TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Generalized File System Dependencies

Christopher Frost *et al.*

Paper Reading Group Presentation
by Carsten Weinhold

Dresden, 2008-02-20

Motivation

- Goal: Durability of file system changes:
  - Do not loose data after crash
  - File system in consistent state every write
- Many approaches:
  - Journaling
  - Soft updates
  - WAFL
  - ACID transactions
- In practice:
  - Developers trade one for the others
  - Little flexibility for applications (fsync(), ...)
  - ***Hard to get right***

Featherstitch

- New architecture for building file systems
- Generalized abstraction of *write-before dependencies*
- Based on **patches**:
  - Describe changes to blocks
  - Depend on other patches
  - Used to implement consistency models in file system code
- Applications can express dependencies using **Patchgroup** abstraction

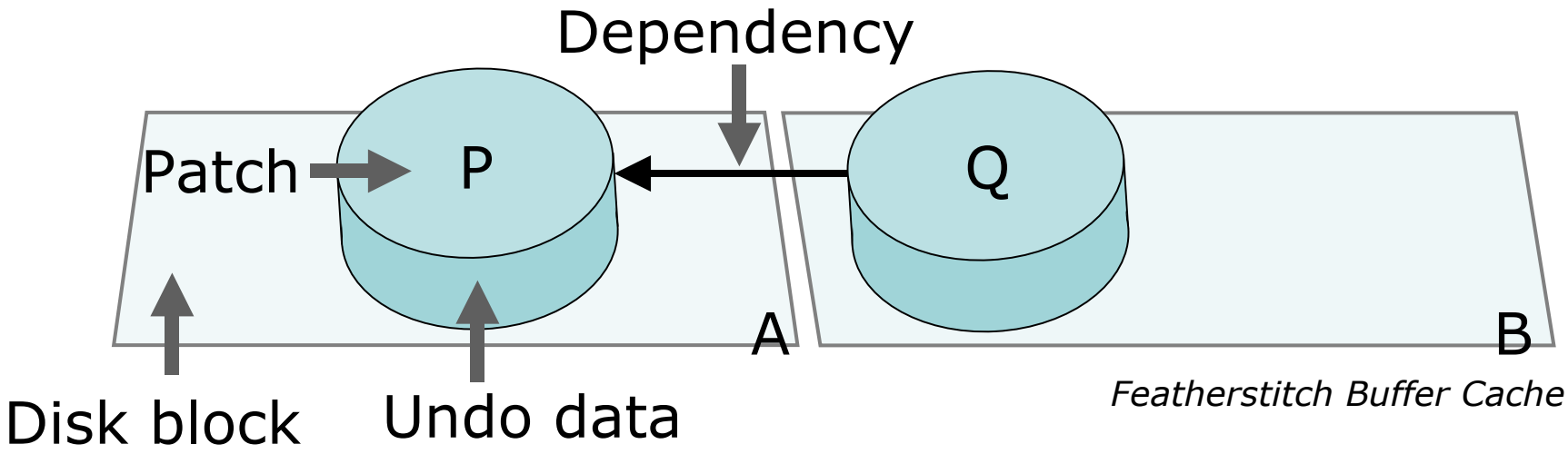
Featherstitch Contributions

- Usable for various consistency models
- Patches and patchgroups are agnostic to specific file system
- New implementation:
  - Ext2 and UFS file systems
  - Patch-aware buffer cache
  - Journaling and soft updates based on patches
  - Other approaches not mentioned in paper implemented as well
  - Available for Linux, Mac OS X, FUSE

Patches

- Describe data change in a block (+ undo data)
- Specify dependencies on other patches

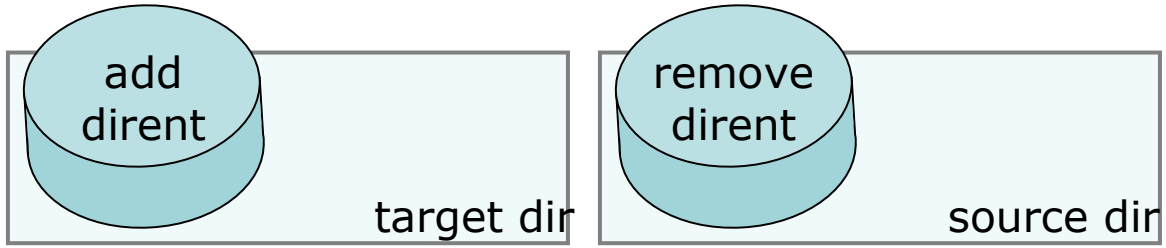
```
patch_create(block *b, int ofs, int length,
             char *data, patch *dep)
```



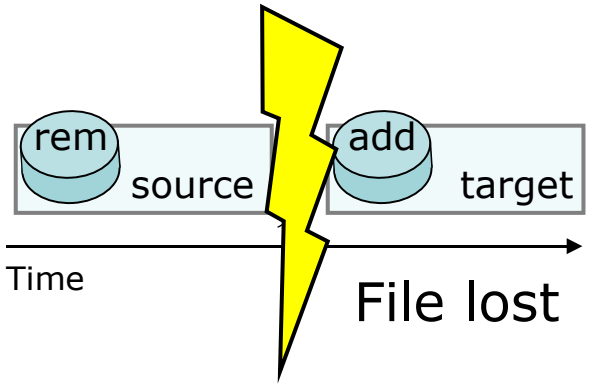
(Figures taken from the original SOSP 2007 presentation)

Example: Asynchronous rename()

- Filename in source dir needs to be removed
- Filename in destination needs to be added

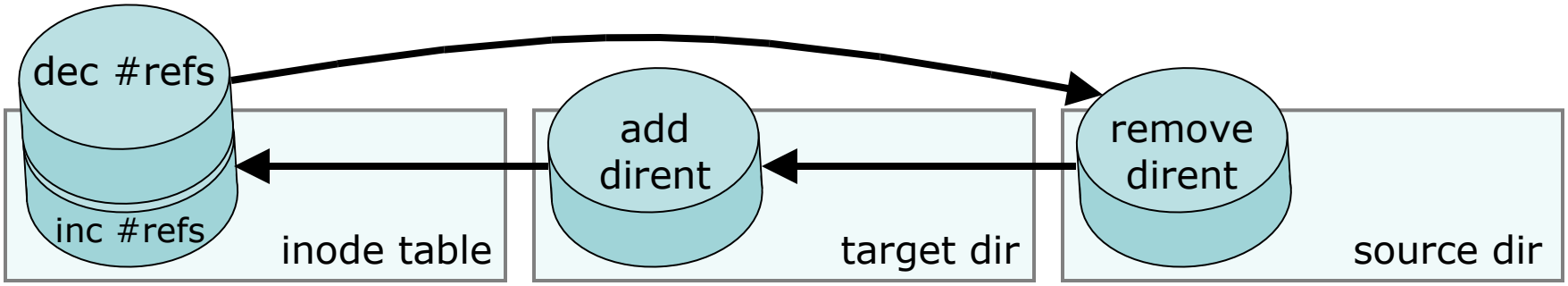


- Problem:



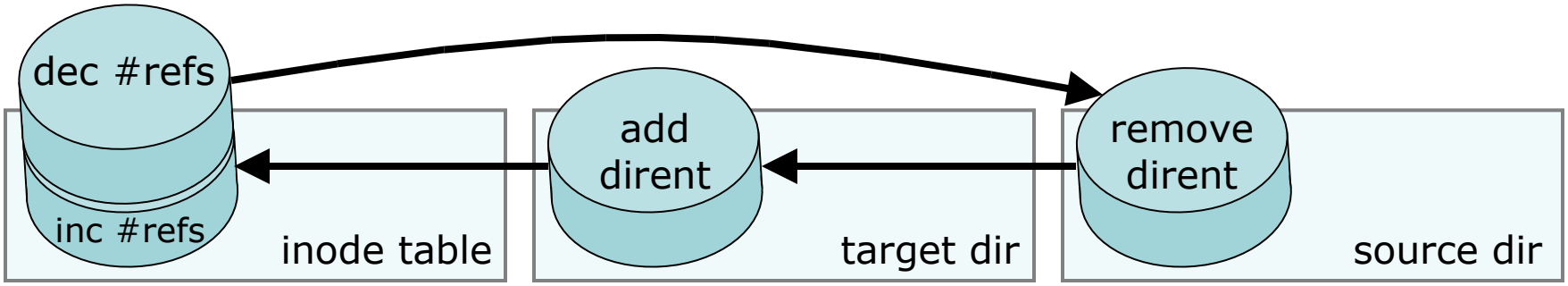
(Figures taken from the original SOSP 2007 presentation)

Example: rename() with Soft Updates

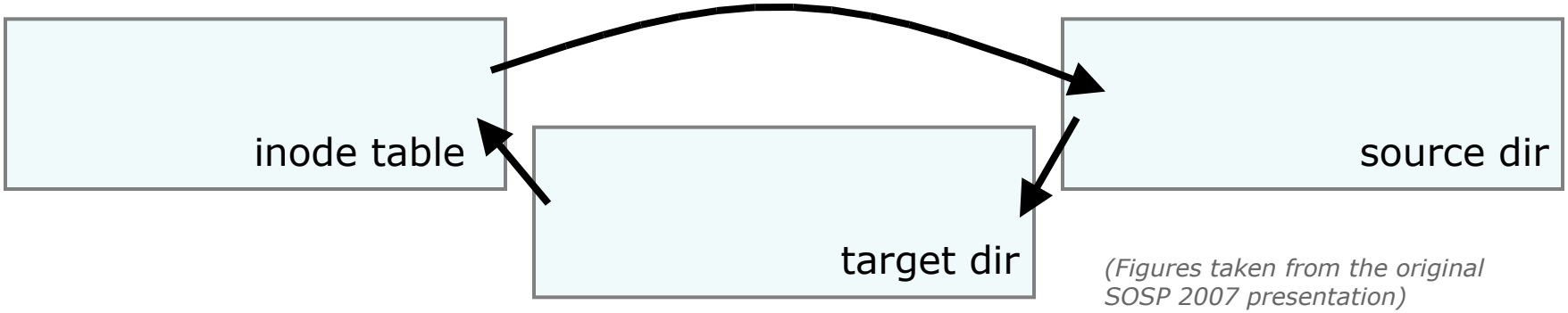


(Figures taken from the original SOSP 2007 presentation)

Example: rename() with Soft Updates

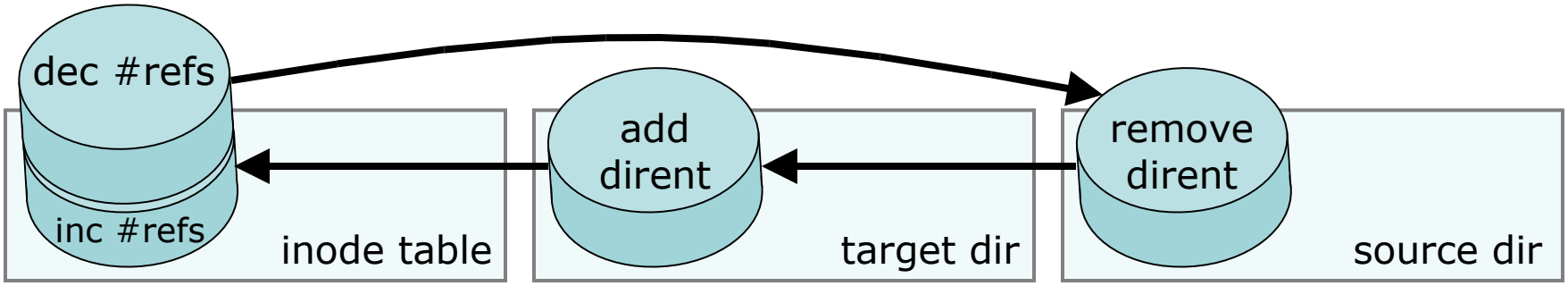


Block level cycle:

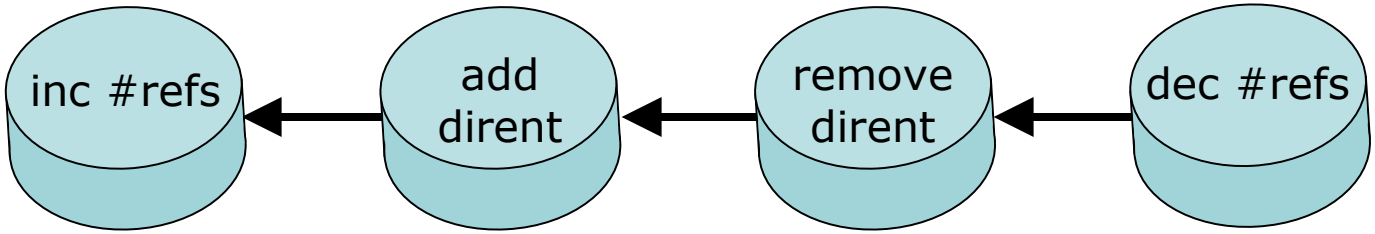


(Figures taken from the original SOSP 2007 presentation)

Example: rename() with Soft Updates

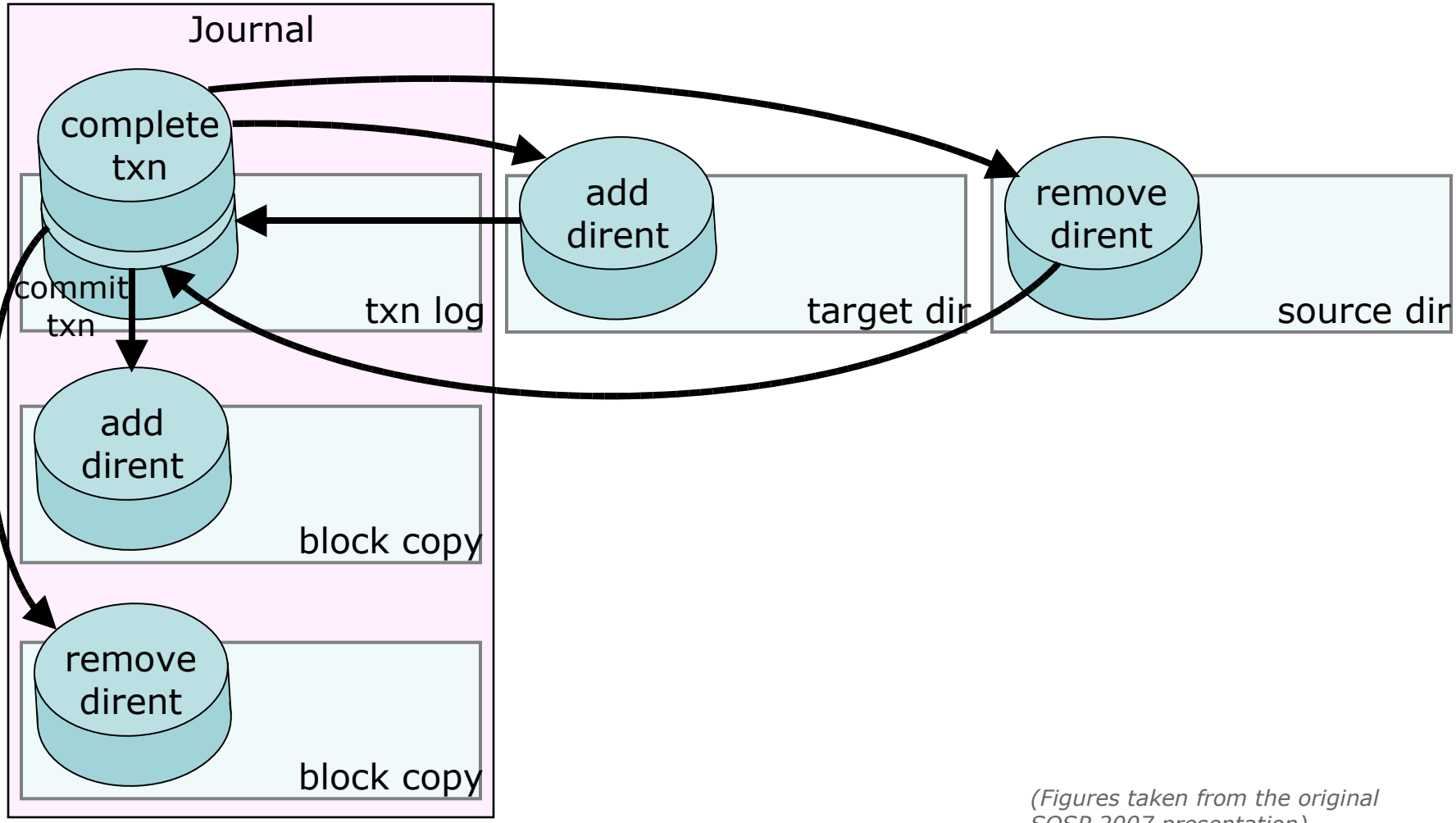


Not a *patch level* cycle:



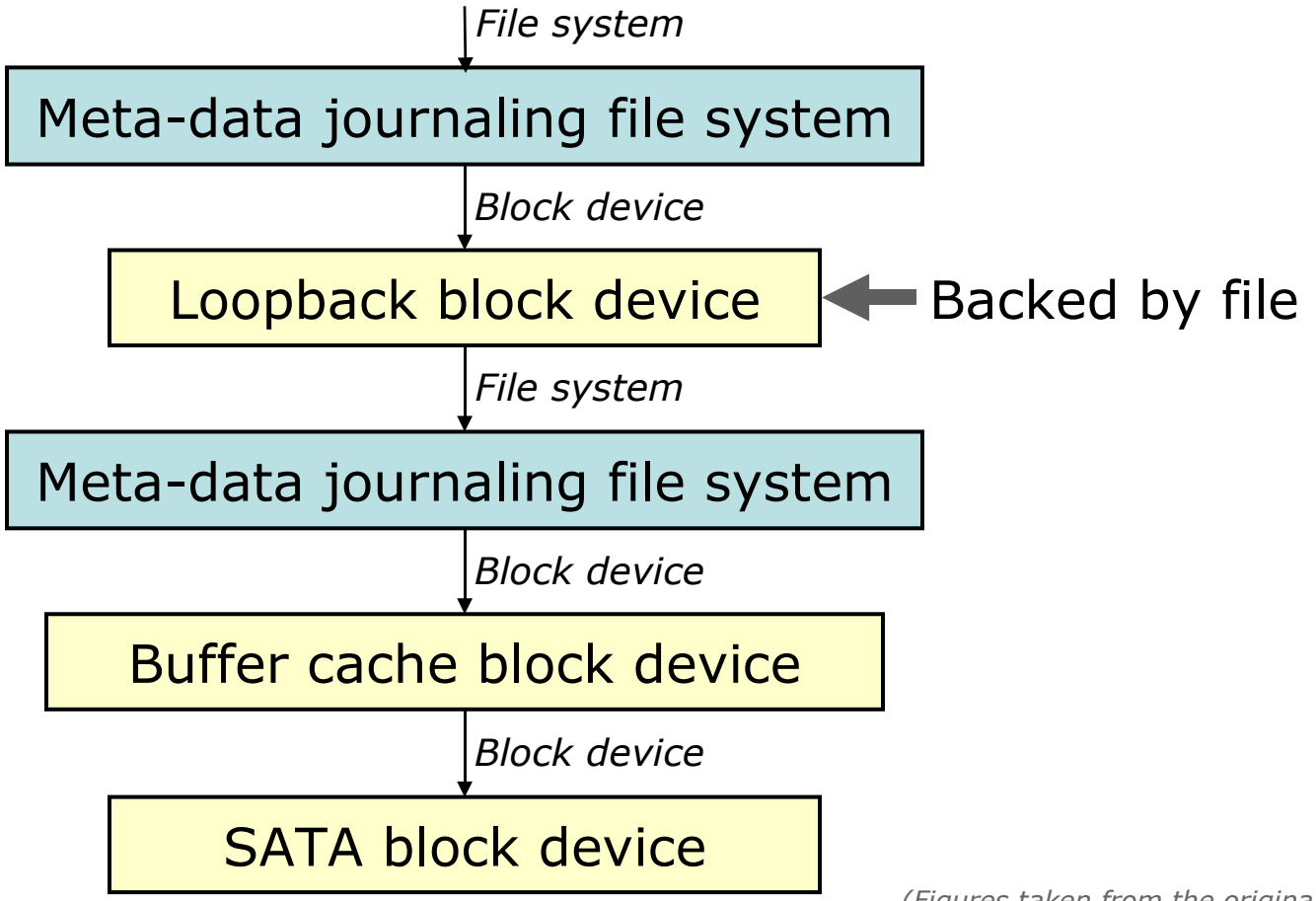
(Figures taken from the original SOSP 2007 presentation)

Example: rename() with Journaling



(Figures taken from the original SOSP 2007 presentation)

Example: Loopback Block Device



(Figures taken from the original SOSP 2007 presentation)

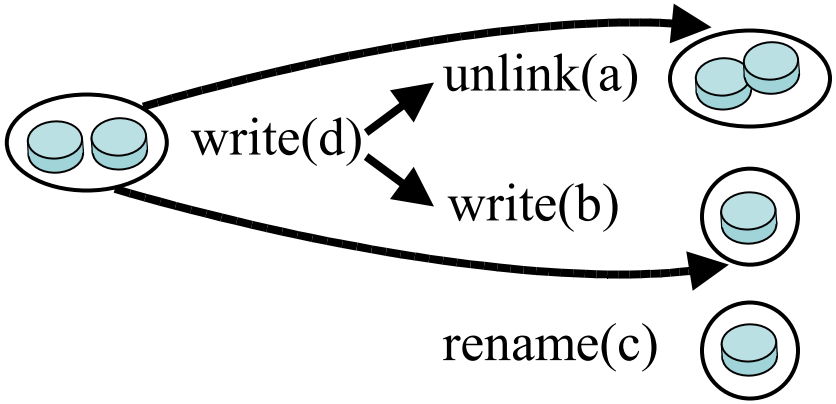
Meta-data journaling file system obeys file data requirements

Application-Level Consistency

- Consistency required in many use cases:
  - Databases
  - E-mail servers
  - Version control systems
- Common solutions:
  - Expensive sync operations (fsync(), ...)
  - Rely on underlying file system (data journaling)
  - (Re-)Implement all consistency code (e.g., database systems operating on raw disks)

Patchgroups

- Applications specify write-before dependencies using **patchgroups**
- Can span multiple files

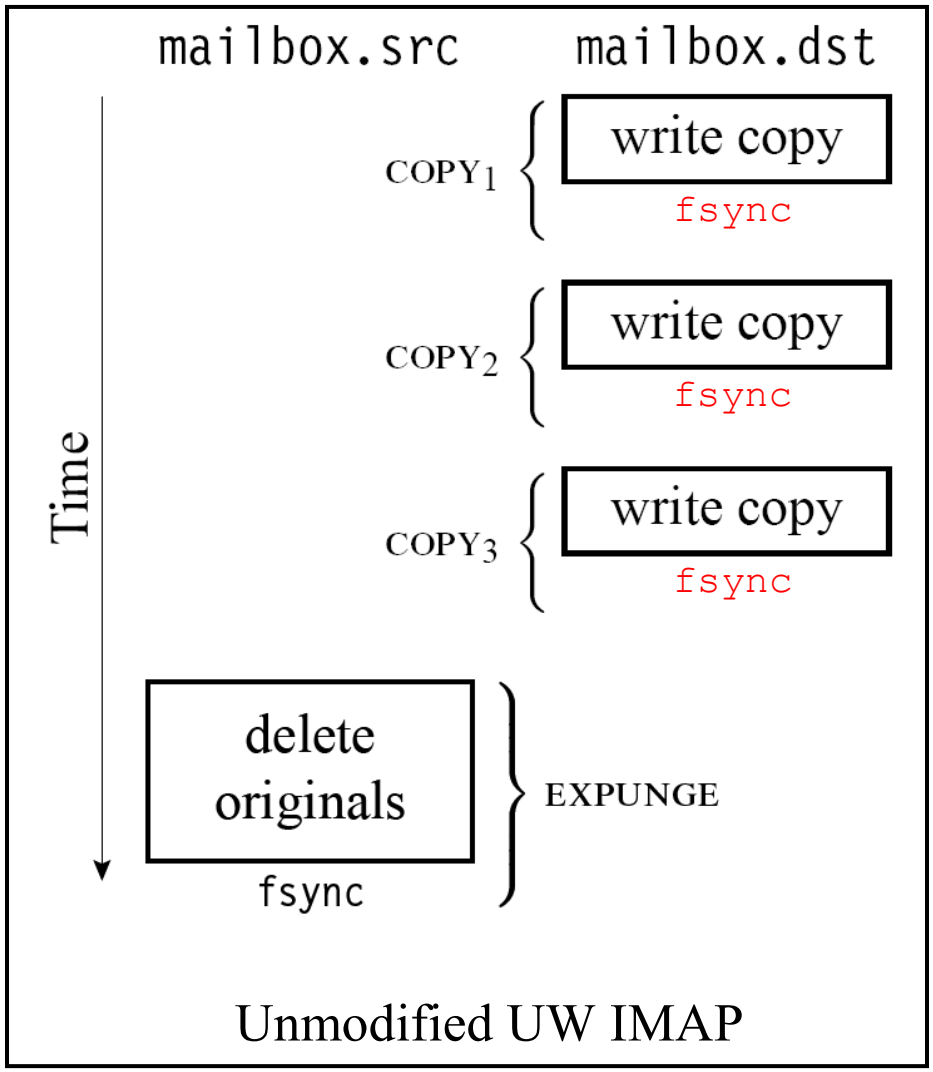


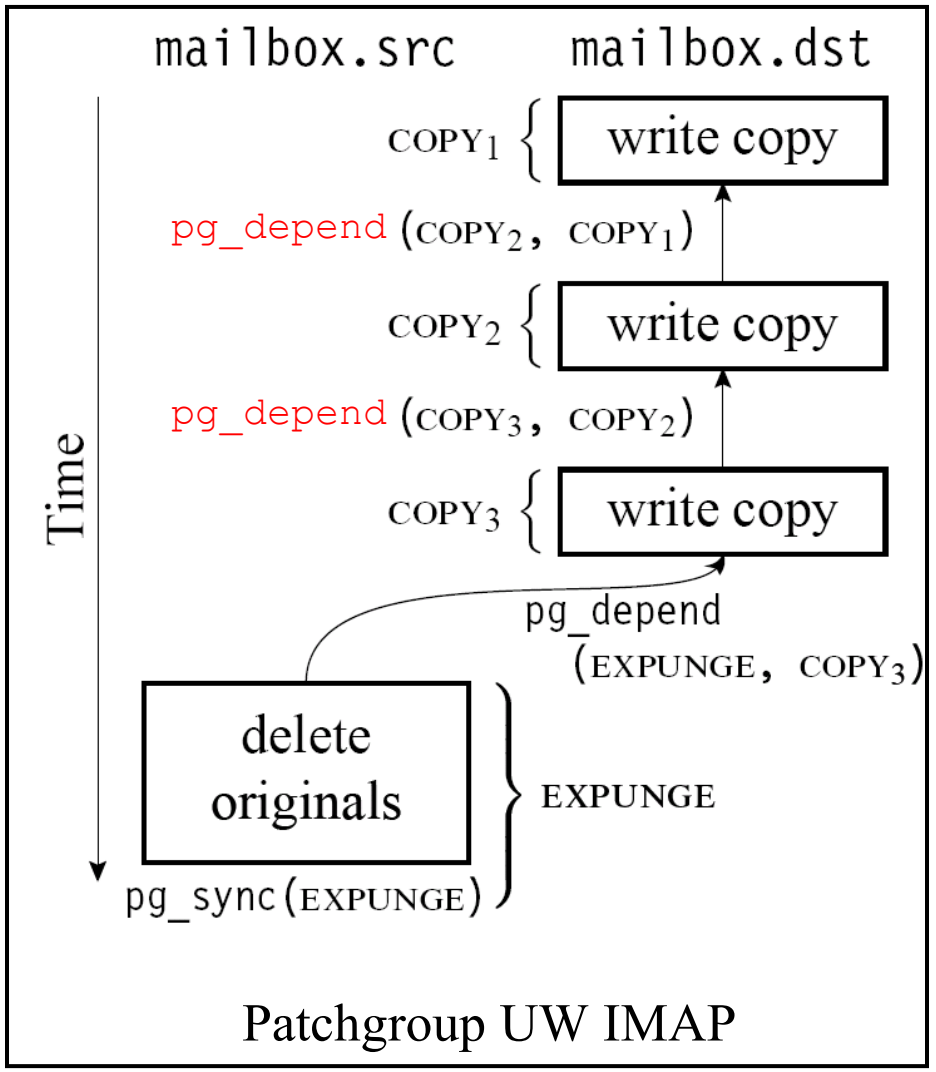
(Figures taken from the original SOSP 2007 presentation)

Patchgroup API

```
typedef int pg_t;                pg_t pg_create(void);
int pg_depend(pg_t Q, pg_t P);   /* adds Q ⇝ P */
int pg_engage(pg_t P);           int  pg_disengage(pg_t P);
int pg_sync(pg_t P);             int  pg_close(pg_t P);
```

Example: UW IMAP



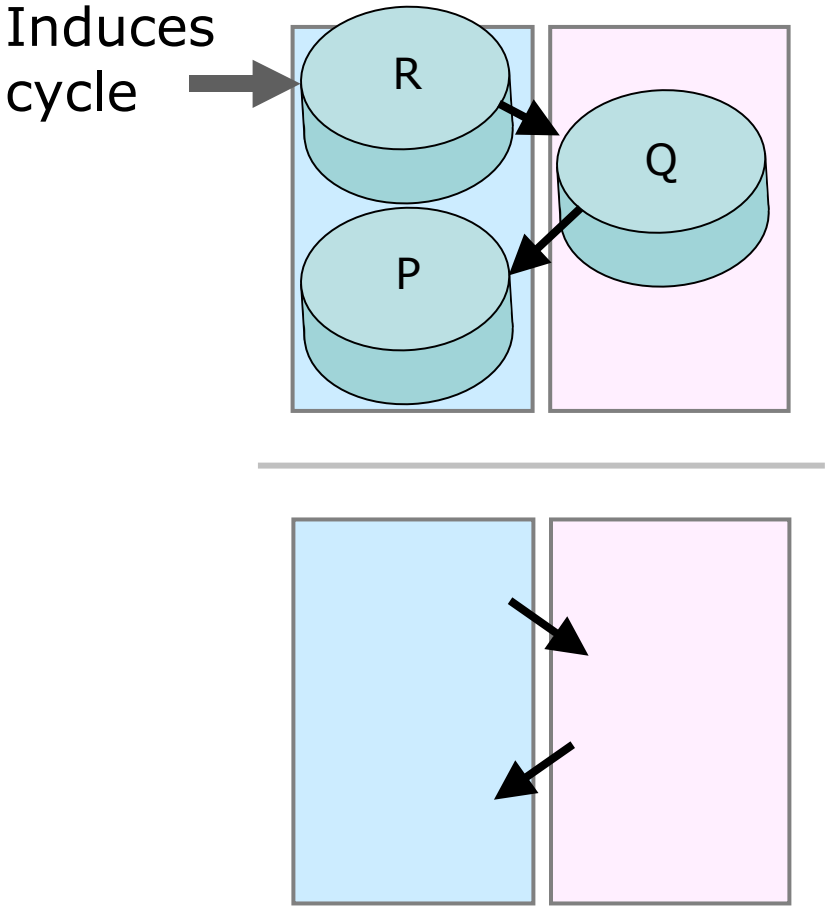


Patch Optimizations

- First naïve implementation performed badly:
  - Patches and their undo data consumed too much memory
  - CPU time for handling patches too high
- Authors optimized Featherstitch:
  - Omit undo data if not needed
  - Merge patches to reduce number of patches and dependencies
- Problem: Undo data can only be omitted if future is known!

Block-Level Cycles

- Dependency cycles are forbidden
- Block-level cycles can happen
- Idea: Restrict API
  - Dependencies can only be specified when creating a patch (more or less)
  - Block-level cycles are known at creation



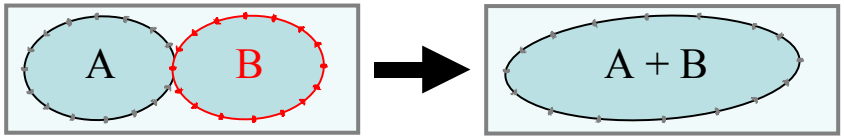
(Figures taken from the original SOSP 2007 presentation)

Hard Patches vs. Soft Patches

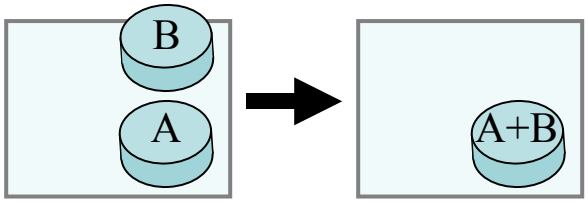
- Any patch that needs to be reverted must induce block-level cycle
- Two types of patches:
  - Hard patches
    - No block-level cycle
    - No undo data
  - Soft patches
    - Need to be reversible
    - Undo data

Patch Merging

- Hard patch merging:
  - No undo data
  - Only one hard patch per block



- Overlap merging of soft patches:
  - Can be merged with soft patch
  - Can be merged with hard patch, if previous dependencies no longer exist

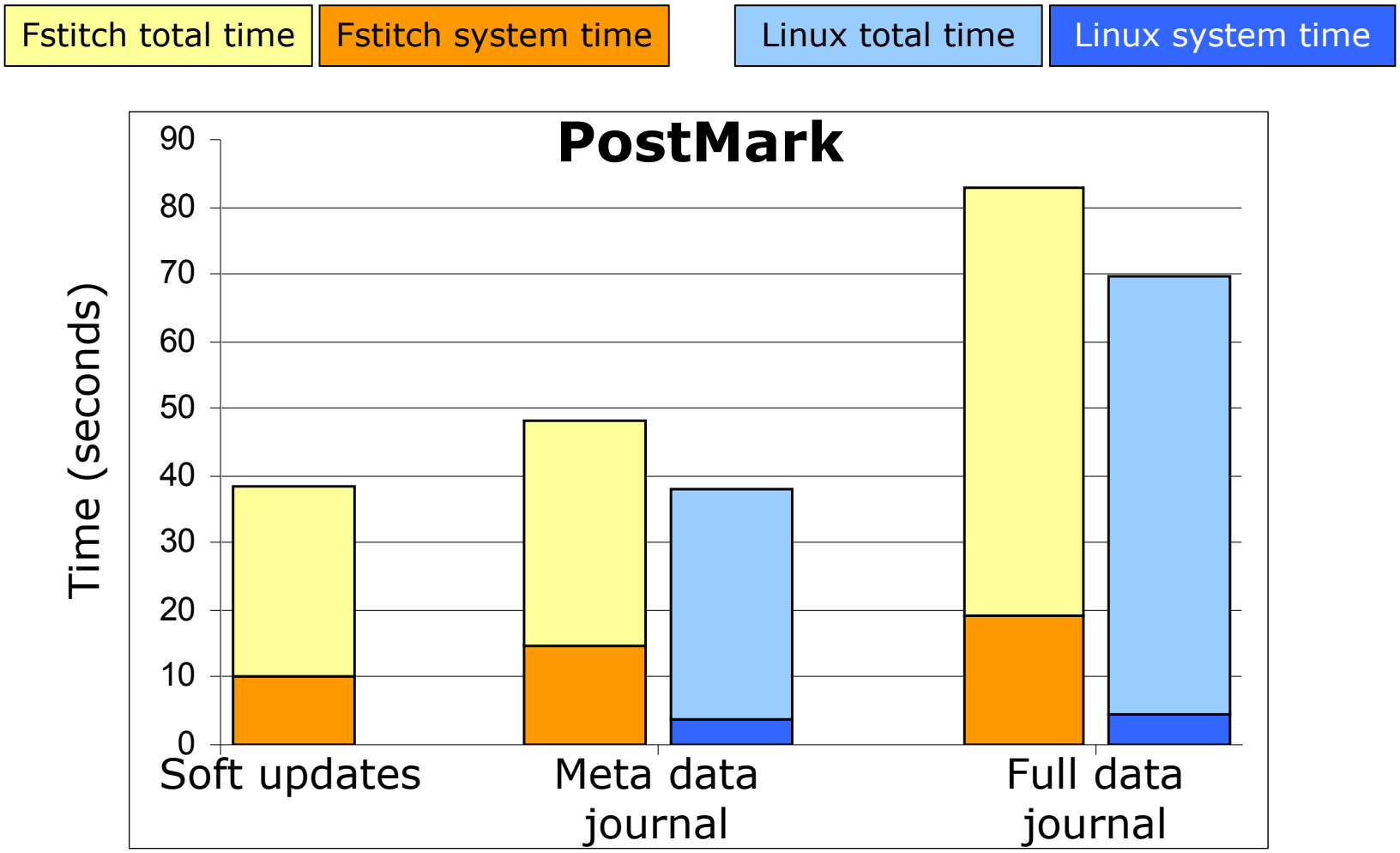


(Figures taken from the original SOSP 2007 presentation)

Other Optimizations

- Ready patch list:
  - Maintained per block in in the buffer cache
  - Contains patches that are safe for writing (making them in-flight)
  - Patches are added, if there are no more direct dependencies that are in uncommitted or in-flight state
- General optimization in the file system itself
  - Minimal dependencies, block layout
  - ...

Evaluation: Postmark Performance



Evaluation: UW IMAP Performance

Implementation	Time (sec)	# Writes
Featherstitch ext2		
soft updates, fsync per operation	65.2 [0.3]	8,083
full journal, fsync per operation	52.3 [0.4]	7,114
soft updates, patchgroups	28.0 [1.2]	3,015
full journal, patchgroups	1.4 [0.4]	32
Linux ext3		
full journal, fsync per operation	19.9 [0.3]	2,531
full journal, fsync per durable operation	1.3 [0.3]	26

Figure 11: IMAP benchmark: move 1,000 messages. System CPU times shown in square brackets. Writes are in number of requests. All configurations are safe.

Sources

- ***Generalized File System Dependencies***, Christopher Frost, Mike Mammarella, Eddie Kohler, Andrew de los Reyes, Shant Hovsepian, Andrew Matsuoka, Lei Zhang SOSP 2007, Stevenson, WA, USA

- These slides are based on the original SOSP 2007 presentation slides created by the paper's authors: http://www.sosp2007.org/talks/sosp169-frost.ppt

Discussion Points

- Featherstitch is really cool stuff!
- Is it feasible to have consistency enforcement outside the TCB operating on encrypted blocks / patches?
- It is generic, can this be done in hardware (e.g., harddisk firmware)?