Ottawa Linux Symposium 2005

State of the Art: Where we are with the ext3 filesystem

Mingming Cao, Theodore Y. Ts'o,
Badari Pulavarty, Suparna Bhattacharya
IBM Linux Technology Center

Andreas Dilger, Alex Tomas
Cluster Filesystem Inc.

Introduction

- Why ext3 filesystem has a large number of community and many users?
- Why improving current ext3 filesystem is important?
- The plan to “modernize” ext3 filesystem.
- What happened to ext3 in the past three years?

Activities Summary

Included features	Linux-2.4	Linux-2.6.11
Directory Indexing	patch/vendor	yes
Online Resizing	patch	yes
Reduce Locking Contention	no	yes
Block Reservation	no	yes
Extended Attributes	patch/vendor	yes
Under development	Linux-2.4	Linux-2.6
Extents	patch	patch
Delayed Allocation	no	patch
Multiple Block Allocation	no	patch
Reduce File Removal Latency	patch	easy to port
Increased subdir support	patch	patch
Parallel directory operations	patch	patch
Finer Timestamp	no	patch

Overview

- Part A: New ext3 features added to Linux 2.6
- Some ext3 features under development:
  - Part B: Features imply filesystem format change
  - Part C: Improvements with current ext3 layout
- Future work

Part A: Features Added to Linux 2.6

- Directory indexing
  (By Daniel Phillips, Theodore Y. Ts'o, 2.5)
- Online resizing
  (By Andreas Dilger, Stephen Tweedie, 2.6.10)
- Removing BKL and improve scalability
  (By Andrew Morton, Alex Tomas, 2.5)
- Extended attributes
  (By Andreas Gruenbacher, 2.5)
- Reservation based block preallocation
  (By Mingming Cao, Andrew Morton, Stephen Tweedie, Badari Pulavarty 2.6.10)

Directory Indexing

- Scalability issues with old ext2/3 directories: simple linked list.
- A simplified tree structure (Htree) was designed for directories.
- HTree features: 32-bit hashes for keys, high fanout factor, constant depth
- Boots ext3 performance on large directories.

Online Resizing

- Online resizing allows filesystem growing without having to take down time. A very useful feature in server environments.
- Handles three primary phases that a filesystem can grow:
  - Grow within the last block group
  - Need a new block group
  - Need a new block group descriptor

Reduce Lock Contention

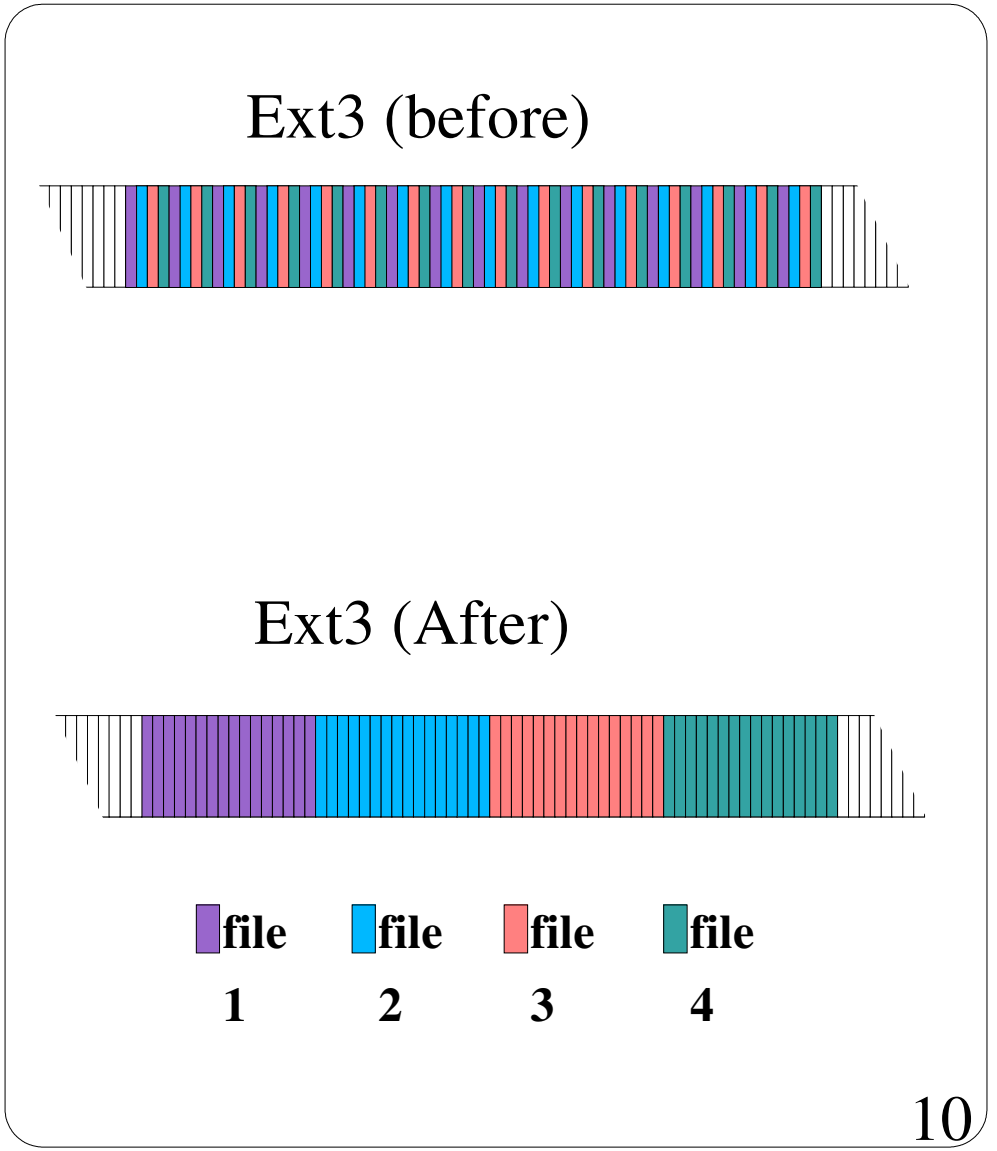
- Scaling issue for 2.4 ext3/JBD with concurrent IO
- A series of effort were made:
  - Ext3: replaced per-filesystem superblock lock with finer-grained locks
  - Journalling layer: pushing BKL out of JBD
- SDET benchmark throughput improved by a factor of 10

Extended Attributes

- Extended Attributes:Small amount of custom metadata with files or directories
- Added to ext2/3 to support ACL
- EAs are stored in a single EA block, shareable by inodes have same extended attributes
- Can be stored in expanded inode itself(2.6.11+)
- EA-in-inode noticeably speed up ext3 performance on Samba4 benchmark

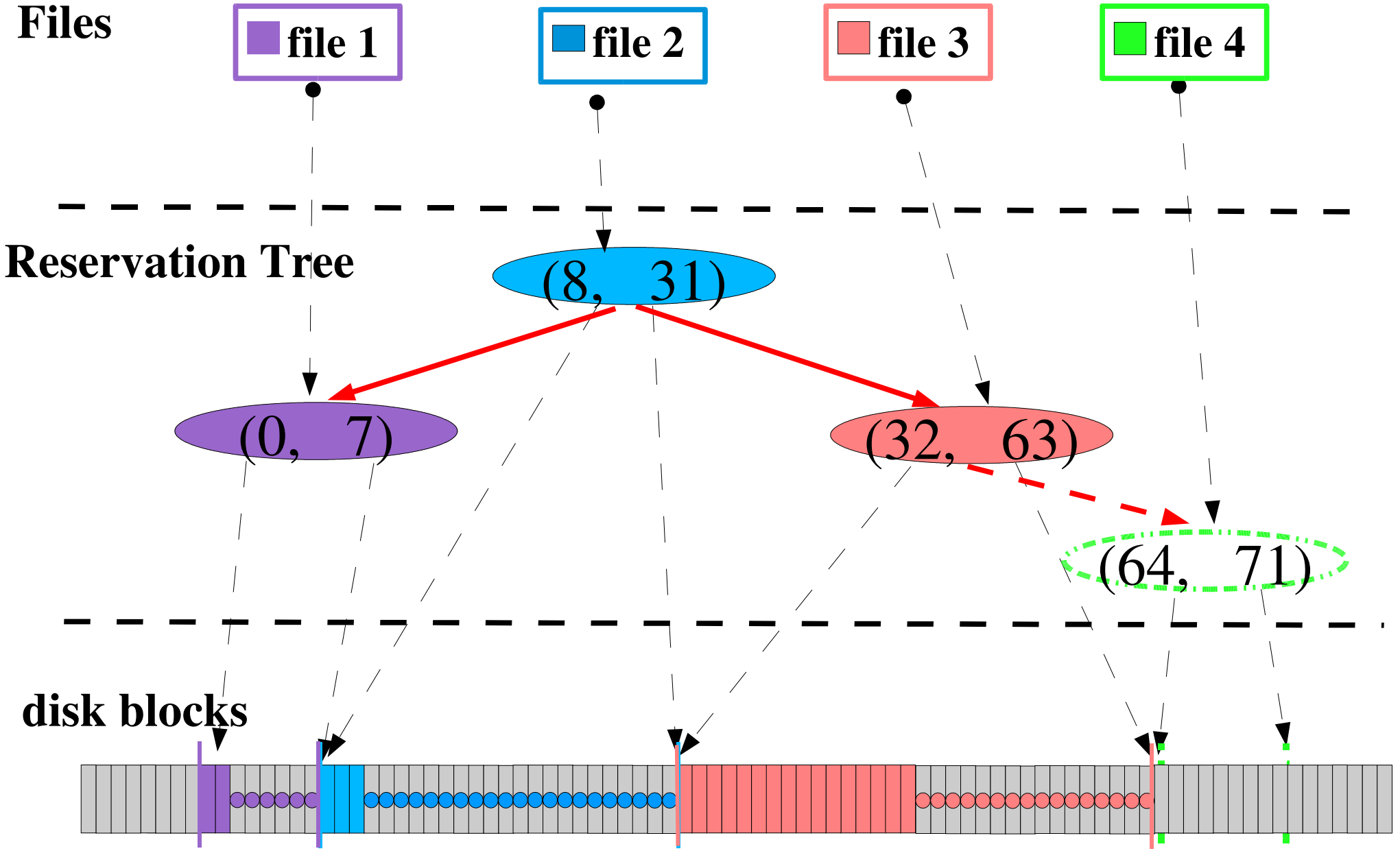
Reservation based block preallocation

- Block preallocation helps reduce file fragmentation
- Ext3 uses in-memory block reservation to support a large preallocation
- Results in significant throughput improvements on concurrent sequential writes and the subsequent sequential read

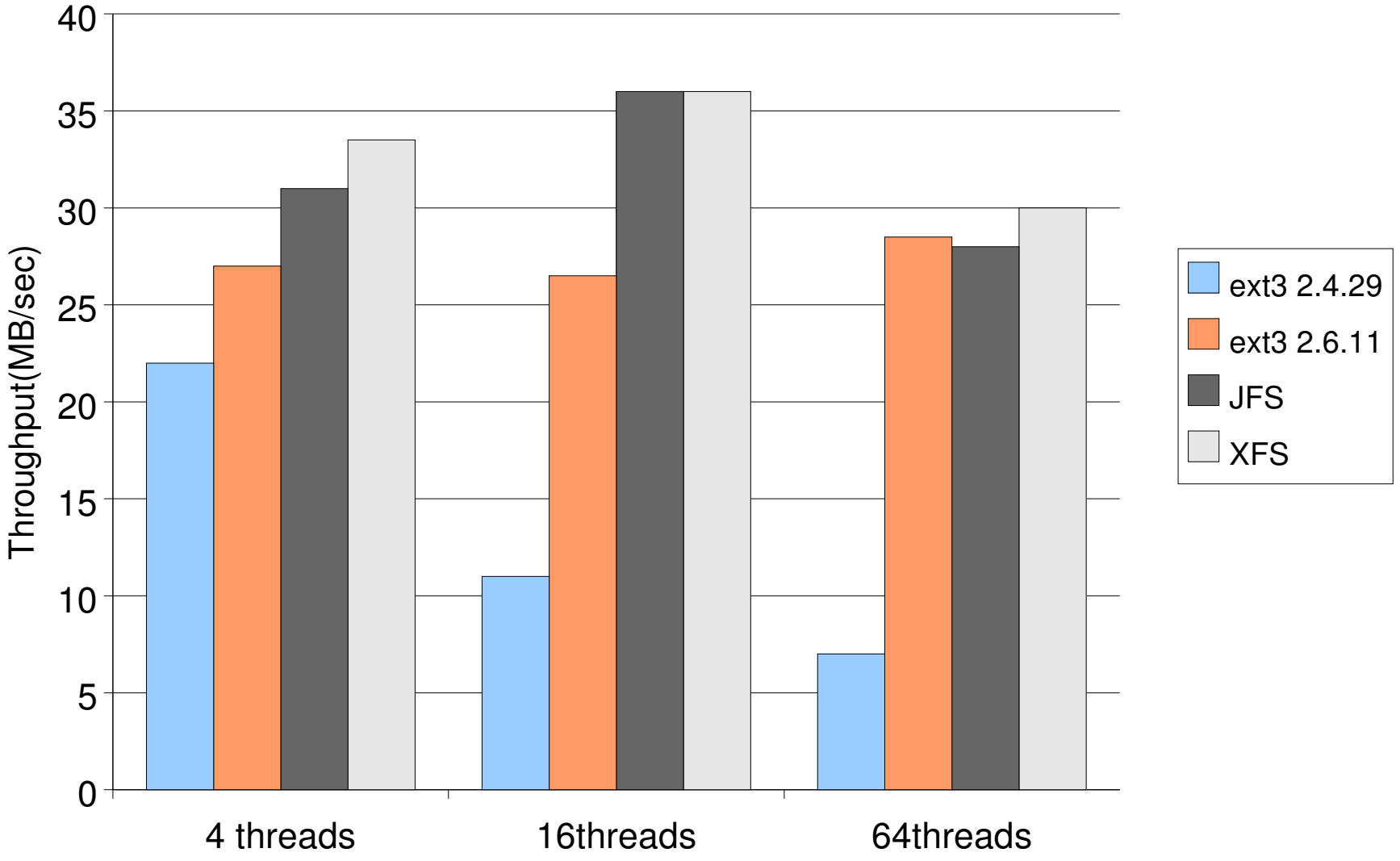


Ext3 Block Reservation

- Key difference: Reservation in memory, rather on disk
- Each inode has it's own reservation window, windows cannot overlapped, indexed by a per-filesystem red-black tree
- Allocation is within the window. Window could dynamically move and grow
- Discard window at the last file close


Files
file 1
file 2
file 3
file 4
Reservation Tree
(8, 31)
(0, 7)
(32, 63)
(64, 71)
disk blocks

tiobench sequential write

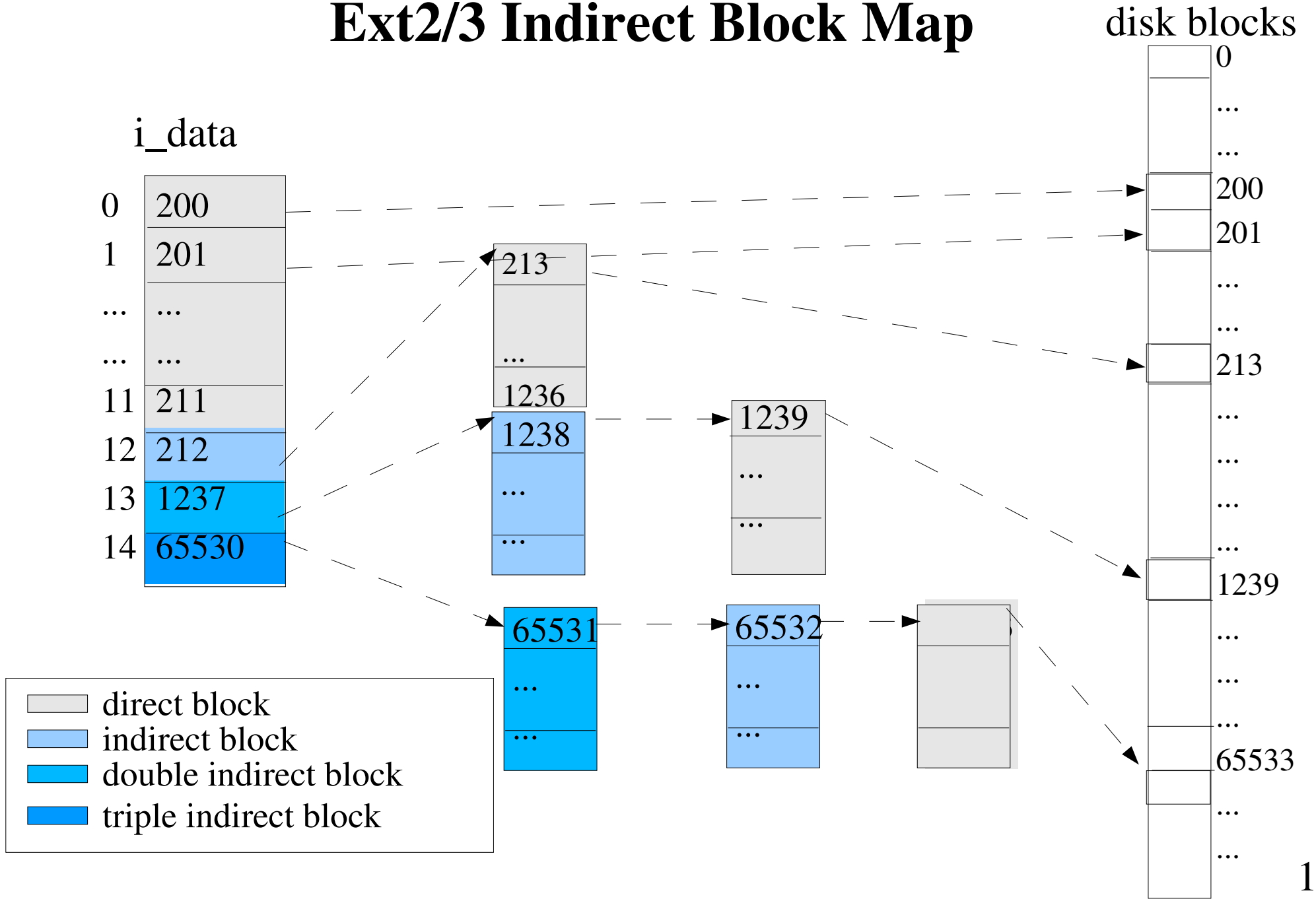


Part B: Extents and Related Work

- Extents
- Extents Allocation
- Delayed allocation for extents

(By Alex Tomas)

Ext2/3 Indirect Block Map



Extents

- Extent is an efficient way to represent large contiguous file
- An extent is a single descriptor for a range of contiguous blocks

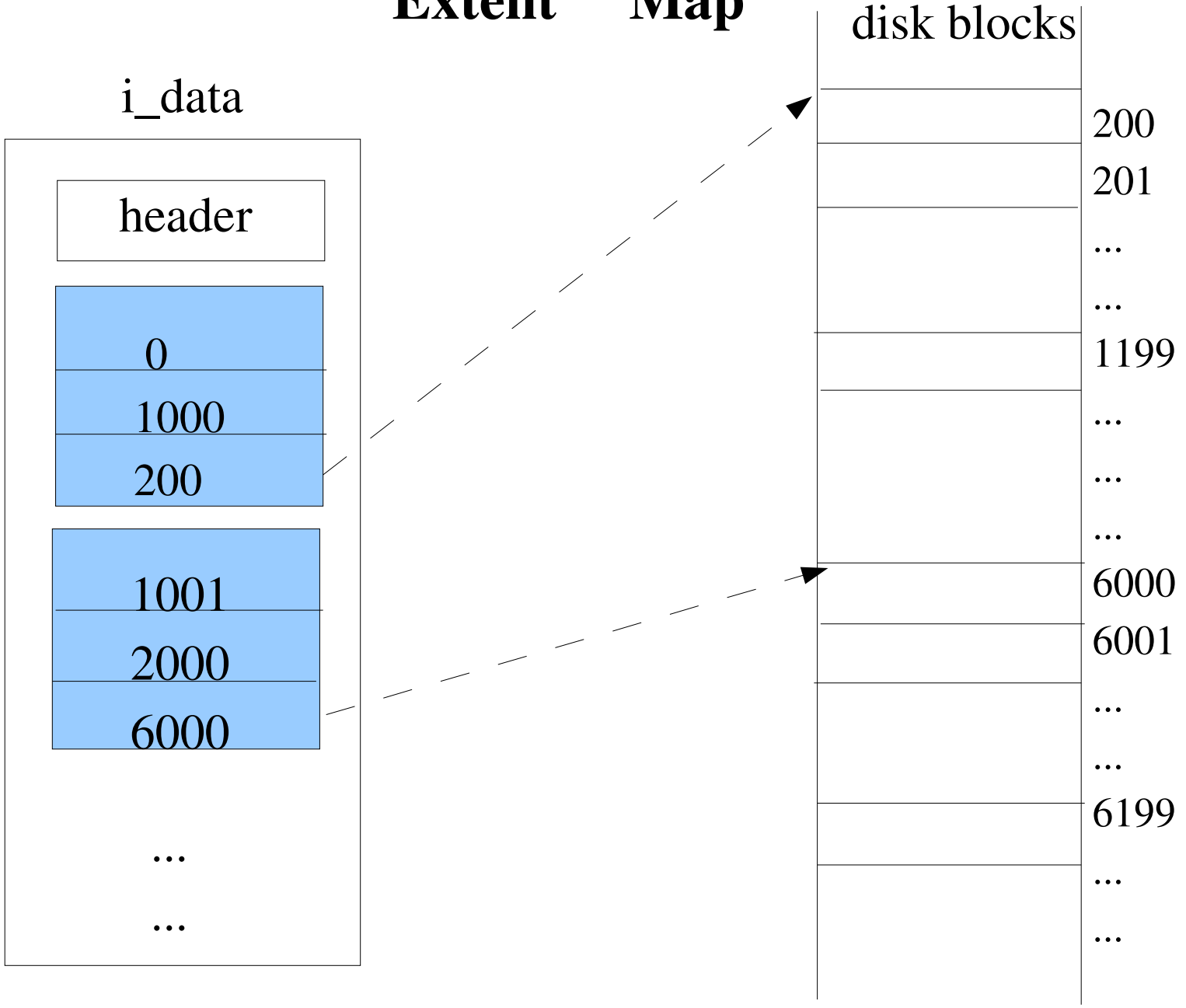
logical	length	physical
0	1000	200
32 bit	16 bit	48 bit

Extents Data Structures

```
struct ext3_extent {
  __u32    ee_block;  /* logical */
  __u16    ee_len;     /* length */
  __u16    ee_start_hi;
  __u32    ee_start;   /*physical*
};
```

```
struct ext3_extent_header {
  __u16    eh_magic;
  __u16    eh_entries;
  __u16    eh_max;
  __u16    eh_depth;
  __u32    eh_generation;
};
```

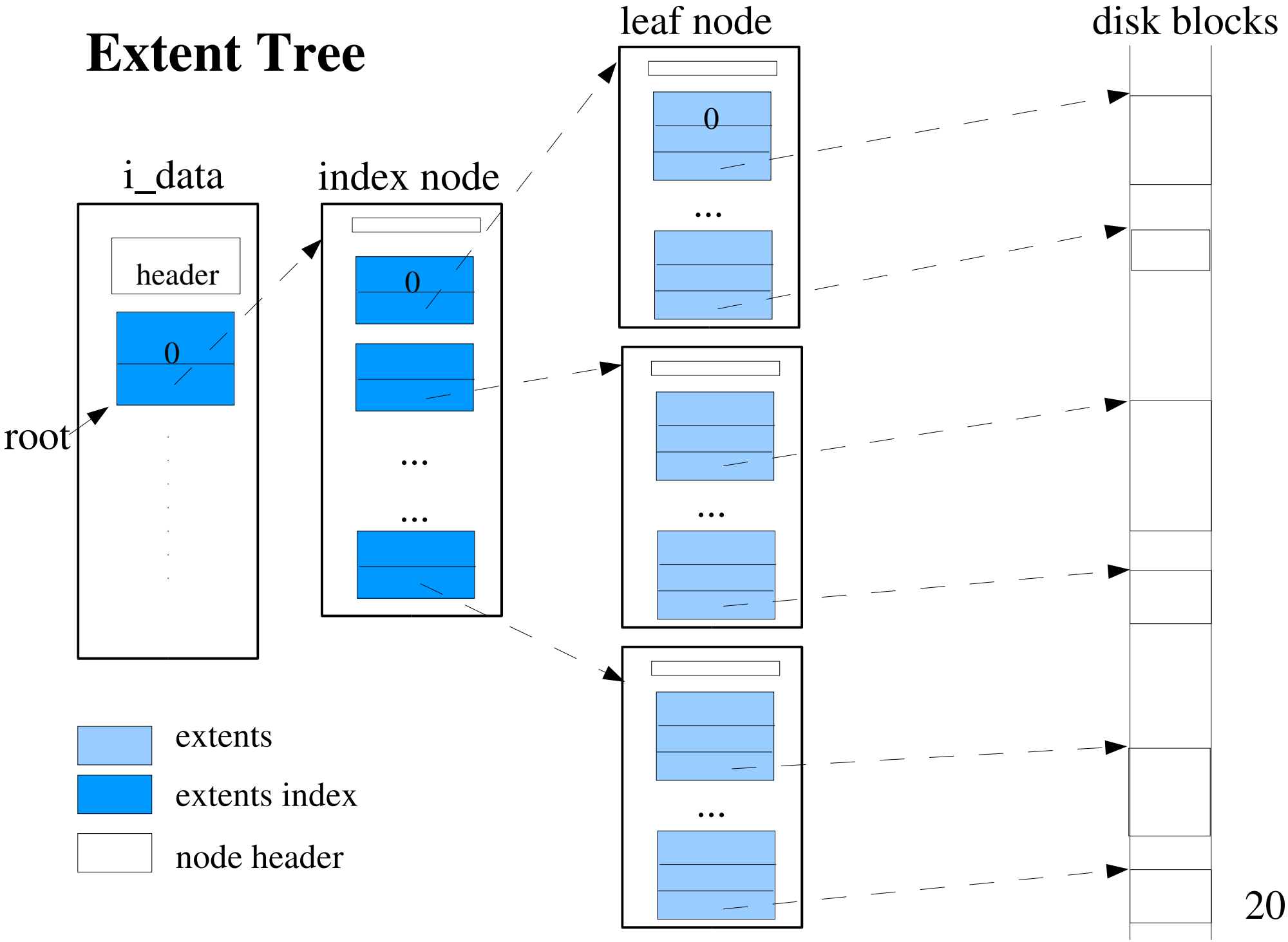
Extent Map


i_data
header
0
1000
200
1001
2000
6000
...
...
disk blocks
200
201
...
...
1199
...
...
...
6000
6001
...
...
6199
...
...

Extents Tree

- A simplified tree – like Htree used in directories
- Constant depth
- Tree nodes including index node and leaf node
- Each node start from a header structure
- A flag in inode indicating the block addressing type: extent map or indirect block mapping

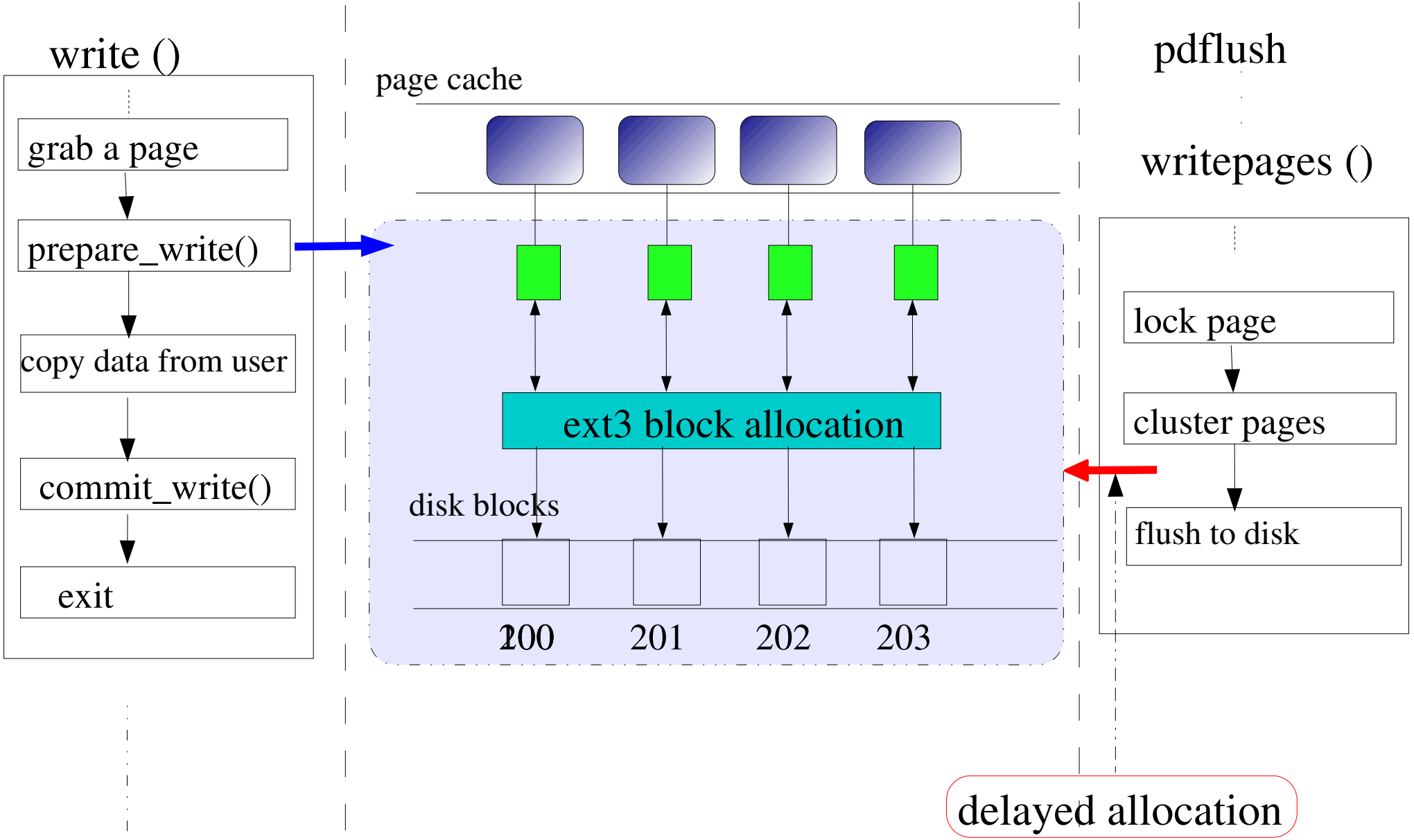
Extent Tree


leaf node
disk blocks
i_data
index node
header
0
0
0
...
...
...
root
extents
extents index
node header

Extent Related Work

- Multiple block allocation
  An efficient way to allocating a chunk of contiguous blocks at a time
- Delayed allocation
  Enable multiple block allocation for buffered IO by deferring and clustering single block allocations

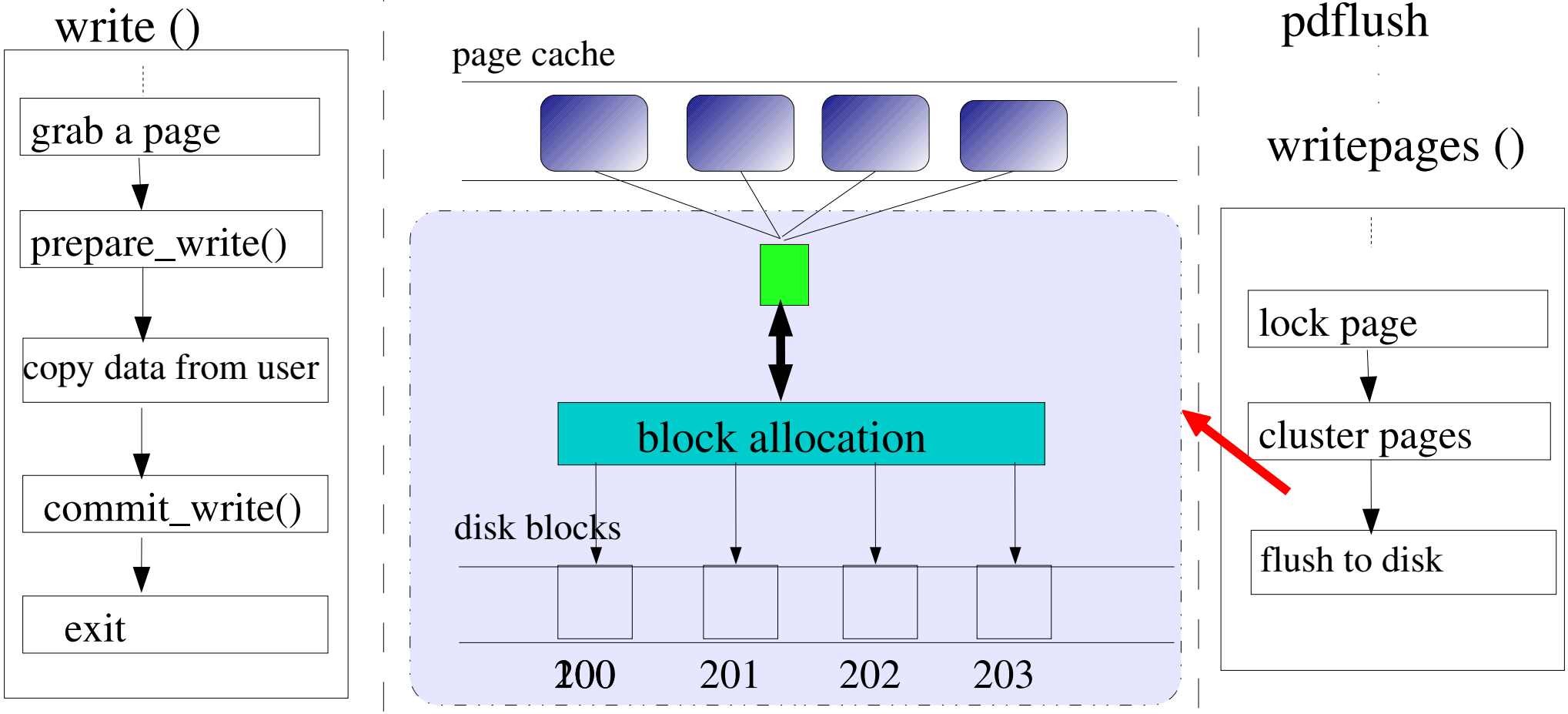
Buffered I/O Write Path



Benefits of Delayed Allocation

- May avoid the need for block allocation for temporary files
- Improves chances of allocating contiguous blocks on disk for a file
- Reduces CPU cycles spent in repeated single block allocation calls, by clustering allocation for multiple blocks together.

Delayed Allocation



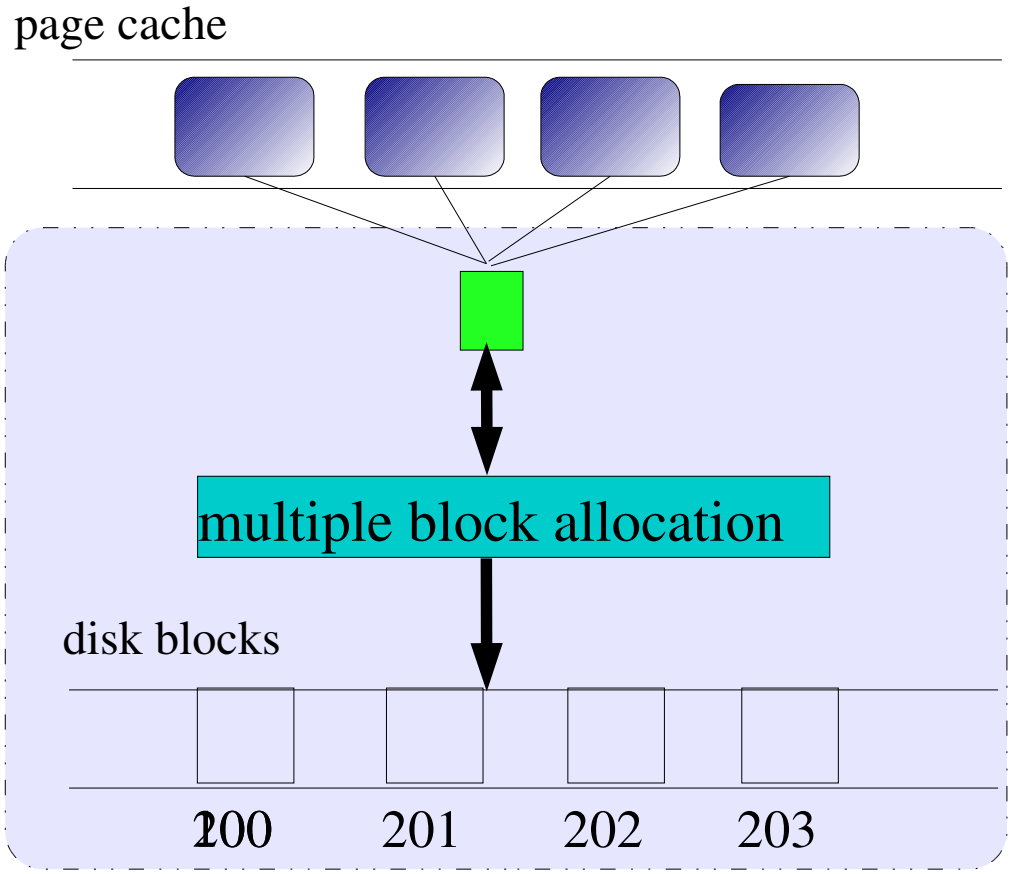
Cost of Single Block Allocation

To add single block to a file ext3 has to:

- Open a transaction
- Load the inode's indirect blocks from disk to memory
- Search filesystem block bitmap to find a free block
- update indirect mapping with new block number
- Add the modified blockmap blocks to the transaction
- Add the modified inode to the transaction

Multiple Block Allocation

- Increase the possibility to get contiguous blocks
- Reduces CPU cycles spent in repeated single block allocation call
- Able to batch metadata update and journaling once

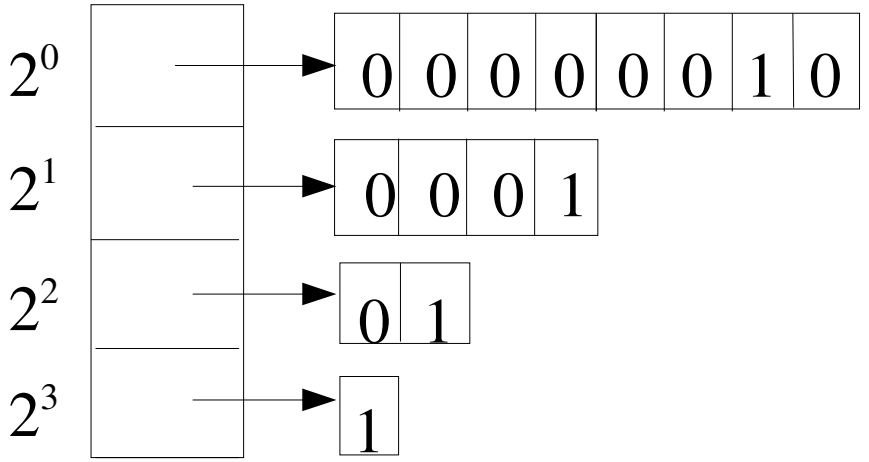


Buddy Based Extent Allocation

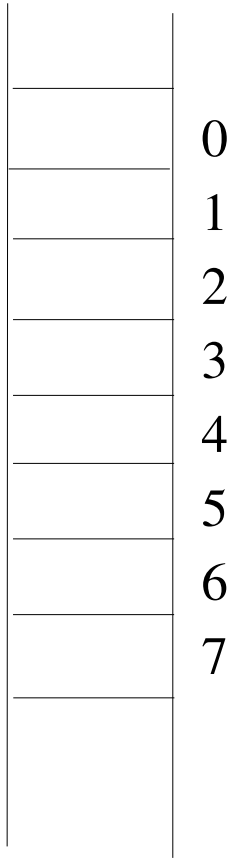
- Collect per block group free extent info and store it in buddy data
- Buddy data is an array of metadata, where each entry describes the status of a cluster of 2^n blocks
- Combine buddy data and traditional block bitmap to quickly search free extent length

Buddy Multiple Block Allocation Example

free extents buddy info



disk block bitmap



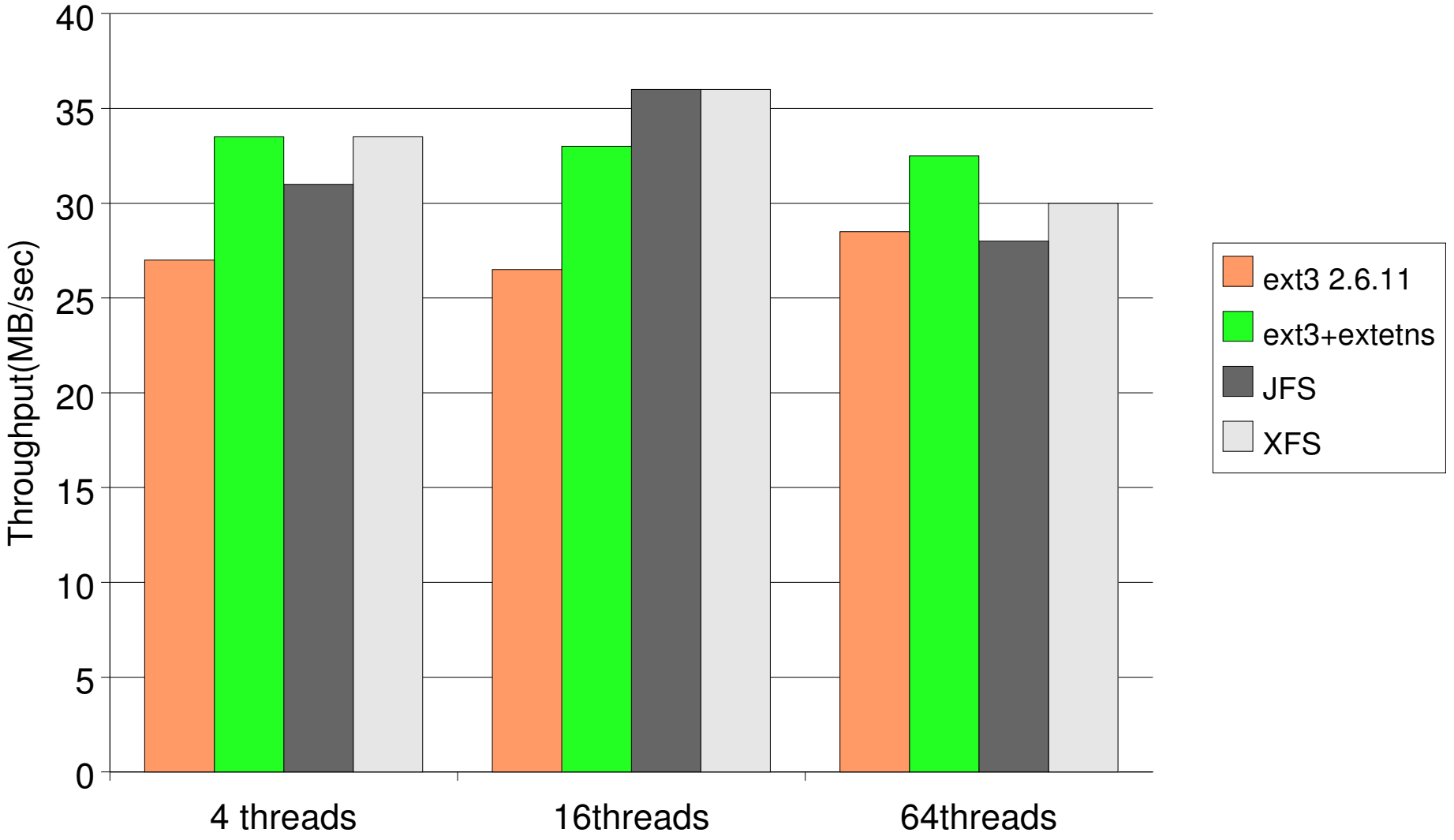
block	bitmap	free exent
0	free	2^2
4	free	2^1
6	allocated	

Total extent from block 0 $2^2 + 2^1 = 6$

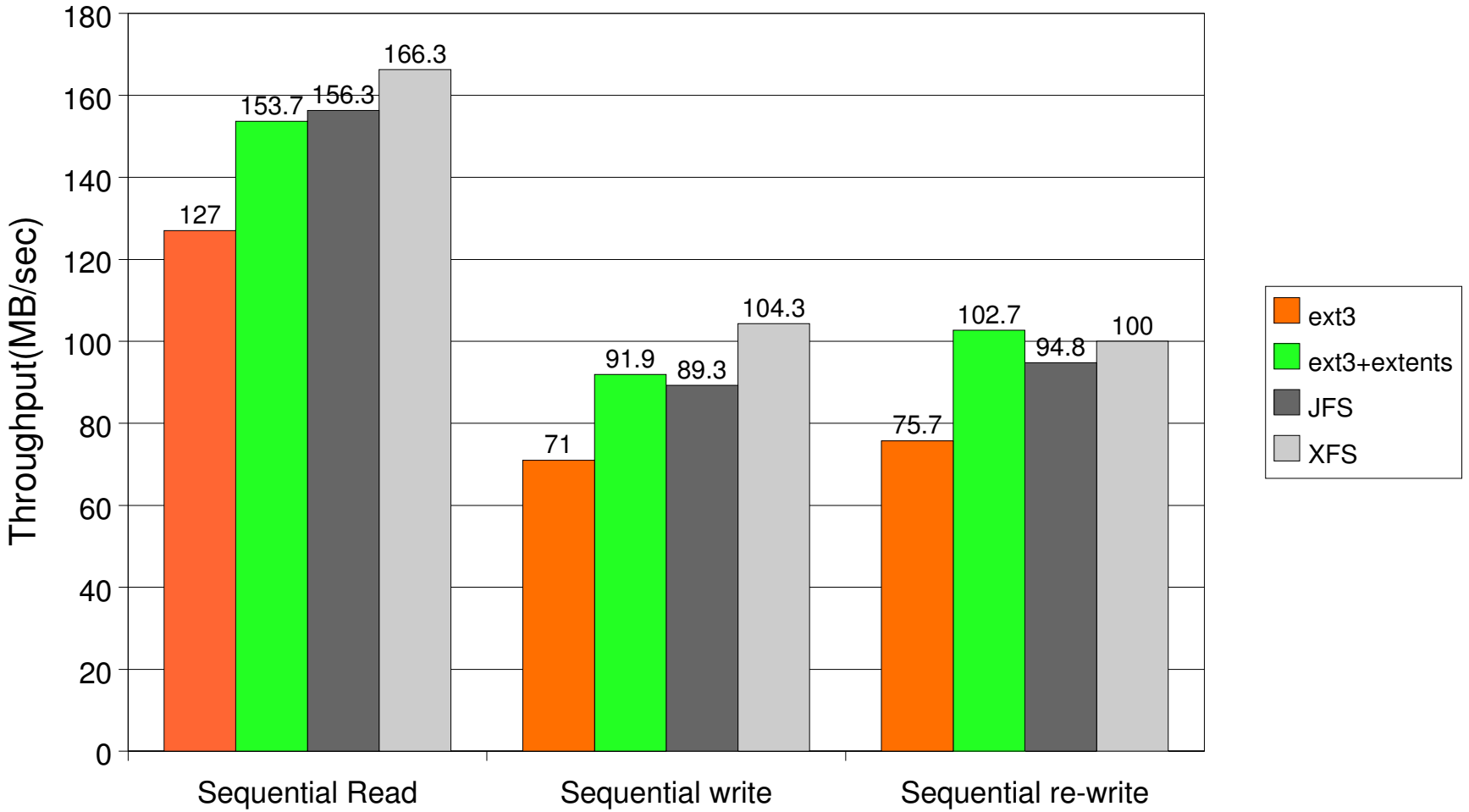
Evaluation of Extents Patches

- Improvements for large file creation/removal/sequential read/sequential rewrite
- Various benchmarking was done: dbench, tiobench, FFSB filemark, sqlbench ,iozone etc.
- Initial analysis indicates that
  - Extents patch helps file sequential read/rewrite/removal
  - Multiple block allocation and delayed allocation help sequential write(file creation)

Tiobench Sequential Write Comparison With Extents



Large File Sequential I/O Comparison Using FFSB



Part C: Improving Current Ext3

- Delayed allocation without extents
  (By Badari Pulavarty, Suparna Bhattacharya)
- Allocating multiple blocks without extents
  (By Mingming Cao)
- Reduce file unlink and truncate latency
  (By Andreas Dilger)
- Increased number of subdirectories support
  (By Andreas Dilger)
- Parallel directory operations
  (By Alex Tomas)
- Finer timestamp
  (By Alex Tomas, Andreas Gruenbacher)

Delayed Allocation for Current Ext3

- Concept: deferring block allocation from the prepare write time to page flush out time
- Reserve filesystem free blocks at prepare-write time to avoid allocation failure later
- Delayed allocation for different journalling modes
  - data=writeback mode
  - data=ordered mode

Delayed Allocation for Ordering Mode

- Delayed allocation and Bufferheads
  - bufferhead is used to link a page to a disk block
  - bufferhead is also used to link the page with the related journal for ordering purpose
  - delayed allocation defers bufferhead creation for a page to writepages time, late for ordering purpose
- Proposed solution: add a ordered-like journalling mode which doesn't use bufferheads

Allocating Multiple Blocks for Current Ext3

- A simple, efficient way to allow allocating multiple blocks at a time
- Based on existing indirect block mapping, make use of block reservation
- Allocating the first block in the existing way, then allocating the rest on a best effort basis

Allocating Multiple Blocks Example

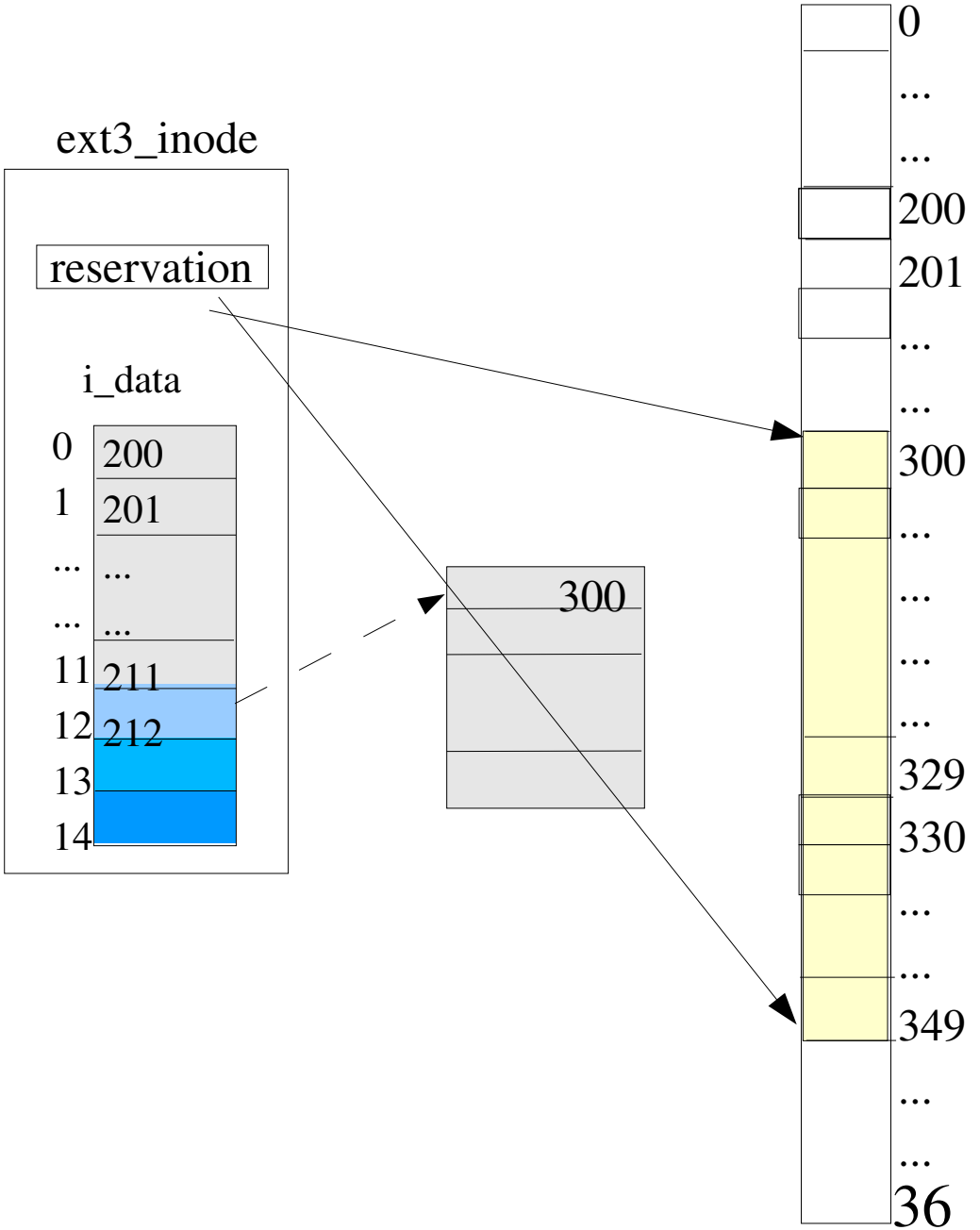
file offset 13: goal block: 300, request 50 blocks, RSV window(300,307)

RSV window extended (300,349)

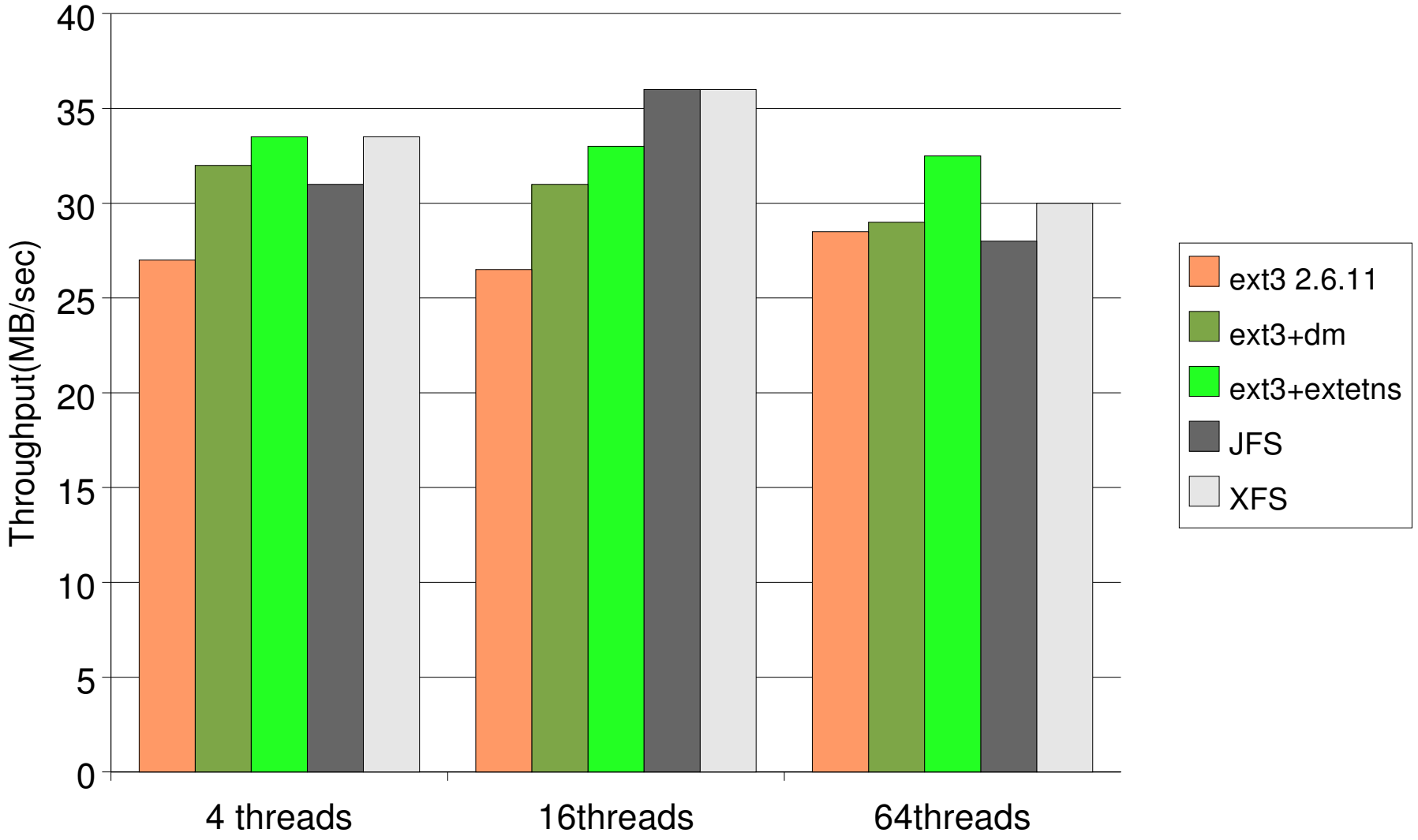
first block 300 allocated

block 301-329 allocated (30 blocks total)

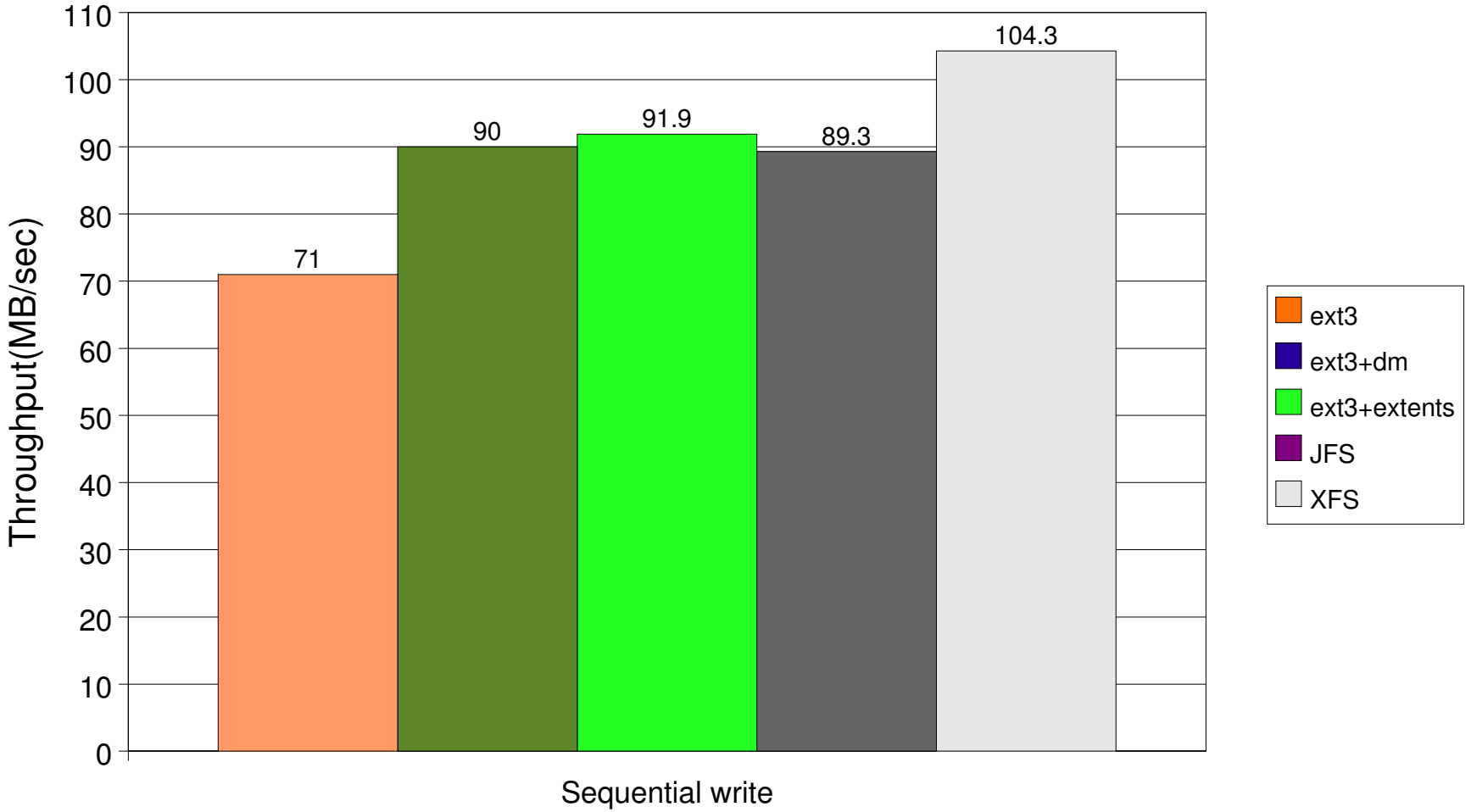
Indirect block is being updated

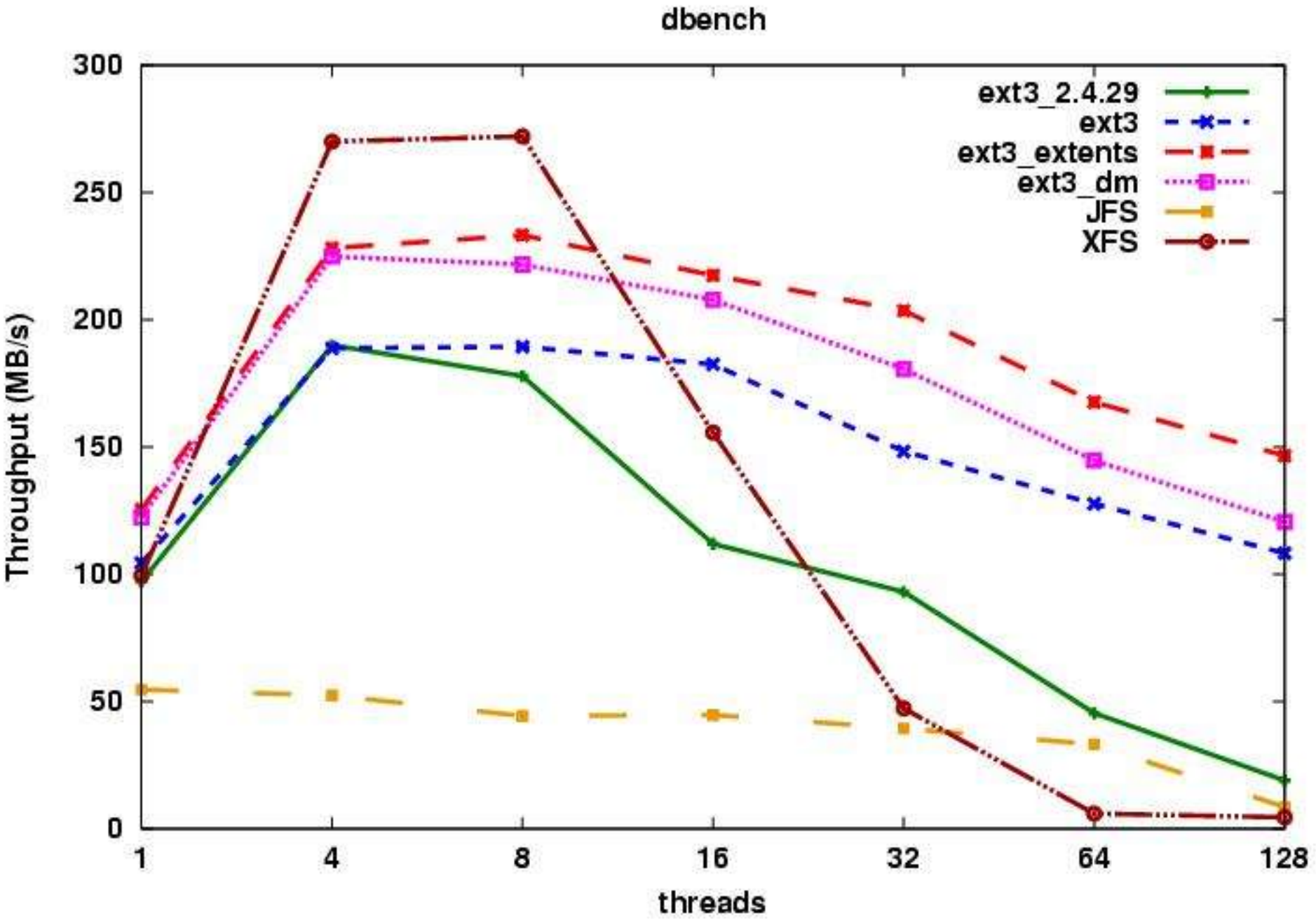


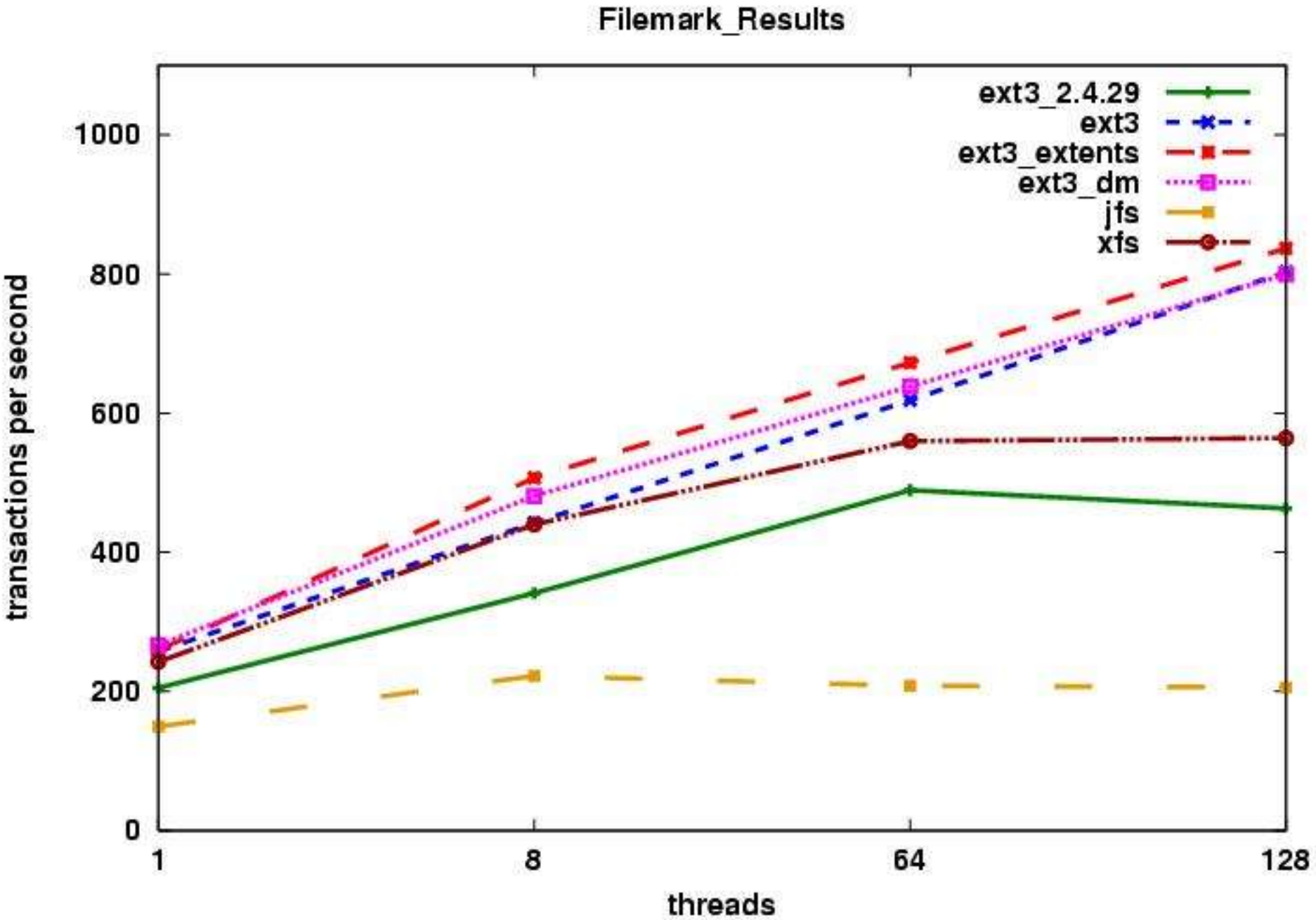
tiobench sequential write comparison with extents



Large File Sequential Write Comparison Using FFSB




dbench
Throughput (MB/s)
threads
0
50
100
150
200
250
300
1
4
8
16
32
64
128
ext3_2.4.29
ext3
ext3_extents
ext3_dm
JFS
XFS


Filemark_Results
ext3_2.4.29
ext3
ext3_extents
ext3_dm
jfs
xfs
transactions per second
0
200
400
600
800
1000
1
8
64
128
threads

Other ext3 work

- Reduce file unlink and truncate
- Increasing ext3 subdirectory limit
- Parallel directory operations
- Finer granularity time stamps for ext3

Reduce File Unlink/Truncate Latency

- Truncating a large indirect-mapped files is slow and synchronous. Root cause: there is limit to the size of a single journal transaction.
- Proposed solutions:
  - Background file unlink/trucate
  - Attempt to fit into one transaction if possible
  - Store i_disksize to expanded inode.

Increasing Ext3 Subdirectory Limit

- Each subdirectory has a hard link to its parent
- Number of subdirectories under a single directory is limited by type of inode's link count(16 bit)
- Proposed solution to overcome this limit:
  - Not counting the subdirectory limit after counter overflow, storing link count of 1 instead. (every directory start with a link count of 2)

Parallel Directory Operations

- Concurrent file operations in a single directory is serialized by per-directory semaphore
- Allow concurrently create/unlink/rename files within a single directory
  - VFS:lock individual hash entries in a directory
  - Ext3:add per-directory-leaf-block lock

Finer granularity time stamps for ext3

- Regular ext3 on-disk inode doesn't have enough space to store high-precision time stamp
- Proposed solution: store nanoseonds time stamp on expanded inode
- Concern:
  - additional dirtyings and writeout for atime/mtime/ctime updates
  - expanded inode may be filled up by extended attributes

Future work

- Continue to improve current ext3 filesystem
  - Improve features already accepted in mainline
  - Finish work in progress for current ext3 filesystem
- Moving ext3 forward: extents, and other potential future work
  - Increase 8/16TB filesystem limit (64 bit block number)
  - Extensible inode table(avoid too much inodes pre-allocation)

Questions?

Contact the authors:

Mingming Cao	cmm@us.ibm.com
Theodore Y. Ts'o	tytso@us.ibm.com
Badari Pulavarty	pbadari@us.ibm.com
Suparna Bhattacharya	suparna@in.ibm.com
Andreas Dilger	adilger@clusterfs.com
Alex Tomas	alex@clusterfs.com

sources of the paper and this presentation are at ext2.sourceforge.net

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of IBM.

IBM and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.

Lustre is a trademark of Cluster File Systems, Inc.

Unix is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

This document is provied ``AS IS," with no express or implied warranties. Use the information in this document at your own risk.