Lustre IO on 25,000 clients
CUG 2007

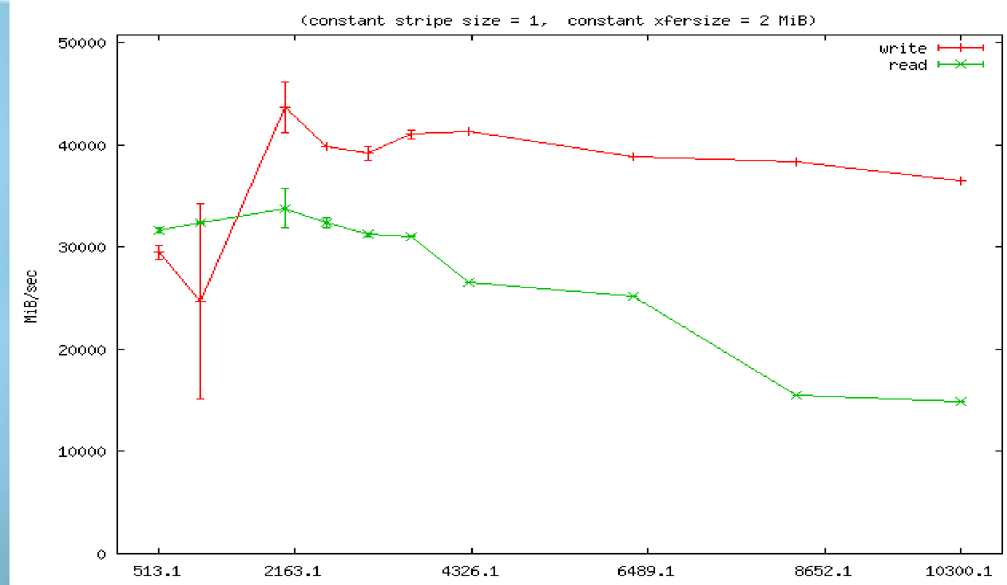
Peter J. Braam, PhD
Cluster File Systems, Inc.

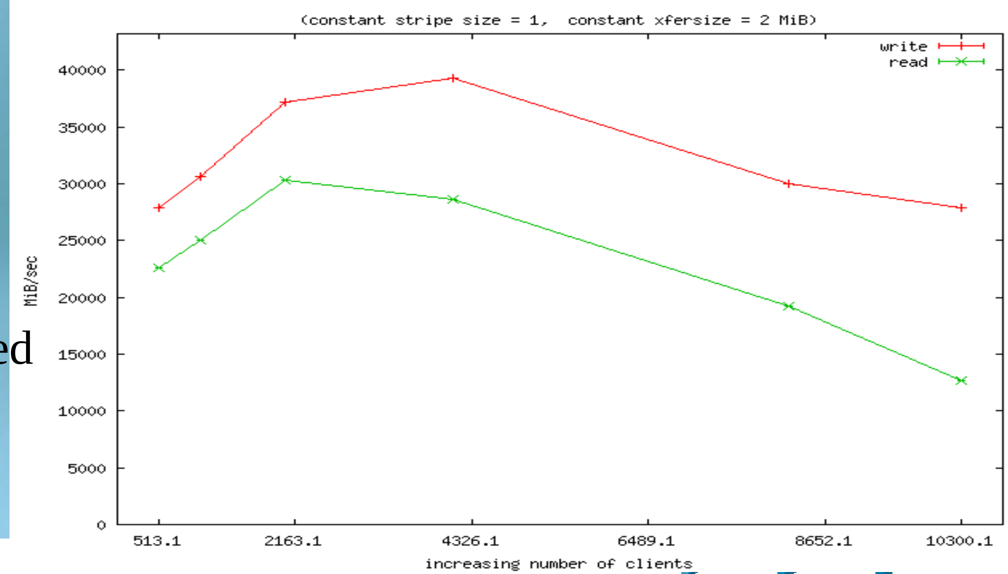
Contents

- Summary graph
- Scalability enhancements
- Dealing with small IO
- Petascale file systems
- Hardening the disk FS
- User level servers

Red Storm – a summary graph

- ~40GB/sec
- File per process (top)
- Shared file (bottom)
  - 160 wide stripe
- Scales to 10,000 clients
- Reads are too slow
  - Array misconfigured for reads
  - Too much read ahead
- Shared file too slow
  - OST's misconfigured for shared file
  - Not enough disks





Scalability

A bit of background

- **Locks allow client caching and coordinated updates**
- **Lustre metadata locks**
  - Directory data (reading directories and modifying them)
  - FID to name associations (lookup)
  - Opened files
- **Lustre file extent locks**
  - Protect extents in files
  - Single writer, multiple reader usage
- **When locks are enqueued scan for conflicting locks**
  - Send callbacks when there are conflicting locks
  - Callbacks cause cache flushes

Connection & file open scalability

- **25,000 nodes was the last straw**
  - For a few poor algorithms in the servers
- **Connection**
  - Searched a list
    - All clients connect so this is a quadratic problem
  - Lustre now has a hash for connection UUIDs
- **Locks - e.g. for file open**
  - Searched a linear list of locks to find conflicting locks
  - The structure of compatible and incompatible locks is complicated
    - Lock modes - EX, PW, PR, CW, CR, NL
    - Inode bits - Open Bit, Lookup bit, Data bit
- **We introduced a skip list mechanism**
  - They allow us to efficiently find conflicting locks

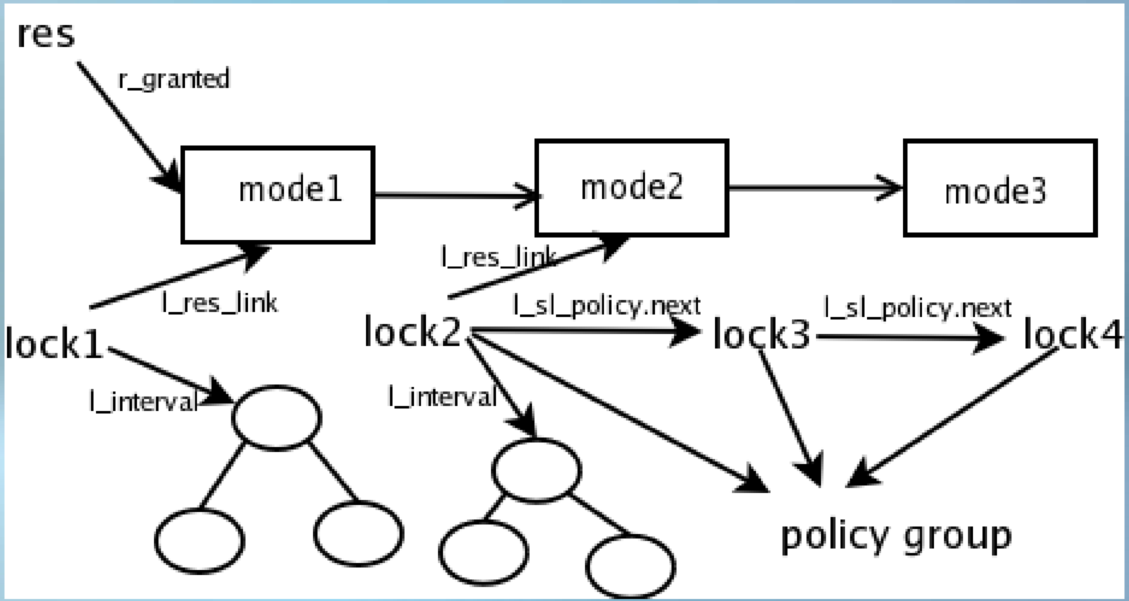
Lock mode compatibility

- **Requested vs Granted lock mode compatibility.**

	NL	CR	CW	PR	PW	EX
NL	Yes	Yes	Yes	Yes	Yes	Yes
CR	Yes	Yes	Yes	Yes	Yes	No
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	No	Yes	No	No
PW	Yes	Yes	No	No	No	No
EX	Yes	No	No	No	No	No

IO regions - conflicting locks

- **Introduce an interval tree in the extent lock handling**
  - Previously there was a list of extents that were locked
  - Now there is a tree
  - Scalable search for conflicting locks

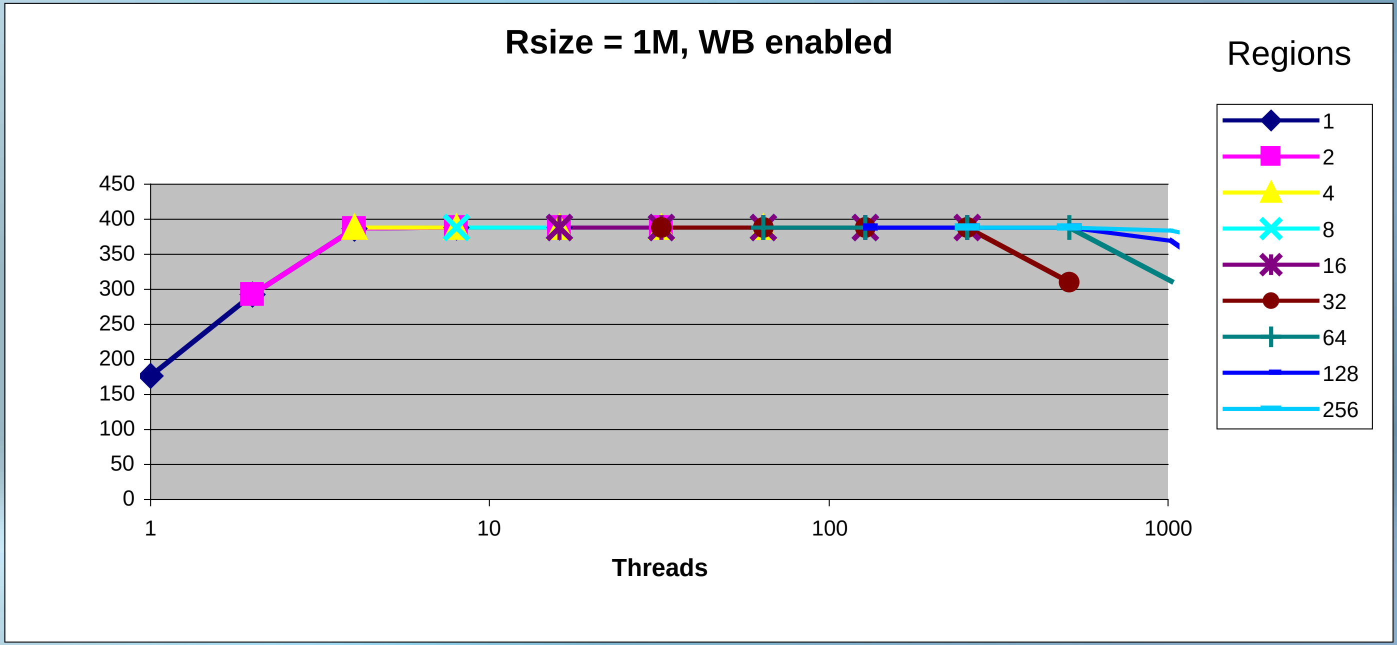


IO and locking

- Stripe locking
  - Change from
    - Lock all stripe extents, do all IO in parallel, unlock all
  - To
    - For all stripes in parallel: lock, do IO, unlock
  - Holding locks from multiple servers
    - Can lead to cascading recovery events on many servers
    - Is necessary for truncate and O_APPEND writes
- Disallow client locks under contention
  - When an extent in a file sees concurrent access
    - Ask the client to write through to the server
  - This eliminates callback traffic and cache flushes

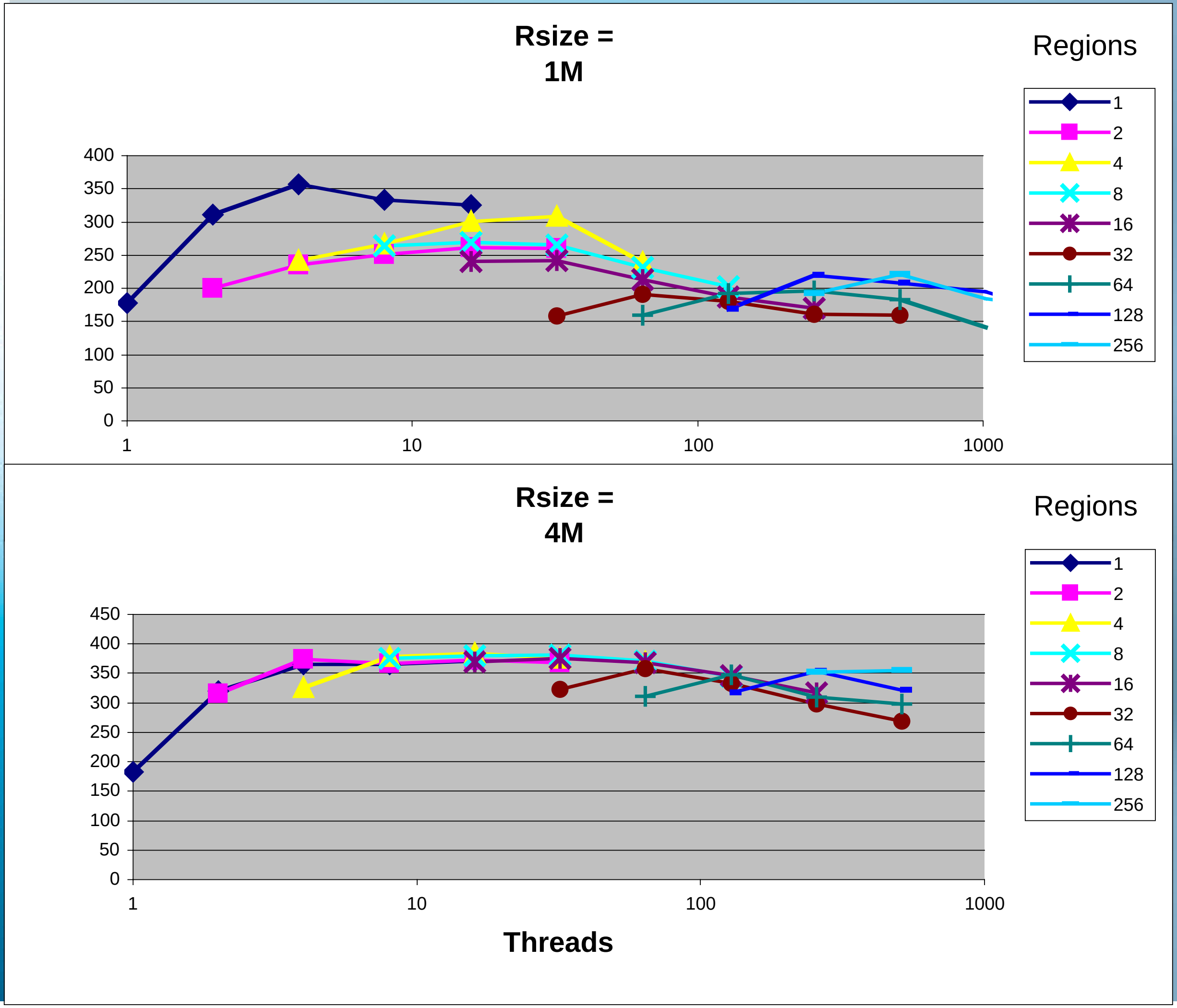
Disk arrays

- **The IO is typically done against a DDN 9500 array**
  - We don't understand well how to do IO with it
- **Some instability for high region counts**



Reads - even more complicated

- Reading from the array - cannot find a sweet spot

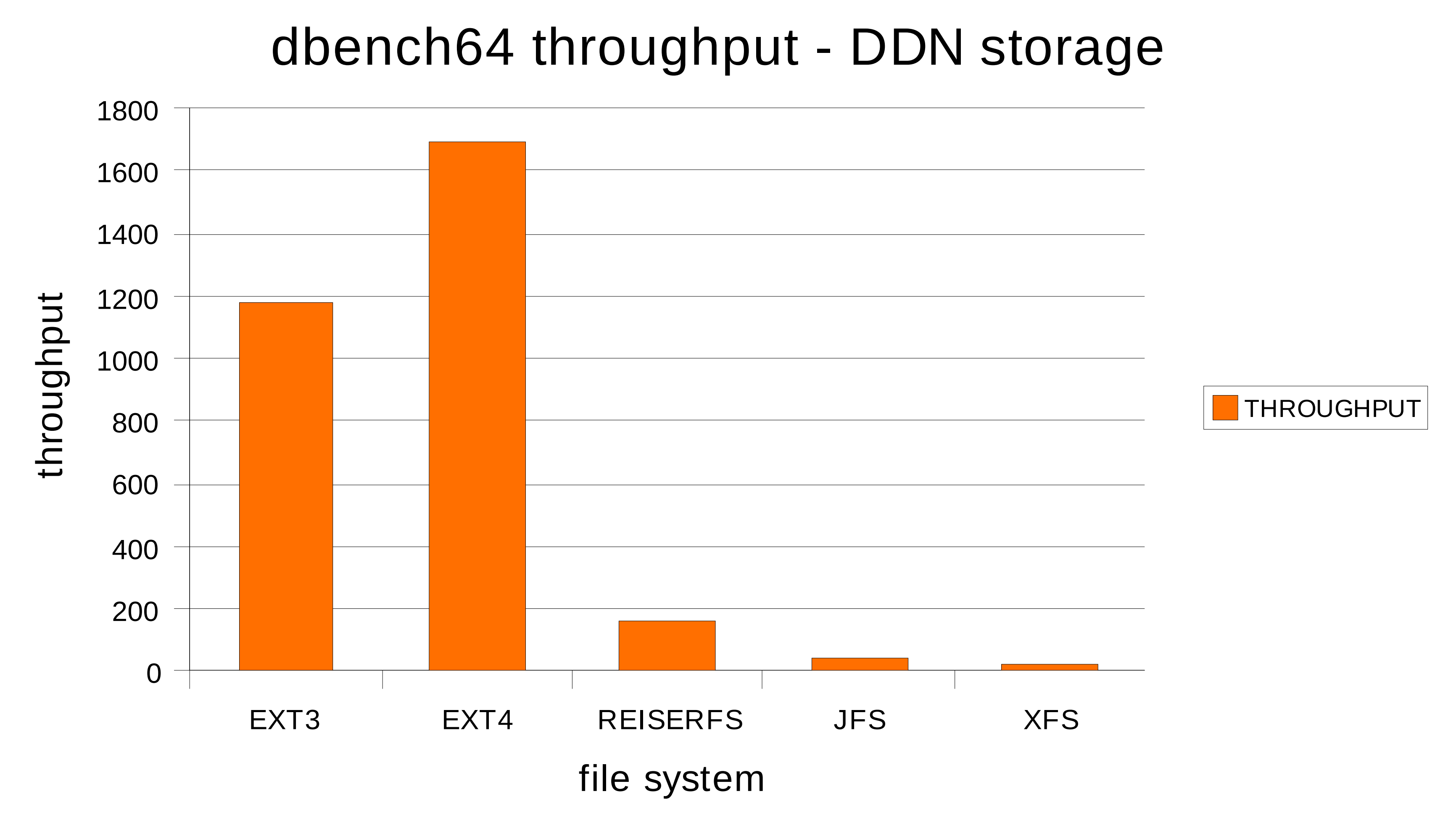


Dealing with Small IO

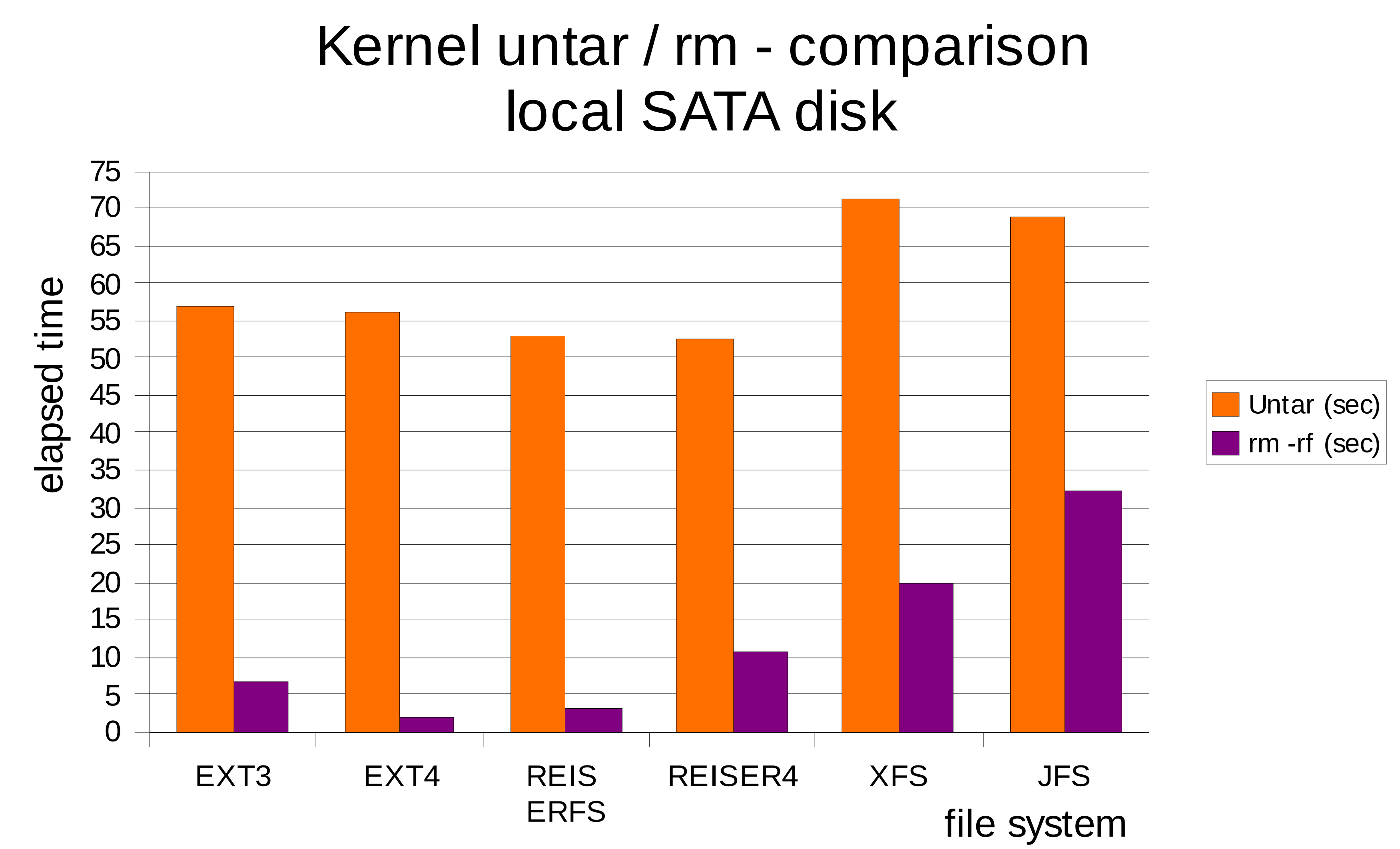
New disk allocator

- **Block allocation policies**
  - Write a little (e.g. <64K) before small offset (e.g. 64K)
    - Place the write in a “small file” area on the disk
  - Keep such small writes together
  - Large writes are aligned in 1-4MB chunks
  - Writes at significant offset are logically and physically aligned
- **Outcome – smoking performance**
  - It appears that this is the crux for small file performance
  - The secret of Reiser was to write things close together
- **Typical use cases**
  - liblustre
  - small file performance

New allocator dbench64 throughput



kernel untar / remove with new allocator



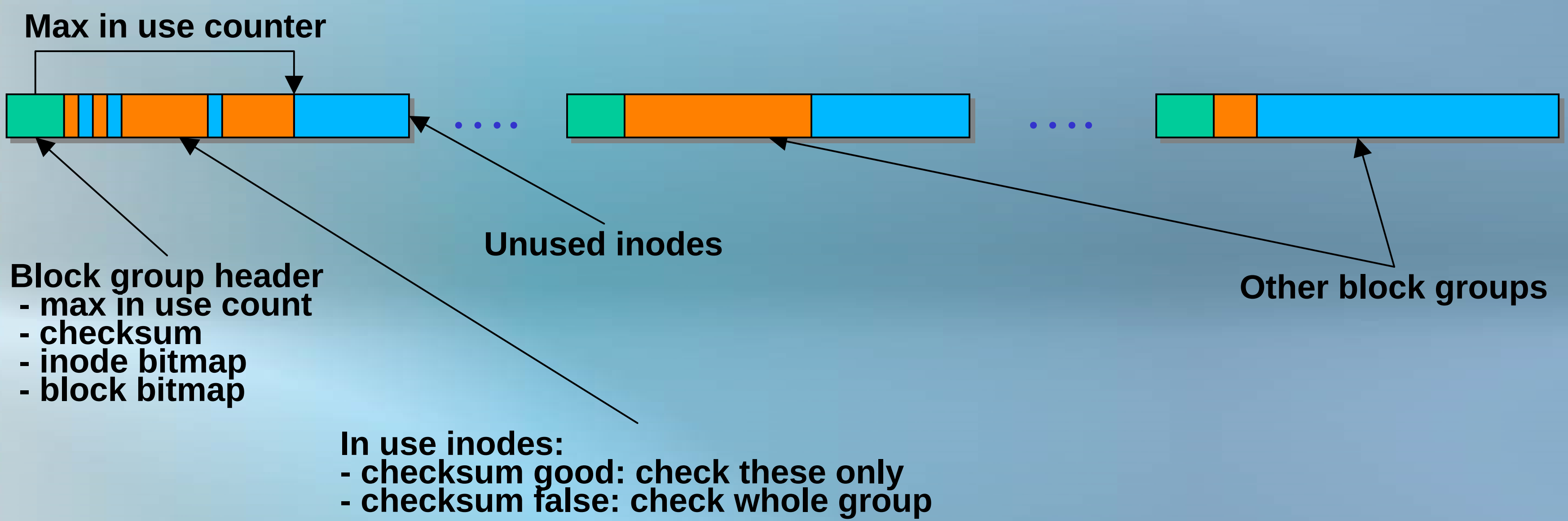
OSS writeback cache

- **Some jobs send very small IO's to the disk arrays**
  - aggregation is important
    - Lustre so far does no caching on the OSS
    - Liblustre clients have no cache (Linux clients do)
- **Lustre OSS servers will get a cache**

Scaling & Killing FSCK

Fast FSCK & Format

- **FSCK has changed**
  - Previously fsck scanned all inodes
  - Now only inodes that are possibly in use
- **The most interesting part of this is a checksum**
  - The checksum indicates if the metadata that follows is consistent
  - If it is the counter can be used to check up to the maximum inode
- **Speedups of 4x to 10x**
  - Good, but fsck needs to disappear completely, it doesn't scale



Extremely large File Systems

- **Do you math – 1PB/fs, 500GB disks**
  - 1 disk blows quickly
  - Estimates vary –
    - mfr: every 12 days,
    - Pessimists - 10 hours
  - Double failure 2 months – 20 years
  - We have interesting practical experiences here …
- **Key features**
  - No limits: #files, #capacity
  - Integrity: FS should be usable after disastrous events
  - Harden: detect and repair corruption where reasonable
- **Port ZFS approach**
  - ZFS seems to have correct design
  - Will probably be ported to Linux
  - The port will probably take long
- **CFS “iron” ext4**
  - University of Wisconsin first steps
  - Checksum much of the data
    - Replicate metadata
    - Detect and repair corruption
  - Handle relational corruption
    - Accidental re-ordering of writes
- **CFS approach**
  - A sequence of small fixes
  - starting now
  - each with benefits

FS for 1PF system

- **Required 1TB/sec, FS will be many PBs**
- **CEA has servers: 2GB/sec**
  - Most promising solution: 500 OSS servers of this type
- **Lustre**
  - Already has installations with ~500 servers
  - Already has installations with ~2GB/sec servers
  - Already handling 25,000 clients on one FS in production today
- **10TB/sec requires some scalability improvements**

User level servers

- **The Solaris OSS port layers the OSS server on ZFS**
  - The server will be a user space server
  - It will not use any custom interfaces to the file system
- **On Linux we are exploring the same**
  - Layer on ext4
  - Preparations
    - Give ext4 / Linux the capability of concurrent writes to one file
    - Improve the direct IO / VM cache relationship
- **Evaluate the performance**
  - For this we have written a simple server simulation program
    - pios – Parallel IO Simulator
- **High likelihood of success**
  - If confirmed the OSS will become a user space server
  - If ZFS is good, we can benefit from it, or have options

Thank you.