Towards a Highly Adaptable Filesystem Framework for Linux

Suparna Bhattacharya, LTC, IBM India Systems and Technology Lab
Dilma Da Silva, IBM T.J. Watson Research Center
Orran Krieger, IBM T.J. Watson Research Center

Ottawa Linux Symposium, July 2006

Linux has a very flexible file system framework...

- Powerful VFS
  - Object oriented framework
    - Abstractions for superblock, inodes etc
    - Variety of file system types
  - Common helpers for data caching, libfs etc
- Rising no. of general purpose file systems
  - Different “sweet-spot” usage patterns
  - Advantage of parallel innovation
- Each filesystem is evolving independently
  - Advances in storage, protocols, appl reqmts
  - Change with compatibility
    - Adding options for new features (tune2fs, chattr)
    - Occasionally break off next gen as a new filesystem

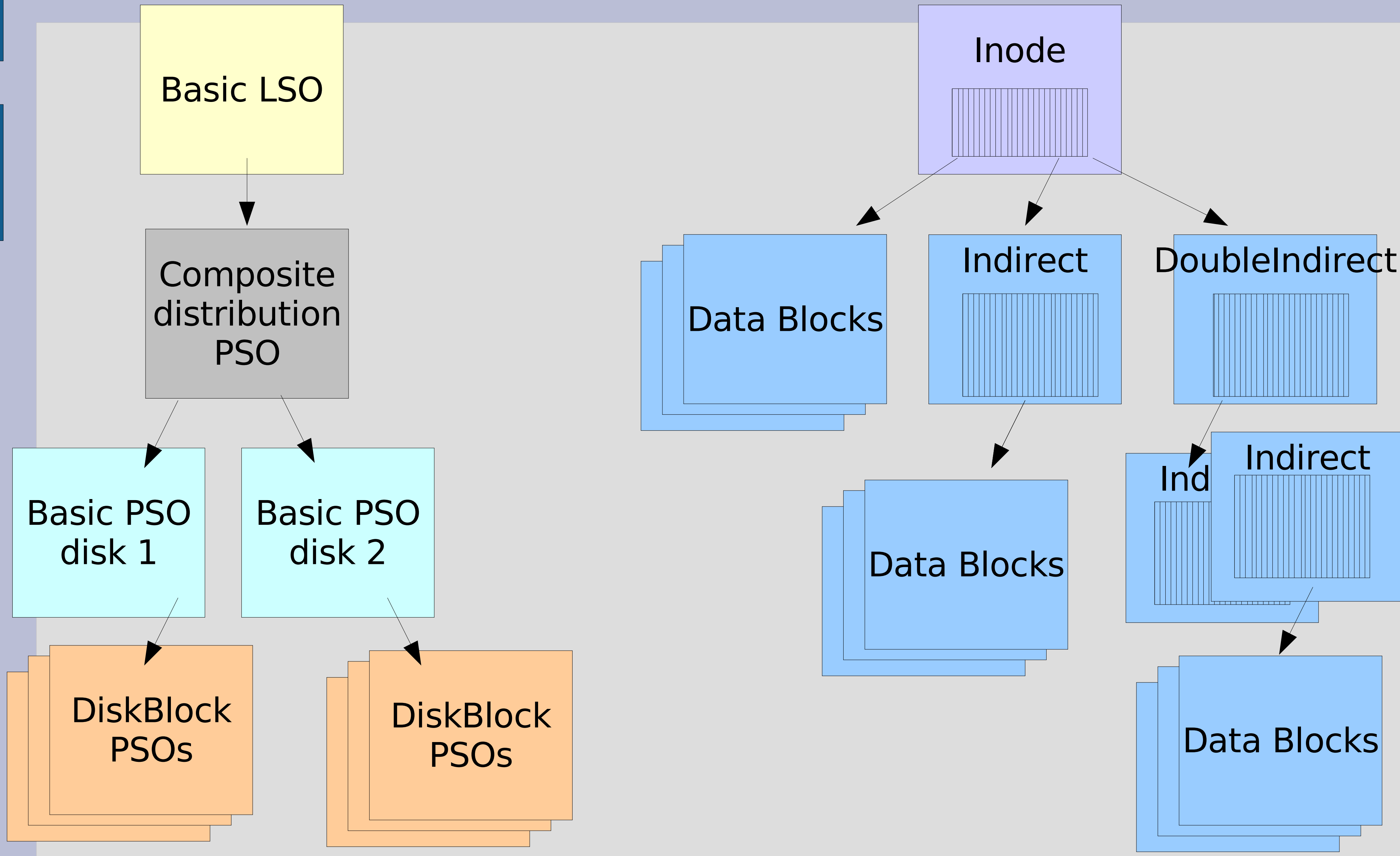
... but only a limited degree of adaptability

- Problem of switching on-disk formats
  - Virtual lock-in to a choice
  - High lead time to adoption of format enhancements
  - Compounded on distribution across multiple disks
- Fragmentation from a user perspective
  - Choosing the right Linux filesystem charts
  - Low reuse of low-level building blocks
- One file layout does not fit all
  - Small vs large, streaming vs slow growing vs random, dense vs sparse, read-mostly vs r/w

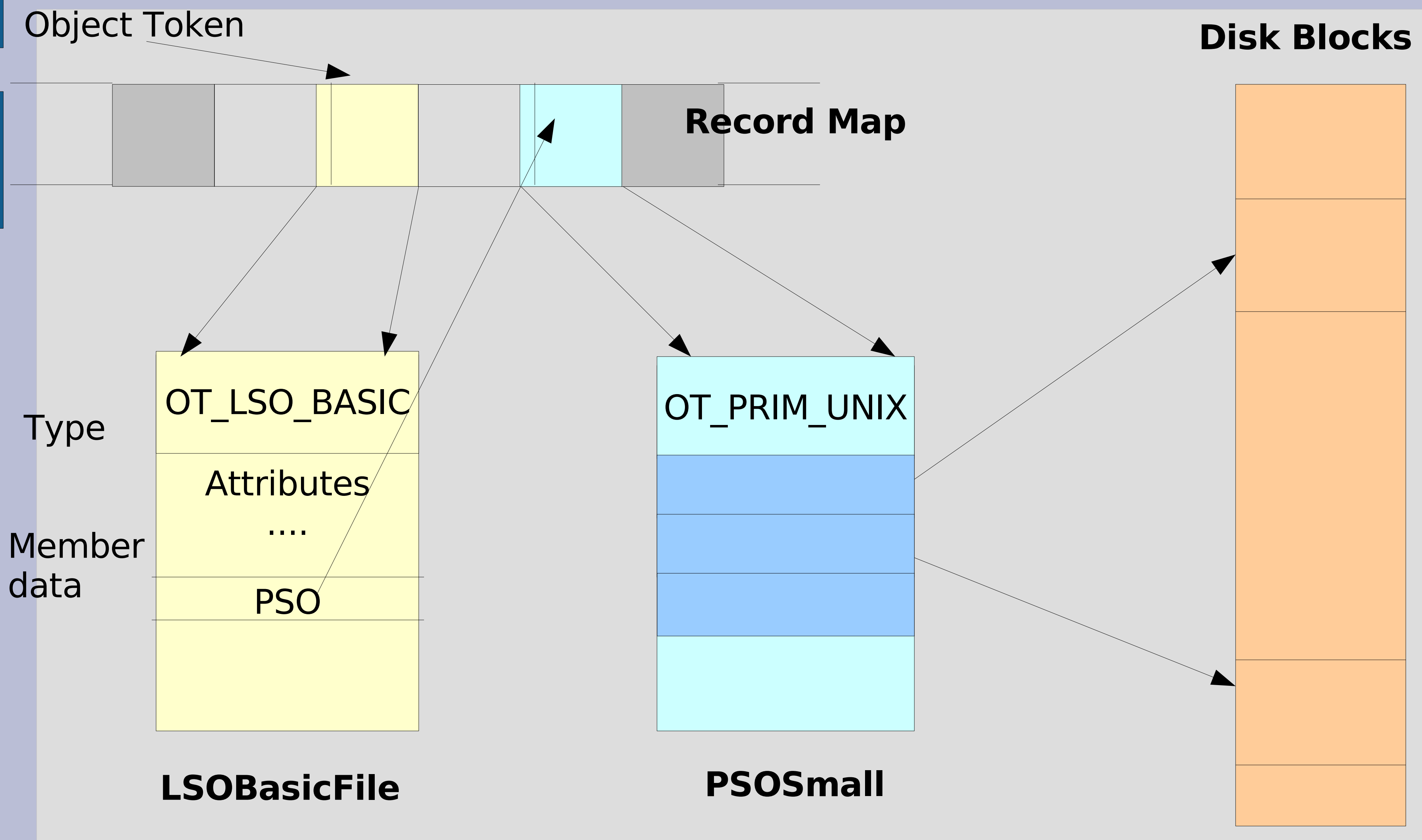
What if we push this flexibility further all the way ?

- All resources represented by storage objects (self describing)
  - e.g., No disk region for inodes
- Separation of concerns into re-usable objects
  - e.g., block mapping (physical) separated from inode (logical)
- Recursive use of OO design
  - Object stored within other object
  - Objects partition work to other objects
- Could enable
  - Per-file-element level adaptability
  - Addition of new formats while retaining old ones
  - Reuse of low-level layout structures

How is this different from what we do today ?



How meta-data may be stored recursively



Intuitively the overhead of flexibility in mapping ext2 is small

- Cached data handled mostly by VFS
- Cached meta data access has small extra overhead.
- IO for meta data should be about the same

While enabling new layout implementations to be added with ease

- Extent based
- Write near disk head
- File in inode
- Inode in directory
- Replication across disks
- Redundancy on per-file, per-directory basis

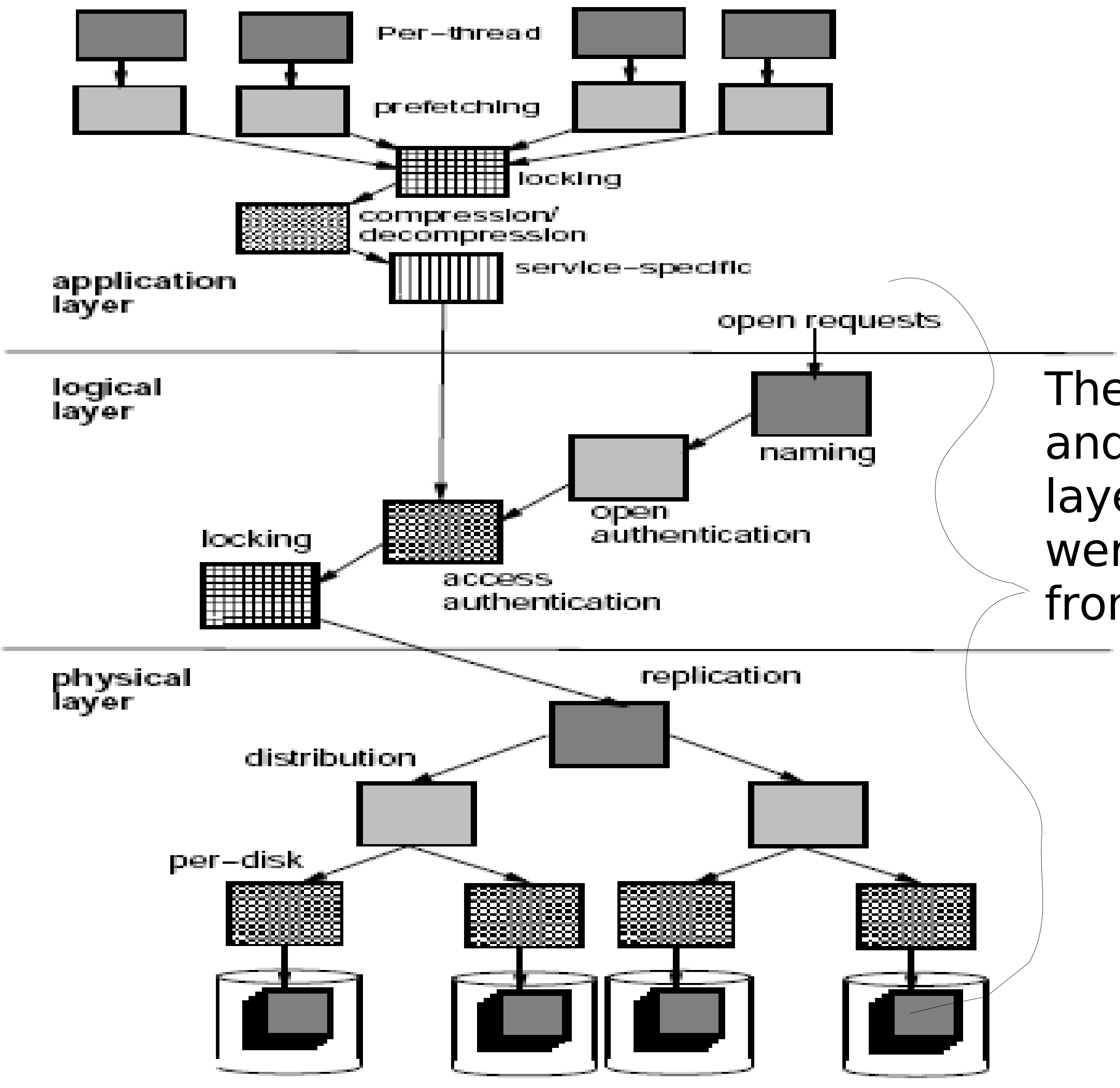
A few ways in which flexibility may be exploited

- Simple hints, different defaults
- HPC, Database, Application specific formats
- Old data can retain its old format
- Natural transition/decision points
  - File growth, aging/hot, multiple links ...
- Agents that determine workload characteristics, usage patterns, CPO

There is a framework to explore such an extreme approach to flexibility

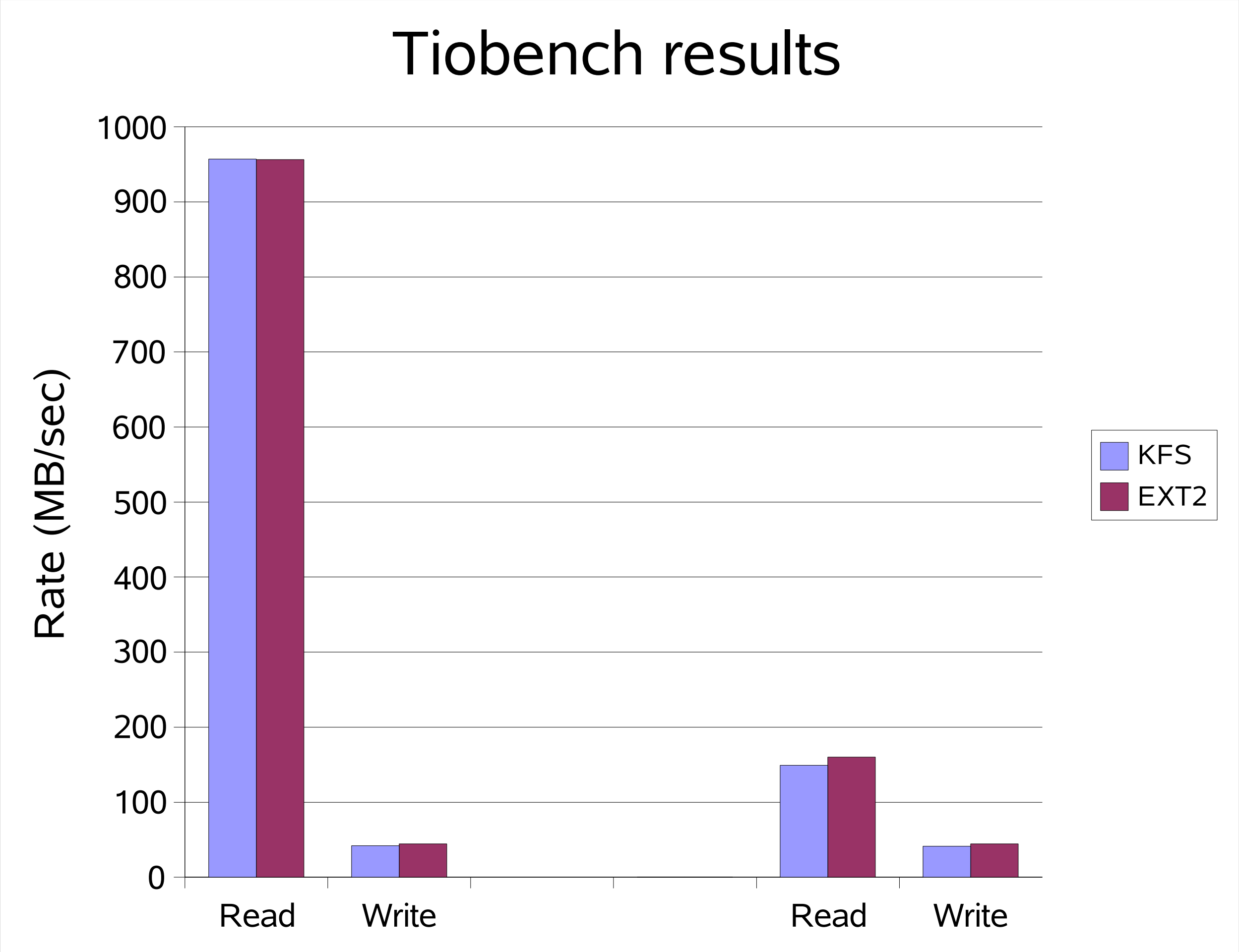
- HFS (Hurricane Filesystem) [Krieger, 1994]
  - A research filesystem designed to support:
    - fine grained flexibility
    - a wide variety of file structures and policies
    - dynamic changes in representation
  - Intended for large-scale SMP running diverse loads
    - Explored flexibility to maximize perf & scalability
    - Object-oriented building block approach
    - Achieved with low processing and I/O overhead
  - Led to the evolution of Tornado and K42 OS
- KFS (K42 filesystem)
  - Implemented the ideas of HFS in a Linux-compatible OS

A Full Example of an HFS file : 3 layer architecture



The logical and physical layers of KFS were derived from HFS

Initial results on Linux 2.6 with tiobench on unoptimized KFS



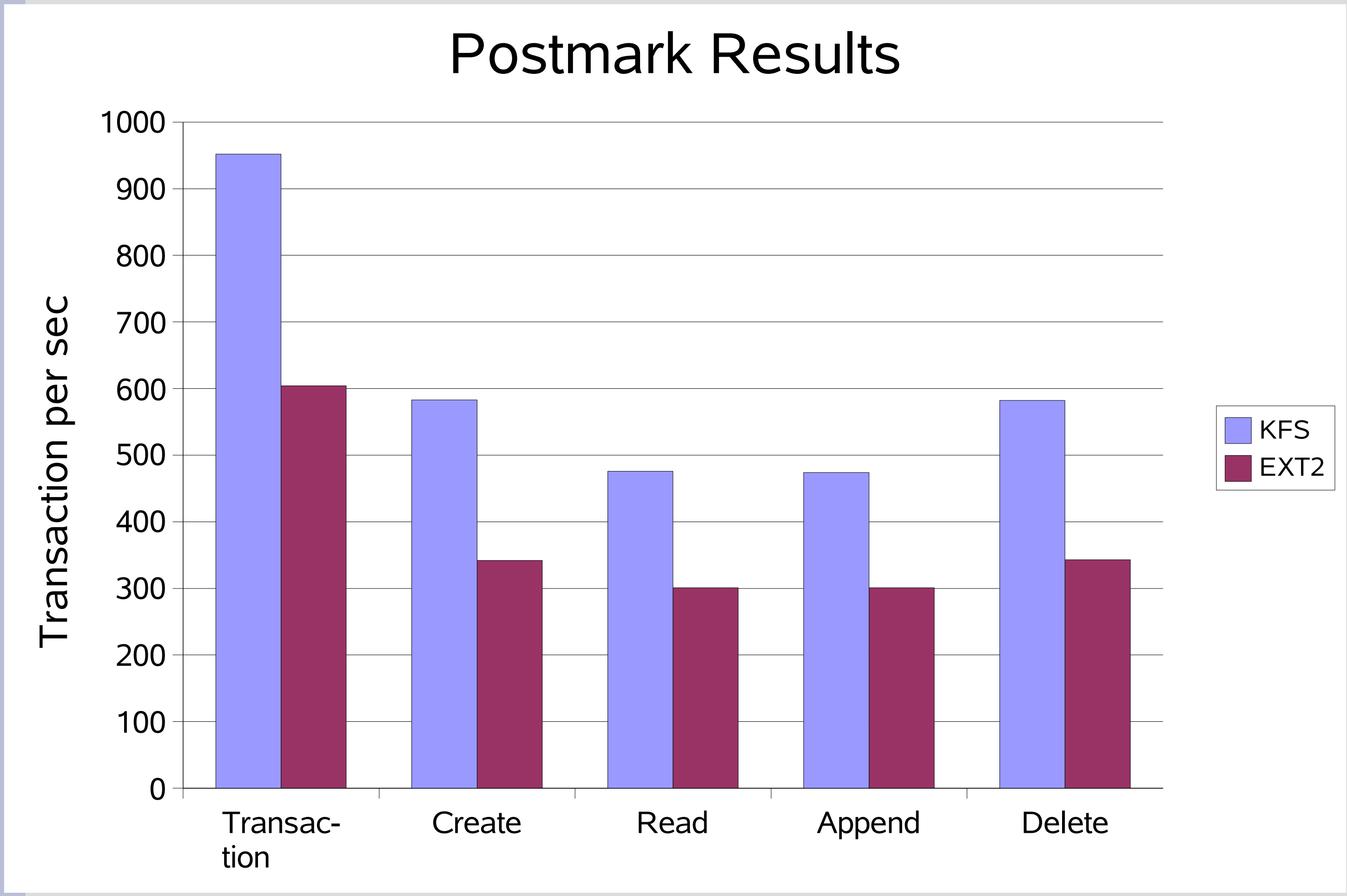
1 thread 4 threads

Avg of 3 runs. (Each run followed by reboot and format).

File size : 100MB per thread

Sequential Write

Initial results on Linux 2.6 with Postmark on unoptimized KFS



Base No of files : 20000

Transactions : 100000

Sub-directories : 400

(Avg of 3 runs and each run is followed by a reboot and remount)

The way forward: Pursue multiple aspects to adaptable Linux filesystems

- Full embedding of an existing filesystem format to evaluate if KFS is an appropriate starting point
- Demonstrate 2 or more embedded filesystems with negligible performance overhead
  - Explore building block sharing across filesystems
- Switch formats to suit access patterns
  - Add new formats, work with multiple formats
    - Discard experimental layout changes
  - Continuation inodes (ArjanV, Val Henson)
- Evaluate alternate layouts and policies
  - Collocation of file data and meta-data
  - Other ideas from file system workshop
- Study reliability aspects
  - Adaptability in the file system checker

In Summary

- The very flexibility that has been the strength of the Linux VFS, appears to have exposed its own set of problems over time
- We hypothetize that pushing the flexibility all the way through to on-disk layouts could help us address these problems in the long run
- While we are exploring KFS as one possible starting point towards such a framework, we have yet to understand whether it is an appropriate approach
- There are many open questions to explore, and we welcome community involvement in this exciting endeavour

Our Thanks to ...

- KFS Contributors
  - Livio Soares (KFS on K42 & Linux 2.4)
  - Ajit Burad & Tarun Mittal (KFS on Linux 2.6)
- And to
  - Suzuki K.P., V Srivatsa, Mingming Cao, Dave Kleikamp, Stephen Tweedie, Theodore Tso, Andreas Dilger, Val Henson, Paul McKenney, H. Peter Anvin, Chris Mason, Arjan Van De Ven, Christoph Hellwig, Mel Gorman

Availability: http://www.research.ibm.com/k42

Legal Statement

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

IBM and DB2 are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or 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.

The benchmarks discussed in this presentation were conducted for research purposes only, under laboratory conditions. Results will not be realised in all computing environments

BACKUP

KFS is based on composition & specialization of building blocks

- Physical Server Object (PSO)
  - PSODiskBlock
  - PSOSmall, PSOSmallMeta
  - PSOExtent
  - PSOReplicated, PSOStriped
- Logical Server Object (LSO)
  - LSOBasicFile,
  - LSOBasicDir,
  - LSOEmbDir
  - LSOBasicSymLink

A variety of file system element types may be defined

```
OT_PRIM_UNIX = 2,   /* primitive unix like file <- replace this */
OT_PRIM_UNIX_META = 3,   /* primitive unix like file for meta-data */
OT_STRIPED = 4,     /* striped  file <- replace this */
OT_BASIC_RW = 5,       // A basic per/disk read/write object
OT_BASIC_SPARSE = 6,   // A basic per/disk sparse object
OT_BASIC_DENSE = 7,    // A basic per/disk dense object
OT_NM_SMALL = 8,       // A non-mapped small object
OT_RECORD_MAP = 9,     // A non-mapped record store object
OT_COMP_STR = 10,      // A composite striped object
OT_COMP_REP = 11,      // A composite replicated object
OT_COMP_DIS = 12,
OT_COMP_CHCK = 13,
OT_COMP_PAR = 14,
OT_LSO_BASIC = 16,
OT_LSO_BASIC_DIR = 17,
OT_LSO_BASIC_LNK = 18,
OT_DISK_BLOCK = 20,    // Low-level disk-based object
OT_BASIC_EXTENT = 21,
OT_SYMLINK_EXT = 22,
OT_LSO_DIR_EMB = 23
```

Example Scenario for adaptation to access patterns

- Start with a PSORecEmb which stores data in the PSO record itself
- As file grows, switch to a PSOSmall which employs direct-block mapping
- As file grows further, depending on contiguity of blocks, add a PSO sub-obj that is either PSOExtent (extent maps) or PSOBasicRW (indirect blocks) or PSOSparse (sparse maps) or PSOPreallocSeq/PSOPreallocRan (preallocated)
- As file grows to require 64 bit relative block number, add a PSO64 sub-obj of the corresponding type (or use a distribution PSO).

Example Collocated Meta-data

- 64 bit Object ID = recmap id + rec id
- Local record maps vs global record map
- LSODirEmb embeds records within the directory

KFS for Linux 2.6 is currently under stabilization

- KFS on Linux 2.4 [Silva, Soares, Krieger]
  - Proof of concept
- KFS on Linux 2.6
  - Under stabilization and optimization by Ajit Burad and Tarun Mittal
  - Results will be made available at http://k42.ozlabs.org/Wiki/KfsExperiments