JG|U

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

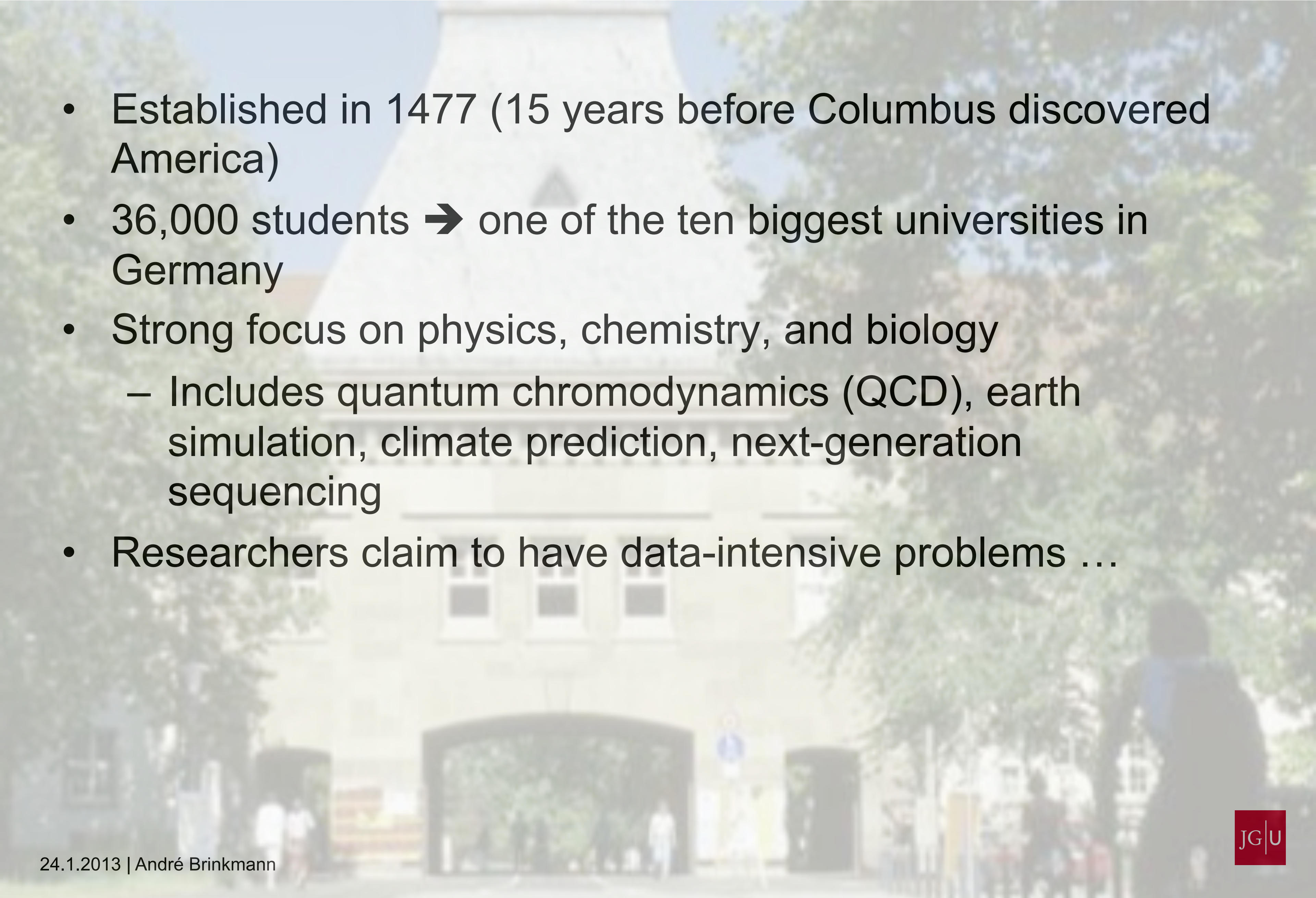
HPC Storage: Challenges and Opportunities of new Storage Technologies

André Brinkmann
Zentrum für Datenverarbeitung
Johannes Gutenberg-Universität Mai

Agenda

- Challenges for scientific storage environments
- Checkpointing
- Metadata Management
- Application Hints
  - How to extend POSIX Advices?
  - Applying in-kernel scripting engines to support application specific data layouts

Johannes Gutenberg University Mainz



- Established in 1477 (15 years before Columbus discovered America)
- 36,000 students ➔ one of the ten biggest universities in Germany
- Strong focus on physics, chemistry, and biology
  - Includes quantum chromodynamics (QCD), earth simulation, climate prediction, next-generation sequencing
- Researchers claim to have data-intensive problems …

File Systems

- Disk drives are used to store huge amounts of data

Files are logical resources

- Differentiation of logical blocks of a file and physical blocks on storage media ➔ Just one interface to storage
- Parallel file systems support parallel applications
  - All nodes may be accessing the same files at the same time, concurrently reading and writing
  - Data for a single file is striped across multiple storage nodes to provide scalable performance to individual files
- Do we need this kind of support and what are the associated costs?

What are the challenges?

- Important Scenarios
  - Checkpointing/restart with large I/O requests
  - Extreme file creation rates
  - Small block random I/O to a single file
  - Multiple data streams with large data blocks working in full duplex mode

H. Newman: What is HPCS and How Does Impact I/O, May 19th, 2009

What are the challenges?

- Important Scenarios
  - Checkpointing/restart with large I/O requests
  - Extreme file creation rates
  - Small block random I/O to a single file
  - Multiple data streams with large data blocks working in full duplex mode
  - Stage-in of data to 1.000.000 cores

Metadata - Coordination

H. Newman: What is HPCS and How Does Impact I/O, May 19th, 2009

Checkpointing I

- Defensive I/O required to overcome high failure rates of Exascale systems
- Checkpointing seems to put highest pressure on storage subsystem
  - Nodes have to write back 64 Pbyte in minutes or even seconds !!
  - 64 Pbyte is also the size of the tape-archive at the German Climate Research Center (DKRZ)
- Can a file system help or is it a burden?

Checkpointing II

- Systems like the checkpointing file system PLFS indicate that parallel file systems do not work well with checkpointing
  - Rearrange patterns (N-N, N-1 segmented, N-1 strided) so that they fit the underlying file system
  - Works as layer above parallel file system

J. Bent, G. A. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, M. Wingate: PLFS: a checkpoint filesystem for parallel applications. SC 2009

Checkpointing III

- SCR: The Scalable Checkpoint/Restart Library
  - Do not use a file system to checkpoint data
  - Assign peers to do the job in main memory
- SCR is not perfect, but it starts with the correct idea:
  - Do not (miss-)use sophisticated file systems to do a dumb man’s job
- Thoughts beyond SCR / generalization:
  - You only have to assign free storage space to each checkpoint
  - This only has to be done once at start-up and is the task of the Resource Management System (RMS)
  - Can be combined with upper layers like plfs
  - Checkpointing is only **about static co-ordination!**

A. Moody: Overview of the Scalable Checkpoint / Restart (SCR) Library, 2009

Metadata I

“95% of I/O time was spent performing metadata operations”[1]

- Why the 🌩 do we need file system metadata for scientific applications?

---

[1] P. Carns: HEC I/O Measurement and Understanding Panel Presentation. HEC FSIO 2011 Workshop

Metadata II

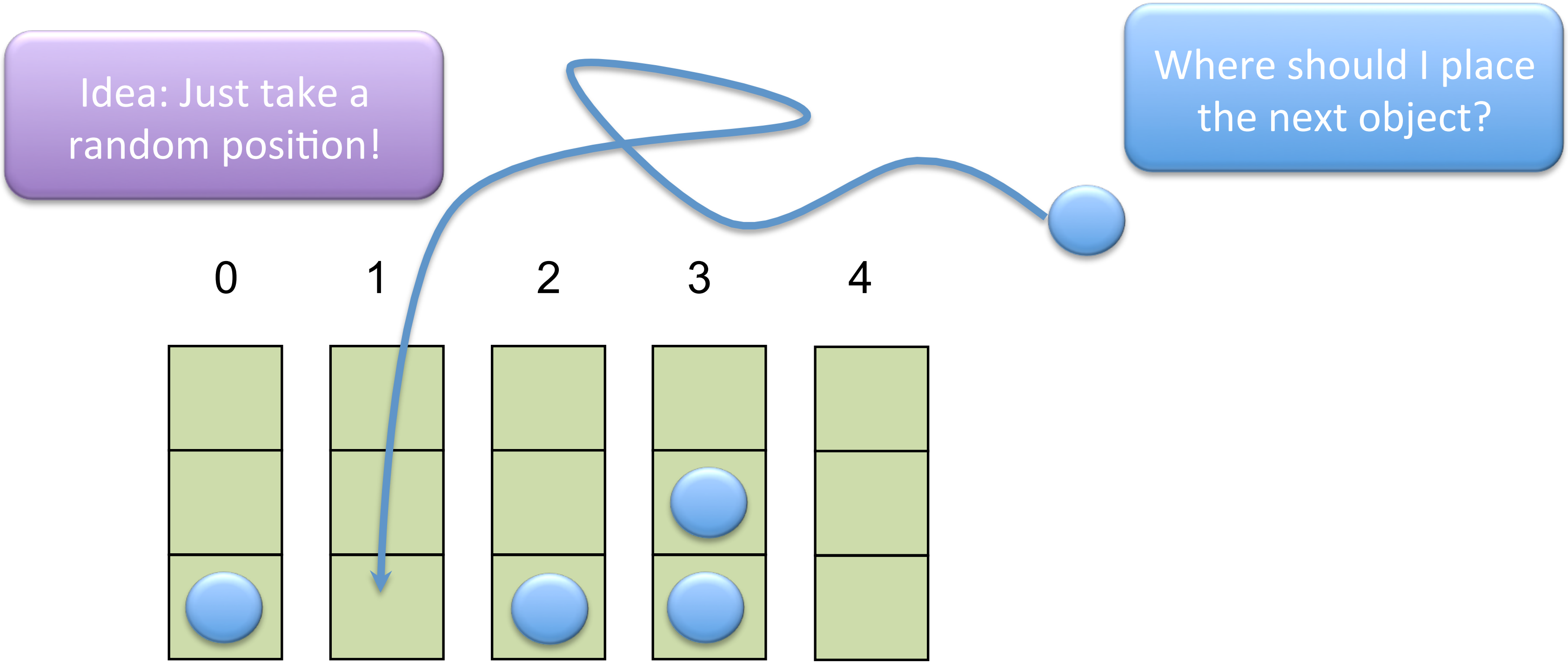
- File systems use metadata to
  - grant access permissions
  - store data layout
  - organize data in directories
  - …

- Directories seem to be the non-scalable data structure
- It is also their fault that it is incredible difficult to create millions of files per second!

- Do we need such centralized structures for exascale computing?

Metadata III

- No, because …
  - … we can organize data in objects, not files
  - … objects can be assigned to disks based on pseudo-randomized data distribution functions



Idea: Just take a random position!

Where should I place the next object?

A. Miranda, S. Effert, Y. Kang, E. Miller, A. Brinkmann, T. Cortes: Reliable and Randomized Data Distribution Strategies for Large Scale Storage Systems, HiPC 2011

Metadata III

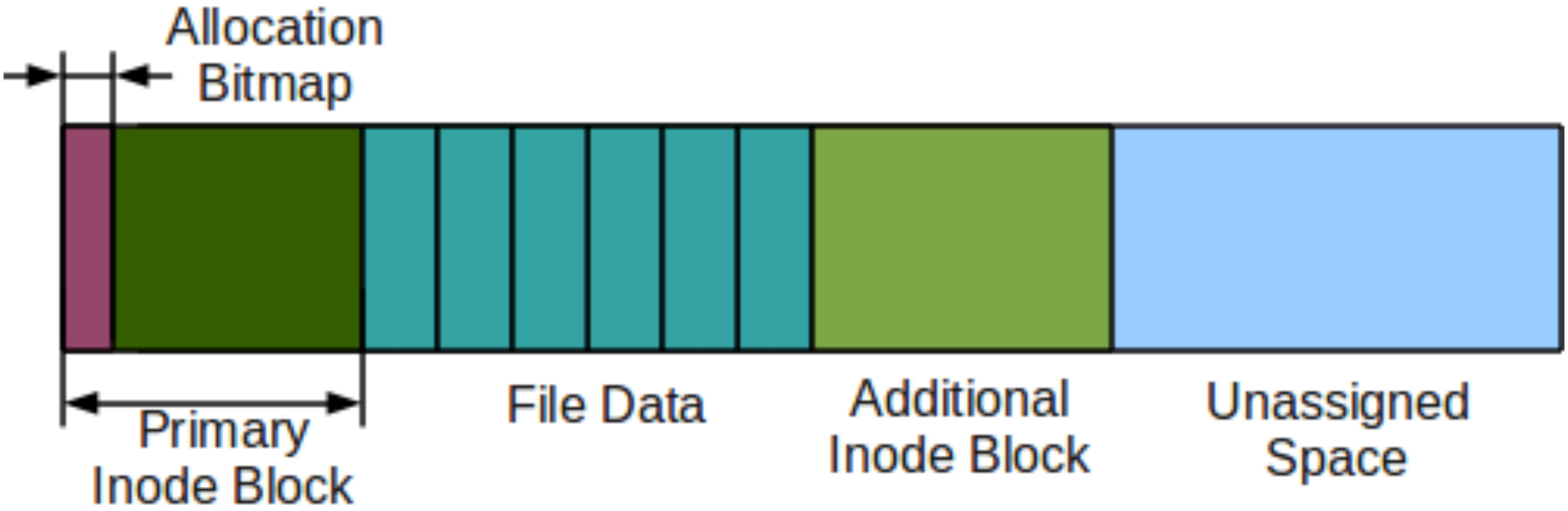
- No, because …
  - … we can organize data in objects, not files
  - … objects can be assigned to disks based on pseudo-randomized data distribution functions
- Location and distribution can be easily calculated by each client / optimal adaptivity / directory support

A. Miranda, S. Effert, Y. Kang, E. Miller, A. Brinkmann, T. Cortes: Reliable and Randomized Data Distribution Strategies for Large Scale Storage Systems, HiPC 2011

Hash-Based Metadata Placement for Local File Systems

- DLFS aims to achieve one-access lookups using hash based metadata placement
- In Linux, file system functions are called from the Virtual File System, which uses path components
  - Modifications of the VFS Layer necessary
  - If original request contains a relative path, then full path needs to be reconstructed before calling file system
- Hash buckets primarily store metadata while big file spaces primarily store data of big files

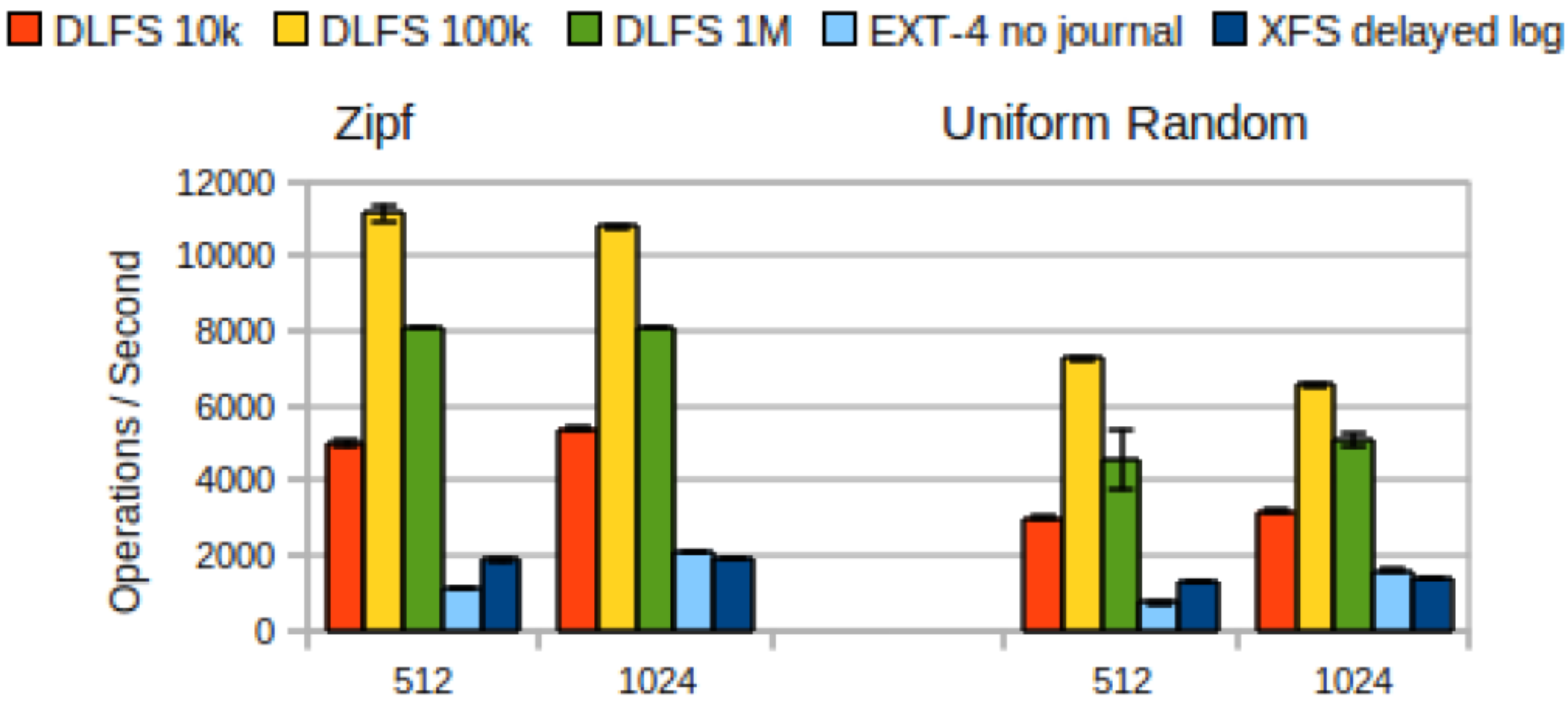
Hash-bucket Design



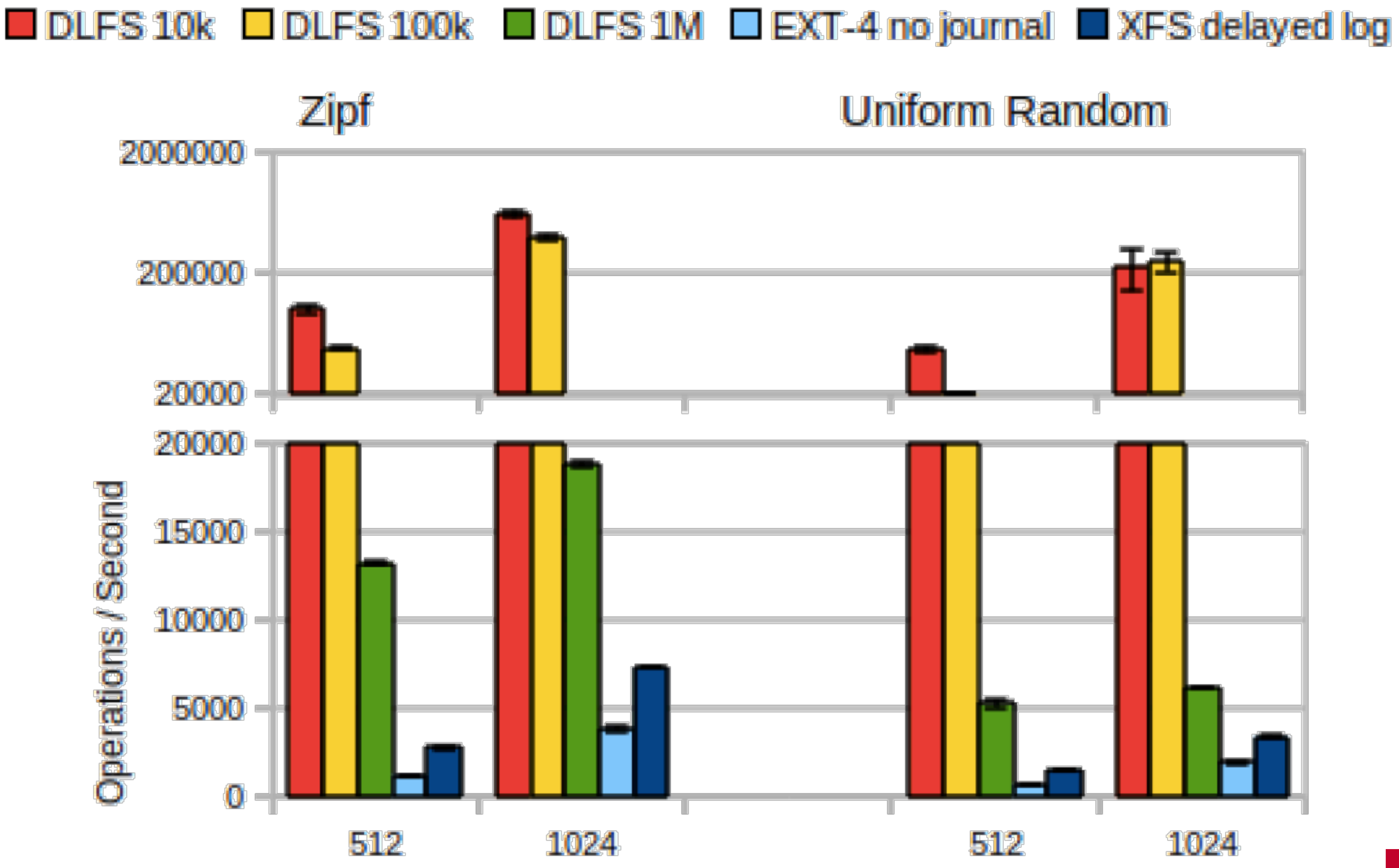
Retaining Access Permissions

- Permissions are typically resolved traversing each path-component
- Number of accesses linear in the number of traversed directories ➔ Contradicts DLFS design goals
- DLFS Approach:
  - Use reachability sets, which describe access permissions
  - Every inode inherits reachability set of its parent directory when it is created
  - Changes of reachability set of parent directory have to be recursively forwarded to all children
- Empirical Evaluation at BSC file systems shows that reachability sets remain small

Performance Results: SSD Cold Cache



Performance Results: SSD Hot Cache



Metadata-limits for Parallel File Systems

- Transfer idea of Direct-Lookup based file system using randomized metadata placement in a distributed environment
- Prototype implementation integrated into the modular GlusterFS distributed file system

- How can we support renames / move operations?

Metadata-limits for Parallel File Systems

- Transfer idea of Direct-Lookup based file system using randomized metadata placement in a distributed environment
- Prototype implementation integrated into the modular GlusterFS distributed file system

- Do we have to support renames / move operations?

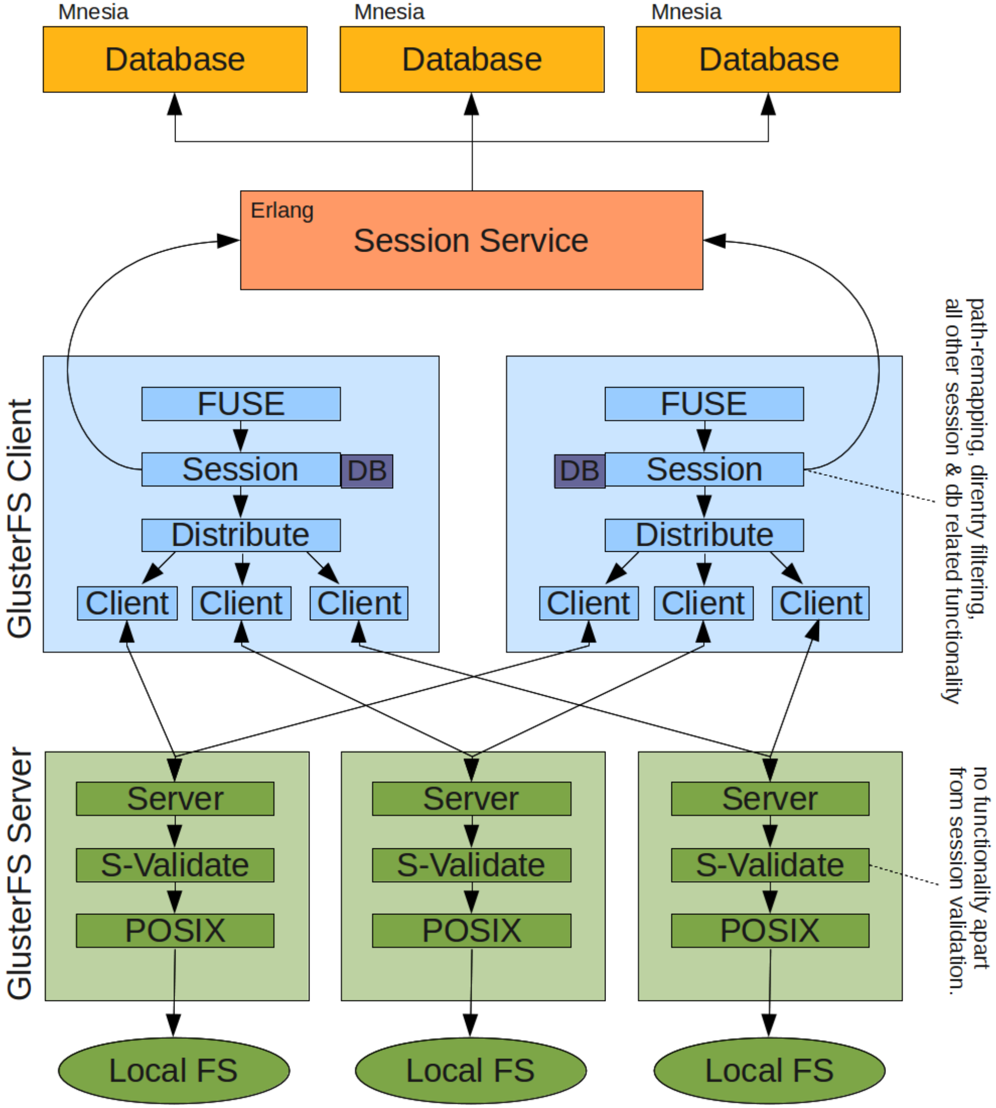
Metadata-limits for Parallel File Systems

- Transfer idea of Direct-Lookup based file system using randomized metadata placement in a distributed environment
- Prototype implementation integrated into the modular GlusterFS distributed file system

- Do we have to support renames / move operations?

- Approach based on asynchronously mirrored versioned database to mitigate direct lookup related problems

How to overcome Metadata-limits?



path-remapping, direntry filtering, all other session & db related functionality

no functionality apart from session validation.

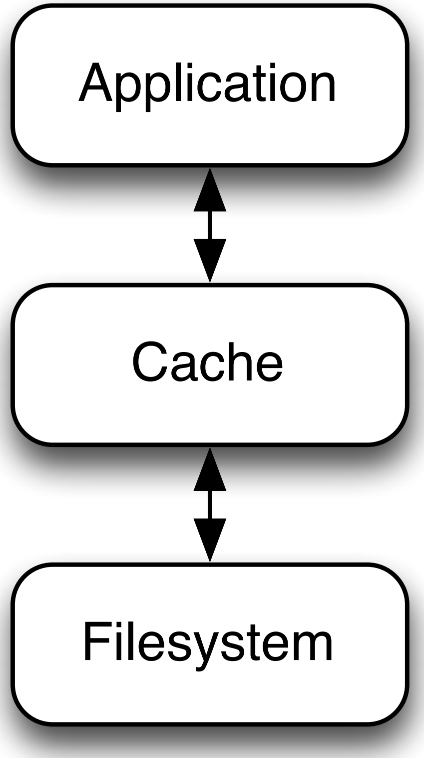
Application Hints

- Core question: How can application knowledge be used to optimize the performance of file systems?
- Here: Focus on cache management and data layout
- Coordination between different nodes / applications can be based on similar approaches

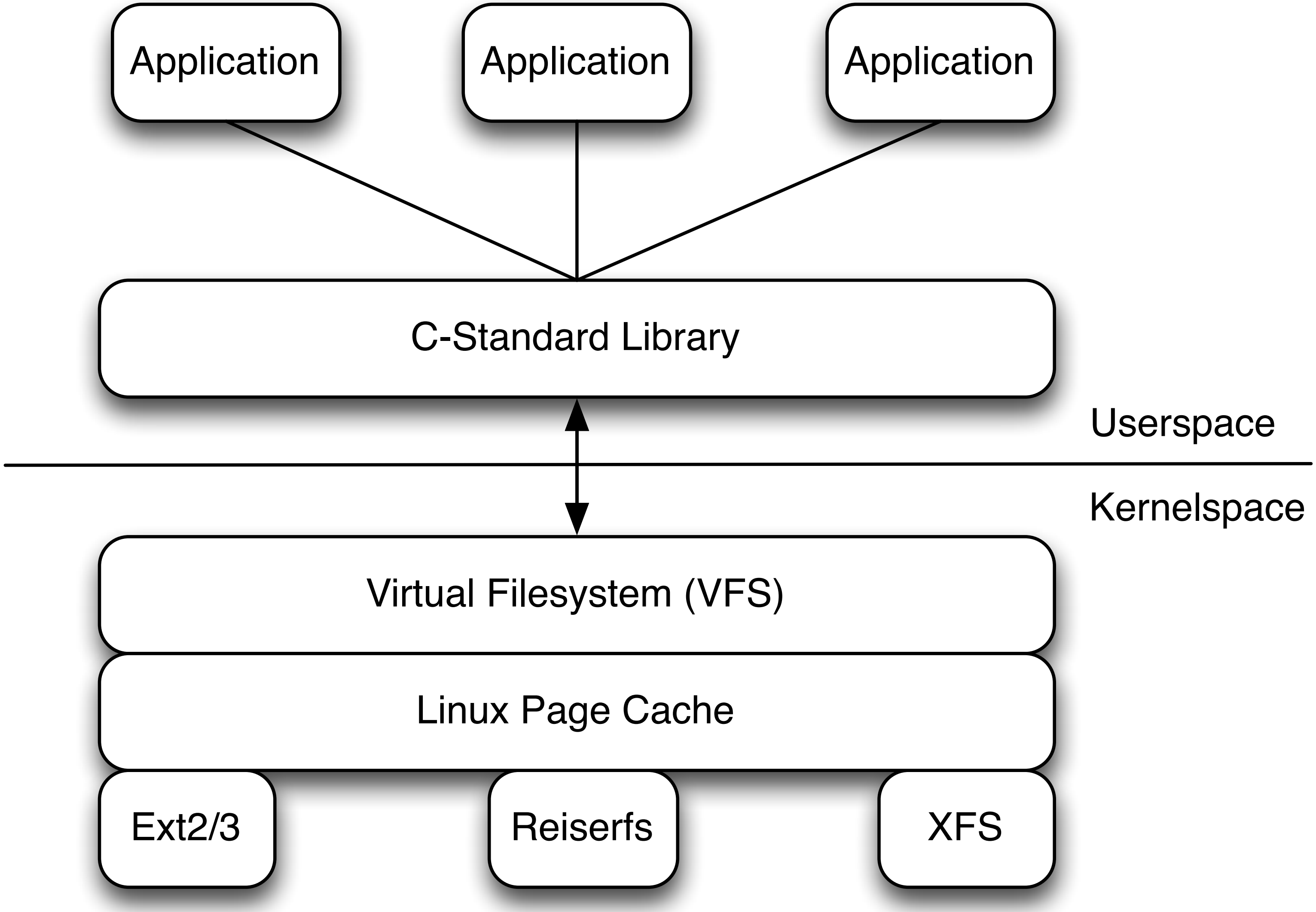
Caching

- A file system cache offers:
  - Storing / retrieving of data
  - Read ahead / write behind
  - Eviction strategy (e.g. LRU)

- Potential advices:
  - Storing of data: Do not store data that is used only once
  - Read ahead: Adjust read ahead size
  - Eviction strategy: Don‘t evict useful pages



Implementation of Advices in Linux



Application

Application

Application

C-Standard Library

Userspace

Kernelspace

Virtual Filesystem (VFS)

Linux Page Cache

Ext2/3

Reiserfs

XFS

Implementation of Advices in Linux

- POSIX_FADV_NORMAL: Sequential mode
- POSIX_FADV_SEQUENTIAL: Seq. Mode + bigger readahead
- POSIX_FADV_RANDOM: Random mode

- POSIX_FADV_WILLNEED: Prefetch file synchronously
- POSIX_FADV_DONTNEED: Invalidate desired pages
- POSIX_FADV_NOREUSE: Not implemented

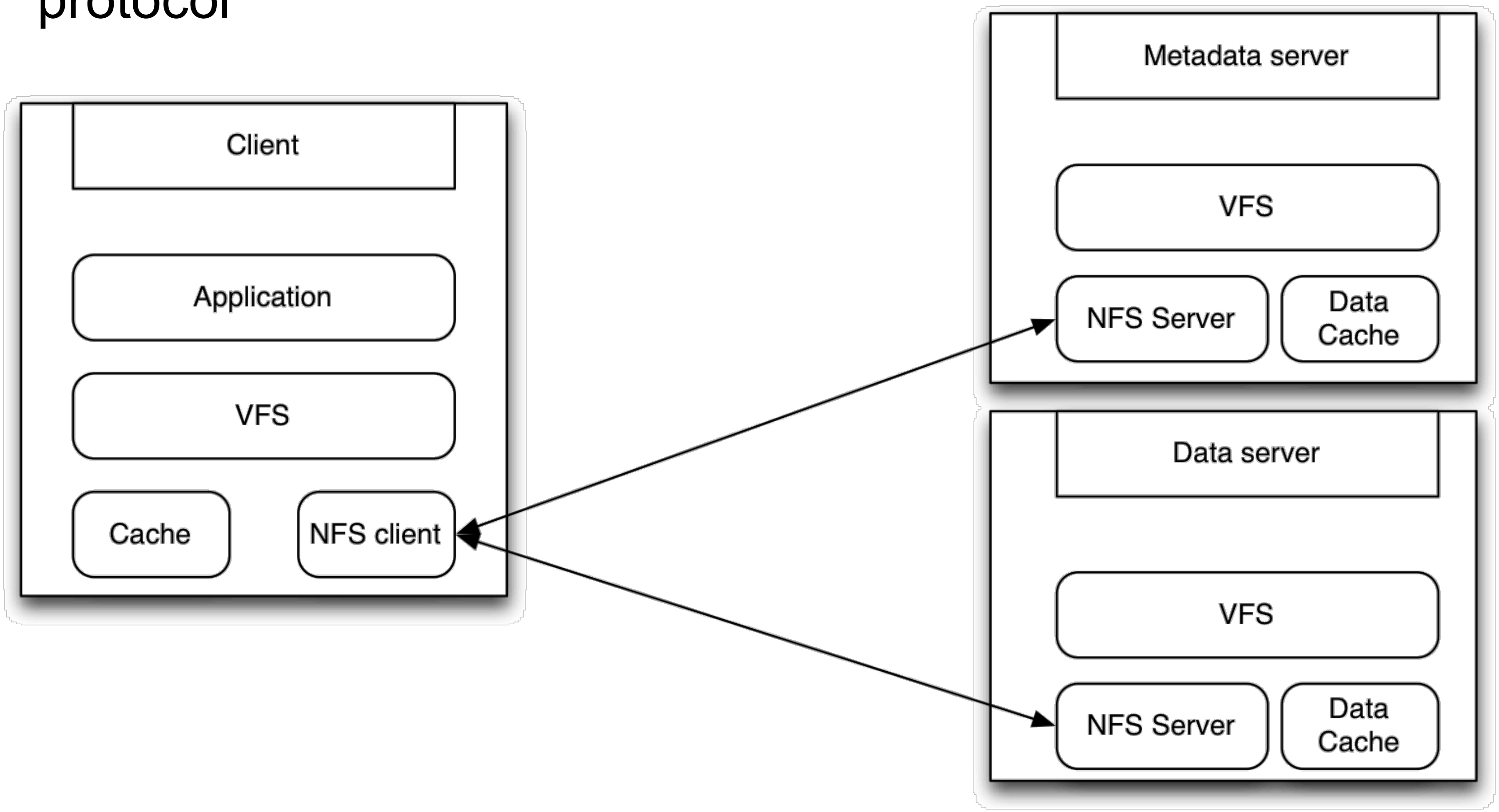
Implementation of Advices in Linux

- Synchronous prefetching can slow down applications
- Sequential/random advices cannot be applied to files
- Documentation does not match
- Hints are not getting passed to the file system

- Static / fixed interface
- No feedback about the state of advices (accepted/rejected)
- Features of modern file systems are not reflected

Advices in distributed settings

- Goal: Use application knowledge to enhance the (p)NFS protocol



Requirement Analysis

1. Predictable behavior: The interface should be well documented and the implementation should stick as closely to the description as possible
2. Extensibility: Vendor-defined advices for specific file systems should be supported
3. Notification of the file systems: Advices should be passed through to the system
4. Asynchronous behavior: The interface should include the possibility to choose between asynchronous and synchronous behavior
5. Support for directives: The new interface should support directives in addition to advices

Implementation of Advices

- Implementation of an advice interface between VFS layer of the Linux kernel and file systems
    - New method `advise()` has been added to VFS methods
    - Interface is called from the posix_fadvise() system call
    - Whenever system call is invoked, the advice will be redirected to the file system (if implemented)
    - File system has choice to act upon the advice or to delegate the handling to default implementation

Asynchronous Behaviour

- Standard Linux implementation of posix_fadvise() handles all advices synchronously
  - Problems for advices that may take significant amount of time to execute, e.g., POSIX_FADV_WILLNEED and POSIX_FADV_DONTNEED

- Compatible extension to current interface is proposed: Use a logical OR operation in combination with the type of the advice

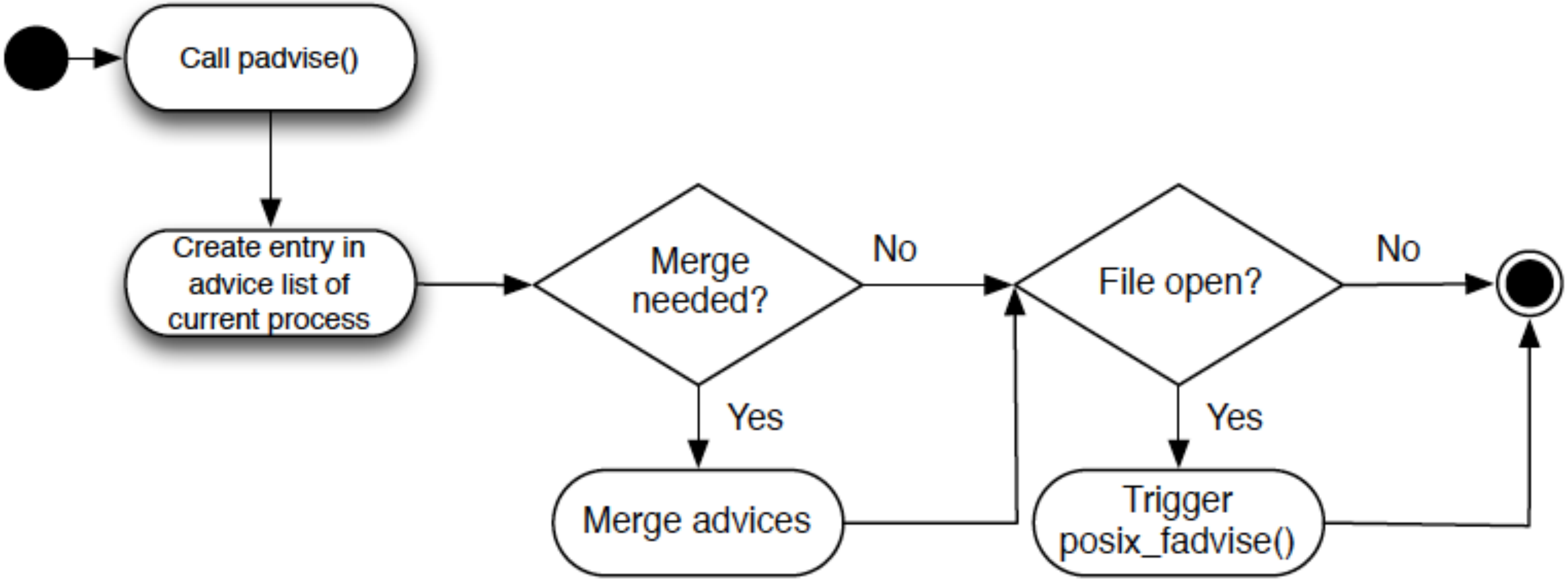
Interface for Directives

- System call provides user-space applications with a possibility to issue directives to the Linux kernel
- Kernel dispatches the directive to the file system
- New method `fdirective` added to the VFS interface
- Set of possible directives includes:
  - `DRCTV_PREFETCH`
  - `DRCTV_ASYNC_PREFETCH`
  - `DRCTV_SET_STRIPE_SIZE`
  - `DRCTV_SET_STRIPE_COUNT`
- Directives are more predictable and reliable than advices

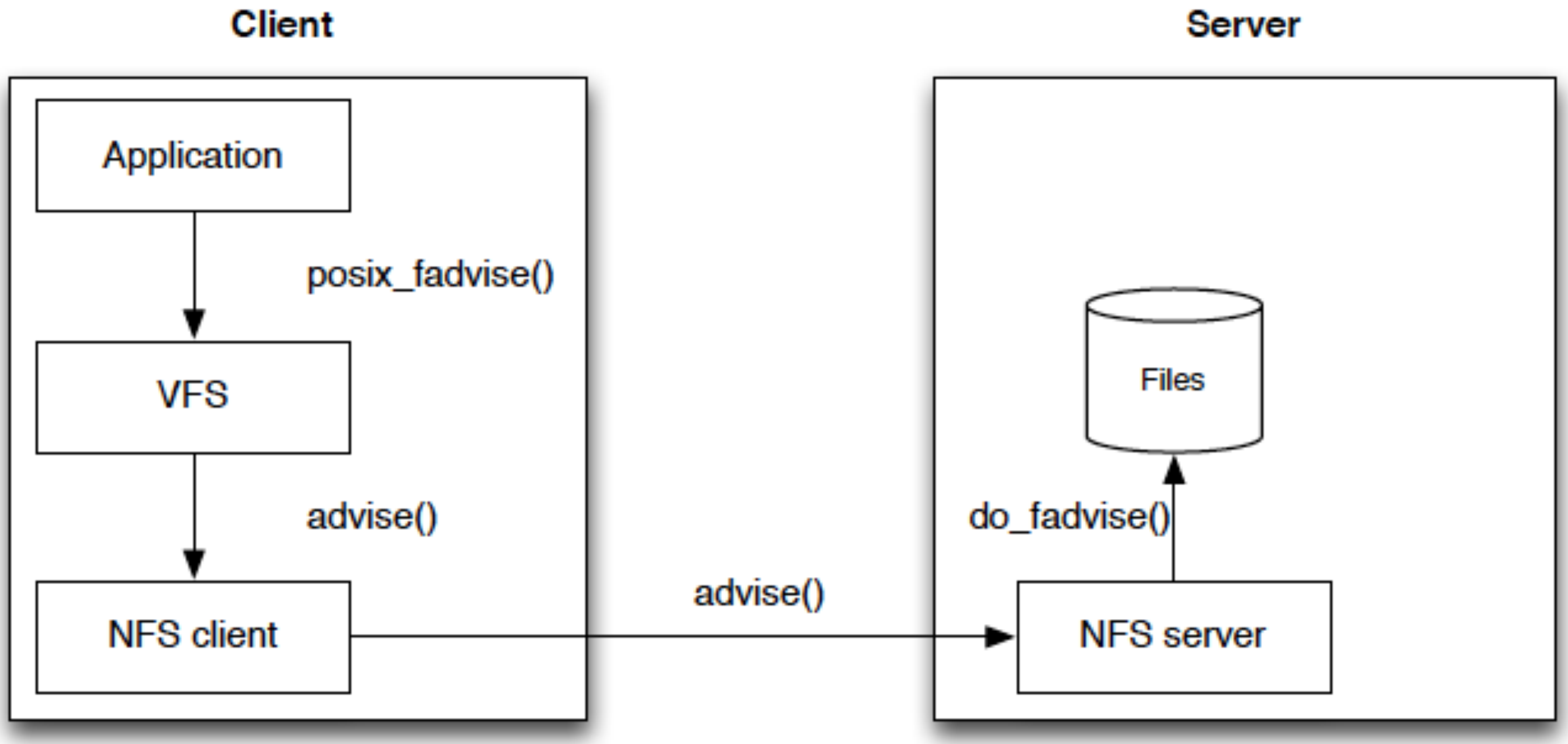
Process-Specific Advices

- Integration of advice-support itself is not always possible in legacy applications
  - Not every language supports posix_fadvise()
  - Closed-source applications cannot be modified
- Process-specific advice is an advice which is not applied to an open file, but to a complete process
  - If this process creates a new child process, the advices are inherited
  - Parent program can be written in a language which supports the desired advice interface
- Management of active advices can be implemented by adding a list of advices to process control block (PCB)

Process-Specific Advices



Transport of Advices to NFS Server



Client

Server

Application

posix_fadvise()

VFS

advise()

NFS client

advise()

Files

do_fadvise()

NFS server

Integration of Advices and Delegations

- Next Steps (not implemented): How could we combine advices and NFS delegations?

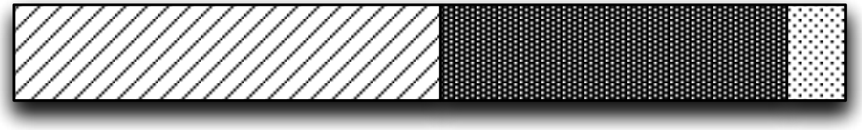
- A delegation is a promise from the server to a client that it will not be changed as long as the client holds the delegation
- When should a delegation be handed out to a client?
  - At the first open()
  - At the second open()
  - At the n-th open()
- These heuristics might work as a rule-of-thumb for generic applications, but fail for most HPC applications

Integration of Advices and Delegations

- Improvement: Use advices for the delegation hand-out algorithm
  - Grant delegations to clients that have signaled that a file is important
  - Revoke or give back delegations that belong to files which will not be used again

Integration of Advices and Delegations

- NFS 4.1 file delegations can be applied only to whole files. Worst case scenario:



Multiple clients accessing the same file in parrallel

Integration of Advices and Delegations

- Solution: Create a new type of delegations that work on byte-ranges, using the semantics of byte-range locking. Use advices to determine which byte-range should belong to which client

- NFS can handle asynchronous and synchronous I/O operations. The behavior could be switched at runtime with an appropriate advice, per-file and per-byte range

Integration of Advices and Delegations

- Apply client-side strategies to server-side caching:
  - Prefetch data
  - Evict unnecessary data
  - Make use of direct I/O if the clients requests it
- Use Metadata servers to control caches of data servers

Integration of Advices and Delegations

- NFS uses close-to-open consistency, which is not strict enough for some applications
- Switch between different consistency models at run-time
- New model: delegation-only consistency
- Configurable per file

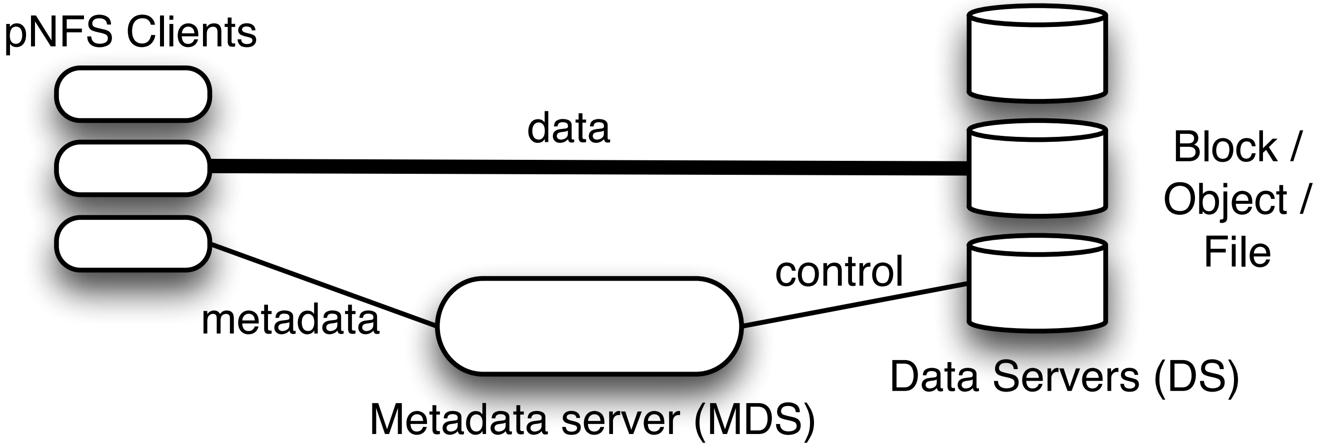
Access patterns and Data Layout

- Mismatch of access pattern and storage system can have severe impact on performance!

- Ideas to improve this situation:
  - Shift some responsibility to clients
  - Extend application’s hints on resource usage
  - Use reconfigurable, script based file layout descriptors

M. Grawinkel, T. Süß, G. Best, I. Popov, A. Brinkmann: Towards Dynamic Scripted pNFS Layouts, PDSW 2013

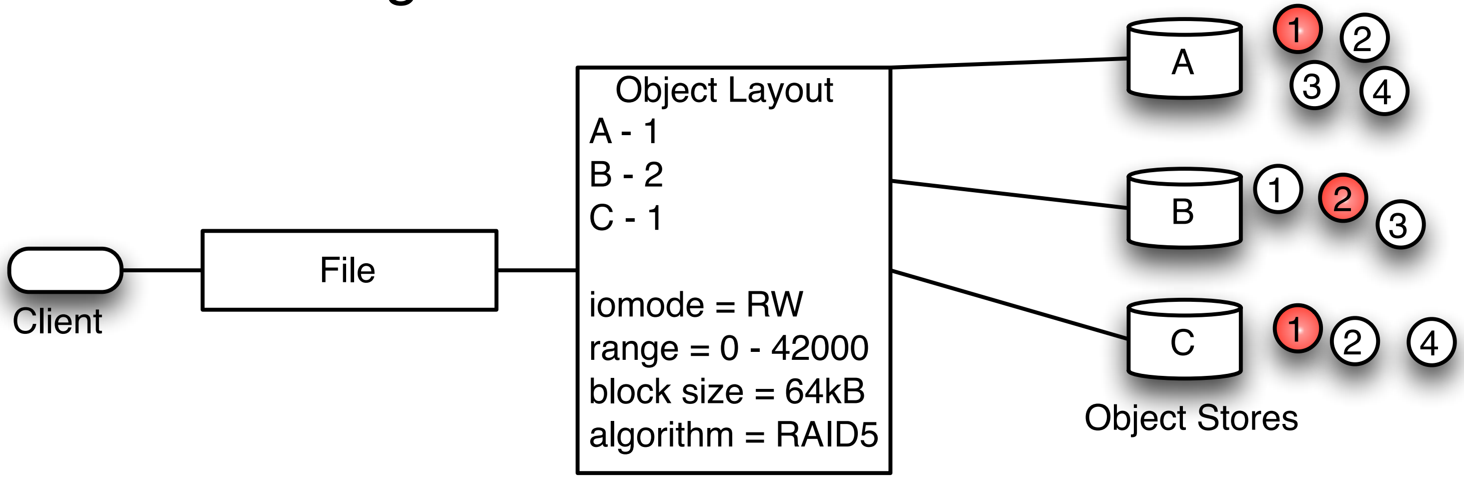
NFSv4.1 / pNFS



- NFSv4.1 extension for parallel and direct data access
- Namespace and metadata operations on MDS
- Direct data path to data servers (Block, Object, File layouts)

Data Access

- File layout organized by MDS, client calls GETLAYOUT for a file handle
- Layout contains
  - Locations: Map of files, volumes, blocks of file
  - Parameters: iomode (R/RW), range, striping information, access rights, ...
- Current layouts define fixed algorithms to calculate target resources for logical file‘s offsets



M. Grawinkel, T. Süß, G. Best, I. Popov, A. Brinkmann: Towards Dynamic Scripted pNFS Layouts, PDSW 2013

Scripted Layouts

- Introduce scripting engine within pNFS stack
  - Layout uses script instead of fixed algorithm
  - Flexible placement strategies
  - RAID 0/1/4/5/6, Share, CRUSH, Clusterfile, ...
  - Flexible mapping to storage classes

- Application can:
  - Provide own layout script
  - Reconfigure storage driver
  - Update layout script, parameters (LAYOUTCOMMIT)
  - Move storage resources between layouts

M. Grawinkel, T. Süß, G. Best, I. Popov, A. Brinkmann: Towards Dynamic Scripted pNFS Layouts, PDSW 2013

Scripting Engine

- Lua
  - Very fast scripting language
  - Embeddable with bindings for C/C++
  - In-kernel scripting engine - lunatik-ng[1]
  - Stateful: Can hold functions, tables, variables
  - Callable from kernel code
- Syscall for applications
  - Administrators / Applications can get/set (global) variables and functions
- Extendable by bindings
  - kernel crypto API
  - pNFS

[1] http://github.com/lunatik-ng/lunatik-ng

Performance Impact

- Calculate stripe unit index from file_layout: 0.87µs / call (±0.03)
- Creating a new file_layout object: 2.18µs / call (±0.05)

```
function lua_create_filelayout (buf)
  rv = pnfs.new_filelayout()
  rv.stripe_type = "sparse"
  rv.stripe_unit = buf[1] + buf[3]
  rv.pattern_offset = buf[2] + buf[4]
  rv.first_stripe_index = buf[5] + buf[6]
  return rv
end
```

- Calling kernel.crypto.sha1(20 bytes): 1.25µs / call (±0.02)
- Creating new file_layout with sha1() calculation: 3.25µs / call (±0.02)

Closing Remarks

- "Interpreting behavior and suggesting improvements is a manual process that requires knowledge of the storage system and I/O tuning expertise"[1]
- Storage is too important not to think about it!
- Changes might be easier (and much less expensive) than to simply continue the old way!

---

[1] P. Carns: HEC I/O Measurement and Understanding Panel Presentation. HEC FSIO 2011 Workshop

Thank you very much for your attention!

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ