Lustre Client IO

CLIO

Nikita Danilov

Xyratex

14-Oct-2012

Overview

CLIO is a re-structured Lustre client IO code. The project started in mid-2007 and was completed by the end of 2008. Less than 1 person worked on it on average.
Why the re-write was needed?

- Client IO was a constant source of bugs. The number of bugs was roughly constant over time. Careful re-write might change this situation.
- Plans for porting Lustre client to other platforms were considered: Windows, OS X, Solaris.
- A host of new features: "parallel IO", peer-to-peer caching, *&c.*
- Similar re-write of the meta-data server (new MDT) in the previous year was a success.

Design goals

- Reduce the number of bugs in the IO path.
- Introduce more logical layer interfaces instead of existing all-in-one obd device interface.
- Define clear and precise semantics for the interface entry points;
- Simplify structure of the client code.
- Support upcoming features:
  - server network striping (SNS);
  - p2p caching (read-only);
  - parallel non-blocking IO;
  - pNFS;
  - file migration re-striping and, more generally, dynamically mutable file layouts.
- Reduce kernel stack consumption.

Design restrictions

- No meta-data changes.
- No old kernel (2.4) support.
- Portable code; isolate non-portable code.
- No changes to recovery.
- No changes to LDLM.
- The same layers with mostly the same functionality.
- As few changes to the core logic of each Lustre data-stack layer as possible (*e.g.*, no changes to the read-ahead or osc RPC logic)—one change at a time.

Major differences

with the pre-existing client code

- Locks on files (as opposed to locks on stripes) are first-class objects that can be cached.
- Sub-objects (stripes) are first class objects.
- Stripe-related logic moved out of llite (almost).
- io control flow is different:
  - pre-CLIO: llite implement control flow, calling underlying obd methods as necessary;
  - CLIO: generic code (`cl_io_loop()`) controls IO logic calling all layers, including vvp.

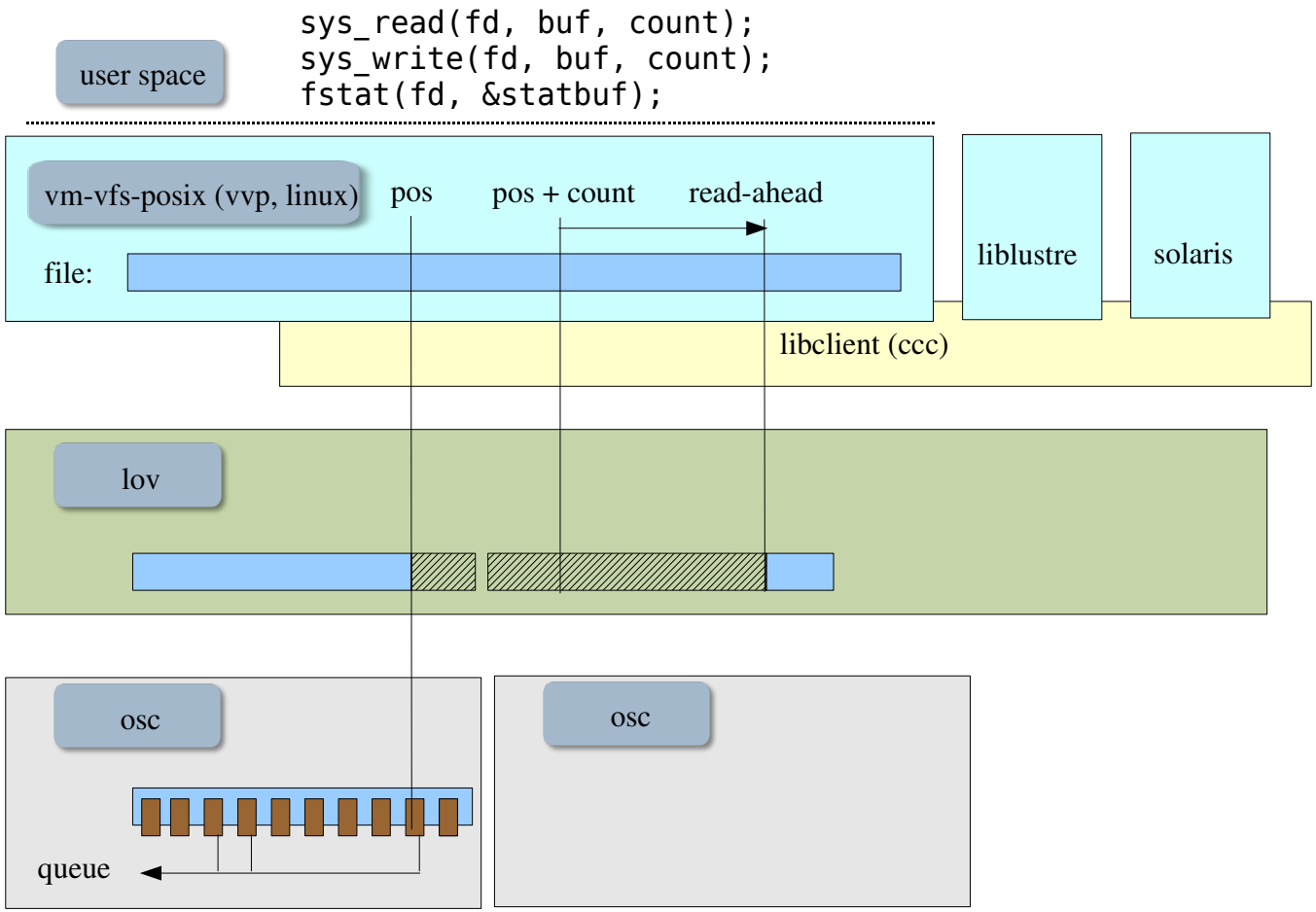
Major differences

continued

- Using `lu_context` for allocations.
- Coöperative caching: each layer might request page cache write-out (needed for SNS).
- fid-based object lookup.
- Page tables internal to Lustre.
- All major abstraction are implemented as non-blocking state machines (to support parallel IO).

CLIO layers

a nice picture



vvp: VFS, VM, POSIX

no, these are nobody's initials

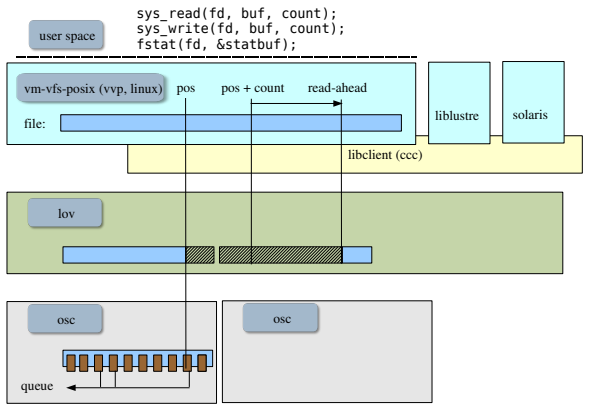
`llite/vvp_{lock,io,page,dev,object}.c`

- top-most layer;
- interacts with the host kernel's VFS and VM sub-systems;
- mostly non-portable;

Alternative existing implementations of the top-most layer:

- echo-client for `echoclient`. Used for testing;
- slp for liblustre client in user space;

Other possible implementations: ports to other operating systems, pNFS client, proxy server.

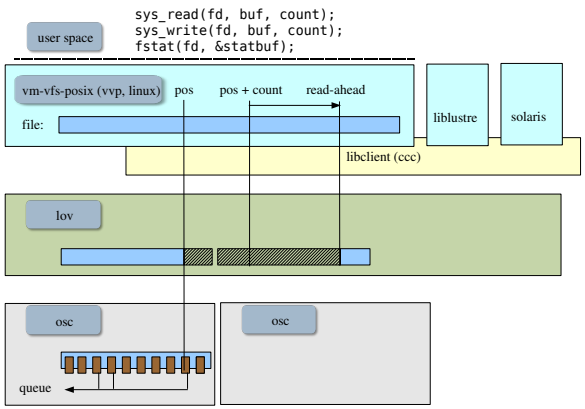


vvp: VFS, VM, POSIX

continued

This layer implements:

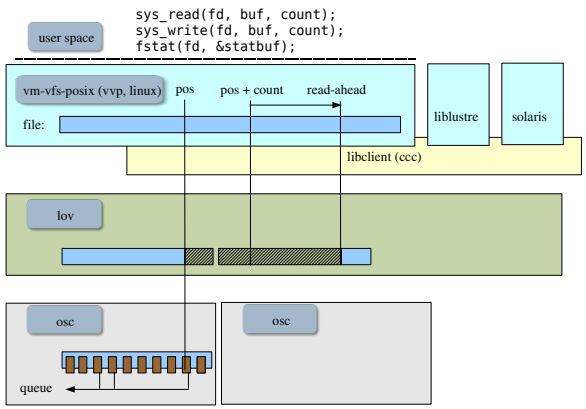
- mapping from Linux specific entry points (`readv`, `writev`, `sendfile`, *&c.*) to the Lustre IO loop;
- mmap,
- POSIX features like short reads, `O_APPEND` atomicity, *&c.*,
- read-ahead (this is arguably not the best layer to implement read-ahead in, because it requires a network-aware read-ahead algorithm).



lov: logical object volume

`lov/lov_{lock,io,page,dev,object}.c` lov layer implements "file layouts". It supports 2 layouts:

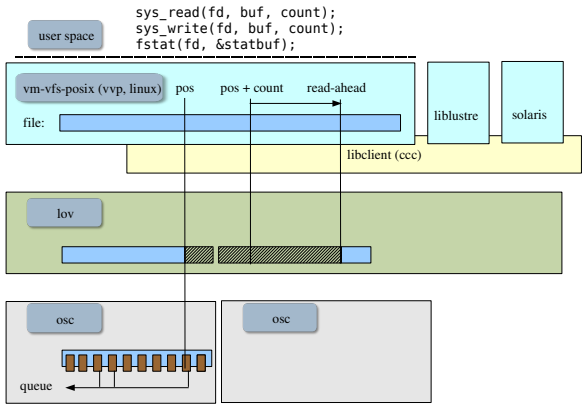
- raid0 striping,
- "empty" file, created on MDS, but without OST storage assigned.



osc: OST Connector

`osc/osc_{lock,io,page,dev,object}.c` This bottom-most layer interacts with LDLM and LNet:

- it decides when efficient rpc can be formed from cached data;
- it calls LNET to initiate a transfer and to get notification of completion;
- it calls LDLM to implement distributed cache coherency, and to get notifications of lock cancellation requests.



CLIO abstractions

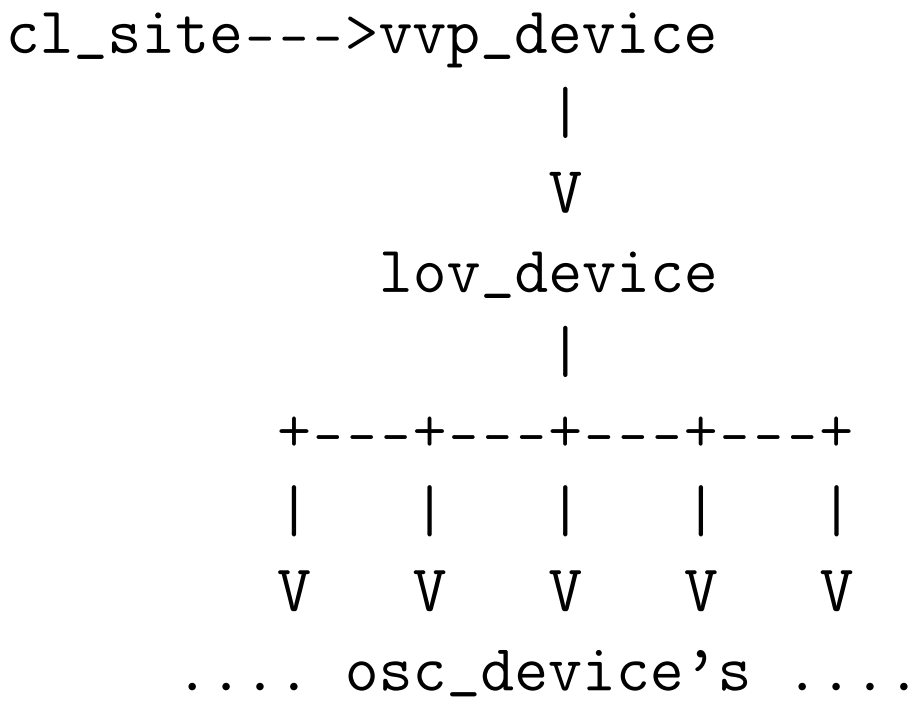
CLIO introduces the following "entities":

- object (`cl_object`). Files and stripe objects are instances of this type;
- page (`cl_page`). Data pages that store object's contents;
- lock (`cl_lock`). Extent lock on an object;
- io (`cl_io`). High-level io activity such as whole read/write system call, or write-out of pages from under the lock being canceled;
- request (`cl_req`). Represent a collection of pages to be sent over network. A request is constructed by req-forming engine, which is part of osc layer, that tries to saturate transport with large and continuous transfers.

Device

cl_device

Device represents a layer in IO "stack"

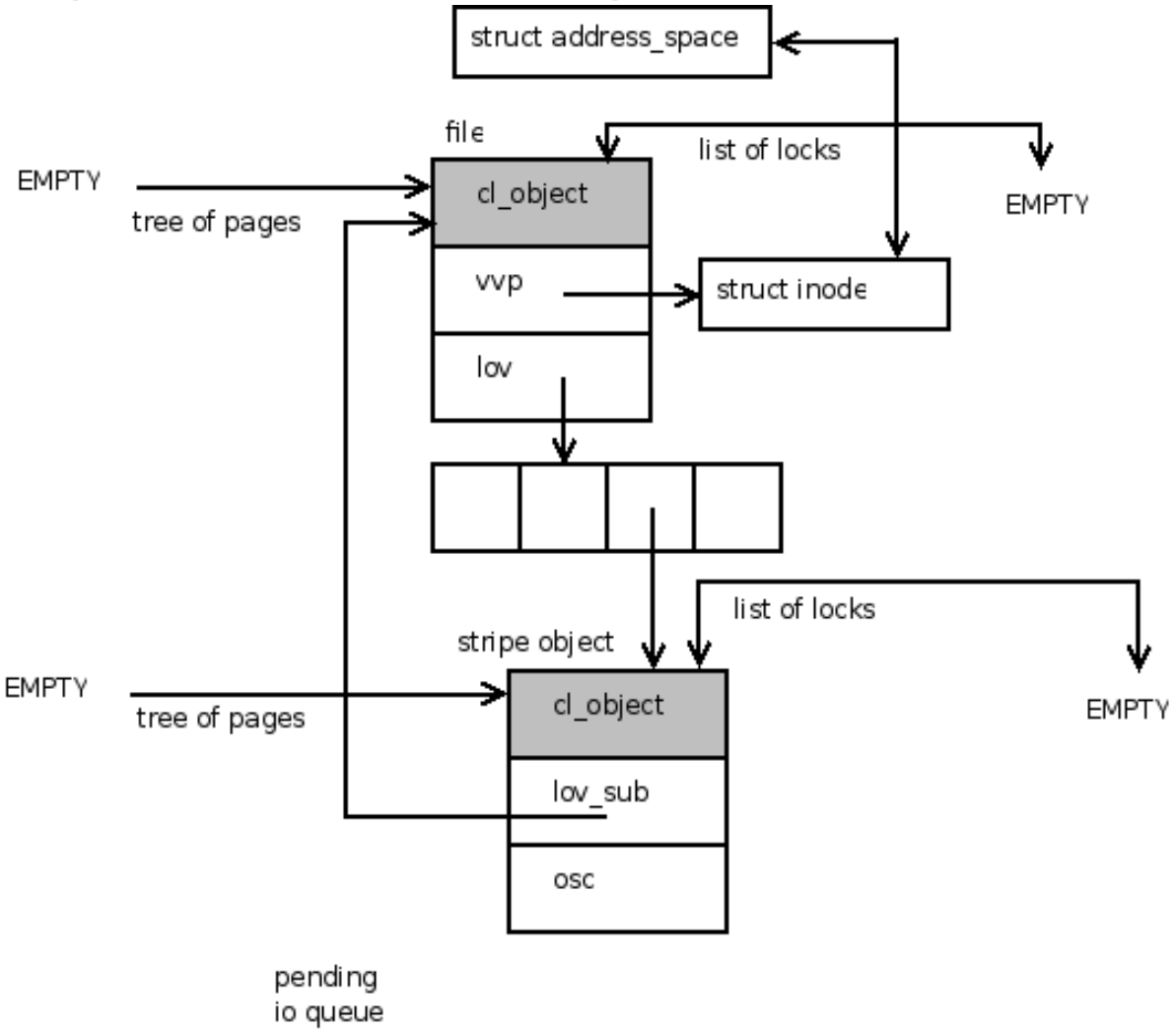


Devices are responsible for creation of objects, defining layers in object. Objects are responsible for layering of locks, ios and pages.

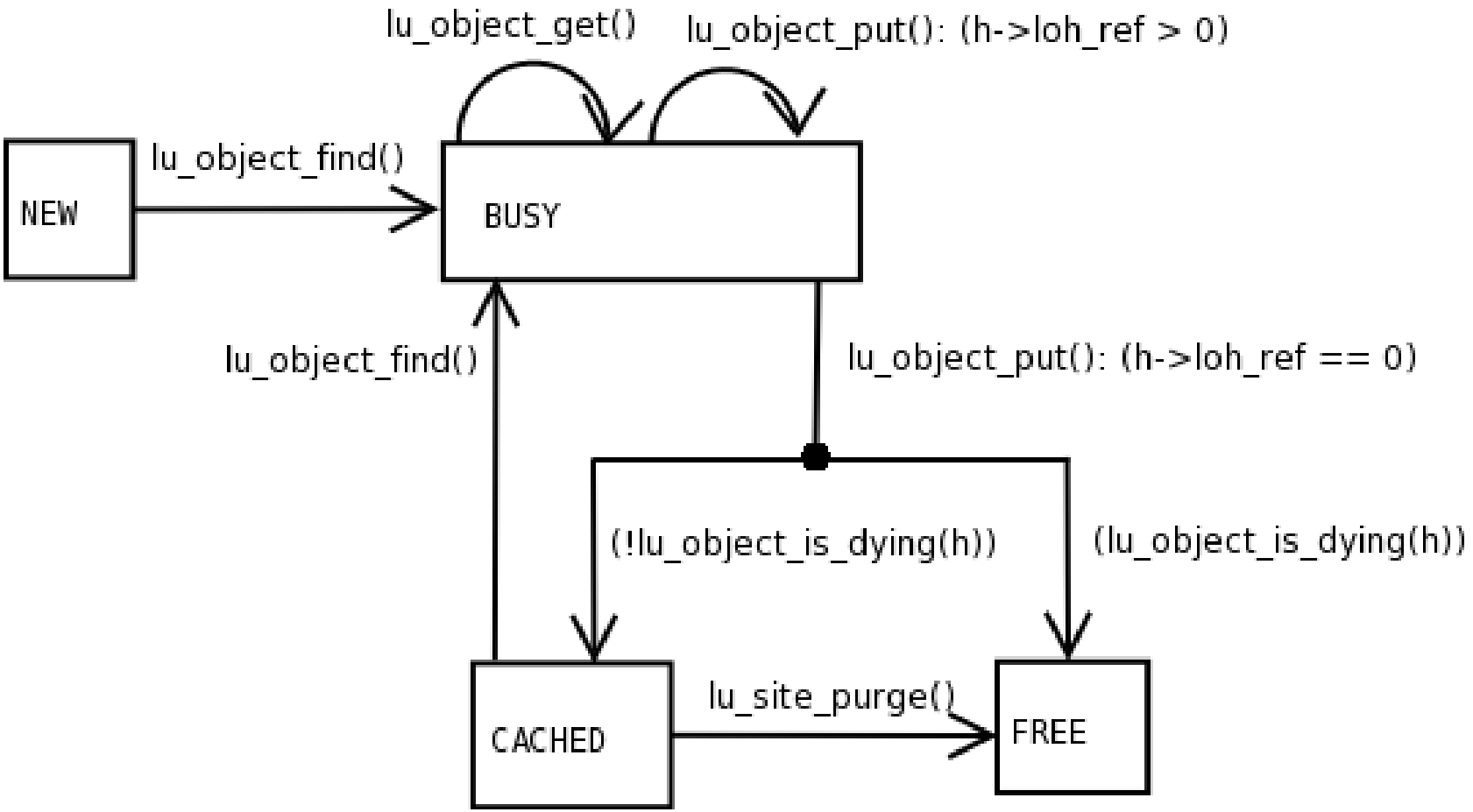
Object

cl_object

Represents file and stripe object



Object state diagram



The same as for server-side `lu_object`

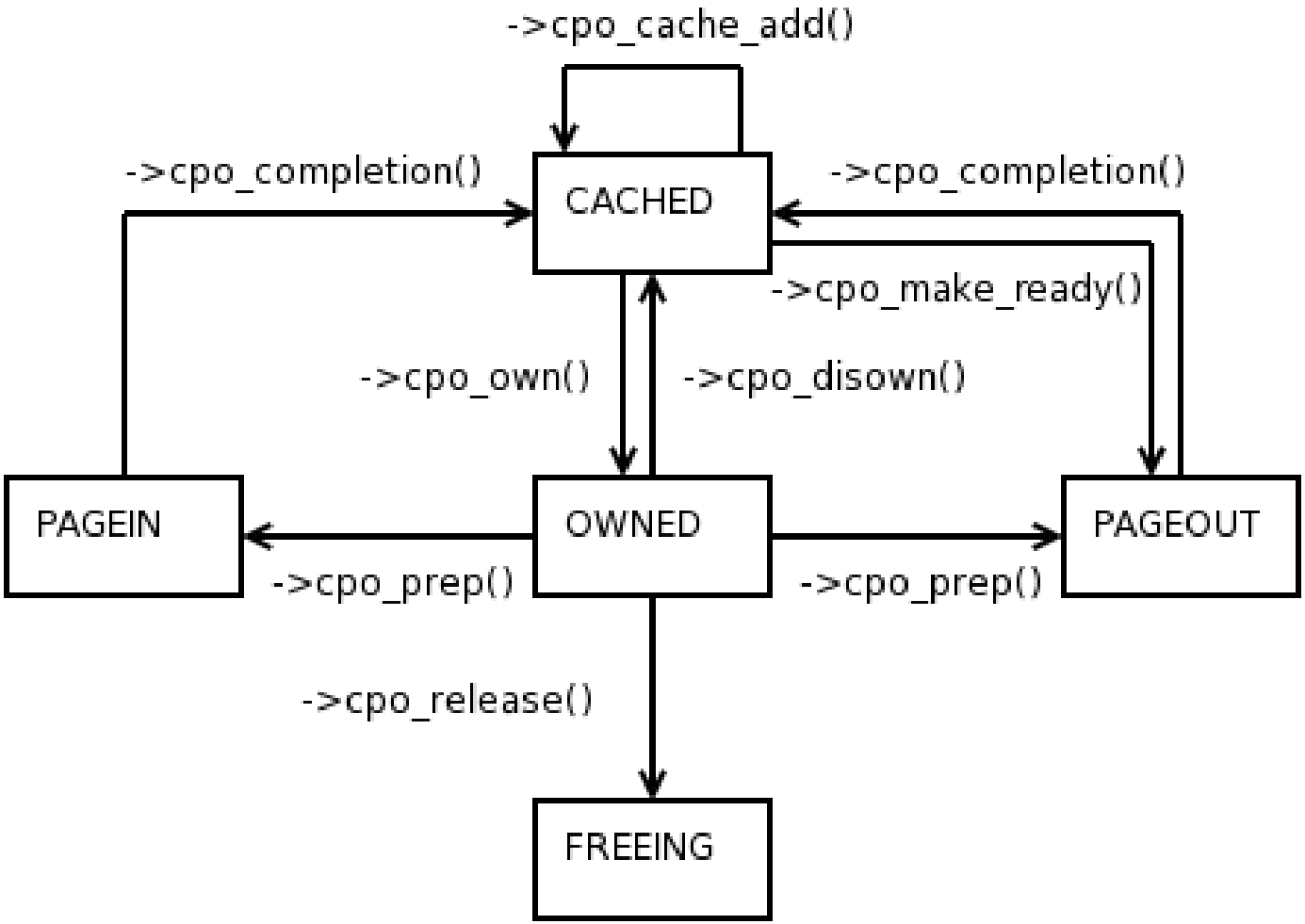
Page

cl_page

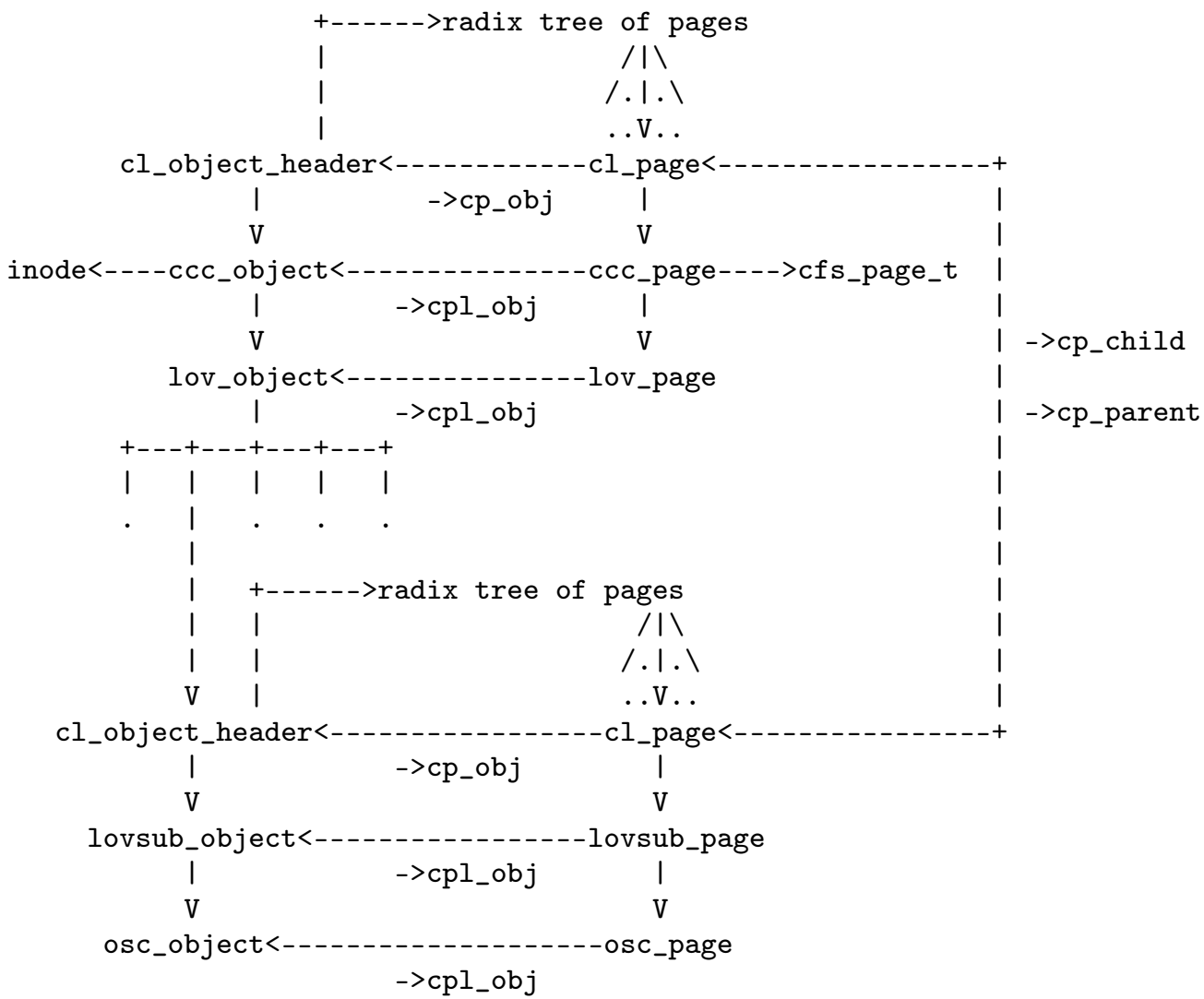
Page with data in page cache

- all pages in the object are of the same size;
- associated with `cfs_page_t`;
- indexed within object (`cl_page::cpo_index`);
- owned by IO;
- the same lock for read and write (portability);

Page state machine



Objects and pages



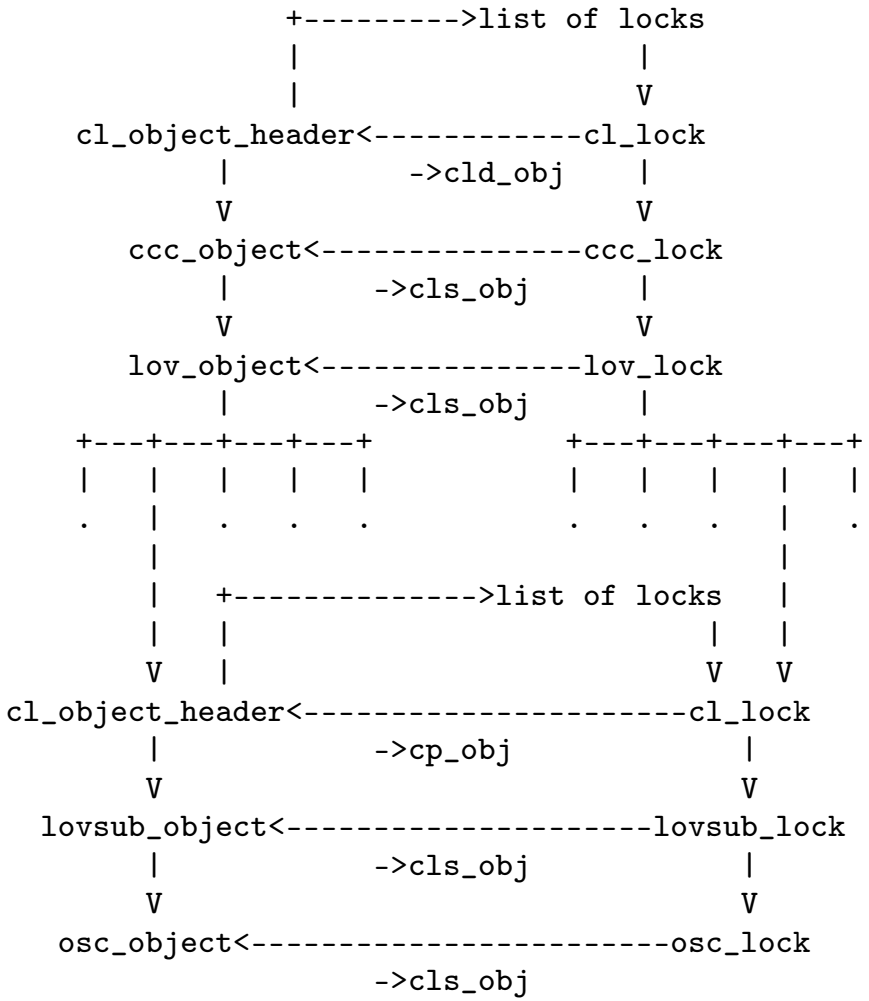
Lock

cl_lock

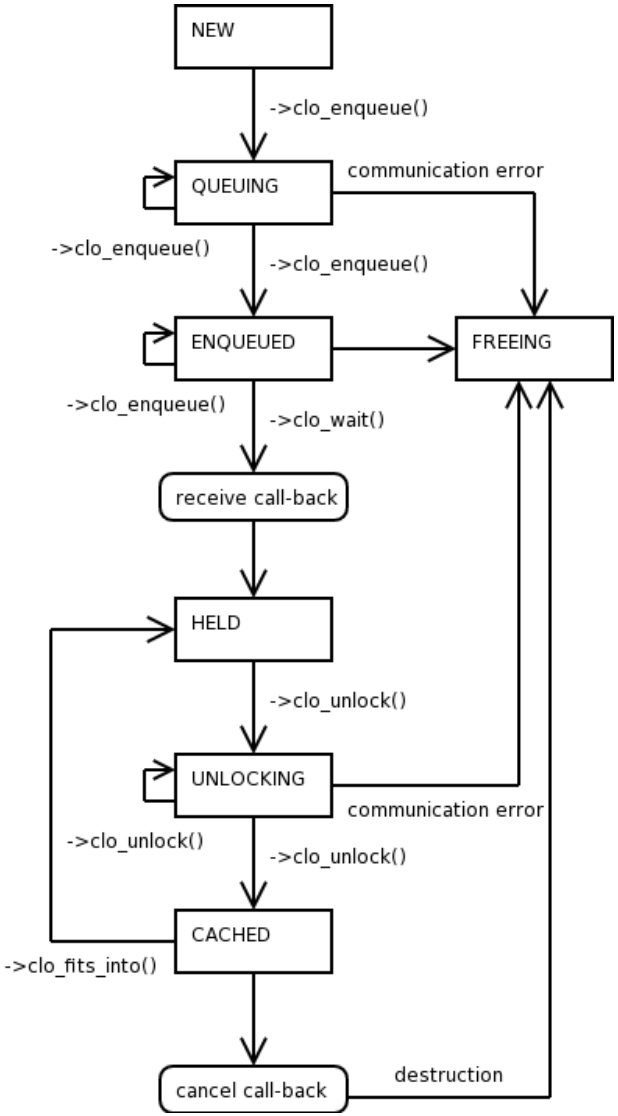
Extent lock on a file or extent lock on a stripe object.

- protects cached pages;
- reacts to DLM callbacks (blocked, conflict);
- mode: `READ`, `WRITE`, `PHANTOM` (for glimpse);
- locks are cached;
- fine-grained locking of locks;
- partial invalidation;
- subtle cases: cascading evictions (multi-stripe locks);

Objects and locks

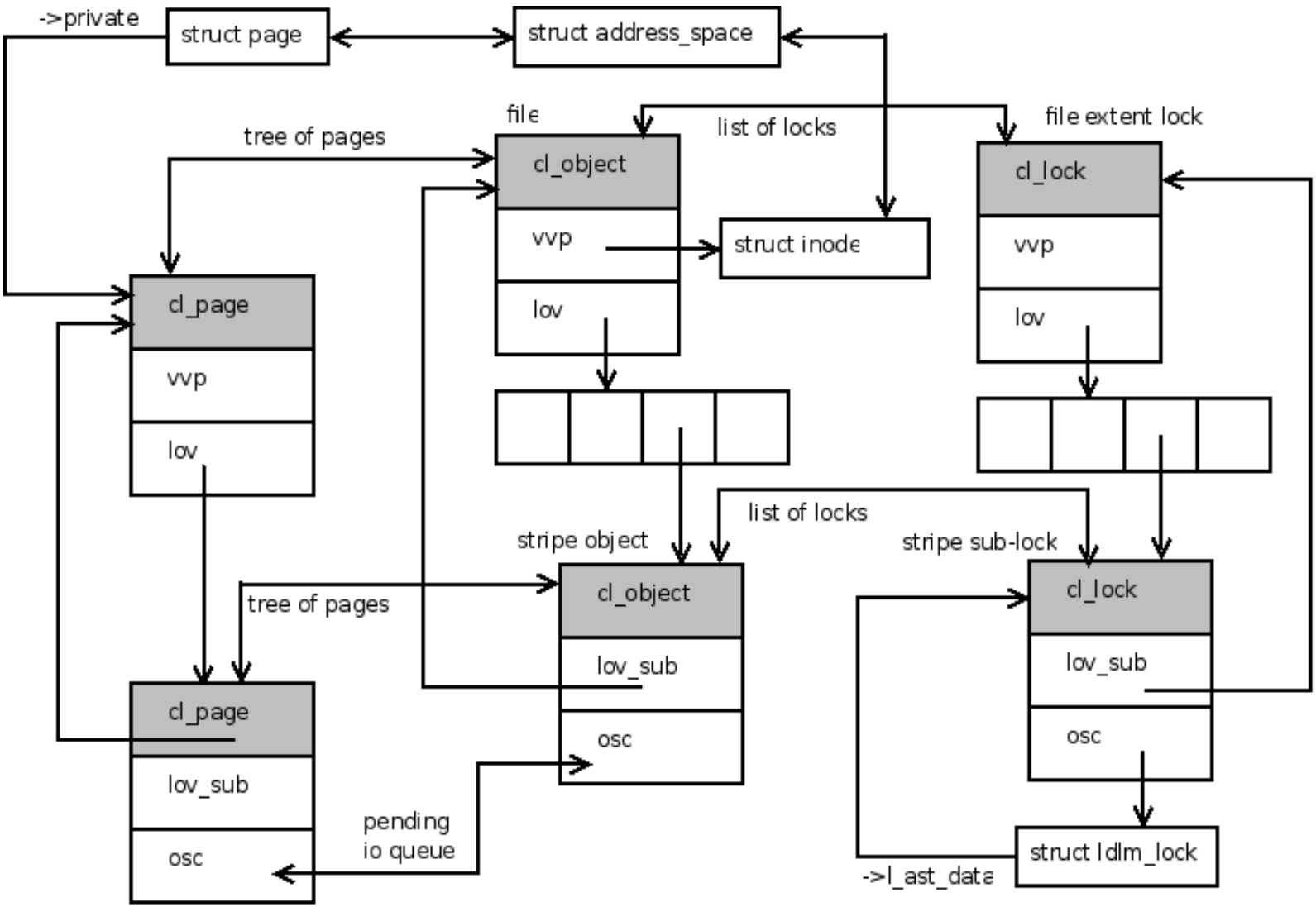


Lock state machine

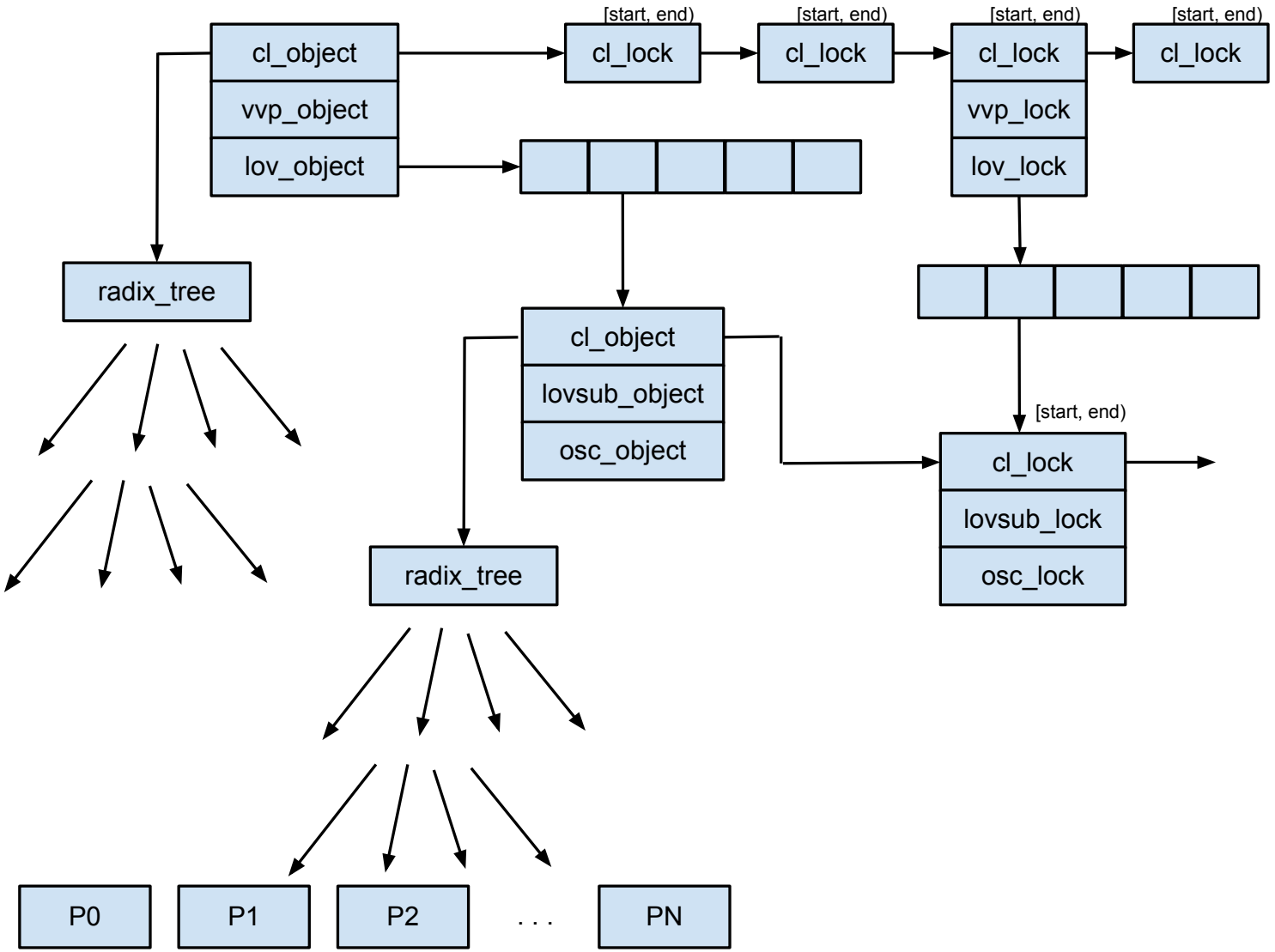


- non-blocking state machine;
- state transitions protected by `cl_lock->cll_guard`;
- each state transition goes through all layers in one or other direction;
- locally initiated operation: top-down;
- call-backs: down-top;
- no good lock ordering!
- solution (sort of): closures (`cl_lock_closure`).

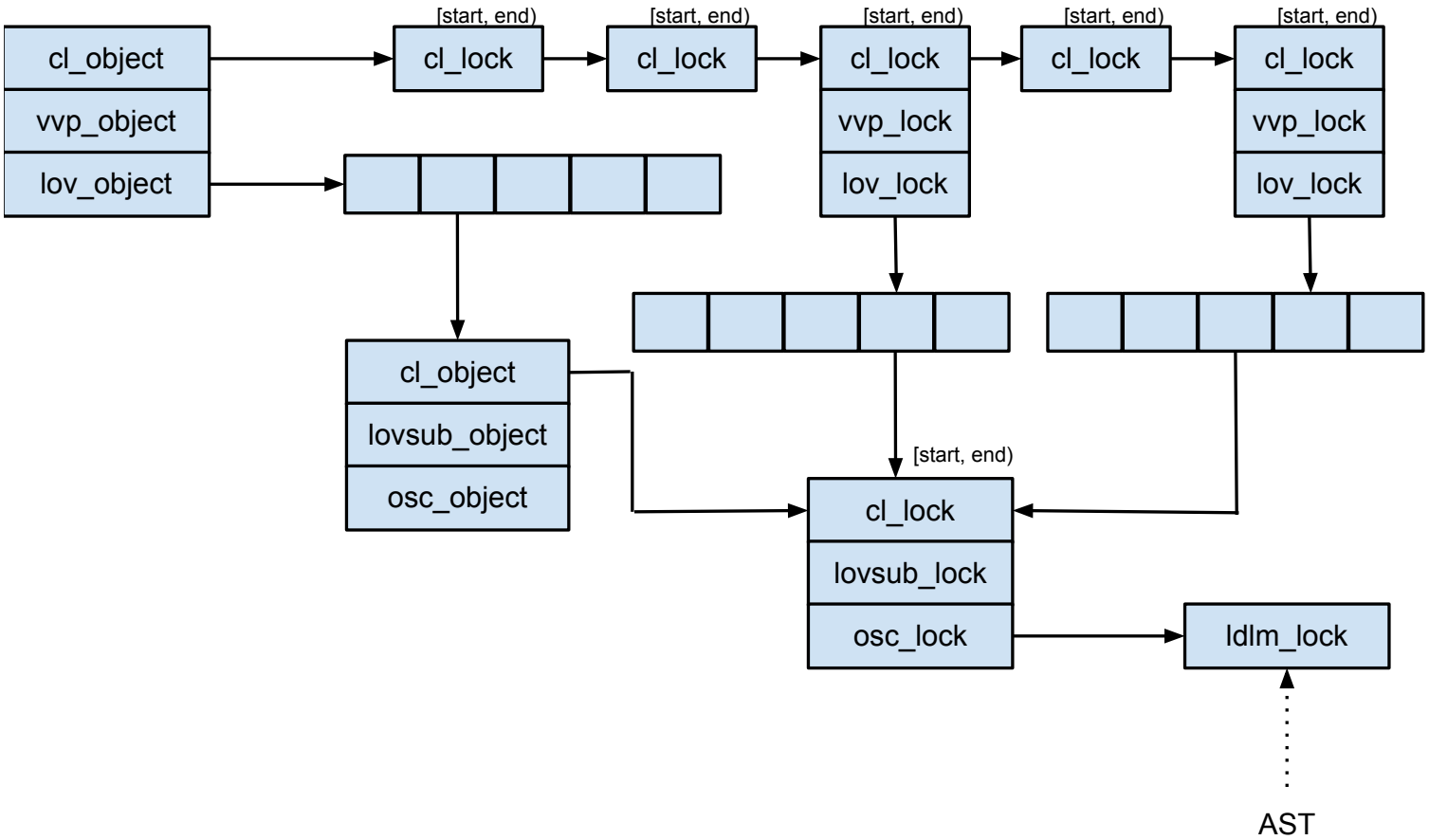
Locks and objects (and pages)



Example0: extent writeout



Example1: sub-lock sharing, closures



Lessons learned

by other people

- Non-blocking state machines (together with `cl_env`) are difficult to understand.
- First-class locks proved to be a major source of complexity. Probably a design mistake.
- First-class sub-objects, on the other hand, were very useful, uncovered some problems with recovery.
- CLIO was designed to accommodate for the future extensions that never materialized (yet). Its complexity is difficult to justify without reference to these extensions.

That's all

Thank you for your attention!

References

- http://wiki.lustre.org/images/3/37/CLIO-TOI-notes.pdf
- http://wiki.lustre.org/images/6/66/CLIO-TOI.pdf
- http://wiki.lustre.org/doxygen/b_client_io_layering/api/html/group__clio.html