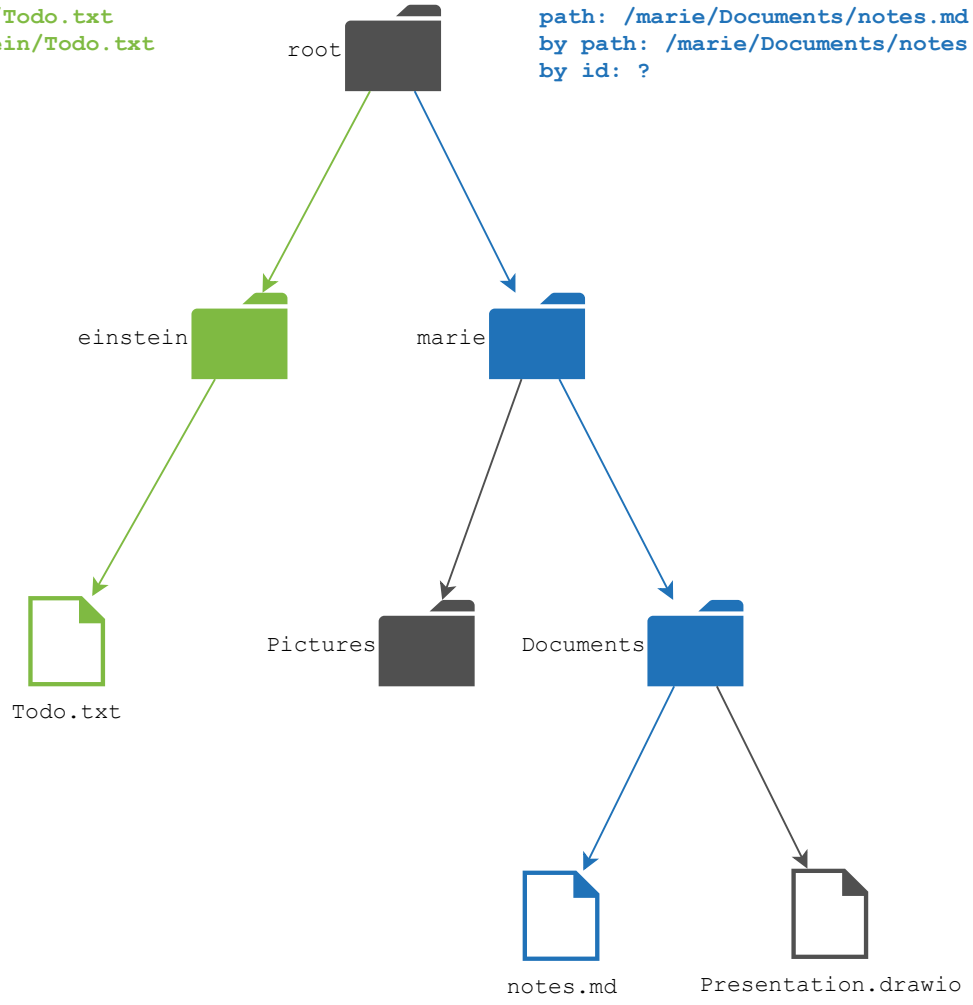
POSIX filesystem

path: /einstein/Todo.txt
by path: /einstein/Todo.txt
by id: ?

path: /marie/Documents/notes.md
by path: /marie/Documents/notes.md
by id: ?



Every node can also be referenced by id. A CS3 resource id consists of `<storageid>` and `<opaqueid>`. The gateway uses the storage id to route requests to the responsible storage provider.
The storage provider is responsible for looking up the file by id as well as by path.
To prevent renames from breaking shares, metadata is attached to file ids, not paths.

Pros:
- path on disk = requested path
- reuses existing layout
- metadata can be stored in extended attributes

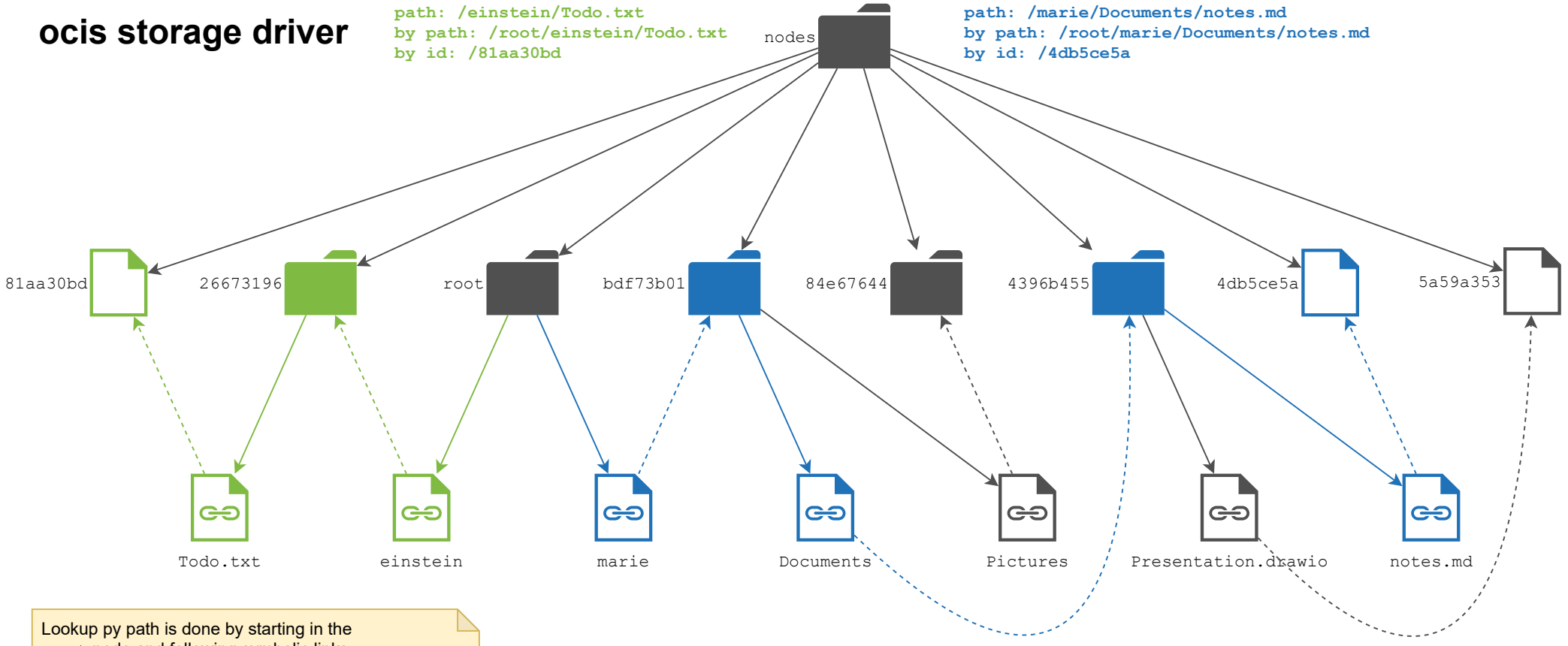
Cons:
- not all storage systems provide fast lookup by id.
- POSIX would have to `find .../root -inum 12345`
- Caveat: inodes are reused. So an additional file id is still necessary.

Outlook:
- check out btrfs and zfs. They may have a fast lookup by id on online filesystems
- inotify could trigger etag / tmtime propagation and size accounting
- index metadata including fileid in a dedicated service
- protip: using the full path as the key in a kv store requires O(n) updates when renaming directories where n = number of children in the affected subtree

ocis storage driver

path: /einstein/Todo.txt
by path: /root/einstein/Todo.txt
by id: /81aa30bd

path: /marie/Documents/notes.md
by path: /root/marie/Documents/notes.md
by id: /4db5ce5a



Lookup py path is done by starting in the `root` node and following symbolic links.

Lookup by id is done by starting in the `nodes` directory and accessing the dir or file with the correct id.

Pros:
- no additional database needed
- path and id based lookups are cached by the linux kernel VFS
- metadata like owner and permissions in extended attributes
- extended attributes are cached by the kernel VFS as well
- versions and trashed files can be stored as nodes
- cs3 references can be stored as a node with metadata
- can still be navigated on the cli
- works with local posix, or network filesystems like cephfs, spectrum scale, glusterfs, and even nfs (relies on the underlying filesystem atomicity)
- rename of a large tree is an O(2) write operation.

Cons:
- you are not suposed to see this as an end user
- doubles the amount of required inodes

Outlook:
- shard nodes into multiple subdirs `84e67644` -> `84/e6/76/44`
- optionally deduplicate using content addressed storage for blobs
- optionally store metadata in dedicated service, eg. redis
- optionally store blobs in dedicated service, eg. s3
- fuse based filesystem for ocis layout