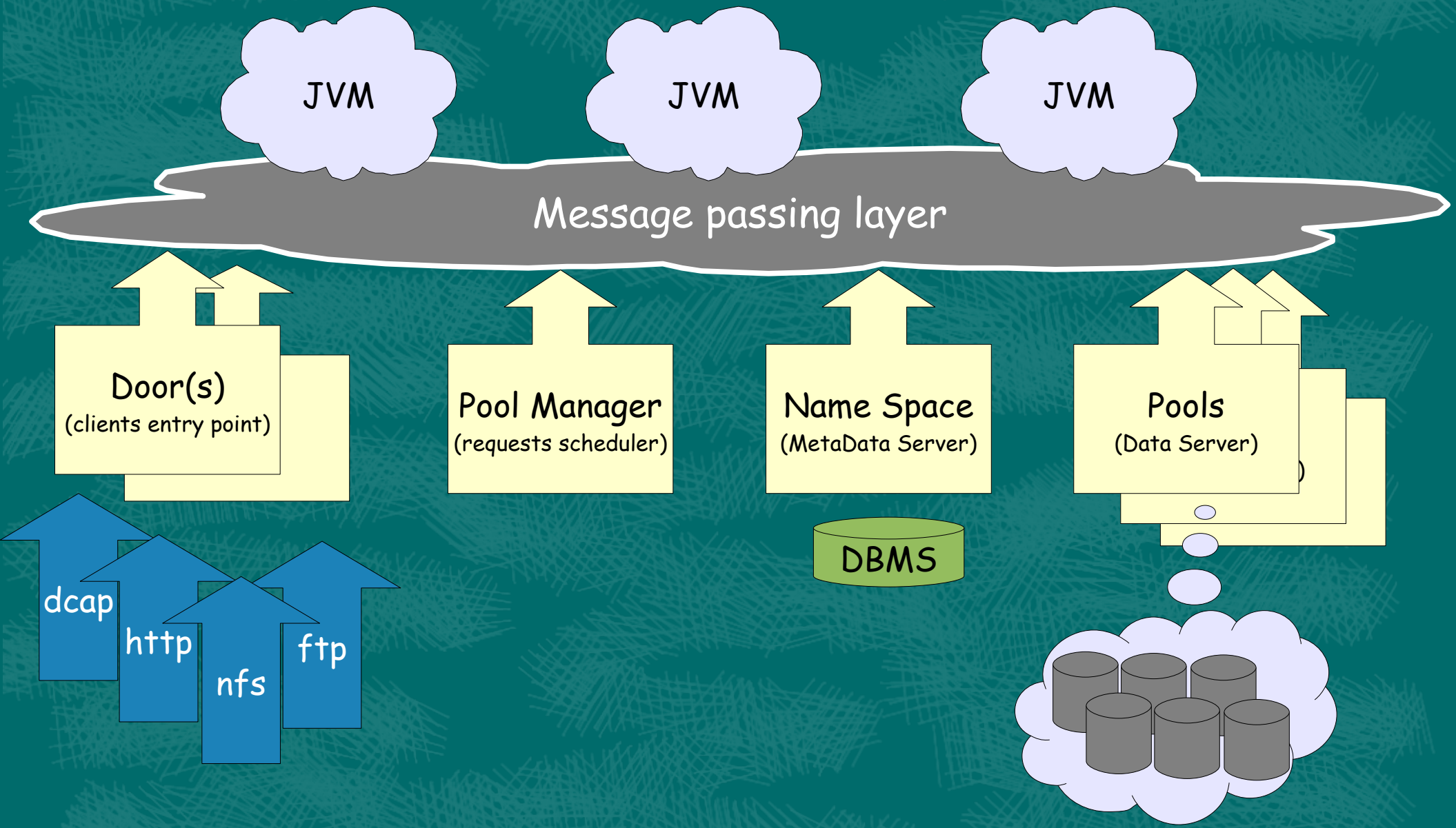
dCache NFSv4.1 server

Tigran Mkrtchyan
for dCache Team

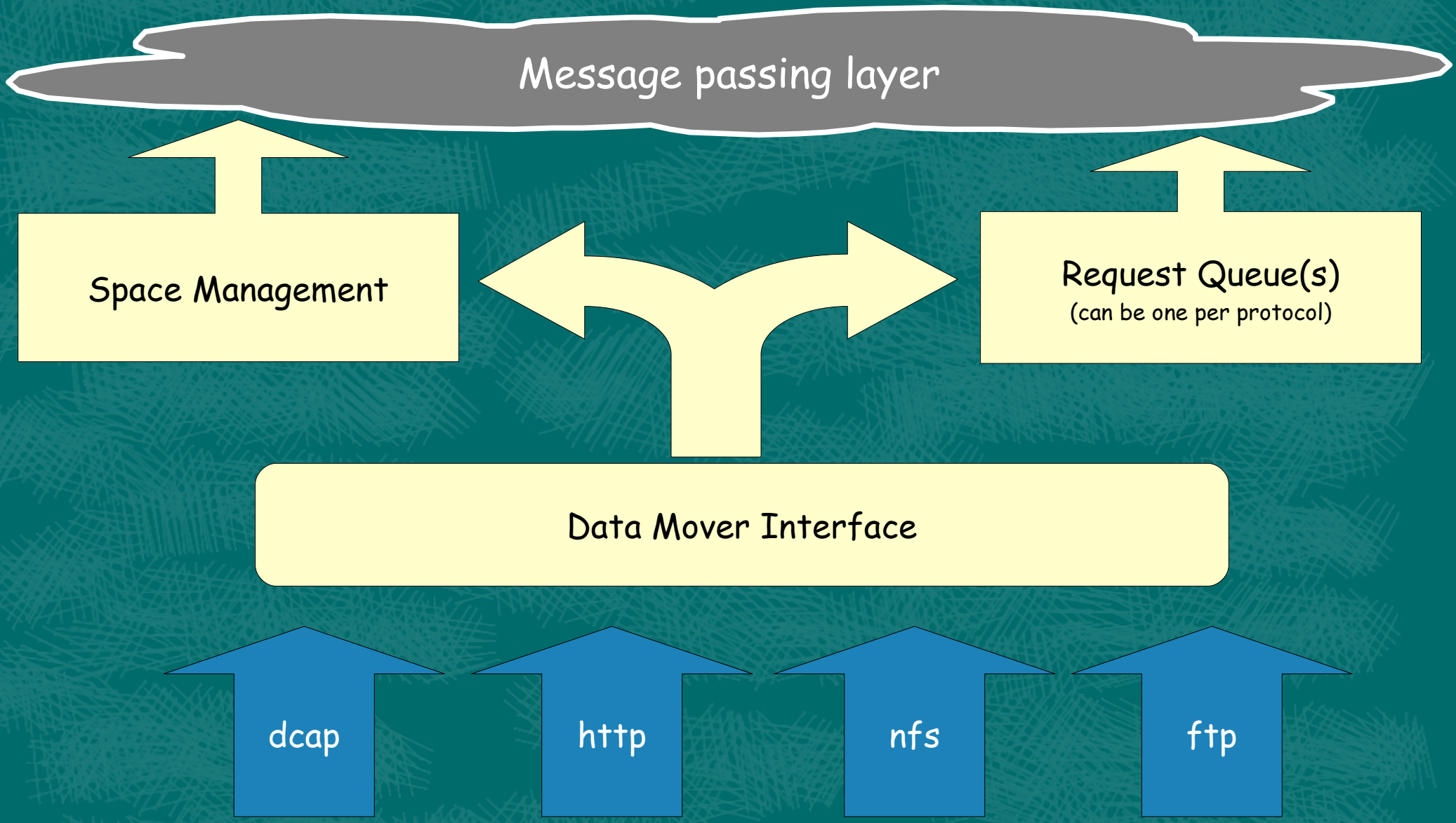
dCache design



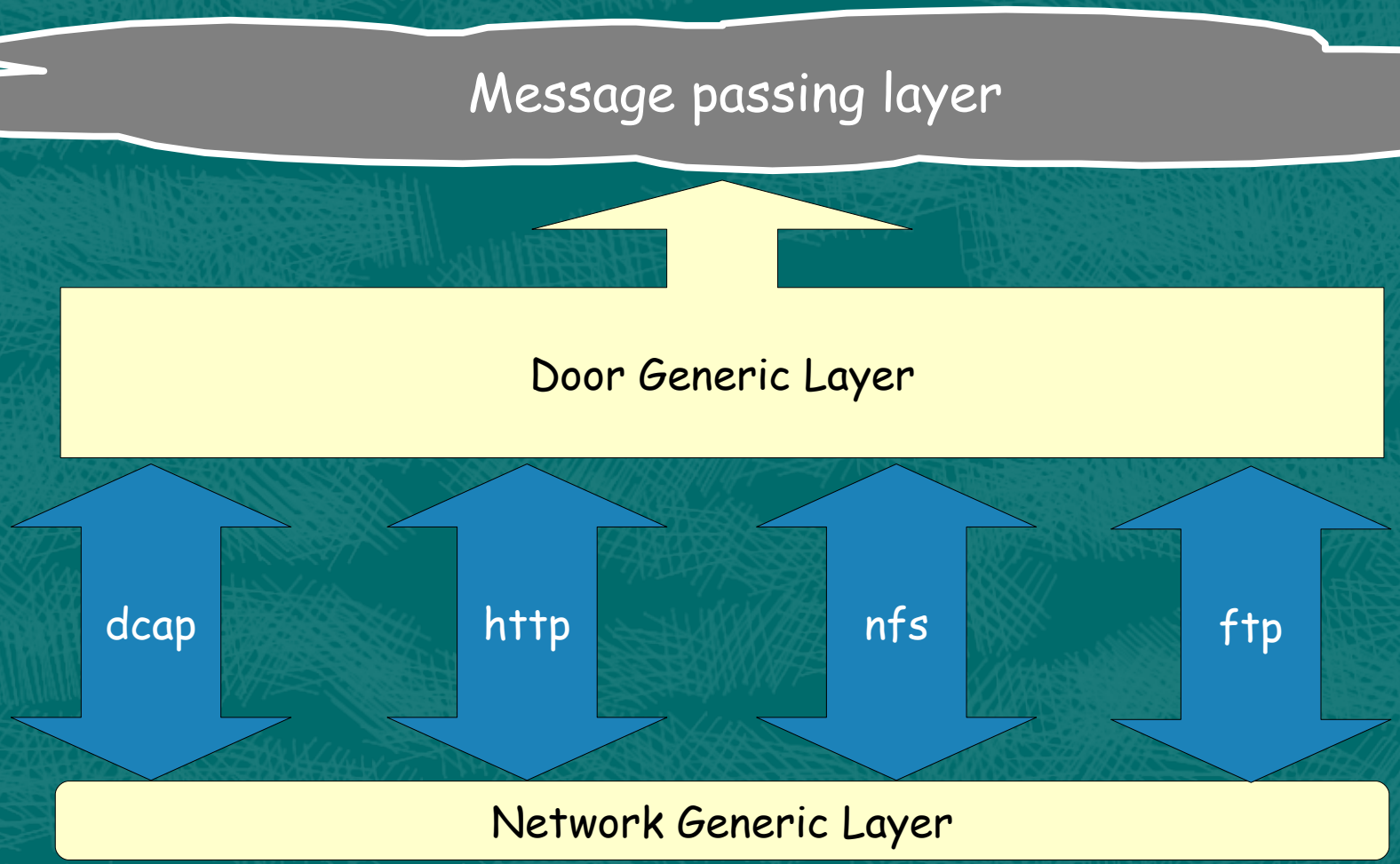
Pool Internals

- More or less like an ***Object Store***
- Metadata (size, checksum, acl, ...) stored in the namespace
- Local cached copy of metadata for inventory verification on startup
- Late data mover protocol binding
- Multiple protocols can be used to access same file
- policy driven (last access, total time, protocol) idle clients can be dropped
- Updates matadata ***on close***

Pool internals



Door internals

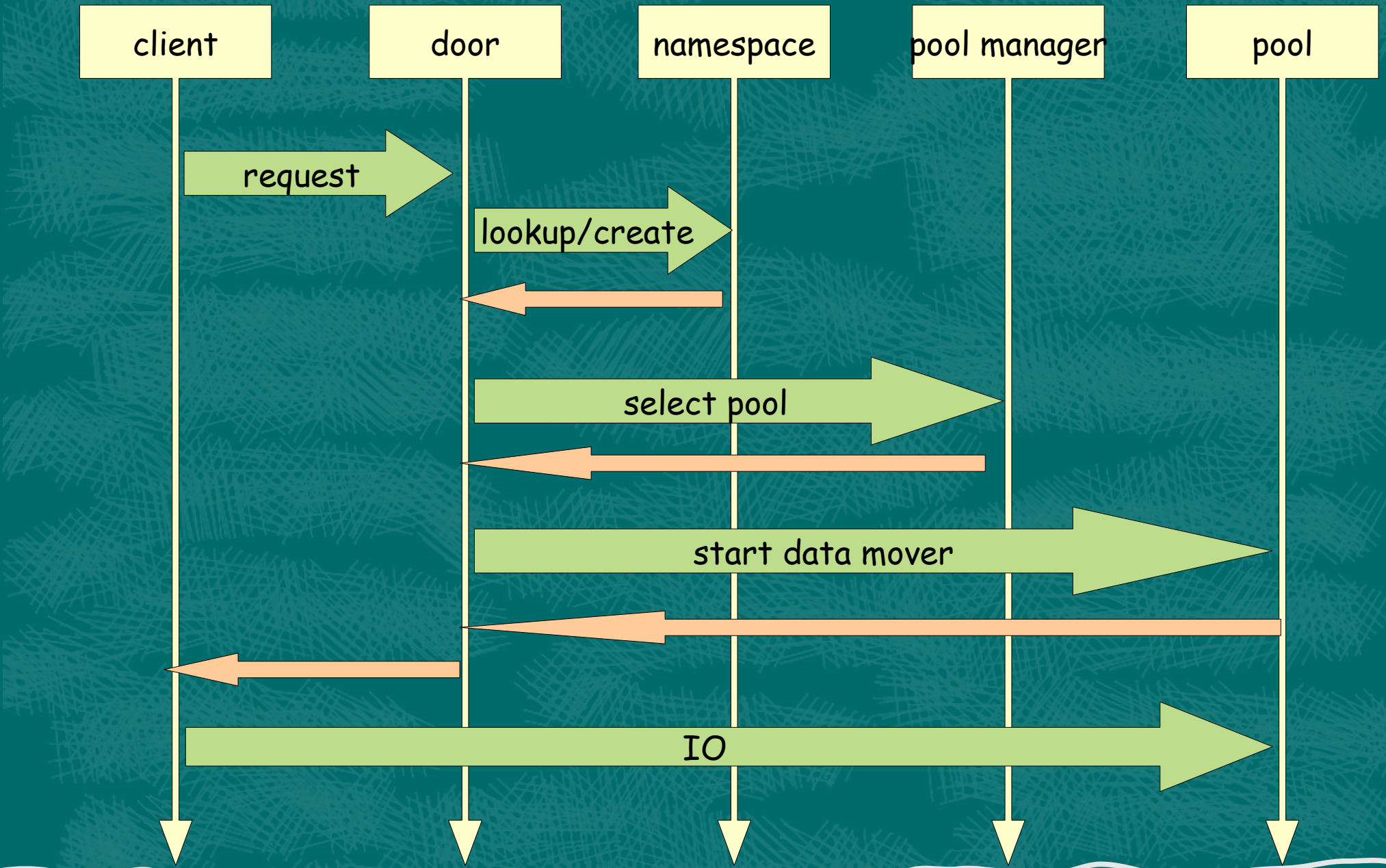


Disk Cache Access Protocol (dcap)

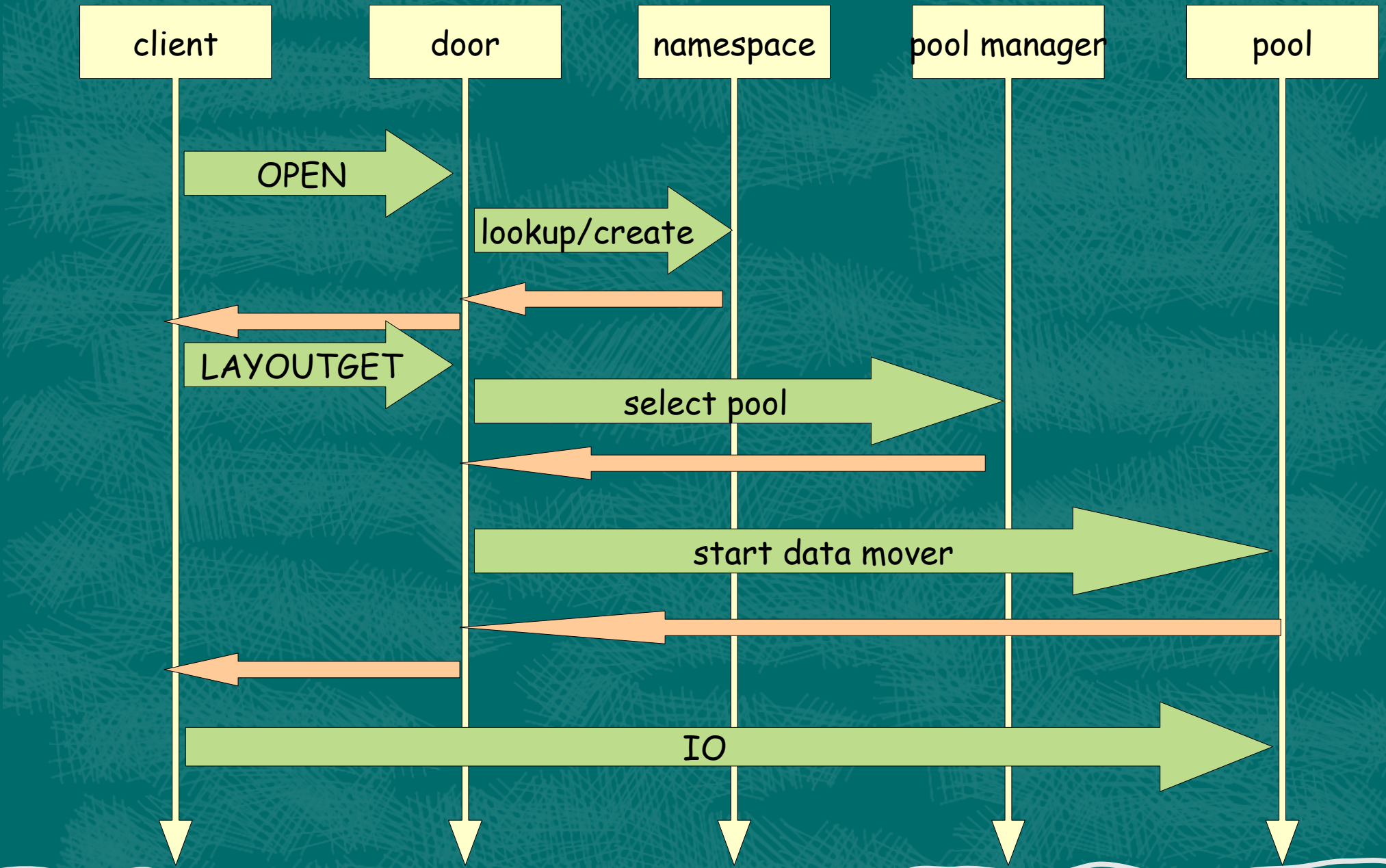
- Our attempt to invent NFSv4.1
  - independent path for metadata operations and IO (control and data lines)
  - *poor man* sessions on control line
  - READ/WRITE/CLOSE on data line
  - data line bind to TCP connection (which is actually bad)
    - on client disconnect all associated resources are freed
  - client library for the most platforms used in Physics

	DCAP		NFSv4.1	
	mds	ds	mds	ds
OPEN	X		X	
READ		X		X
WRITE		X		X
CLOSE		X	X	
LOOKUP	X		X	
READDIR	x	X	X	

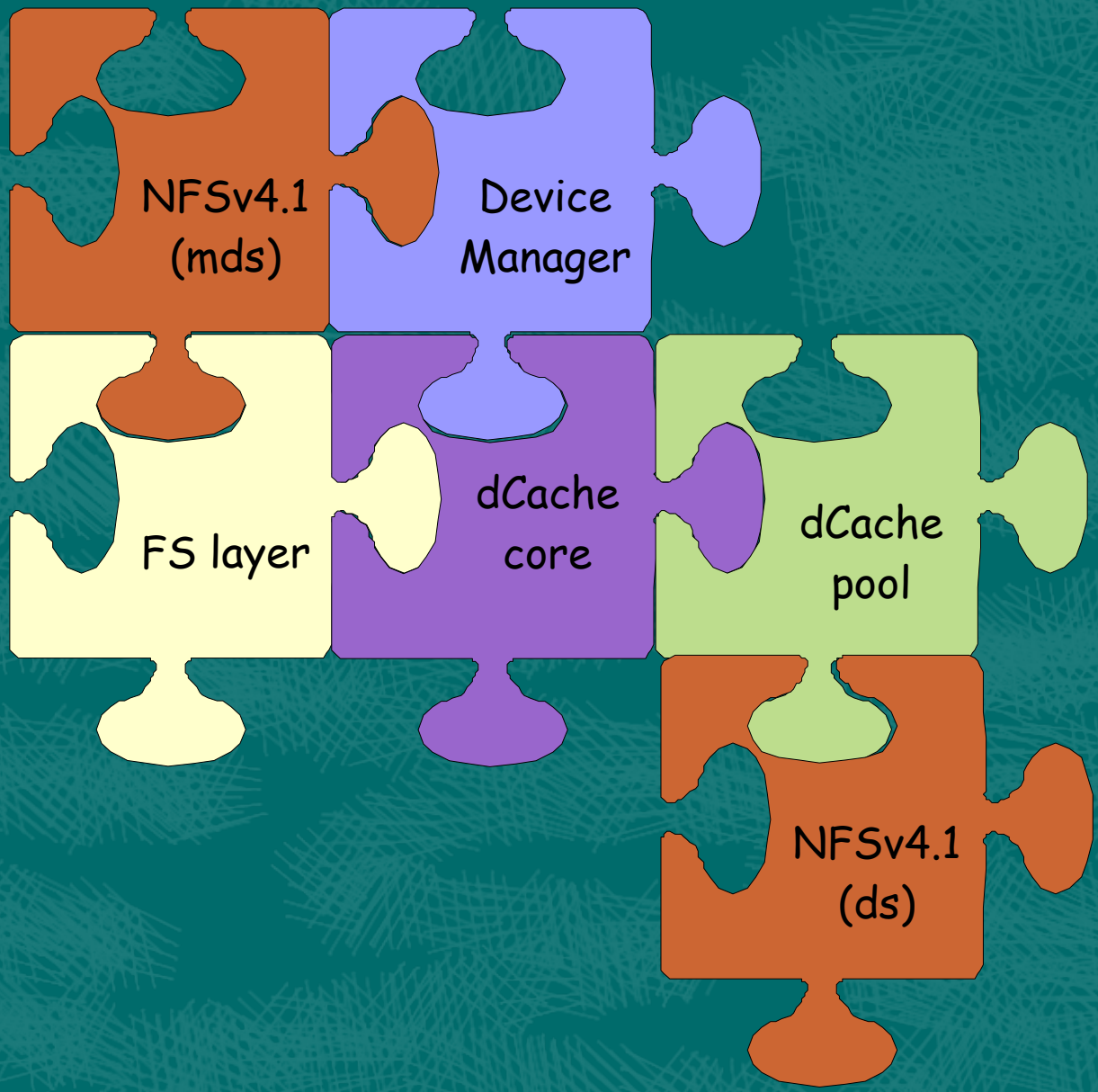
Request Sequence Diagram



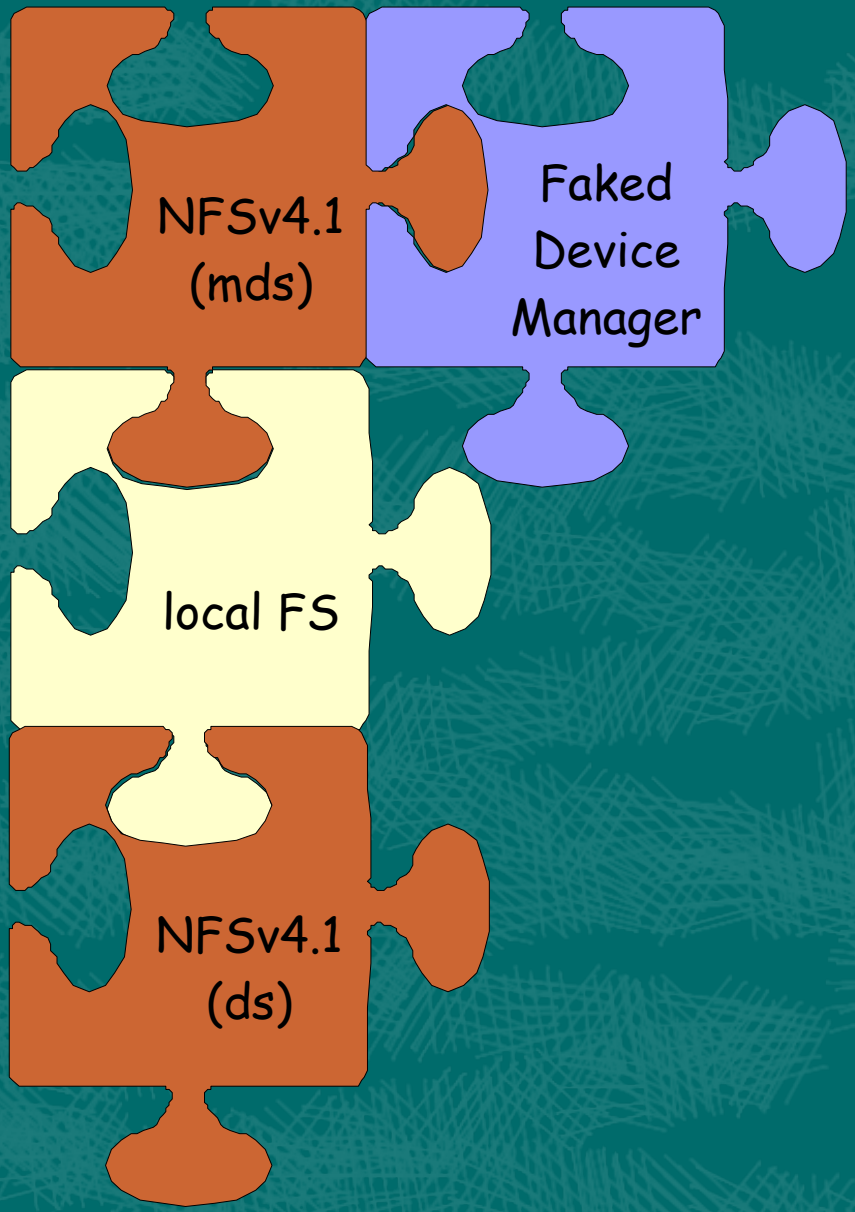
Request Sequence Diagram (pNFS)



NFS server internals



(what I usually run on Bake-a-thon)



Little bit of OO

```
case nfs_opnum4.OP_PUTFH:
       return new OperationPUTFH(call);
case nfs_opnum4.OP_READ:
      return new OperationREAD(call);
case nfs_opnum4.OP_WRITE:
      return new OperationWRITE(call);
```

MDS

DS

```
case nfs_opnum4.OP_PUTFH:
       return new OperationPUTFH(call);
case nfs_opnum4.OP_READ:
      return new DSOperationREAD(call);
case nfs_opnum4.OP_WRITE:
      return new DSOperationWRITE(call);
```

Ok, not a rocket science, but

- No stared globals between operations
- No requirement to support NFSv3/2 at the same time
- No shared API with other components (no vnode, nnode and so on)
- User space code (!!!)
- I pay for it with performance, you pay for it with lines of code
  - our NFS server is ~ 8K lines of Java code

Easy to create custom servers

```
case nfs_opnum4.OP_PUTFH:
      return new OperationPUTFH(call);
case nfs_opnum4.OP_READ:
     return new OperationFailOnSecondREAD(call);
case nfs_opnum4.OP_WRITE:
     return new OperationWRITE(call);
```

Benefits for me

```
@Test
 public void testWrongIOmode() {
    .....
        call = ... ; // generate request with BAD IO MODE
         operationLayoutGet = NFSv4OperationFactory.getOperation(call);
        opResult = operationLayoutGet.process();
        assertNFSState("Invalid IO mode did not return BADIOMODE",
                NFS4ERR_BADIOMODE, opResult.getStatus());
}
```

- ~130 different unit tests (do not require server to run)
- some test cases stacked into poor man client
- can be turned into functional test suite

/* FIXME: */

- Callbacks
  - Finally I got bidirectional RPC to work
- Infrastructure for byte-range lock
  - dCache's internal architecture supports create once read many
- Striping on read and write
  - we can't really stripe on write due to backend tape system
- GSS authentication
  - what we need is actually X509
- Re-implementation of sessions
  - current one is ugly
  - reply cache is missing

Thank You!

Code and Info

@

http://www.dcache.org