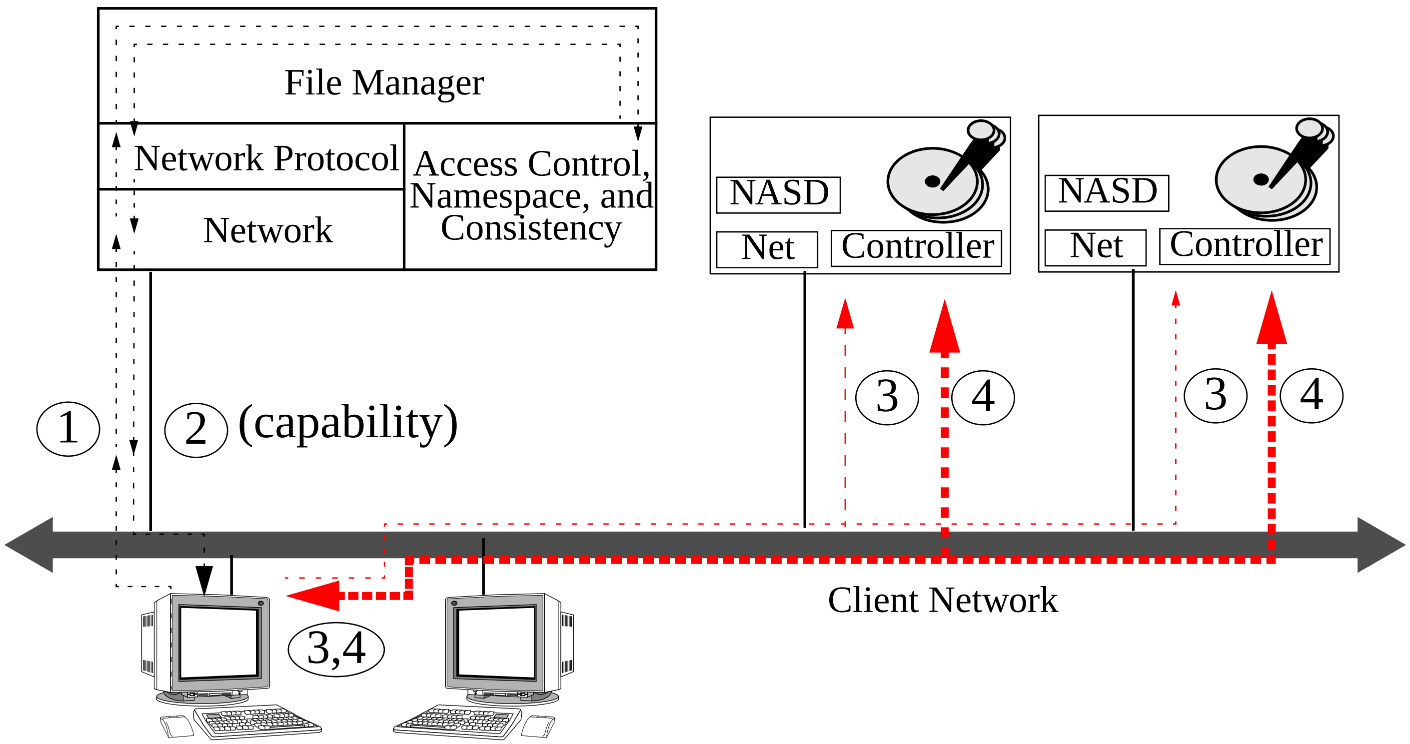
CMU's NASD:
Network-Attached Secure Disks

Garth Gibson
D. Nagle, K. Amiri, H. Gobioff, E. Riedel, J. Zelenka
Computer Science and Computer Engineering, CMU

Sponsored by DARPA/ITO Quorum/Scalable Systems
and HP, Quantum, Seagate, STK, Symbios, Clariion, Compaq, Wind River, Intel, 3Com

Avoid file manager unless policy decision needed

- access control once (1,2) for all accesses (3,4) to drive object
- spread access computation over all drives under manager
- Scalable BW, off-load manager, “file” knowledge



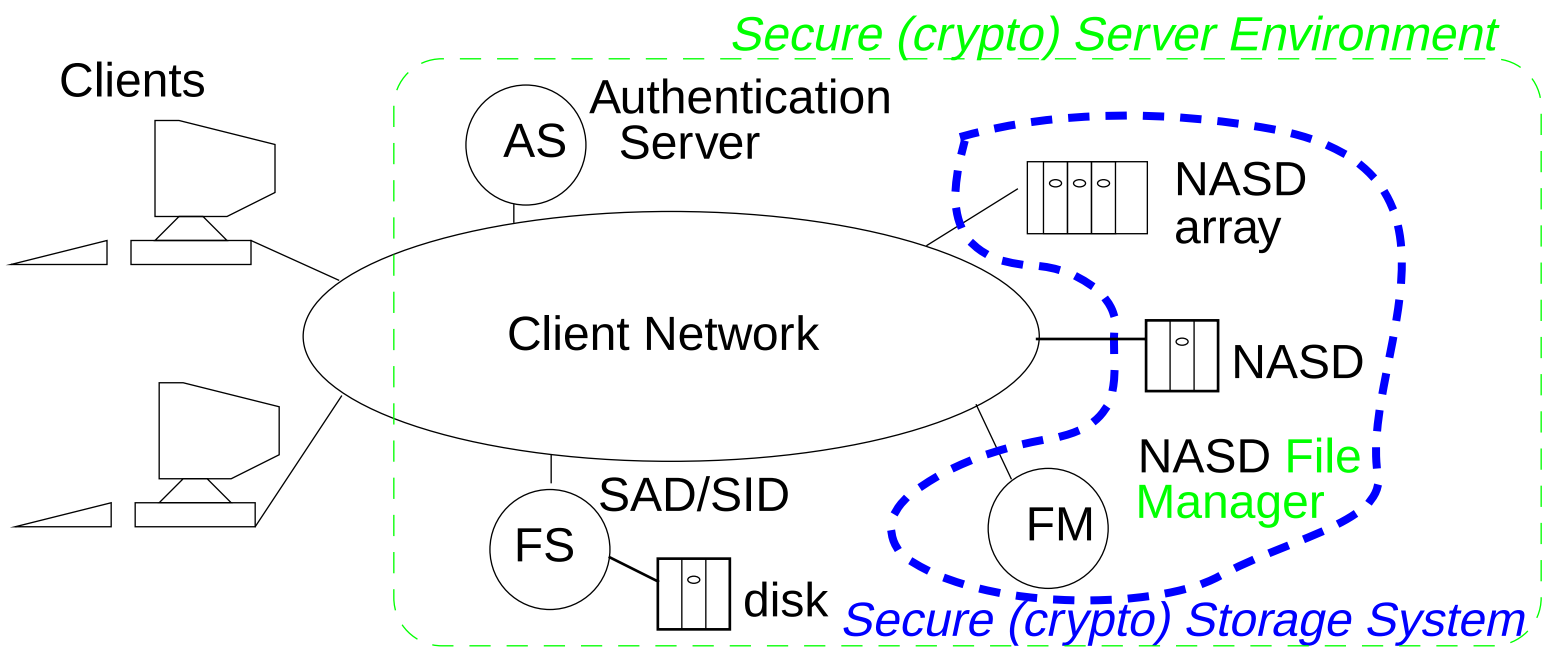
Security implications of network-attached storage

Not tied to any specific higher-level security system

- **ie., not Kerberos, authenticated RPC, x.509**

Authenticates command to be file manager approved

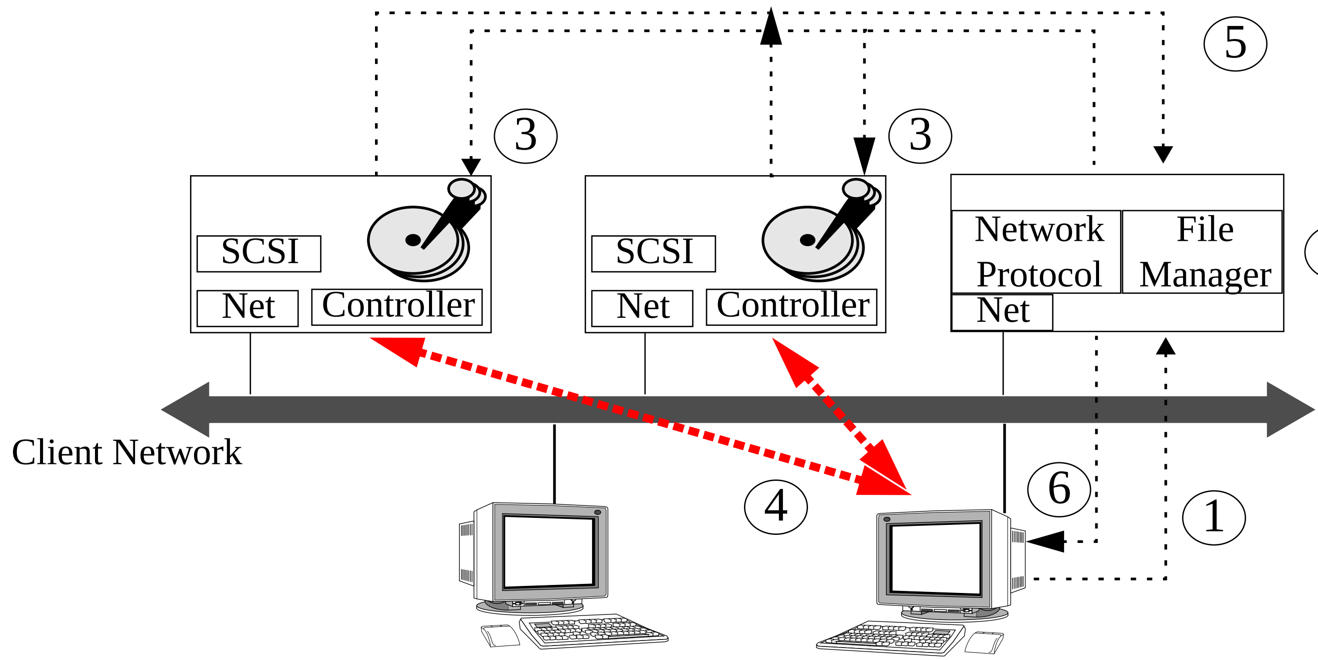
- **rests on secrecy of file manager key (hierarchy) only**



Applies to Networked SCSI too

Third-party transfer drive/client exposed to tampering

- **integrity of transfer step (4) requires support from drive**
- **private, secure storage bus (steps (3) and (5)) not enough**
- **cryptographic checksum on step (4) provides integrity**



NASD security protocol: integrity protection

Clients “carries” access rights to NASD drive

- manager builds Capability, sends to client to “carry” to drive
- Capability = Digest(Key,Drive,Object,Version,Rights,Expiry)
- Key is secret between manager and drive (really 1 of 4 keys)
- request for Operation on Object sent by client to Drive:
  Operation,Object,Rights,Expiry,Digest(Capability,Operation)

Drive must enforce prior manager authorization

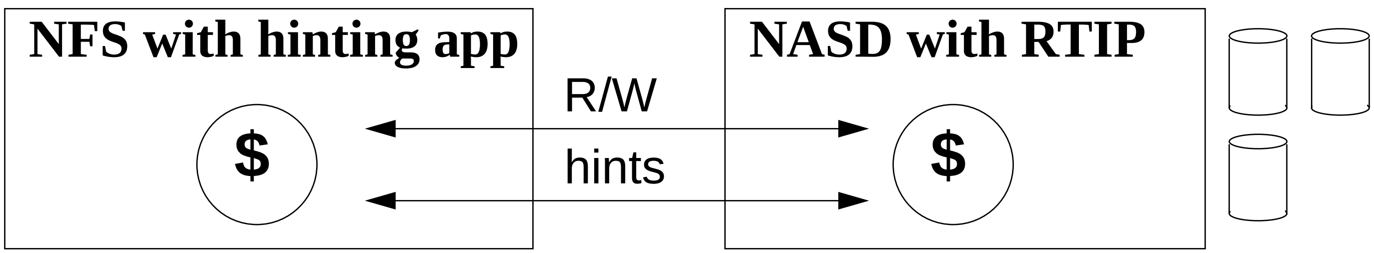
- drive computes capability, operation digest on each request
- manager revokes Capability by 1) letting it expire, or
  2) advancing Object’s Version on drive
- no explicit message to drive with each client open
- drive can reduce digest costs by caching capabilities

NASD object interface: Where is file metadata?

Not at client: don't rest integrity on trusted client

Data Layout in storage device ?

- **avoid distributing per-drive block lists to drive**
- **enables on-drive, drive-subsystem optimization**
  ie. AutoRAID; deleted space recovery
  ie. interposed/stackable NASD - striping, RAID
  ie. remote Transparent Informed Prefetching

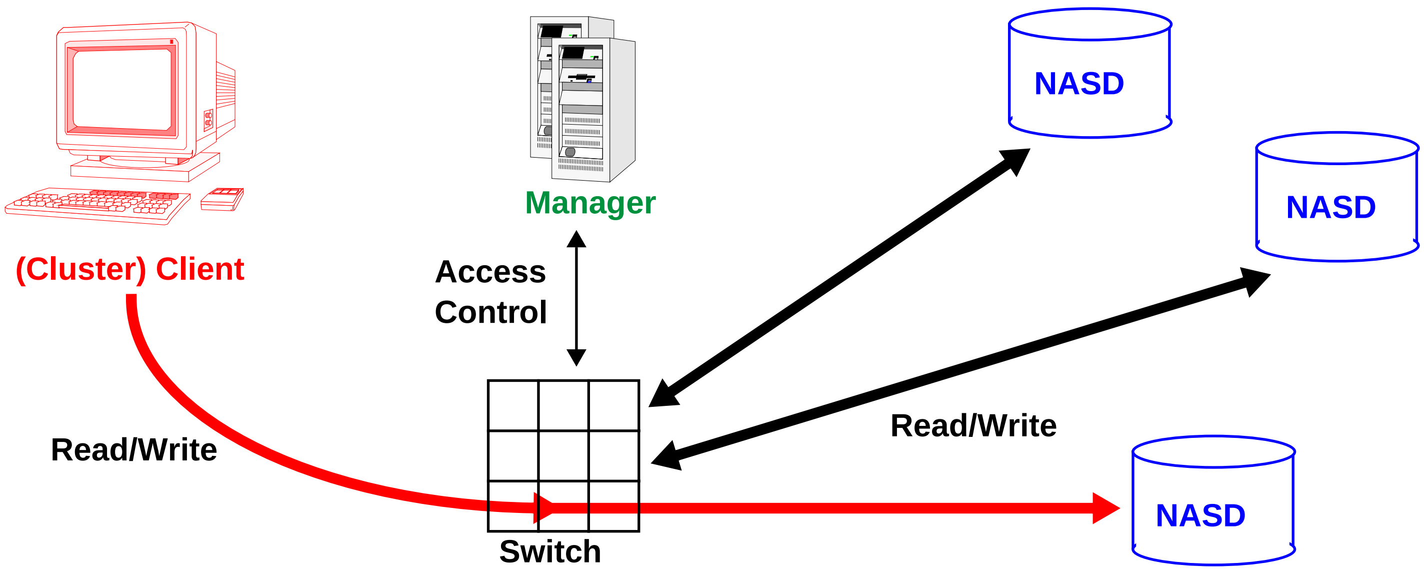


- **RTIP in NASD**
- **XDS rendering 25 planes from 64 MB**
- **data striped on 3 disks**

	NFS
Hints	68s
Nohints	120s

CMU's Functional Definition of NASD

- **Direct** **client/drive** **transfer** **in networked environment**
- **Asynchronous** **filesystem** **oversight** **of rights, semantics**
- **Cryptographic** **capabilities ensure command integrity**
- **Self-management** **by more abstraction, independence**
- **Extensible** **features for application, not just client OS**



NASD Interface Design: Storage Objects

Layout is best (actually) done below SCSI-4

- real-time support possible; accurate geometry
- simplifies, strengthens transparent performance optimization

Drive serves storage objects on behalf of manager

Objects have attributes

- Inodes: (name), size, protection, type, timestamps, layout

Object layout guidance from higher level

- sequential within object requests contiguous (can preallocate)
- related objects can be clustered with “nearby to” attribute

Attributes are extensions of object name

- give higher level (filesystem) uninterpreted attribute
- big enough to contain another object name (soft link)

Supporting File Managers: Soft Partitions, Well-known Objects

Split capacity among different managers: partions

- managers can use attributes differently; no need to integrate
- boundary should be soft: resize partition should be fast
- flush partition enables fast acquiese
- partition key hierarchy: (partition key, working keys)

Well-known objects replace SCSI mode sense/select

- published format and interpretation
- Per-drive and per-partition separated
- Partition table of contents (mini-disk directory)
- Assist simple boot code with easily found “first object”

Adapting Filesystems to NASD

Reorganize decomposition of function (aka port)

Primitives become drive responsibility

- data transfer, synchronous/automatic metadata updates

Policy remains manager responsibility

- namespace definition/navigation
- access control policy
- client cache managment
- multi-access atomicity

Managers retain control through capabilities

- exploiting attributes for naming and revocation
- restricting client operations to protect “set attribute”

Mapping Filesystem to NASD Objects

Objects: attributes, access control, clustering

Simple model

- each file and directory bound to separate NASD object
- file attributes inherit object attributes (times, logical size)

Multiple objects per file?

- internal structure: database pages, mpeg group-of-pictures
- NASD striping, redundancy

Multiple files/directories per object?

- probable contiguity, prefetching; shared metadata overhead
- capabilities can be restricted to object region

NFS, AFS simple model; Striped NFS multiple per file

Prototyping NASD: NFS & AFS on NASD

File -> NASD object; Directory -> NASD object

NASD object: private metadata, exposed attributes

- **allocation: length, blocks used; times: create, data modify**
- **FS specific: NFS: owner, group, mode**
- **FS specific: AFS: above and modify time**

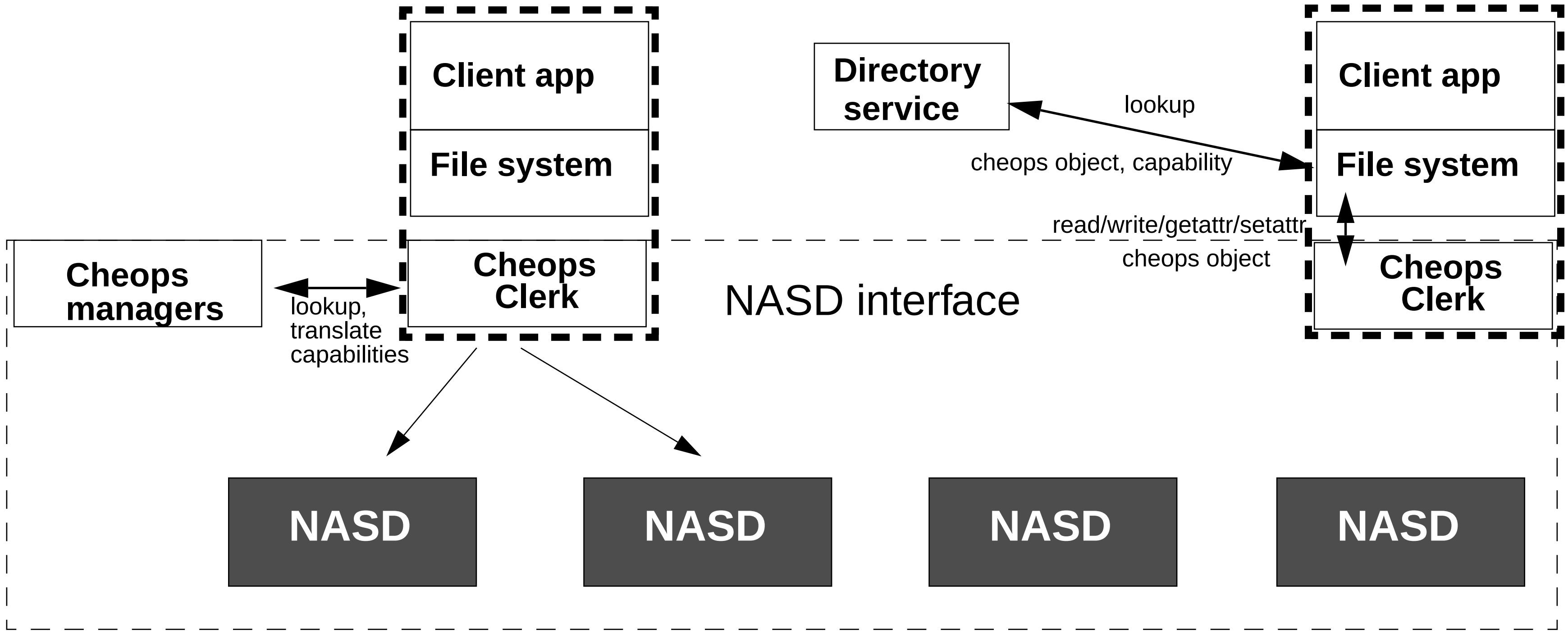
Operation disposition

- **NFS: to drive: get attribute, read, write**
- **AFS: to drive: FetchStatus, BulkStatus, FetchData (w/cap), StoreData (w/cap)**
- **AFS: Read w/o cap: GetCap (callback, attributes), (GetAttr from drive), FetchData**
- **AFS: Write w/o cap: GetWCap, StoreData, ReturnCap**
- **AFS cache coherence protocol independent of NASD**
- **AFS quota enforced by capability escrow (using write range)**

Cheops: Striping storage middleware

Transparent, scalable bandwidth, RAID, optimistic client synchronization (fs-specific attributes)

Storage management architecture parallels file management architecture (uses capabilities)

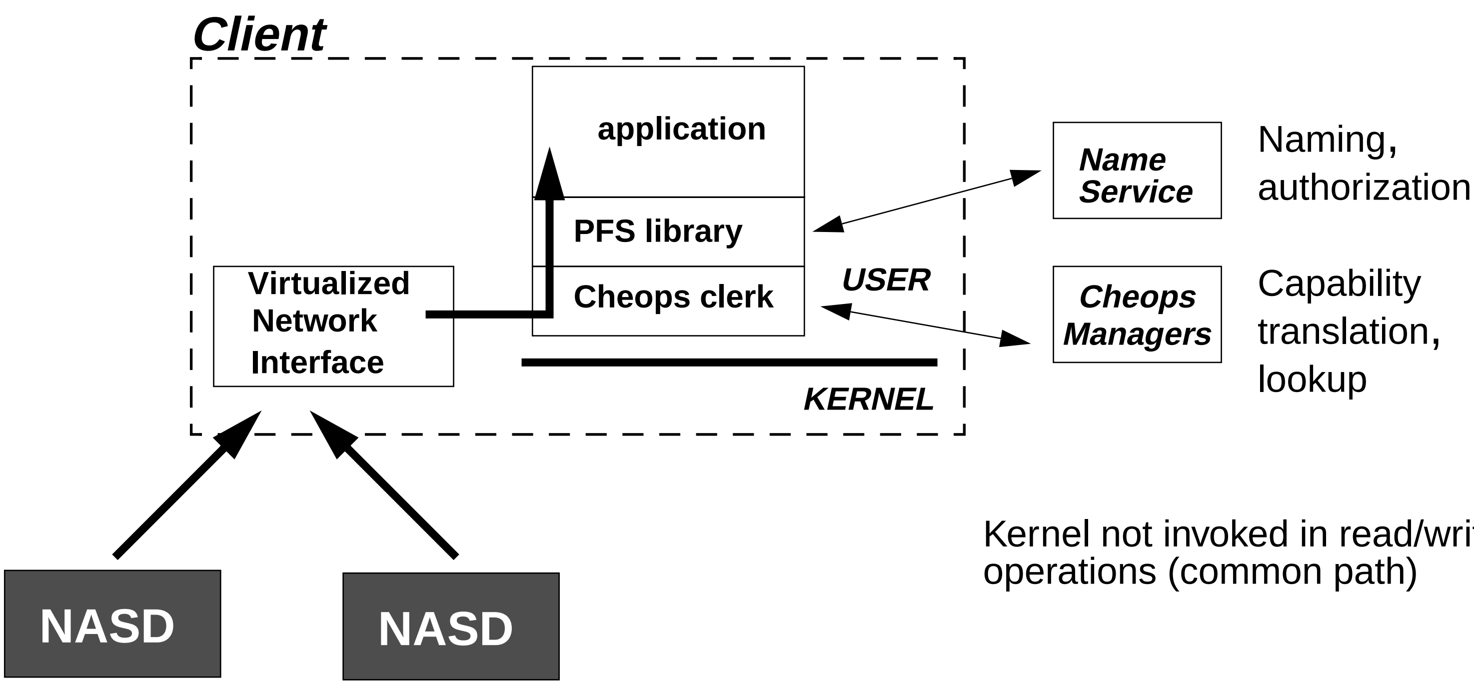


Simple Cluster Parallel FS on Cheops on NASD

Client asking for service pays for it (synchronizer)

- striping, RAID, consistent caches, collective operations

Entirely user level, incl. messaging, for low latency



Kernel not invoked in read/write operations (common path)

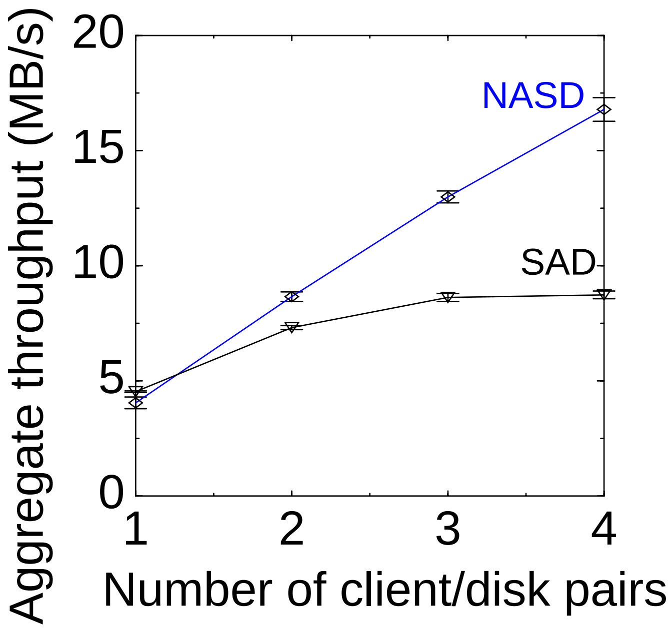
Experimenting with striped NFS-NASD prototype

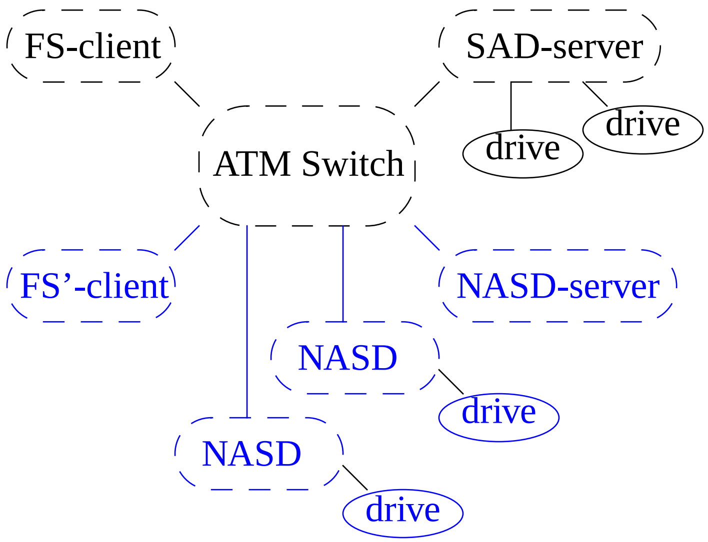
Transparent function extension through NASD stacks

- **NFS-NASD FM issues capabilities on a psuedo-object**
- **Psuedo-object managed by NASD-striper**
- **After first touch by each, direct client-drive transfers**

Experiments on DEC Alpha testbed; DCE on OC3

- **aggregate random large read BW scales with client/drives**

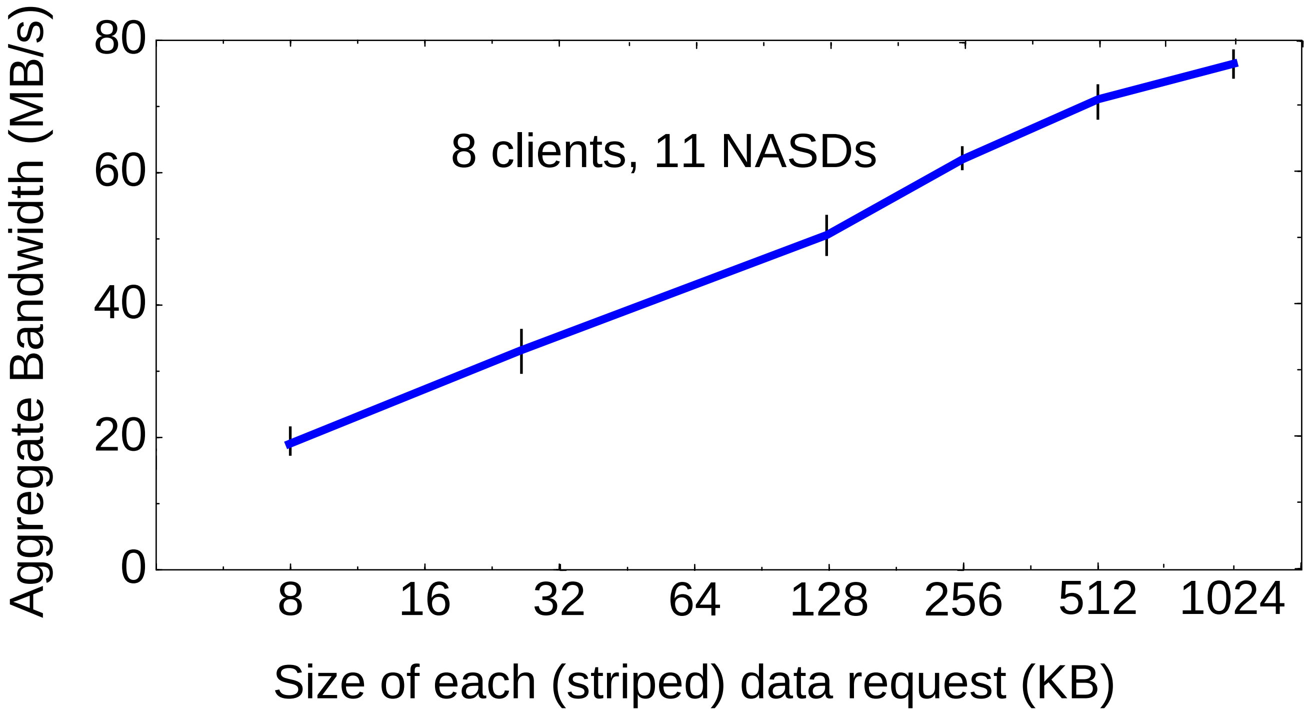




Its about Scaling, remember?

Concurrent reads of data striped on many NASDs

- **cached data in 2nd prototype NASD (peak 9MB/s)**
- **with small requests, DCE/RPC overhead substantial**



Recap: NASD Filesystems are Policy Servers

Direct transfer for wire-once, scalable bandwidth

- NetSCSI for large object bandwidth
- NASD for object bandwidth and server offloading

NASD filesystems serve policy (async oversight)

- namespace, access control, consistency, atomicity
- capabilities encode policy, metadata; crypto integrity
- capabilities cause drive to understand variable length object

Storage management middleware

- clients pay for requested synchronizing semantics
- striping, RAID, incremental capacity, migration
- optimistic synchronization using fs-specific attributes

CMU's Functional Definition of NASD

- **Direct** client/drive **transfer** in networked environment
- **Asynchronous** filesystem **oversight** of rights, semantics
- **Cryptographic** capabilities ensure command integrity
- **Self-management** by more abstraction, independence
- **Extensible** features for application, not just client OS