Storage services

requirement to support:

* program/document storage, development

* application/system program-load and run-time data access

* application-level services

  databases, CAD

  email, naming directories
  web, newsgroups

* ubiquitous environments

  storing applications to run on home systems

* mobile users - detached operation

  copy, disconnect, reconnect

* integration of new media

  continuous media: audio, video
  need Quality-of-Service QoS guarantees

  servers may be dedicated e.g. VOD

* new services

  e.g. indexing and retrieval support
  over different media

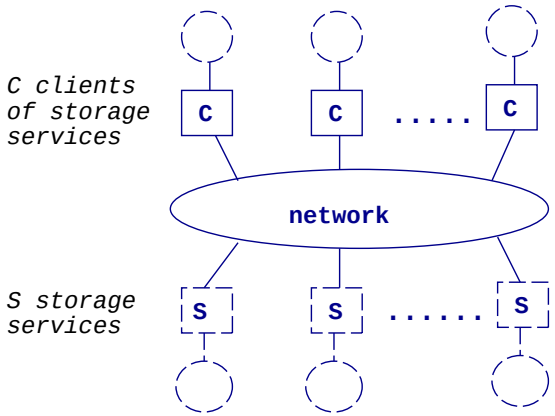
* composite documents

  with components of different media types

* linking related information across files

Storage in a Distributed System

consider a single domain:



clients may not have local discs:
  V system
  network computer

client local discs:
  for private system (local desktop - Xerox)
  part of shared filing system - Unix mount
  cached files only?
  system files for bootstrapping?

Storage service functionality

* open or closed?

  is it bound into a single OS model
  e.g. single pathname format?

* distribution of functionality

  - storage and retrieval of data
  - name resolution (directory service)
  - access control
  - existence control (garbage collection)
  - concurrency control

* level of interface

  - remote blocks

    minimal overhead at server
    client system does block layout?

  - remote UID-named files

    server does block layout - more overhead

  - remote path-named files

    bound into a single OS

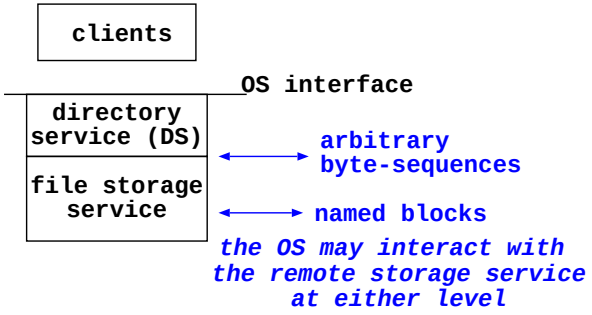
* caching and replication

  is the service responsible for managing:

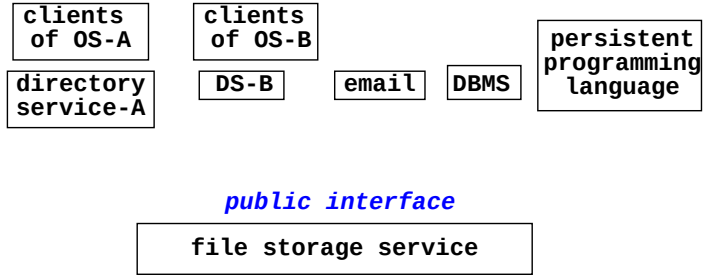
  - multiple cached copies of a file?
  - replicas of a file (replicated for reliability)

  or are these application-level concerns?

a)closed architecture

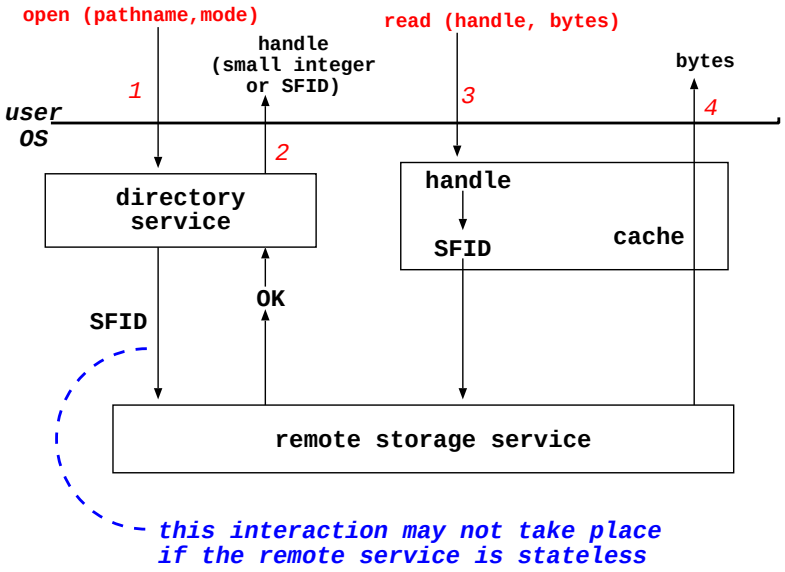


b)open architecture



remote interface at file-level

SFID = system file-identifier



operations in remote interface

SFID ⟷

create		
delete	(SFID)	?
read	(SFID,range)	
write	(SFID,range)	
lock	(SFID)	?
open	(SFID)	?
close	(SFID)	?

? depend on design decisions

Does the server hold state?

* NO specified as stateless e.g. NFS

  - simple crash recovery

  - can't help with concurrency control
  - can't help with cache management
    (have to ask for time-last-modified)

* YES
  - supports open/close
    holds who has files open/mode

  - crash recovery - need to interact with
    clients to rebuild state

  - concurrency control
    exclusive or shared lock
    better than write or read locks

Existence control

* storage service at file level can't help
  (doesn't see naming graphs)

* a directory service (multiple instances?)
  can do existence control for its own objects

  OK for a closed architecture

* what about
  - objects shared by different systems?
    (e.g. video clip in document)

  - objects not stored in directories?

* lost object problem

  SFID ⟷ create (...)

  reaches client's main memory only
  - crash -

* consider a "touch" operation in storage service
  untouched files deleted (archived)

* Cambridge File Server (ref paper 1980)

  - open - many OS clients
  - minimal support for structure
    without enforcing path-naming

CFS - the Cambridge File Server

developed as part of
the Cambridge Distributed Computing System (CDCS)
in the late 1970's and used as the Lab's research
environment through the 1980's

CFS provides:
  two primitive types: byte and UID

  two abstractions:
    file - an uninterpreted sequence of bytes
           named persistently by a PUID
           with a random component

    index - a sequence of PUIDs
            itself named by a PUID

Indexes are used by CFS's clients to mirror their
directory structures

all index operations are failure atomic
(all or nothing is done)

* existence control:

  indexes form a general naming network starting from
  a specific root index

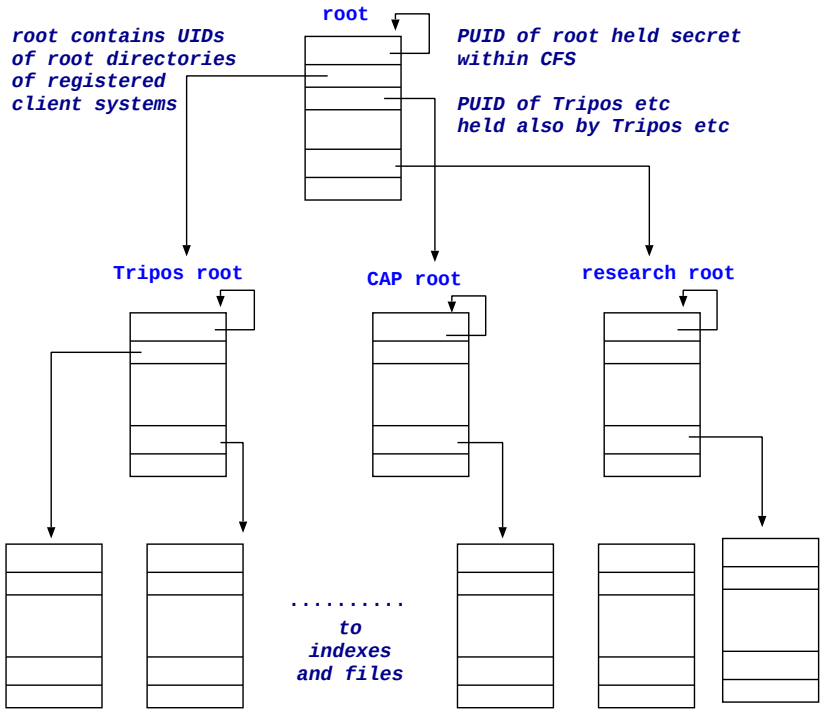
  objects are preserved while they are reachable from
  the root

  - reference counts are used - the number of times
    a UID is preserved in an index and
  - an asynchronous garbage collector is used for cyclic
    structures

* concurrency control - just MRSW

  open - a TUID is issued as a handle
         and the PUID locked
  TUIDs are timed out, 15 mins, reset on access
  close - release PUID lock

CFS index structure



some CFS operations

object-ID = file or index ID
PUID = permanent/preserved ID for closed object
TUID = transient ID for open object

open [object-ID, {read/write}] -> TUID

close [TUID, {commit,abort}] -> commit/abort

create-index [existing index-ID, entry] -> index-ID

preserve [index-ID, entry, object-ID] -> done

retrieve [index-id, entry] -> object-ID

delete [index-ID, entry] -> done

NOTE - no delete index - garbage collection

create-file [index-ID, entry, ...] -> file-ID

read [file-ID, offset, length] -> data

write [file-ID, offset, length, data] -> done

NOTE - no delete file - garbage collection

MSSA

Multi-Service Storage Architecture

- **Three file abstractions:**

  flat, structured, continuous medium

- **A file is built from byte segments**

  different byte segment implementations
  may be used to cater for
  different file characteristics

- **Files are named by storage-service identifiers SSIDs**

  -used as embedded references
  -applications can choose their
  own(textual)naming schemes

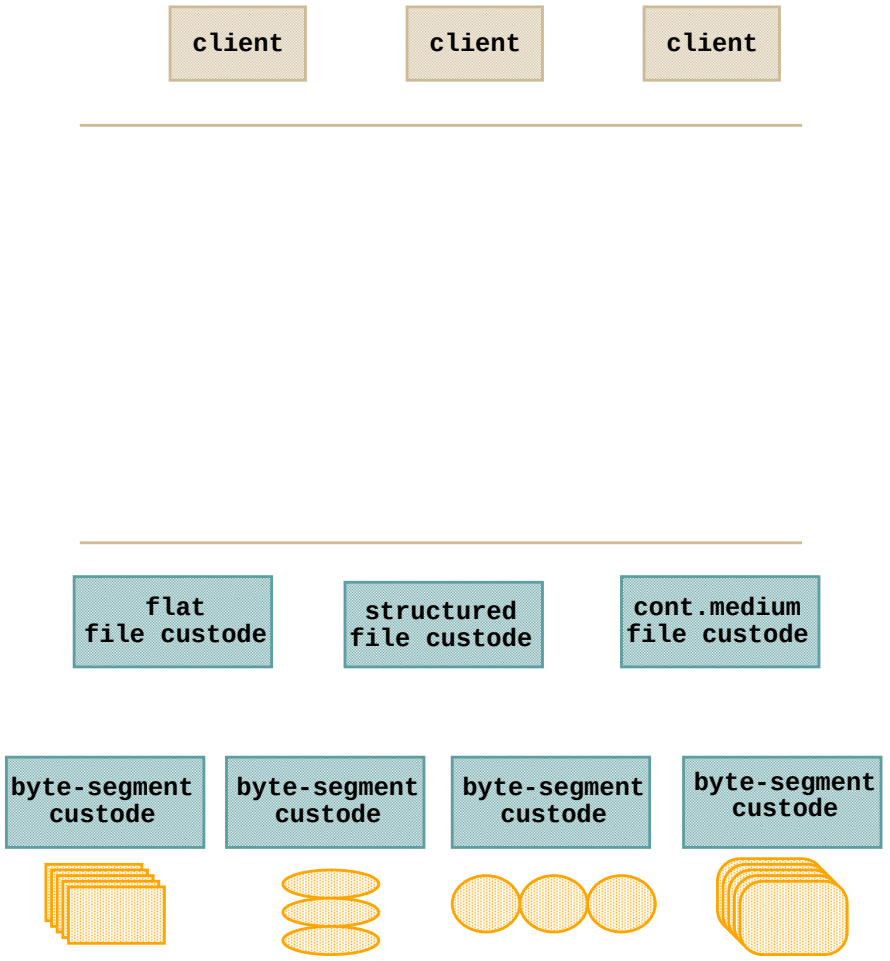
- **NVRAM used in storage substrate**

  high performance atomic update

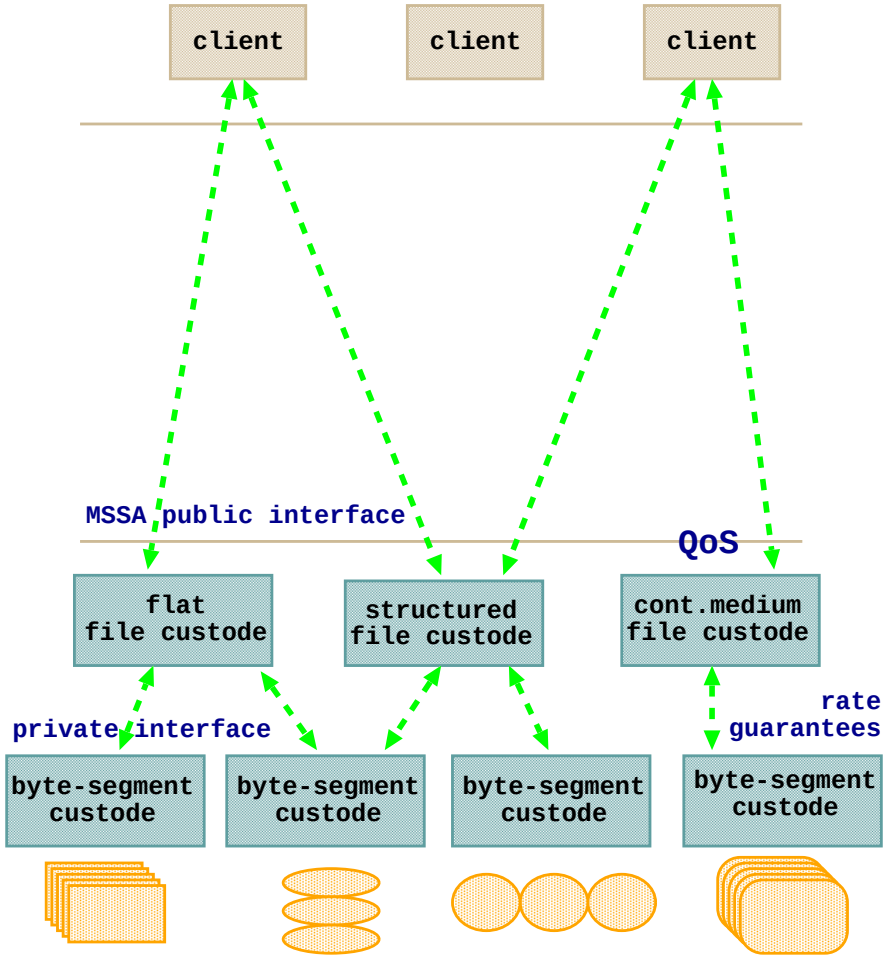
- **Value-adding service layers above MSSA**

  MSSA access control extends
  indefinitely through the layers

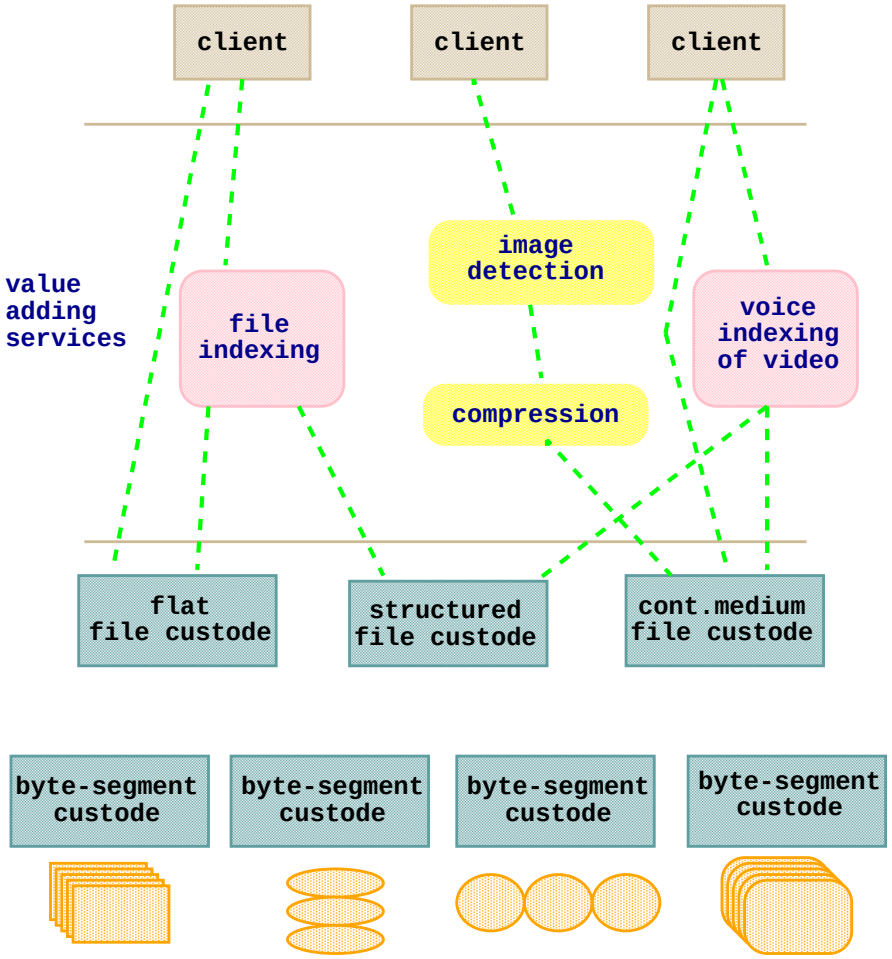
MSSA: Multi Service Storage Architecture



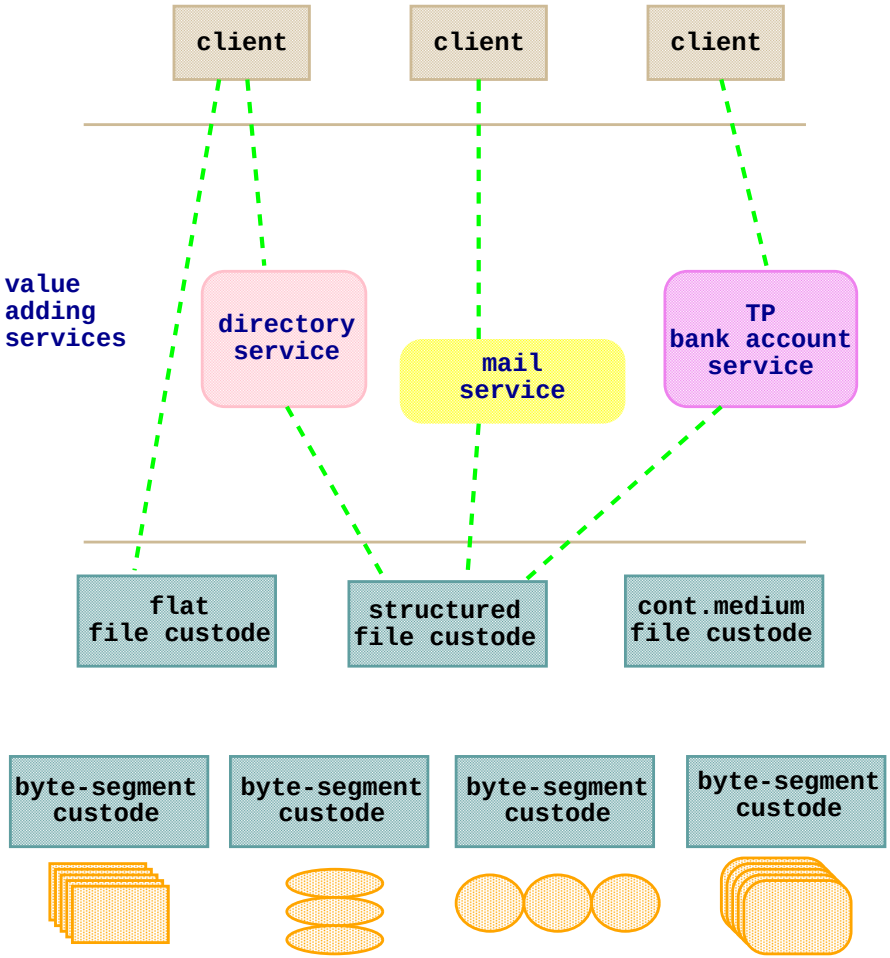
MSSA: Multi Service Storage Architecture



MSSA: Multi Service Storage Architecture



MSSA: Multi Service Storage Architecture



MSSA: Naming

- Any naming trees are external to the MSSA
- MSSA names are bit patterns: SSIDs
- SSIDs may be stored persistently e.g. in structured files
- SSID = container-name, object-name
  (32) (64)
  - unique within the object's class
  - never reused
- A single object can therefore be moved to another container
- optimisation: can often reuse the byte-segment instance number at higher levels

MSSA Files, byte segments and containers

STRUCTURED FILE CUSTODE (SFC)

Idea abstract out storage requirements common to emerging applications

Not a (language) type manager

But storage structure sufficient to locate SSIDs for existence control

(Sue Thomson PhD thesis 1990)

base types byte sequence

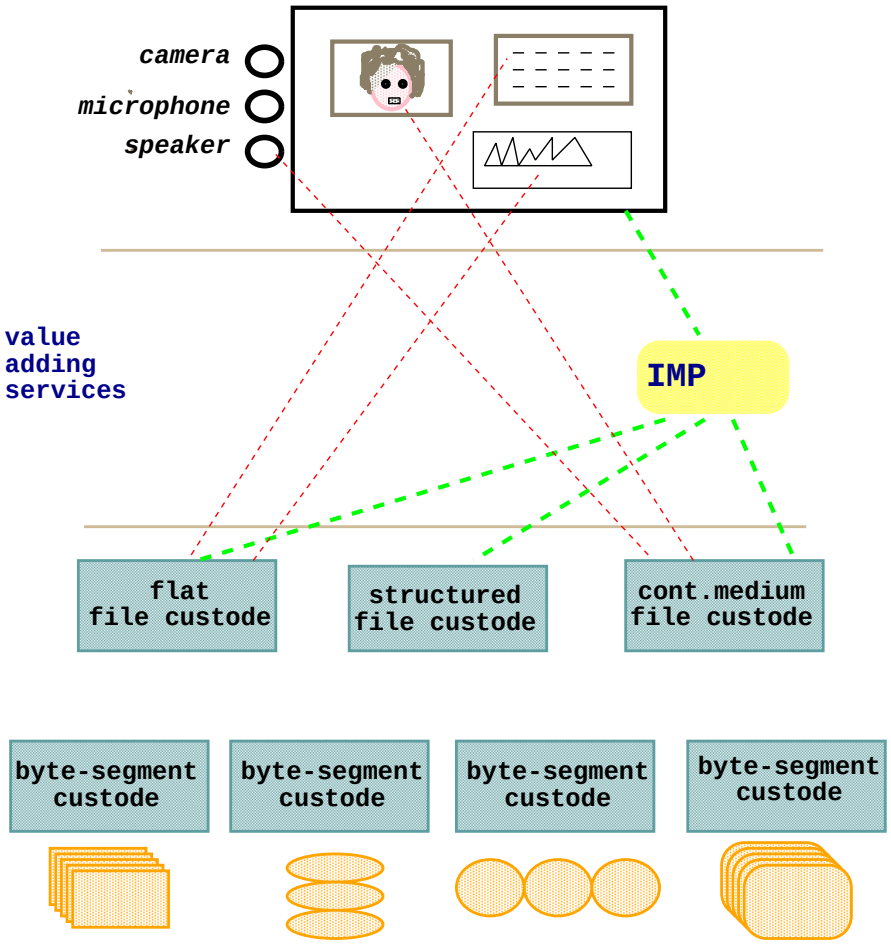
SSID

generators

sequence T*
record (T1,T2,...Tk)
union [T1+T2+.. +Tk]

set ?
bag ?

components may vary in length

storage type-structure may be changed to support structured documents

MSSA: Multi Service Storage Architecture