InformationSpace Design

ANSA Technical Committee

9 April 1998

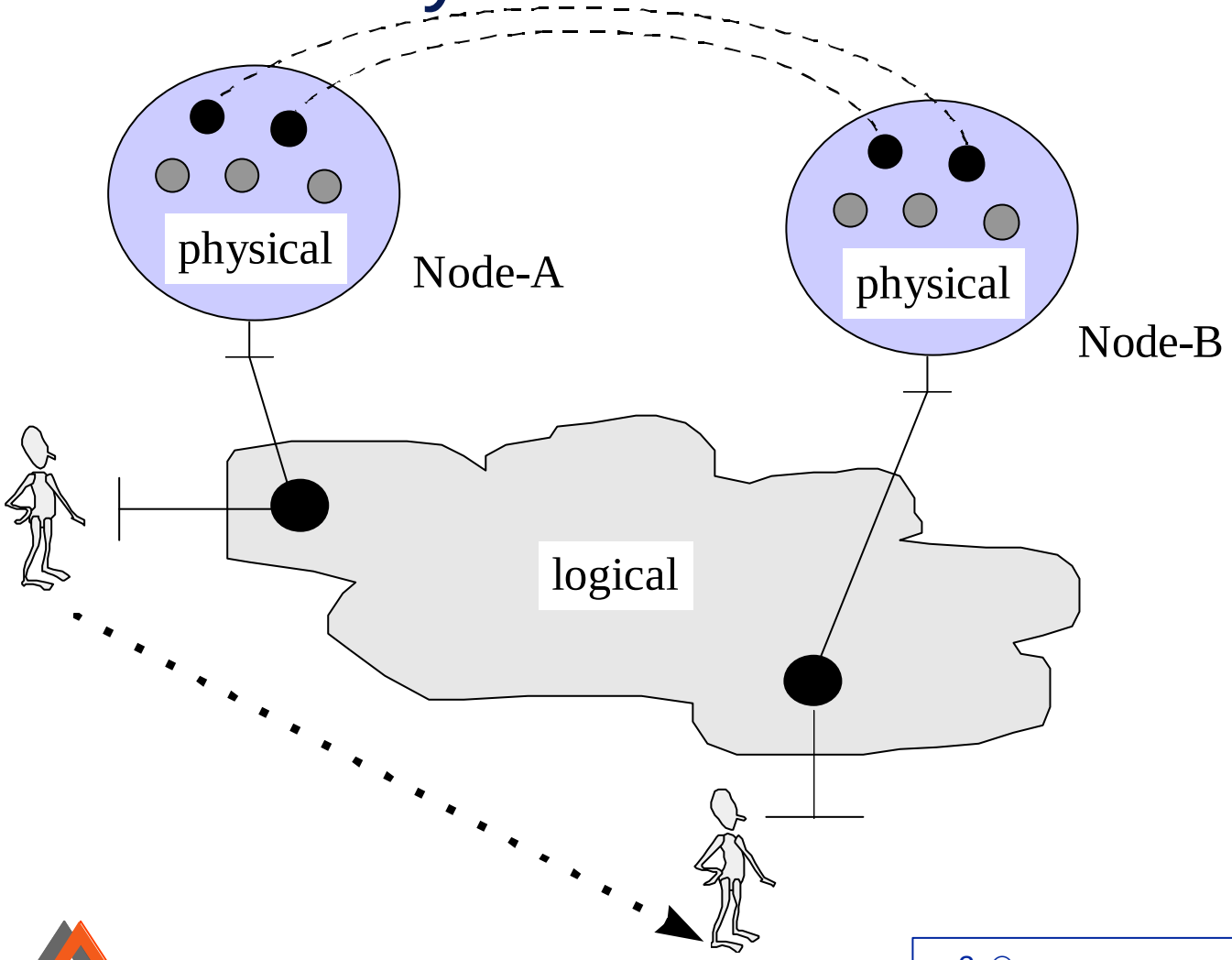
Douglas Donaldson, APM Ltd.

(presented by Andrew Herbert).

Research Goals

- MOW gives migration transparency (‘Mobile Objects’)
- InformationSpace will give persistence transparency (‘Robust Objects’)
  - Persistence of the server will be transparent to both the client *and* the server (with caveats).
  - It aims to allow migration and replication of robust objects…
    - … with the choices of server behaviour configured in middleware using FlexiNet binding.

Logical and Physical Views of Information



Immediate Requirements for FollowMe

- Agents cannot be relied on to deliver results
  - Their host or network may crash
- They need a service allowing information to be stored reliably
- For example, the Newspaper Application would benefit from additional transparencies
  - Both news gathering agents (‘personal editors’) and news delivery agents (‘paper boys/girls’) would gain performance benefits from transparent replication of news information objects.

Simple Black Box Model

- InformationSpace as an abstraction of a file system

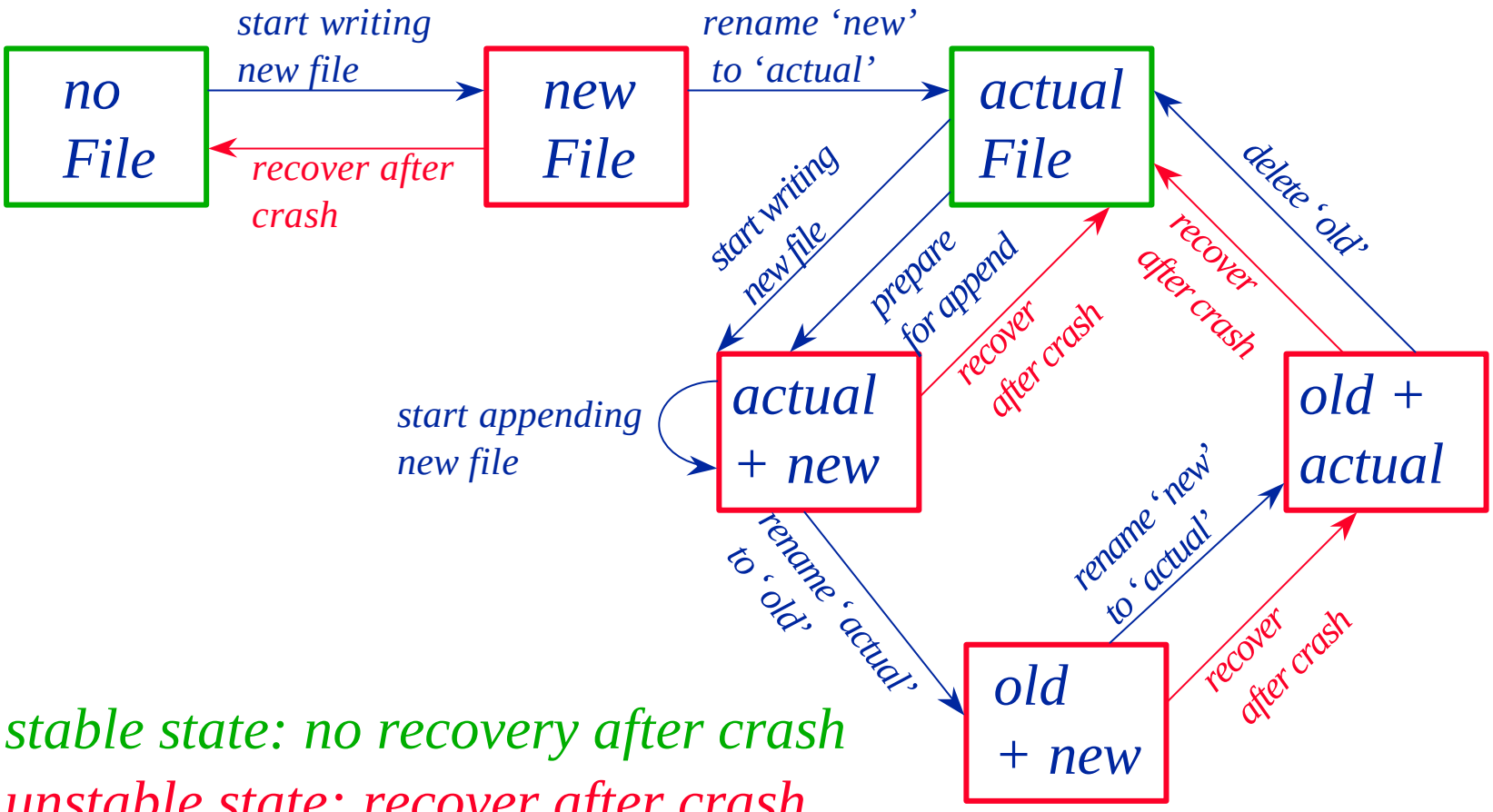
```
void copyInto(String name,
              Object obj)
Object copyOut(String name);
void remove(String name);
String[] listNames();
```

```
write(filename, buffer);

buffer = read(filename);
delete(filename);
list();
```

  - InformationSpace prevents conflicting copyIntos
  - Objects copied by value (FlexiNet model)
  - Extensible with locking, permission modes,…
  - Client manages copies, names, versioning

State Model for Atomic Write



stable state: no recovery after crash

unstable state: recover after crash

Less Simple White Box Model

- InformationSpace as an Object Database

```
Interface newStorable(String name, Class class, Object[]args);
Interface lookup(String name);
Object copyOut(String name);
void remove(String name);
```

  - The Storable Objects are *encapsulated* behind the IS
  - The Storable is just a (remote) object to clients
  - The Storable is (nearly) an arbitrary data type
    - so the White Box Model subsumes the Black Box Model
  - The InformationSpace prevents conflicts *and more…*

Storable Example

- ```
  public interface List {
     public void insertAtHead(Object object);
     public Object removeFromTail();
  };
  ```

- ```
  listRef = infospace.newStorable(ListImpl.class,
                                  "liszt");
  ```

- ```
  listRef.insertAtHead("item");
  ```

- The InformationSpace keeps the List persistent
  - Transparent to the client using listRef, and to List itself

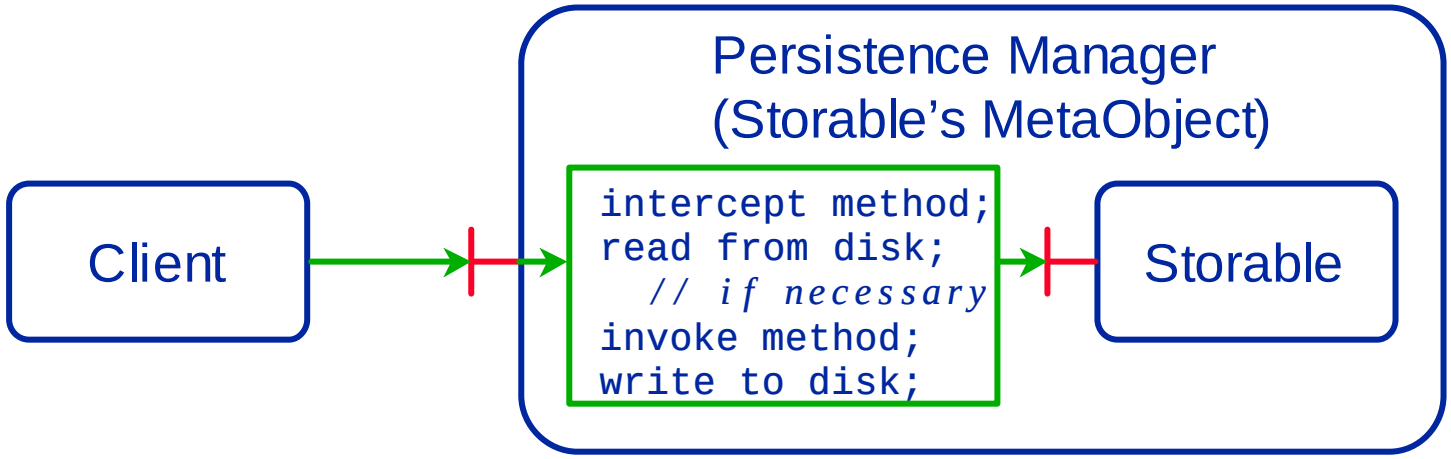

  ```

# The Encapsulation Allows Hooks

- Server Side:
  - Persistence
  - Concurrency Control
  - Transactions
- Client Side:
  - Failure Recovery
  - Migration Transparency
  - Replication Transparency

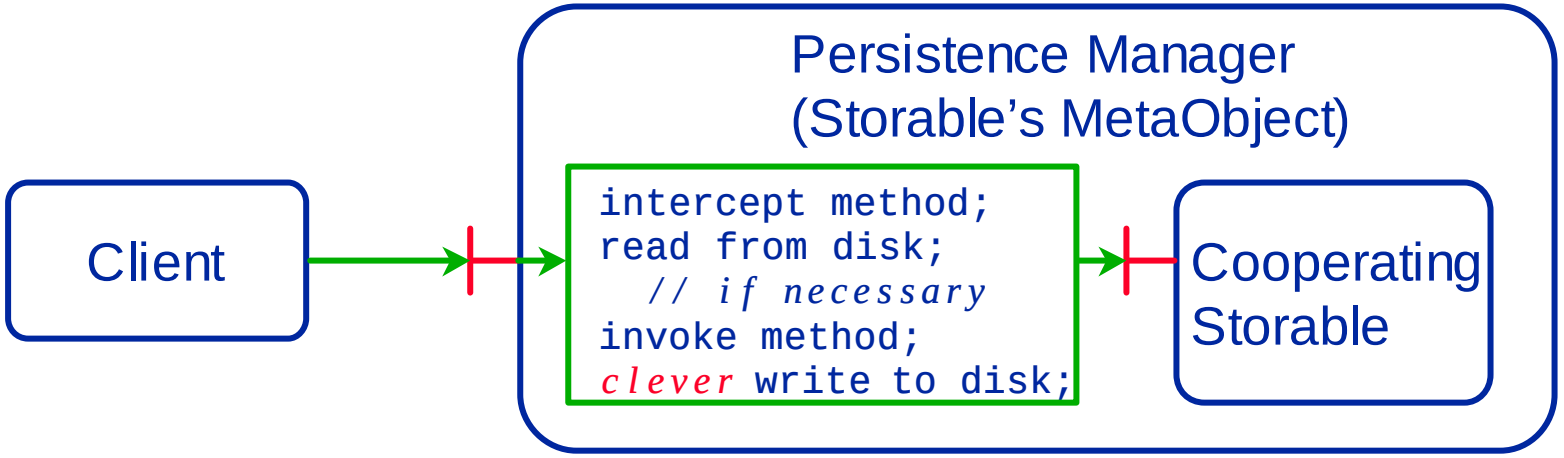
# Default Persistence




- Constraints on Storable:
  - Serializable Somehow
  - Well Behaved

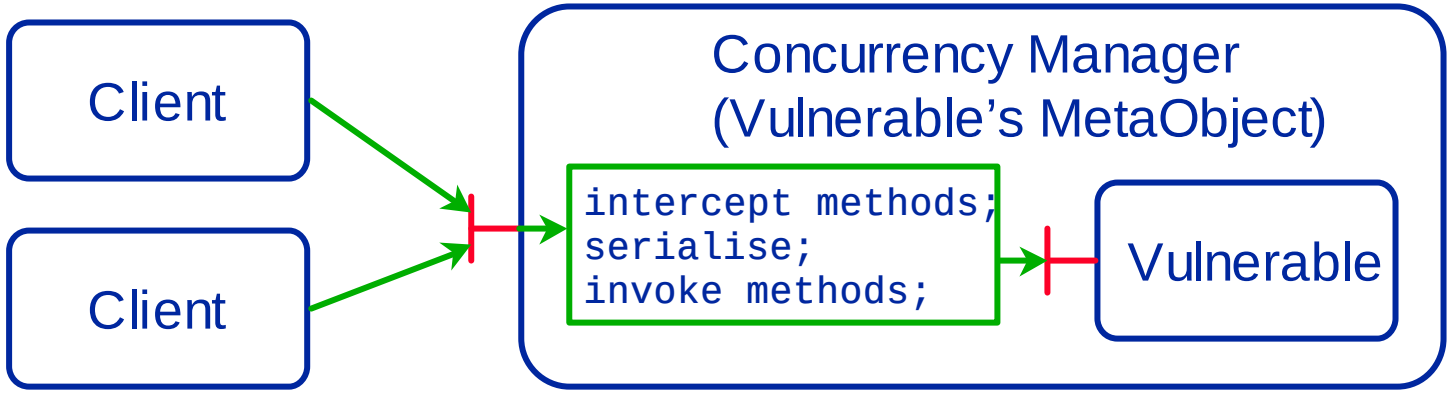
# Advanced Persistence




- Cleverness such as:
  - Incremental write of changes to Storable
  - Stable store
  - Versioning
  - Rollback

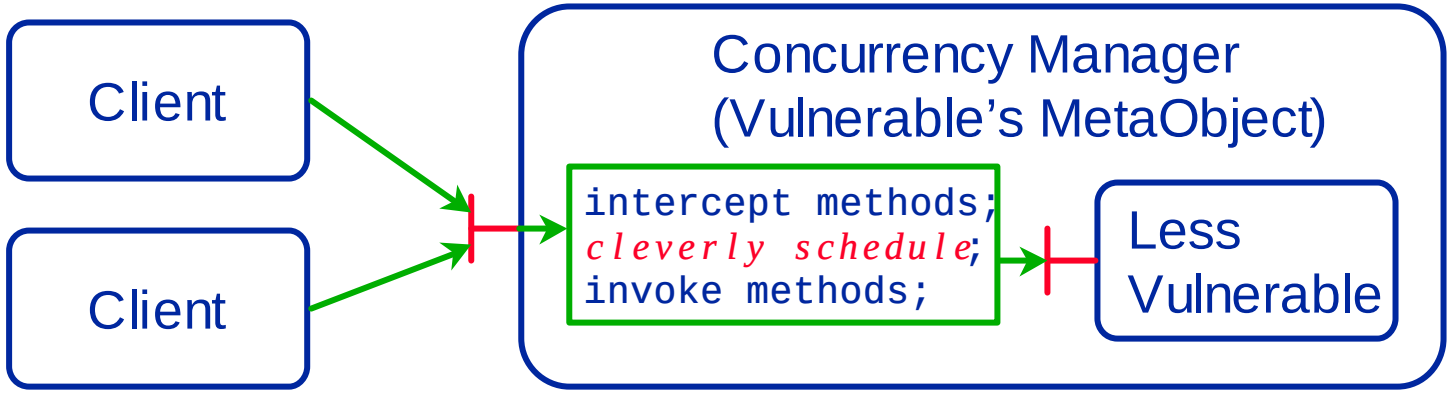
# Default Concurrency Control




- Given the default persistence model, and unknown concurrency semantics for vulnerable, all methods must be invoked sequentially.
  - Policy for mutual exclusion could be:
    - Fair (no starvation)
    - Unfair (possible starvation)

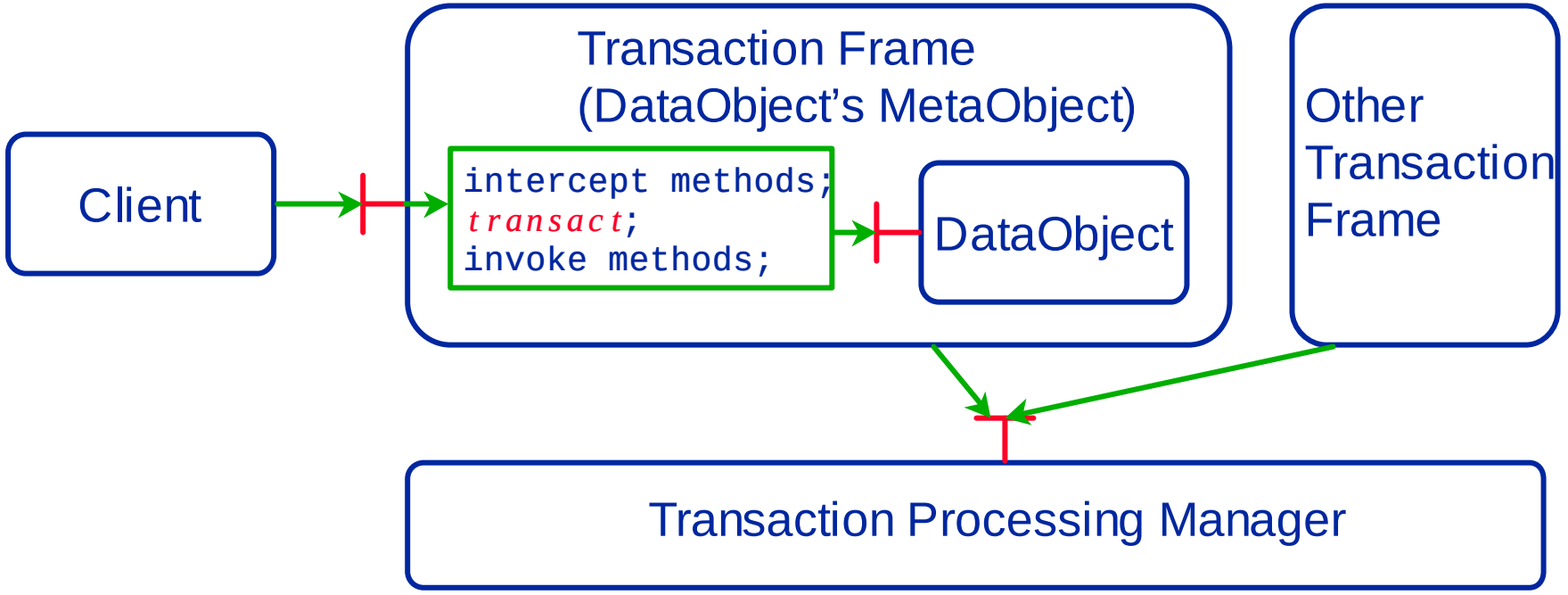
# Advanced Concurrency Control




- Use knowledge about read only methods
  - Allow concurrent reads, only save state back after changes.
- Cleverer scheduling according to:
  - Vulnerable's state
  - Client priority
  - Method parameters
  - Invocation history

# Transactions




- Client must have some transactional awareness:
  - begin, end, abort_transaction

# Default Failure Recovery

- When a host restarts, it restores its InformationSpaces, which contain persistant objects.
- The clients' FlexiNet references to the IS and Storables become invalid.
- The client has to rebind using a NameServer and / or:
  - `interfaceRef = infospace.lookup(objname);`

# Advanced Failure Recovery

- The client-side interface reference is to a ‘clever’ stub which:
  - Encapsulates enough information to automatically rebind on failure
  - Attempts to rebind the FlexiNet reference after failure
  - Less work for the application programmer
- Existing migration transparency mechanisms can be used
  - Restart after failure is like restart after movement

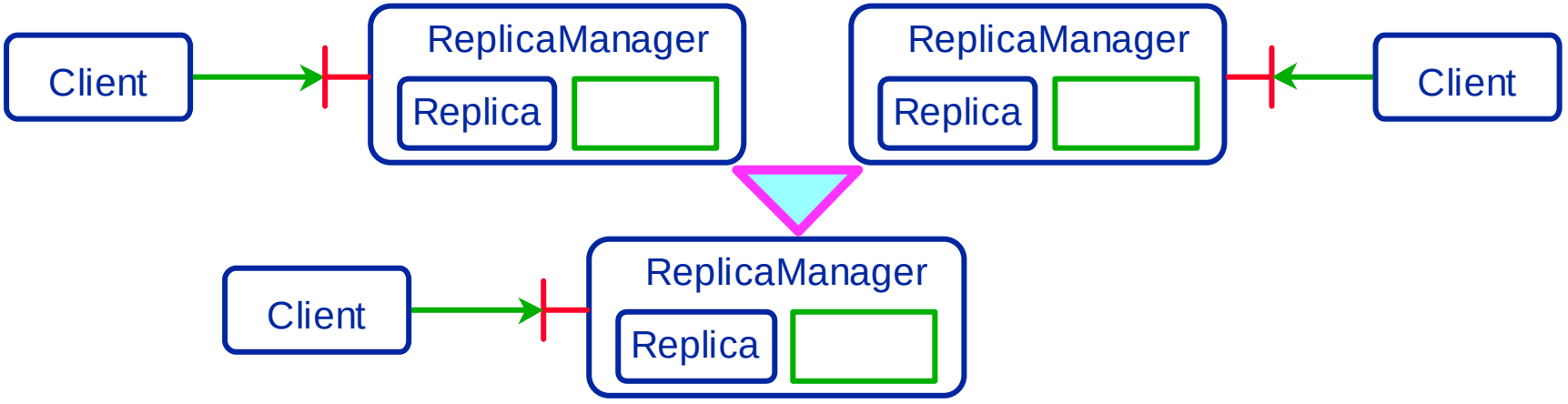
# Migration Transparency

- A Client (or for transparency, a clever client stub) may request that a Storable *FollowsMe*.
  - `void move(String name, InformationSpace ispace);`
- FollowMe migration transparency would need augmented with:
  - Reliable movement protocol
  - Reliable MobileName servers

# Replication Transparency

- Clients (or for transparency, client stubs) may request that a Storable *FollowsThem*.
  - `is.replicate(objname, iss);`




- Requires a consistency model for operations on InformationSpaces and Storables

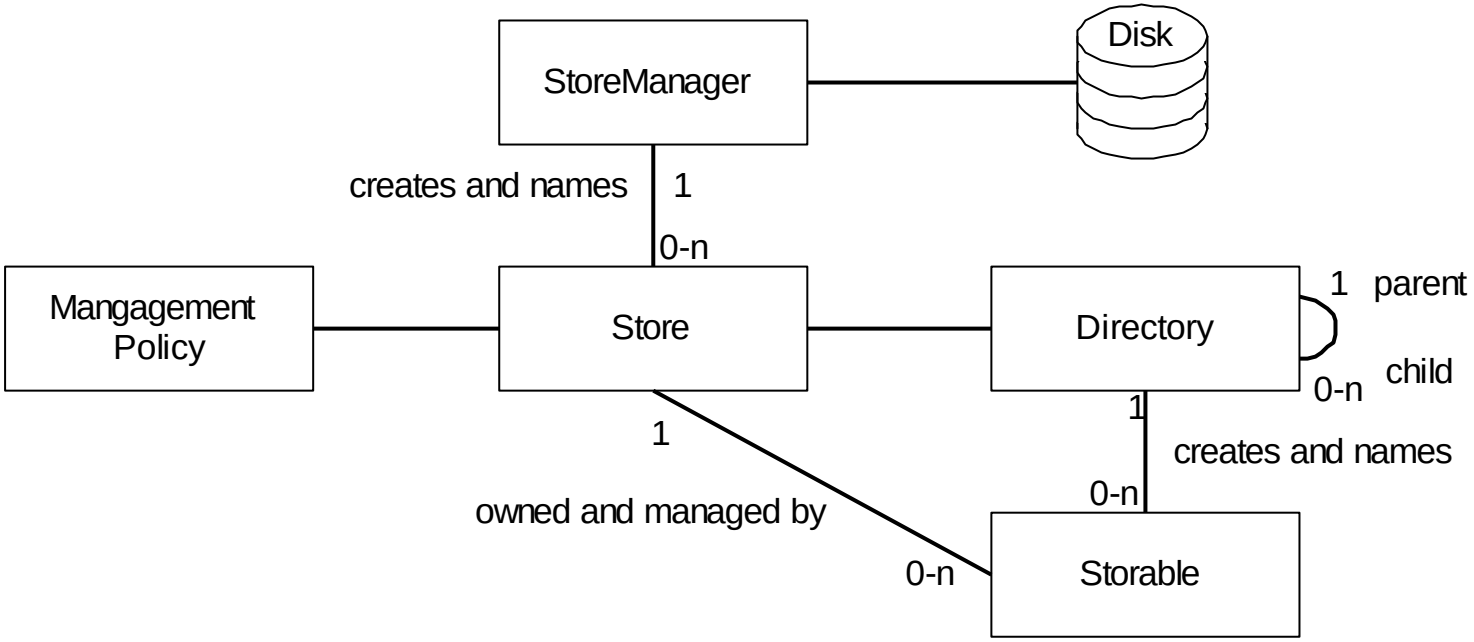
 

# MetaObjects are Owners and Managers

- The MetaObjects in the previous diagrams act as persistence, concurrecy control, transaction, migration and replication managers.
- Generalising, a managed object needs an *owner* with vested interested in the object's behaviour
  - The owner's policy can also control access, lifespan, resource usage, etc.
- Owners can be a 'group object' for 'dumb' data
- 'Smart' objects may own themselves

# Default Component Configuration




- The StoreManager can create Stores with different policies.
- The policy covers all the Storables in the Store

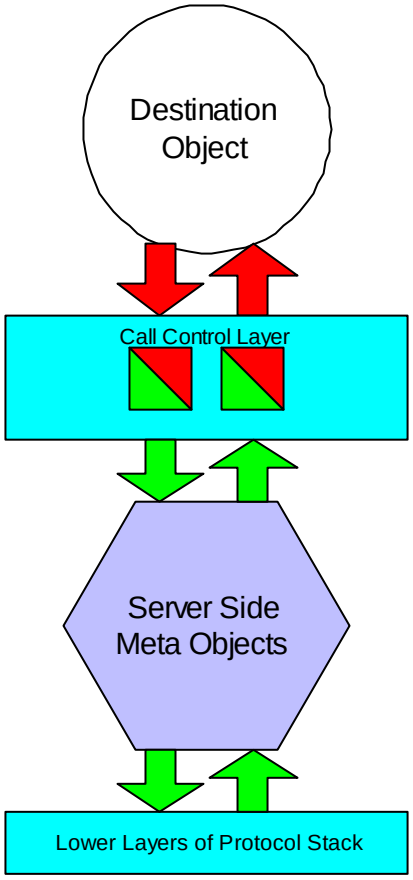
# MobileObjects vs Storables

- A Storable is like a Cluster, but is not a kind of Cluster. Both rely on
  - Encapsulation
  - Thread management
    - The conditions for a Storable to be stored are the same as those for a MobileObject to be moved
  - Clever interface references returned on creation
- There are differences too
  - Storables are not autonomous / active
- The symmetry is not enforced

# Integration with FlexiNet

- newStorable() can put the Store (Meta Object) into the CallControlLayer
- The InformationSpace is then integrated into FlexiNet as an example of Binding, rather than horizontally separated
- Similarly, suitably clever stubs can be built into the client side



# Progress

- Black Box model finished, end March 1998
- White Box model (transparent store of arbitrary data types) due April 1998
- Future work probably on replication of read only information