Manifestos

- OODBMS Manifesto
  - M. Atkinson, U. Glasgow
  - D. Dewitt, U. Wisconsin
  - D. Maier, Oregon Graduate Center
  - F. Bancilhon, Altair
  - K. Dittrich, U. Zurich
  - S. Zdonik, Brown U.

OODBMS View

Manifestos

- 3GDBMS Manifesto
  - M. Stonebraker, UC Berkeley
  - L.Rowe, UC Berkeley
  - B. Lindsay, IBM Research
  - P. Bernstein, DEC
  - J. Gray, Tandem Comp
  - M. Carey, U. Wisconsin
  - M. Brodie, GTE
  - D. Beech, Oracle

ORDBMS View

Manifestos

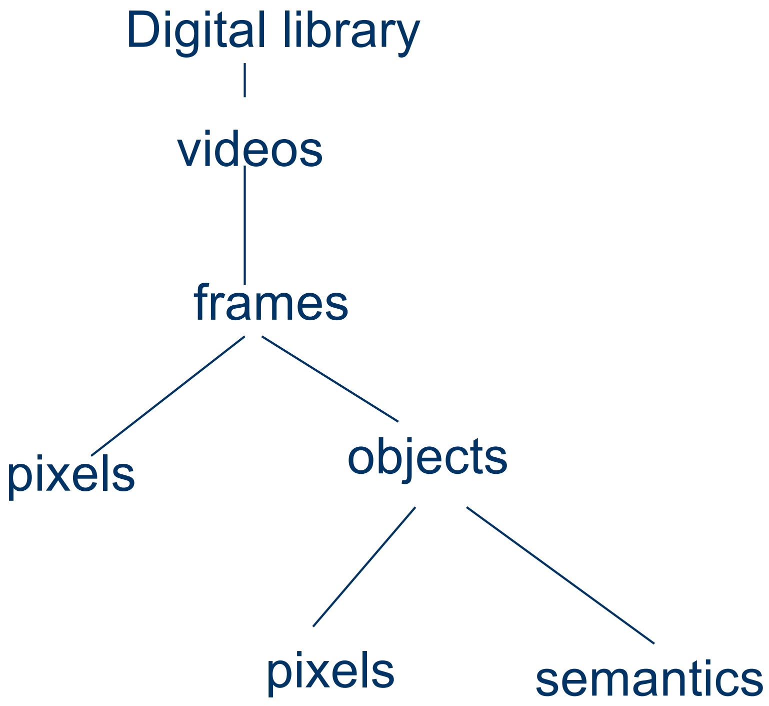
- Comments on “3GDBMS Manifesto”
  - D. Maier, Oregon Graduate Institute

OODBMS View

OO Concepts - Complex/composite objects

- each object has a set of attributes
  - simple objects do not have attributes; e.g. integer
- each attribute can contain
  - an object or
  - a set/sequence of objects

Complex objects/aggregation hierarchy



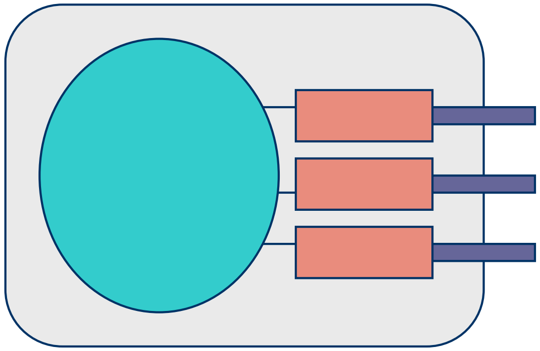
Collections

- Collections:
  - List <T>
  - Bag <T>
  - Set <T>
  - Array <T> (sequence <T>)
- T can be any class name

OODMBS

- Encapsulation
  - Each object has a state (the value of the attributes)
  - Each object also has a set of (methods/interfaces) pairs to modify or manipulate the state.

    - State (attribute values)
    - Methods (procedures)
    - Interfaces



OO Concepts - Identity

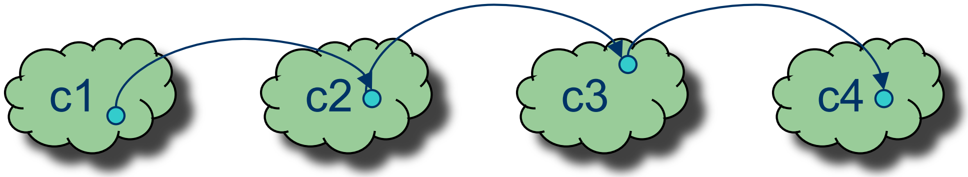
- Objects and Identity
  - each real world entity is modeled as an object
  - each object has a unique identifier (UID)

Identity and Equality

- Obj1 = Obj2
  - Objects have the same object id
- Obj1 == Obj2
  - Objects have the same values for the corresponding attributes
    - deep or shallow

Path Expressions

- $\underbrace{\underbrace{\underbrace{\underbrace{O}_{c1}.att1}_{c2}.att2}_{c3}.att3}_{c4}$



- Instead of foreign keys, references are implemented through explicit pointers/objectids
- This may be better than joins(??)!!!

OO Concepts – Types and Classes

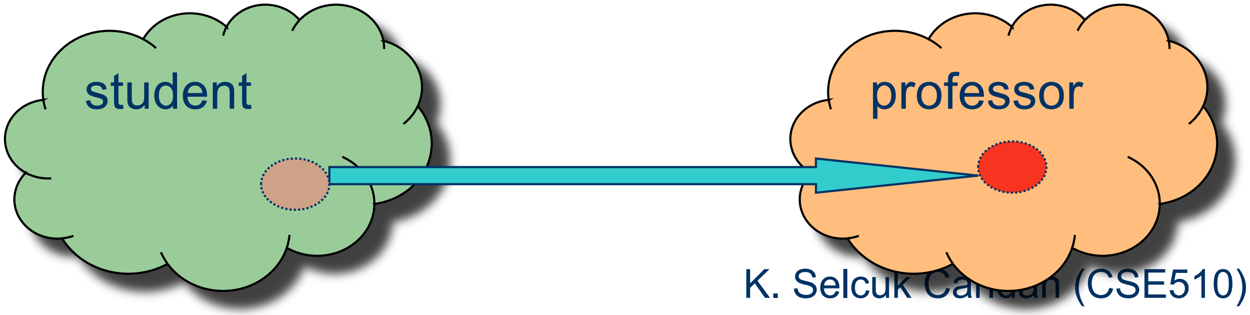
- Sometimes these terms are used interchangeably…
  - type is a compile time concept
  - class is a run time concept
    - classes are usually associated with their *extensions*. I.e., the set of objects that are of the corresponding class.

Classes

- Objects which share the same set of attributes and methods are grouped together in classes
  - the implementations may still differ
- Each object belongs to some class.

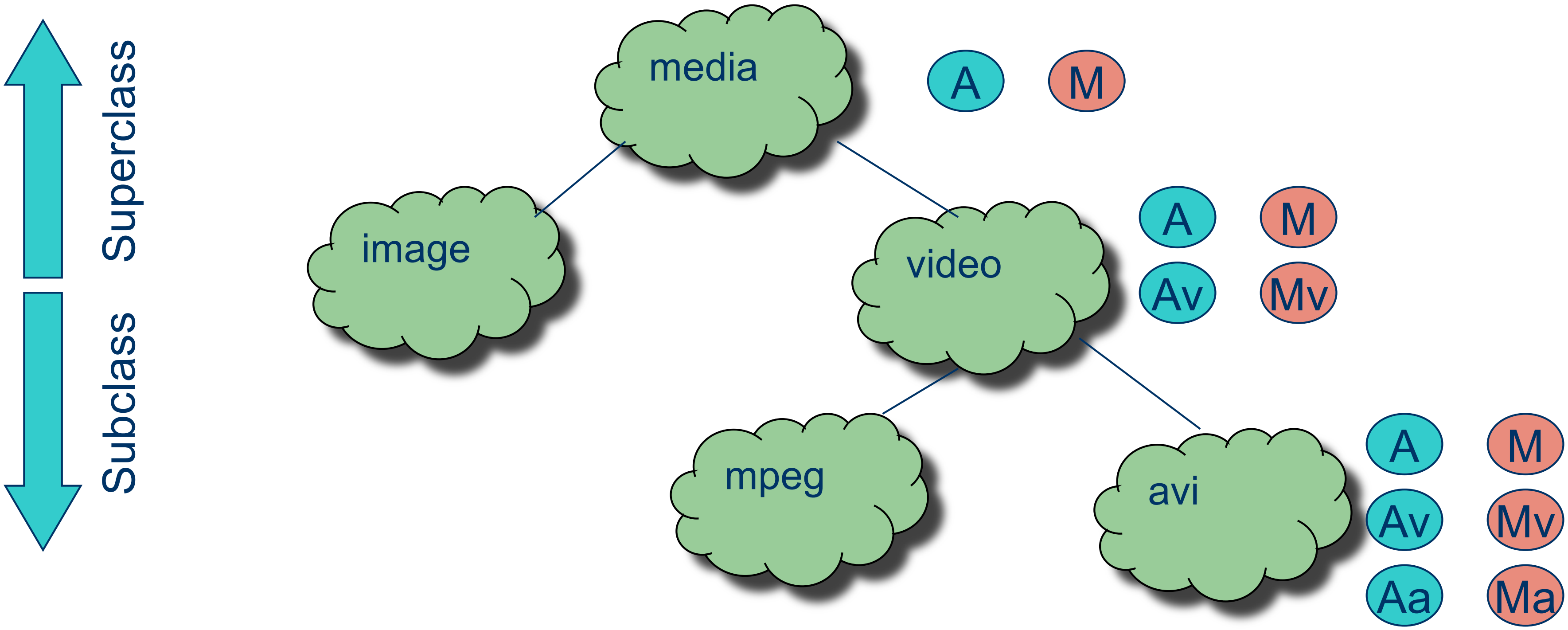
Classes

- Objects which share the same set of attributes and methods are grouped together in classes
  - the implementations may still differ
- Each object belongs to some class.
- Objects can migrate from one class to another.



Class/inheritance hierarchy

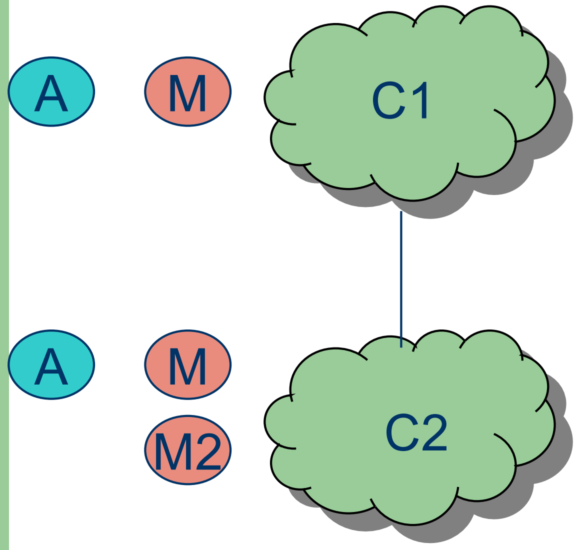
- …object membership

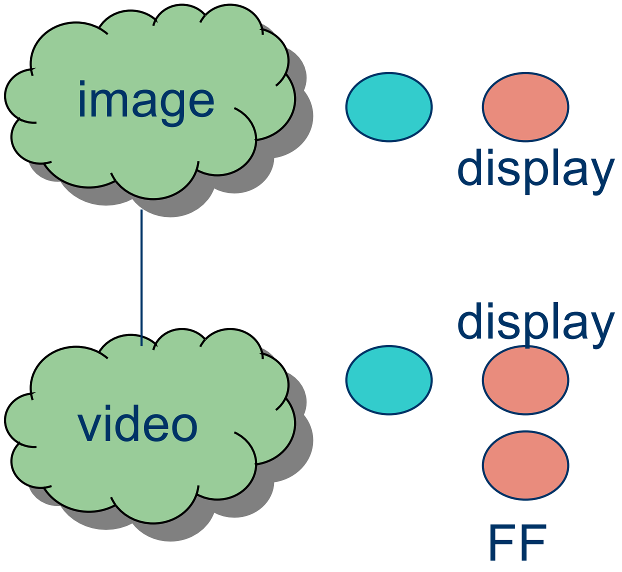


Class or inheritance hierarchy

Substitution inheritance

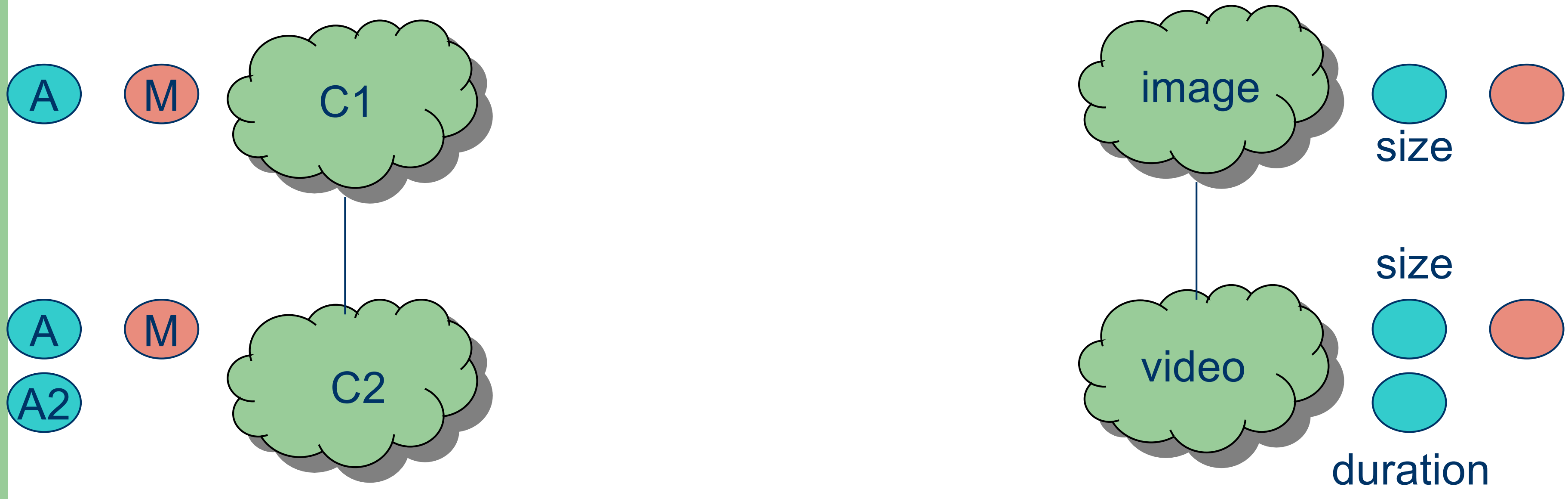
- …more operations





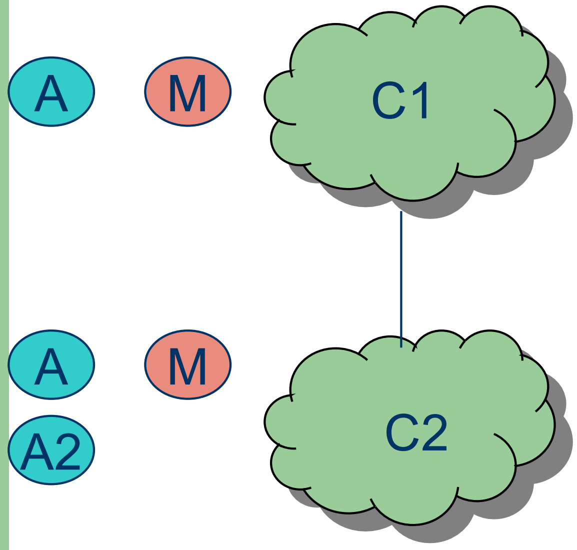
Inclusion inheritance

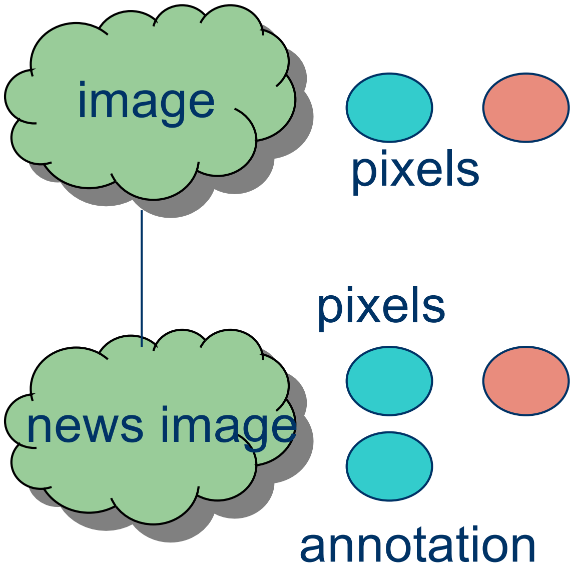
- …more attributes



Specialization inheritance

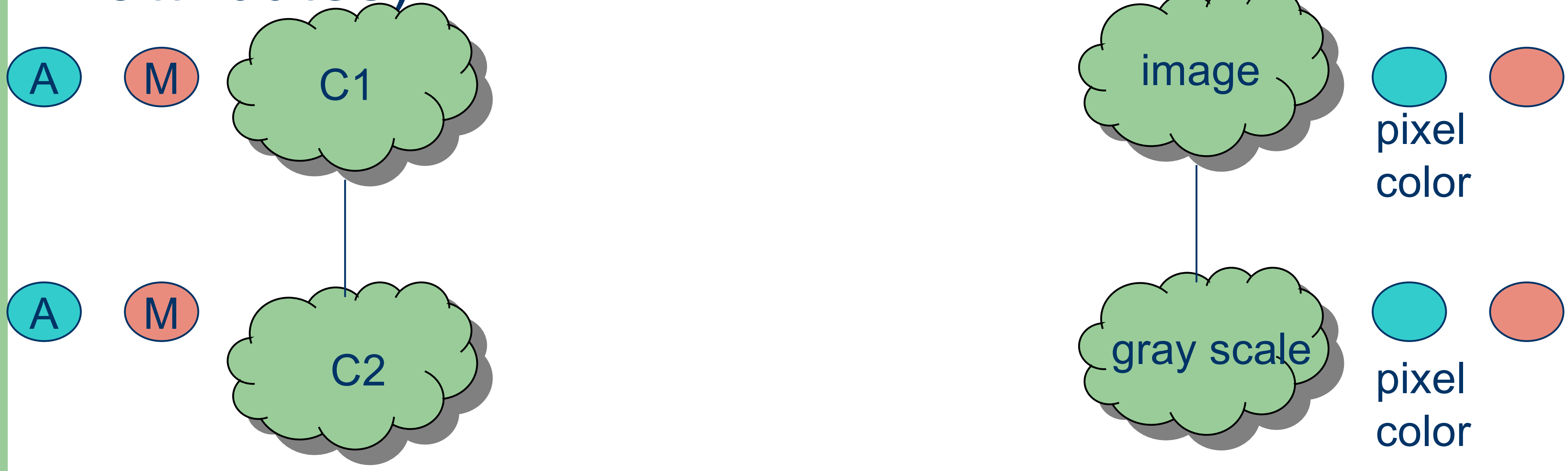
- …more attributes



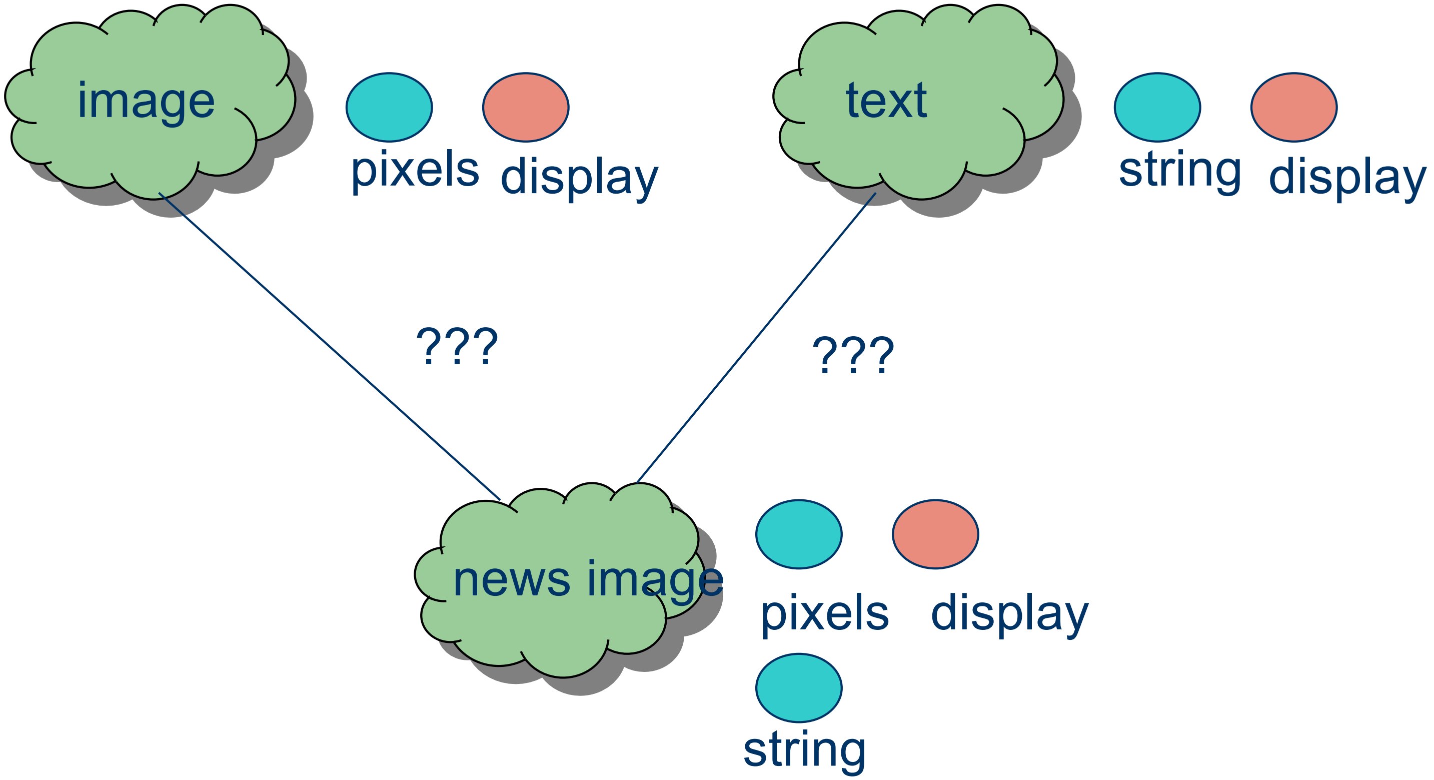


Constraint inheritance

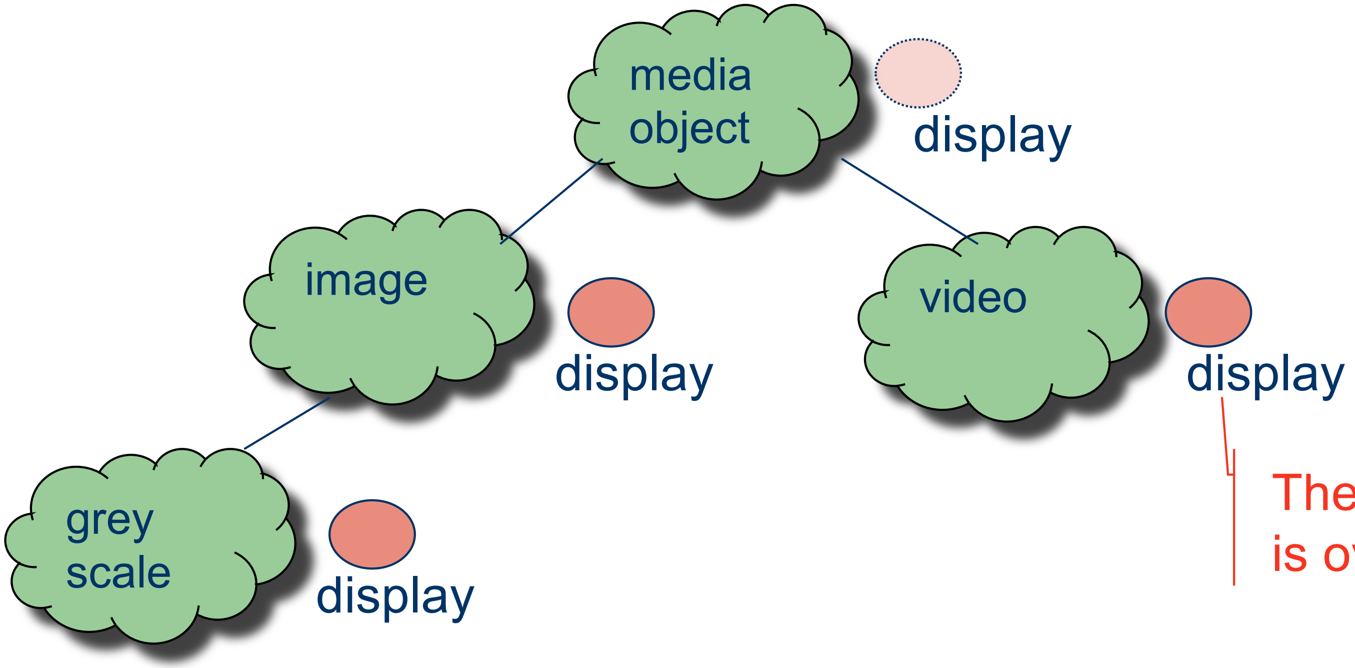
- …more strict constraints (domain of the attributes)



Multiple inheritance

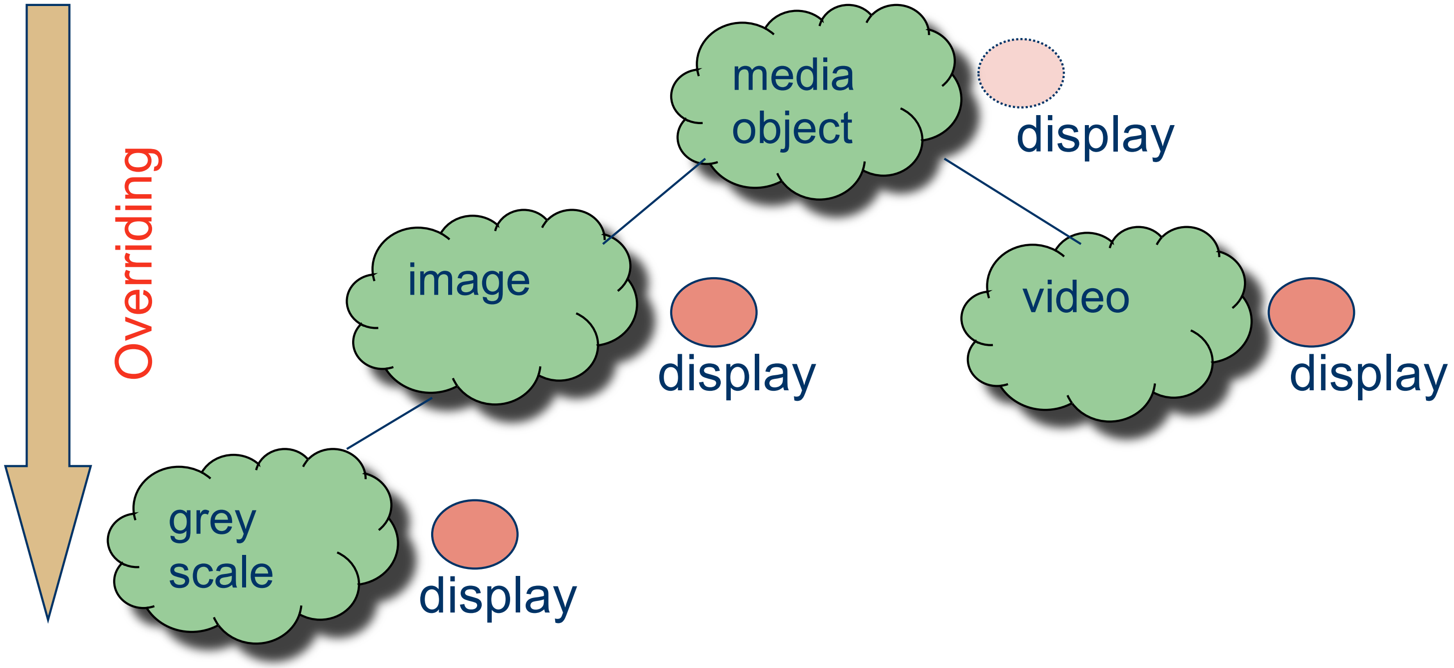


Overriding, overloading, late binding

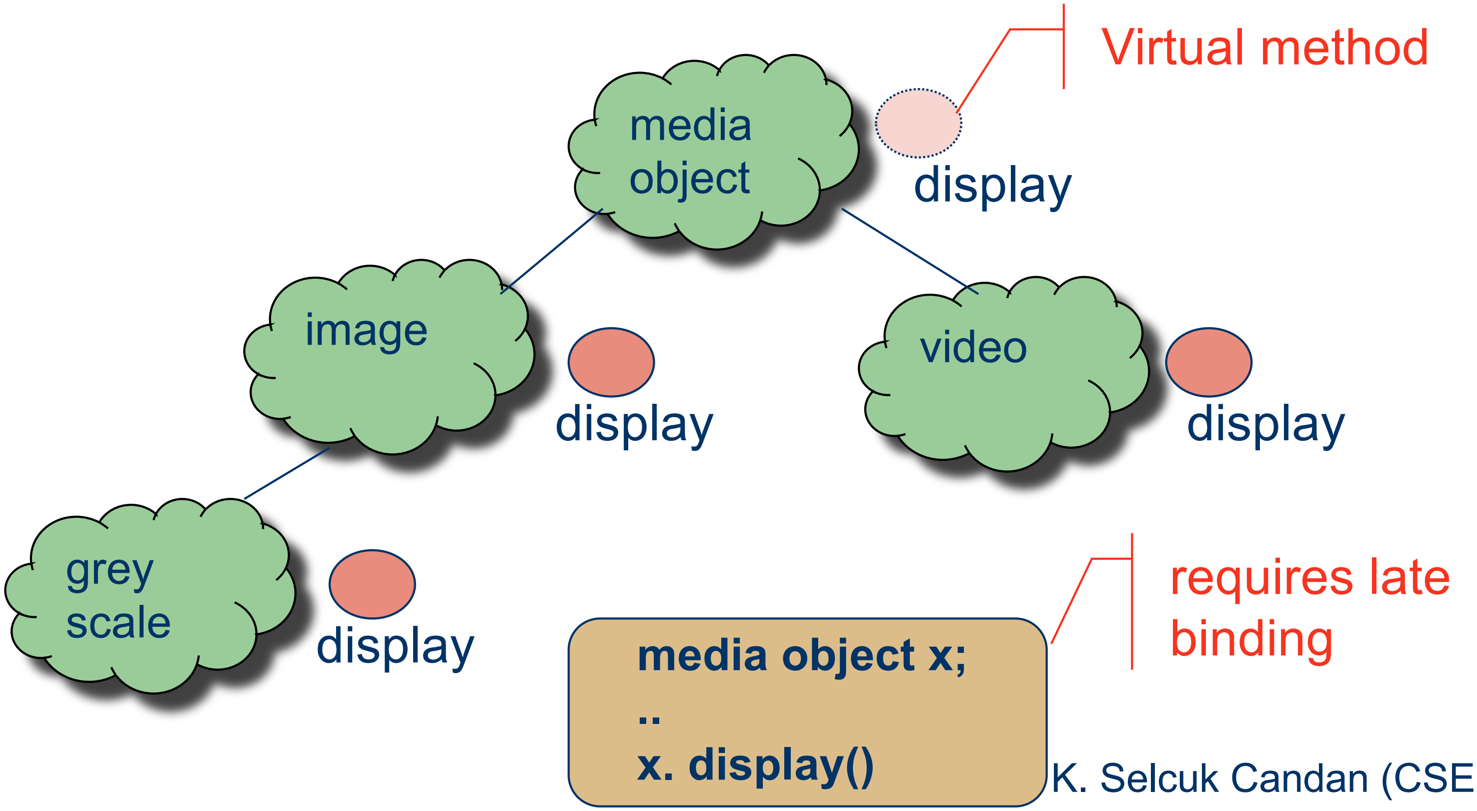


The method name is overloaded

Overriding, overloading, late binding



Overriding, overloading, late binding



```
media object x;
..
x. display()
```

The OODBS Manifesto

Mandatory

- Complex objects
- Object identity
- Encapsulation
  - Types and classes
- Class or type hierarchies
- Overriding, overloading, and late binding
- Computational completeness
- Extensibility
- Persistence
- Secondary storage management
- Concurrency
- Recovery
- Ad hoc querying

OODBs

- Integration of structure and behavior
- Classes, types, and inheritence
- Object identity
- Encapsulation and views

- Relational model is a striped down version of E-R model
- OODB design without dynamics is nothing but ER.
- OO allows more semantics to be captured.

The OODBS Manifesto

- Optional
  - Multiple inheritance
  - Type checking and type inferencing
  - Distribution
  - Design transactions
    - long & nested transactions
  - Versions

How do we store objects in an OODBMS??

- OO data is complex and variable in size and structure.

Clustered storage…

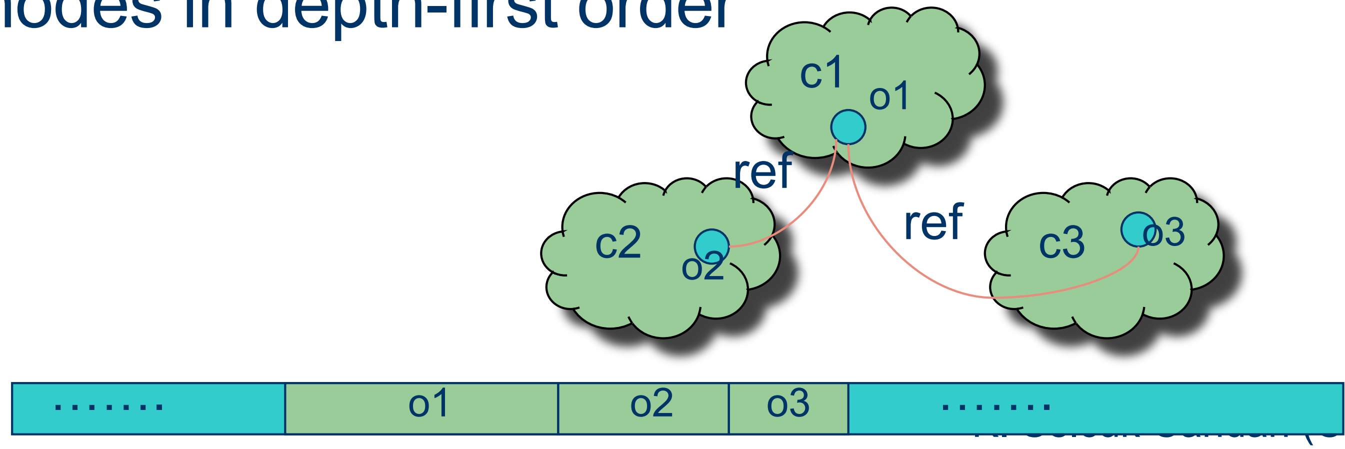
- Storing relevant pieces of data closer to each other on the disk

Clustered storage…

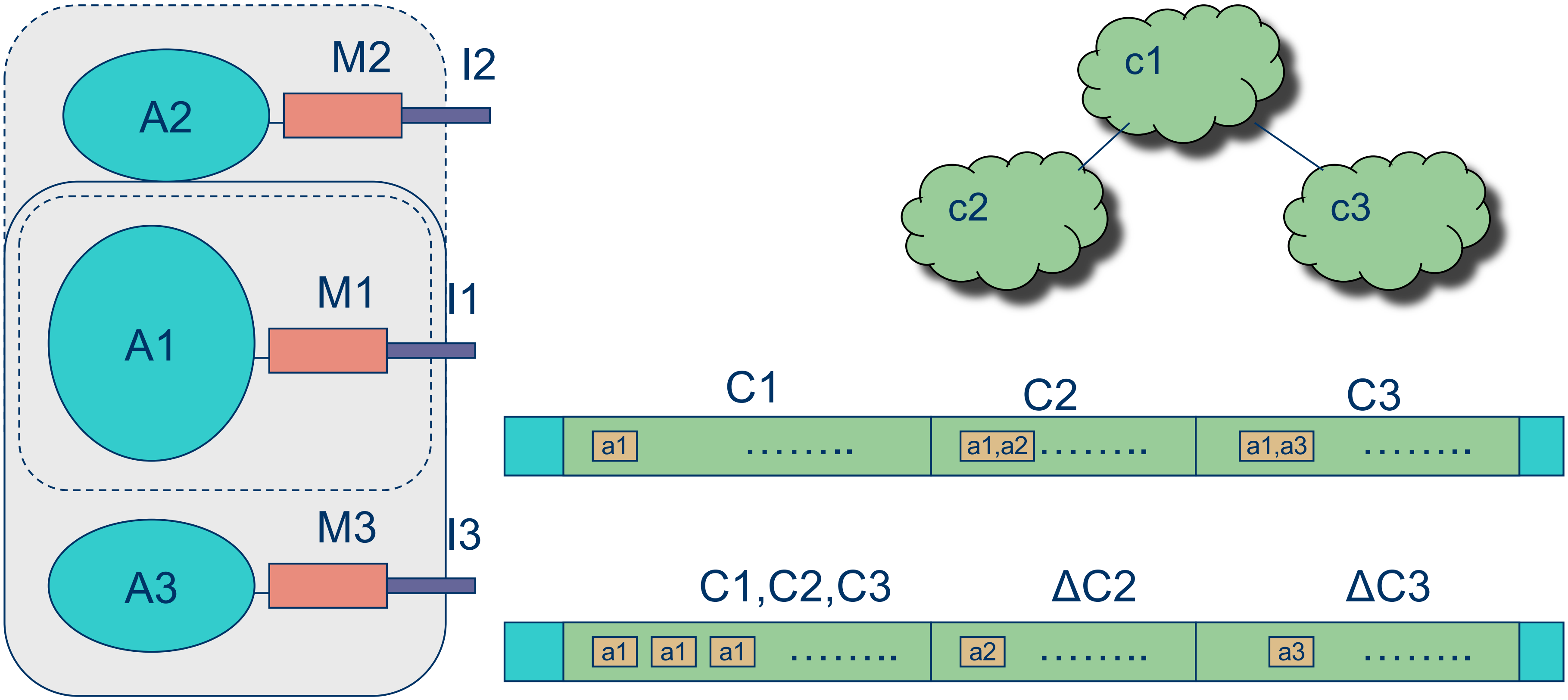
- Storing relevant pieces of data closer to each other on the disk
  - Method1: store all object instances of classes contiguously as in RDBMSs

Clustered storage…

- Storing relevant pieces of data closer to each other on the disk
  - Method1: store all object instances of classes contiguously as in RDBMSs
  - Method2: given an aggregation hierarchy, store nodes in depth-first order



Inheritence-based storage



OODB Design Steps

- Problems
  - redundancy (no normalization)
  - no clear distinction between the design of application and database semantics
    - lost logical independence
  - integrity constraints
    - not declarative
    - implemented within methods
    - model static connection between classes

- 3GDBMS Manifesto (ORDBMS View)

Object Relational DBMS

- Collections (of objects)
- New types (… and inheritance)
  - row, table
  - set, multiset
  - reference
- Transformations
  - collection to table

    TABLE(e.projects)
  - single row with a single column to object

    THE (SELECT e.projects from employees where SID = “-”)
  - table to set

    SET(SELECT * FROM EMPLOYEES)

Example queries

- Select ITEM e.salary
  From employees e
  - returns a multiset
- Select DISTINCT ITEM e.salary
  From employees e
  - returns a set
- Select e.salary
  From employees e
  - returns a table

Type Storage

- Primitive Types
  - No Specific requirements
- Row Types
  - In tables/columns
- Abstract/Opaque types
  - Store in columns
- REF type
  - Store RowID, OID, OID/TableID
- Collections (arrays)
  - Inline
  - Out of line in a LOB

Type Storage

- Collections (multisets/nested tables)
  - Inline
  - Out of line (individual refs inline)
  - Out of line (single ref inline)

- 3GDBMS Manifesto (ORDBMS View)
- Comments on “3GDBMS” Manifesto (OODBMS View)

Tenet 1:

- 3GDBMSs will provide support for rich object-structures and rules.
  - Richer object structures: non-traditional data elements.
  - Rules: data elements, records, and collections.
    - integrity constraints
    - if_then_do

Tenet 1:

- Richer object structures ??
  - type extensibility
  - support must be for *manifest* types.
    - first class (no difference in treatment of base types and others)
    - immediate (availability to any programmer)
      - schema definition time
      - only with DDL & DML
    - Abstract (hidden implementation)
- Rules??
  - inferencing?
      - rules are not the only way to do inferencing.
  - integrity constraints?
      - if_then_do rules are not the best way to achieve integrity.
  - event sequencing?
      - dynamicity

Prop 1.1:

- 3GDBMS must have a rich type system:
  - array, sequence, record, set, functions, union, ADT
  - query language support is essential
  - relational databases can handle this.

Prop 1.1:

- 3GDBMS must have a rich type system:
  - array, sequence, record, set, functions, union, ADT
  - query language support is essential
  - relational databases can handle this.

- 3GDBMSs must support “manifest types”
  - object reference type?
  - types must be “user produced” not vendor supplied.
  - RDBMSs can not (efficiently) handle all extensions

Prop 1.2

- Inheritance is a good idea
  - Multiple inheritance is essential.
  - Collections without additional fields.

Prop 1.2

- Inheritance is a good idea
  - Multiple inheritance is essential.
  - Collections without additional fields.
- Inherited? What?
  - type hierarchy?
  - implementation hierarchy?
  - subset hierarchy?
- Are all instances of a subtype instances of a super type?
- Is multiple inheritance necessary in subset hierarchy?

Prop 1.3:

- Functions including database procedures and methods and encapsulation are a good idea.
  - Encapsulation encourages modularity.
    - Moving functions inside the DBMS improves performance **(*)**
  - Inheritance and overriding of functions.
  - All functions should be written in HLL.
    - Access to DBMS should be through nonprocedural access language (except in exceptional cases) **(**)**
  - Types shall be transparent
  - RDBMSs can do this

Prop 1.3:

- Impedance mismatch !!!
- Navigation within DML is not a problem (can be optimized)
  - It is only a representation
- Transparency is not essential
  - Transparency from the user and from the system are different things!
- If we move functions inside DBMS **(*)** function call would be the unit of communication, contradicting **(**)**

Proposition 1.4

- UIDs should be assigned only if a user-defined primary key is not available.
  - Primary keys never change
  - Human readable.

Proposition 1.4

- UIDs should be assigned only if a user-defined primary key is not available.
  - Primary keys never change
  - Human readable.

- Keys are good within a collection. If an instance belongs to more than one collection than key may not uniquely identify the instance.
- Never is a long time
- Immutability is not enough
  - Existence & 1-to-1 may not always hold.

Proposition 1.5

- Rules(triggers, constraints) will become a major feature in future systems. They should not be associated with a specific function or collection.
  - OODBs ignores rules, RDBs support them.
  - Two disadvantages of putting rules in functions:
    - Too much responsibility for programmers
    - If two rules interact a single function may not capture the whole semantics.
    - Queries about rules.

Prop. 1.5

- Rules, triggers, and constraints are not the same!
- Looking at a state (using horn clauses) cannot capture all semantics.
- Chaining if_then_do rules is not wise afterall.
- RDBMSs support them?

Tenet 2

- 3GDBMSs must subsume 2GDBMSs
  - non-procedural access
  - data independence
    - optimization,
    - views

Tenet 2:

- “include features of” ……….yes,
- “be directly compatible” …...no.
  - 2GDBMS is not a superset of 1GDBMS
- QLs must extract proper information & display it in intelligible manner ??
  - SQL is limited to “structurally homogenous records”
  - What if the regularity is in the operations, not in the structure?
- OODBMSs can have views
- OODBMSs provide a higher degree of data independence (masked by message interface)

Proposition 2.1

- All programmatic access to a database should be through a non-procedural language.
  - OODBMSs allows navigation…Navigation is bad!
    - hard to optimize,
    - hard to evolve! (schema)
    - Navigation does not provide performance gains after all

Proposition 2.1

- Navigation is only a representation..
- Methods and application programs are not the same.
  - Methods->DBMSs can optimize
  - Application -> high level message expressions.
  - SQL is limited
  - Pointers aren’t the only way to implement object references.
    - Hence could be optimized.

Proposition 2.2

- At least two ways to specify collections:
  - enumeration of members and a QL to specify membership.
  - OODB suggests enumeration is the way to go.
  - Inefficient (large and overlapping)
  - Intentional specifications can be optimized further within a query….. Q(IS) ->Q and IS

Proposition 2.2

- At least two ways to specify collections:
  - enumeration of members and a QL to specify membership.
  - OODB suggests enumeration is the way to go.
  - Inefficient (large and overlapping)
  - Intentional specifications can be optimized further within a query….. Q(IS) ->Q and IS
- Both are necessary….application dependent.
    - Specification -> independent, derived.
    - Implementation -> expression, explicit list of elements.
  - Extensional representation can be fast.
  - For derived collections, SQL is not the way
    - Possible heterogeneity

P 2.3

- Updateable views are essential!
  - Functions are hard to update,
  - Updating views is a hard problem!
  - Relational DBMSs are doing reasonable with this.

P 2.3

- Updateable views are essential!
  - Functions are hard to update,
  - Updating views is a hard problem!
  - Relational DBMSs are doing reasonable with this.
- OODBs also provide views!
  - In addition, OODBs provide views with virtual objects (through UIDs)

P2.4

- Performance indicators must have almost nothing to do with data models and must not appear in them:
  - compilation techniques
  - location of buffer pool
  - kind of indexing used
  - clustering to be performed.

P 2.4

- True…
- ….but complex object structures and logical groupings of objects can be (and are) part of the semantics of data.
- … they can be used in optimization:
  - archiving,
  - concurrency control,
  - copying,
  - versioning, etc.

Tenet 3: 3GDBMSs must be open to other DBMSs

P 3.1: 3GDBMSs must be accessible from multiple HLLs.

- Multilingual databases
  - impedance mismatch between type system of HLL and database

P 3.1: 3GDBMSs must be accessible from multiple HLLs.

- Multilingual databases
  - impedance mismatch between type system of HLL and database
- ...not necessarily
  - have one language for writing methods
  - have other (application) languages invoke these methods

  - 20 years from now, no one will write HLL code directly!

P 3.2 Persistent X for a variety of Xs is a good idea.

- Any variable in user’s program should optionally be persistent.
- Use a cache in user space to implement persistency.

P 3.2 Persistent X for a variety of Xs is a good idea.

- Any variable in user’s program should optionally be persistent.
- Use a cache in user space to implement persistency.

- …maybe...
- …but, using user space (application space) for implementing persistency is not a good idea.

P 3.3

- SQL is intergalactic dataspeak.

P 3.3

- SQL is intergalactic dataspeak.

- SQL is a changing standard.
- SQL does not support persistent variables for anything except relations.

P.3.4

- Queries & their resulting answers should be the lowest form of communication between a client and a server..

P.3.4

- Queries & their resulting answers should be the lowest form of communication between a client and a server..

- SQL is changing
- SQL may call functions
- A call to a single function could be a legal SQL query.

P.3.4 (cont..)

- Sometimes, it may be better to do bulk of the processing at the workstation instead of the server.

- If functions are legal, maybe query languages should be functional instead of being based on predicate logic

Tenet 4

- 3GDBMSs should be simple, formally defined and clear.