Tsimmis

- Mediator-based integration.
- Unlike IM, there is no notion of universal predicates.
  - ❖ Mediators are linked together and to sources.
  - ❖ Each mediator answers queries about its own concepts and uses its sources (wrapped “raw” sources or other mediators) to answer those queries.

- Automatic generation of wrappers and mediators is a feature.
  - ❖ Mediators are described in a special language (MSL) that in effect describes views that the mediator can support + the algorithm the mediator must execute (queries to its own sources, e.g.) to support queries about those views.
  - ❖ MSL is not Datalog, but can be "encoded" in Datalog.
  - ❖ All of Tsimmis is based on an "object-oriented" model called OEM (object-exchange model).

Object-Exchange Model

Objects have four components:

1. *Object ID* (optional).
2. *Label* tells what the object represents (roughly, its class).
3. *Type* of its value, either a set or a primitive type like string.
4. *Value*, either a primitive value or a set of object references.

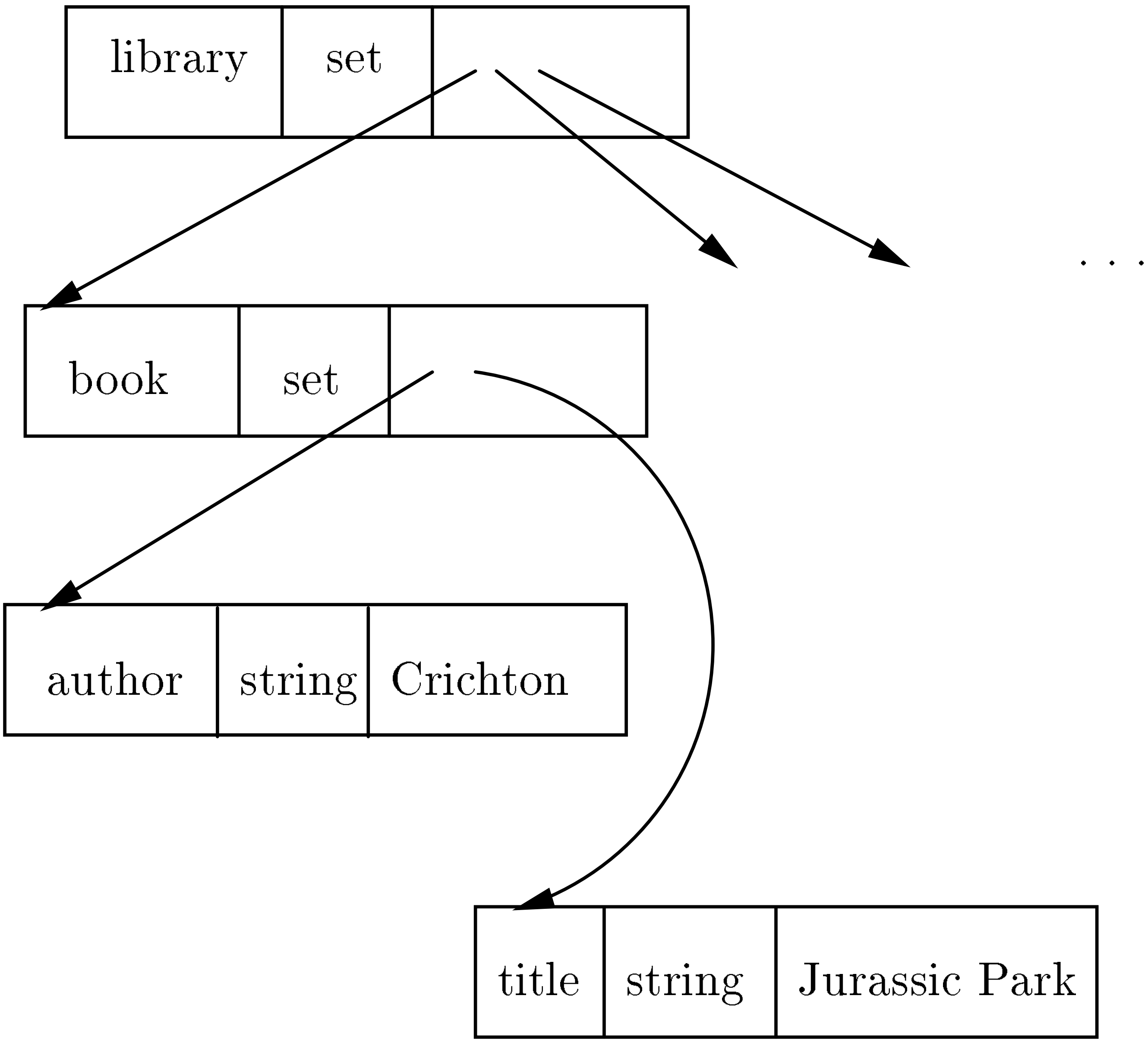
Object form:

OID:	label	type	value

OEM Objects are "Lightweight"

- Label = class, but there is no notion of field or instance variable.
  - ❖ We can fake fields by using labels.
  - ❖ Aside: "New OEM" uses fields but *not* classes. It can simulate classes via fields.
- OEM is tuned to "semistructured" data: unlike strongly typed object models, we cannot know what the subobjects are, i.e., what the "type" of an object is.

An Example Object



More About OEM

- There is no "schema." Objects with a given label are not required to have a particular set of subobjects.
  - ❖ But queries will ignore objects whose form doesn't match the query.
- Labels are expected to have human-understandable definitions that may be retrieved easily by the user.
- Object ID's are optional and may be ephemeral and/or local to the client.
  - ❖ They may be constructed by the mediators to be expressions describing where the object came from.

MSL

MSL statements are logical rules, but the underlying model is OEM objects rather than relations (as in Datalog).

- Labels and values are connected using triangular brackets, <...>.
- It is also possible to include an object-ID inside triangular brackets as an optional first component.
- Object-ID's may be constructed using function symbols.
- Some (not necessarily all) members of a set of objects may be described by enclosing them in curly braces {...}.
    - Important for *semistructured* data, where we cannot be sure what the subobjects are.

Example

Let's redo the employee-phone-office-etc. example in MSL. Suppose:

- Source 1 produces employee-phone-manager information
- Source 2 produces employee-office-department information
- Source 3 produces employee-phone information for members of the Toy Department.

Each of these sources will be assumed to export appropriate OEM objects. For example, Source 1 exports objects with atomic subobjects labeled `name`, `phone`, and `mgr`.

We wish to describe, using MSL rules, a mediator named `med` that uses these three sources and exports two types of objects:

1. Employee-phone-office objects with label `epo`.
2. Employee-department-manager objects with label `edm`.

- Each object of these types will have subobjects with the appropriate labels.

The MSL Description

```
(1) <f(E) epo{<name E> <phone P>}>@med :-
        <emp{<name E> <phone P>}>@source1
(2) <f(E) epo{<name E> <phone P>}>@med :-
        <emp{<name E> <phone P>}>@source3
(3) <f(E) epo{<name E> <office O>}>@med :-
        <emp{<name E> <office O>}>@source2

(4) <edm{<name E> <dept D> <mgr M>}>@med :-
        <emp{<name E> <mgr M>}>@source1 AND
        <emp{<name E> <dept D>}>@source2
```

Explanations

- Rule (1) says that whenever there is an `emp` object at Source 1 with a `name` subobject having value E and a `phone` subobject with value P, we "create" at the mediator `med` an object whose ID is $f(E)$ and whose label is `epo`.
    - ❖ This object has a subobject with label `name` and value E and a second subobject with label `phone` and value P.
- Rules (2) and (3) are similar.
- Rules (1)–(3) can each create subobjects for the *same* object, i.e., the object with ID `f(E)`.
    - ❖ OK as long as employee names uniquely identify the employee.

- Variables like E or P might not get atomic values in some instantiations of the rules.
    - ❖ They could be structured objects.
    - ❖ Different employees may have different structures, e.g., one phone object could be a string and another a structure with area-code and 7-digit number as subobjects.
- Rule (4) is different. An `edm` object is produced only if we are successful in finding, for employee E, a department at Source 2 and a manager at Source 1. If either is missing, then there is no object for employee E at `med`.

A Simplified Example

We shall use Datalog in place of MSL, but retain MSL's approach to answering queries by utilizing sources.

- Again use views $v_1(E, P, M)$, $v_2(E, O, D)$, and $v_3(E, P)$ from our IM examples. We could describe the mediator `med` as producing *epo* and *edm* "objects" by:

```
epo(E,P,O) :- v1(E,P,M) & v2(E,O,D)
epo(E,P,O) :- v3(E,P) & v2(E,O,D)

edm(E,D,M) :- v1(E,P,M) & v2(E,O,D)
```

- Unlike the MSL rules, we only get *epo* facts for employees such that among the three views we find both a phone and office for that employee.

Querying Tsimmis Mediators.

When we query an MSL mediator, we are effectively querying the objects exported by the mediator.

- There is no notion of "global" predicates as there is in IM. Rather, we must refer to the labels (equivalent to predicates) that the mediator exports.

Example

Again let us ask "what are Sally's phone and office?" using the mediator `med` in its Datalog incarnation. The appropriate query is thus:

```
answer(P,O) :- epo(sally,P,O)
```

MSL-generated mediators answer their queries by expanding the rules by which the mediator is defined, in order to get the same query in terms of information at the sources.

- Thus, we replace the *epo* subgoal in the query by the bodies of the two rules that define *epo* at `med`, obtaining the same set of solutions as IM:

```
answer(P,O) :-
    v1(sally,P,M) & v2(sally,O,D)
answer(P,O) :-
    v3(sally,P) & v2(sally,O,D)
```

Comparing the IM and Tsimmis Query Processors

- Both in a sense synthesize queries from views.
- IM uses exhaustive search to find all solutions that "fit."
- Tsimmis expands queries using explicit rules to get solutions in terms of views at the sources.
  - This approach too can be exponential, since there may be several ways to expand each subgoal of the query.

Example

Here, Tsimmis appears to flounder, but we should emphasize that it is an atypical example, contrived for the sake of illustration.

- What are Sally's office and department?

```
q2(O,D) :-
    office(sally,O) & dept(sally,D)
```

IM finds the only minimal solution to the query:

```
answer(O,D) :- v2(sally,O,D)
```

However, using Tsimmis we can only express our query as:

```
q3(O,D) :-
    epo(sally,P,O) & edm(sally,D,M)
```

- The mediator `med` would process query q_3 by expanding each subgoal, resulting in the pair of rules:

```
answer(O,D) :-
    v1(sally,P,M1) & v2(sally,O,D1) &
    v1(sally,P1,M) & v2(sally,O1,D)
answer(O,D) :-
    v3(sally,P) & v2(sally,O,D1) &
    v1(sally,P1,M) & v2(sally,O1,D)
```

- "Tsimmis" cannot eliminate the subgoals whose predicates are the irrelevant views v_1 and v_3. Thus, it produces an empty answer in the case that we do not know a phone or manager for Sally.
- The apparent failure of Tsimmis is due only to the fact that we contrived the mediator to export inconvenient objects.
- The motivation for the design of Tsimmis is that the mediators it creates may perform some very complex processing of source data to produce its exported objects.
- In comparison, IM is limited in the way it can combine its sources; it uses LMSS.

Levels of Mediation.

- IM is designed to have two levels: the sources and the “global mediator.”
- Tsimmis supports an indefinite number of mediator levels.

Adding Sources.

- IM makes it easy to add new sources.
    - ❖ You write a wrapper for the sources and define its views in terms of the global concepts.
    - ❖ No change to the query-processing algorithm is needed. The new views will be used whenever they are appropriate for the query.
- New Tsimmis sources not only must be wrapped, but the mediators that use them have to be redefined and their MSL definitions recompiled.