

DB Semantics and Advanced Applications

VUB Computer Science 2001-2002

Prof. Dr. Robert Meersman

meersman@vub.ac.be

room: 10G 730a

0. Introduction

- Motivation 1: new applications of DBs
 - ◆ Example
- Motivation 2: pervasiveness of DBs
 - ◆ Example
- Data(-base) semantics
- Database computing principles etc.
 - ◆ Generic Design Methodology
 - ◆ Generic DB Functionality
 - ◆ Constraints (“business rules”)

Motivation 1

- 1. New applications argument:
- Applications drive computing and IT; divide into domain (-types)
- Desirable to have DB, DBMS specifically adapted to application domain-types
- Leads to specialized new DB architectures, systems and tools

New application requirements for DBMS

- 1. Active rule DBs
 - 2. (Deductive DBs)
 - 3. Temporal DBs
 - 4. Spatial and Multimedia DBs
-
- special cases of “semantic” DBMSs
-

Active Rules: example

DOMAIN

Inventory control; DB to store information about parts on hand

Inventory: Part, PartsOnHand, ReorderTreshold, ...

PendingOrder: Part, OrderQuantity, Date

DOMAIN RULE:

“Inventory levels for a part should always be above a given threshold that may depend on that part”

“ACTIVE” VERSION:

“When registering a withdrawal from inventory, check whether the level is still OK, if not automatically issue a reorder”

Active Rules: example

(ORACLE®) SQL

```
update Inventory  
set PartsOnHand = PartsOnHand - 70  
where Part = 1
```

```
update Inventory  
set PartsOnHand = PartsOnHand - 50  
where Part >= 1
```

Active Rules: example

```
create trigger Reorder
after update of PartsOnHand on Inventory
when (new.PartsOnHand < new.ReorderThreshold)
for each row
declare number #_of_orders
begin
    select count(*) into #_of_orders
    from PendingOrders
    where Part = new.Part

    if #_of_orders = 0 then
    insert into PendingOrders
    values (new.Part, new.OrderQuantity, sysdate)
    end if;
end
```

Implementation in DBMS (ORACLE[®] v. 7.2→)

- add (domain-independent) parser/compiler for the rules
- extend DBMS update runtime with algorithm:
 - execute before-triggers at statement level;
 - for each row in the target table:
 - execute before-triggers at row level;
 - perform update;
 - perform referential integrity and assertion check, and rollback update if necessary;
 - execute after-triggers at row level.
 - execute after-triggers at statement level.

Motivation 2

- 2. Pervasiveness argument:
- DB, DBMS basic tools in almost *all* aspects of modern computing
- Internet connectivity
- Strong need to share and make interoperate
- Leads to new uses for existing data(-bases)

Reuse of existing data(-bases)

- 5. Interoperability & Mediation
 - 6. Data Mining
 - 7. Data Warehousing
 - 8. Web-based Data
-
- ~~need to know what data “means”~~

Describing data(-bases) with XML

Example

```
<starlab>
  <description> STARLab people on 10G </description>
  <members>
    <member>
      <name> Robert </name>
      <since> 1994 </since>
      <email> meersman@vub.ac.be </email>
    </member>
    <member>
      <name> Ismael </name>
      <since> 1998 </since>
      <email> isanz@vub.ac.be </email>
    </member>
    <member>
      <name> Peter </name>
      <since> 1996 </since>
      <email> pstuer@vub.ac.be </email>
    </member>
  </members>
</starlab>
```

Describing data(-bases) with XML

Example

Data description using XML DTD “schema”:

```
<!DOCTYPE starlab
[
  <!ELEMENT starlab (member*)>
  <!ELEMENT member (name, since, email)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT since (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
]>
```

Element names are at document (“ontology”) level;
Data schemas may be shared-accessible

```
<!DOCTYPE starlab SYSTEM
"http://www.starlab.vub.ac.be/courses/AdvDBApp/schema.dtd">
```

Describing data(-bases) with XML

Example

DTD elements may have “non-document” *attributes*:

```
<!DOCTYPE starlab
[
  <!ELEMENT starlab (member*)>
  <!ELEMENT member (name, since, email)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT since (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
  <!ATTLIST member      pid          ID          #REQUIRED
                       reports_to  IDREF       #IMPLIED>
]>
```

Describing data(-bases) with XML

Example

Example XML document conforming to this DTD:

```
<starlab>
  <member pid="rm">
    <name> Robert </name>
  </member>
  <member pid="mj" reports_to="rm">
    <name> Mustafa </name>
  </member>
  <member pid="is" reports_to="rm">
    <name> Ismael </name>
  </member>
  <member pid="sc" reports_to="ps">
    <name> Sven </name>
  </member>
</starlab>
```

→ Common to *both* arguments:

- DBMSs become explicitly concerned with *meaning* of stored data: **data semantics**
- Formal representations of meaning are desirable/required:
 - ◆ “*hard-wired*”: extensions of DBMS
 - ◆ *stored (accessible)*: **meta data (-base)** needed for description
 - ◆ *combination* of both

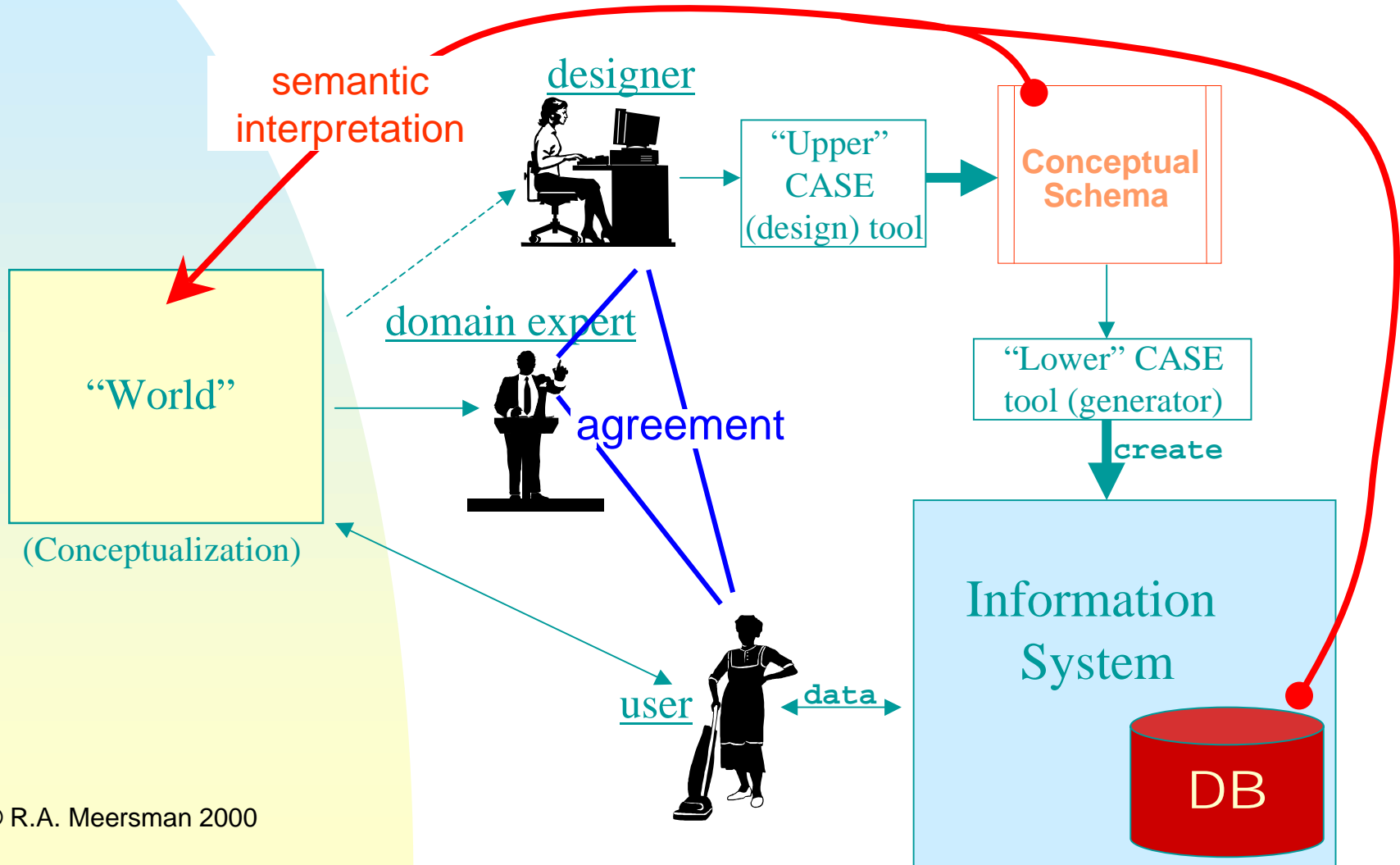
Treatment of semantics in this course

- Theory: separate future course subject “Database Semantics”
 - ◆ declarative (Tarski) semantics
 - ◆ interpretation mappings
 - ◆ ontologies & Kripke semantics
- Application: in this course, domain- and problem-specific solutions, subject “Advanced DB Applications”

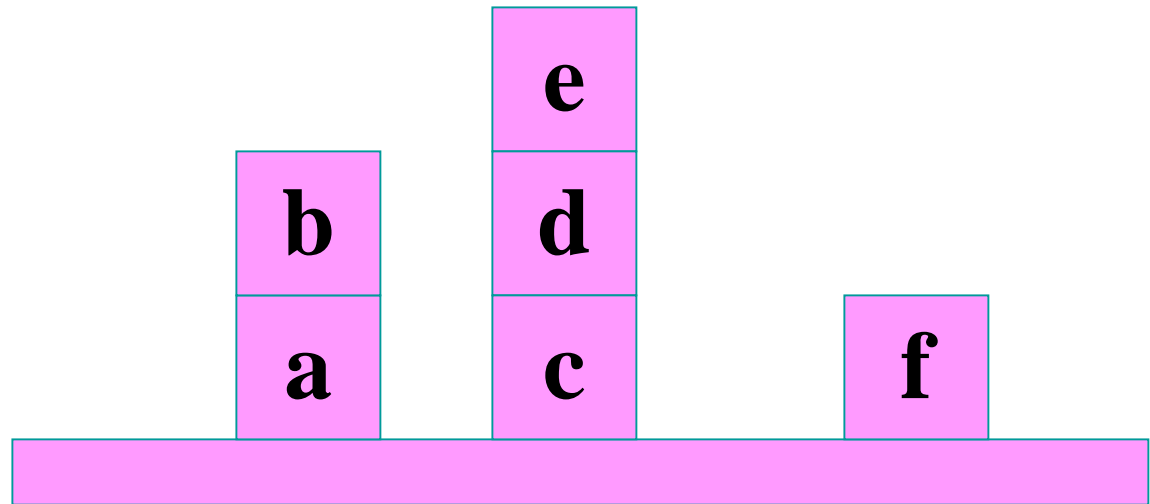
Declarative Semantics for Information Systems

- **Based on formal and strict separation** of language (schema, lexical expressions) and the “world” it describes
- so-called *Tarski semantics* defined by *interpretation mapping* from schema to “world”
- cf. *M. Genesereth & N. Nilsson*, “Logical Foundations of Artificial Intelligence”, chapters 1-2

Declarative semantics



Blocks World example

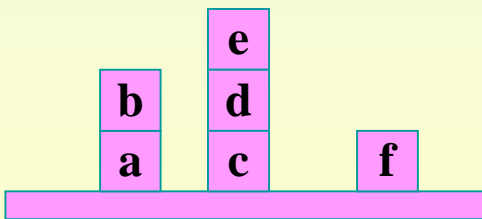


Universe of Discourse (UoD): {a, b, c, d, e, f}

→ table? Other stuff?

Conceptualization

- Replaces “real world” in modeling of semantics, for mathematical purposes
- semantics = mapping symbols onto “real world” *objects, relationships, functions, events*
- ignore events —for the moment
- **Conceptualization =**
 - ◆ 1. set of objects
 - ◆ 2. set of (mathematical) functions
 - ◆ 3. set of (mathematical) relations



Functions and Relations

Names (symbols) will be used *only* in conceptual schema language, where convenient

But need some way of identifying and referring to elements of conceptualization, too
→ use *italics* for function and relation names

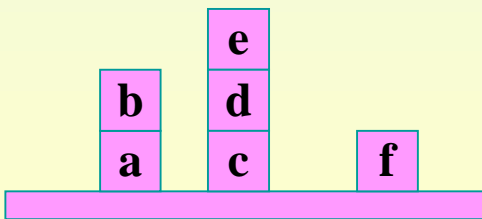
Example: function *bottom_block_of* :

$\{ \langle b, a \rangle, \langle d, c \rangle, \langle e, c \rangle \}$

expresses a mapping $UoD \rightarrow UoD$:

$a \rightarrow b, c \rightarrow d, c \rightarrow e$

for instance “ $c = \textit{bottom_block_of}(e)$ ”



Functions and Relations

Functions and relations are subsets of Cartesian products of underlying domains:

$$\textit{bottom_block_of} \subseteq \text{UoD} \times \text{UoD} = \text{UoD}^2$$

Functions, as usual, “compute” a result (= last component of product). Relations “express” boolean true facts; example: (binary) relation

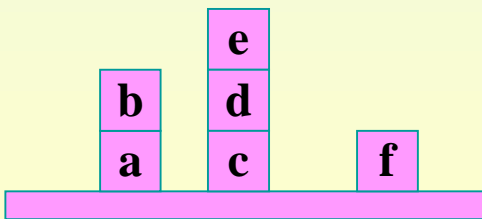
$$\textit{sits_on} \subseteq \text{UoD} \times \text{UoD} \quad \text{defined by}$$

$$\{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle \}$$

expresses that “b *sits_on* a”, etc.

Example: relation *is_above* $\subseteq \text{UoD} \times \text{UoD}$ defined by

$$\{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle, \langle e, c \rangle \}$$



Functions and Relations

Relations may be unary:

example: $is_clear \subseteq UoD^1$ defined by

$$\{ \langle b \rangle, \langle e \rangle, \langle f \rangle \}$$

expresses no blocks are on top of these; we also write $\{ b, e, f \}$.

Example: $sits_on_table \subseteq UoD^1$ defined by

$$\{ \langle a \rangle, \langle c \rangle, \langle f \rangle \} \approx \{ a, c, f \}$$

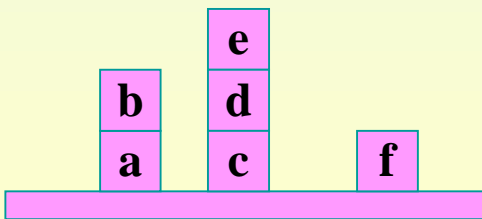
Conceptualization for Blocks World is now a triple $\langle UoD, F, R \rangle$:

$\langle \{ a, b, c, d, e, f \}; \{ \langle b, a \rangle, \langle d, c \rangle, \langle e, c \rangle \};$

$\{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle \}, \{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle,$

$\langle e, c \rangle \}, \{ \langle b \rangle, \langle e \rangle, \langle f \rangle \}, \{ \langle a \rangle, \langle c \rangle, \langle f \rangle \} \rangle$

\rightarrow *is this complete?*



Conceptual Schema

- Language
- part of (usually first order) predicate calculus
- expressed in formal syntax
 - ◆ requires symbols for objects, relations, operators, quantification, ...
 - ◆ constructs terms, formulas (= sentences)
 - ◆ formula expresses “true” fact about UoD
- cf. relational domain- & tuple-calculi!

Conceptual Schema languages

- Propositional logic
 - First-order predicate calculus (FOPC)
 - ◆ terms
 - ◆ well-formed formulas (wff)
 - declarative semantics
 - ◆ interpretation mapping
 - models vs. proofs (Reiter paper)
 - proof-theoretic relational database
-

Propositional logic

- Formal language
- Symbols: p, q, r, \dots : denote facts, sentences
- operators: $\wedge, \vee, \rightarrow, \neg, \dots$: defined by truth tables
- syntax:

p

$p \wedge q$

$(p \vee q) \vee \neg r \rightarrow s$

Propositional logic

- “Semantics”:

p	q	$p \vee q$
t	t	t
t	f	t
f	t	t
f	f	f

- But: no variables
- no quantification \forall, \exists
- no “type” abstraction

First order predicate calculus (FOPC): language L

- Alphabet of *characters*

A, B, C, ..., a, b, c, ..., 1, 2, ..., +, -, \forall , \exists , ...

- Symbols

- ◆ variables **x, y, z, abc, index, e19, ...**

- ◆ constants **Jan, E19, 2001, 587.12, ...**

- Function constants

- ◆ constant symbol: prefix notation + (...)

Weight, Number, Age, ...

- ◆ operator symbol : infix notation

+, -, *, \cup , \cap , ...

FOPC functions in L

- Functions (cont'd.)
 - ◆ *arity*: # of arguments resp. operands
 - ◆ function arity > 0

Weight (.)
1

Sum (. , .)
2

$x + y$
2

FOPC relation symbols in L

- Relation symbols

- ◆ relation constant --prefix notation

Is_father, Odd, Sits_Above, ...

- ◆ relation operator --infix notation

=, <, >, ≠, ≥, ≤, ∈, ⊂, ⊆

- ◆ arity ≥ 0 (=0: equiv. to “proposition constant”)

Is_father(Robert, Daniel)

2

Odd(6)

1

Cool

0

FOPC: well-formed formulas (wffs) of the language L

- sentences of the FOPC language, denote supposed facts in the “real world” (modulo formal *interpretation*)
 - ◆ atomic sentences: “relation expressions”
 - ◆ logical sentences: combined with logical operators
 - ◆ quantified sentences: binding free variable(s)
 - ◆ set of wffs of L is denoted $Wff(L)$

wffs: atomic sentences

- $R(t_1, t_2, \dots, t_n)$ “relational expression”
terms

relation constant

`Is_Father(x, Sarah)`

`Is_Sum(2, 3, 6)`

- n-ary function \rightarrow (n+1)-ary relation

`Age(Methuselah) = 969`

`Equal(Age(Methuselah), 969)`

`Age(Methuselah, 969)`

wffs: logical sentences

- If φ_1 and φ_2 are wffs then also
 - ◆ $(\varphi_1 \wedge \varphi_2)$
 - ◆ $(\varphi_1 \vee \varphi_2)$
 - ◆ $(\varphi_1 \rightarrow \varphi_2)$
 - ◆ $(\varphi_1 \leftrightarrow \varphi_2)$
 - ◆ $\neg\varphi_1$

$(\text{Up}(\text{Heads}) \vee \text{Up}(\text{Tails}))$

$((\text{Dark} \wedge \text{Cat}(x)) \rightarrow \text{Grey}(x))$

$\text{Rolling}(\text{stone}) \leftrightarrow \neg\text{Gathers}(\text{stone}, \text{Moss})$

wffs: quantified sentences

- universally \forall
- existentially \exists
- if φ is a wff, and x is a *free variable* in φ then also
 - ◆ $(\forall x \varphi)$ is a wff
 - ◆ $(\exists x \varphi)$ is a wff
- φ is called the *scope* of x

wffs: free and bound variables

- $(\text{Man}(x) \rightarrow \text{Mortal}(x))$
x is free
 - $(\forall x (\text{Man}(x) \rightarrow \text{Mortal}(x)))$
x is bound
 - $(\text{Employee}(z) \vee (\exists z (\text{Manager}(z))))$
x is both free and bound
-
- wff without free variables: *closed wff*
 - wff without variables: *ground formula*

(note) second order logics

- if domain of variables is allowed to be functions or relations → *second order language/calculus/logic*

$$(\forall f \forall x, y1, y2 ((f(x)=y1) \vee (f(x)=y2)) \rightarrow (y1=y2)))$$

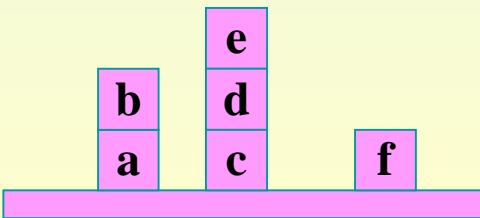
...back to the Blocks World...

- Conceptualization is a triple $\langle U, F, R \rangle$:

$\langle \{a, b, c, d, e, f\};$

$\{ \langle b, a \rangle, \langle d, c \rangle, \langle e, c \rangle \};$

$\{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle \}, \{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle, \langle e, c \rangle \}, \{ \langle b \rangle, \langle e \rangle, \langle f \rangle \}, \{ \langle a \rangle, \langle c \rangle, \langle f \rangle \} \rangle$



Interpretation of a first order language L

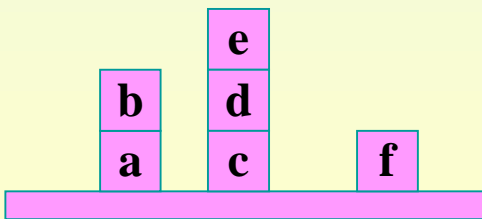
- Notation: $|I|$ for Universe of Discourse U
- An *interpretation* is a mapping I :
Conceptual Schema \rightarrow Conceptualization
 $\sigma \rightarrow I(\sigma)$

such that:

if σ is an object constant, $I(\sigma) \in |I|$;

if ϕ is a n -ary function constant, $I(\phi): |I|^n \rightarrow |I|$;

if ρ is a relation constant, $I(\rho) \subseteq |I|^n$



Example: Interpretation of Blocks World constants

- Symbols of Blocks World language =
{ **A, B, C, D, E, F, Bottom,**
On, Above, Clear, On_Table}

$I(\mathbf{A}) = a, I(\mathbf{B}) = b, I(\mathbf{C}) = c,$

$I(\mathbf{D}) = d, I(\mathbf{E}) = e, I(\mathbf{F}) = f;$

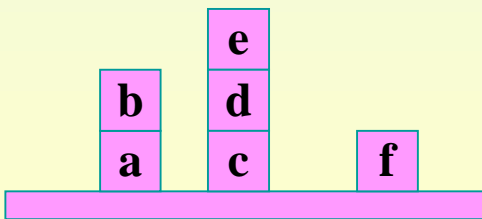
$I(\mathbf{Bottom}) = \{ \langle b, a \rangle, \langle d, c \rangle, \langle e, c \rangle \};$

$I(\mathbf{On}) = \{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle \},$

$I(\mathbf{Above}) = \{ \langle b, a \rangle, \langle e, d \rangle, \langle d, c \rangle, \langle e, c \rangle \},$

$I(\mathbf{Clear}) = \{ \langle b \rangle, \langle e \rangle, \langle f \rangle \},$

$I(\mathbf{On_Table}) = \{ \langle a \rangle, \langle c \rangle, \langle f \rangle \}.$



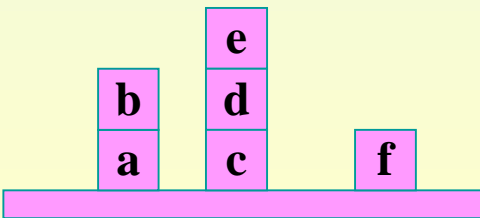
Interpretation of variables

- Let L be a F.O. language
- A *variable assignment* is a mapping
$$U: \mathbf{Var}(L) \rightarrow U$$
- Example assignment for Blocks World:

$\text{MyU}(\mathbf{x}) = a,$

$\text{MyU}(\mathbf{y}) = b,$

$\text{MyU}(\mathbf{z}) = a, \dots$

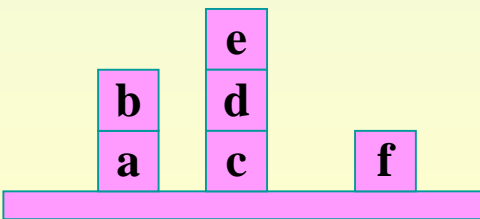


Interpretation of terms

- “Combine” a variable assignment U with an interpretation I of object constants:

$$IU: \mathbf{Term(L)} \rightarrow \mathbf{U}$$

- is a mapping such that
 - if τ is an object constant, $IU(\tau) = I(\tau)$;
 - if τ is a variable, $IU(\tau) = U(\tau)$;
 - if τ is $\phi(\tau_1, \dots, \tau_n)$ where $I(\phi) = f$ and $IU(\tau_i) = x_i$, $IU(\tau) = f(x_1, \dots, x_n)$ or equivalently : $\langle x_1, \dots, x_n, IU(\tau) \rangle \in f$



Example: Interpretation of Blocks World terms

- Let I be the “natural” interpretation
- Let MyU be a variable assignment

$$MyU(\mathbf{x}) = a,$$

$$MyU(\mathbf{y}) = b,$$

$$MyU(\mathbf{z}) = a, \dots$$

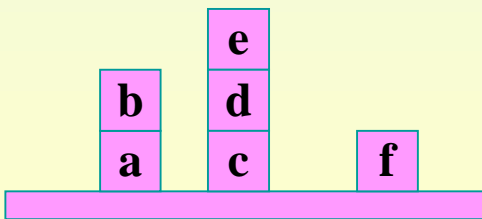
- The corresponding term assignment is

$$MyIU(\mathbf{E}) = e$$

$$MyIU(\mathbf{z}) = a$$

$$MyIU(\mathbf{Bottom}(\mathbf{E})) = c \quad \text{since}$$

$$\langle e, c \rangle \in I(\mathbf{Bottom})$$

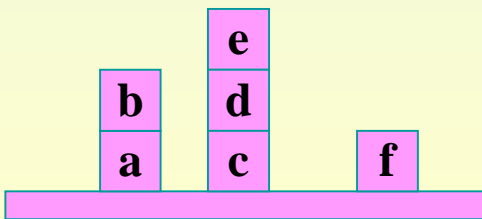


Satisfaction and truth

- Denote a sentence $\varphi \in \mathbf{Wff(L)}$ as *satisfied* by an interpretation I and variable assignment U by

$$\models_I \varphi[U]$$

- Also say that “ φ is *true* under interpretation I and relative to variable assignment U ”
- Relative notion of truth...
- Definition?

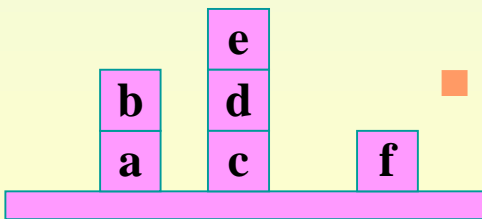


Satisfaction: definitions

meta symbols

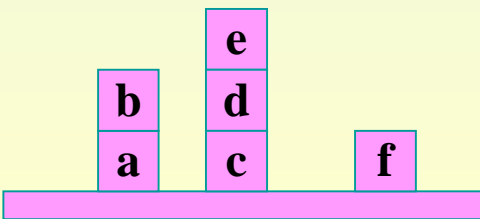
basis for recursion

- $\models (\sigma = \tau)[U]$ iff $IU(\sigma) = IU(\tau)$
- $\models \rho(\tau_1, \dots, \tau_n)[U]$ iff $\langle IU(\tau_1), \dots, IU(\tau_n) \rangle \in I(\rho)$
- $\models (\neg\varphi)[U]$ iff not $\models \varphi[U]$
- $\models (\varphi_1 \wedge \varphi_2)[U]$ iff $\models \varphi_1[U]$ and $\models \varphi_2[U]$
- $\models (\varphi_1 \vee \varphi_2)[U]$ iff $\models \varphi_1[U]$ and/or $\models \varphi_2[U]$
- $\models (\varphi_1 \rightarrow \varphi_2)[U]$ iff not $\models \varphi_1[U]$ or $\models \varphi_2[U]$
- $\models (\varphi_1 \leftrightarrow \varphi_2)[U]$ iff $\models (\varphi_1 \rightarrow \varphi_2)[U]$ and $\models (\varphi_2 \rightarrow \varphi_1)[U]$



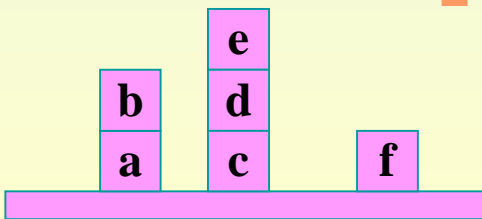
Satisfaction: definitions_(cont'd.)

- Quantified sentences are satisfied depending on variable assignment:
- $\models (\forall \mathbf{x} \varphi)[U]$ iff for all $s \in |I|$ we have $\models \varphi[V]$ where the assignment V is defined by $V(\mathbf{x}) = s$ while $V(\mathbf{y}) = U(\mathbf{y})$ for all other variables $\mathbf{y} (\neq \mathbf{x})$
- $\models (\exists \mathbf{x} \varphi)[U]$ iff for some $s \in |I|$ we have $\models \varphi[V]$ where the assignment V is defined by $V(\mathbf{x}) = s$ while $V(\mathbf{y}) = U(\mathbf{y})$ for all other variables $\mathbf{y} (\neq \mathbf{x})$



Satisfaction in the Blocks World

- $((y = \text{Bottom}(x)) \rightarrow \text{Above}(x, y))$
- \rightarrow true for which variable assignments and interpretations?
- Under “natural” interpretation, true for *all* assignments – check!
- We say that such interpretation is a ***model*** for the sentence

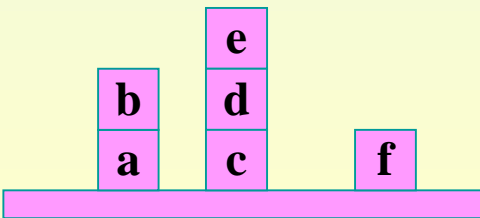


Models

- Interpretation I is **a model** for a set Γ of sentences, denoted

$$\models_I \Gamma$$

iff for each sentence φ in Γ
the interpretation I satisfies φ
for all variable assignments



Example of models: databases!

- Database instance: viewed as logical model of its schema (= relations + integrity constraints)
- “classical” (historical) view of DB
- not unique (in general ∞) \rightarrow need to specify which is the *intended* answer
- *Herbrand interpretation*: identifies the constants of the language (=DB instance) with the “objects” of the UoD

Study Assignment: Reiter Paper

- “Towards a Logical Reconstruction of Relational Database Theory”
- Raymond Reiter, 1984

Introduction: model-theoretic DB

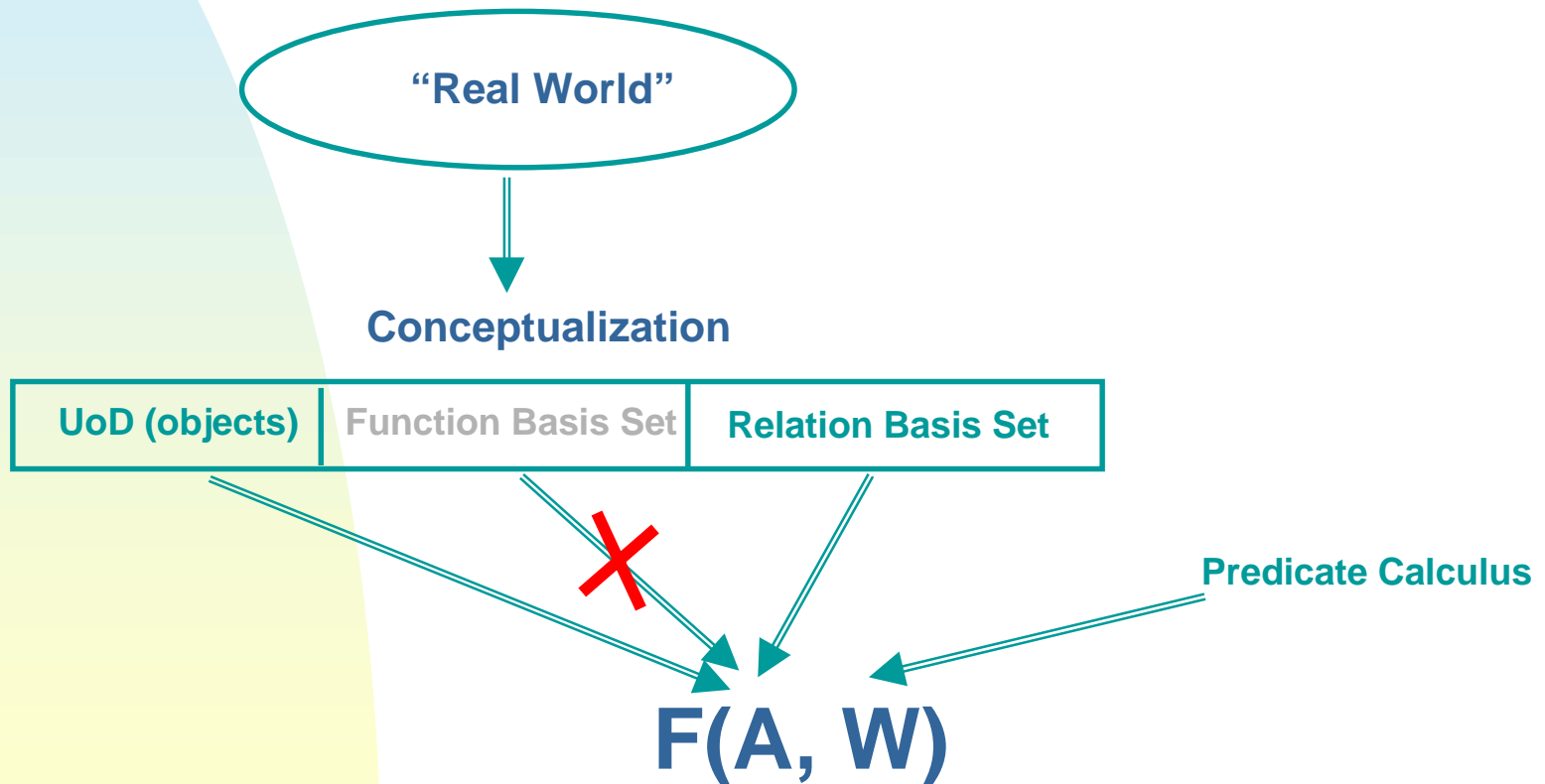
- “Classical” relational database theory is defined *model-theoretically*:
 - ◆ A DB is viewed as a particular kind of First Order Interpretation
 - ◆ Query of DB becomes a process of truth-functional evaluation of F.O. formulas with respect to this interpretation
 - ◆ Imposes limitations on representation of certain “real world situations”

Introduction: proof-theoretic DB

- A so-called *proof theoretic* view of relational DB is possible, and to a certain extent (claimed by Reiter) more fruitful
 - ◆ Idea has capacity for generalization
 - ◆ Permits to extend the relational theory in order to include more real world knowledge

The Model-theoretic Perspective

- 1. First Order Languages “re-defined”



UoD / Conceptualization



- Suppliers
- Parts
- “Supplies” relationship
- Cities
- Names
- Shipments
- ...

The Model-theoretic Perspective

- 1. First Order languages (based on predicate calculus --*note*: Reiter's notation slightly different)
 - ◆ $F=(A, W)$ where A is an alphabet en W is a set of wffs
 - ◆ A is composed of
 - Constant symbols
 - Predicate symbols
 - Variable symbols
 - Punctuation signs : () ,
 - Logical Constants : \supset (implies), \wedge , \vee , \neg , \equiv (iff)
 - ◆ a term is either
 - a constant
 - a variable

The Model-theoretic Perspective

◆ Atomic formulas

$P(t_1, \dots, t_n)$

P is n -ary predicate symbol

t_i are terms, $i = 1, \dots, n$

◆ Well Formed Formulas (wffs)

◆ An atomic formula is a wff

◆ If w_1 and w_2 are wff's then also

$(w_1 \wedge w_2), (w_1 \vee w_2), (w_1 \supset w_2), (w_1 \equiv w_2), (\neg w_1)$

◆ If x is a variable and W a wff then also

$(\forall x)W$ [equiv. to $(\forall x)W$]

$(\exists x)W$

The Model-theoretic Perspective

- Describing relational DB as first order language

$$\mathbf{F}=(A, W)$$



Relational Language

$$\mathbf{R}=(A, W)$$

Finitley many constants, at least one
Finitley many predicates, at least one
Presence of the = Predicate
Some unary P called *simple types*

First Order Relational Language

- A first order language $R=(A, W)$ is a *relational language* if its only syntactical elements in A are
 - ◆ **CONSTANTS:** S1, S2, Smith, Jones, London, Paris, P1, P2, Screw, Screwdriver, 5, 100, 200
 - ◆ **PREDICATES:** S#(.), Sname#(.), City(.), P#(.), Pname#(.), QTY#(.), Supplier#(.,.,.), Part#(.,.), Shipment#(.,.,.)
 - ◆ **SIMPLE TYPES:** S#(.), Sname#(.), City(.), P#(.), Pname#(.), QTY#(.)



The Model-theoretic Perspective

- Semantics of a FOL (in Reiter's terminology)
 - ◆ As is standard, give an *interpretation* to each of the symbols of A and use this mapping as a basis to define truth values of arbitrary wffs in W constructed from these symbols.
 - ◆ Define interpretation $I = (D, K, E)$ for $\mathbf{F}=(A, W)$:
 - D (Domain, was UoD) is the nonempty set of objects over which the variables of A are meant to range
 - K is a mapping of the constants of A into D
 - E is a mapping of the predicate symbols of A into given sets of *tuples* of elements of D ("populations"). For a predicate symbol P , $E(P)$ is called the *extension* of P .

An interpretation for $R=(A,W)$

- Define $I = (D, K, E)$ such that
 - ◆ $D = \{ S1, S2, Smith, Jones, London, Paris, P1, P2, Screw, Screwdriver, 5, 100, 200 \}$
 - ◆ $K : A \rightarrow D: \forall c \in A \ K(c) \in D$
 - ◆ E : defined by “populations” or “extensions”:

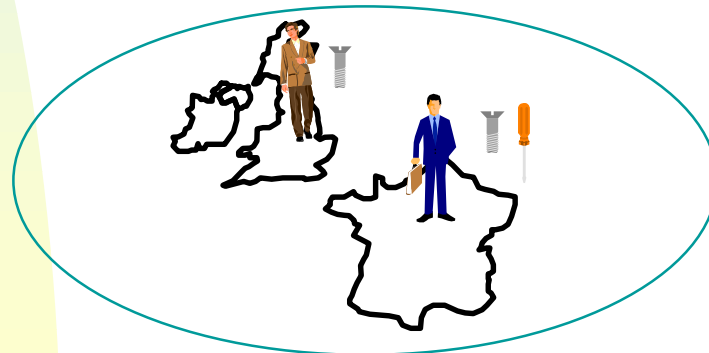
S#	Sname	City	P#	Pname	Qty
S1	Smith	London	P1	Screwdriver	5
S2	Jones	Paris	P2	Screw	100
					200

Supplier

S1 P1
S2 P2

Shipment

S1 P2 200
S2 P1 5
S2 P2 100



=	
S1	S1
S2	S2
Smith	Smith
Jones	Jones
London	London
Paris	Paris
P1	P1
P2	P2
Screwdriver	Screwdriver
Screw	Screw
5	5
100	100
200	200

The Model-theoretic Perspective

- 2. (Classical, declarative) semantics of FOL:
 - ◆ Let ρ be a variable assignment (= partial mapping of A into D)
 - ρ is also called an *environment* for the variables of A
 - ◆ w is *true in the interpretation* I iff w is true under I for each possible environment
 - ◆ I is a **model** for w if w is true under I
 - ◆ I is a **model** for a set W of wffs if each $w \in W$ is true under I
 - ◆ *Note.* In the case of finite interpretations, determining the truth of an arbitrary wff reduces to purely propositional truth tables evaluation

An interpretation for $R=(A,W)$

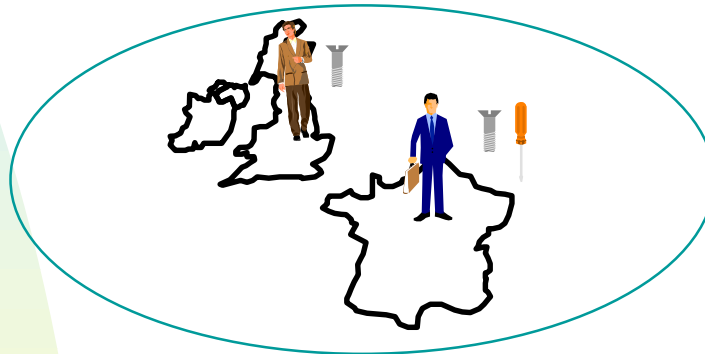
S#	Sname	City	P#	Pname	Qty
S1	Smith	London	P1	Screwdriver	5
S2	Jones	Paris	P2	Screw	100

Supplier

S1	P1
S2	P2

Shipment

S1	P2	200
S2	P1	5
S2	P2	100



=	
S1	S1
S2	S2
Smith	Smith
Jones	Jones
London	London
Paris	Paris
P1	P1
P2	P2
Screwdriver	Screwdriver
Screw	Screw
5	5
100	100
200	200

- This I is a model, for example, for the sentence $(x/S\#)(y/P\#)(z/Qty)(\text{Shipment}(x, y, z) \supset \text{Supplier}(x, y))$

The Model-theoretic Perspective

- Relational databases redefined... (cont'd.)
 - ◆ Class of *relational* interpretations
 - R** = (A, W)
 - I** = (D, K, E) is a *relational interpretation* iff
 - K**: Constants of A $\xrightarrow{1-1}$ D (so D is finite!)
 - E**(=) = { (d,d) | d ∈ D }
 - ◆ A *relational DB* is a triple (**R**, I, IC) such that
 - R** is a relational language
 - I** is a relational interpretation
 - IC** is a set of wffs of R called *integrity constraints*, for each n-ary predicate P containing at least the wff $(x_1) \dots (x_n) (P(x_1, \dots, x_n) \supset \tau_1(x_1) \dots \tau_n(x_n))$ for simple types $\tau_1 \dots \tau_n$

The Model-theoretic Perspective

- Relational databases redefined ... (cont'd.)
 - ◆ For each predicate (except = and simple types), the extension $E(P)$ is called, naturally, a *relation*
 - ◆ The IC of a relational DB = (\mathbf{R}, I, IC) are said to be *satisfied* if I is a model for IC
- 4. First Order *query languages*
 - ◆ $Q = \langle \vec{x}/\vec{\tau} \mid w(\vec{x}) \rangle$ is a query (“vector” notation)
 - ◆ If $DB = (\mathbf{R}, I, IC)$ then a query Q for \mathbf{R} is said to be *applicable* to DB
 - ◆ A tuple \vec{c} of constants of \mathbf{R} 's alphabet is an answer to Q with respect to DB iff

$\tau_i(c_i)$ is true in I , $i = 1, \dots, n$ for simple types τ_i
 $w(\vec{c})$ is true in I

Model-theoretic query evaluation

S#	Sname	City	P#	Pname	Qty
S1	Smith	London	P1	Screwdriver	5
S2	Jones	Paris	P2	Screw	100

Supplier

S1	P1
S2	P1
S2	P2

Shipment

S1	P2	200
S2	P1	5
S2	P2	100

=	
S1	S1
S2	S2
Smith	Smith
Jones	Jones
London	London
Paris	Paris
P1	P1
P2	P2
Screwdriver	Screwdriver
Screw	Screw
5	5
100	100
200	200

- A query in the relational model reduces to formula truth value evaluation, for example:

$\langle (x / Pname) (y / City) (z / Qty) \mid (\exists u / S\#)(\exists v / Sname)(\exists w / P\#)$
 $(Part(w, x) \wedge Supplier(u, v) \wedge Shipment(u, v, z)) \rangle$

Pname	City	Qty
Screw	London	200
Screwdriver	Paris	5
Screw	Paris	100

The Model-theoretic Perspective

- 5. Some perceived problems with the model theoretic perspective (see Reiter paper for details)
 - ◆ DB with incomplete information
 - Disjunctive information
 - NULL values
 - ◆ Extending the Relational Model to incorporate more real world knowledge
 - General facts & integrity constraints
 - Events: sequencing and time of occurrence
 - Generalization hierarchies (IS-A hierarchies) with property inheritance

The Proof-theoretic Perspective

■ 1. Relational *theories*

◆ $DB = (\mathbf{R}, I, IC) \Rightarrow DB = (\mathbf{R}, T, IC)$, where

T (First Order Theory) is like interpretation *I*, but now viewed as a set of *ground atomic formulas*,

+ *Domain Closure Axiom*

+ *Unique Name Axiom*

+ *Equality axioms* (reflexivity, commutativity, transitivity of equality and the principle of substitution of equal terms)

+ The ground atomic facts

+ *Completion axioms* for each predicate

The Proof-theoretic Perspective

■ 1. Relational *theories*

◆ Theorems :

Suppose $\mathbf{R}=(A, W)$ is a relational language.

- ① If T is a relational theory of \mathbf{R} , then T has a *unique model* I which is a relational interpretation for \mathbf{R} .
- ② If I is a relational interpretation for \mathbf{R} then there is a relational theory T of \mathbf{R} such that I is the *only model* of T

◆ Corollary:

- ③ Suppose T is a relational theory of a relational language \mathbf{R} and that I is a model of T . Then for any wff w of \mathbf{R} , w is true in I iff it is possible to *prove w from T* , denoted $T \vdash w$.

The Proof-theoretic Perspective

- 2. A proof-theoretic reconstruction of Relational Database theory:
 - ◆ $DB = (\mathbf{R}, T, IC)$ where
 - \mathbf{R} and IC as before;
 - T is a relational theory of \mathbf{R} ;
 - The IC are said to be *satisfied* iff for each $w \in IC$, $T \mid\text{---} w$
 - Q (query) applicable to a database

Equivalence of model- and proof-theoretic perspectives

- Need *Closed World Assumption (CWA)*
“what’s not in the DB, is false”
- → meaning of NULL values...?
- *Minimal model*: --under CWA, only the facts that are satisfied in *all models* of the database schema are assumed to be true
- **Theorem** (Reiter): in this case, proof-theoretic approach is \approx equivalent to model-theoretic approach

Summarizing some practical aspects of DB semantics

- ① Relational DB: structure (data “syntax”) only —or almost...
- ② → relational model is nearly completely data- (& application-) independent
- ③ → all domain-specific meaning modeled in application programs
- ④ happy and continuous re-invention of wheel & warm water 😊
- ⑤ → inconsistency, lack of interoperability

Summarizing some practical aspects of DB semantics...

- ⑥ May package domain-specific knowledge into “domain libraries”
- ⑦ Use through instantiation and parameterization (design patterns)
- ⑧ Handle “universal” semantic aspects in extensions to DBMS (e.g. time, space, document structure, ...)
- ⑨ Further generalize DBMS technology: objects, rules, constraints, ...

“More” semantics in DB : *Advantages*

- Standardizes “documentation” of meaning of data and data handling
 - Guarantees agreement on meaning
 - Enables enforcement
 - Reduces redundancy and size
 - Increases maintainability

“More” semantics in DB : *Disadvantages?*

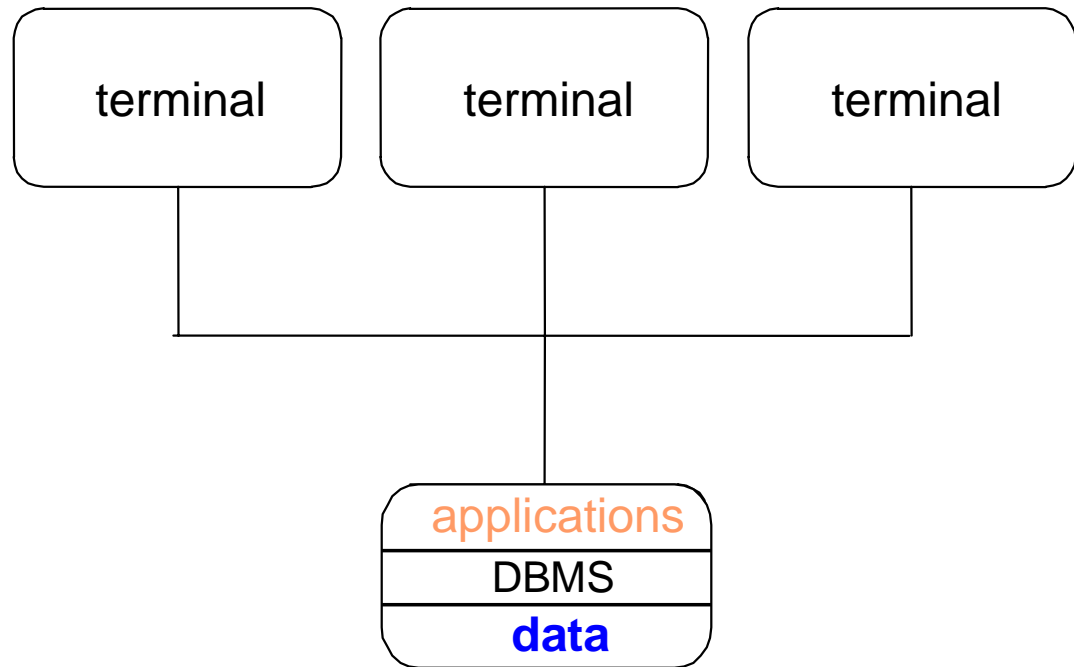
- Efficiency problems
 - ◆ generality of solution → common overhead independent of problem complexity
 - ◆ interpretation vs. compilation
- Methodology issues & R.o.I.
 - ◆ complexity & price of tools
 - ◆ adoption of new paradigms
 - ◆ long term stability & support

Database computing paradigms

- A. Functional and Methodological
 - ◆ DBMS \leftrightarrow Application software
 - ◆ 3-schema model: Conc, Int, Ext
 - ◆ Conceptual Schema and data semantics
- B. Architectural
 - ◆ Centralized
 - ◆ Client/Server
 - ◆ On-line (multi-server)

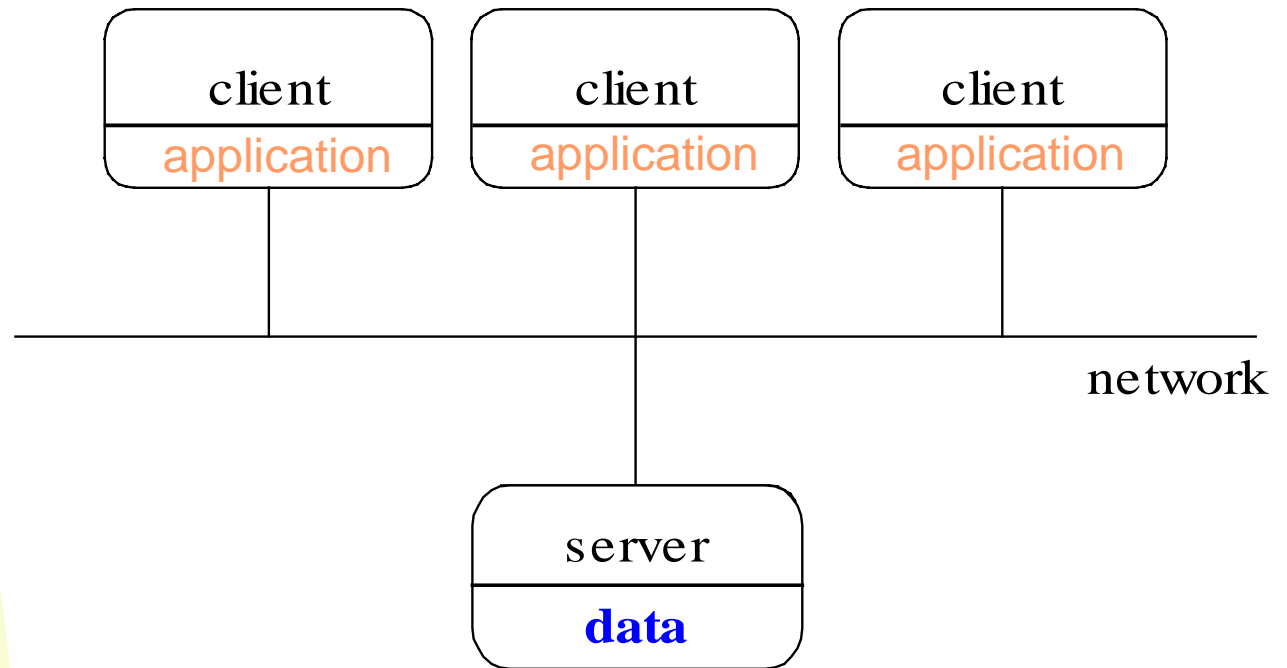
DB system architectures

1. *Centralized*



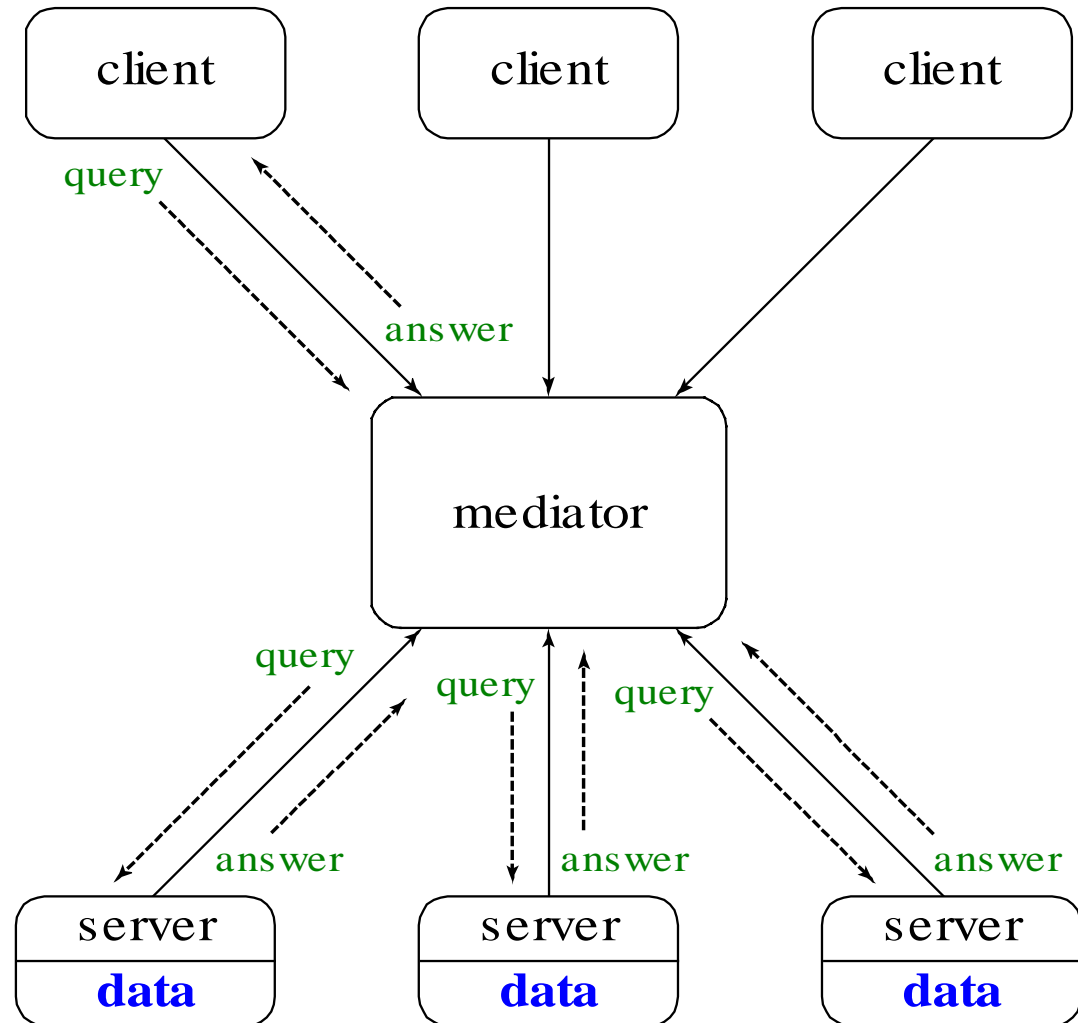
DB system architectures

2. *Client-Server*



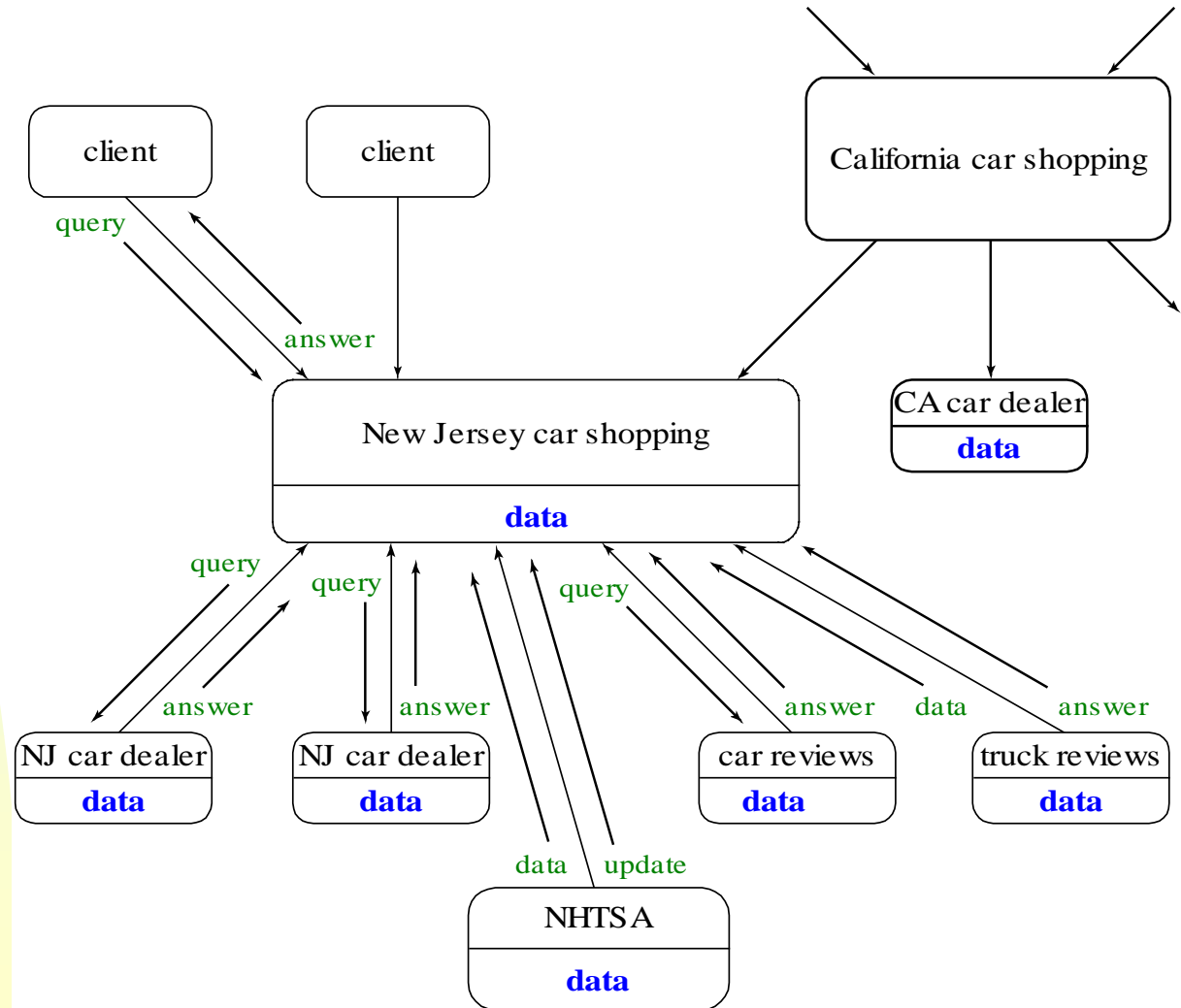
DB system architectures

3a. On-line (multi-server), mediated

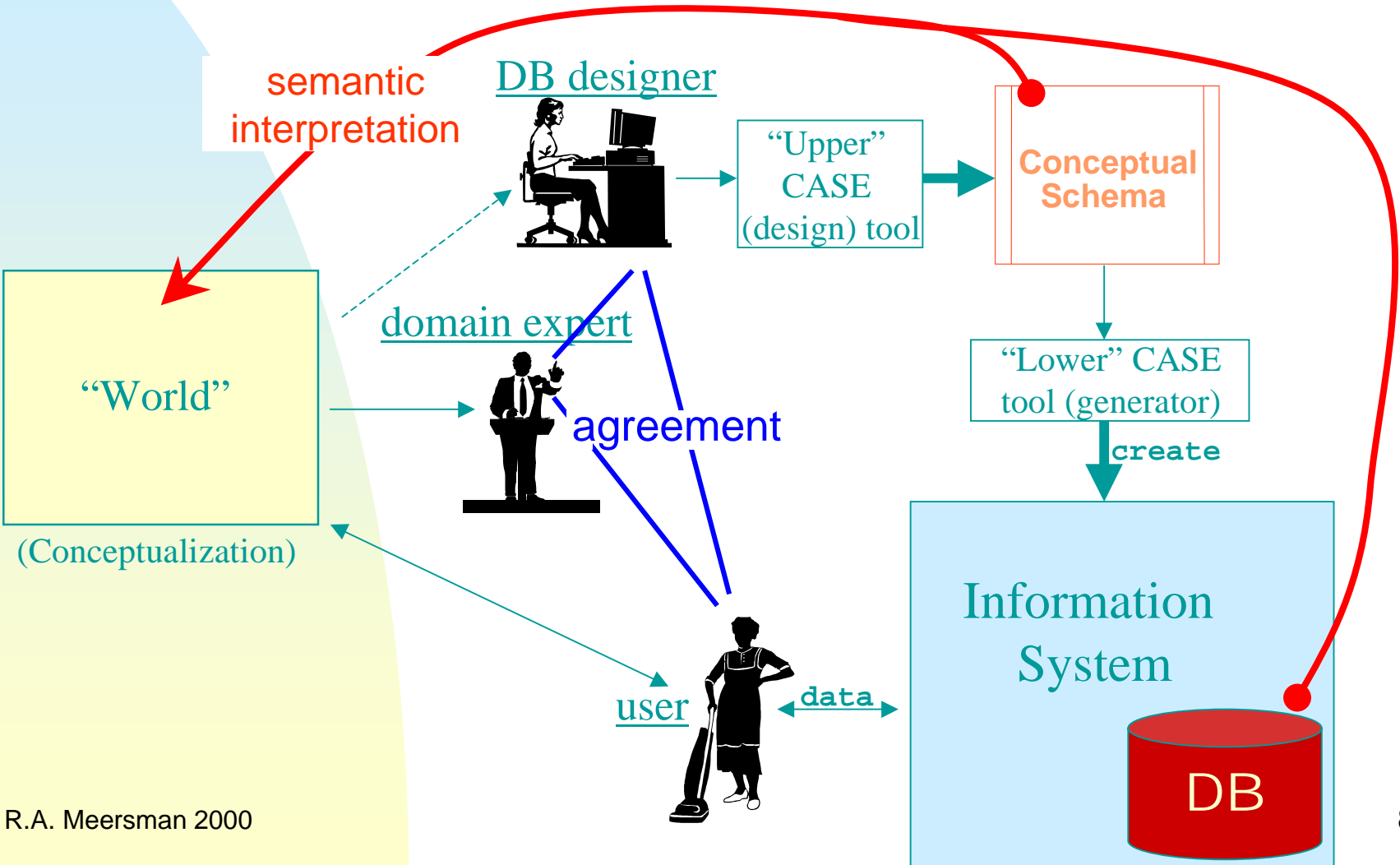


DB system architectures

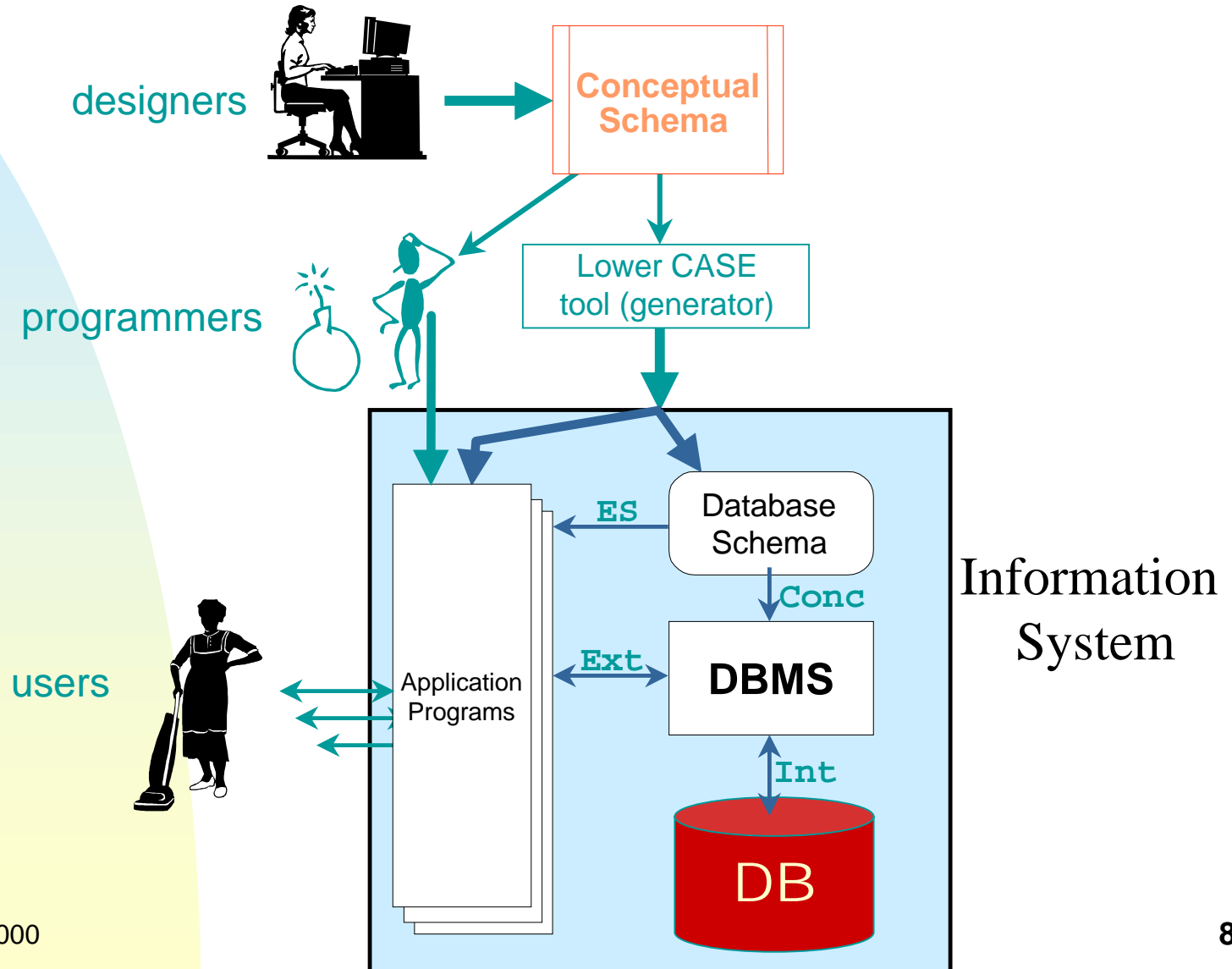
3b. On-line (multi-server, multi-tier)



Generic DB system design methodology



Generic DB system functionality



Constraints (“business rules”)

1. *Specification*

- “Oldest” form of explicit DB semantics
 - ◆ require a “special” syntax → “rules”
 - ◆ part of Conceptual Schema
 - ◆ syntax may be domain-specific
- Predicates must hold for the entire DB, in logic terms the DB is a *model* for (among other things) the constraints
 - ◆ in principle, no indication on **how** truth is to be enforced (e.g. by DBMS)
 - ◆ procedural vs. declarative

Constraints (“business rules”)

Specification (cont’d.)

Domain: Inventory control

Inventory: Part, PartsOnHand, ReorderTreshold, ...

PendingOrder: Part, OrderQuantity, Date

Domain “business rule”:

“Inventory levels for a part should always be above a given threshold that may depend on that part”

```
for each p in dom(Part) holds
((select PartsOnHand from Inventory
  where Inventory.Part = p)
 - select(sum(Orderquantity) from PendingOrder
  where PendingOrder.Part = p)
 >= select ReorderThreshold from Inventory
  where Inventory.Part = p)
```

Constraints (“business rules”)

Specification (cont’d.)

- Q.: what are the key differences between the declarative and procedural approaches?
 - ◆ Specification & design
 - ◆ Implementation

Constraints (“business rules”)

2. Implementation (enforcement)

- Spectrum of implementation choices:
 - ◆ (some) hard-wired in DBMS
 - ◆ (all?) interpreted by DBMS++
 - ◆ compiled & packaged, between DBMS and application (e.g. *design patterns*)
 - ◆ hand-coded inside applications
 - methodically & systematically?
 - free-form & ad-hoc?

1. Active Databases

- “ECA” model
 - ◆ Event-Condition-Action
 - ◆ for relational, object-based DBs
- translation of *constraints* into *triggers*
 - ◆ Boolean condition → procedure
- transaction-based, SQL2 or SQL3
 - ◆ Oracle[®], DB2[®], Sybase[®], Informix[®] (incl. Illustra), StarBurst, ...

Active Database Models

- “ECA” model: abstract syntax
trigger <t-name> **on** <object>
events <e₁, e₂, ...>
condition <c₁, c₂, ...>
actions <a₁, a₂, ...>
before <t-name₁, ...>
after <t-name₁, ...>
- **events** ∈ {create, insert, delete, modify}
- trigger sequence must be acyclic

Triggers: *example*

Employee

Name	Mgr	Salary
------	-----	--------

```
define trigger AdjustSalary for Employee
  events      create, modify(Salary)
  condition  self.Salary > self.Mgr.Salary
  actions    modify(Employee.Salary, self,
                    self.Mgr.Salary)
end
```

Active DB terminology

- Events

- ◆ primitives for DB state changes
- ◆ internal, external, scheduled, ...

- Conditions

- ◆ predicate or query
- ◆ truth-valued: for query, empty = **false**

- Actions

- ◆ “arbitrary” data manipulation program
- ◆ may include transaction commands

Active DBs applications

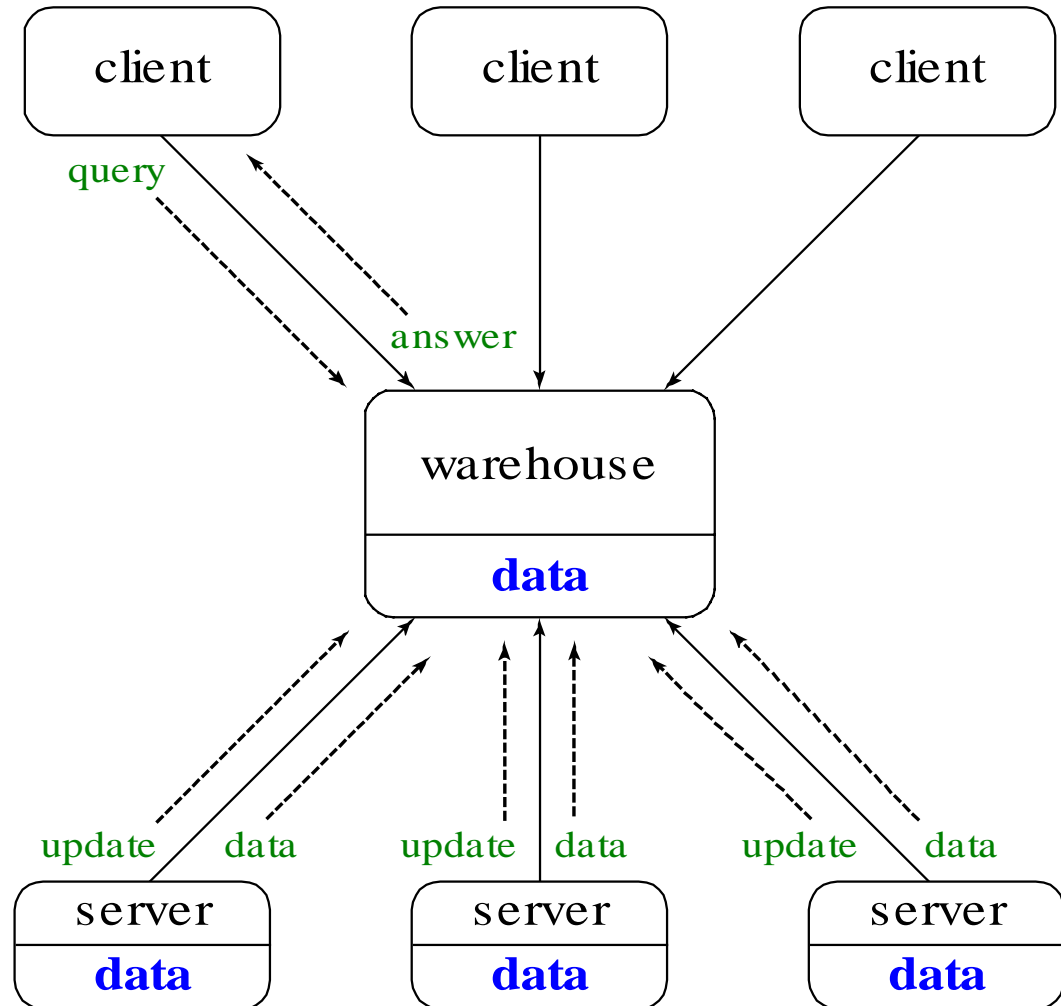
- DB integrity enforcement
- Data warehouse maintenance
- DB replication and synchronization
- Workflow management
- Business rule modeling
- ...

➔ *Any application domain where runtime control over DB state changes is essential*

DB integrity enforcement

- Constraint taxonomy
 - ◆ *static vs. dynamic* constraints
 - ◆ “*built-in*” (e.g. keys, referential integrity)
 - ◆ *generic*
- Generating rules from integrity constraints
 - ◆ *abort* rules: “if constraint violated then refuse update”
 - ◆ *repair* rules: bring DB in new state consistent with rules (non-monotonicity!)

Data warehouse maintenance



Data warehouse maintenance

- *Derived data*: view, stored query result
 - ◆ virtual (interpreted query code)
 - ◆ materialized (compiled, pre-evaluated)
 - refresh
 - incremental

Design and methodology issues for active DBs

- rule set termination
 - ◆ any transaction (triggering rules) terminates in final DB state
- rule set confluence
 - ◆ any transaction terminates in final state independent of rule execution order
- observable determinism
 - ◆ confluence + all output sequences caused by rule actions are the same

(2. Deductive Databases)

- <not covered 2002; to be supplied 2003>

3. Temporal Databases

- <student project 1, to be supplied 2002>

4. Spatial and Multimedia Databases

- <student project 2, to be supplied 2002>

5. Interoperability, ORBs, Mediators, Standardization

- <student project 3, to be supplied 2002>

6. Data Mining

- <student project 4, to be supplied 2002>

7. Data Warehousing

- <student project 5, to be supplied 2002>

8. XML & Web databases

- XML
- Web applications
- XML query languages
 - ◆ XML-QL (AT&T)
 - ◆ XSL
- Research Topics
- NB. Slides derived from and ©Dan Suciu of AT&T Research

What Is XML ?

```
<book>
  <author> Serge Abiteboul </author>
  <author><first-name> Rick </first-name>
    <last-name> Hull </last-name>
  <author> Victor Vianu </author>
  <title> Foundations of Databases </title>
  <year> 1995 </year>
</book>
```

= a data exchange format

XML Types

DTD (Document Type Descriptor):

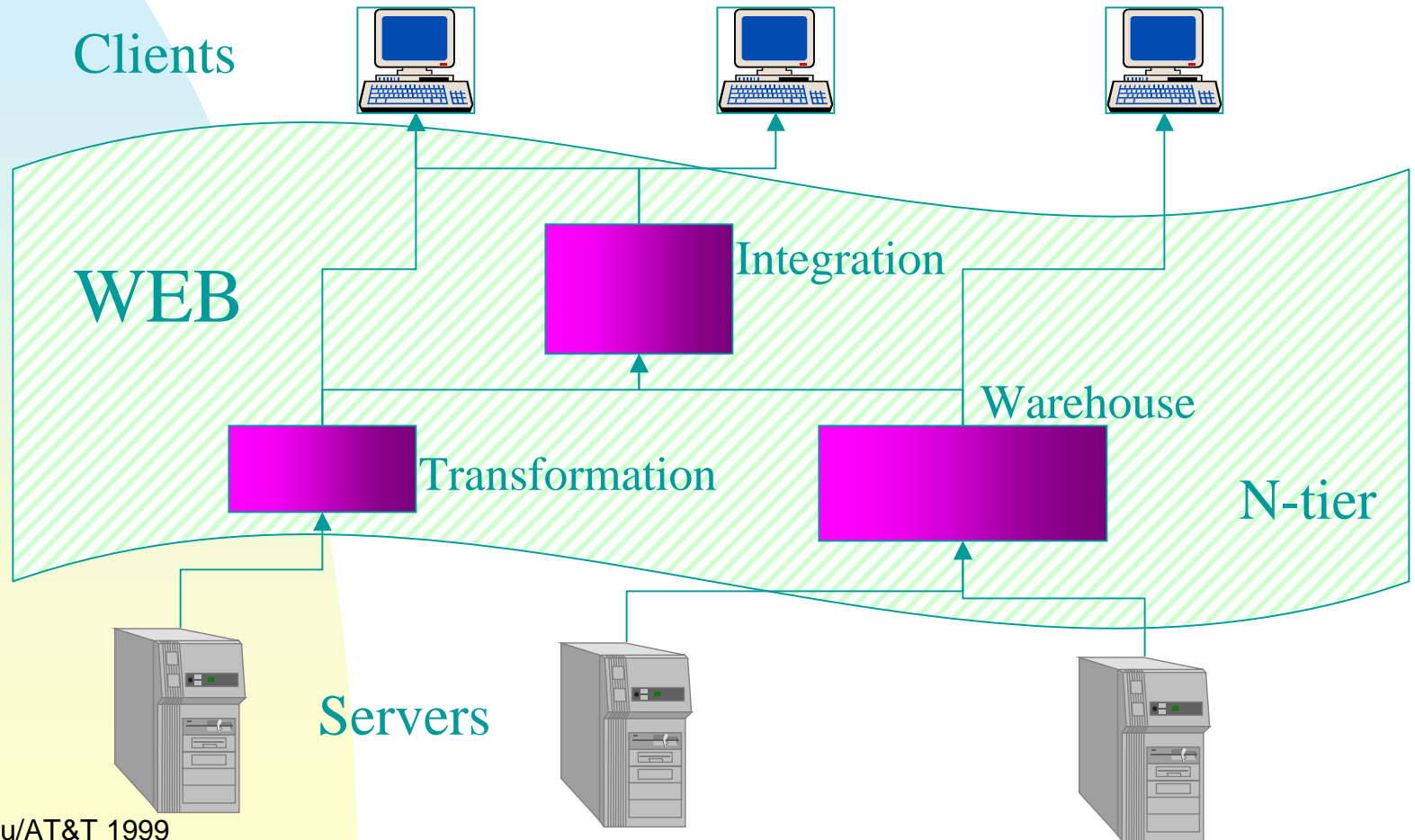
```
<!ELEMENT book (author*,title,year?,publisher?)>  
<!ELEMENT author (#PCDATA|first-name|last-name)*>  
<!ELEMENT title #PCDATA>  
<!ELEMENT year #PCDATA>
```

Better XML types are being considered

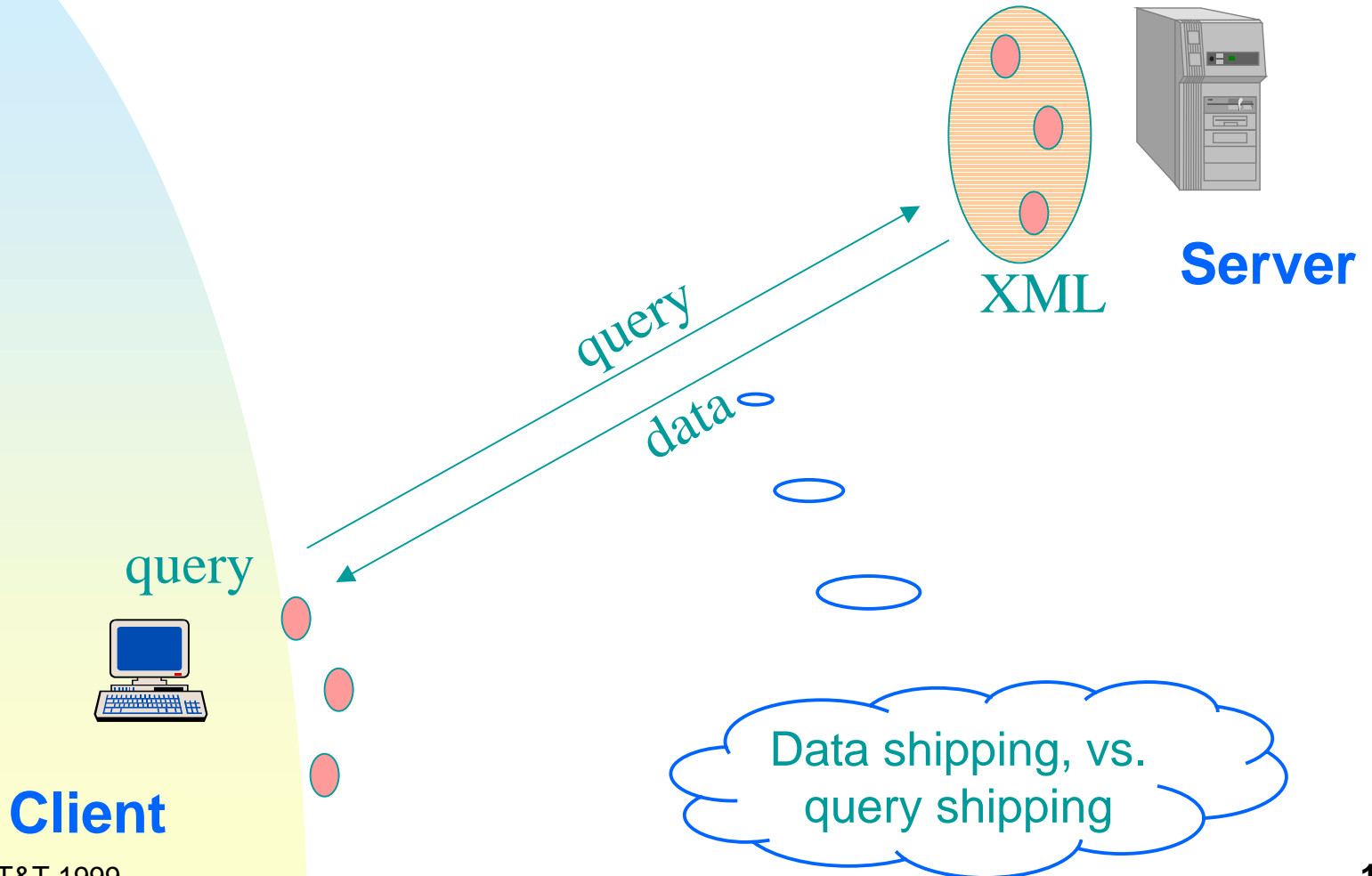
Web Applications

- Today's enterprise DB systems: 3-tier, typed
- Web applications: N-tier, untyped:
 - ◆ Simplicity wins
 - ◆ XML creates “common illusion”
 - ◆ Multiple data sources
 - ◆ Cross platforms, cross enterprises
 - ◆ Standards
- Companies (amazon.com) under pressure from portals (yahoo.com) to support queries

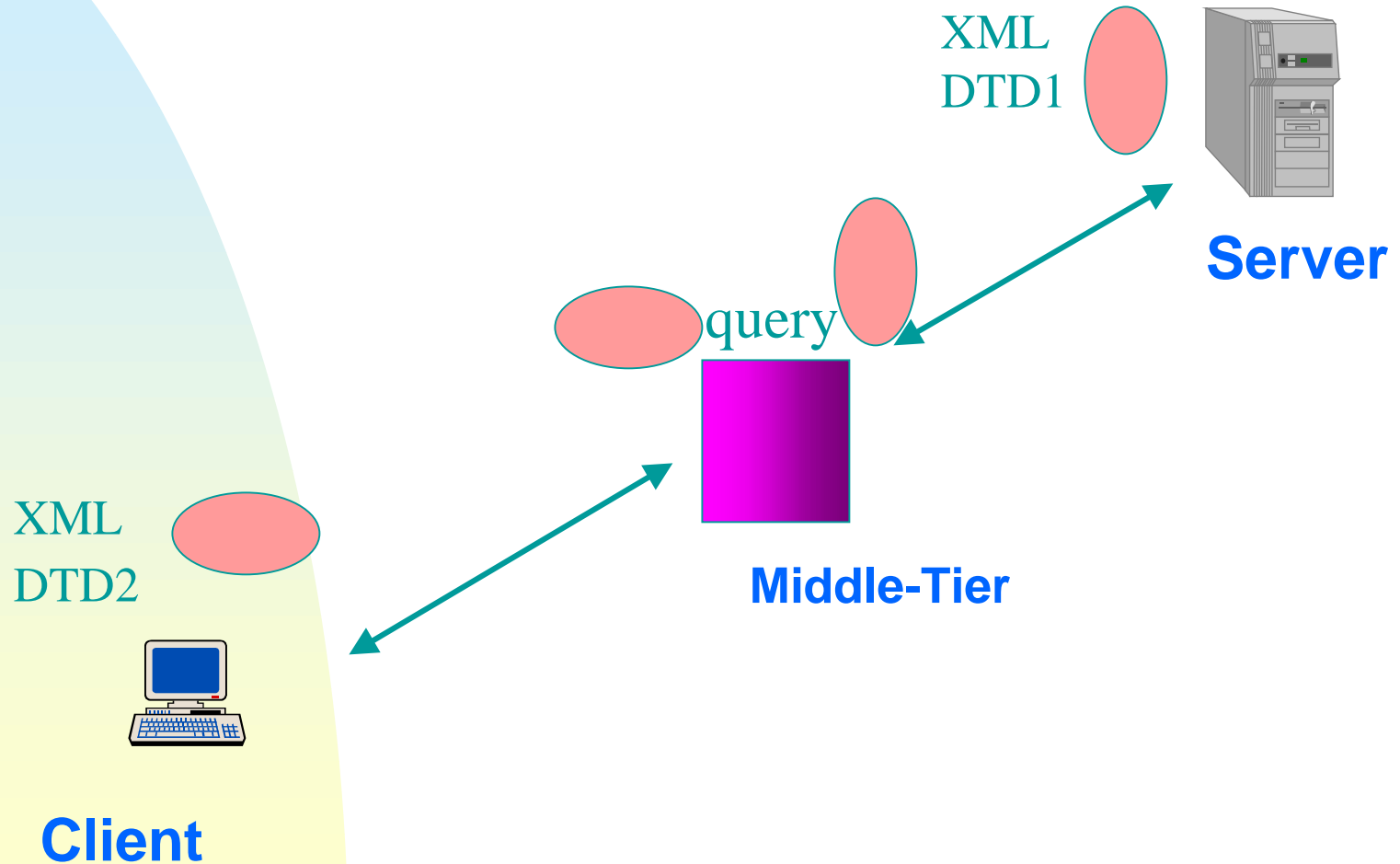
Web Applications Architecture



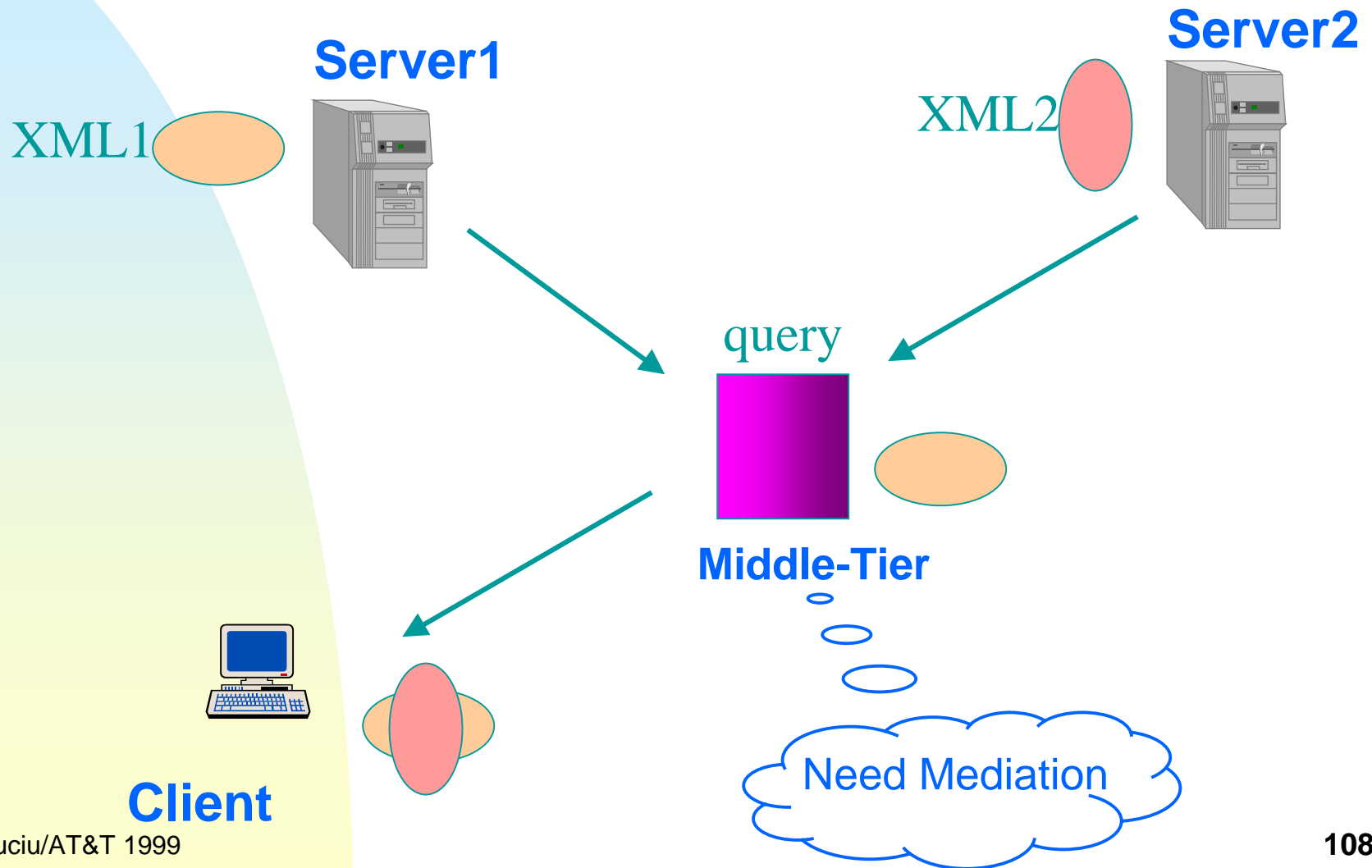
Simple Web Application 1: *Data Extraction*



Simple Web-Application 2: *Data Transformation*



Simple Web-Application 3: *Data Integration*



XML Query Language

Desiderata from David Maier's list:

- XML output
- Server-side processing
- Query operations (“Relational complete”)
- No schema required
- But exploit schema (DTD) when available
- Precise semantics

What Is an “XML Query Processor” ?

- Anecdote from W3C Workshop:
 - ◆ Tim Bray: a query language should admit a “simple” processor
- Processor can be:
 - ◆ lightweight ‘grep’-like, or
 - ◆ heavyweight, cost-based optimizer, or
 - ◆ translate XML query to SQL, or
 - ◆ translate XML query to XML query

Note: A processor is different things in different applications...

A Query Language for XML: XML-QL

- Designed in AT&T Labs (w. Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy)
- Implementation on top of Strudel (Alin Deutsch, Mary Fernandez)
- Prototype:
<http://www.research.att.com/sw/tools/xmlql>

XML-QL: *Pattern Matching and Selections*

```
WHERE <book>
    <publisher>Springer</publisher>
    <author> $a </author>
    <year> $y </year>
</book> IN "www.a.b.c/bib.xml" ,

1991 <= $y AND $y <= 1994

CONSTRUCT $a
```

XML-QL

Construction of New XML Data

```
WHERE <book> <publisher> Springer </>
      <title> $t </>
      <author> $a </>
      </> IN "www.a.b.c/bib.xml"
CONSTRUCT <result> <author> $a </>
          <title> $t </>
          </>
```

XML-QL: Joins

```
WHERE <article> <author> $a </>  
      </article> ELEMENT_AS $p  
      IN "www.a.b.c/bib.xml",
```

```
<book loc = $x> <author> $a </>  
</book>  
      IN "www.a.b.c/bib.xml",  
      $x = "Main Library"
```

```
CONSTRUCT $p
```

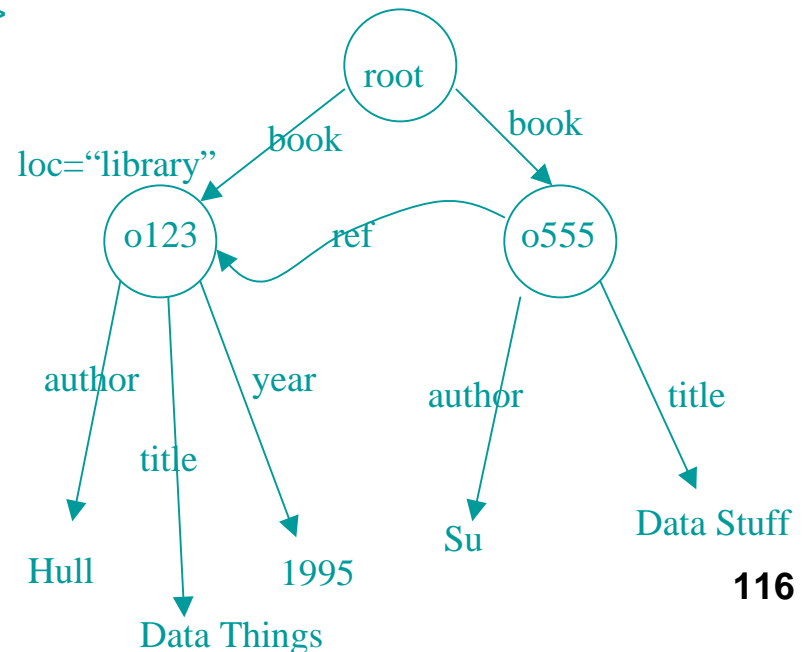
XML-QL Data Model

- Directed, labeled graph
- Tags on edges (not nodes)
 - ◆ here we are against the tide
- Two models: ordered and unordered

XML Data Model: Example

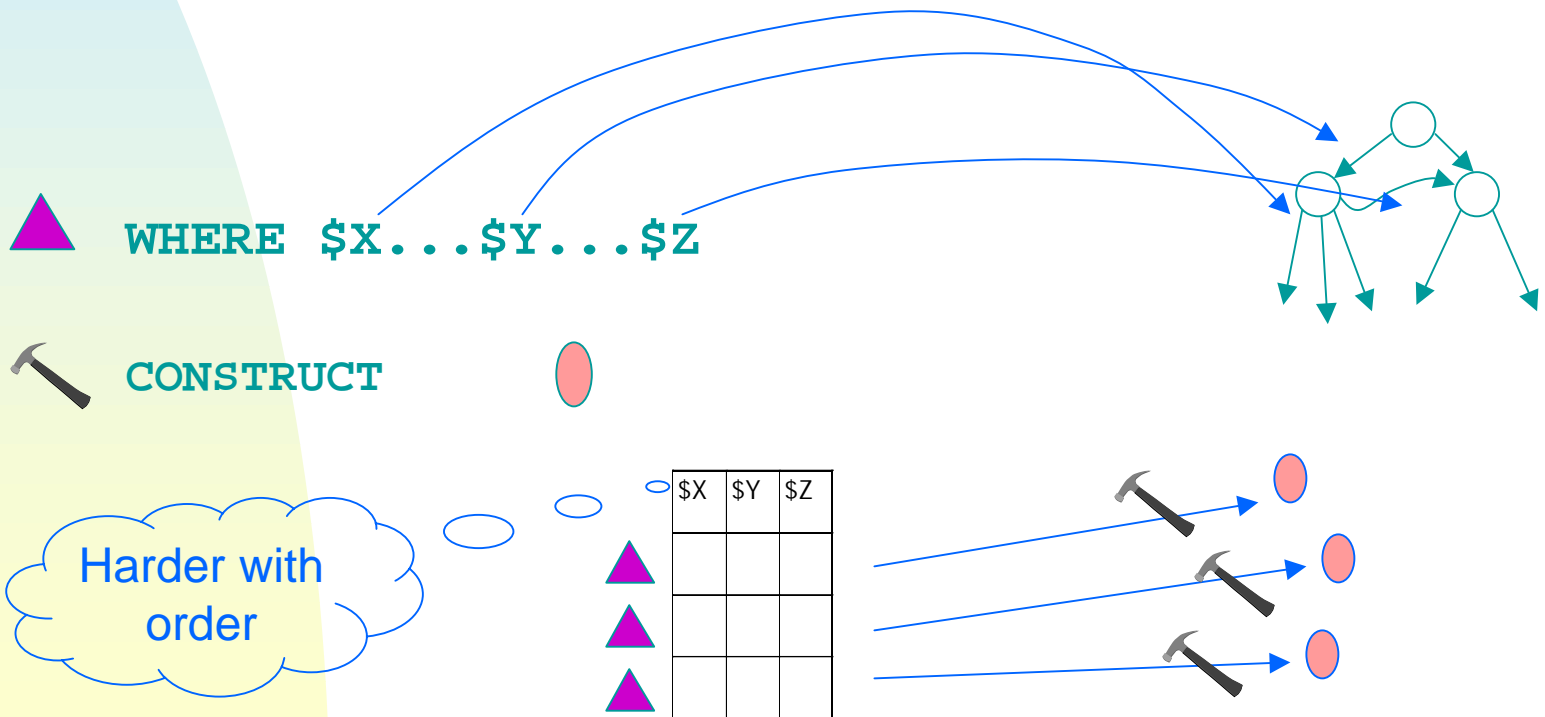
```
<book id = "o123", loc = "library">  
  <author> Hull </author>  
  <title> Data... </title>  
  <year> 1995 </year>  
</book>
```

```
<article id="o555", ref="o123">  
  <author> Su </author>  
  <title> ... </title>  
</article>
```



XML-QL Semantics

- *Step 1: find all substitutions*
- *Step 2: construct XML result*



XML-QL Advanced Features:

1. *Tag Variables*

```
WHERE <$p> <title> $t </title>
      <$e> Smith </>
      </> IN "www.a.b.c/bib.xml",
      $e IN {"author", "editor"}
CONSTRUCT <$p> <title> $t </title>
          <$e> Smith </>
          </>
```

XML-QL Advanced Features:

2. Regular Path Expressions

```
WHERE <(part | subpart)*>
      <name> $r </>
      <brand> Ford </>
      </> IN "www.a.b.c/bib.xml"
CONSTRUCT <result> $r </>
```

XML-QL Applications:

1. Data Transformations

```
WHERE <$pers> <author> $a </>
      <title> $t </>
      </>   IN      "www.a.b.c/bib.xml",
CONSTRUCT <$pers id = PersonID($a)>
          <name> $a </>
          <publication_title> $t </>
          </>
```



XML-QL Applications:

2. Data Integration

```
{ WHERE <person> <name> $n </>
      <ssn> $ssn </>
      </> IN "www.a.b.c/data.xml"
  CONSTRUCT <result id = SSNID($ssn)>
            <name> $n </>
            </> }
{ WHERE <taxpayer> <ssn> $ssn </>
      <income> $i </>
      </> IN "www.irs.gov/tax.xml"
  CONSTRUCT <result id = SSNID($ssn)>
            <income> $i </>
            </> }
```

XML-QL Order:

1. Query the Input Order

```
WHERE <paper>
    <title> $t </title>
    <author[$i]> Abiteboul </author>
    <author[$j]> Vianu </author>
</paper> IN "www.a.b.c/my-bib",
    $j < $i
CONSTRUCT <result> $t </result>
```

XML-QL Order:

2. Control Output Order

```
WHERE <book> <title> $t </title>  
        <year> $y </year>  
        <publisher> $p </publisher>  
        </book>  
ORDERED-BY $p, $y  
CONSTRUCT <result> $t </result>
```

XML-QL Loose Ends

- XML links in the data model
- “Locators”: return pointer, not value
- Aggregates (sum, min, max, etc)
- Non-monotone operators (not-member)
- Functions -- facilitate code reuse

XML-QL Summary

- XML output
- Relational complete
- No schema required
- Precise semantics
- Suitable for:
 - ◆ XML extraction
 - ◆ XML transformation
 - ◆ XML integration

XSL (= XML Stylesheet Language)

- Designed mainly for *formatting*
- *Can be used for querying/transformations*
- Three (conflicting) sources:
 - ◆ W3C's XSL specification
 - ◆ Microsoft's XSL specification
 - ◆ Microsoft's XSL implementation in IE 5.0

XSL

- XSL program = set of *template rules*
- template rule = *pattern + template*
 - ◆ Corresponds to **where + construct**
- No variables, no joins
- *Syntax: directly in XML*

XSL Template Rules

```
<xsl:template match = "book">  
  <result> <xsl:apply-templates/> </result>  
</xsl:template>
```

```
<xsl:template match = "title">  
  <xsl:value-of/>  
</xsl:template>
```

```
WHERE <book> $x </>  
CONSTRUCT  
  <result>  
    WHERE <title> $t </>  
    CONSTRUCT $t  
  </result>
```

XSL patterns

`book` matches any “book” tag
`book/title` matches the “title” tag in “book”
`book/*/first-name` ...any tag
`//first-name` ...any path
`book[author/first-name]/title` conditions
`//book[author,date/year]//title[subtitle]`

Equivalent to tree patterns with one variable

XSL summary

- XML output
- Not relationally complete (such as Datalog)
- No schema required
- Complex semantics, example:
 - ◆ XSL document (no examples) = 448k
 - ◆ XML-QL document (plenty examples) = 75k
- Suitability:
 - ◆ querying, formatting = yes
 - ◆ transformation = maybe
 - ◆ integration = no

Note on XSL in IE

- Microsoft ships XSL in IE5.0.
 - ◆ Anecdote: 10,000,000 edge-traversals/second

XML Query Languages: What's next ?

- Common agreement on the usefulness of a *unique* language
- Little agreement beyond that
 - ◆ culture clash (document v.s. database)
- W3C is organizing a WG on an XML query language

XML Research Topics

- Compression
- Type checking/inference
- Storage

Research Topic: Compression

- Separate element compression ?

```
<object> <height> 55 </height>  
          <width> 700070007 </width>  
</object>  
<object> <width> 700070003 </width>  
          <height> 54 </height>  
</object>
```

- Distinct algorithms for ordered/unordered
- May exploit DTD or other schema info

Research Topic: Compression

- Anecdotes from Microsoft:
 - ◆ Adam Bosworth observed a 50% size reduction when moving a 1MB SQLServer database to XML
 - ◆ Jim Gray observed similar reduction for a 1TB database (TeraServer)
 - ◆ Why ?

Is this a statement about XML or about SQL Server ?

Research Topic: Type Inference

- No “most general output DTD”:

Input DTD: `<!ELEMENT Document Book*>`

Query: “Create an `<A>` for each `<Book>`;
then a `` for each `<Book>`;
then a `<C>` for each `<Book>`”

Input: `<Book />...<Book />`

Output: `<A />...<A /> ... <C />...<C />`
not regular, not even context-free !

The output of a query does not have a most general type

Research Topic: Storage

- Should exploit data regularities when present
- Use data mining to derive relational storage
- Overlap with compression ?

Conclusions

- XML today: emphasizes simplicity
- Related to many fields:
 - ◆ database
 - ◆ programming languages, object-orientation
 - ◆ data compression
- Exciting research topics (and pressing too !)

