Emerging paradigms...

- ARE:
  - flexible
  - easy to use
    - for naïve users
  - standard
    - will be, anyway...
- ARE NOT:
  - safe
    - “static” safety...
  - computationally complete
    - eg transitive closure
  - efficient
  - cycle-aware

example

```
<people>
    <person age="23">
        <name>Richard</name>
        <address>23 City Road</address>
    </person>
    <person age="34">
        <name>
            <forename>Fabio</forename>
            <surname>Simeoni</surname>
        </name>
    </person>
</people>
```

The problem

- How to introduce the desirable “database” features…
  - … without losing the flexibility inherent in the paradigm?

Step 1: a semantics

- Bizarrely (?!) XML has no semantic model
  - one is emerging...
- basic semantics are easy to assign
  - scalars, records, collections are all based upon a "fact"-based model of edge-labeled graphs
- model is hard to stretch to all of XML
  - strange "committee" decisions have already been made, e.g. attributes, mixed content, ...

Typing the impossible...

- A semantics allows the concept of typing a collection
  - wrt a programming algebra
- Many collections contain significant heterogeneity
  - will result in impossibly complex typings
    - can approximate to same size as collection
  - beyond understanding of programmer

example

People : collection(Person)

Person : union(Person1, Person2)

Person1 : record(age : int, name : Name, address : string)

Person2 : record(age : int, name : Name)

Name : union(string, record(forename, surname : string)

Step 2 : retrospective typing

- Don’t type a whole collection a priori ...
  - as in standard systems
- … describe a view type retrospectively
  - to capture a subset of the original data
  - subset concept based on “fact” interpretation
- gives “programming language”-level views
  - incrementally maintainable over evolving data

example

People : collection(Person)

Person : record(age : int, name : string)

Defining a subset...

- We require to identify a subset of the original data, amenable to “standard” query techniques
  - static knowledge
  - computationally complete
  - potentially efficient
    - structural indexing...
  - amenable to cyclic data
    - through recursive typings
- Subtyping...

Intuitive type system

- T ::= record($l_1 : t_1 \ldots l_n : t_n$) |
  union(T,T) |
  collection(T) |
  int | string

Intuitive type rules

- record(l : t ; SIG) $\subseteq$ record(SIG)
- t $\subseteq$ union(s, t)
- record(l : t ; SIG)

  $\subseteq$

  record($coll_l : collection(t) ; SIG)
- record(l : t, $coll_l : collection(t) ; SIG)

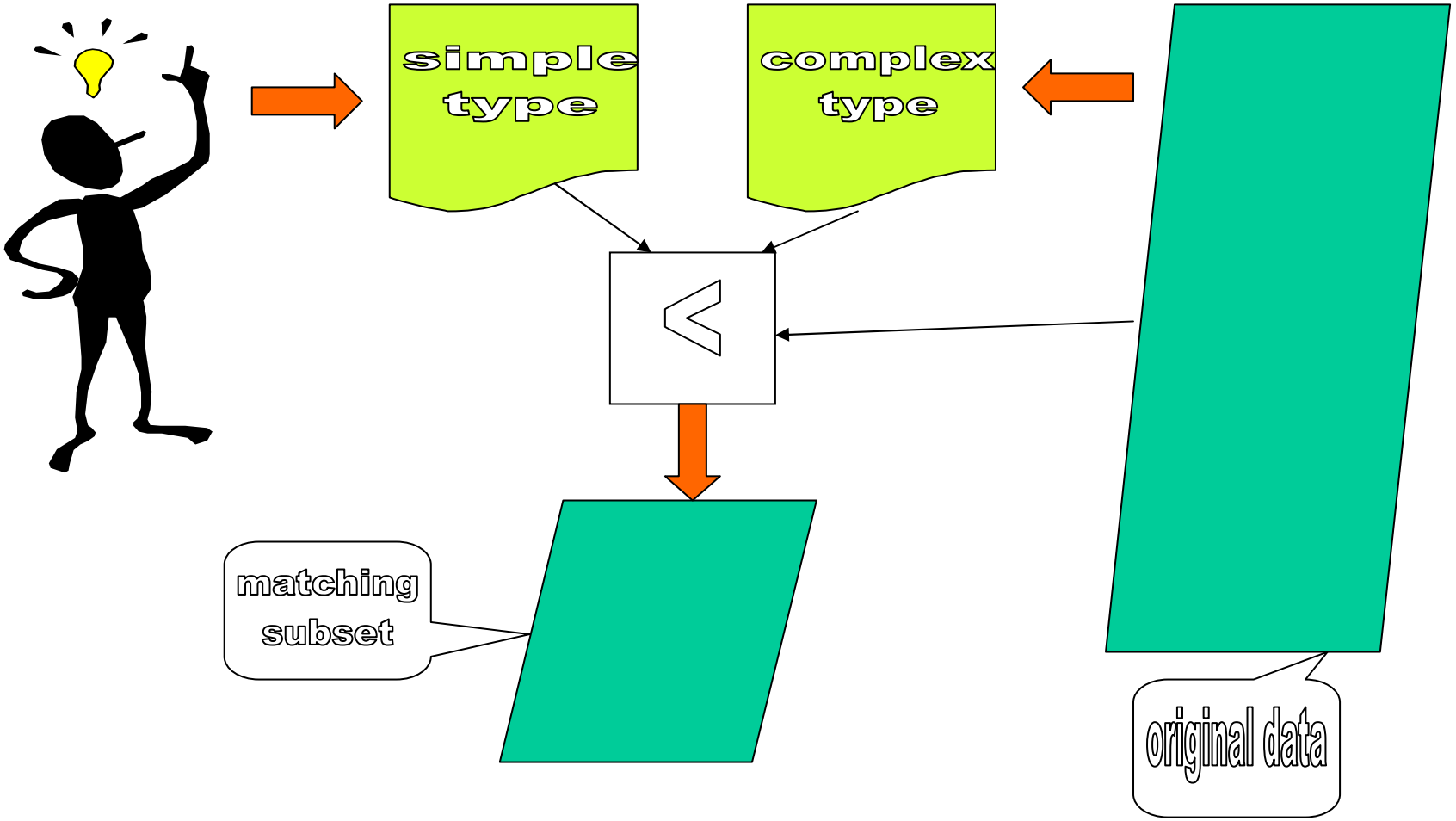
  $\subseteq$

  record($coll_l : collection(t) ; SIG)

Quantification

- Programmer needs to know
  - how much relevant data is included
  - how much relevant data is not included
- a difficult issue, but
  - we create a structural index into the data
  - various mechanisms can be used to provide feedback
    - **before** query execution time...

Simple type views



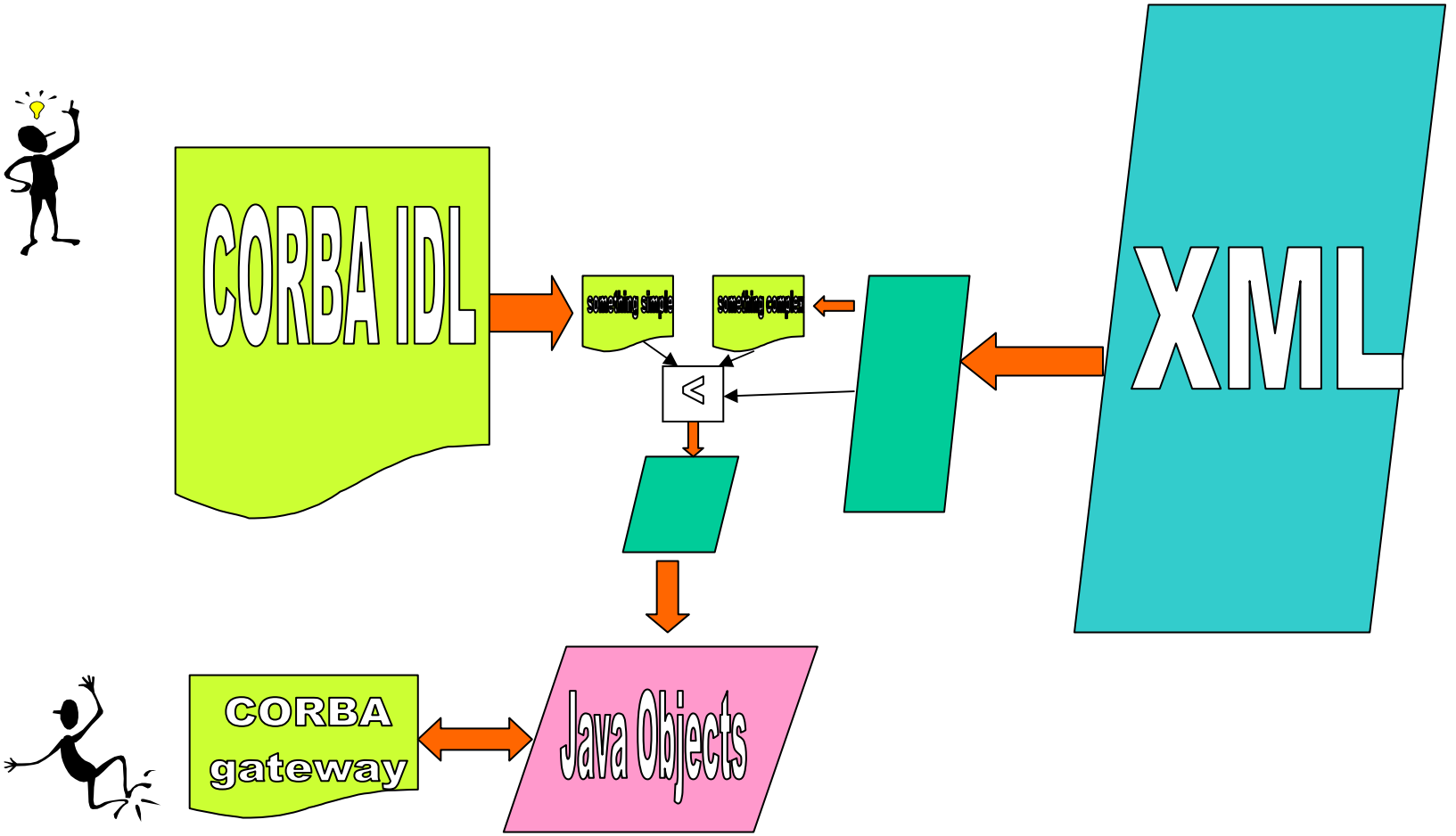
Formalism...

- radically new typing domain - require proof of soundness
- problem - only interesting when including cycles
- standard set theory doesn't support it!
  - long-standing problem in type systems….

Prototype

- 1st prototype - proprietary type systems with OEM
- 2nd prototype - CORBA onto XML
  - working, demonstrable
  - doesn’t handle cycles (yet)
  - based on intuitive typing model...

CORBA/XML gateway



Example: Shakespeare

- entire works of Shakespeare are available on the Web as XML
- want to know how many major speeches Juliet makes
- steps:
  - 1. Define a suitable data type likely to work
  - 2. Attempt to impose this on the file
  - 3. If successful, write a Java program over the new interface

interface description

- struct speech(string speaker ;
  sequence< string > lines)
  typedef sequence< speech > play;

interface creation

- X = createInterface(“RomeoAndJuliet.xml”, “play.idl”)

program code

- let RandJ = X.getData();

```
for( i = 0 ; i < X.max ; i++ )
{
    if( RandJ.i.speaker = “Juliet” &&
            RandJ.i.lines >= 5 )
    speeches++ ;
}
```