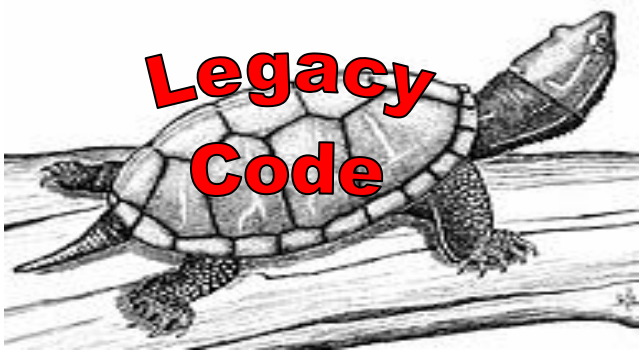
When less is more

G. Ramalingam
Microsoft Research India

The Dark Side of Programming

- Common scenario
  - huge existing legacy code base
  - building on top of existing code
  - transforming existing code
  - integrating legacy systems

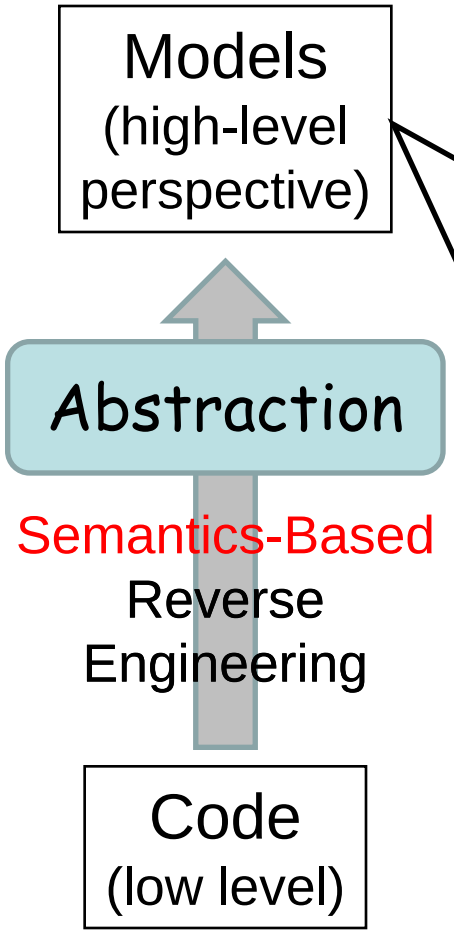


- Legacy code can be surprisingly hard to work with
  - lack of documentation and understanding of existing code
- Need tools to help understand (legacy) code

Towards program-understanding

- Stumbling blocks
  - large and complex code-base
  - low-level code
  - implementation details
  - tangling of multiple concerns
  - scattering (of a logical aspect all over code)
- Understanding requires models that
  - are high level
  - focus on some desired aspect
    - eliminate irrelevant details

Reverse Engineering: From Code To Models



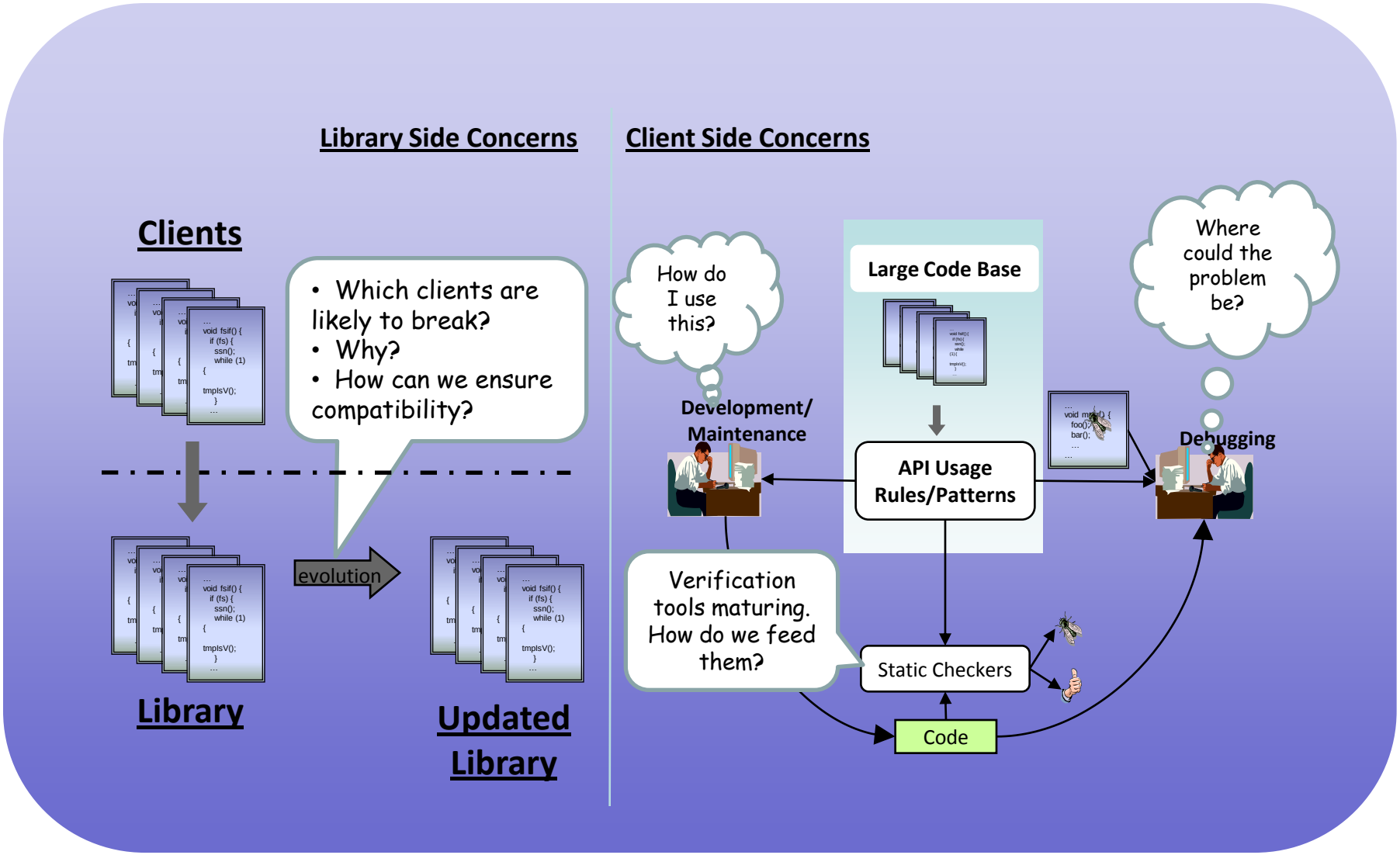
- foundation for understanding
- *and* transforming systems
  - maintain
  - enhance
  - redevelop
- for use by people *as well as other analyses/tools*
- use not restricted to *old* code

1. Mining Library Usage Rules

(Joint work with
Kanika Nema, Kapil Vaswani, Venkatesh-Prasad,
Microsoft Research India)

- Libraries typically have usage constraints
  - data preconditions
    - input parameter p must be non-null
    - parameter x must point to array of size > y
  - temporal constraints
    - call m1(p) before calling m2(p)
    - do not call m3(p) after calling m4(p)
- Goal: Recover (part of) API contract from large code base that uses the API

Abstracting Client-Library Interaction: What & Why



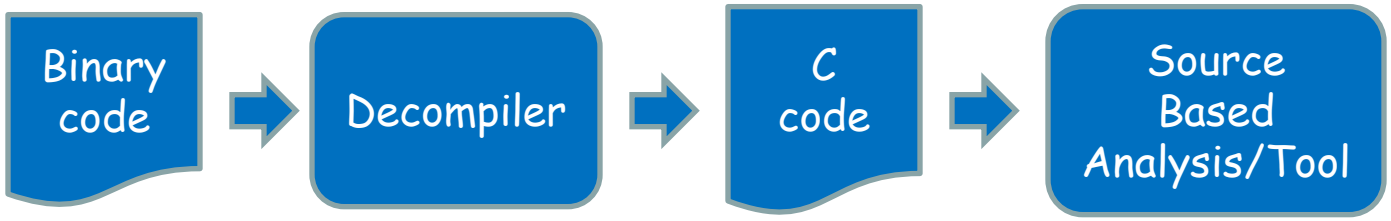
2.Static Analysis of Binaries

(Joint work with:
Pushkar Tripathi, V.L. Subrahmanyam, Venkatesh-Prasad,
Microsoft Research India)

- Many useful analyses and tools exist
  - e.g., Static Driver Verifier (SLAM)
- Most target source languages, such as C
- But sometimes only binaries are available
  - third party code (drivers)
  - potentially malicious code

The Problem

- Decompile binary code to C …
- … permitting reuse of source-code based analyses and tools



- Naïve transliteration will not be very effective
  - Existing analyses exploit higher level abstractions, such as variables and types
- Problem: Inferring *variables* and *types*

Variable Identification: Issues

source code

```
int i, j, k;
int *p;

i = 0;
p = &j;
*p = 1;
k = i;
```

Verifier

executable

```
mov   DWORD PTR [ebp-8], 0

lea     eax, DWORD PTR [ebp-16]
mov   DWORD PTR [ebp-4], eax

mov   ecx, DWORD PTR [ebp-4]
mov   DWORD PTR [ecx], 1

mov   edx, DWORD PTR [ebp-8]
mov   DWORD PTR [ebp-12], edx
```

- Many existing analyses do not handle arrays or structures well
- Modeling stack frame as single array is *correct*, but can lead to *imprecision* downstream

imprecise

```
dword v[4];
v[2]  = 0;
eax   = &v[0];
v[3]  = eax;
ecx   = v[3];
*ecx = 1;
edx   = v[2];
v[1]  = edx;
```

Verifier

?

Variable Identification: Issues

source code

```
typedef struct {
  int x; int y;
} S;
main() {
  S s;
  S *p;
  p = &s;
  p->x = 3;
  p->y = 5;
}
```

Verifier

executable

```
lea     eax, DWORD PTR [ebp-12]
mov   DWORD PTR [ebp-4], eax

mov   ecx, DWORD PTR [ebp-4]
mov   DWORD PTR [ecx], 3

mov   edx, DWORD PTR [ebp-4]
mov   DWORD PTR [edx+4], 5
```

- Representing single array / structure as a set of variables can lead to *incorrectness*

incorrect

```
int *a;
int b;
int c;
eax   = &c;
a       = eax;
ecx   = a;
*ecx = 3;
edx   = a;
*(edx+1) = 5;
```

Verifier

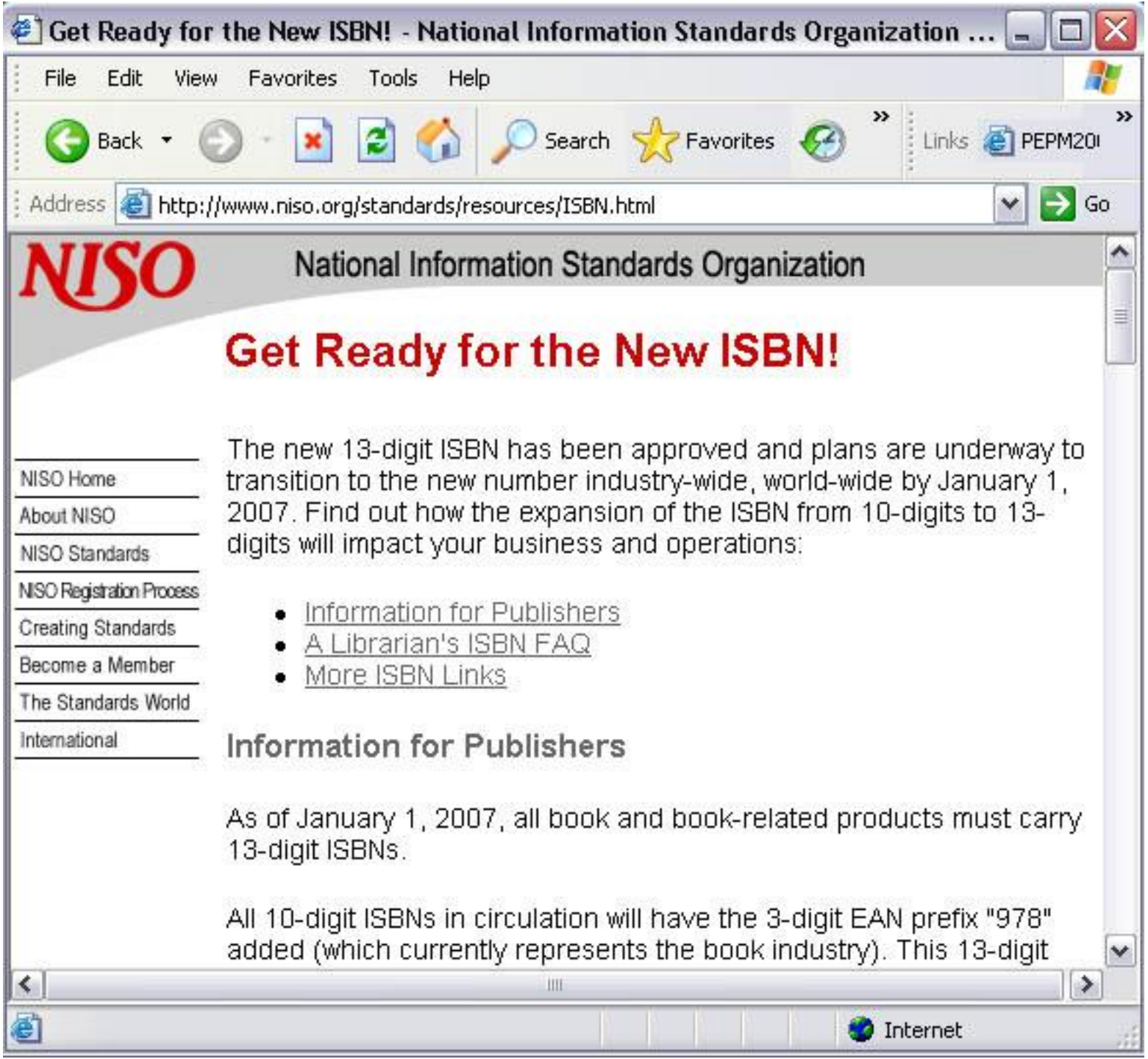
3. Maintenance of Legacy Code

(Joint work with: Raghavan Komondoor, John Field, Satish Chandra, Saurabh Sinha, IBM Research)

- Focus:
  - legacy business applications, written in (weakly-typed) languages such as Cobol
  - assist in common legacy transformation and maintenance tasks

Motivating Example

- Field Expansion
  - a frequently recurring scenario
  - e.g., expand "account number" from 8 to 10 digits
  - Y2K: well-known instance of field expansion


Get Ready for the New ISBN! - National Information Standards Organization ...
File Edit View Favorites Tools Help
Back Search Favorites Links PEPM20
Address http://www.niso.org/standards/resources/ISBN.html
Go
NISO
National Information Standards Organization
Get Ready for the New ISBN!
NISO Home
About NISO
NISO Standards
NISO Registration Process
Creating Standards
Become a Member
The Standards World
International
The new 13-digit ISBN has been approved and plans are underway to transition to the new number industry-wide, world-wide by January 1, 2007. Find out how the expansion of the ISBN from 10-digits to 13-digits will impact your business and operations:
Information for Publishers
A Librarian's ISBN FAQ
More ISBN Links
Information for Publishers
As of January 1, 2007, all book and book-related products must carry 13-digit ISBNs.
All 10-digit ISBNs in circulation will have the 3-digit EAN prefix "978" added (which currently represents the book industry). This 13-digit
Internet

An Example Cobol Program

```
01 input-rec
   05 ir-tr
   05 ir-us
   05 ir-ac
   05 ir-da
01 withdraw
   05 wi-am
   05 wi-da
01 account-r
   05 ar-acc-num pic x(8).
   05 ar-user-name pic x(8).
   05 ar-data pic x(8).
...
MOVE ar-acc-num TO ir-acc-r
IF ir-trans-code = 'w' THEN
   MOVE ir-data TO withdrawal-info
...
```

Cobol

- no user-defined types
- lack of typing constraints
- primitive type: string (sequence of bytes)
- facility to define structured variables
- memory overlay (like *union* in C)
- declarative info: no semantic guarantee

Is this an account-number?

Logical Data Model Inference

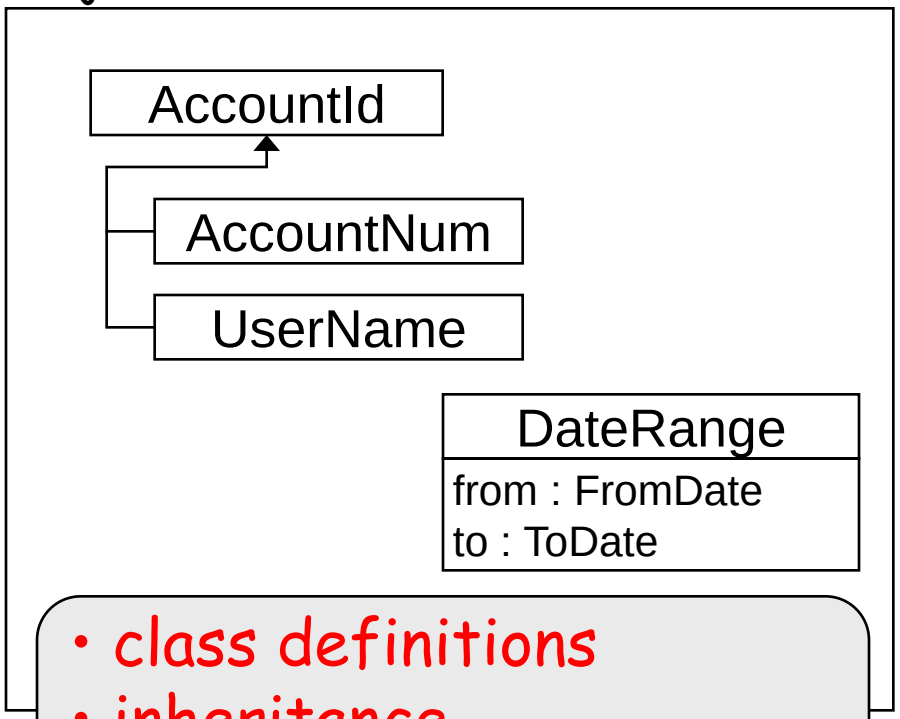
- Problem: Reverse engineer a *logical data model* of a given (legacy) program

The Inference Problem

Source Program

```
...
01 account-rec.
   05 ar-acc-num pic x(8).
   05 ar-user-name pic x(8).
   05 ar-data pic x(5).
...
MOVE ar-acc-num TO ir-acc-num.
...
MOVE ir-data TO ...
...
```

Object-Oriented Data Model



- class definitions
- inheritance
- typed fields

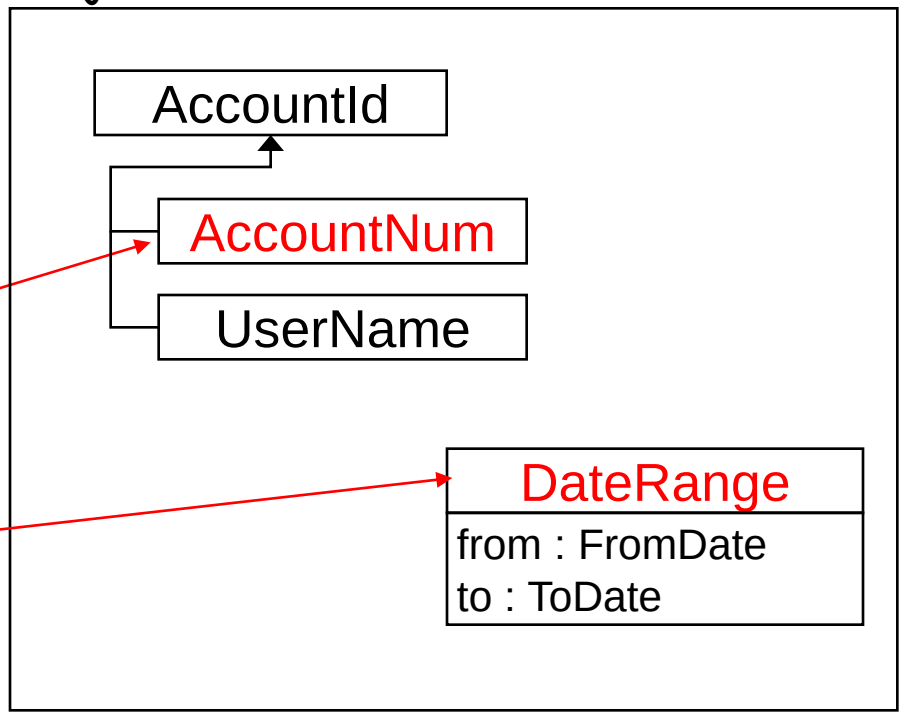
- Infer
  - Object-Oriented Data Model
  - + Links
    - connect a source program to an OO model

Object-Oriented Data Model + Links (LOOM)

Source Program

```
...
01 account-rec.
   05 ar-acc-num pic x(8).
   05 ar-user-name pic x(8).
   05 ar-data pic x(5).
...
MOVE ar-acc-num TO ir-acc-num.
...
MOVE ir-data TO ...
...
```

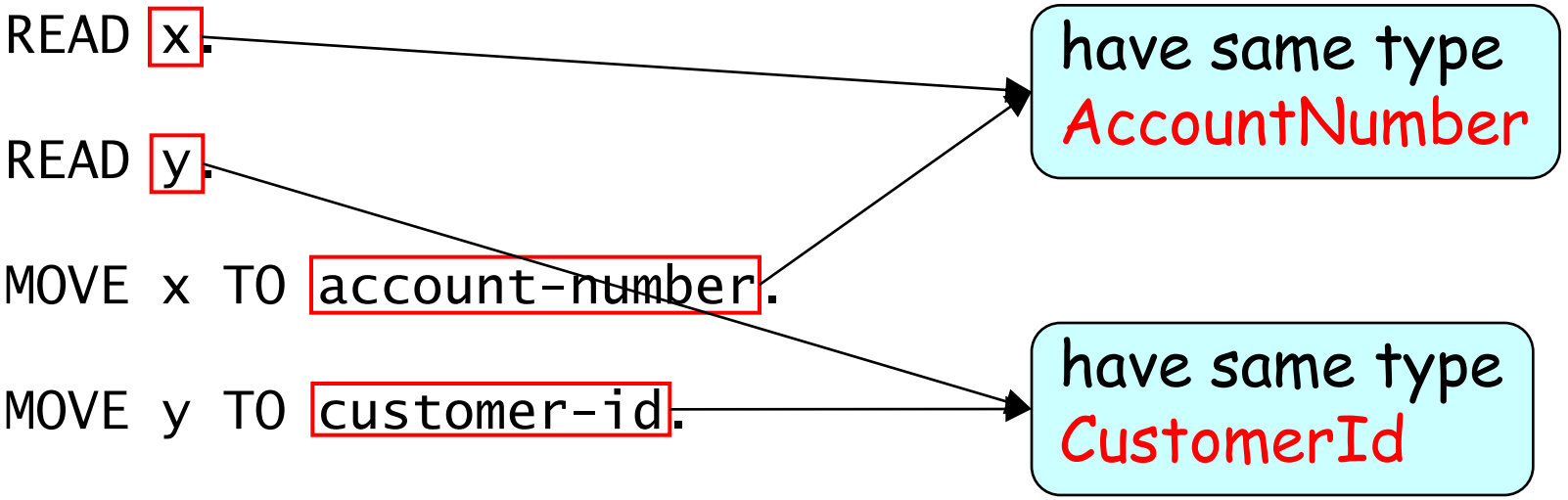
Object-Oriented Data Model



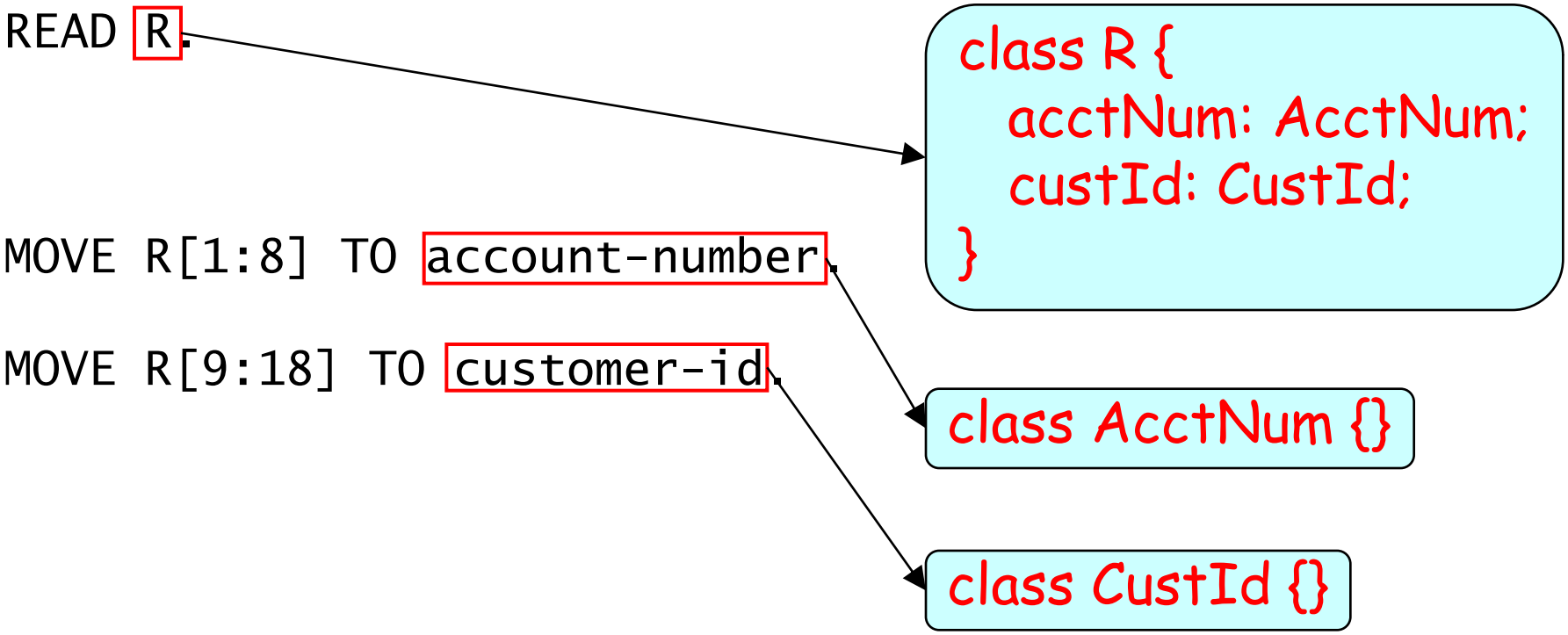
- Infer
  - Object-Oriented Data Model
  - + Links
    - connect a source program to an OO model
    - map variable occurrences to types (& more)

The Inference Algorithm

Basic Idea:
1. Infer Type Equivalence

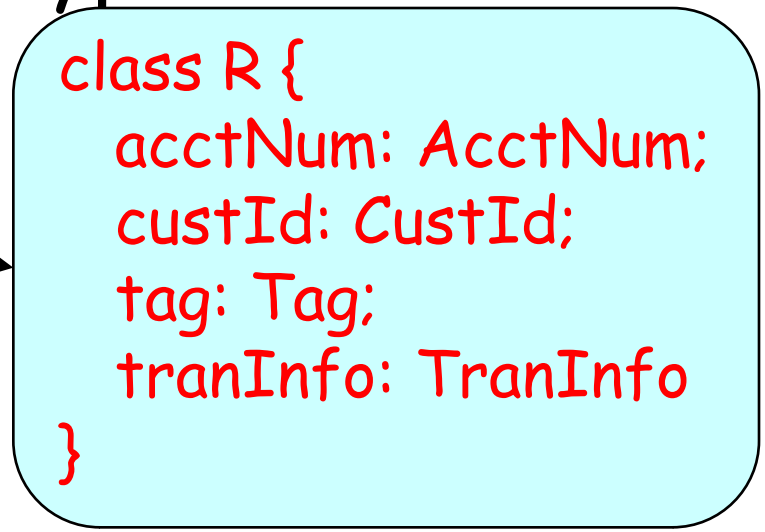


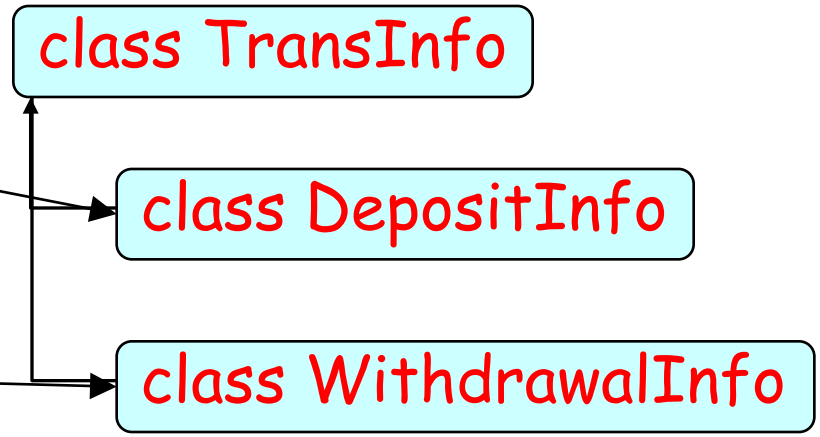
Basic Idea:
2. Infer Record Structure



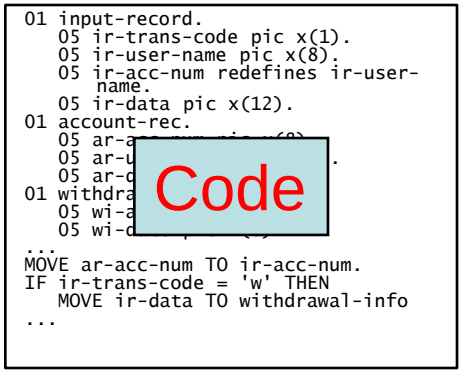
Basic Idea:
3. Infer Subtypes

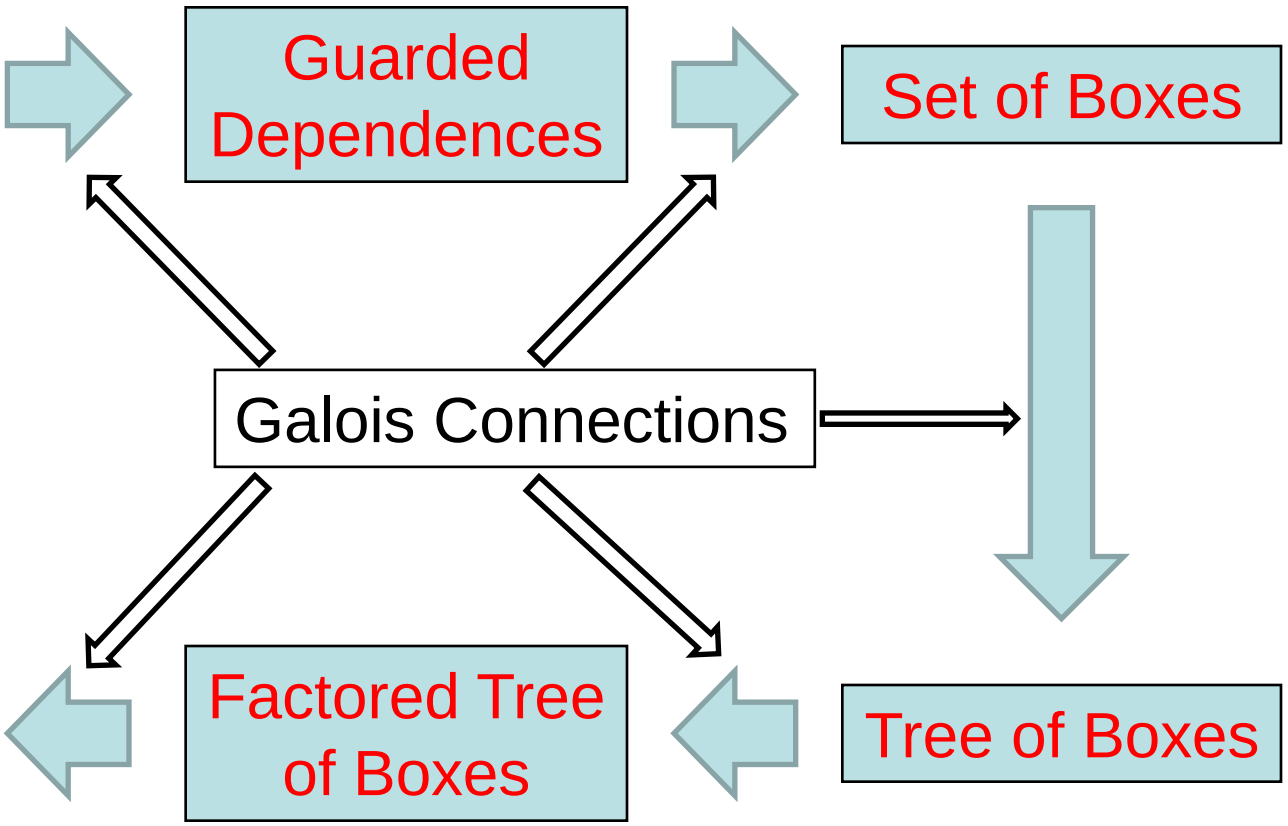
```
READ R.

MOVE R[1:8] TO account-number.

MOVE R[9:18] TO customer-id.

IF (R[19:19] = 'D') THEN

    … process deposit …

ELSE

    … process withdrawal …

END-IF.
```

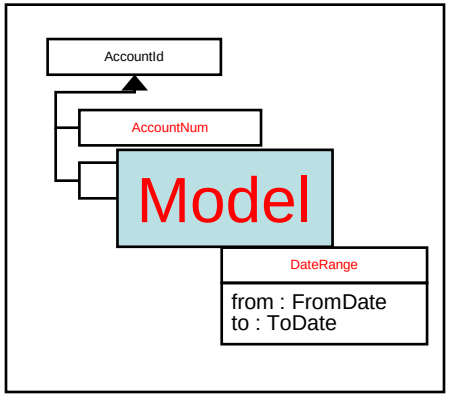




Algorithm Overview



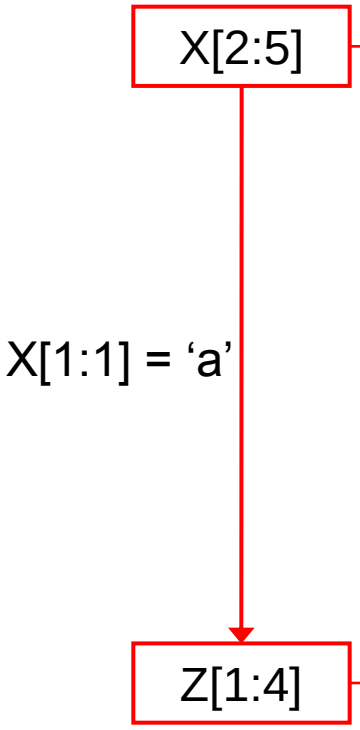




Guarded Dependence Definition

- guard $\Rightarrow$ source $\rightarrow$ target
  - source is a pair range @ program-pt
  - target is similar
  - guard is a predicate on the state at source program-point.
- when guard is true, value at source may reach target (via some sequence of copies)
- example:
  - (x ='a') $\Rightarrow$ y @ pp1 $\rightarrow$ z @ pp2
  - ([1:1] ='a') $\Rightarrow$ [2:5] @ pp1 $\rightarrow$ [6:9] @ pp2

Guarded Dependence Example

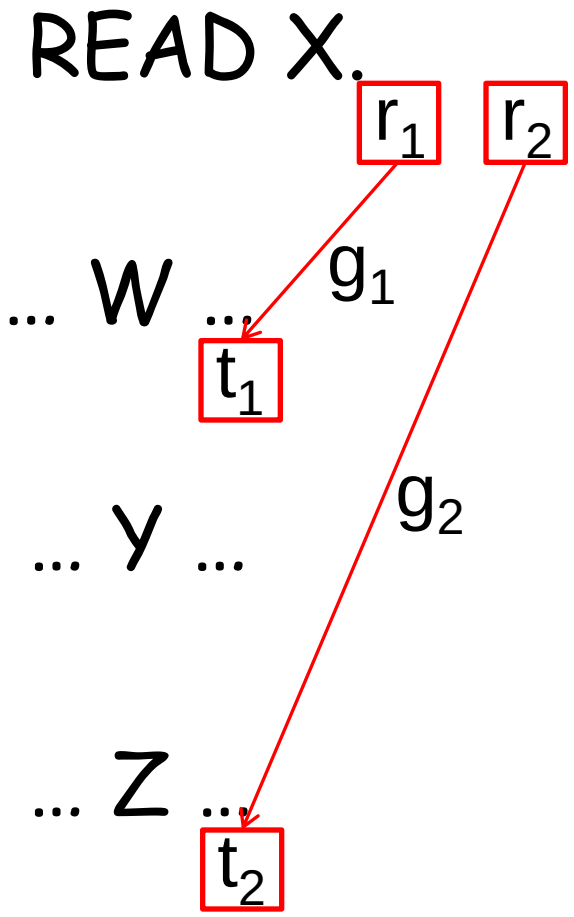



```
READ X.
IF (X[1:1] = ‘a’) THEN
   MOVE X[2:5] TO Y
ELSE
   MOVE X[6:9] TO Y
END-IF.
IF (X[1:1] = ‘a’) THEN
   MOVE Y TO Z
END-IF
```

Guarded Dependence Analysis

- Guarded dependences
  - capture transitive data-dependences
  - capture conditions under which dependence is manifested
- Parametric guarded dependence computation
  - parameterized by abstraction for guards
  - can be computed in polynomial time for simple (common type of) guards

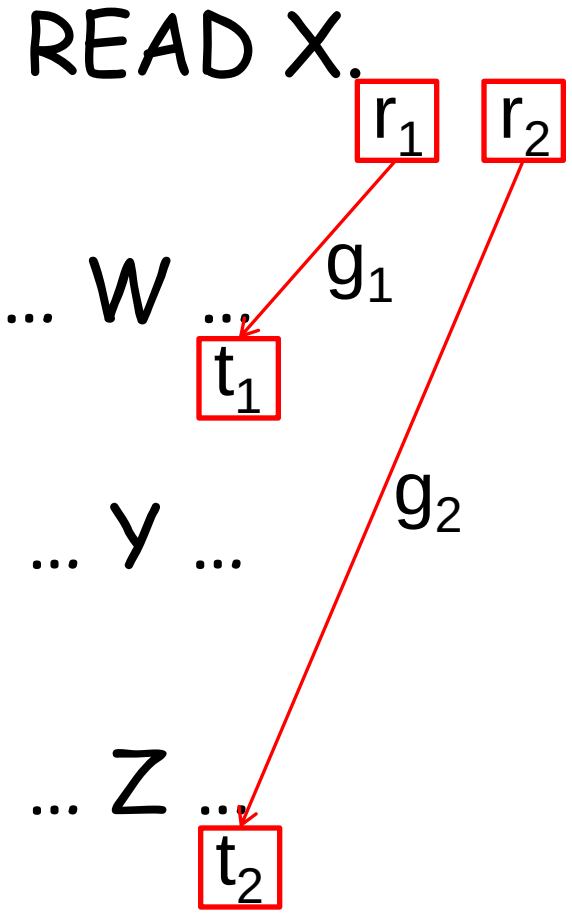
Step 2
From Dependences to Boxes



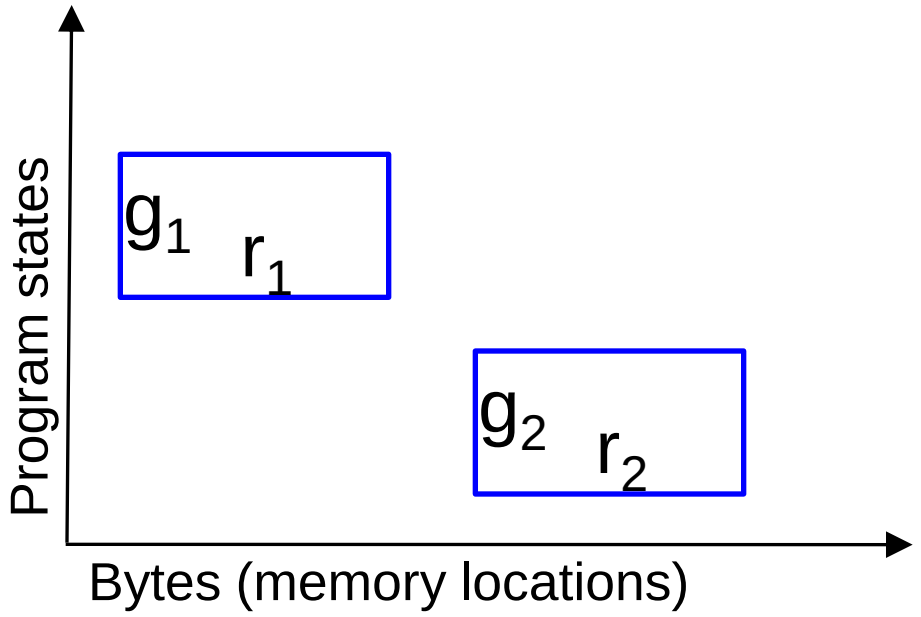
for each data-source:

- consider all guarded dependences originating at that source and ending at data-references
- accumulate the set of all (guard, source-range) pairs from these dependences

Step 2
From Dependences to Boxes

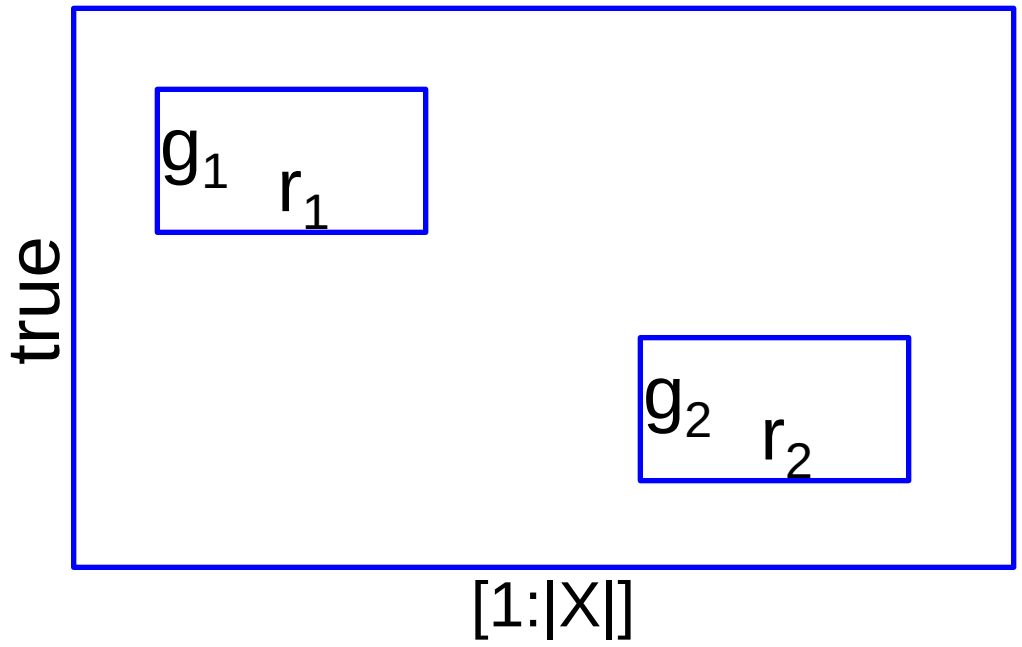


$\{ (g_1, r_1), (g_2, r_2) \}$



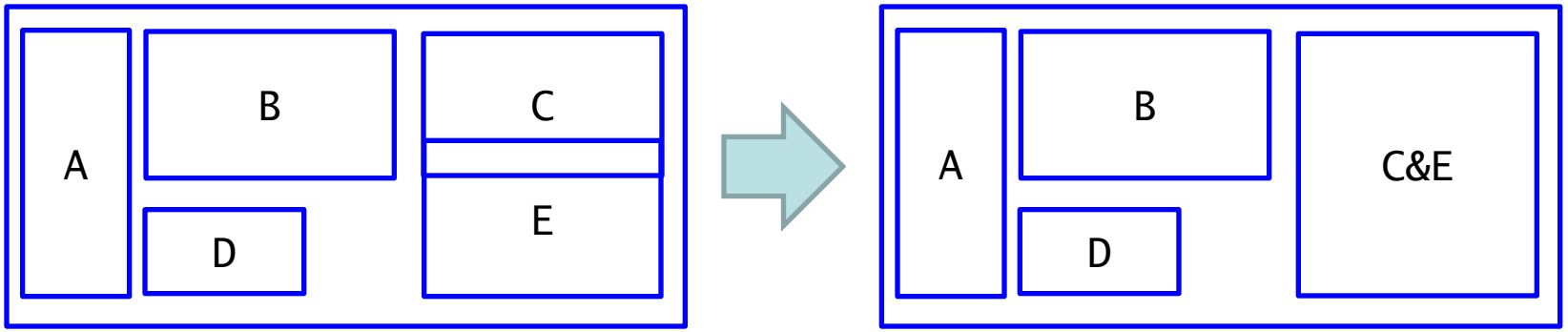
Step 3
From a Set of Boxes …

READ X.

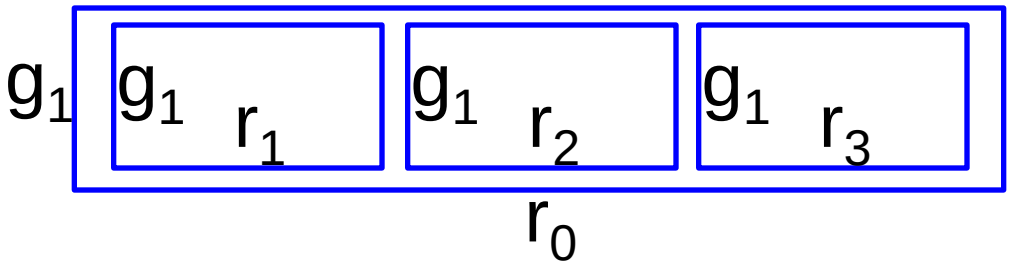


Step 3
… to a Tree of Boxes

- Ensure boxes are properly nested
  - any two boxes must be disjoint, or one must properly contain the other
  - ensure property by widening box vertically as necessary
  - if proper nesting can not be ensured, the algorithm halts (with failure)

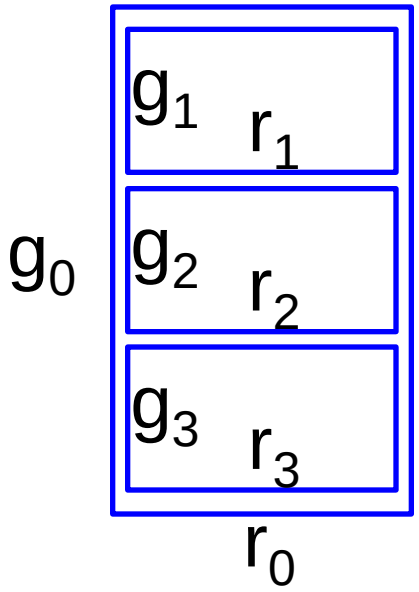


Step 4: Factoring the Tree Special Case 1



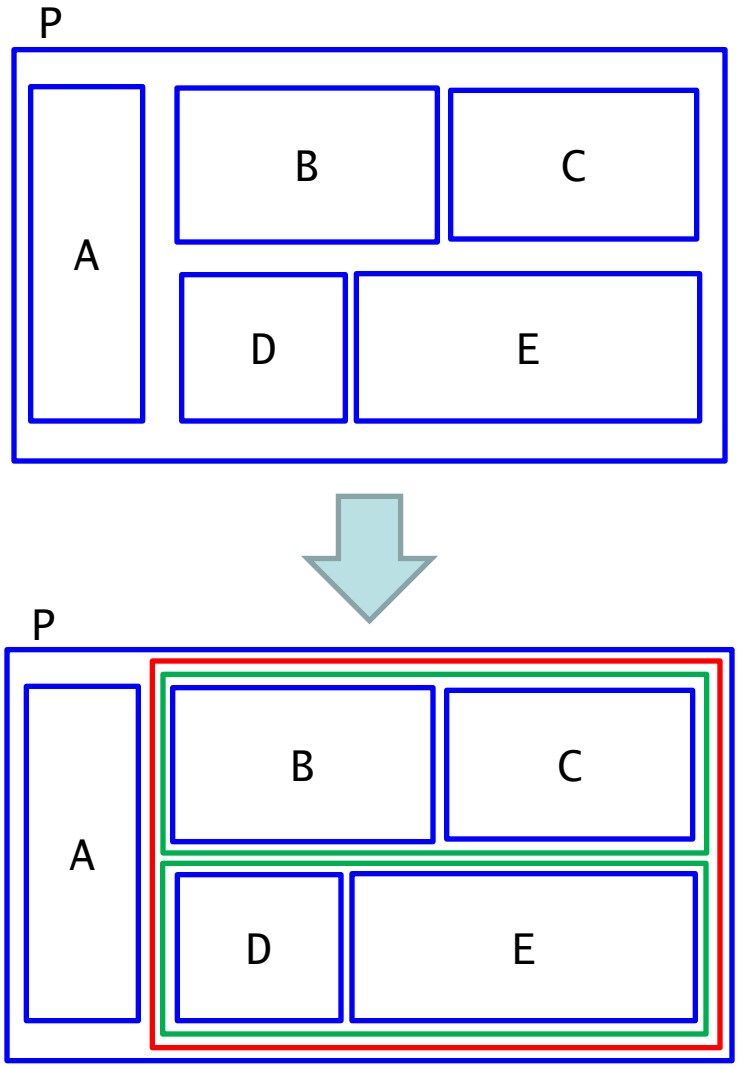
- All children have same guard as parent
  - parent corresponds to a record (class)
  - children correspond to fields
- (Horizontal Partitioning)

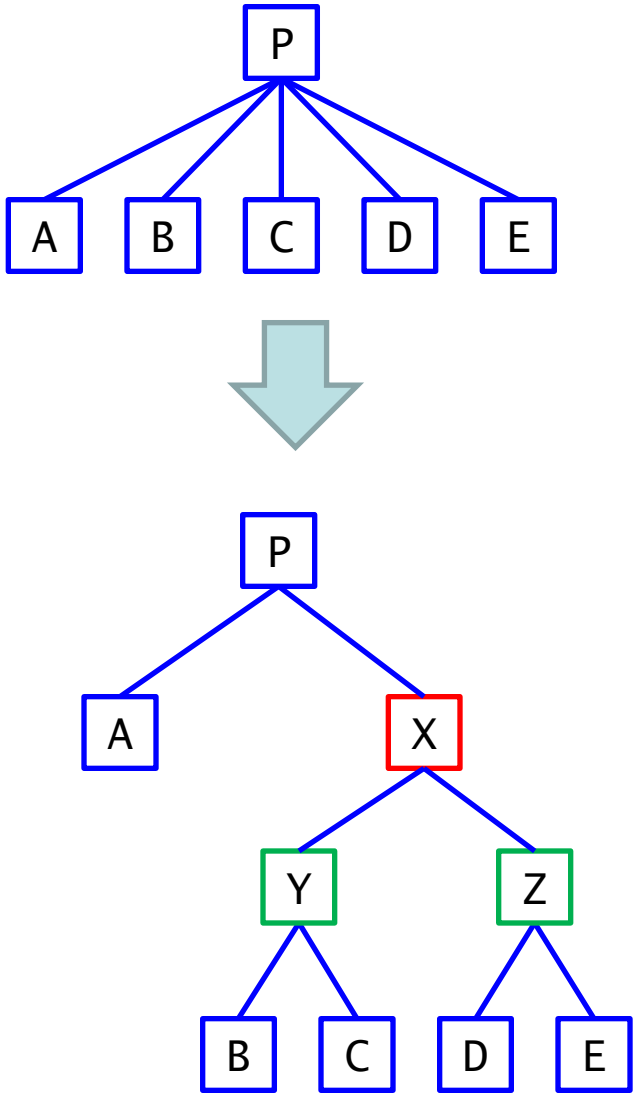
Step 4: Factoring the Tree Special Case 2



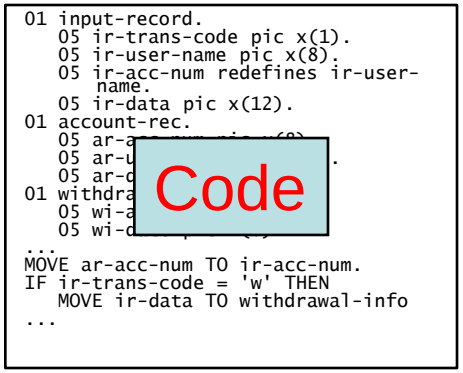
- All children have mutually disjoint predicates
  - parent corresponds to a base class
  - children correspond to derived classes
- (Vertical Partitioning)

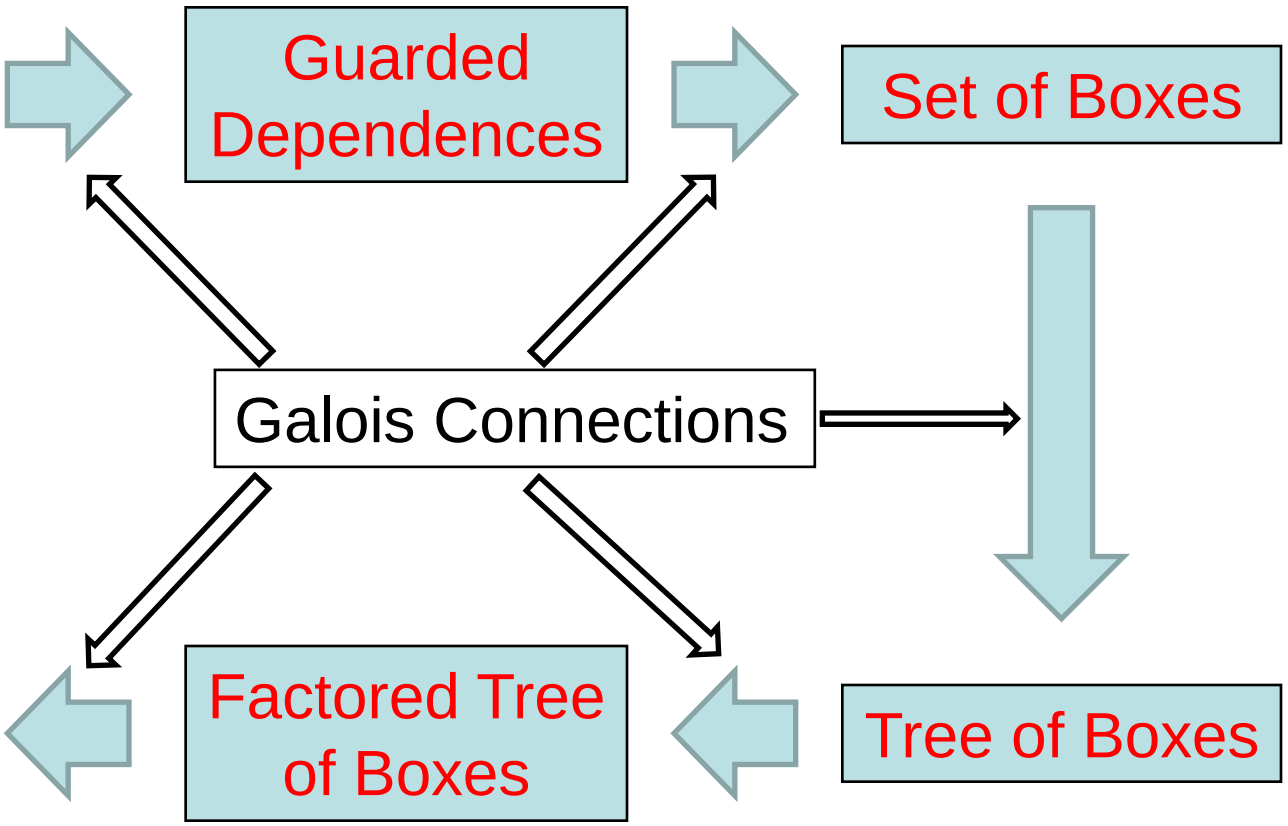
Step 4: Factoring the Tree
The General Case

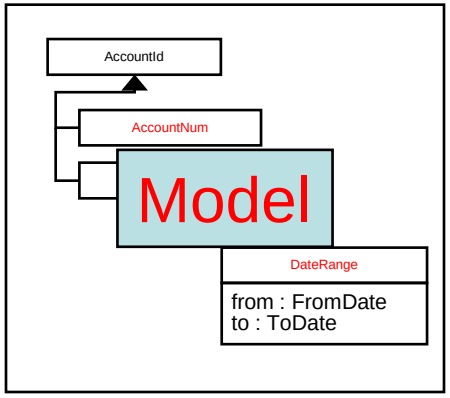




Algorithm Overview







thank you!