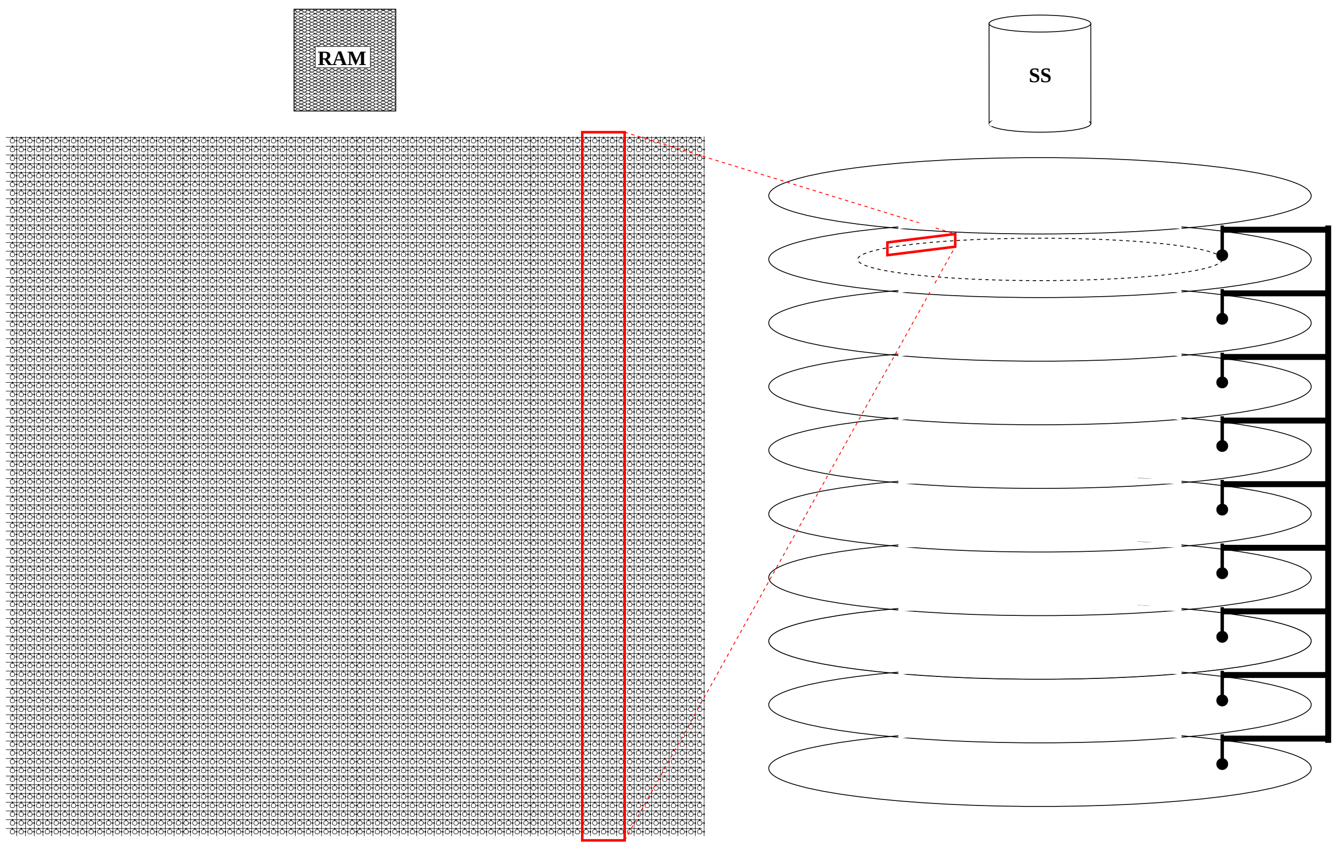
The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

Secondary storage must transfer *large blocks* of data to and from RAM ..




.. because of *latency*, the *relative* cost, of finding it.

CS++: Reinventing Computer Science
(for Secondary Storage)

1. Algorithms & Data Structures ~cs420

- variable multidimensional arrays
- finding all substrings
- variable-resolution maps
- data compression

2. Programming Language ~cs612

- software engineering
- parallel algorithms
- expert systems
- object-orientation
- data mining
- semistructured data
- Internet distributed db

# 1. Algorithms & Data Structures

## Variable-sized arrays

E.g., a Leontieff matrix for the economy:

| | 1. out\in | C | F | H | *earned* |
|---|---|---|---|---|---|
| Charles makes Clothes | C | 2 | 1 | 3 | 6 |
| Fred makes Food | F | 2 | 2 | 3 | 7 |
| Harry makes Houses | H | 2 | 3 | 4 | 9 |
| | *spent* | 6 | 6 | 10 | |

Woops, we forgot Pete, who supplies Power:

| | 2. out\in | C | F | H | P | *earned* |
|---|---|---|---|---|---|---|
| Charles makes Clothes | C | 2 | 1 | 3 | 3 | 9 |
| Fred makes Food | F | 2 | 2 | 3 | 3 | 10 |
| Harry makes Houses | H | 2 | 3 | 4 | 2 | 11 |
| Pete makes Power | P | 1 | 3 | 5 | 1 | 10 |
| | *spent* | 7 | 9 | 15 | 9 | |

Represent these in memory:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2? | CC | CF | CH | CP | FC | FF | FH | FP | HC | HF | HH | HP | PC | PF | PH |
| 1. | CC | CF | CH | FC | FF | FH | HC | HF | HH | | | | | | |
| 2! | CC | CF | CH | FC | FF | FH | HC | HF | HH | CF | FP | HP | PC | PF | PH |

| | j | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| i | | 0 | 1 | 2 | 9 |
| 0 | 0 | 0 | 1 | 2 | 9 |
| 1 | 3 | 3 | 4 | 5 | 10 |
| 2 | 6 | 6 | 7 | 8 | 11 |
| 3 | 12 | 12 | 13 | 14 | 15 |

1. $a = j + 3i$

2? $a = j + 4i$

2! $a = \max(\text{rowbase}(i), \text{colbase}(j))$

$+$ the other one, $i$ or $j$

Refs: E. J. Otoo '83; D. E. Knuth '97

## Finding all substrings

E.g., Mycobacterium tuberculosis from codon 729

16 letters: 32 bits

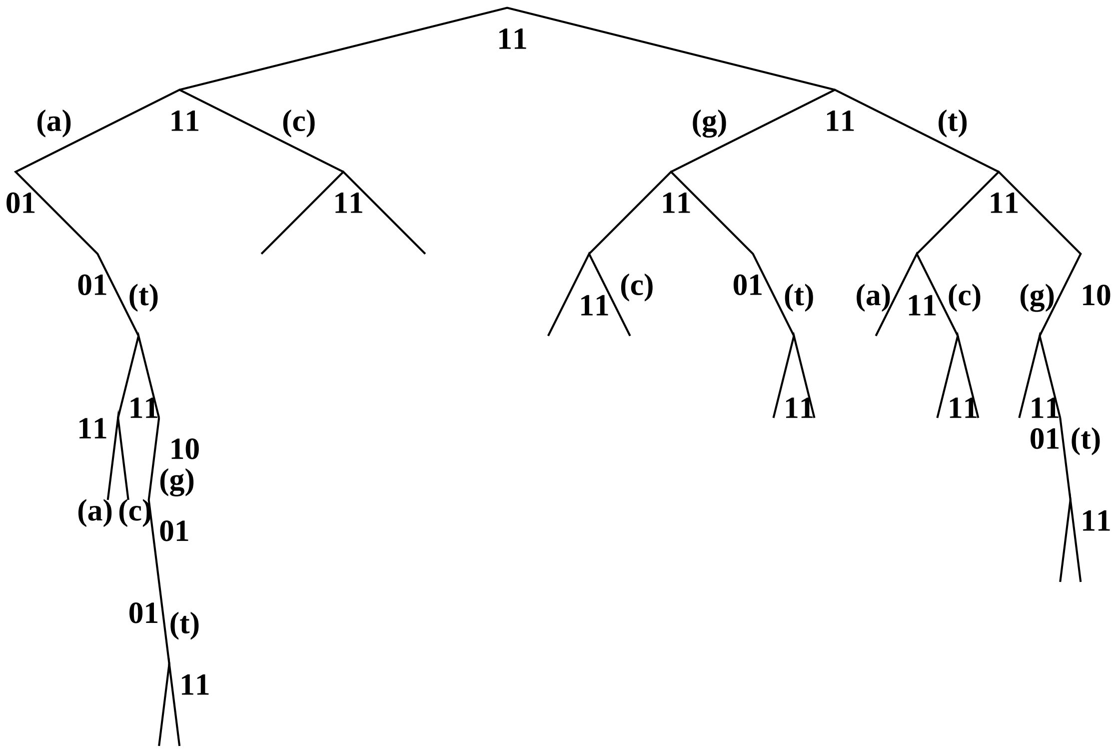
16×17/2 letters: 272 bits

```
atgtcatatgtgatcg
 tgtcatatgtgatcg
  gtcatatgtgatcg
   tcatatgtgatcg
    catatgtgatcg
     atatgtgatcg
      tatgtgatcg
       atgtgatcg
        tgtgatcg
         gtgatcg
          tgatcg
           gatcg
            atcg
             tcg
              cg
               g
```

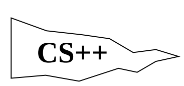
CS++ Trie: 174 bits

# Trie [ref De la Briandais '59]

```
          1
0123456789012345
atgtcatatgtgatcg
```



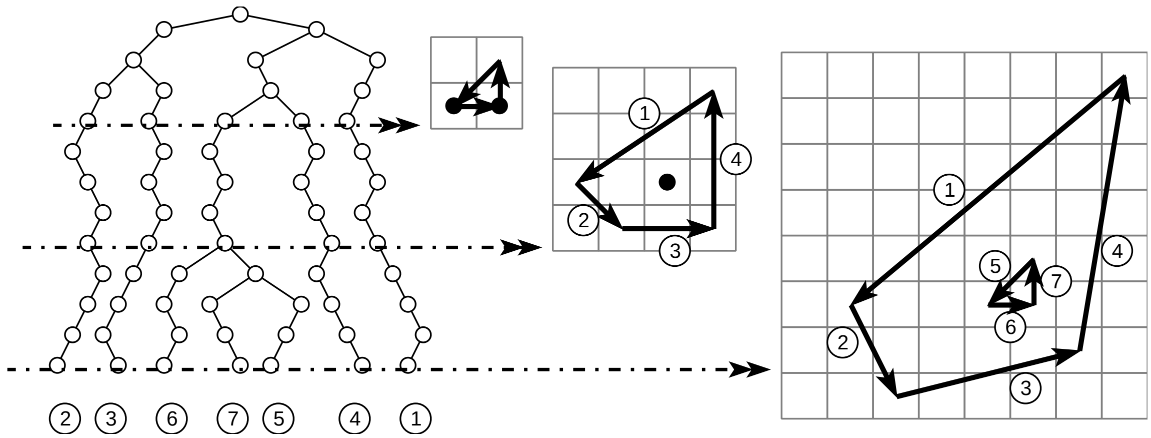

11
11 11
01 11 11 11
01 000100 001110 11 01 11 10
:

Sequential? 

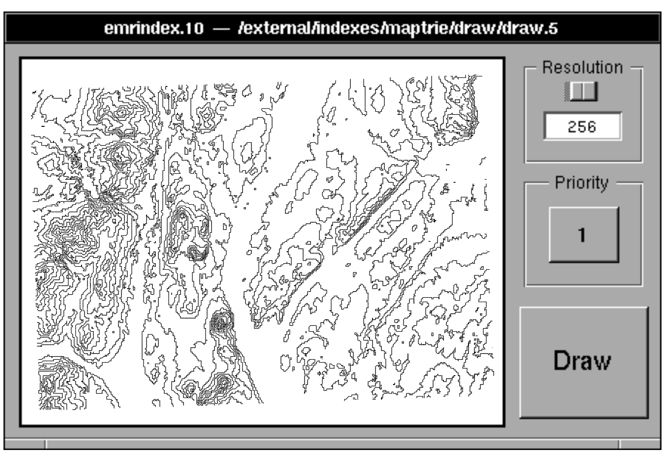
Logarithmic!

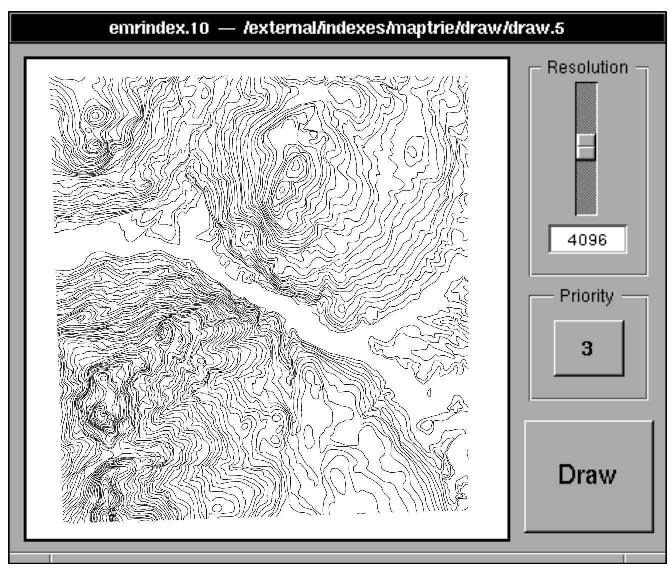
Ref.: J. A. Orenstein '83

# 1. Algorithms & Data Structures
## Variable-resolution maps

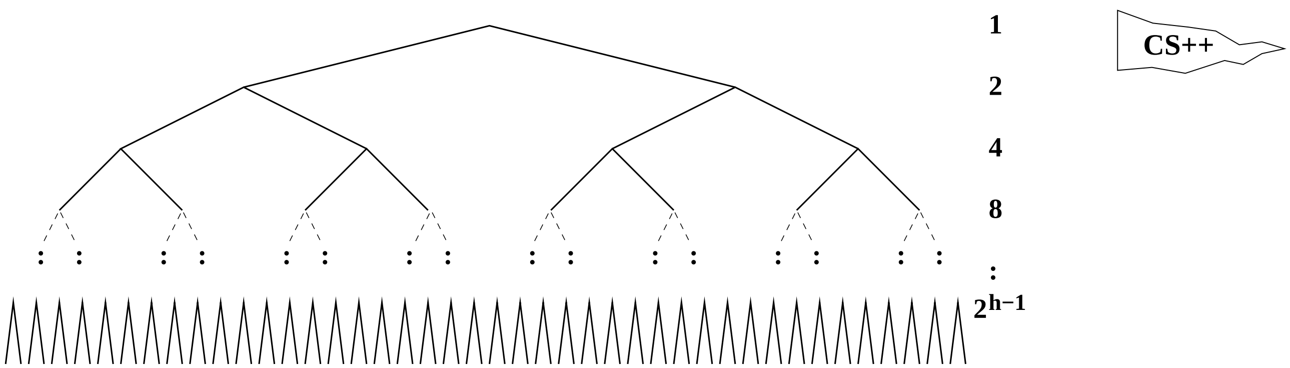



## Map zooming (Ref.: H. Shang '94)



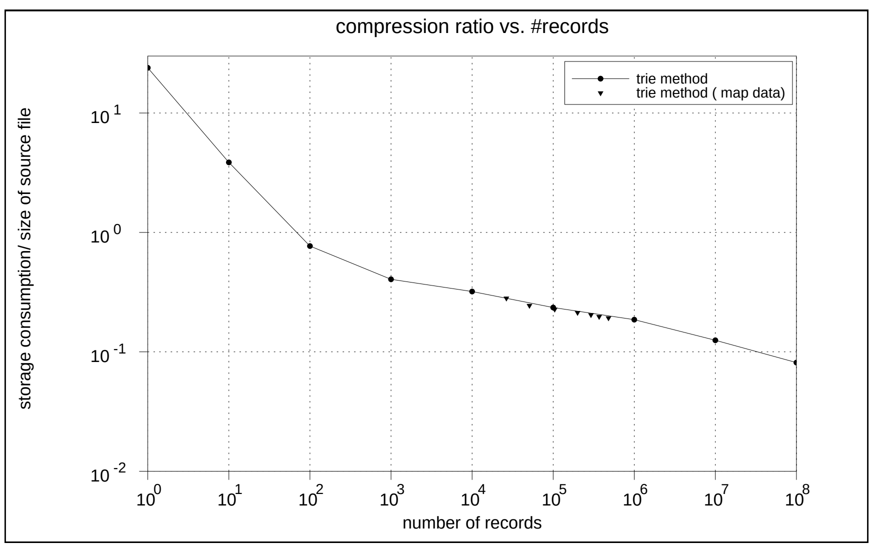
# 1. Algorithms & Data Structures
## Compression by tries



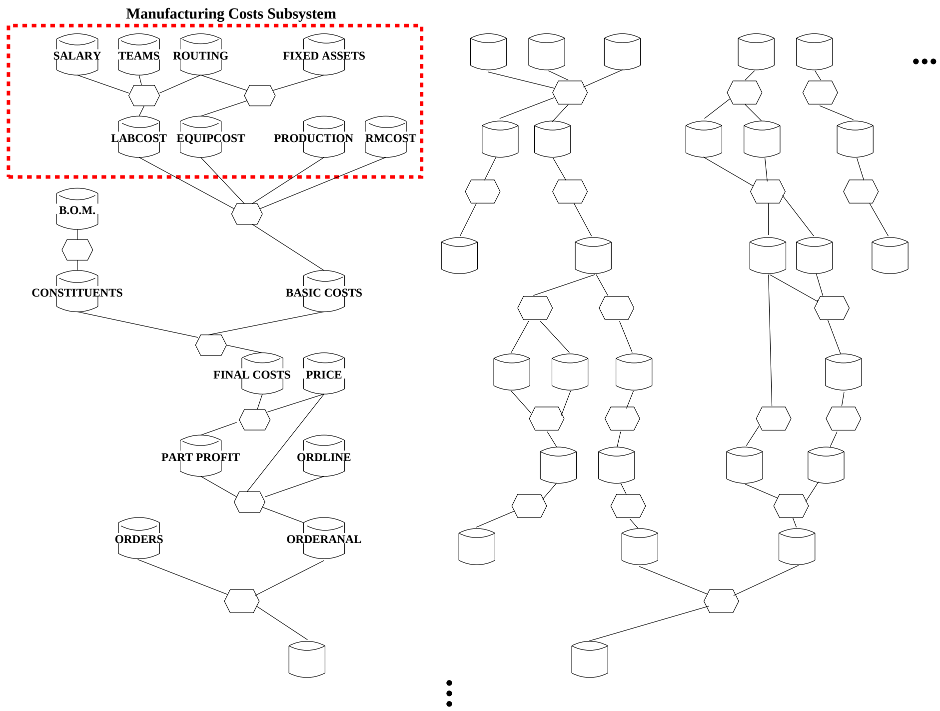

$$h \times 2^h \text{ vs } (2^h - 1) \times 2 : \frac{2}{h} = \frac{2}{\lg n}$$

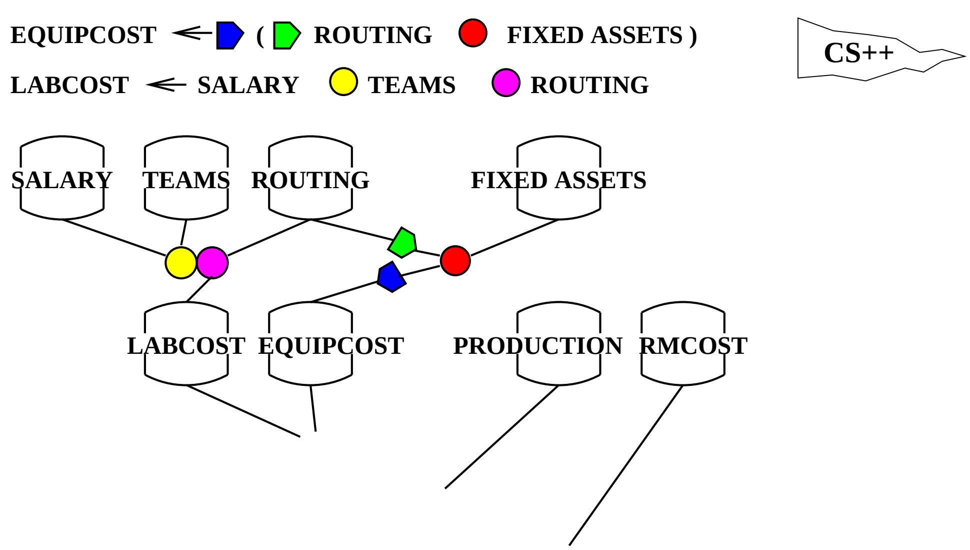
| $n$ | $10^3$ | $10^6$ | $10^9$ | $10^{12}$ |
|---|---|---|---|---|
| $2/\lg n$ | 1/5 | 1/10 | 1/15 | 1/20 |
| lossless compression | 80% | 90% | 93% | 95% |

## Experimental (Ref.: X. Y. Zhao, '00)



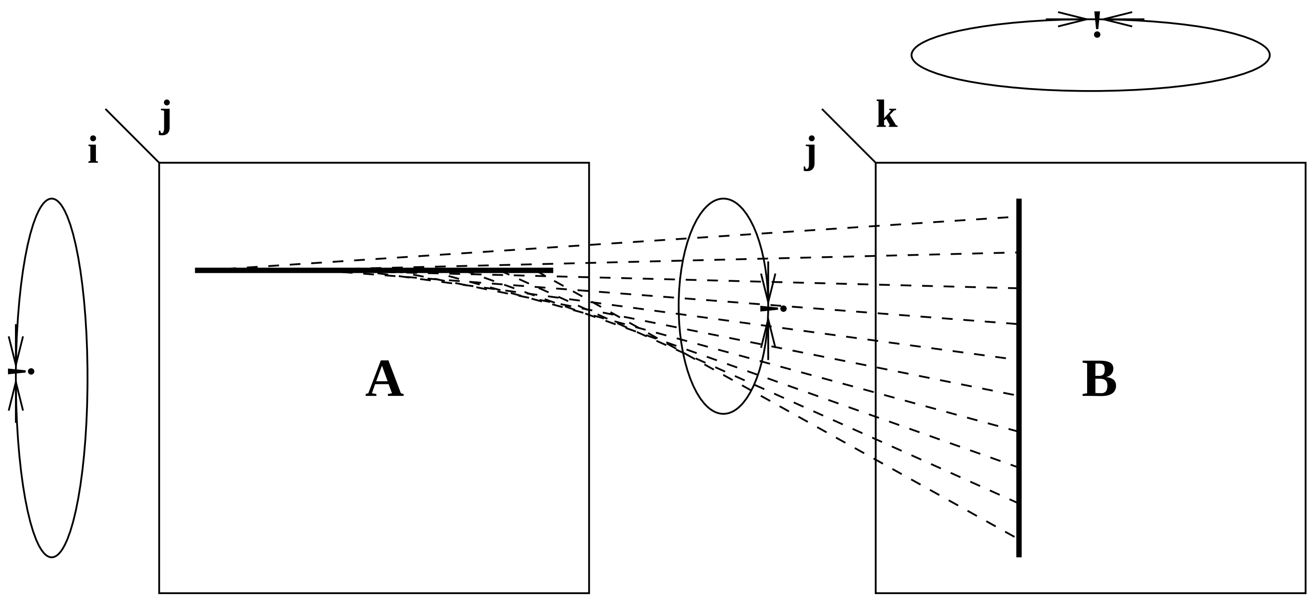
# 2. Programming Language
## Software engineering

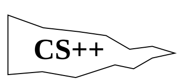



Ref.: Merrett '84

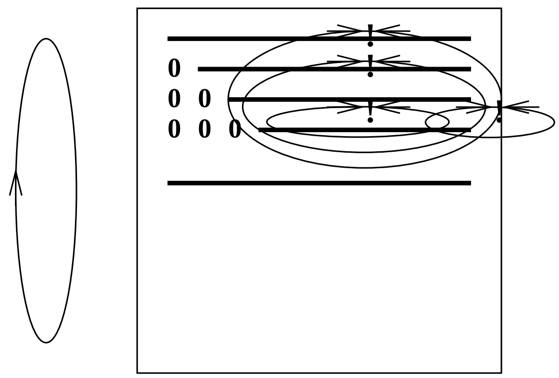
## Matrix multiplication



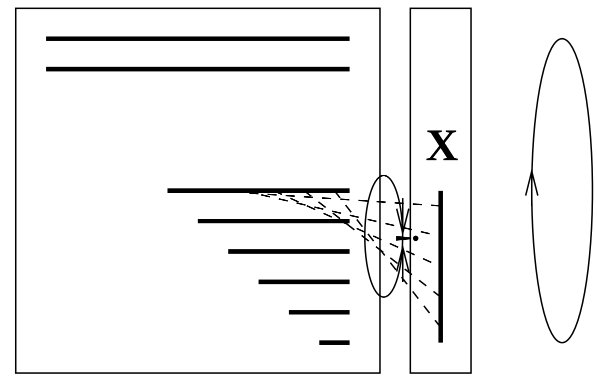




**let** *ab* **be equiv + of** $a \times b$ **by** $i, k$;
*AB* <−[$i, k$,*ab*] **in** ($A$ **natjoin** $B$);
Leave ordering to implementation:
already "parallelized"
NB Domain algebra ⊥ Relational algebra

# Gaussian elimination




**Triangularization**




**Back−substitution**

**let** $a'$ **be** $a$; **let** $a''$ **be** $a$;

**for** *row* $<-$ 1 **to**

  [**red max of** $i$] **in** $A$

$\{$ $A' <-[j, a']$ **where** $i$=row

  **in** $A$;

  $A$'' $<-[i, a'']$**where** $j$=row

  **in** $A$;

  **let** $aa$ **be** $(a' \times a'')/A$[row,row];

  **update** $A$ **change** $a <-$

  **if** $i \leq$*row* **then** $a$ **else** $a - aa$

  **using** $[i, j, aa]$ **in**

  ($A''$ **natjoin** $A'$)

$\}$

**relation** $X(j, x)$;

**let** *ax* **be equiv** $+$ **of** $a \times x$

  **by** $j$;

**for** *row* $<-$ [**red max of** $i$]

  **in** $A$ **to** 1 **by** $-1$

$\{$ $AX <-[ax]$ **in**

  ($A$ **natjoin** $X$);

  **let** $x$ **be** ($X$[*row*,**red max**

  **of** $i + 1$] $-ax)/A$[*row,row*];

  **let** $j$ **be** *row*;

  **update** $X$ **add** $[j, x]$ **in** $AX$

$\}$

CS++

# 2. Programming Language
## Expert systems

## **Horn Clauses** An Inference Engine

| [*New*]*Facts* | *Horn* | | | |
|---|---|---|---|---|
| (*Concl*) | (*Rule#* | *Ante* | *Concl* | ) |
| `lays eggs` | 1 | `lays eggs` | `is bird` | |
| `has feathers` | 1 | `has feathers` | `is bird` | |
| `swims` | 2 | `flies` | `is bird` | |
| ———— | 2 | `is not mammal` | `is bird` | |
| `is bird` | 3 | `is bird` | `is duck` | |
| ———— | 3 | `swims` | `is duck` | |
| `is duck` | 3 | `is brown` | `is duck` | |
| | 4 | `is bird` | `is duck` | |
| | 4 | `swims` | `is duck` | |
| | 4 | `is green` | `is duck` | |
| | 4 | `is red` | `is duck` | |
| | 5 | `is duck` | `migrates` | |
| | 5 | `is not tame` | `migrates` | |

CS++

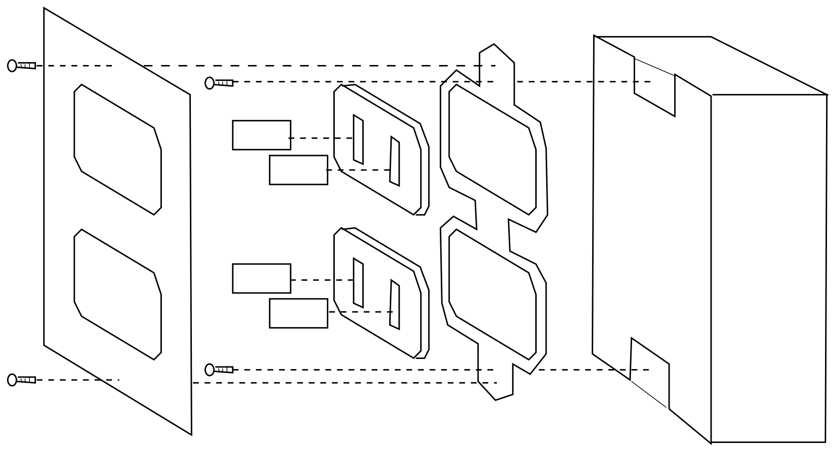
*NewFacts* **is** *Facts* **union**

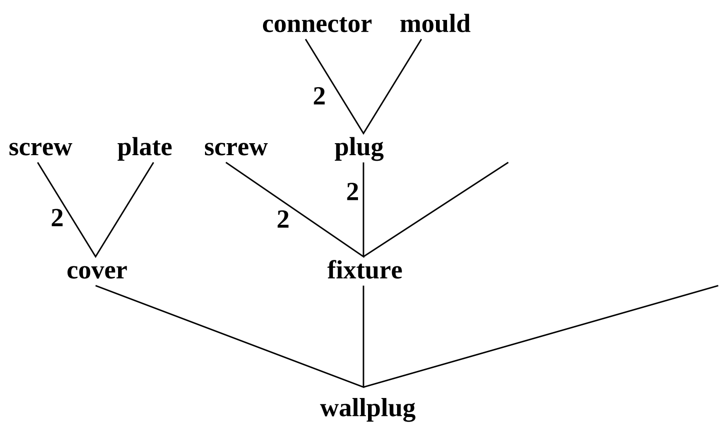
[*Concl*] **in** (*NewFacts*[*Concl* $\supseteq$ *Ante*]*Horn*)

*Relixpert* expands this 1-line inference engine to 50, in a 200-line expert system shell: TDKE **6** (1991) 151

## Bill of materials

E.g., a wallplug






CS++

**let** $A'$ **be** $A$; **let** $S'$ **be** $S$; **let** $Q'$ **be** $Q$;
**let** $Q''$ **be equiv** $+$ **of** $Q \times Q'$ **by** $A, S'$;
**let** $Q'''$ **be** $Q + Q''$; **let** $Q$ **be** $Q'''$;

E.g., `wallplug` has 4 `connector`s.

*Explo* **is** $[A, S, Q]$ **in** $[A, S, Q''']$ **in** (*PartOf* $[A, S$ **union** $A, S']$
$[A, S', Q'']$ **in** (*Explo* $[S$ **natjoin** $A']$ $[A', S', Q']$ **in** *PartOf*));

# 2. Programming Language
## Object orientation

**proc** *bankAccount* (*Balance, Deposit*) **is**
**state** *BAL* **intg**
{ **proc** *Deposit*(*dep*) **is**
  { *BAL* <– *BAL* + *dep*};
  **proc** *Balance*(*bal*) **is**
  { *bal* <– *BAL*;
  *BAL* <– 0
}

Instantiation is join.

**relation** *accts*(*acctno, client*) <–
  {(`1729, "Pat"`),(`4104, "Jan"`)};
*Accounts* <– *accts* **natjoin** *bankAccount*;

| (*accno* | *client* | *Balance* | *Deposit* | [*BAL*]) |
|---|---|---|---|---|
| 1729 | Pat | | | [0] |
| 4104 | Jan | | | [0] |

# Object orientation

**update** *Accounts* **change** *Deposit*(`100`)
  **using where** *acctno*=`4104`;

Ref.: Zheng '02

Inheritance

**proc** *interest*(*Interest*) **is**
**state** *BAL* **intg**;
{ **proc** *Interest*(*int*) **is**
  { $BAL <\!- BAL \times (1 + int/100.0)$};
}

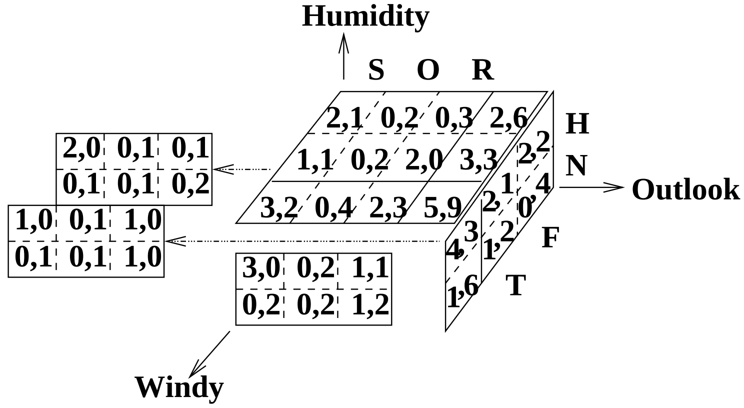
**relation** *intaccts* (*acctno, intrate*) <− {(`4104, 3`)};
*InterestAccounts* <− *intaccts* **natjoin** *interest*;
*InterestAccounts* **isa** *Accounts*;
**update** *InterestAccounts* **change** *Interest*(*intrate*)
  **using** *intaccts*;

| (*accno* | *client* | *intrate* | *Balance* | *Deposit* | *Interest* | [*BAL*]) |
|---|---|---|---|---|---|---|
| 1729 | Pat | – | | | – | [0] |
| 4104 | Jan | 3 | | | | [103] |

# E.g., Classification by Decision tree using Datacube

*Training*

| (*Outlook* | *Humidity* | *Windy* | $N$ | $P$) |
|---|---|---|---|---|
| sunny | high | f | 2 | 0 |
| sunny | high | t | 1 | 0 |
| sunny | normal | f | 0 | 1 |
| sunny | normal | t | 0 | 1 |
| overcast | high | f | 0 | 1 |
| overcast | high | t | 0 | 1 |
| overcast | normal | f | 0 | 1 |
| overcast | normal | t | 0 | 1 |
| rain | high | f | 0 | 1 |
| rain | high | t | 1 | 0 |
| rain | normal | f | 0 | 2 |
| rain | normal | t | 1 | 0 |



# Datacube

```
let N be totN;
let P be totP;
domain attr attribute;
relation AllAttribs(attr) <- AttribsOf Training;
//Outlook, Humidity, Windy, N, P
relation ClassAttribs(attr) <- {(N),(P)};
relation TotAttribs(attr) <- {(totN), (totP)};
PropAttribs <- AllAttribs diff ClassAttribs;
LoopAttribs <- PropAttribs;
while [] in LoopAttribs
{ Attrib <- pick LoopAttribs;
   update LoopAttribs delete Attrib;
   let eval Attrib be "ANY";
   let totN be equiv + of N by (PropAttribs diff Attrib);
   let totP be equiv + of P by (PropAttribs diff Attrib);
   update Training add [AllAttribs] in
      [PropAttribs diff Attrib union TotAttribs] in Training;
}
```

The decision tree analysis follows directly; “one-rule” and Bayesian classification methods are special cases.

# 2. Programming Language
## Semistructured data

Text:

Ted married Alice in 1932. Their children, Mary (1934) married Alex in 1954 (Joe was born to Mary and Alex in 1956) and James (1935) married Jane in 1960 (James and Jane had Tom in 1961 and Sue in 1962).

Marked up text ($x$ML):

```
<Person>
  <Name>Ted</Name> married
  <Family><Conj>Alice</Conj> in
   <Wed>1932</Wed>. Their children,
    <Children><Name>Mary</Name> (<DoB>1934</DoB>) married
      <Family><Conj>Alex</Conj> in <Wed>1954</Wed>
       :
      </Family>
     :
    </Children>
  </Family>
</Person>
```

Convert to (recursively nested) relation:

**let** *FAMILY* **be** [*Conj, Wed, CHILDREN*] **mu2nest** *Family*;
**let** *CHILDREN* **be** [*DoB, Name, FAMILY*] **mu2nest** *Children*;
*PERSON* $<-$ [*Name, FAMILY*] **mu2nest** *Person*;

# Semistructured data

## Here's the relation

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *PERSON* | | | | | | | | | | |
| (*Name* | *FAMILY* | | | | | | | | | ) |
| | (*Conj* | *Wed* | *CHILDREN* | | | | | | | ) |
| | | | (*DoB* | *Name* | *FAMILY* | | | | | ) |
| | | | | | (*Conj* | *Wed* | *CHILDREN* | | | ) |
| | | | | | | | (*DoB* | *Name* | ) | |
| Ted | Alice | 1932 | 1934 | Mary | Alex | 1954 | 1956 | Joe | | |
| | | | 1935 | James | Jane | 1960 | 1961 | Tom | | |
| | | | | | | | 1962 | Sue | | |

## Queries:

*PERSON*/*Name*

`Ted`

*PERSON*/*FAMILY*/*CHILDREN*/*Name*

`Mary, James`

*PERSON*/*FAMILY*/*CHILDREN*/*FAMILY*/*CHILDREN*/*Name*

`Joe, Tom, Sue`

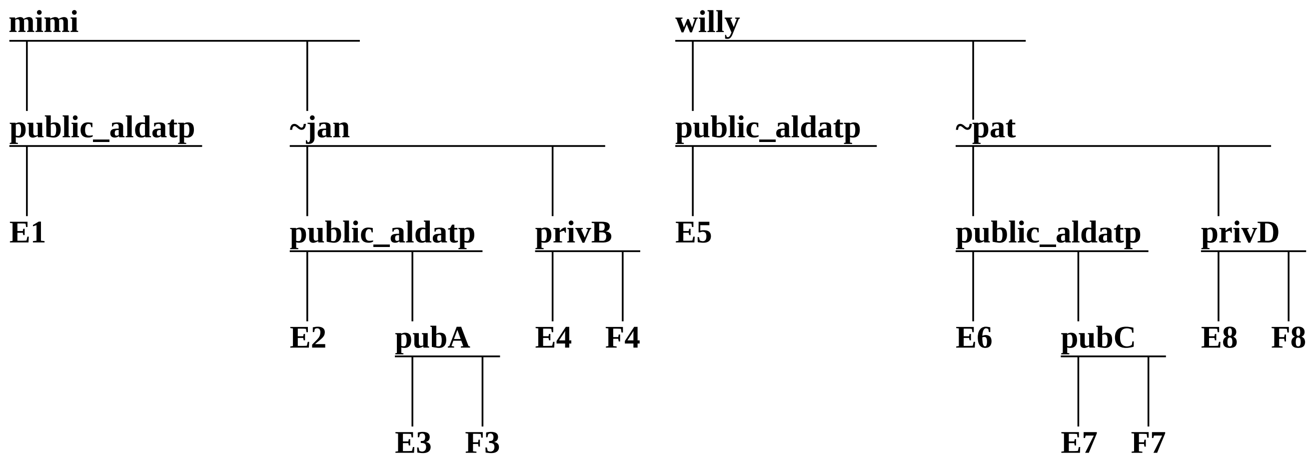
*PERSON*/(./)**Name*

`Ted, Mary, James, Joe, Tom, Sue`

*Name* **where** *FAMILY*/*Conj*=`"Alice"` **in** *PERSON*

`Ted`

E.g., aldat protocol




Extended names may be used anywhere permissions allow:

*F4* $<-$ *aldatp*://*mimi*/$\sim$*jan*/*pubA*/*E3*;
*aldatp*://*mimi*/$\sim$*jan*/*pubA*/*F3* $<-$ *E2*;
*aldatp*://*willy*/$\sim$*pat*/*pubC*/$\{$*F7* $<-$ *E7*$\}$;

Joining *E3*$(A, B)$ with *E7*$(B, C)$ by semijoin:

(*aldatp*://*mimi*/$\sim$*jan*/*pubA*/(*E3* **natjoin**
*aldatp*://*willy*/$\sim$*pat*/*pubC*/($[B]$ **in** *E7*))) **natjoin** *E7*

Ref.: Wang '02

# Conclusions

SS different from RAM => new thinking about:

Computer Science:

- object orientation
- parallel programming
- artificial intelligence
- networking

Applications:

- numerical analysis
- bioinformatics
- G.I.S.
- semistructure

Current work: visualization

Future work:

- constraint databases
- peer-to-peer cooperative work
- agent programming