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

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

T. H. Merrett
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:

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>F</th>
<th>H</th>
<th>earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charles makes Clothes</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Fred makes Food</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Harry makes Houses</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

spent | 6 | 6 | 10

Woops, we forgot Pete, who supplies Power:

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>F</th>
<th>H</th>
<th>P</th>
<th>earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charles makes Clothes</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Fred makes Food</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Harry makes Houses</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>Pete makes Power</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

spent | 7 | 9 | 15 | 9

Represent these in memory:

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>1?</td>
<td>CC</td>
<td>CF</td>
<td>CH</td>
<td>CP</td>
<td>FC</td>
<td>FF</td>
<td>FH</td>
<td>FP</td>
<td>HC</td>
<td>HH</td>
<td>HP</td>
<td>PC</td>
<td>PF</td>
<td>PH</td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>CC</td>
<td>CF</td>
<td>CH</td>
<td>FC</td>
<td>FF</td>
<td>FH</td>
<td>HC</td>
<td>HF</td>
<td>HH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2!</td>
<td>CC</td>
<td>CF</td>
<td>CH</td>
<td>FC</td>
<td>FF</td>
<td>FH</td>
<td>HC</td>
<td>HF</td>
<td>HH</td>
<td>CF</td>
<td>FP</td>
<td>HP</td>
<td>PC</td>
<td>PF</td>
<td>PH</td>
</tr>
</tbody>
</table>
```

```cpp
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
1. Algorithms & Data Structures

Finding all substrings

E.g., Mycobacterium tuberculosis from codon 729

atgtcatatgtgatcg
tgcatatgtgatcg
gtcatatgtgatcg
tcatatgtgatcg
catatgtgatcg
atatgtgatcg
tatgtgatcg
atgtgatcg
tgtgatcg
gtgatcg
gtgatcg
tgatcg
atcg
tcg
cg
g

16
letters:
32 bits
16×17/2
letters: 272 bits

Trie: 174 bits

T. H. Merrett

©05/1
Trie [ref De la Briandais '59]

0123456789012345
atgtcatatgtgatcg

Sequential?

Ref.: J. A. Orenstein '83

©05/1
1. Algorithms & Data Structures
Variable-resolution maps

Map zooming (Ref.: H. Shang ’94)

T. H. Merrett
### 1. Algorithms & Data Structures

Compression by tries

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

<table>
<thead>
<tr>
<th>( n )</th>
<th>10^3</th>
<th>10^6</th>
<th>10^9</th>
<th>10^{12}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/\lg n</td>
<td>1/5</td>
<td>1/10</td>
<td>1/15</td>
<td>1/20</td>
</tr>
<tr>
<td>lossless compression</td>
<td>80%</td>
<td>90%</td>
<td>93%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Experimental (Ref.: X. Y. Zhao, ’00)

![Diagram](image)

T. H. Merrett

©05/1
2. Programming Language
Software engineering

Ref.: Merrett '84

T. H. Merrett
Matrix multiplication

\[
\begin{array}{c}
\text{let } ab \text{ be equiv} + \text{ of } a \times b \text{ by } i, k; \\
AB \leftarrow [i, k, ab] \text{ in } (A \text{ natjoin } B); \\
\text{Leave ordering to implementation: already “parallelized”}
\end{array}
\]

NB Domain algebra \(\perp\) Relational algebra

T. H. Merrett
Gaussian elimination

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 $(a' \times a'')/A[\text{row}, \text{row}]$;

update $A$ change $a ←$

if $i ≤ row$ then $a$ else $a - aa$

using $[i, j, aa]$ in

$(A'' \text{ natjoin } A')$

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

$(A \text{ natjoin } X)$;

let $x$ be $(X[\text{row}, \text{red max of } i + 1] - ax)/A[\text{row}, \text{row}]$;

let $j$ be row;

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

CS++
2. Programming Language  
Expert systems

**Horn Clauses** An Inference Engine

<table>
<thead>
<tr>
<th>NewFacts</th>
<th>Horn</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Concl)</td>
<td>(Rule#</td>
</tr>
<tr>
<td>lays eggs</td>
<td>1</td>
</tr>
<tr>
<td>has feathers</td>
<td>1</td>
</tr>
<tr>
<td>swims</td>
<td>2</td>
</tr>
<tr>
<td>is bird</td>
<td>2</td>
</tr>
<tr>
<td>is duck</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>is duck</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

NewFacts is Facts union

\[\text{[Concl \ in (NewFacts[Concl \supseteq \ Ante]Horn)}\]

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

T. H. Merrett

©05/1
2. Programming Language

Bill of materials

E.g., a wallplug

```
let A' be A; let S' be S; let Q' be Q;
let Q'' be equiv + of Q x Q' by A, S';
let Q''' be Q + Q''; let Q be Q''';
```

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;

(acctno client Balance Deposit [BAL])

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>Pat</td>
<td></td>
<td>[0]</td>
</tr>
<tr>
<td>4104</td>
<td>Jan</td>
<td></td>
<td>[0]</td>
</tr>
</tbody>
</table>

T. H. Merrett
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 x (1 + int/100.0) };
}

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

(acctno client intrate Balance Deposit Interest [BAL])
1729 Pat — — — [0]
4104 Jan 3

T. H. Merrett
2. Programming Language

Data mining

E.g., Classification by Decision tree using Datacube

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Humidity</th>
<th>Windy</th>
<th>N</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>high</td>
<td>f</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>sunny</td>
<td>high</td>
<td>t</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>sunny</td>
<td>normal</td>
<td>f</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>sunny</td>
<td>normal</td>
<td>t</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>overcast</td>
<td>high</td>
<td>f</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>overcast</td>
<td>high</td>
<td>t</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>overcast</td>
<td>normal</td>
<td>f</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>overcast</td>
<td>normal</td>
<td>t</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>rain</td>
<td>high</td>
<td>f</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>rain</td>
<td>high</td>
<td>t</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>rain</td>
<td>normal</td>
<td>f</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>rain</td>
<td>normal</td>
<td>t</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Datacube

let $N$ be $\text{tot}N$;
let $P$ be $\text{tot}P$;
domain $\text{attr}$ attribute;
relation $\text{AllAttribs}(\text{attr}) \leftarrow \text{AttribsOf Training}$;
    // Outlook, Humidity, Windy, $N$, $P$
relation $\text{ClassAttribs}(\text{attr}) \leftarrow \{(N),(P)\}$;
relation $\text{TotAttribs}(\text{attr}) \leftarrow \{(\text{tot}N),(\text{tot}P)\}$;
$\text{PropAttribs} \leftarrow \text{AllAttribs} \text{diff} \text{ClassAttribs}$;
$\text{LoopAttribs} \leftarrow \text{PropAttribs}$;
while $\square$ in $\text{LoopAttribs}$
    { $\text{Attrib} \leftarrow \text{pick LoopAttribs}$;
      update $\text{LoopAttribs delete Attrb}$;
      let $\text{eval Attrb}$ be "ANY";
      let $\text{tot}N$ be equiv + of $N$ by ($\text{PropAttribs}$ diff $\text{Attrib}$);
      let $\text{tot}P$ be equiv + of $P$ by ($\text{PropAttribs}$ diff $\text{Attrib}$);
      update $\text{Training add [AllAttribs in}$
        $[\text{PropAttribs diff Attrb union TotAttribs}]$ in $\text{Training}$;
    }

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

T. H. Merrett

©05/1
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 (XML):

```xml
<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>
      :
    </Children>
  </Family>
  :
</Family>
</Person>
```

Convert to (recursively nested) relation:

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

T. H. Merrett
Semistructured data

Here's the relation

<table>
<thead>
<tr>
<th>PERSON (Name)</th>
<th>FAMILY (Conj)</th>
<th>Wed</th>
<th>CHILDREN (DoB)</th>
<th>Name</th>
<th>FAMILY (Conj)</th>
<th>Wed</th>
<th>CHILDREN (DoB)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ted</td>
<td>Alice</td>
<td>1932</td>
<td>1934</td>
<td>Mary</td>
<td>Alex</td>
<td>1954</td>
<td>1956</td>
<td>Joe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1935</td>
<td>1934</td>
<td>Mary</td>
<td>Alex</td>
<td>1954</td>
<td>1956</td>
<td>Joe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1960</td>
<td>Jane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1960</td>
<td>Jane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1960</td>
<td>Jane</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1962</td>
<td>Sue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1962</td>
<td>Sue</td>
</tr>
</tbody>
</table>

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

T. H. Merrett
2. Programming Language

Internet

E.g., aldat protocol

Extended names may be used anywhere permissions allow:

\[
F4 \leftarrow \text{aldatp://mimi/\simjan/pubA/E3}; \\
\text{aldatp://mimi/\simjan/pubA/F3} \leftarrow E2; \\
\text{aldatp://willy/\simpat/pubC/\{F7} \leftarrow E7); \\
\]

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

\[
(\text{aldatp://mimi/\simjan/pubA/(E3 \text{natjoin} \\
\text{aldatp://willy/\simpat/pubC/([B} \text{ in } E7)))} \text{natjoin } E7
\]

Ref.: Wang '02

T. H. Merrett
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