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.

Algebraic Principles

- Things

- Operations on things

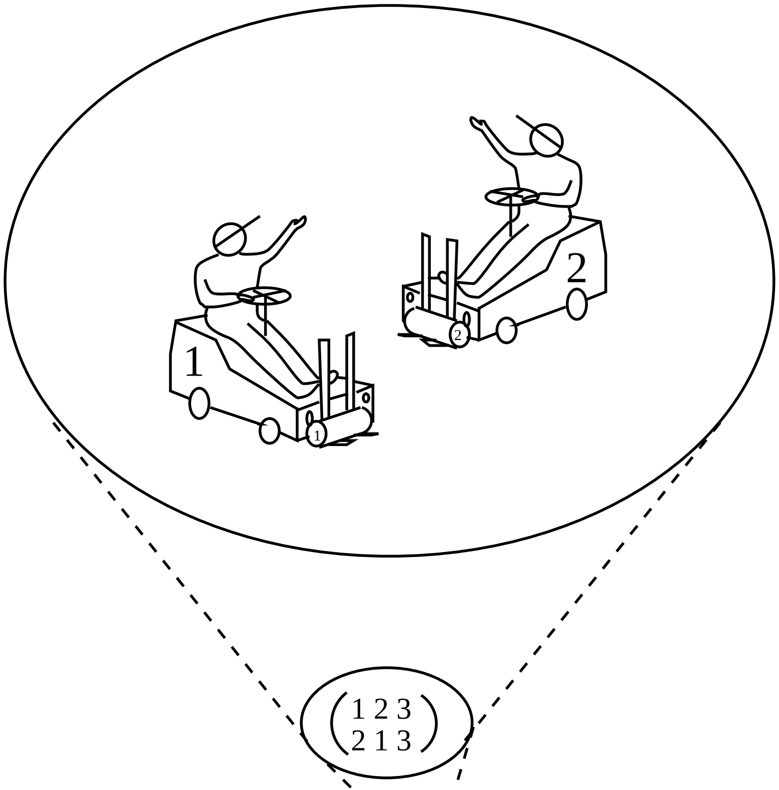
The *Principle of Abstraction*

the structure and the context of a thing should be of no concern to the operation

The *Principle of Closure*

operations on a thing should produce things of the same type

Abstraction

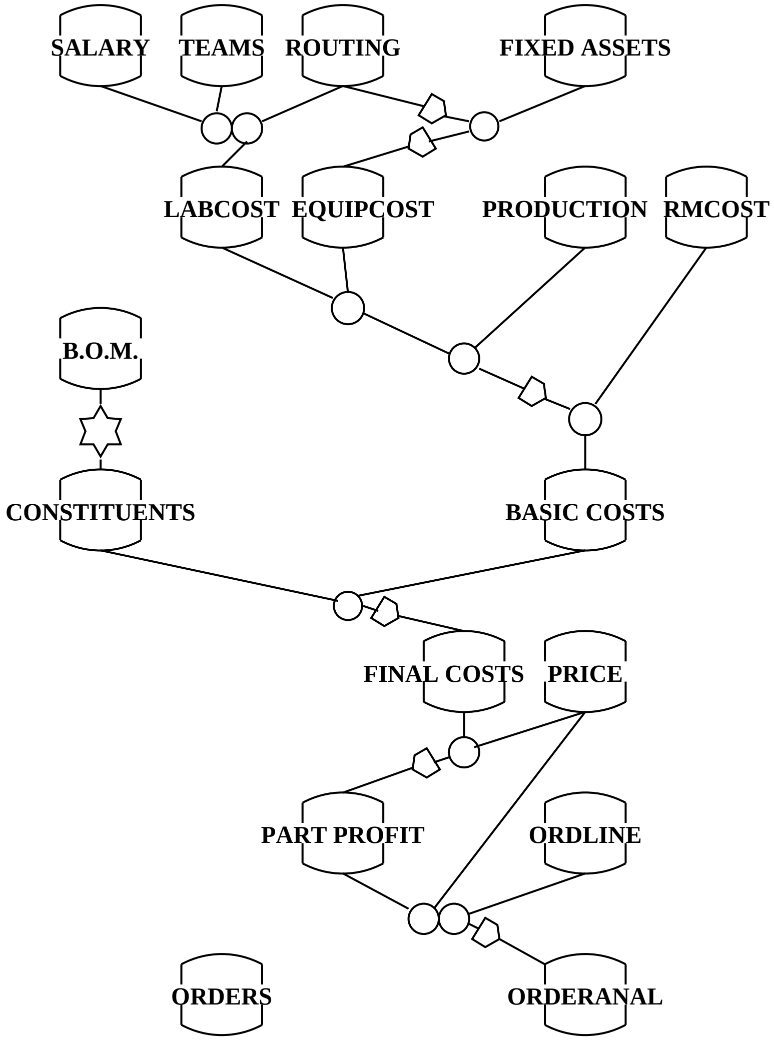


.	A	B	C	D	E	F
A	A	B	C	D	E	F
B	B	C	A	E	F	D
C	C	A	B	F	D	E
D	D	F	E	A	C	B
E	E	D	F	B	A	C
F	F	E	D	C	B	A

Closure

EQUIPCOST <— ⬠ (⬠ ROUTING ○ FIXED ASSETS)

LABCOST <— SALARY ○ TEAMS ○ ROUTING



Relations

Orderbook

(*Ord#*	*Cust*	*Sales*	*Assembly*	*Qty*)
4	PR	H	Car	37
3	L&S	E	Car	23
2	NYC	N	CabooseLocomotive	1
7	GTRC	N	Locomotive	47
3	L&S	E	Caboose	3
5	NYC	H	Locomotive	13
7	GTRC	N	Caboose	43
8	GNS	E	Toy Train	37
1	GNS	E	Locomotive	2
5	NYC	H	Car	31
6	B&O	H	Car	17
4	PR	H	Toy Train	11
3	L&S	E	Locomotive	5
1	GNS	E	Toy Train	7
7	GTRC	N	Car	139

re-ordered 1

Orderbook

(*Ord#*	*Cust*	*Sales*	*Assembly*	*Qty*)
1	GNS	E	Locomotive	2
			Toy Train	7
2	NYC	N	Locomotive	1
3	L&S	E	Car	23
			Caboose	3
			Locomotive	5
4	PR	H	Car	37
			Toy Train	11
5	NYC	H	Locomotive	13
			Car	31
6	B&O	H	Car	17
7	GTRC	N	Locomotive	47
			Caboose	43
			Car	139
8	GNS	E	Toy Train	37

re-ordered 2

Orderbook

(*Cust*	*Ord#*	*Sales*	*Assembly*	*Qty*)
B&O	6	H	Car	17
GNS	1	E	Locomotive	2
			Toy Train	7
	8	E	Toy Train	37
GTRC	7	N	Locomotive	47
			Caboose	43
			Car	139
L&S	3	E	Car	23
			Caboose	3
			Locomotive	5
NYC	2	N	Locomotive	1
	5	H	Locomotive	13
			Car	31
PR	4	H	Car	37
			Toy Train	11

re-ordered 3

Orderbook

(*Sales*	*Ord#*	*Cust*	*Assembly*	*Qty*)
E	1	GNS	Locomotive	2
			Toy Train	7
	3	L&S	Car	23
			Caboose	3
			Locomotive	5
	8	GNS	Toy Train	37
H	4	PR	Car	37
			Toy Train	11
	5	NYC	Locomotive	13
			Car	31
	6	B&O	Car	17
N	2	NYC	Locomotive	1
	7	GTRC	Locomotive	47
			Caboose	43
			Car	139

Properties of Relations

- All rows are distinct.
- The ordering of rows is immaterial.
- Each column is labelled, making the ordering of columns insignificant.
- The value in each row under a given column is “simple”.

Terminology

Relation

Attribute — the label of a column.

Tuple — a row.

Key — a key of a relation is a minimal subset of its attributes, which can be used to identify each tuple uniquely.

Decomposition (Normalization)

(*Ord#*	*Cust*	*Sales*)
4	PR	H
3	L&S	E
2	NYC	N
7	GTRC	N
5	NYC	H
8	GNS	E
1	GNS	E
6	B&O	H

(*Ord#*	*Assembly*	*Qty*)
4	Car	37
3	Car	23
2	Locomotive	1
7	Locomotive	47
3	Caboose	3
5	Locomotive	13
7	Caboose	43
8	Toy Train	37
1	Locomotive	2
5	Car	31
6	Car	17
4	Toy Train	11
3	Locomotive	5
1	Toy Train	7
7	Car	139

Database:

a collection of relations

Orders(*Ord#*, *Cust*, *Sales*)

Ordline(*Ord#*, *Assembly*, *Qty*)

Keys

Orders(**Ord#**, *Cust, Sales*)

Ordline(**Ord#, Assembly**, *Qty*)

Orderbook(**Ord#, Assembly**, *Cust, Sales, Qty*)

Functional Dependence

Ord# → *Cust*

Ord# → *Sales*

Ord#, Assembly → *Qty*

Telephone Book Dependence

(Place a √ where there is a functional dependence!)

Tbook(*Name*, *Address*, *Phone*)

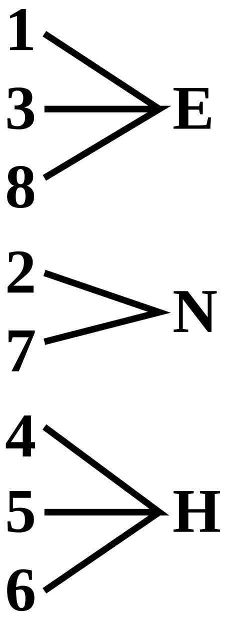
→	*Name*	*Address*	*Phone*
Name			
Address			
Phone			
Name, *Address*			
Name, *Phone*			
Address, *Phone*			

Table, Graph and Matrix Forms

OS(*Ord#*, *Sales*)

Ord#	*Sales*
1	E
2	N
3	E
4	H
5	H
6	H
7	N
8	E

Table

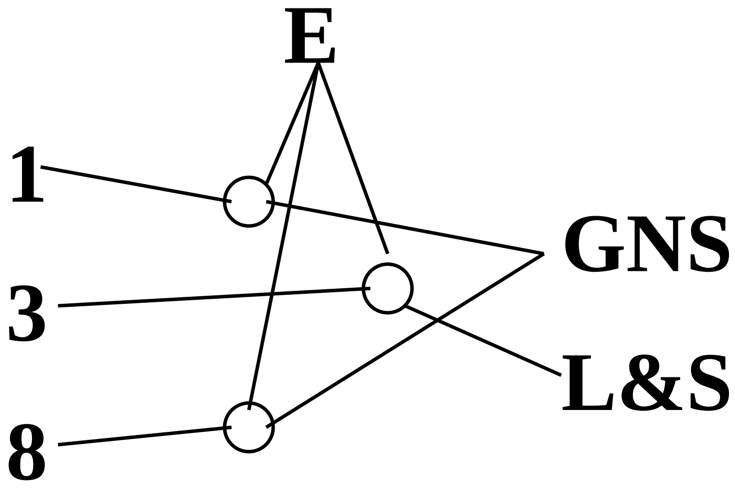


Graph

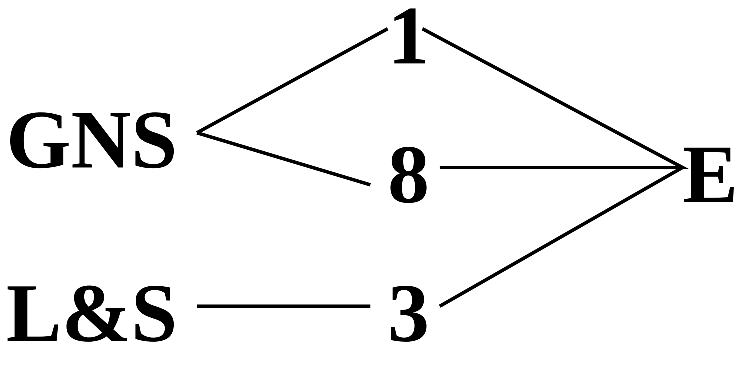
	H	E	N
1		1	
2			1
3		1	
4	1		
5	1		
6	1		
7			1
8		1	

Matrix

Exploiting the Graph Form

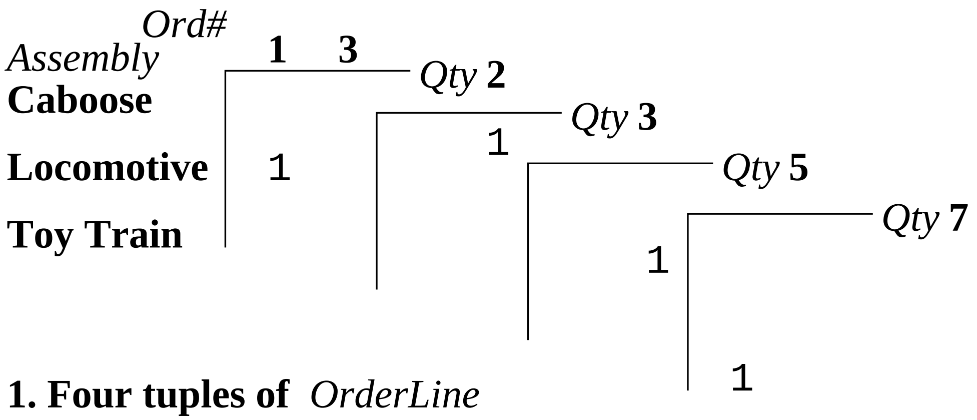


1. Three tuples of *Orders*



2. Special case: revealing key

Exploiting the Matrix Form



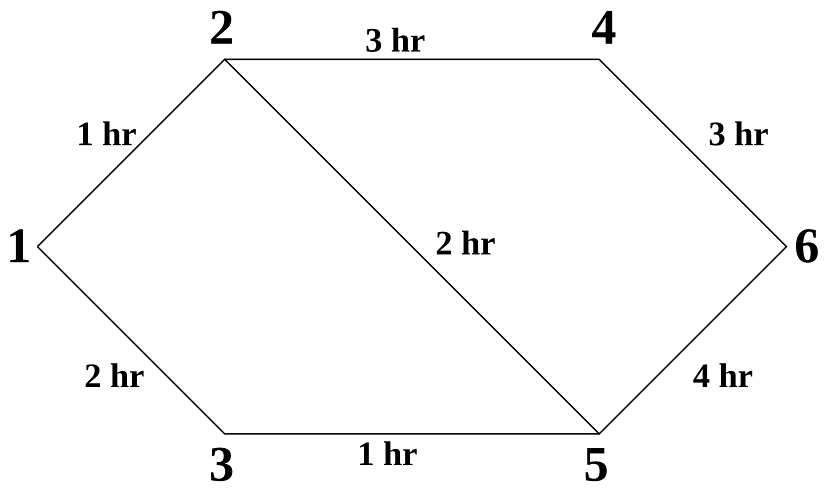
1. Four tuples of *OrderLine*

Assembly \ *Ord#*	**1**	**3**
Caboose		**3**
Locomotive	**2**	**5**
Toy Train	**7**	

2. Special case, revealing key

Some Relations

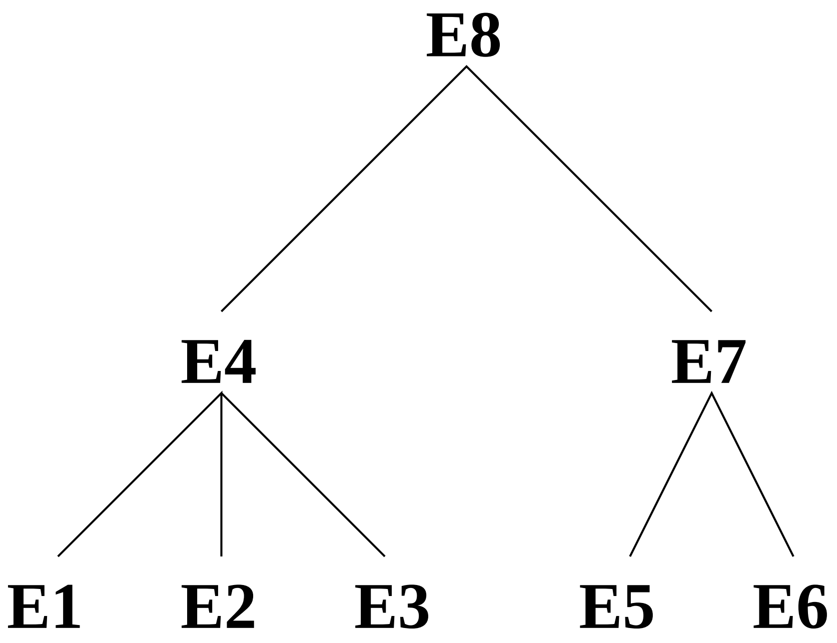
PERT Network



PERT

(*Start*	*Finish*	*Duration*)
1	2	1hr
1	3	2hr
2	4	3hr
2	5	2hr
3	5	1hr
4	6	3hr
5	6	4hr

Organization Chart



Org

(*Manager*	*Employee*)
E8	E4
E8	E7
E4	E1
E4	E2
E4	E3
E7	E5
E7	E6

Text

Text (*Word*	*Seq*)
Algebraic	1
data	2
processing	3
techniques	4
can	5
enable	6
applications	7
programmers	8
to	9
work	10
with	11
units	12
of	13
data	14
larger	15
than	16
a	17
single	18
computer	19
word	20

Diagrams

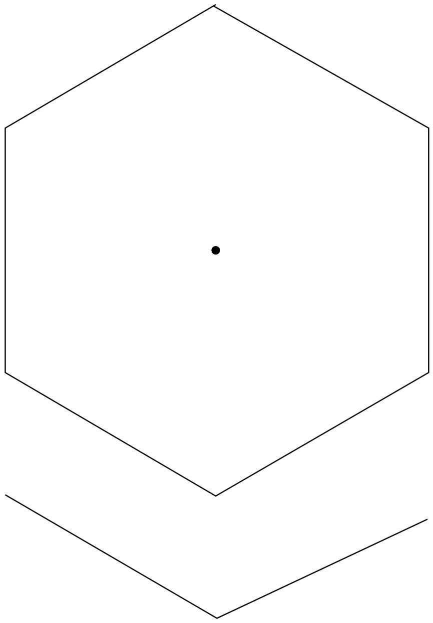
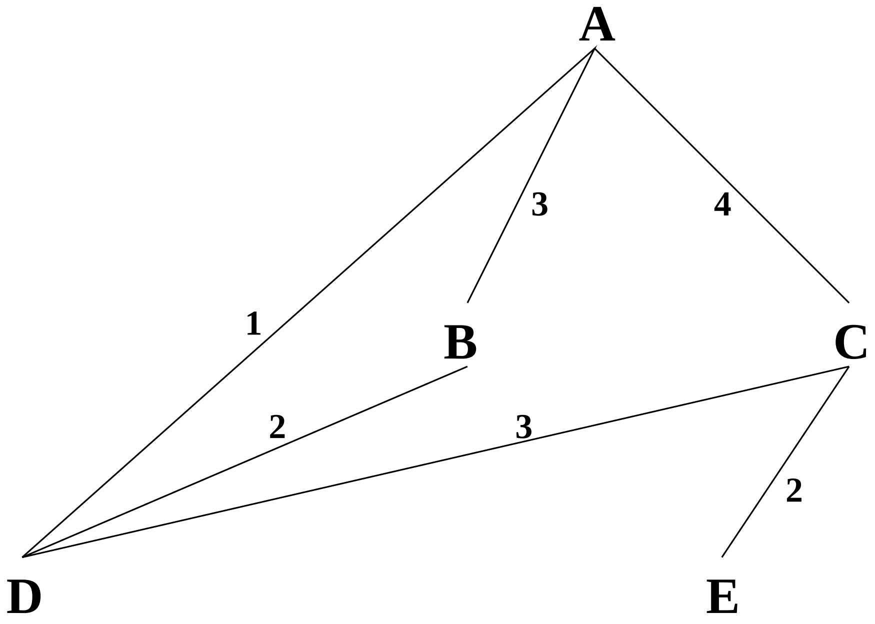


Diagram					
(*Feature*	*Group*	*Type*	*Seq*	X	Y)
Hex	1	region	1	0	-1
Hex	1	region	2	.866	-.5
Hex	1	region	3	.866	.5
Hex	1	region	4	0	1
Hex	1	region	5	-.866	.5
Hex	1	region	6	-.866	-.5
Rest	1	line	1	-.866	-1
Rest	1	line	2	0	-1.5
Rest	1	line	3	.866	-1
Rest	2	line	1	0	0

Bill of Materials



PartOf

(A	S	Q)
A	B	3
A	C	4
A	D	1
B	D	2
C	D	3
C	E	2

with Costs

Cost

(A	C)
A	.7
B	.4
C	.1
D	.2
E	.3

Implementing Relations

(Briefly: to reinforce the ideas, not to dwell on the machinery underneath)

Sequential Files

Logarithmic Files

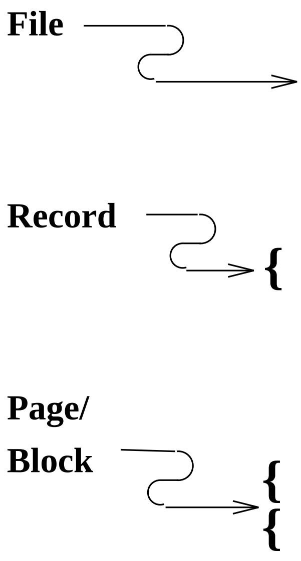
Direct Access Files

Hybrid Files

Z-Ordering

Sequential Files

Ordered



Ord#	*Cust*	*Sales*
1	GNS	E
2	NYC	N
3	L&S	E
4	PR	H
5	NYC	H
6	B&O	H
7	GTRC	N
8	GNS	E

$N =$ 8 records

$n =$ 4 blocks

Unordered

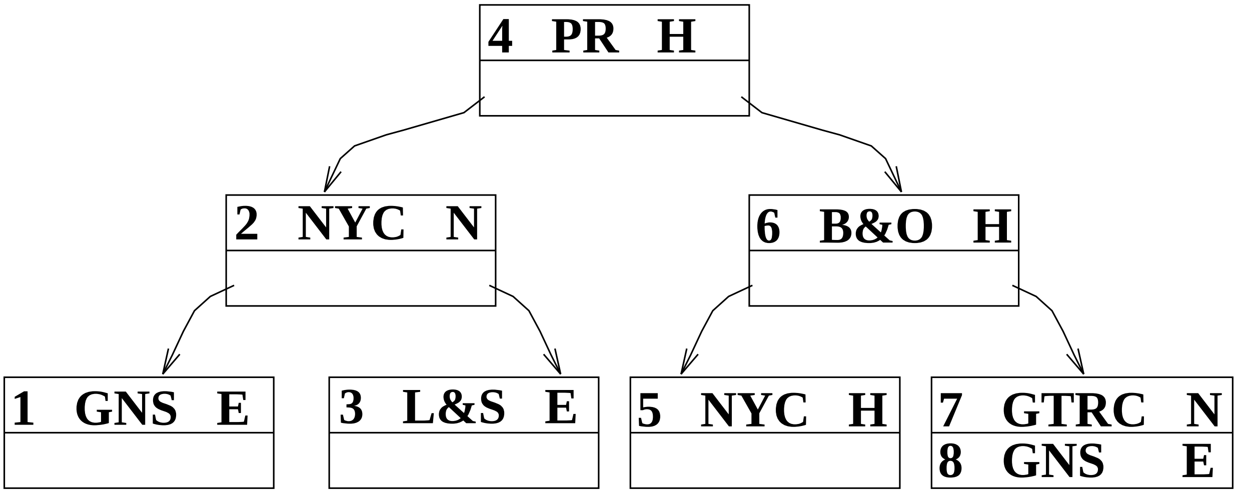
Average cost of a successful search: $n/2$ accesses.

Sequential files are best for *high activity*.

i.e. $>\sim$% of records accessed.

Logarithmic Files

e.g., B-trees



4 PR H

2 NYC N

6 B&O H

1 GNS E

3 L&S E

5 NYC H

7 GTRC N

8 GNS E

Average cost of a successful search:
log n accesses.

e.g., $n = 6$ $\lceil \log_2 n \rceil = 3$

B-trees are very flexible, good for *dynamic data*

Direct Access Files

e.g., Multipaging

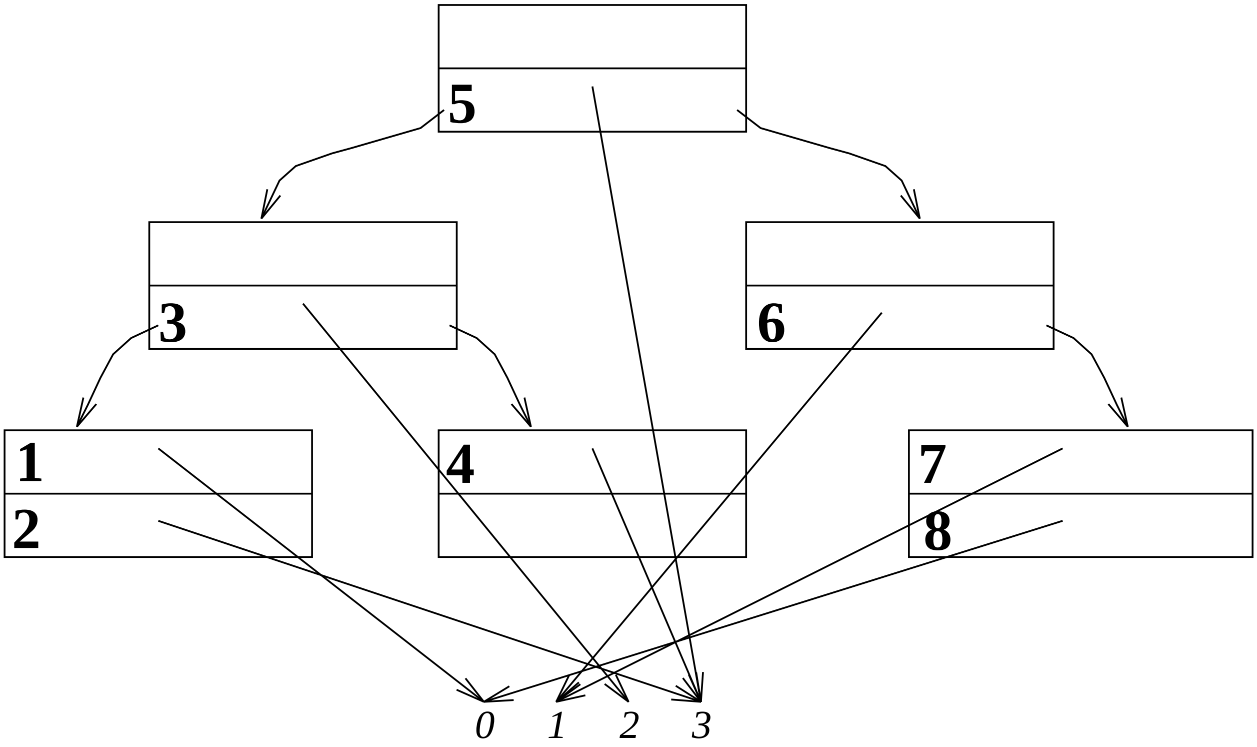
Average cost of a successful search:
1 access.

Order-preserving,
thus good for high activity.

Can be built up dynamically.

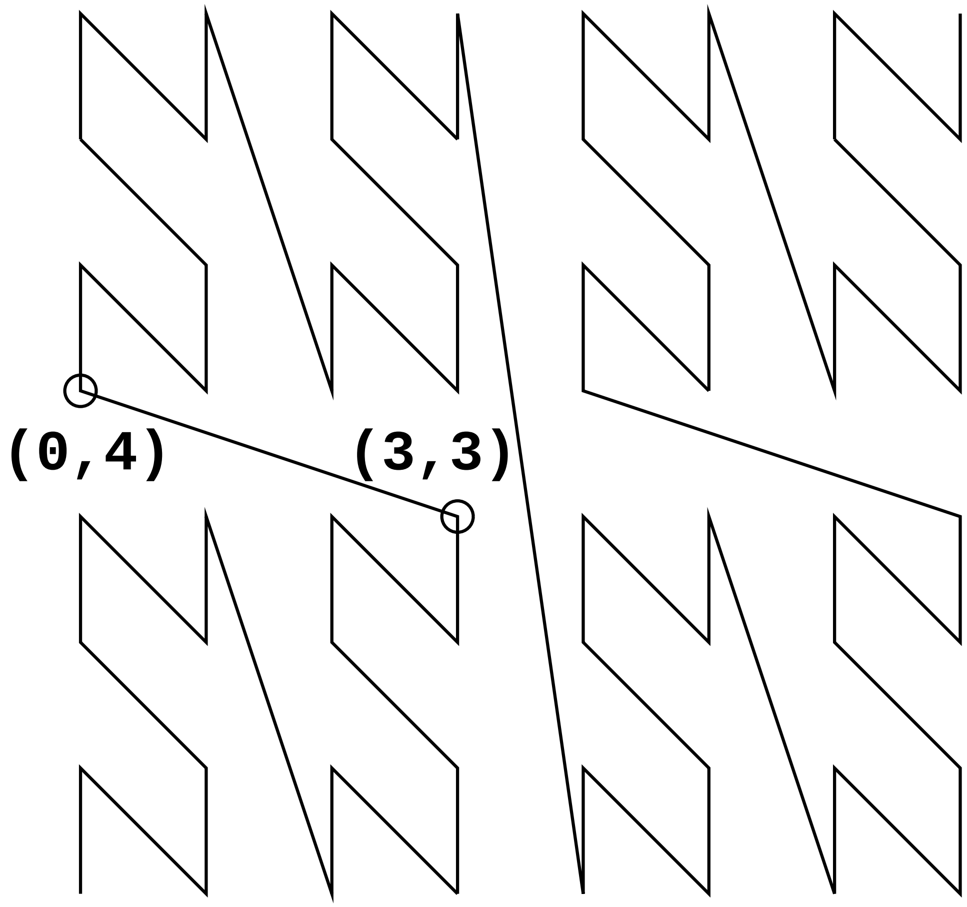
B&O		6	
GNS	1,8		
GTRC			7
L&S	3		
NYC		5	2
PR		4	
	E	H	N

Hybrid Files



B&O	*0*	6	*1*
GNS	1,8		
GTRC			7
L&S	3 *2*		*3*
NYC		5	2
PR		4	
	E	H	N

Z-Ordering



1-dimensional ordering of m-dimensional data

So can use existing structures (e.g., B-tree)

Based on *kd-trie*, or on interleaving of bits:

(3,3)=(0011,0011) shuffles to 0000111 $<$

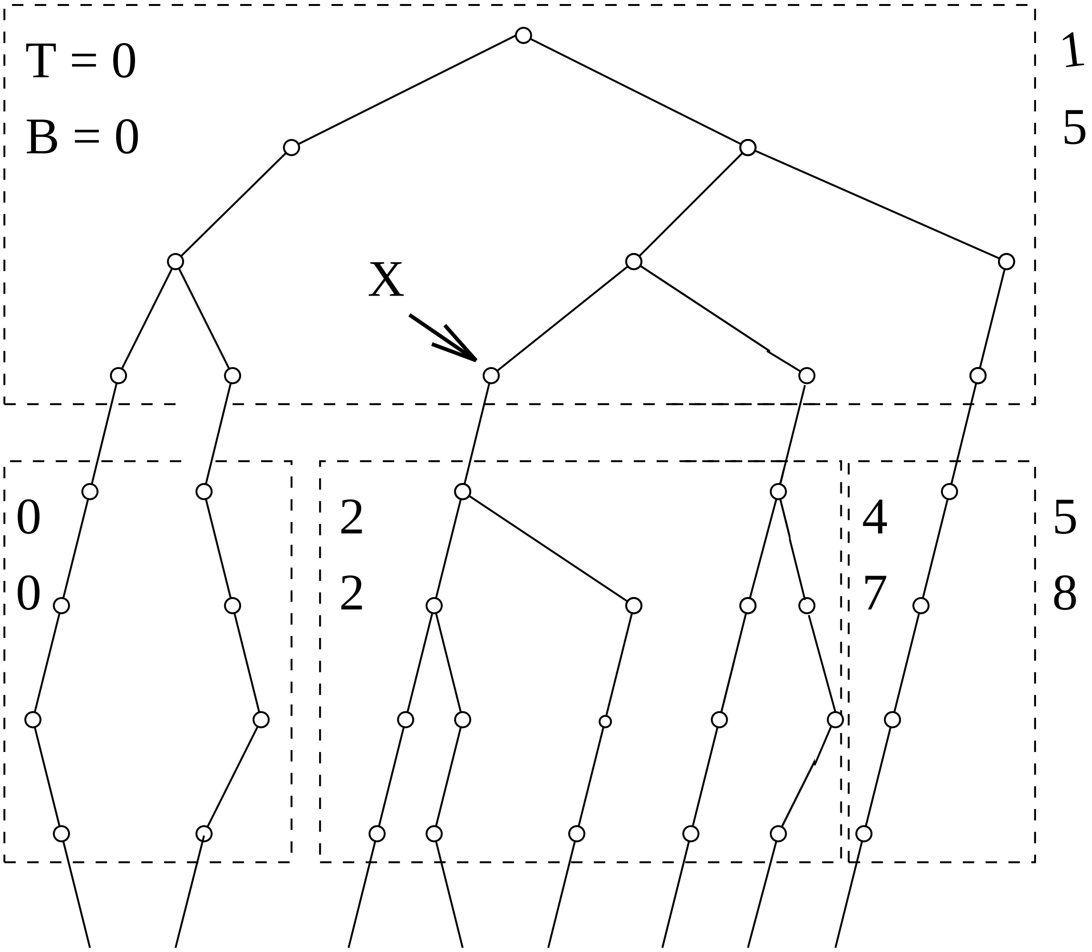
00010000 unshuffles to (0000,0100)=(0,4)

Tries

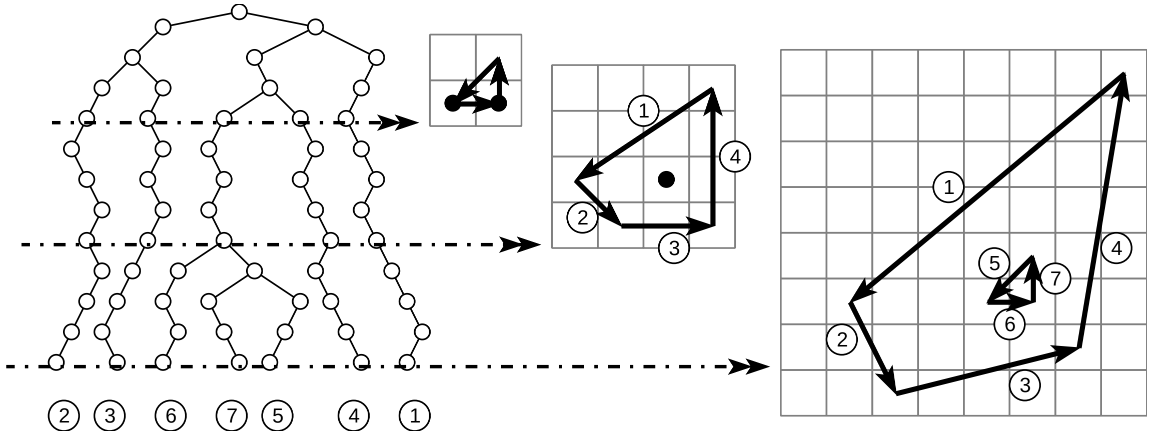
(Digital trees   Information re*trie*val)

Sample data:

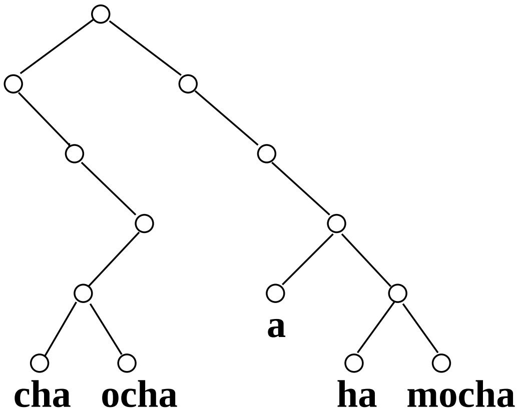
```
00000011
00101100
10000000
10000101
10001000
10100000
10101100
11000000
```



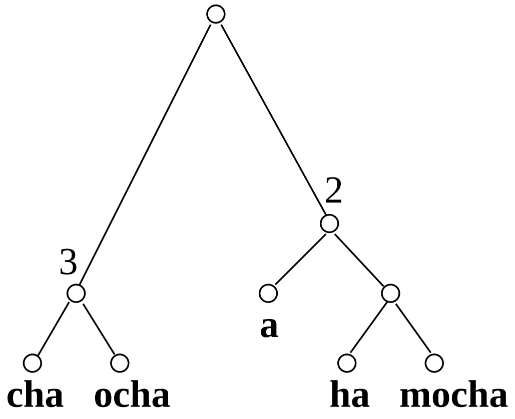
Kd-Tries and Variable Resolution



Truncated Tries and Text Data



1) Truncated Trie



2) PATRICIA Trie

Sample “text”:

`mocha :` 1110110101101111011000111110100011100001

with “starts” every eight bits.