High Performance Data Mining

Vipin Kumar
Computer Science Dept.
University of Minnesota
Minneapolis, MN, USA.
kumar@cs.umn.edu
www.cs.umn.edu/~kumar

Mohammed J. Zaki
Computer Science Dept.
Rensselaer Polytechnic Institute
Troy, NY, USA.
zaki@cs.rpi.edu
www.cs.rpi.edu/~zaki

Tutorial overview

- Overview of KDD and data mining
- Parallel design space
- Classification
- Associations
- Sequences
- Clustering
- Future directions and summary

Overview of data mining

- What is Data Mining/KDD?
- Why is KDD necessary
- The KDD process
- Mining operations and methods
- Core issues in KDD

What is data mining?

The iterative and interactive process of discovering valid, novel, useful, and understandable patterns or models in

Massive databases

What is data mining?

- Valid: generalize to the future
- Novel: what we don't know
- Useful: be able to take some action
- Understandable: leading to insight
- Iterative: takes multiple passes
- Interactive: human in the loop

Data mining goals

- Prediction
  - What?
  - Opaque
- Description
  - Why?
  - Transparent

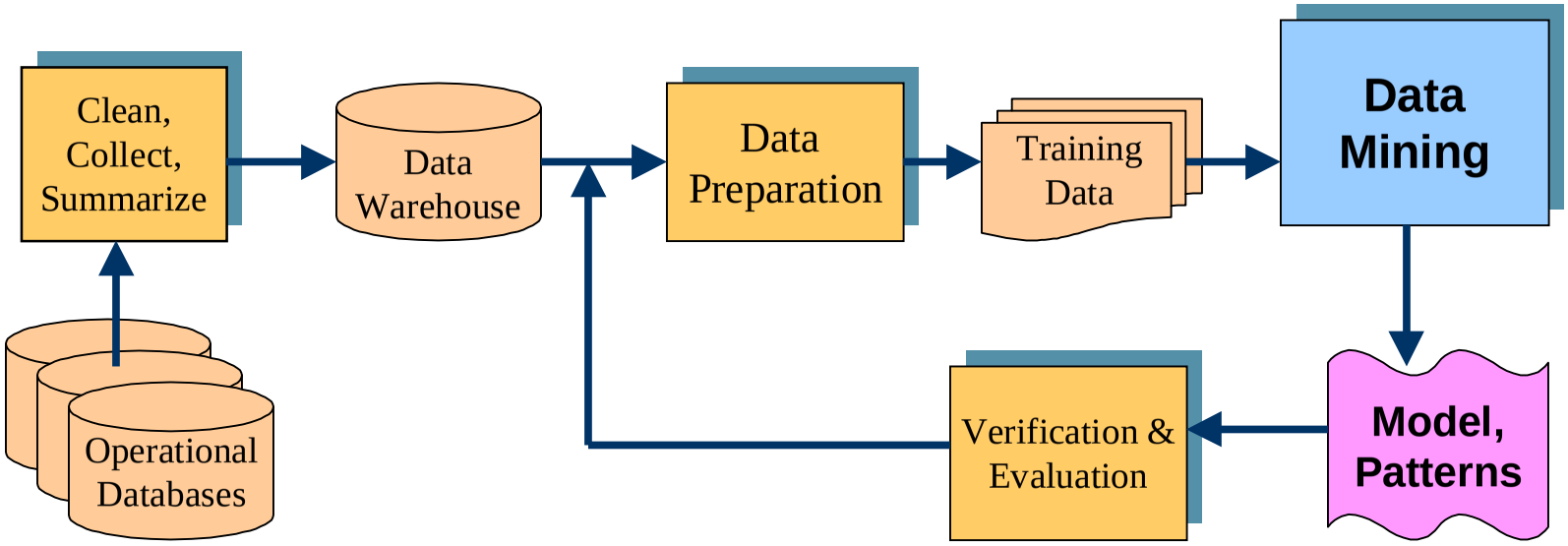
Data mining operations

- Verification driven
  - Validating hypothesis
  - Querying and reporting (spreadsheets, pivot tables)
  - Multidimensional analysis (dimensional summaries); On Line Analytical Processing
  - Statistical analysis

Data mining operations

- Discovery driven
  - Exploratory data analysis
  - Predictive modeling
  - Database segmentation
  - Link analysis
  - Deviation detection

KDD Process


Clean,
Collect,
Summarize
Data
Warehouse
Data
Preparation
Training
Data
Data
Mining
Operational
Databases
Verification &
Evaluation
Model,
Patterns

Data mining process

- Understand application domain
  - Prior knowledge, user goals
- Create target dataset
  - Select data, focus on subsets
- Data cleaning and transformation
  - Remove noise, outliers, missing values
  - Select features, reduce dimensions

Data mining process

- Apply data mining algorithm
  - Associations, sequences, classification, clustering, etc.
- Interpret, evaluate and visualize patterns
  - What's new and interesting?
  - Iterate if needed
- Manage discovered knowledge
  - Close the loop

Why Mine Data? Commercial ViewPoint...

- Lots of data is being collected and warehoused.d
- Computing has become affordable.
- Competitive Pressure is Strong
  - Provide better, customized services for an *edge*.
  - Information is becoming product in its own right.

Why Mine Data? Scientific Viewpoint...

- Data collected and stored at enormous speeds (Gbyte/hour)
  - remote sensor on a satellite
  - telescope scanning the skies
  - microarrays generating gene expression data
  - scientific simulations generating terabytes of data
- Traditional techniques are infeasible for raw data
- Data mining for data reduction..
  - cataloging, classifying, segmenting data
  - Helps scientists in Hypothesis Formation

Origins of Data Mining

- Draws ideas from machine learning/AI, pattern recognition, statistics, database systems, and data visualization.
- Traditional Techniques may be unsuitable
  - Enormity of data
  - High Dimensionality of data
  - Heterogeneous, Distributed nature of data

Data mining methods

- Predictive modeling (classification, regression)
- Segmentation (clustering)
- Dependency modeling (graphical models, density estimation)
- Summarization (associations)
- Change and deviation detection

Data mining techniques

- Association rules: detect sets of attributes that frequently co-occur, and rules among them, e.g. 90% of the people who buy cookies, also buy milk (60% of all grocery shoppers buy both)
- Sequence mining (categorical): discover sequences of events that commonly occur together, .e.g. In a set of DNA sequences ACGTC is followed by GTCA after a gap of 9, with 30% probability

Data mining techniques

- Classification and regression: assign a new data record to one of several predefined categories or classes. Regression deals with predicting real-valued fields. Also called supervised learning.
- Clustering: partition the dataset into subsets or groups such that elements of a group share a common set of properties, with high within group similarity and small inter-group similarity. Also called unsupervised learning.

Data mining techniques

- Similarity search: given a database of objects, and a “query” object, find the object(s) that are within a user-defined distance of the queried object, or find all pairs within some distance of each other.
- Deviation detection: find the record(s) that is (are) the most different from the other records, i.e., find all outliers. These may be thrown away as noise or may be the “interesting” ones.

Data mining techniques

- Many other methods, such as
  - Neural networks
  - Genetic algorithms
  - Hidden Markov models
  - Time series
  - Bayesian networks
  - Soft computing: rough and fuzzy sets

Main challenges for KDD

- Scalability
  - Efficient and sufficient sampling
  - In-memory vs. disk-based processing
  - High performance computing
- Automation
  - Ease of use
  - Using prior knowledge

Tutorial overview

- Overview of KDD and data mining
- Parallel design space
- Classification
- Associations
- Sequences
- Clustering
- Future directions and summary

Speeding up data mining

- Data oriented approach
  - Discretization
  - Feature selection
  - Feature construction (PCA)
  - Sampling
- Methods oriented approach
  - Efficient and scalable algorithms

Speeding up data mining (contd.)

- Methods oriented approach (contd.)
  - Parallel data mining
    - Task or control parallelism
    - Data parallelism
    - Hybrid parallelism
  - Distributed data mining
    - Voting
    - Meta-learning, etc.

Need for Parallel Formulations

- Need to handle very large datasets.
- Memory limitations of sequential computers cause sequential algorithms to make multiple expensive I/O passes over data.
- Need for scalable, efficient (fast) data mining computations
  - gain competitive advantage.
  - Handle larger data for greater accuracy in shorter times.

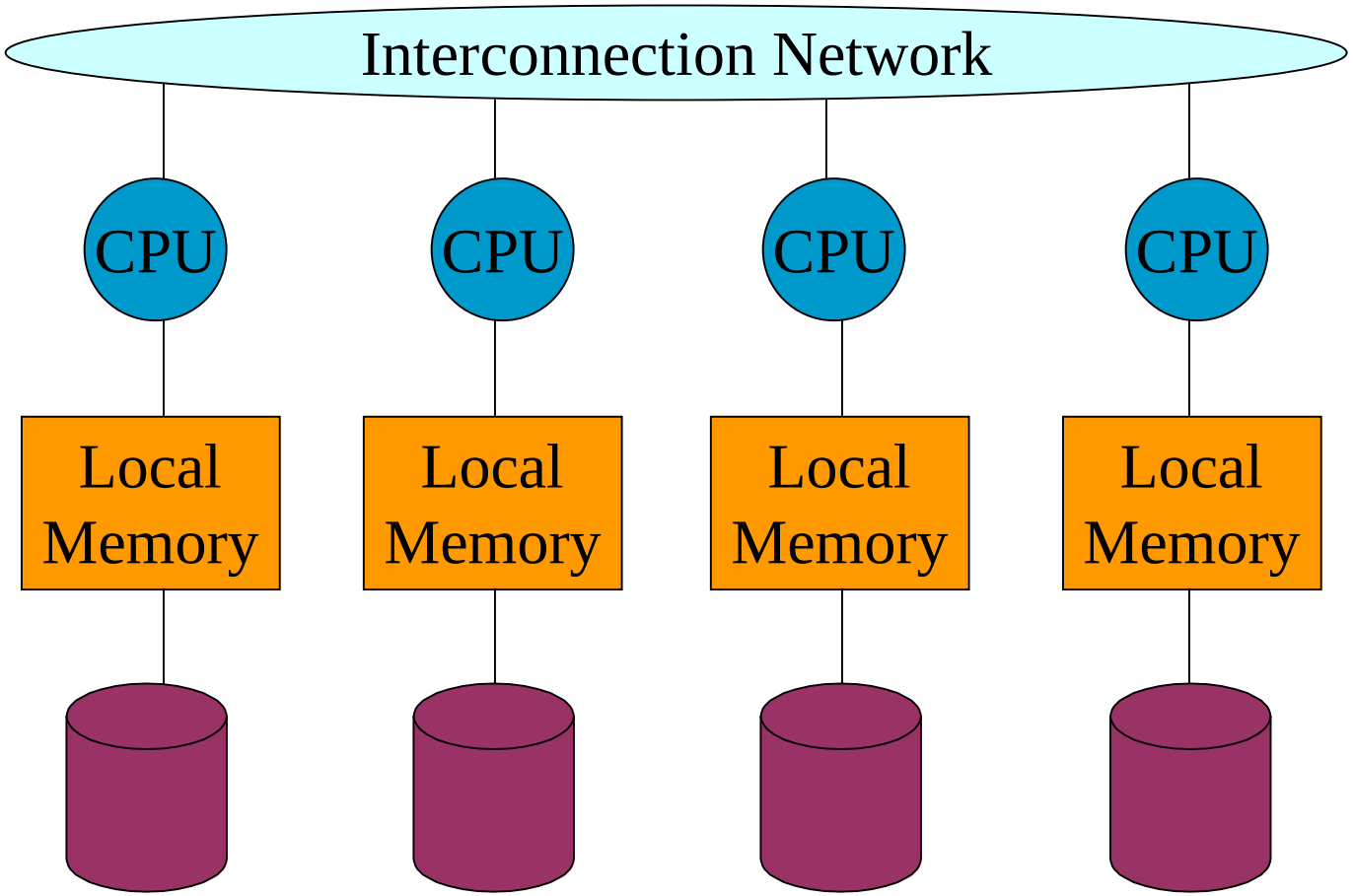
Parallel Design Space

- Parallel architectures
  - Distributed memory
  - Shared disk
  - Shared memory
  - Hybrid cluster of SMPs
- Task and data parallelism
- Static and dynamic load balancing
- Horizontal and vertical data layout
- Data and concept partitioning

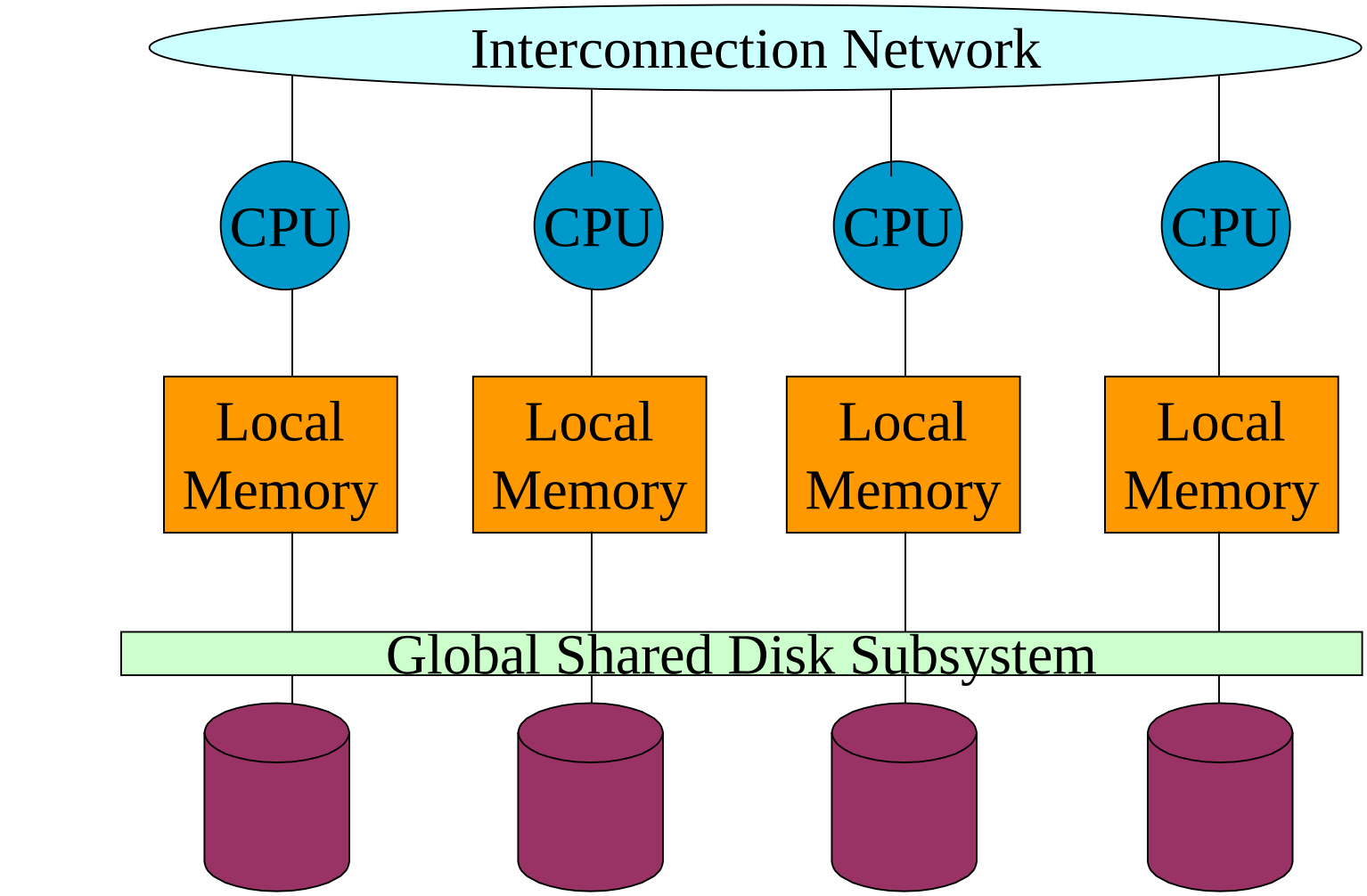
Parallel Hardware

- Distributed-memory machines
  - Each processor has local memory and disk
  - Communication via message-passing
  - Hard to program: explicit data distribution
  - Goal: minimize communication
- Shared-memory machines
  - Shared global address space and disk
  - Communication via shared memory variables
  - Ease of programming
  - Goal: maximize locality, minimize false sharing
- Current trend: Cluster of SMPs

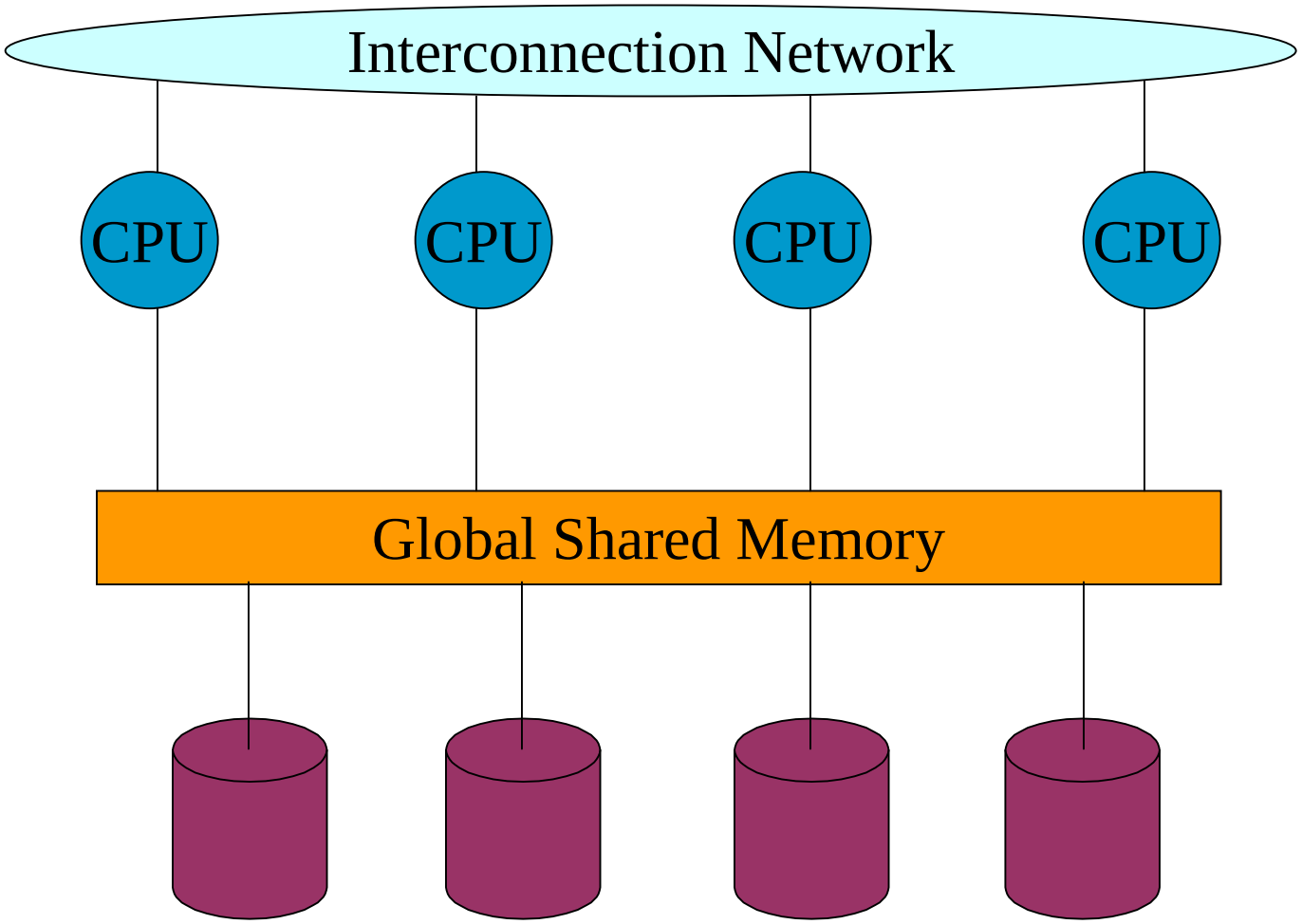
Distributed Memory Architecture (Shared Nothing)



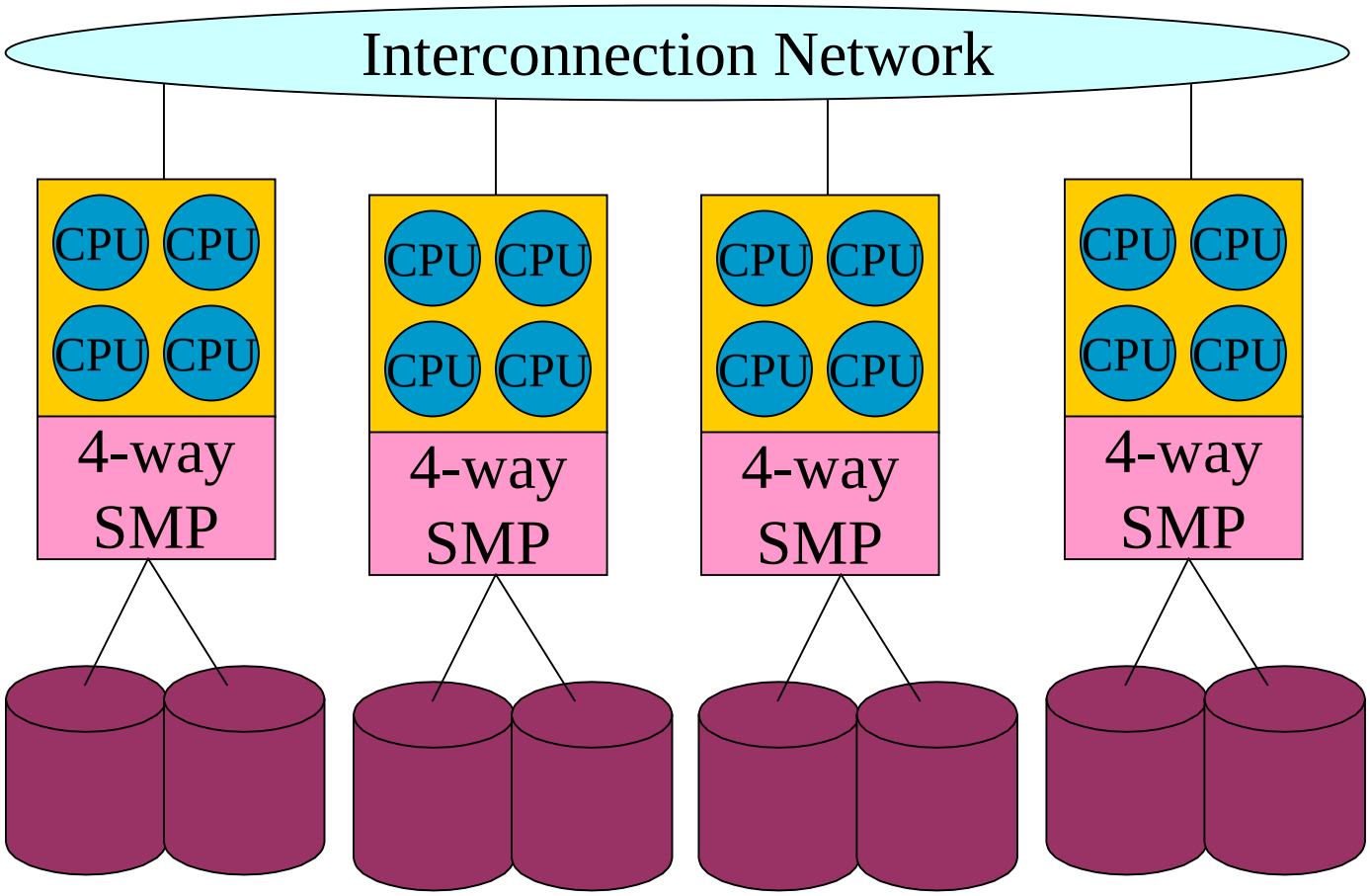
DMM: Shared Disk Architecture



Shared Memory Architecture (Shared Everything)



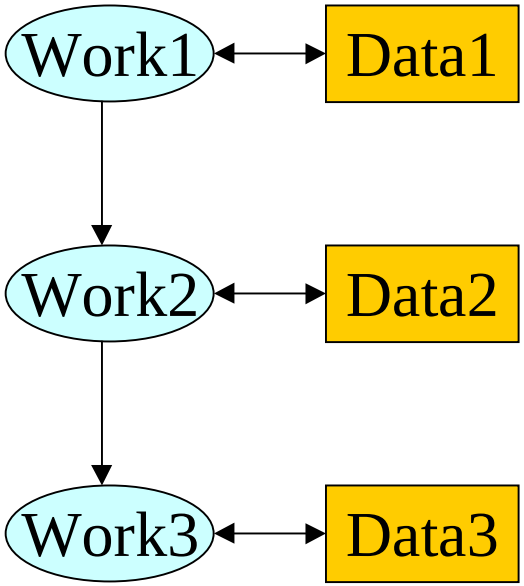
Cluster of SMPs



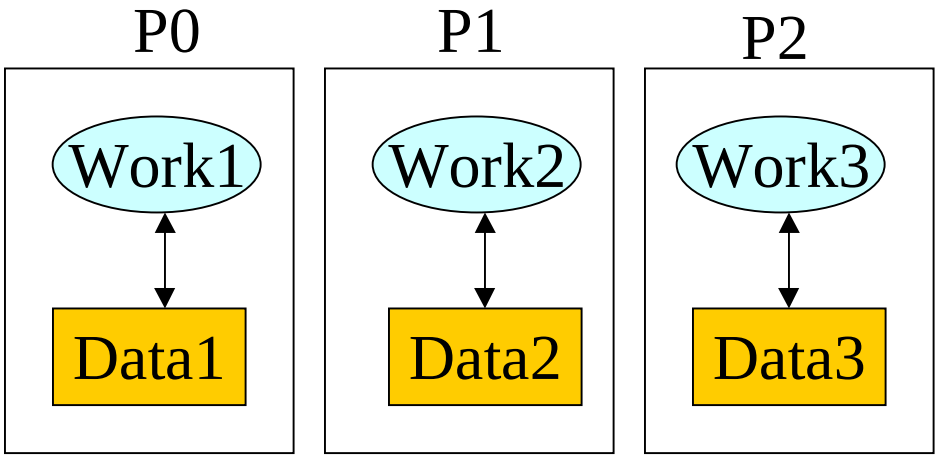
Task vs. Data Parallelism

- Data Parallelism
  - Data partitioned among P processors
  - Each processor performs the same work on local partition
- Task Parallelism
  - Each processor performs different computation
  - Data may be (selectively) replicated or partitioned

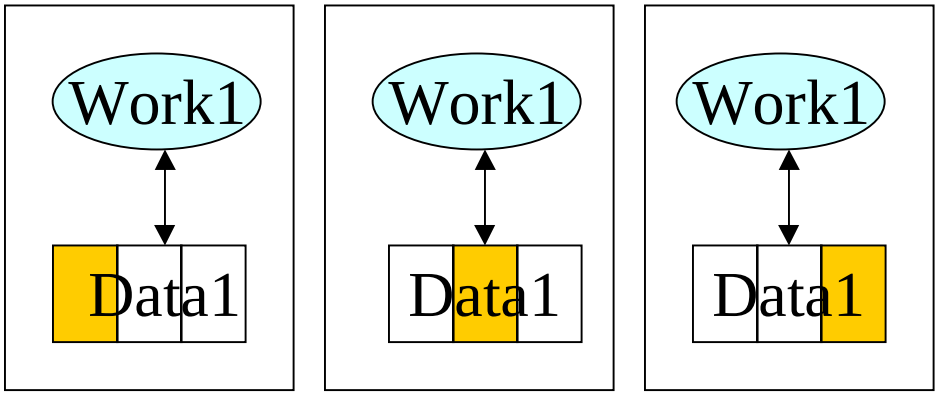
Task vs. Data Parallelism



Uniprocessor



Task Parallelism



Data Parallelism

Static vs. Dynamic Load Balance

- Static Load Balancing
  - Work is initially divided (heuristically)
  - No subsequent computation or data movement
- Dynamic Load Balancing
  - Steal work from heavily loaded processors
  - Reassign to lightly loaded processors
  - Important for irregular computation, heterogeneous environments, etc.

Horizontal/Vertical Data Format

Horizontal

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Vertical

Tid	Refund	Cheat
1	Yes	No
2	No	No
3	No	No
4	Yes	No
5	No	Yes
6	No	No
7	Yes	No
8	No	Yes
9	No	No
10	No	Yes

Tid	Marital Status	Cheat
1	Single	No
2	Married	No
3	Single	No
4	Married	No
5	Divorced	Yes
6	Married	No
7	Divorced	No
8	Single	Yes
9	Married	No
10	Single	Yes

Tid	Taxable Income	Cheat
1	125K	No
2	100K	No
3	70K	No
4	120K	No
5	95K	Yes
6	60K	No
7	220K	No
8	85K	Yes
9	75K	No
10	90K	Yes

Data and concept partitioning

- Shared
  - SMP or shared disk architectures
- Replicated
  - Partially or totally
- Partitioned
  - Round-robin partitioning
  - Hash partitioning
  - Range partitioning

Tutorial overview

- Overview of KDD and data mining
- Parallel design space
- Classification
- Associations
- Sequences
- Clustering
- Future directions and summary

What is Classification?

Classification is the process of assigning new objects to predefined categories or classes

- Given a set of labeled records
- Build a model (decision tree)
- Predict labels for future unlabeled records

Classification learning

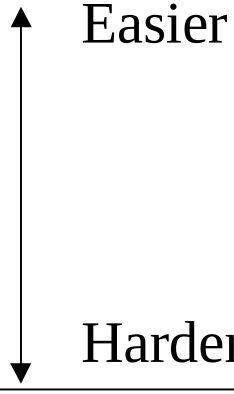
- Supervised learning (labels known)
- Example described in terms of attributes
  - Categorical (unordered symbolic values)
  - Numeric (integers, reals)
- Class (output/predicted attribute): categorical for classification, numeric for regression

Classification learning

- Training set: set of examples, where each example is a feature vector (i.e., a set of (attribute,value) pairs) with its associated class. Model built on this set.
- Test set: a set of examples disjoint from the training set, used for testing the accuracy of a model.

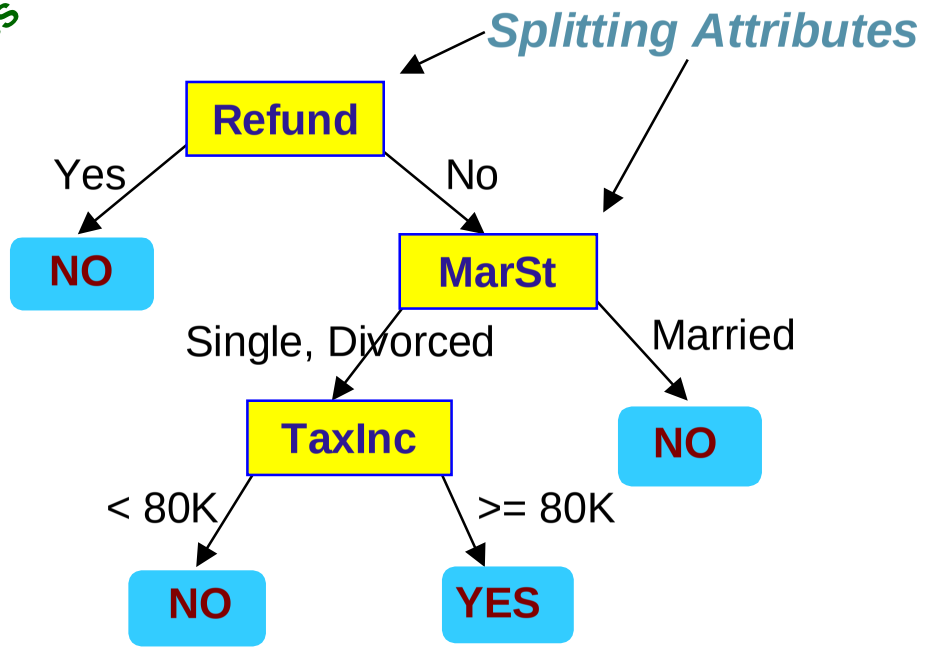
Classification Models

- Some models are better than others
  - Accuracy
  - Understandability
- Models range from easy to understand to incomprehensible
  - Decision trees
  - Rule induction
  - Regression models
  - Neural networks



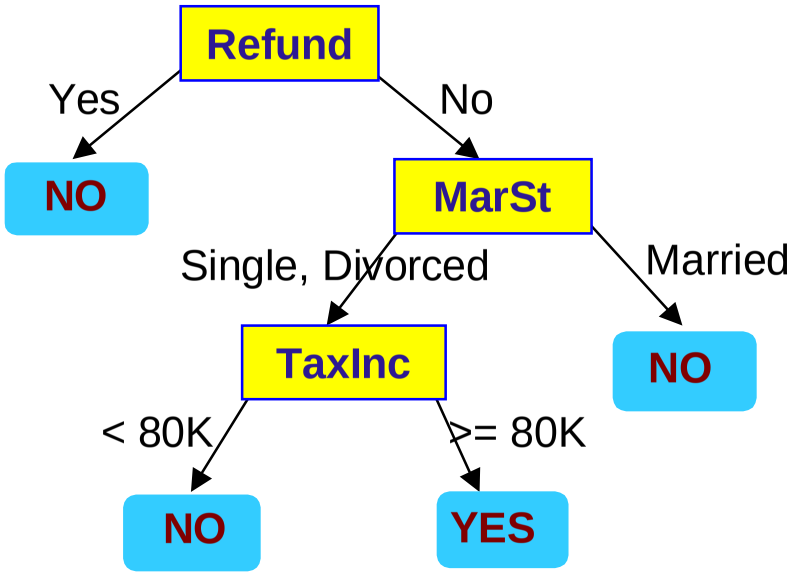
Decision Tree Classification

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



The splitting attribute at a node is determined based on the Gini index.

From Tree to Rules



1) Refund = Yes ⇒ NO

2) Refund = No and
MarSt in {Single, Divorced}
and TaxInc < 80K ⇒ NO

3) Refund = No and
MarSt in {Single, Divorced}
and TaxInc >= 80K ⇒ YES

4) Refund = No and
MarSt in {Married} ⇒ NO

Classification algorithm

- Build tree
  - Start with data at root node
  - Select an attribute and formulate a logical test on attribute
  - Branch on each outcome of the test, and move subset of examples satisfying that outcome to corresponding child node

Classification algorithm

- Recurse on each child node
- Repeat until leaves are “pure”, i.e., have example from a single class, or “nearly pure”, i.e., majority of examples are from the same class

- Prune tree
  - Remove subtrees that do not improve classification accuracy
  - Avoid over-fitting, i.e., training set specific artifacts

Build tree

- Evaluate split-points for all attributes
- Select the “best” point and the “winning” attribute
- Split the data into two
- Breadth/depth-first construction
- CRITICAL STEPS:
  - Formulation of good split tests
  - Selection measure for attributes

How to capture good splits?

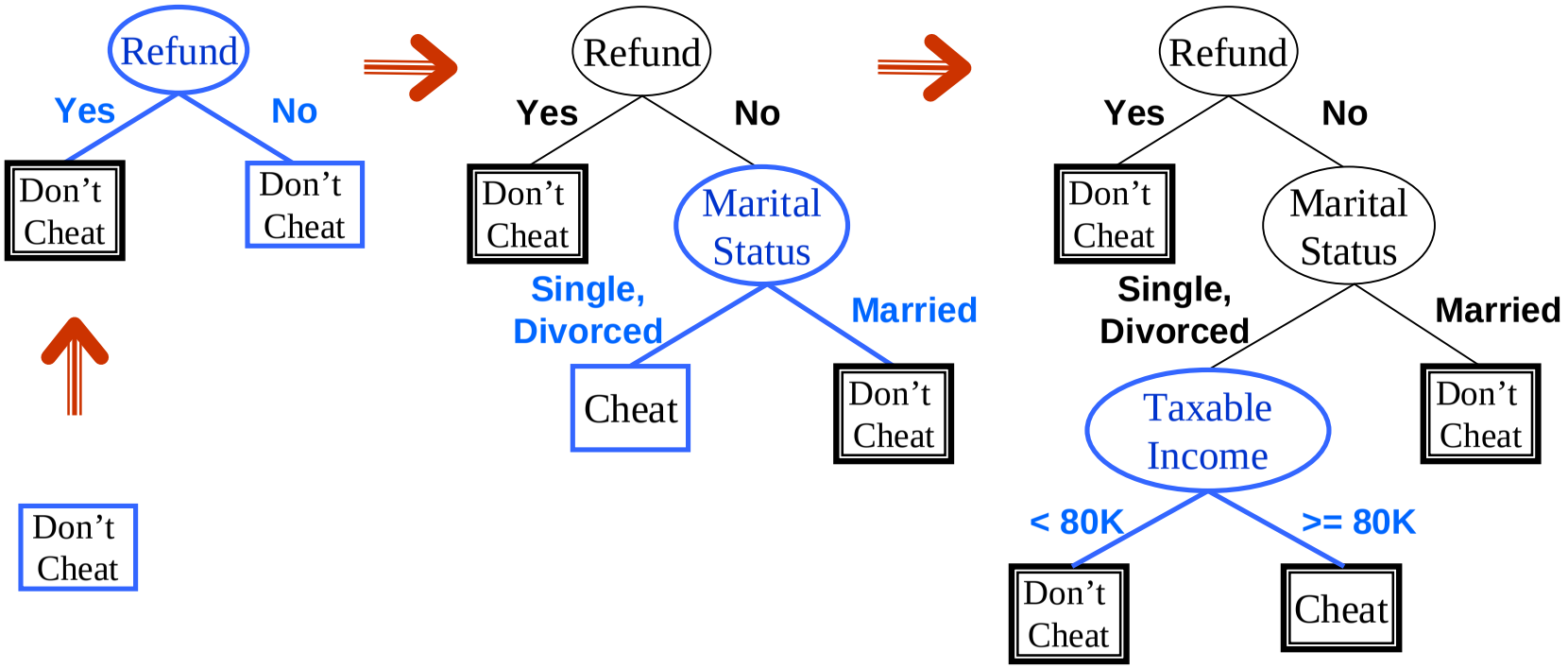
- Occam's razor: Prefer the simplest hypothesis that fits the data
- Minimum message/description length
  - Dataset D
  - Hypotheses H1, H2, …, Hx describing D
  - MML(Hi) = Mlength(Hi)+Mlength(D|Hi)
  - Pick Hk with minimum MML
- Mlength given by Gini index, Gain, etc.

Tree pruning using MDL

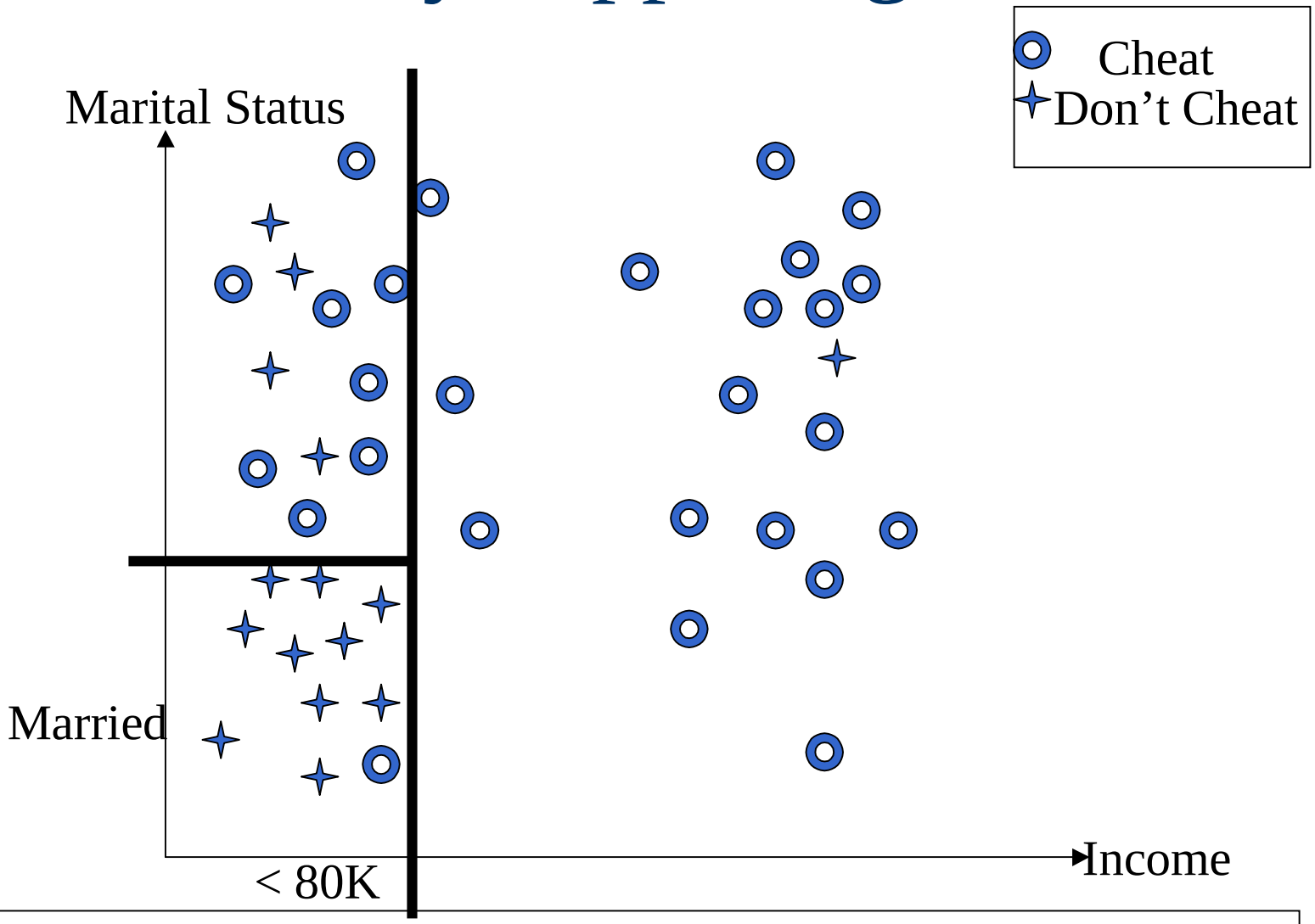
- Data encoding: sum classification errors
- Model encoding:
  - Encode the tree structure
  - Encode the split points
- Pruning: choose smallest length option
  - Convert to leaf
  - Prune left or right child
  - Do nothing

Hunt's Method

- Attributes: Refund (Yes, No), Marital Status (Single, Married, Divorced), Taxable Income
- Class: Cheat, Don't Cheat



What's really happening?



Finding good split points

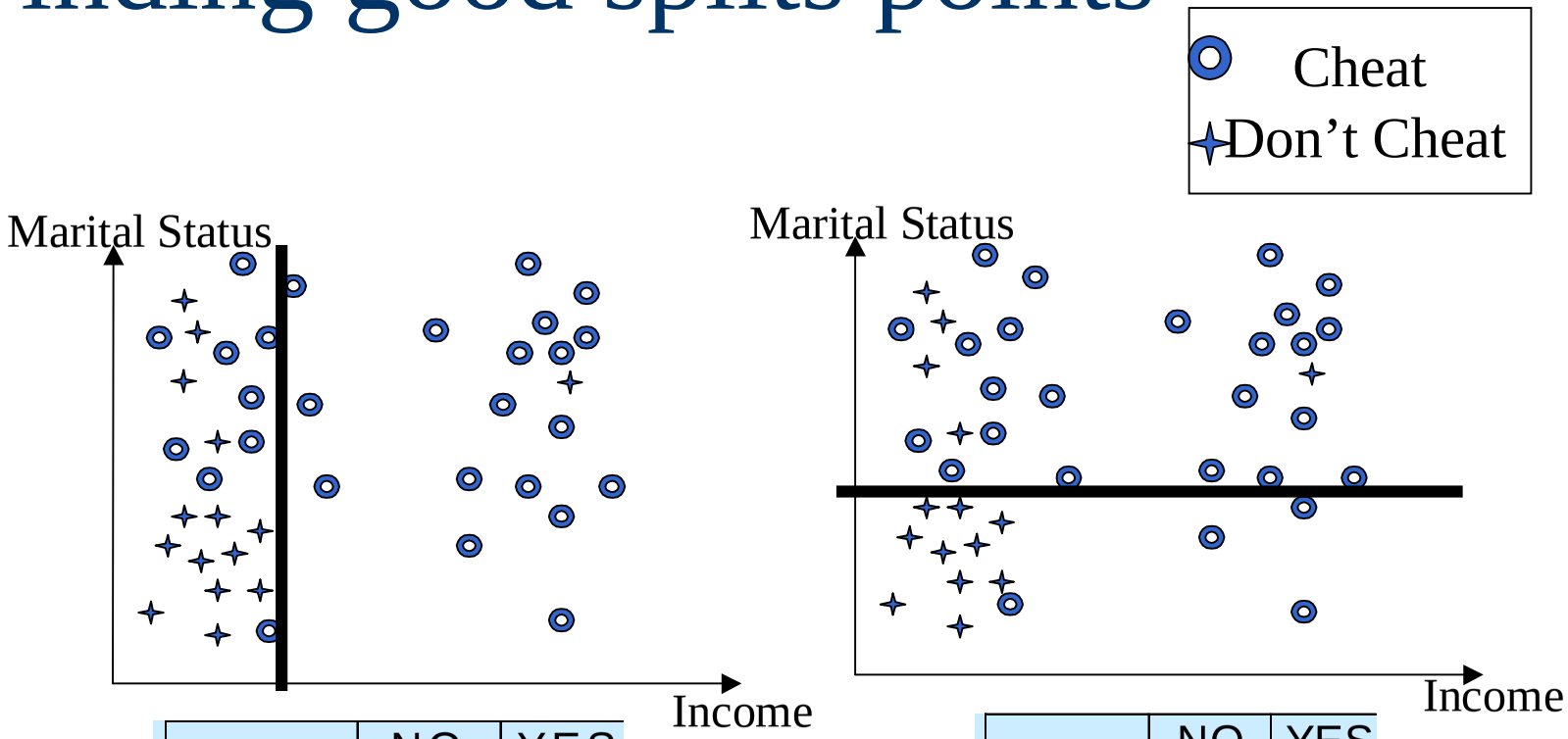
- Use Gini index for partition purity

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

$$Gini(S_1, S_2) = \frac{n_1}{n} Gini(S_1) + \frac{n_2}{n} Gini(S_2)$$

- If S is pure, Gini(S) = 0
- Find split-point with minimum Gini
- Only need class distributions

Finding good splits points



	NO	YES
Left	14	9
Right	1	18

Gini(split) = 0.31

	NO	YES
Top	5	23
Bottom	10	4

Gini(split) = 0.34

Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Taxable Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	230

	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

C4.5

- Simple depth-first construction.
- Sorts Continuous Attributes at each node.
- Needs entire data to fit in memory.
- Unsuitable for Large Datasets.
  - Needs out-of-core sorting.
- Classification Accuracy shown to improve when *entire* datasets are used!

SPRINT [Shafer,Agrawal,Mehta]

Attribute Lists:

Tid	Refund	Cheat
1	Yes	**No**
2	No	**No**
3	No	**No**
4	Yes	**No**
5	No	**Yes**
6	No	**No**
7	Yes	**No**
8	No	**Yes**
9	No	**No**
10	No	**Yes**

Tid	Marital Status	Cheat
1	Single	**No**
2	Married	**No**
3	Single	**No**
4	Married	**No**
5	Divorced	**Yes**
6	Married	**No**
7	Divorced	**No**
8	Single	**Yes**
9	Married	**No**
10	Single	**Yes**

Tid	Taxable Income	Cheat
1	125K	**No**
2	100K	**No**
3	70K	**No**
4	120K	**No**
5	95K	**Yes**
6	60K	**No**
7	220K	**No**
8	85K	**Yes**
9	75K	**No**
10	90K	**Yes**

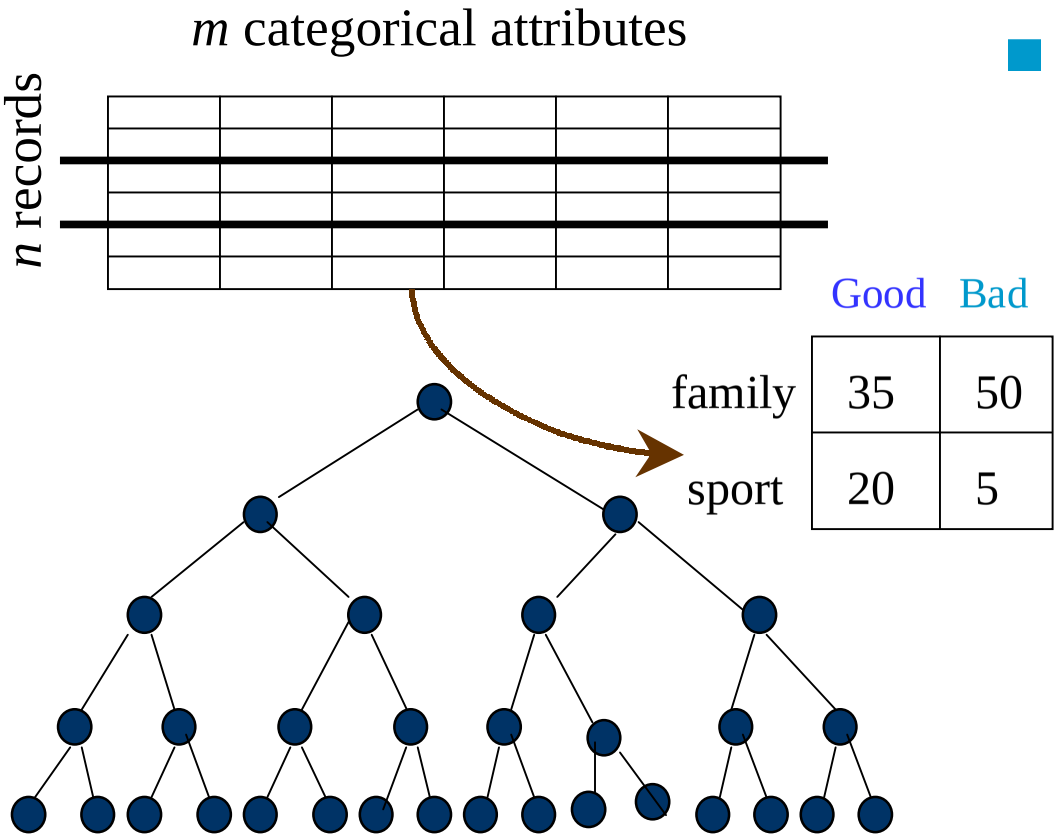
SPRINT (contd.)

- The arrays of the continuous attributes are pre-sorted.
  - The sorted order is maintained during each split.
- The classification tree is grown in a breadth-first fashion.
- Class information is clubbed with each attribute list.
- Attribute lists are physically split among nodes.
- Split determining phase is just a linear scan of lists at each node.
- Hashing scheme used in splitting phase.
  - tids of the splitting attribute are hashed with the tree node as the key.
    - lookup table
  - remaining attribute arrays are split by querying this hash structure.

SPRINT Disadvantages

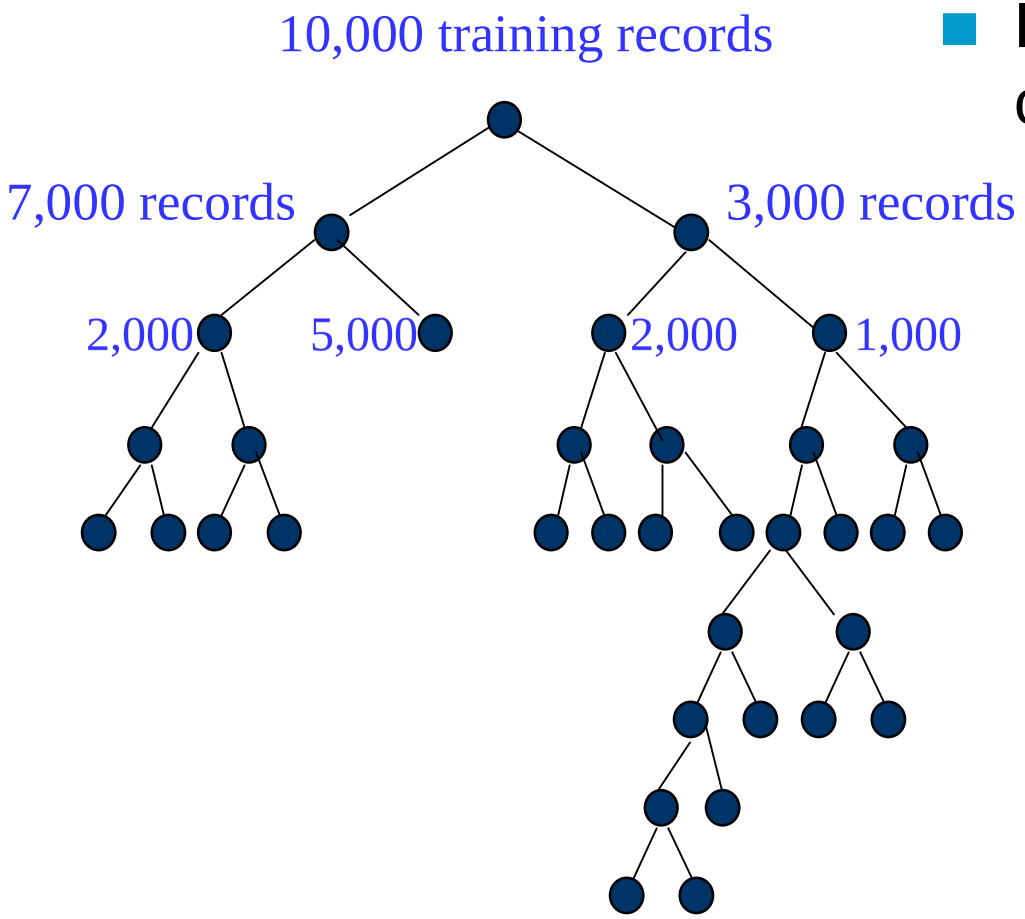
- Size of hash table is O(N) for top levels of the tree.
- If hash table does not fit in memory (mostly true for large datasets), then build in parts so that each part fits.
  - Multiple expensive I/O passes over the entire dataset.

Constructing a Decision Tree in Parallel



- Partitioning of data only
  - global reduction per node is required
  - large number of classification tree nodes gives high communication cost

Constructing a Decision Tree in Parallel

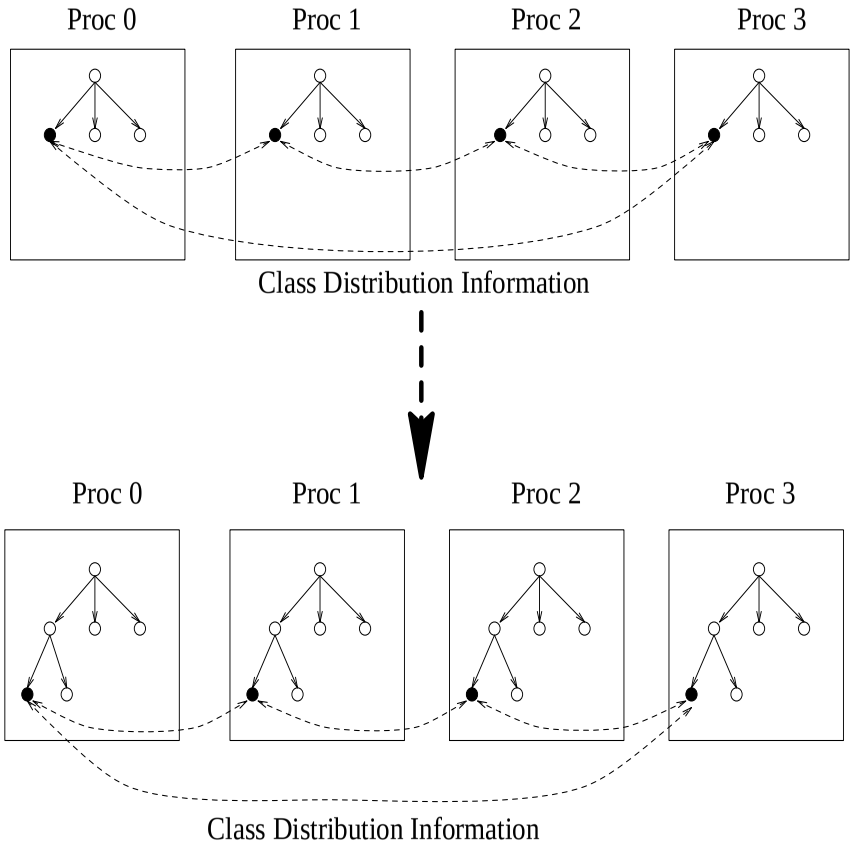


- Partitioning of classification tree nodes
  - natural concurrency
  - load imbalance as the amount of work associated with each node varies
  - child nodes use the same data as used by parent node
    - loss of locality
    - high data movement cost

Challenges in Constructing Parallel Classifier

- Partitioning of data only
  - large number of classification tree nodes gives high communication cost
- Partitioning of classification tree nodes
  - natural concurrency
  - load imbalance as the amount of work associated with each node varies
  - child nodes use the same data as used by parent node
    - loss of locality
    - high data movement cost
- How do we efficiently perform the computation in parallel?

Synchronous Tree Construction Approach



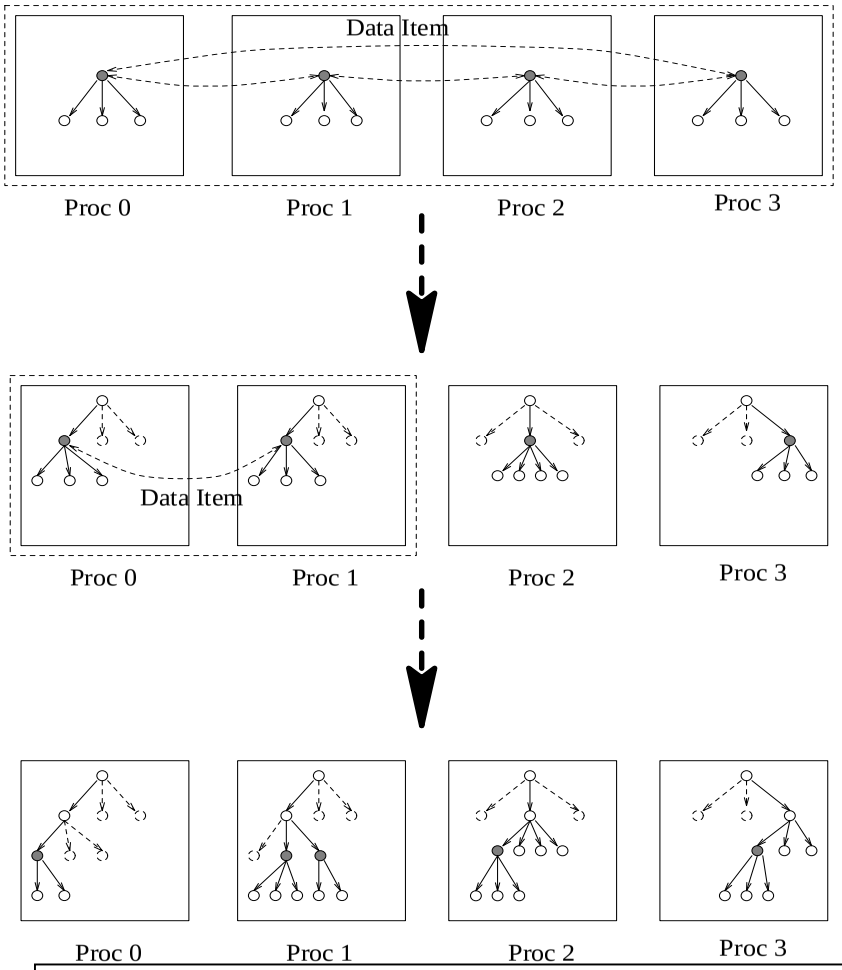
Partition Data Across Processors

\+ No data movement is required

– Load imbalance
- can be eliminated by breadth-first expansion

– High communication cost
- becomes too high in lower parts of the tree

Partitioned Tree Construction Approach



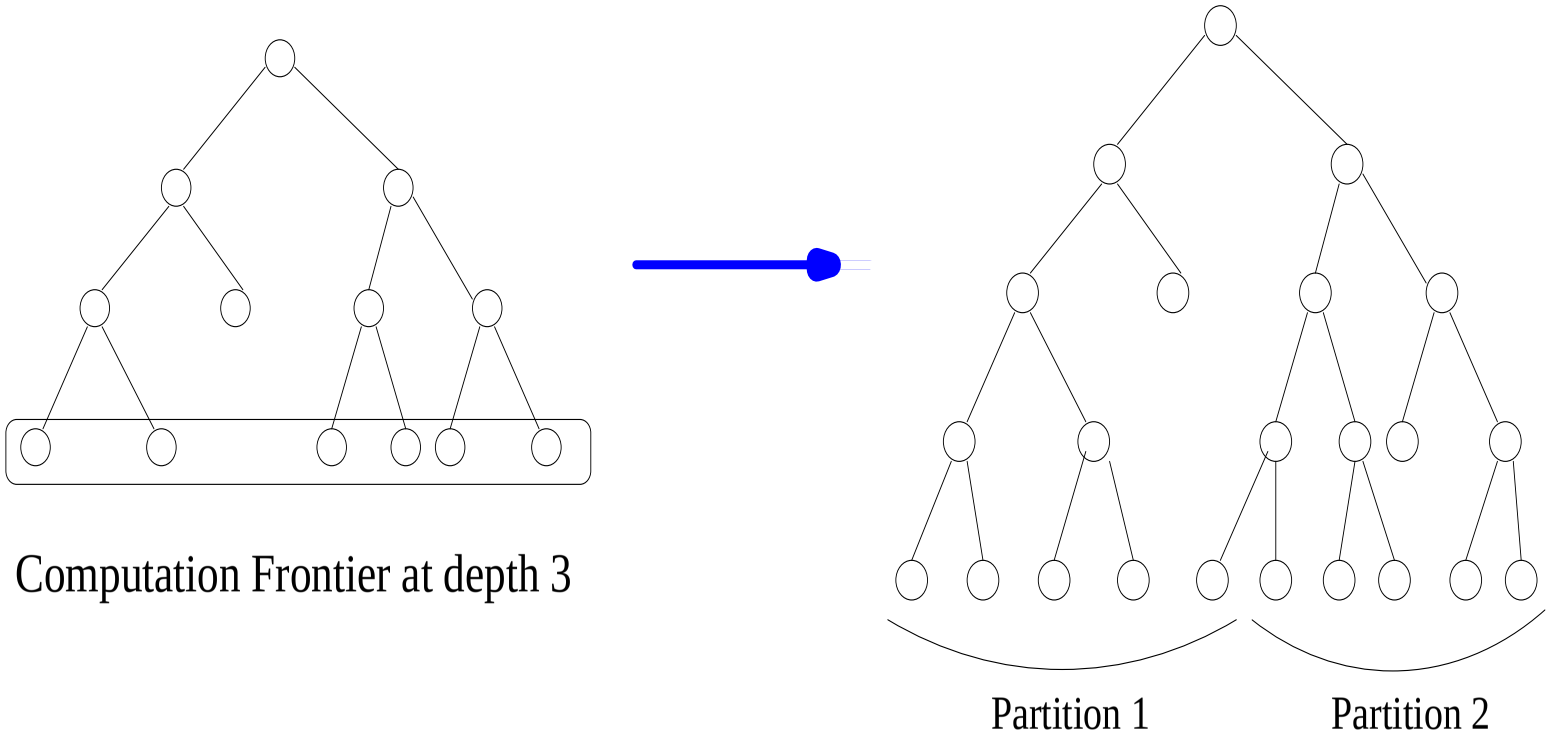
Partition Data *and* Nodes

\+ Highly concurrent

\- High communication cost due to excessive data movements

\- Load imbalance

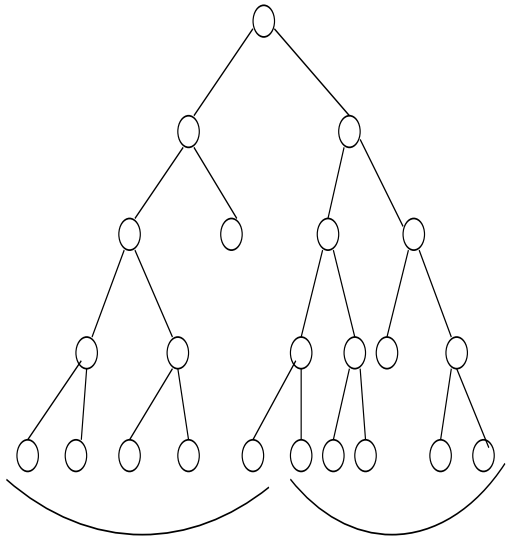
Hybrid Parallel Formulation



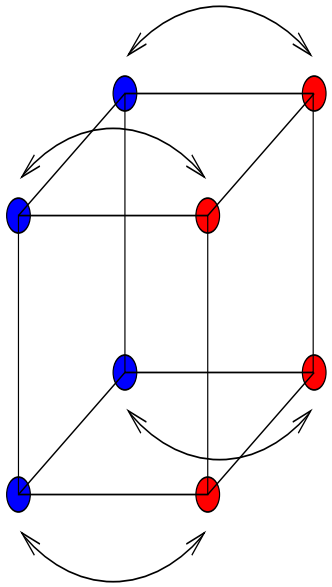
Synchronous Tree Construction Approach

Partitioned Tree Construction Approach

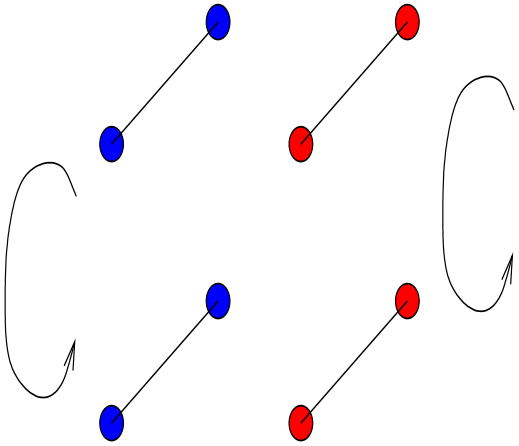
Load Balancing



Partition 1 Partition 2



Step 1: exchange data between processor partitions



Step 2: load balance within processor partitions

Splitting Criterion

- Switch to Partitioned Tree Construction when

$$\sum Communication\ Cost \geq Moving\ Cost + Load\ \ Balancing$$

- Splitting criterion ensures

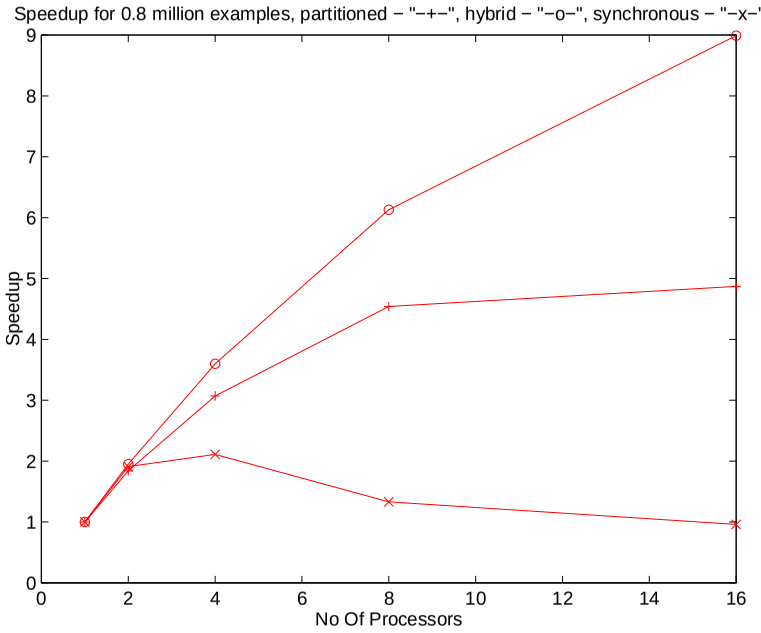
$$Communication\ Cost \leq 2 * Communication\ Cost_{optimal}$$

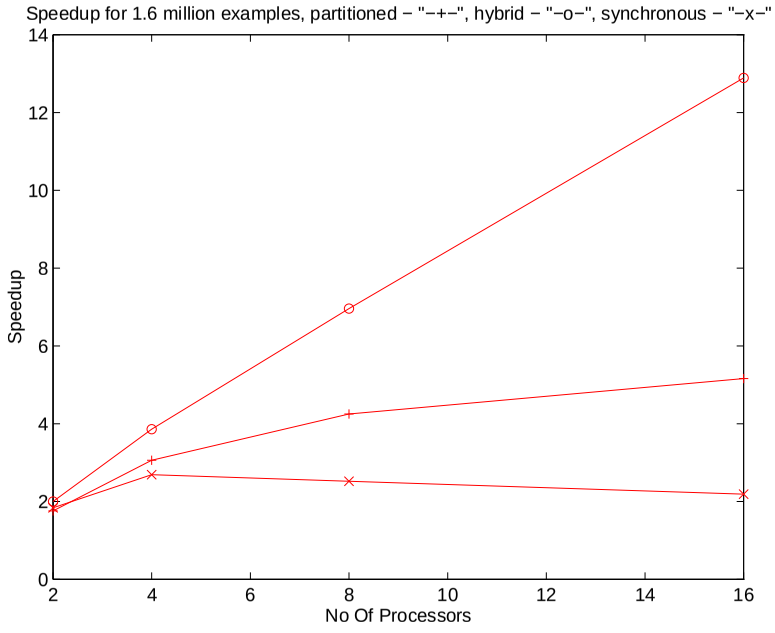
  – G. Karypis and V. Kumar, IEEE Transactions on Parallel and Distributed Systems, Oct. 1994

Experimental Results

- Data set
  - function 2 data set discussed in SLIQ paper (Mehta, Agrawal and Rissanen, EDBT'96)
  - 2 class labels, 3 categorical and 6 continuous attributes
- IBM SP2 with 128 processors
  - 66.7 MHz CPU with 256 MB real memory
  - AIX version 4
  - high performance switch

Speedup Comparison of the Three Parallel Algorithms



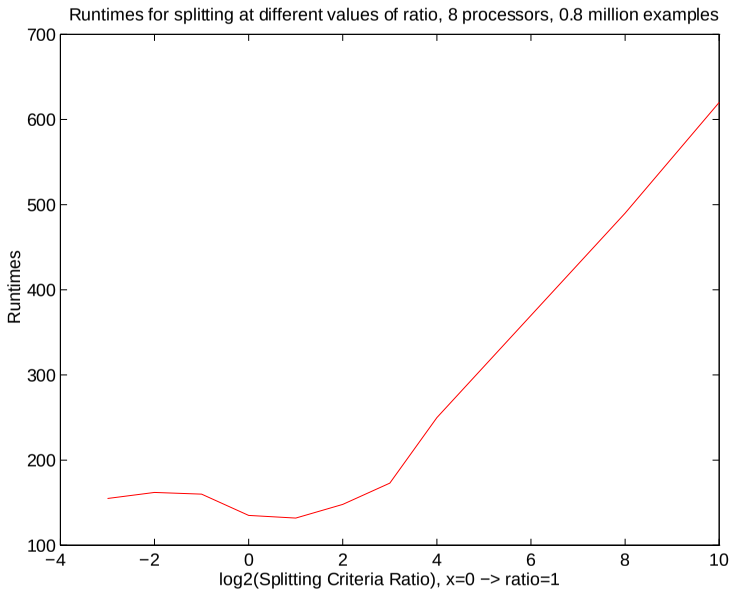


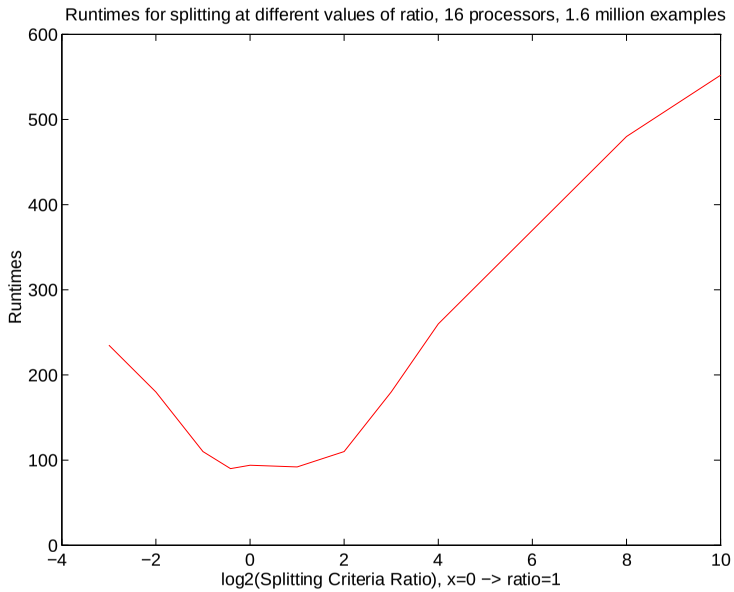
0.8 million examples

1.6 million examples

Splitting Criterion Verification in the Hybrid Algorithm

$$Splitting\ Criterion\ Ratio = \frac{\sum Communication\ Cost}{Moving\ Cost + Load\ Balancing}$$

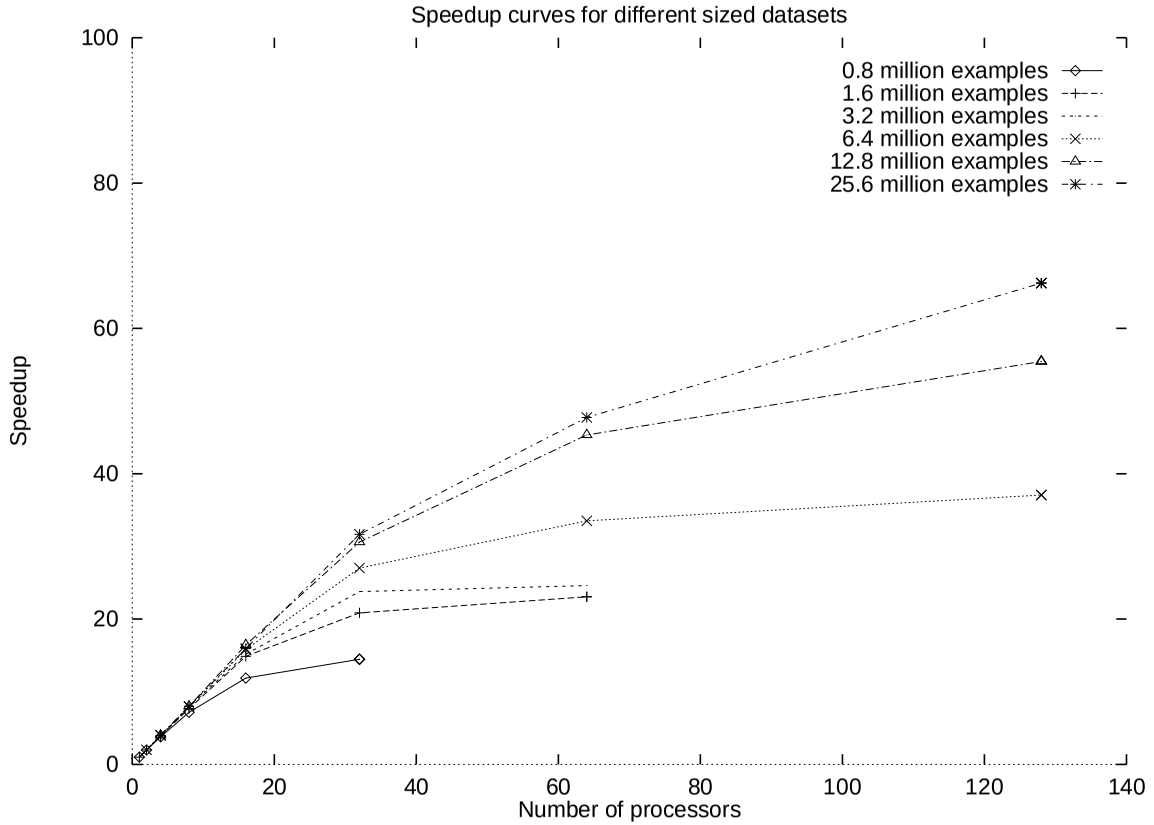




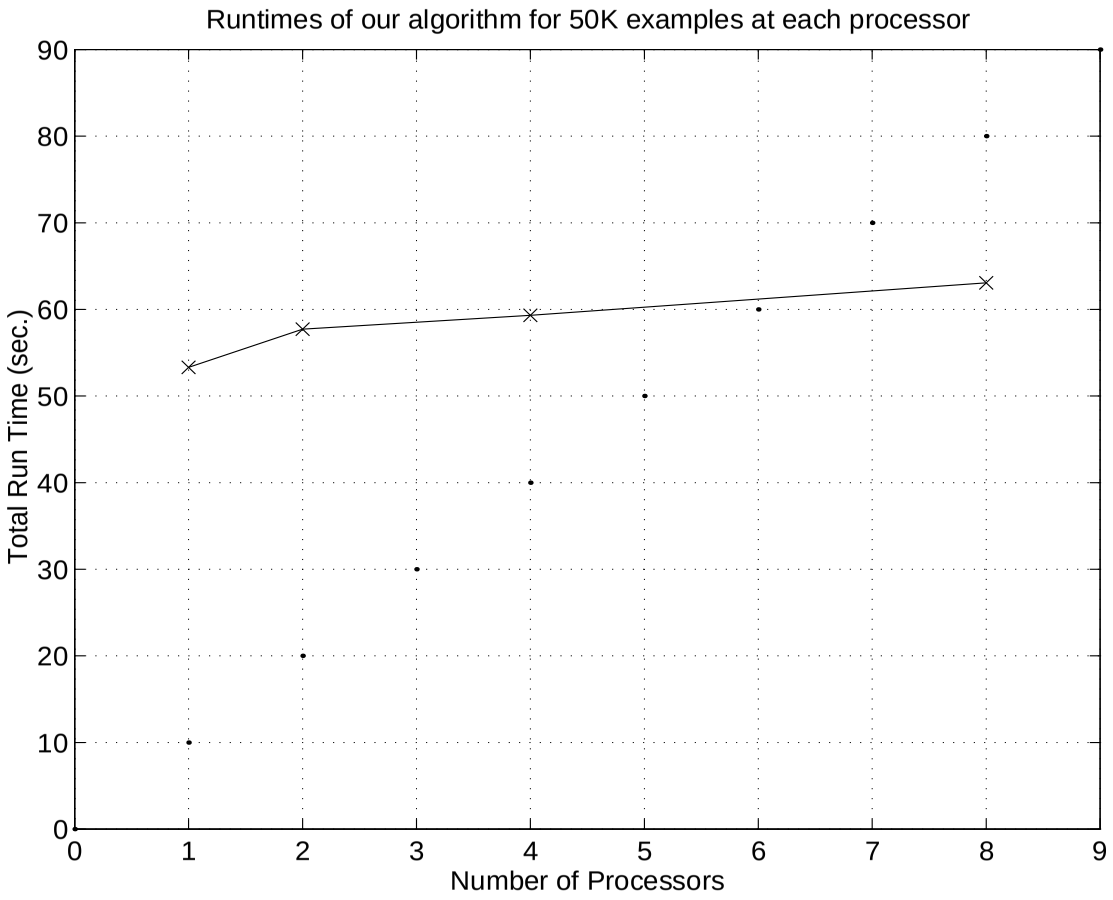
0.8 million examples on 8 processors

1.6 million examples on 16 processors

Speedup of the Hybrid Algorithm with Different Size Data Sets



Scaleup of the Hybrid Algorithm



Summary of Algorithms for Categorical Attributes

- Synchronous Tree Construction Approach
  - no data movement required
  - high communication cost as tree becomes bushy
- Partitioned Tree Construction Approach
  - processors work independently once partitioned completely
  - load imbalance and high cost of data movement
- Hybrid Algorithm
  - combines good features of two approaches
  - adapts dynamically according to the size and shape of trees

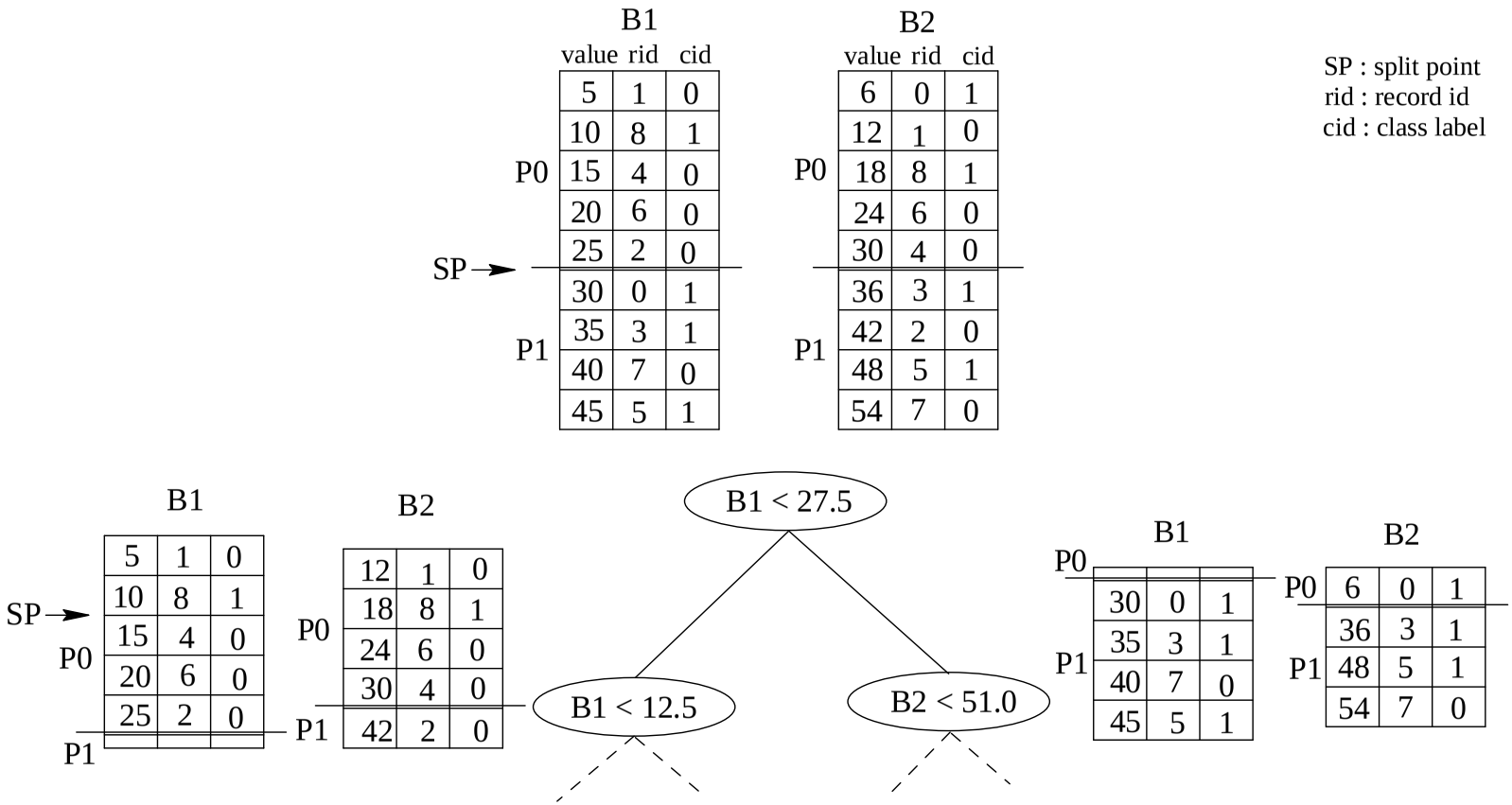
Handling Continuous Attributes

- Sort continuous attributes at each node of the tree (as in C4.5). Expensive, hence Undesirable!
- Discretize continuous attributes
  - *CLOUDS* (Alsabti, Ranka, and Singh, 1998)
  - *SPEC* (Srivastava, Han, Kumar, and Singh, 1997)
- Use a pre-sorted list for each continuous attributes
  - *SPRINT* (Shafer, Agrawal, and Mehta, VLDB'96)
  - *ScalParC* (Joshi, Karypis, and Kumar, IPPS'98)

Design Approach

- Goal: Scalability in *both* Runtime and Memory Requirements.
- Parallelization overhead: $T_o = p\ T_p - T_s$
- $T_o\ \alpha\ O(T_s)$ for scalability. Per processor overhead should not exceed $O(T_s/p)$.
- Two components of T_o:
  - Load Imbalance.
  - Communication time.

Load Balancing

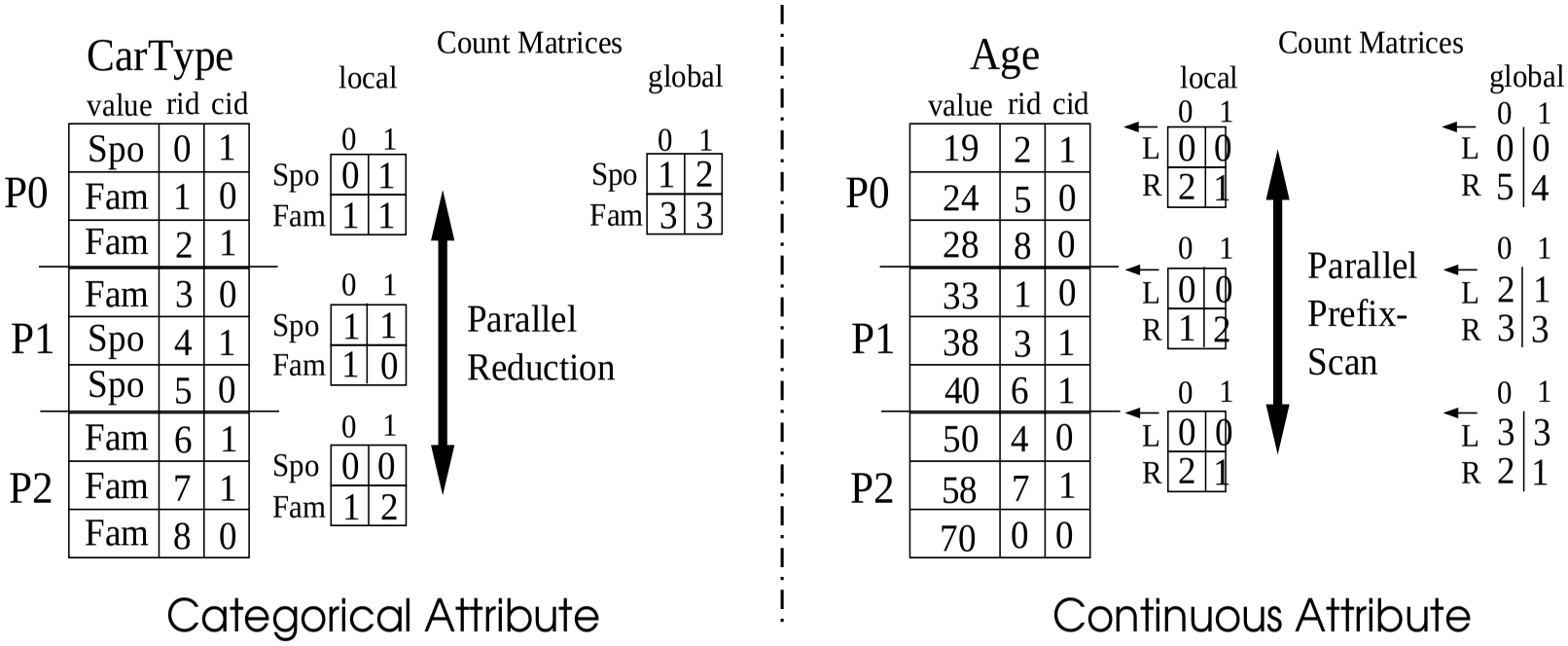


Parallel SPRINT

- Attribute lists are partitioned among processors
  - Each processor gets *N/p* records of each attribute list
- Attribute lists of continuous attributes are pre-sorted
- Split Determining Phase
  - Categorical attributes
    - Local construction of count matrices followed by a reduction to add them up
  - Continuous attributes
    - Prefix-scan followed by local Gini computations followed by a gini index reduction to find the maximum point

Parallelizing Split Determination Phase

- Easy.



Parallelizing Splitting Phase..

- SPRINT Approach:
  - Builds hash table on *all processors* by an all-to-all broadcast operation.
  - So, O(N) data sent & received by *each processor* --> Not Scalable in Runtime as well as Memory Requirements.

Scalable Parallel Formulation of Splitting Phase [ScalParC]

- **Challenging!!**
  - Requirement for *consistent* splitting, given the relatively random sorted order among attributes, makes this phase interesting.
- Four cases:

Splitting Attribute Type	Non-Splitting Attribute Type
Categorical	Categorical
Categorical	Continuous
Continuous	Categorical
Continuous	Continuous

Categorical/Categorical

- Splitting a categorical attribute list requires access exactly to those entries of the hash-table that are created by the same processor

	tid	attr1	Class
P1	1	A	0
	2	B	0
	3	A	1
P2	4	A	1
	5	B	0
	6	A	1
P3	7	A	0
	8	B	0
	9	B	1

Node
L
R
L
L
R
L
L
R
R

	tid	Attr2	Class
P1	1	Y	1
	2	N	0
	3	N	0
P2	4	Y	1
	5	Y	1
	6	N	0
P3	7	Y	0
	8	N	1
	9	N	0

Continuous/Continuous

	tid	attr1	Class
P1	1	2454	0
	3	4942	0
	9	6491	1
P2	5	9476	1
	6	9767	0
	2	9881	1
P3	4	12614	0
	7	13683	0
	8	15303	1

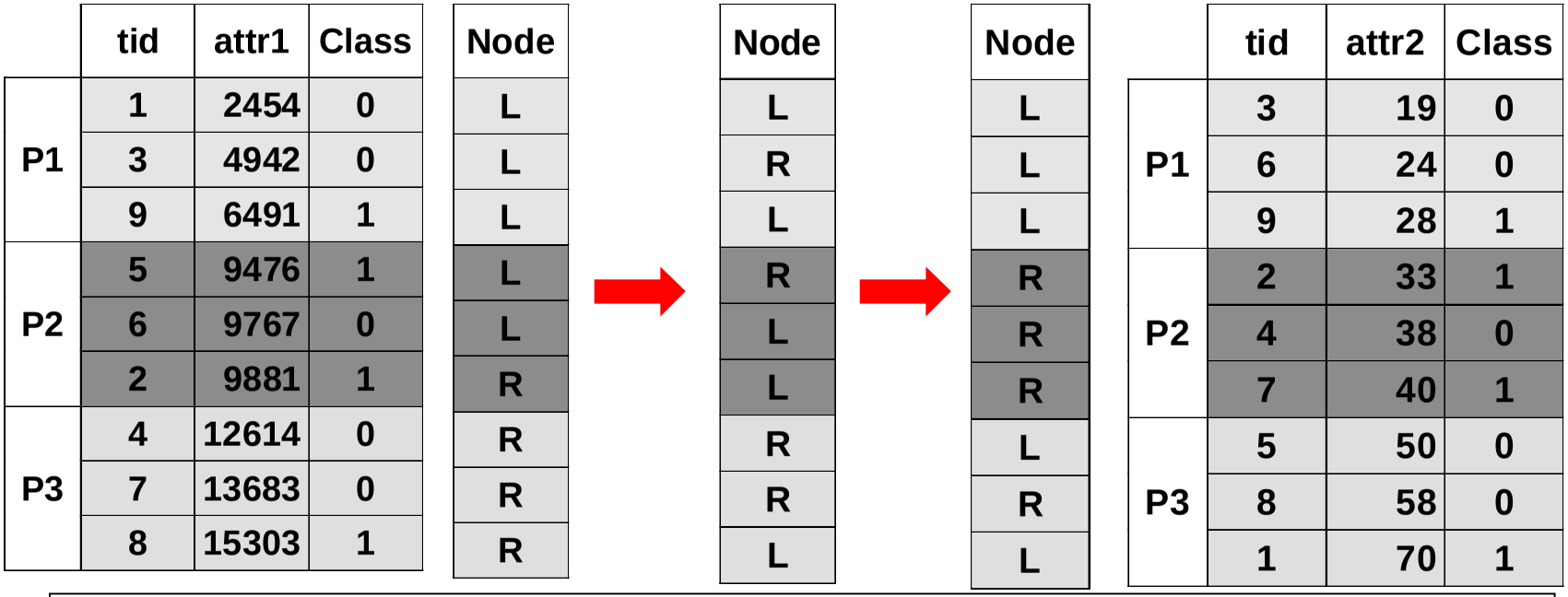
Node
L
L
L
L
L
R
R
R
R

	tid	attr2	Class
P1	3	19	0
	6	24	0
	9	28	1
P2	2	33	1
	4	38	0
	7	40	1
P3	5	50	0
	8	58	0
	1	70	1

- The hash-table entries required by a processor are generated by other processors
  - Need to develop a scheme to transfer these entries to the processors that need them

Getting the required entries of the hash table

- The required entries are transferred into two steps
  - From splitting attribute order to *tid* sorted order
  - From *tid* sorted order to attribute list order



	tid	attr1	Class
P1	1	2454	0
	3	4942	0
	9	6491	1
P2	5	9476	1
	6	9767	0
	2	9881	1
P3	4	12614	0
	7	13683	0
	8	15303	1

Node
L
L
L
L
L
R
R
R
R

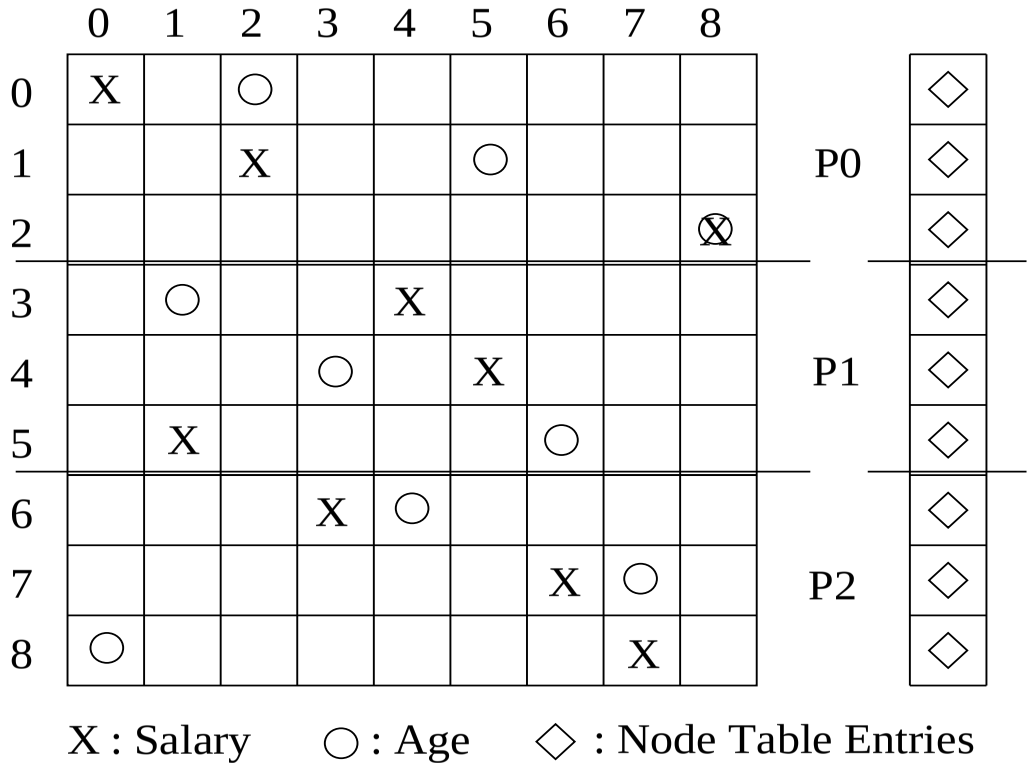
Node
L
R
L
R
L
L
R
R
L

Node
L
L
L
R
R
R
L
R
L

	tid	attr2	Class
P1	3	19	0
	6	24	0
	9	28	1
P2	2	33	1
	4	38	0
	7	40	1
P3	5	50	0
	8	58	0
	1	70	1

This Design is Inspired by..

- Communication Structure of Parallel Sparse Matrix-Vector Algorithms.



Parallel Hashing Paradigm

- Distributed Hash Table.
- Hash Function: $h(k) = (p_i , l)$
- Construction:
  - (k,v) --> (p_i,l) --> form send buffers {(l,v)} for each p_i --> all-to-all-personalized communication.
- Enquiry:
  - (k) --> (p_i , l) --> form {(l)} buffers for each p_i --> all-to-all-personalized comm --> each p_i replaces received {(l)} with {(v)} --> all-to-all-personalized comm.
- If each processor has m keys to hash, then time is O(m) if m is $\Omega(p)$; i.e. $\Omega(p^2)$ overall keys.

Applying the paradigm to ScalParC : Update

rid	Salary	Age	cid
0	24542	70	0
1	98816	33	0
2	49241	19	1
3	126146	38	1
4	94766	50	0
5	97672	24	0
6	136838	40	1
7	153032	58	1
8	64911	28	0

(a)

Salary

	value	rid	cid
P0	24542	0	0
	49241	2	1
	64911	8	0
P1	94766	4	0
	97672	5	0
SP →	98816	1	0
P2	126146	3	1
	136838	6	1
	153032	7	1

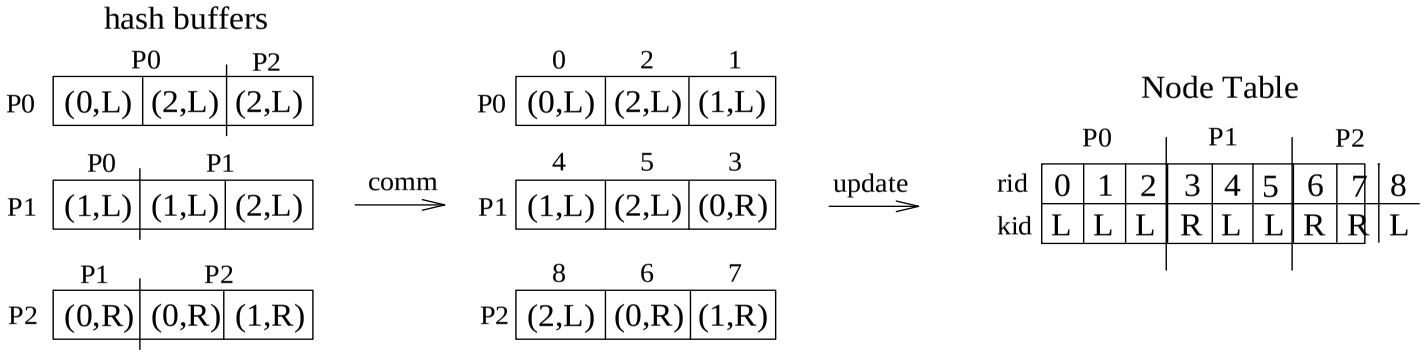
Age

	value	rid	cid
P0	19	2	1
	24	5	0
	28	8	0
P1	33	1	0
	38	3	1
	40	6	1
P2	50	4	0
	58	7	1
	70	0	0

Node Table

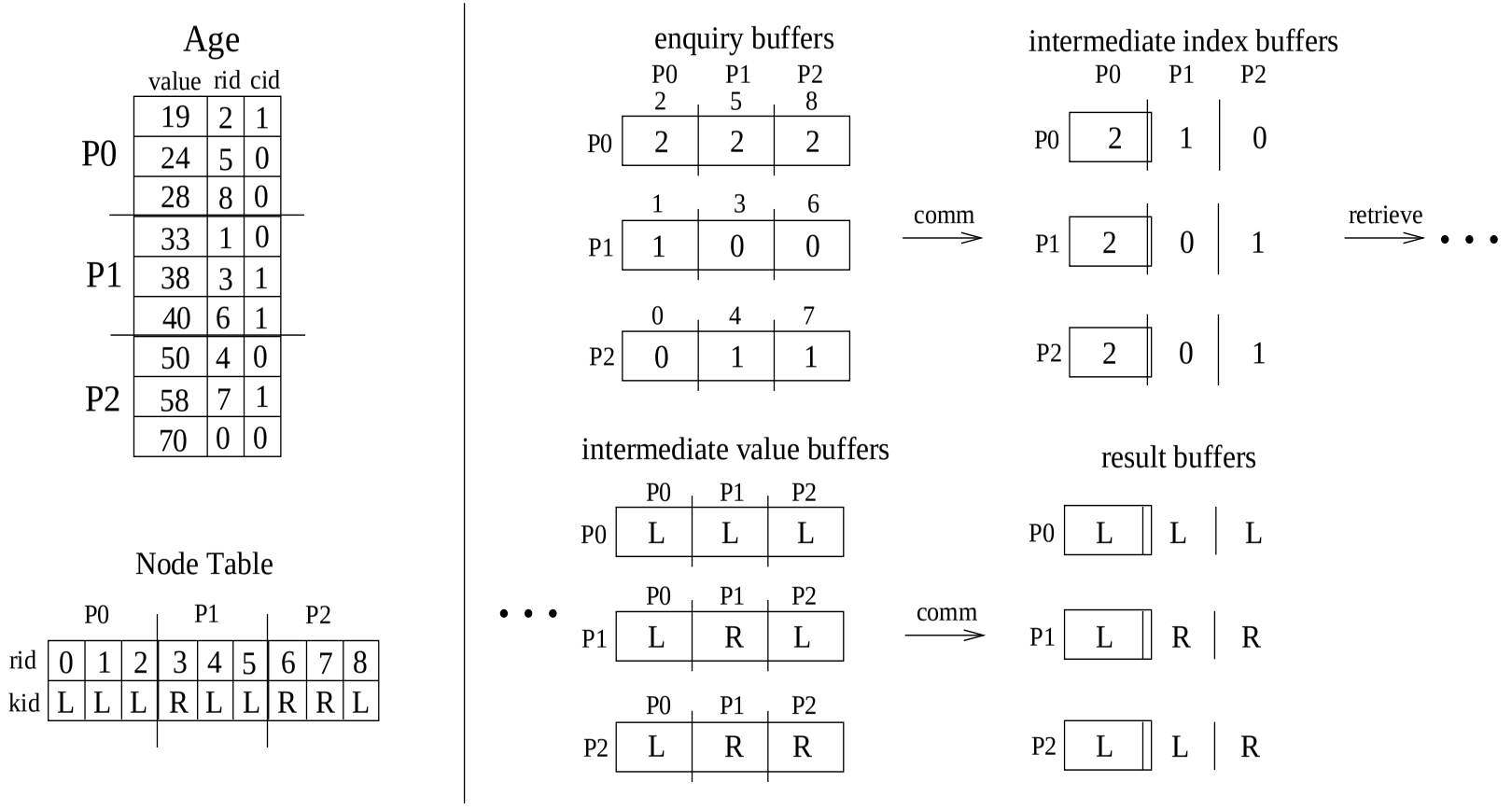
	rid	kid
P0	0	-
	1	-
	2	-
P1	3	-
	4	-
	5	-
P2	6	-
	7	-
	8	-

(b)



(c)

Applying the Paradigm to ScalParC: Enquiry

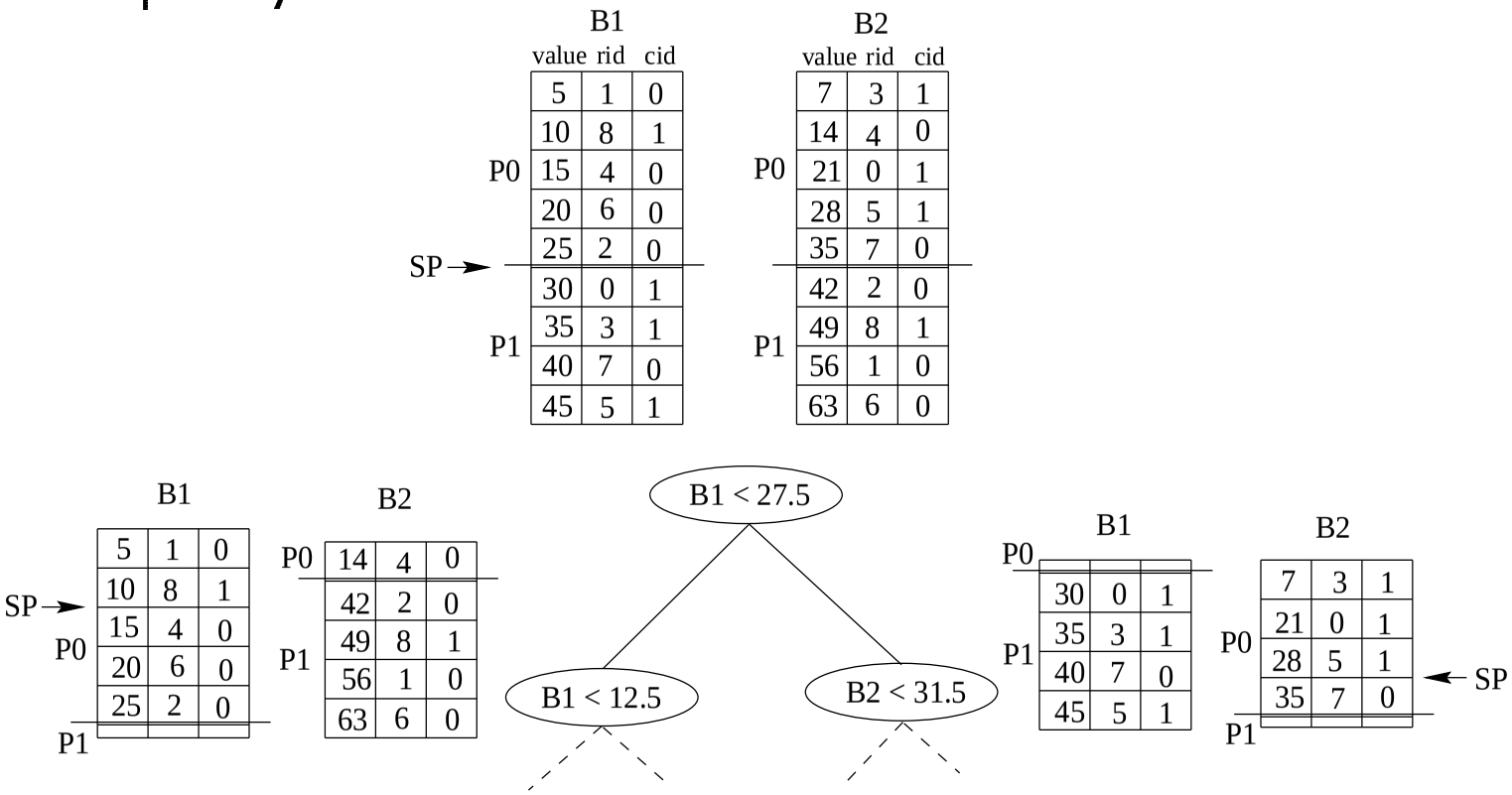


Verifying Scalability

- Updating the distributed hash table
  - Each processor receives at most *N/p* entries
  - Each processor can send at most *Nk/p* entries (k is the number of continuous attributes)
    - This is a worst case situation that does not happen frequently
- Inquiring the distributed hash table
  - Each processor will perform *k* inquires, one for each continuous attribute
  - Each processor receives at most *N/p* entries
  - Each processor sends at most *N/p* entries
- The per-processor communication overhead is in general *O(N/p)*
- Each processor requires only *O(N/p)* memory making ScalParC memory efficient

A worst case scenario for Updating Hash Table

- One Processor might need to send O(kN/p) data! Happens Infrequently.



Categorical/Continuous and Continuous/Categorical

- Special cases of the Continuous/Continuous
  - Categorical/Continuous does not require communication for loading the hash table
  - Continuous/Categorical does not require communication for inquiring the hash table

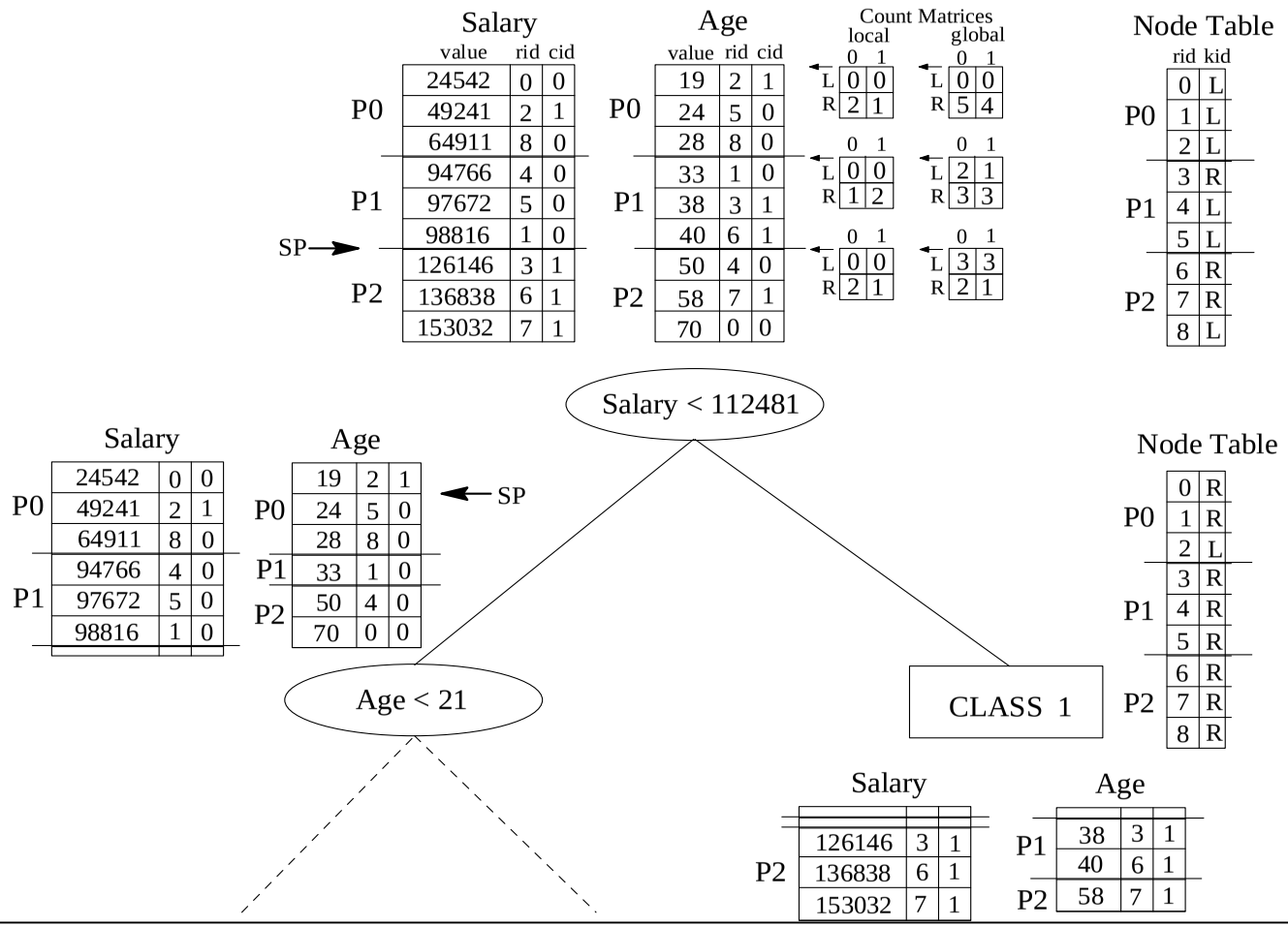
Algorithm: Level-wise Communications

- Tree is built in a breadth-first manner.
- At each level of the decision tree
  - Count Matrices for all attributes for all nodes are reduced in one single communication approach.
  - Loading the hash-table for all the nodes is combined into a single communication operation
  - Inquiring the hash-table to split a particular attribute list is combined in a single communication operation
  - A total of *k+1* All-to-All personalized communication operations are performed for each level of the tree

Structure of the Algorithm

- Sort Continuous Attributes (Pre-Sort)
- do level-wise while (at least one node requiring split)
  - Compute count matrices (Find-Split I)
  - Compute best gini index for nodes requiring split (Find-Split II)
  - Partition splitting attributes and update Node Table (Perform-Split I)
  - Partition non-splitting attributes by inquiring Node Table (Perform-Split II)
- end do

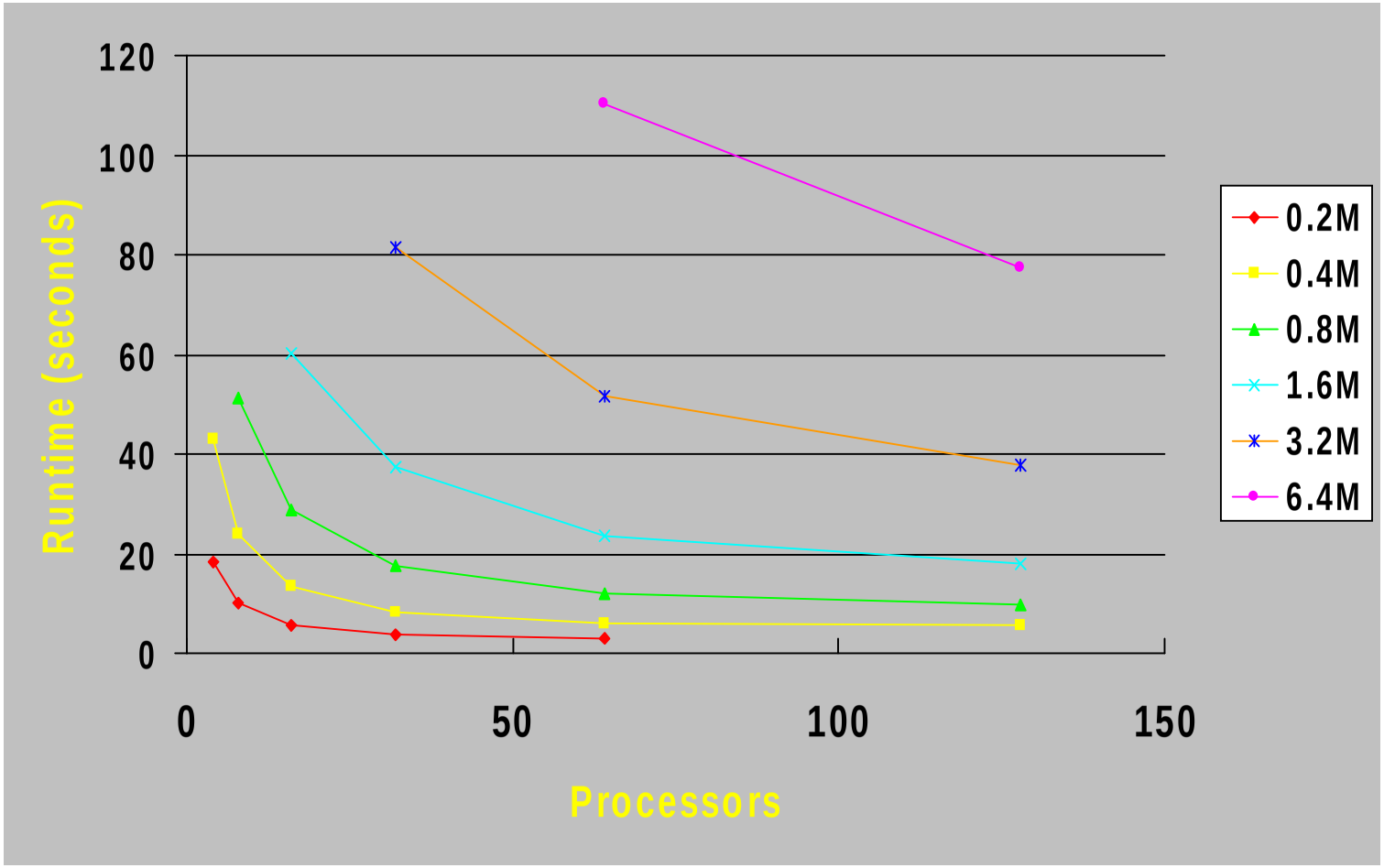
Example


Salary
value rid cid
P0
24542 0 0
49241 2 1
64911 8 0
P1
94766 4 0
97672 5 0
98816 1 0
SP
126146 3 1
P2
136838 6 1
153032 7 1
Age
value rid cid
P0
19 2 1
24 5 0
28 8 0
P1
33 1 0
38 3 1
40 6 1
50 4 0
P2
58 7 1
70 0 0
Count Matrices
local
global
L 0 0
R 2 1
L 0 0
R 5 4
L 0 0
R 1 2
L 2 1
R 3 3
L 0 0
R 2 1
L 3 3
R 2 1
Node Table
rid kid
P0
0 L
1 L
2 L
P1
3 R
4 L
5 L
P2
6 R
7 R
8 L
Salary < 112481
Salary
P0
24542 0 0
49241 2 1
64911 8 0
P1
94766 4 0
97672 5 0
98816 1 0
Age
P0
19 2 1
24 5 0
28 8 0
P1
33 1 0
P2
50 4 0
70 0 0
SP
Node Table
P0
0 R
1 R
2 L
P1
3 R
4 R
5 R
P2
6 R
7 R
8 R
Age < 21
CLASS 1
Salary
P2
126146 3 1
136838 6 1
153032 7 1
Age
P1
38 3 1
40 6 1
P2
58 7 1

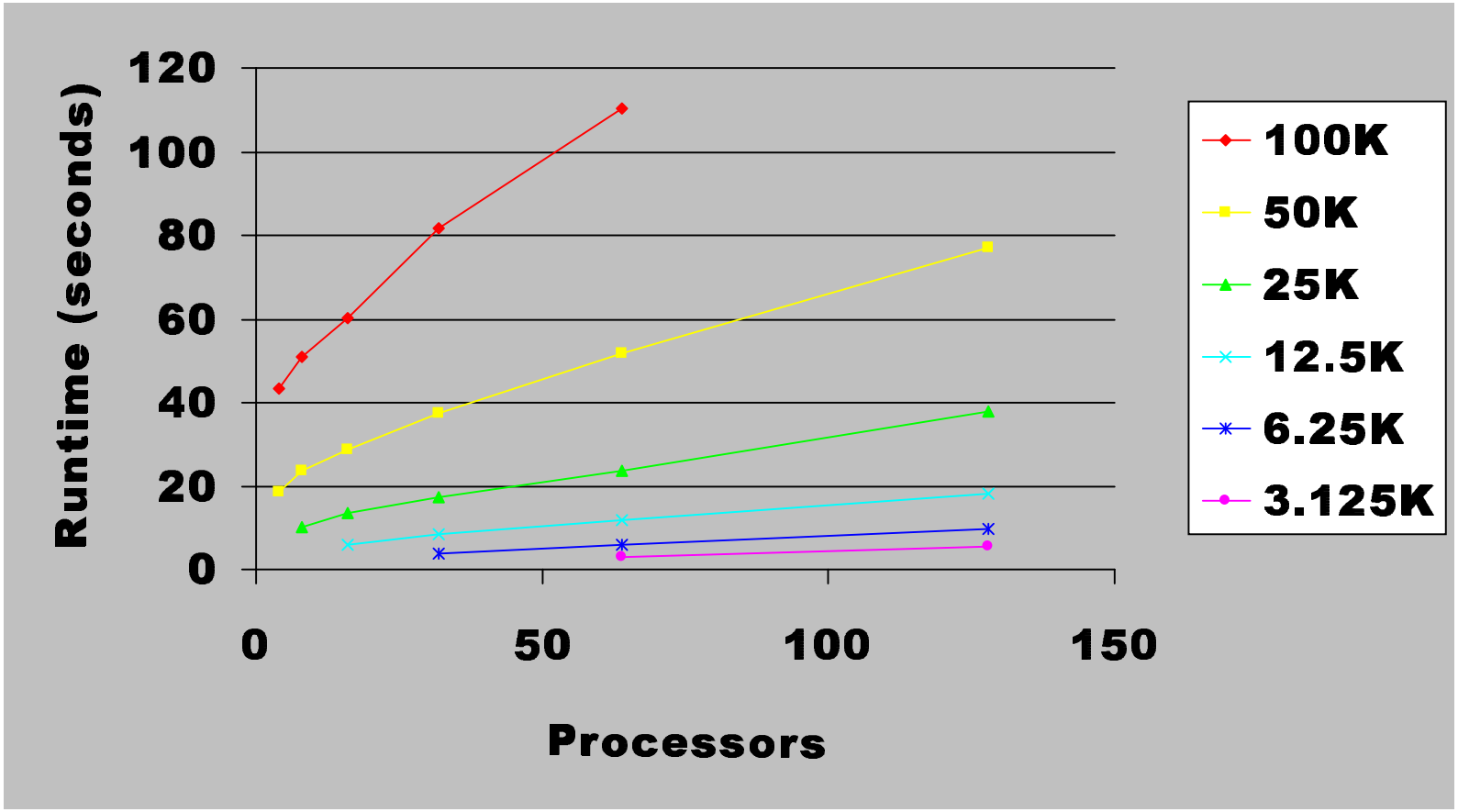
Experimental Results

- Experiments were performed on a 128-processor Cray T3D
- Training sets were synthetically generated
  - Each contained only continuous attributes (5-9)
  - The were two possible class labels

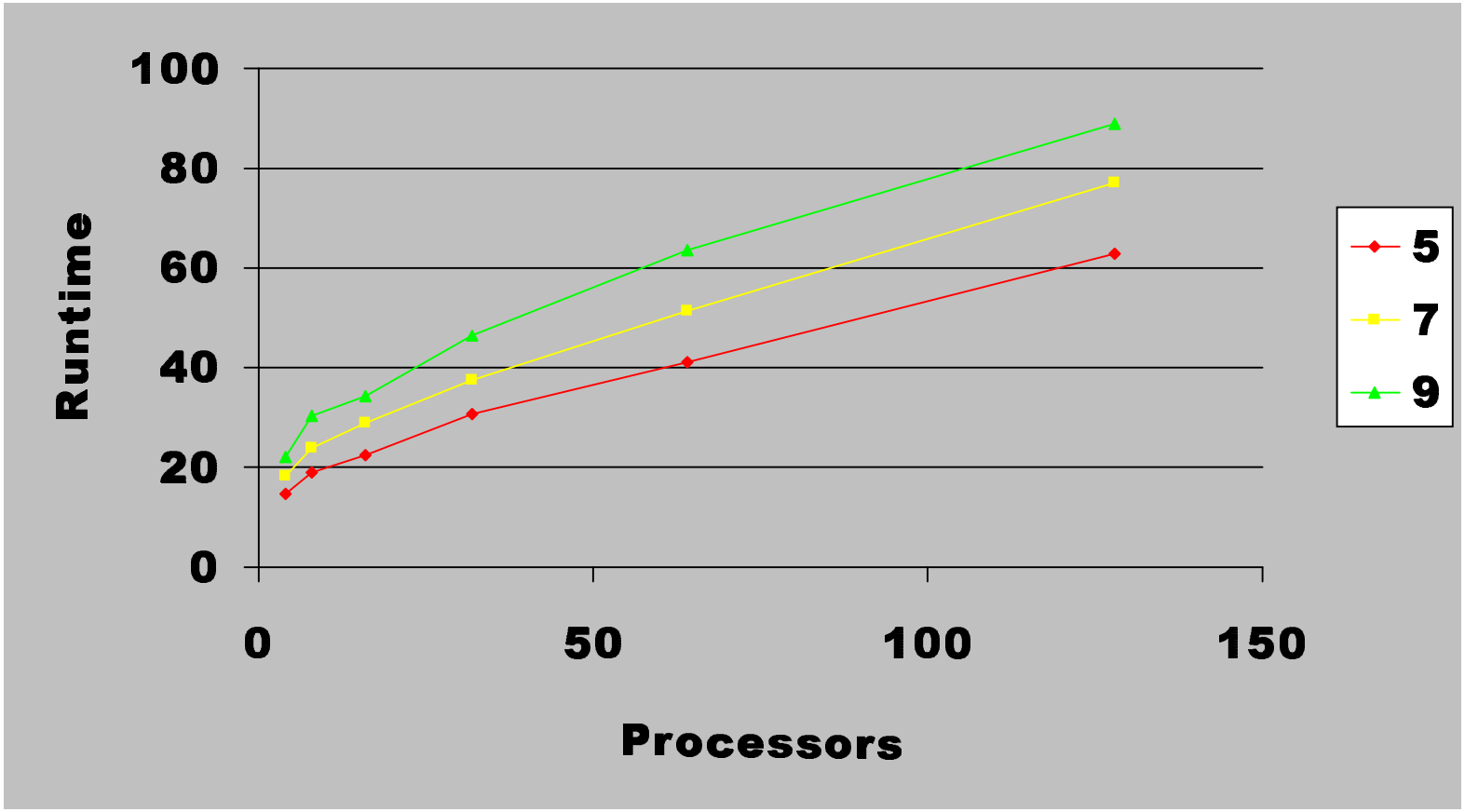
Parallel Runtime



Constant size/processor



Different Number of Attributes



SMP Parallel Design Space

- Data parallelism: within a tree node
  - Split attributes: vertical
    - BASIC
    - Fixed Window
    - Moving Window
  - Split data (attribute lists): horizontal
- Task parallelism: between tree nodes
  - SUBTREE
- Static vs. dynamic load balancing

SPRINT: Attribute Lists

Training Set

Tid	Age	Car Type	Class
0	23	family	High
1	17	sports	High
2	43	sports	High
3	68	family	Low
4	32	truck	Low
5	20	family	High

Attribute lists

Age	Class	Tid
17	High	1
20	High	5
23	High	0
32	Low	4
43	High	2
68	Low	3

continuous (sorted)

Car Type	Class	Tid
family	High	0
sports	High	1
sports	High	2
family	Low	3
truck	Low	4
family	High	5

categorical (orig order)

Splitting Attribute Lists

Training Set

Tid	Age	Car Type	Class
0	23	family	High
1	17	sports	High
2	43	sports	High
3	68	family	Low
4	32	truck	Low
5	20	family	High

Attribute lists

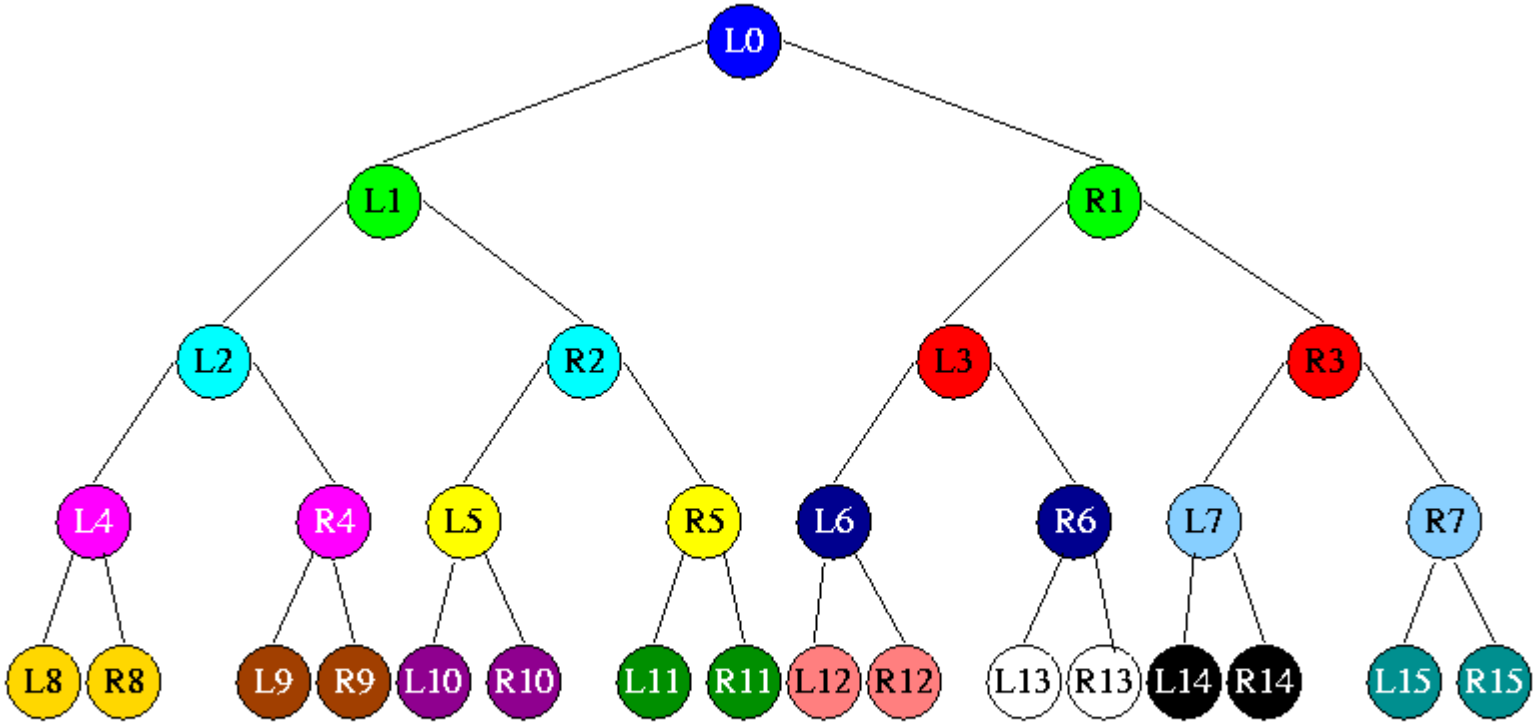
Age	Class	Tid
17	High	1
20	High	5
23	High	0
32	Low	4
43	High	2
68	Low	3

continuous (sorted)

Car Type	Class	Tid
family	High	0
sports	High	1
sports	High	2
family	Low	3
truck	Low	4
family	High	5

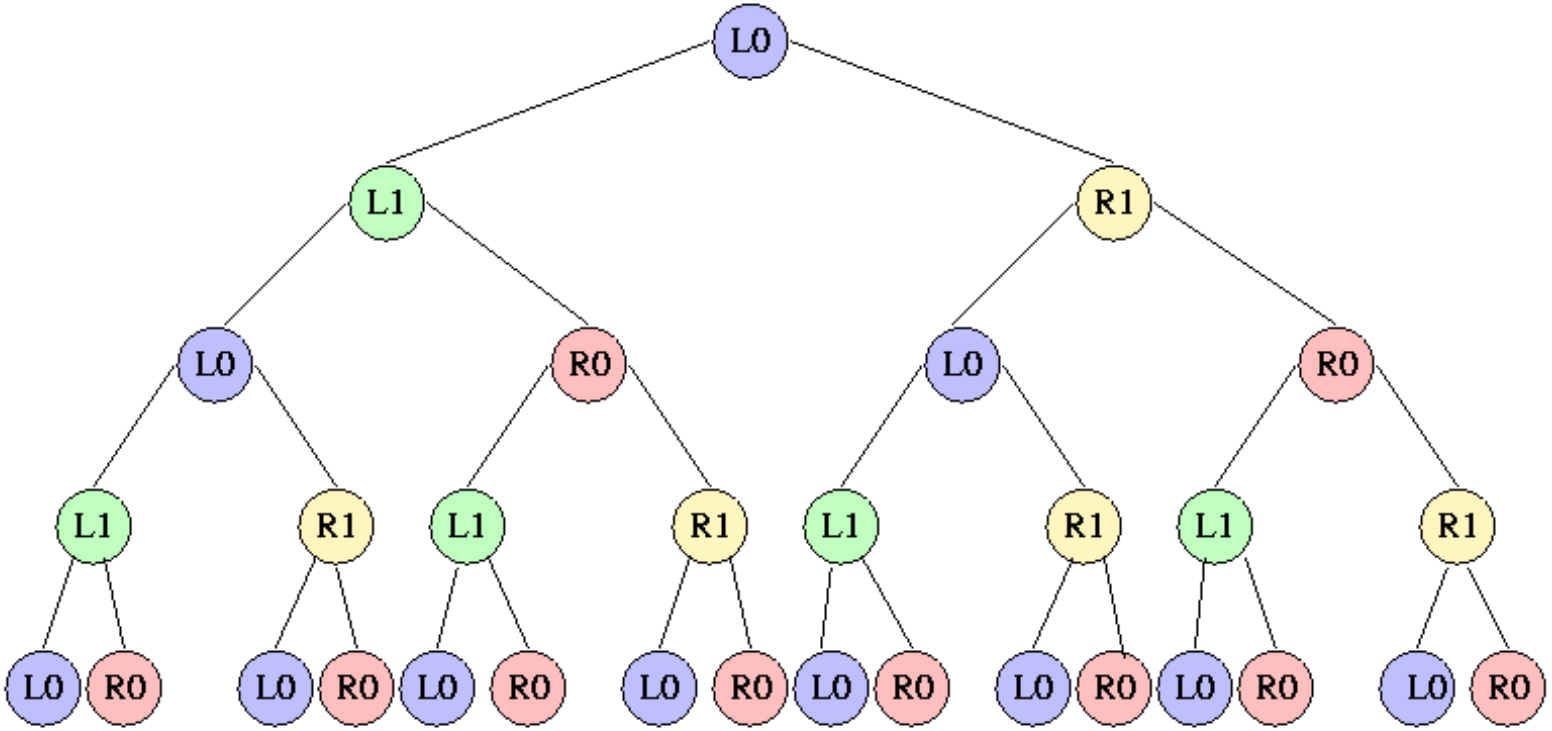
categorical (orig order)

SPRINT: File per Attribute



TOTAL FILES PER ATTRIBUTE: 32

Optimized File Usage

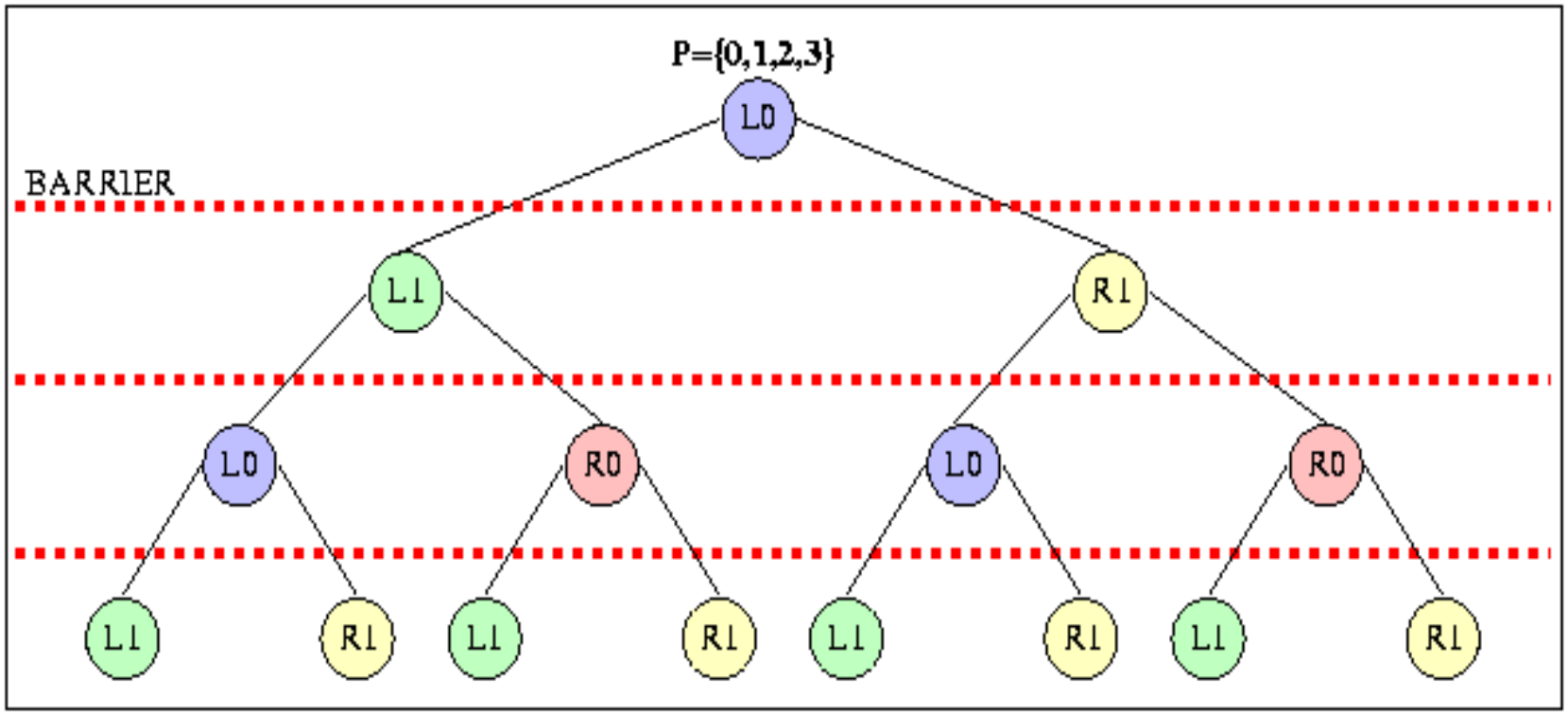


TOTAL FILES PER ATTRIBUTE: 4

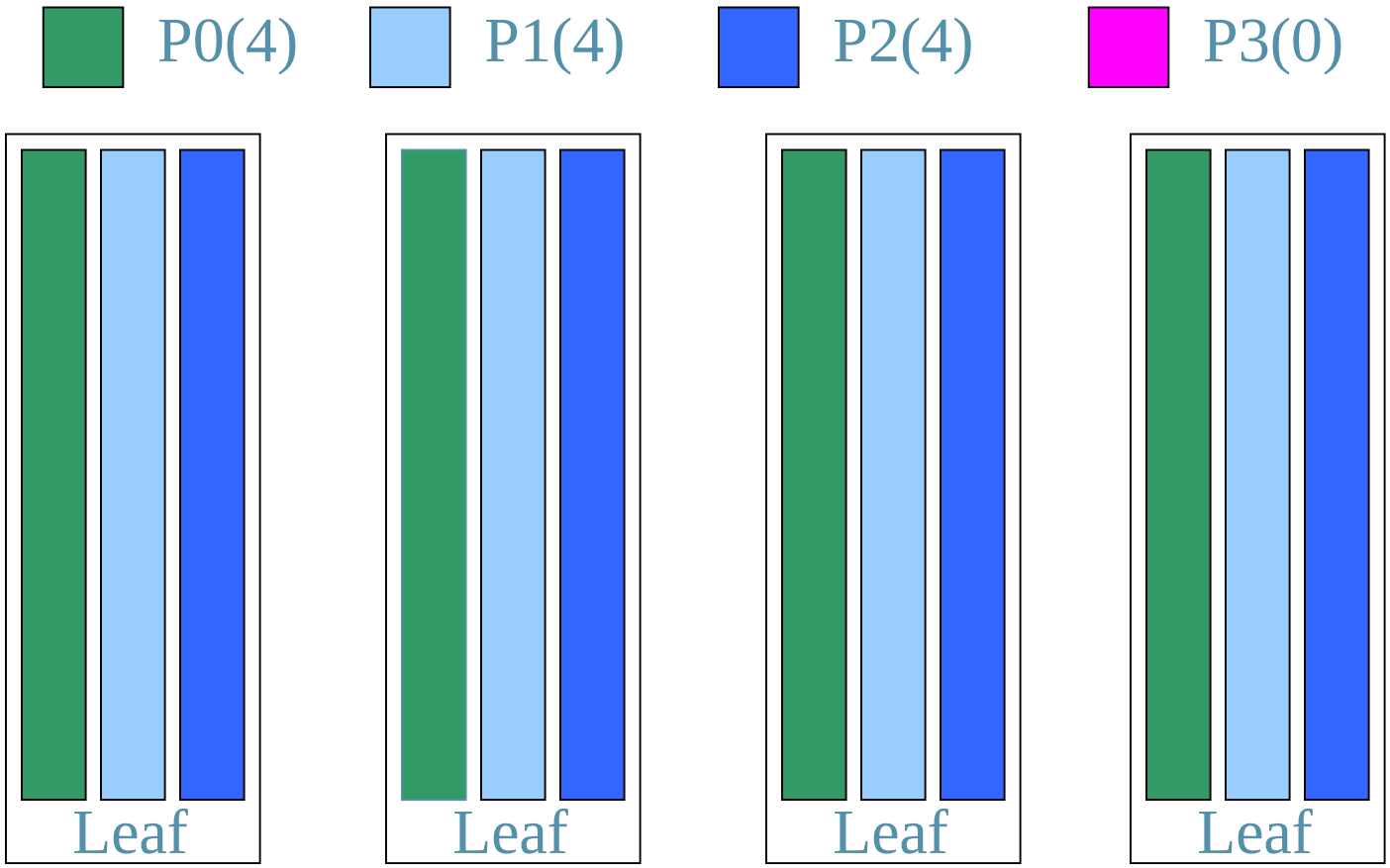
BASIC: Data Parallel

- Given current leaf frontier
- Atomically acquire an attribute and evaluate it for all leaves
- Master finds winning attribute and constructs hash table
- Atomically acquire an attribute and split its list for all leaves
- Barrier synchronization between phases

BASIC (Tree View)



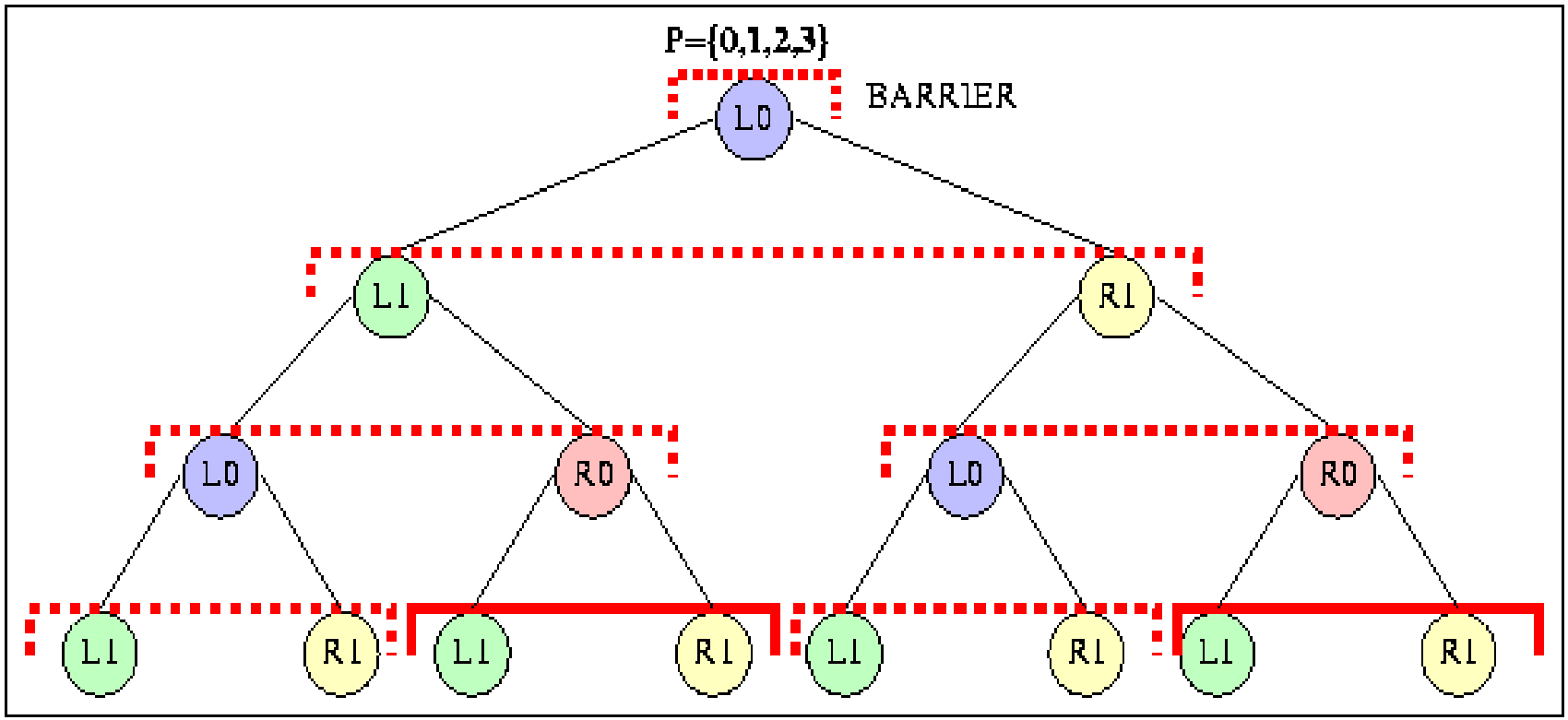
BASIC (Level View, A=3, P=4)



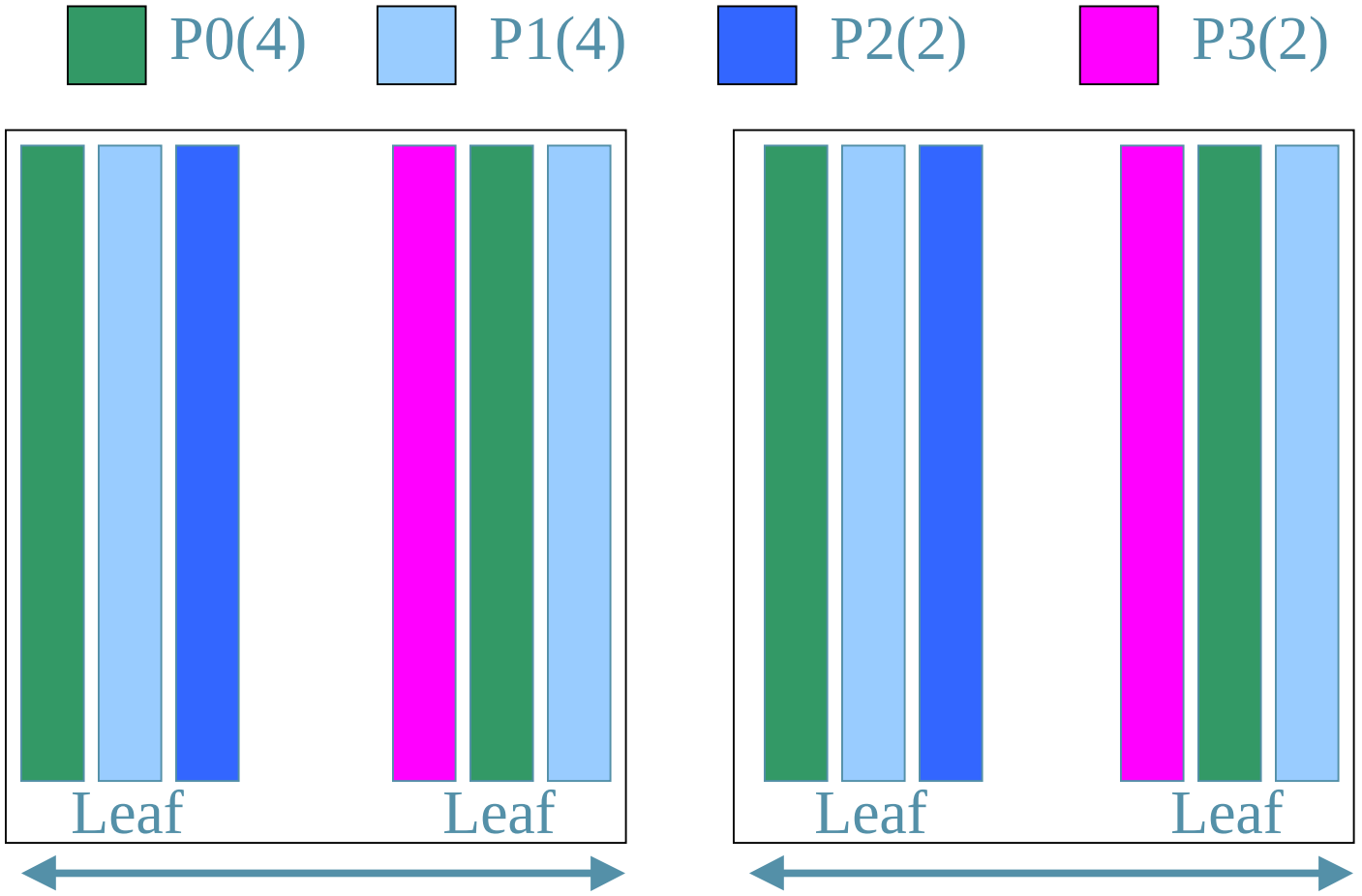
Fixed Window: Data Parallel

- Partition leaves into blocks/window of K
- Dynamically acquire any attribute for any leaf within current block and evaluate it
- Last processor to work on a leaf notes the winning attribute and builds the hash table
- Split data as in BASIC
- Barrier synchronization between each block of K leaves

FWK (Tree View)



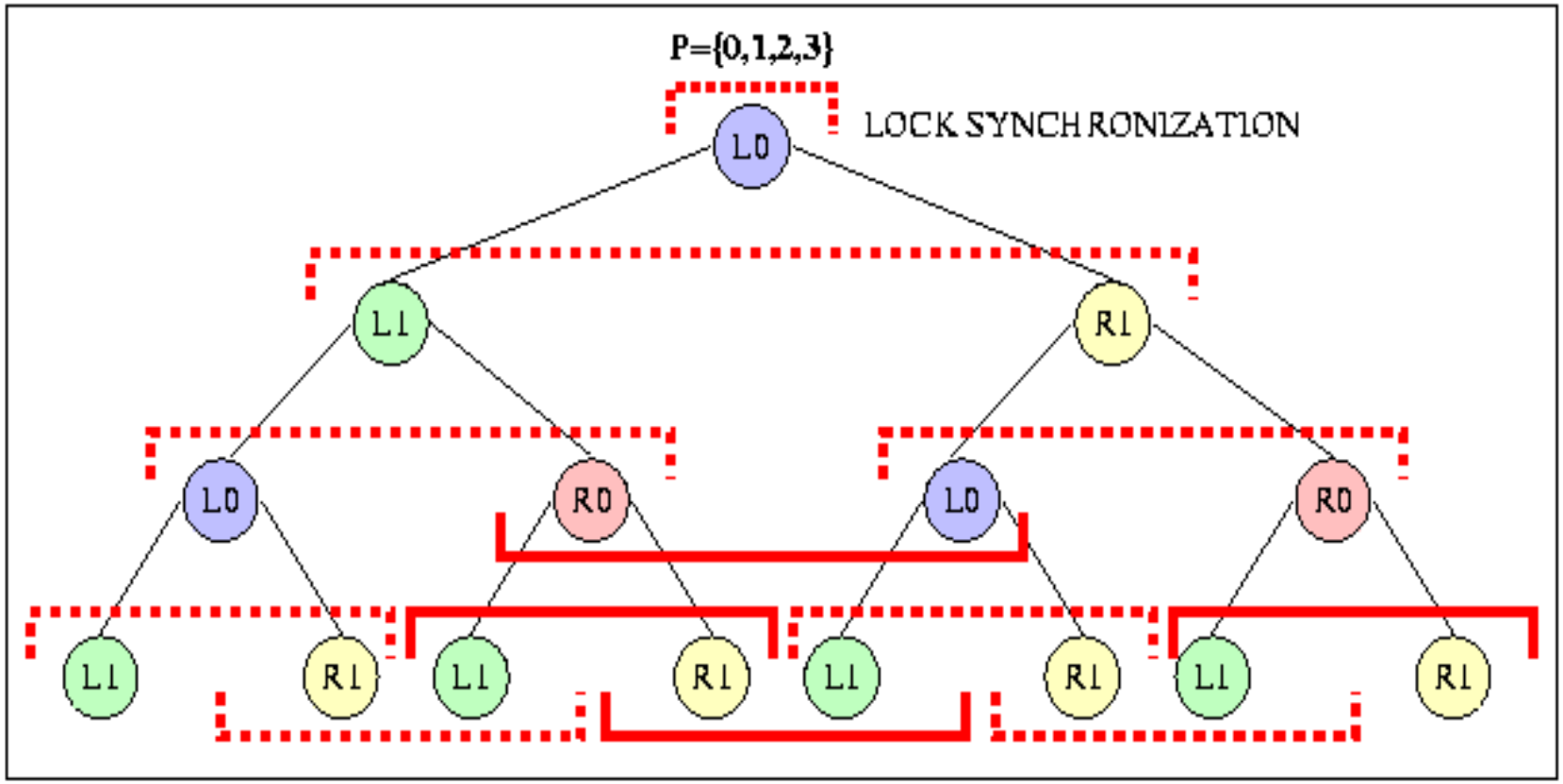
FW2 (Level View)



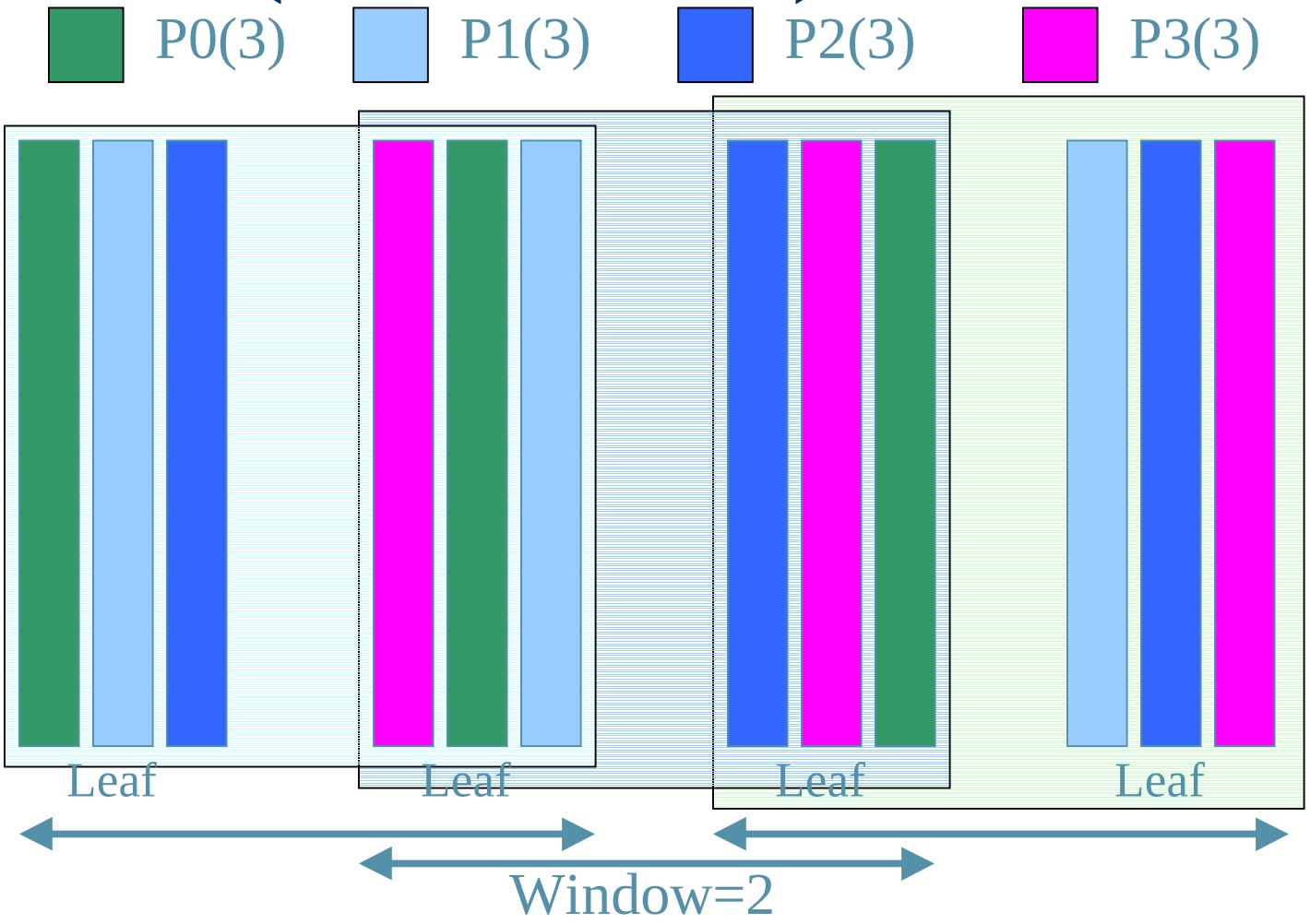
Moving Window: Data Parallel

- Partition leaves into blocks/window of K
- Dynamically acquire any attribute for any leaf, say i, within current block
- Wait for last block's i-th leave
- Last processor to work on a leaf notes the winning attribute and builds the hash table
- Split data as in BASIC
- No barriers, only conditional wait

MWK (Tree Level)



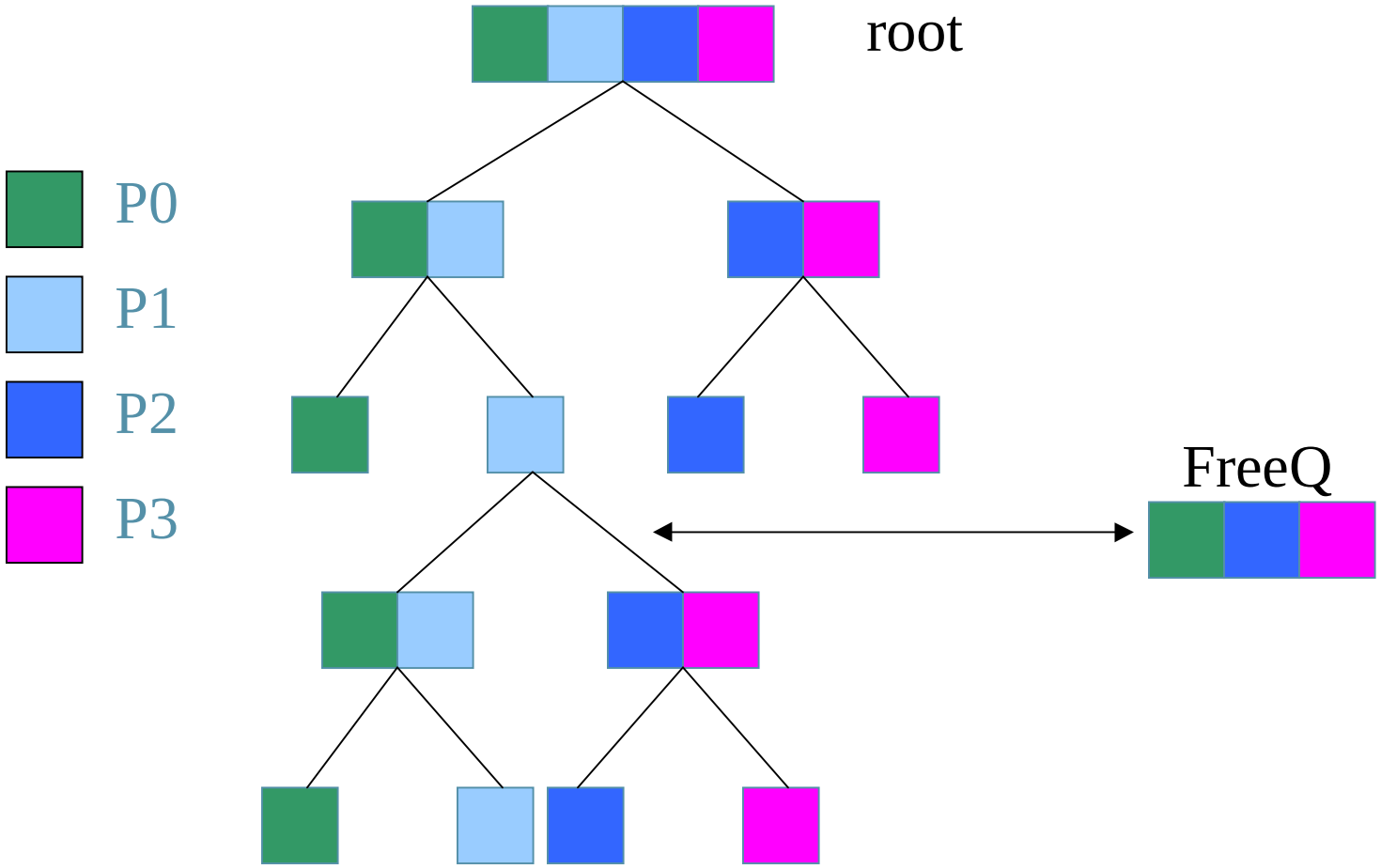
MW2 (Level View)



SUBTREE: Task Parallel

- Processor group (P) and leaf frontier (L)
- Apply BASIC, get new leaf frontier (NL)
- if NL is empty, put processors in FreeQ, else new group, NP = P + FreeQ
- Master divides NP and NL into two parts
- NP1 works on NL1, NP2 works on NL2
- Initially all processors work on root

SUBTREE (4 procs)



Experimental Setup

- SMP Machines
  - 112 MHz PowerPC-604 processors
  - 16KB L1-cache, 1MB L2-cache
  - Machine: 4-way SMP
    - 128MB memory, 300MB disk
- Synthetic Datasets: simple (F2) and complex (F7) decision tree

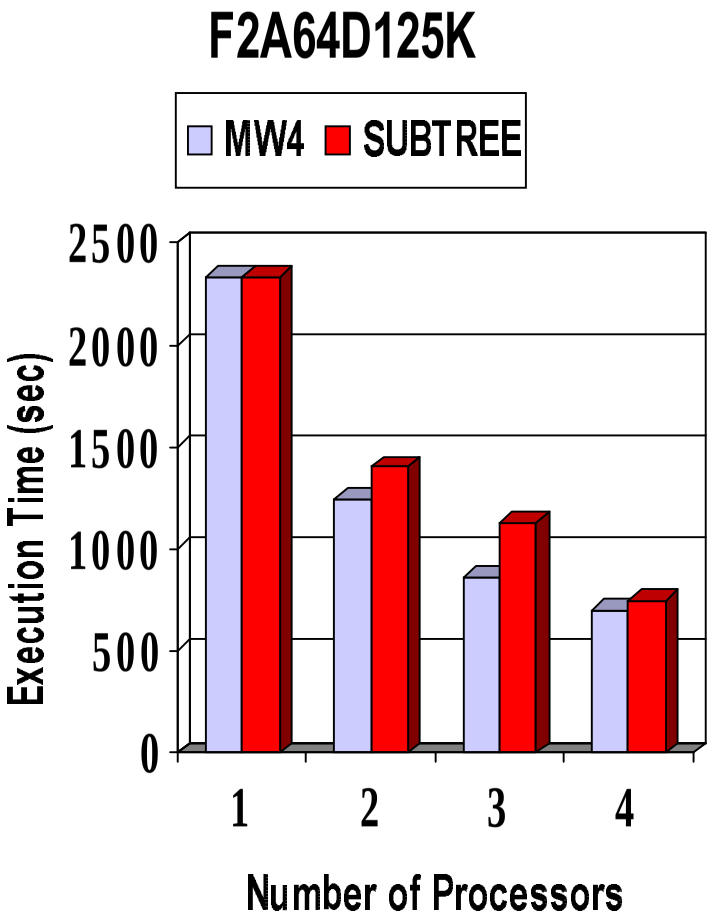
Experimental Datasets

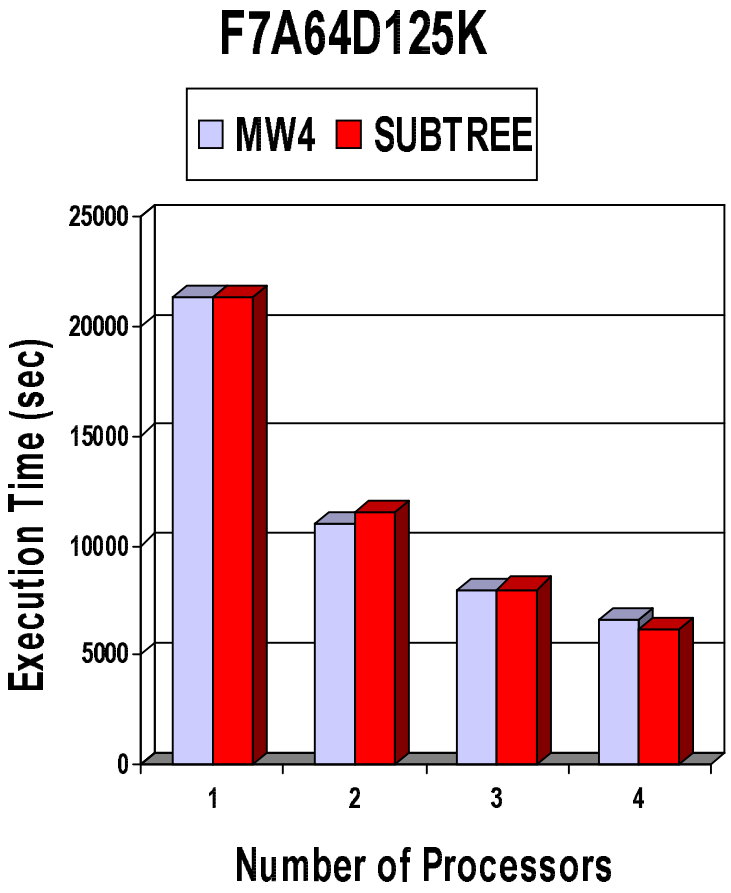
Dataset	Function	#Attr	Num Records	DBSize	Num Levels	Max Nodes
F2A8D1M	F2	8	1000K	192MB	4	2
F2A32D250K	F2	32	250K	192MB	4	2
F2A64D125K	F2	64	125K	192MB	4	2
F7A8D1M	F7	8	1000K	192MB	60	4662
F7A32D250K	F7	32	250K	192MB	59	802
F7A64D125K	F7	64	125K	192MB	55	384

Setup and Sort Time

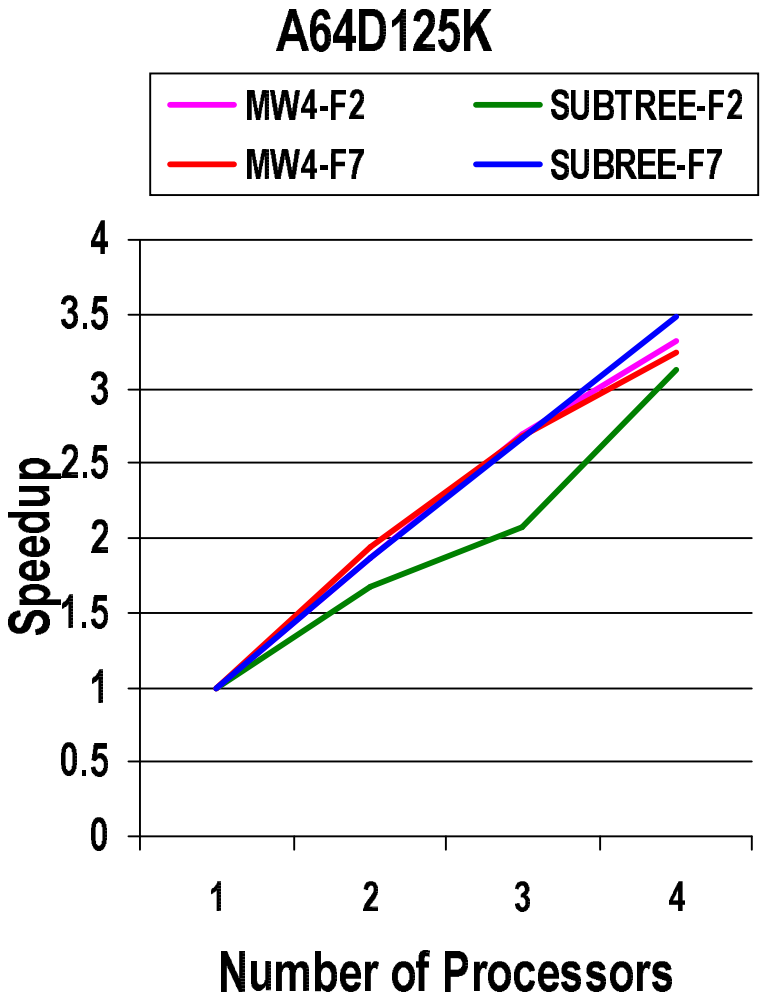
Dataset	Setup (sec)	Sort (sec)	Total (sec)	Setup%	Sort%
F2A8D1M	721	633	3597	20%	18%
F2A32D250K	685	598	3584	19%	17%
F2A64D125K	705	626	3665	19%	17%
F7A8D1M	989	817	23360	4%	4%
F7A32D250K	838	780	24706	3%	3%
F7A64D125K	672	636	22664	3%	3%

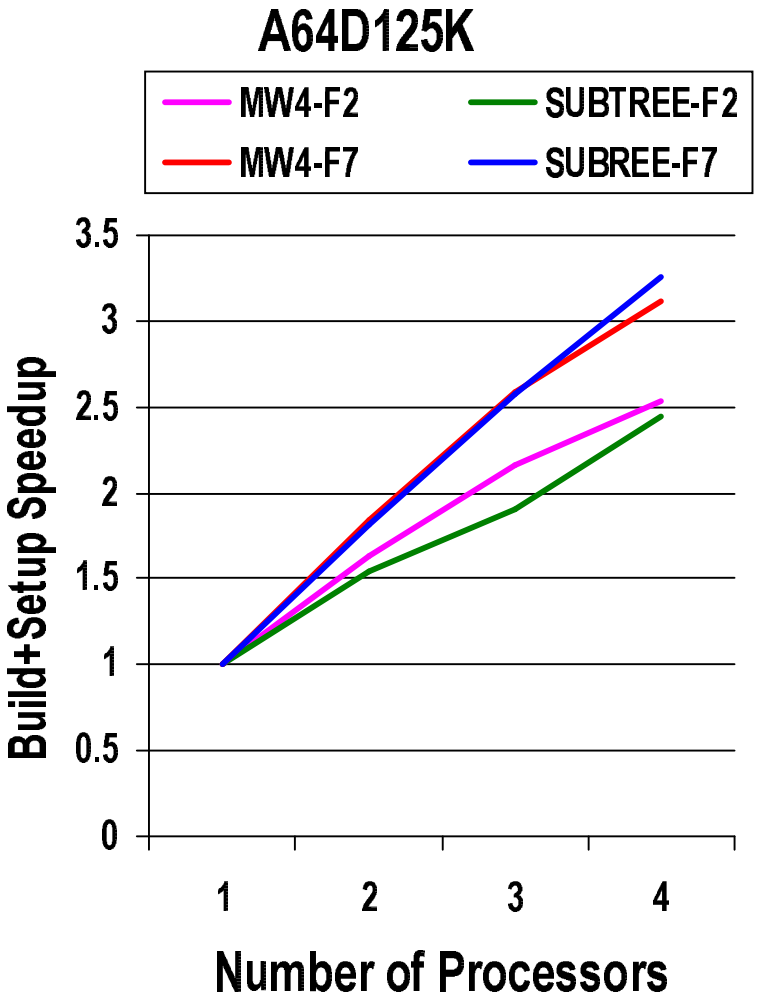
Parallel Performance





Parallel Performance





Tutorial overview

- Overview of KDD and data mining
- Parallel design space
- Classification
- Associations
- Sequences
- Clustering
- Future directions and summary

What is association mining?

- Given a set of items/attributes, and a set of objects containing a subset of the items
- Find rules: if I1 then I2 (sup, conf)
- I1, I2 are sets of items
- I1, I2 have sufficient support: P(I1+I2)
- Rule has sufficient confidence: P(I2|I1)

Association mining

- User specifies “interestingness”
  - Minimum support (minsup)
  - Minimum confidence (minconf)
- Find all frequent itemsets (> minsup)
  - Exponential Search Space
  - Computation and I/O Intensive
- Generate strong rules (> minconf)
  - Relatively cheap

Association Rule Discovery: Support and Confidence

TID	Items
1	Bread, Milk
2	Beer, Diaper, Bread, Eggs
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Bread, Diaper, Milk

Association Rule: $X \Rightarrow_{s,\alpha} y$

Support: $s = \frac{\sigma(X \cup y)}{|T|} (s = P(X, y))$

Confidence: $\alpha = \frac{\sigma(X \cup y)}{\sigma(X)|} (\alpha = P(y \mid X))$

Example:

$\{Diaper, Milk\} \Rightarrow_{s,\alpha} Beer$

$s = \frac{\sigma(Diaper, Milk, Beer)}{\text{Total Number of Transactions}} = \frac{2}{5} = 0.4$

$\alpha = \frac{\sigma(Diaper, Milk, Beer)}{\sigma(Diaper, Milk)|} = 0.66$

Handling Exponential Complexity

- Given *n* transactions and *m* different items:
  - number of possible association rules: $O(m2^{m-1})$
  - computation complexity: $O(nm2^m)$
- Systematic search for all patterns, based on support constraint [Agarwal & Srikant]:
  - If {A,B} has support at least α, then both A and B have support at least α.
  - If either A or B has support less than α, then {A,B} has support less than α.
  - Use patterns of *k*-1 items to find patterns of *k* items.

Apriori Principle

- Collect single item counts. Find large items.
- Find candidate pairs, count them => large *pairs* of items.
- Find candidate triplets, count them => large *triplets* of items, And so on...
- Guiding Principle: *Every subset of a frequent itemset has to be frequent.*
  - **Used for pruning many candidates.**

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

Minimum Support = 3

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
$6 + 6 + 2 = 14$

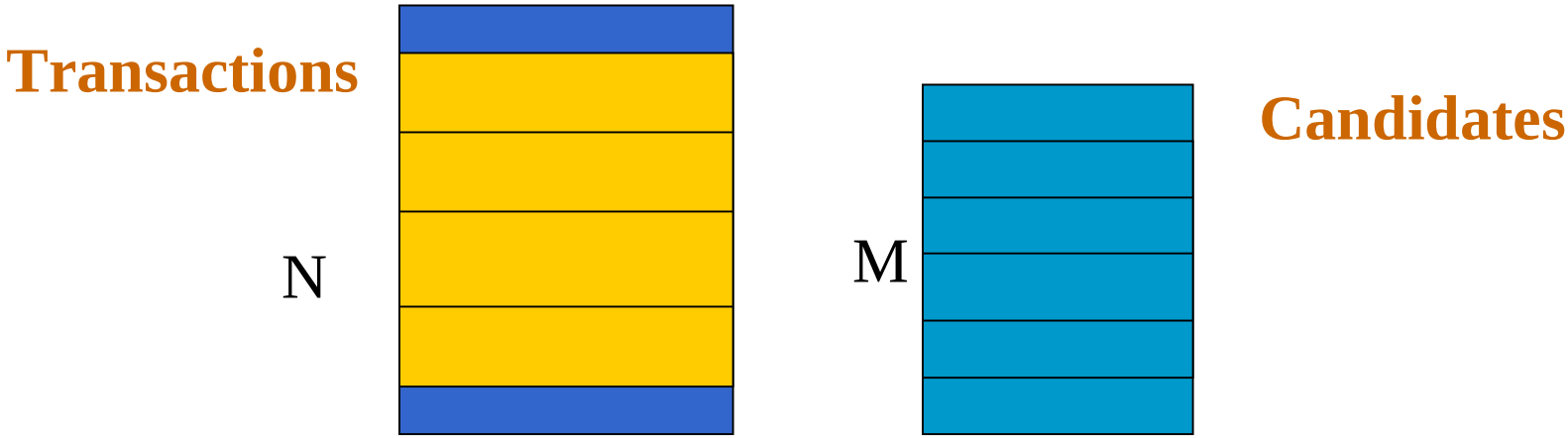
Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3
{Milk,Diaper,Beer}	2

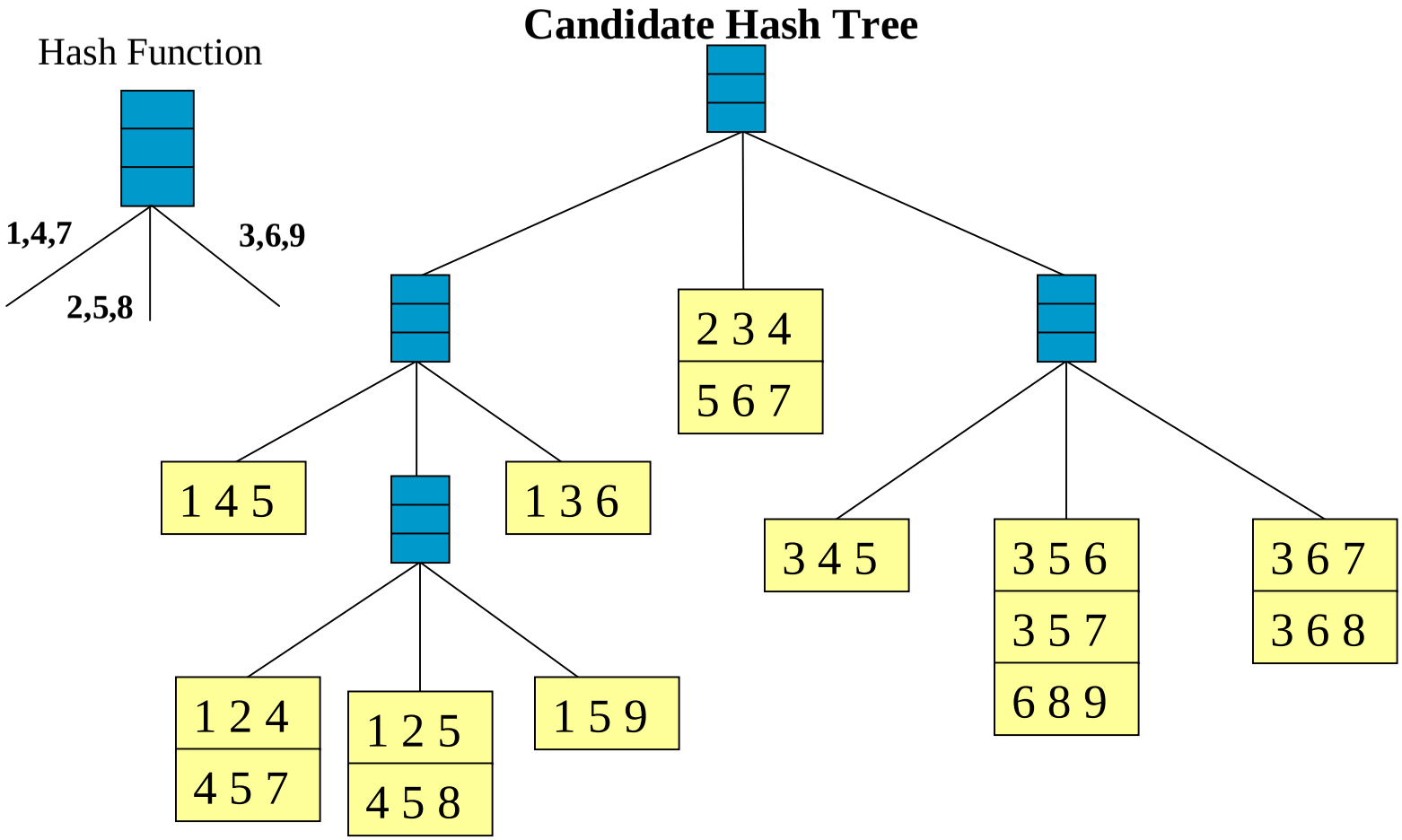
Counting Candidates

- Frequent Itemsets are found by counting candidates.
- Simple way:
  - Search for each candidate in each transaction.

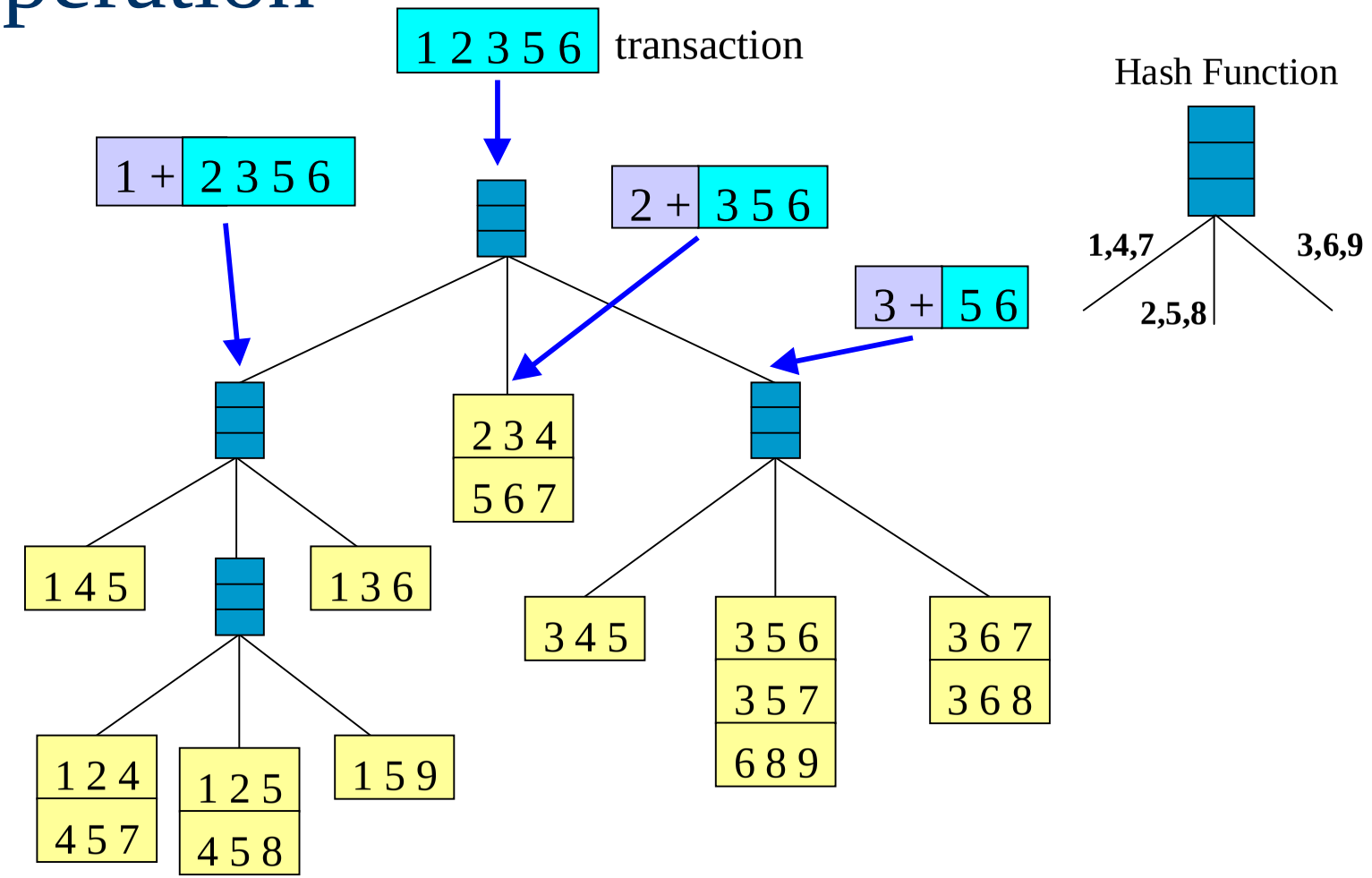
Expensive!!!



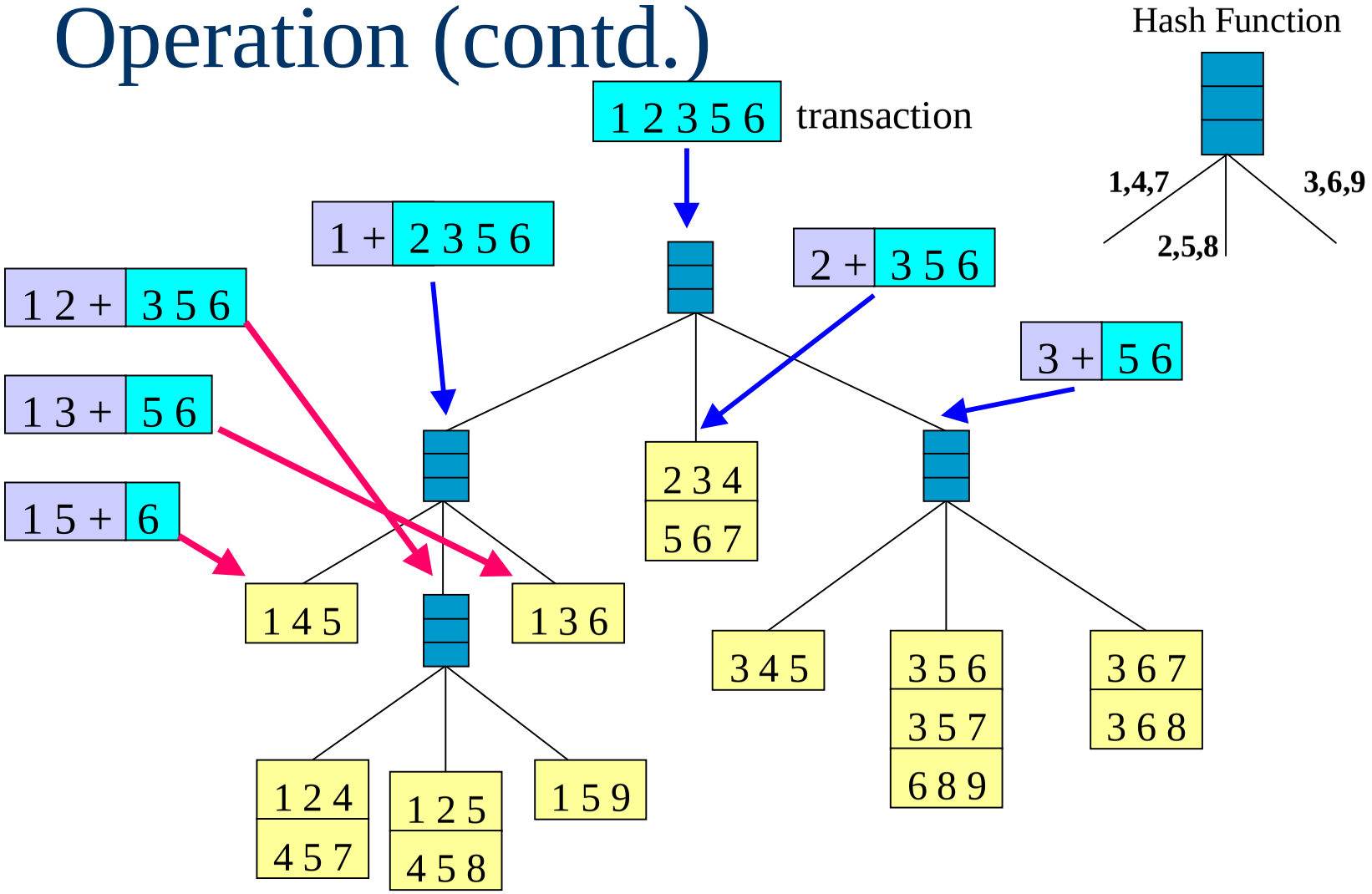
Association Rule Discovery: Hash tree for fast access.



Association Rule Discovery: Subset Operation



Association Rule Discovery: Subset Operation (contd.)



Parallel Formulation of Association Rules

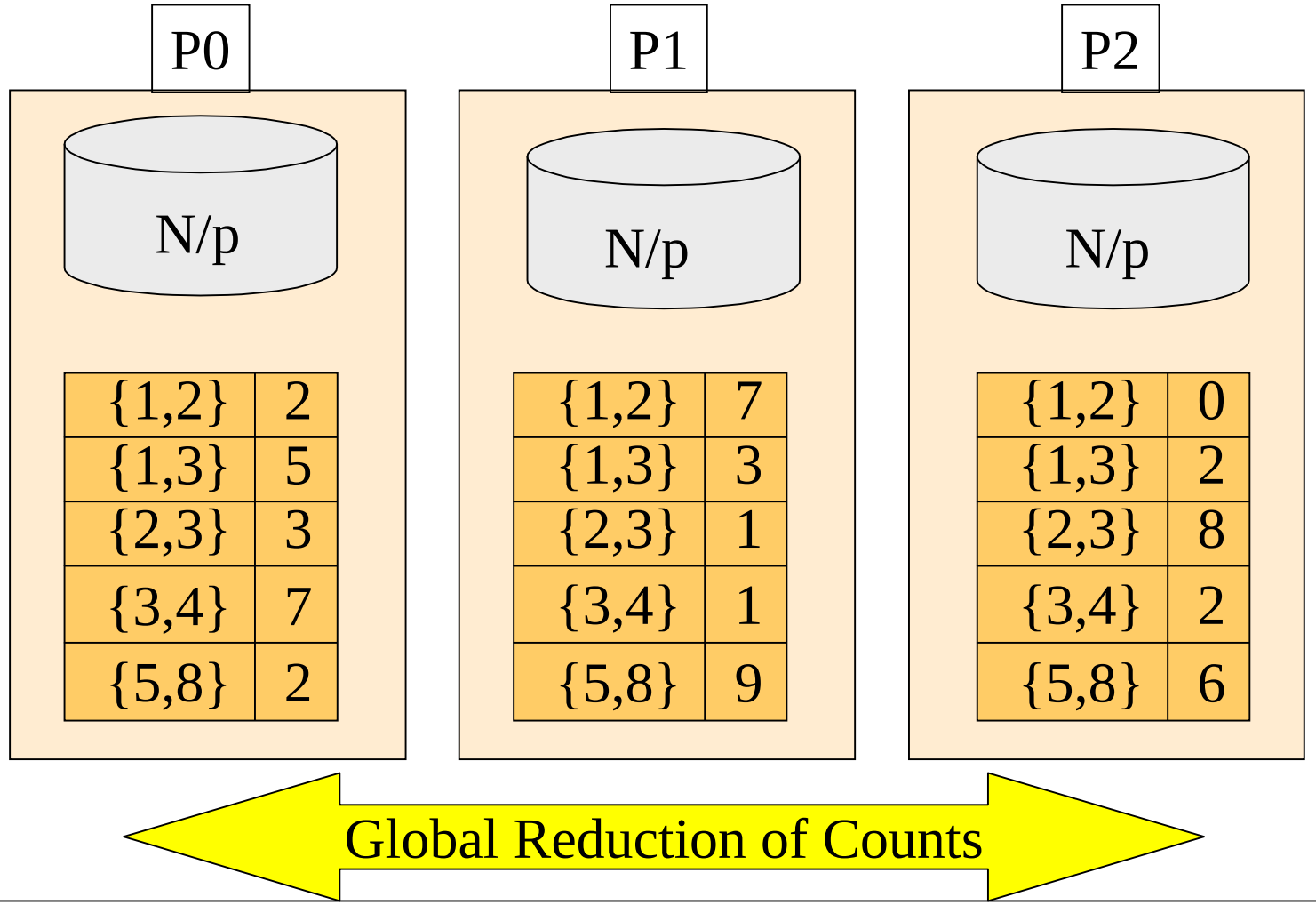
- Need:
  - Huge Transaction Datasets (10s of TB)
  - Large Number of Candidates.

- Data Distribution:
  - Partition the Transaction Database, or
  - Partition the Candidates, or
  - Both

Parallel Association Rules: Count Distribution (CD)

- Each Processor has complete candidate hash tree.
- Each Processor updates its hash tree with local data.
- Each Processor participates in global reduction to get global counts of candidates in the hash tree.
- Multiple database scans per iteration are required if hash tree too big for memory.

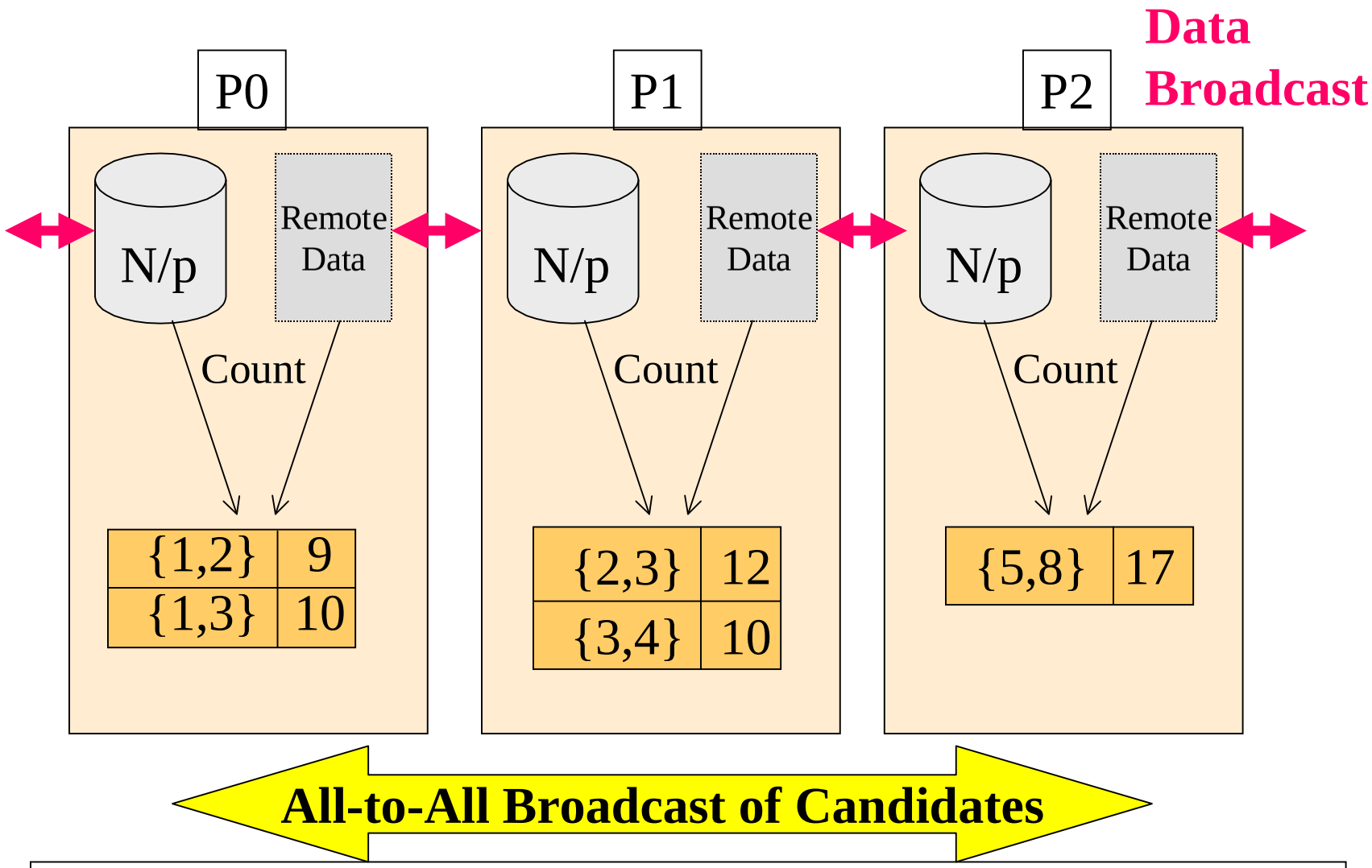
CD: Illustration



Parallel Association Rules: Data Distribution (DD)

- Candidate set is partitioned among the processors.
- Once local data has been partitioned, it is broadcast to all other processors.
- High Communication Cost due to data movement.
- Redundant work due to multiple traversals of the hash trees.

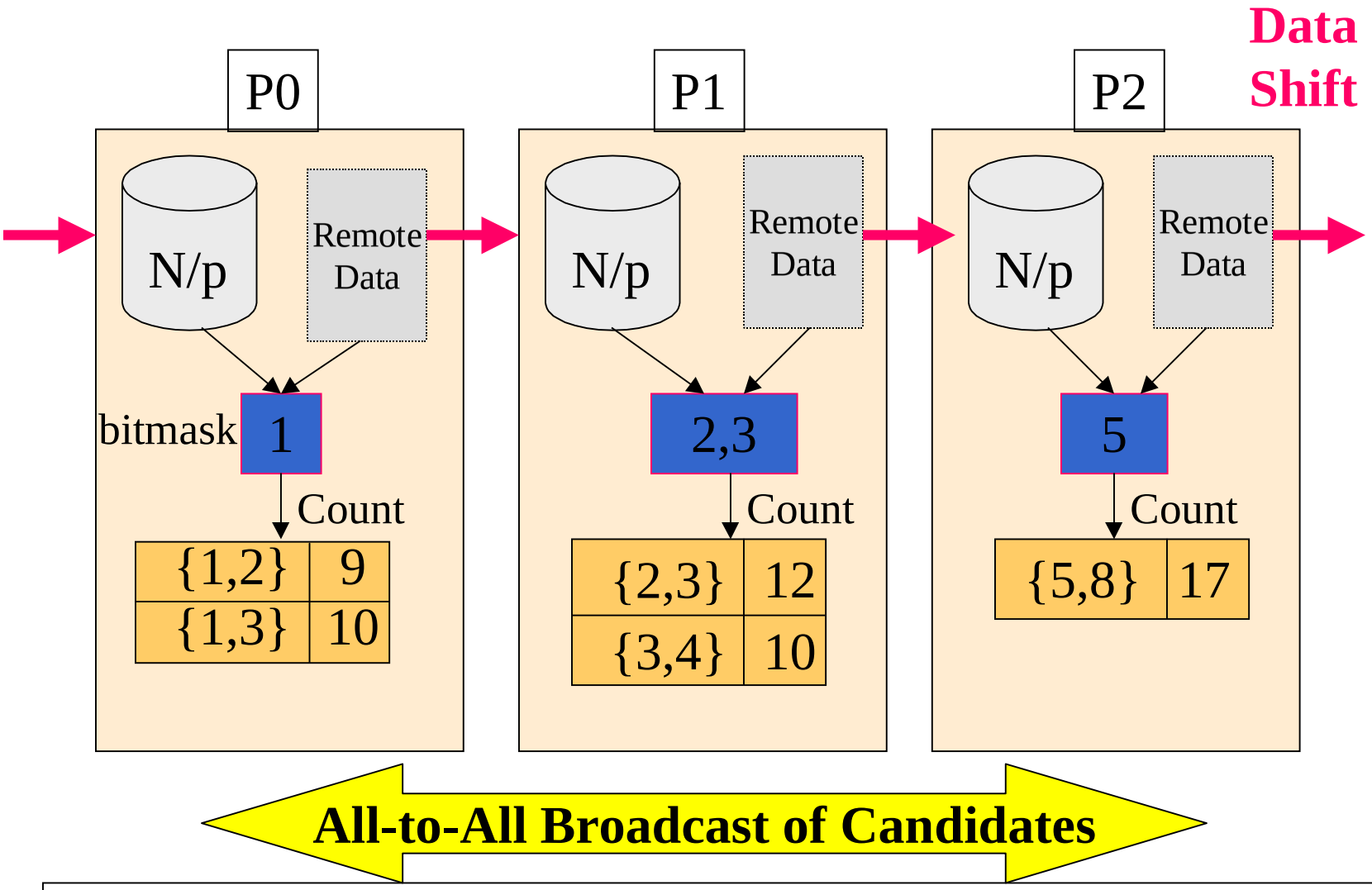
DD: Illustration



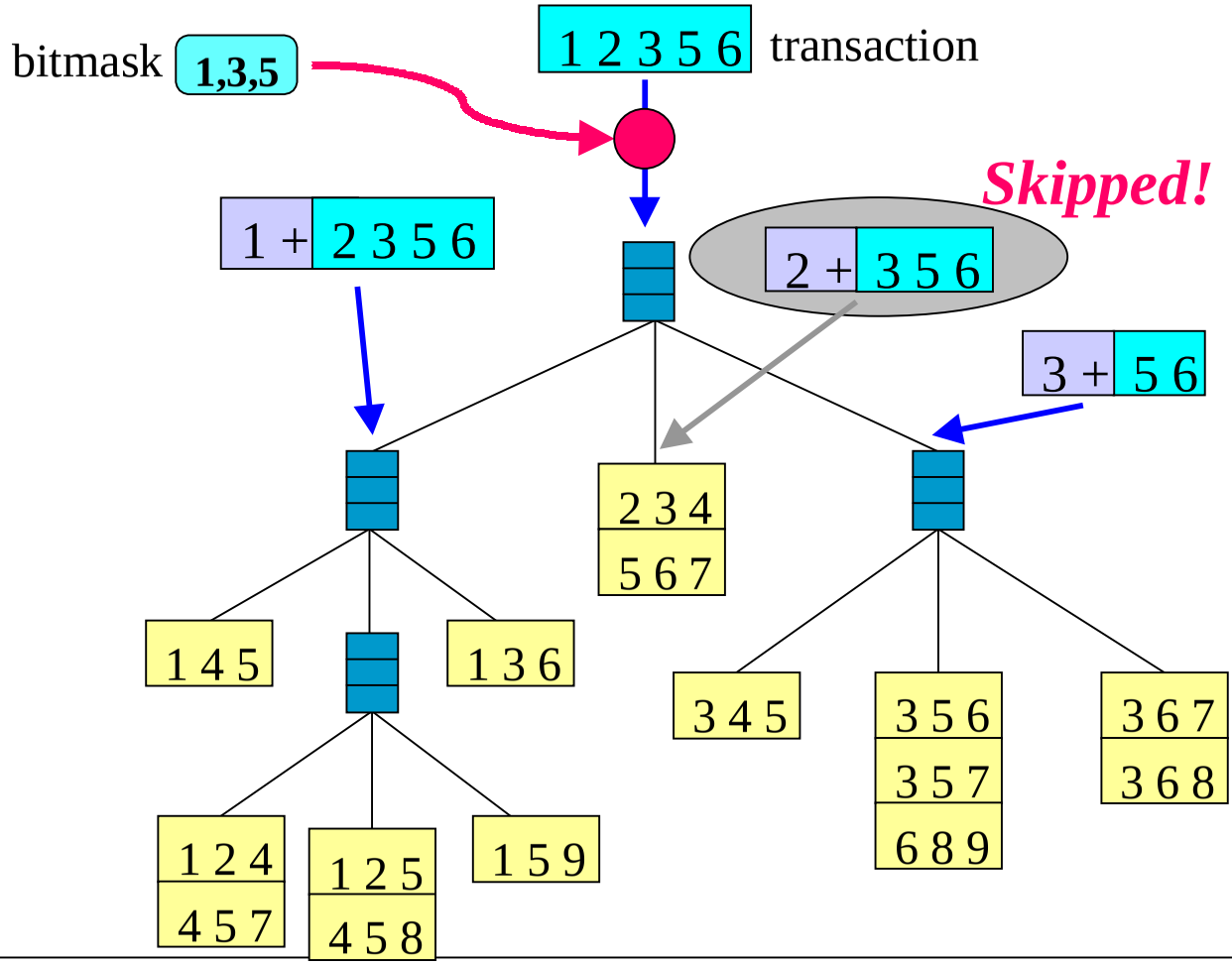
Parallel Association Rules: Intelligent Data Distribution (IDD)

- Data Distribution using point-to-point communication.
- Intelligent partitioning of candidate sets.
  - Partitioning based on the first item of candidates.
  - Bitmap to keep track of local candidate items.
- Pruning at the root of candidate hash tree using the bitmap.
- Suitable for single data source such as database server.
- With smaller candidate set, load balancing is difficult.

IDD: Illustration



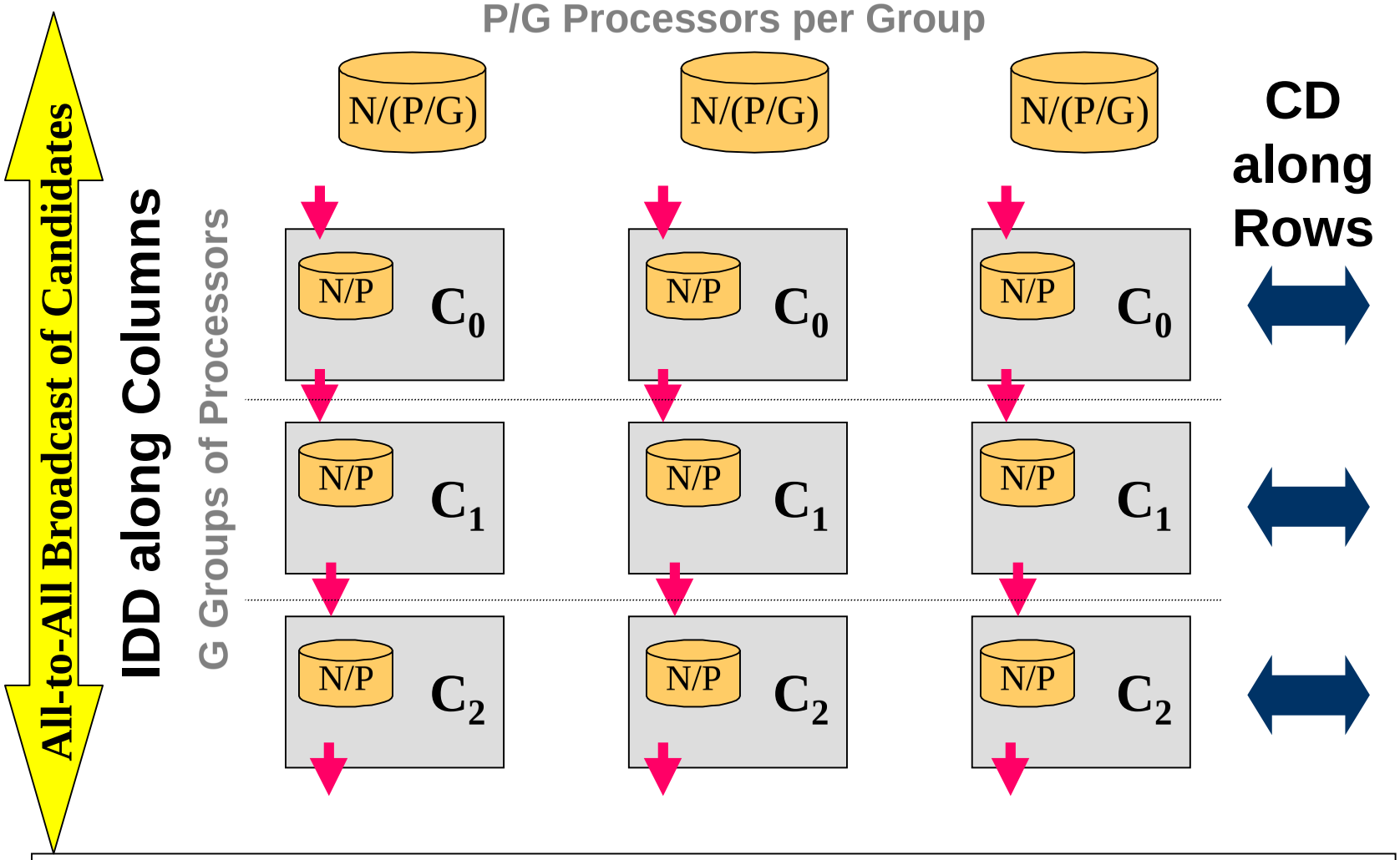
Filtering Transactions in IDD



Parallel Association Rules: Hybrid Distribution (HD)

- Candidate set is partitioned into G groups to just fit in main memory
  - ***Ensures Good load balance with smaller candidate set.***
- Logical processor mesh G x P/G is formed.
- Perform IDD along the column processors
  - ***Data movement among processors is minimized***.
- Perform CD along the row processors
  - ***Smaller number of processors is global reduction operation.***

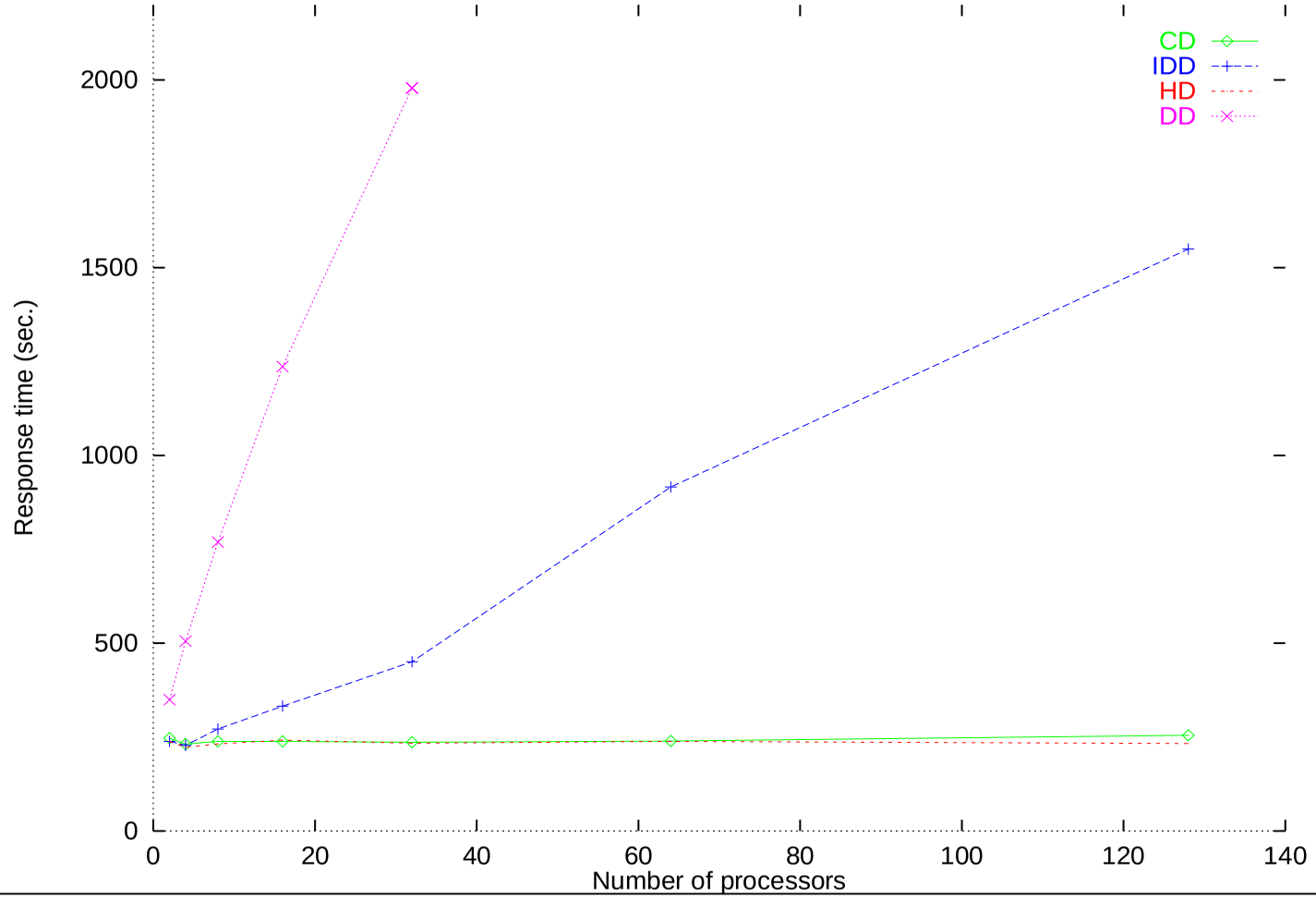
HD: Illustration



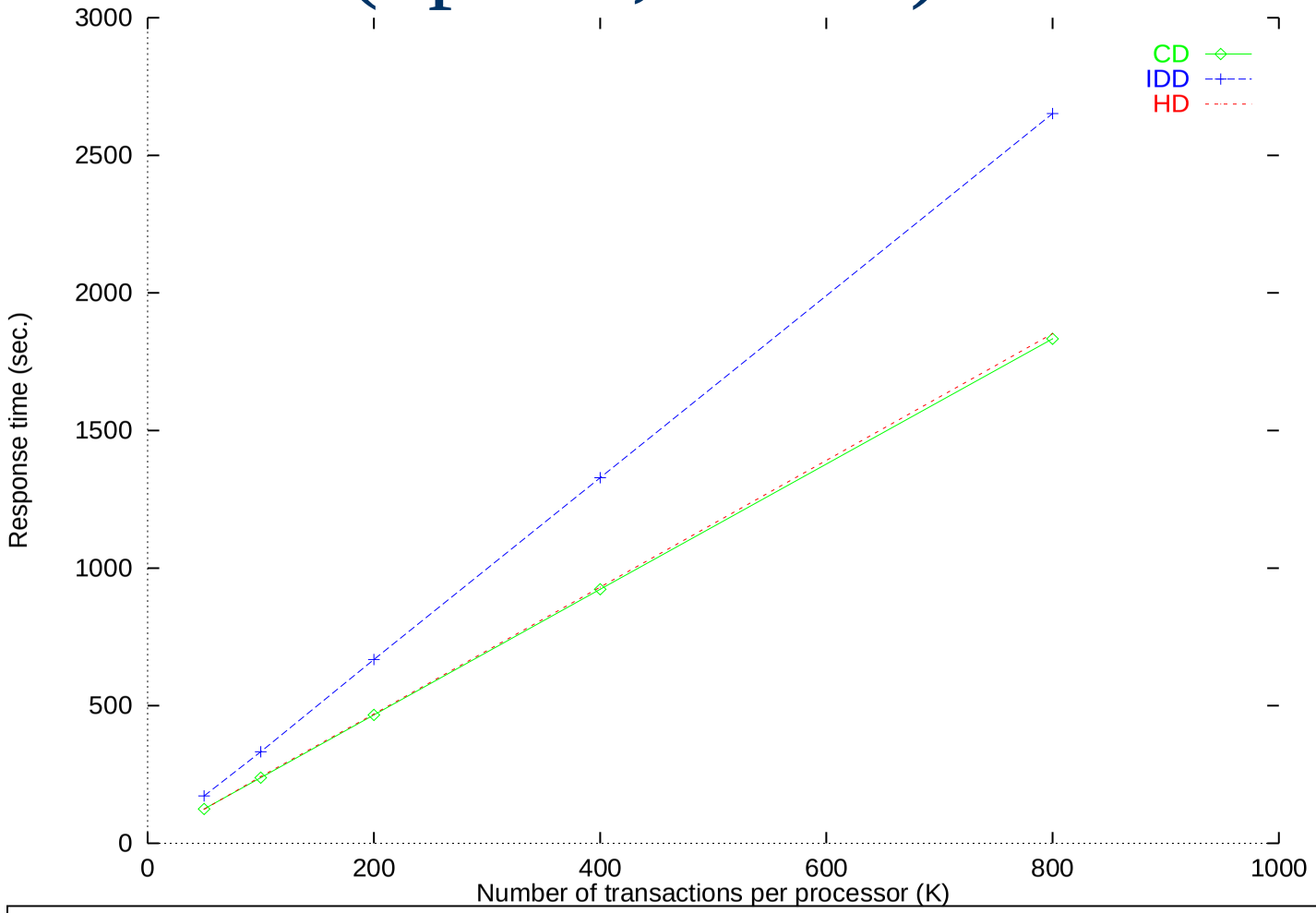
Parallel Association Rules: Experimental Setup

- 128-processor Cray T3D
  - 150 MHz DEC Alpha (EV4)
  - 64 MB of main memory per processor
  - 3-D torus interconnection network with peak unidirectional bandwidth of 150 MB/sec.
- MPI used for communications.
- Synthetic data set: avg transaction size 15 and 1000 distinct items.
- For larger data sets, multiple read of transactions in blocks of 1000.
- HD switch to CD after 90.7% of the total computation is done.

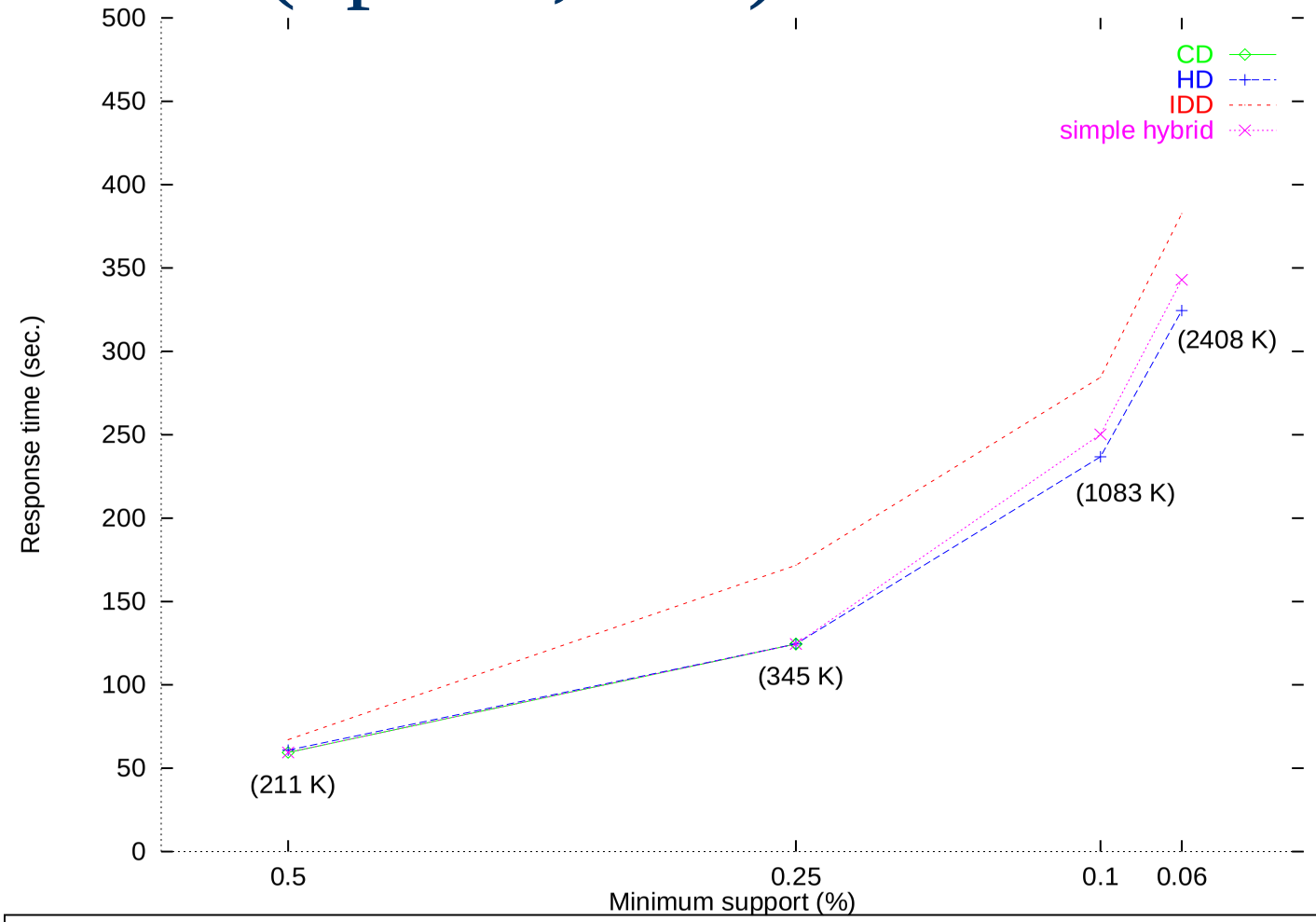
Parallel Association Rules: Scaleup Results (100K,0.25%)



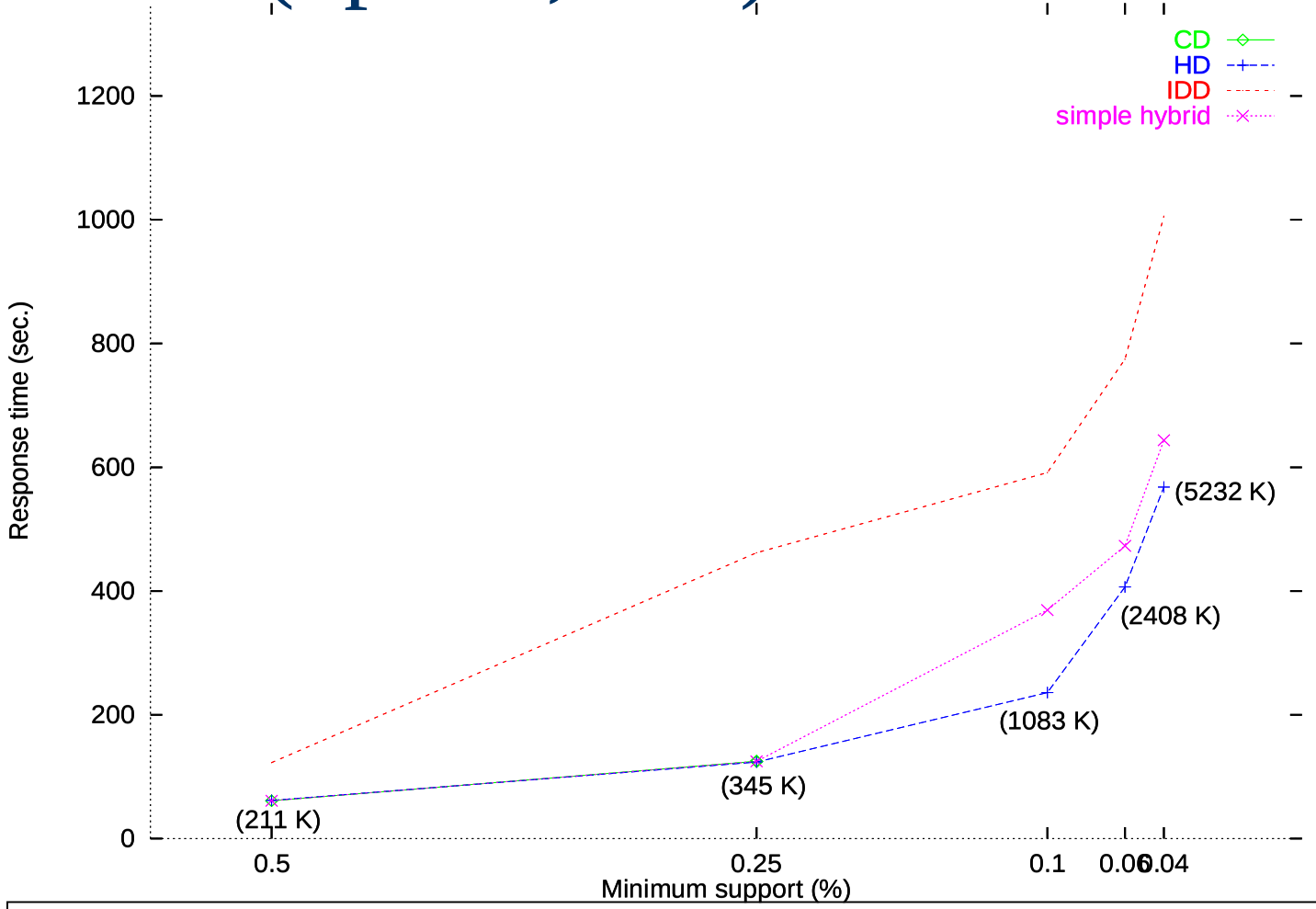
Parallel Association Rules: Sizeup Results (np=16,0.25%)



Parallel Association Rules: Response Time (np=16,50K)



Parallel Association Rules: Response Time (np=64,50K)



Parallel Association Rules: Minimum Support Reachable

Number of Processors	1	2	4	8	16	32	64
Successful Down to	0.25	0.2	0.15	0.1	0.06	0.04	0.03
Ran out of memory at	0.2	0.15	0.1	0.06	0.04	0.03	0.02

Parallel Association Rules: Processor Configuration in HD

64 Processors and 0.04 minimum support

Pass	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Configuration	**8 x 8**	**64 x 1**	**4 x 16**	**2 x 32**	**2 x 32**	**2 x 32**	**2 x 32**
# of Candidates	**351 K**	**4348 K**	**115 K**	**76 K**	**56 K**	**34 K**	**16 K**

Parallel Association Rules: Summary of Experiments

- HD shows the same linear speedup and sizeup behavior as that of CD.
- HD Exploits Total Aggregate Main Memory, while CD does not.
- IDD has much better scaleup behavior than DD

Eclat Approach

- Frequent itemset lattice
- Vertical or “inverted” tid-list format
- Support counting via intersections
- Lattice decomposition: break into subproblems
- Efficient search strategies
- Independent solution of subproblems

Example Database

DISTINCT DATABASE ITEMS

Jane Austen	Agatha Christie	Sir Arthur Conan Doyle	Mark Twain	P. G. Wodehouse
A	C	D	T	W

DATABASE

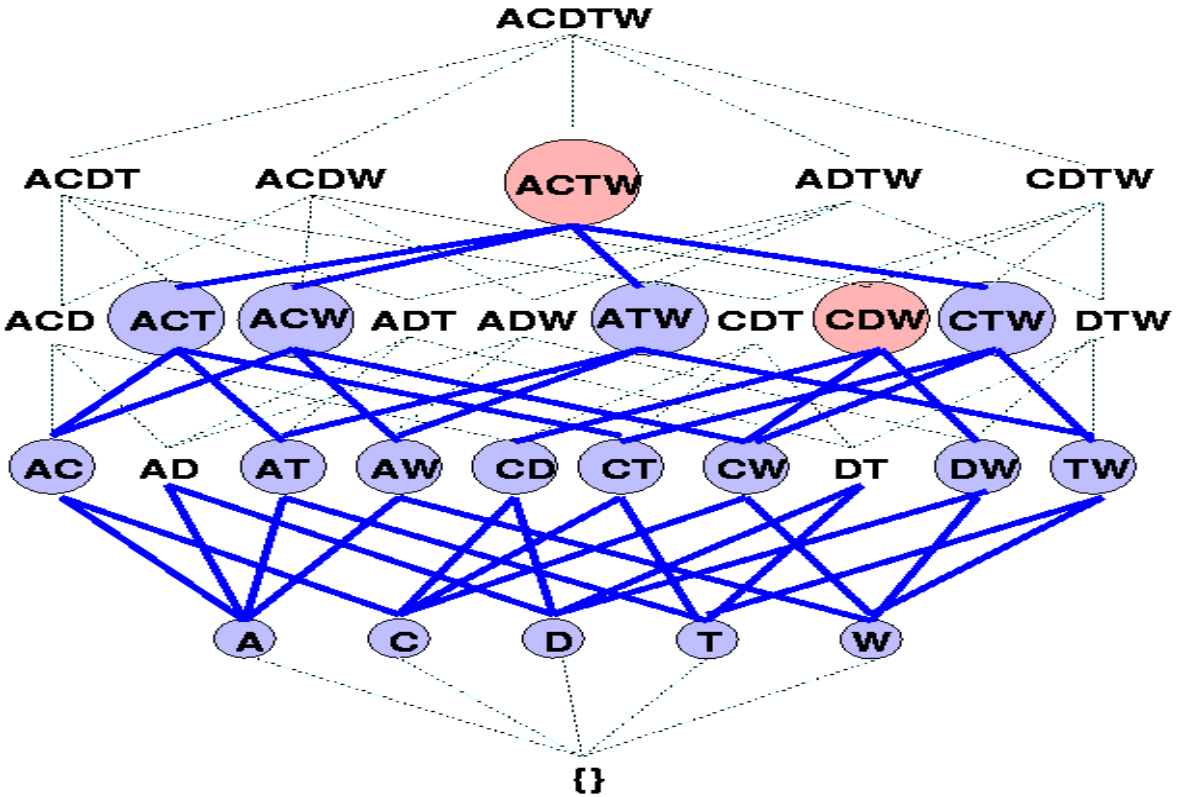
Transcation	Items
1	A C T W
2	C D W
3	A C T W
4	A C D W
5	A C D T W
6	C D T

ALL FREQUENT ITEM-SETS

Support	Item-Sets
100% (6)	C
83% (5)	W, CW
67% (4)	A, D, T, AC, AW CD, CT, ACW
50% (3)	AT, DW, TW, ACT, ATW CDW, CTW, ACTW
33% (2)	AD, DT, ACD, ADW CDT, ACDW
17% (1)	ADT, DTW, ACDT, ADTW CDTW, ACDTW

MAXIMAL ITEM-SETS (MIN SUPPORT = 50%): ACTW, CDW

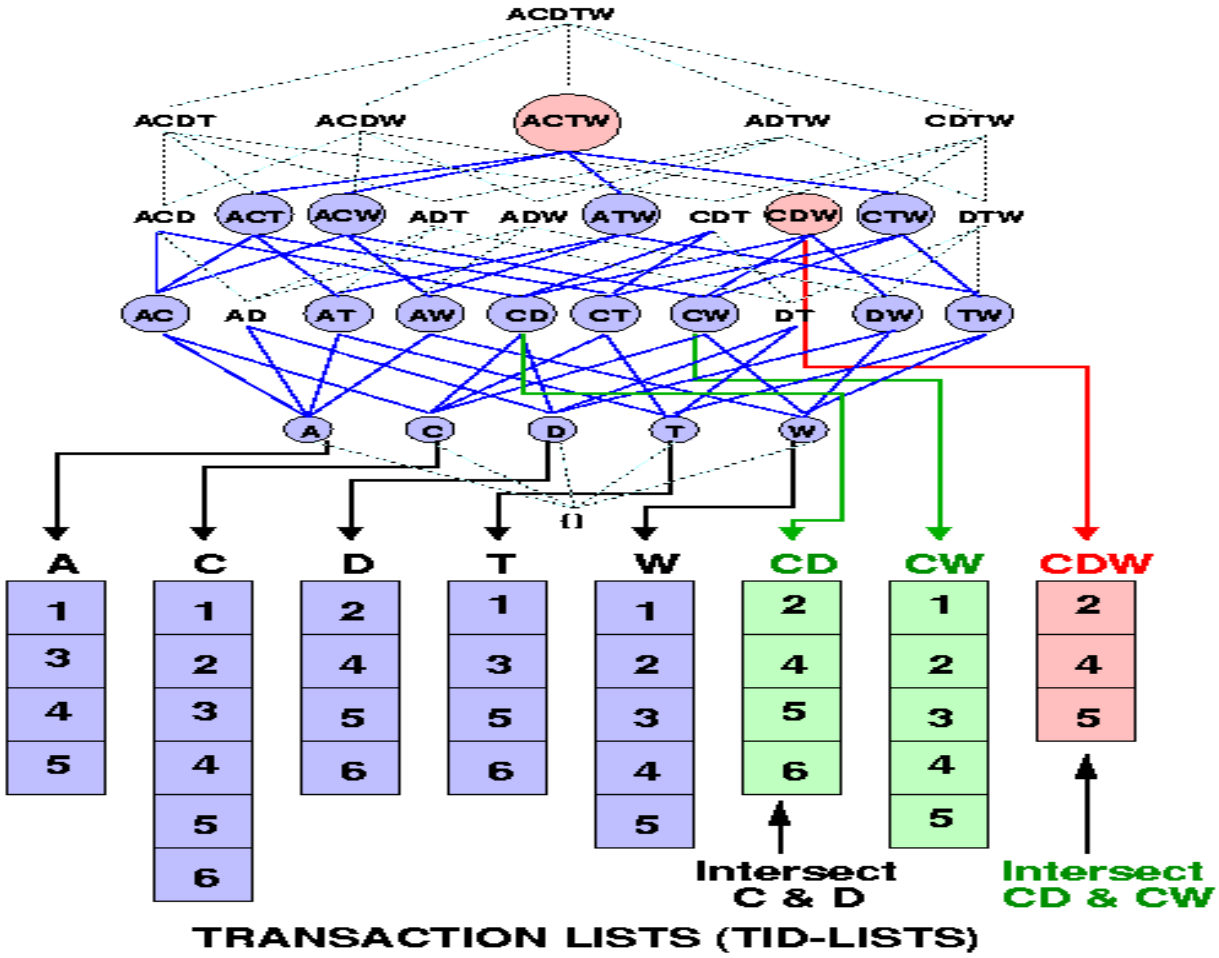
Frequent Itemset Lattice



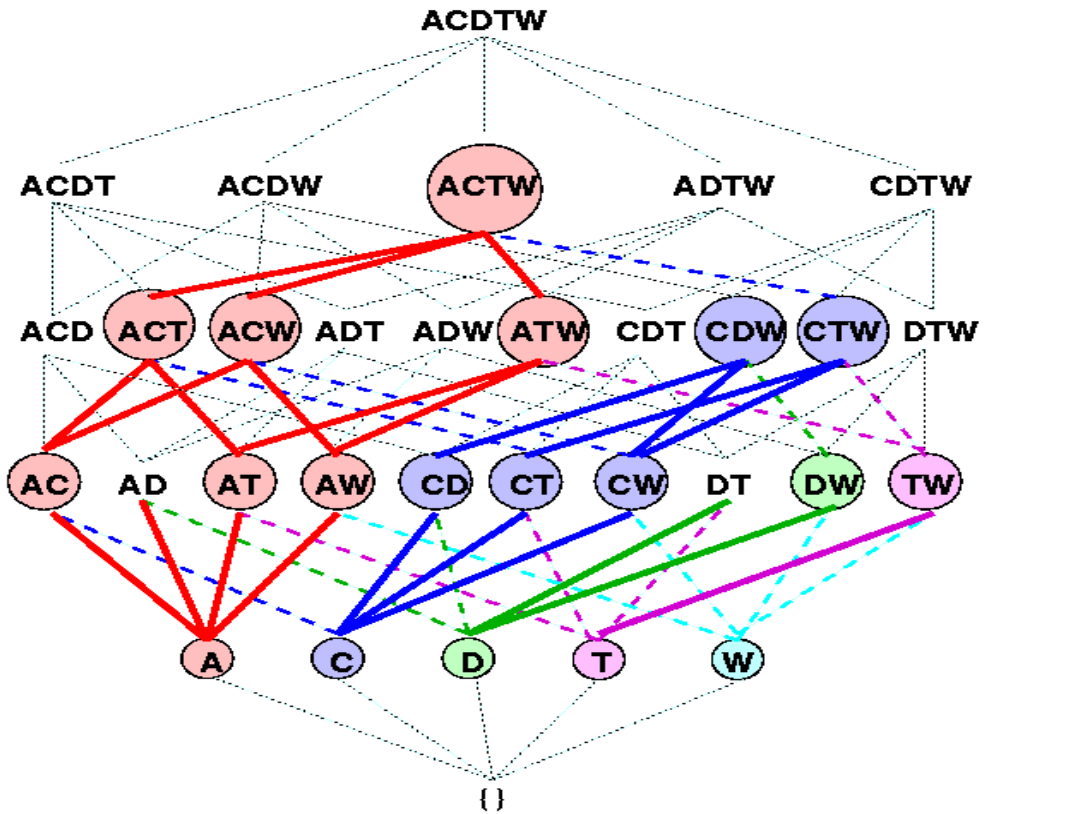
DOWNWARD CLOSED ON ITEM-SET SUPPORT

MAXIMAL FREQUENT ITEM-SETS: ACTW, CDW

Eclat: Support Counting



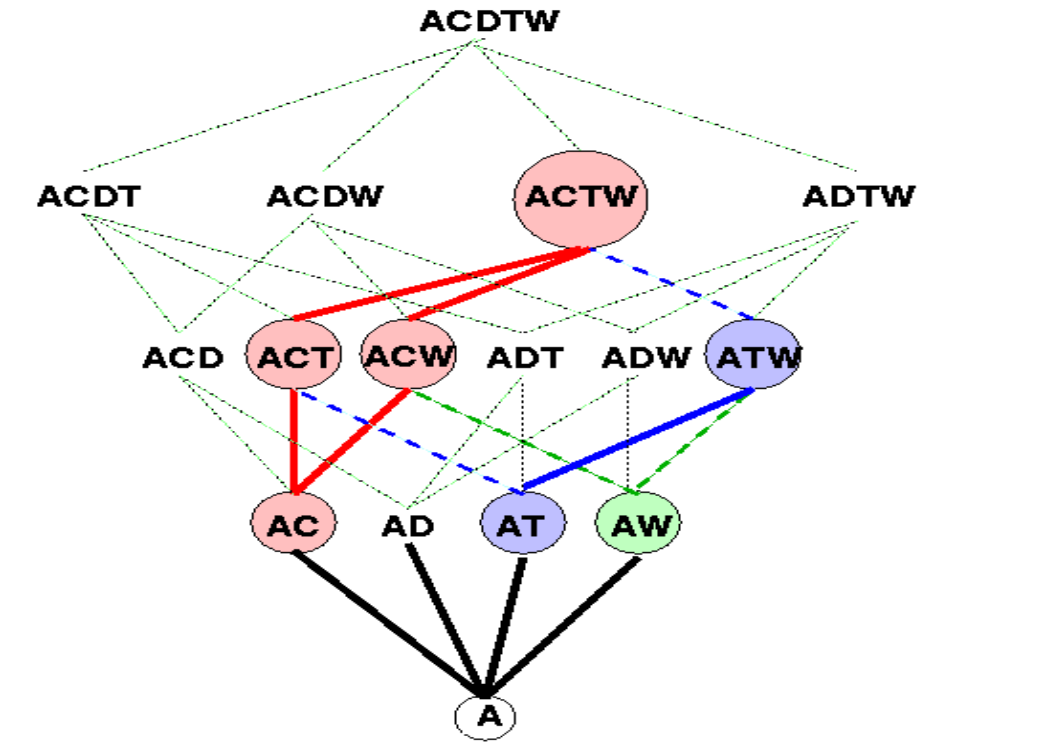
Eclat: Lattice Decomposition



Lattice Search Strategies

- Bottom-up
  - Level-wise like Apriori
- Top-down
  - Start with largest element. If frequent, done! Else look at subsets
- Hybrid
  - Find long frequent/maximal itemsets
  - Find remaining using bottom-up search

Bottom-up Lattice Search

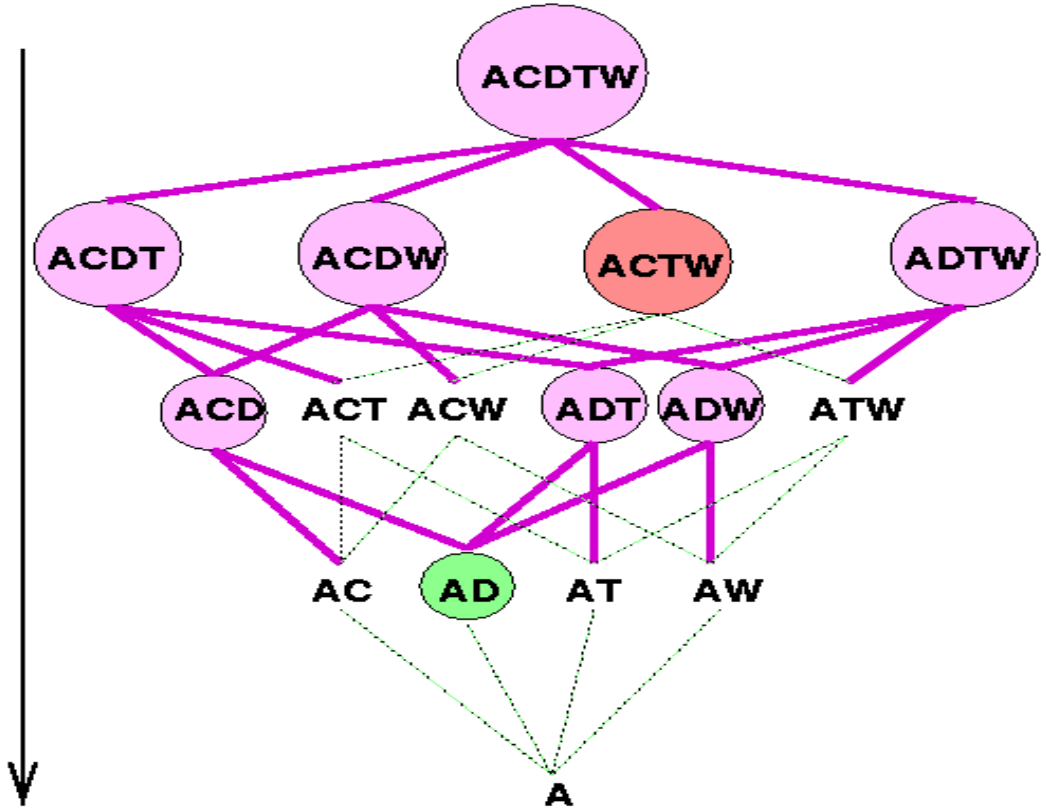


NEW EQUIVALENCE CLASSES

[AC] = { AC, ACT, ACW, ACTW }
[AT] = { AT, ATW }
[AW] = { AW }

KNOWN MAXIMAL FREQUENT ITEMSETS: CDW, CTW

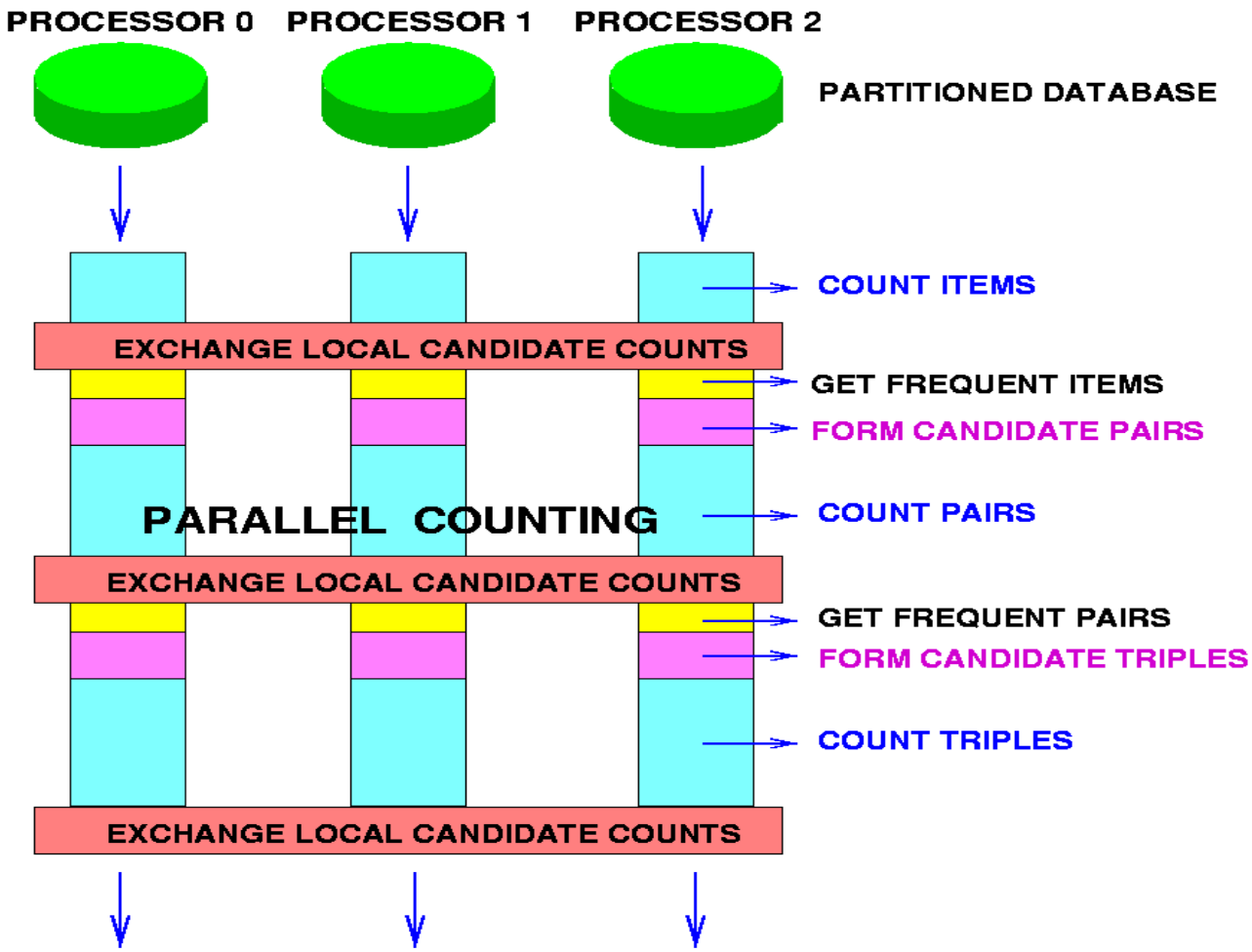
Top-down Lattice Search



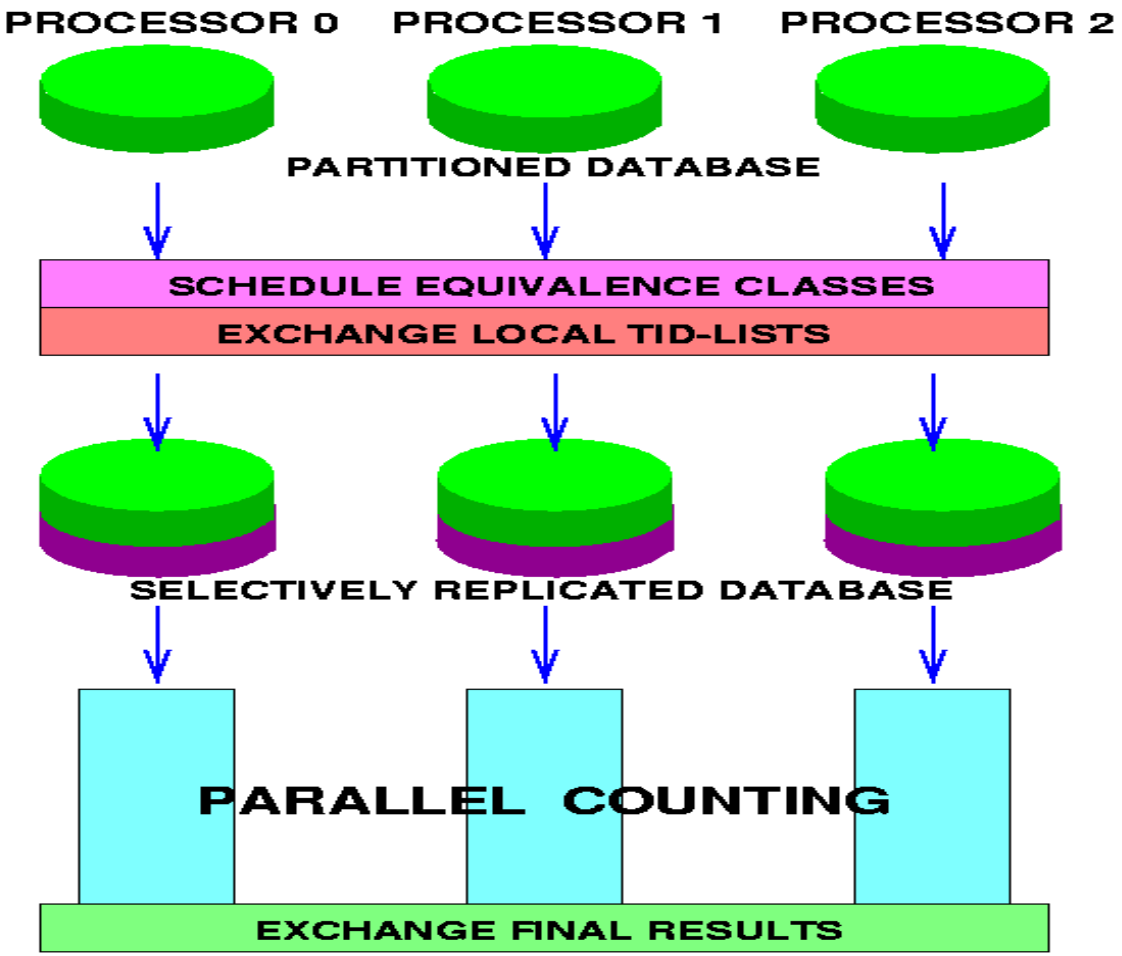
Parallel Eclat/MaxEclat

- Partition tidlists among processors
- Schedule equivalence classes
- Selectively replicate database
- Solve Asynchronously/Independently
- Algorithm Space
  - Eclat (Bottom-up)
  - MaxEclat (Hybrid)
- Effective aggregate memory utilization

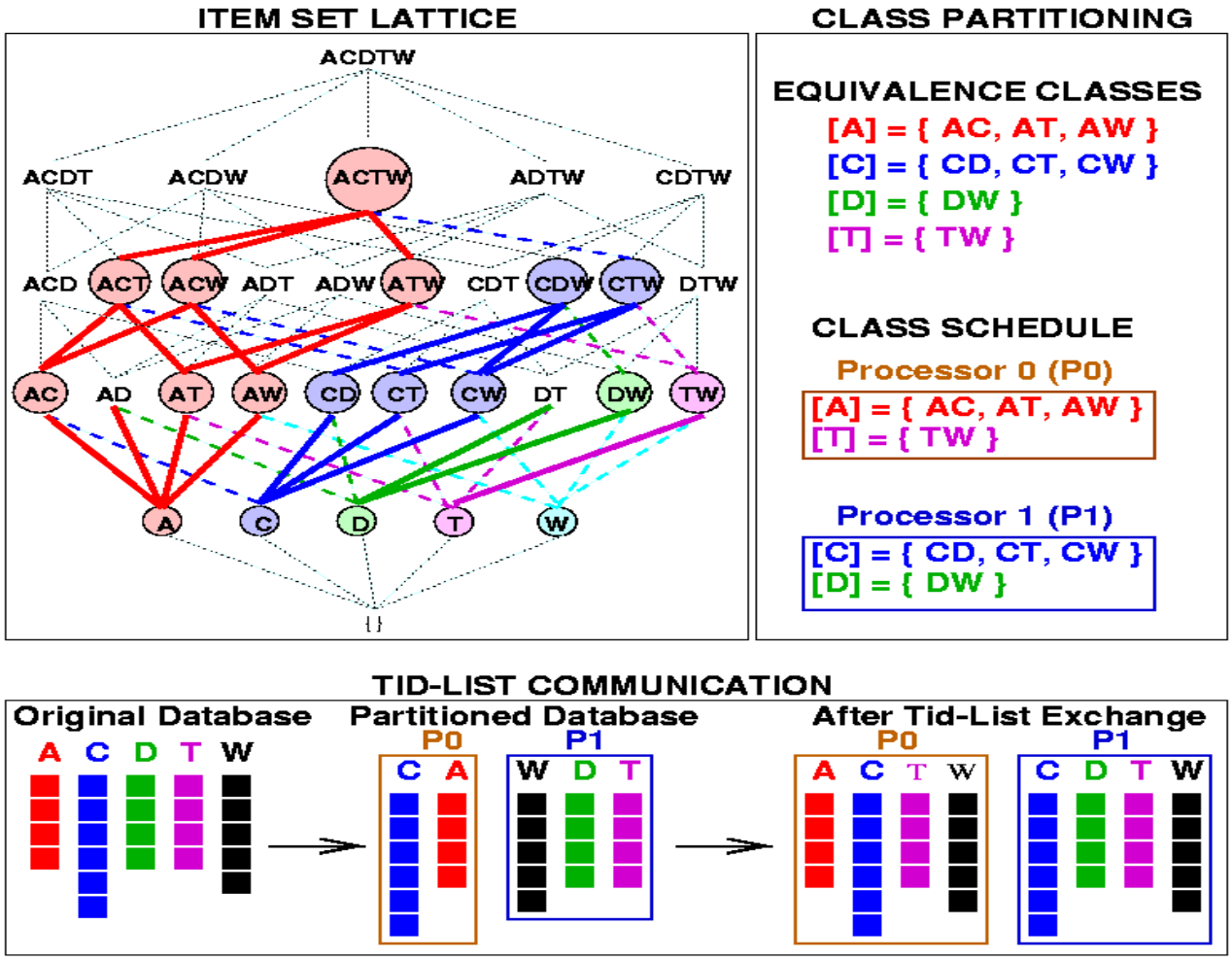
Count Distribution



Parallel Eclat



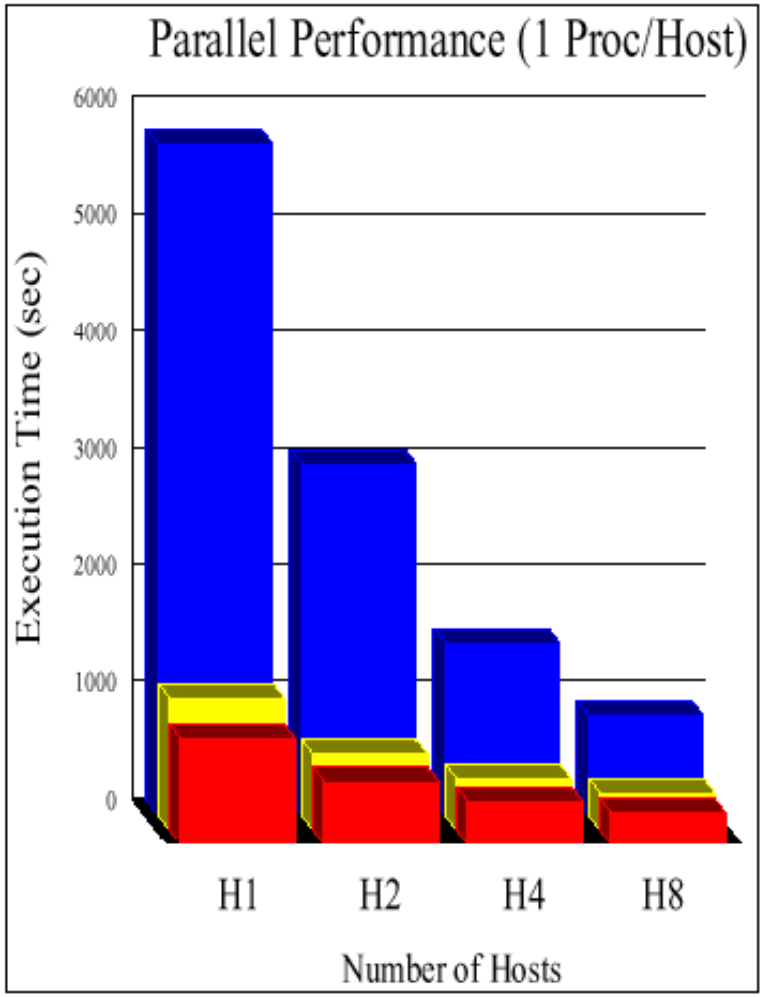
Parallel Eclat Algorithm

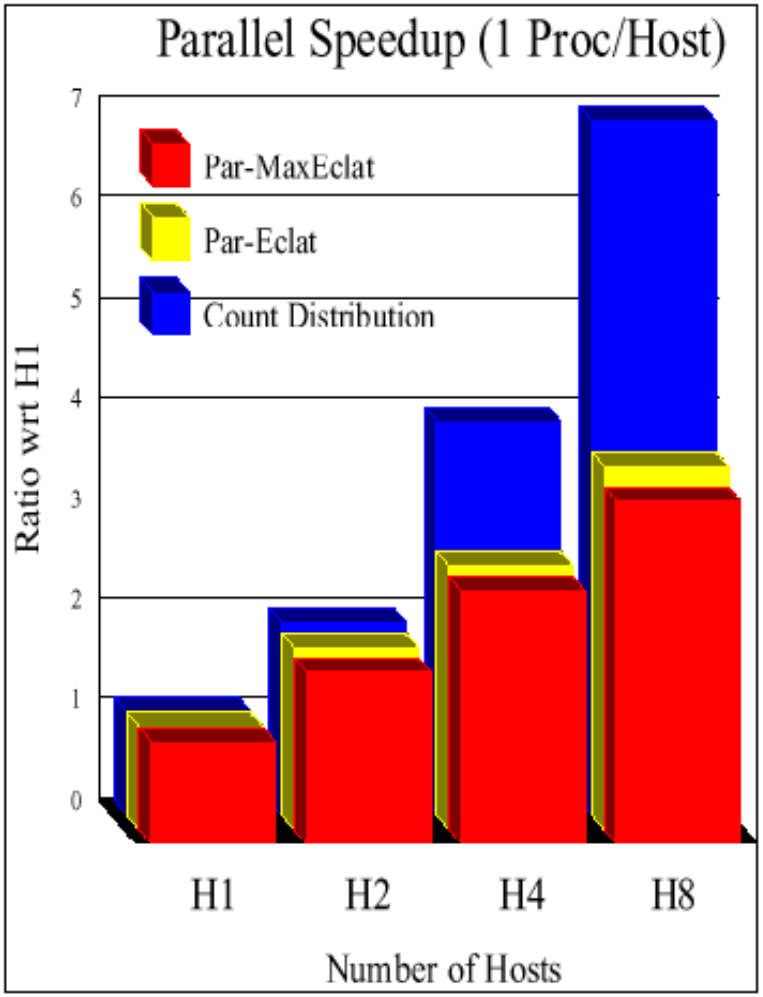


Eclat Experiments

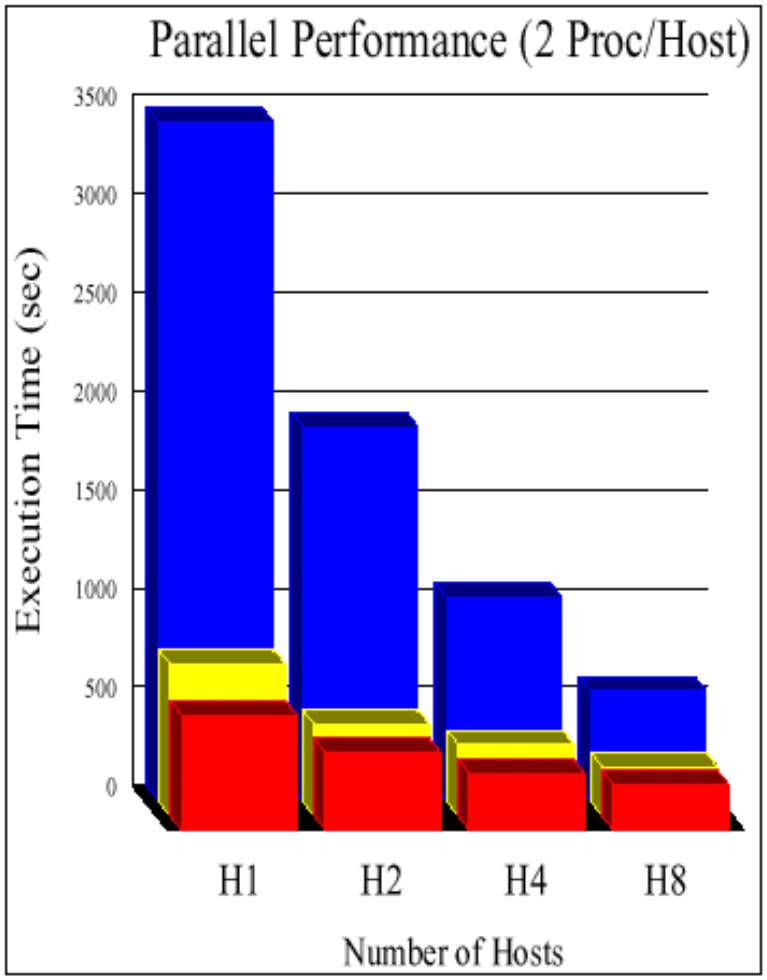
- Database: T20.I6.D4550K
  - 1000 Items
  - 4550,000 Transactions
- Machine: Eight 4-way SMP nodes
  - Digital's Memory Channel (5μs,30Mb/s)
  - 233 Mhz, 256 MB, 1MB L2 Cache
- Hierarchical Parallelization
  - Message passing + SMP

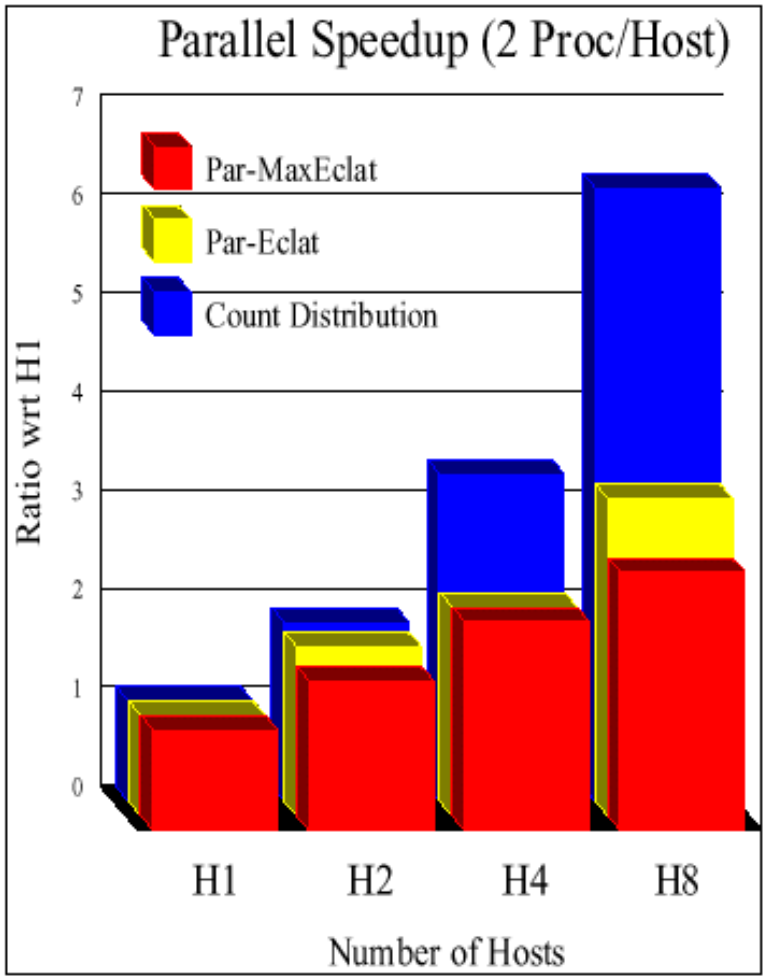
Parallel Performance





Parallel Performance





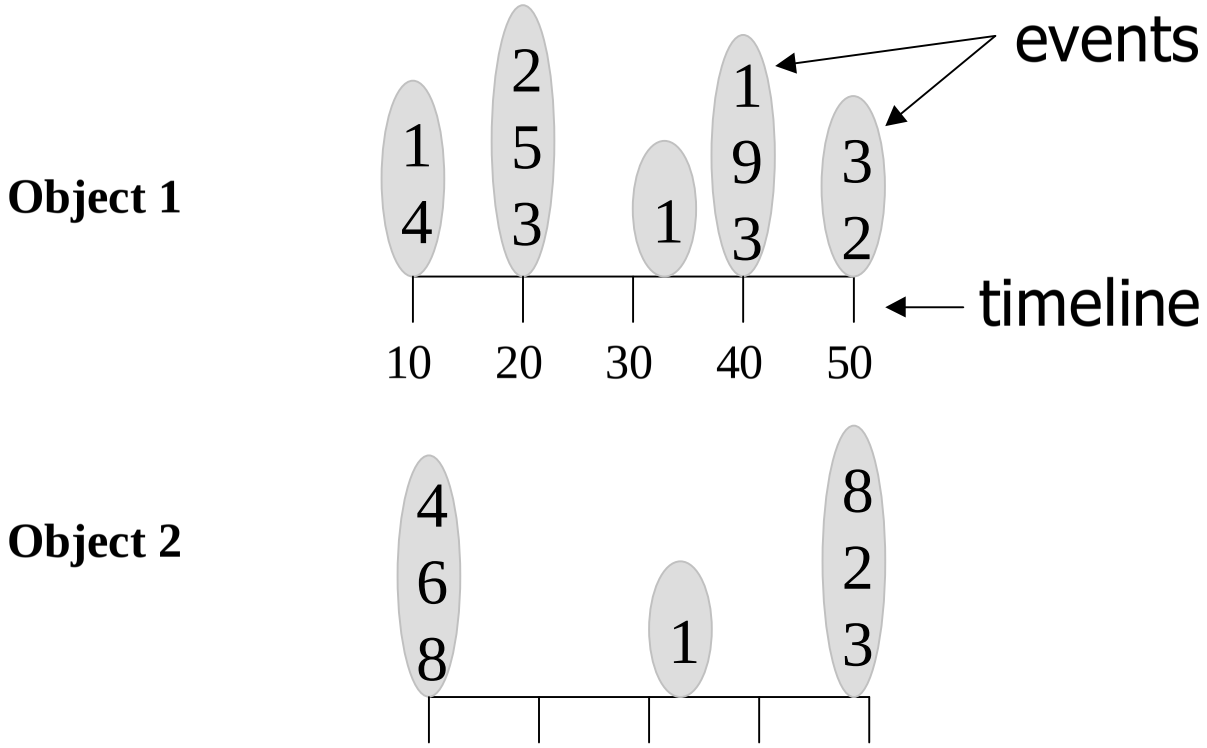
Tutorial overview

- Overview of KDD and data mining
- Parallel design space
- Classification
- Associations
- Sequences
- Clustering
- Future directions and summary

Discovering Sequential Associations

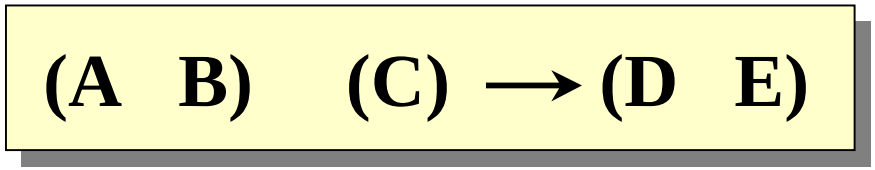
Given:

A set of objects with associated event occurrences.

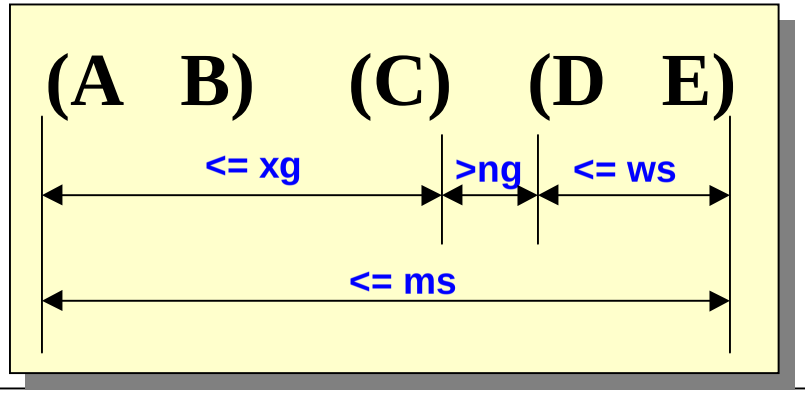


Sequential Pattern Discovery: Definition

- Given is a set of *objects*, with each object associated with its own *timeline of events*, find rules that predict strong sequential dependencies among different events.



- Rules are formed by first disovering patterns. Event occurrences in the patterns are governed by timing constraints.



Sequential Patterns: Complexity

- Much **higher computational complexity** than association rule discovery.
  - **O($m^k 2^{k-1}$) number of possible sequential patterns having *k* events, where *m* is the total number of possible events.**
- Time constraints offer some pruning. Further use of **support based pruning contains complexity**.
  - A subsequence of a sequence occurs at least as many times as the sequence.
  - A sequence has no more occurrences than any of its subsequences.
  - Build sequences in increasing number of events. [GSP algorithm by Agarwal & Srikant]

Apriori-like Approach

- Make multiple scans
- Join frequent patterns to generate new candidate sequences
- Example:
  - (2 3) (4 6) (7) joins with (3) (4 6) (7) (8) to create (2 3) (4 6) (7) (8)
  - (1 2) with (2 3) creates (1 2 3)
  - (1) with (2) creates (1) (2) and (1 2)

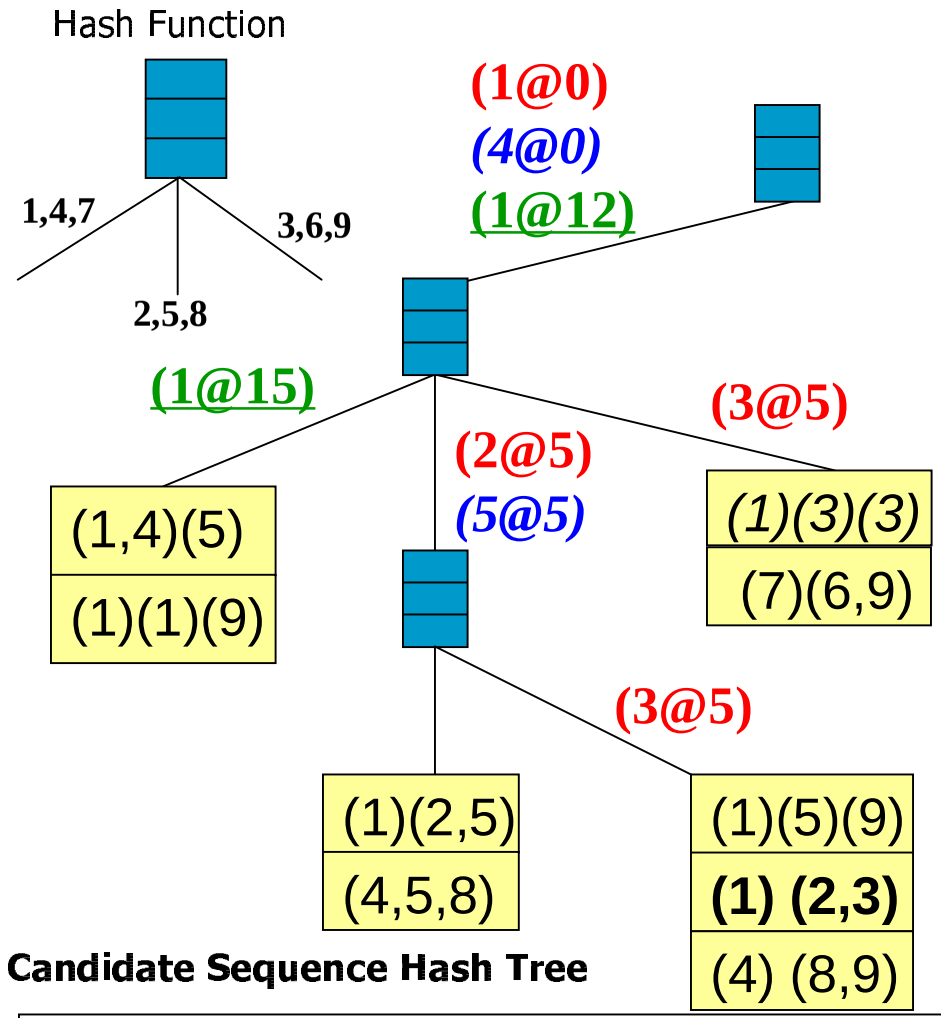
Sequential Apriori: Count operation

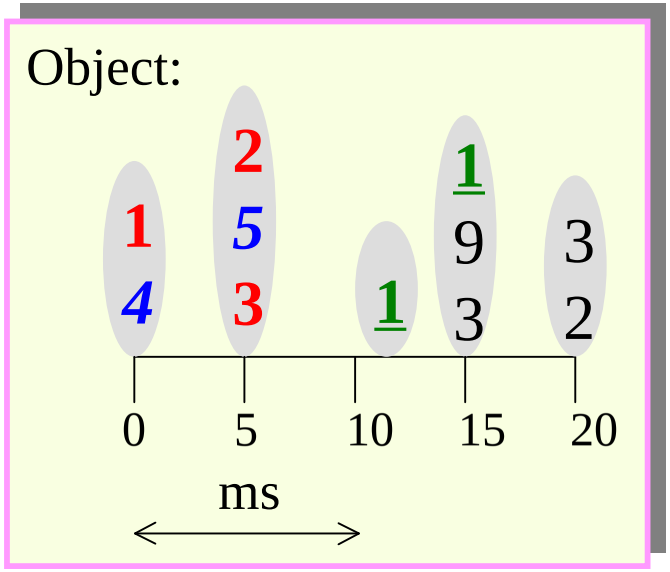
- Various possibilities for counting occurrences of a sequence in an object's timeline
  - Count only one occurrence per object.
  - Count the number of span-size windows the sequence occurs in.
  - Count the number of distinct occurrences of a sequence:
    - Each event-timestamp pair considered at most once.
    - Each counted occurrence has at least one new event-timestamp pair.

Sequential Apriori: Count operation (contd..)

- Hash Tree used for fast search of candidate occurrences. Similar to association rule discovery, except for following differences.
  - *Every* event-timestamp pair in the timeline is hashed at the root.
  - Events eligible for hashing at the next level are determined by the maximum gap (xg), window size (ws), and span (ms) constraints.

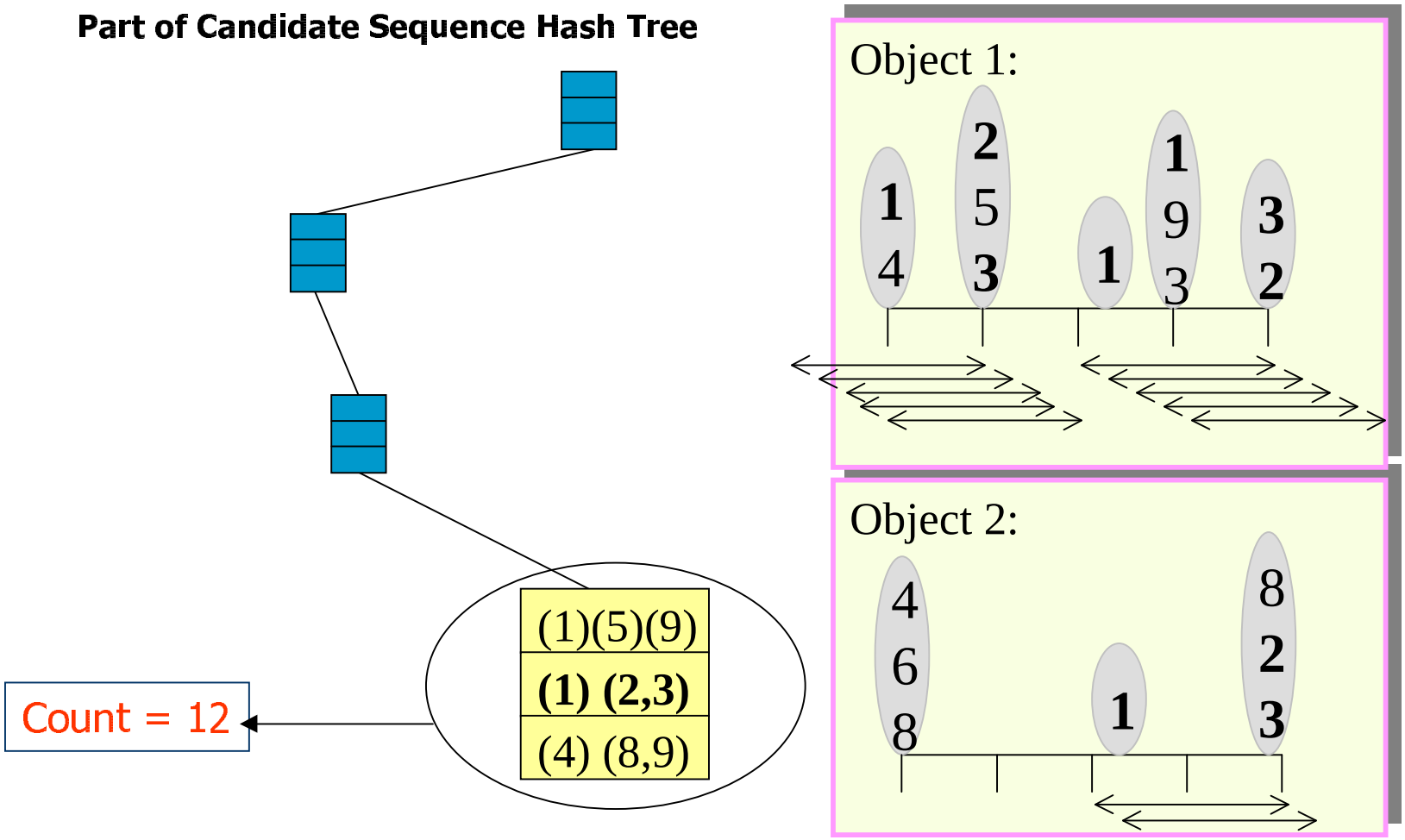
Sequence Hash Tree for Fast Access





- Four Interesting Paths:
  - 1@0, 2@5, 3@5
  - 1@0, 3@5
  - *4@0, 5@5*
  - 1@12, 1@15

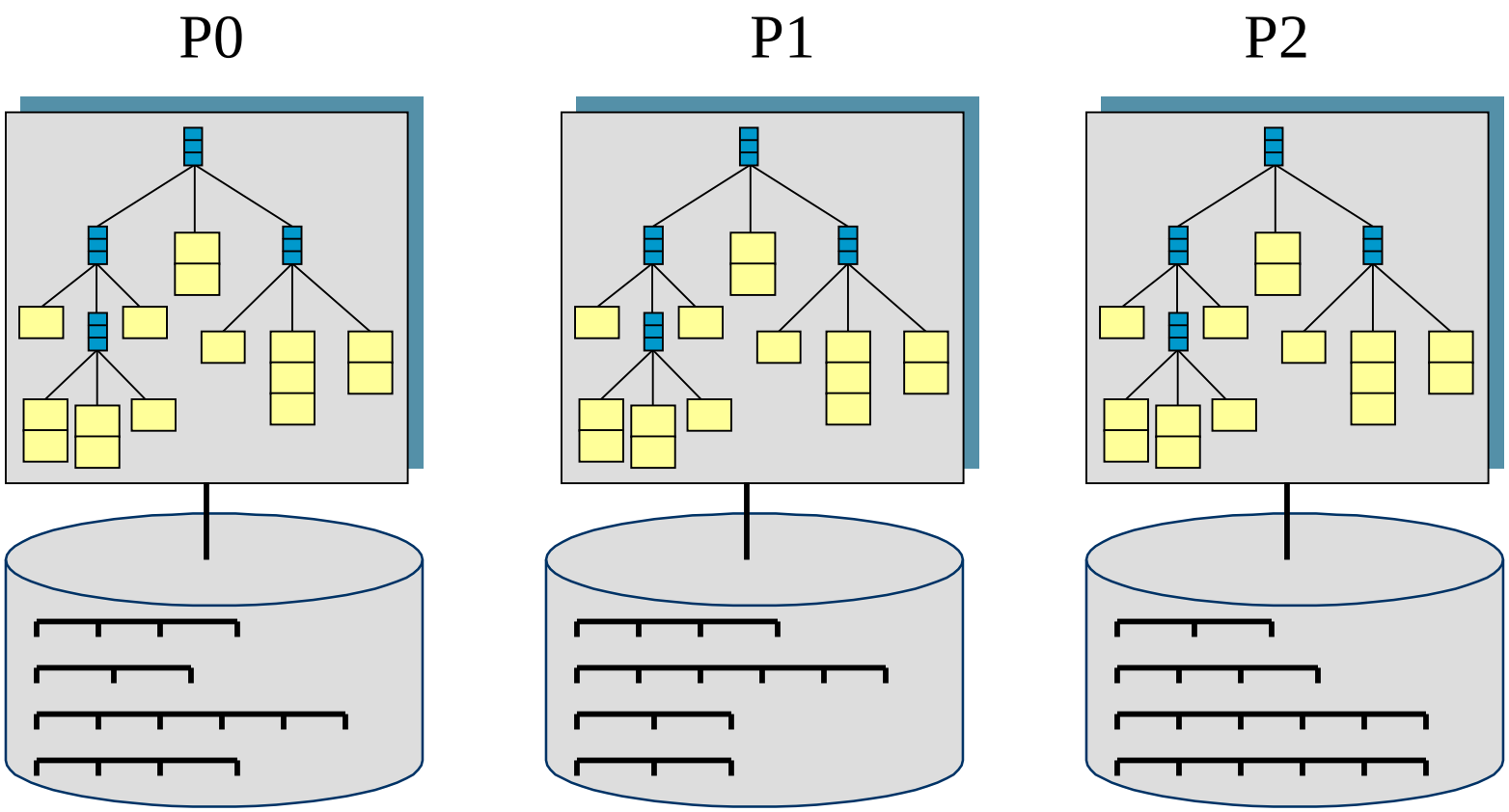
Counting Leaf Level Candidates



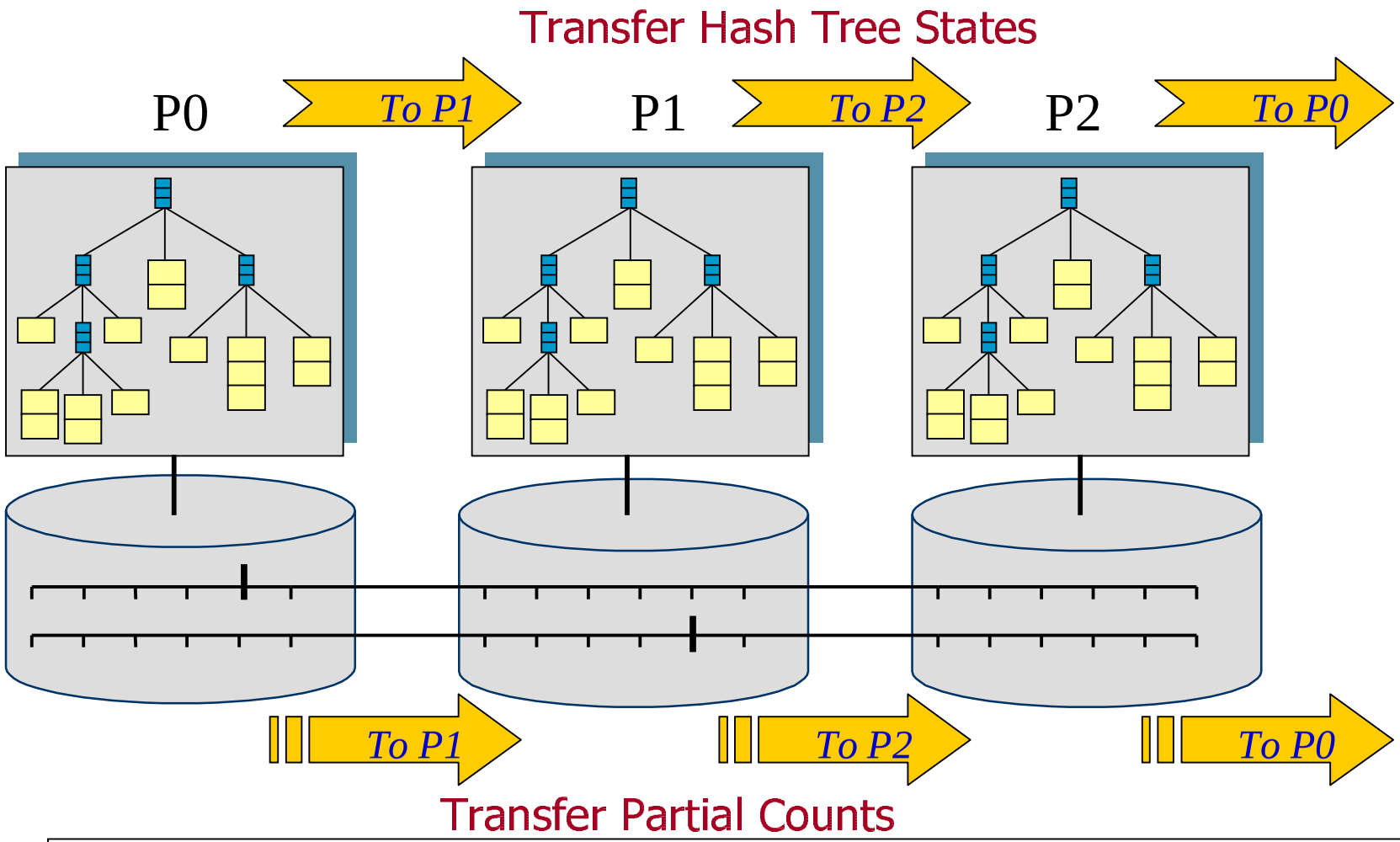
Parallel Sequential Associations

- Need:
  - Enormity of Data.
  - Memory and Disk limitations of serial algorithms running on single processor.
- Can algorithms for non-sequential associations be extended easily?
  - Sequential Nature gives rise to complex issues:
    - ***Longer Timelines***
    - ***Large Span values***
    - ***Large Number of Candidates***

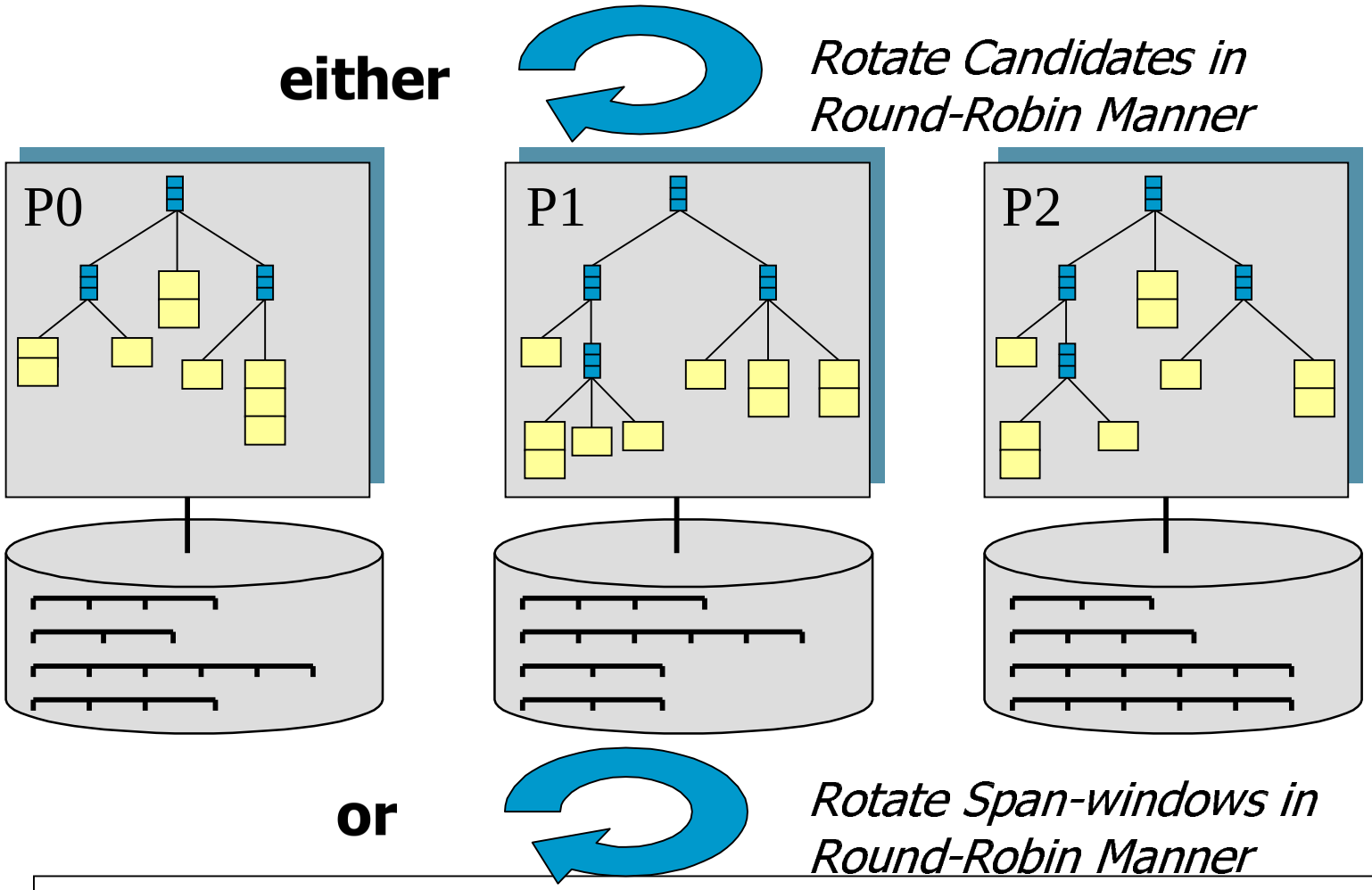
Parallel Sequential Associations: Event Distribution (EVE)



EVE Algorithm: Challenging Case



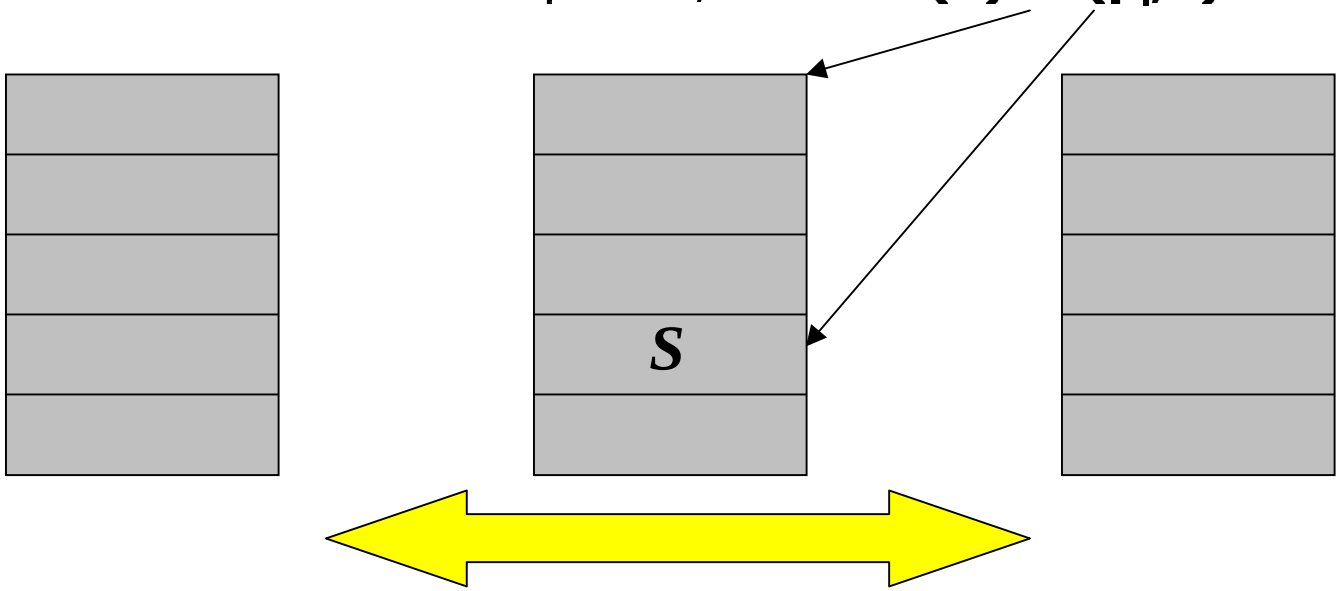
Event and Candidate Distribution (EVECAN)



EVECAN: Parallelizing Candidate Generation

- Candidates Stored in a Distributed Hash Table
- Hash Function:

  Candidate Sequence, S => **h(S) = (p_i, l)**



Scalable Communication Mechanism to construct and probe

The SPADE Approach

- Search space is a lattice
- Use “inverted” vertical lists
- Find support via list intersections
- Break large search space into small independent suffix-based chunks
- Process each chunk recursively using breadth/depth-first search
- Takes only a few data scans

Example Database

DATABASE

Customer-Id	Transaction-Time	Items
1	April 1	C
1	April 7	D
1	April 25	T W
2	April 2	A C
2	April 5	D
2	April 20	C T W
3	April 15	A C D
3	April 27	D T W

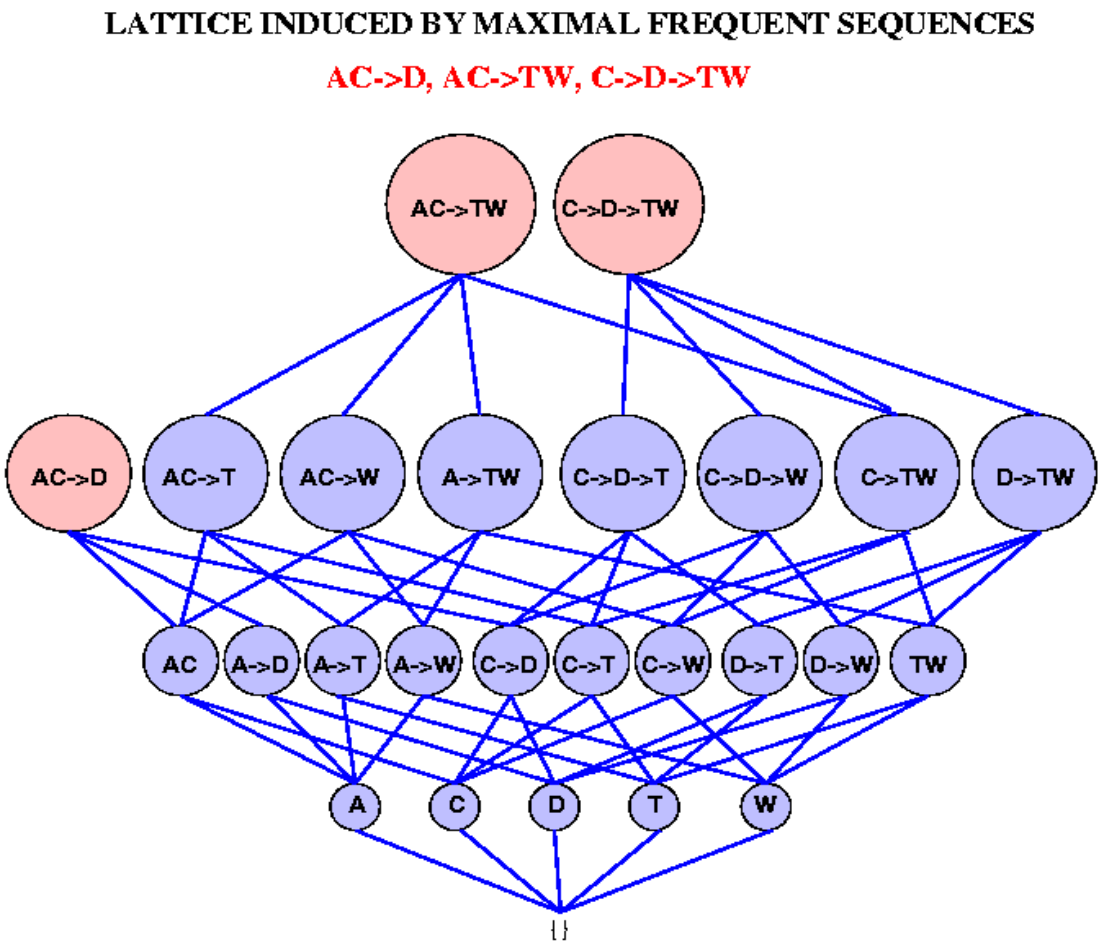
FREQUENT SEQUENCES (75%)

Support	Sequences
100% (3)	C, D, T, W, C->D, C->T C->W, D->T, D->W TW, C->TW, D->TW
75% (2)	A, AC, A->D, A->T A->W, AC->T, AC->D AC->W, A->TW AC->TW C->D->TW

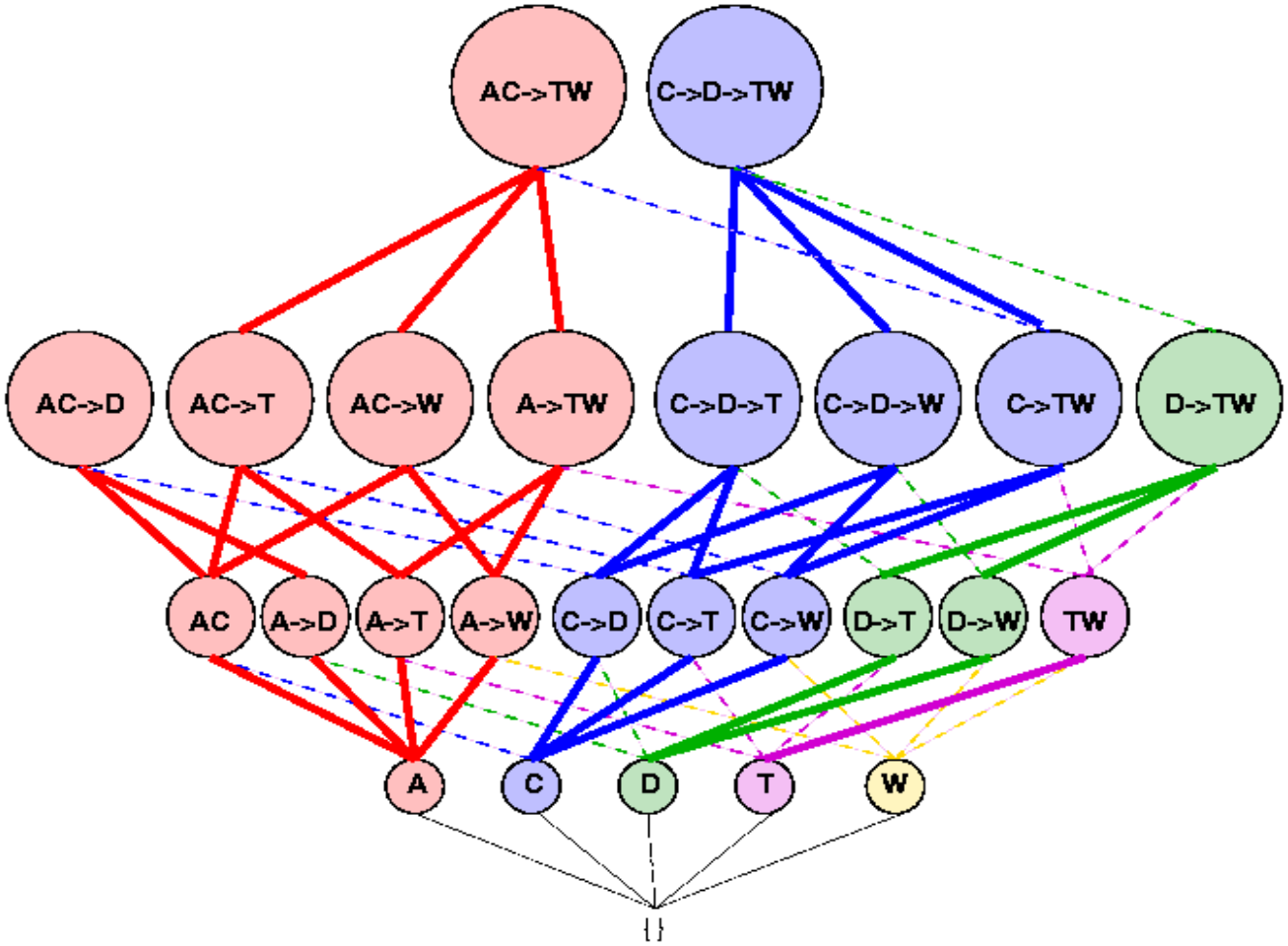
MAXIMAL FREQUENT SEQUENCES

AC->D AC->TW C->D->TW

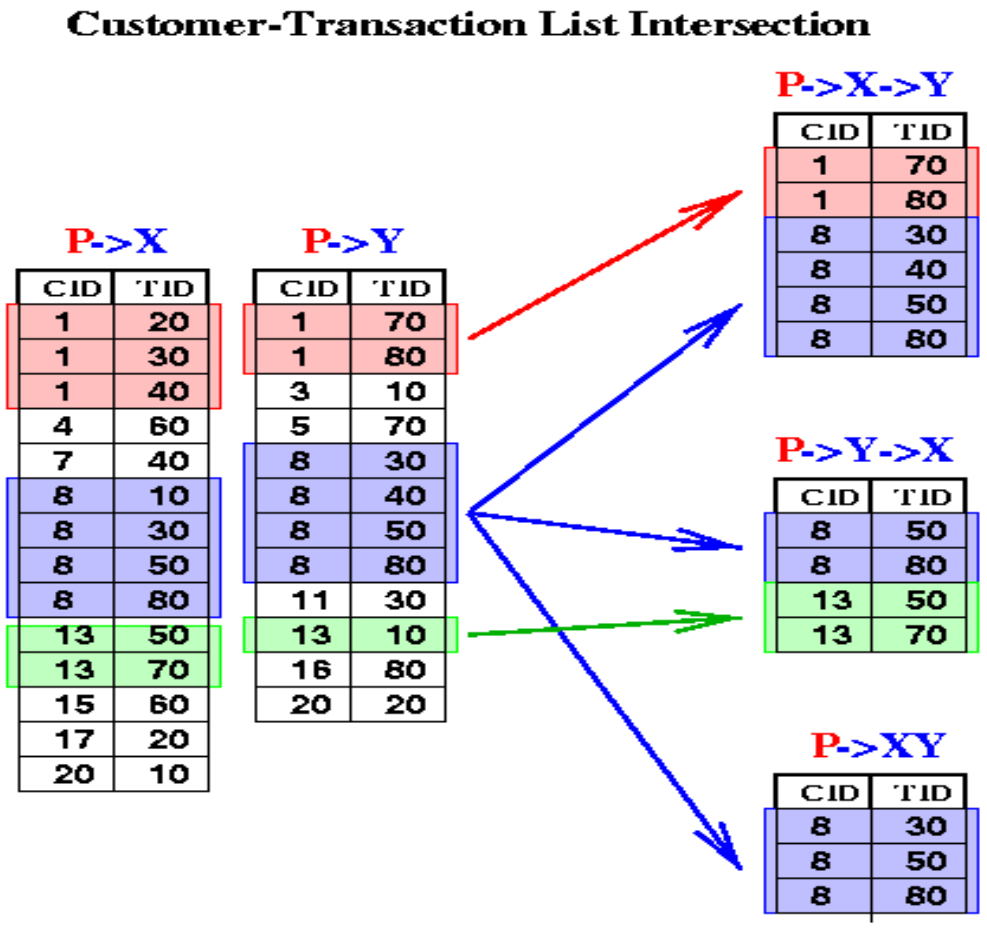
Frequent sequence lattice



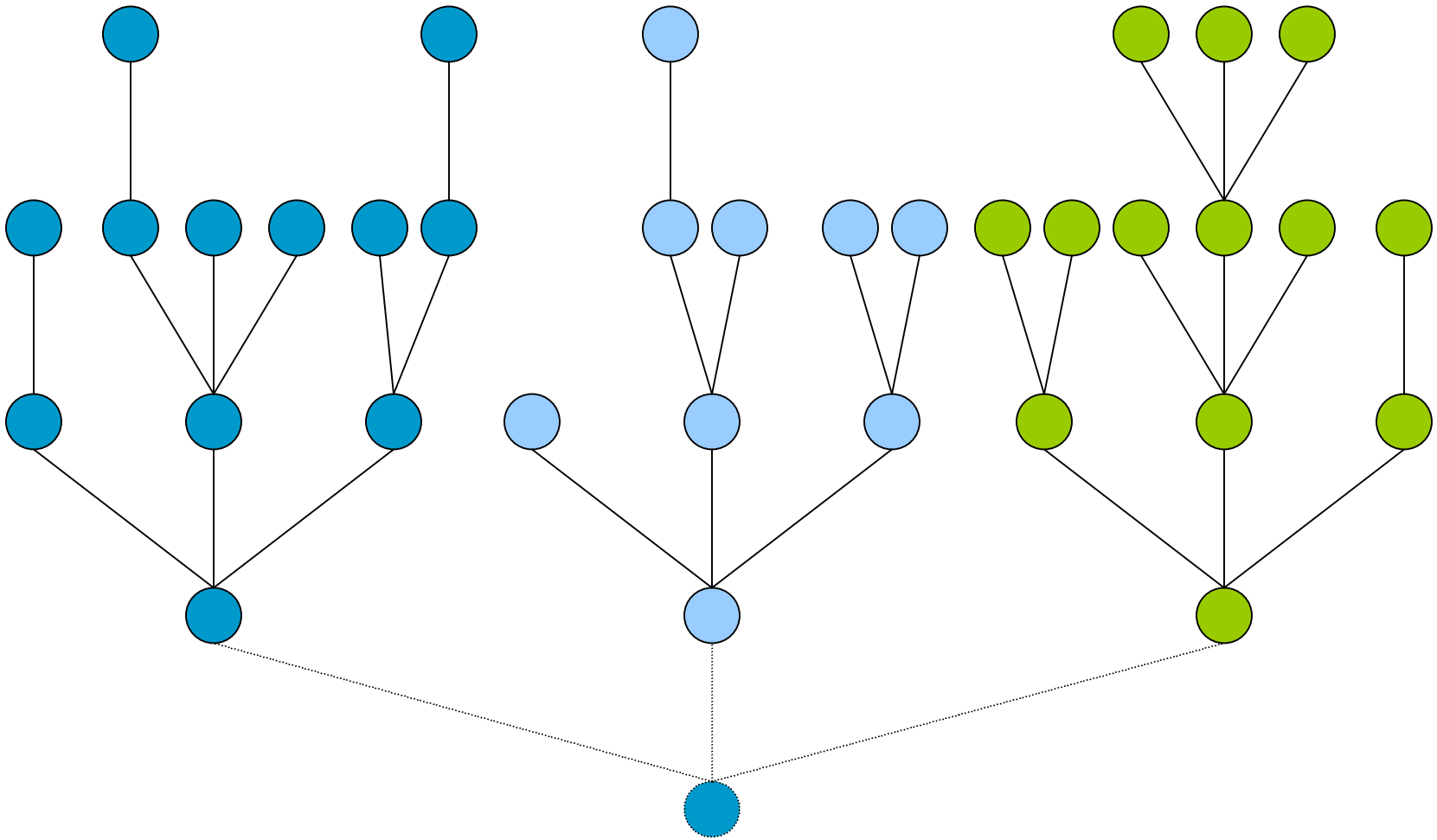
Sequence lattice decomposition



Temporal Joins



Dynamic Computation Tree



Parallel Design Space

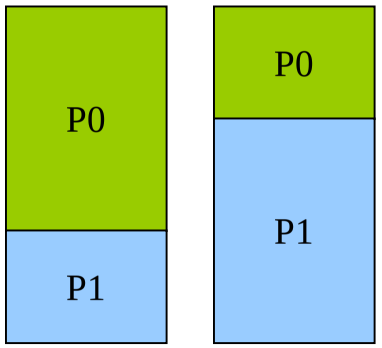
- Data parallelism: within a class
  - Idlist parallelism
    - Single idlist join
    - Level-wise idlist join
  - Join parallelism
- Task parallelism: between classes
  - pSPADE
- Static vs. dynamic load balancing

Idlist Data Parallelism

- Single idlist join
  - Split idlists on cid range
  - Each proc intersects over its cid range
  - Barrier synchronization for each join

P0: cid in range 1-500
P1: cid in range 501-1000

Parallel Idlist Intersection

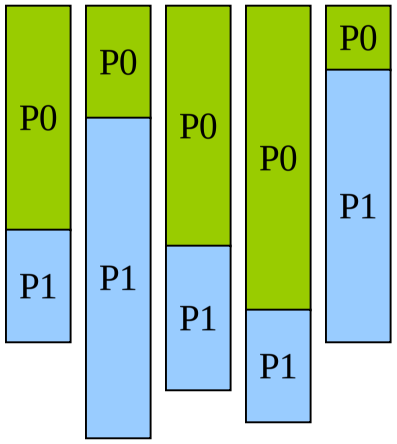


Idlist Data Parallelism

- Level-wise idlist join
  - Process all classes at current level
  - Each proc still intersects over its local DB
  - Barrier synchronization and sum-reduction for each level

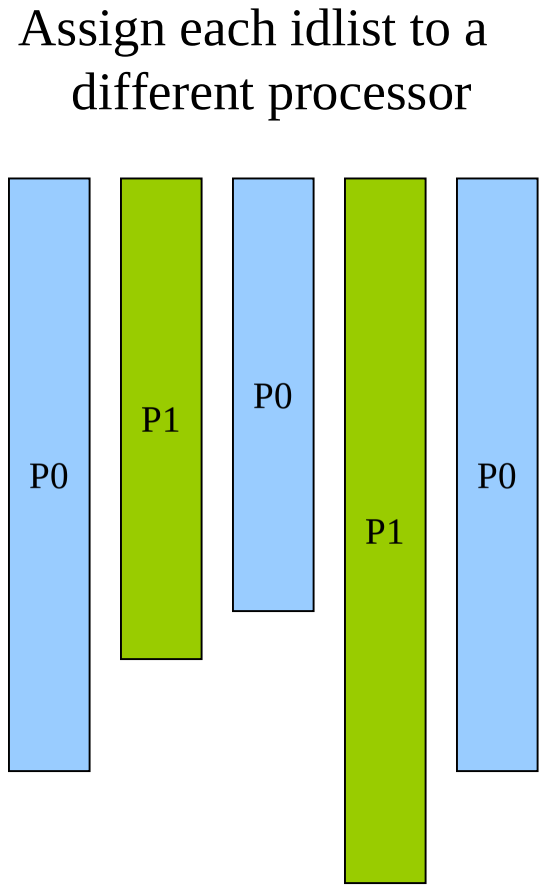
P0: cid in range 1-500
P1: cid in range 501-1000

Parallel Pair-Wise Intersections



Join Data Parallelism

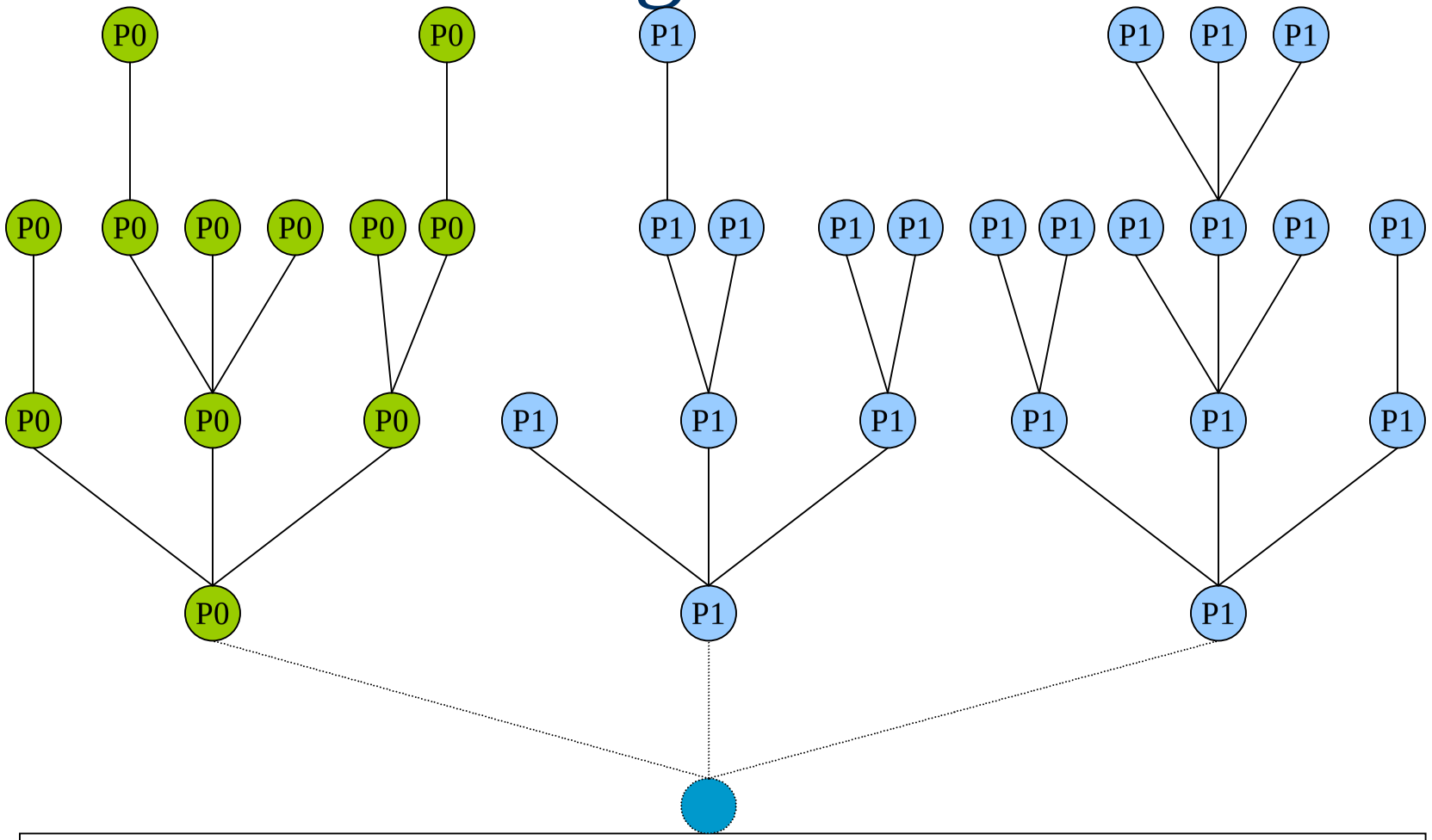
- Each processor performs different intersections within a class
- Barrier after self-joining within a class, before processing classes at next level
- # synchronizations is between Single and Level-wise parallelism



Task Parallelism: Static Load Balancing

- Given level 1 classes, C1, C2, C3, …
- Assign weights W1, W2, W3, … on the class size (# sequences in the class)
- Greedy scheduling of entire classes
  - Sort classes on weight (descending)
  - Assign class to proc with least total weight
- Each proc solves assigned classes asynchronously
- Hard to get accurate work estimate

Static and Dynamic Load Balancing



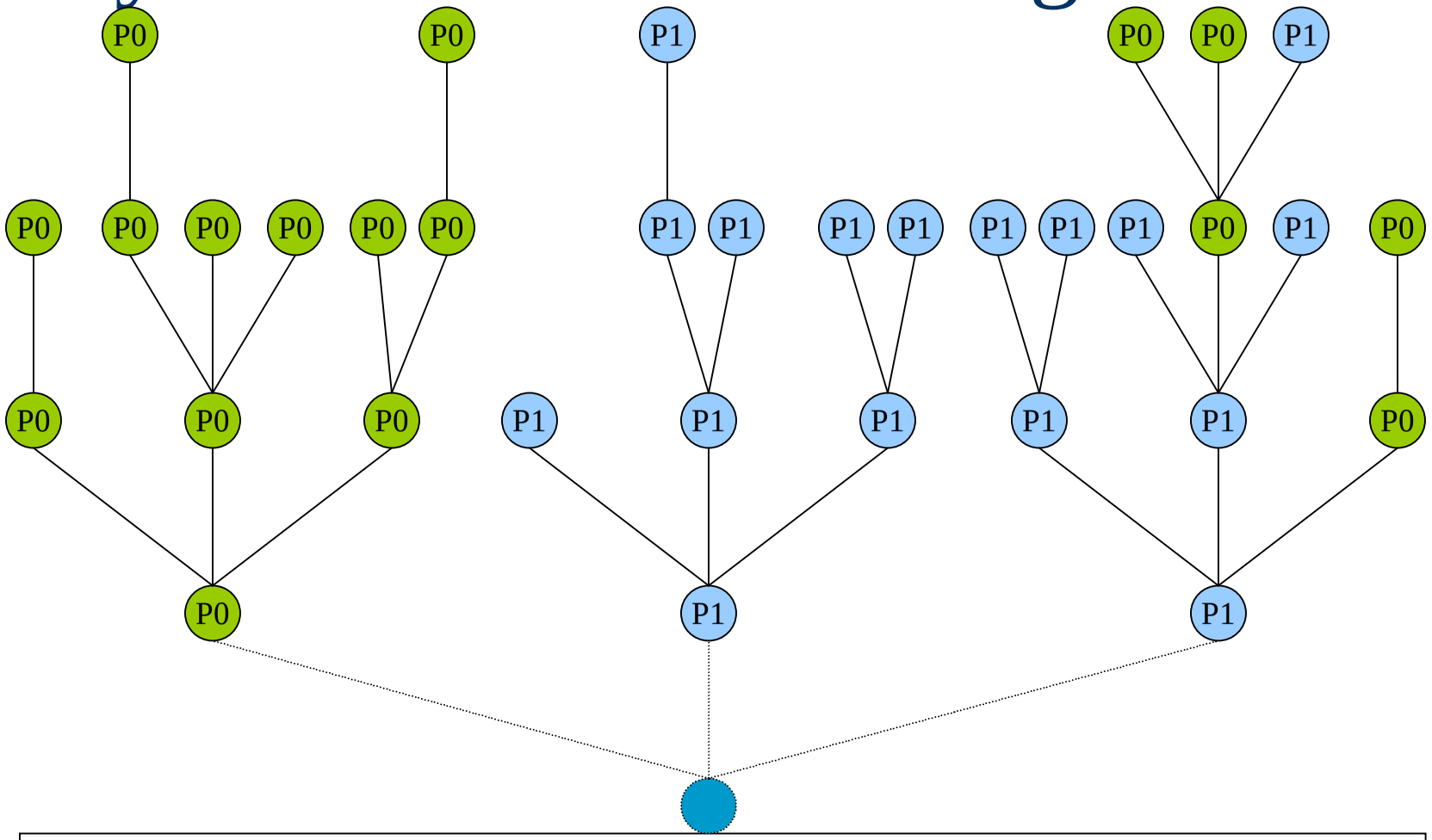
Task Parallelism: Dynamic Load Balancing

- Given level 1 classes, C1, C2, C3, … and their weights
- Sort classes on weight (descending) and insert in a logical task queue
- A proc atomically grabs the first available class and solves it completely
- Repeat until no more classes in queue
- Uses only inter-class parallelism
- Queue contention negligible since classes are coarse-grained

Task Parallelism: Recursive DLB (pSPADE)

- Process available classes in parallel
- Worst case: P-1 procs free, 1 proc busy
- Classes sorted on weight, last class usually small (but it could be large)
- Provide mechanism for free procs to join busy group
- At each level, get free procs, insert the classes into a shared task queue; process available classes in parallel

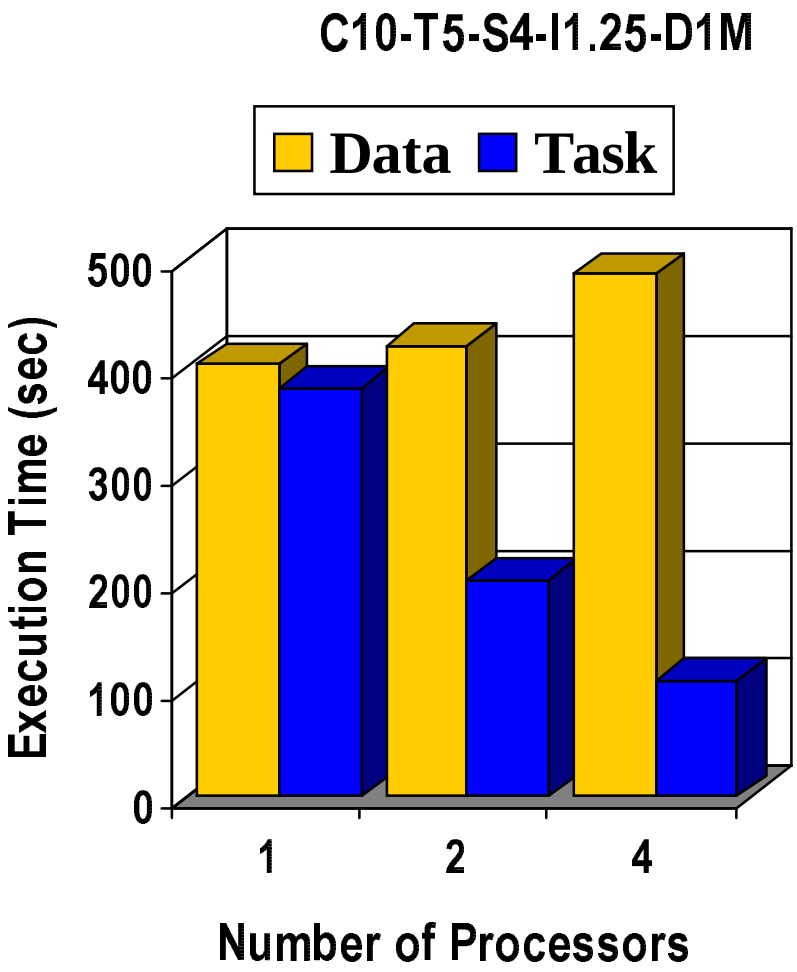
pSPADE: Recursive Dynamic Load Balancing

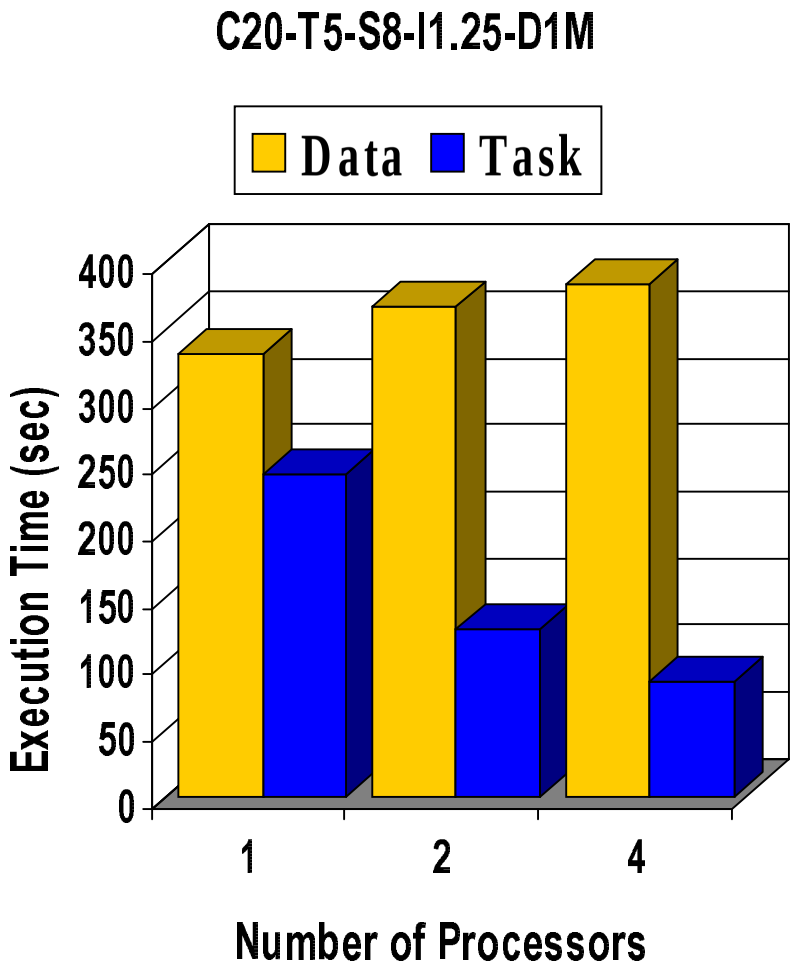


Experimental Setup

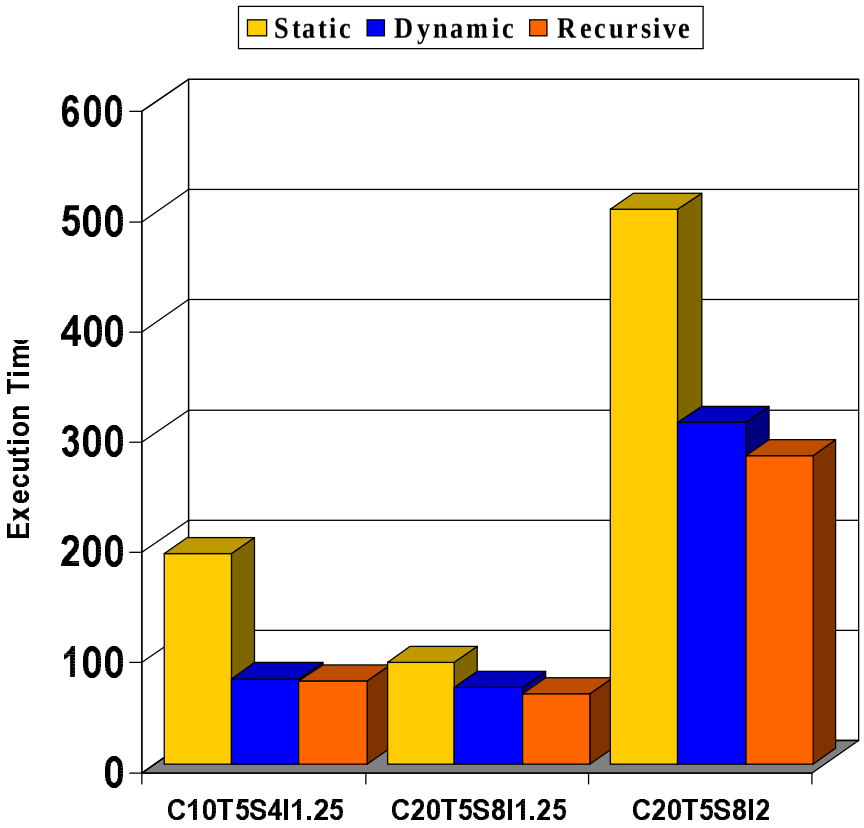
- SGI Origin 2000
  - 12 195Mhz R10K MIPS processors
  - 4MB cache, 2GB memory
- Synthetic datasets
  - C: number of transactions/customer
  - T: average transaction size
  - S/I: average sequence/itemset size
  - D: number of customers

Data vs. Task Parallelism



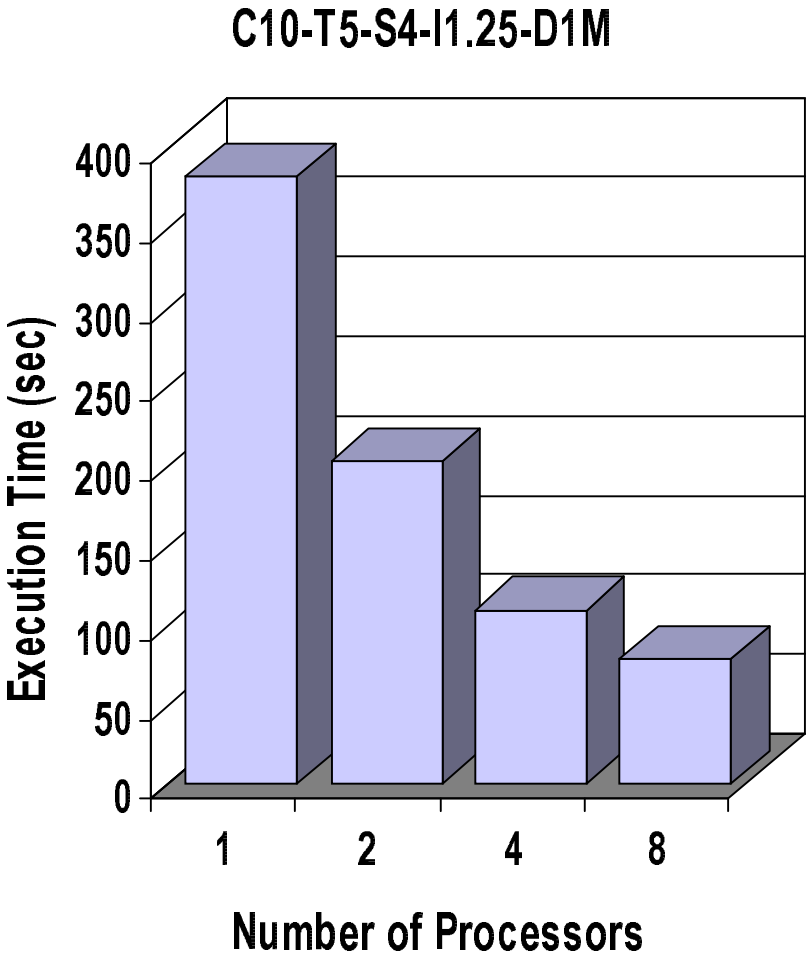


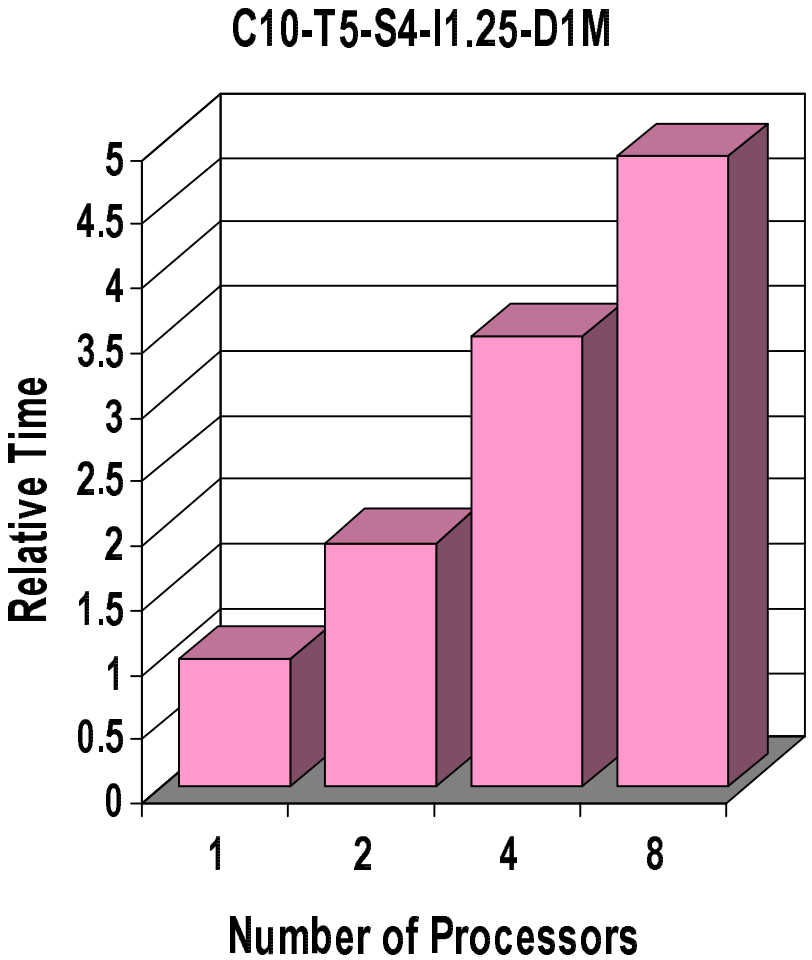
Static vs. Dynamic DLB



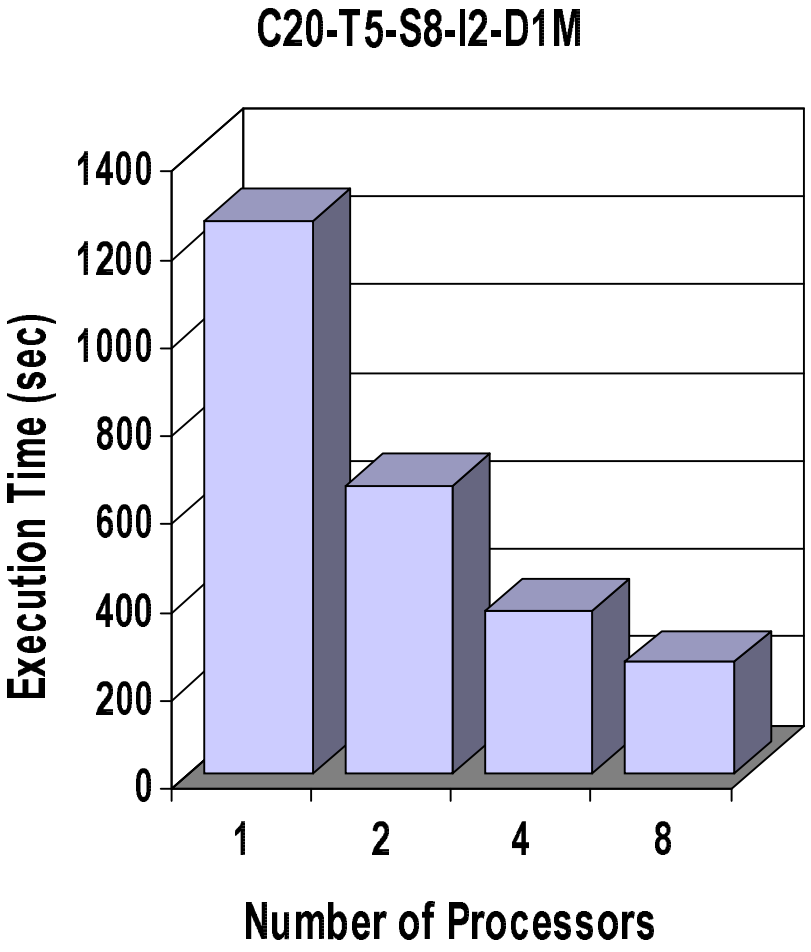
- Dynamic is up to 38% better than Static
- Recursive is up to 12% better than Dynamic
- Over all Recursive is 44% better than Static

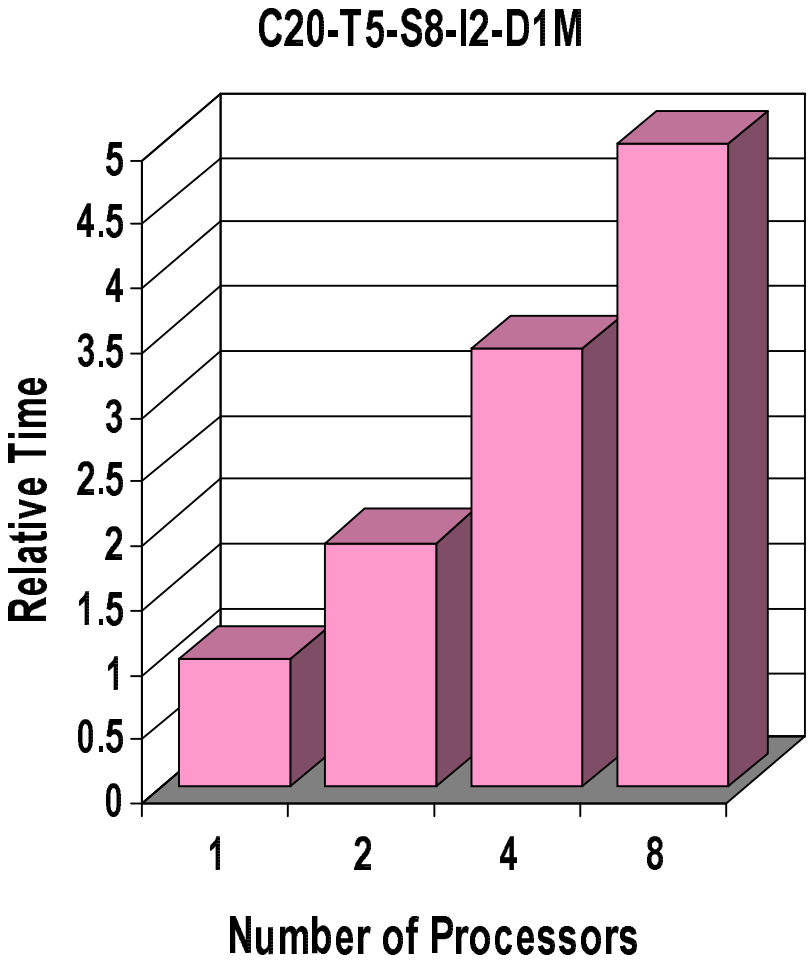
Parallel Performance



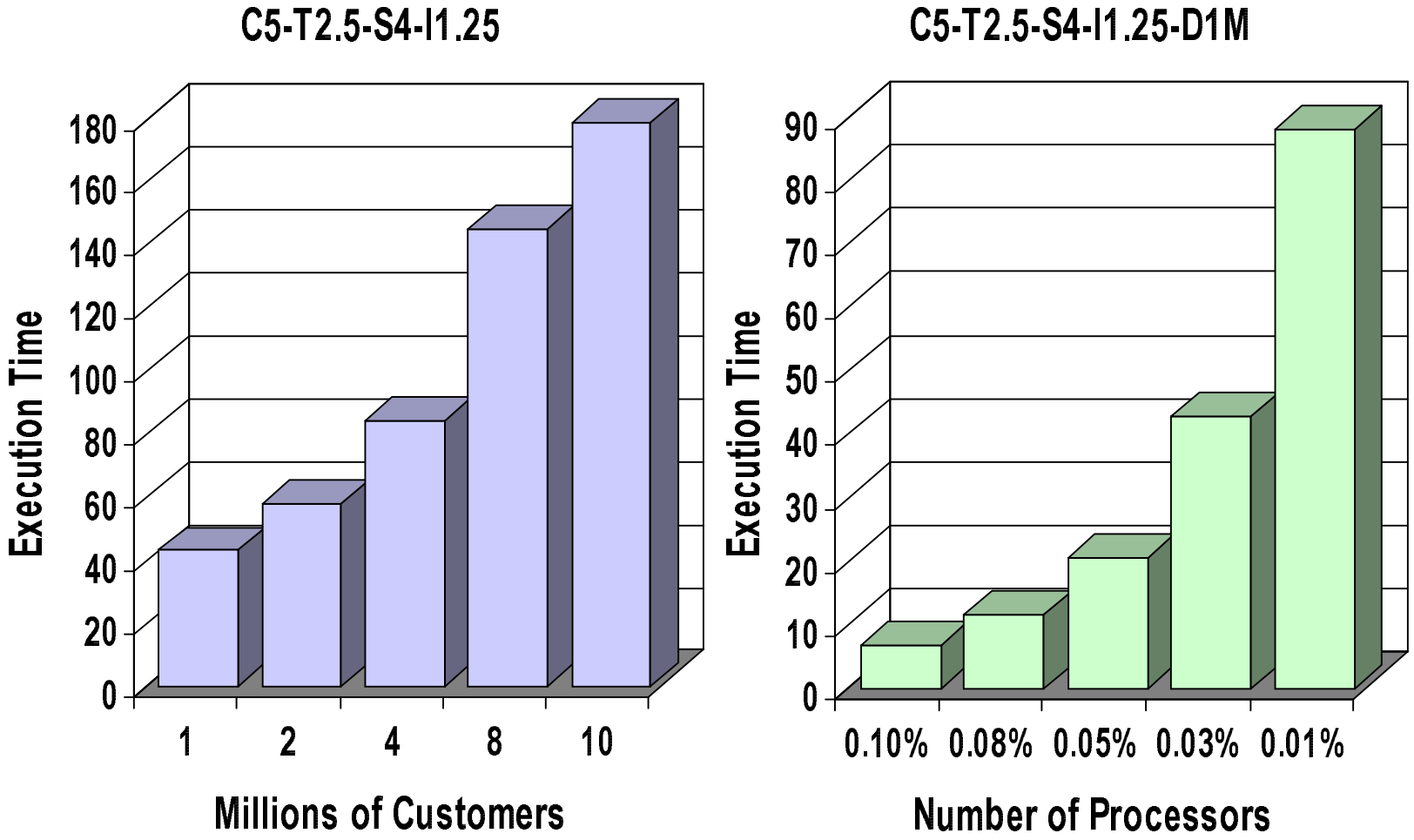


Parallel Performance





Scaleup and Support



pSPADE Summary

- Task parallel better than data parallel
- Dynamic load balancing
- Asynchronous algorithms (independent classes)
- Good locality (uses only intersections)
- Good speedup and scaleup
- What next? Gigabyte databases

Tutorial overview

- Overview of KDD and data mining
- Parallel design space
- Classification
- Associations
- Sequences
- Clustering
- Future directions and summary

What is clustering?

Given N k-dimensional feature vectors, find a “meaningful” partition of the N examples into c subsets or groups

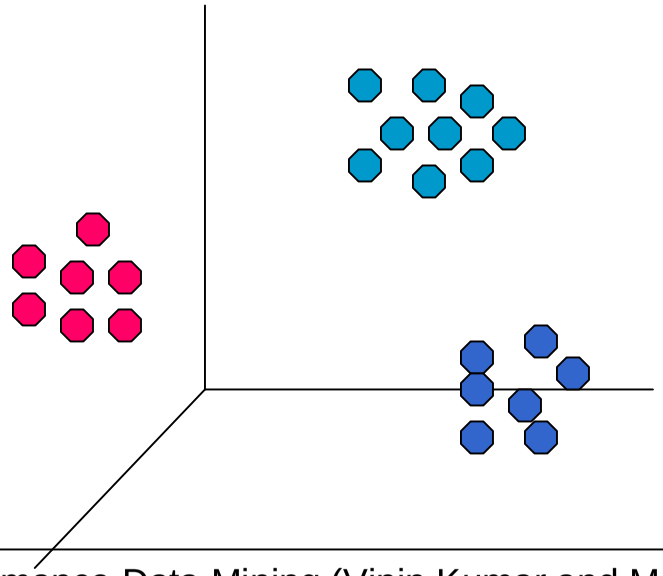
- Discover the “labels” automatically
- c may be given, or “discovered”
- much more difficult than classification, since in the latter the groups are given, and we seek a compact description

Clustering Illustration

- Euclidean Distance Based Clustering in 3-D space.

Intracluster distances are minimized

Intercluster distances are maximized



Clustering

- Have to define some notion of "similarity" between examples
- Goal: maximize intra-cluster similarity and minimize inter-cluster similarity
- Feature vector be
  - All numeric (well defined distances)
  - All categorical or mixed (harder to define similarity; geometric notions don't work)

Clustering schemes

- Distance-based
  - Numeric
    - Euclidean distance (root of sum of squared differences along each dimension)
    - Angle between two vectors
  - Categorical
    - Number of common features (categorical)
- Partition-based
  - Enumerate partitions and score each

Clustering schemes

- Model-based
  - Estimate a density (e.g., a mixture of gaussians)
  - Go bump-hunting
  - Compute P(Feature Vector i | Cluster j)
  - Finds overlapping clusters too
  - Example: bayesian clustering

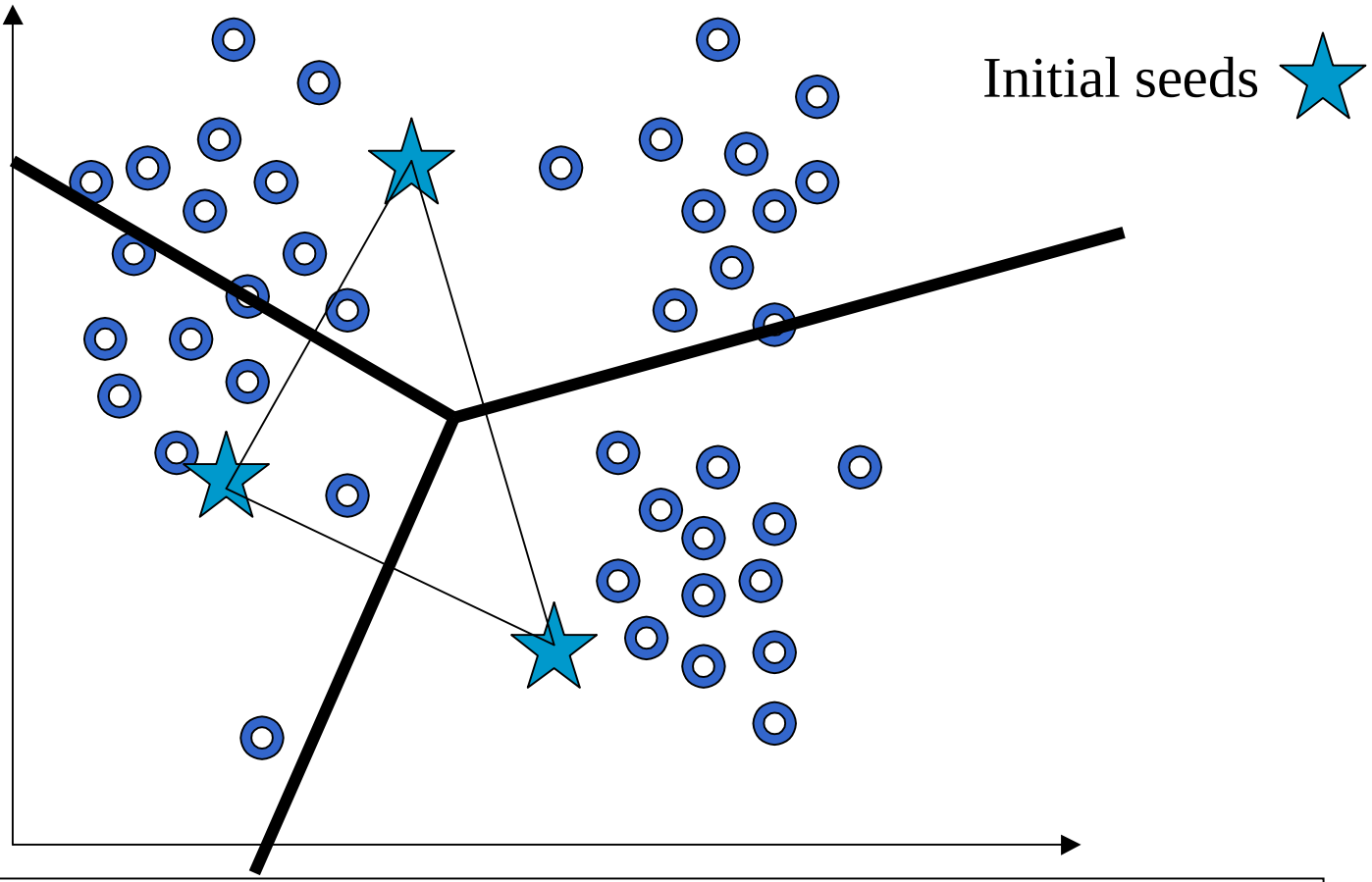
Before clustering

- Normalization:
  - Given three attributes
    - A -- micro-seconds
    - B -- milli-seconds
    - C -- seconds
  - Can't treat differences as the same in all dimensions or attributes
  - Need to scale or normalize for comparison
  - Can assign weight for more importance

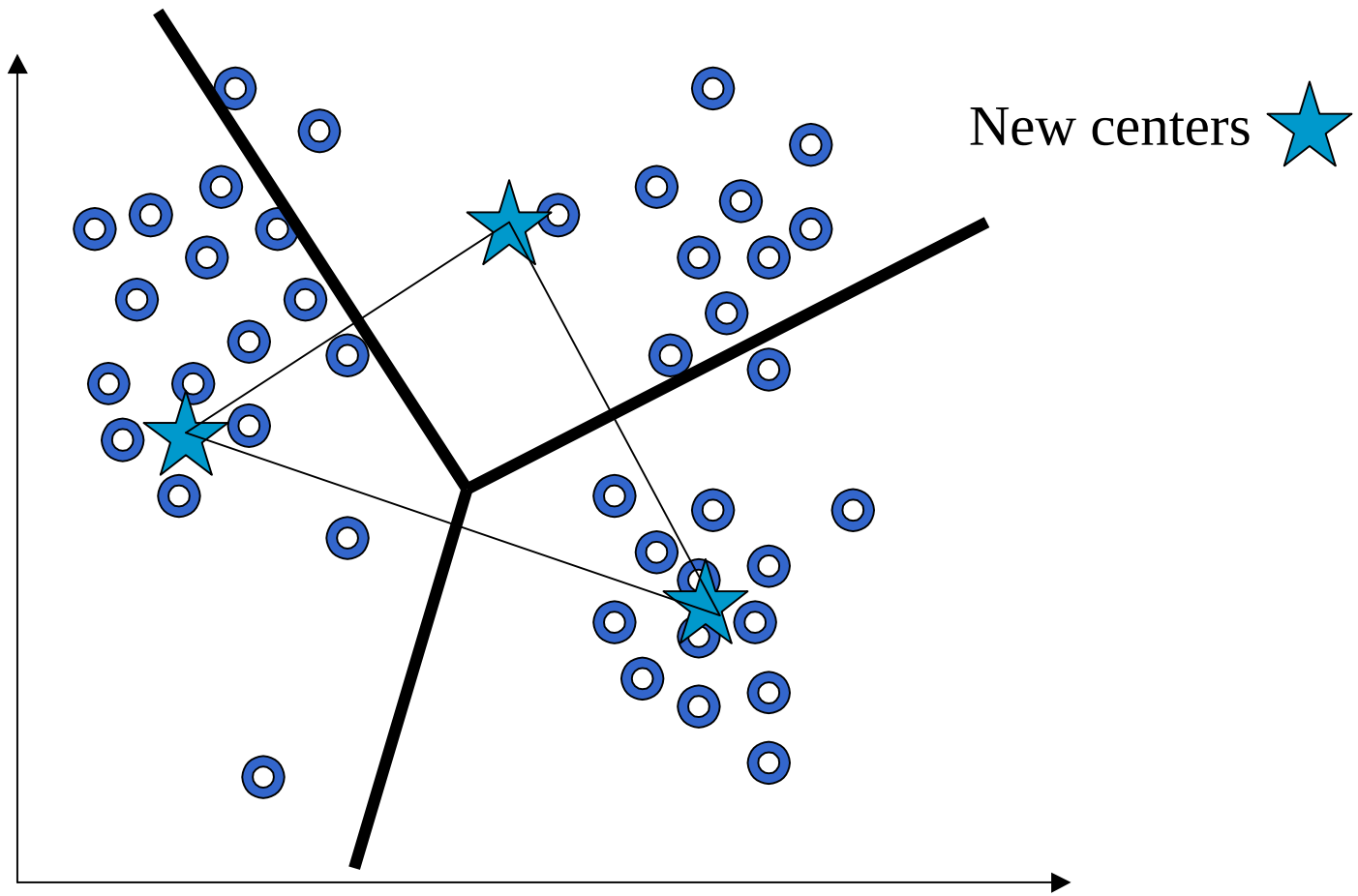
The k-means algorithm

- Specify ‘k’, the number of clusters
- Guess ‘k’ seed cluster centers
- 1) Look at each example and assign it to the center that is closest
- 2) Recalculate the center
- Iterate on steps 1 and 2 till centers converge or for a fixed number of times

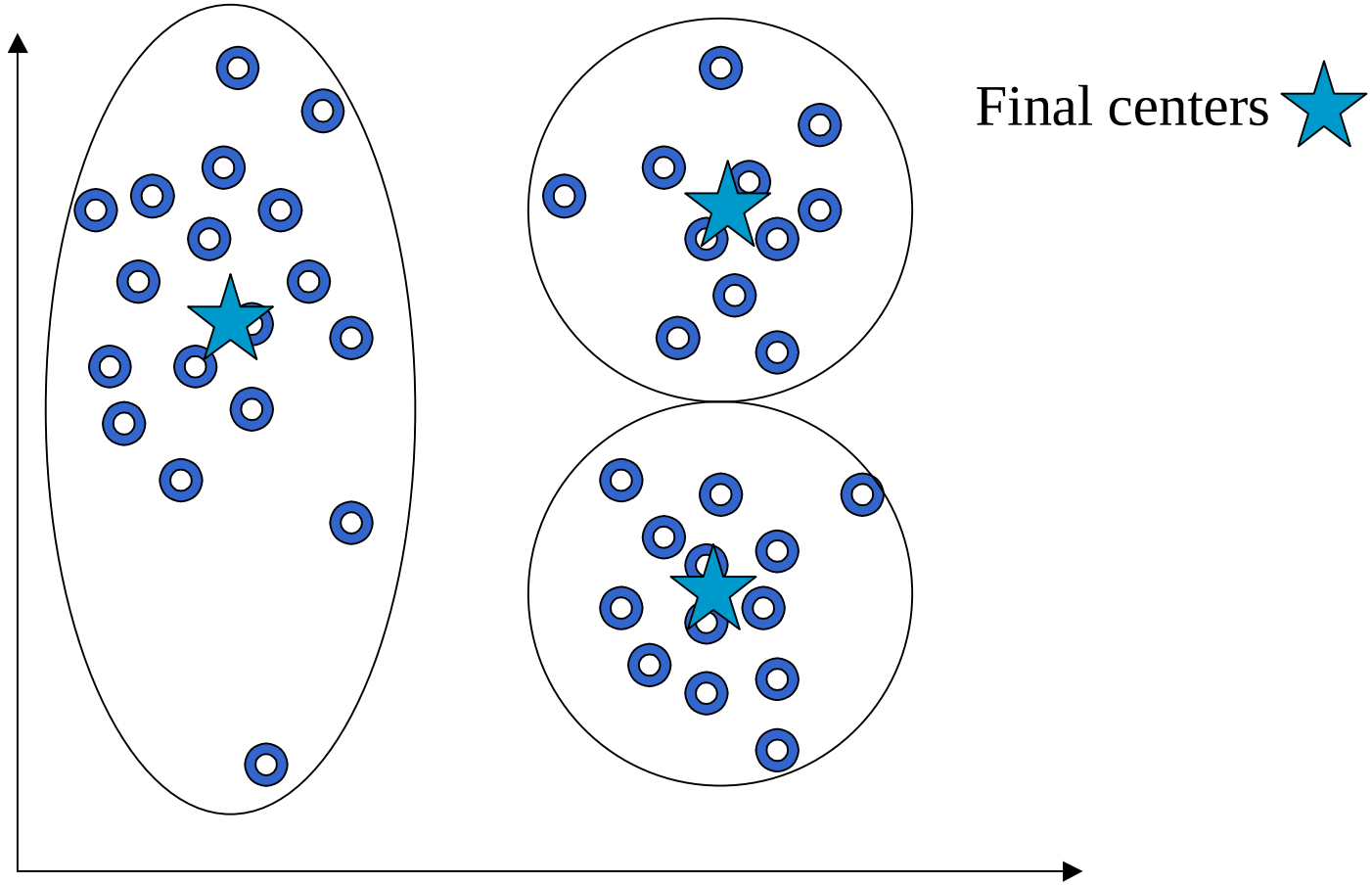
K-means algorithm



K-means algorithm



K-means algorithm



Operations in k-means

- Main Operation: Calculate distance to all k means or centroids
- Other operations:
  - Find the closest centroid for each point
  - Calculate mean squared error (MSE) for all points
  - Recalculate centroids

Parallel k-means

- Divide N points among P processors
- Replicate the k centroids
- Each processor computes distance of each local point to the centroids
- Assign points to closest centroid and compute local MSE
- Perform reduction for global centroids and global MSE value

Gaussian mixture models

- Drawbacks of K-means
  - Doesn’t do well with overlapping clusters
  - Clusters easliy pulled off center by outliers
  - Each records is either in or out of a cluster; no notion of some records begin more or less likely than others to really belong to cluster they have been assigned
- GMM: probabilistic variants of K-means

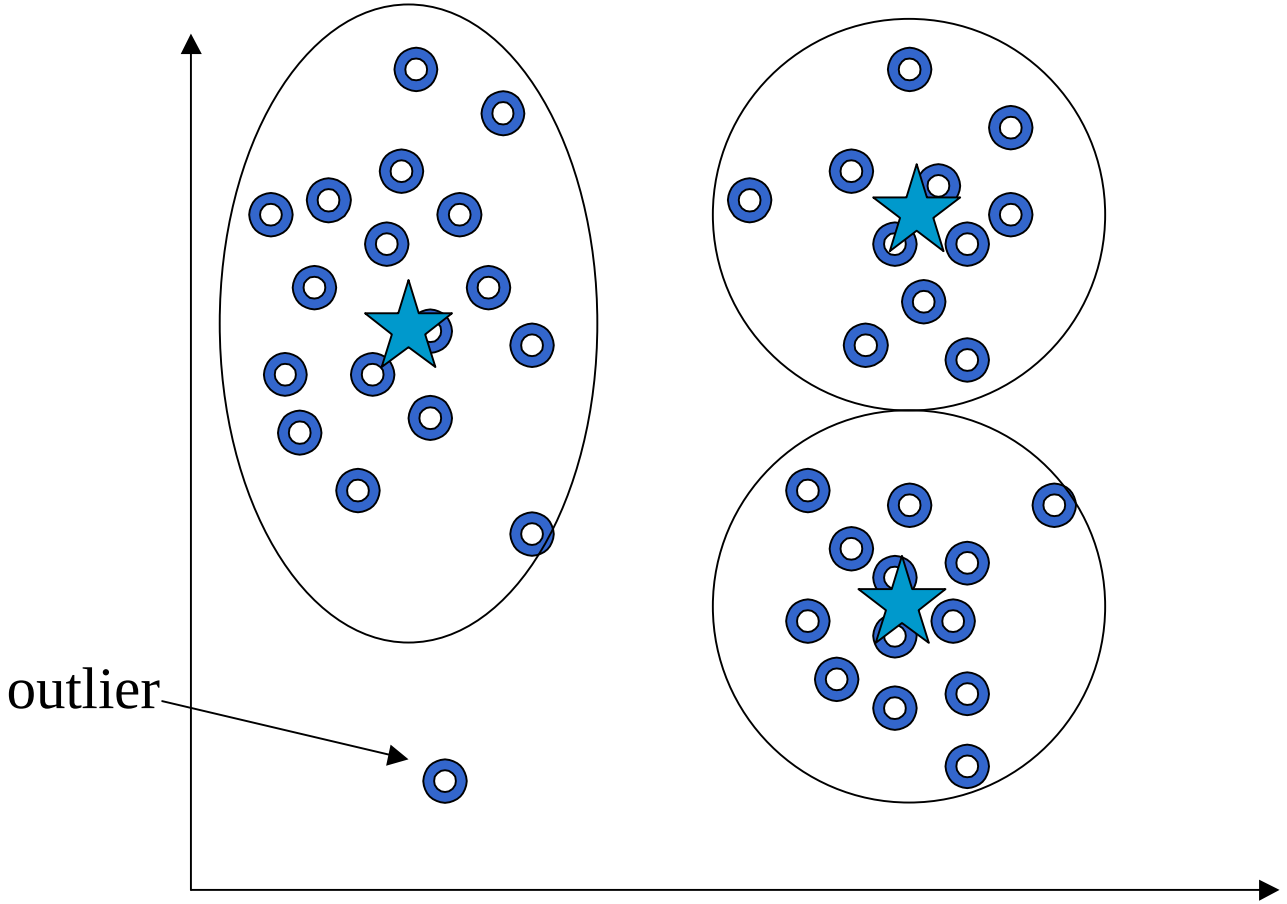
Estimation-maximization

- Choose K seeds: means of a gaussian distribution
- Estimation: calculate probability of belonging to a cluster based on distance
- Maximization: move mean of gaussian to centroid of data set, weighted by the contribution of each point
- Repeat till means don’t move

Deviation/outlier detection

- Find points that are very different from the other points in the dataset
- Could be “noise”, that causes problems for classification or clustering
- Could be the really “interesting” points, for example, in fraud detection, we are mainly interested in finding the deviations from the norm

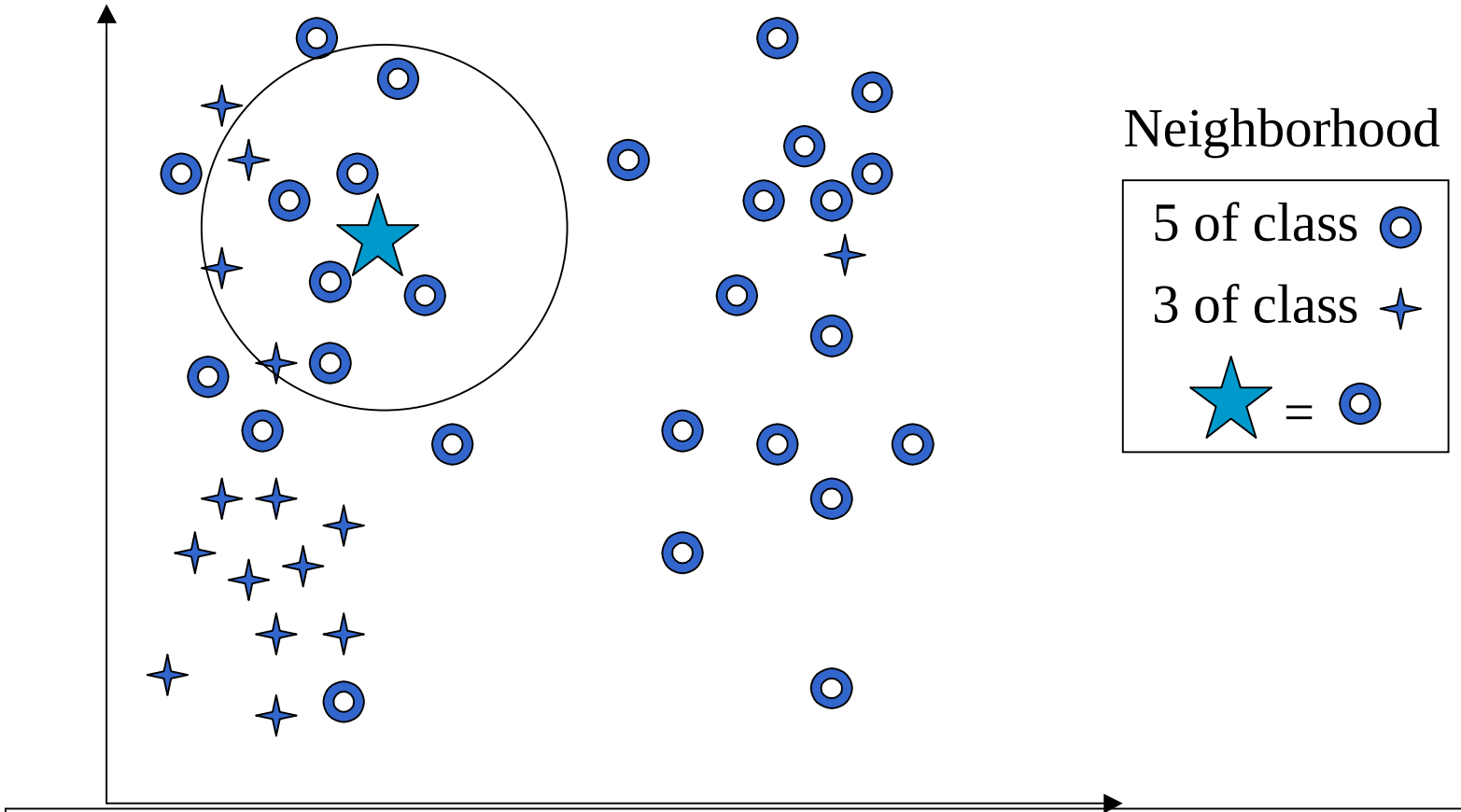
Deviation detection



K-nearest neighbors

- Classification technique to assign a class to a new example
- Find k-nearest neighbors, i.e., most similar points in the dataset (compare against all points!)
- Assign the new case to the same class to which most of its neighbors belong

K-nearest neighbors



Tutorial overview

- Overview of KDD and data mining
- Parallel design space
- Classification
- Associations
- Sequences
- Clustering
- Future directions and summary

Large-scale Parallel KDD Systems

- Terabyte-sized datasets
- Centralized or distributed datasets
- Incremental changes
- Heterogeneous data sources
- Pre-processing, mining, post-processing
- Modular (rapid development)
- Benchmarking (algorithm comparison)

Research Directions

- Fast algorithms: different mining tasks
  - Classification, clustering, associations, etc.
  - Incorporating concept hierarchies
- Parallelism and scalability
  - Millions of records
  - Thousands of attributes/dimensions
  - Single pass algorithms
  - Sampling
  - Parallel I/O and file systems

Research Directions (contd.)

- Data locality and type
  - Distributed data sources (www)
  - Text and multimedia mining
  - Spatial data mining
- Incremental mining: refine knowledge as data changes
- Interactivity: anytime mining

Research Directions (contd.)

- Tight database integration
  - Push common primitives inside DBMS
  - Use multiple tables
  - Use efficient indexing techniques
  - Caching strategies for sequence of data mining operations
  - Data mining query language and parallel query optimization

Research Directions (contd.)

- Understandability: Too many patterns
  - Incorporate background knowledge
  - Integrate constraints
  - Meta-level mining
  - Visualization
- Usability: build a complete system
  - Pre-processing, mining, post-processing, persistent management of mined results

Summary Of the Tutorial

- Data mining is a rapidly growing field
  - Fueled by enormous data collection rates, and need for intelligent analysis for business and scientific gains.
- Large and high-dimensional nature data requires new analysis techniques and algorithms.
- Scalable, fast parallel algorithms are becoming indispensable.
- Many research and commercial opportunities!!!

Resources

Workshops

- HiPC Special Session on Large-Scale Data Mining, 2000.
  http://www.cs.rpi.edu/~zaki/LSDM/
- ACM SIGKDD Workshop on Distributed Data Mining, 2000.
  http://www.eecs.wsu.edu/~hillol/DKD/dpkd2000.html
- 3rd IEEE IPDPS Workshop on High Performance Data Mining, 2000.
  http://www.cs.rpi.edu/~zaki/HPDM/
- ACM SIGKDD Workshop on Large-Scale Parallel KDD Systems, 1999.
  http://www.cs.rpi.edu/~zaki/WKDD99/
- ACM SIGKDD Workshop on Distributed Data Mining, 1998.
  http://www.eecs.wsu.edu/~hillol/DDMWS/papers.html
- 1st IEEE IPPS Workshop on High Performance Data Mining 1998.
  http://www.cise.ufl.edu/~ranka/

Books

- A. Freitas and S. Lavington. Mining very large databases with parallel processing. Kluwer Academic Pub., Boston, MA, 1998.
- M. J. Zaki and C.-T. Ho (eds). Large-Scale Parallel Data Mining. LNAI State-of-the-Art Survey, Volume 1759, Springer-Verlag, 2000.
- H. Kargupta and P. Chan (eds). Advance in Distributed and Parallel Knowledge Discovery, AAAI Press, Summer 2000.

Resources (contd.)

Journal Special Issues

- P. Stolorz and R. Musick (eds.). Scalable High-Performance Computing for KDD, Data Mining and Knowledge Discovery: An International Journal, Vol. 1, No. 4, December 1997.
- Y. Guo and R. Grossman (eds.). Scalable Parallel and Distributed Data Mining, Data Mining and Knowledge Discovery: An International Journal, Vol. 3, No. 3, September 1999.
- V. Kumar, S. Ranka and V. Singh. High Performance Data Mining, Journal of Parallel and Distributed Computing, forthcoming, 2000.

Survey Articles

- F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. Data Mining and Knowledge Discovery: An International Journal, 3(2):131--169, 1999.
- A. Srivastava, E.-H. Han, V. Kumar and V. Singh. Parallel formulations of decision-tree classification algorithms. Data Mining and Knowledge Discovery: An International Journal, 3(3):237--262, 1999.
- M. J. Zaki. Parallel and distributed association mining: A survey. In IEEE Concurrency special issue on Parallel Data Mining, 7(4):14-25, Oct-Dec 1999.
- D. Skillicorn. Strategies for parallel data mining. IEEE Concurrency, 7(4):26--35, Oct-Dec 1999.
- M. V. Joshi, E.-H. Han, G. Karypis and V. Kumar. Efficient parallel algorithm for mining associations. In Zaki and Ho (eds.), Large-Scale Parallel Data Mining, LNAI 1759, Springer-Verlag 2000.
- M. J. Zaki. Parallel and distributed data mining: An introduction. In Zaki and Ho (eds.), Large-Scale Parallel Data Mining, LNAI 1759, Springer-Verlag 2000.

References: Classification

- J. Chattratichat, J. Darlington, M. Ghanem, Y. Guo, H. Huning, M. Kohler, J. Sutiwaraphun, H. W. To, and Y. Dan. Large scale data mining: Challenges and responses. In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, August 1997.
- S. Goil and A. Choudhary. Efficient parallel classification using dimensional aggregates. In Zaki and Ho (eds), Large-Scale Parallel Data Mining, LNAI Vol. 1759, Springer-Verlag 2000.
- M. Holsheimer, M. L. Kersten, and A. Siebes. Data surveyor: Searching the nuggets in parallel. In Fayyad et al.(eds.), Advances in KDD, AAAI Press, 1996.
- M. Joshi, G. Karypis, and V. Kumar. ScalParC: A scalable and parallel classification algorithm for mining large datasets. In Intl. Parallel Processing Symposium, 1998.
- R. Kufrin. Decision trees on parallel processors. In J. Geller, H. Kitano, and C. Suttner, editors, Parallel Processing for Artificial Intelligence 3. Elsevier-Science, 1997.
- S. Lavington, N. Dewhurst, E. Wilkins, and A. Freitas. Interfacing knowledge discovery algorithms to large databases management systems. Information and Software Technology, 41:605--617, 1999.
- F. Provost and J. Aronis. Scaling up inductive learning with massive parallelism. Machine Learning, 23(1), April 1996.
- F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. Data Mining and Knowledge Discovery: An International Journal, 3(2):131--169, 1999.
- John Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In Proc. of the 22nd Int'l Conference on Very Large Databases, Bombay, India, September 1996.
- M. Sreenivas, K. Alsabti, and S. Ranka. Parallel out-of-core divide and conquer techniques with application to classification trees. In 13th International Parallel Processing Symposium, April 1999.
- A. Srivastava, E-H. Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. Data Mining and Knowledge Discovery: An International Journal, 3(3):237--261, 1999.
- M. J. Zaki, C.-T. Ho, and R. Agrawal.Parallel classification for data mining on shared-memory multiprocessors. In 15th IEEE Intl. Conf. on Data Engineering, March 1999.

References: Associations

- R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In U. Fayyad and et al, editors, Advances in Knowledge Discovery and Data Mining, pages 307--328. AAAI Press, Menlo Park, CA, 1996.
- R. Agrawal and J. Shafer. Parallel mining of association rules. IEEE Trans. on Knowledge and Data Engg., 8(6):962--969, December 1996.
- D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In 4th Intl. Conf. Parallel and Distributed Info. Systems, December 1996.
- D. Cheung, V. Ng, A. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. IEEE Transactions on Knowledge and Data Engg., 8(6):911-922, December 1996.
- D. Cheung, K. Hu, and S. Xia. Asynchronous parallel algorithm for mining association rules on shared-memory multi-processors. In 10th ACM Symp. Parallel Algorithms and Architectures, June 1998.
- D. Cheung and Y. Xiao. Effect of data distribution in parallel mining of associations. Data Mining and Knowledge Discovery: An International Journal. 3(3):291--314, 1999.
- E-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In ACM SIGMOD Conf. Management of Data, May 1997.
- M. Joshi, E.-H. Han, G. Karypis, and V. Kumar. Efficient parallel algorithms for mining associations. In M. Zaki and C.-T. Ho (eds), Large-Scale Parallel Data Mining, LNAI State-of-the-Art Survey, Volume 1759, Springer-Verlag, 2000.
- S. Morishita and A. Nakaya. Parallel branch-and-bound graph search for correlated association rules. In Zaki and Ho (eds), Large-Scale Parallel Data Mining, LNAI Vol. 1759, Springer-Verlag 2000.

References: Associations (contd.)

- A. Mueller. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, College Park, August 1995.
- J. S. Park, M. Chen, and P. S. Yu. Efficient parallel data mining for association rules. In ACM Intl. Conf. Information and Knowledge Management, November 1995.
- T. Shintani and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In 4th Intl. Conf. Parallel and Distributed Info. Systems, December 1996.
- T. Shintani and M. Kitsuregawa. Parallel algorithms for mining generalized association rules with classification hierarchy. In ACM SIGMOD International Conference on Management of Data, May 1998.
- M. Tamura and M. Kitsuregawa. Dynamic load balancing for parallel association rule mining on heterogeneous PC cluster systems. In 25th Int'l Conf. on Very Large Data Bases, September 1999.
- M. J. Zaki. Parallel and distributed association mining: A survey. IEEE Concurrency, 7(4):14--25, October-December 1999.
- M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. In Supercomputing'96, November 1996.
- M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, August 1997.
- M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithms for fast discovery of association rules. Data Mining and Knowledge Discovery: An International Journal, 1(4):343-373, December 1997.
- M. J. Zaki, S. Parthasarathy, W. Li, A Localized Algorithm for Parallel Association Mining, 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), June, 1997.

References: Sequences

- R. Agrawal and R. Srikant. Mining sequential patterns. In 11th Intl. Conf. on Data Engg., 1995.
- H. Mannila and H. Toivonen and I. Verkamo. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery: An International Journal. 1(3):259-289, 1997.
- T. Oates, M. D. Schmill, and P. R. Cohen. Parallel and distributed search for structure in multivariate time series. In 9th European Conference on Machine Learning, 1997.
- T. Oates, M. D. Schmill, D. Jensen, and P. R. Cohen. A family of algorithms for finding temporal structure in data. In 6th Intl. Workshop on AI and Statistics, March 1997.
- T. Shintani and M. Kitsuregawa. Mining algorithms for sequential patterns in parallel: Hash based approach. In 2nd Pacific-Asia Conf. on Knowledge Discovery and Data Mining, April 1998.
- R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In 5th Intl. Conf. Extending Database Technology, March 1996.
- M. J. Zaki. Efficient enumeration of frequent sequences. In 7th Intl. Conf. on Information and Knowledge Management, November 1998.
- Mohammed J. Zaki, Parallel Sequence Mining on SMP Machines, ACM SIGKDD Workshop on Large-Scale Parallel KDD Systems, 1999 (LNAI Vol 1759).

References: Clustering

- K. Alsabti, S. Ranka, V. Singh. An Efficient K-Means Clustering Algorithm. 1st IPPS Workshop on High Performance Data Mining, March 1998.
- I. Dhillon and D. Modha. A data clustering algorithm on distributed memory machines. In Zaki and Ho (eds), Large-Scale Parallel Data Mining, LNAI Vol. 1759, Springer-Verlag 2000.
- L. Iyer and J. Aronson. A parallel branch-and-bound algorithm for cluster analysis. Annals of Operations Research Vol. 90, pp 65-86, 1999.
- E. Johnson and H. Kargupta. Collective hierarchical clustering from distributed heterogeneous data. In Zaki and Ho (eds), Large-Scale Parallel Data Mining, LNAI Vol. 1759, Springer-Verlag 2000.
- D. Judd, P. McKinley, and A. Jain. Large-scale parallel data clustering. In Int'l Conf. Pattern Recognition, August 1996.
- X. Li and Z. Fang. Parallel clustering algorithms. Parallel Computing, 11:270--290, 1989.
- C.F. Olson. Parallel algorithms for hierarchical clustering. Parallel Computing, 21:1313--1325, 1995.
- S. Ranka and S. Sahni. Clustering on a hypercube multicomputer. IEEE Trans. on Parallel and Distributed Systems, 2(2):129--137, 1991.
- F. Rivera, M. Ismail, and E. Zapata. Parallel squared error clustering on hypercube arrays. Journal of Parallel and Distributed Computing, 8:292--299, 1990.
- G. Rudolph. Parallel clustering on a unidirectional ring. In R. Grebe et al., editor, Transputer Applications and Systems'93: Volume 1, pages 487--493. IOS Press, Amsterdam, 1993.
- H. Nagesh, S. Goil and A. Choudhary. MAFIA: Efficient and scalable subspace clustering for very large data sets. Technical Report 9906-010, Center for Parallel and Distributed Computing, Northwestern University, June 1999.
- X. Xu, J. Jager and H.-P. Kriegel. A fast parallel clustering algorithm for large spatial databases. Data Mining and Knowledge Discovery: An International Journal. 3(3):263--290, 1999.

References: Distributed DM

- J. Aronis, V. Kolluri, F. Provost, and B. Buchanan. The WoRLD: Knowledge discovery from multiple distributed databases. In Florida Artificial Intelligence Research Symposium, May 1997.
- R. Bhatnagar and S. Srinivasan. Pattern discovery in distributed databases. In AAAI National Conference on Artificial Intelligence, July 1997.
- J. Chattratichat, J. Darlington, Y. Guo, S. Hedvall, M. Kohler, and J. Syed. An architecture for distributed enterprise data mining. In 7th Intl. Conf. High-Performance Computing and Networking, April 1999.
- R. L. Grossman, S. M. Bailey, H. Sivakumar, and A. L. Turinsky. Papyrus: A system for data mining over local and wide area clusters and super-clusters. In Supercomputing'99, November 1999.
- Y. Guo and J. Sutiwaraphun. Knowledge probing in distributed data mining. In 3rd Pacific-Asia Conference on Knowledge Discovery and Data Mining, August 1999.
- L. Hall, N. Chawla, K. Bowyer and W. P. Kegelmeyer. Learning rules from distributed data. In Zaki and Ho (eds), Large-Scale Parallel Data Mining, LNAI Vol. 1759, Springer-Verlag 2000.
- H. Kargupta, I. Hamzaoglu, and B. Stafford. Scalable, distributed data mining using an agent based architecture. In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, August 1997.
- H. Kargupta, B-H. Park, D. Hershberger, and E. Johnson. Collective data mining: A new perspective toward distributed data mining. In Kargupta and Chan (eds), Advances in Distributed DM, AAAI Press, 2000.
- S. Parthasarathy and R. Subramonian. Facilitating data mining on a network of workstations. In Kargupta and Chan (eds), Advances in Distributed DM, AAAI Press, 2000.
- A. Prodromidis, S. Stolfo, and P. Chan. Meta-learning in distributed data mining systems: Issues and approaches. In Kargupta and Chan (eds), Advances in Distributed DM, AAAI Press, 2000.
- S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, W. Fan, and P. Chan. Jam: Java agents for meta-learning over distributed databases. In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, August 1997.