SQL Server Performance Analysis

Joe Chang

jchang6@yahoo.com

www.sql-server-performance.com/joe_chang.asp

Objectives

- Estimate DB performance early in development
  - What design/architecture decisions impact performance?
  - Is the project/architecture feasible?
- Production database performance tuning
  - Reduces guess work, but there are easier ways
- Server Performance Characteristics
  - Processor Architecture: PIII – Xeon – Itanium 2 – Opteron
  - System Architecture: 2, 4, 8, 16-way etc
- Performance from one DB to another ?
  - ex. SQL Server 2000 to 2005 (Yukon)

Topics

- Overview
  - Predict performance in design phase – can it be done?
  - Interpreting performance benchmarks
- Execution Plan Cost Formulas
  - Cost formulas for common SQL operations
  - Statistics on data distribution to estimate rows & pages
- Quantitative Performance Analysis
  - Actual query cost structure same as optimizer formulas?
  - Platform specific processor & system architecture
    - Memory & Disk

HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

SQL Server Performance Analysis Overview

Joe Chang

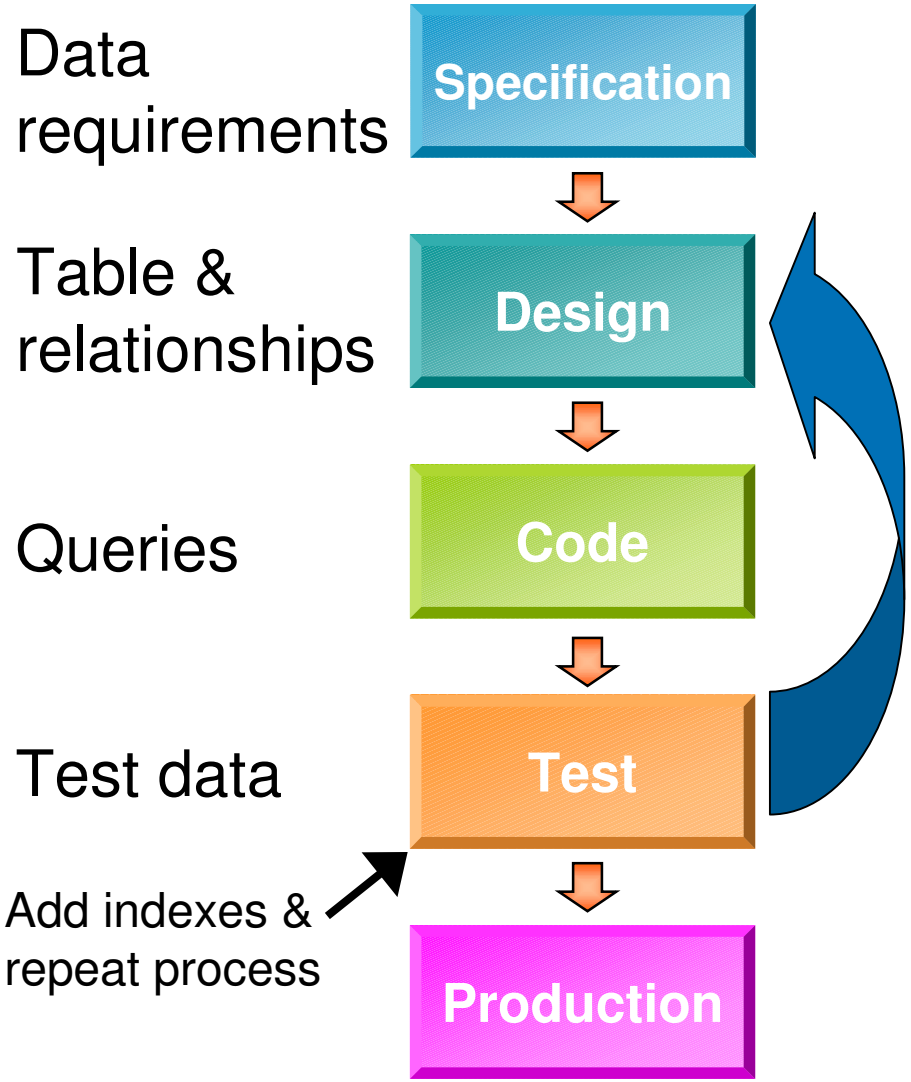
jchang6@yahoo.com

www.sql-server-performance.com/joe_chang.asp

Development Project Challenges

- Database application fails to meet performance objectives
  - Discovered in test?
  - Discovered by live users?
- Resolution?
  - Massive redesign effort?
  - Redefine performance objectives?

Development Life Cycle



No methods for predicting performance in design & code phase.

Performance testing cannot be started until a functional database is available.
At this stage it may be too late for redesign.

How to test for performance?
Will production perform similarly?

Designing to Requirements

- How to design to a specific performance target?
  - Normalization?
    - To avoid update anomalies
      - Not directly related to performance
  - Objects?
    - For code reuse,
      - Not related to performance
- Other Performance Methodologies
  - No way to estimate performance early in design
  - Need to test completed application with actual data
    - Test environment performance may differ from production?

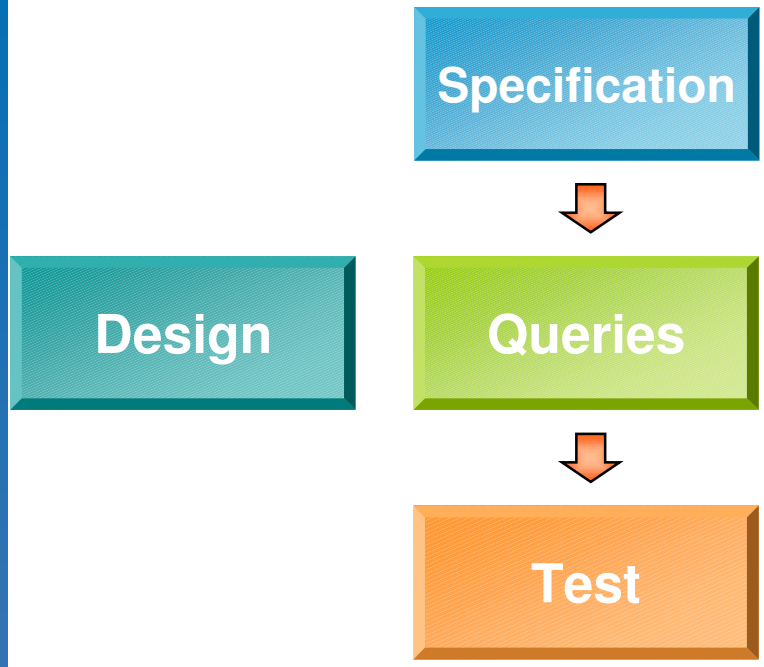
Database Performance Factors

- Code
  - Tables, Indexes, Queries
    - Does not completely determine performance
- Data
  - Rows involved, statistical distribution
    - Still don't know performance
- Execution Plan
  - Depends on Code and data distribution statistics
    - Now we can estimate performance!

Model Assumptions

- No excessive contention
  - Locking contention is minimal
- Component operation costs are:
  - Predictable & consistent
    - Good disk & memory configuration
- Costs independent of other operations

Code - Design for Performance



Include data cardinality in specification

Tables, queries and indexes in parallel

Test with same cardinality as expected in production. Scale memory, data size & disks

Inefficient Design Indicators:

1. **Excessive use of SELECT DISTINCT**
2. **Excessive use of TEMP tables**
3. **Excessive logic in query**
4. **Expensive Table scans or bookmark lookups**

Data & Distribution Statistics

- Overall database size
  - Should be a non-issue for transactional applications
    - Small queries should never require a large table scan
  - Index B-tree organization
    - Ex. index key is 80Bytes wide, 100 rows per 8K page
    - 100X increase in rows for each index level
- Estimated rows & pages involved versus Actual
  - Look for wide variations, data skew, correlation
- Testing – Data population
  - Cardinality is more important than size
    - 1 Customer – 10 Orders – 10 Line Items per order
  - Scale: Data Size / Memory / Disks

Execution Plan

- Query Plan
  - One or more component operations
- Component operations
  - Index Seek, Table Scan, Bookmark Lookup, Loop, Hash & Merge Joins, etc
- Cost Structure
  - Rows & Pages involved
  - Scalar Operations – not really zero cost
- Is optimizer cost model same as actual cost?
  - Model assumptions
  - System architecture changes over time

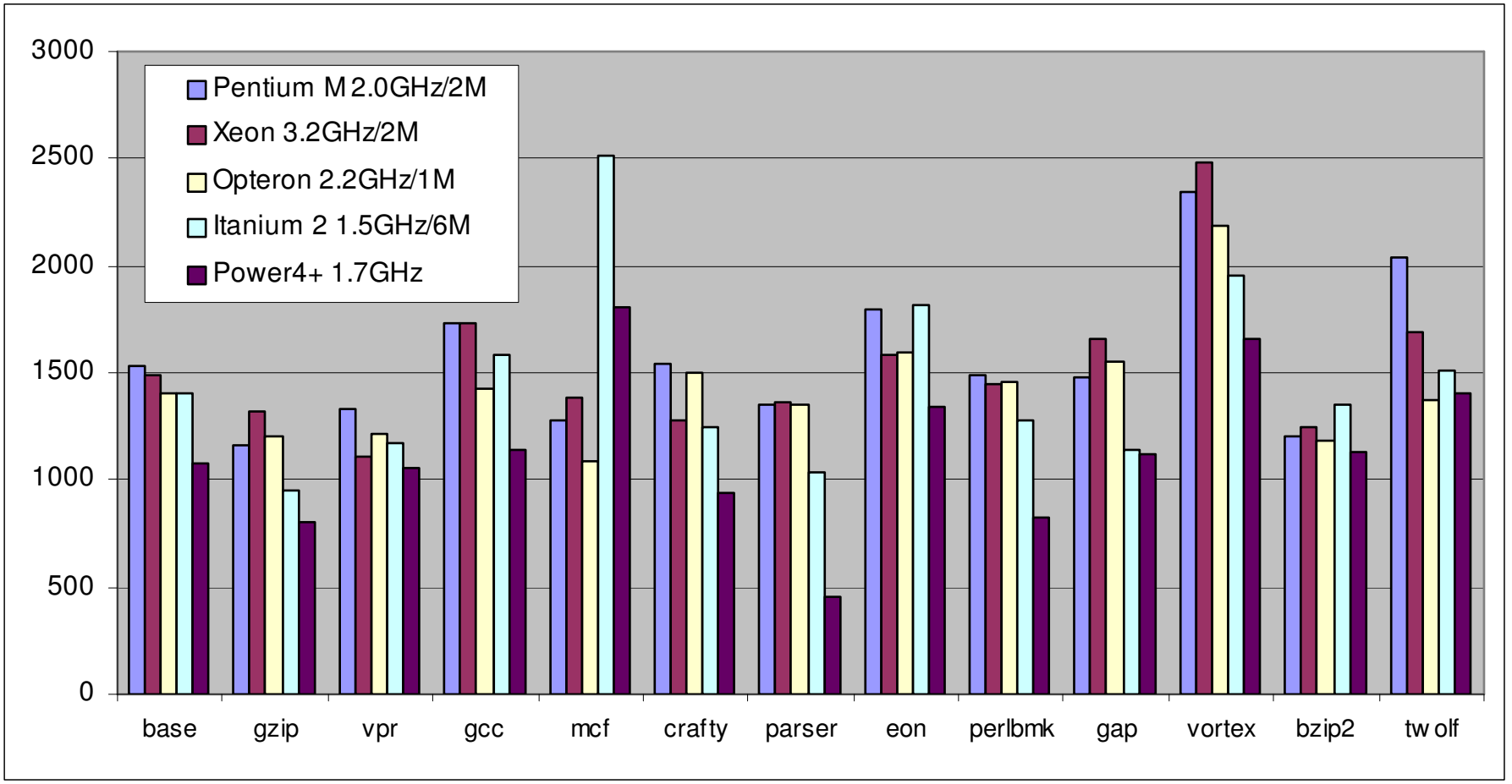
Performance Strategy

- Know how optimizer determines execution plan
- True cost structure of SQL operations
  - Derive strategies for database architecture
- Design Tables, Indexes & SQL code
  - Minimize row & page count
  - Enable most efficient operations
  - Avoid contention

Performance Benchmarks

- SPEC CPU Integer Base (www.spec.org)
  - Pros: Available for most processors, frequency, etc
  - Cons: Single CPU, Intel Compilers*
- TPC-C OLTP workload (www.tpc.org)
  - Pros: Multi-Processor platforms, Database specific, reasonable range of published results
  - Cons: more disk and memory intensive than most actual OLTP apps? (12.5tx/warehouse, 84MB/Wh)
- TPC-H (DSS)
  - Very limited publications
- Others: SAP, Siebel, PeopleSoft

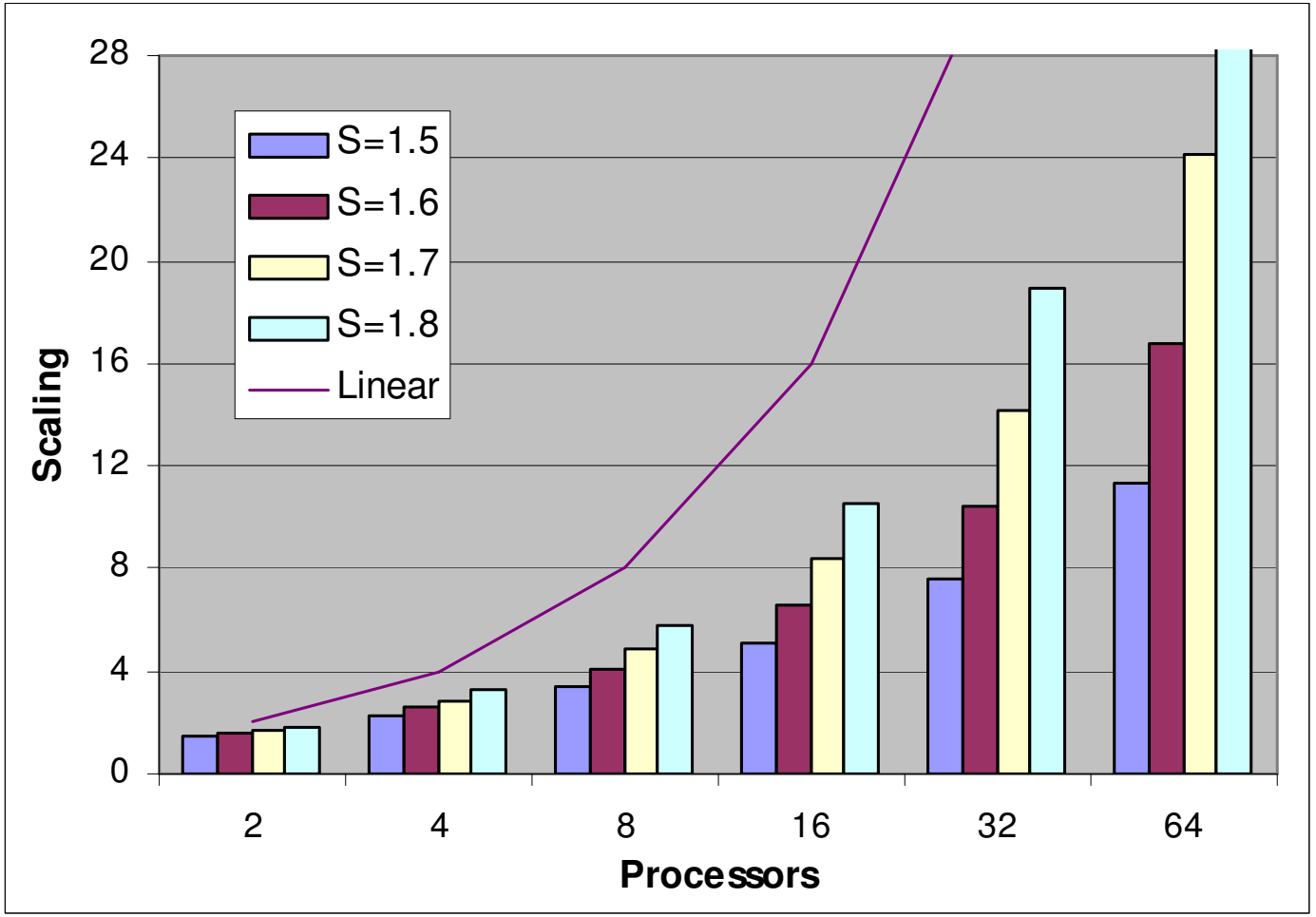
SPEC CPU 2000 Integer



Pentium M 2.0GHz 90nm, all others 130nm

Xeon 2.4GHz/512K base: 913 (used later in this presentation)

Scaling



$$P_n / P_1 = S ** \log_2(n)$$

P_n	Performance with n processors	P_1	Perf. with 1 processor
S	Scale Factor	n	Number of processors

Published Performance – SQL Server

Xeon MP 3.0GHz/4M

# CPUs	System	tpm-C	$/tpm-C	Mem(GB)	# Disks
4	IBM x365	102,667	$3.52	32	266
8	IBM x445	156,486	$4.31	64	616
16	Unisys ES7000	237,869	$5.08	64	700+10
32	Unisys ES7000	304,148	$6.18	64	1092+12

Itanium 2 1.5GHz/6M

# CPUs	System	tpm-C	$/tpm-C	Mem(GB)	# Disks
4	HP rx5670	121,065	$4.49	64	448+20
8	Bull	175,366	$4.54	128	225
16	Unisys ES7000	309,037	$4.49	128	770+24
32	NEC Express	577,531	$7.74	512	1150
64	HP Superdome	786,646	$6.49	512	1792+60

IA-32 limited max memory (64GB),
AWE overhead ,
bus architecture

Published Performance 2

IBM Power 4+/5 1.9GHz

# CPUs	System	tpm-C	$/tpm-C	Mem(GB)	# Disks
4	Pwr5 570 / Oracle	194,391	$5.62	128	432+16
8	Pwr5 570 / Oracle	371,044	$5.26	256	720+30
16	Pwr5 570 / DB2	809,144	$4.95	512	1600+40
32	Pwr4+ 690 / DB2	1,025,486	$5.43	1024	1950+40

Itanium 2 1.5GHz/6M

# CPUs	System	tpm-C	$/tpm-C	Mem(GB)	# Disks
4	HP rx5670	121,065	$4.49	64	448+20
8	Bull	175,366	$4.54	128	225
16	Unisys ES7000	309,037	$4.49	128	770+24
32	NEC Express	577,531	$7.74	512	1150
64	HP Superdome	786,646	$6.49	512	1792+60

No SPEC integer benchmark published for Power5
Compared to Power4+: Integrated memory controller, more/shorter IO paths, simultaneous multi-threading

Benchmark Limitations

- Many benchmarks are well designed
  - It really does take ~ 2X faster system to get 2X faster results
  - Difficult to generate ridiculous results
- Not representative of actual applications and usage
  - No method for translating benchmark results to actual application characteristics

Moore's Law

- Shouldn't we have all the CPU power we want?

Year	**Processor**	**Freq. MHz**	Bus BW (GB/sec)	SPEC CPU 2000 int_base
1994	Pentium	100	0.5	
1996	Pentium Pro	200	0.5	
1998	Pentium III	500	0.8	231
2000/1	Pentium 4(Xeon)	1600	3.2	552
2004	Xeon/Itanium 2/Opteron	3.0/1.5/2.4	3.2/6.4/+	1455/1380/1346

Facts that didn't make the spec sheet

Total Memory Latency:
Processor initiates a request, memory controller, DRAM access time, return trip

10 years ago	~180ns?
Today	~140ns?

Similar for disk drives, Random Disk Access

10 years ago	15ms
Today	7ms

Build to applications to favor sequential access instead of random access

HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

Execution Plan Cost Formulas

Joe Chang

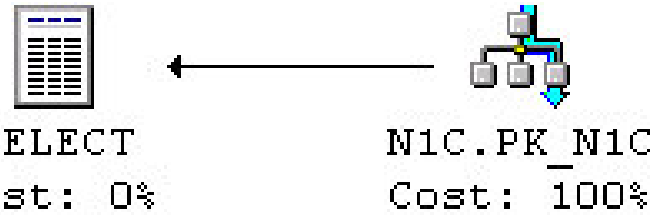
jchang6@yahoo.com

www.sql-server-performance.com/joe_chang.asp

Topics

- Component Operation Cost Model
  - Cost formulas for basic operations
- Dependencies
  - Rows & Pages involved - yes
  - Index depth – no,
  - Locks level – no
  - WHERE conditions – no cost for logic
    - only if row count affected

Index Seek - 1 row




```
SELECT xx
FROM N1C
WHERE ID = @ID
```

Clustered Index Seek

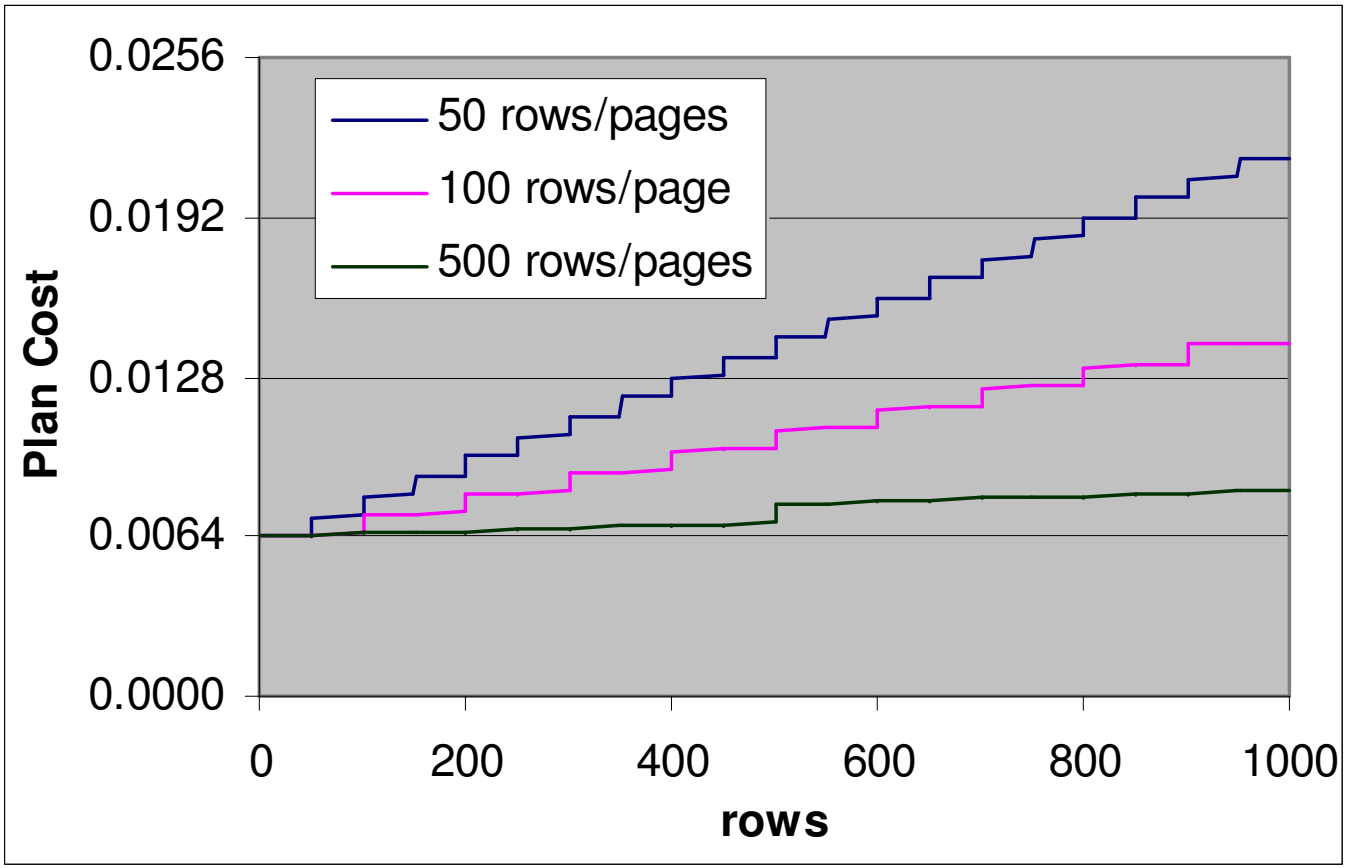
Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek
Logical operation:	Clustered Index Seek
Estimated row count:	1
Estimated row size:	21
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000080
Estimated number of executes:	1.0
Estimated cost:	0.006408(100%)
Estimated subtree cost:	0.00640

Argument:
OBJECT:([pubs].[dbo].[N1C].[PK_N1C]), SEEK:([N1C].[ID]=Convert([@1])) ORDERED FORWARD

	I/O	CPU	Total
≤ 1GB	0.006328500	0.0000796	0.006408100
> 1GB	0.003203425	0.0000796	0.003283025

Index Seek Cost Formula




```
Multiple rows, ≤ 1GB
I/O:    0.00632850 + 0.00074074 per additional page (leaf level)
CPU:    0.00007960 + 0.00000110 per additional row
Total: 0.00640810 + 0.00074074 /add. page + 0.0000011 / add. row
```

Bookmark Lookup


```
SELECT xx
FROM N1N
WHERE ID = @ID
```

Same cost for bookmark lookup on Heap and Clustered Index

Bookmark Lookup

Use a Bookmark (RID or Clustering Key) to look up the corresponding row in the Table or Clustered Index.

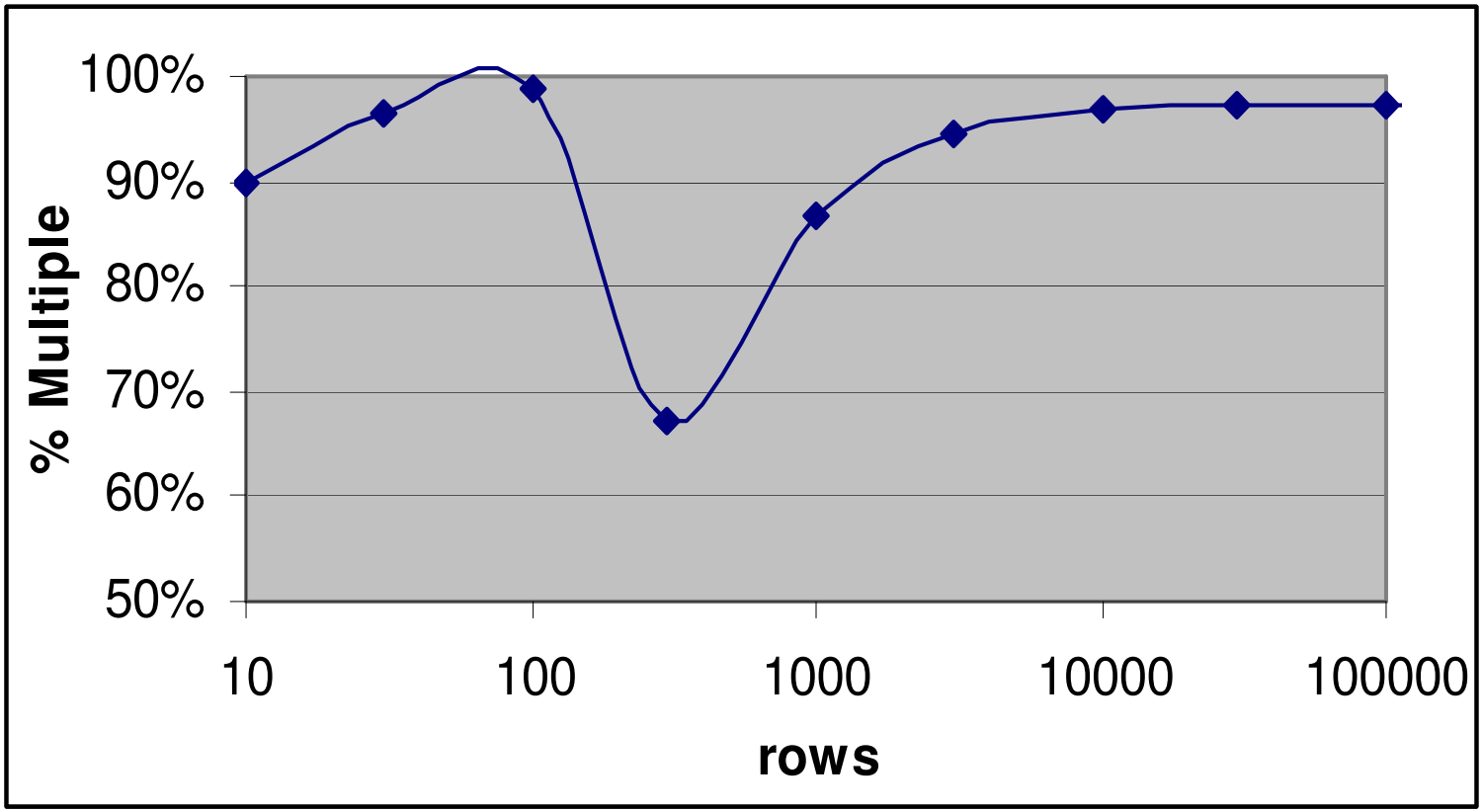
Physical operation:	Bookmark Lookup
Logical operation:	Bookmark Lookup
Estimated row count:	1
Estimated row size:	21
Estimated I/O cost:	0.00625
Estimated CPU cost:	0.000001
Estimated number of executes:	1.0
Estimated cost:	0.006251(49%)
Estimated subtree cost:	0.0126

Argument:
BOOKMARK:([Bmk1000]), OBJECT:([pubs].[dbo].[N1N])

	I/O	CPU	Total
≤ 1GB	0.0062500	0.0000011	0.0062511
> 1GB	0.0031249	0.0000011	0.0031260

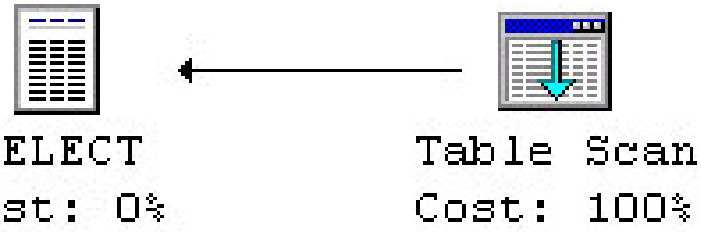
Bookmark Lookup Cost Formula

Example:
500,000 rows,
99 rows / page




```
I/O:    multiple 0f 0.0062500 (≤1GB), 0.0031249 (>1GB)
CPU:    0.0000011 per row
Total: multiple of I/O + (# of rows) x 0.0000011
```

Table Scan




```
SELECT xx
FROM N1H
WHERE ID = @ID
```

Table Scan

Scan rows from a table.

Physical operation:	Table Scan
Logical operation:	Table Scan
Estimated row count:	1
Estimated row size:	21
Estimated I/O cost:	0.0375
Estimated CPU cost:	0.000430
Estimated number of executes:	1.0
Estimated cost:	0.038009(100%)
Estimated subtree cost:	0.0380

Argument:
OBJECT:([pubs].[dbo].[N1H]), WHERE:([N1H].[ID]=Convert([@1]))

```
I/O: 0.03757850 + 0.00074074/page
CPU: 0.00007850 + 0.00000110/row
```

Plan Cost Unit of Measure

- Time? CPU-usage? time, in seconds

```
0.0062500sec -> 160/sec
```

Too fast for 7200RPM disk random I/Os.

```
0.000740741  ->1350/sec (8KB)
->169/sec(64K)-> 10.8MB/sec
```

About right for 1997 sequential disk transfer rate?

S2K BOL: Administering SQL Server, Managing Servers, Setting Configuration Options: cost threshold for parallelism Opt
Query cost refers to the estimated elapsed time, in seconds, required to execute a query on a specific hardware configuration.

Disk Drive Performance

Access time = rotational latency + seek time
7200RPM = 4.17ms Rotational Latency
10000RPM = 3ms, 15000RPM = 2ms

Year	Model	Rot. RPM	Avg. Seek	Sequential Transfer
1994	ST12550	7.2K	8.0	3.5–6.0 MB/sec
1996	ST34371	7.2K	9.4	7.1–11.7
1997	ST34572	7.2K	9.4	7.9–12.5
1998	ST39102	10K	5.4	19.0–28.9*
1999	ST39103	10K	5.4	22.7–36.2*
2000	ST318451	15K	4.1	37.4–48.9*
2002	ST373453	15K	3.8	49–75

Bookmark versus Scan

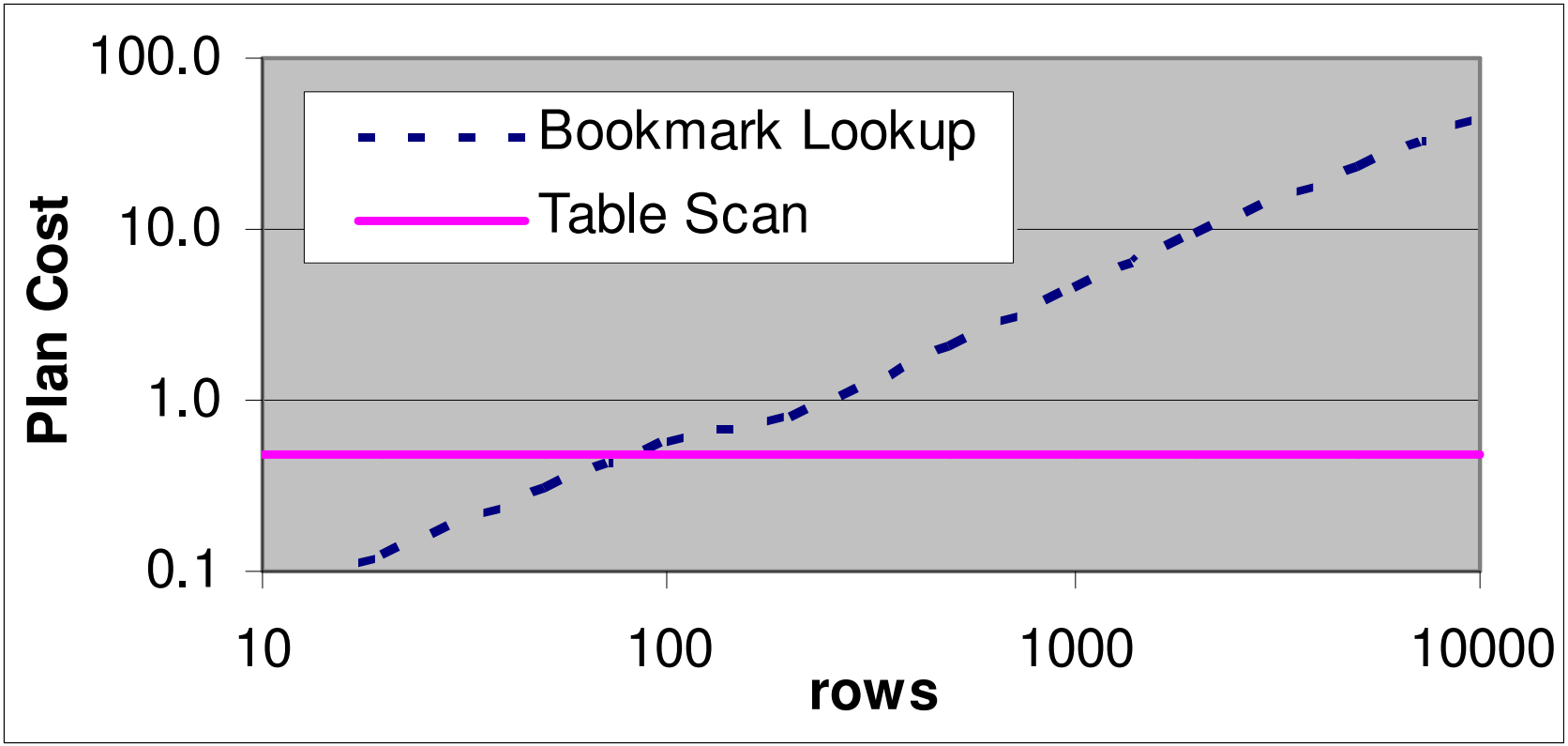
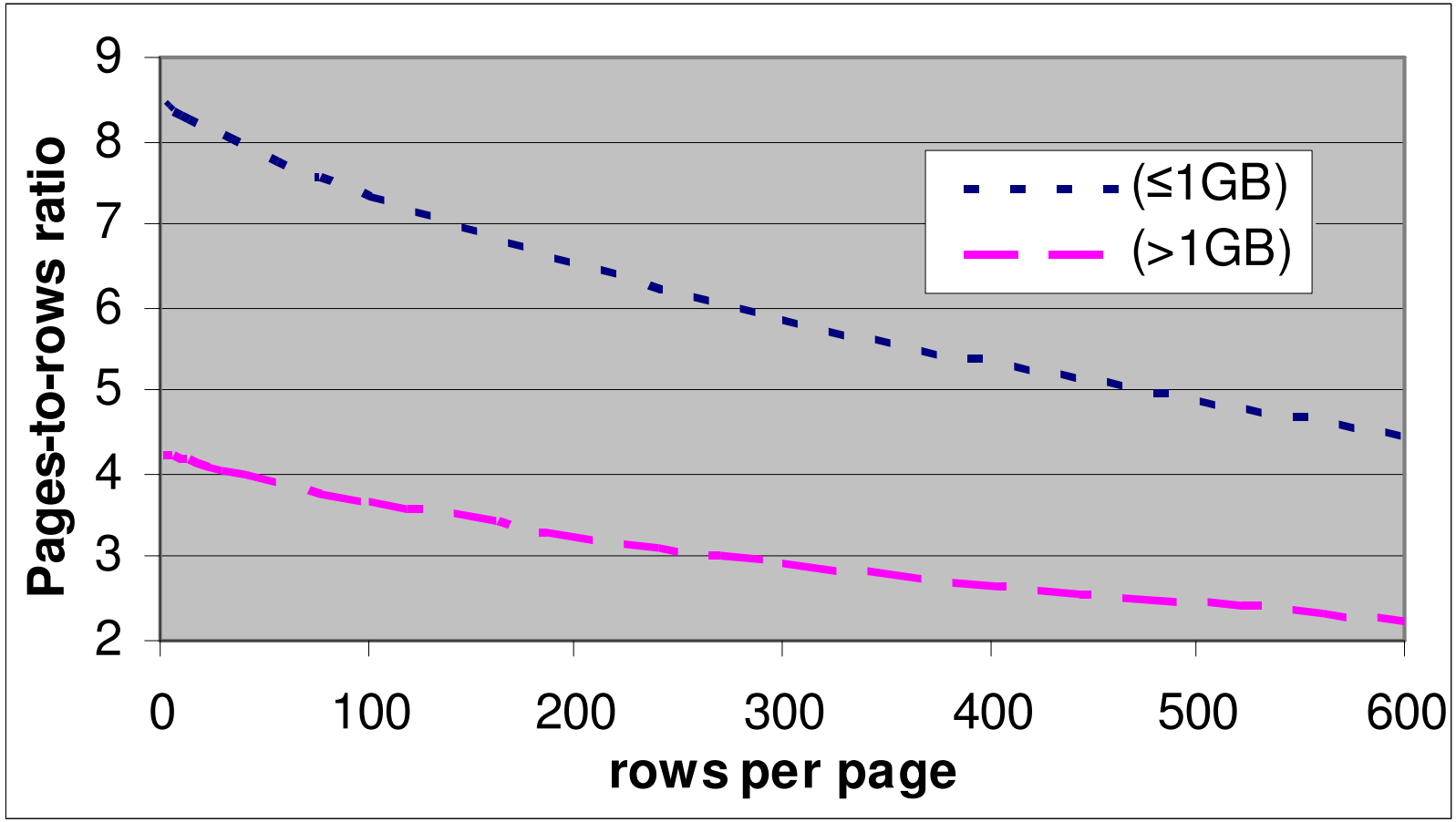


Table scan cost for 50,000 row, 506 pages
Index Seek and Bookmark Lookup cost for ≤ 1GB

≤ 1GB

Bookmark-Table Scan Crossover



Rows ~ 5 + Pages ÷ (CF×(Pages-to-rows ratio)) (≤ 1GB)

~ 11 + Pages ÷ (CF×(Pages-to-rows ratio)) (> 1GB)

1 Bookmark Lookup costs ~7 (≤1GB)
or 3.5(> 1GB) times a scan of 1 page

Bookmark & Table Scan Costs

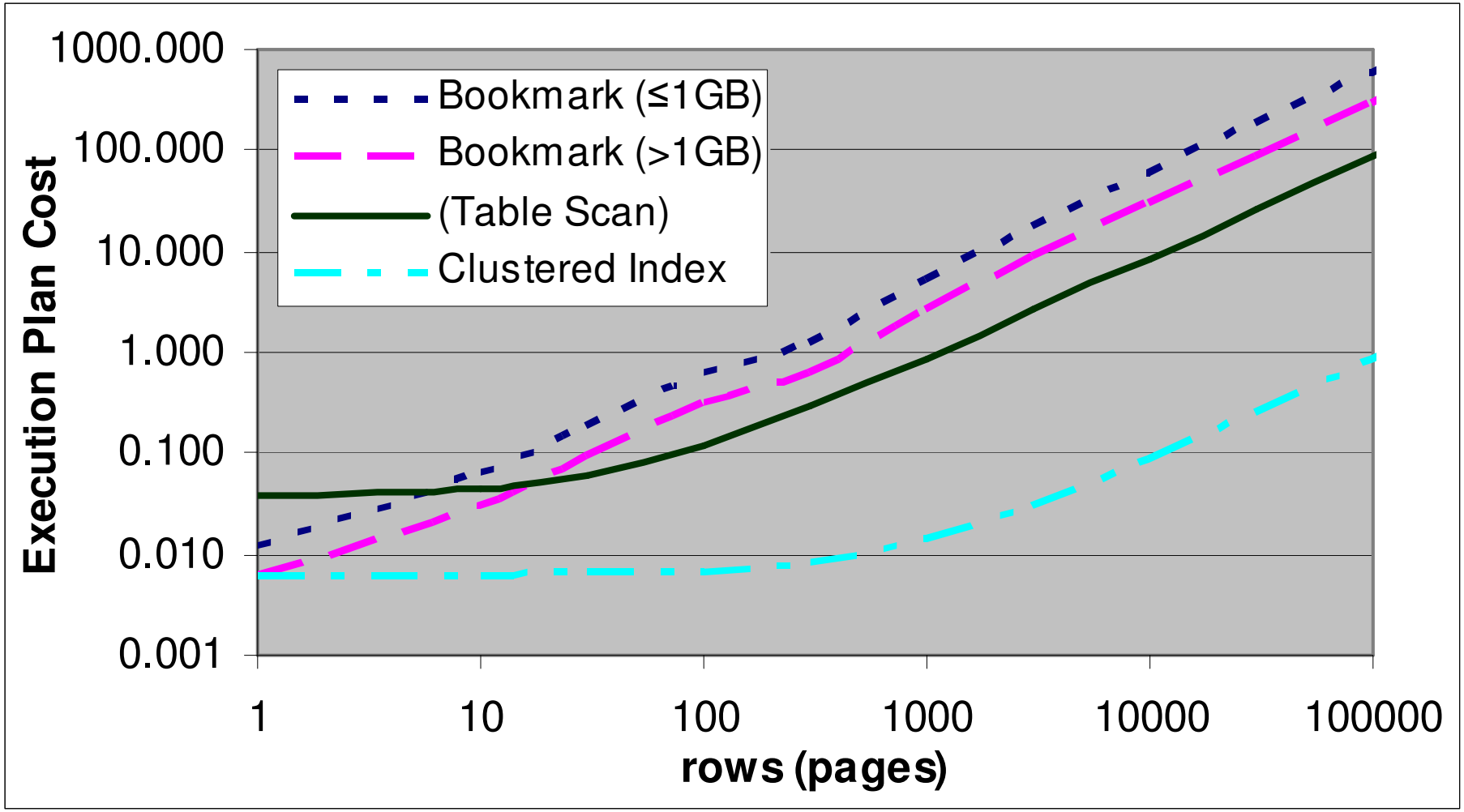


Table scan: cost per page, others: cost per row

HP WORLD 2004
Solutions and Technology Conference & Expo

Aggregates


```
SELECT MIN(x)
FROM M2C
WHERE GroupID = 1
```

Aggregate:
MIN & MAX

Aggregate & Compute Scalar
AVG & SUM

Stream Aggregate/Aggregate

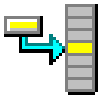
Computing summary values for groups of rows in a suitably sorted stream.

Physical operation:	Stream Aggregate
Logical operation:	Aggregate
Estimated row count:	1
Estimated row size:	28
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.000050
Estimated number of executes:	1.0
Estimated cost:	0.000050(1%)
Estimated subtree cost:	0.00700

Argument:
[Expr1012]=Count(*), [Expr1013]=SUM([M2C].[randDecimal])

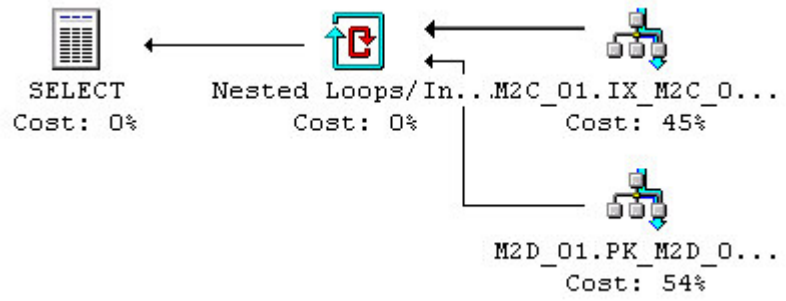
For single row result `I/O: None CPU: 0.0000001/row`

Loop, Hash and Merge Joins

- SQL Server supports 3 types of joins
  - Loop  , Hash  , Merge 
- Hash join subtypes
  - In memory, Grace, Recursive
    - Different settings for SQL Batch & RPC
- Merge join
  - one-to-many
  - many-to-many

Loop Joins

```
SELECT M2C_01.ID, N1C.Value
FROM M2C_01 INNER JOIN M2D_01
ON M2D_01.ID = M2C_01.ID2
WHERE M2C_01.Group1 = @Group1
```



Outer Source (Top Input)

Index Seek
Scanning a particular range of rows from a non-clustered index.

Physical operation:	Index Seek
Logical operation:	Index Seek
Estimated row count:	10
Estimated row size:	38
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000090
Estimated number of executes:	1.0
Estimated cost:	0.006418(45%)
Estimated subtree cost:	0.00641

Argument:
OBJECT:([pubs].[dbo].[M2C_01].[IX_M2C_01_GroupID]), SEEK:([M2C_01].[GroupID]=1) ORDERED FORWARD

Inner Source (Bottom Input)

Clustered Index Seek
Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek
Logical operation:	Clustered Index Seek
Estimated row count:	1
Estimated row size:	42
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000080
Estimated number of executes:	10.0
Estimated cost:	0.007697(54%)
Estimated subtree cost:	0.00769

Argument:
OBJECT:([pubs].[dbo].[M2D_01].[PK_M2D_01]), SEEK:([M2D_01].[ID]=[M2C_01].[ID]) ORDERED FORWARD

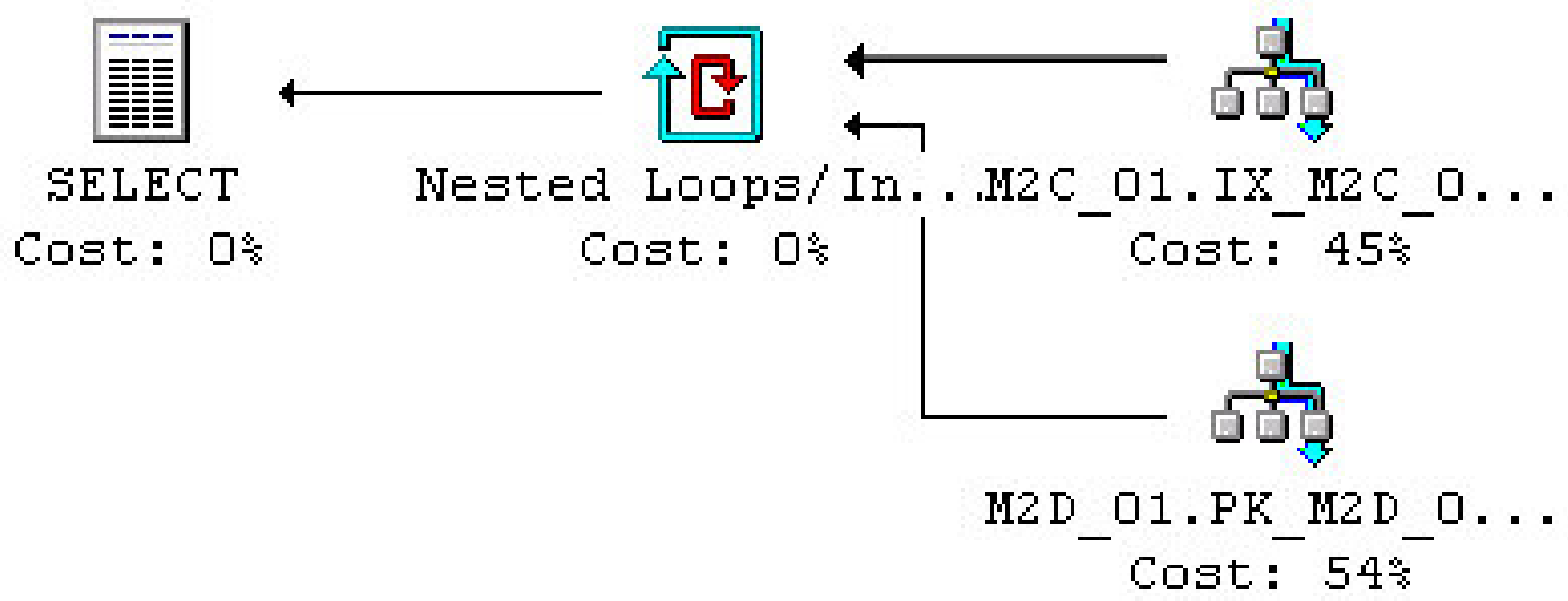
Join

Nested Loops/Inner Join
For each row in the top (outer) input, scan the bottom (inner) input, and output matching rows.

Physical operation:	Nested Loops
Logical operation:	Inner Join
Estimated row count:	9
Estimated row size:	72
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.000042
Estimated number of executes:	1.0
Estimated cost:	0.000042(0%)
Estimated subtree cost:	0.0141

Argument:
OUTER REFERENCES:([M2C_01].[ID])

Loop Join Cost



Complete Loop Join Cost

= Outer Source Cost + Inner Source Cost + Join Cost

Loop Join, cont

Nested Loops/Inner Join

For each row in the top (outer) input, scan the bottom (inner) input, and output matching rows.

Physical operation:	Nested Loops
Logical operation:	Inner Join
Estimated row count:	9
Estimated row size:	72
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.000042
Estimated number of executes:	1.0
Estimated cost:	0.000042(0%)
Estimated subtree cost:	0.0141

Argument:
OUTER REFERENCES:([M2C_01].[ID])

I/O Cost: 0
CPU Cost 0.00000418 per row

Loop Join, Inner Source

Clustered Index Seek

Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek
Logical operation:	Clustered Index Seek
Estimated row count:	1
Estimated row size:	42
Estimated I/O cost:	0.00632
Estimated CPU cost:	0.000080
Estimated number of executes:	10.0
Estimated cost:	0.007697(54%)
Estimated subtree cost:	0.00769

Argument:
OBJECT:([pubs].[dbo].[M2D_01].[PK_M2D_01]), SEEK:([M2D_01].[ID]=[M2C_01].[ID]) ORDERED FORWARD

row count is expected matches per row for each row from outer source (rounded down)

I/O and CPU cost is for 1 execute

Number of executes is row count from outer source

Cost is for all executes

Loop Join Examples

SARG on Outer Source Only

```
SELECT …
FROM M2C
INNER JOIN N1C ON N1C.ID = M2C.CodeID
WHERE M2C.GroupID = @GroupID
```

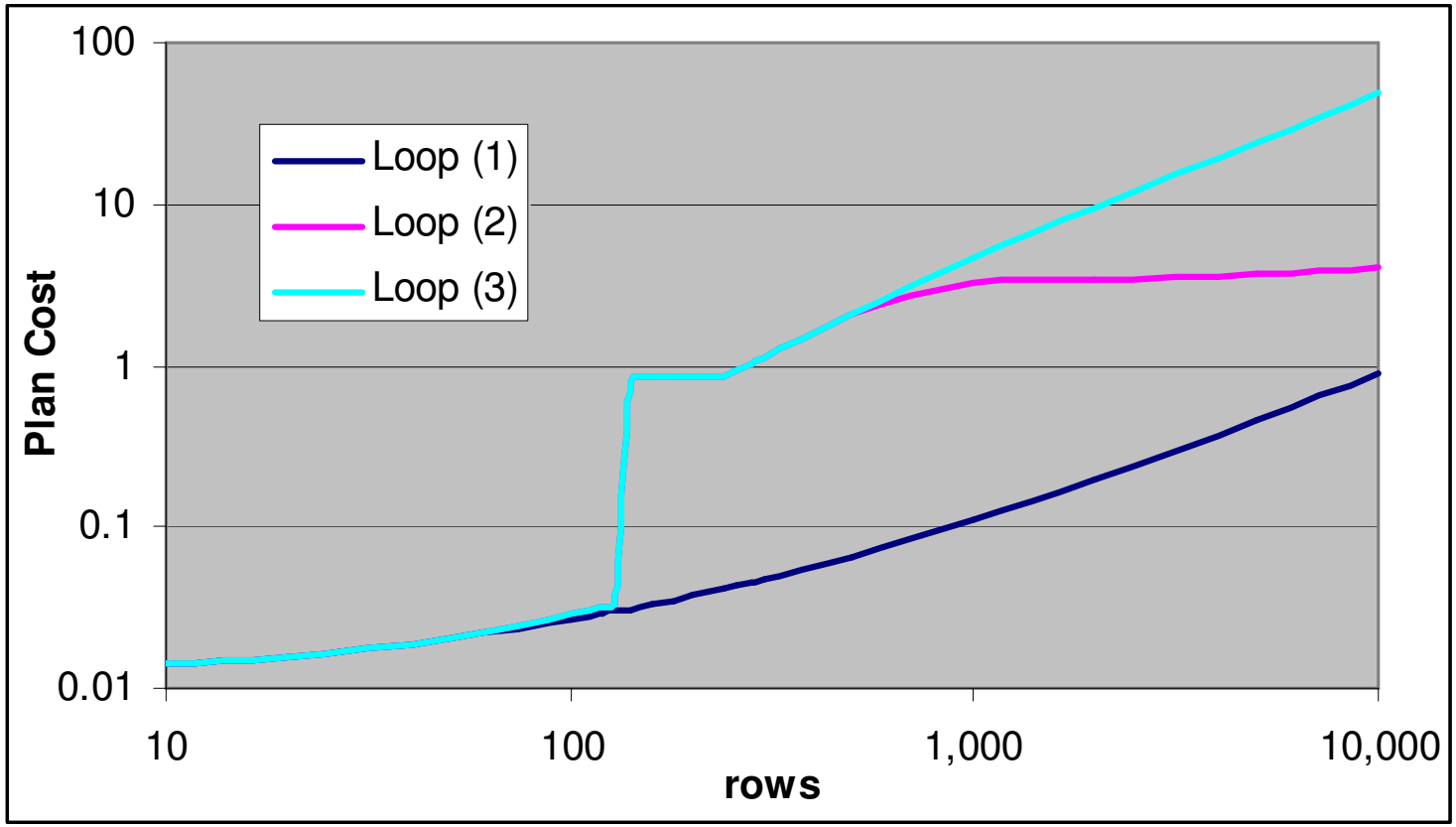
OS Index on SARG
IS Index on join condition

SARG on both sources

```
SELECT …
FROM M2C
INNER JOIN M2D ON M2D.ID = M2C.ID2
WHERE M2C.GroupID = @GroupID AND M2D.GroupID = @GroupID
```

OS Index on SARG
IS Index on SARG followed by join condition

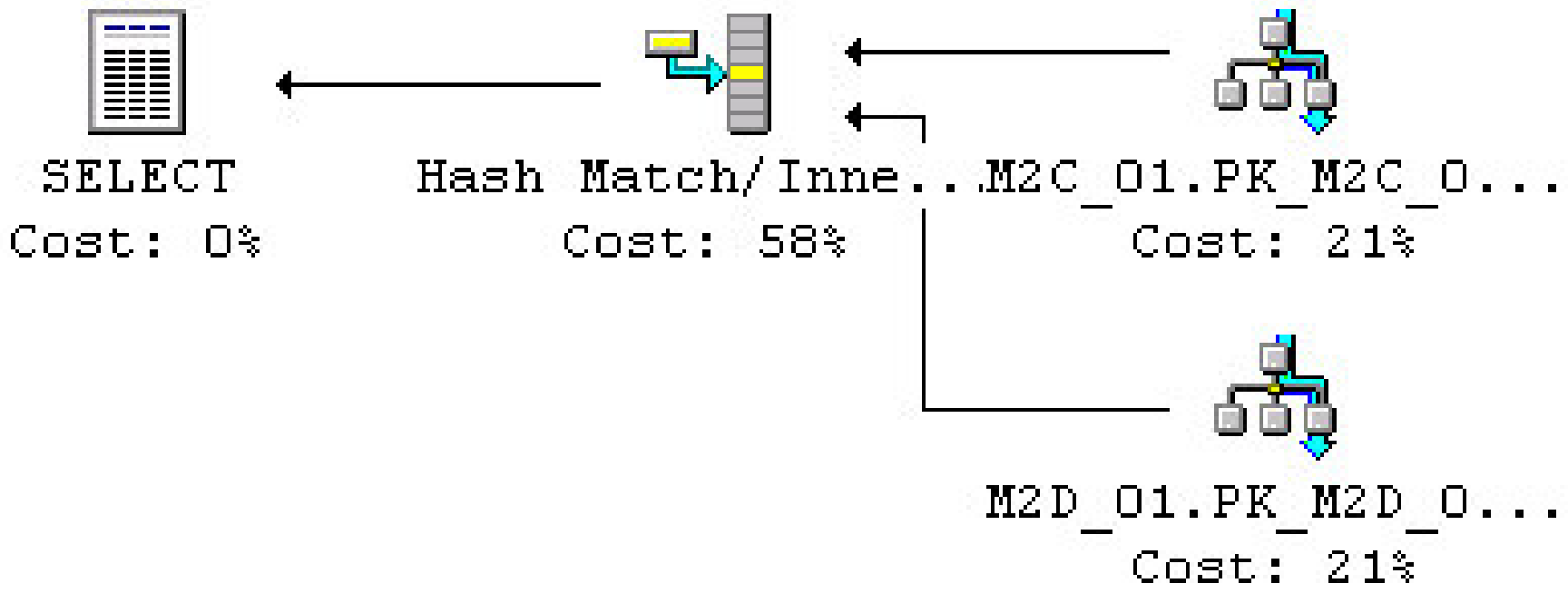
Loop Join Costs



(1) SARG on OS, IS small table
(2) SARG on OS, IS not small
(3) SARG on OS & IS and IS not small

(1) or SARG on both sources and IS is effectively small

Hash Join



SELECT * FROM M2C m INNER HASH JOIN M2D n ON n.ID = m.ID
WHERE m.GroupID = @Group1 AND n.GroupID = @Group2

Hash Join Cost = Outer Source + Inner Source + Hash Match

Hash Join Cost

Outer Source & Inner Source are index seek or scan

1 execute, 1 or more rows

Hash Match/Inner Join

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

Physical operation:	Hash Match
Logical operation:	Inner Join
Row count:	10
Estimated row size:	25
I/O cost:	0.000000
CPU cost:	0.0179
Number of executes:	1
Cost:	0.017942(58%)
Subtree cost:	0.0307
Estimated row count:	9

Argument:
HASH:([m].[ID])=([n].[ID])

Hash Join Cost

SELECT

Retrieves rows from the database. Allows selection of one or many rows or columns from one or many tables.

Physical operation:	SELECT
Logical operation:	SELECT
Estimated row count:	500,000
Estimated cost:	0.049999(0%)
Estimated subtree cost:	16.2

Clustered Index Seek

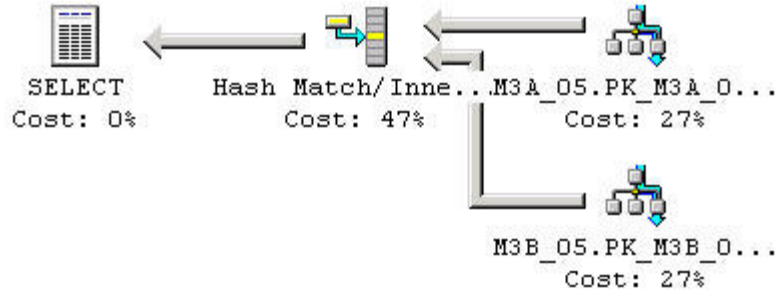
Scanning a particular range of rows from a clustered index.

Physical operation:	Clustered Index Seek
Logical operation:	Clustered Index Seek
Estimated row count:	500,000
Estimated row size:	35
Estimated I/O cost:	3.74
Estimated CPU cost:	0.551
Estimated number of executes:	1.0
Estimated cost:	4.295523(27%)
Estimated subtree cost:	4.30

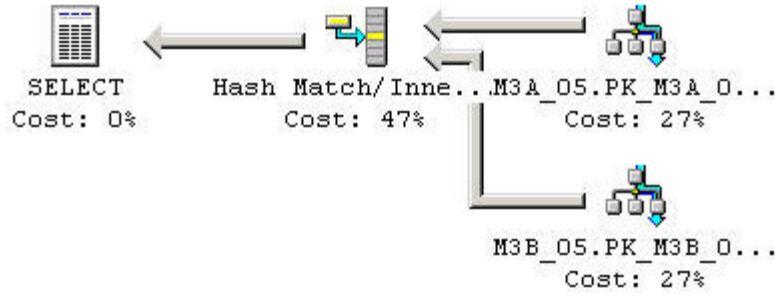
Argument:
OBJECT:([Test2].[dbo].[M3A_05].[PK_M3A_05] AS [a]), SEEK:([a].[GroupID]=1) ORDERED FORWARD

Hash join cost independent of IS column count or size

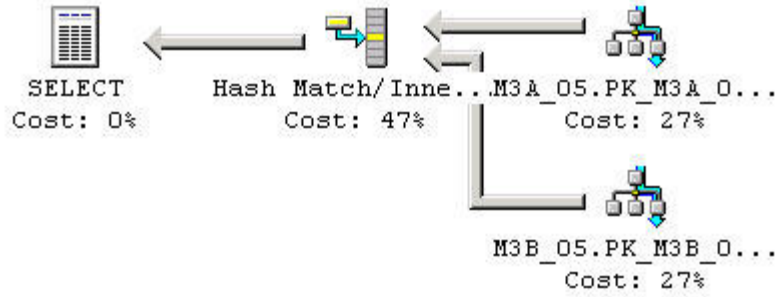
```
Query 1: Query cost (relative to the batch): 33.33%
Query text: SELECT 1 FROM M3A_05 a INNER HASH JOIN M3
```




```
Query 2: Query cost (relative to the batch): 33.33%
Query text:  SELECT a.ID FROM M3A_05 a INNER HASH JOI
```




```
Query 3: Query cost (relative to the batch): 33.33%
Query text:  SELECT b.* FROM M3A_05 a INNER HASH JOIN
```



Hash Join Cost

Q1

SELECT
Retrieves rows from the database. Allows selection of one or many rows or columns from one or many tables.

Physical operation:	SELECT
Logical operation:	SELECT
Estimated row count:	500,000
Estimated cost:	0.049999(0%)
Estimated subtree cost:	16.2

Q2

SELECT
Retrieves rows from the database. Allows selection of one or many rows or columns from one or many tables.

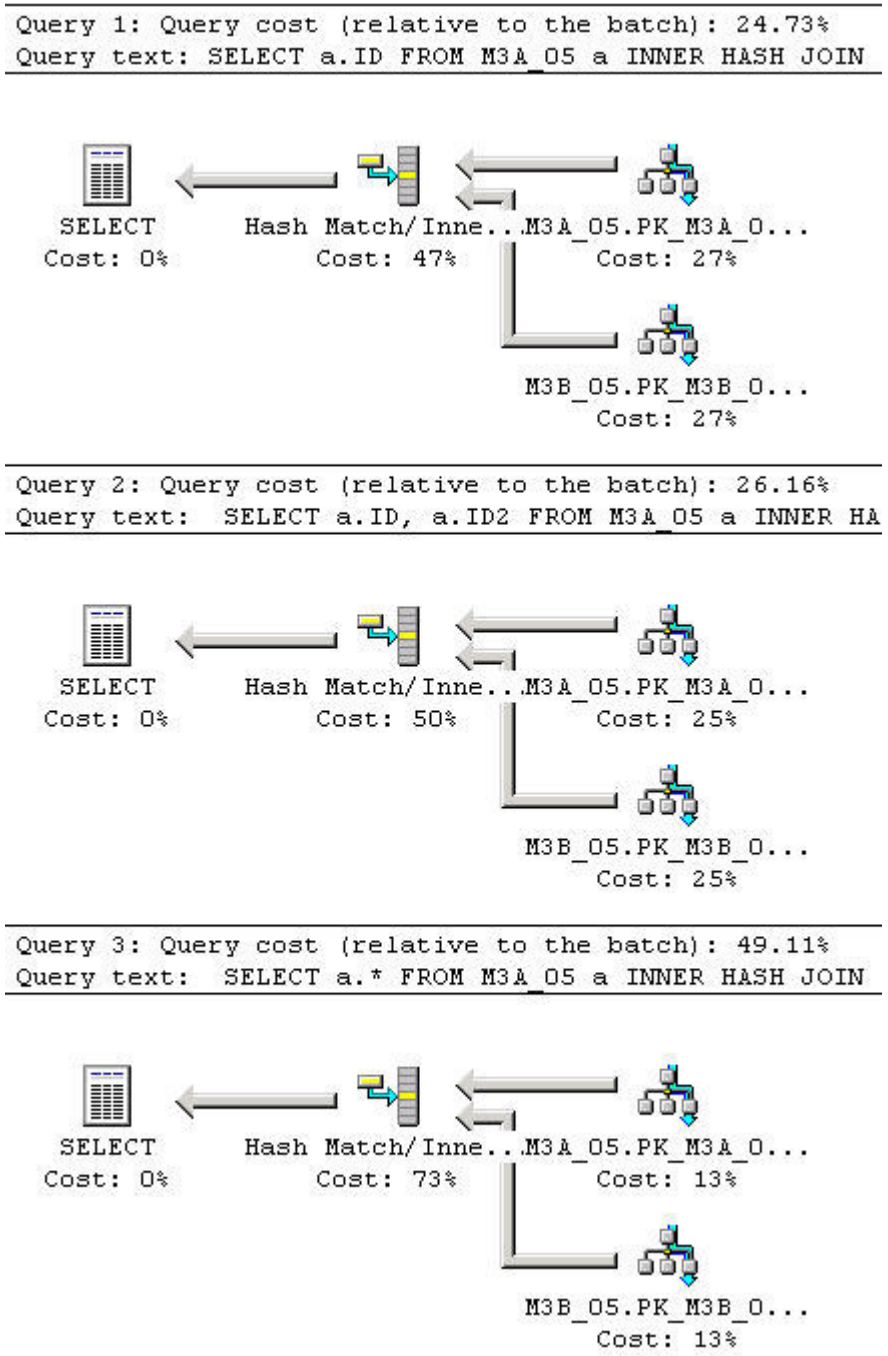
Physical operation:	SELECT
Logical operation:	SELECT
Estimated row count:	500,000
Estimated cost:	0.050002(0%)
Estimated subtree cost:	17.1

Q3

SELECT
Retrieves rows from the database. Allows selection of one or many rows or columns from one or many tables.

Physical operation:	SELECT
Logical operation:	SELECT
Estimated row count:	500,000
Estimated cost:	0.049999(0%)
Estimated subtree cost:	32.2

Hash join cost dependent on OS select size



Hash Join Cost

Q1

Hash Match/Inner Join

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

Physical operation:	Hash Match
Logical operation:	Inner Join
Estimated row count:	500,000
Estimated row size:	15
Estimated I/O cost:	0.000000
Estimated CPU cost:	7.56
Estimated number of executes:	1.0
Estimated cost:	7.563254(47%)
Estimated subtree cost:	16.2

Argument:
HASH:([a].[ID])=([b].[ID])

Q2

Hash Match/Inner Join

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

Physical operation:	Hash Match
Logical operation:	Inner Join
Estimated row count:	500,000
Estimated row size:	19
Estimated I/O cost:	0.000000
Estimated CPU cost:	8.50
Estimated number of executes:	1.0
Estimated cost:	8.503252(50%)
Estimated subtree cost:	17.1

Argument:
HASH:([a].[ID])=([b].[ID])

Q3

Hash Match/Inner Join

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

Physical operation:	Hash Match
Logical operation:	Inner Join
Estimated row count:	500,000
Estimated row size:	83
Estimated I/O cost:	0.000000
Estimated CPU cost:	23.5
Estimated number of executes:	1.0
Estimated cost:	23.543255(73%)
Estimated subtree cost:	32.1

Argument:
HASH:([a].[ID])=([b].[ID])

Hash Join Cost Formula

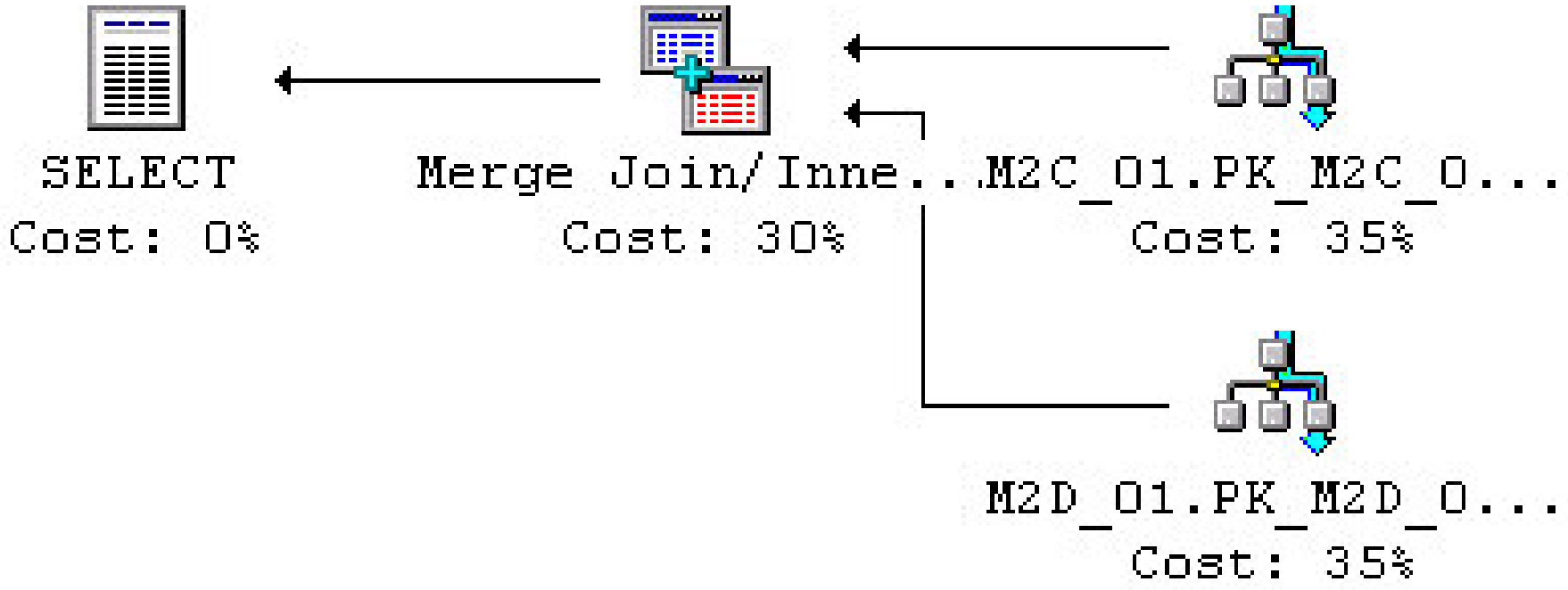
Hash Join

```
Base CPU Cost = 0.017750000 base
Fudge factors + 0.0000001749 (2-30 rows)
              + 0.0000000720 (100 rows)
Cost per row    0.000015091 per row (1:1 join)
                0.000015857 (parallel)
              + 0.000001880 per row per 4 bytes in OS
              + 0.000005320 per additional row in IS

I/O Cost      = 0.0000421000 per row over >64-102MB?
                0.0000036609 per row per 4 byte
```

Hash join spills to tempdb at 64-102MB in 32-bit 1-2GB memory
700MB+ in 64-bit with 32GB memory

Merge Join



SELECT xx **FROM** M2C m **INNER MERGE JOIN** M2D n **ON** n.ID = m.ID
WHERE m.GroupID = @Group1 **AND** n.GroupID = @Group2

Merge Join Cost = Outer Source + Inner Source + Merge cost

Merge Join Cost

```
Cost
CPU: 0.0056046
 + 0.00000446/row

Discrepancy:
0.0000030

1:n
additional rows:
+0.000002370 / row
```

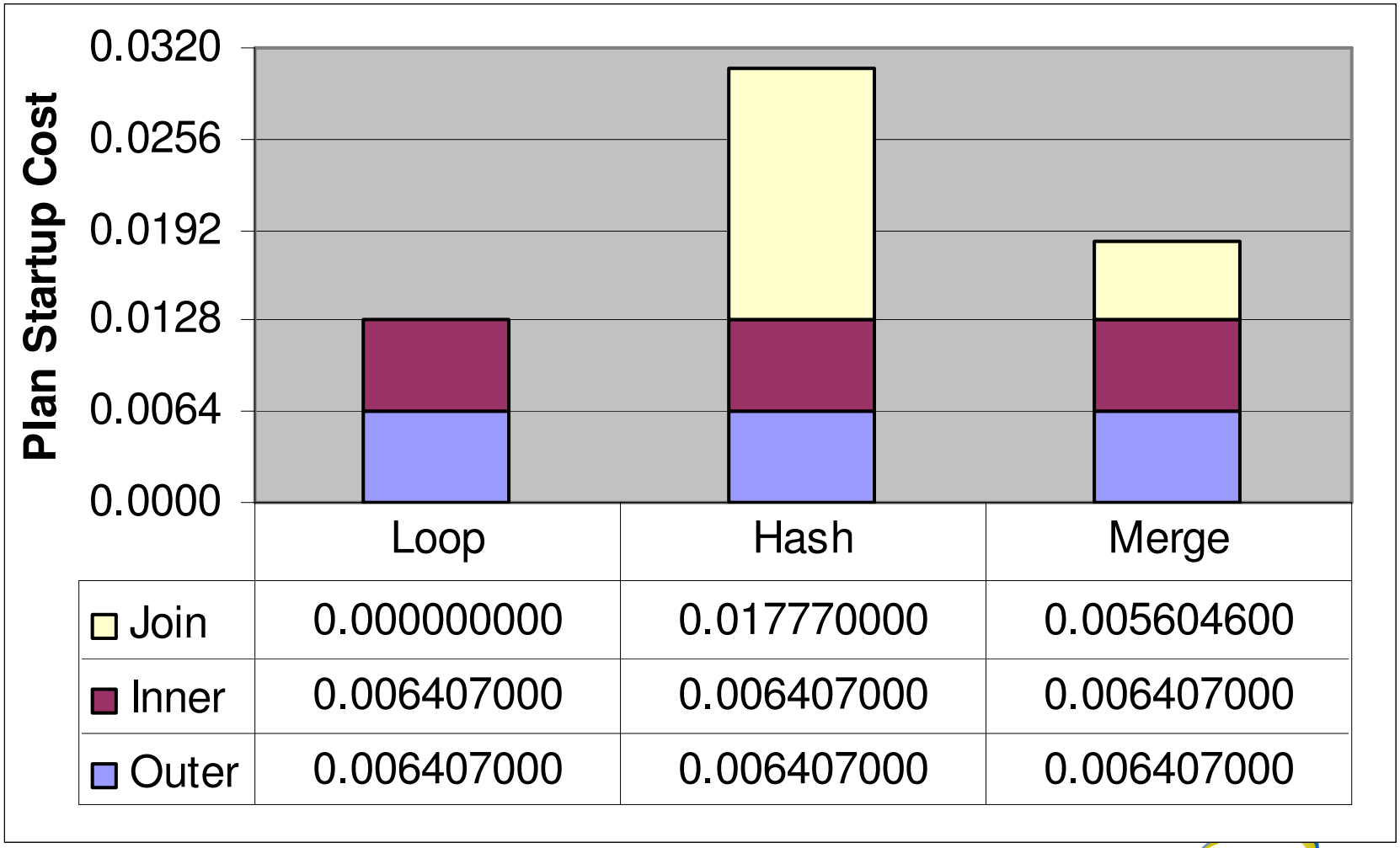
Merge Join/Inner Join

Matching rows from two suitably sorted input tables exploiting their sort order.

Physical operation:	Merge Join
Logical operation:	Inner Join
Row count:	10
Estimated row size:	47
I/O cost:	0.000000
CPU cost:	0.00564
Number of executes:	1
Cost:	0.005648(31%)
Subtree cost:	0.0184
Estimated row count:	9

Argument:
MERGE:([m].[ID])=([n].[ID]), RESIDUAL:([m].[ID]=[n].[ID])

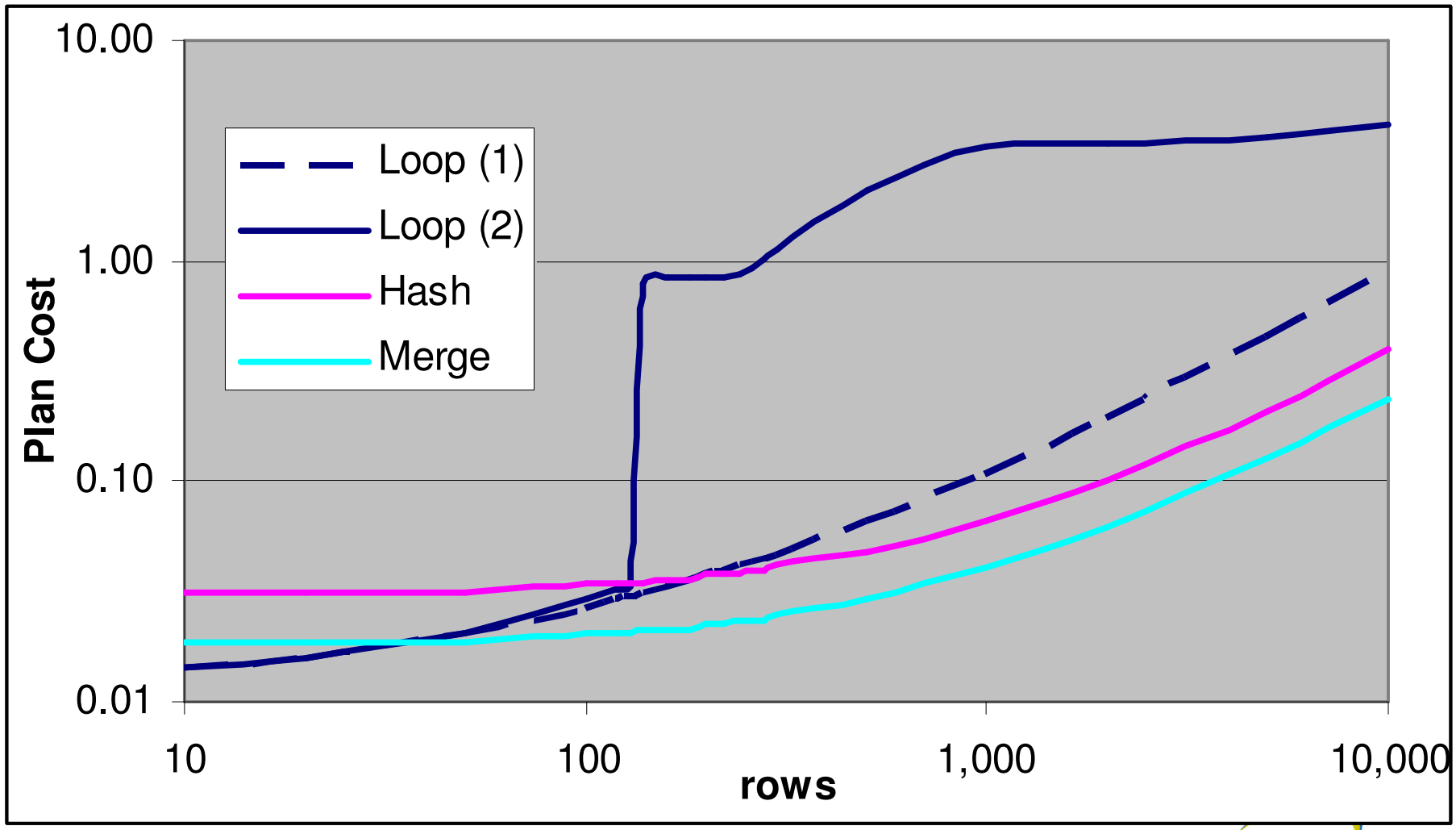
Joins – Base cost



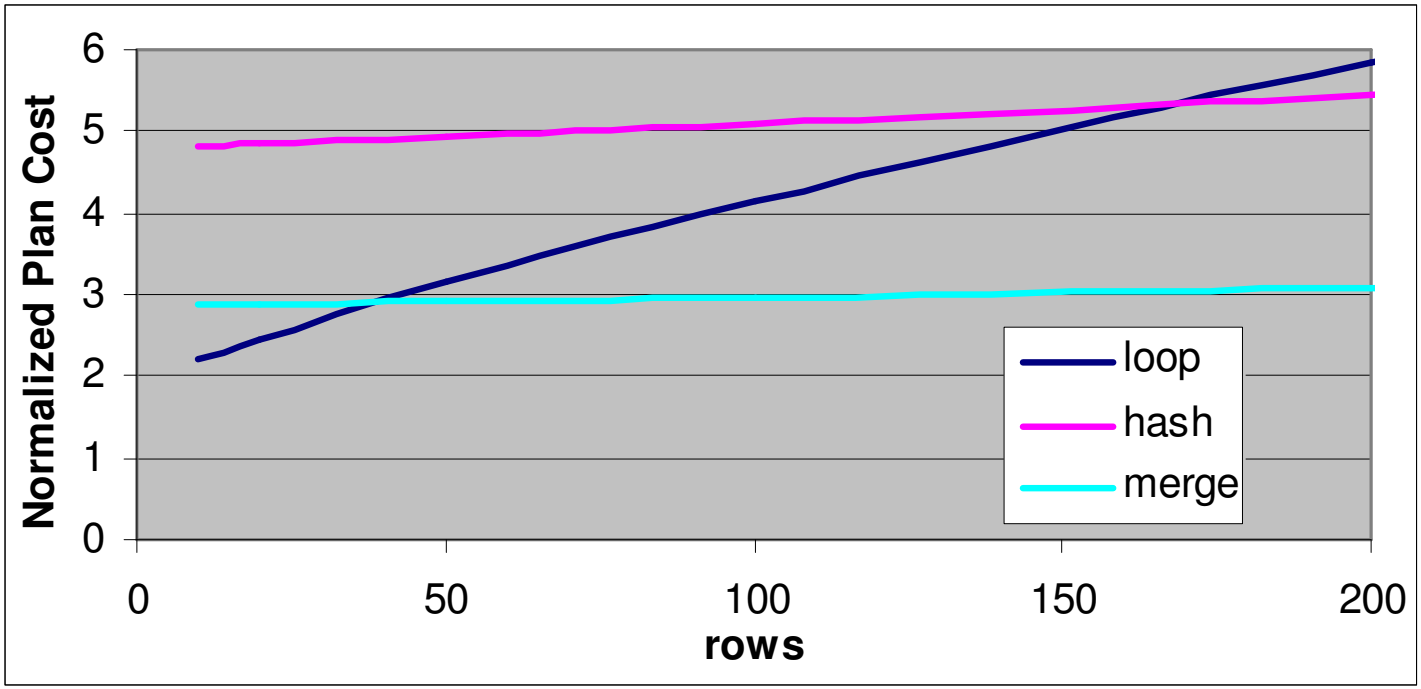
	Loop	Hash	Merge
Join	0.000000000	0.017770000	0.005604600
Inner	0.006407000	0.006407000	0.006407000
Outer	0.006407000	0.006407000	0.006407000

Base cost excludes cost per row

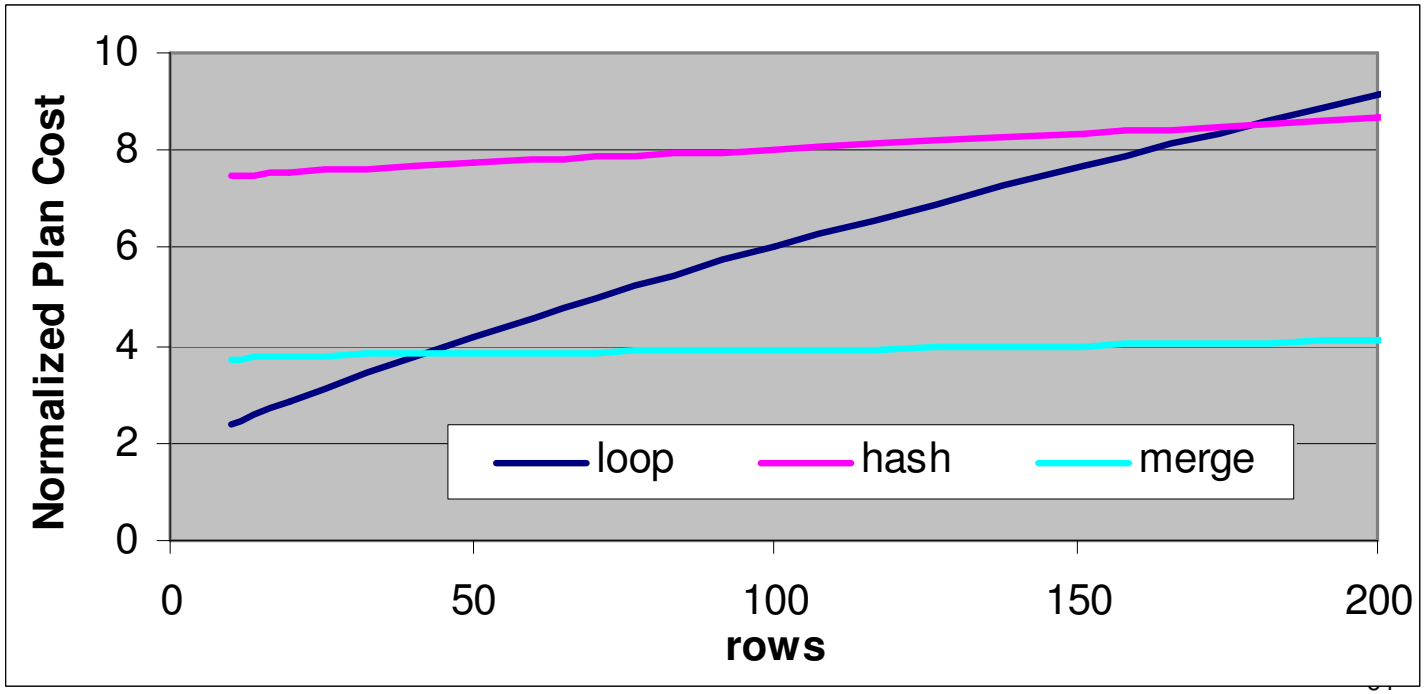
Loop, Hash and Merge Join Costs



≤1GB



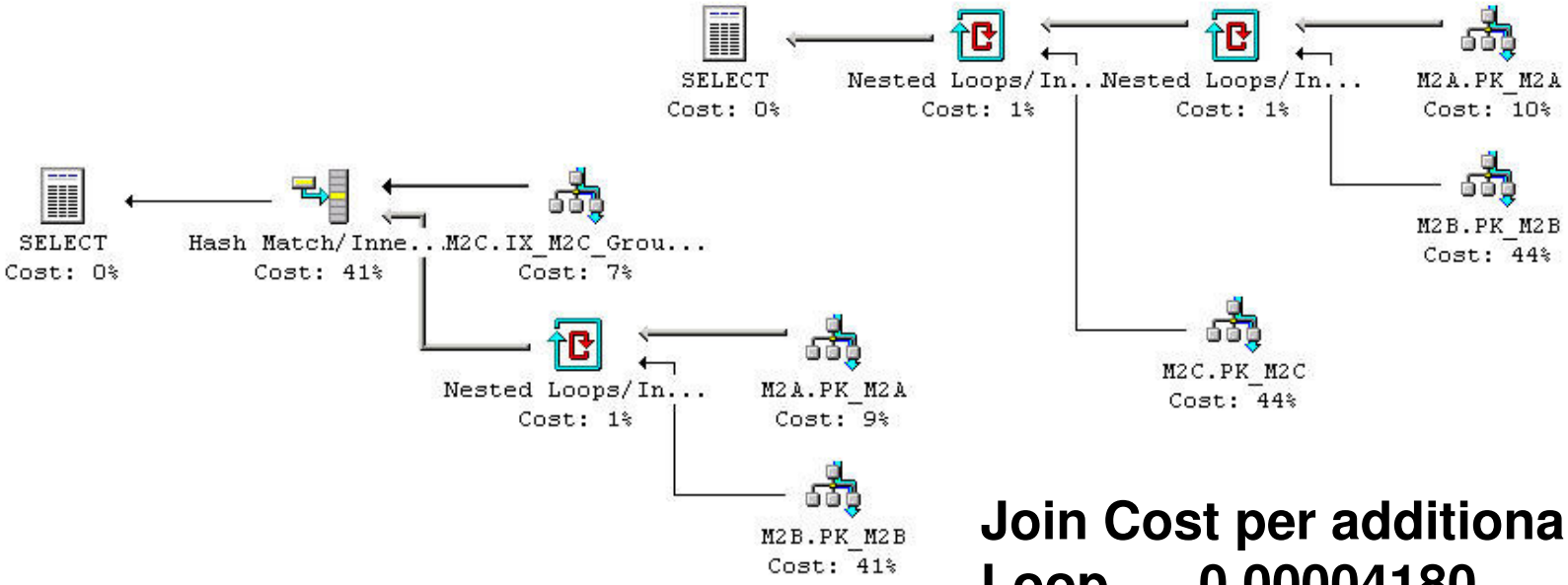
>1GB



Loop(1)

1 to Many Joins

- Each row from OS joins to n rows in IS



Join Cost per additional IS row
Loop 0.00004180
Hash ~0.00000523-531
Merge ~0.00000237
IS Index Seek cost:
0.0000011/row + IO costs

Many-to-Many Merge

Merge Join/Inner Join

Matching rows from two suitably sorted input tables exploiting their sort order.

Physical operation:	Merge Join
Logical operation:	Inner Join
Estimated row count:	3
Estimated row size:	51
Estimated I/O cost:	0.00125
Estimated CPU cost:	0.00582
Estimated number of executes:	1.0
Estimated cost:	0.007079(36%)
Estimated subtree cost:	0.0199

Argument:
MANY-TO-MANY MERGE:([m].[ID2])=([n].[ID]), RESIDUAL:([m].[ID2]=[n].[ID])

```
I/O:   0.000310471 per row
CPU:    0.0056046 + 0.00004908 per row
```

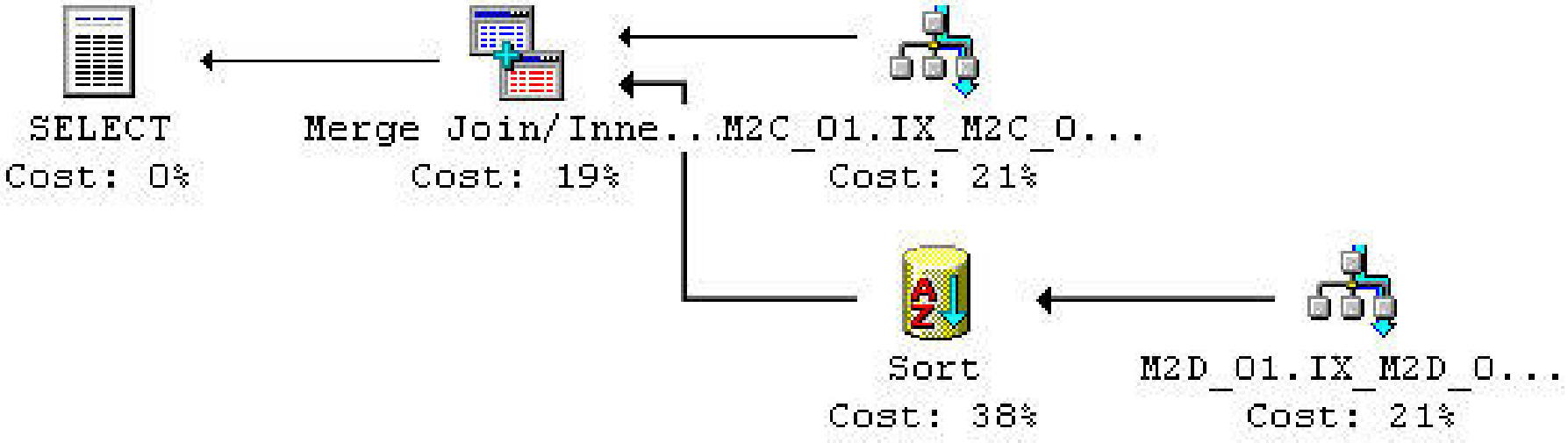
Merge with Sort

Sort

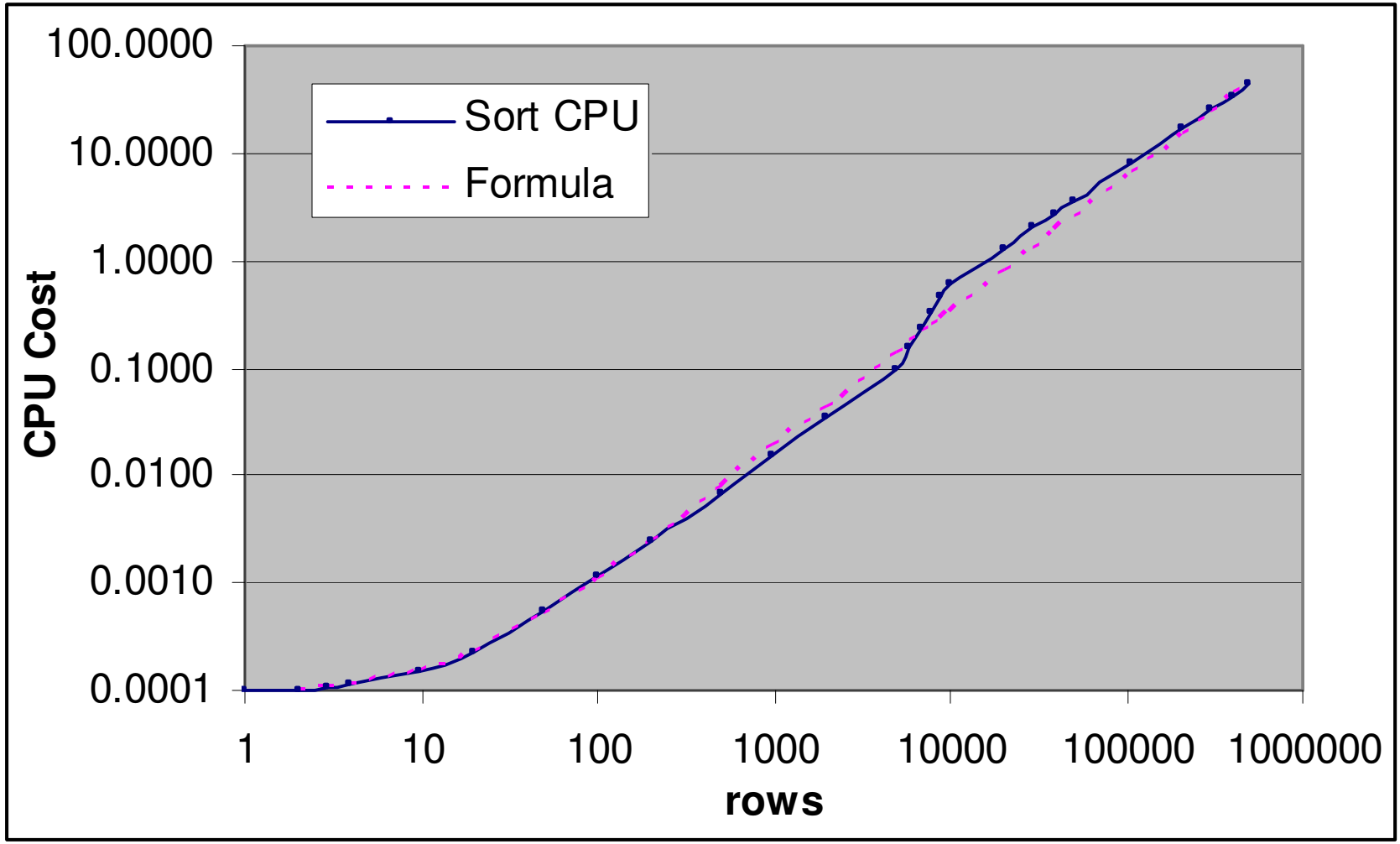
Sorting the input.

Physical operation:	Sort
Logical operation:	Sort
Estimated row count:	10
Estimated row size:	16
Estimated I/O cost:	0.0112
Estimated CPU cost:	0.000152
Estimated number of executes:	1.0
Estimated cost:	0.011413(38%)
Estimated subtree cost:	0.0178

Argument:
ORDER BY:([n].[ID] ASC)

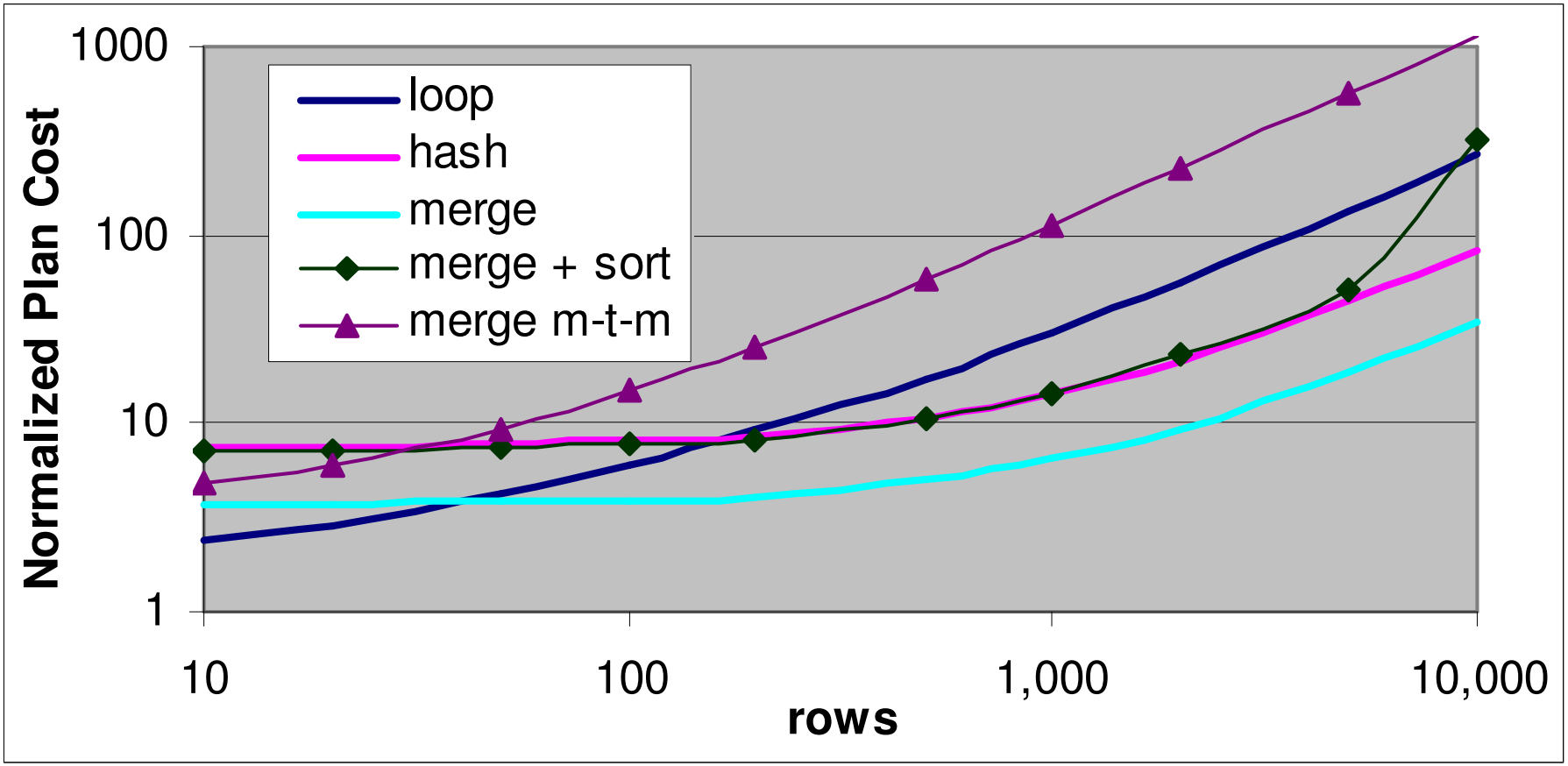


Sort Cost cont.




```
I/O: 0.011261261
CPU: 0.000100079 + 0.00000305849*(rows-1)^1.26
Probably depends on size per row
```

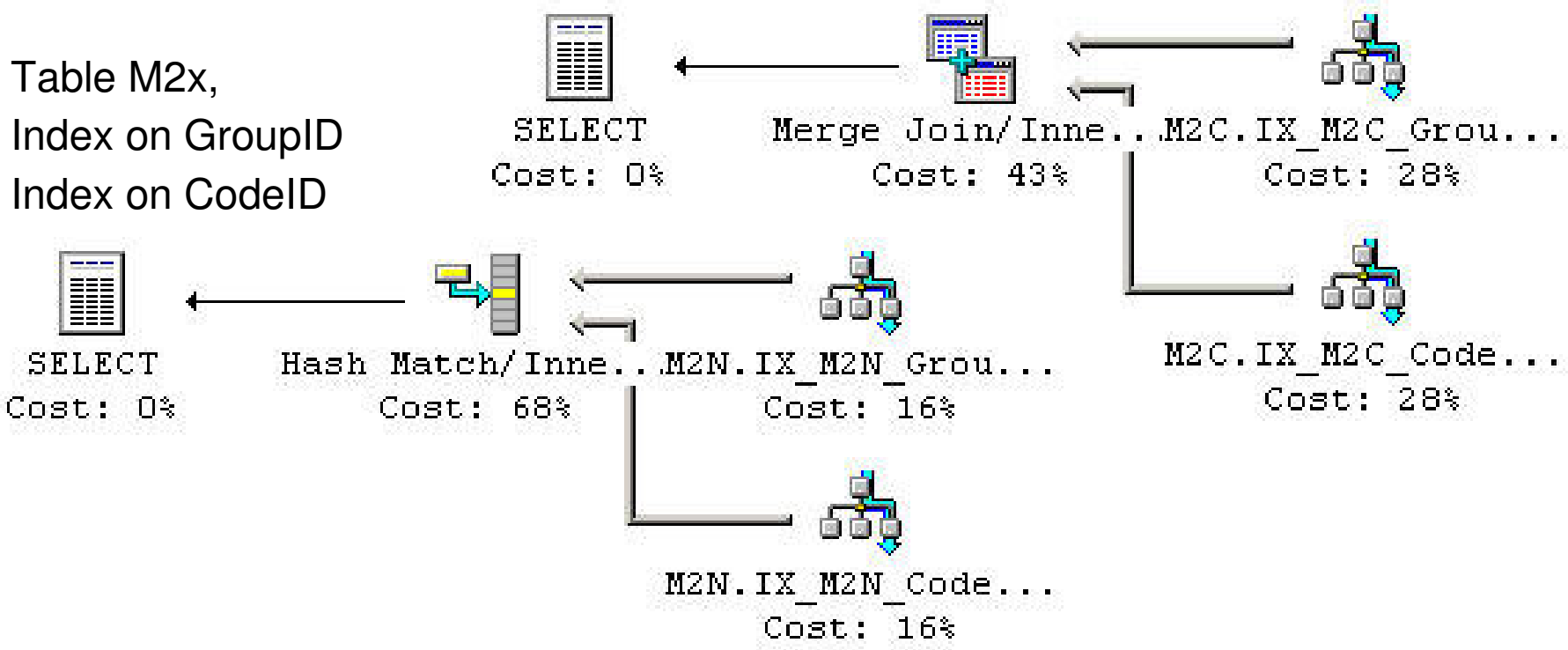
Join Costs Compared



Merge + Sort slightly less expensive than hash join at lower row counts

Index Intersection

Table M2x,
Index on GroupID
Index on CodeID



SELECT xx **FROM** M2C **WHERE** GroupID = @Group **AND** CodeID = @Code

SELECT xx **FROM** M2C a **INNER JOIN** M2C b ON b.ID = a.ID
WHERE a. GroupID = @Group **AND b.**CodeID = @Code

Merge Join cost formula different than previously discussed

Execution Plan Costs Recap

Index Seek	I/O	CPU	Total
≤ 1GB	0.006328500	0.0000796	0.006408100
> 1GB	0.003203425	0.0000796	0.003283025
Additional page	0.00074074/p		
Additional rows		0.00000110/r	

Bookmark Lookup	I/O	CPU	Total
≤ 1GB	0.0062500	0.0000011	0.0062511
> 1GB	0.0031249	0.0000011	0.0031260

Table Scan	I/O	CPU	Total
Base	0.0375785	0.0000785	
Additional page	0.00074074/p		
Additional row		0.0000011/r	

Logical IO count

Example: Index Depth 2, rows per page: 100

I/O per additional row	
Bookmark Lookup (Heap)	1
Bookmark Lookup (Clustered)	2
Loop Join (IS)	2

I/O per addition 100 rows	
Index Seek	1
Hash & Merge join	2

Very little relation between IO count and plan cost for different component operations

IO count comparisons more relevant for similar operations

Accurate Performance Testing

- Execution Plan - match
  - Raw size of DB not as important:
    - 1M customers actual, 10K test
  - Cardinality more important
    - 1 Customer – 10 orders – 10 order items per order
- Statistics & actual data queried
  - Statistics could be accurate but actual queries favors different distribution

Aggregates – multiple output rows

Stream Aggregate/Aggregate

Computing summary values for groups of rows in a suitably sorted stream.

Physical operation:	Stream Aggregate
Logical operation:	Aggregate
Estimated row count:	2,000
Estimated row size:	28
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.0149
Estimated number of executes:	1.0
Estimated cost:	0.014900(56%)
Estimated subtree cost:	0.0264

Argument:
GROUP BY:([M2C_08].[ID2]) [Expr1002]=SUM([M2C_08].[randDecimal])

CPU
Cost per result row: 0.000007450/row

Hash Match/Aggregate

Insert each input row into a hash table, grouping on the GROUP BY columns and computing aggregate expressions.

Physical operation:	Hash Match
Logical operation:	Aggregate
Estimated row count:	2,000
Estimated row size:	28
Estimated I/O cost:	0.000000
Estimated CPU cost:	0.0554
Estimated number of executes:	1.0
Estimated cost:	0.055452(83%)
Estimated subtree cost:	0.0670

Argument:
HASH:([M2C_08].[ID2]) [Expr1002]=SUM([M2C_08].[randDecimal])

CPU Cost per result row:
0.01777 + 0.0000188

Execution Plan Cost Summary

- Plan costs do not include RPC cost
- Plan costs are a model
- Index seek independent of index depth
- Bookmark L/U independent of table organization
- Logic by itself does not influence cost
- Costs are not influenced by lock hints
- Populate test DB with accurate cardinality

HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

Quantitative Performance Analysis

Joe Chang
jchang6@yahoo.com

Subjects

- Cost measurement
  - Test procedure, Unit of measure
- Query Costs
  - Single row, table scan, multi-row, joins
    - Discrepancies with plan costs
    - Logical I/O, Lock Hints, Conditions
- Database design implications
  - Design, coding and indexes

Test Procedure

- Load generator
  - Drive DB server CPU utilization as high as possible (>85%)
  - Multiple load generators running same query
    - Single thread will not fully load DB server
    - Network propagation time is significant
    - Most queries run on single processor
    - Server may have more than one processor
- Component operation cost derived
  - Comparing two queries differing in one op

Unit of Measure – CPU-Cycles

- Query costs measured in CPU-cycles
  - Alternative: CPU-sec

Cost = Runtime (sec) × CPU Util. × Available CPU-cycles ÷ Iterations

Available CPU-cycles = Number of CPUs × Frequency

Example 4 x 700MHz = 2.8B cycles/sec

CPU-cycles does not imply CPU instructions,
Unit of time same as CPU clock

1GHz CPU: time unit = 1ns
2GHz CPU: time unit = 0.5ns

CPU-Cycles dependencies

CPU-cycles on one processor architecture has no relation to another – ex. Pentium III, Pentium 4, Itanium, Opteron

Some platform dependencies – cache size, bus speed, SMP

Processor	System/Cache/Mem	Performance	Cost / trans.
Itanium 2	4 x 1.5GHz/6M /64G	121,065	2.974M
Xeon	4 x 3.0GHz/4M /32G	102,667	7.013M
Opteron	4 x 2.2GHz/1M /32G	105,687	4.996M

Notes:
Some platform dependencies – cache size, bus speed, SMP

Scaling dependencies

2 CPUs does not imply 2X performance compared to 1 CPU. This implies that costs are higher for more CPUs.
Example:

1 CPU/1GHz – 1000 ops/sec – 1.00M CPU-cycles/op
2 CPU/1GHz – 1800 ops/sec – 1.11M CPU-cycles/op

Scaling Expectation
1.8X from 1 to 2 CPUs, 1.7X from 2 to 4CPUs, 1.6X for each additional doubling

Itanium2 1.5GHz/6M TPC-C performance, W2K3, S2K

CPU	tpm-C	tpm/CPU	Scale factor
4	121,065	30,266	
32	577,531	18,048	1.68X
64	786,646	12,291	1.59X

Cost Structure - Model

Stored Procedure Call Cost =

RPC cost (once per procedure)
+ Type cost (once per procedure?)
+ Query costs (one or more per procedure)

Query – one or more components

Component Cost =

Cost for component operation base
+ Cost per additional row or page

Only stored procedures are examined

RPC Cost

Cost of RPC to SQL Server includes:

1) Network roundtrip
2) SQL Server handling costs

Calls to SQL Server are made with RPC (not SQL Batch)
Profiler -> Event Class: RPC

ADO.NET Command Type Stored Procedure or Text with parameters
Command Text without parameters: SQL Batch

Type Cost?

Blank Procedure: ~250,000 CPU-cycles

```
CREATE PROC p_ProcBlank AS
RETURN
```

Proc with Single Query ~320K CPU-cycles

```
CREATE PROC p_ProcSingleQuery AS
SELECT … FROM TableA WHERE ID = @ID
```

Proc with Two Queries ~360K CPU-cycles

```
CREATE PROC p_ProcTwoQueries AS
SELECT … FROM TableA WHERE ID = @ID
SELECT … FROM TableB WHERE ID = @ID
```

RPC Cost ~250K CPU-cycles,
Type Cost ~30K CPU-cycles,
Query Cost ~40K CPU-cycles

RPC Cost

Costs in CPU-Cycles

Processor	**PIII**	**PIII X**	**Xeon**	**Opteron**	**Itanium 2***		**It2**
CPUs	**2**	**4**	**2**	**2**	**2**	**4**	**8**
RPC cost	**140K**	**200K**	**250** 270K **(2.xx driver)**	**140K**	**155K**	**290K**	**350K**
Type Cost Select	**20-30K**	**~5K**	**35-55K**	**~20K**	**~8K**		

Systems:

Pentium III	**2x 600MHz/256K, 2x 733MHz/256K**,	
PIII Xeon	2x 500MHz/2M, 4x 700MHz/2M, 4x900/2M	
Xeon (P4)	2x 2.0GHz/512K	**2x 2.4GHz/512K**
Opteron		**2x 2.2GHz/1M**
Itanium 2	2x 900MHz/1.5M	**8x1.5GHz/6M**

OS: W2K, W2K3, various sp
SQL Server 2000, various sp
PIII: Intel PRO/100+, Others: Broadcom Gigabit Ethernet driver 5.xx+

*Itanium 2 system booted with 2, 4 or 8 processors
(4P config may have had procs from more than 1 cell)

RPC Cost – Fiber versus Threads

Costs in CPU-Cycles

PIII Xeon - TCP	**1P**	**2P**	**4P**
FE-Thread	105K	150K	200K
FE-Fiber	95K	120K	170K
Xeon - TCP			
GE-Thread	210K	250K	
GE-Fiber	200K	230K	
Xeon - VI			
VI Thread		190K	
VI Fiber	160K	180K	

Itanium 2 - TCP	**1P**	**2P**	**4P**	**8P**
Thread	105K	155K	290K	350K
Fiber	95K	145K	260K	300K

Broadcom Gigabit Ethernet driver 5.xx, 6.xx, 7.xx (270K for 2P 2.xx driver)
VI: QLogic QLA2350, drivers: qla2300 8.2.2.10, qlvika 1.1.3

RPC Cost TCP vs Named Pipes

Costs in CPU-Cycles

PIII Xeon 4P	**TCP**	**named pipes**
FE-Thread	200K	315K
FE-Fiber	170K	370K

Xeon, Thread	**1P**	**2P**
GE, TCP	210K	250K
GE, Named Pipes	320K	360K

Broadcom Gigabit Ethernet driver 5.xx, 6.xx, 7.xx (270K for 2P 2.xx driver)
VI: QLogic QLA2350, drivers: qla2300 8.2.2.10, qlvika 1.1.3

RPC Costs – owner, case

Costs in CPU-Cycles

	PIII	PIII X	P4/Xeon
RPC cost	140K	140K?	250K
sp_executesql		210K	

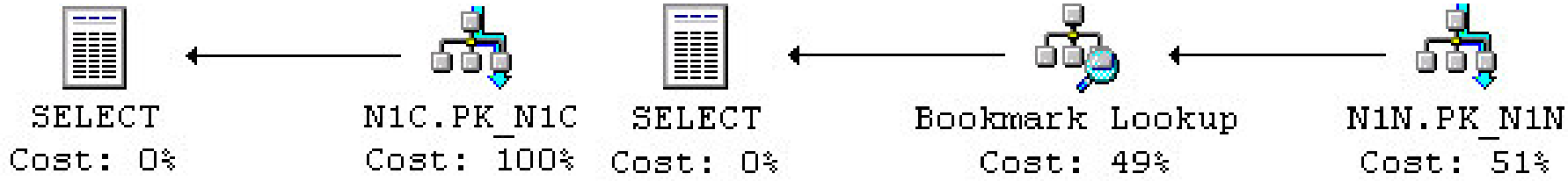
Unspecified owner, Ex: user1 calls procedure owned by dbo
+100K on 4P PIII, +100K on 2P Xeon, 300K on 8P Itanium2
Case mismatch:
Actual procedure: p_Get_Rows
Called procedure: p_get_rows
+100K on 4P PIII, +150K on 2P Xeon, +300K on 8P Itanium2

Single Row Select Costs

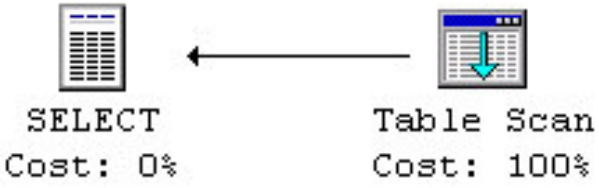
- Clustered Index Seek
  - Does cost depend on index depth?
  - Role of I/O count in cost
- Index Seek with Bookmark Lookup
  - Does cost depend on table type?
    - Heap versus clustered index
- Table Scan

Single Row Logical IO Count

SELECT Value FROM N1x WHERE ID = 1



Clustered Index



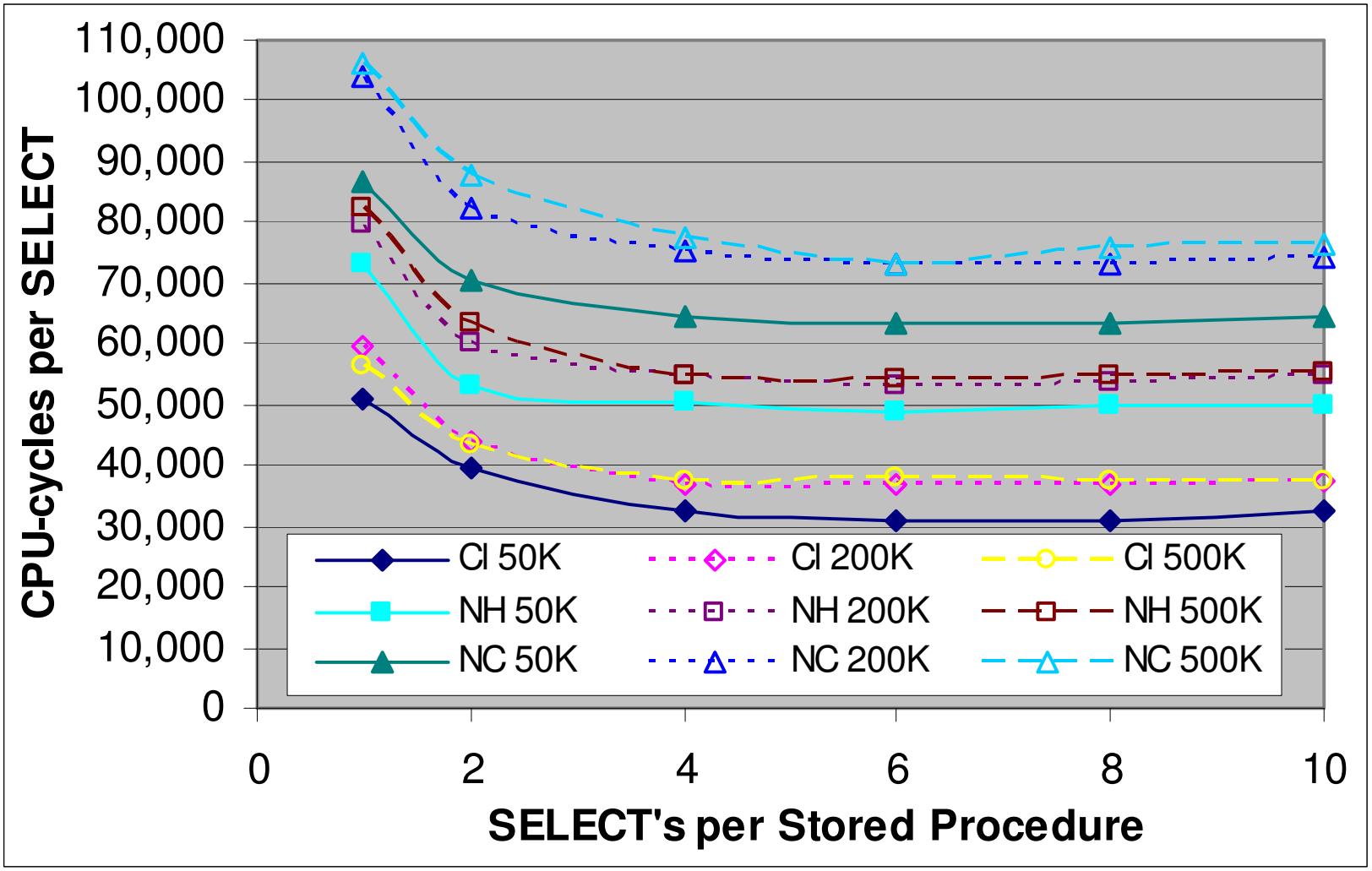
No index

Nonclustered Index

All data fits in 1 page

Index Type	Index IO	Table I/O	Total IO
Clustered	2	0	2
Covered	1	0	1
Nonclustered	1	1	2
Table Scan	0	1	1

Single Row Index Seek Cost per Query



Index depth: 50K rows -2 both, 200K 3 CI, 2 NC, 500K 3

HP WORLD 2004
Solutions and Technology Conference & Expo

Single Row Cost Summary

- Index depth
  - Plan – no, True cost -yes
  - Cost versus index depth not fully examined
  - Fill factor dependence not tested
- Bookmark Lookup – Table type
  - Plan cost -no
  - True cost higher for clustered index than heap

Multi-row Select Queries

Queries return single decimal aggregate
- Not a test of network bandwidth

Single Table Example

```
SELECT @Count = count(*), @Value1 = AVG(randDecimal)
FROM M3C_01 WHERE GroupID = @ID
```

1 Count, 1 AVG on 5 byte decimal

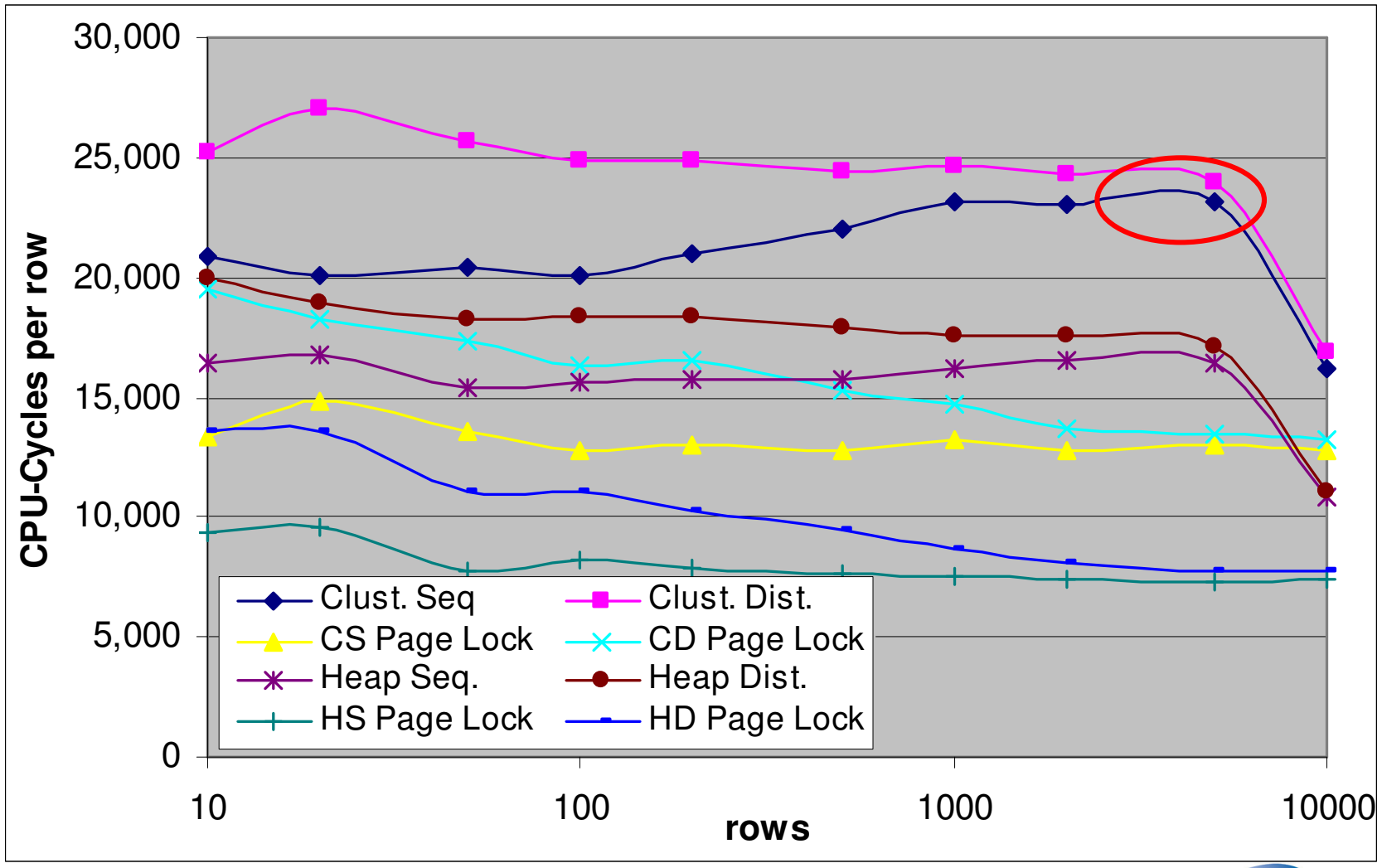
Join Example

```
SELECT count(*), AVG(m.randDecimal), min(n.randDecimal)
FROM M2A_02 m
INNER LOOP JOIN M2B_02 n ON n.ID = m.ID
WHERE m.GroupID = @ID AND n.GroupID = @ID
```

1 Count, 2 AVG on 5 byte decimal

Table Scan tests return either
- a single row
- 1 Count, 1 aggregate on 5 byte decimal

Multi-row Bookmark Lookups



2x2.4GHz/512KB Xeon

Table lock shift at 5K rows ?

Table Scan – Component Cost

Total cost = RPC + Type + Base + per page costs

	PIII (X)	P4/Xeon	Opteron	It2 2P	8P
Type + Base cost	60K/40K	145K		35K	90K
Cost per page:					
NOLOCK:	24K	-		16K	23-35K
TABLOCK:	24K	25K		16K	
PAGLOCK:	26K	26K	20K	17K	23-35K
ROWLOCK:	140K	250K	110K	100K	150K

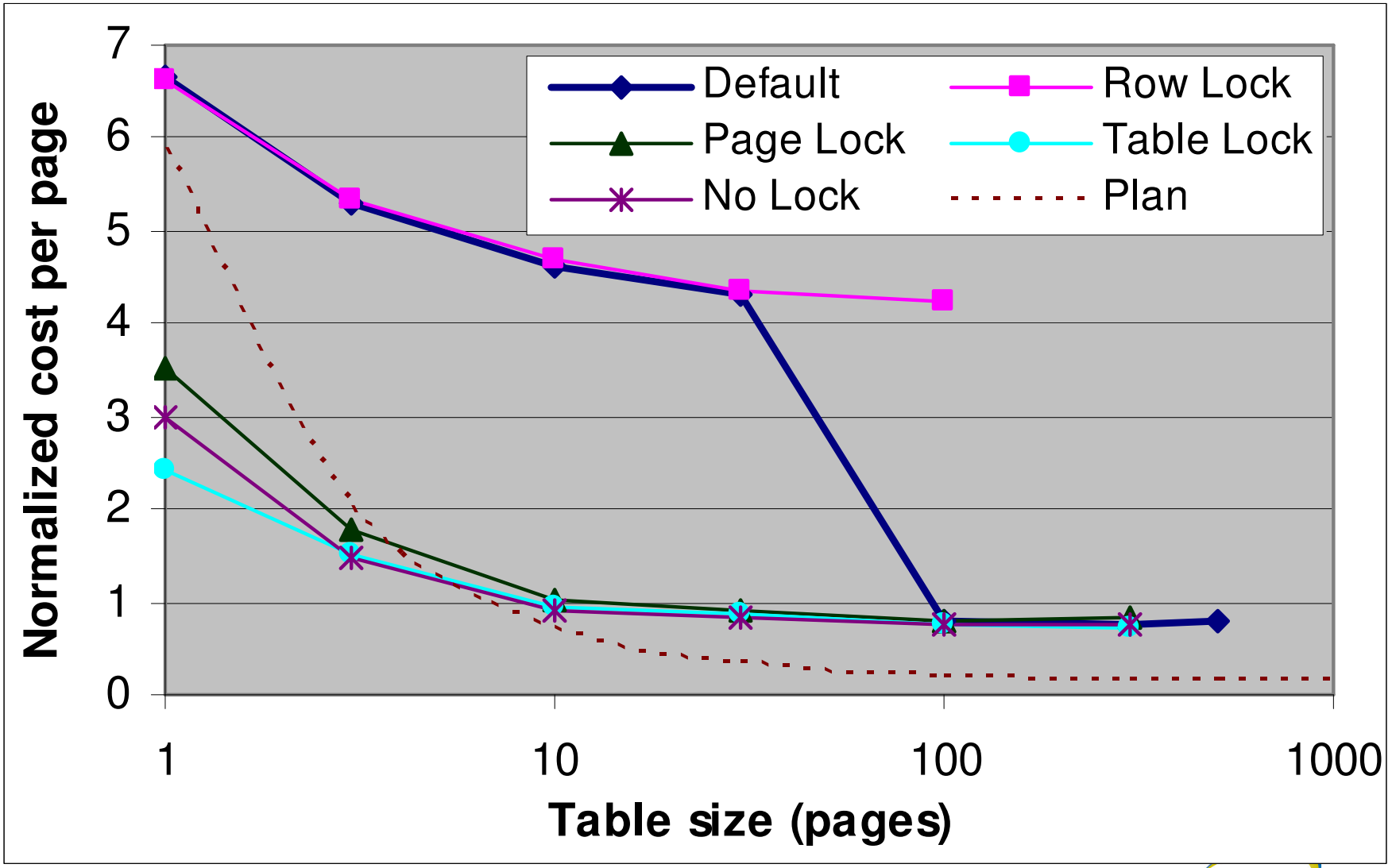
Measured Table Scan cost formulas for 99 rows per page

Costs in CPU-Cycles per page

Table Scan or Index Scan – Plan Formula

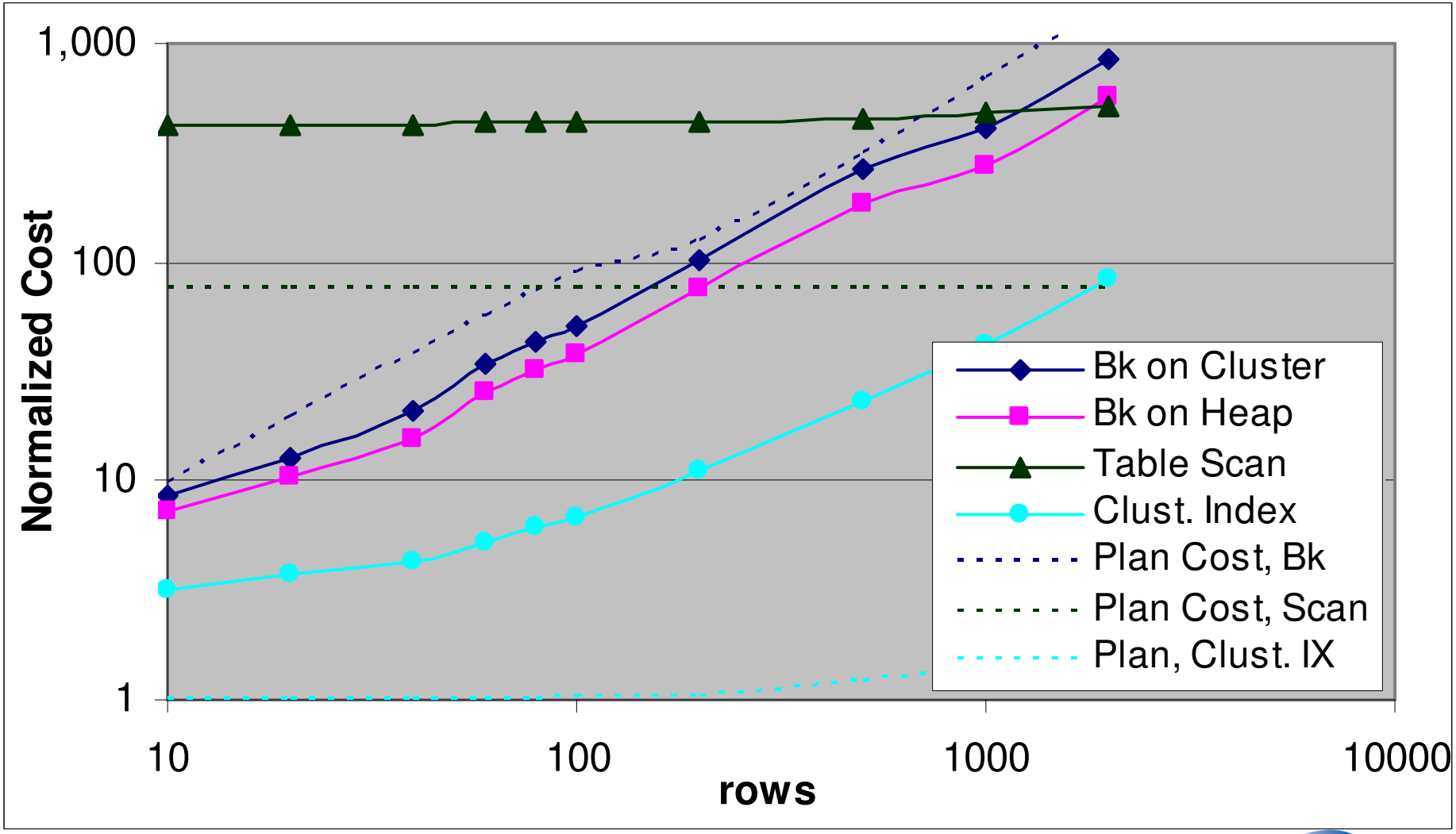
```
I/O:   0.0375785 + 0.0007407 per page
CPU:   0.0000785 + 0.0000011 per row
```

Table Scan Cost per Page



2x733MHz/256KB Pentium III

Bookmark – Table Scan Cross-over

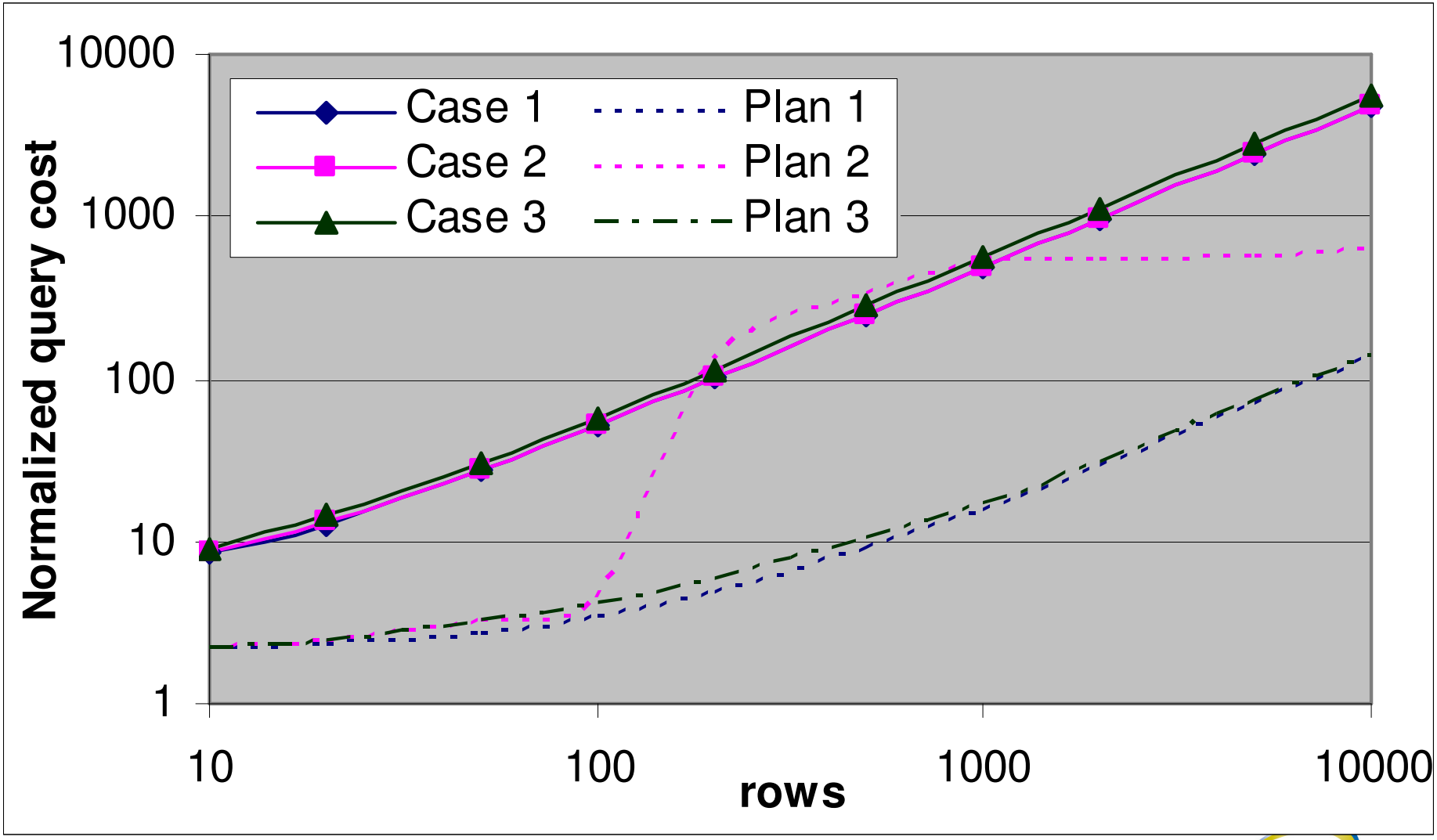


Plan Costs: ≤1GB 2x733MHz/256K Pentium III

Loop, Hash and Merge Join Costs

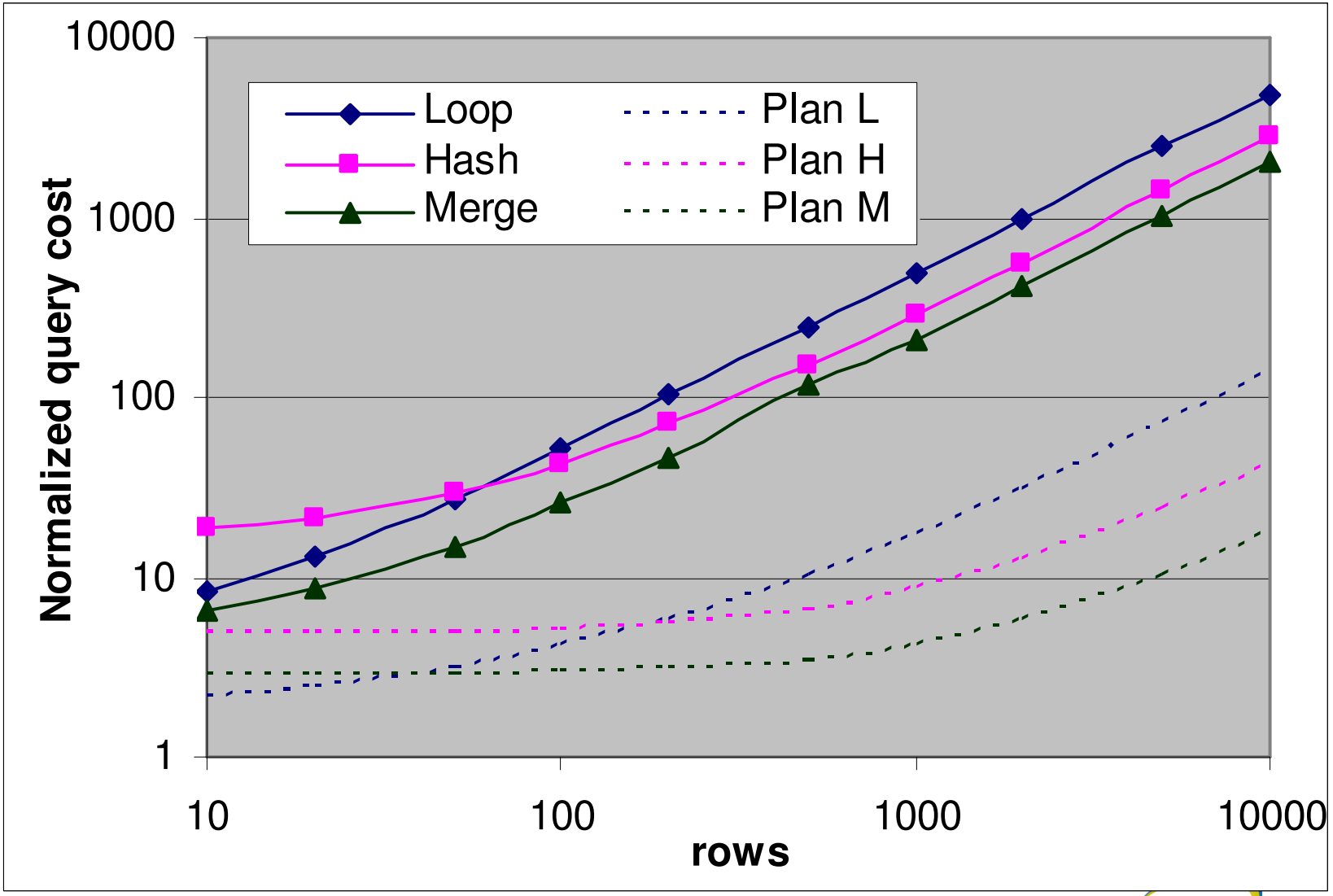
- Loop joins: Case 1, 2, 3 etc
- Hash joins: in-memory versus others
- Merge: regular versus many-to-many
- Merge join with sort operations
- Locks – page lock

Loop Joins



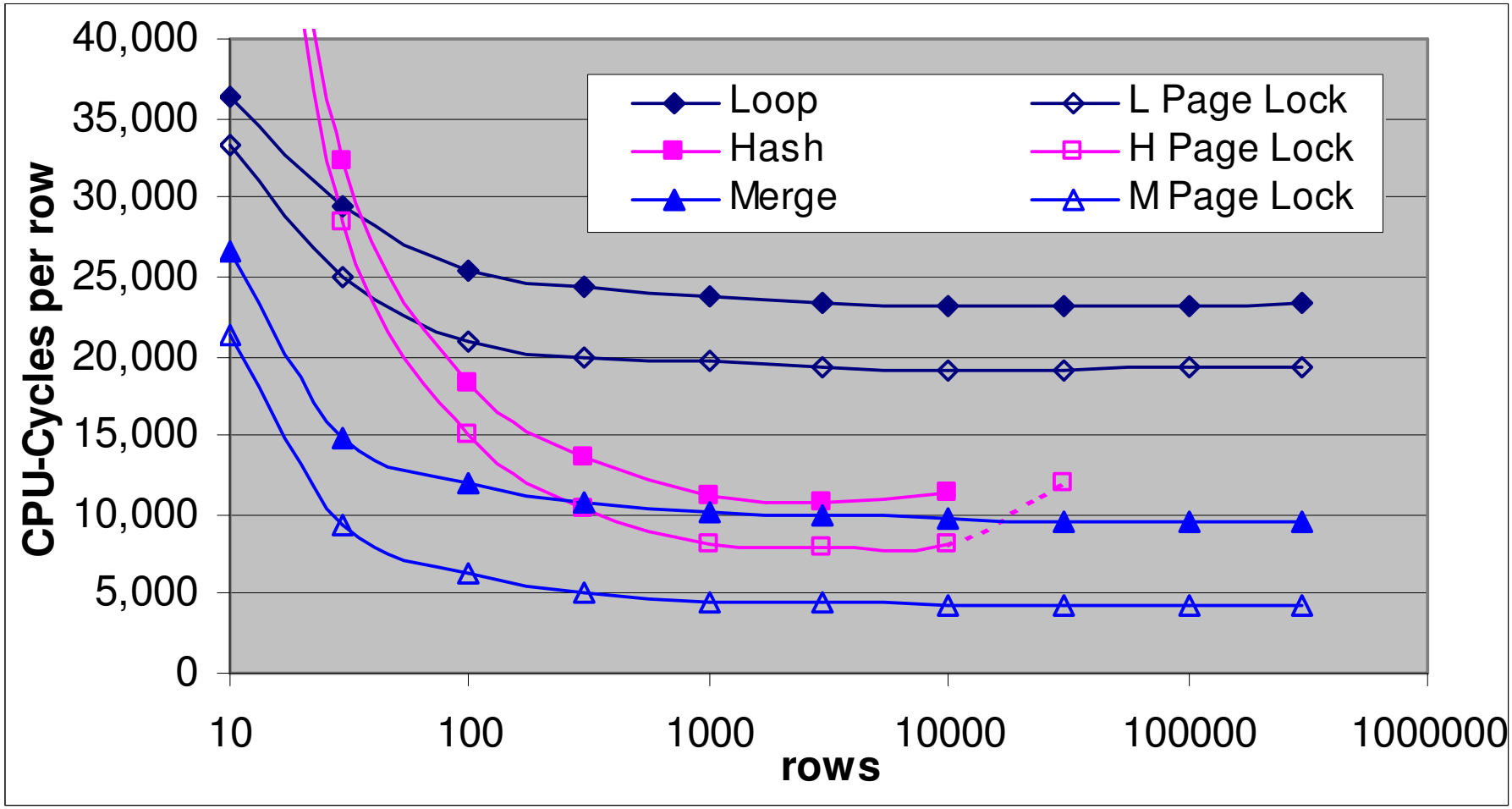
2x600MHz/256K Pentium III

Loop, Hash & Merge Joins



2x600MHz/256K Pentium III

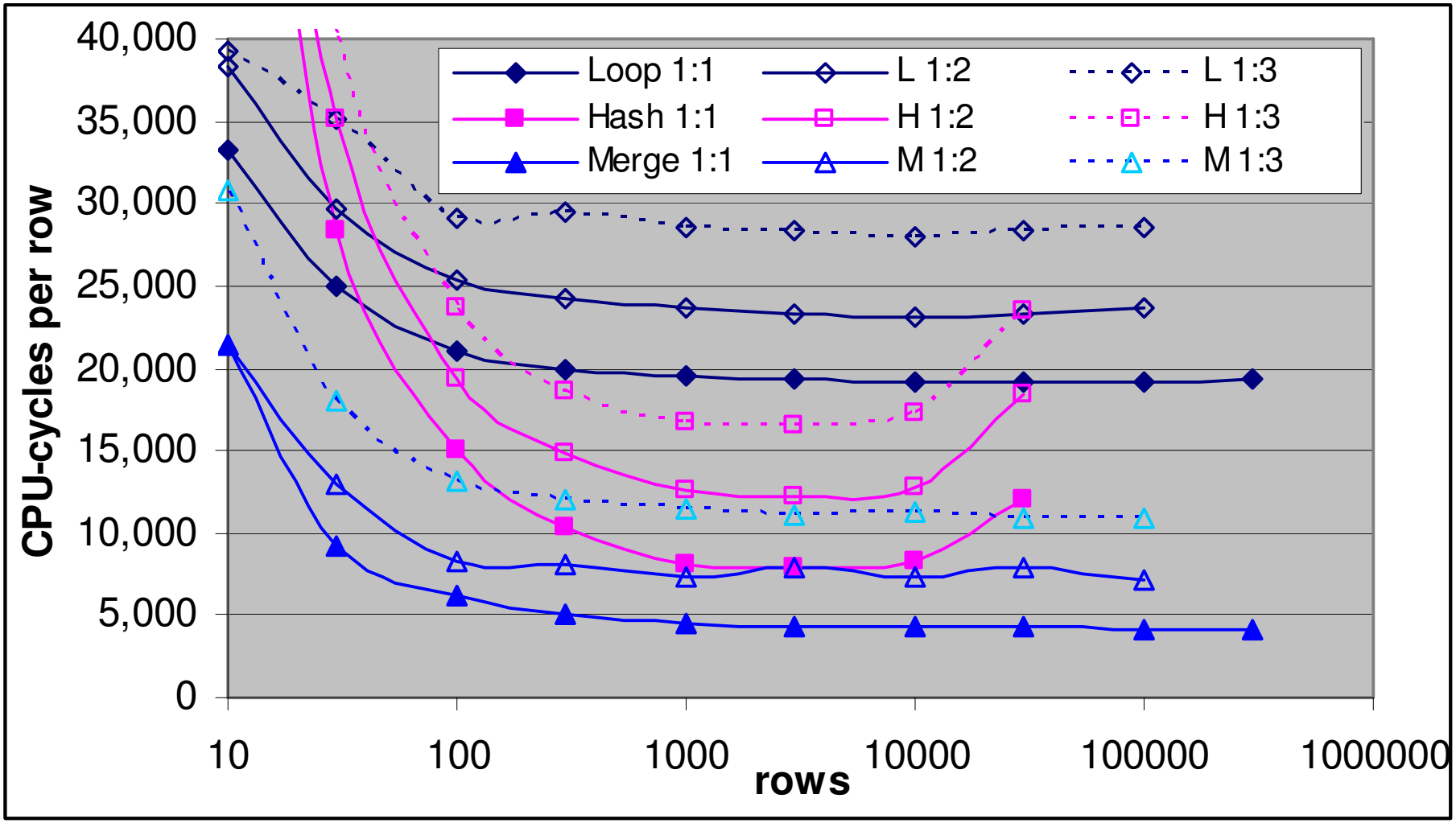
Joins – Locking Granularity



Xeon 2x2.0GHz/512K

Hash join spools to tempdb at much lower point in RPC calls versus SQL Batch

1:n Loop, Hash, Merge Joins



Xeon 2x2.0GHz/512K

Type + Component Base Cost

Costs in CPU-Cycles

Processor	PIII	PIII X	Xeon	Opt	Itanium 2		
# of CPUs	2	4	2	2	2	4	8
Frequency	~733	700	2-2.4	2.2	900M	1.5G	1.5G
Cache	256K	2M	512K	1M	1.5M	6M	6M
Aggregate	100K	40K	135K	45K	45K	35K	66K
Table Scan	60K	40K	145K	50K	35K	40K	90K
Loop Join	110K	24K	130K	50K	45K	10K	15K
Hash Join	500K	250K	620K	250K	225K	210K	320K
Merge Join	150K	54K	170K	80K	75K	55K	110K
Merge + Sort		145K	320K		140K		230K
Many-to-Many		250K	550K		200K		320K

Big cache lowers base cost?

Cost per Row

Costs in CPU-Cycles

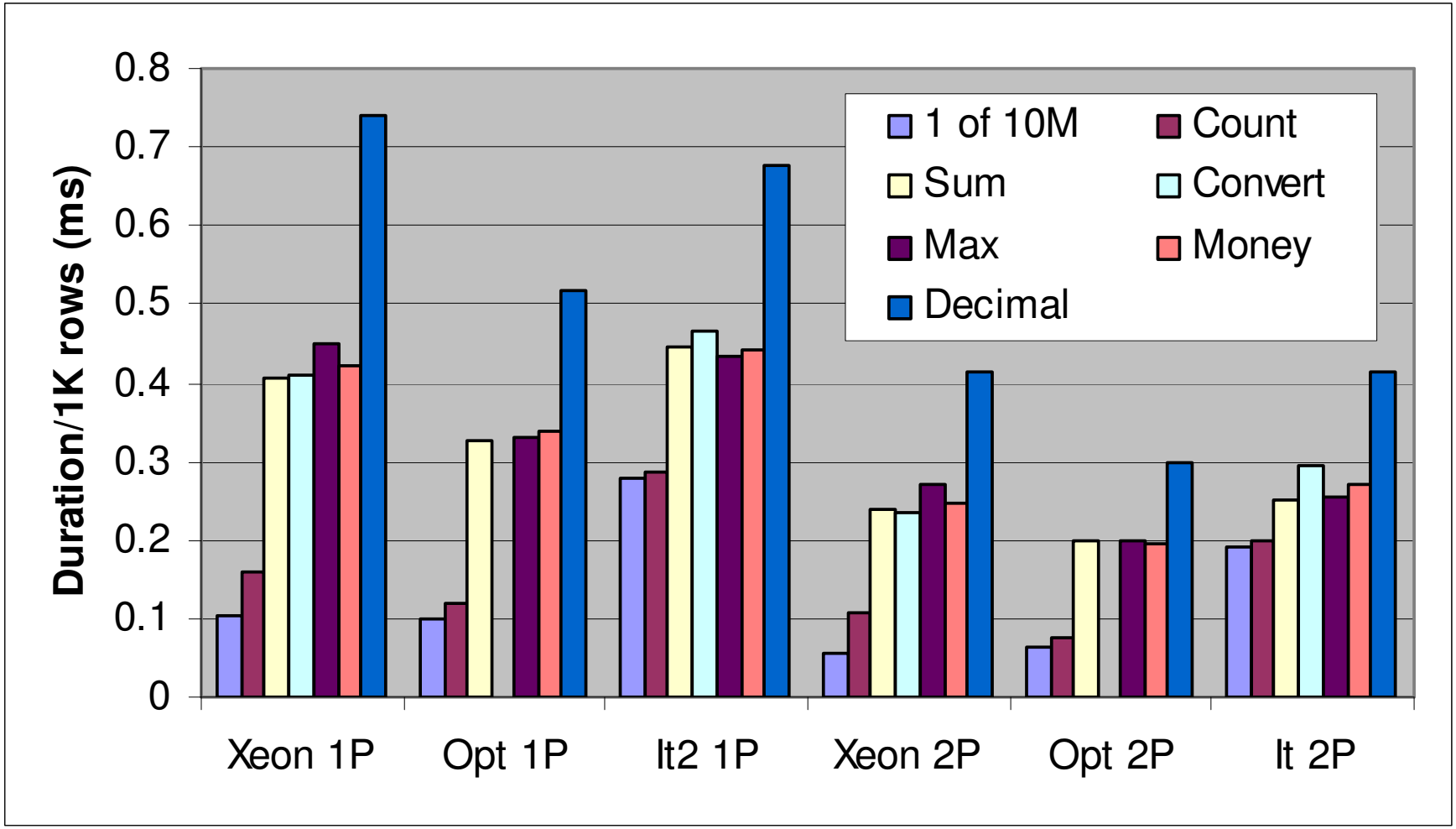
Processor	PIII	Xeon	Opt	Itanium 2 2P, 4P, 8P		
Index Seek	1.3K	1.8K	1.1K	1.0K	1.0K	1.0K
Bookmark LU (CL)	16K	20K	12K	11K	16K	27K
BL (CL) page lock	11K	15K	8K	8K	13K	18K
Bookmark LU (Heap)	11K	14K	9K	10K	11K	16K
BL (Heap) page lock	7K	9K	5K	6K	8K	8K
Loop Join	16K	25K	13K	11K	16K	23K
Loop Join, page lock	13K	20K	11.5K	9K	14K	21K
Hash Join	8.5K	11K	7.5K	7K	7K	7K
Hash Join, page lock	6.5K	8K	6.0K	5K	5K	5K
Merge Join	6.5K	10K	5.0K	6K	6K	6K
Merge Join, page lock	3.0K	4K	2.5K	3K	3K	3K
Merge+Sort	7.5K	9K		6K		7K
Many-To-Many PL	32K	40K		18K		31K

Peak Theoretical Performance

Rows or Pages/sec

Processor	PIII	Xeon	Opt	Itanium 2 2P, 4P, 8P		
RPC	9K	19K	31K	19K	21K	34K
Index Seek 1 row	46K	120K	200K	120K	200K	300K
Table Scan (pages)	61K	192K	200K	176K	352K	521K
Index Seek	1,100K	2,600K	4,000K	3,000K	6,000K	12M
Bookmark LU (CL)	91K	240K	366K	250K	375K	444K
BL (CL) page lock	133K	320K	550K	333K	461K	666K
Bookmark LU (Heap)	133K	342K	488K	300K	545K	750K
BL (Heap) page lock	209K	554K	880K	500K	750K	1,500K
Loop Join	92K	225K	338K	230K	375K	521K
Loop Join, page lock	113K	257K	382K	272K	428K	571K
Hash Join	172K	436K	586K	375K	857K	1,700K
Hash Join, page lock	225K	600K	733K	500K	1,200K	2,400K
Merge Join	225K	480K	880K	500K	1,000K	2,000K
Merge Join, page lock	489K	1,200K	1,700K	1,000K	2,000K	4,000K

Index Seek by Aggregates



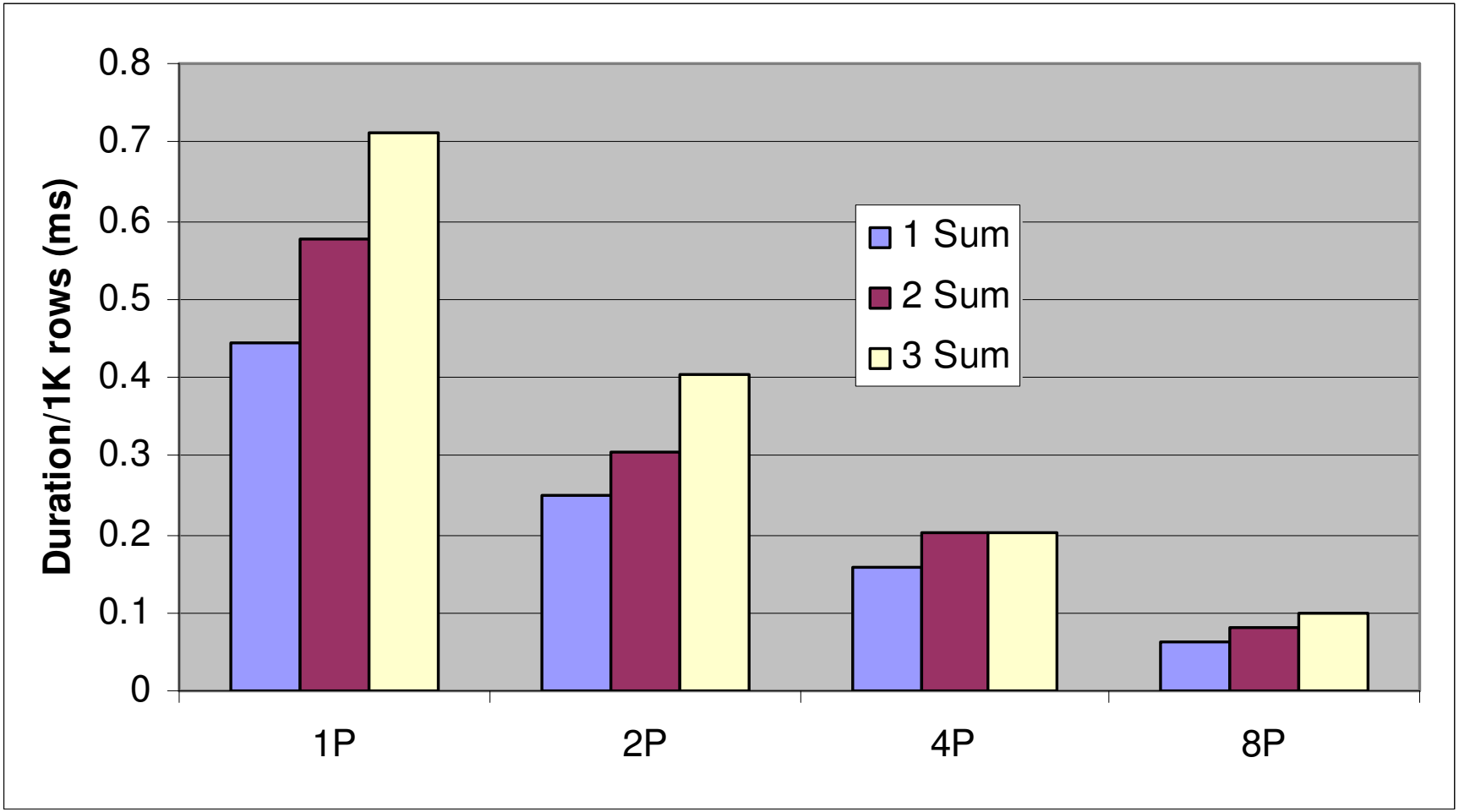
SELECT COUNT(*)

SELECT COUNT(*), SUM(int)

SELECT COUNT(*), SUM(Money)

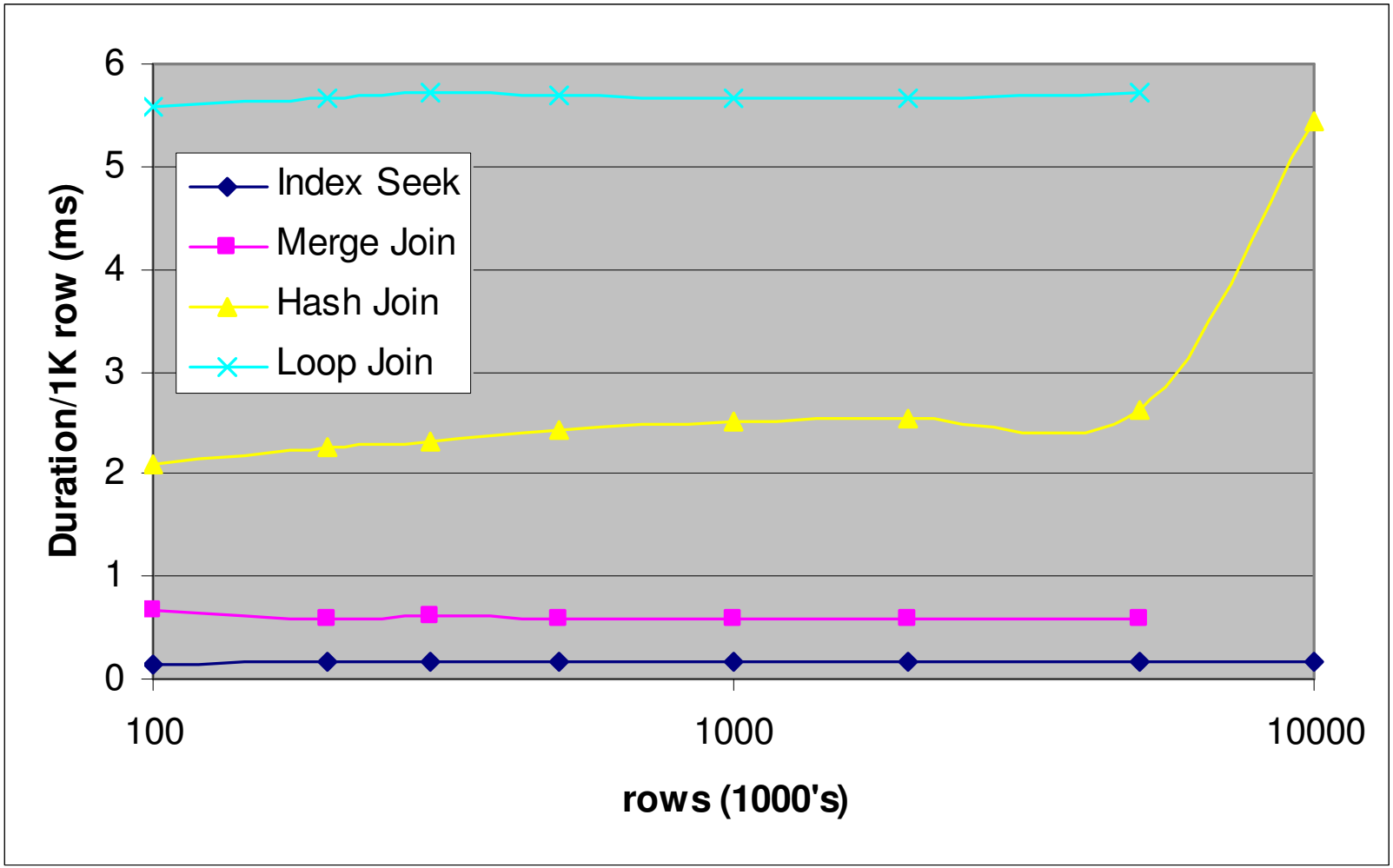
SELECT COUNT(*), CONVERT int to bigint, etc

Aggregates – Itanium 2



SELECT COUNT(*), SUM(int)
SELECT COUNT(*), SUM(int), SUM(int)
SELECT COUNT(*), SUM(int), SUM(int), SUM(int)

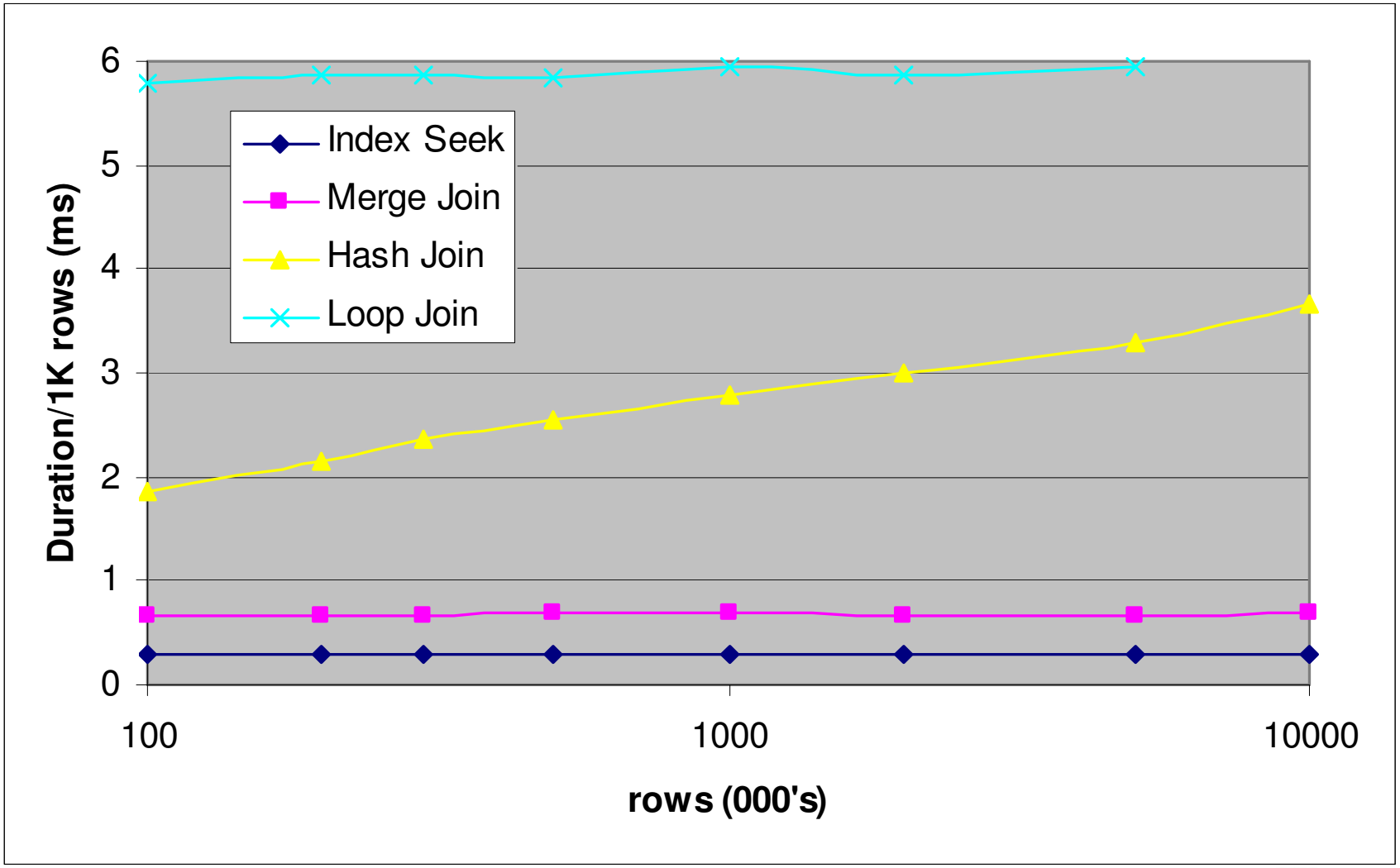
Hash Join Linearity on 32-bit



Hash join cost per row is somewhat linear up to ~5M rows
Duration then jumps due to disk IO

HP WORLD 2004
Solutions and Technology Conference & Expo

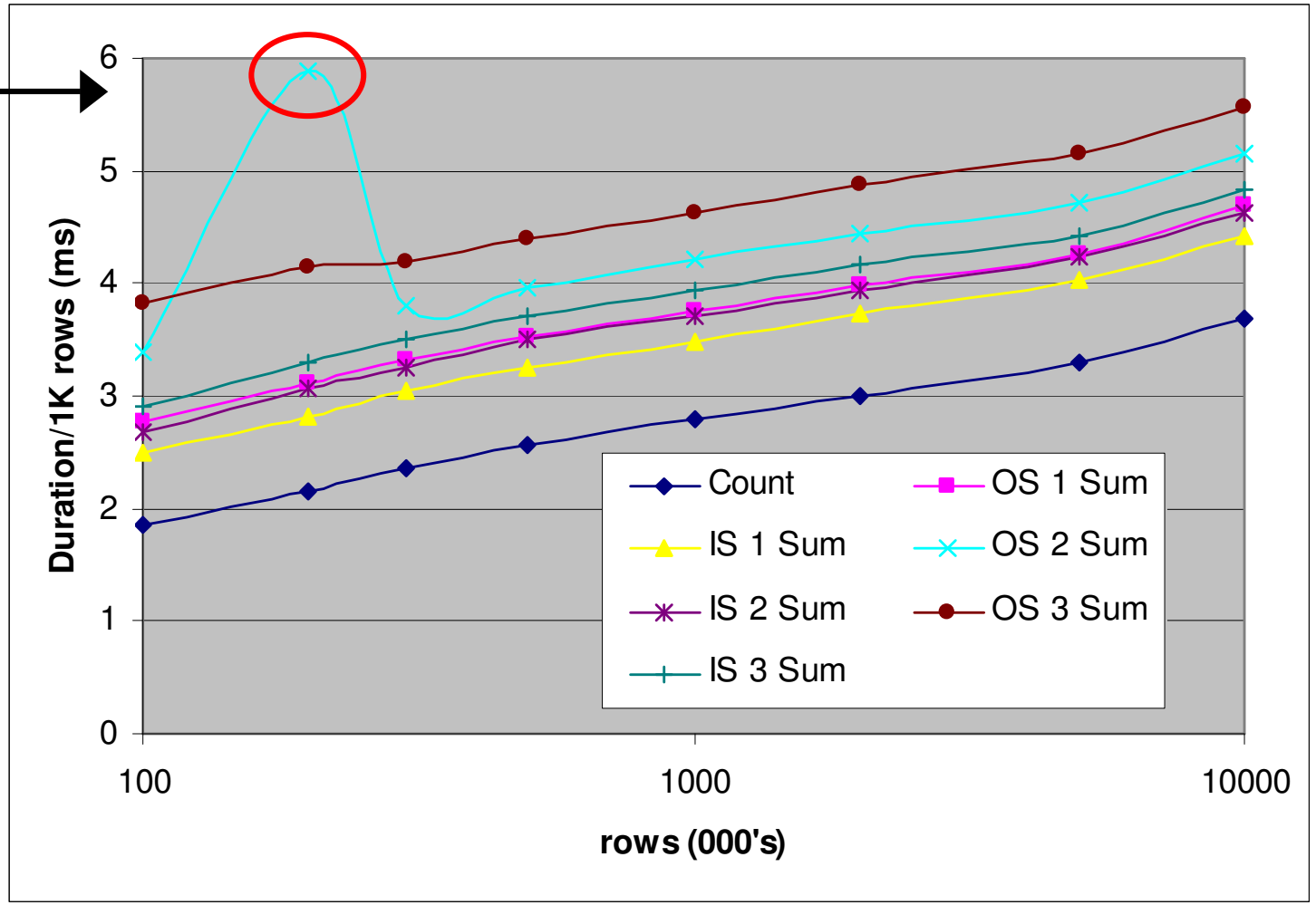
Hash Join Linearity on Itanium 2



Loop & Merge join cost per row independent of row count
Hash join cost per row is not (no spooling to temp)

Hash Join – row size

Occasional performance anomalies



Optimizer cost: depends on OS size, not IS size
Actual cost: depends on # of columns and OS/IS

IUD Cost Structure

*Use Windows NT fibers on

	2xXeon*	Notes
RPC cost	**240,000**	**Higher for threads, owner m/m**
Type Cost	**130,000**	**once per procedure**
IUD Base	**170,000**	**once per IUD statement**
Single row IUD	**300,000**	**Range: 200,000-400,000**
Multi-row Insert Cost per row	**90,000**	**cost per additional row**

INSERT, UPDATE & DELETE cost structure very similar
Multi-row UPDATE & DELETE not fully investigated

INSERT Cost Structure

Index and Foreign Key not fully explored

Early measurements:

50-70,000 per additional index
50-70,000 per foreign key

Assumes inserts into “sequential” segments
Cluster on Identity or Cluster Key + Identity
Cluster on row GUID can cause substantial disk loading!

Database Design for Performance

- Round-trip minimization – RPC cost
- Row count management – Cost per row
  - Indexes – isolate queried rows to a limited number of adjacent pages quickly, not most selective columns 1st
- Design for low cost operations
  - Covered index instead of bookmark lookups
  - Merge joins, Indexed views
  - Avoid excessive logic
- NOLOCK on non-transactional data

Statistics

- Accuracy & Relevance
  - More than keeping statistics up to date
  - Data queried needs to reflect data in table
  - Avoid populating database with test data having different distribution than live data

Performance Issues

- Index + Bookmark Lookup vs. Table Scan
  - Query optimizer switches to table scan too soon for in-memory, too late for disk bound data
- Row count plan versus actual cost issues
  - May be related to WHERE clause conditions
- Lock hints
- Merge and Hash joins vs. Loop joins
- Fixed costs favor consolidation
  - Both in RPC and queries

Summary

- Query cost structure
- Fixed costs identified
  - Costs applied once per RPC
- Component operations examined
  - Base cost and cost per row or page
- Lock hints – Row, Page, Table, No Lock

Additional Information

www.sql-server-performance.com/joe_chang.asp

SQL Server Quantitative Performance Analysis
Server System Architecture
Processor Performance
Direct Connect Gigabit Networking
Parallel Execution Plans
Large Data Operations
Transferring Statistics
Backup Performance Analysis with SQL LiteSpeed (soon)

jchang6@yahoo.com

HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP