Vertica's Design: Basics, Successes, and Failures

Chuck Bear

CIDR 2015 – January 5, 2015

1. Vertica Basics: Storage Format

Design Goals

- SQL (for the ecosystem and knowledge pool)
- Clusters of commodity hardware (for cost)
  - Linux, x86, Ethernet
- Software-only solution (for flexibility)
  - Special purpose hardware has poor track record in databases
- Shared Nothing MPP
  - Cheaper, but puts more complexity in the software
- Analytics: Run large queries many times faster than a legacy DB, load as fast, but feel free to snarl and growl at small UPDATEs and DELETEs
- Work smart, and work hard.
  - Robust algorithms, query optimizer, vectorize, JIT, etc.

Start from how data is stored on disk...

- **SELECT SUM(volume) FROM trades WHERE symbol = 'HPQ' AND date = '5/13/2011'**

SYMBOL	DATE	TIME	PRICE	VOLUME	ETC
...	...	...	...	...	...
HPQ	*05/13/11*	*01:02:02 PM*	*40.01*	*100*	...
IBM	05/13/11	01:02:03 PM	171.22	10	...
AAPL	05/13/11	01:02:03 PM	338.02	5	...
GOOG	05/13/11	01:02:04 PM	524.03	150	...
HPQ	*05/13/11*	*01:02:05 PM*	*39.97*	*40*	...
AAPL	05/13/11	01:02:07 PM	338.02	20	...
GOOG	05/13/11	01:02:07 PM	524.02	40	...
...	...	...	...	...	...

Sorted Data

- **Sort by Symbol, Date, and Time**

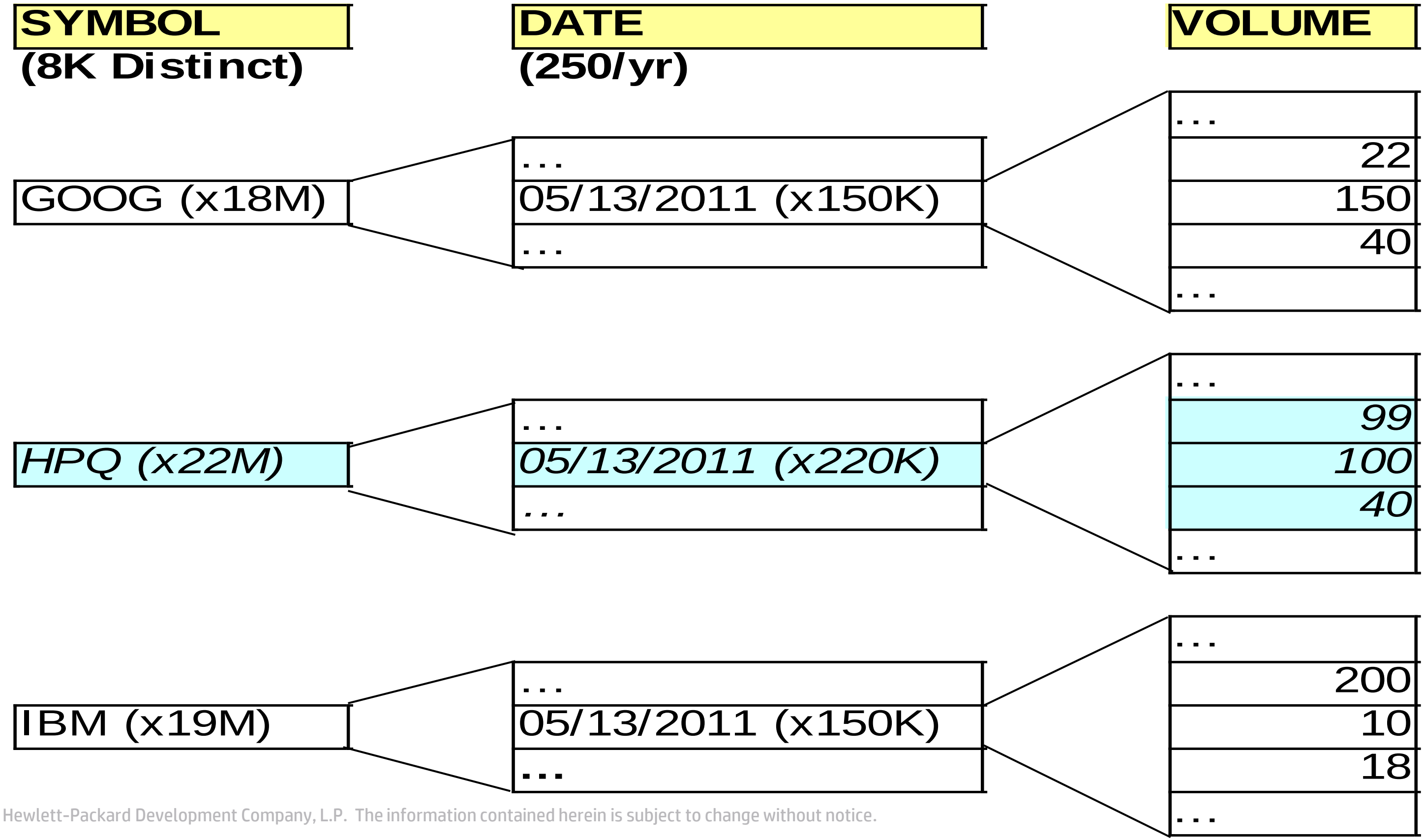
SYMBOL	DATE	TIME	PRICE	VOLUME	ETC
...	...	...	...	...	...
AAPL	05/13/11	01:02:07 PM	338.02	20	...
AAPL	05/13/11	01:02:03 PM	338.02	5	...
...	...	...	...	...	...
GOOG	05/13/11	01:02:04 PM	524.03	150	...
GOOG	05/13/11	01:02:07 PM	524.02	40	...
...	...	...	...	...	...
HPQ	*05/13/11*	*01:02:02 PM*	*40.01*	*100*	...
HPQ	*05/13/11*	*01:02:05 PM*	*39.97*	*40*	...
...	...	...	...	...	...
IBM	05/13/11	01:02:03 PM	171.22	10	...
...	...	...	...	...	...

Column Files

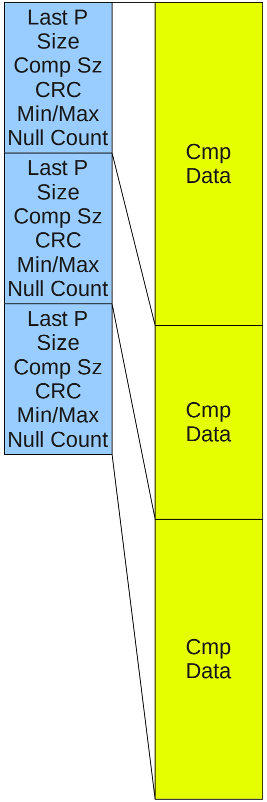
- Split into columns

SYMBOL	DATE	TIME	PRICE	VOLUME	ETC
...	...	...	...	...	...
AAPL	05/13/11	01:02:07 PM	338.02	20	...
AAPL	05/13/11	01:02:03 PM	338.02	5	...
...	...	...	...	...	...
GOOG	05/13/11	01:02:04 PM	524.03	150	...
GOOG	05/13/11	01:02:07 PM	524.02	40	...
...	...	...	...	...	...
HPQ	*05/13/11*	*01:02:02 PM*	*40.01*	*100*	...
HPQ	*05/13/11*	*01:02:05 PM*	*39.97*	*40*	...
...	...	...	...	...	...
IBM	05/13/11	01:02:03 PM	171.22	10	...
...	...	...	...	...	...

Compression + RLE

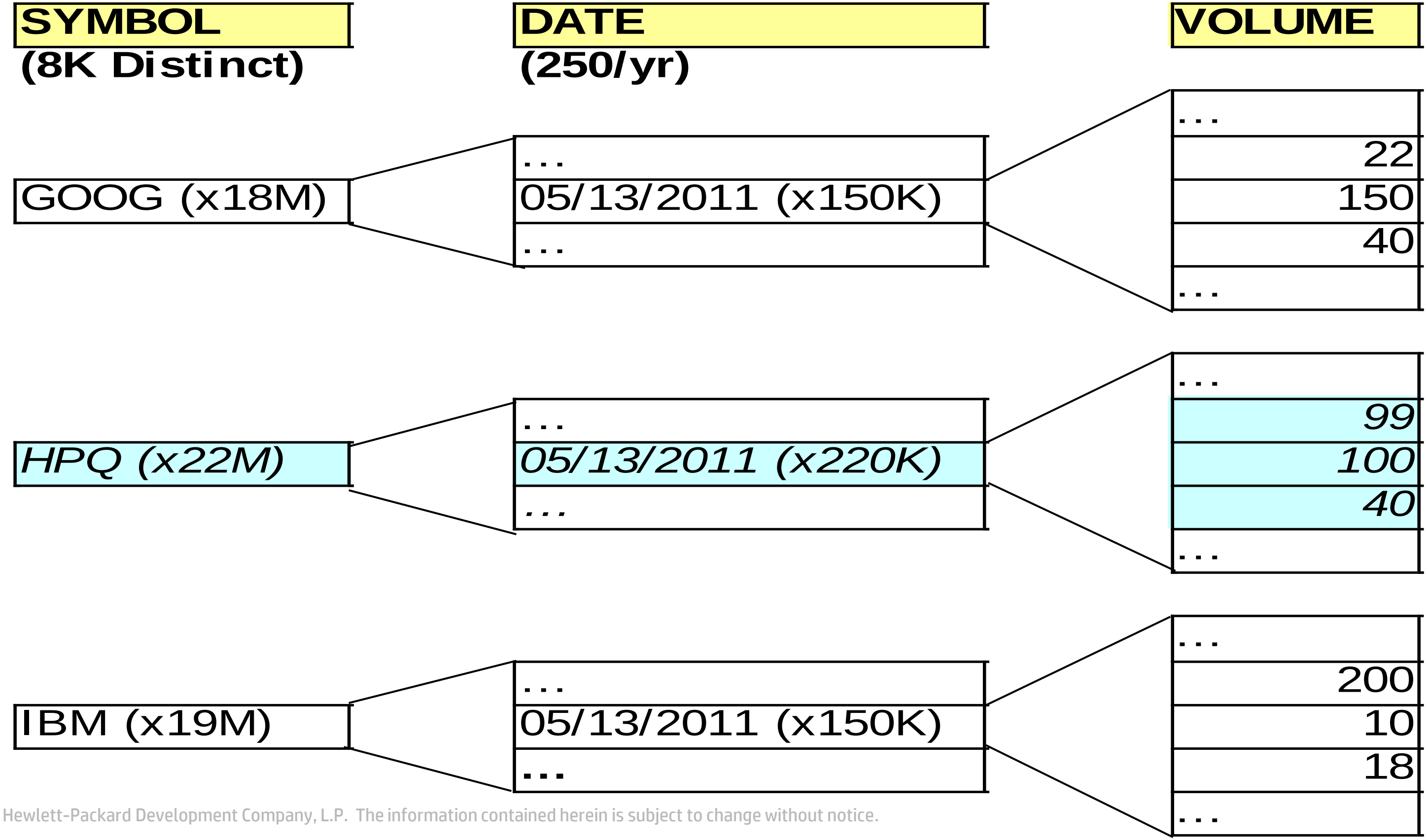


Position Index (NOT Row ID)



2. Vertica Basics: Updates & Deletes

Q: How do you update this?

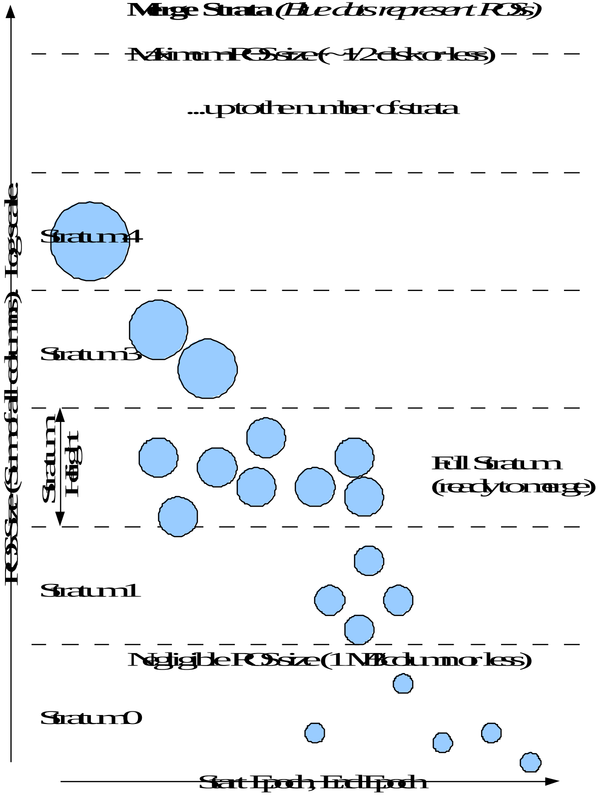


A: You Do Not!

- Multiple sets of sorted files loaded
  - Or keep things in memory for a while
- Update is INSERT+DELETE
- Delete is just a mark – nice sorted list of positions

It’ll Get Dirty….

So you need to compaction, or whatever. We call ours the tuple mover.



How Do You Judge a Tuple Mover?

Not by glamour, etc.

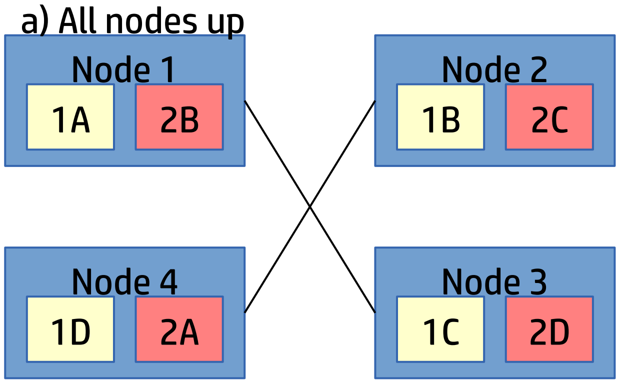
- Magical: no problems, no backlogs, no errors
- Latency and freshness: How much batching is needed?
- Sustained load rate (consider machine capacity + retention interval)

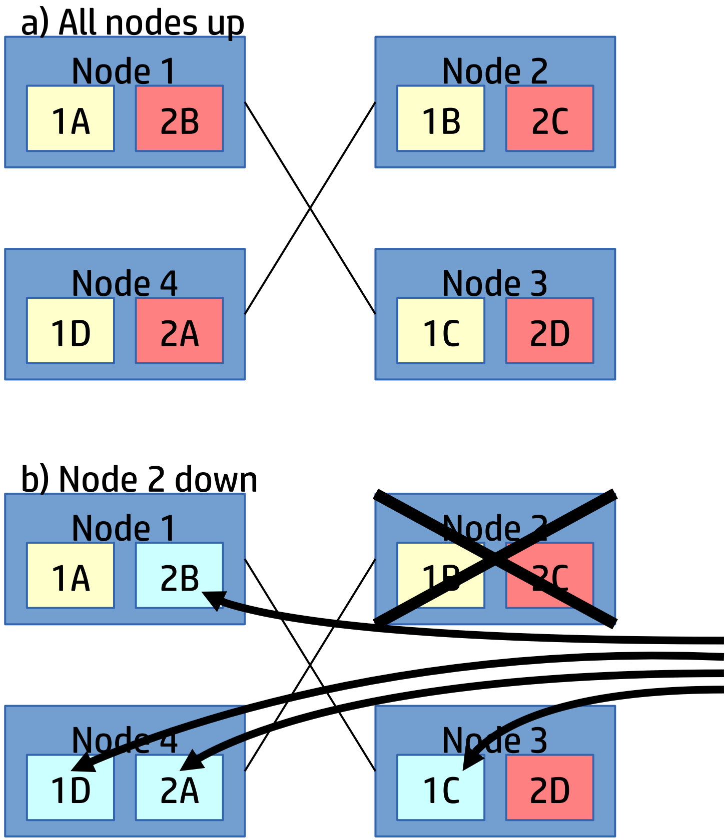
- Efficiency will be required

3. Vertica Basics: Transactions & Recovery

Transactions

- **Vertica offers full ACID (just at low TPS)**
- **Queries take a snapshot of the relevant list of files, and need no locks at READ COMMITTED isolation**
  - Tuple Mover (etc.) doesn't interfere
- **Loads do not conflict with each other**
  - COMMIT – keep the new files
  - ROLLBACK – discard them
- **Table level locks for SERIALIZABLE**
- **All Operations are On-Line**
- **Database is essentially its own undo / redo log**
  - Recovery can be as simple as file copies


a) All nodes up
Node 1
1A
2B
Node 2
1B
2C
Node 4
1D
2A
Node 3
1C
2D



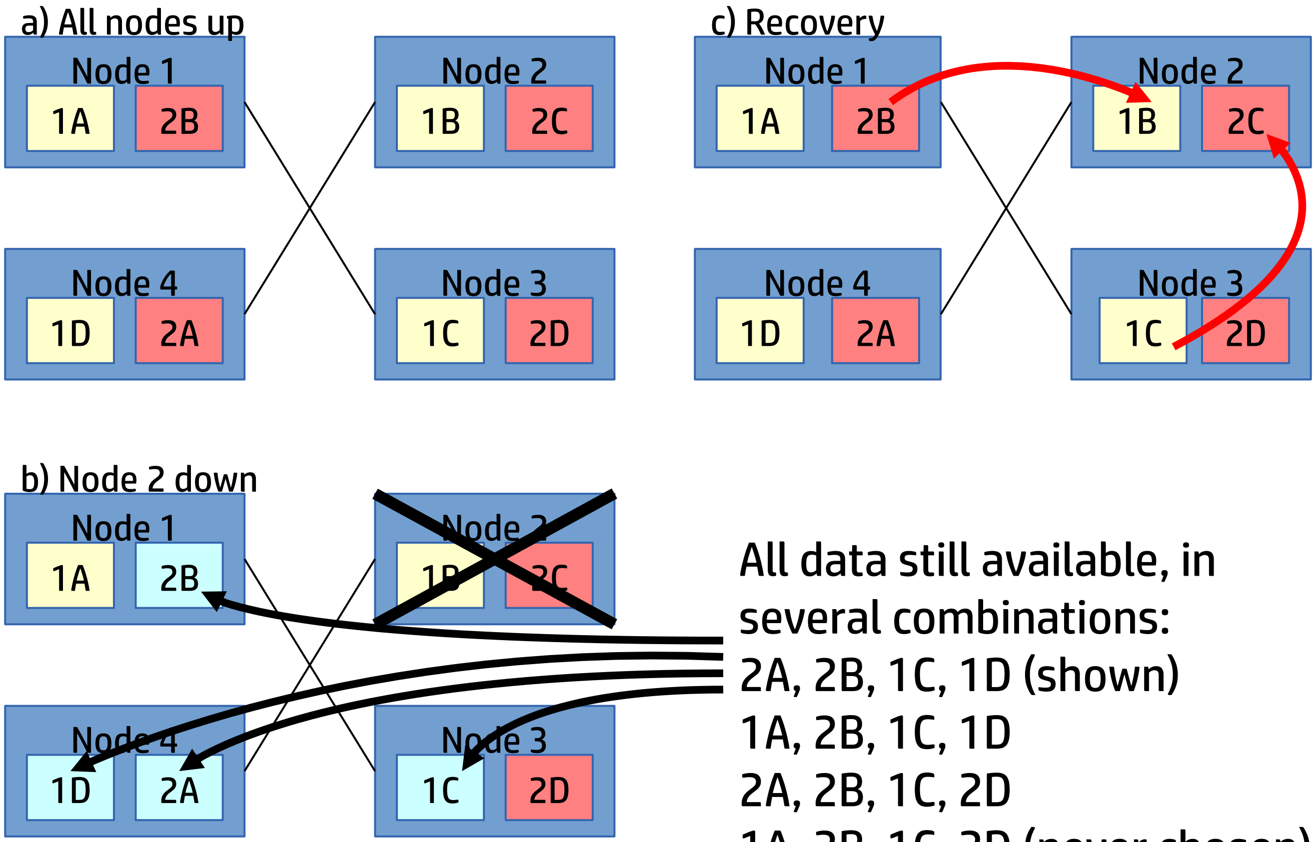
All data still available, in several combinations:

2A, 2B, 1C, 1D (shown)

1A, 2B, 1C, 1D

2A, 2B, 1C, 2D

1A, 2B, 1C, 2D (never chosen)



All data still available, in several combinations:

2A, 2B, 1C, 1D (shown)

1A, 2B, 1C, 1D

2A, 2B, 1C, 2D

1A, 2B, 1C, 2D (never chosen)

4. Mistake: Execution Engine Design

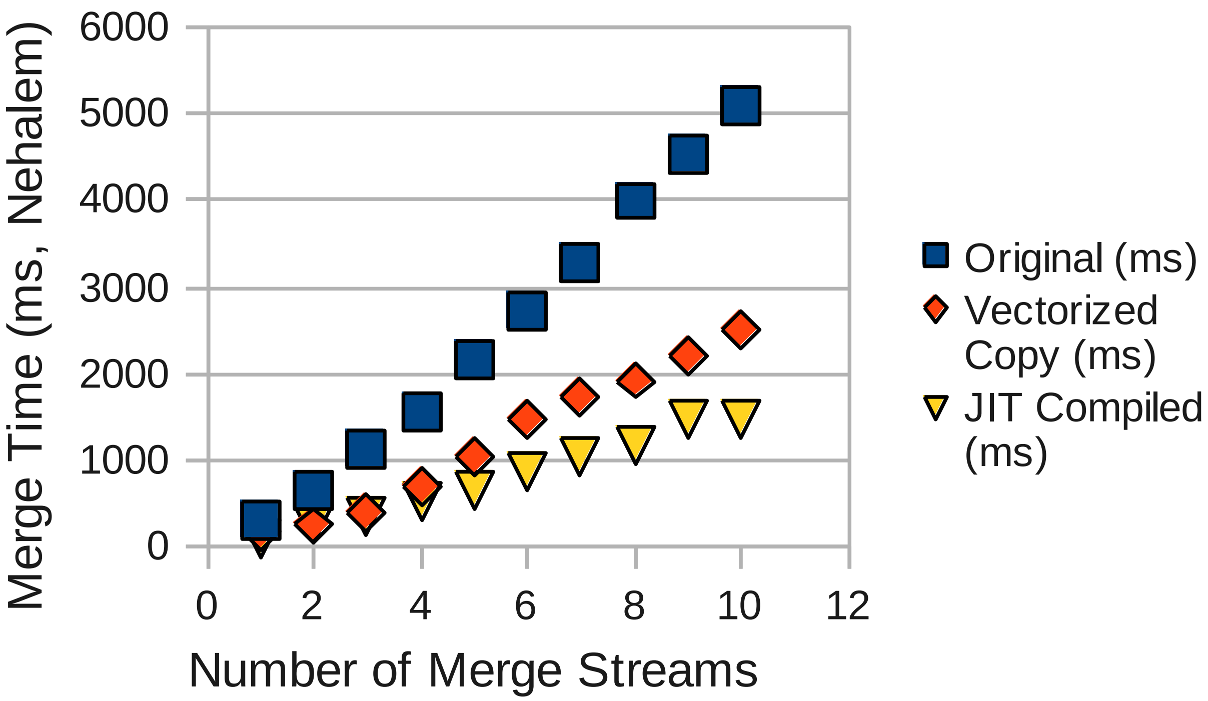
Simple Design

- Use iterators
  - open
  - getNext
  - close

- If there's trouble, use a temp relation

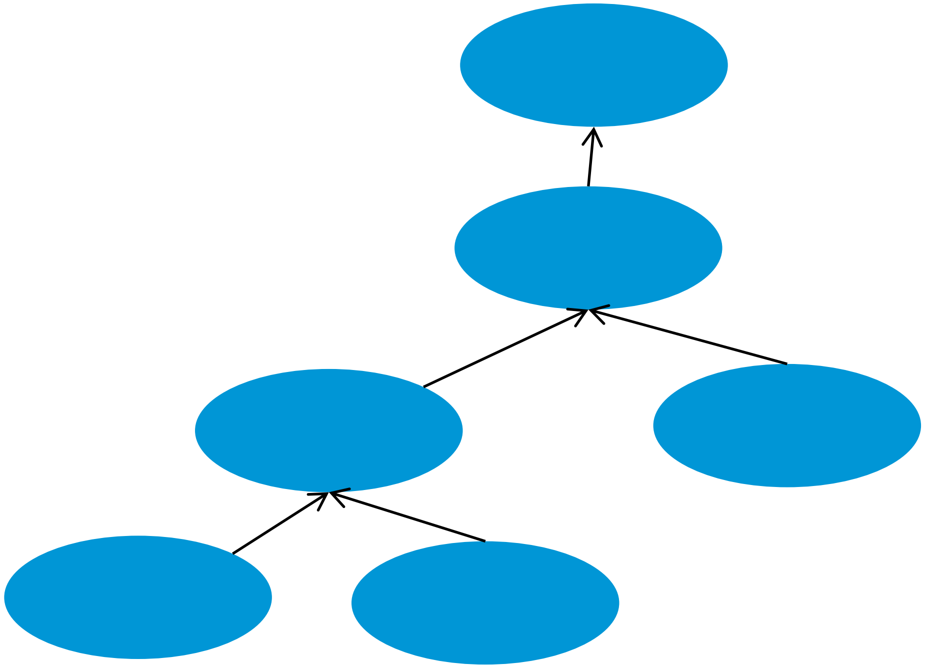
Too Slow!

(You have to vectorize. And do JIT compiling.)

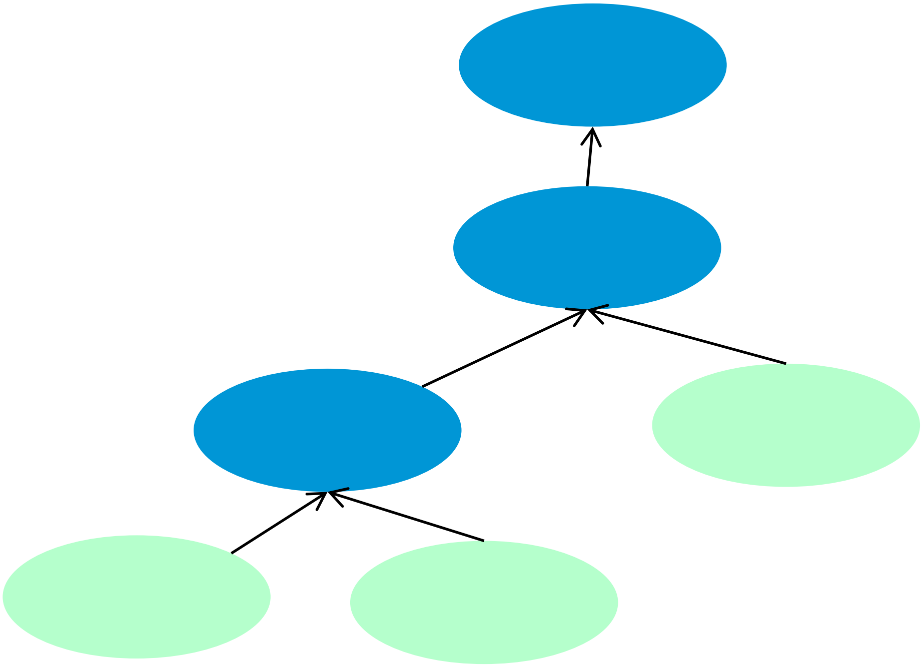


“Push Model” DAG Executor

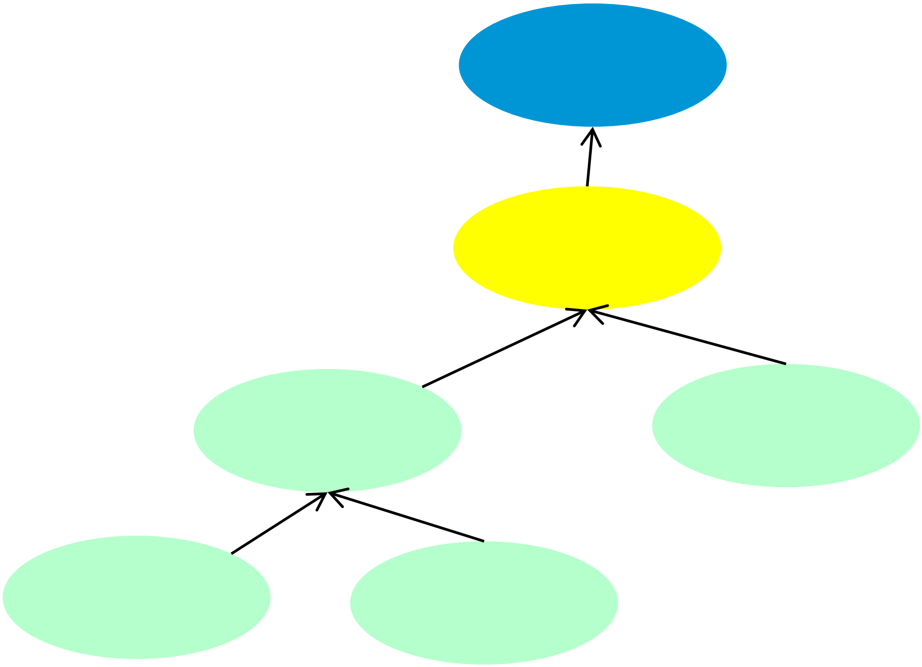
You might even get parallelism for free ☺



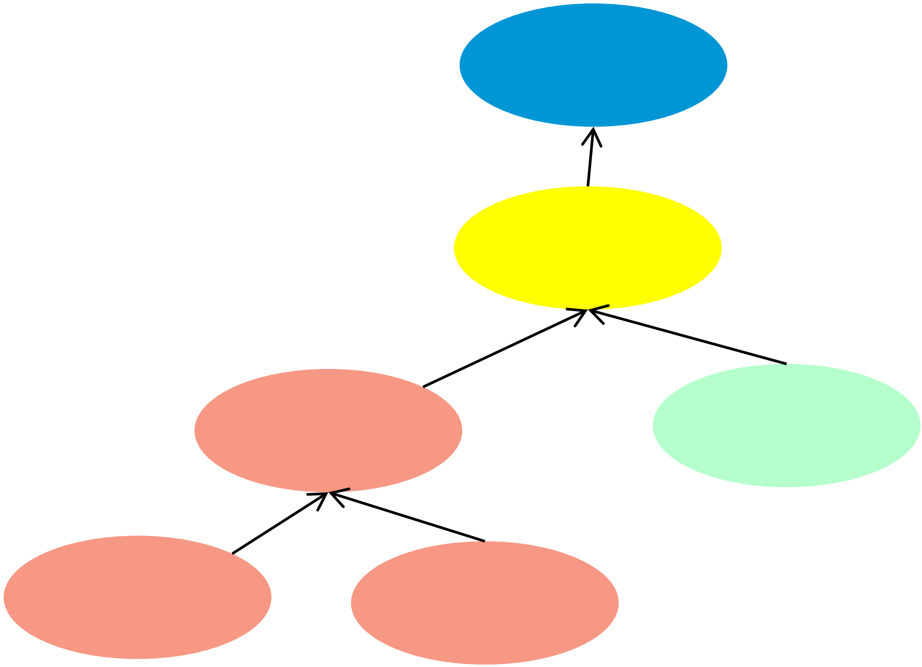
“Push Model” DAG Executor



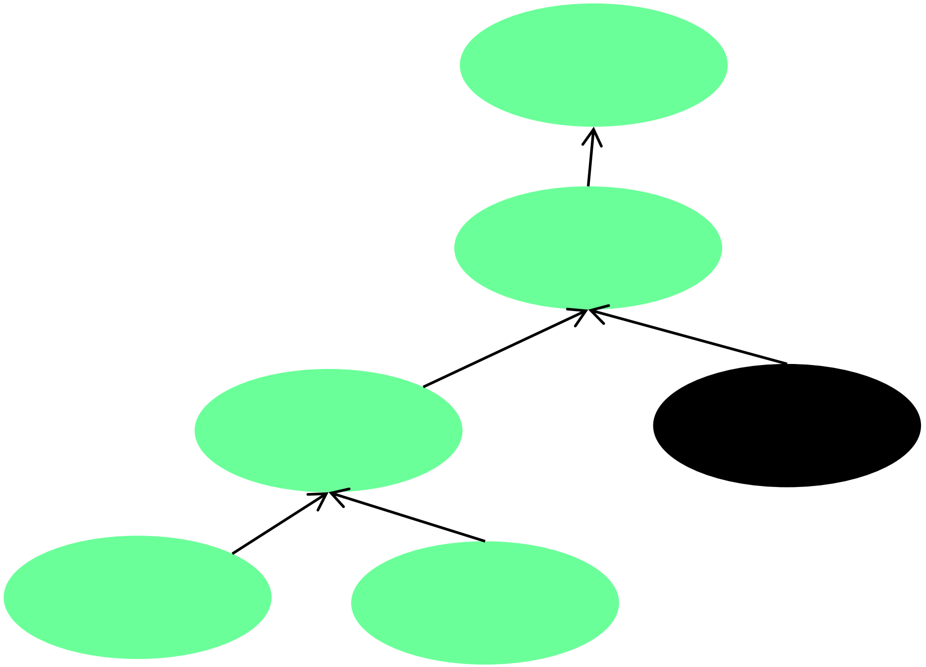
“Push Model” DAG Executor



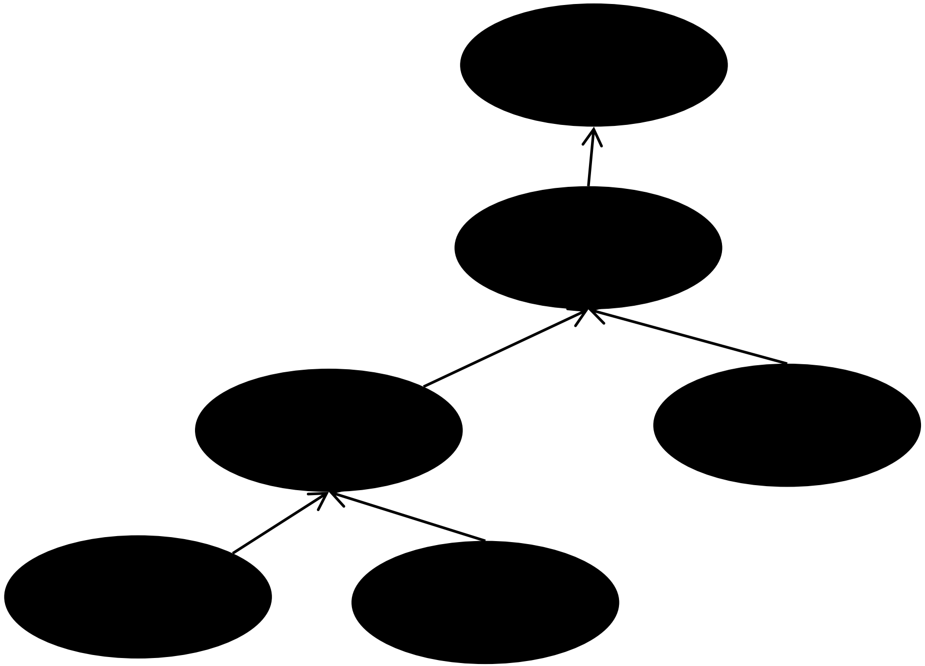
“Push Model” DAG Executor



"Push Model" DAG Executor



"Push Model" DAG Executor



Problems with DAG Execution

- Free-for-all
  - And that parallelism thing didn't pan out after all
- Resource usage: could do better
- Diamond problem
- Need to give clues to upstream operations
  - Imagine subqueries?

End Result?

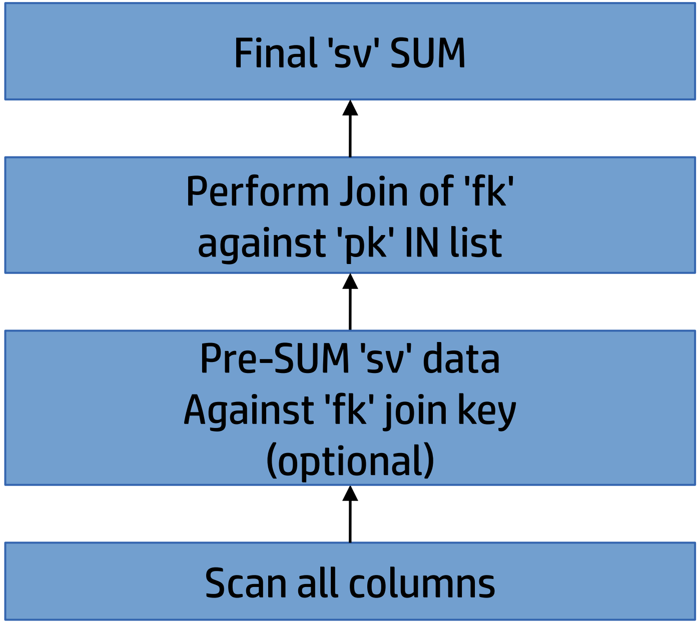
We threw it away, and went back to the "pull" model

- Block iterators
  - open
  - getNextBatch (w/ optimizations to avoid tuple copies)
  - close
  - Also, send information back upstream
- When it gets tricky, use coroutines or other tactics
- We still push data when there are multiple targets
  - Such as loading multiple projections, UPDATEs, etc.

5. Evolution of Joins in Vertica: The Good, Bad, and Ugly

Early Materialized Joins

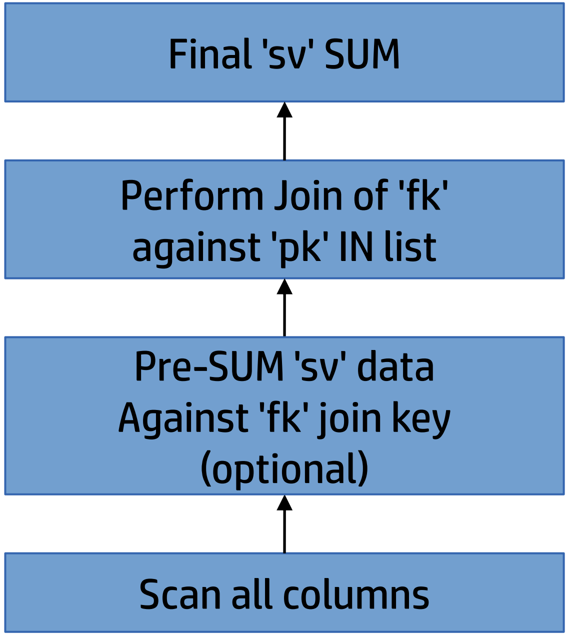
a) No SIPS, EMJ



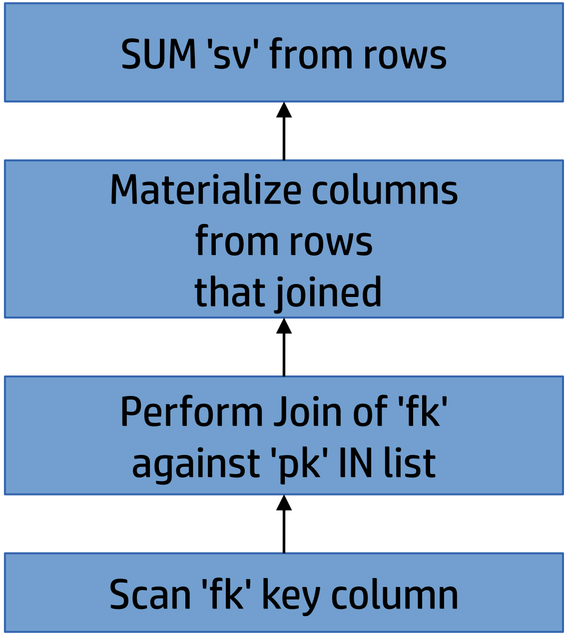
SELECT SUM(sv) FROM fact WHERE fk IN (SELECT pk FROM d)

Late Materialized Joins

a) No SIPS, EMJ

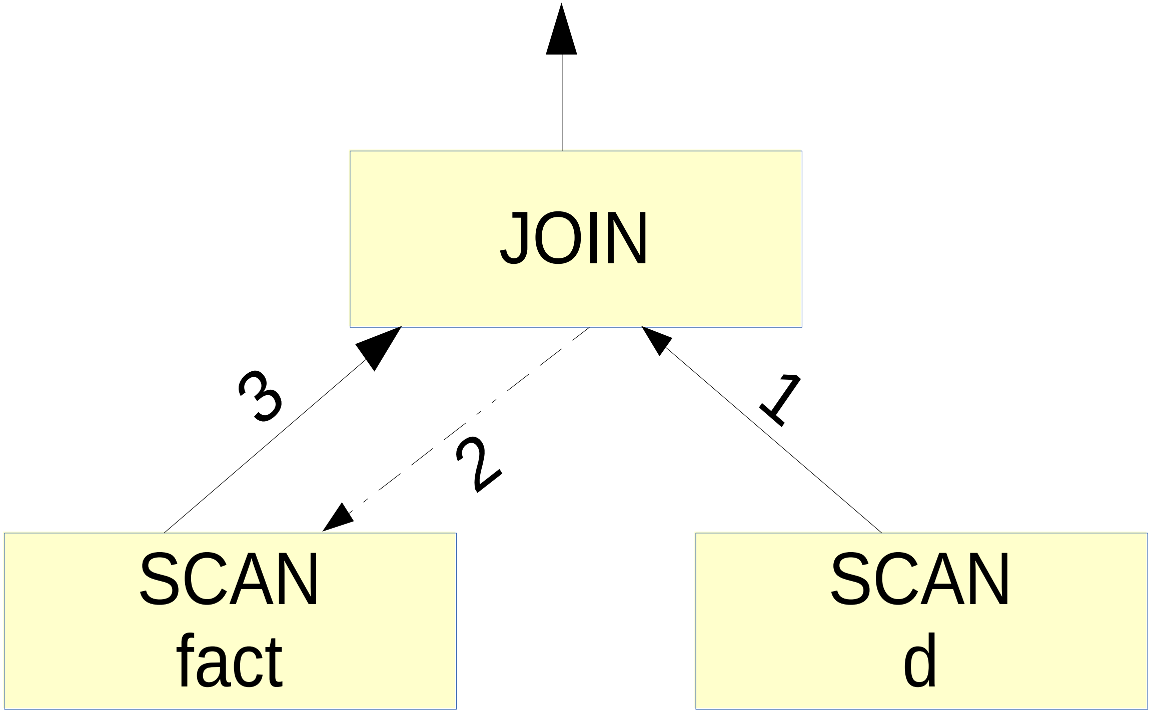


b) No SIPS, LMJ

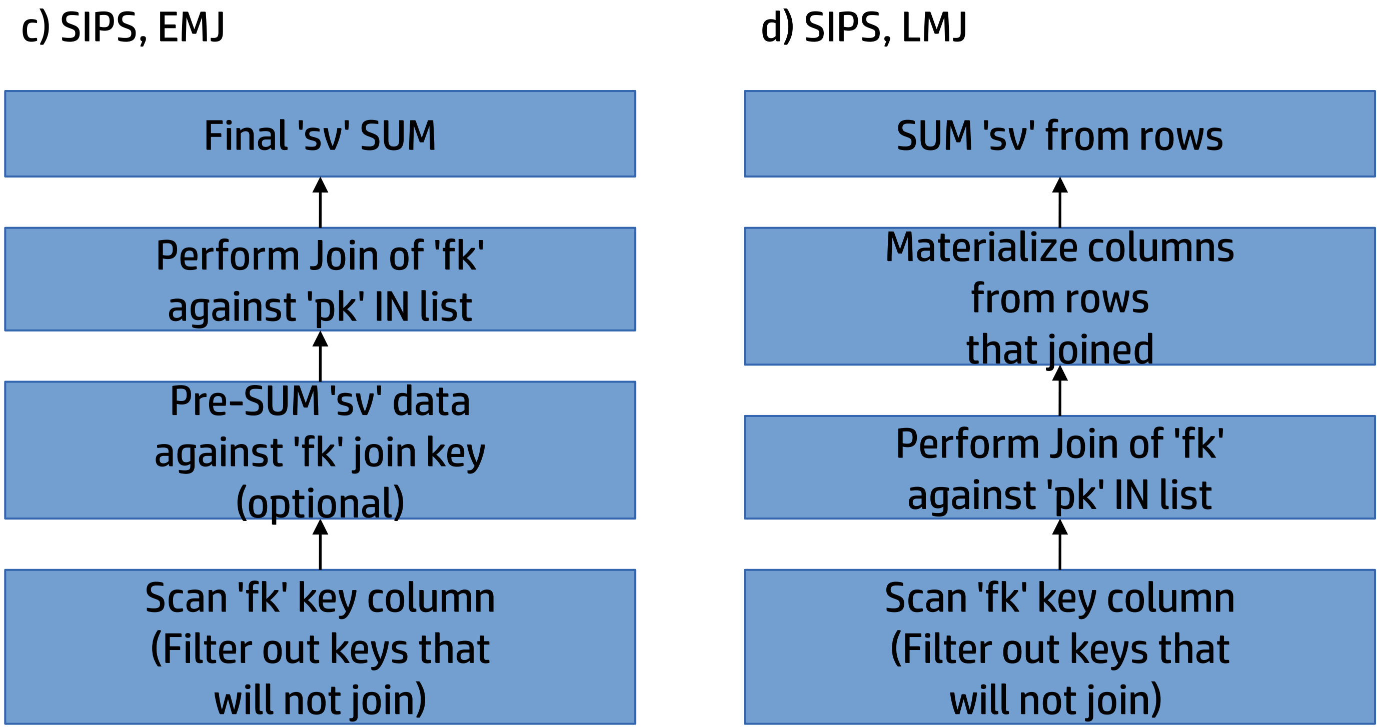


SELECT SUM(sv) FROM fact WHERE fk IN (SELECT pk FROM d)

Sideways Information Passing (SIPS)



Late Materialized Joins

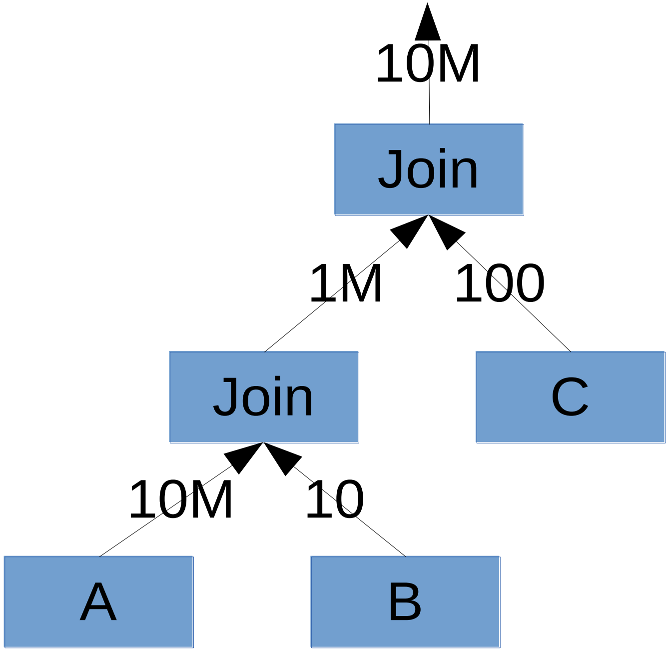


SELECT SUM(sv) FROM fact WHERE fk IN (SELECT pk FROM d)

Outcome?

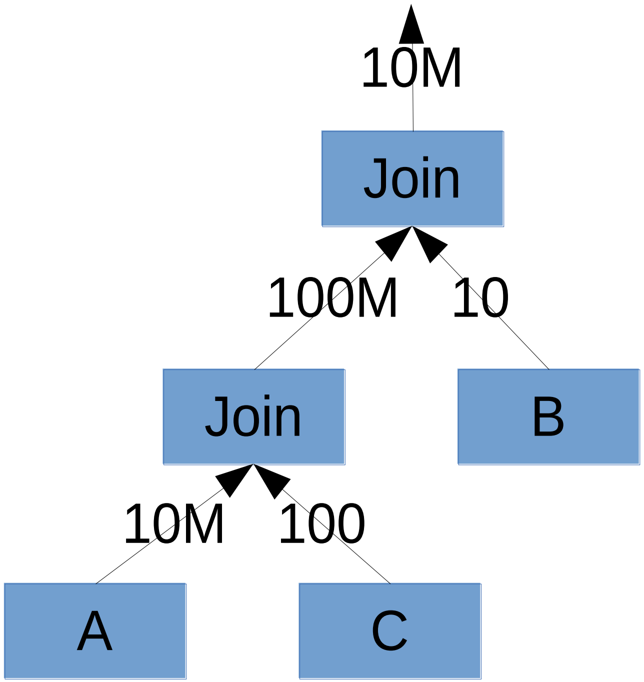
Selectivity	Neither Feature	LMJ only	SIPS only	SIPS+LMJ
0.00%	1206	39	23	27
1.00%	1202	63	33	39
2.00%	1200	75	50	57
3.00%	1208	121	75	79
5.00%	1207	151	93	116
10.00%	1200	195	141	191
20.00%	1202	362	405	360
50.00%	1202	1050	1086	1047
100.00%	1204	1720	1222	1724

Robustness to Join Order Errors



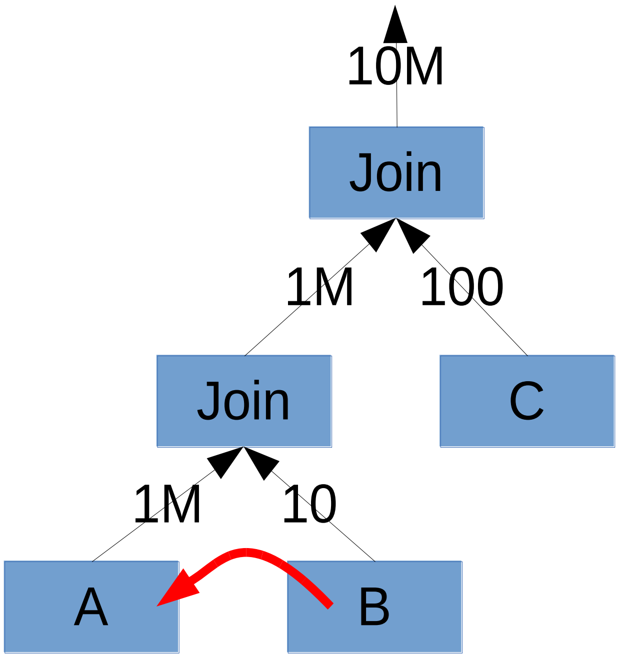
a) Good join order, no SIPS

Robustness to Join Order Errors

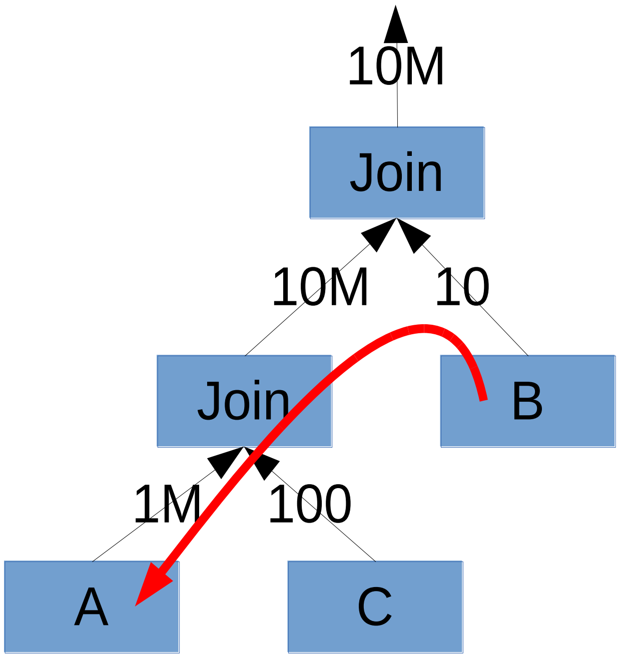


b) Bad join order, no SIPS

Robustness to Join Order Errors



c) Good join order, w/ SIPS



d) Bad join order, w/ SIPS

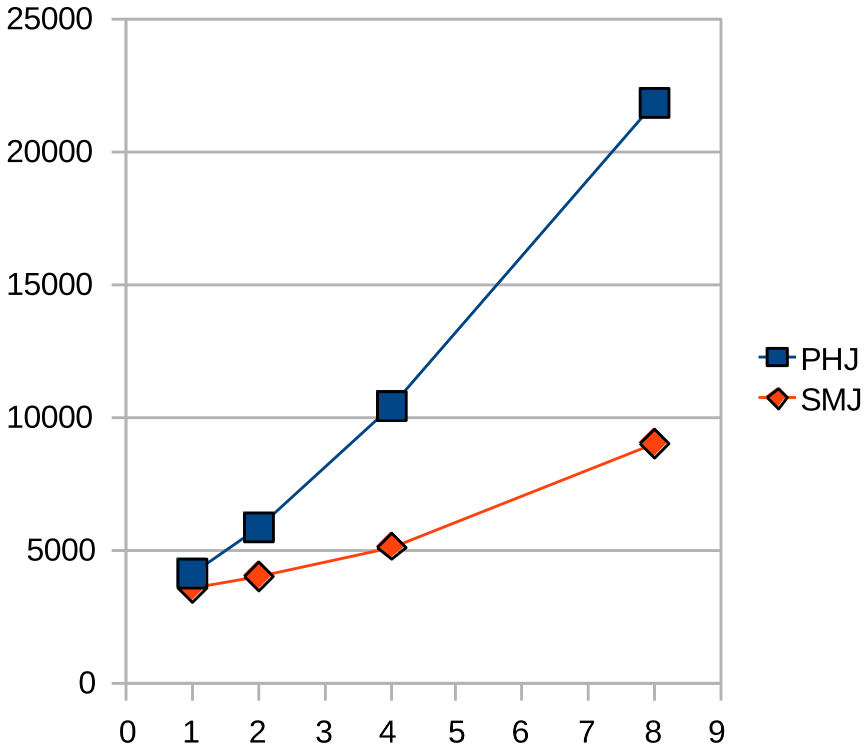
6. Mistake: Partitioned Hash Join

Partitioned Hash Join vs. Sort Merge Join

- (There are papers about these)

- PHJ was the first one tried
- SMJ was simpler to implement
- Sometimes one relation is sorted already
- Sometimes, you need to sort for other reasons
- Much more compatible with SIPS

Also, There's Performance



7. Good Idea: Data Collection

Big Data Mentality

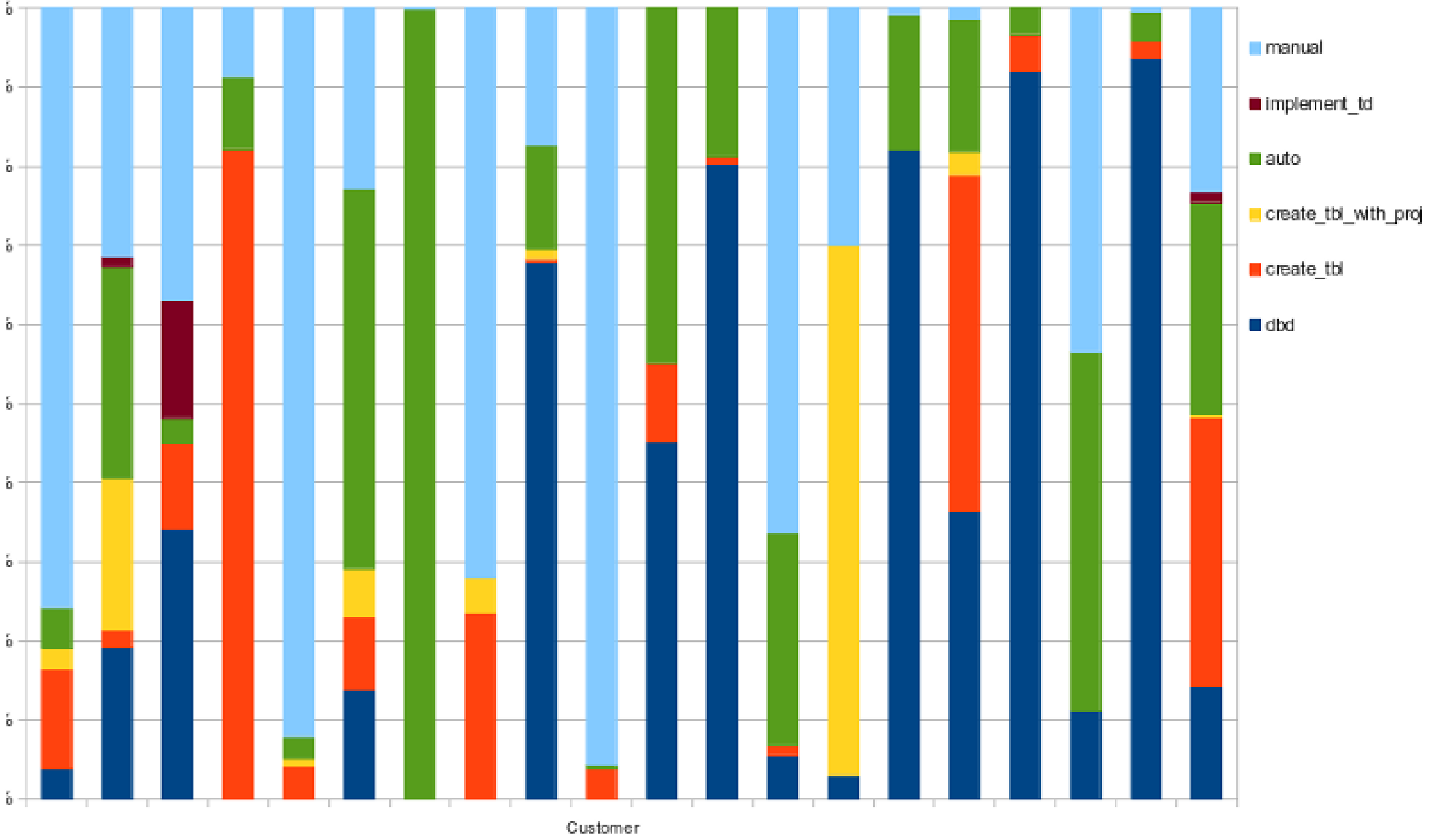
A database that doesn't self-collect is hypocrisy at its worst

- How busy is the machine compared to historical trends?
- What have my users been doing?
- How long will this job take to finish?
- What is the most common error?
- When was the last time we made a backup?
- My request's run-time changed… why?

- Have there been changes from the standard configuration?
- Are there problems that the customer hasn't called about?

- Which features have been used?
- Where do customer machines burn the most CPU cycles?

Unexpected Questions



Don’t Compromise on the Design

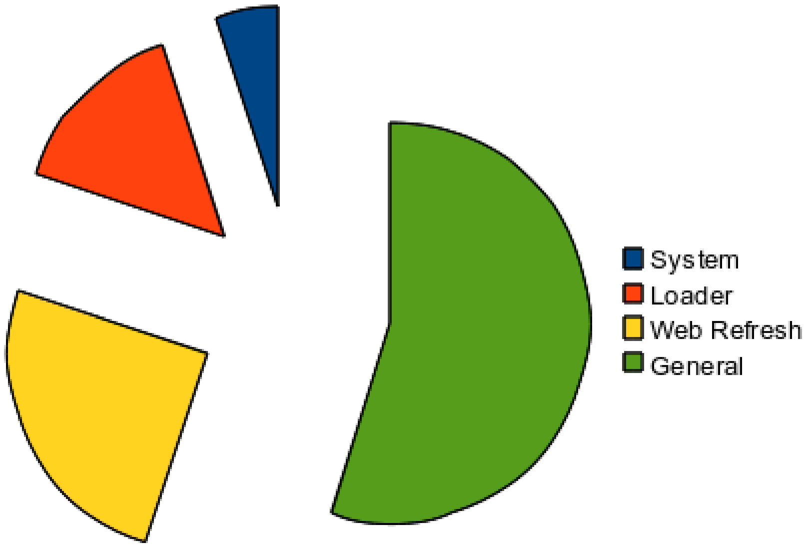
- Data collector can’t kill the system
- Like a log, lots of little appends
- Shouldn’t accidentally monitor itself
- Should be able to analyze off-line

- Result: Separate data management scheme

8. Good Idea: Dynamic Workload Management

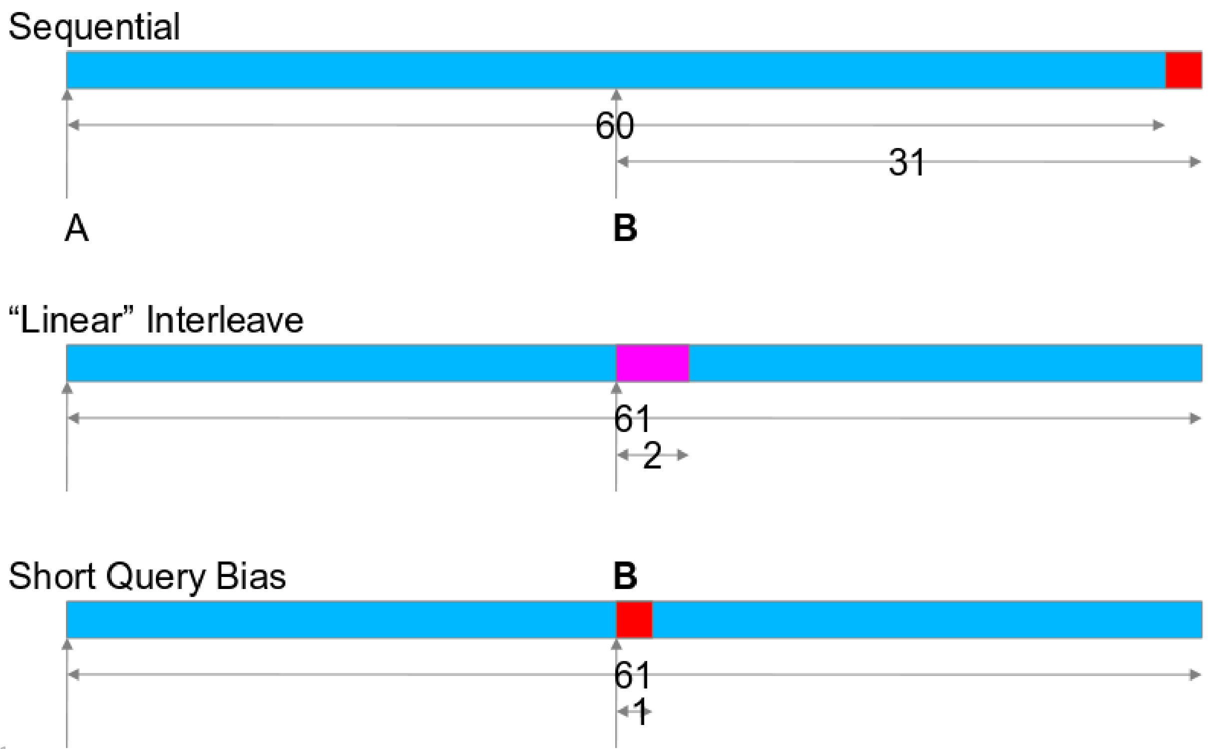
Static (Known) Workload Management

- Don't want reports to take over the entire system, preventing loads or tactical queries
- Keep some resources (e.g. memory) reserved so that high-priority queries can always begin
- Apply run-time prioritization to manage CPU and I/O



Unpredictable Workload: Short Query Bias

Independent: A=60s, B=1s



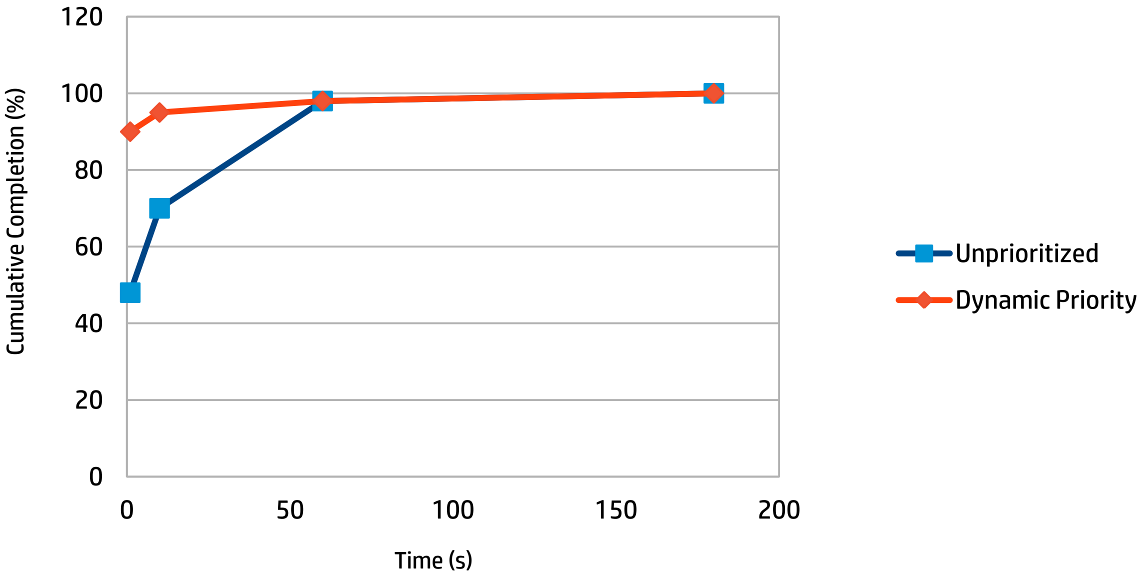
Dynamic Prioritization

Q: Are optimizer cost model estimates really that bad?

Dynamic Prioritization

Q: Are optimizer cost model estimates really that bad?

A: Doesn't matter!



Thank you

Please come visit our development team in:

Boston (Cambridge and Andover), MA

Pittsburgh, PA

Sunnyvale, CA