

**“One Size Fits All”
An Idea Whose Time Has
Come and Gone**

by

Michael Stonebraker

Current DBMS Gold Standard

- ◆ **Store fields in one record contiguously on disk**
- ◆ **Use B-tree indexing**
- ◆ **Use small (e.g. 4K) disk blocks**
- ◆ **Align fields on byte or word boundaries**
- ◆ **Conventional (row-oriented) query optimizer and executor**

Terminology -- “Row Store”

Record 1

Record 2

Record 3

Record 4

E.g. DB2, Oracle, Sybase, SQLServer, ...

Row Stores are **Write Optimized**

- ◆ **Can insert and delete a record in one physical write**
- ◆ **Good for OLTP**
- ◆ **But not for the data warehouse and other read-mostly markets**

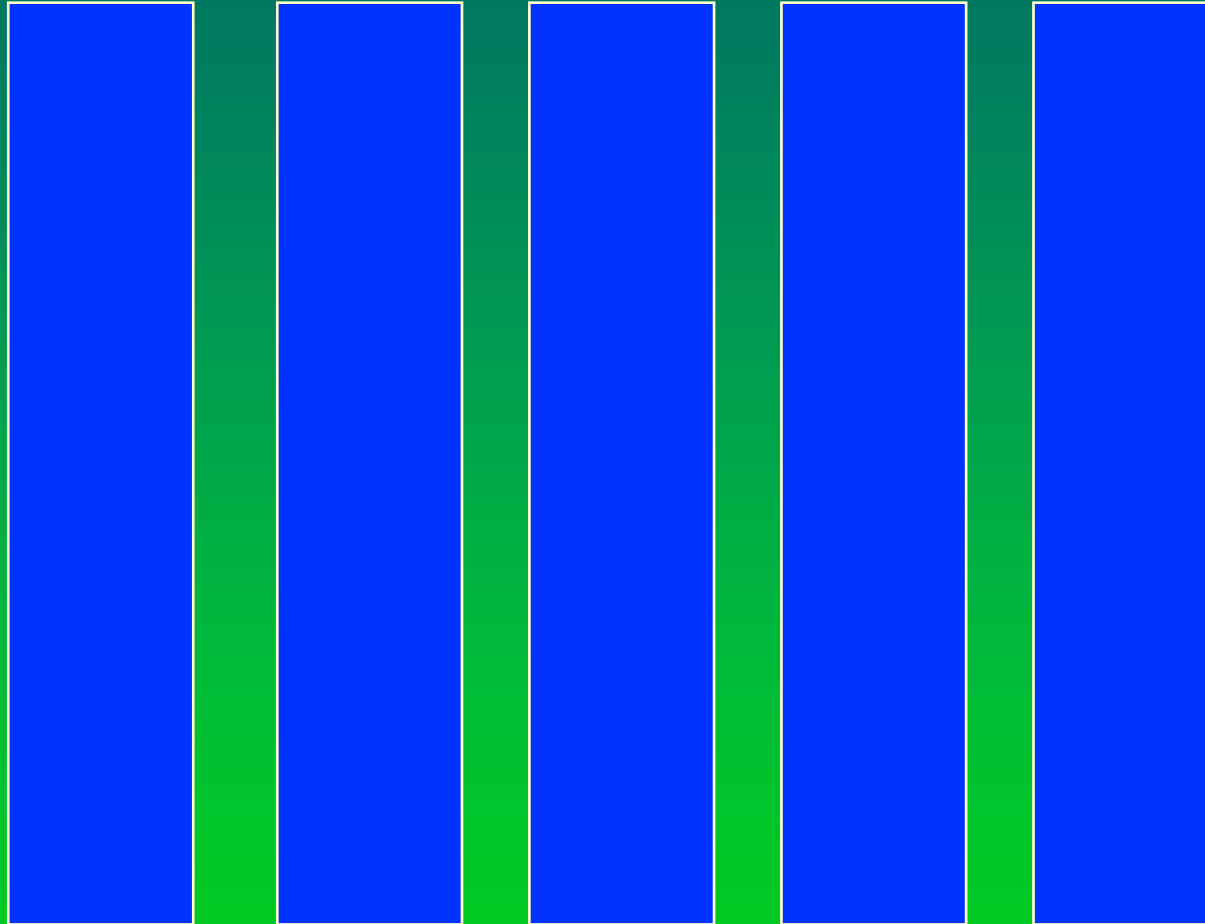
The Elephants and Warehouses

- ◆ **Bitmap indexes**
- ◆ **Star schema optimization**
- ◆ **Materialized views**
- ◆ **Compression (coding) or attributes**

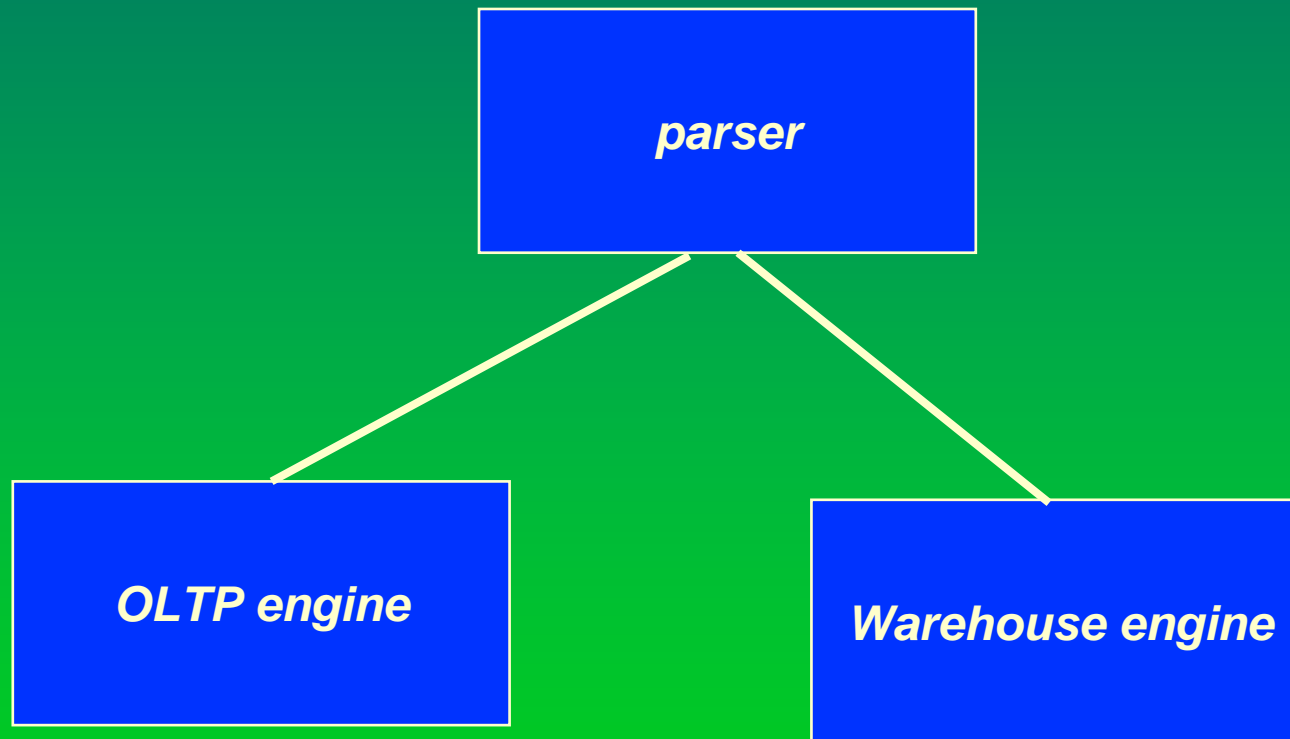
Ultimate Result....

- ◆ **Read optimized store**
- ◆ **Instead of a write optimized store**

A Column Store (Like Sybase IQ)



What is Fast Approaching...



Two Engines United by a Common Parser

- ◆ Marketing can preserve the “one size fits all” fiction
- ◆ Good idea because two engines causes
 - ◆ sales confusion
 - ◆ Marketing confusion
 - ◆ Compatibility issues

But it won't work in stream processing!

Example Application – Feed Alarms



Characteristics of Feed Alarm Pilot

- ◆ 500 rapidly updating tickers (5 sec. interval) + 4000 slowly updating tickers (60 sec. interval) in each FEED.
- ◆ Problem Types
 1. Low-level alarm ⇒
Ticker not seen within update interval.
 2. Problem in Feed ⇒
More than 100 low-alarms from Feed A or Feed B
 3. Problem in Exchange ⇒
More than 100 low-level alarms from NASDAQ or NYSE
- ◆ Suppression:
 - When problems of type 2 or 3 detected, do not emit (distracting) problems of type 1.

Results

- ◆ **Aurora/Grassy Brook implementation:**
 - ~ 160K msgs/sec on a 3.2GHz Linux pentium
- ◆ **Elephant solution**
 - ~900 msgs/sec on the same hardware

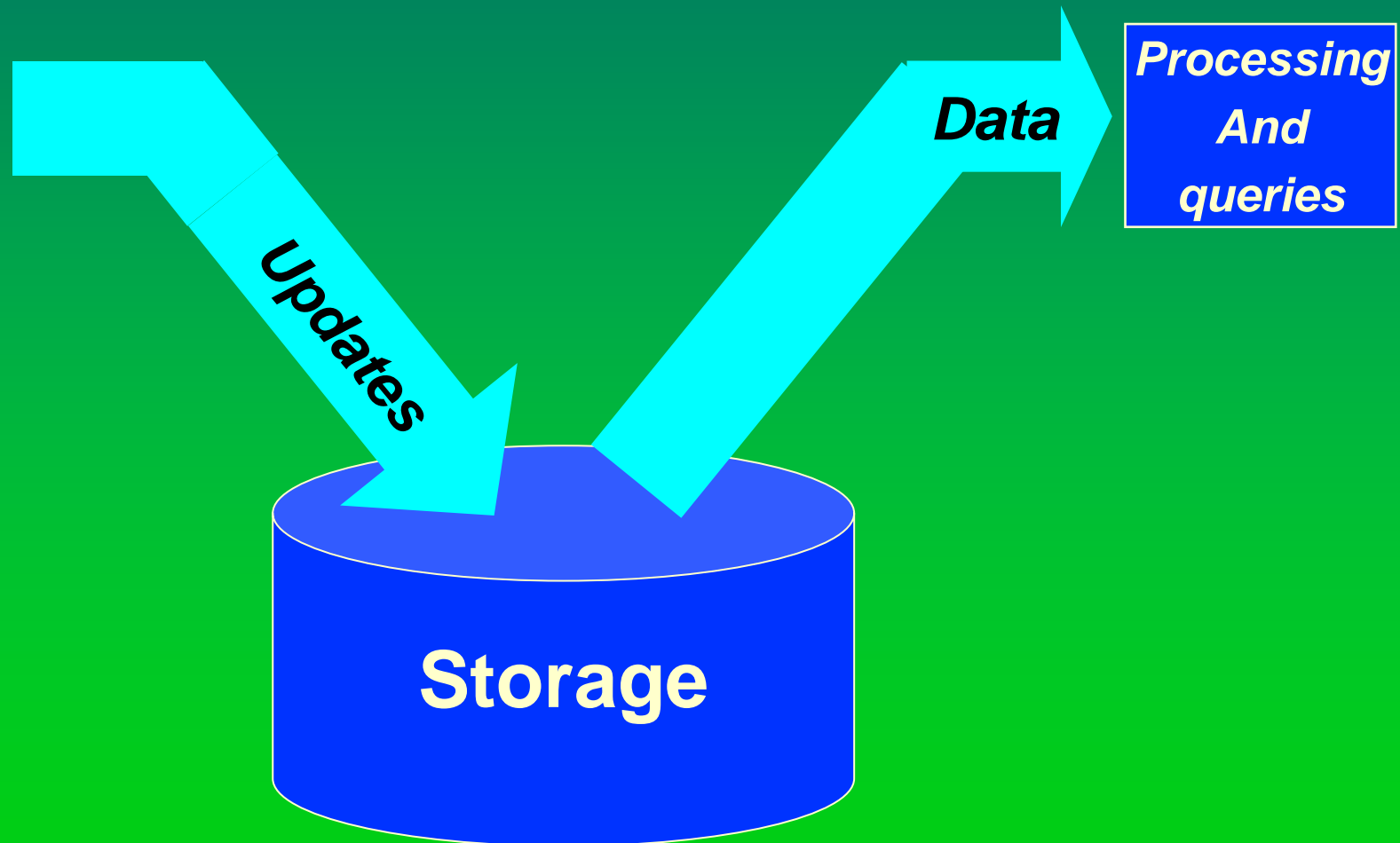
More than 2 orders of magnitude difference.....

Why?

- ◆ **Inbound vs outbound processing**
- ◆ **The right primitives**
- ◆ **Integration of application logic**

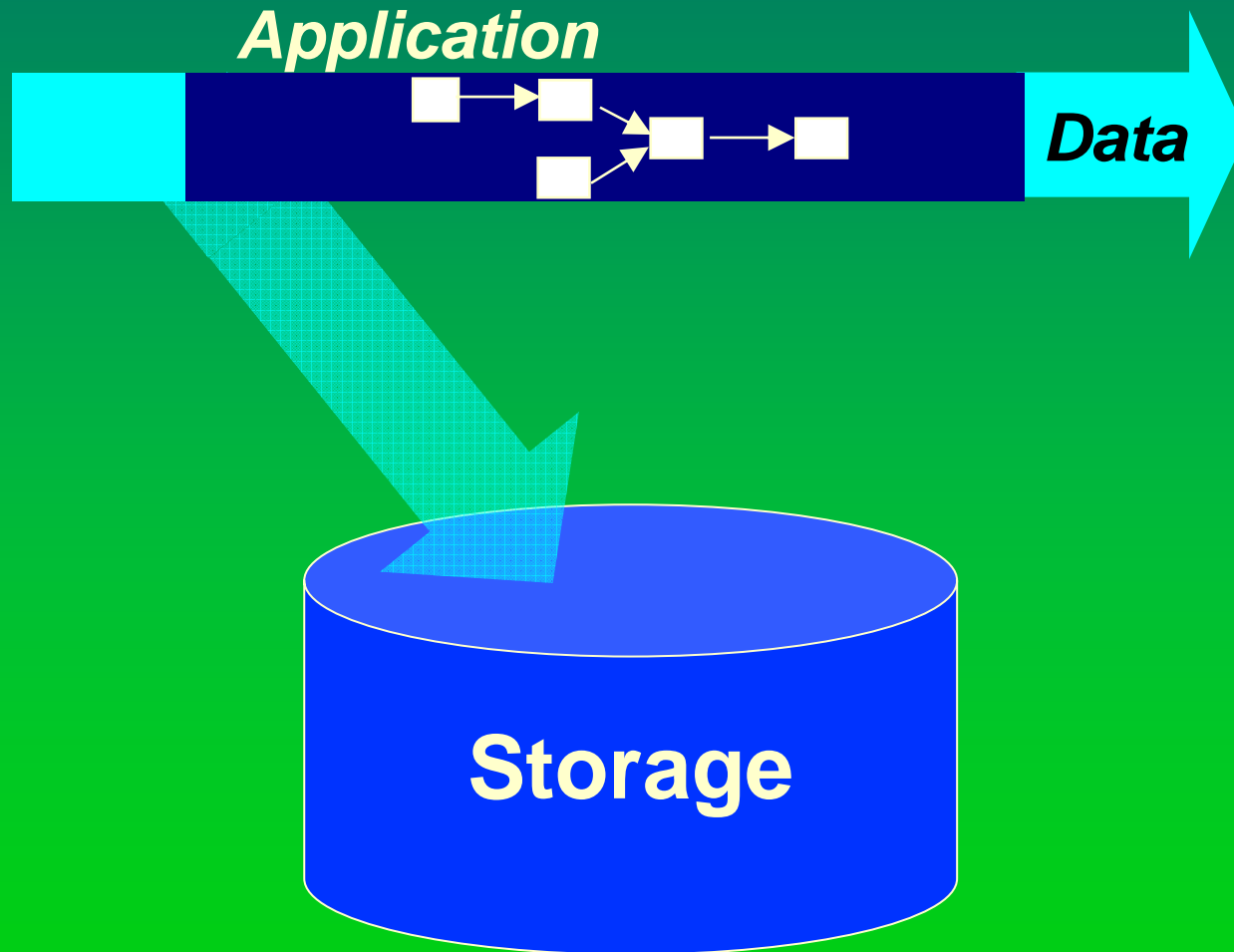
Traditional Model

Outbound Processing



Stream Processing Model

Inbound Processing



Inbound Processing

- ◆ **Never store the data!**
- ◆ **Lower overhead**
- ◆ **Lower latency**

Inbound Processing in DBMSs

- ◆ Triggers (glue-on)
- ◆ Limited support
- ◆ Often slow

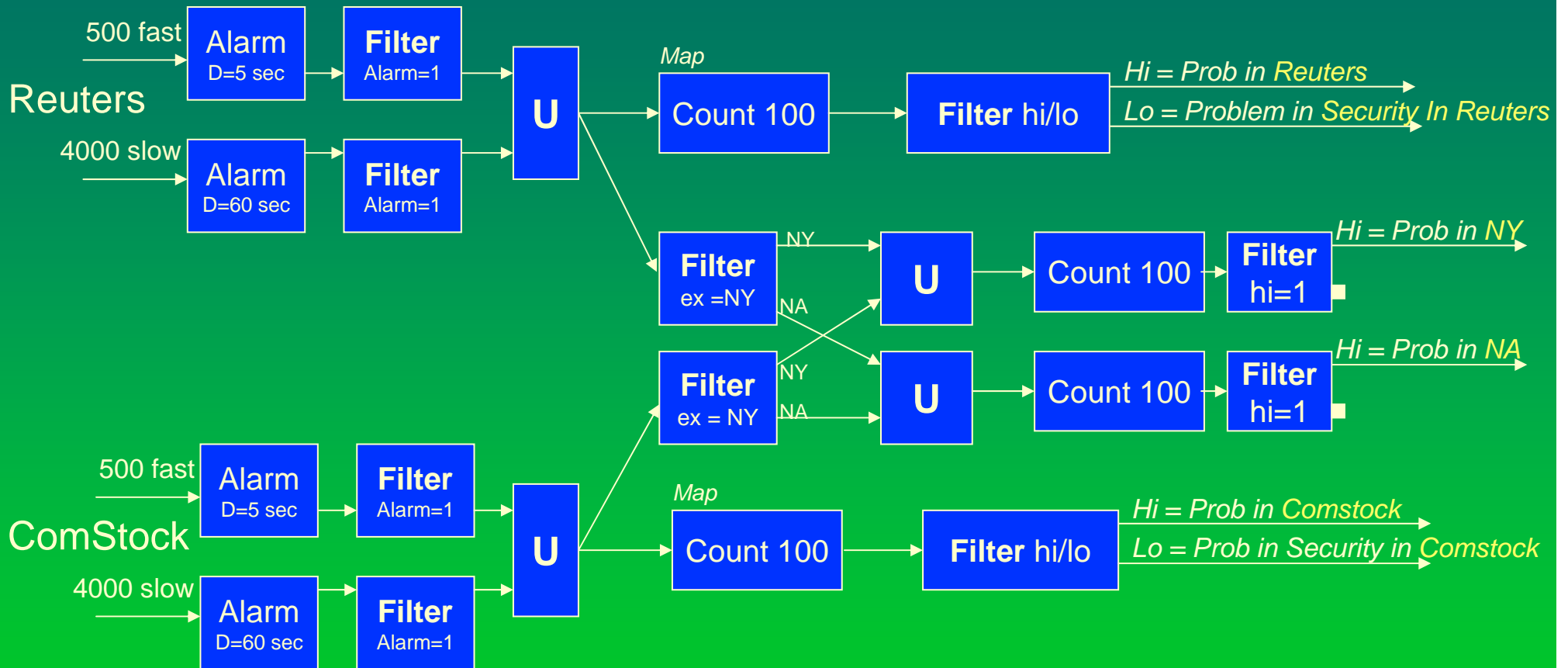
In theory, a DBMS could be both inbound and outbound, but this is a research project....

Just hooking a query plan up to a stream is not good enough.....

Windowed Time Series Operators

- ◆ **Windowed time series operators**
 - *Group by stock_id*
 - *Window is 2 ticks*
 - *Slide by 1 tick*
- ◆ **Resilient to stream imperfections**
 - *User-specified timeouts for late data*

Alarm Correlation Application



Windowed Aggregates With Timeouts

Alarm
D=5 sec

*same
as*

Aggregate
Group by ticker
Window (size = 2 tuples,
step = 1 tuple)
Timeout = 5 sec.

Windowed Aggregates with Timeout in DBMSs

- ◆ In the trigger system?
- ◆ On stored data (polling)?

Integration of Application Logic

- ◆ **All required capabilities in single system**
 - *No process switches*
 - *Integrated storage (not client-server)*

Integrated Code

Count 100

same as

Map

F.evaluate:

cnt++

if (cnt % 100 != 0) if !suppress emit *lo-alarm*

else emit *drop-alarm*

else emit *hi-alarm*, set suppress = true

- Lets first 100 *low-alarms* through.
- Emits one *high-alarm* for every 100 *low-alarms*.
- Suppresses *low-alarms* after 1st *high-alarm*.

Application Integration in DBMSs

- ◆ Client-server present for protection
- ◆ Stored procedures are a start
 - tough to do control flow
- ◆ Object-relational blades are better
 - But still tough to do control flow
- ◆ Unified programming language never made it
 - E.g. Rigel or Pascal R
- ◆ No support for embedded DBMS applications

Transactions in Streams

- ◆ **Locking**
 - Critical sections are enough; no need for xacts
- ◆ **Crash recovery**
 - Log-based recovery slow
 - doesn't recover whole state
 - System unavailable during recovery
- ◆ **Much better to just do HA**
 - Failover to a backup (Tandem-style)
 - Forget about state recovery

Net-Net

- ◆ **Inbound vs outbound processing**
- ◆ **Windowed primitives vs end-of-table primitives**
- ◆ **Separate app vs embedded app**
- ◆ **HA failover vs transactions**

Whenever These Matter a Lot

- ◆ Separate engine
- ◆ To get 2 orders of magnitude benefit

Candidates for a Separate Engine

- ◆ OLTP
- ◆ Warehouses
- ◆ Stream processing
- ◆ Sensor networks (TinyDB, etc.)
- ◆ Text retrieval (Google, etc.)
- ◆ Scientific data bases (lineage, arrays, etc.)
- ◆ XML (argued by some)

Obvious Research Template

- ◆ Pick an area where “one size doesn’t fit”
- ◆ And figure out what does

More Generally

- ◆ **Current system software factored into**
 - App server (e.g. Websphere)
 - Messaging system (e.g, MQSeries)
 - DBMS (e.g. DB2)
- ◆ **Stream processing engines integrate pieces of all three**
 - To avoid process switches
- ◆ **How many other interesting factorings are there?**