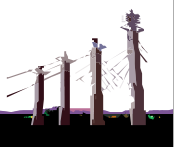

Financial Solutions with IDS and Time Series
Kevin Brown, Informix Lead Architect, IBM,
kbrown3@us.ibm.com
C06
Monday April 28th 4:40-5:40 PM
2008 IIUG Informix Conference
www.iiug.org

Agenda

- What is a Time Series?
- Key Concepts
- Building Time Series Applications
- Why do we need a Real-Time Loader?
- Overview of the RTL
- Putting it all Together



What is Time Series Data?

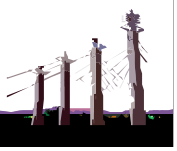
- Time series data is:
  - Always time-stamped
  - Typically read and written in ascending timestamp order
  - Access to one time series is usually completed before moving to the next time series.

Examples of Time Series Data

- Financial market data
- Temperature over time
- Network monitoring
- Phone usage
- Manufacturing processes
- Location of a fleet of trucks over time
- …

Key Strengths of Informix Time Series

- Performance
  - Extremely fast data access
    - Data layout optimized on disk
  - Handles operations hard or impossible to do in standard SQL
- Space Savings
  - Can be over 50% space savings in best case
- Toolkit approach allows users to develop their own algorithms
  - Algorithms run in the database to leverage buffer pool
- Conceptually closer to how users think of time series



Typical RDBMS Time Series Representation

S_P_500

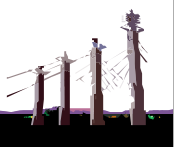
Primary Key

name	date	open	high	.....	close
IBM	Mon	Value 1	Value 2	.....	Value N
IBM	Tue	Value 1	Value 2	.....	Value N
IBM	Wed	Value 1	Value 2	.....	Value N
...	...	...	...	.....	...
ATT	Mon	Value 1	Value 2	.....	Value N
ATT	Tue	Value 1	Value 2	.....	Value N
ATT	Wed	Value 1	Value 2	.....	Value N
...	...	...	...	.....	...

Example of a Table with a Time Series Column

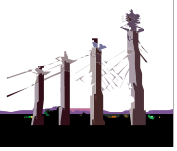
S_P_500

Name (lvarchar)	Series timeseries(daybar)
IBM	[(Mon, v1, ...)(Tue,v1…)]
ATT	[(Mon, v1, ...)(Tue,v1…)]
IFMX	[(Mon, v1, ...)(Tue,v1…)]
NAG	[(Mon, v1, ...)(Tue,v1…)]
SUN	[(Mon, v1, ...)(Tue,v1…)]
MSFT	[(Mon, v1, ...)(Tue,v1…)]
SGI	[(Mon, v1, ...)(Tue,v1…)]
HP	[(Mon, v1, ...)(Tue,v1…)]



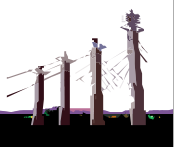
Key Concepts: Regular Time Series

- Timestamps have a regularly repeating pattern of intervals
  - daily, hourly, etc...
- Might be breaks in the pattern
  - A work week is data captured for 5 days in a row then nothing for 2 days
- Only one piece of data per interval
  - If an interval has not been inserted into then it has the value of NULL
  - Intervals not inserted into occupy some space
    - Heuristic: intervals not inserted into at the end of series do not take up space
- Can be thought of as an array
  - Optimized to return data at offset, not timestamp



Key Concepts: Irregular Time Series

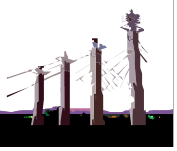
- Data in an irregular time series does not have a regularly repeating pattern of intervals
  - Any interval may have zero or more pieces of data
- Missing data takes no space on disk
  - There really is no concept of missing data
- Only efficient way to access data is by timestamp
  - You can retrieve the Nth piece of data, but the code does a linear search
- Data can be stair stepped
  - Value persists until next value arrives – for example stock prices
- Data can be discrete points
  - Value is valid only at the given time – for example heart beats



Key Concepts: Calendars

- The Calendar is a User Defined Type (UDT) that contains:
  - A start time
    - Determines the earliest time data can be stored
  - A pattern of “on” and “off” periods
    - Determines when data may be stored
  - An interval
    - year, month, day, hour, minute, second
- The SQL Syntax to create a calendar is:

insert into Calendartable (c_name, c_calendar) values
(‘daily’, ‘start(2001-01-01 00:00:00), pattern({5 on, 2 off}, day)’);



Key Concepts: Row Types

- A time series is a constructor of row types
  - The SQL used to create a row type is:

    create row type new_type_name(colname1 type, colname2 type,…)

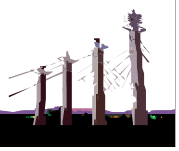
  - Time series requires the type of the first column to be "datetime year to fraction(5)"
- The other columns may be of any type except:
  - blob, clob, text, serial
- All the normal restrictions that Informix has for tables on the number of columns, total size of row apply
- Rows in a time series are referred to, sometimes, as "elements"

Key Concepts: Containers

- Containers are data structures that hold data for one or more time series
- You cannot mix data for regular and irregular time series in the same container
- A container is created in a dbspace with this SQL

  execute procedure TsContainerCreate('cont_name', 'dbspace_name', 'rowtype_name', first_extent_size, next_extent_size);

- A container is made up of index pages and data pages
- Containers allow time series data to be spread onto many disk partitions



Putting it all Together

- Create a calendar:

  insert into Calendartable (c_name, c_calendar) values ('daily', 'start(2001-01-01 00:00:00), pattern({5 on, 2 off}, day)');

- Create a row type:

  create row type daybar (tstamp datetime year to fraction(5), bid float, ask float, bidsize int, asksize int);

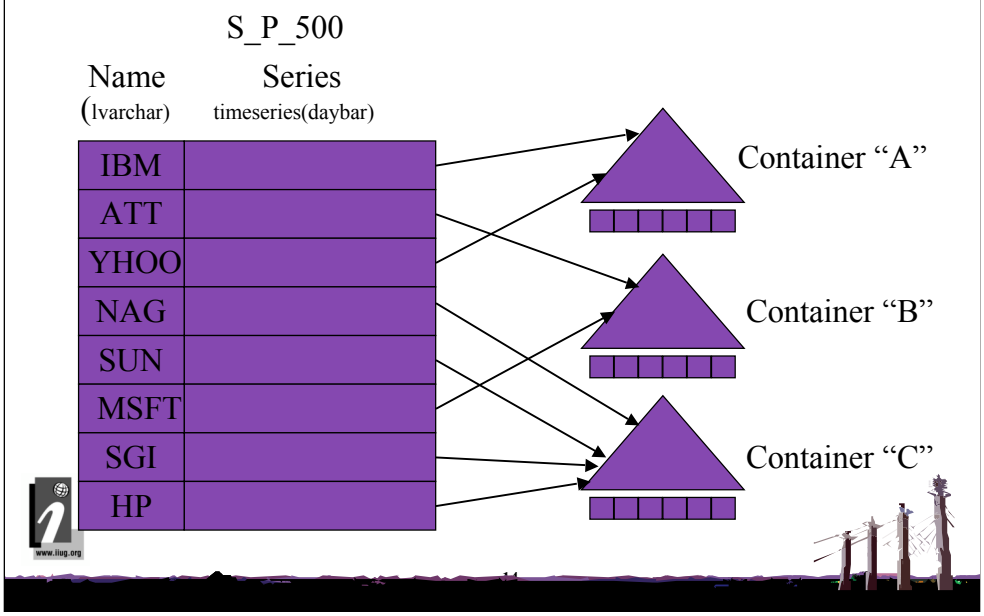
- Create a container:

  execute procedure TsContainerCreate('stock_data', 'stock_parition', 'daybar', 1 GB, 1GB);

- Create a table and insert a row:

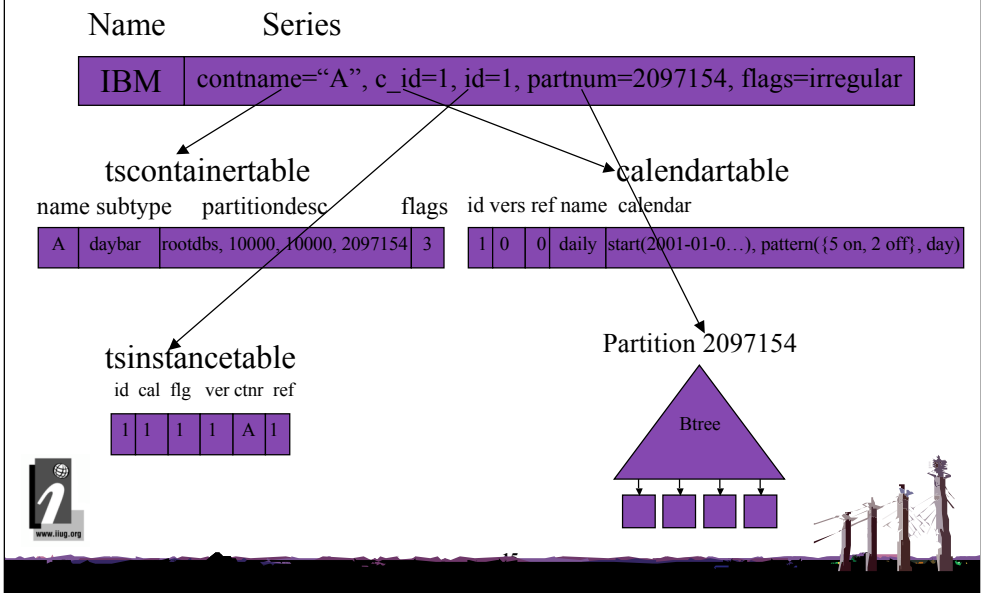
  create table S_P_500(name char(32), series timeseries(daybar));
  insert into table S_P_500 values ("IBM", "calendar(daily), container(stock_data), regular");

An Informix Table Containing a Time Series Column



TimeSeries Drill Down

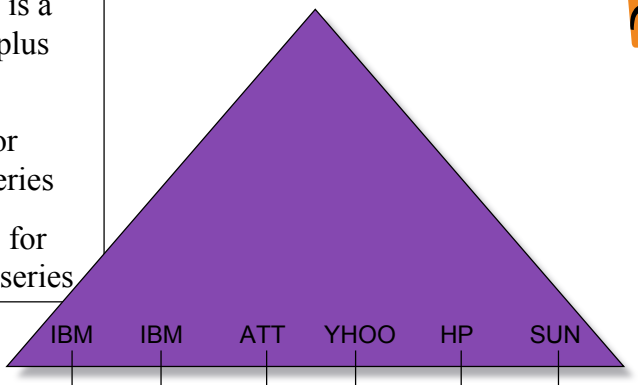
A row from the S_P_500 Table:



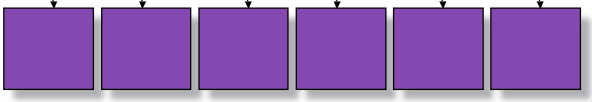
Container Layout

The index key is a time series id plus either:

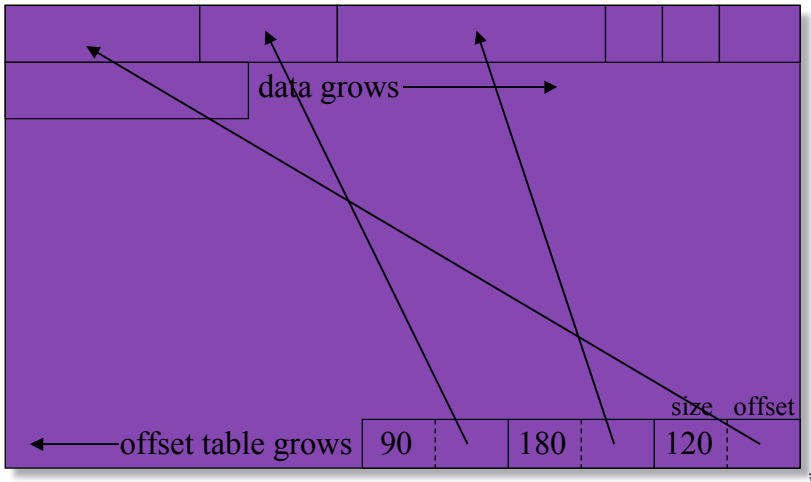
- An integer for regular time series
- A timestamp for irregular time series



Data Pages:

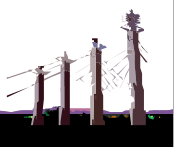


Page Structure in the Container



Row Structure in a Container

- A time series row contains a header, data and a footer
- The header contains 1 bit for each column in the row rounded to an 8 bit boundary
  - If a column is null the bit is 1, otherwise it is 0
  - Leads to great compression!
- The data portion contains the data from the row
- The footer is one byte and has flags indicating whether the row is entirely null or hidden
  - Bit 2 = 0 indicates row is null
  - Bit 3 = 1 indicates the row is hidden

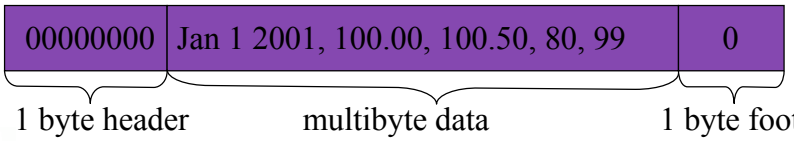


Rows, Continued...

- If the row type was declared as

  create row type daybar(tstamp datetime year to fraction(5), bid float, ask float, bidsize int, asksize int);

- Then a row for an irregular time series with no null columns and not hidden might look like:



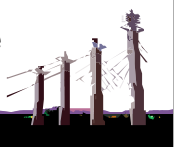
Building Applications with the TimeSeries Datablade

- Several interfaces are available:
  - SQL
  - VTI
  - SPL
  - Java
  - C-API

- It's a toolkit approach!
  - Allow people to build their analytics

TimeSeries SQL Interface

- Time series data is usually accessed through user defined routines (UDR's) from SQL:
  - Clip() - clip a range of a time series and return it
  - LastElem(), FirstElem() - return the last (first) element in the time series
  - Apply() - apply a predicate and expression to a range of elements in a time series
  - AggregateBy() - change the interval of a time series using a aggregate function
  - SetContainerName() - move a time series from one container to another
  - BulkLoad() - load data into a time series from a file



TimeSeries SQL Examples

- Get all of IBM since the beginning of the 2001

  Select Clip(series, '2001-01-01 00:00:00.00000', Current) from S_P_500 where name = 'IBM';

- Get the last closing price of IBM

  Select GetLast(series).close from S_P_500 where name = 'IBM';

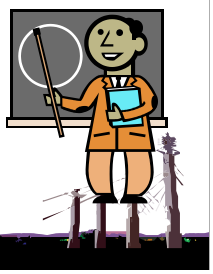
- Find the high for each week of IBM

  Select

  AggregateBy('max($high)', 'weeklycal', series, '2001-01-01 00:00:00', Current) from S_P_500 where name = 'IBM';

TimeSeries VTI Interface

- Makes time series data look like standard relational data
  - useful for programs that can't handle objects
  - useful when application connects with ODBC
  - There is a small (10%) penalty for using VTI
- Restrictions
  - A VTI table can only reference one time series column from the base table
  - No secondary indices are allowed
- SQL to create a VTI table:
  execute procedure tscreatevirtualtab('t_vti', 't');



VTI Interface: Continued

TimeSeries Table
S_P_500

Name (lvarchar)	Series timeseries(daybar)
IBM	[(Mon, v1, ...)(Tue,v1…)]
ATT	[(Mon, v1, ...)(Tue,v1…)]
IFMX	[(Mon, v1, ...)(Tue,v1…)]
NAG	[(Mon, v1, ...)(Tue,v1…)]
SUN	[(Mon, v1, ...)(Tue,v1…)]
MSFT	[(Mon, v1, ...)(Tue,v1…)]
SGI	[(Mon, v1, ...)(Tue,v1…)]
HP	[(Mon, v1, ...)(Tue,v1…)]

TimeSeries Virtual Table
S_P_500_VT

Name	date	open	high	
IBM	Mon	Value 1	Value 2	...
IBM	Tue	Value 1	Value 2	...
IBM	Wed	Value 1	Value 2	...
...	...	...	...	...
ATT	Mon	Value 1	Value 2	...
ATT	Tue	Value 1	Value 2	...
ATT	Wed	Value 1	Value 2	...
...	...	...	...	...

Execute procedure tscreatevirtualtable ('S_P_500_VT', 'S_P_500');

Stored Procedure (SPL) Example

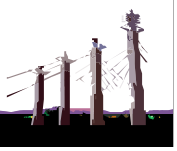
```
-- count non-null elements in a time series
create function spl_test(arg lvarchar) returning integer
    define var daybar;
    define cnt integer;

    let cnt = 0;
    foreach execute function
            transpose((select series from S_P_500 where name = arg))
      into var
          let cnt = cnt + 1;
    end foreach
    return cnt;
end function;
```

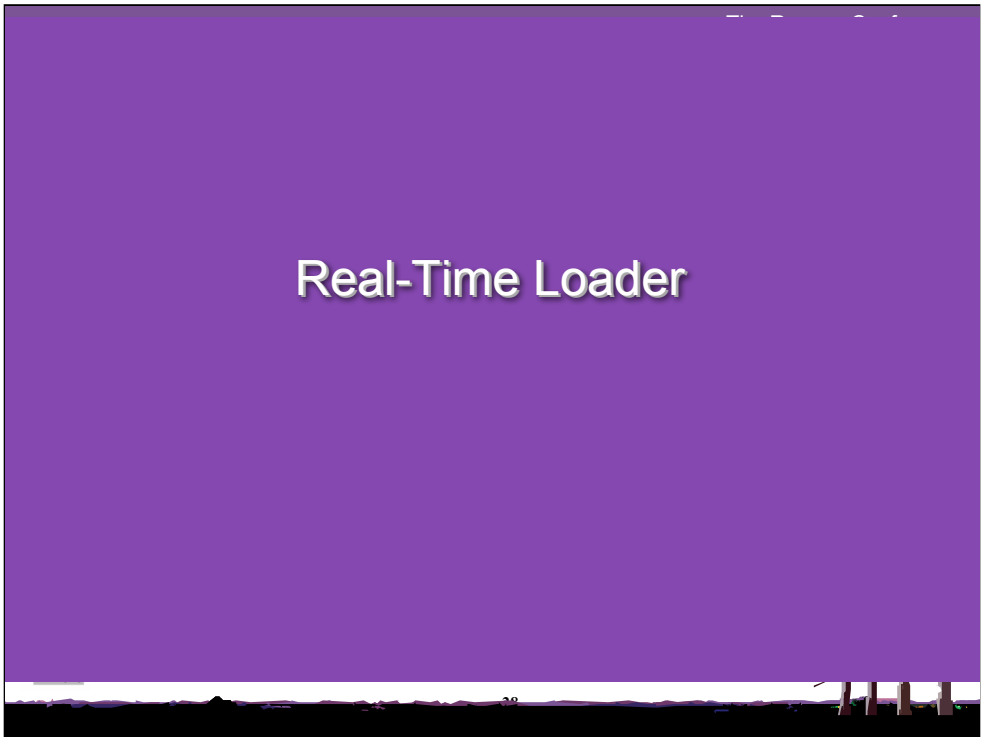
TimeSeries C-API Interface

- Client and server versions of the API
- Treats a time series like a table (sort of)
  - Functions to open and close a time series
  - Functions to scan a time series between 2 timestamps
  - Functions to create a time series
  - Functions to retrieve, insert, delete, update
- Plus another 70 functions defined

C-API example

```
#include "tseries.h"
/* Example of counting non null elements */
mi_integer
nelems(ts_timeseries *ts, MI_FPARAM *fParam)
{
        ts_tsdesc               *tsdesc;
        ts_tscan                *tsscan;
        ts_tselem                 elem;
        mi_integer                cnt;
        mi_integer                ret;
        MI_CONNECTION *conn;

        conn = mi_open(NULL, NULL, NULL);

        /* open the source timeseries */
        tsdesc = ts_open(conn, ts, mi_fp_argtype(fParam, 0), 0);

        /* scan the entire timeseries */
        cnt = 0;
        tsscan = ts_begin_scan(tsdesc, 0, NULL, NULL);
        while ((ret = ts_next(tsscan, &elem)) != TS_SCAN_EOS)
                if (ret != TS_SCAN_NULL)    /* don't count nulls */
                        cnt++;    /* clean up */

        ts_end_scan(tsscan);
        ts_close(tsdesc);

        return(cnt);
}
```



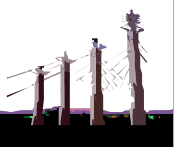

Real-Time Loader

Informix Real Time Loader

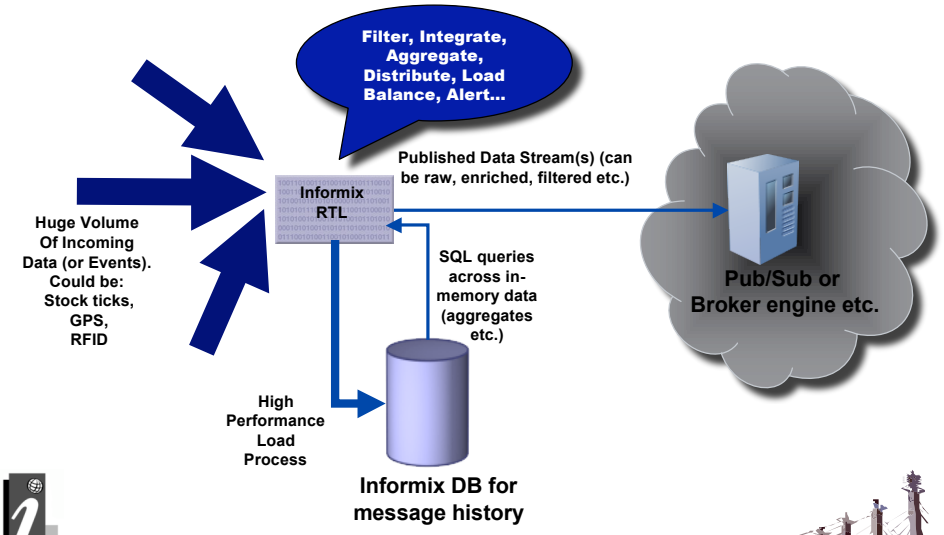
- Customer Pain
  - Growing requirement to capture and store huge large volumes of “real time” data
    - The data tends to be generated from electronic sources such as Market Data feeds, Geo-locations from GPS or Product IDs from RFID tags
    - The number of messages (each is small) can be huge (e.g. for Market Data 100,000+/sec)
    - Key Business drivers are Optimization (military, supply chain, equity trading, logistics) and Enforcement (risk or insider-trading, bio-security, anti-terror)

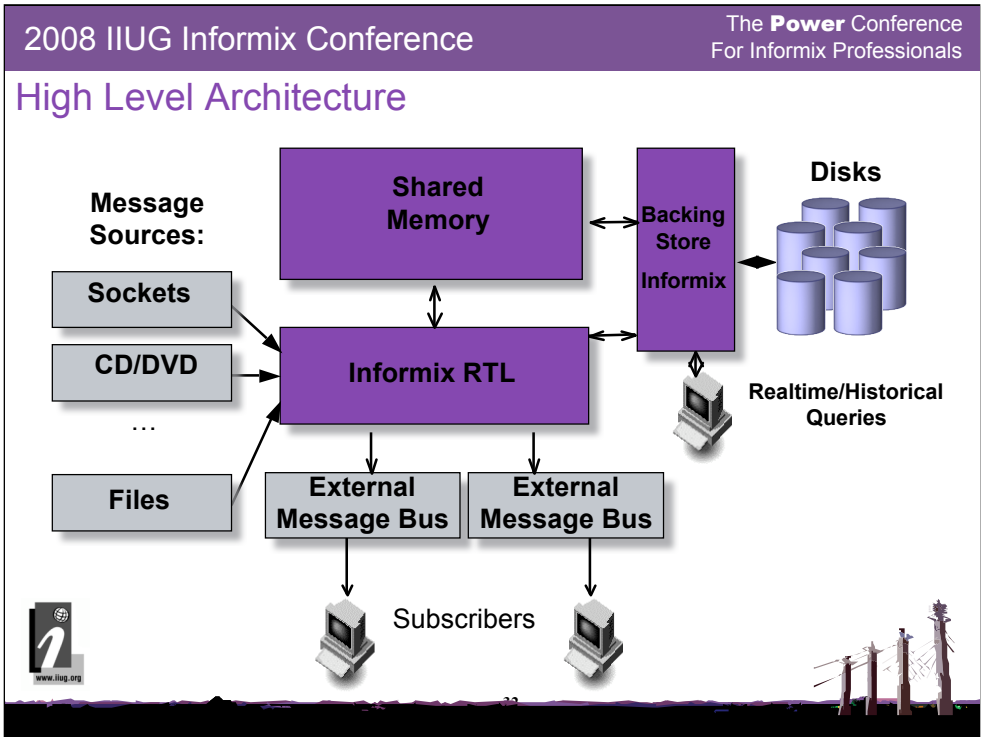
RTL Features

- Persists over 100,000 messages/sec to a database
  - 4 CPU dual core linux box
  - 16 GB memory
  - 25 messages
- Simultaneously handles data from many continuous data feeds
- Simultaneously persists data to many different database servers
- Filters and enriches messages and publishes results
- Provides real-time access (< .1 secs) to data
- Gateway to multiple databases for simultaneous access

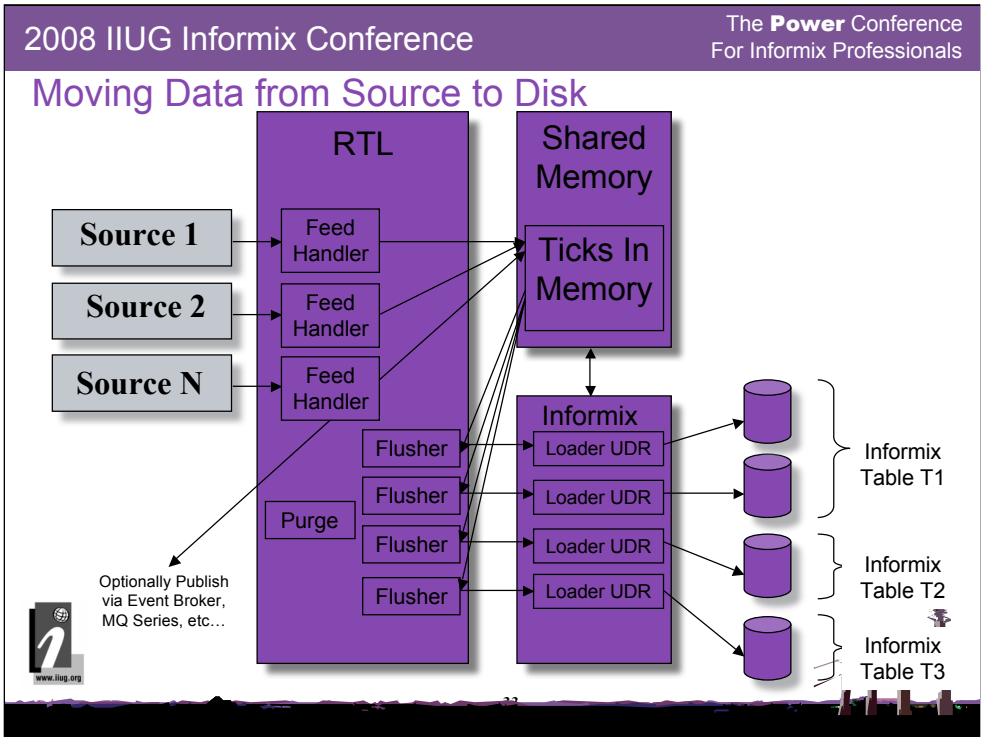


Generic RTL Scenario





See notes for next slide…



This slide shows how records flow from a data source to the persistence layer and then eventually get removed. A row first enters through one of the feed handler library threads. From there the data is written to shared memory. Some time later a flusher thread retrieves blocks of ticks that have not been persisted and writes them to the persistence store by making calls to the persistence library. For Informix the flusher threads are setup to each flush data for a particular partition. Multiple rows are sent to the database in a single block and inserted with a prepared insert statement. Some time later the purge thread wakes up and goes through shared memory removing all records that have been persisted to disk and have remained in memory longer than a user configured amount of time.

RTL Virtual Table Interface Example

S_P_500_RT_VTI

name	tstamp	price	volume
IBM	Jan 1, 2004 12:00:00	90.2	1000
IBM	Jan 1, 2004 12:00:01	90.3	500
HPQ	Jan 1, 2004 12:00:00	21.7	700
HPQ	Jan 1, 2004 12:00:04	21.6	200
IBM	Curtime – 2 minutes	90.2	1000
IBM	Curtime – 1 minute	90.2	1000
HPQ	Curtime	90.2	1000

Requests for these records causes the VTI code to use the Timeseries blade to retrieve them from disk

Request for these records causes the VTI code to retrieve them from shared memory

Kevin Brown (kbrown3@us.ibm.com)

http://www.ibm.com/informix

Kevin Brown: kbrown3@us.ibm.com