In-memory Tables

Larry Strickland
Chief Product Officer

Abstract

In-Memory tables have the promise of drastically improving the performance of an application, so much so, that previously rejected application designs may become possible. This presentation will introduce In-Memory tables; how to find applications that may benefit from In-Memory tables; how to adapt applications to use In-Memory tables; and, discuss what needs to be considered when designing a system to manage the In-Memory tables. Finally some real world results will be presented showing performance improvements as a result of the use of In-Memory tables.

Agenda

- Introduction to IN-MEMORY tables
- Designing IN-MEMORY tables
- Where IN-MEMORY tables are most useful
- Some performance results
- Adapting for IN-MEMORY
- Identifying IN-MEMORY table Candidates
- Summary

Introducing IN-MEMORY tables

IN-MEMORY – Shortest Path to Data

- Obtaining the shortest path to Data is essential in improving performance of applications

- There are two major components to that path
  1. Accessing data from permanent storage
  2. Executing the machine instructions necessary to find and retrieve the right data

Putting data in memory

- Clearly putting data into memory can improve the performance

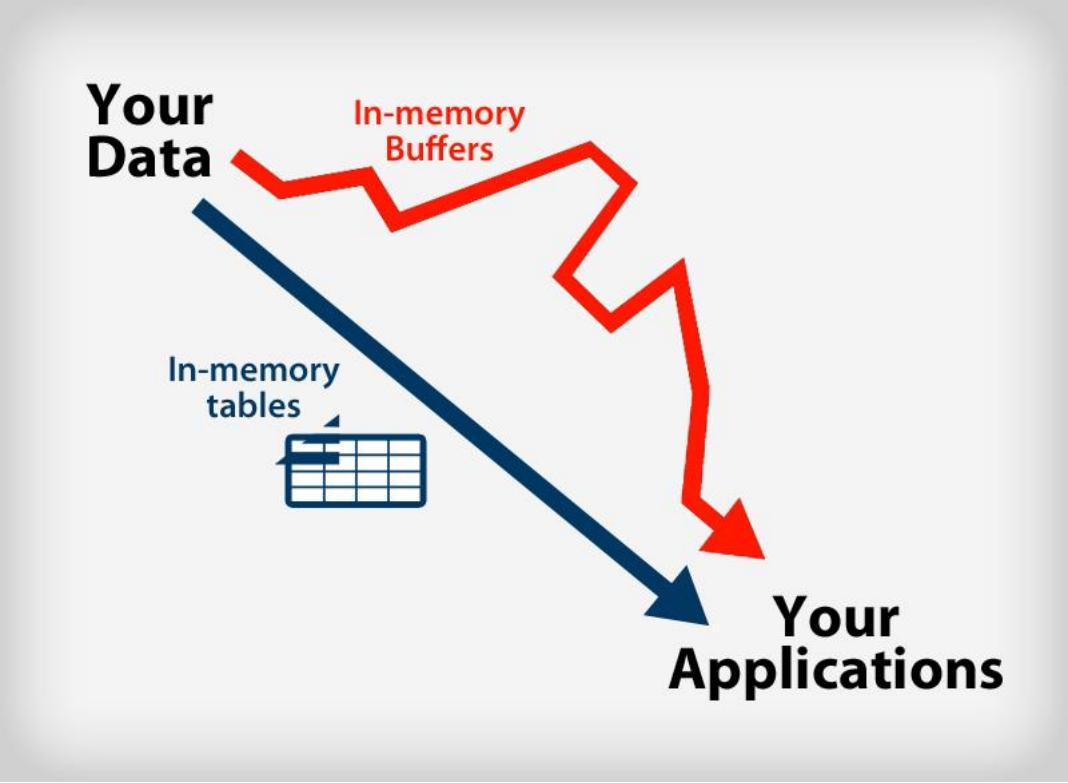
- There are a number of methodologies that have achieved this:
  - In memory tables
  - Data buffers
  - Caches

Reducing the machine instructions

- Even after data has been moved to some sort of storage in memory the access path can still be very long:
  - Often the same code is used for all sorts of access, including on disc. The large number of possible branches increases the code length.
- Only systems designed to be optimized for in memory access are going to reduce the code path.

IN-MEMORY table manager

Shortest and fastest path to data

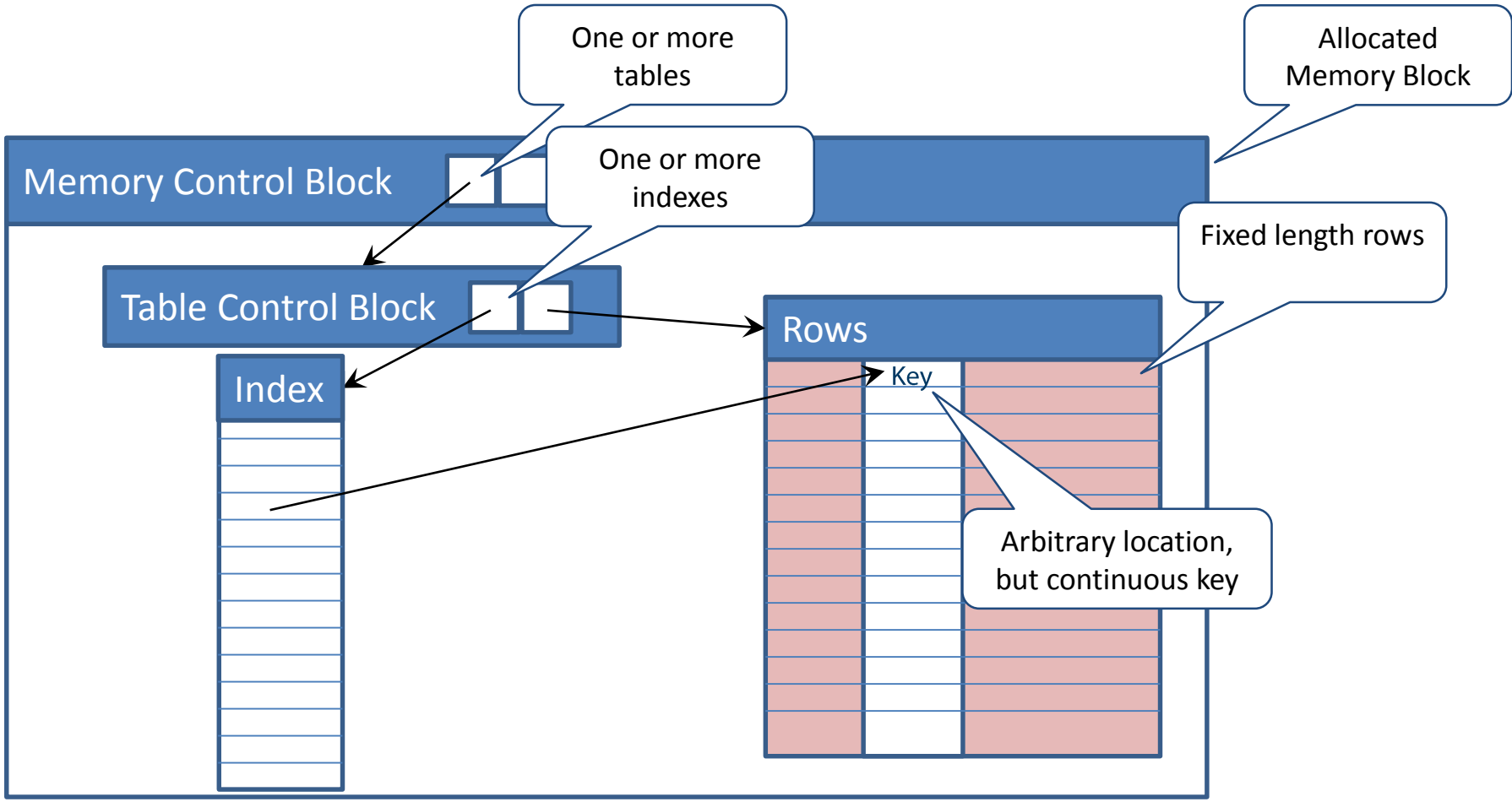


Which path would you prefer?

DataKinetics

Designing IN-MEMORY tables

Designing in-memory system



DataKinetics

Multiple Organizations/Retrieval

- Organizations
  - Sequential (asc, desc)
  - Hash
  - Random
  - User Defined Sequence
  - B-tree
- Retrieval Methods
  - Serial
  - Binary
  - Queued Sequential
  - Bounded Binary
  - Hash

Creating the index

Using the index

DataKinetics

Hash Organization and Retrieval

- Hash Organization and Retrieval uses a hash algorithm – many algorithms available.
  - Ideally looking for low collisions, and not too sparse an index – there is much in the literature about the best algorithms for this.
  - However, when you are looking for speed:
    - The time to calculate the value on retrieval is very important.
    - May need to use less efficient algorithm from a collision point of view, to maintain speed.

Access

- Fixed row length means very fast access:
  - row_start_address = data_start_address + (row_number-1) * row_length
  - key_position = row_start_address + key_offset
  - Index holds key_position
- Reads – very fast
- Updates
  - If key changes – need to recalculate index (in part)
- Insert
  - Add row to end of row data (or unused row)
  - Need to extend indexes – but can take advantage of block moves
- Deletes
  - Similar to Insert – simpler if memory is not recovered

Design Considerations

- Avoid I/O
- Avoid OS services in general
- Avoid getmains
- Avoid locking (assumption of mostly read only)
- Use Cookies to get to existing position (index entry) – for subsequent calls
- Implicit open (assumed tables are backed)
- Consider page size (multiples of 4k work well)

Shareable IN-MEMORY tables

- Eliminate the loading of local data space on initialization
- Ensure all applications have identical data
- Share data between applications
- Overcome processing bottlenecks between applications
- Use tables as a write through cache, message queues
- Tables in a shared data space outlive their transaction or application

Where to use IN-MEMORY tables

Where is this most useful?

Three common scenarios:

1. Temporary data
2. Frequently read data

   (note, this is frequently read rows, not frequently read tables)

3. Rules Engines

   Replacing the if…then…else with table lookups

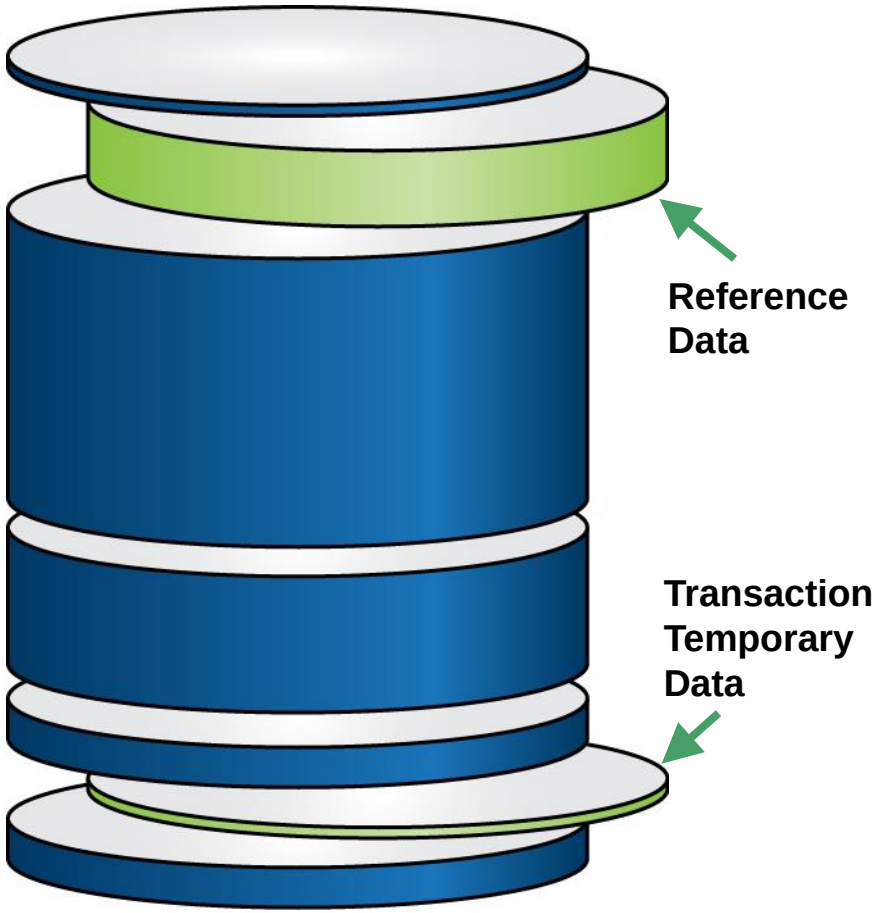
Temporary Data

- Temporary Data Tables
  - Load all data before generating indexes
  - Use indexes to perform virtual sorts, and data organization
- Examples:
  - Replace Cobol SORT or Sort All
  - Build reports for display, such as consolidated bank statements

Frequently Read Data

- Data that is read the most frequently provides the greatest opportunity for improvement
- Consider
  - A credit card transaction system of a billion transactions a day, with 100 different card types.
  - During reconciliation, each transaction is read once, however, each row in the card type table is read 10 million times (on average)
  - The card type table will benefit greatly from optimized in-memory access
  - The transaction table won't !

Not all data is handled the same way



Reference data

- Is 5-15% of your total data
- Changes infrequently
- Is accessed often, may represent as much as 80% of your accesses

Temporary data

- Is created, processed and then deleted
- Generates a high volume of data accesses for the volume of data

Remaining data

- The largest volume of data
- Read often followed by a write
- The lowest number of accesses

DataKinetics

Rules Engines

- Moving logic from programs into rules based tables is great for flexibility, however it can really hurt performance
- If rules tables are used appropriately with optimized in-memory methods, both speed and flexibility is possible
- Example construct
  - Inputs are converted to a vector
  - Vector is used as the key to rules table (matching zero or more)
  - Inputs are given to programs identified by matching rules (could be a start address)
    - May be in parallel or series
    - Can be iterative (the output of a program goes back into the rules engine)

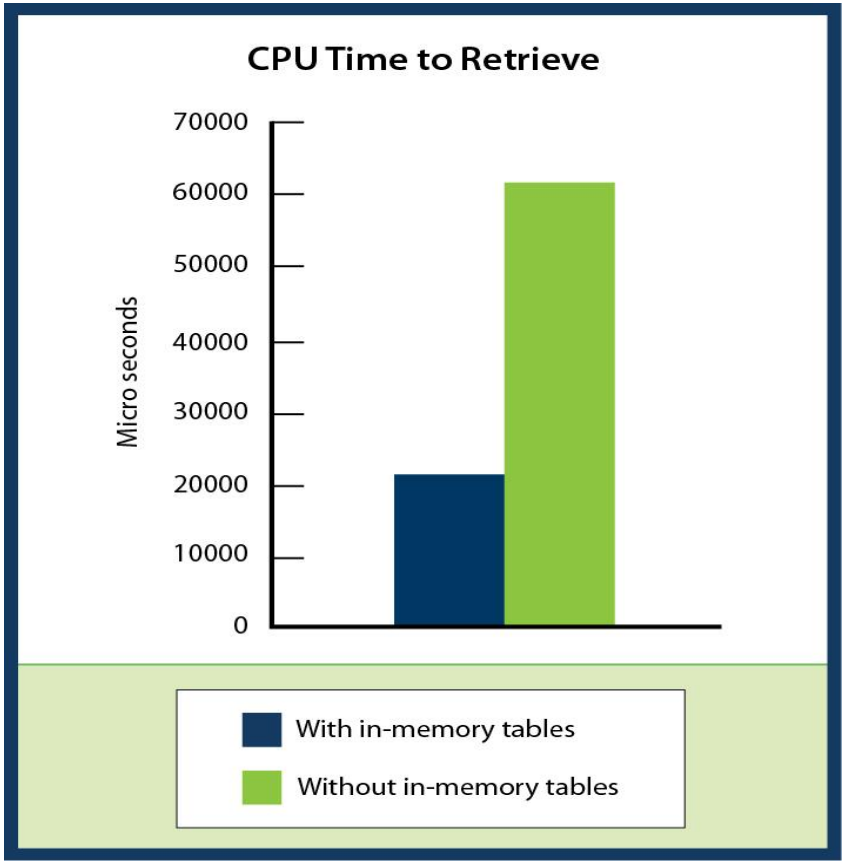
IN-MEMORY some performance results

IN-MEMORY tables perform faster than CMDT

CICS supports CICS managed tables (CMDT) backed by a VSAM data set

For this test:

- **The VSAM data set (of 5000 records for this test) was loaded into CICS managed tables in memory and measurements taken**
- **The same 5000 records were loaded into IN-MEMORY tables and measurements taken**
- **There was no buffering optimization for the VSAM KSDS file.**
- **Note that CMDT retrieves data only by key**



Retrieving data from IN-MEMORY tables is faster then CMDT

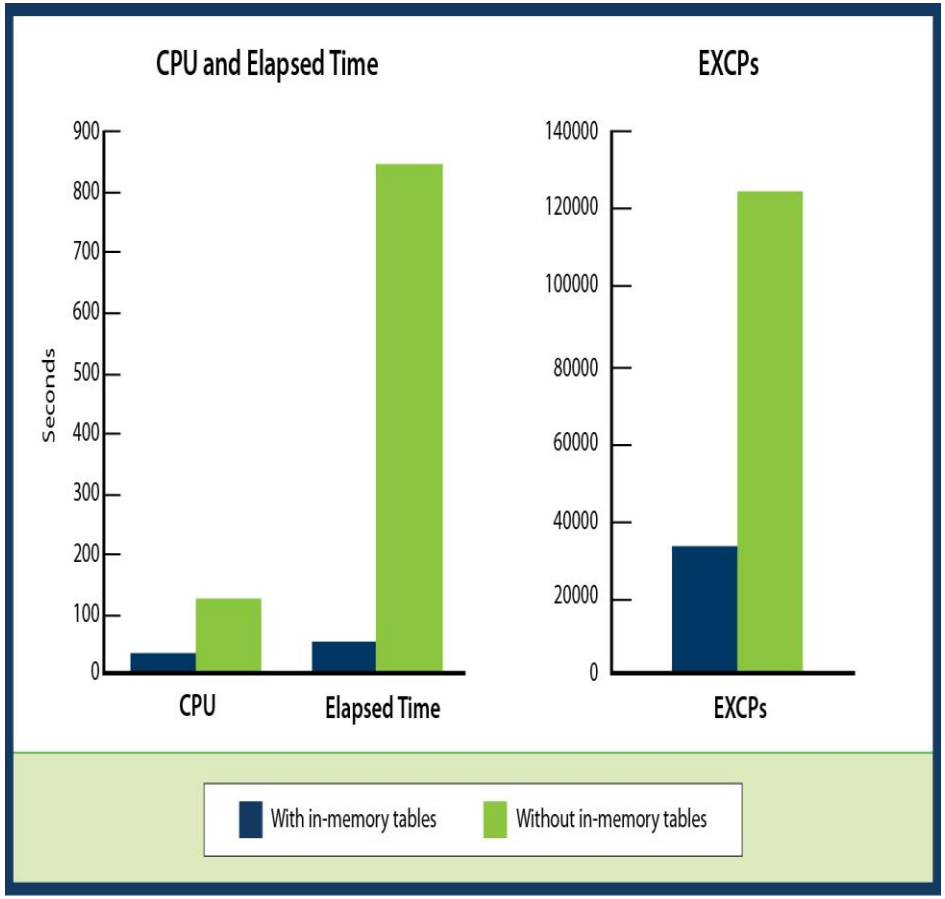
DataKinetics

IN-MEMORY tables perform faster than VSAM

The speed of direct VSAM access was compared with IN-MEMORY access.

For this test:

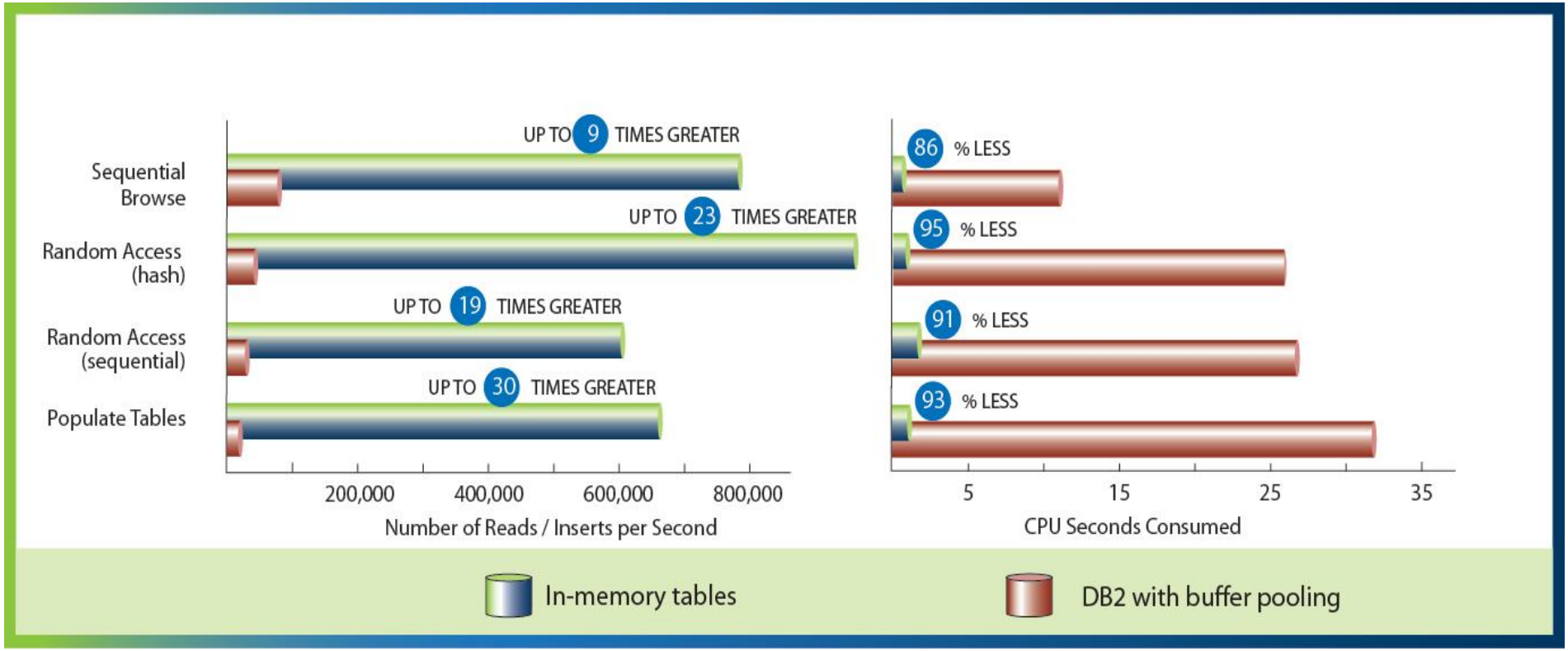
- The VSAM files are KSDS files buffered with Batch LSR. The elapsed time to access the data from IN-MEMORY tables ***includes the time to load the IN-MEMORY tables***
- 999,900 records in VSAM KSDS file
- 999,300 records were retrieved 50 times
- It takes fewer EXCPs to load the same amount of data into IN-MEMORY tables as it does to load the data into VSAM direct access.
- The percentage reduction in elapsed time is more significant as you increase the number of retrievals because the elapsed time includes the time taken to load the IN-MEMORY tables.



The more data retrieved from IN-MEMORY tables, the greater the benefit

DataKinetics

Faster than DB2 buffer pools



DataKinetics

Adapting for IN-MEMORY tables

Example

4 tables joined; two transactional tables and two reference tables

```
SELECT T2.LAST_NAME, T2.FIRST_NAME, T1.BALANCE_DUE, T3.ZIP_CITY, T4.STATE_NAME
        INTO LAST-NAME, FIRST-NAME, BALANCE-DUE ZIP-CITY STATE-NAME
FROM       CUSTACCT T1,
           CUSTINFO T2,
           ZIPLIST T3,
           STATELIST T4
WHERE T1.CUSTID = T2.CUSTID
AND        T2.ZIP5 = T3.ZIP5
AND        T2.STATE = T4.STATE;
```

```
EXEC SQL
SELECT T2.LAST_NAME, T2.FIRST_NAME, TI.BALANCE_DUE, T2.ZIP5, T2.STATE
        INTO LAST-NAME, FIRST-NAME, BALANCE-DUE, T2-ZIP5, T2-STATE
FROM        CUSTACCT T1,
            CUSTINFO T2
WHERE T1.CUSTID=T2.CUSTID
END-SQL
Call API    using TB_PARM
            TB-ZIPLIST-CMDAREA
            TB-ZIPROW
            T2-ZIP5
Move TB-ZIP-CITY to ZIP-CITY
Call API    using TB_PARM
            TB-STATE-CMDAREA
            TB-STATEROW
            T2-STATE
Move TB-STATE-NAME to STATE-NAME
```

Example - Before

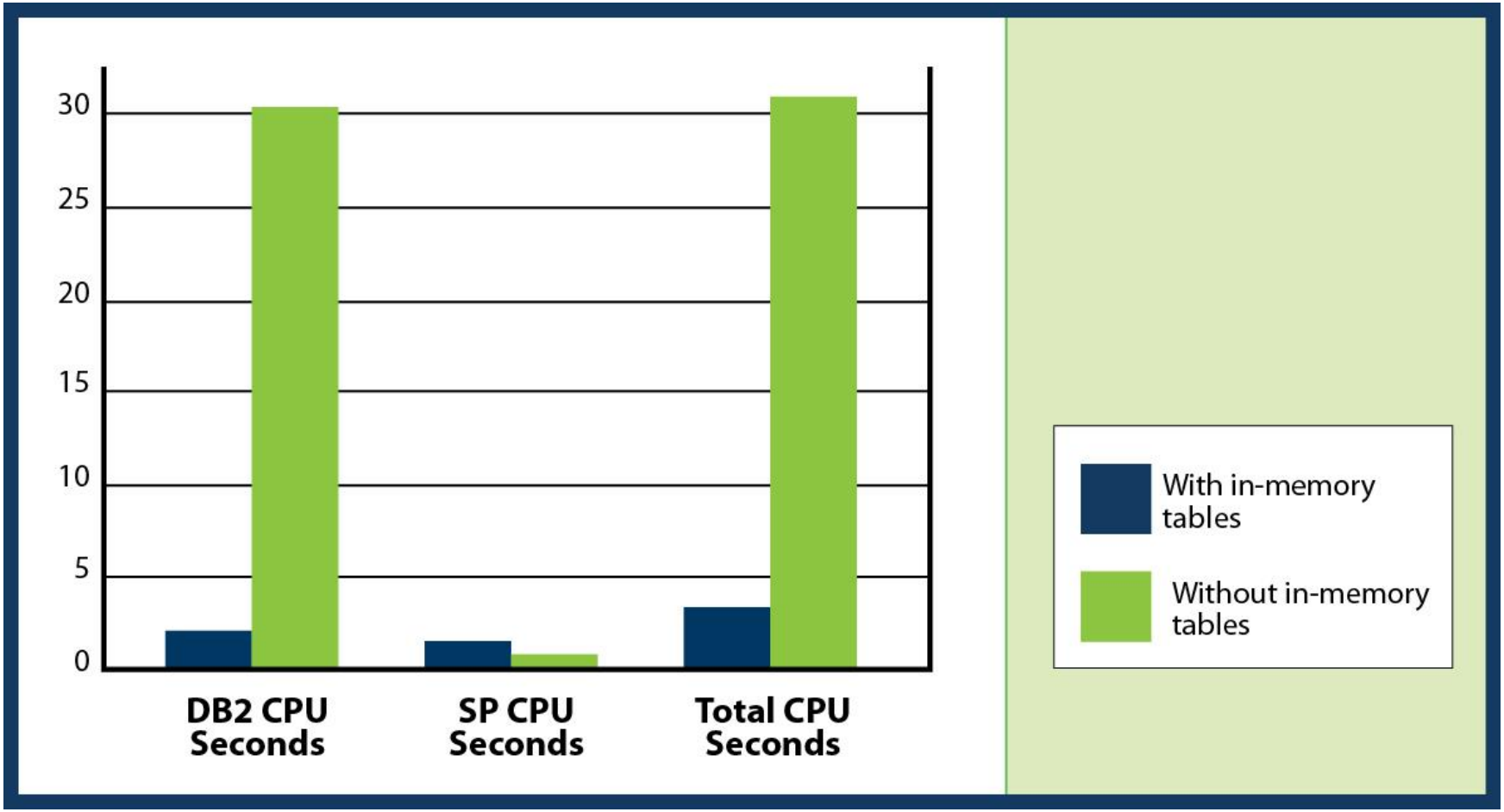
```
EXEC SQL DECLARE CUR01 CURSOR FOR
        SELECT A.ACCT_NBR
        FROM DKLDB001.USB_ACCOUNT A,
             DKLDB001.USB_PRODUCT P
        WHERE A.CLNT_ID = :W-CLNT-ID
        AND A.BNK_NBR = :W-BNK-NBR
        AND A.AGT_NBR = :W-AGT-NBR
        AND A.PRODUCT_ID = P.PRODUCT_ID
        AND A.BNK_NBR = P.BNK_NBR
        AND P.CARD_TYP_CDE = :W-CARD-TYP-CDE
        FOR FETCH ONLY
END-EXEC.
```

Example - After

```
        EXEC SQL DECLARE CUR01 CURSOR FOR
          SELECT ACCT_NBR, CLNT_ID, BNK_NBR, PRODUCT_ID
          FROM DKLDB001.USB_ACCOUNT
          WHERE CLNT_ID = :L-CLNT-ID
            AND BNK_NBR = :L-BNK-NBR
            AND AGT_NBR = :L-AGT-NBR
           FOR FETCH ONLY
        END-EXEC.
*****************************************************
           MOVE PRODUCT-ID      TO  IN-MEMORY-PRODUCT-ID.
           MOVE L-BNK-NBR       TO  IN-MEMORY-BNK-NBR.
           MOVE L-CARD-TYP-CDE  TO  IN-MEMORY-CARD-TYP-CDE.
           CALL 'API'               USING  W-IN-MEMORY-PARM
                                    W-IN-MEMORY-COMMAND-AREA
                                    IN-MEMORY-PRODUCT-REC.
```


DataKinetics

Example – CPU Consumption



DataKinetics.

IN-MEMORY usage

- Replacing CICS tables and TS queues with IN-MEMORY tables.
- Replace hard coded module tables with IN-MEMORY tables.
- Replace VSAM (KSDS and RRDS) files with IN-MEMORY tables.
- Unload high referenced DB2 table data into IN-MEMORY tables.
- Replace BDAM files with IN-MEMORY tables.
- Use IN-MEMORY tables for summarizing and grouping data for reports.
- IN-MEMORY is used for storing cursor results for DB2 cursor scrolling in CICS.
- IN-MEMORY can be used to provide session continuation data for online applications.

How to identify potential candidates

Tools

- SMF records
  - VSAM – SMF64 records
  - Jobs SMF 30 subtype 4
  - IAM – IAM SMF Records - (IAMSMFVS IAM EXCP Report)
- DB2
  - SQL query against SYSTABLES & SYSTABLESPACESTATS
  - Vendor tools, APPTUNE or DETECTOR
- IMS – Change accumulation statistics
- CICS – Shutdown statistics
- Measurement tools
  - STROBE
  - APA
  - Monitors

DataKinetics.

Criteria for identifying tables to be optimized

Item	Criteria
1	The ratio of Reads to Writes needs to be high, generally a ratio of 100:1 is good and 500:1 would be great. This applies to: • DB2 • IMS • VSAM
2	The ratio of Reads to Rows in the table needs to be good
3	The reads per day should be significant
4	Limit table size. *Consider Memory required (real or virtual). Normally smaller tables gain more. Say <2G*
5	GLOBS are generally bad

DataKinetics

Discovery VSAM

Data Set name	JOB NAME	DD NAME	EXCPS	INSERTS	DELETES	UPDATES	RETRIEVALS	# RECORDS	LRECL	GB
DKLFE.#LPS.APOR.DATA	CICSP6P3	APOR	24,883,173	1,123	940	41,308	121,193,241	52,946	600	0.029586
DKLI9.#P1BHI.N1ZXFDR.DATA	ZOODZICA	N1ZXFDR	28,821,494	0	0	0	114,137,907	231,896	4089	0.883101
DKLXW.#ENTRDX.VPF.DK	XWSRVP	SYS00001	2,319,082	95,220	163,653	70	104,925,834	585,381	136	0.074144
DKLI9.#P1BHI.PE01RCR.X00001.DATA	ZOODZIAA	PE01RCRX	26,092,817	10,561	2	0	92,958,864	2,952,090	18	0.049488
DKLI9.#P1BHI.PG15IDF.X00021.DATA	ZOODZIGA	PG15IDFX	32,675,941	97,947	73,731	0	88,488,668	6,452,284	38	0.228348
DKLI9.#P1BHI.E10FRPM.A00015.DATA	ZOODZIAA	E10FRPMA	3,180,727	0	0	0	68,269,497	130,461	8185	0.994488
DKLI9.#P1BHI.E104CIF.A00004.DATA	ZOODZIAA	E104CIFA	5,942,502	0	0	0	63,679,458	173,605	4089	0.661119
DKLI9.#P1BHI.N1ZXFDRX.DATA	ZOODZICA	N1ZXFDRX	24,330,441	17,670	19,696	0	61,776,087	4,364,814	40	0.162602
DKLI9.#P1BHI.E103CIF.A00003.DATA	ZOODZIAA	E103CIFA	4,433,407	0	0	0	60,130,646	124,992	4089	0.475992
DKLI9.#P1BHI.N105AFS.A00005.DATA	ZOODZIGA	N105AFSA	12,132,415	0	0	0	54,528,900	156,271	4089	0.595108
DKLE6.OS.PAYEE.INDEX.CICS.DATA	CE6070RP	HBDD004	79,206	0	0	0	46,876,875	147,422	100	0.01373
DKLI9.#P1BHI.PG0WAIF.X00032.DATA	ZOODZIFA	PG0WAIFX	11,813,441	86,676	0	0	42,542,171	2,882,518	34	0.091275
DKLI9.#P1BHI.N113PAC.A00039.DATA	ZOODZIAA	N113PACA	4,467,981	0	0	0	35,373,809	87,734	10233	0.836125
DKLI9.#P1BHI.PD01RCR.A00001.DATA	ZOODZIAA	PD01RCRA	9,576,590	0	0	0	35,305,591	10,983	4089	0.041825
DKLI9.#P1BHI.E10FRPM.A00015.DATA	ZOODZIBA	E10FRPMA	496,074	0	0	0	34,503,793	130,461	8185	0.994488

DataKinetics

Discovery DB2

TABLE NAME	GETPAGES COUNT	GETPAGES TIME	SYNC I/O COUNT	SYNC I/O TIME	READ EXECUTIONS	NON-READ EXECUTIONS	# OF ROWS	ROW LENGTH
PITBPREF	139,102,000	12:02.4	6590	00:03.2	93,327,028	0	1755	210
T_DELIVERY_CHANNEL	72,922,988	07:11.9	31	00:00.0	41,195,936	0	105	59
PROCESS_LOCATION	39,691,687	03:30.5	10	00:00.0	26,434,286	0	396	21
REGIONS	39,564,353	03:30.6	3	00:00.0	26,376,234	0	108	14
CUSTOM_PHOTO_RULES1	31,777,736	02:22.8	214	00:00.2	15,888,868	0	19	22
CALL_TIMES_TB	21,738,834	02:17.8	1928	00:01.0	12,037,498	0	366	167
PITBSPRI	9,633,519	00:48.8	32	00:00.1	6,422,346	0	8	23
SLCTN_PREFN_DFNTN	7,361,714	00:43.2	65	00:00.1	4,907,787	0	6	145
PITBDKLC	15,389,568	01:10.0	2602	00:01.2	3,080,381	0	53	250
PROD_CORP	18,696,038	00:54.8	14006	00:06.4	1,696,716	0	487	102
TEST_TB	5,205,402	00:13.1	634	00:00.3	1,671,653	0	727	45
ACCT_RELCODE_PROCESS	2,833,120	00:10.0	14079	00:06.2	1,416,667	0	2	32
NBTBTRN	1,301,645	00:02.3	176	00:00.1	635,822	0	34	75
CARD_STATUS_TB	10,275,312	00:36.6	612	00:00.3	619,848	0	11	38
PITBCHGC	2,949,616	00:05.5	13	00:00.0	589,925	0	262	58

DataKinetics

Discovery IMS

Date	DBD Type	DL/I Total	DL/I Reads	DL/I Updates	GU Count	GN Count	ISRT Count	REPL Count	DLET Count	Other Count	Tran Count
10/23/2012	DATA1	202,585,821	202,472,358	0	202,472,358	0	0	0	0	113,463	214,004
10/23/2012	DATA2	38,898,112	38,897,313	799	25,266,537	13,630,776	0	799	0	0	962,327
10/23/2012	DKL1	38,742,393	38,658,675	83,718	19,313,741	19,344,934	76,692	0	7,026	0	17,402,470
10/23/2012	DKL2	28,702,133	28,701,975	158	15,341,058	13,360,917	46	108	4	0	3,828,475
10/23/2012	DKL3	17,845,734	0	0	0	0	0	0	0	17,845,734	8,922,867
10/23/2012	DKL4	7,072,987	132	22	132	0	0	22	0	7,072,833	3,047,122
10/23/2012	DKL5	6,953,822	132	22	132	0	0	22	0	6,953,668	2,994,085
10/23/2012	DKL6	6,846,180	132	22	132	0	0	22	0	6,846,026	3,020,075
10/23/2012	DATA5	6,708,378	6,613,242	95,136	6,613,242	0	1,157	93,979	0	0	648,575
10/23/2012	DKL8	6,546,443	132	22	132	0	0	22	0	6,546,289	2,869,934
10/23/2012	DKL9	6,519,354	1,821,462	4,697,892	1,821,194	268	4,697,852	0	40	0	320
10/23/2012	DKL10	6,478,828	132	22	132	0	0	22	0	6,478,674	2,865,231
10/23/2012	DKL11	6,219,972	6,153,359	66,613	3,065,342	3,088,017	46,356	19,754	503	0	1,051,683

DataKinetics

Summary

Top 5 features of IN-MEMORY tables

- Shortest path
- Optimize search
- Implicit commands
- Dynamic indexes
- Flexible API

Results that IN-MEMORY tables can provide

With in-memory tables you can:	Decrease MSU	Decrease elapsed time	Increase flexibility and market adaptation	Reduce maintenance	Enable new paradigms
Place reference data in tables	Yes	Yes			
Replace temporary files with temporary tables	Yes	Yes			Yes
Use tables for rules			Yes	Yes	
Use tables as a message queue	Yes	Yes			Yes
Use decision tables to replace logic trees	Often		Yes	Yes	
Use tables for process control			Yes	Yes	
Use temporary tables for implementing complex algorithms	Yes	Yes	Yes	Yes	Yes

DataKinetics


DataKinetics
Larry Strickland
Chief Product Officer
lstrickland@dkl.com