# In-memory Tables
Technology overview and solutions

Verna Bartlett
Head of Marketing

Gary Weinhold
Systems Analyst

DataKinetics®

## Agenda

- **Introduction to in-memory tables**
- **What data works best in in-memory tables**
- **What you can do with in-memory tables**
- **The details of how in-memory tables can achieve its speed**
- **How in-memory tables solves critical business challenges**
  - Provide consolidated customer statements
  - Create a market adaptive application for credit card offerings
  - Provide capacity and scale to service ever increasing volume of transactions
  - Increase capacity of batch window and meet contractual SLAs
- **Summary**

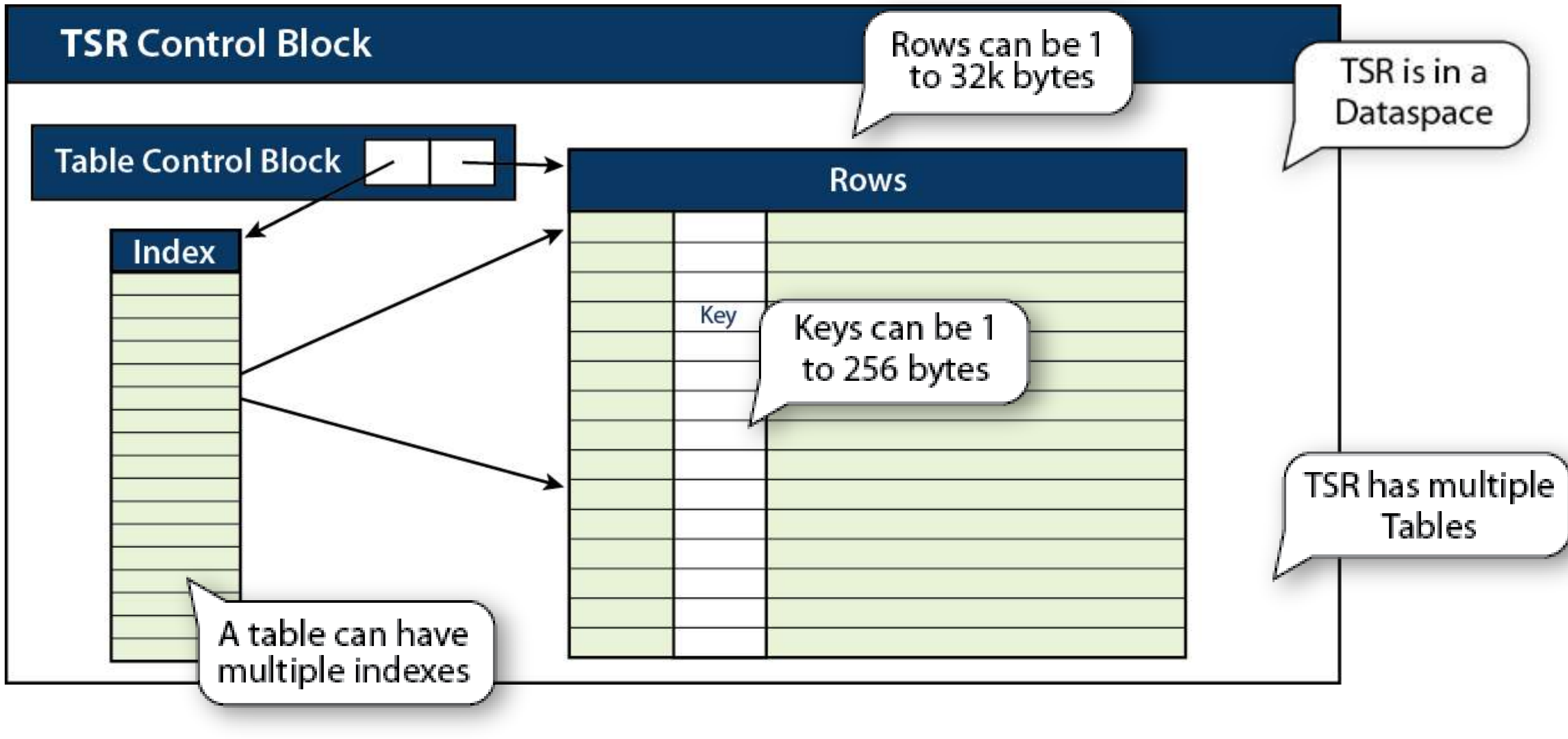
DataKinetics.

## Introduction to in-memory tables

## What is an in-memory table

Data Space in memory for table processing

- In-memory tables are kept in a Table Share Region (TSR)
- Tables are made up of Rows, Keys, Index, Organization and Search Methods



DataKinetics.

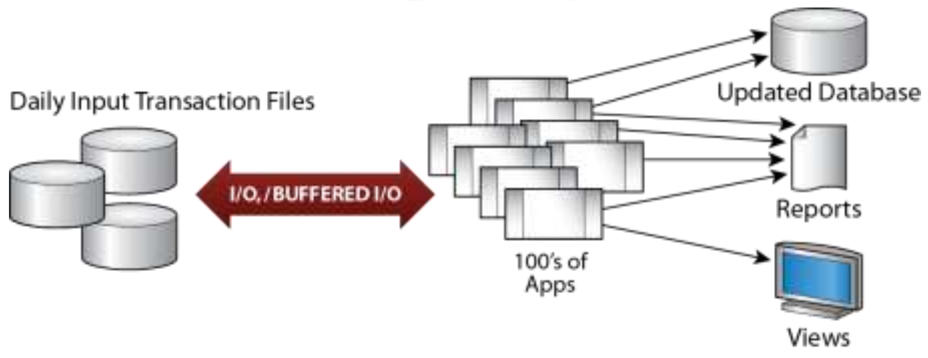
# In-memory tables

## Options for in-memory tables

- Define, autoload and index tables
- Tables have a fixed row length
- Each row contains a key and structured data of variable format
  - Key can be multiple fields
  - Data can be values, instructions, locations, rules or decisions
- Group multiple rows from different tables with different formats into a single table
- Create alternate indices on the fly
  - Use alternate indices as a virtual sort
- Optimize table search
  - Options for table organization and table search

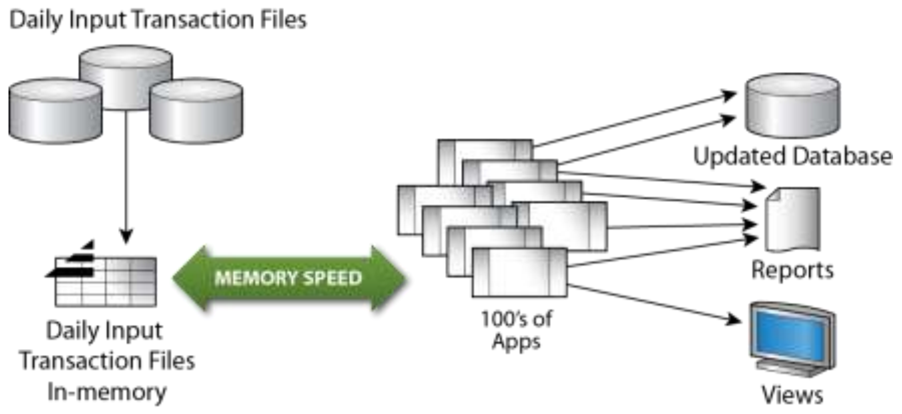
## How in-memory tables reduces elapsed time

### Access data @ I/O speed



- Each request for data can result in one or more I/Os
- IMS or DB2 can buffer the data, but it still needs to be reformatted
- IMS or DB2 brings back a block, data is extracted and reformatted

### Access optimized @ memory speed

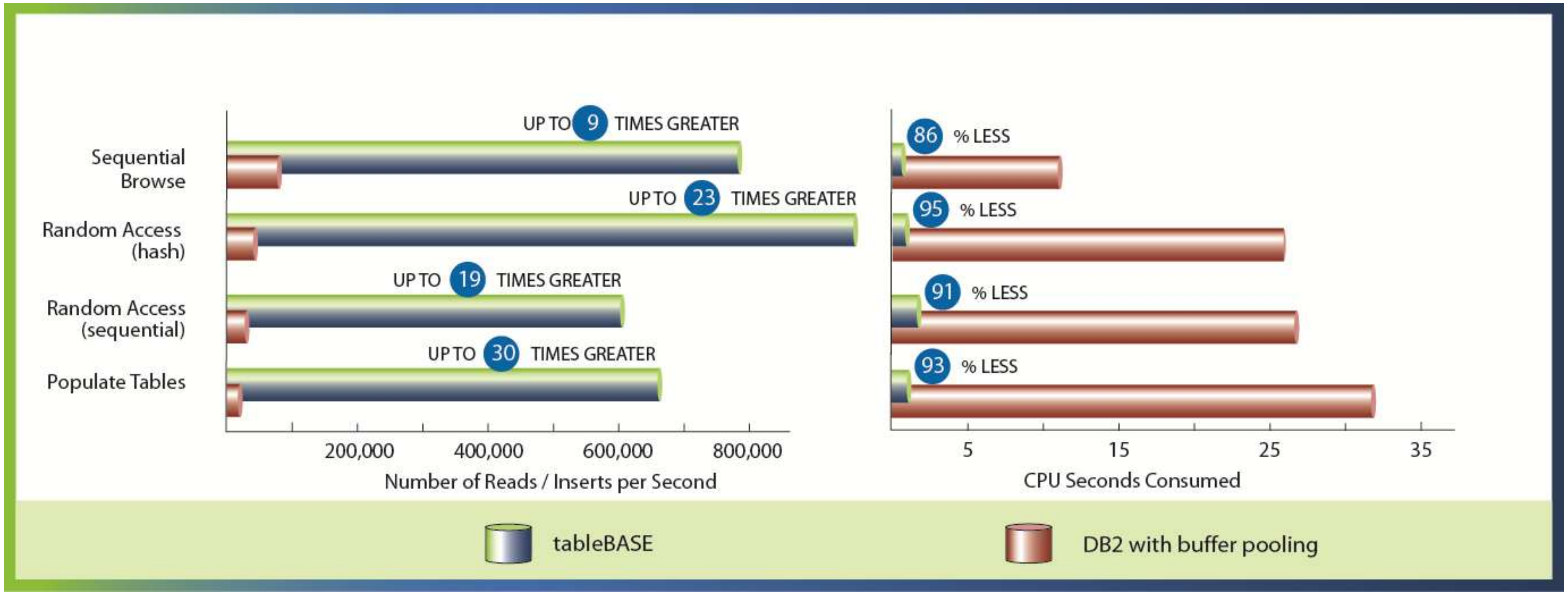

Daily Input Transaction Files
MEMORY SPEED
Daily Input Transaction Files In-memory
100's of Apps
Updated Database
Reports
Views

- First call for data loads the table, creates index
- All other calls access in-memory tables
- Returns entire row
- Create alternate indices on the fly
- Virtual sorts using alternate indices
- Optimize search method
- No DBMS or OS overhead

**With tableEXTENZ the path to data is shorter and much, much faster**

## Performance of in-memory tables

- Shortest, fastest possible path to data
- Use in-memory tables with a DBMS to offload read-only I/Os and replace creation of temporary files
- When the amount of time taken for each transaction is reduced, and the number of transactions per unit of time is increased, performance and capacity are increased significantly

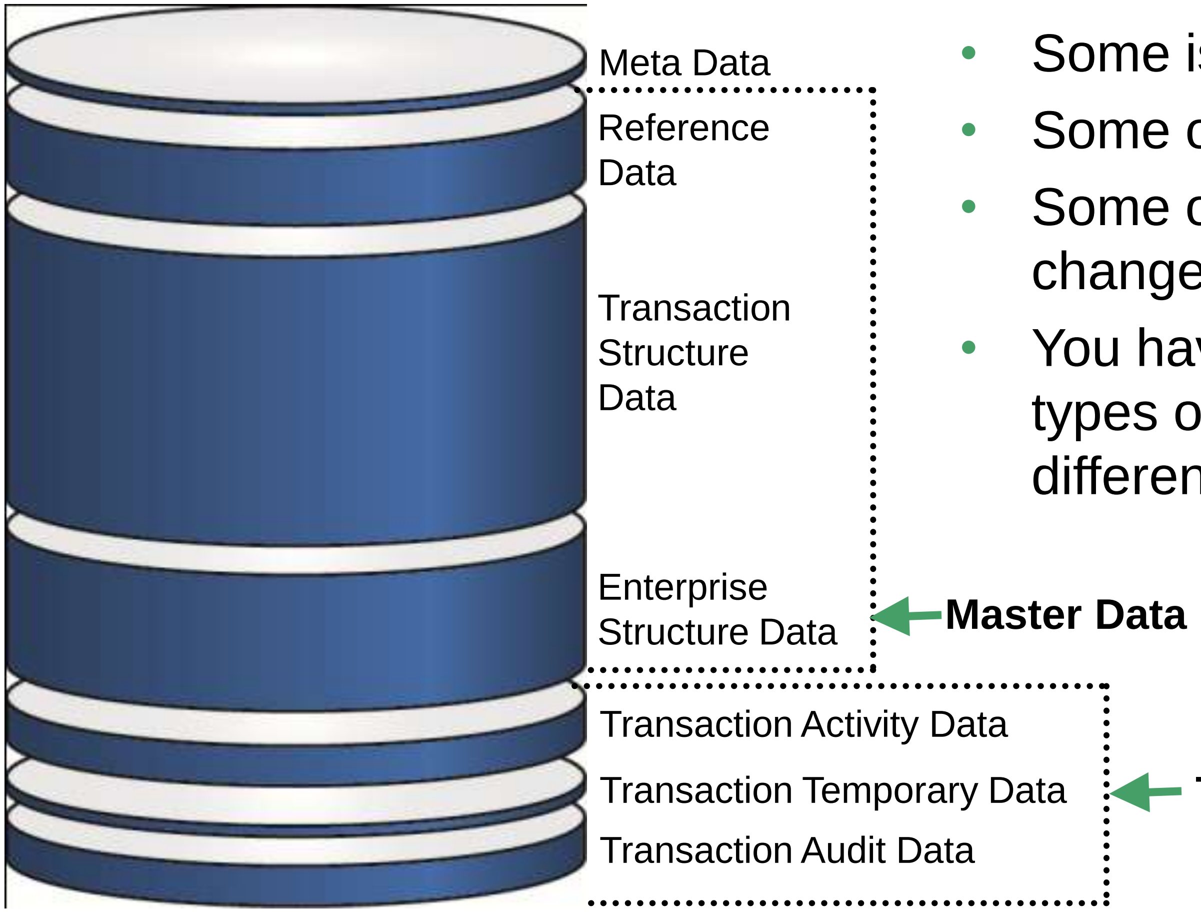


DataKinetics

## What data works best for in-memory tables

## The data just keeps growing!

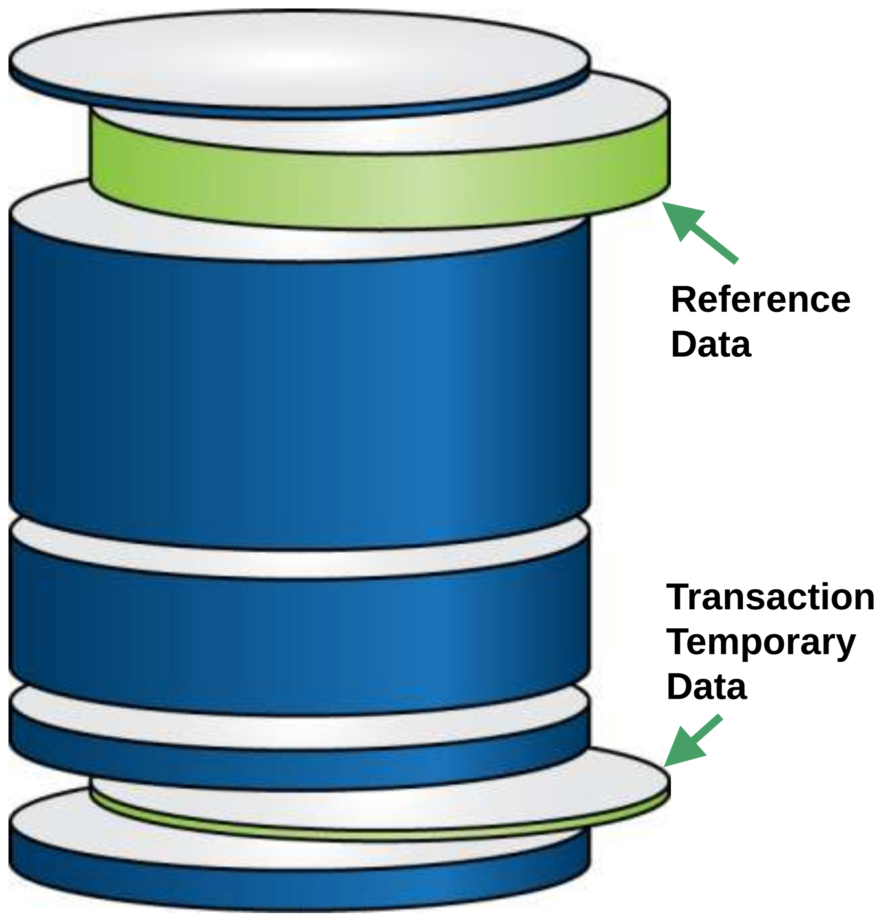
**Your enterprise data**


Meta Data
Reference Data
Transaction Structure Data
Enterprise Structure Data
Master Data
Transaction Activity Data
Transaction Temporary Data
Transaction Audit Data
Transaction Data

**Your enterprise has many different kinds of data:**

- Some is used for transactions
- Some of it is temporary
- Some of it changes often, some changes infrequently
- You have all these different types of data, but you may have different naming conventions.

## 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

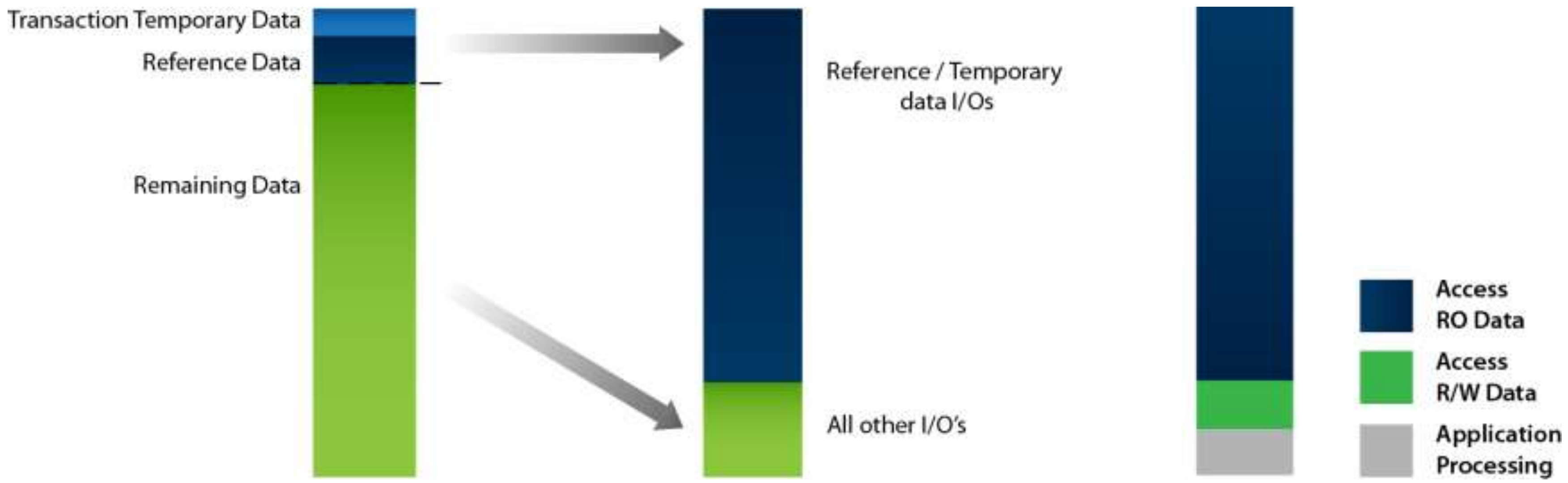
- Is created, processed and then deleted
- Generates a high volume of data accesses for the volume of data

### Remaining data

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

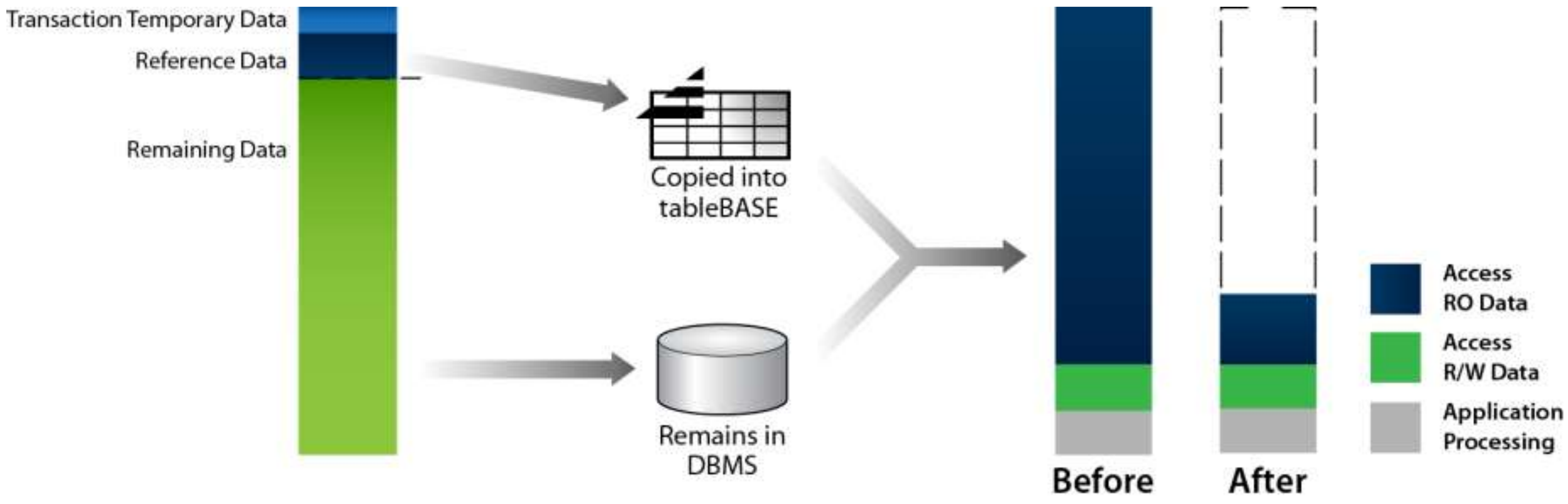
DataKinetics.

# A small proportion of your data generates the most I/Os



**Between 5-15 % of your total data** → **Generates 80% of your I/Os** → **Takes time and CPU**

# How to optimize your reference and temporary data



1. Identify the highly accessed reference data

2. Put it into table BASE

3. Reduce I/Os and elapsed time by 85%
4. Reduce CPU by 65%

DataKinetics

## What is reference data?

**Data that is used to validate transactions, or data that is looked-up as part of a transaction, or that categorizes other data**

- As part of credit card transactions – name, address, credit card number, expiry date
- As part of settlement – vendor name, address, account information; purchaser name, address, account number, bank information
- Price tables
- Lists of cities, states and countries
- Rate tables, which may be differ depending on geographic location
- Tax tables
- Product or part numbers
- SIC classification

## What is temporary data?

Data that is collected often from multiple sources, usually sorted, and then used as inputs for subsequent applications – which may be a printing program

- Consolidating financial information from the various accounts a customer may have to create a consolidated statement
- Calculating instantaneous net worth

## What you can do with in-memory tables

## Barriers to performance and flexibility

- **What slows data down?**
  - Moving data from disk to memory and back again
  - Repeated retrieval of identical information
  - Looping serially through information to find the piece you want
  - Waiting for one transaction to complete before starting another

- **What slows down market adaptiveness?**
  - Relearning complex logic in order to enhance or repair
    - Rules embedded in program logic
    - Logic trees embedded in program logic
  - Updates that need to be propagated to many programs
  - Comprehensive testing cycles

## Optimize Access to Your Data

Optimize access to your data

- Copy your Reference Data into in-memory tables,
  - Access the data from there to minimize I/O and CPU resource usage
- Use in-memory tables to store your Temporary Data
  - Avoid unnecessary and wasteful I/O access
  - Any I/O you can save is a good I/O
- Access the rest of your data directly from the DBMS

DataKinetics.

# Temporary tables

## Efficient use of temporary tables

- Temporary tables can be defined and used in lieu of DASD temporary tables
- Temporary processing is done in memory and the table is deleted at the end of the processing
- Can be shared between two or more programs
- Can be shared between two or more transactions

## Build more powerful applications

**Build more powerful applications using table-drive design techniques, using in-memory tables:**

- Business rules contained in DBMS tables are easy enough to maintain, but have performance issues, as every transaction experiences many I/O accesses to the disk-based rules.
- Business rules contained in application code run very fast, but are difficult to maintain— change control involves recompiles, and redeployment, and inefficient use and over-use of programming staff
- Business rules contained in in-memory tables process very fast and are easily maintained— changes to tables and new tables can be managed by non-technical staff.

## Capabilities of in-memory tables

- Define, build, maintain and manage in-memory tables
- Assemble data from multiple sources into a temporary table for subsequent processing or renderings
- Optimize table search
- Place reference data and rules in read-only tables
- Replace sort with a virtual sort using alternate indices that can be defined on the fly
- Replace logic trees with decision tables
- Use tables as a write through cache or as message queues
- Share tables between applications
- Applications simultaneously switched to access new data

## 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.

## Where can in-memory tables be used?

| Challenges | Results |
|---|---|
| **Performance issues** | • Improved efficiency & performance of existing applications<br>• Eliminated the cost, complexity, risk of hardware upgrade / migration plans |
| **Application complexity** | • Reduced their application support and maintenance costs<br>• Redeployed their people to revenue-generating tasks |
| **Adding new services is painful** | • Enhances  ability to add new services quickly w/o additional cost<br>• Turn around time is dramatically reduced |
| **M&A activity** | • Integrated operations with success<br>• Reduced costs by maximizing existing investments |

**Improve operational efficiency while reducing costs**

DataKinetics

## The details of how in-memory tables can achieve its speed

# Designed for performance

- **Does not do metadata translation on a field by field basis**
- **Avoids OS to do access, eg, link command, or I/O**
- **Avoid getmains**
- **Avoid locking**
- **Efficient algorithms for search**
  - Choose the search that is the most efficient for the data
- **Returns entire rows or portions thereof**
- **Uses implicit commands to reduce changes to calling application**
  - Application asks for row, and if table not open tableBASE opens the table – avoids explicit changes to application to first open and then find row
- **Uses shortcuts to reduce path to data**
  - Searches list of tables first time, and next time, uses a shortcut so doesn’t have to search again
- **Allows dynamic creation of indexes;  can populate and then create index, can create multiple indices after population of tables**

DataKinetics.

## Other advantages of In-memory tables

- **Efficient use of temporary tables**
  - Temporary tables- can be shared by multiple programs
- **Row addressing efficiency**
  - Get next 1, get next n, get last n, etc. All built in; no special structures/set up required
- **Multiple search strategies**
  - Dynamically switch between binary/serial/hash search; by programmer, no help from DBA needed
- **Multiple dynamic indexing**
  - Dynamically add new indexes; multiple indexes for one table; no help from DBA needed
- **Indirect opens**
  - Abstracts out the data allowing simple indirect references to tables/groups of data; tables designed by programmer, no help from DBA needed
- **Date-sensitive processing**
  - Data based on based on effective dates can be automatically selected using a built-in function rather than by program logic; no need for WHERE clause, can be used by any tableBASE app.

DataKinetics.

## Multiple search strategies

**Multiple search strategies:**

- Can dynamically switch between binary, serial and hash search
- This can be done by the programmer on the fly
- No need for DBA involvement to analyze tables, build new indexes and deploy.

## Row addressing

**Rich command set with positional addressing of rows:**

- Insert, update and retrieval commands that work by position in a row
  - Insert by count, replace by count
  - Fetch by count
  - Get next, get next n
  - Get previous, previous n
  - Last, last n
  - First, first n
- Useful especially for processing that involves scrolling of rows in a table
- Requires no programmatic set up

## Multiple dynamic indexing

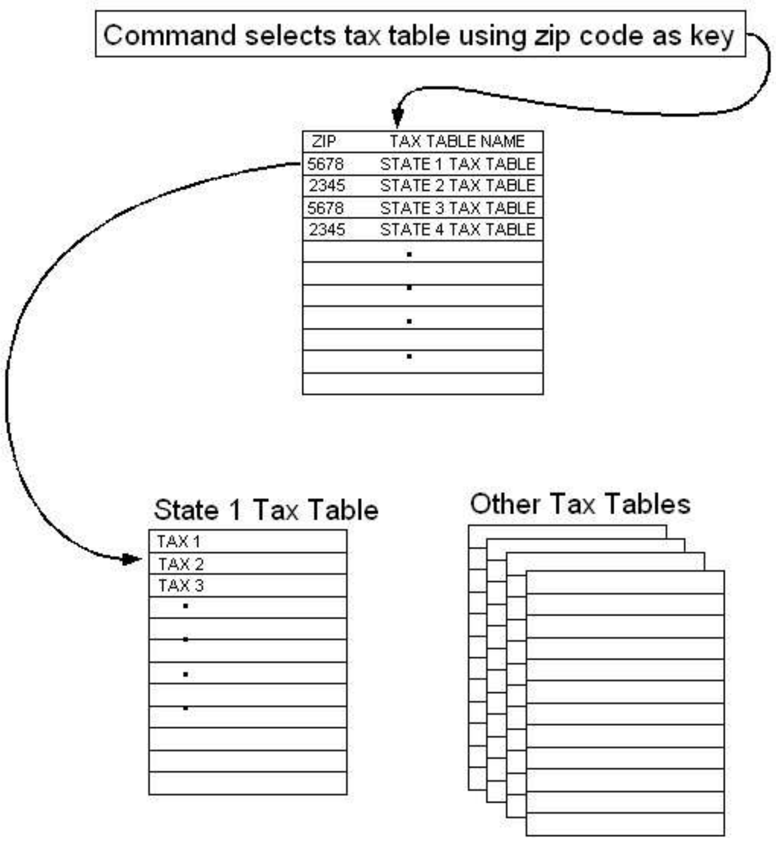
**Multiple dynamic indexing**

- Can dynamically create new indexes for a table
- Can be multiple indexes for a table at one time
- No need for DBA involvement to analyze tables, build new indexes and deploy

## Simplifies, better manages generational / time-based data

**Indirect open processing :**

- A built-in feature of tableBASE
- Example:
  - Two or more price tables exist
  - Another table contains the names of the price tables
  - Zip code is used as a key to select the correct tax table
  - tableBASE checks the key table name, and passes back the name of the correct table
  - The correct tax table is opened indirectly
  - You don't need program logic (or extra columns) to select the correct table
- Tables can be designed by the programmer.
- No need for DBA to define tables, no need for complex SQL design.



## Simplifies, better manages generational / time-based data

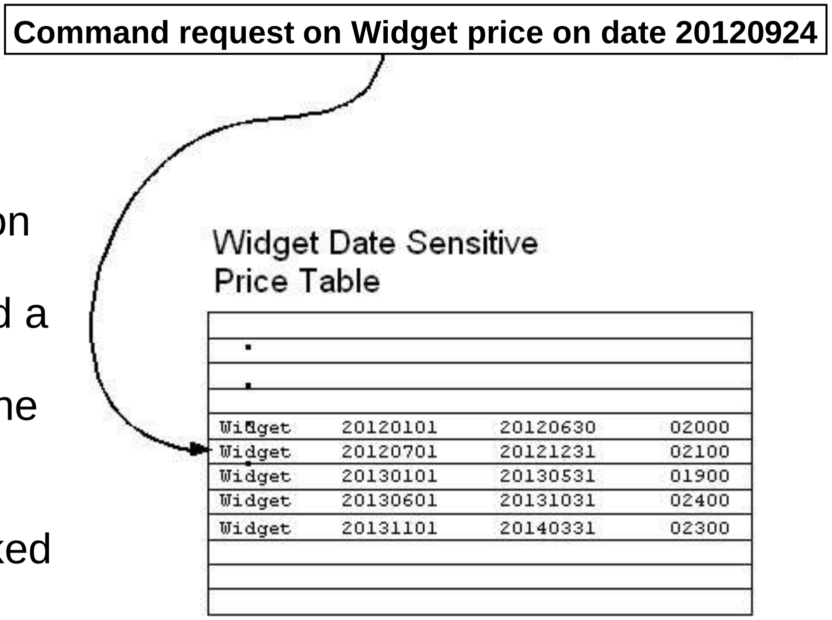
**Date sensitive processing :**

- A built-in feature of tableBASE

- Example:
  - Normally a table is searched via a match on the key
  - In this case, you can match on the key and a date
  - Date is a match on the date range within the price table
  - All with a single command
  - Using same technique, price can be checked for last month, two years ago, etc.
  - You don't need program logic (or extra columns) to obtain the date-sensitive pricing data

- Access to table of this type is built in to tableBASE- no need for WHERE clause.

- A program can access either a date-sensitive table or a non-date-sensitive table *without change*.



**Command request on Widget price on date 20120924**

Widget Date Sensitive Price Table

| | | | |
|---|---|---|---|
| | | | |
| . | | | |
| . | | | |
| Widget | 20120101 | 20120630 | 02000 |
| Widget | 20120701 | 20121231 | 02100 |
| Widget | 20130101 | 20130531 | 01900 |
| Widget | 20130601 | 20131031 | 02400 |
| Widget | 20131101 | 20140331 | 02300 |
| | | | |
| | | | |
| | | | |

## Why in-memory tables are such a good match for optimizing batch applications

DataKinetics

## Key attributes of batch processing

- **Process large volumes, or batches, of instructions or files sequentially**
- **A large volume of work requires repetitive actions or repetitive logic**
- **Jobs often queued, with the outputs of one job providing the inputs for the next job**
- **Often have repeated reads of static data**
- **May create temporary files as part of the data record processing**
- **Batch jobs do a lot with reading, writing and sorting sequential and VSAM files**
- **Manual intervention or inputs not required**
- **Runs automatically once a batch job begins, it continues until it is done or until an error occurs**
- **Traditionally there has been a batch window – a period of time when there would not be any OLTP, and batch jobs would run uninterrupted**
  - Data for OLTP would then always be current

# Options to address batch window compression

- **Scheduling solutions**
- **More hardware**
- **Use grid workflows**
- **Application re-architecture or optimization or DBMS optimization**
- **Run batch and OLTP concurrently**
  - May reduce performance of OLTP (as seen at Home Depot)
  - May be some challenges in accessing the data concurrently
    - Lockouts
    - Data not current
- **Move data into memory – caching**
- **DataKinetics in-memory solution**

***Many of these solutions require hardware, and ongoing monitoring and optimization of the processes.***

## One of IBMs recommendations...

- **“Normally batch programs process a huge amount of data. Thereby often, depending on the business logic, some data is static and does not change during the program run (for example company address information).**
- **It is recommended to read such data only once from the database and cache it somewhere for further processing. This will prevent your system from running unnecessary round trips to the database.**
- **Eliminate repeated reads of the same data”**

**BUT**

- **“Implementing data in memory (DIM) techniques is a complex task”**

***Whereas implementing in-memory tables is not a complex task.***

From IBM Redbook on Batch Modernization on z/OS, December 2009

## The easiest solution to implement

| Solution to batch window compression | Hardware | Software | Ongoing monitoring and optimization | Code changes | Complexity |
|---|---|---|---|---|---|
| **Scheduling solutions** | No* | Yes | Yes | No* | Medium |
| **More MIPS** | Yes | Licensing* | No change | None | Low |
| **Grid workflows** | Yes | Yes | Yes | Some | Medium to High |
| **Application re-architecture, optimization** | No* | No* | No* | Could be extensive | Medium to High |
| **DBMS optimization** | No* | Yes** | Yes | Yes | Medium |
| **Run batch and OLTP concurrently** | Maybe*** | Maybe*** | Increased | Maybe*** | Medium |
| **IBMs data in memory (DIM) solution (caching)** | Spare memory and spare CPU | Yes | Yes | Could be extensive | High |
| **DataKinetics in-memory solution** | No | Yes | No | Minor | Low |

* Typical  ** DBMS optimization tools are needed in many solutions  *** Depends on specific environment and specific solution

## Why in-memory tables are such a good solution for batch processing

- *Process large volumes*, or batches, of instructions or files sequentially
- A large volume of work requires *repetitive actions or repetitive logic*
- Jobs often queued, with the *outputs of one job providing the inputs for the next job*
- Often have *repeated reads of static data*
- May *create temporary files* as part of the transaction processing
- Batch jobs do a lot with *reading*, writing and *sorting* sequential and VSAM files
- Manual intervention or inputs not required
- Runs automatically once a batch job begins, it continues until it is done or until an error occurs
- Traditionally there has been a batch window – a period of time when there would not be any OLTP, and batch jobs would run uninterrupted
  - Data for OLTP would then always be current

***These are the areas where in-memory tables excels at improving performance***

DataKinetics.

## Typical results of customer implementations

Optimize data access in batch processing

Provide consolidated customer statements

Provide capacity and scale to service ever increasing volume of transactions

## Pressures on the batch window

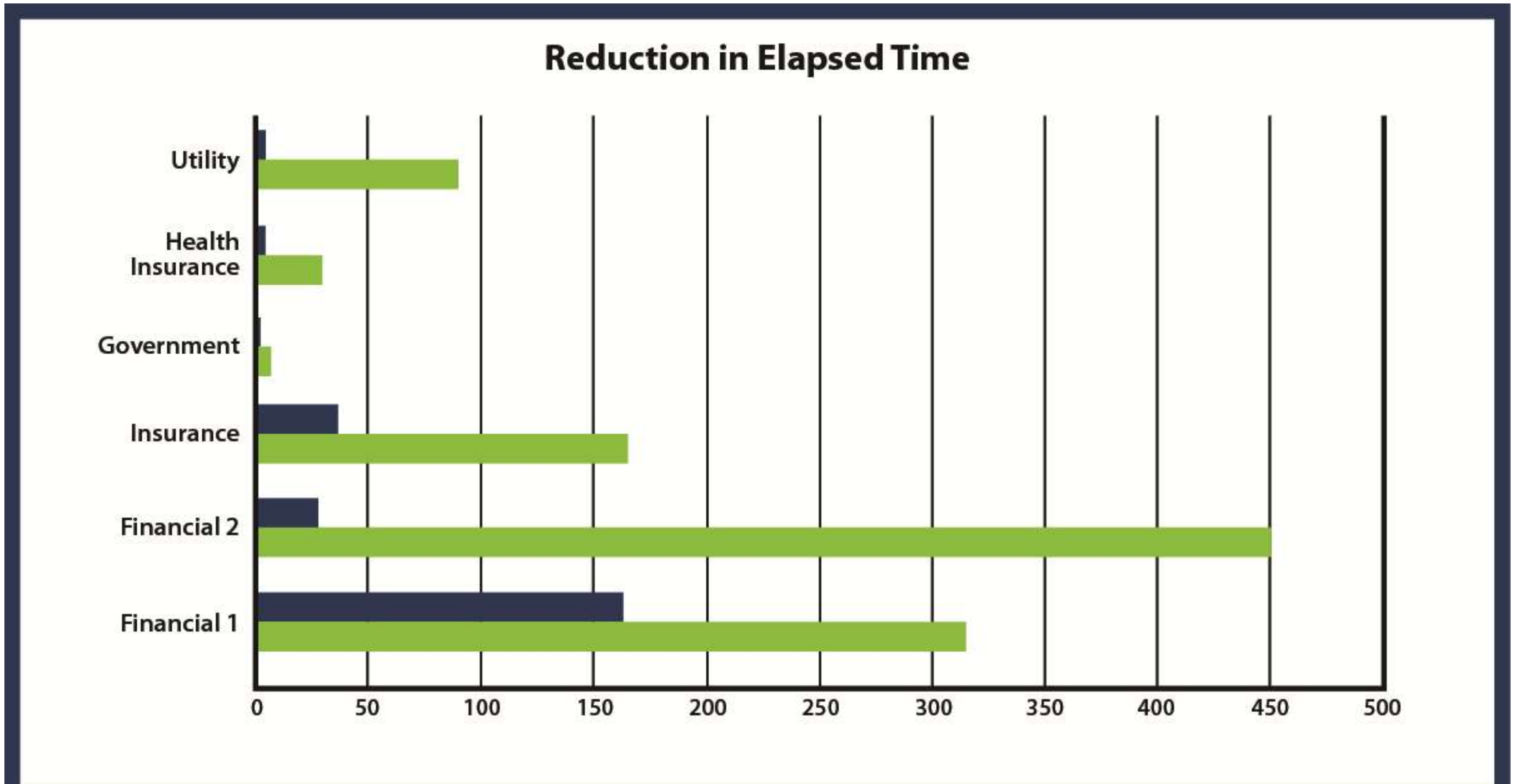
**In today's web world, demands for 24/7 OLTP constantly increasing**

- Global business
- Ecommerce
- Customers demand access to services such as banking 24/7

**Batch processing work load keeps increasing as business changes**

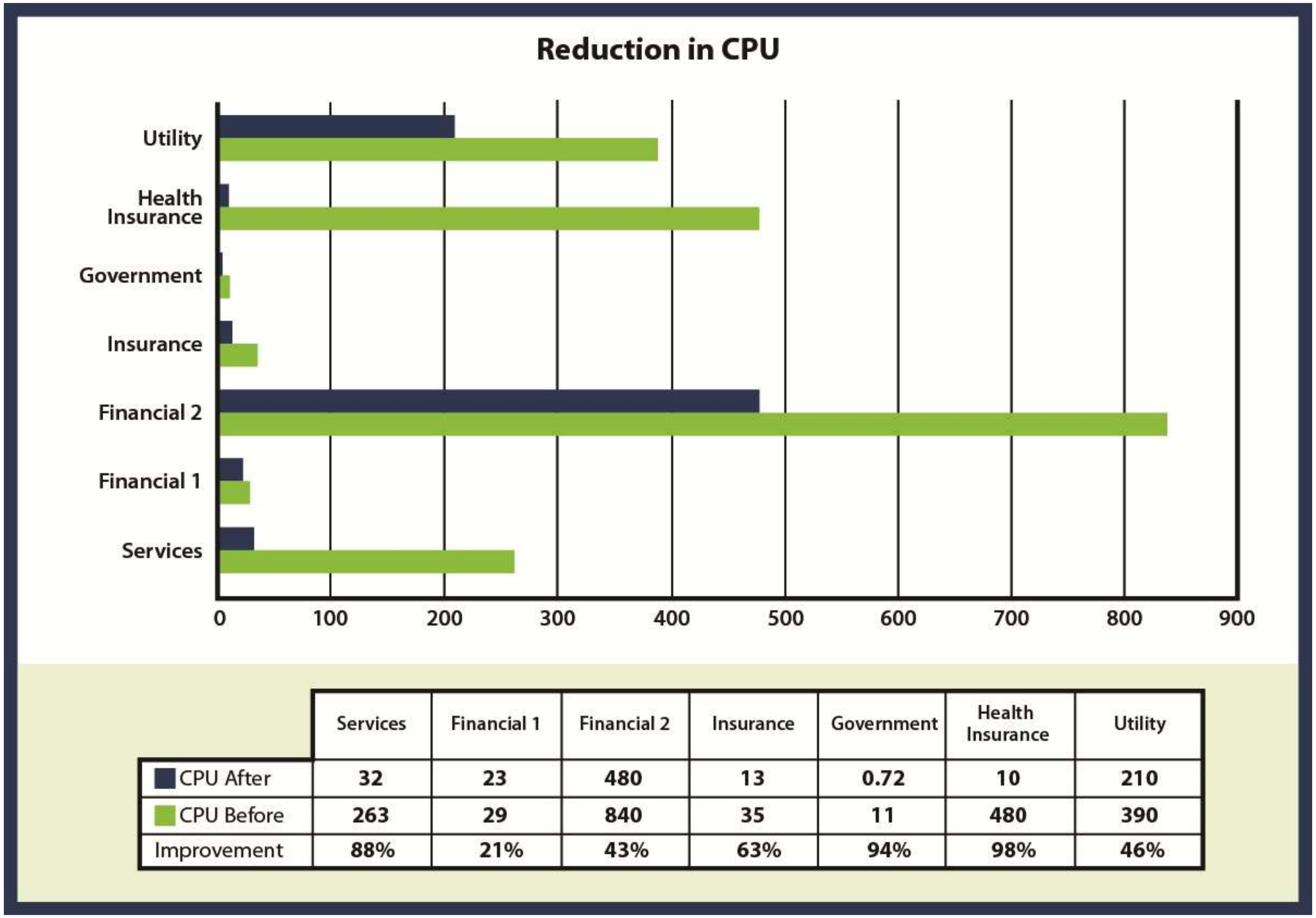
- The need to handle larger volumes of data
- The need to incorporate additional functions
- The need to integrate corporate acquisitions

## Typical results from some of our customers



| | Financial 1 | Financial 2 | Insurance | Government | Health Insurance | Utility |
|---|---|---|---|---|---|---|
| ■ Elapsed time After | 163 | 28 | 37 | 0.11 | 5 | 5 |
| 🟩 Elapsed time Before | 315 | 450 | 165 | 7 | 30 | 90 |
| Improvement | 48% | 94% | 78% | 98% | 83% | 94% |

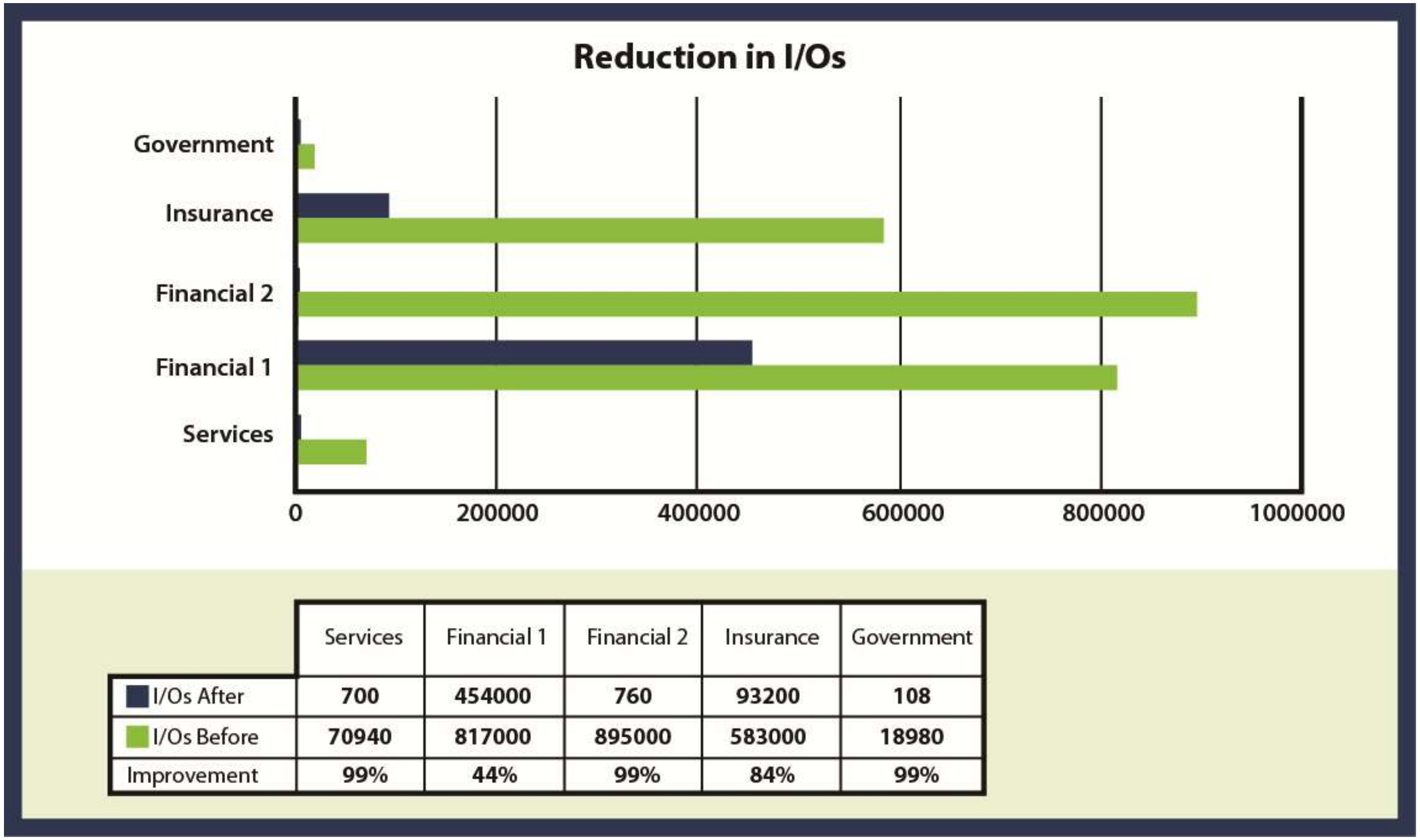
# Typical customer results - CPU



| | Services | Financial 1 | Financial 2 | Insurance | Government | Health Insurance | Utility |
|---|---|---|---|---|---|---|---|
| ■ CPU After | 32 | 23 | 480 | 13 | 0.72 | 10 | 210 |
| 🟩 CPU Before | 263 | 29 | 840 | 35 | 11 | 480 | 390 |
| Improvement | 88% | 21% | 43% | 63% | 94% | 98% | 46% |

DataKinetics.

## More average results from some of our customers – I/Os



| | Services | Financial 1 | Financial 2 | Insurance | Government |
|---|---|---|---|---|---|
| ■ I/Os After | 700 | 454000 | 760 | 93200 | 108 |
| 🟩 I/Os Before | 70940 | 817000 | 895000 | 583000 | 18980 |
| Improvement | 99% | 44% | 99% | 84% | 99% |

DataKinetics.

## Typical reductions

| Customer | CPU | Elapsed Time | I/Os |
|---|---|---|---|
| **Services** | 88% | | 99% |
| **Financial 1** | 21% | 48% | 44% |
| **Financial 2** | 43% | 94% | 99.9% |
| **Insurance** | 63% | 78% | 84% |
| **Government** | 93% | 98% | 99% |
| **Heath Insurance** | 98% | 83% | |
| **Utility** | 46% | 94% | |
| **Average** | **65%** | **83%** | **85%** |

# Consolidated customer statements - challenge

*Challenge* **Produce a single customer statement for all their financial products, provide a summary of their net worth, and promote those products the customer does not have**

- When each financial product that the customer uses resides in a different data base and is accessed by a different application, consolidating that information takes considerable CPU and elapsed time
- Creating a statement of net worth requires integrating the information after consolidating it
- Identifying what products the customer does not have and promoting those products requires more processing

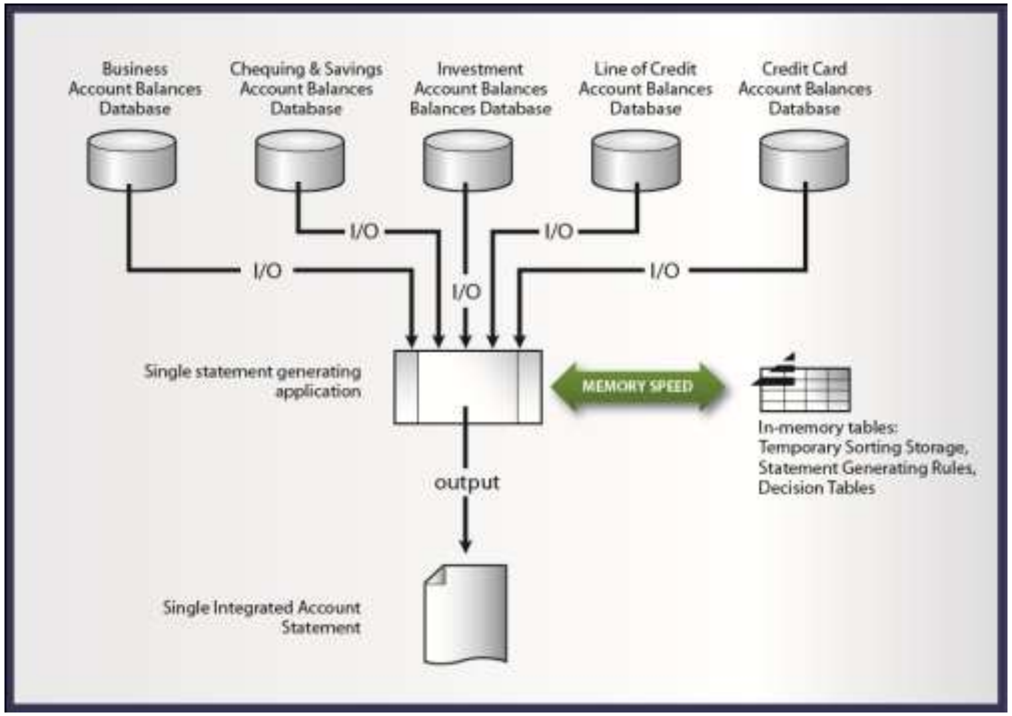
DataKinetics

## Consolidated customer statements - solution

***Solution***

- Populate account consolidation table in memory with account data
- Render contents via virtual sorts and using formatting tables (rules table)
- Provide a summary of their net worth at that moment taken from consolidation table which is summarized in the table
- Based on what services the customer has and the customer profile taken from consolidation table, provide customized offers for new services – using decision tables

***Do it all in tables***


Business Account Balances Database
Chequing & Savings Account Balances Database
Investment Account Balances Balances Database
Line of Credit Account Balances Database
Credit Card Account Balances Database
I/O
I/O
I/O
I/O
I/O
Single statement generating application
MEMORY SPEED
In-memory tables: Temporary Sorting Storage, Statement Generating Rules, Decision Tables
output
Single Integrated Account Statement

***Results***

**Reduced cost and elapsed time to generate statements**

- **By replacing temporary files with in-memory tables**
- **Read all data once into in-memory tables and use virtual sorts**

DataKinetics.

## Credit card settlement - challenge

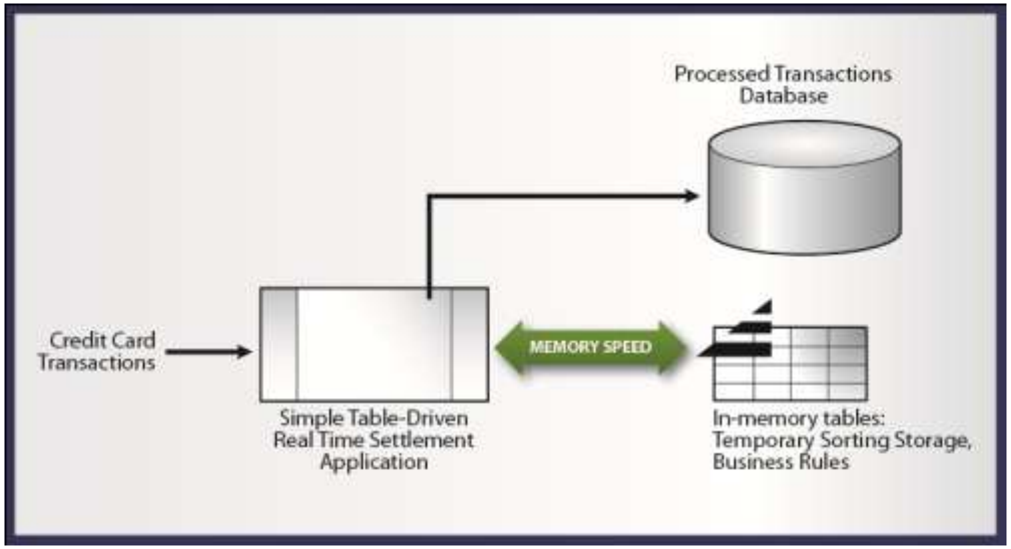
***Challenge*** **Be able to handle a higher volume of transactions, and be able to implement changes to the settlement process**

- As credit card use increases, the volumes of transactions that need to go through the settlement process increases
- When a customer uses their credit card, the transaction goes through an approval process (and some customers use tableBASE for transaction approvals), and the settlement is done to ensure the vendor gets their money, and the customer gets charged
- There is both a capacity challenge and a challenge in adapting the processes to accommodate changes to rules, such as currency exchange

## Credit card settlement – solution

***Solution***

- With DataKinetics tableBASE each transaction runs through only the business rules that are applicable to it – a far more efficient process.
- New accounts, users, card types, regions, jurisdictions, vendors, etc. can be added quickly to the appropriate table.


Processed Transactions Database
Credit Card Transactions
MEMORY SPEED
Simple Table-Driven Real Time Settlement Application
In-memory tables: Temporary Sorting Storage, Business Rules

***Results***

**Financial company gets the capacity and throughput required for this business critical application**

- **handles 45B accesses per hour using tableBASE for settlement or 12.5M accesses per second**

## Summary

## How in-memory tables can optimize data access

- Optimize data access
  - ➢ Use DBMS for read / write updating of data
  - ➢ Use in-memory tables to off-load read only I/Os
- Optimize application performance by reducing I/Os
  - ➢ Check to see how many I/Os are read only
  - ➢ Check to see how many read accesses there are to different tables
  - ➢ Identify any temporary files that are created
  - ➢ Use temporary tables for consolidation and continuous summation
- Optimize application performance by sharing data
  - ➢ Check to see how many applications use the same data
  - ➢ Place one copy in in-memory tables for all applications to use
  - ➢ Check to see if data is passed from one application to another
  - ➢ Use tables to pass that data at memory speed

## 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 |

## In-memory tables

**In-memory tables are used by enterprises to:**

- Optimize computing efficiency and cost reductions
- Achieve superior performance, capacity, and scale
- Choose where you put your data
- Make your infrastructure change transparent to customers, employees and partners
- Adapt quickly to market changes and be more competitive

## Thank you for your time.

Verna Bartlett
Head of Marketing
613 523 5500 ext 208
vbartlett@dkl.com