IN-MEMORY COMPUTING: IT'S NEW AND IT'S NOT...

LARRY STRICKLAND
DATAKINETICS

LARRY STRICKLAND

Chief Product Officer

DATAKINETICS

So why am I presenting here today

IS THE MAINFRAME STILL RELEVANT?

Used by **92%** of the top 100 global banks

Processes nearly **100%** of all credit card transactions

Manages **80%** of all corporate data

Processes **nearly all** airline reservations

70% of all business transactions

95% of daily ATM transactions

80% of Point Of Sale transactions

WHY IS IN-MEMORY CONSIDERED (ON MAINFRAMES)

- It's nearly always about the $
- However, when looking deeper, the rational is always one of:
  - Improve Response Time
  - Reduce Elapsed Time
  - Reduce CPU Usage

TWO PARTS ....

- Reducing I/O wait times
  - Improves Response Time
  - Reduces Elapsed Time
  - (minimal impact on CPU used)

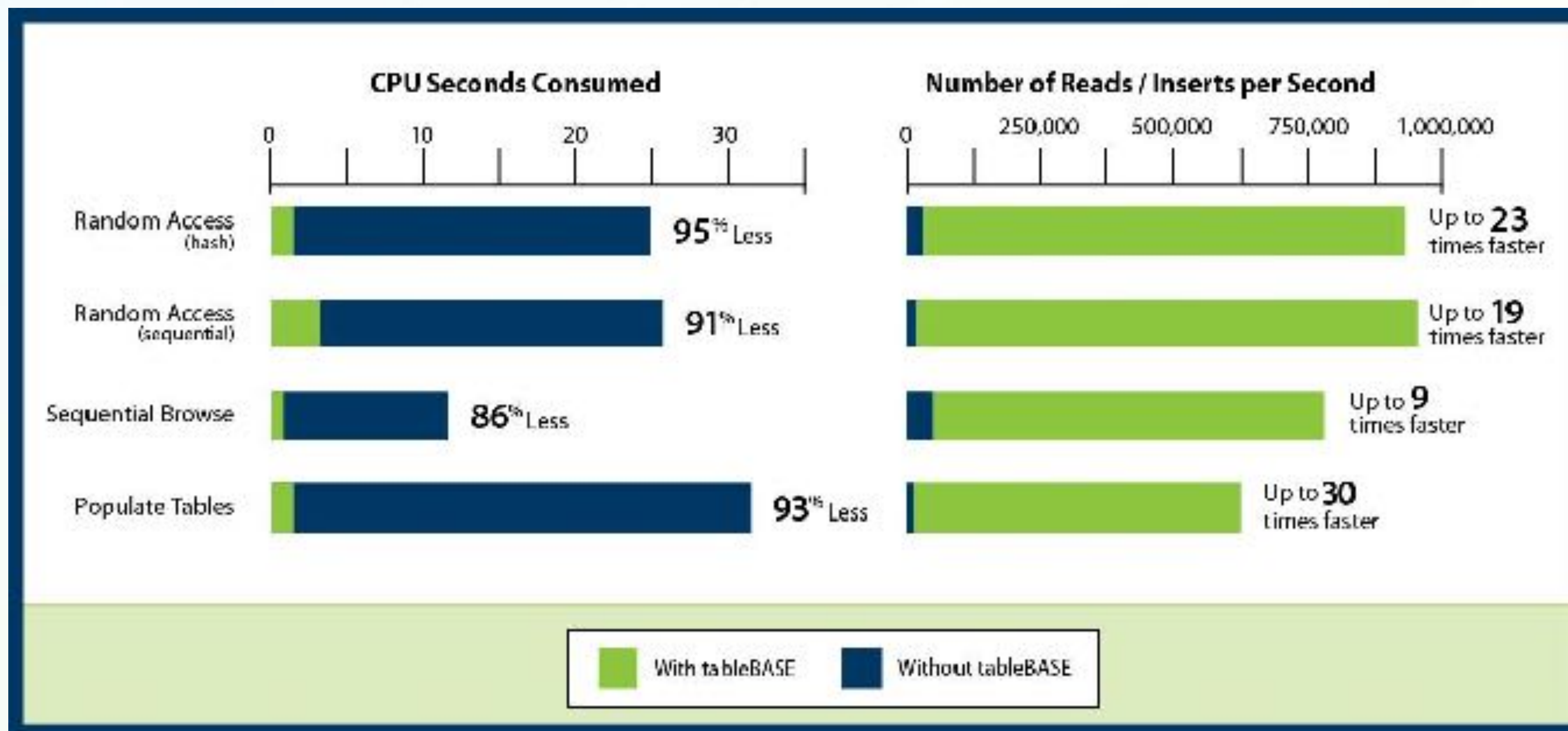
- Reduced Code Path
  - Improves Response Time
  - Reduces Elapses Time
  - Reduces CPU Usage

MAINFRAME USES MANY TECHNIQUES FOR REDUCING I/O

- Caching
- Buffering
  - DB2 buffering
  - Buffer pools
  - 3rd-party buffer tools like BPT, BPA4DB2
  - VSAM Buffers
- CICS managed data tables
- COBOL internal tables
- SSD ?

TABLEBASE – IN-MEMORY TABLE MANAGER

- Removes I/O
- Reduces Code Path



WHAT WE'VE LEARNED ALONG THE WAY

- WHICH DATA?
- INDEXING IS VERY IMPORTANT
- NOT ALL HASHES ARE CREATED EQUAL
- RULES, RULES, RULES
- SEPARATE OUT READ-ONLY
- ACCUSATIONS FLY

WHICH DATA?

WHAT TO PUT IN-MEMORY

BIG OR SMALL TRANSACTIONAL DATA

- Large data takes longer to search, so has huge Elapsed time advantages in being accessed from Memory
  - Great Response Time Improvement
  - Great Elapsed Time Improvement
  - CPU impact is minimal

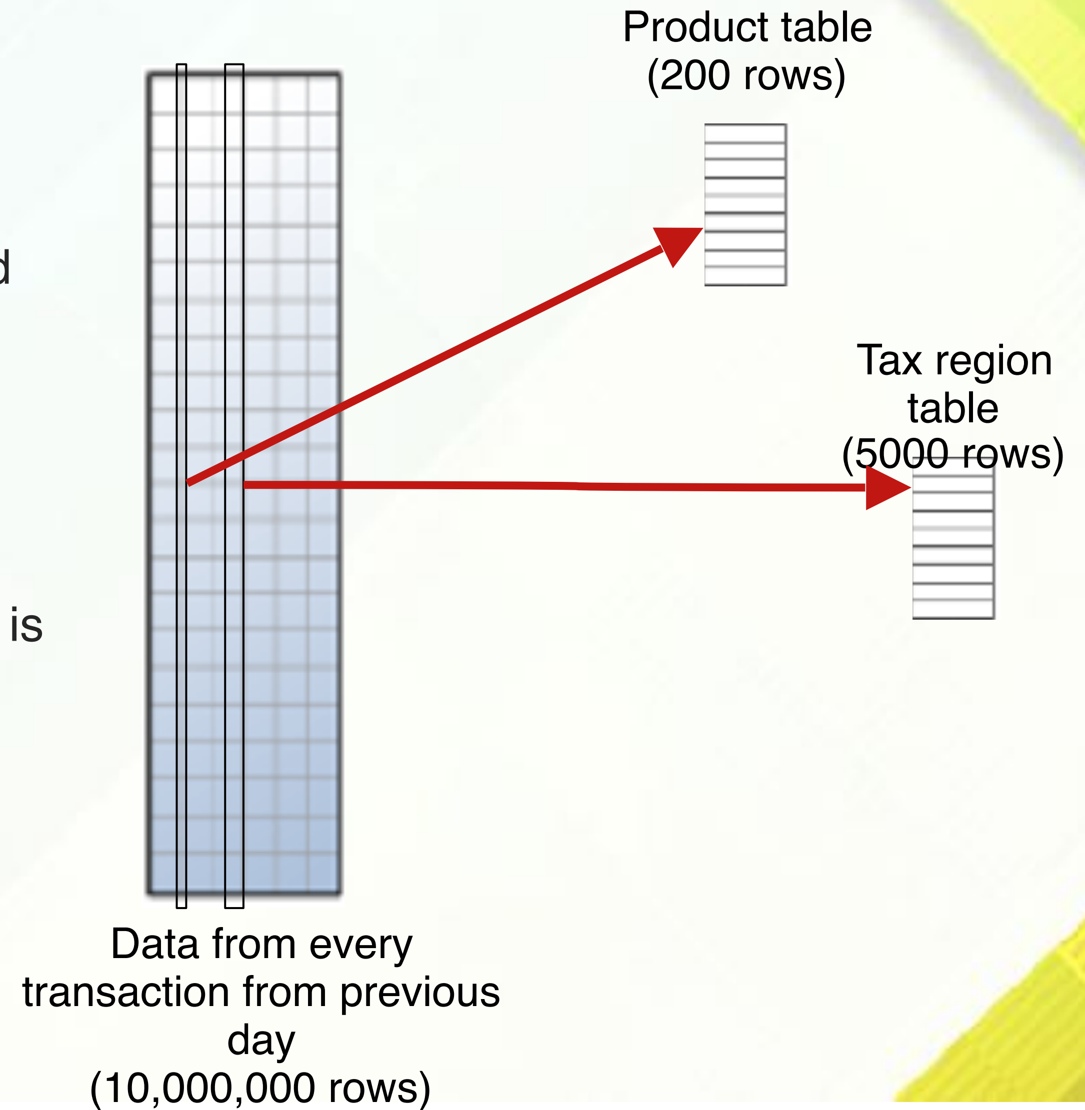
Every row read into memory
Not every row read once it is there

- Small data - small in size, accessed very frequently (Reference Data)
  - Good Response time Improvement
  - Good Elapsed Time Improvement
  - CPU impact is huge

Every row read into memory
Every row read potentially 1,000's of times

IN-MEMORY TECHNOLOGY: LOOKING AT CPU

- Consider the large table here
  - You won't gain much my reading it into memory and accessing the data from there – as each row isn't read frequently
- Different story for smaller reference data tables
  - Top table is read once into memory, then each row accessed 50,000 times from memory
  - Bottom table is read once into memory then each row is accessed 2,000 times from memory
- In actual use, some rows are read once into memory and accessed from there many millions of times per day…



RESULTS FROM CREDIT CARD PROCESSING

Challenge

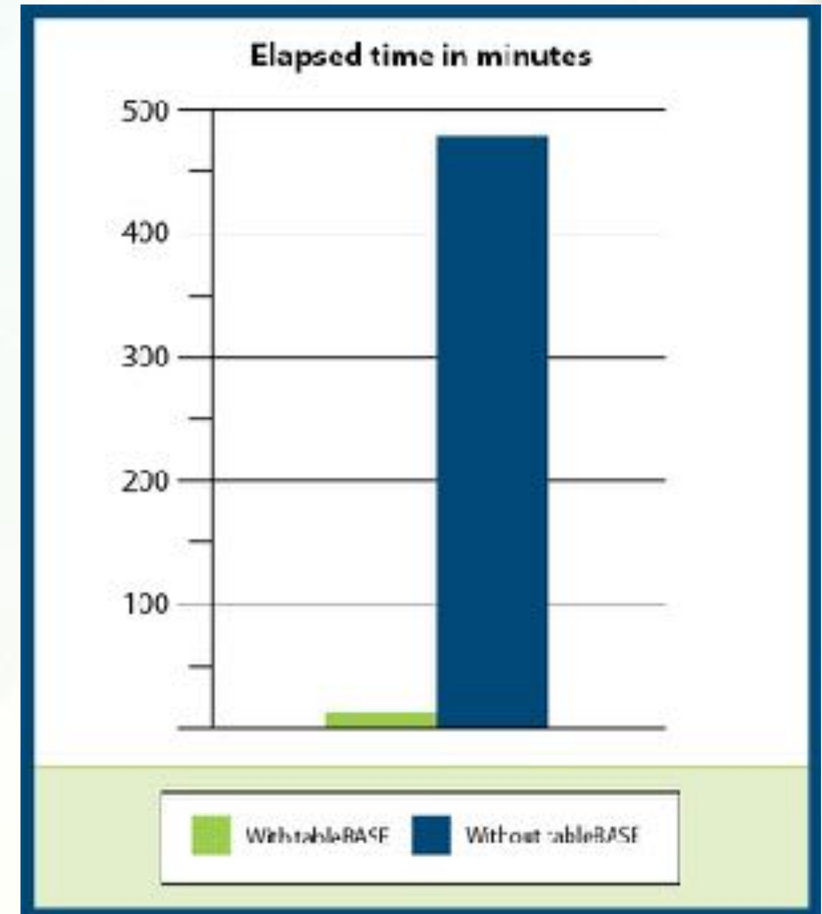
- Reconciliation batch processing taking too long

Solution

- Move a table describing the credit card options into tableBASE
- Each transaction required data from that table

Results

- 97% reduction in CPU time
- Batch job that took 8 hours to complete now takes 15 min



BIG OR SMALL DATA - ECONOMICS

- Large data takes longer to search, so has huge Elapsed time advantages in being accessed from Memory
  - Great Response Time Improvement
  - Great Elapsed Time Improvement
  - CPU impact is minimal

Cost neutral or more expensive (increased memory requirements)

- Small data - small in size, accessed very frequently (Reference Data)
  - Good Response time Improvement
  - Good Elapsed Time Improvement
  - CPU impact is huge

Reduces cost

INDEXING IS IMPORTANT

PROBABLY OBVIOUS BUT…

INDEXING IS IMPORTANT

- COBOL Internal Tables are in Memory
- Often used to manage temporary tables
- Primary index – no alternative indexes
  - Serial Search required if alternative searches required

ONE CUSTOMER'S EXPERIENCE

Challenge

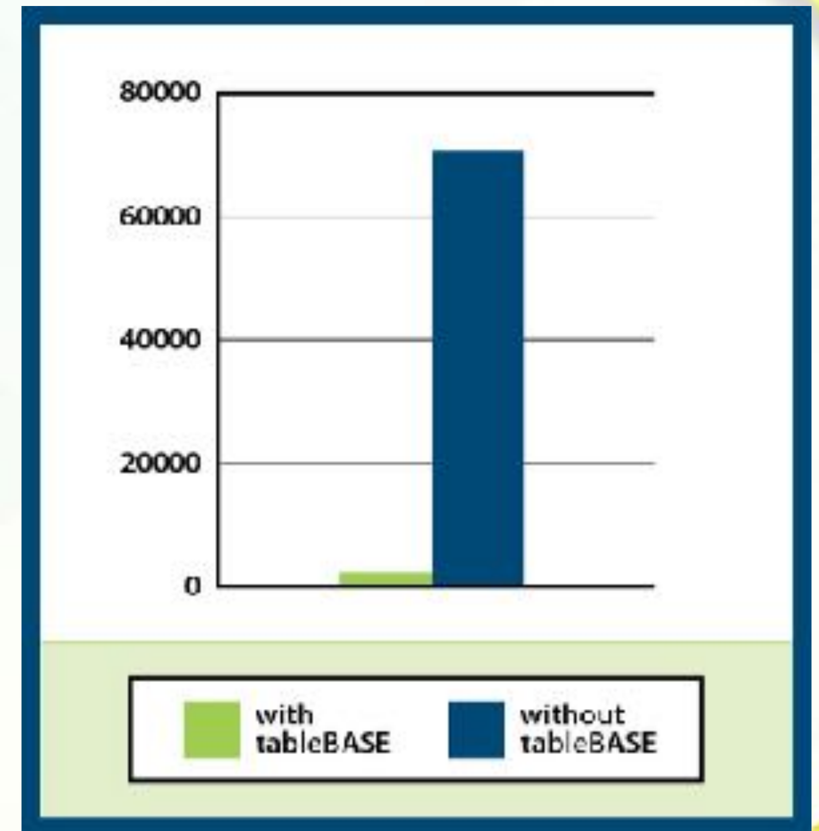
- A COBOL program was using an internal table and a binary search
- The search code was called 1.25 million times and had 4 searches in it
- Took over an hour of CPU to execute

Solution

- Replace the 4 searches with calls to tableBASE

Results

- 98.3% reduction in CPU
- Now takes less than a minute to execute



INDEXES

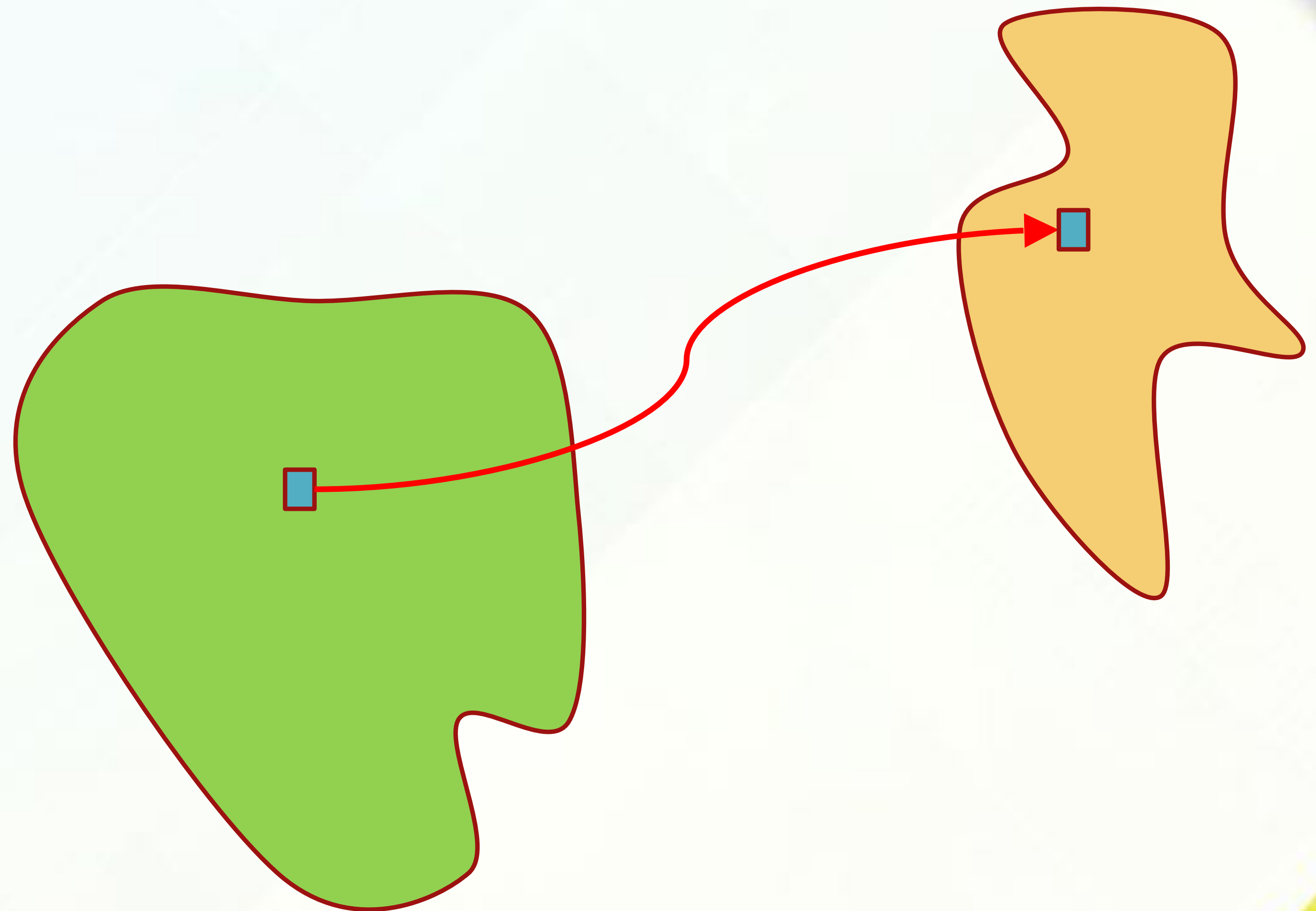
- Indexing for Speed (with tableBASE – but probably generally applicable for other implementations)
  - <10 rows – serial search
  - >10, <100 rows – binary search
  - >100 rows – Hash search

HASH INDEXING

NOT ALL HASHES
ARE CREATED EQUAL

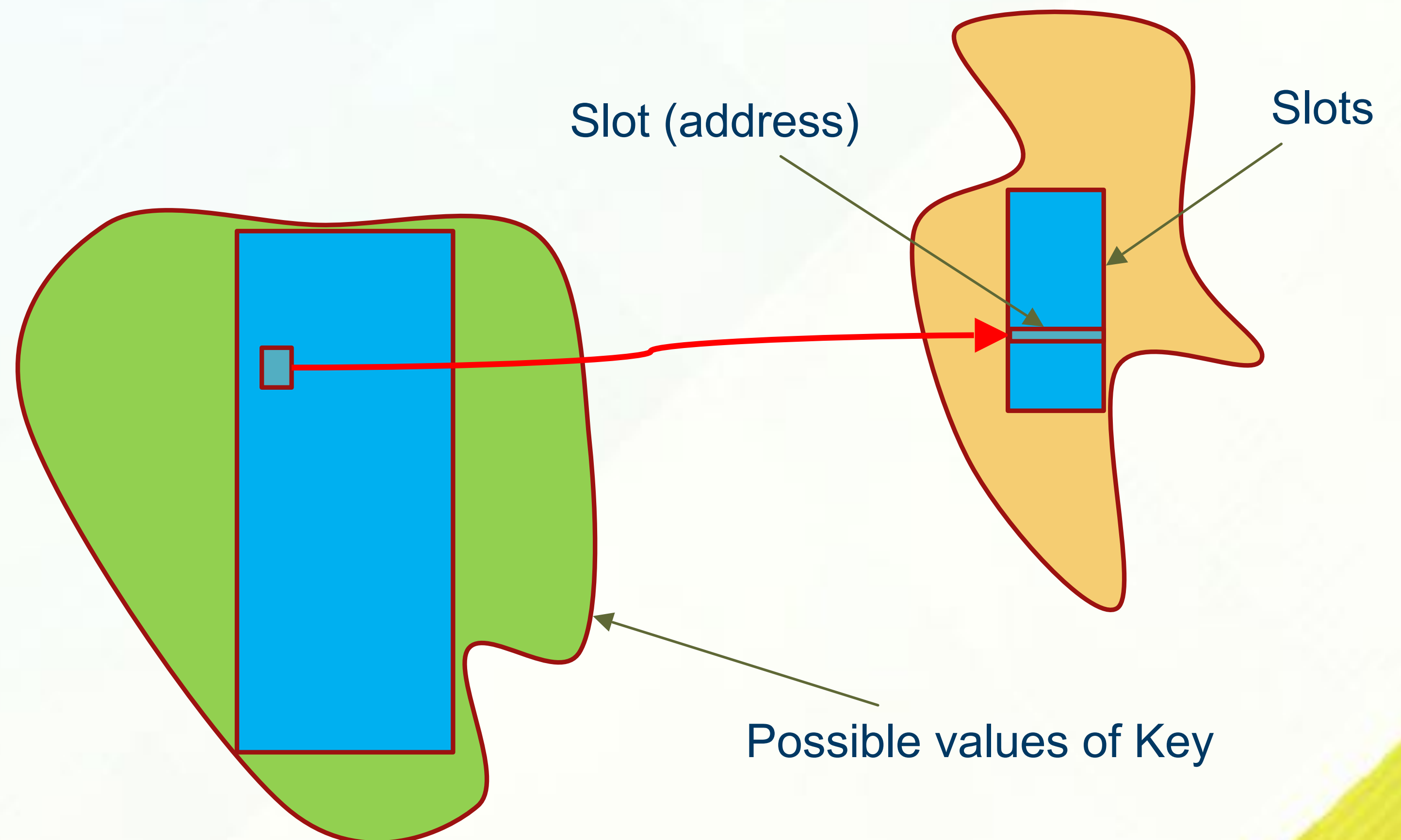
WHAT DOES HASH DO?

- Maps space to another space
  - One way
  - Typically shrinks (doesn't have to)
  - Arbitrary bytes to number
  - Can encrypt

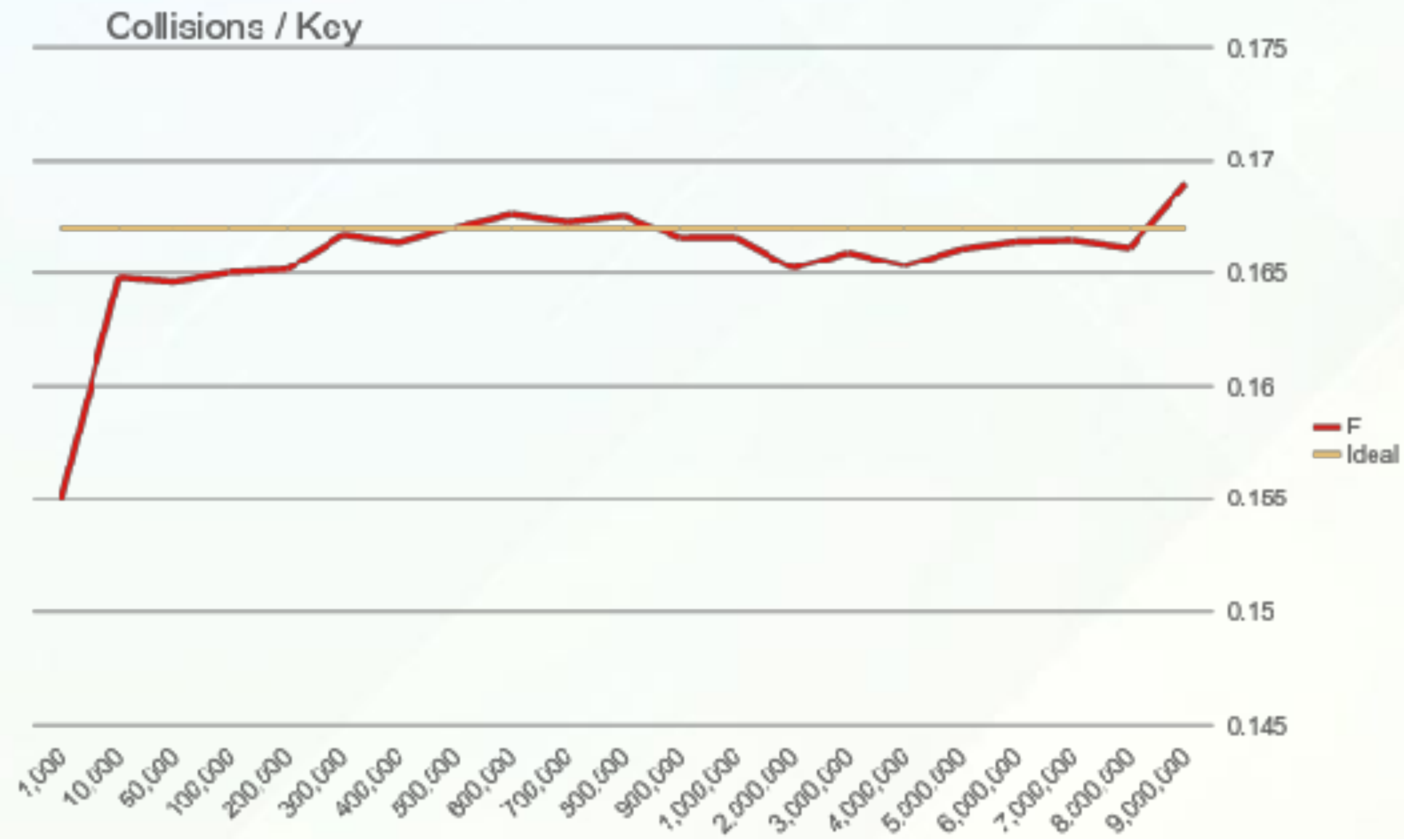


WHEN USING HASH TO INDEX

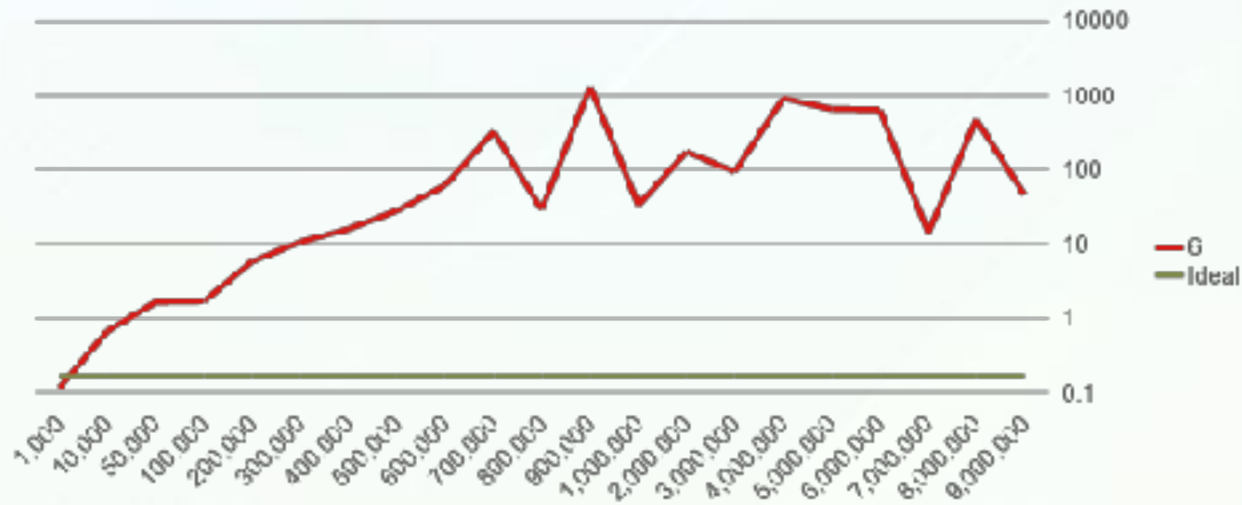
- Hash is used to calculate a slot
  - Slot calculated can simply be a pointer to the key (if in memory)
  - Need to deal with collisions
- Density is #keys/#slots
  - Higher value
    - less memory used
    - More collisions
  - Lower value
    - more memory
    - Less collisions

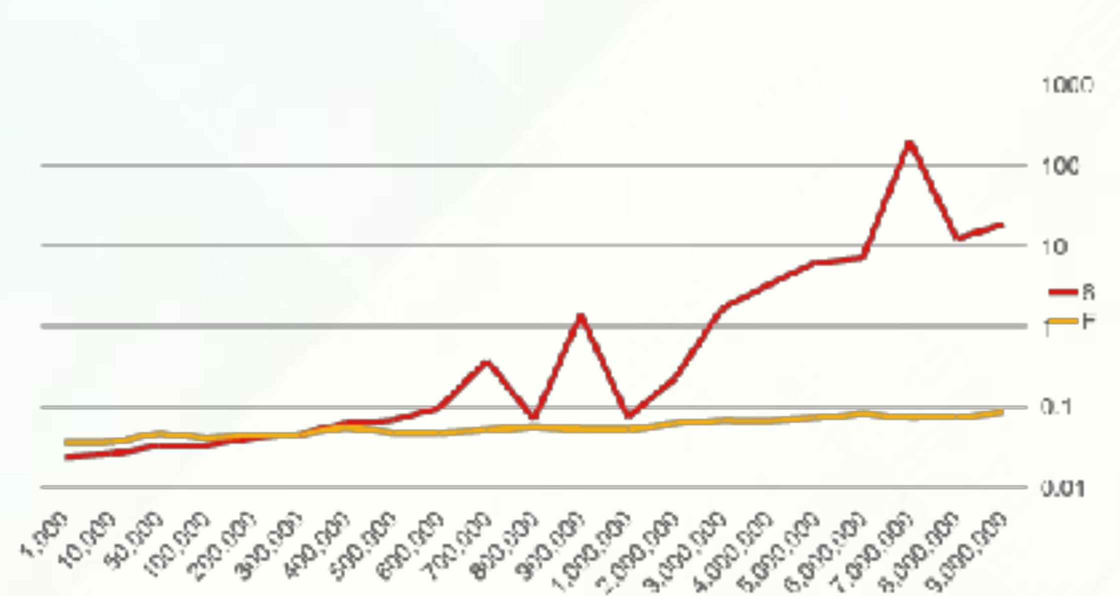


HASH ALGORITHM BEHAVIOR - FIRST ATTEMPT



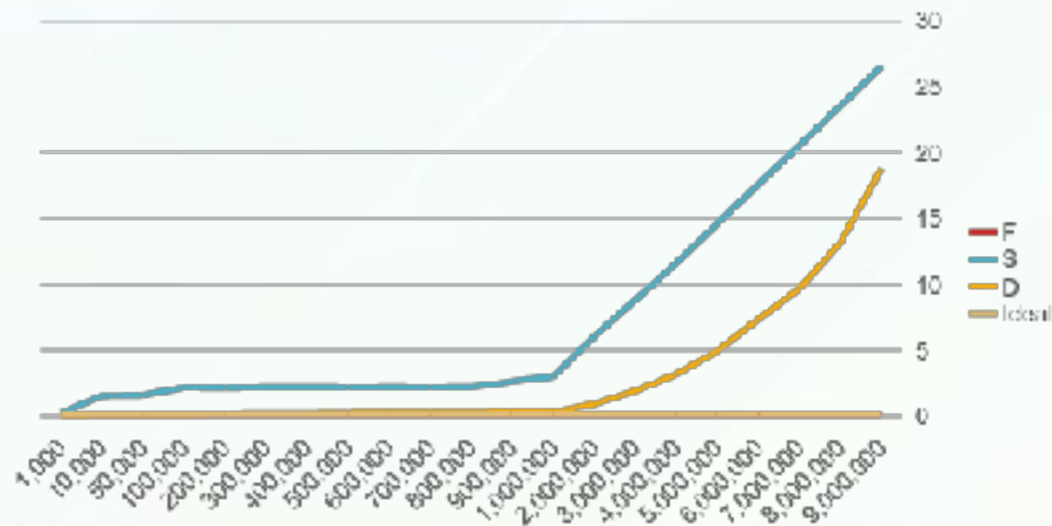
SOME RESULTS (CORRELATED KEYS)

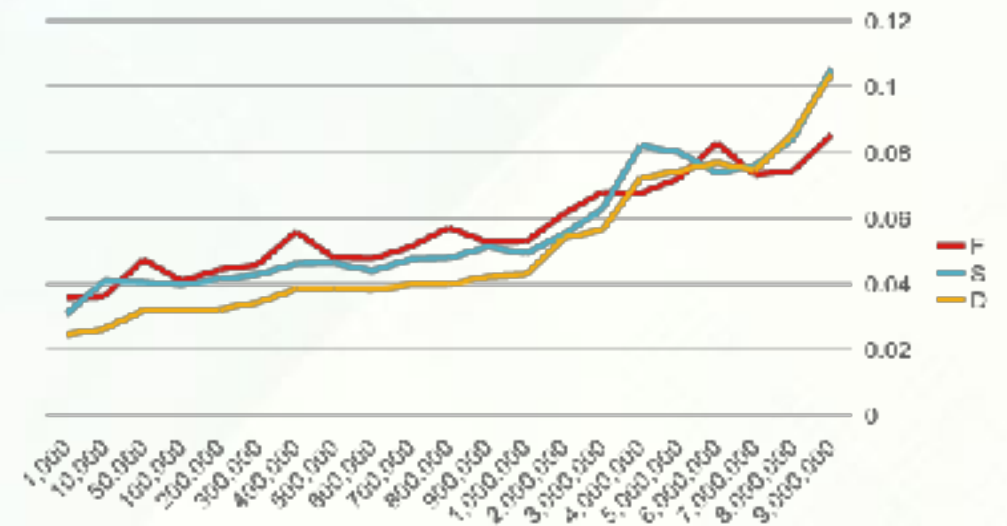




In-Memory Computing SUMMIT 2017

LOOKING AT SOME ALTERNATIVES





SO WHERE DOES THIS LEAVE US?

- If we don’t know much – should use a Hash with low collisions
  - I recommend the Fowler-Noll-Vo Hash function (FNV)
- But, if we know
  - Well distributed key
  - Small number of keys
  - V. Low Density

….. we may consider a cheaper function to calculate Hash

SPECIFIC HASHES

- With some knowledge of a key, we can create some very effective (high performance, low collisions) Hashes.

- E.g. Canadian Postcodes e.g K1A 3M2
  - Letters D, F, I, O, Q or U are not used
  - Letters W, or Z are not used in first position
  - 6 bytes have 300,000,000,000 combinations
  - Can limit to 7,400,000 with knowledge of distribution
  - Only about 830,000 in use

STANDARD HASH

- There are standard Hash algorithms out there
  - Linux 32 and 64 bit Hash algorithm

    $F(byte\,[\,]) = (\sum_{k=0}^{n-1} p^k .\text{byte}[k]\,)\ mod\ (2^b)$

    where p is a prime (31), and

    n is the number of bytes

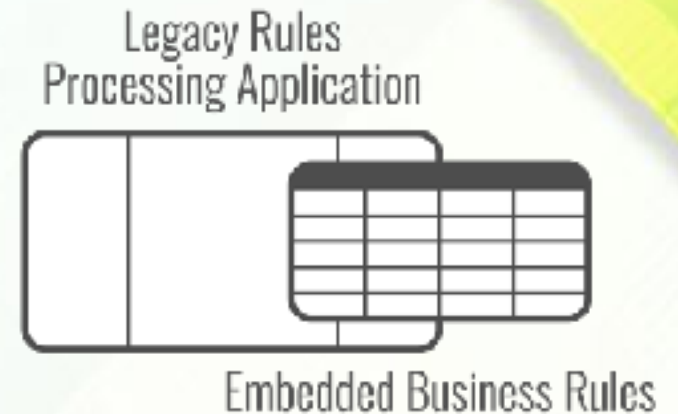
    b is 32 or 64
  - Maps any string to either 32 or 64 bit number
- Doesn't behave well with high densities
- However, combinations 2^{32} or 2^{64} - so low densities should be guaranteed

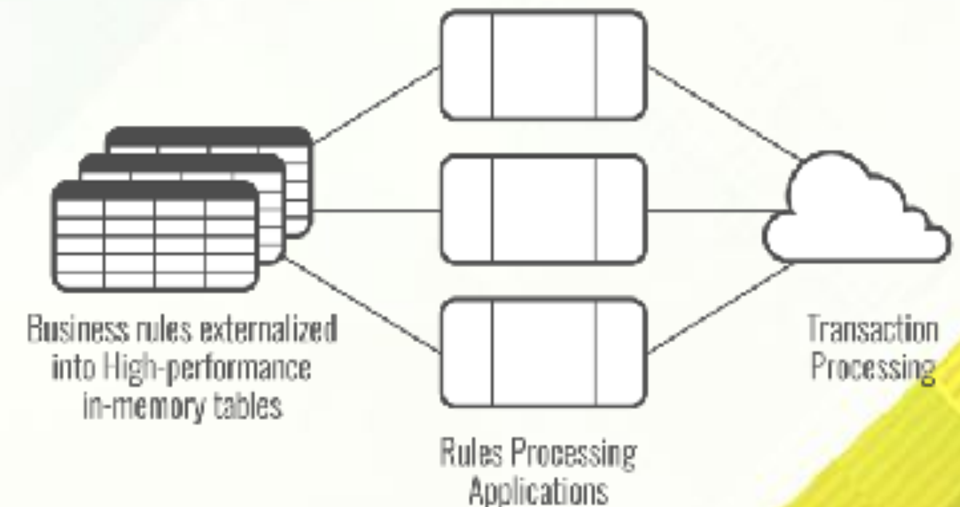
RULES, RULES, RULES

MOST FREQUENTLY READ TABLES

RULES PROCESSING

- Business rules are among the organization's most valuable intellectual property.
- For speed of processing, business rules were often embedded within mainframe applications.
- For business flexibility, these are often externalized into rules tables
- Rules tables accessed potentially 100's of times per transaction
  - Processing transaction logic
  - Fraud Rules





SEPARATE OUT READ-ONLY

GETTING MORE EFFICIENT

SHARED MEMORY TABLES

- Read and Write locks are standard practices to allow multiple programs to access the same table (almost) simultaneously
  - Routines required to deal with failures to remove locks and clean up
  - 60-85% of code path!

- Alternatives
  - Separate out Read-Only data (no locks required) 3 to 4 times improvement
  - Use table versioning and logical switches

LET THE ACCUSATIONS FLY

WHAT HAPPENS WHEN YOU REMOVE THE IO WAIT TIME

ACCUSATIONS

- You're using all the CPU!
- You're using all the memory

CONCLUSION

CONCLUSION

- The Mainframe is still relevant
- In-memory can help on multiple fronts
  - But needs a business case
- In-memory small data has a bigger impact on $
- Indexing (including the appropriate Hash function) is essential
- Rule tables are often the most read
- Careful what you wish for