# Ten Pounds of Batch In a Five Pound Window

# Can I push more workload through my existing hardware configuration?

**Batch window problems can often be reduced down to two basic problems:**

- **Increasing Business Volume**
- **Less time to process because of online commitments**

# What Are Your Options?

# How do you increase throughput?

- Reduce/Remove wait time
- Increase efficiency
- Reduce overhead

**Realize the true potential of your system**

# MAINVIEW Batch Optimizer: the Easy Alternative

## Reduces batch run times (elapsed time)

- No application changes
- No JCL changes
- Based on proven technology

# Batch Optimizer
# What does it do?

**Removes embedded wait times from batch jobs**

- I/O wait
- Serialization wait

# Batch Optimizer- How does it do it?

- Shrinks Batch Window processing with high performance and predictability
  - Enhanced I/O performance
  - Parallelize batch processes

# MVBO Data Optimization
## I/O Wait

- For Both VSAM and non-VSAM processing
  - Exploits RMODE(31) buffering
  - Dynamically selects optimum buffer values and processing techniques based on current system resource availability
    - Paging rate, CPU rate, below-the-line storage, etc.
  - Dynamically adjusts user region values to make any required additions to below-the-line storage transparent to the application

# MVBO Data Optimization
(I/O Wait)

- Flat Files - (QSAM/BSAM)
  - Moves data buffers above the line
  - Replaces low-level I/O processing providing complete control of buffer management and physical I/O requests
  - All I/O requests satisfied by MVBO's internal buffer manager
  - For sequential processing reads large amount of data and overlaps I/Os to maximize performance regardless of blocking characteristics

# MVBO Data Optimization
(I/O Wait)

- VSAM Processing
  - Optimizes buffer values
  - Activates VSAM options such as deferred write which increases optimization
  - For random access builds LSR buffer pools and dynamically switches to LSR processing
    - When LSR processing with sequential accesses performs read-ahead for greater performance benefit
  - For NSR and LSR processing moves buffers and control blocks above the line to aid virtual storage control relief

# MVBO Data Optimization
(I/O Wait)

- Centrally managed through internal tables called Policies
  - No JCL changes
  - No application changes

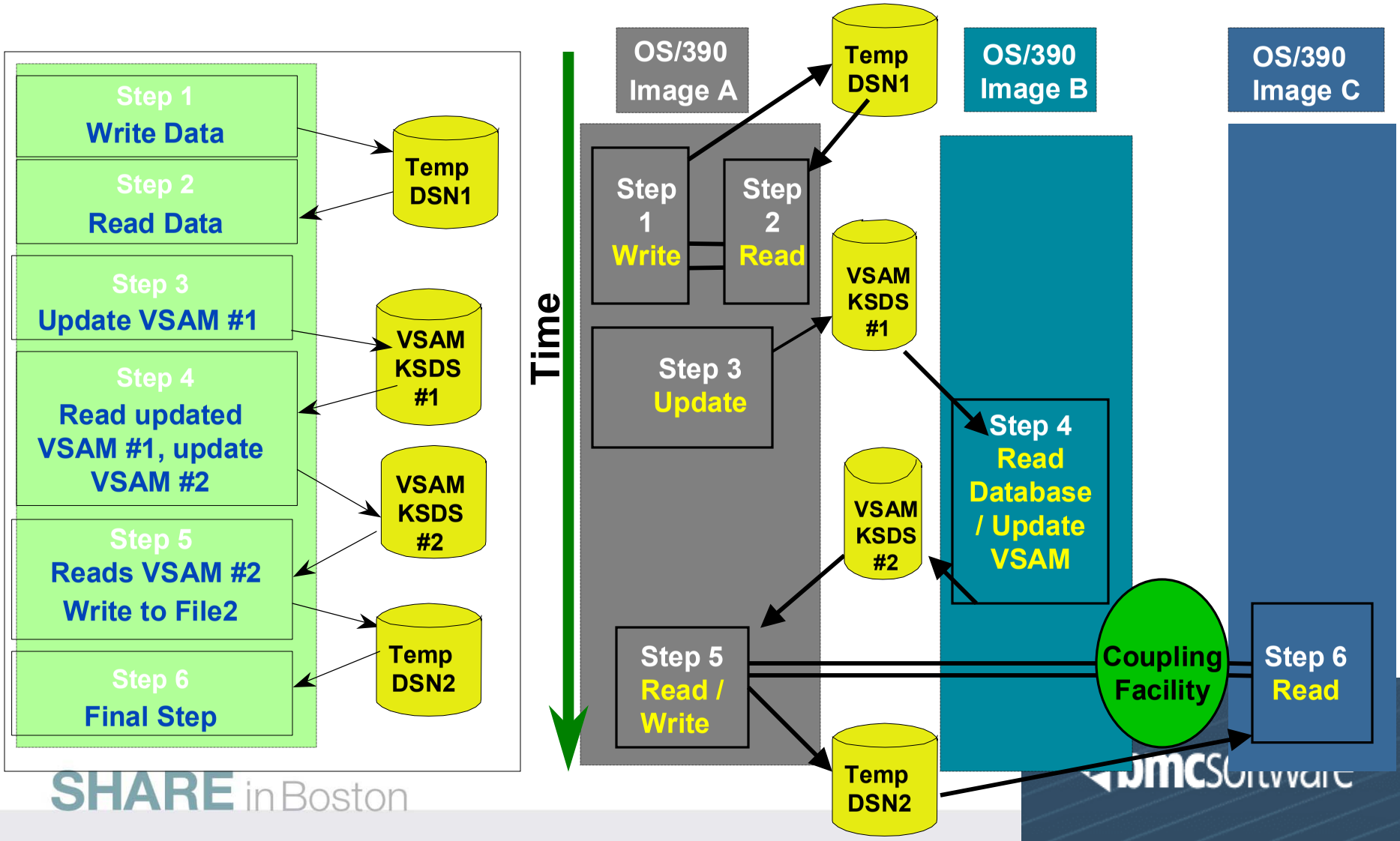
# MVBO Job Optimization
(Serialization Wait)

- Parallel Execution of Job Steps
  - Splits job steps for piping and concurrent execution based on history and policy definitions
    - Steps with no data dependencies
    - Dependent steps where a data pipe can be established, e.g., reader/writer pair
  - Data is passed to split steps via Pipe
    - Essentially a data buffer shared by two programs

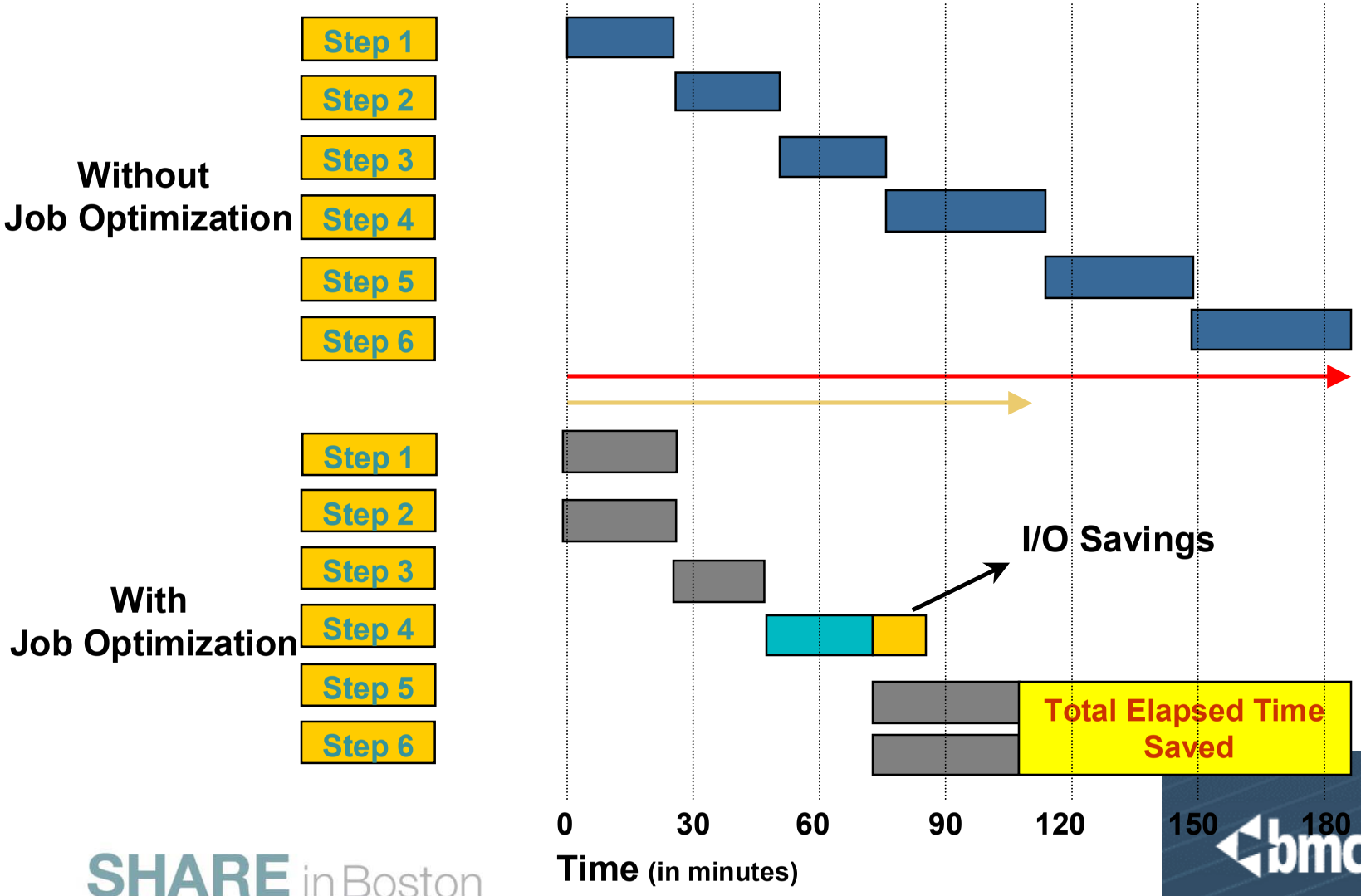
# MVBO Job Optimization
(Serialization Wait)

- Step to Step Piping
  - 2 data-dependent steps executing in parallel
  - Data pipe forms between reader step and a writer step: reader can access record as soon as it is written
  - Step-to-step piping passes data directly into the reading program

# ‘Typical’ Batch Job



# MVBO Job Optimization

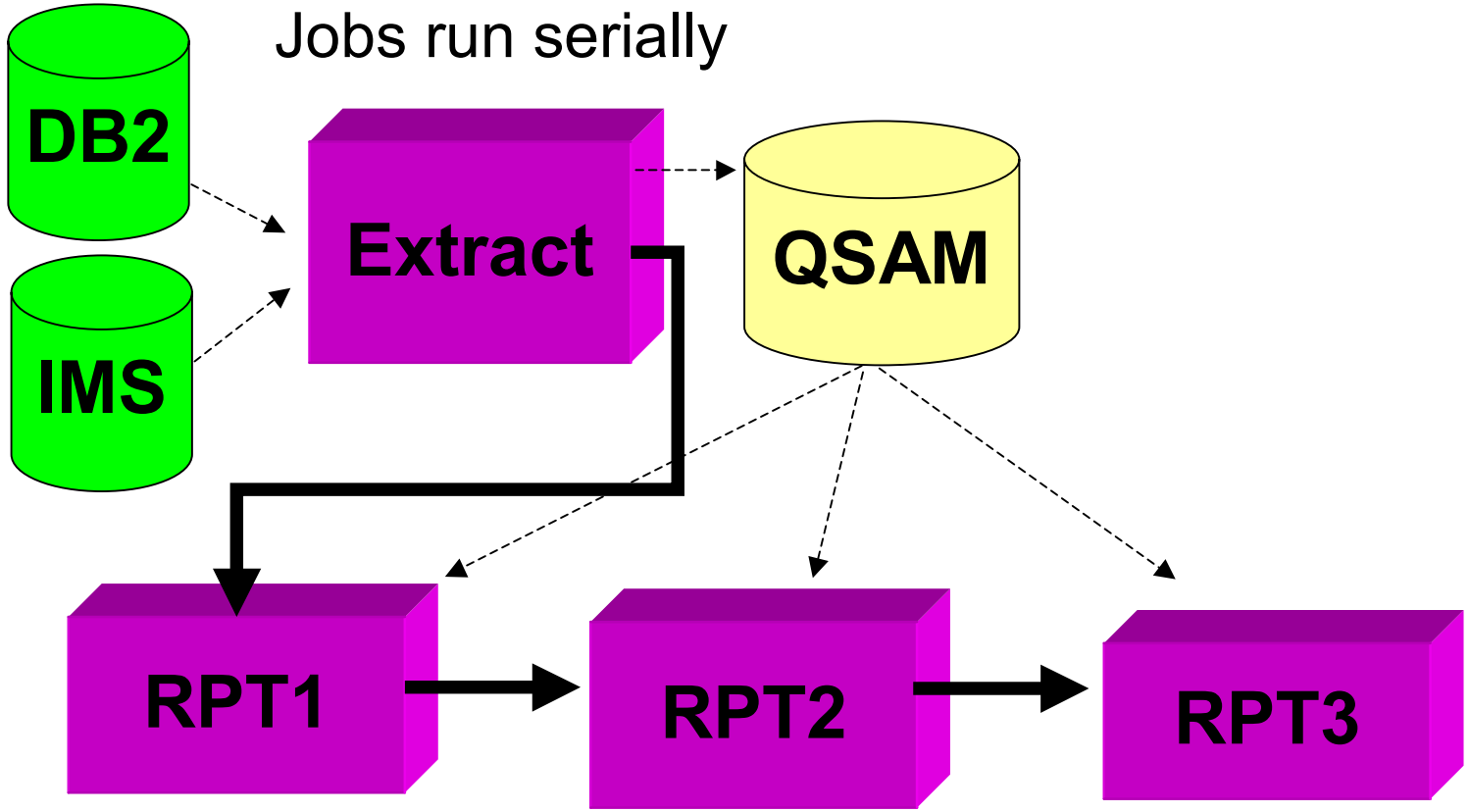


# MVBO Piping
(Serialization Wait)

- Passing data to another job
  - 2 data-dependent jobs executing in parallel
  - Data pipe forms between writer step in one job and reader step in another job
  - Direct communication with CONTROL-M

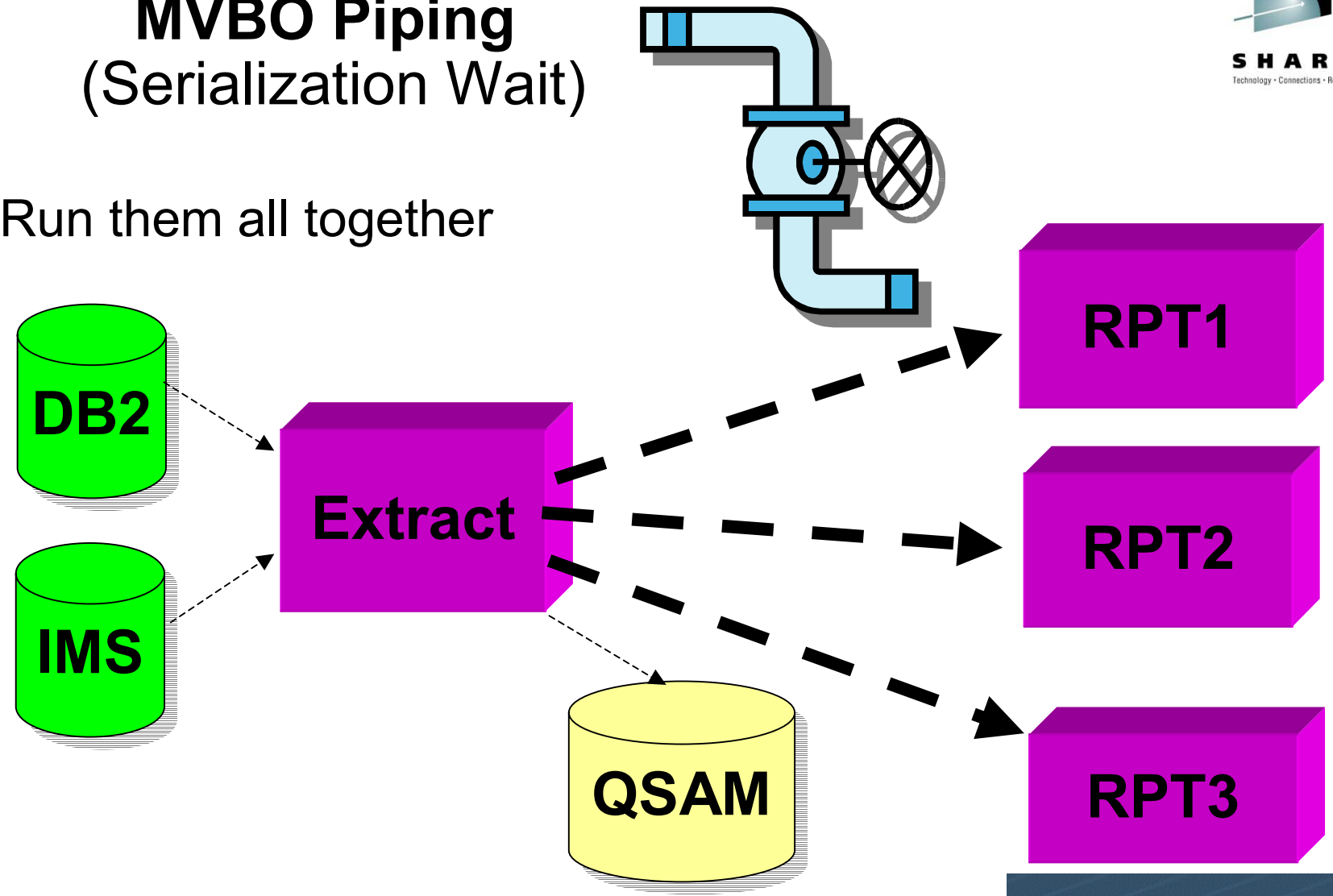
# Without MVBO Piping
(Serialization Wait)

- Jobs run serially



# MVBO Piping
(Serialization Wait)

- Run them all together



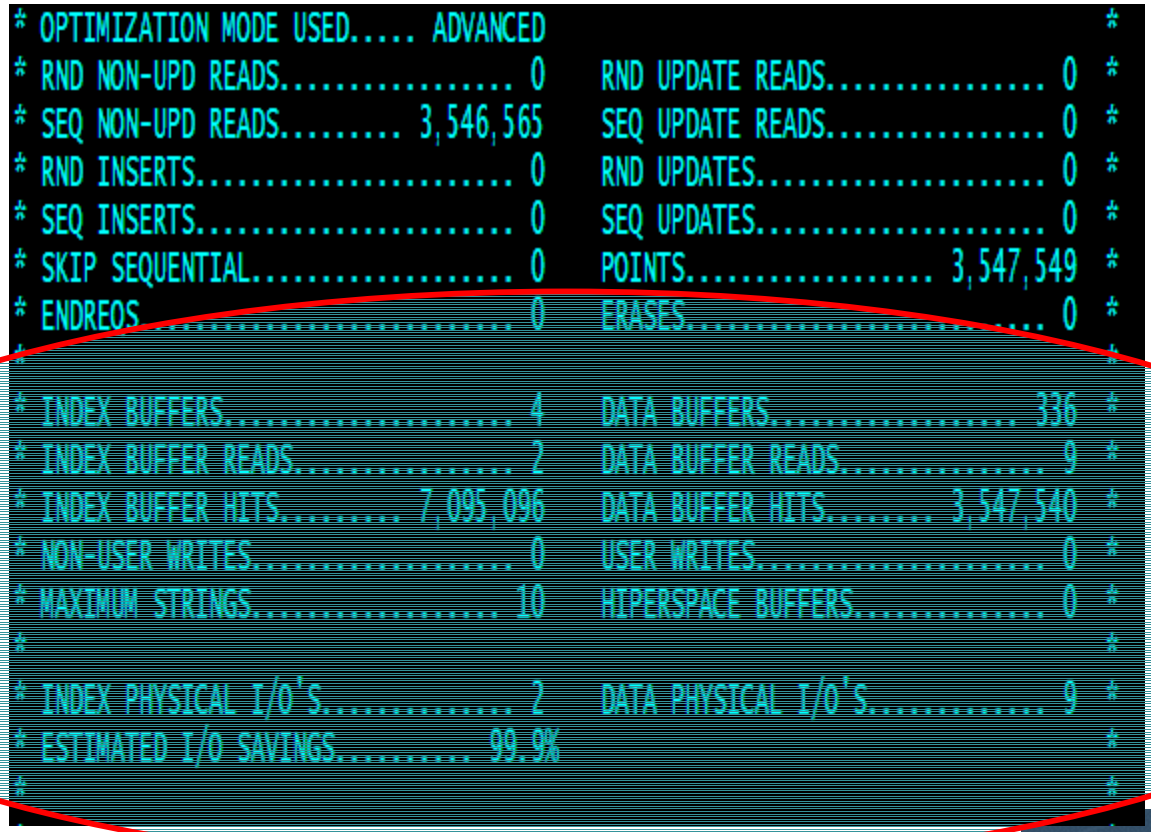
# MVBO Candidate Utility
## How much improvement can I expect?

- RPL460 JOBU360U

```
History source:                 SMF
Elapsed time (Normal):          3:01:35
Elapsed time (Optimized):       2:33:40
Savings (Potential):            0:27:55   16%
```

```
27 RPL465W2 --------          IKJEFT1B
   CALENDAR RPL.RPL020.RPL150.CALENDAR                    QSAM, Read
            DD ALREADY TUNED                  EXCPS:          2
   CUSTFILE RPL.RPL460.RPL460.INPUT                       QSAM, Read
            DD ALREADY TUNED                  EXCPS:     36,559
   WORKFILE SYS08182.T054528.RA000.RPL460.TEMP.H01        QSAM, Read
            DD ALREADY TUNED                  EXCPS:     58,447
   SORTWK01 SYS08182.T054528.RA000.RPL460.R0106813         EXCP, Write
                                             EXCPS:      1,237  EST SAVE  31% EST TIME  0:00:05
   SORTWK02 SYS08182.T054528.RA000.RPL460.R0106814         EXCP, Write
                                             EXCPS:      1,193  EST SAVE  31% EST TIME  0:00:04
   SORTWK03 SYS08182.T054528.RA000.RPL460.R0106815         EXCP, Write
                                             EXCPS:      1,204  EST SAVE  31% EST TIME  0:00:04
   SORTWK04 SYS08182.T054528.RA000.RPL460.R0106816         EXCP, Write
                                             EXCPS:      1,205  EST SAVE  31% EST TIME  0:00:04
   TBL094   SLM.MISC.TABLE.FILE                           VSAM, Read
                                             EXCPS:        304K EST SAVE  40% EST TIME  0:02:36
```

# We got a pretty good bang for our buck with this one



# Data Accelerator Compression

Notice the word "Accelerator"

# How Can Software Based Compression Help?

- Major Benefits
  - “Footprint” reduction of data
  - Performance advantages of compressed data

Hardware based compression only reduces the “Footprint of the data”

It does nothing for performance

# DATA ACCELERATOR Compression (DAC)

- Data Accelerator has a very good compression engine
- The worst compression percentage I have seen is 57% on IMS LOG data
- Typical compression savings are greater than 66%
- I had one absurd example where I got 97% compression on some of my generated test data
  - A 800 cylinder dataset was reduced to 15 cylinders!!

# Supported Data Set Types

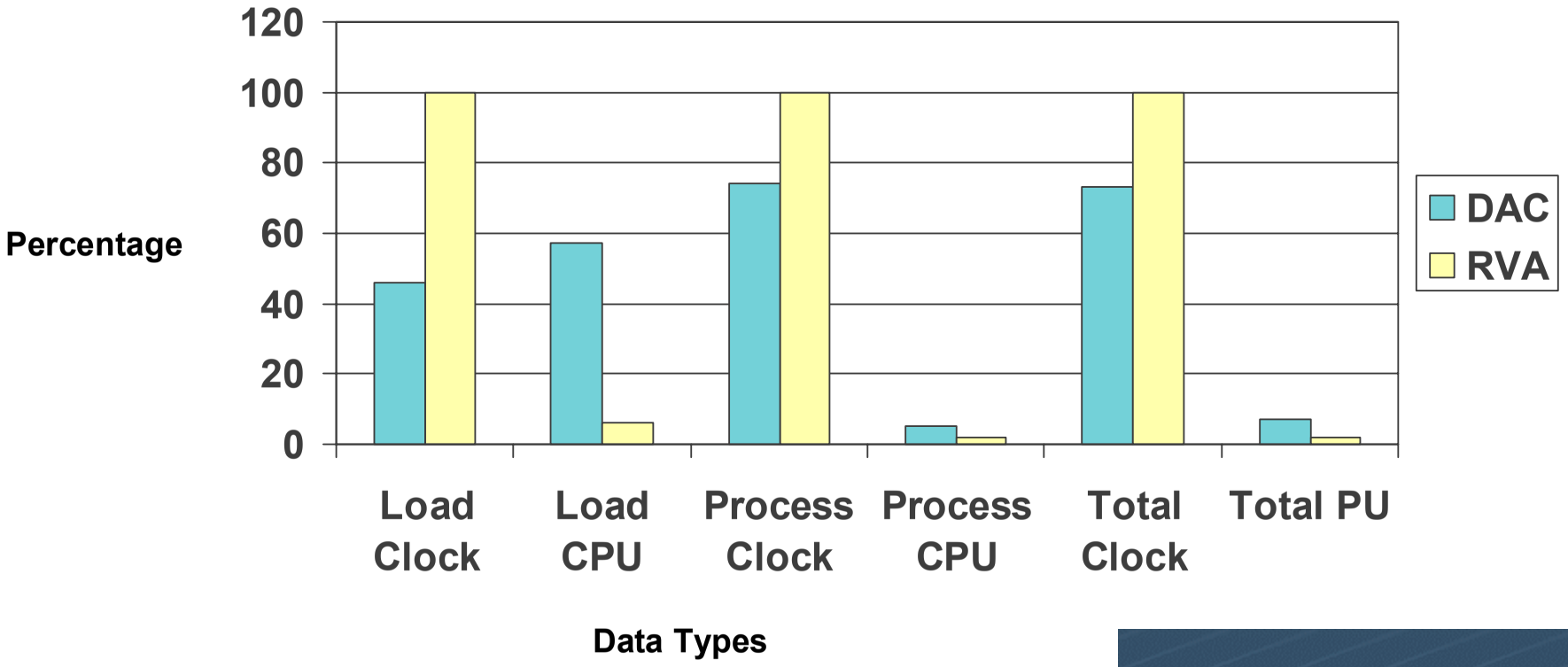
- Non-VSAM — BSAM, QSAM, BPAM
- VSAM — KSDS, ESDS, VRRDS

# DATA ACCELERATOR Compression

- Centrally administered
  - Candidate DSN patterns added via ISPF Interface
    - Ex. Datasets that begin with CWA.DAC.**
  - Compression is automatic with the next load of the file
  - Can be administered via SMS Class
- Requires no JCL changes
- Requires no application changes

# DATA ACCELERATOR Compression

Compression (Host vs. DASD) Benchmark Comparison



# Now that we have compression…

- Remember that I said to pay attention to the word “Accelerator” in the product title?

- Let’s talk about performance…

# Seeing is believing

- Let's take a look at compression in action

DAC

# How does software compression help?

- If you compress a file, it takes fewer resources to process the file
  - Data remains compressed as it crosses the I/O channels
  - Data remains compressed as it resides in the I/O buffers
  - Physical I/O is much more efficient using compressed data
    - The path length of one EXCP is between 25k – 50k assembler instructions !!
    - In addition, your program has to wait for the physical data transfer!!

# Where does it help?

- Most batch processing is sequential in nature
- Anything that makes a sequential pass of a compressed file will benefit
  - Backups
  - Application sweep programs
  - SORT processing

- Best candidates are datasets that are re-read often
  - VSAM files
  - Master files
  - Extract files
  - Archives

# What kind of benefit are we talking about?

- VSAM file 8,000,000 fixed length records
- Compression % = 81%

| | Compressed | Uncompressed |
|---|---|---|
| • | Elapsed 2.22 minutes / CPU 30.58 sec | Elapsed 6.65 minutes / CPU 30.03 sec |
| • | 2.18 minutes / CPU 30.71 sec | 5.33 minutes / CPU 29.93 sec |

# Even better if we avoid expansion overhead

- DFDSS backup without expansion overhead
  - 75% improvement –
    - *11,000 cylinders of IMS log data average 62% compression*
- VSAM Backups benefit as well

# What are the trade-offs?

- Software compression does cost CPU time
  - More expensive to compress data
  - Less expensive to expand data
- Some CPU is offset by more efficient I/O to write the compressed blocks

- Elapsed time is the big savings

  "You can always add more MIPS to your environment… but you can't add more time to the day."

# Let's bring it all together

**What happens when you blend MVBO with DAC?**

- 8,000,000 VSAM records
- Batch process updated 800,000 records (10% of the file)
- Native processing time 43 minutes elapsed – CPU (1.30.59 minutes)
- Batch Optimizer only 13 minutes elapsed – CPU (43.30 seconds)
- MVBO & DAC 4 minutes elapsed – CPU (30.01 seconds)

81% compression achieved

# What is a good strategy?

- Sequential processes yield the best benefit
  - Look for files with favorable Read/Write ratio 2:1 or greater is best
    - Avoid temporary files - && type datasets
  - VSAM files tend to be good candidates
    - Might need to embed some freespace if records grow after an update
  - Extract files
  - Historical data
- Combination of VSAM random processing and LSR buffering
  - Compression enhances the effect of buffering
  - Increases probability of buffer hit & reduces physical I/O

# What about IMS and DB2 jobs?

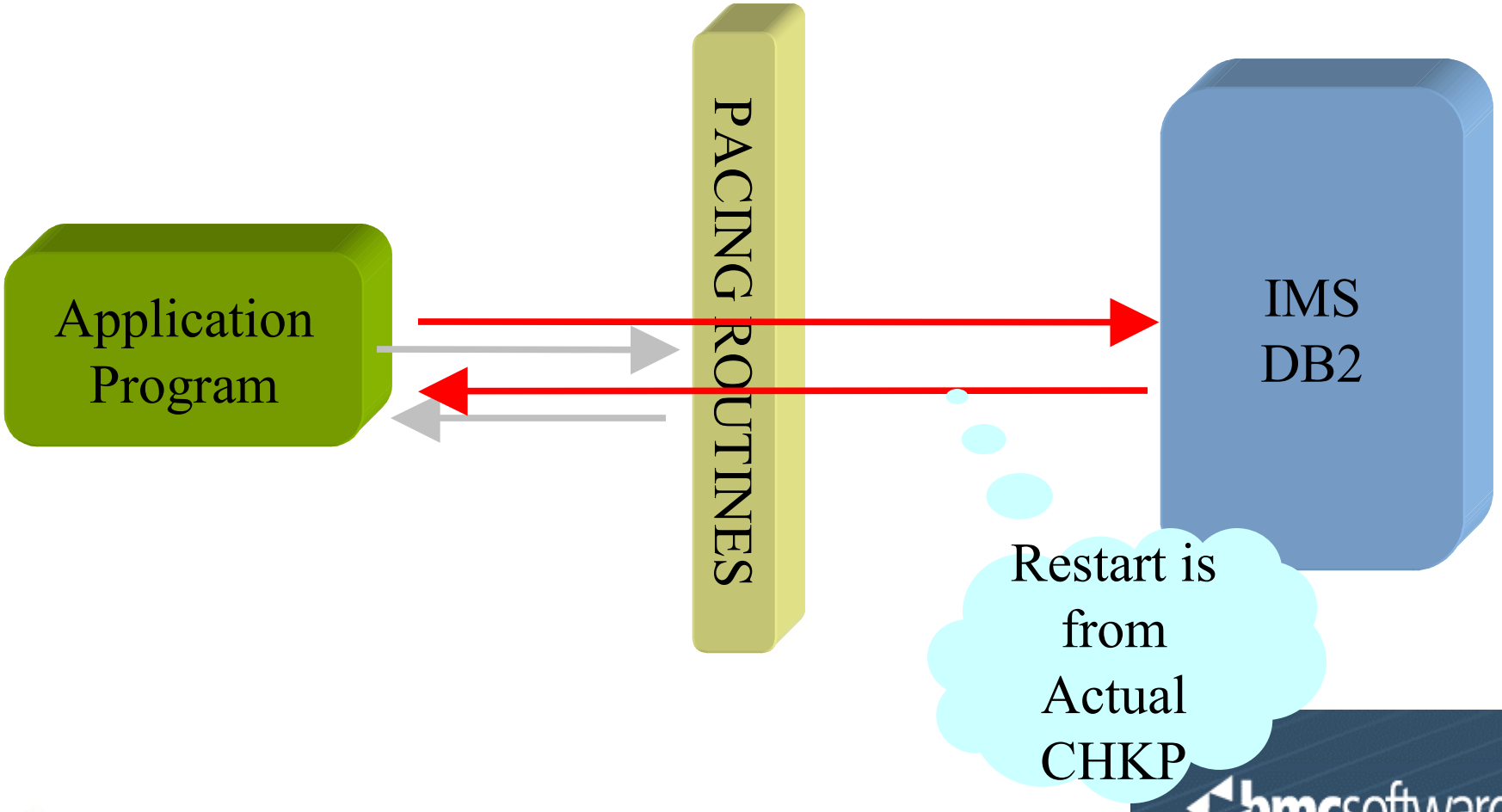
- Most DBMS's require checkpoint/commit processing
  - Required but a necessary evil
  - Extremely expensive
  - 100% overhead

- Removing excessive checkpoint activity can provide significant run time improvements

# AR/CTL
## CAN HELP MANAGE YOUR BATCH

- AR/CTL is part of a family of products by BMC Software that addresses the needs of batch DB2, IMS, and VSAM applications
- AR/CTL is designed to provide a checkpoint restart capability for many environments that do not currently have this ability.

# Let's take out some of the overhead



# What is the benefit of checkpoint filtering?

- CPU Reduction
  - Checkpoints consume a large amount of CPU
- Elapsed time Reduction
  - Checkpoint/Commit activity increases throughput by reducing run time

# Three Technologies – One Focus
# Throughput

## MAINVIEW Batch Optimizer

- Significantly reduces the *elapsed time* for batch cycles

## Data Accelerator Compression

- Software based compression makes I/O more efficient

## Application Restart Control

- Removes excessive checkpoint activity