SAS® Efficiencies

Lunch and Learn Webinar

brought to you by

Systems Seminar Consultants

www.sys-seminar.com

1-800-997-7081

Questions, Comments

- Technical Difficulties: Call 1-800-263-6317
- We have several hundred attendees, so we won't be able to answer questions live.
- Please email questions or comments to train@sys-seminar.com.
- A copy of the presentation and a recording of the webinar will be emailed out to attendees. This can be shared with people who did not attend.

Systems Seminar Consultants, Inc.

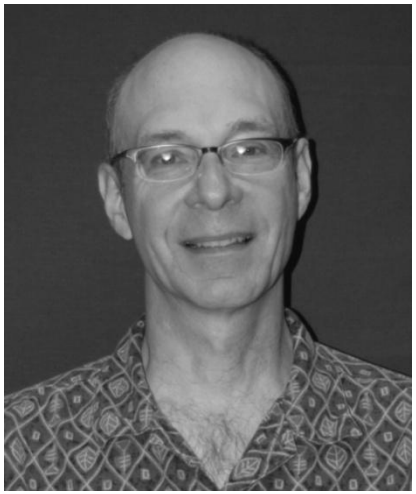
Consulting, Training, and Support

- SAS
- SQL
- ETL
- Reporting Systems
- Business Analytics
- Data modeling
- Automating Processes
- “Big Data”

Apply good IT and systems practices to real life business applications.

Work with Marketing, Finance, Retail, Insurance, Utilities, Manufacturing, Healthcare, Government organizations.

Presenter – Thomas Miron



- Over 20 years of experience as a Senior SAS Consultant
- ETL analysis, process automation, efficiency analysis and reengineering, database modeling, and enterprise level systems assessment.
- Published books: *The How-To Book for SAS/GRAPH Software* and *SAS Software Solutions*.
- Regular presenter at local, regional, and international SAS Users Groups
- Awarded a Best Contributed Paper at SUGI 20

Keys to Efficiency

Efficient programs:

- keep data sizes to a minimum*
- keep data passes to a minimum*
- use efficient data types
- read and write selectively
- make efficient use of SAS statements
- use the correct procedure for the task
- know defaults and override them when desirable
- avoid sorting
- take advantage of data characteristics
- have comments and code clearly
- are easy to understand, run, and maintain

Note:

* These two items matter more than anything else!

Pass and convert less stuff, fewer times!

Today's Topics:

- sorting and indexing
- merging alternatives
- file storage issues

Avoid Sorting

Sorting can often be avoided.

Tips:

- If several procedures require data sorted in the same order, group those PROC steps together rather than re-sorting the data each time.
- If possible, use a CLASS statement.
- Only sort data when necessary.
- If original sort order is not important, specify the NOEQUALS option.
- Take advantage of sorted input files.
- Use indexing when appropriate.
- Remove unnecessary variables and observations.

A Sorting Example

Run procedures that require a given sort sequence as a group…

Adequate

```
proc sort data=midwest;
 by state;
run;
proc print data=midwest;
 by state;
run;
proc sort data=midwest;
 by county;
run;
proc print data=midwest;
 by county;
run;

proc sort data=midwest;
 by state;
run;
proc chart data=midwest;
 by state; vbar x y;run;
```

Better

```
proc sort data=midwest;
 by state;
run;
proc print data=midwest;
 by state;
run;
proc chart data=midwest;
 by state;
 vbar x y;
run;

proc sort data=midwest;
  by county;
run;
proc print data=midwest;
 by county;
run;
```

Sorting Datasets

Some procedures support the CLASS statement

Adequate:

```
proc sort data=midwest;
 by state;
run;
proc means data=midwest;
 by state;
run;
```

Better:

```
proc means data=midwest;
 class state;
run;
```

Sort Data Only When Necessary

If the incoming data is in the correct order, no need to sort in SAS.

Raw Data CENSUS

```
AZ MOHAVE   KINGMAN    132,303
.    .         .          .
IN DUBOIS   JASPER     38,303
.    .         .          .
WI DANE     MADISON    660,393
.    .         .          .
WY NATRONA  CASPER     73,202
```

```
data findcity;
   infile census;
   input @1  state    $2.
         @4  county   $8.
         @13 city     $10.
         @24 populate comma7.;
run;
proc sort data=cities;
  by state;
run;
```

A sort is NOT needed.

Take Advantage of Sorted Input

Two sorted datasets are concatenated and sorted.

```
data both;
  set one
      two;
run;

proc sort data=both;
 by state county city;
run;
```

one

```
AZ MOHAVE   KINGMAN    132,303
IN DUBOIS   JASPER      38,303
WI DANE     MADISON    660,393
WY NATRONA  CASPER      73,202
```

two

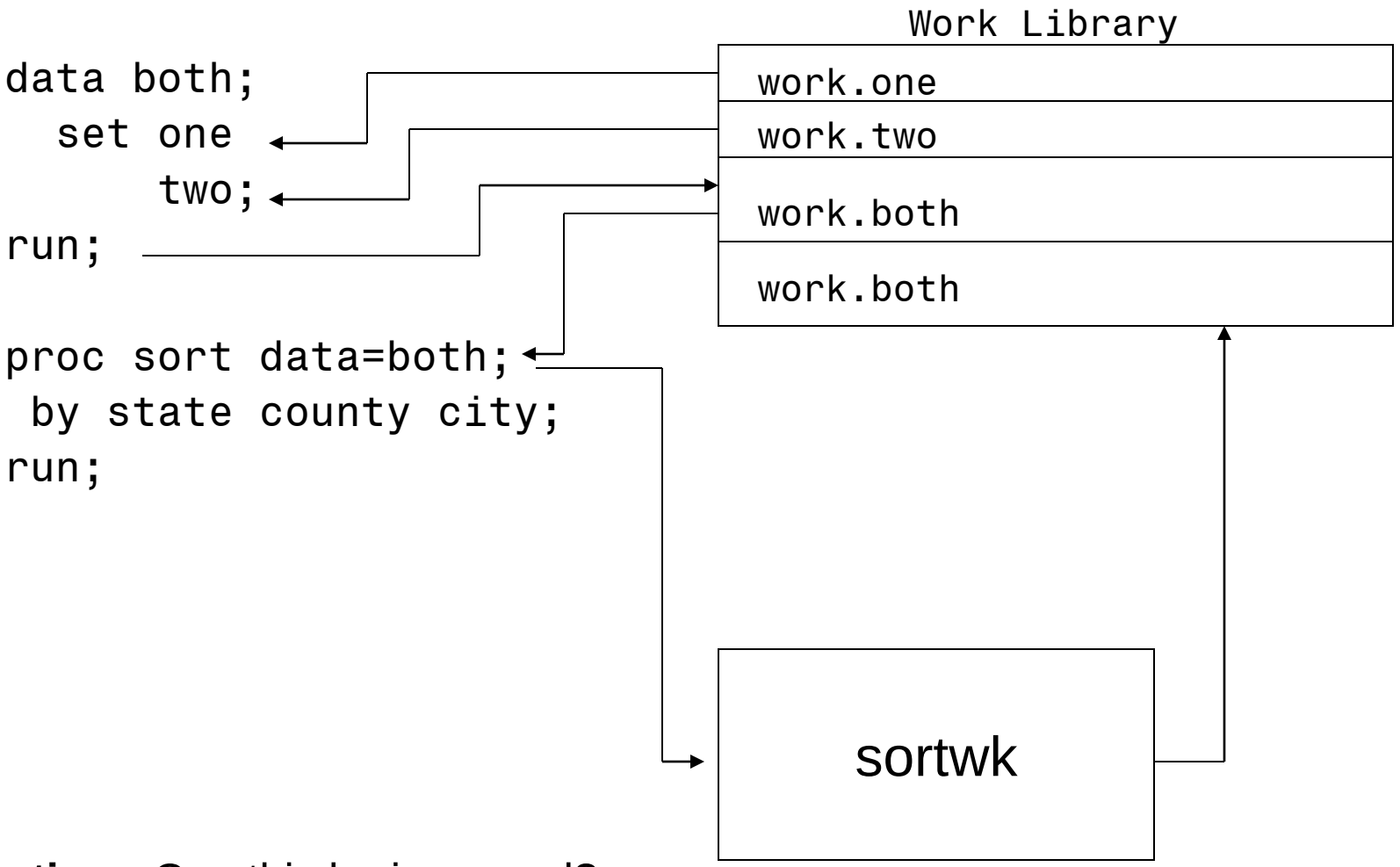
```
IA DUBUQUE  DUBUQUE     15,101
IL COOK     CHICAGI  1,234,212
MN RAMSEY   ST. PAUL   982,393
```

both

```
AZ MOHAVE   KINGMAN    132,303
IA DUBUQUE  DUBUQUE     15,101
IL COOK     CHICAGI  1,234,212
IN DUBOIS   JASPER      38,303
MN RAMSEY   ST. PAUL   982,393
WI DANE     MADISON    660,393
WY NATRONA  CASPER      73,202
```

Space requirements

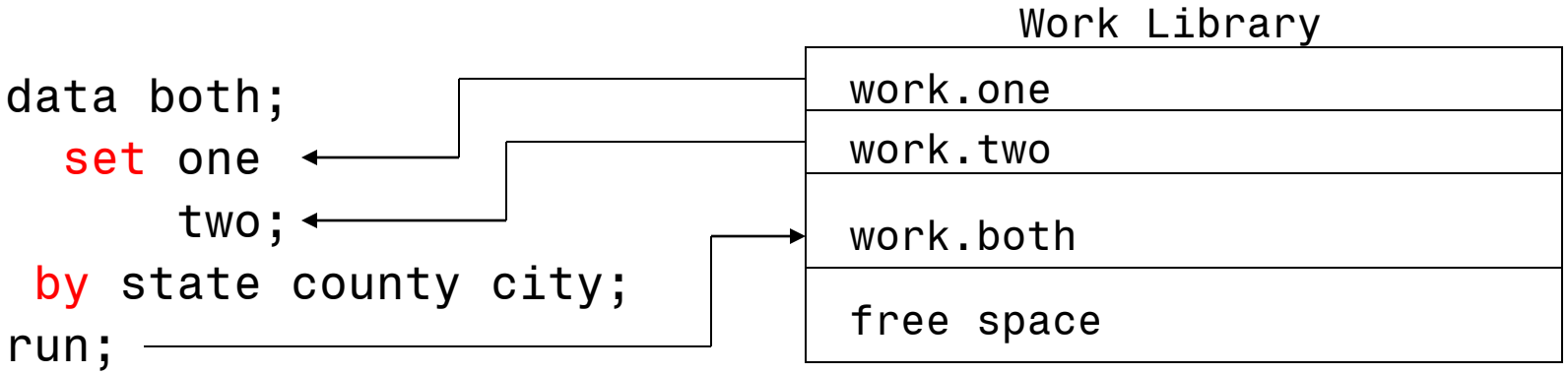
The flow of datasets.



Question: Can this be improved?

Interleave pre-sorted datasets

Interleave = SET statement with BY



Notes:

- Sorting is not required.
- Only one copy of BOTH is built.

Indexing of Datasets

SAS datasets can be indexed by one or more variables.

What is an index?

An auxiliary data structure that assists in the location (selection) of observations specified by the value of one or more key variables

EMP SAS Data Set

Obs	EMPNUM	DEPT	SALES
1	1294	SOFT	55.99
2	1011	**HARD**	22.34
3	3333	DOCU	12.22
4	0938	**HARD**	88.01

Index File

Key	Location
Value	Page(obs,obs...)
SOFT	1(1,...) 2(...)...
HARD	1(2,4,.) 2(...)...
DOCU	1(3,...) 2(...)...

Indexing of Datasets

```
proc print data=emp;
 where dept='HARD';
run;
```

Obs	EMPNUM	DEPT	SALES	
2	1011	HARD	22.34	<----From OBS 2
4	0938	HARD	88.01	<----From OBS 4

Notes:

- SAS Data Views cannot be indexed.

Indexing Advantages

Indexing stores pointers to locate observations more quickly and efficiently.

Advantages of Indexing:

- Indexing allows quick access to a small subset of observations.
- Values returned from the index are returned in sorted order.
- SAS automatically decides whether or not to use an index.
- The data set does NOT need to be sorted.
- SET, MODIFY statements can retrieve specific observations directly.

Indexing Tradeoffs

Disadvantages of Indexing:

- CPU, I/O, memory resources are required to create and maintain the index.
- Manipulation of files outside of SAS (e.g., OS commands) is not recommended.
- Extra disk space on a separate file is required to store the index's data structure. This can be significant with multiple indices on large tables.
- Added complexity.

How Do You Create an Index?

There are a number of ways to index SAS datasets:

- using PROC DATASETS
- using PROC SQL
- using the DATA Step
- Data Set Options on an output SAS dataset.

Creating Indexes in the DATA Step

Index with a Data Set option.

```
* creating indexes in a data step;
data sasdata.sample(index=(state/nomiss
                           name=(lname fname)
                           product));
  set sample1;
run;
```

Notes:

- The input file could also be raw data read with INPUT statements.
- “NOMISS” means rows where the key value is missing (null) are not in the index.

Resulting PROC CONTENTS (Partial)

```
CONTENTS PROCEDURE

Data Set Name: SASDATA.SAMPLE                Observations:          100
Member Type:   DATA                          Variables:             8
Engine:        V612                          Indexes:               3
-----Alphabetic List of Variables and Attributes-----
#    Variable    Type    Len    Pos
-----------------------------------
3    CITY        Char     16     24
7    EXPENSE     Num       8     60
1    FNAME       Char     12      0
2    LNAME       Char     12     12
5    PRODUCT     Char     10     42
8    RATE        Num       8     68
6    SALES       Num       8     52
4    STATE       Char      2     40

-----Alphabetic List of Indexes and Attributes-----
                 Nomiss
#    Index       Option    Var1      Var2
-------------------------------------------
1    NAME                  LNAME     FNAME
2    PRODUCT
3    STATE       YES
```

Index Information

Indexing Guidelines

Some rules of thumb:

- Minimize the number of indexes to reduce the disk storage and update costs.
- Create indexes only if the data file's physical size is large.
- When using WHERE processing to select missing values, do not use the NOMISS attribute.
- Indexing works best when retrieving a small number of observations relative to the total number of observations (<15%).
- Indexing works best on values not frequently updated and values that are uniformly distributed.
- Only index the columns you will actually be working with.

KEY= SET statement Option

The SET statement KEY= option retrieves rows by key (indexed column) rather than reading the complete data set sequentially.

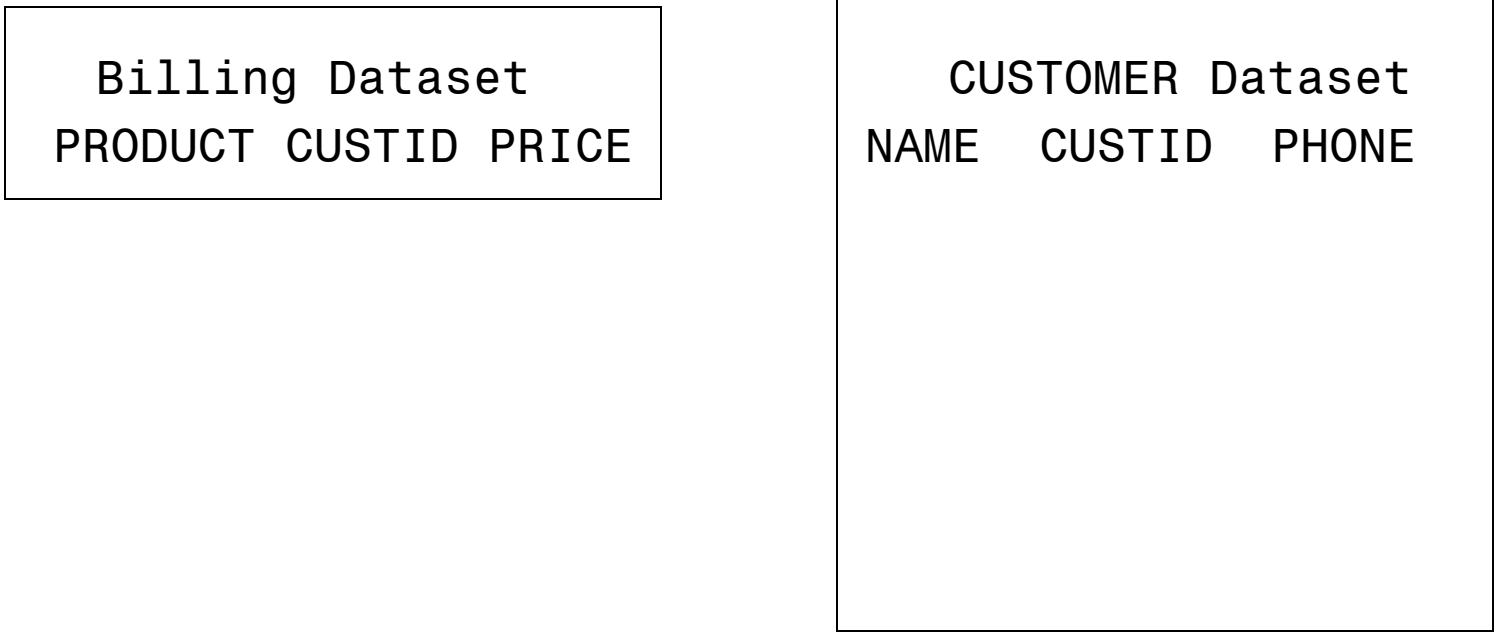
- You must create an index on the KEY= columns.
- This is much faster than reading an entire table sequentially to find a relatively small number of “key” values.

- KEY= can also be used with the MODIFY statement.

Using Indexes to Improve MERGE

Example:

You have a CUSTOMER file with 1 million rows sorted by name. You have a small BILLING file with 100 selected records. You need to join them by the KEY CUSTID.



Traditional Merge

Sort the CUSTOMER file by CUSTID, MERGE discards most records.

```
data customer;                  /* build customer file*/
 . . . rest of data step
proc sort data=customer;        /* sort customer file */
 by custid;                     /* by key to billing  */
run;                            /* end of sort        */
data report;                    /* build report       */
 merge billing(in=onbill)       /* join billing       */
       customer(keep=custid     /* needed cols        */
                name phone);    /* and customer       */
 by custid;                     /* custid is key      */
  if onbill;                    /* only if on billing */
run;                            /* end of step        */
```

Notes:

- Our job sorts all 1 million rows.
- MERGE reads all 1 million rows sequentially, discards all but 100.

Merge Alternative

Index CUSTOMER by CUSTID, retrieves 100 rows only, based on CUSTID.

```
options errors=0;
data customer(index=(custid/unique/nomiss));
 . . . rest of data step
data report;
 set billing;                  /* Read a billing          */
 set customer(keep=            /* Now read the one row    */
    custid name phone)         /* with the just read      */
   key=custid/unique;          /* CUSTID key              */
 if _iorc_ ne 0 then do;       /* If you didn't find it   */
   name='          ';          /* set to null or data from*/
   phone='         ';          /* the last good read will */
 end;                          /* be used                 */
run;                           /* end of step             */
```

Notes:

- This program only reads CUSTOMER 100 times.

Merge Alternative (continued)

Logic of the previous program:

- reads one row from the BILLING data set
- read row with CUSTID directly from CUSTOMER table
- if it doesn't exist, _IORC_ will be non-zero
- set values to missing as needed

Merging

The following times are from a test case with 1 million customer records and 50 desired products (under v.6.12):

- Data step MERGE: 14.15 seconds
- Data step with indexed SET statement: 0.55 seconds.

Notes:

- Times do not include sorting or building the index.

INDEX Keys Example

This technique can also be used if the first file is a flat file.

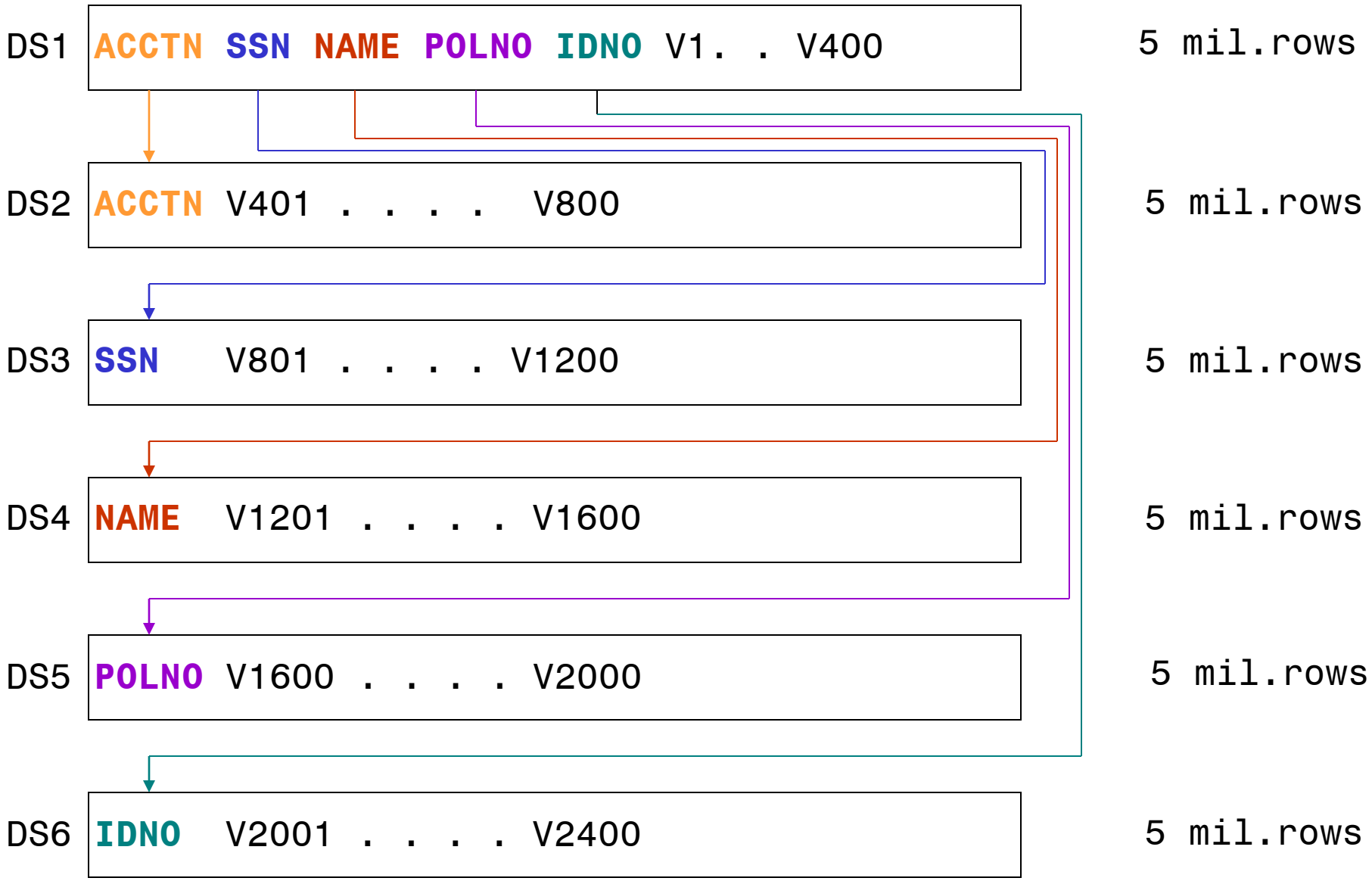
```
options errors=0;
data mydata;
infile rawdata;
input @1 custid $char5.;       /* read from flat file  */
set numbers key=custid / unique;
                               /*is in the desired list*/
if _iorc_  ne 0 then           /* not found            */
  do;
    delete;                    /* delete it            */
  end;
output mydata;                 /* found it, output     */
run;


UNIQUE means always start the key search from row 1.
```

A Difficult Space Problem

An insurance company needs to join six huge SAS files.

- Each of the files has app. 5 million rows and hundreds of variables.
- Each file is sorted by a different key variable.
- The first file is the driver file.
  (i.e. keep all rows from ds1 regardless if there is a match)

A Diagram of the Files



Our First Merge

The File to be sorted now has 5 million rows, 800+ variables.

```
data d12;
 merge ds1(in=on1)
       ds2;
 by acctn;
 if on1;
run;

proc sort data=d12;
 by ssn;
run;
```

```
ACCTN SSN NAME POLNO IDNO V1. .   V800
  .    .    .    .    .    .  .    .
```

Our Second Merge

The File to be sorted now has 5 million rows, 1200+ variables.

```
data d13;
 merge d12(in=on1)
       ds3;
 by ssn;
 if on1;
run;

proc sort data=d13;
 by name;
run;
```

```
ACCTN SSN NAME POLNO IDNO V1. .   V1200
  .    .    .    .    .    .  .    .
```

Our Third Merge

The File to be sorted now has 5 million rows, 1600+ variables.

```
data d13;
 merge d12(in=on1)
       ds3;
 by ssn;
 if on1;
run;

proc sort data=d13;
 by name;
run;
        OUT OF SPACE IN WORK LIBRARY !!!!
```

Question: We are out of space with three files to go. What can we do now?

Our Choices

Possible solutions:

- increase work space (couldn't get enough)
- tried some views (couldn't get sort space)
- go to permanent disk (couldn't get enough)
- go to tape (still couldn't get enough sort space)

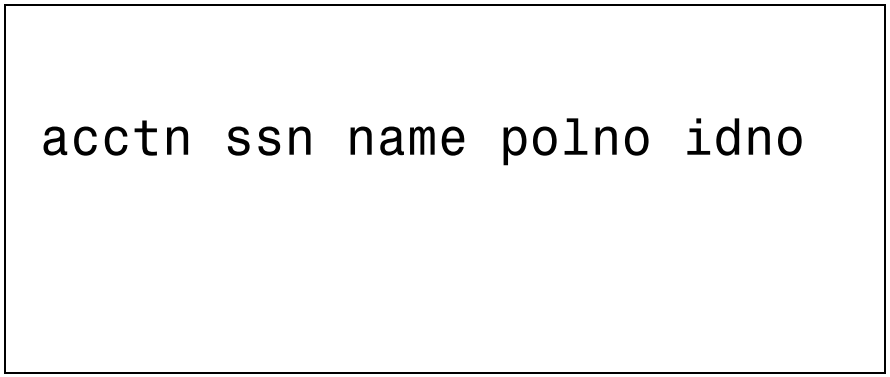
Question: What now? Divide and Conquer!

Our Solution

Take advantage of sorts, pass less data.

1. Build a file with all rows but only the key variables.
2. Decide what the final sort will be (ACCTN).

```
data keyds;
 set ds1(keep=acctn ssn name polno idno);
run;
```



Our Solution (continued)

3. Sort and merge the narrower KEYDS file with the DS3, sort by ACCTN.

```
proc sort data=keyds;
 by ssn;
run;
data key3;
 merge keyds(in=on1 keep=acctn ssn)
       ds3(in=on3);
 by ssn;
 if on1 and on3;
run;
proc sort data=key3;
 by acctn;
run;
```

KEY3

ACCTN SSN V801 . . . . V1200

Notes:

- This file has < 5 mil. Rows, 402 variables and ACCTN.

Our Solution (continued)

4. Repeat for DS4, DS5, DS6.

```
proc sort data=keyds;
 by name;
run;
data key4;
 merge keyds(in=on1 keep=acctn name)
       ds4(in=on4);
 by name;
 if on1 and on4;
run;

proc sort data=key4;
 by acctn;
run;
```

Notes:

- All files are < 5 mil. Rows, app. 400 variables.
- All files are sorted into ACCTN order.

Our Final Step

5. Merge the presorted datasets together to make the final dataset.

```
data final;
 merge ds1(in=on1)
       ds2
      key3
      key4
      key5
      key6;
 by acctn;
 if on1;
run;
```

Notes:

- The final file has 5 million rows and 2405 variables.
- The final file is sorted into ACCTN order.

Disk Space Issues

Topics:

- why we run out of space
- how to increase work space
- how to clean up during a job
- how to use permanent datasets
- how to use SAS data step views
- when to use tape instead of disk
- how to free tape datasets early
- how to minimize tape drives

SAS Space Issues

We frequently run out of disk space in the SAS WORK library and when processing large files, our SAS jobs fail.

Possible causes:

- Too much data for the WORK library.
- Unnecessary temporary SAS files are being kept.
- Unnecessary records and variables are present.
- Multiple copies of a SAS file are kept while recreating or sorting.
- Variable lengths are greater than necessary.

SAS Space Issues

What can we do about it?

Possible solutions:

1. Use LENGTH statement to minimize unneeded wasted space
2. Clean up the program to avoid carrying any unneeded rows
3. Use the COMPRESS= option
4. Use logical *views* of the data instead of physical data files.
5. Make the WORK library bigger.
6. Use an alternate temporary or permanent disk library.
7. Use tape datasets instead of disks (z/OS systems)

z/OS SAS WORK Library

SAS WORK Library:

- This is a required z/OS sequential dataset normally allocated at the beginning of each SAS session.
- It has a finite amount of space set by the invoking JCL batch procedure or CLIST.

SAS WORK Library Space:

- This can be allocated in various increments: bytes, cylinders, tracks, blocks.
- It is usually specified as a single primary space request and a secondary space request that the system will allocate up to 15 times.
- The actual amount of space varies at each site; view the appropriate proc or CLIST to check what is available to your SAS session.
- Your site may restrict space allocations via data management classes.

z/OS SAS WORK Library Example

A partial batch SAS proc:

```
//SAS   PROC   ENTRY=SASXAL,
//             WORK='20,10'
//WORK DD      UNIT=SYSDA,SPACE=(CYL,(&WORK)),
```

A partial SAS CLIST:

```
PROC 0
     WORK('20,10')
ALLOC F(&DDWORK) SP(&WORK) CYL UNIT(SYSDA)
```

Notes:

- Space can be allocated in cylinders (CYL). A CYL is approximately 850,000 bytes on a 3390 device.
- Space can be allocated in tracks (TRK). A TRK is approximately 56,664 bytes.
- Space can also be an actual byte count such as 6160.

z/OS SAS WORK Library Example (continued)

The previous examples contain 20 cylinders plus up to 15 additional allocations of 10 cylinders (150 cylinders) for a total of 170 cylinders.

Formula for Total WORK Library

(CYL or TRK or block allocation unit) * (number required or allocated) = bytes

In this Example

850,000 * 170 = 144.5 million bytes

Estimated Required WORK Space

To estimate how much space we *need*, use a simple calculation with 20% for overhead:

Space-in-Bytes = (obs length) * (number of obs) * (1.2)

Notes:

- This calculation allows 20% for overhead.
- You can either estimate or use the output from PROC CONTENTS.
- The space required for all datasets in the job must be added together.
- This total must fit in the WORK library.

Space Issues

If an existing dataset is being rewritten, the *old* copy is kept until the *new* dataset is successfully written.

Example:

```
proc sort data=x;
 by name;
run;
```

Space Issues

A typical job creates the following datasets:

- A contains 100,000 obs, 290 obs length, or 34.8 million bytes
- B contains 200,000 obs, 187 obs length, or 44.8 million bytes
- C contains 100,000 obs, 333 obs length, or 40.0 million bytes

The work space required:

WORK space is (CYL,(20,10)) or 144.5 million bytes**.**

PROC SORT Example

CODE

```
//  EXEC  SAS
data a;
..........
run;
```

WORKSPACE

A	**34.8 mil bytes**
Free	109.7 mil bytes

```
proc sort data=a;
    by id;
run;
```

A	**34.8 mil bytes**
A	**34.8 mil bytes**
Free	74.9 mil bytes

```
data b;
...........
run;
```

A	**34.8 mil bytes**
B	**44.8 mil bytes**
Free	64.9 mil bytes

PROC SORT Example (continued)

```
proc sort data=b;
     by id;
run;


data c;
     merge a b;
     by id;
     ............
run;

proc sort data=c;
     by name;
run;
```

A	34.8 mil bytes
B	44.8 mil bytes
B	44.8 mil bytes
Free	20.1 mil bytes

A	34.8 mil bytes
B	44.8 mil bytes
C	40.0 mil bytes
Free	24.9 mil bytes

A	34.8 mil bytes
B	44.8 mil bytes
C	40.0 mil bytes
C	Not Enough Space!

Notes:

- The step fails because the work space does not have enough room for 1 copy of A, 1 copy of B, and 2 copies of dataset C.

Alternative A: Increase the WORK Space

Without altering code in the PROC SORT example, the WORK space must be least 159.6 million bytes.

- Divide 159.6 million by 850,000 (1 CYL) , which gives a WORK library of 188 cylinders.
- Add a little extra for growth (without asking for too much) such as:

```
//  EXEC SAS.WORK='200,10'     (Batch)
  or SAS WORK('200,10')        (Interactive)
```

Alternative B: Clean Up the Code

Less WORK space is needed if the code is cleaned up. After C is built, delete A and B.

```
proc sort data=b;
 by id;
run;
data c;
   merge a b;
   by id;
run;
proc datasets library=work;
 delete a b;
run;
proc sort data=c;
  by name;
run;
```

A	34.8 mil bytes
B	44.8 mil bytes
B	44.8 mil bytes
Free	20.1 mil bytes

A	34.8 mil bytes
B	44.8 mil bytes
C	40.0 mil bytes
Free	24.9 mil bytes

C	40.0 mil bytes
C	40.0 mil bytes
Free	64.5 mil bytes

More Cleanup

Decreasing dataset size:

- Use KEEP or DROP to discard unneeded variables.
- Use LENGTH to specify the minimum variable widths.
- Use DELETE, IF, or WHERE to include only necessary rows.

Alternative C: Use Another Library for Space

Route output to another library.

- SAS software refers to these as *permanent* SAS libraries
- SAS does not automatically delete them when the SAS session ends
- These libraries remain after a job ends.

Permanent SAS Libraries

Use permanent SAS libraries when one or more of the following is true:

- The SAS datasets require a lot of resources to build.
- The tables with exactly the same data are being built repeatedly.
- Other SAS sessions may use the data.
- The SAS session may be restarted part way through the job.
- The data is to be stored on tape.
- The amount of storage exceeds the WORK library space.

Permanent SAS Libraries (continued)

Using permanent SAS libraries requires the job to:

- Allocate the space or tape via some system control (JCL), or the SAS LIBNAME statement. (z/OS)
- Associate a *libref* (or DDNAME in z/OS) to the library.
- Reference all datasets in the library with a two-level name, where the first level is the *libref* in the LIBNAME or DDNAME statements.

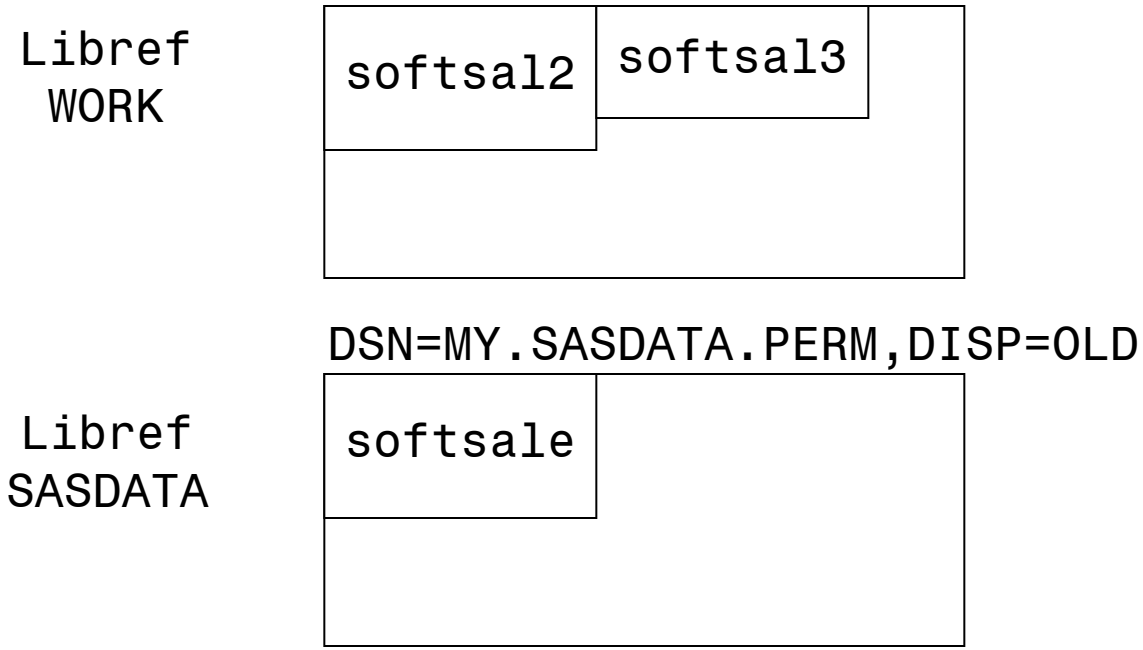
Permanent SAS Library Example (z/OS)

Sample SAS job:

```
//SASJOB1 JOB accting
//  EXEC  SAS
//RAWIN   DD DSN=MY.INPUT,DISP=SHR
//SASDATA DD DSN=MY.SASDATA.PERM,DISP=OLD
data sasdata.softsale;
  infile rawin;
  input . . . ;
run;
data softsal2;
infile rawin;
input . . .;
run;
data softsal3;
set sasdata.softsale;
run;
```

Permanent SAS Library Example (continued)

The diagram below shows the WORK and the SASDATA libraries.



Advantage of Using Permanent Datasets

Permanent datasets can be accessed in a later job without rebuilding the dataset.

```
//SASJOB2 JOB accting
//        EXEC   SAS
//SASDATA DD     DSN=MY.SASDATA.PERM,DISP=SHR
proc print data=sasdata.softsale;
WHERE division='S';
run;
```

Managing SAS Libraries

If data is to remain on disk and tape, we need tools to manage the SAS libraries. Some tools we can use:

```
PROC DATASETS
PROC CONTENTS
PROC COPY
Interactive commands
```

Notes:

- These PROCs can be used to display contents, delete members, copy libraries, and more.
- These PROCs can be used in batch or they can be run interactively.
- The interactive commands (ex. LIBNAME) are friendlier and make it easy to maintain SAS permanent libraries.
- In interactive SAS point and click can be used instead of coding SAS procedures.

COMPRESS= option

COMPRESS applies a compression algorithm to SAS tables.

```
proc sort data=b;
    by id;
run;

data c( compress=yes );
    merge a b;
    by id;
run;

proc sort data=c;
    by name;
run;
```

A Tape Drive Problem (z/OS)

Tape drives are normally tied up for the entire SAS job.

```
//SASJOB1 JOB accting
//  EXEC  SAS
//RAWIN   DD DSN=MY.INPUT,DISP=SHR   * tape file;
   data softsale;
        infile rawin;                 * reads tape;
        input . . . ;
   run;                               * done with tape;
   proc sort data=softsale;           * tape is still  ;
    by name;run;                      * around         ;
   proc means data=softsale;run;
   proc freq data=softsale;
   table division;run;
```

Notes:

- You are keeping a tape drive allocated that someone else could use.

Solving the Previous Tape Drive Problem

FREE=CLOSE deallocates the dataset when it is closed.

```
//SASJOB1 JOB accting
//  EXEC  SAS
//RAWIN   DD DSN=MY.INPUT,DISP=SHR,FREE=CLOSE
   data softsale;
        infile rawin;                  * reads tape;
        input . . . ;
   run;                                * done with tape;
   proc sort data=softsale;            * tape is freed;
    by name;run;
   proc means data=softsale;run;
   proc freq data=softsale;
   table division;run;
```

Notes:

- You cannot reference the tape after the first step.

Another Job That Wastes Tape Drives

Two physical tape drives are allocated, even though only one at a time is needed.

```
//SASJOB1 JOB accting
//  EXEC  SAS
//RAWIN   DD DSN=MY.INPUT,DISP=SHR     * tape one;
//RAWIN2  DD DSN=MY.INPUT2,DISP=SHR    * tape two;
   data softsale;
        infile rawin;                   * reads tape;
        input . . . ;
   run;                                 * done with tape;
   data softsal2;
        infile rawin2;                  * reads tape;
        input . . . ;
   run;
```

Solving the Previous Problem

UNIT=AFF mounts both tapes on the same drive.

```
//SASJOB1 JOB accting
//  EXEC  SAS
//RAWIN   DD DSN=MY.INPUT,DISP=SHR     * tape one;
//RAWIN2  DD DSN=MY.INPUT2,DISP=SHR,   * tape two;
//         UNIT=AFF=RAWIN              * use same drive
                                         as RAWIN
   data softsale;
        infile rawin;                    * reads tape;
        input . . . ;
   run;                                  * done with tape;
   data softsal2;
        infile rawi2;                    * reads tape;
        input . . . ;
   run;
```

Questions, Comments

- Please email questions or comments to train@sys-seminar.com or contact us at 1-800-997-7081.

- A copy of the presentation and a recording of the webinar will be emailed out to attendees. This can be shared with people who did not attend.

Upcoming Lunch and Learn

Getting the Most Out of SAS Enterprise Guide

October 11, Noon Central 2012

Jennifer First

We will email an invitation to our mailing list and attendees

Contact Us

SYSTEMS SEMINAR CONSULTANTS, INC.

2997 Yarmouth Greenway Drive • Madison, WI 53711
(608) 278-9964
www.sys-seminar.com

SAS Alliance Silver Member

Thomas Miron
Senior Consultant
tmiron@sys-seminar.com

General Questions
train@sys-seminar.com