Top 10 SAS coding efficiencies

Charu Shankar
SAS Education
Wisconsin Illinois SAS Users Conference
Milwaukee
26 Jun 2013

Agenda

Data Worker Rule #1

Top 10 SAS coding efficiencies

1. CPU saving - Boiling down or reducing your data
2. CPU saving - Conditional processing
3. CPU saving - Do not reduce the length of numeric variables

4. I/O saving - Reduce multiple passes through data
5. I/O saving - Modify variable attributes
6. I/O saving - Process only necessary observations
7. I/O saving - Process only necessary variables

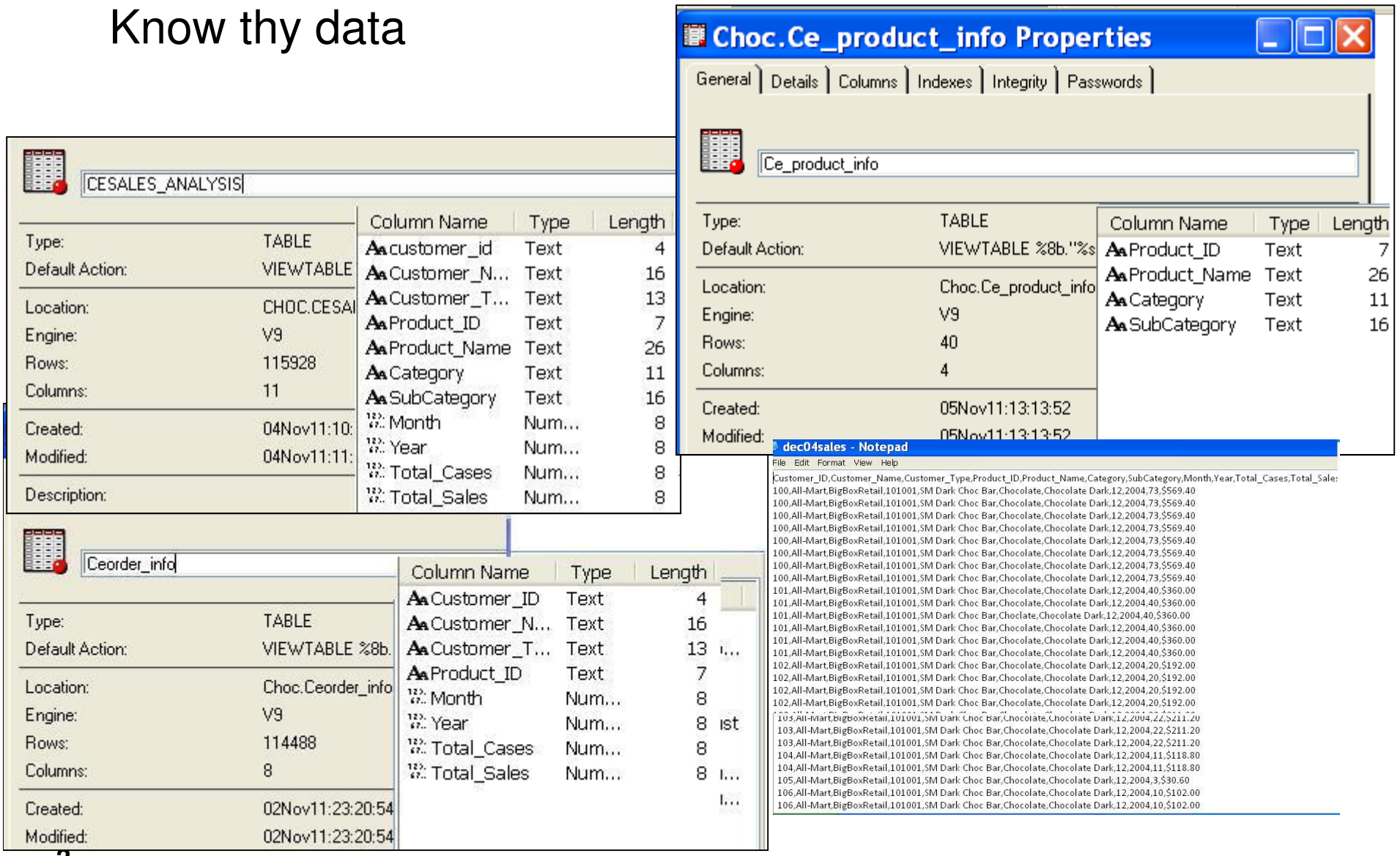
8. Space saving - Store as character data

9. Memory saving - Use the BY statement instead of CLASS

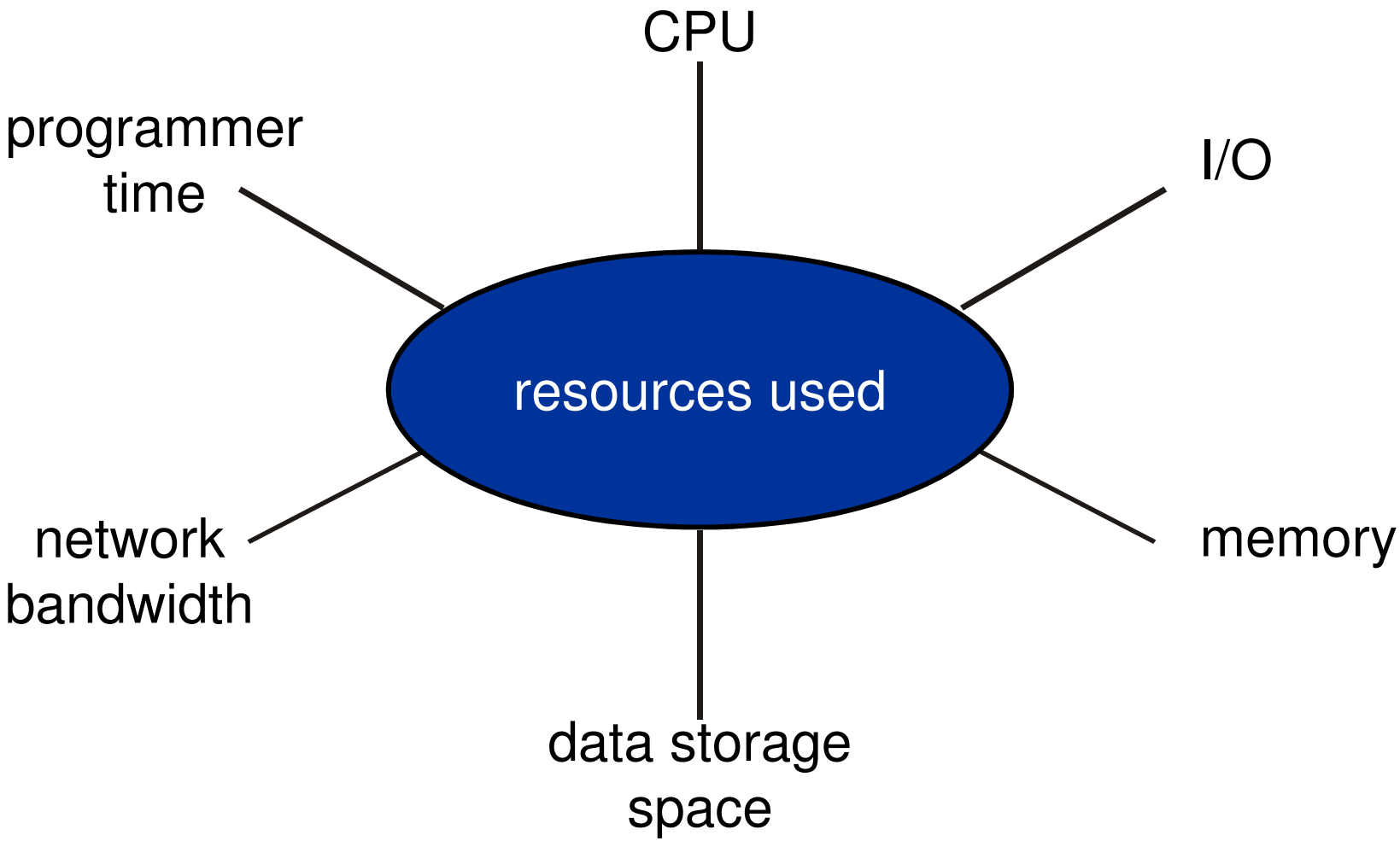
10. Programmer Time Saving
11. Q&A

Data worker's rule #1

Know thy data



What 6 Resources Are Used?



A few definitions

CPU	is a measurement of the amount of time that the central processing unit uses to perform requested tasks such as calculations, reading and writing data, conditional and iterative logic, and so on.
I/O	is a measurement of the read-and-write operations performed when data and programs are moved from a storage device to memory (input) or from memory to a storage or display device (output).
Memory	is the size of the work area required to hold executable program modules, data, and buffers.
Data storage space	is the amount of space that is required to store data on a disk or tape.
Programmer time	is the amount of time required for the programmer to write and maintain the program. This can be decreased through well-documented, best programming practices.
Network bandwidth	is the amount of data that can pass through a network interface over time. This time can be minimized by performing as much of the subsetting and summarizing as possible on the data host before transferring the results to the local computer. The network bandwidth is heavily dependent on network loads.

Computer Processing

You @ your terminal

Applications

O/S
Email
SAS

CPU

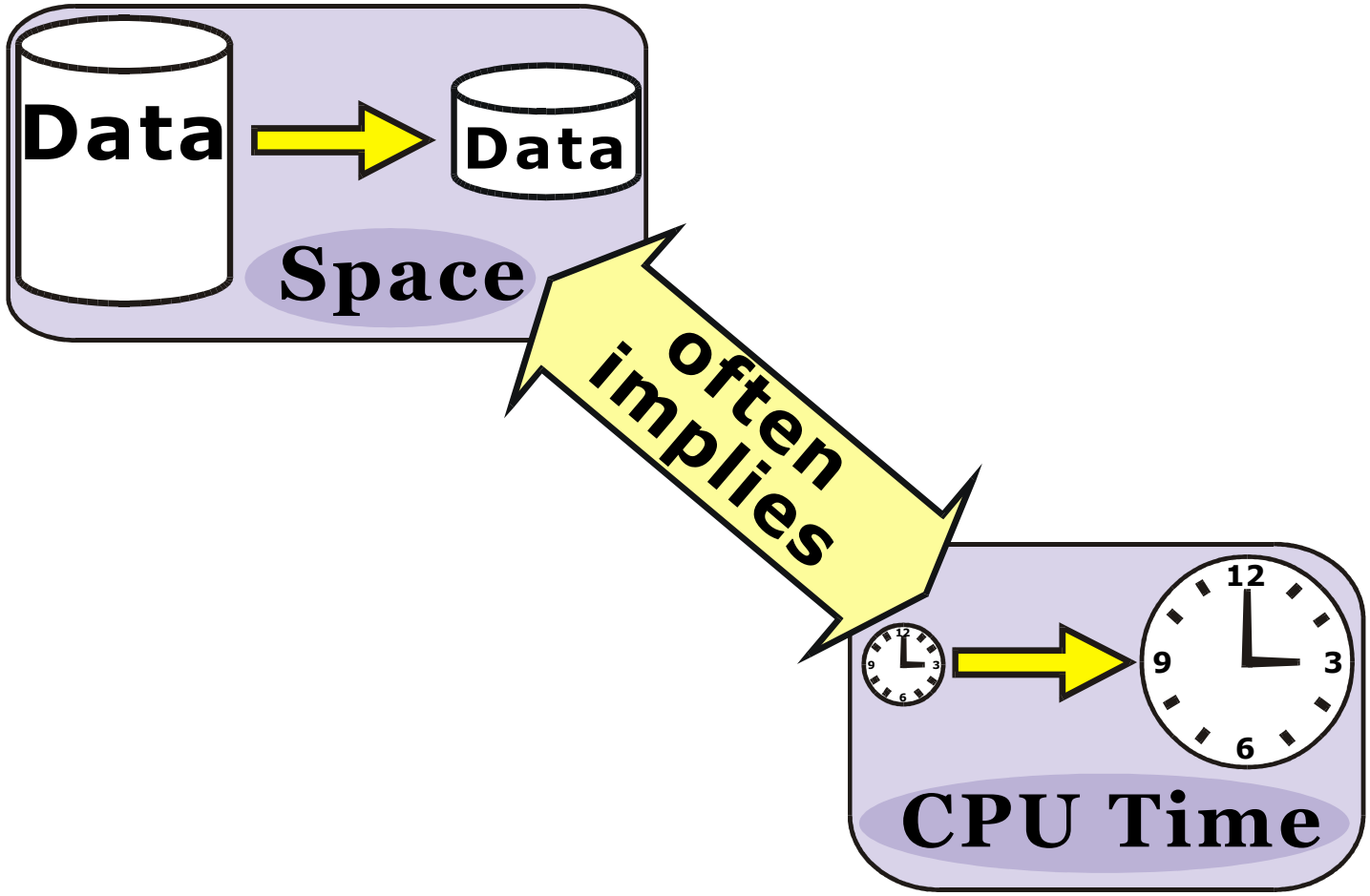
Operating System | Email Software | SAS Software | SAS data

Memory

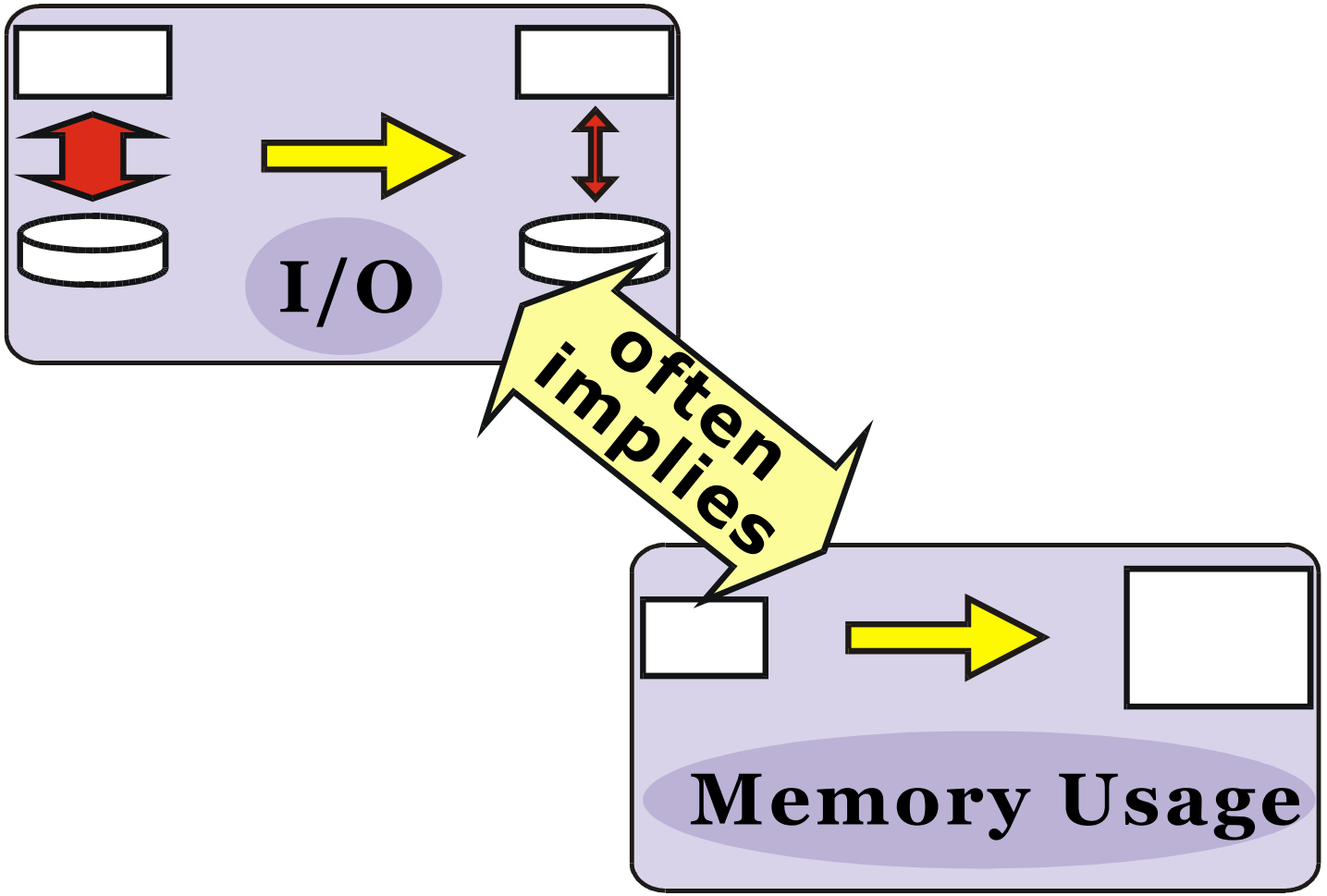
Data

Tables

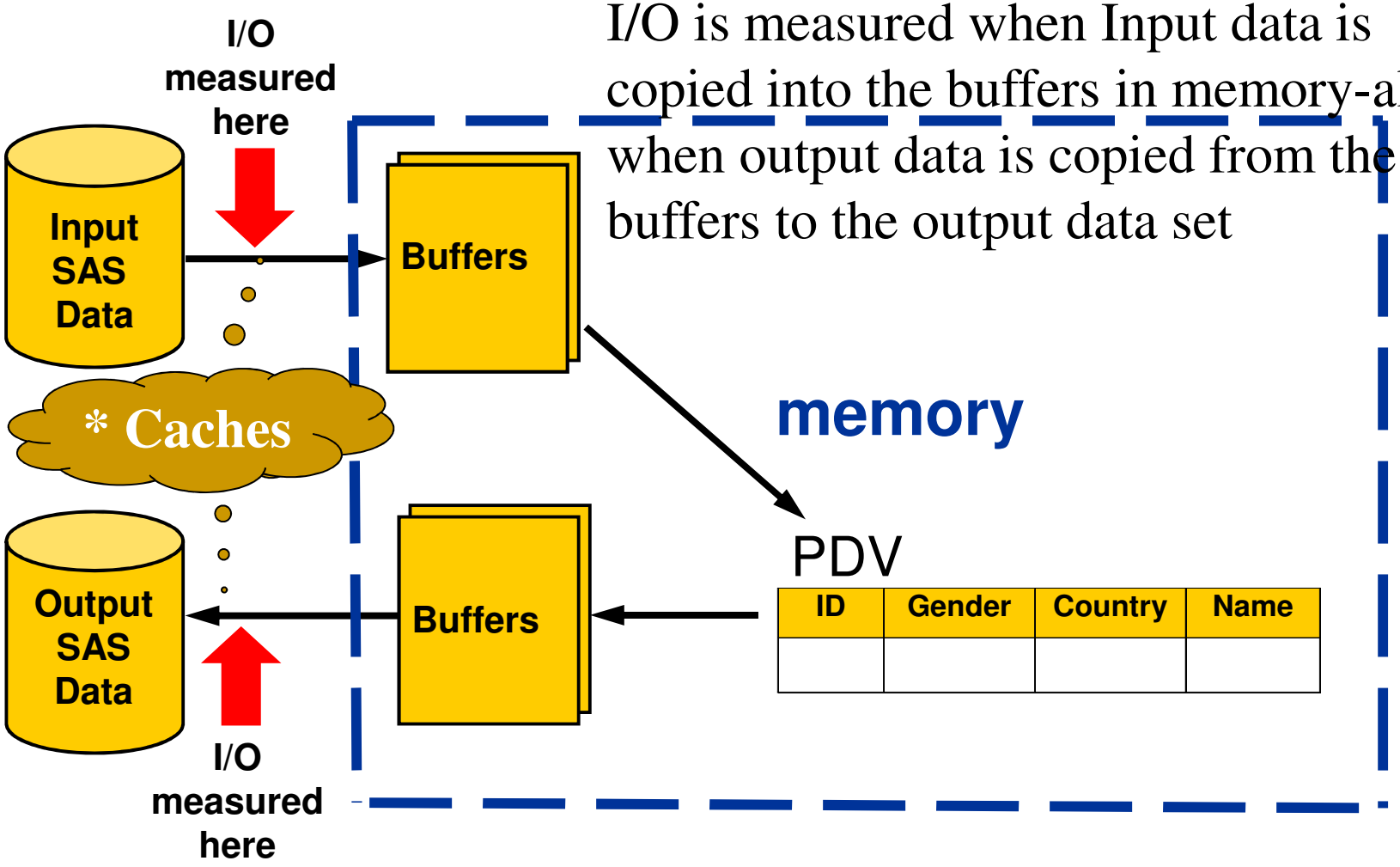
Understanding Efficiency Trade-offs



Understanding Efficiency Trade-offs



Where Is I/O Measured?



I/O is measured when Input data is copied into the buffers in memory-also when output data is copied from the buffers to the output data set

* Windows and UNIX Only

What's the only answer to "What's the best way to do this?"

It Depends!!

Saving CPU-the computer's brain

#1. Boiling down your data

#2. Use conditional logic

#3. Do not reduce the length of numeric variables.

Make room in your brain for important stuff

Technique 1
Subsetting IF Statement at Bottom of Step

Create a new SAS data set from choc.cesales_analysis. The new dataset should contain 2 new variables & we're interested only in the Gourmet customer

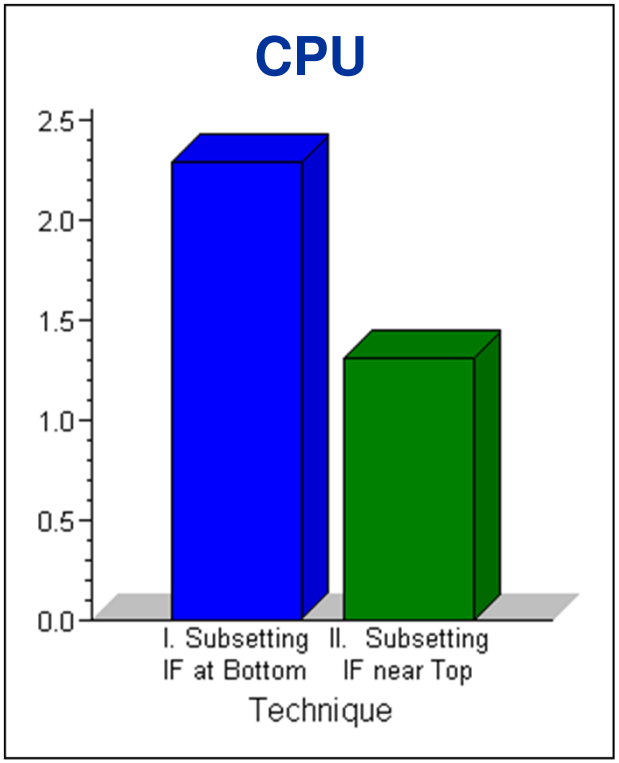
```
data totals;
set choc.cesales_analysis;
do month=1 to 12;
totcase + total_cases;
totsales + total_sales;
end;
cust='******';
custorder=1;
if customer_type='Gourmet';
run;
```

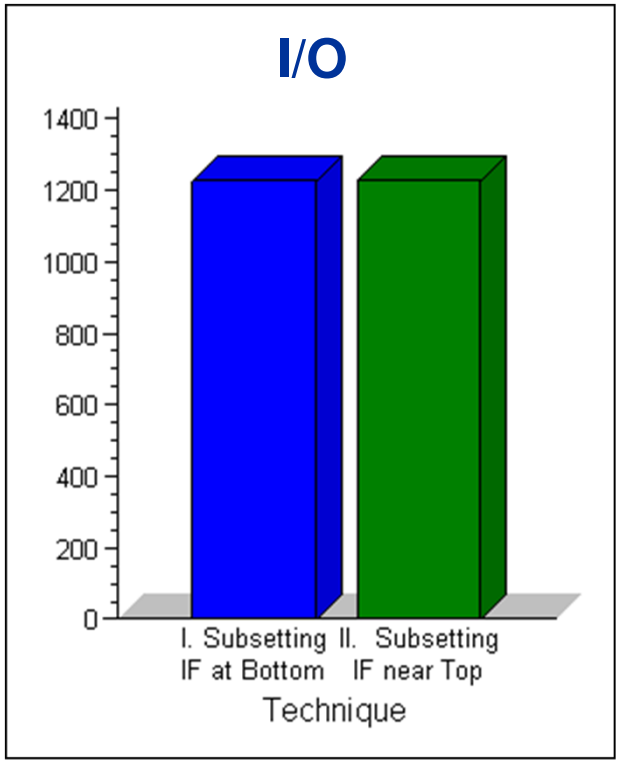
Technique 2
Subsetting IF Statement as High as Possible

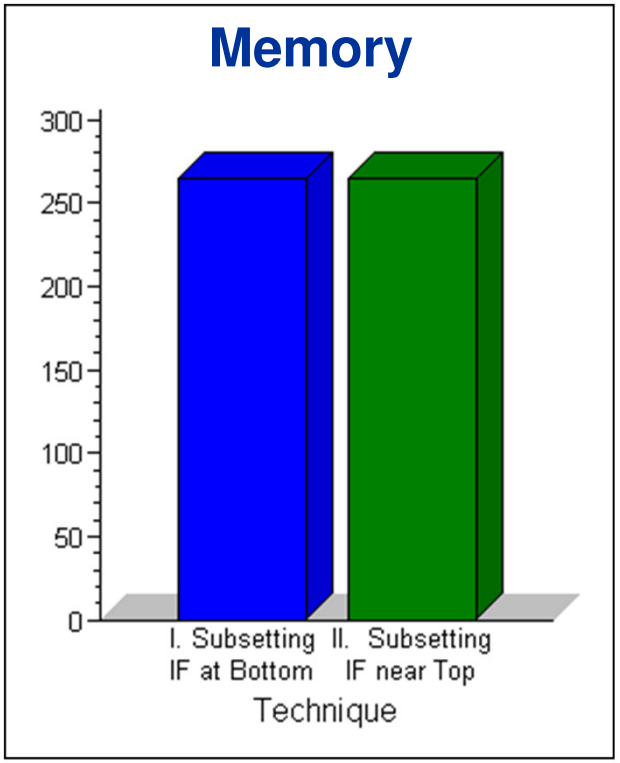
```
data totals;
set choc.cesales_analysis;
do month=1 to 12;
totcase + total_cases;
totsales + total_sales;
end;
if customer_type='Gourmet';
cust='*****';
custorder=1;
run;
```

The Result- Let's compare techniques

Technique	CPU	I/O	Memory
I. Subsetting IF at Bottom	2.3	1226.0	265.0
II. Subsetting IF near Top	1.3	1226.0	265.0
Percent Difference	42.8	0.0	0.0







#2 Use conditional Logic

IF-THEN/ELSE	Executes a SAS statement for observations that meet a specific condition
SELECT	Executes one of several statements or groups of statements

Technique 1
Using Parallel IF Statements

create variable named **`var`**, based on existing variable **`var1`**.

```
data _null_;
   length var $ 30;
   retain var2-var50 0 var51-var100 'ABC';
   do x=1 to 10000000;
      var1=10000000*ranuni(x);
      if var1>1000000 then var='Greater than 1,000,000';
       if 500000<=var1<=1000000 then var='Between 500,000 and
1,000,000';
       if 100000<=var1<500000 then var='Between 100,000 and 500,000';
       if 10000<=var1<100000 then var='Between 10,000 and 100,000';
       if 1000<=var1<10000 then var='Between 1,000 and 10,000';
       if var1<1000 then var='Less than 1,000';
   end;
run;
```

Technique 2
Using ELSE-IF Statements

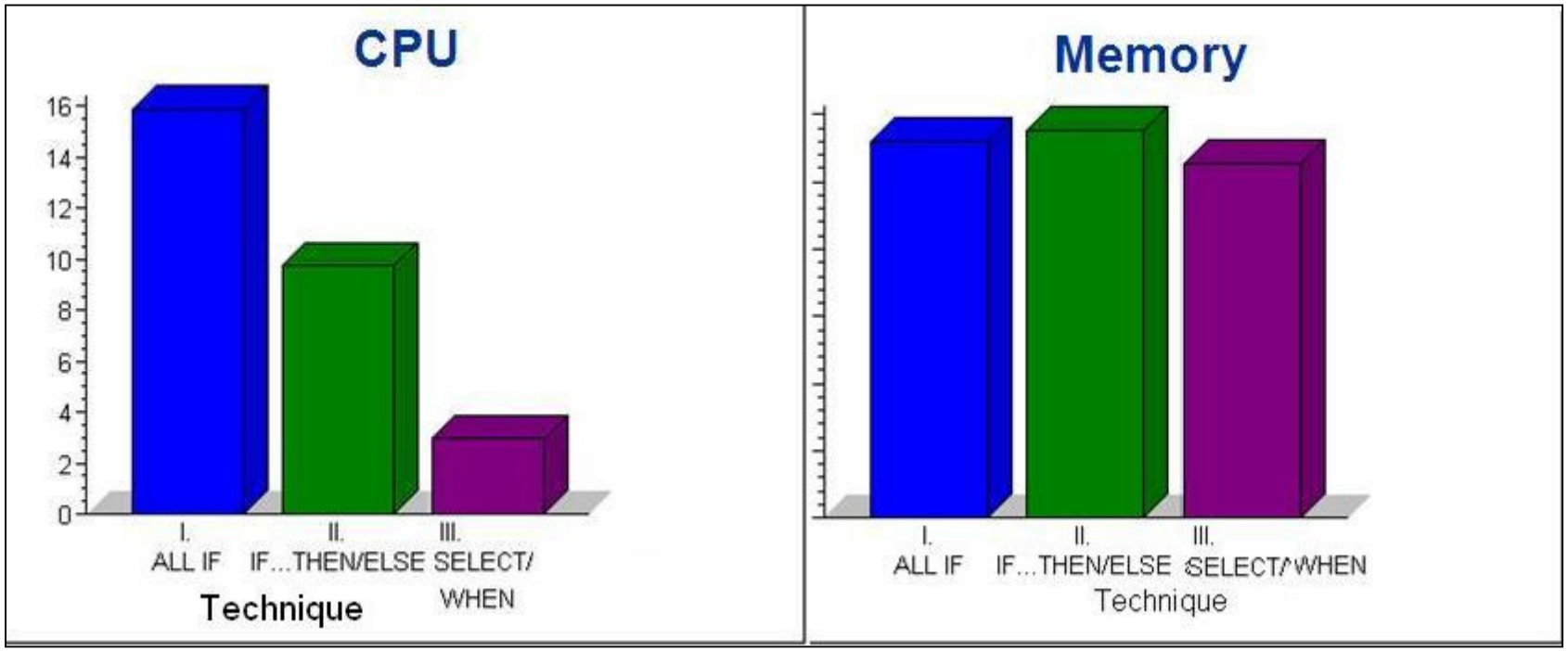
```
data _null_;
   length var $ 30;
   retain var2-var50 0 var51-var100 'ABC';
   do x=1 to 10000000;
      var1=10000000*ranuni(x);
      if var1>1000000 then var='Greater than 1,000,000';
      else if 500000<=var1<=1000000 then var='Between 500,000 and
1,000,000';
      else if 100000<=var1<500000 then var='Between 100,000 and
500,000';
      else if 10000<=var1<100000 then var='Between 10,000 and 100,000';
      else if 1000<=var1<10000 then var='Between 1,000 and 10,000';
      else if var1<1000 then var='Less than 1,000';
   end;
run;
```

Technique 3
Using a SELECT Block

```
data _null_;
   length var $ 30;
   retain var2-var50 0 var51-var100 'ABC';
   do x=1 to 10000000;
      var1=10000000*ranuni(x);
select;
when (var1>1000000) var='Greater than 1,000,000';
 when (500000<=var1<=1000000) var='Between 500,000 and 1,000,000';
 when (100000<=var1<500000) var='Between 100,000 and 500,000';
  when (10000<=var1<100000)  var='Between 10,000 and 100,000';
  when (1000<=var1<10000)  var='Between 1,000 and 10,000';
   when (var1<1000 ) var='Less than 1,000';
end;
end;
run;
```

The result-Let's compare Techniques

Technique	CPU	I/O	Memory
I. ALL IF Statements	15.9	6797.0	280.0
II. ELSE-IF Statements	9.7	6797.0	288.0
III. SELECT/WHEN Block	3.0	6795.0	263.0



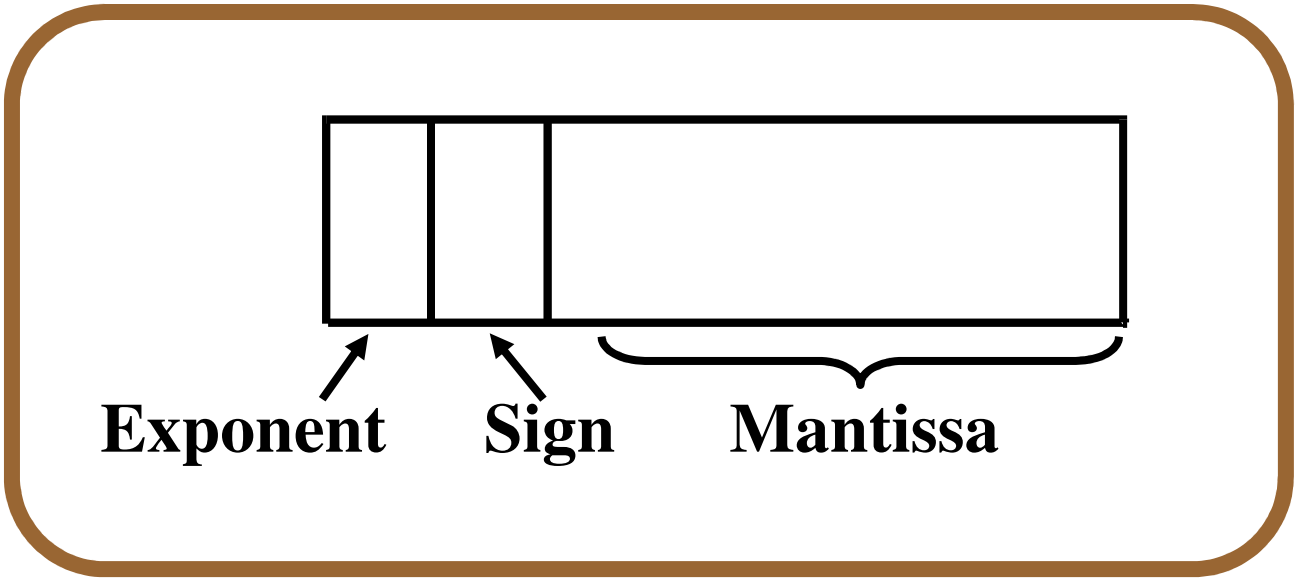
The I/O for each technique is the same.

#3 Do not reduce the length of numeric data

Default Length of Numeric Variables

The number 35,298 can be written as follows:

SAS stores numeric variables in floating-point form:



Possible Storage Lengths for Integer Values

Windows and UNIX

Length (bytes)	Largest Integer Represented Exactly
3	8,192
4	2,097,152
5	536,870,912
6	137,438,953,472
7	35,184,372,088,832
8	9,007,199,254,740,992

Possible Storage Lengths for Integer Values

z/OS

Length (bytes)	Largest Integer Represented Exactly
2	256
3	65,536
4	16,777,216
5	4,294,967,296
6	1,099,511,627,776
7	281,474,946,710,656
8	72,057,594,037,927,936

Assigning the Length of Numeric Variables

The use of a numeric length less than 8 bytes does the following:

- causes the number to be truncated to the specified length when the value is written to the SAS data set

This reduces the number of bytes available for the mantissa, which reduces the precision of the number that can be accurately stored.

- causes the number to be expanded to 8 bytes in the PDV when the data set is read by padding the mantissa with binary zeros

Numbers are always 8 bytes in length in the PDV.

Dangers of Reduced-Length Numeric Variables

It is **not** recommended that you change the length of non-integer numeric variables.

```
data test;
   length x 4;
   X=1/10;
   Y=1/10;
run;

data _null_;
   set test;
   put X=;
   put Y=;
run;
```

What happens if we try to reduce the length of non-integer data?

Partial SAS Log (Windows)

```
7    data test;
8       length x 4;
9       X=1/10;
10      Y=1/10;
11   run;

NOTE: The data set WORK.TEST has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


12
13   data _null_;
14      set test;
15      put X=;
16      put Y=;
17   run;

x=0.0999999642
y=0.1
NOTE: There were 1 observations read from the data set WORK.TEST.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

Look at the log.

Are the values of X and Y equal?

Dangers of Reduced-Length Numeric Variables

It is **not** recommended that you reduce the length of integer numeric variables inappropriately or that you reduce the length of variables that hold large integer numeric values. This example illustrates the effect of inappropriately reducing integer values.

```
data test;
   length X 3;
   X=8193;
run;

data _null_;
   set test;
   put X=;
run;
```

p202d07

Numeric Precision

Partial SAS Log (Windows)

```
120  data test;
121     length X 3;
122     X=8193;
123  run;

NOTE: The data set WORK.TEST has 1 observations and 1
variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

124
125  data _null_;
126     set test;
127     put X=;
128  run;

x=8192
NOTE: There were 1 observations read from the data set
WORK.TEST.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

The results-Let's compare pros and cons

Advantages	Disadvantages
Conserves data storage space	Uses additional CPU to read
Requires less I/O to read	Can alter high-precision values such as non-integer and large integer values

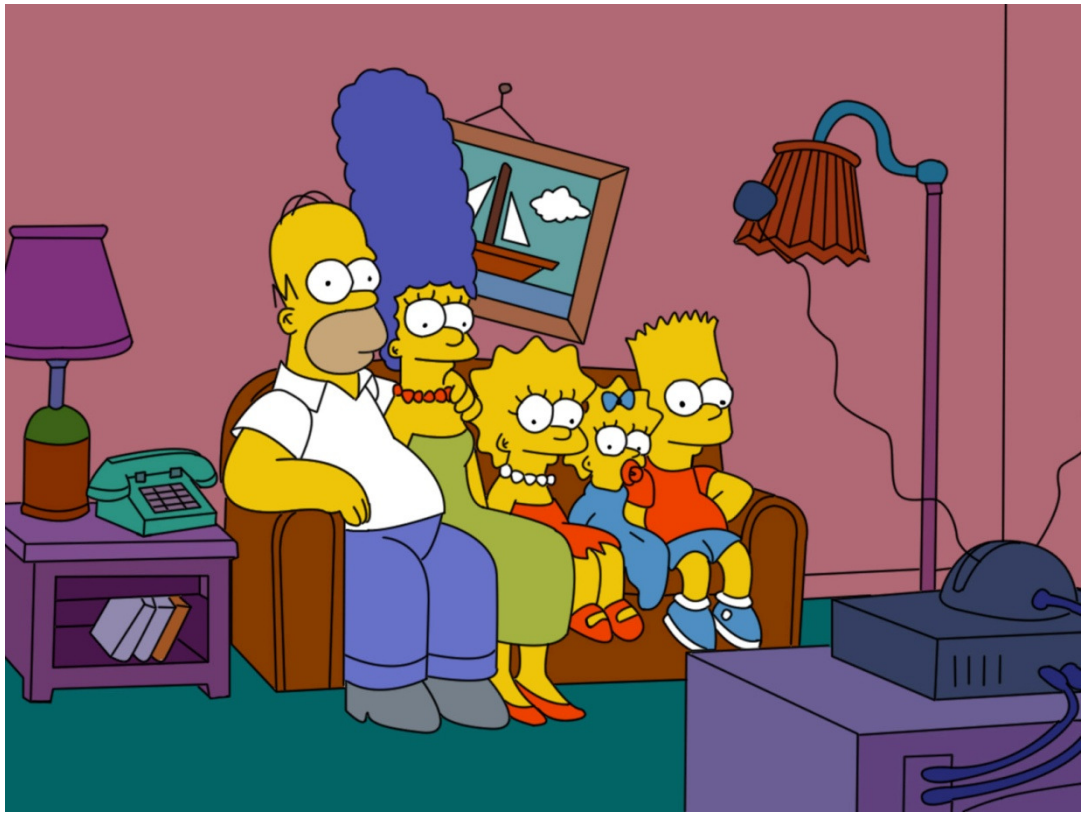
Saving I/O

#4 Reduce multiple & unnecessary passes through data.

#5 Modify variable attributes.

#4 Reduce multiple passes of your data

Watching a favorite TV show. Would you get up from the safety of your couch to the kitchen 4 times to get your chocolate fix
Or would you get all your chocolate at one go?



Technique 1-Multiple data steps

Create 4 subsets from **cesales_analyis** for candy categories

```
data chocolate;
set choc.cesales_analysis;
if category='Chocolate' ;
Run;
data gummy;
set choc.cesales_analysis;
if category='Gummy' ;
Run;
data hard;
set choc.cesales_analysis;
if category='Hard' ;
Run;
data sugarfree;
set choc.cesales_analysis;
if category='Sugar-Free' ;
Run;
```

Technique 2-Multiple select statements

```
proc sql;
create table  chocolate as
select * from choc.cesales_analysis
where category='Chocolate';

create table  gummy as
select * from choc.cesales_analysis
where category='Gummy';

create table  hard as
select * from choc.cesales_analysis
where category='Hard';

create table sugarfree as
select * from choc.cesales_analysis
where category='Sugar-Free';
```

Technique 3-Single DATA Step

```
data chocolate gummy hard sugarfree;
set choc.cesales_analysis;
if category='Chocolate' then output
chocolate;
else if category ='Gummy ' then output gummy;
else if category='Hard' then output hard;
else if category='Sugar-Free' then output
sugarfree;
run;
```

Which is the best?

Technique 4-subset in datastep & then sort data

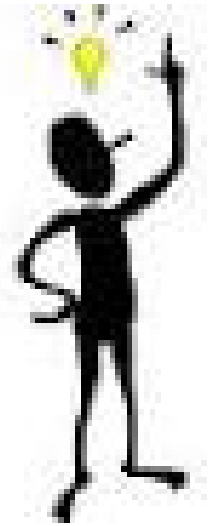
Create a sorted subset of **choc.cesales_analysis** that contains the string Bag.

```
data bag;
set choc.cesales_analysis;
where product_name contains 'Bag';
run;

proc sort data=bag;
by product_name;
run;
```

Technique 5-Sort & filter in one step

```
proc sort data=choc.cesales_analysis out=bag;
by product_name;
where product_name contains 'Bag';
run;
```

Q. Which one is better? A. Technique 5
Handy tip: Many folks don't use the Where clause in PROC preferring to use in the data step.



Did you know ? you can use the WHERE clause in any PROC, it's a powerful way to filter your data as we'll see later

#5 Manage your data with PROC DATASETS

Business task- Rename & format variable attributes in **choc.cesales_analysis** to be consistent with those in other datasets

	Var Name	Var Format
Ceorder_info	Prod_id	$7.
	Total_sales	.
Ceesales_analysis	Product_id	$7.
	Total_sales	Comma9.2.

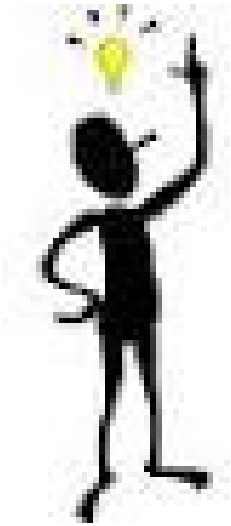
DATA Step / PROC DATASETS

```
data choc.ceorder_info;
set choc.ceorder_info;
rename prod_id=product_id;
format total_cases comma9.2;
run;
```

```
proc datasets library=choc;
modify ceorder_info
rename prod_id=product_id;
format total_cases comma9.2;
run;
```

DATA Step / PROC DATASETS

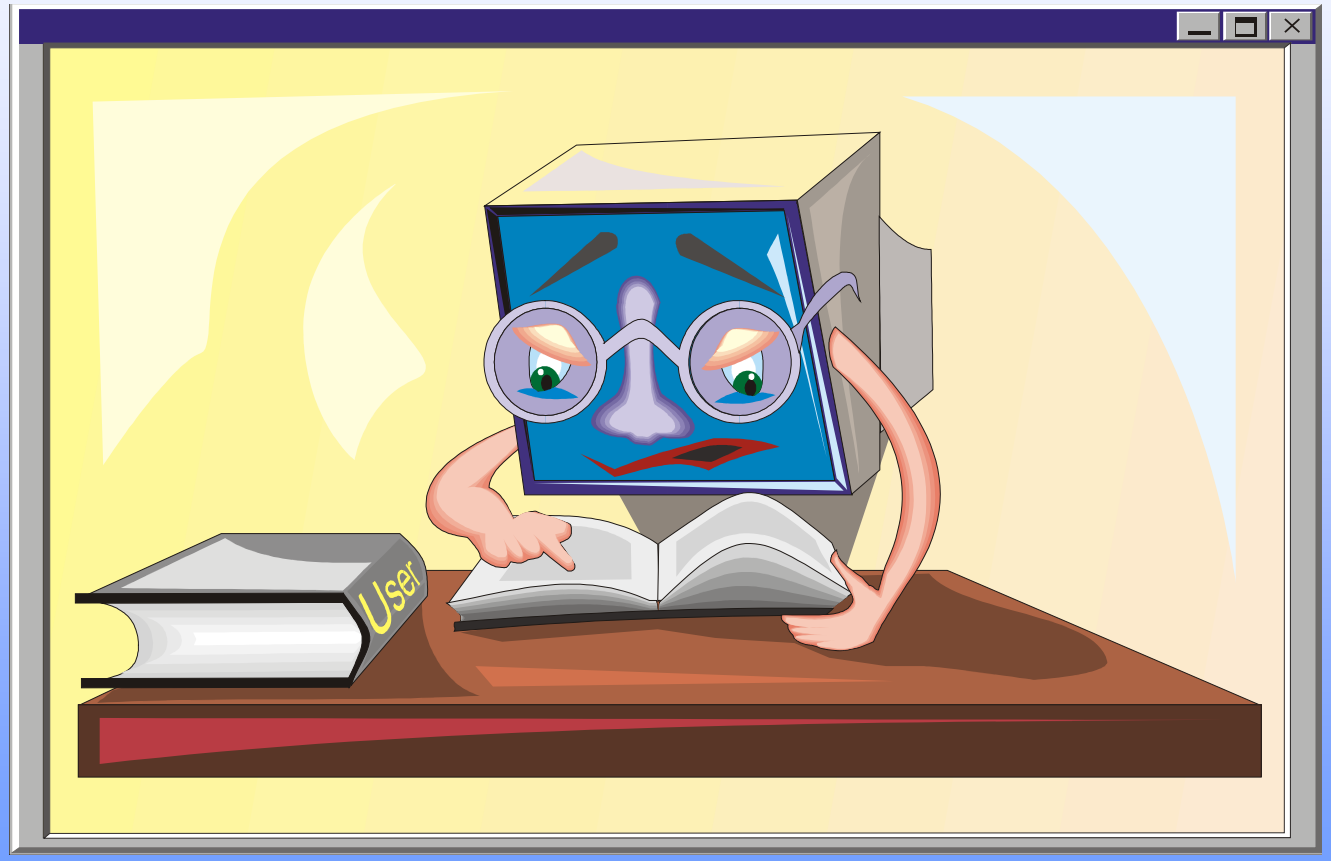
Q. So Which one is better for data management?
The Data Step or PROC Datasets?
A. Important advantage is efficiency, for attributes other than variable index DATASETS only processes the header portion of the SAS data set, while the data step processes the entire dataset.



Did you know ? PROC Datasets needs a QUIT statement otherwise it just sits in memory waiting for you to submit another request.. So remember to end it with a QUIT statement

Techniques affecting CPU and/or IO

If you process fewer variables and observations,
CPU and/or I/O operations can be affected significantly.



6 Process only necessary variables & observations

...

#6 Process only necessary observations

Simple techniques can conserve I/O. The amount of I/O saved depends on the size of the subset being processed.

6.1 Reduce the number of observations - WHERE in the In the Data step or WHERE in the PROC step

6.2. WHERE statement or IF statement

6.1 Reduce observations
Technique 1 - Subsetting in the Procedure

One way to create a subset is to use the WHERE statement in a procedure.

```
data choc.yearend;
set chocxls.'dec04sales$'n;
extracase=total_cases*2;
run;
proc means data=choc.yearend mean sum;
where category='Chocolate';
class customer_type;
var extracase;
run;
```

- The data set **yearend** contains 12 variables and 4640 observations.

Technique 2-Subsetting in the Data Step

Another way to subset would be in the DATA step.

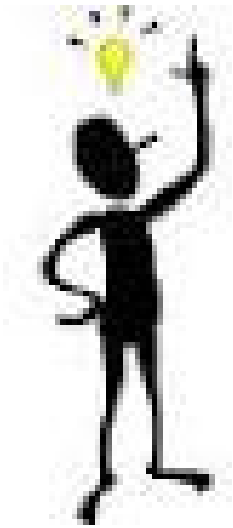
```
data choc.yearend;
set chocxls.'dec04sales$'n;
extracase=total_cases*2;
where category='Chocolate';
run;
proc means data=choc.yearend mean sum;
class customer_type;
var extracase;
run;
```

I/O savings result from reducing the number of observations in the input and output data sets.

- The data set **yearend** contains 12 variables and 2445 observations..

Consider- Which one is more efficient?

The Data Step & then subsetting in PROC MEANS
or
subsetting directly in the Datastep



Did you know ? The data step is a builder – that's why you had to use the data step here because you were creating a new variable. Otherwise PROC MEANS alone would have been enough!

6.2 Reduce Observations

Where or IF – that is the question?

Selecting Observations

We want to subset for **category**= “Chocolate“

Category	SubCategory	Product_Name
Chocolate	Chocolate Dark	SM Dark Choc Bar
Gummy	Gummy Sour	Gummy Lions Bag
Hard	Hard Sweet	Butterscotch Disks Bag
Sugar-Free	SF Chocolate	SF Jelly Beans Bag

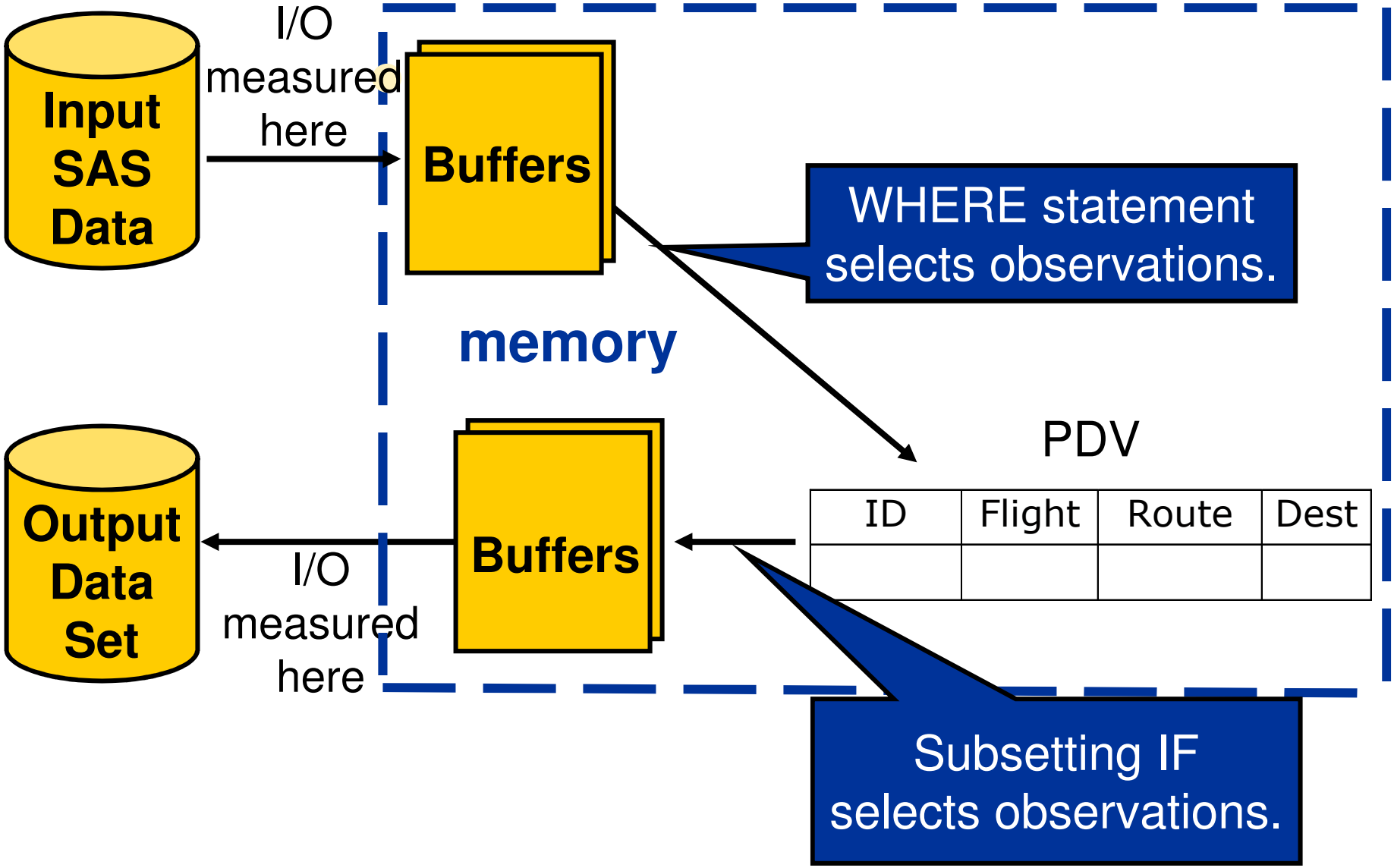
Subsetting IF or the Where clause?

Create a subset of the cesales_analysis dataset that contains data for Chocolate.

```
3  data chocolate;
4  set choc.cesales_analysis;
5  if category='Chocolate' ;
6  Run;

NOTE: There were 115928
observations read from the
data set
CHOC.CESALES_ANALYSIS.
NOTE: The data set
WORK.CHOCOLATE has 50368
observations and 11
variables.
NOTE: DATA statement used
(Total process time):
real time          2.84 seconds
cpu time           0.12 seconds
```

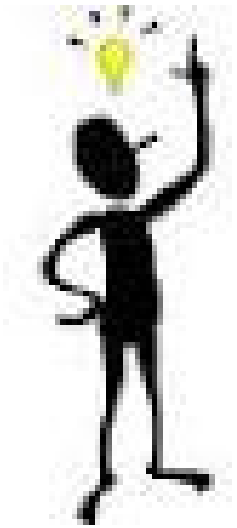
```
7  data chocolate;
8  set choc.cesales_analysis;
9  where category='Chocolate' ;
10 Run;

NOTE: There were 50368
observations read from the data
set CHOC.CESALES_ANALYSIS.
WHERE category='Chocolate';
NOTE: The data set
WORK.CHOCOLATE has 50368
observations and 11 variables.
NOTE: DATA statement used
(Total process time):
real time          2.26 seconds
cpu time           0.06 seconds
```

The Subsetting IF and the WHERE Statements



Consider- When to use which one?

The WHERE clause
Or
The Subsetting IF
The answer lies in this question - do you want to subset existing obs or newly created obs?

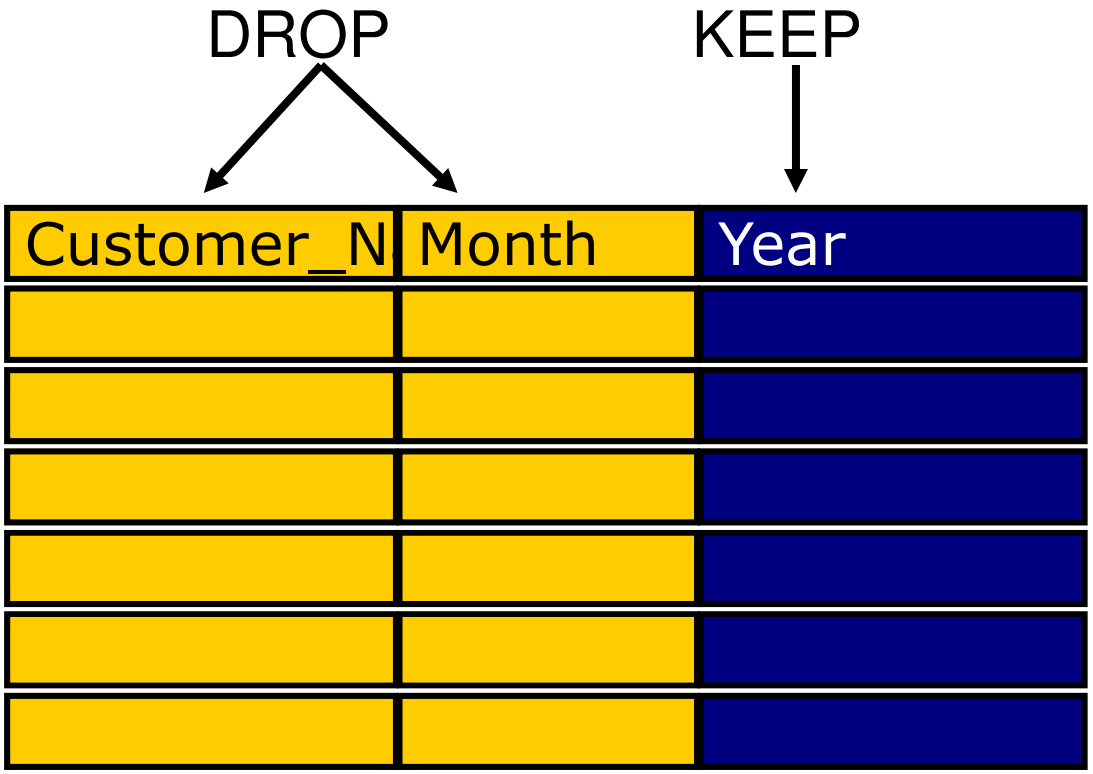


Did you know ? The WHERE clause is the same one used in SQL. If you want to subset existing obs use the WHERE. The powerful WHERE acts on obs before moving it to the PDV. The IF statement works on newly created var but has to read in row by row into the PDV thus slower in comparison

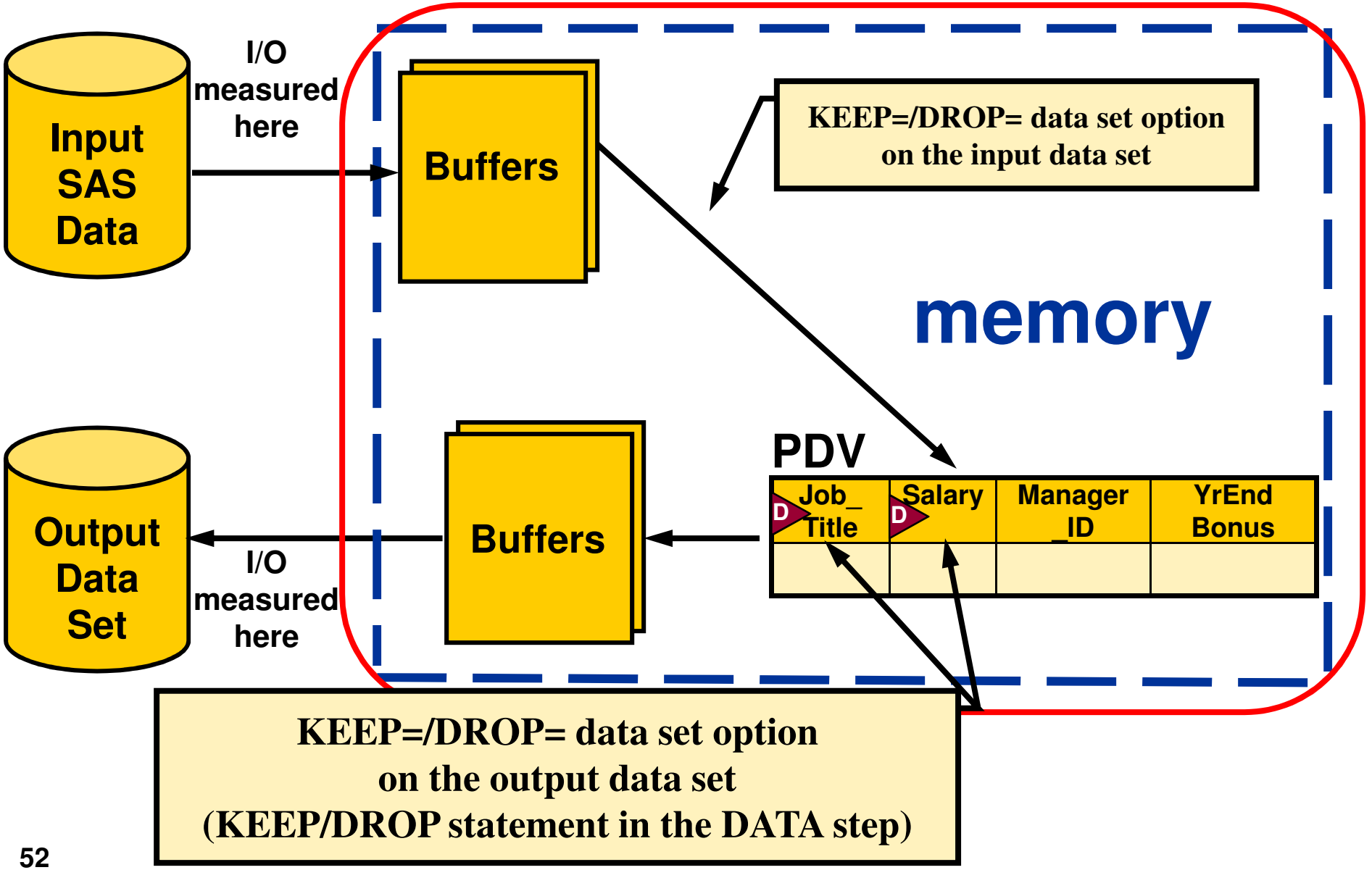
#7 Process only the necessary variables

To subset variables, you can use the following:

- DROP and KEEP statements
- DROP= and KEEP= data set options



Using the KEEP=/DROP= Options




Poll
Quiz

Setup for the Poll

Technique 1

```
data choc.yearend;
set chocxls.'dec04sales$'n;
extracase=total_cases*2;
run;
proc means data=choc.yearend mean sum;
where category='Chocolate';
class customer_type;
var extracase;
run;
```

I/O savings results from reducing the number of variables and observations in the input and output data sets.

Technique 2

```
data choc.yearend(keep=extracase category);
set chocxls.'dec04sales$'n;
extracase=total_cases*2;
where category='Chocolate';
run;
proc means data=choc.yearend mean sum;
class customer_type;
var extracase;
run;
```

Multiple Choice Poll

In addition to the I/O decrease when the DATA step creates **`yearend`**, where does Technique 2 have additional decrease of I/O?

a. Fewer variables are read into the program data vector from **`choc.'dec04sales'n`** in Technique 2 because of the KEEP= data set option.
b. The PROC MEANS in Program 2 loads a smaller version of **`yearend`**.
c. There is no additional decrease in I/O; all of the decrease in I/O occurs when the data set **`yearend`** is created by the DATA step.

Answer

In addition to the I/O decrease when the DATA step creates **yearend**, where does Technique 2 have additional decrease of I/O?

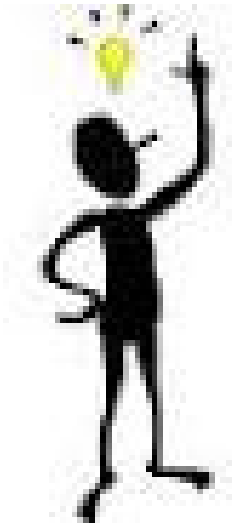
a. Fewer variables are read into the program data vector from **choc.'dec04sales'n** in Technique 2 because of the KEEP= data set option.

b. The PROC MEANS in Program 2 loads a smaller version of **yearend**.

c. There is no additional decrease in I/O; all of the decrease in I/O occurs when the data set **yearend** is created by the DATA step.

Consider- Which one is more efficient?

DROP & KEEP statements or
DROP & KEEP options?



Did you know ? DROP & KEEP options & statements are both a great way to reduce # of variables. Which one is better? Options might be better, as the moment of action is pure & clear. You don't have to go scanning code to see what variable went to building the dataset.

#7 Subsetting variables – contd.

Told you the efficiency question is always answered with the 1 question
IT DEPENDS
so let's take a look at 5 more scenarios

Technique 1- Reading and Writing All Variables

Create a report that contains the average and median of the total number of extracases of chocolates provided to customers in december in **choc.'dec04sales$'n** that has 11 variables.

```
data choc.yearend;
set chocxls.'dec04sales$'n;
extracase=total_cases*2;
where category='Chocolate';
run;

title 'December sales data';
proc means data=choc.yearend mean median;
class customer_type;
var extracase;
run;
```

Technique 2- Reading All Variables/Writing Two Variables

```
data choc.yearend(keep=customer_type extracase);
set chocxls.'dec04sales$'n;
extracase=total_cases*2;
where category='Chocolate';
run;

title 'December sales data';
proc means data=choc.yearend mean median;
class customer_type;
var extracase;
run;
```

Technique 3- Reading Three Variables

```
data choc.yearend;
set chocxls.'dec04sales$'n(keep=customer_type
category total_cases);
extracase=total_cases*2;
where category='Chocolate';
run;

title 'December sales data';
proc means data=choc.yearend mean median;
class customer_type;
var extracase;
run;
```

Technique 4- Reading Three Variables/Writing Two Variables

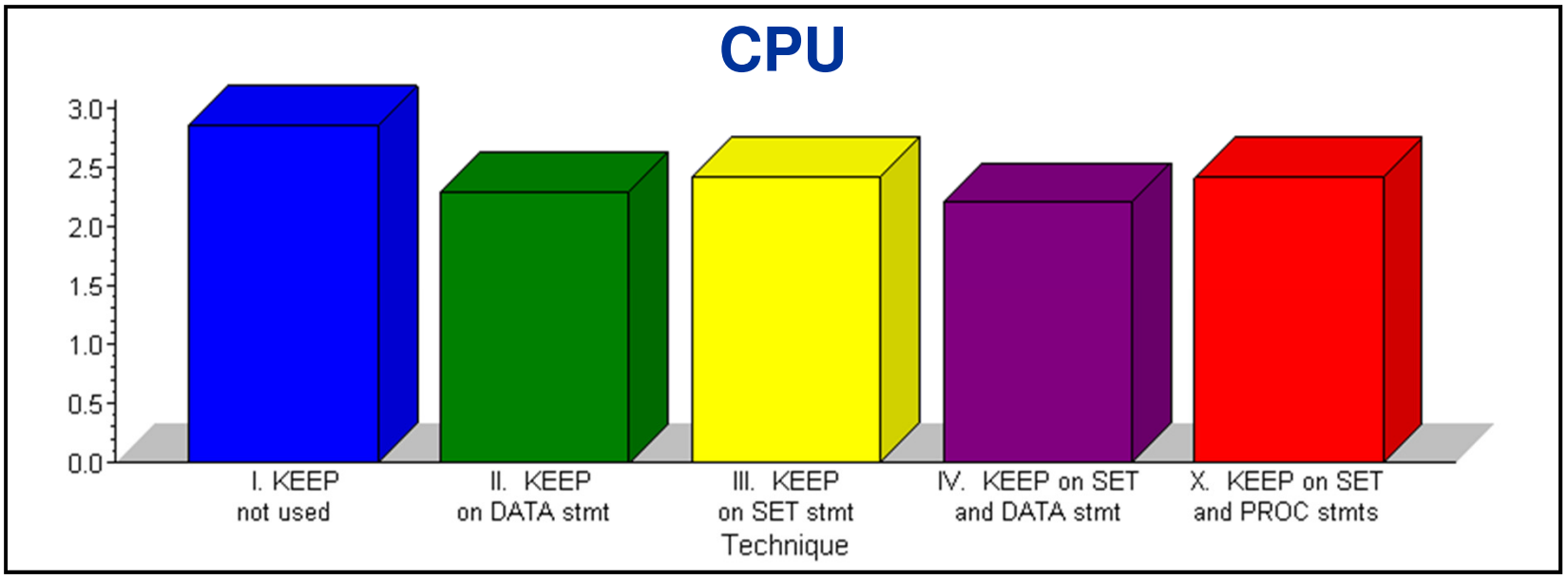
```
data choc.yearend(keep=extracase customer_type);
set chocxls.'dec04sales$'n(keep=category
total_cases customer_type);
extracase=total_cases*2;
where category='Chocolate';
run;

title 'December sales data';
proc means data=choc.yearend mean median;
class customer_type;
var extracase;
run;
```

Technique 5 -Reading Three Variables/Reading Two Variables

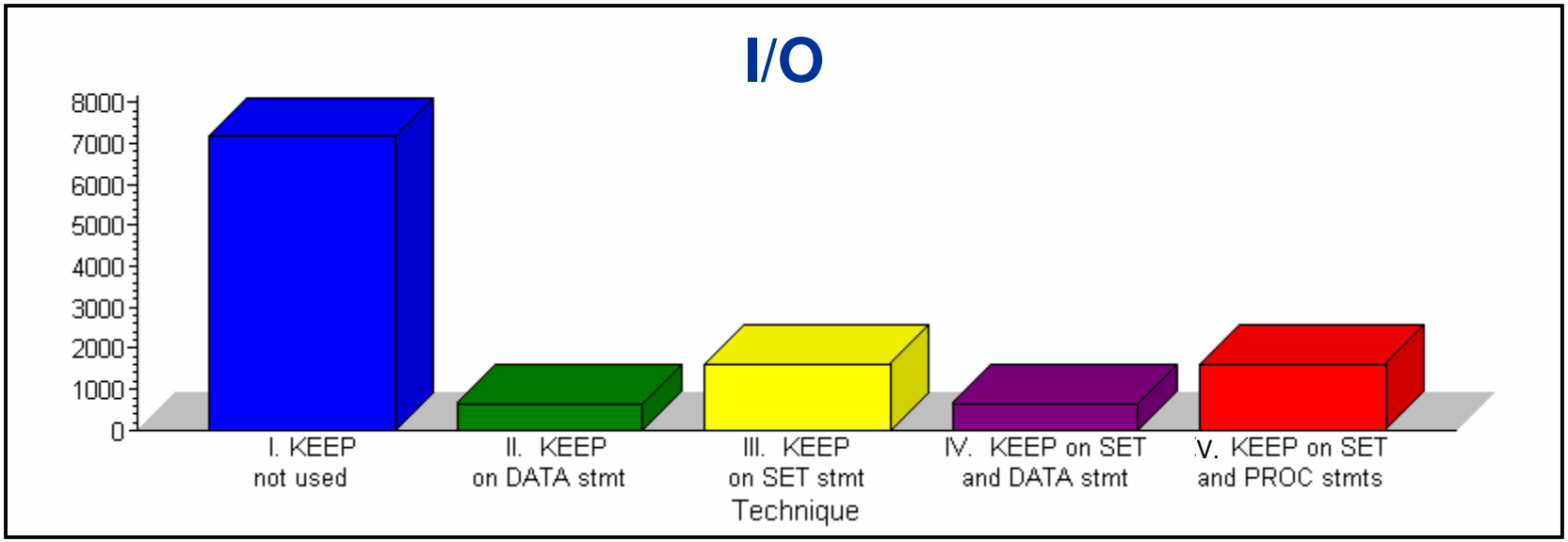
```
data choc.yearend;
set chocxls.'dec04sales$'n(keep=category
total_cases customer_type);
extracase=total_cases*2;
where category='Chocolate';
run;

title 'December sales data';
proc means data=choc.yearend(keep=extracase
customer_type) mean median;
class customer_type;
var extracase;
run;
```

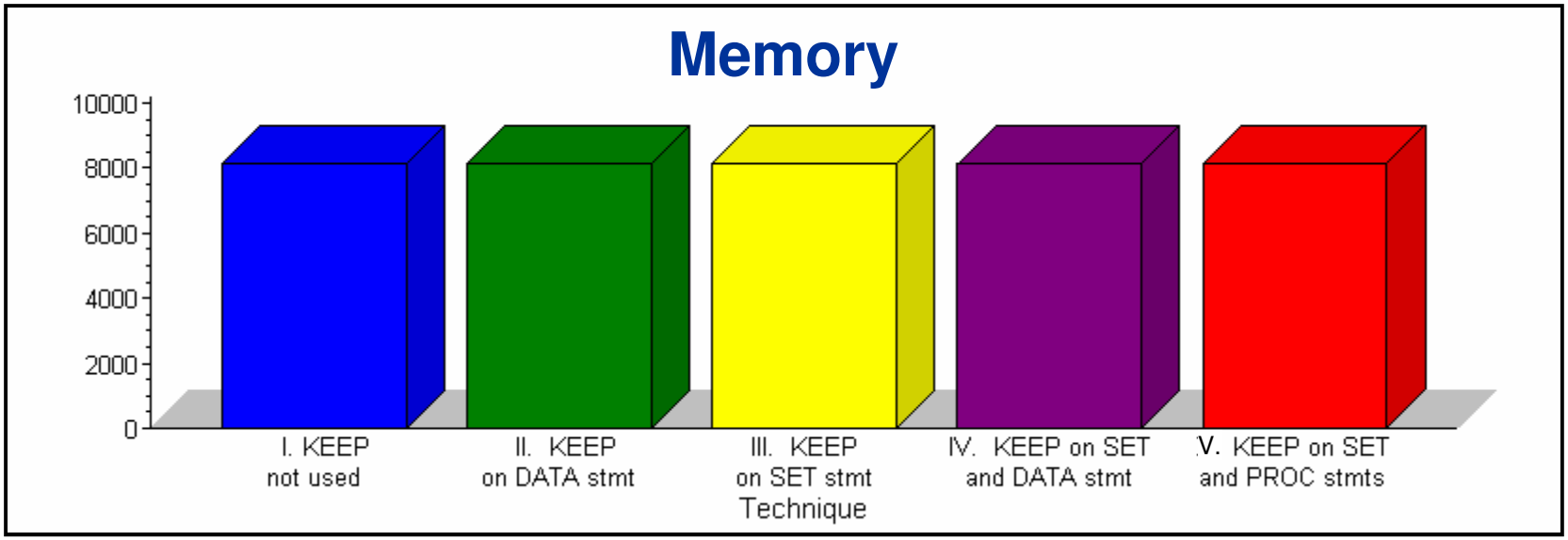
Comparing Techniques

Technique	CPU	I/O	Memory
I. KEEP not used	2.9	7177	8140
II. KEEP on DATA statement	2.3	656	8138
III. KEEP on SET statement	2.4	1625	8138
IV. KEEP on SET and DATA statements	2.2	662	8138
V. KEEP on SET and PROC statements	2.4	1625	8139

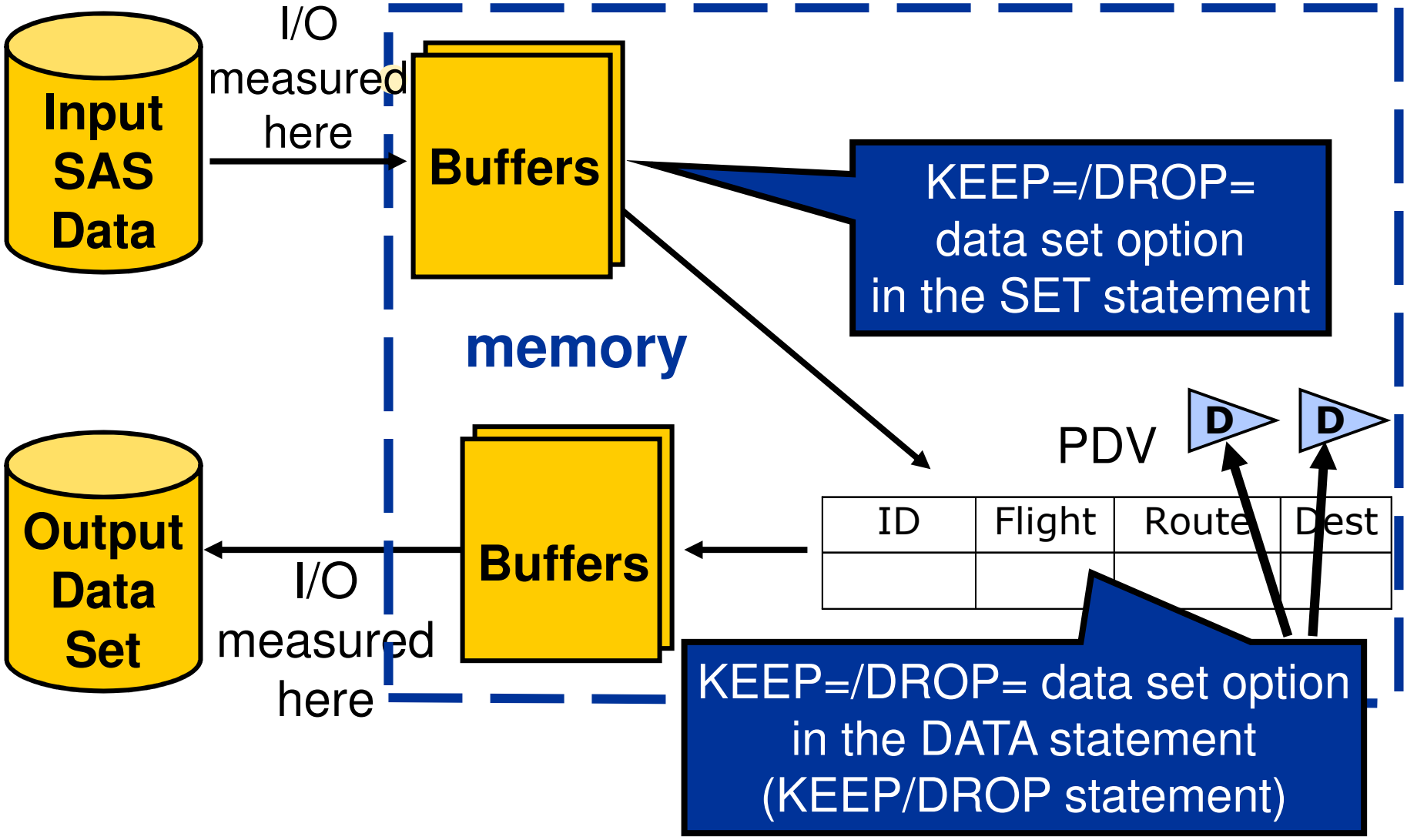


Comparing Techniques



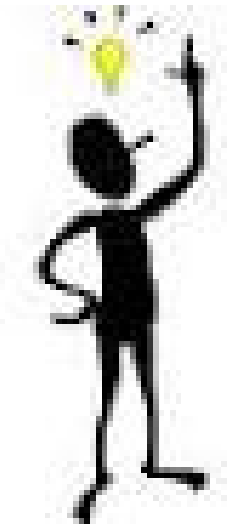


Using the KEEP=/DROP= Options



Consider- Which one is more efficient?

DROP & KEEP options on the DATA statement
Or DROP & KEEP options on the SET statement?

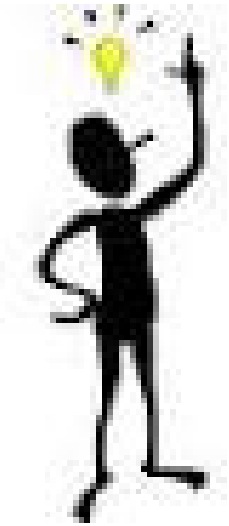


Did you know ? DROP & KEEP options on your input dataset are a great way to reduce # of variables read into the PDV substantially saving your CPU & I/O.

If your new variable's construction depends on an old variable, don't use the DROP/KEEP on the SET statement- otherwise you won't have access to it at all

Recap

- Execute only necessary statements.
- Eliminate unnecessary passes of the data.
- Read and write only the data that you require.



Did you know ? OBS= and FIRSTOBS= options are a great way to test your data & code before bringing into production

Best practice - Saving Space

#8 Store data as character to manage space

What type should my data be—Character or numeric?

Space Recap

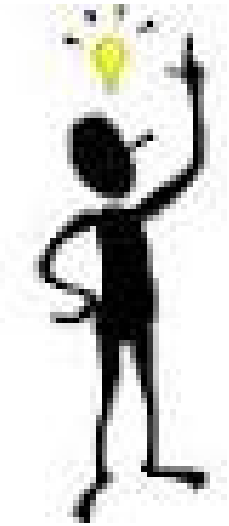
Space is no longer a big issue with most organizations having managed this resource very well.

However if space is an issue at your shop, ask yourself the question:

Am I ever going to do calculations with this data?

Customer_id, order_id, lend themselves very well to character data

If you store them as numeric, then even if your data only occupies e.g. 4 bytes, space taken up would be standard 8 bytes



Did you know ? SAS stores dates as a number so you can do plenty of calculations with dates– you can convert numeric to character data using the PUT function

Best practice - Saving memory

I always have trouble remembering three things: faces, names, and -- I can't remember what the third thing is.

Fred A. Allen

#9 Use the BY statement instead of the CLASS statement

Use BY-group processing instead of CLASS statements TO GROUP DATA in those procedures that support both, especially where you have pre-sorted data or can use an existing index.

Using the BY Statement

What are the differences between using a BY statement and using a CLASS statement in a procedure?

BY Statement	CLASS Statement
The data set must be sorted or indexed on the BY variables.	The data set does not need to be sorted or indexed on the CLASS variables.
BY-group processing holds only one BY group in memory at a time.	The CLASS statement accumulates aggregates for all CLASS groups simultaneously in memory.
A percentage for the entire report cannot be calculated with procedures such as the REPORT or TABULATE procedures.	A percentage for the entire report can be calculated with procedures such as the REPORT or TABULATE procedures.

PROC MEANS with a BY Statement

```
proc means data=orion.order_fact mean median
           maxdec=2;
   format Order_Date year4.;
   by Order_Date;
   var Quantity -- CostPrice_Per_Unit;
run;
```

PROC MEANS with a BY Statement

Partial PROC MEANS Output with a BY Statement

----------Date Order was placed by Customer=2003----------

The MEANS Procedure

Variable	Label	Mean	Median
Quantity	Quantity Ordered	1.82	2.00
Total_Retail_Price	Total Retail Price for This Product	177.59	105.95
CostPrice_Per_Unit	Cost Price Per Unit	42.72	31.80

---------- Date Order was placed by Customer=2004 ----------

Variable	Label	Mean	Median
Quantity	Quantity Ordered	1.69	1.00
Total_Retail_Price	Total Retail Price for This Product	146.13	85.65
CostPrice_Per_Unit	Cost Price Per Unit	37.10	25.68

PROC MEANS with a CLASS Statement

```
proc means data=orion.order_fact mean median
           maxdec=2;
   format Order_Date year4.;
   class Order_Date;
   var Quantity -- CostPrice_Per_Unit;
run;
```

PROC MEANS with a CLASS Statement

Partial PROC MEANS Output with a CLASS Statement

The MEANS Procedure

Date Order was placed by Customer	N Obs	Variable	Label	Mean	Median
2003	128	Quantity	Quantity Ordered	1.82	2.00
		Total_Retail_Price	Total Retail Price for This Product	177.59	105.95
		CostPrice_Per_Unit	Cost Price Per Unit	42.72	31.80
2004	108	Quantity	Quantity Ordered	1.69	1.00
		Total_Retail_Price	Total Retail Price for This Product	146.13	85.65
		CostPrice_Per_Unit	Cost Price Per Unit	37.10	25.68
2005	90	Quantity	Quantity Ordered	1.70	1.00
		Total_Retail_Price	Total Retail Price for This Product	187.20	77.00
		CostPrice_Per_Unit	Cost Price Per Unit	49.45	25.20
2006	143	Quantity	Quantity Ordered	1.57	1.00
		Total_Retail_Price	Total Retail Price for This Product	149.70	92.80
		CostPrice_Per_Unit	Cost Price Per Unit	44.25	29.95
2007	148	Quantity	Quantity Ordered	1.93	2.00
		Total_Retail_Price	Total Retail Price for This Product	157.49	80.95
		CostPrice_Per_Unit	Cost Price Per Unit	37.53	21.35

#10. Programmer's time saving

10.1 Tips & tricks to manage the SAS display manager

- Getting intimate with the SAS display manager
- The log
- Shortcuts-keys, comments
- Using macros to understand your recent log

10.2 Variable shortcuts

Last Word

What is the data worker's rule #1?

What are 3 questions to ask before jumping to data work

Top 10 SAS coding efficiences:

#1. Boiling down or reducing your data

#2. Do conditional processing

#3. Do not reduce the length of numeric variables

#4 Reduce multiple passes of your data

#5 Manage your data with PROC Datasets

#6 Process only necessary observations

#7 Process only necessary variables

#8 Store data as character type to save space

#9 Use the BY statement instead of CLASS to save space

#10 Finally its all about YOU & your time

Thanks for your time

Questions

Contact

Charu Shankar

Technical Training Specialist

SAS Institute, Toronto

Charu.shankar@sas.com

Continue the conversation.

Connect with me on LinkedIn.

http://ca.linkedin.com/pub/charu-shankar/0/b42/892