Merck Serono
Living science, transforming lives

MERCK

Considerations for improving program performance

Martin Gregory

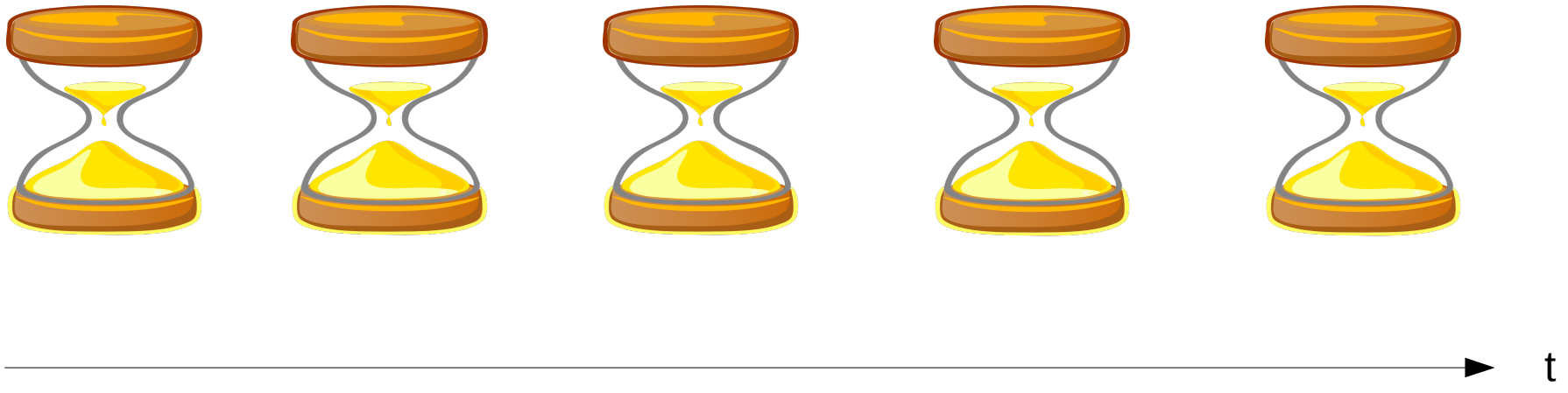
PhUSE SDE

Frankfurt, 13 June 2012

Topics

- Motivation
- Diagnosis of performance problems
- Treatment alternatives
- Generally applicable treatment
- Know your programming language
- SAS-related treatment
- Conclusions

Why improve performance?



Diagnosis: possible causes

- insufficient CPU resources
- slow input/output:
  - to memory
  - to local disk
  - to network disk
- insufficient storage
  - memory
  - disk

Diagnosis: how to

- Observe the running program using appropriate tools
  - UNIX
    - ps command,
    - time command
    - sosview, ksysinfo or similar tools
  - Microsoft Windows
    - task manager
  - SAS
    - FULLSTIMER option
    - writing timing information to the log
- Examine the program carefully
  - does it do complicated calculations?
  - does it read or write a large amount of data?
  - does it break any of the rules discussed in this paper?

Diagnosis: caveats

- I/O is often hidden in “user cpu” time
- Some operating systems provide much more information
  - SAS FULLSTIMER option on both UNIX and Microsoft Windows shows:
    - real time
    - user cpu time
    - system cpu time
    - memory use
  - On UNIX additional information is provided
    - page Faults
    - page Reclaims
    - page Swaps
    - voluntary Context Switches
    - involuntary Context Switches
    - block Input Operations
    - block Output Operations

Diagnosis: caveats

- Single measurements are only indicative:
  - other activity on the system might have an effect
  - especially for I/O, caching done at the system level can mask differences or even lead to false conclusions
- so:
  - Use repeated measurements
  - Measure at different times of the day and on different days
  - Use statistical analysis to check differences:
    - distributions are not normal, even after log transformations
  - No two systems are alike: your measurements are valid for the system on which you make them

Available treatment

- Add resources
  - faster cpu
  - faster i/o
  - more storage
- Tune the program
  - developer time

Both have a cost

Prevention is better than cure

- Make efficient programming techniques part of your toolbox

- Generally very low cost, but...

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%".

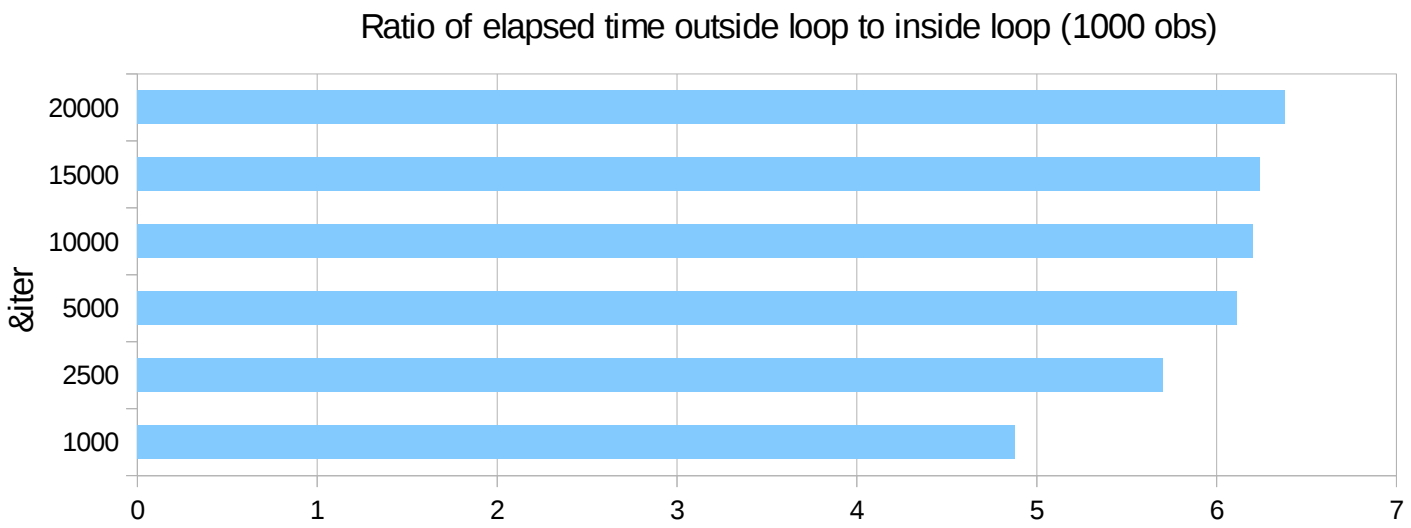
Donald Knuth in Computing Surveys, Vol 6, No 4, December 1974

Treatment: General Rules: CPU

- Avoid unnecessary computations

```
do i=1 to &iter ;
    x=mean(1,2,3) ;
    end ;
```

```
x=mean(1,2,3) ;
do i=1 to &iter ;
   end ;
```



Treatment: General Rules: CPU

- Use less expensive operations when possible

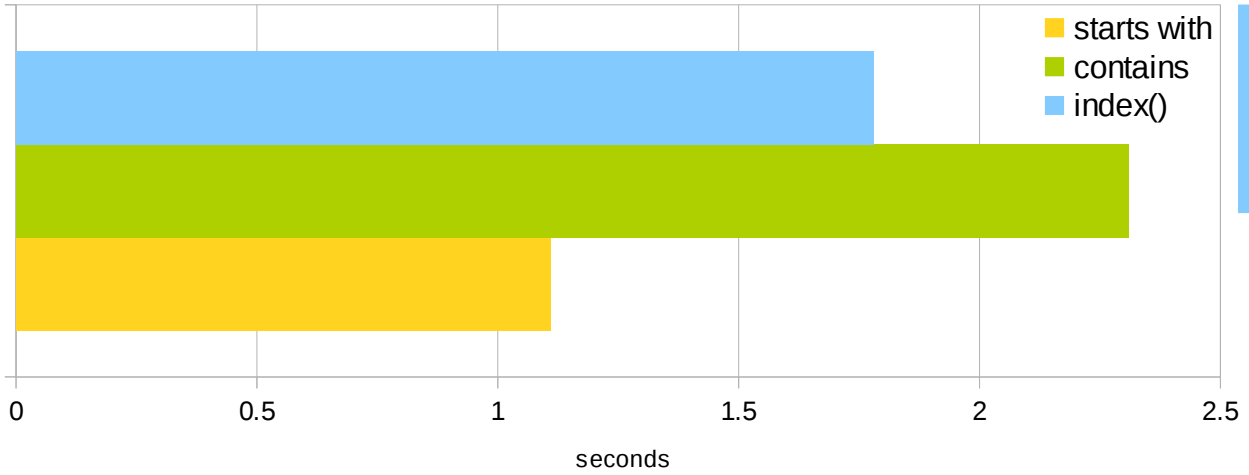
IN_0n has 190,000 obs, cvar1, cvar2 $200

cvar2 has 43,000 obs starting with 'strategically'

```
data out_01 ;
   set in_01 ;
   where cvar2=:'strategically' ;
run ;
```

```
data out_02 ;
   set in_02 ;
   where cvar2 contains 'strategically';
run ;
```

```
data out_03 ;
   set in_03 ;
   where index(cvar2,'strategically') ;
run ;
```



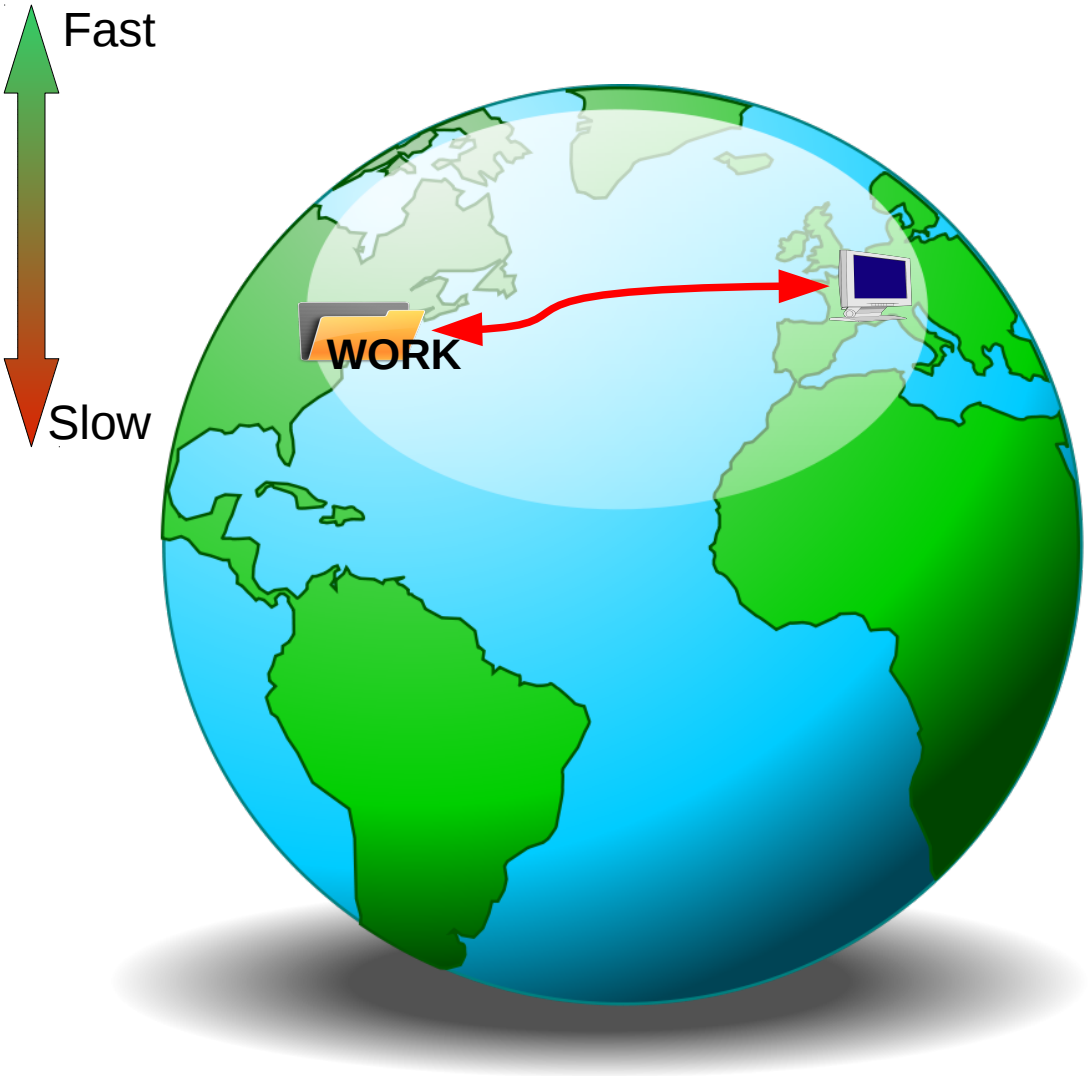
Treatment: General Rules: Input/Output

- Avoid unnecessary I/O –
  - be especially careful with loops
  - sorting is expensive
  - don't read or write data you don't need

Treatment: General Rules: Input/Output

Be aware of the cost of I/O operations:

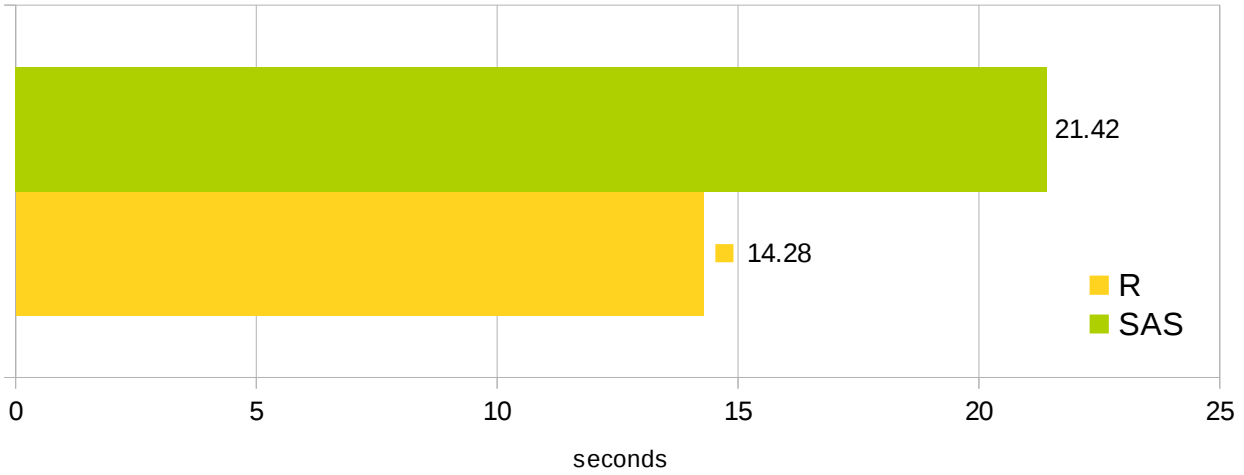
- Memory
- Local disk
- Channel attached disk
- Local network disk
- WAN disk



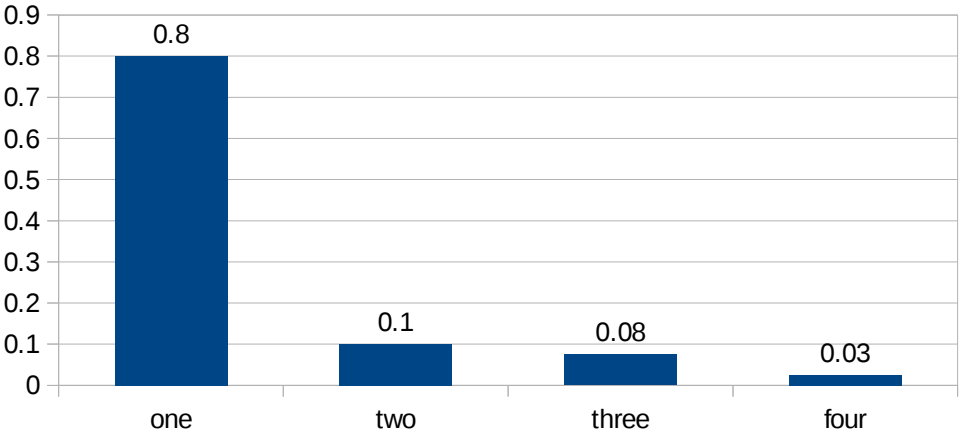
Know your programming language: SAS vs R

```
%macro doit ;
   %do i=1 %to 1000 ;
      proc phreg data=sim ;
         model tanalysis*censanalysis(0)=trt ;
         output out=coxph ;
      run ;
      %end ;
%mend ;
%doit ;
```

```
for ( i in 1:1000) {
  coxph <- coxph(Surv(tanalysis,censanalysis==0)~trt,
                 data=sim)
}
```



Know your programming language: SAS over time



SAS V6: order in select important:

```
if cvar="one" then nvar=1 ;
else if cvar="two" then nvar=2 ;
else if cvar="three" then nvar=3 ;
else if cvar="four" then nvar=4 ;
```

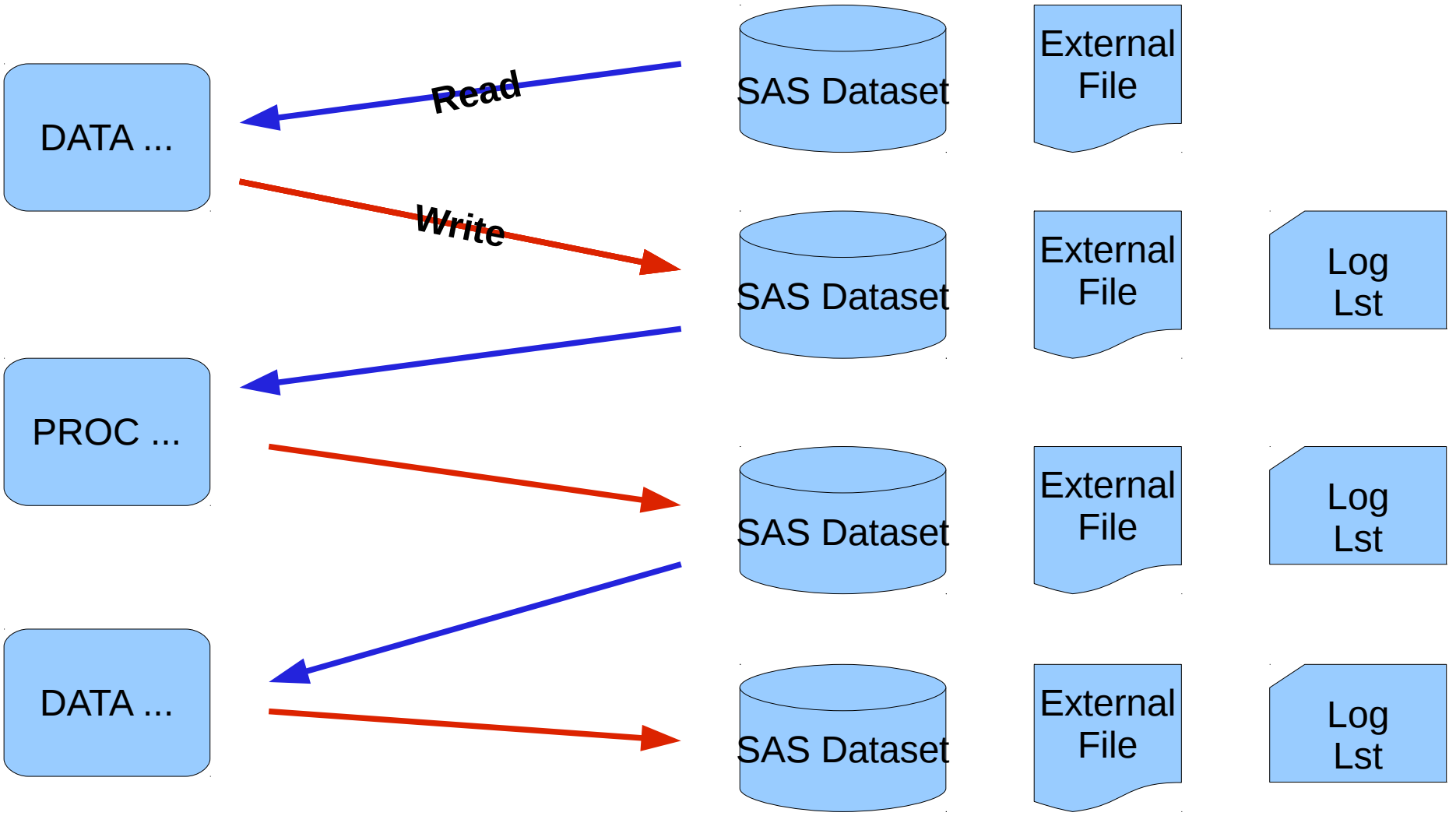
SAS V8: order in select no longer matters

```
if cvar="four" then nvar=4 ;
else if cvar="three" then nvar=3 ;
else if cvar="two" then nvar=2 ;
else if cvar="one" then nvar=1 ;
```

SAS 9.3

Optimizing WHERE conditions with an index is improved with the enhanced SUBSTR (left of=) function.

SAS predisposition: I/O intensive



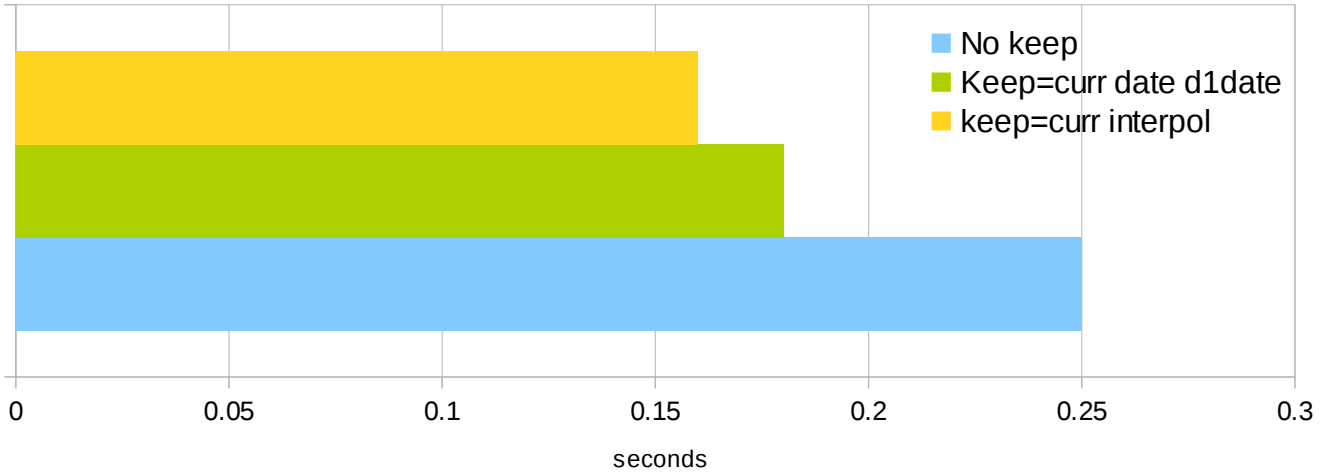
SAS: reducing I/O: KEEP

Variable	Type	Len	275,000 Obs
CURR	Char	3	
DATE	Num	7	
D1RATE	Num	8	
D2RATE	Num	8	
D3RATE	Num	8	
INTERPOL	Num	8	
OBSOLETE	Char	1	

```
data keep_two ;
   set in (keep=curr interpol) ;
run ;
```

```
data keep_three ;
   set in (keep=curr date d1rate) ;
run ;
```

```
data keep_all ;
   set in ;
run ;
```

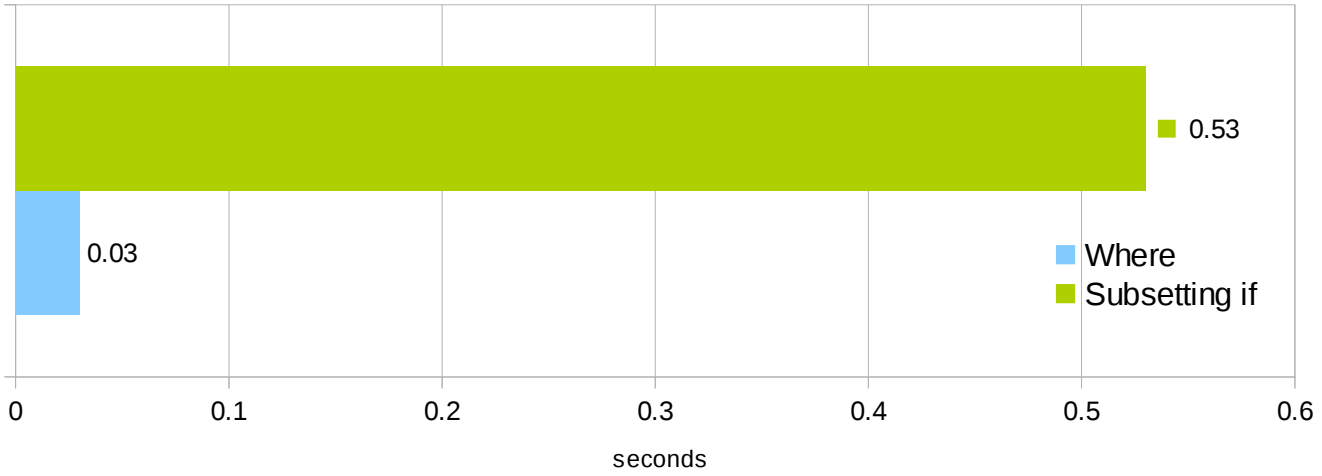


SAS: reducing I/O: WHERE vs IF

Variable	Type	Len	275,000 Obs 2,587 match
CURR	Char	3	
DATE	Num	7	
D1RATE	Num	8	
D2RATE	Num	8	
D3RATE	Num	8	
INTERPOL	Num	8	
OBSOLETE	Char	1	

```
data out_if ;
  set in_if ;
  if curr='EUR';
run ;
```

```
data out_wh ;
  set in_wh ;
  where curr='EUR';
run ;
```

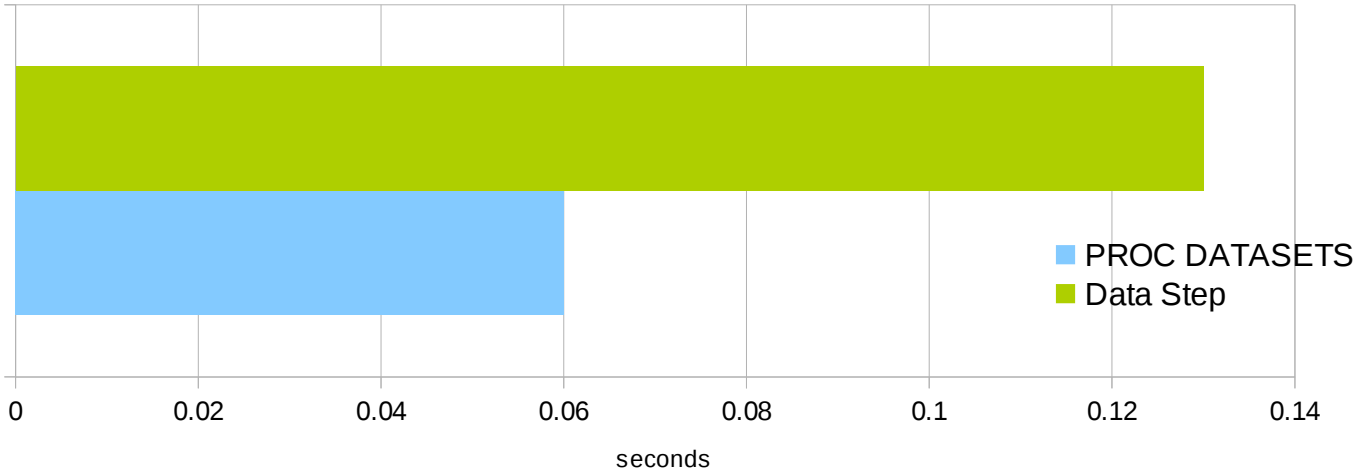


SAS: reducing I/O: DATASETS for attributes

Variable	Type	Len	275,000 Obs
CURR	Char	3	
DATE	Num	7	
D1RATE	Num	8	
D2RATE	Num	8	
D3RATE	Num	8	
INTERPOL	Num	8	
OBSOLETE	Char	1	

```
data lib.in_if ;
  set lib.in_if ;
  format d1rate 8.2;
run ;
```

```
proc datasets lib=x ;
  modify in_wh ;
  format d1rate 8.2;
quit ;
```

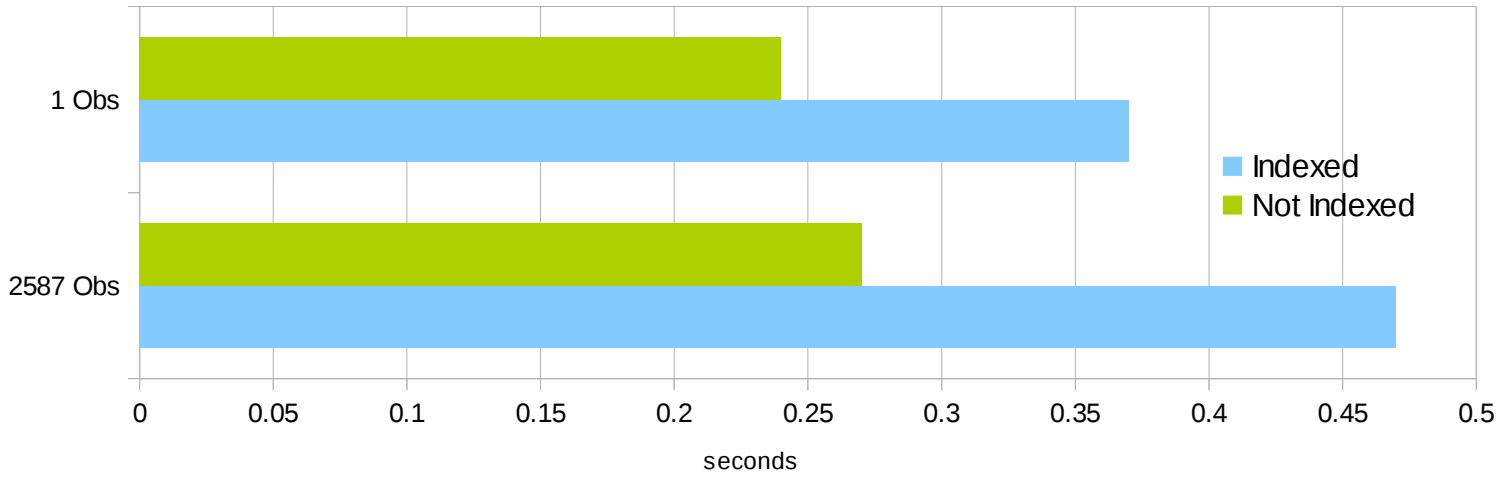


SAS: more efficient I/O: Indexes

Variable	Type	Len	275,000 Obs
CURR	Char	3	Indexes:
DATE	Num	7	curr
D1RATE	Num	8	curr date
D2RATE	Num	8	
D3RATE	Num	8	
INTERPOL	Num	8	
OBSOLETE	Char	1	

```
data _null_ ;
  set lib.not_indexed ;
  where ... ;
run ;
```

```
data _null_ ;
  set lib.indexed ;
  where ... ;
run ;
```

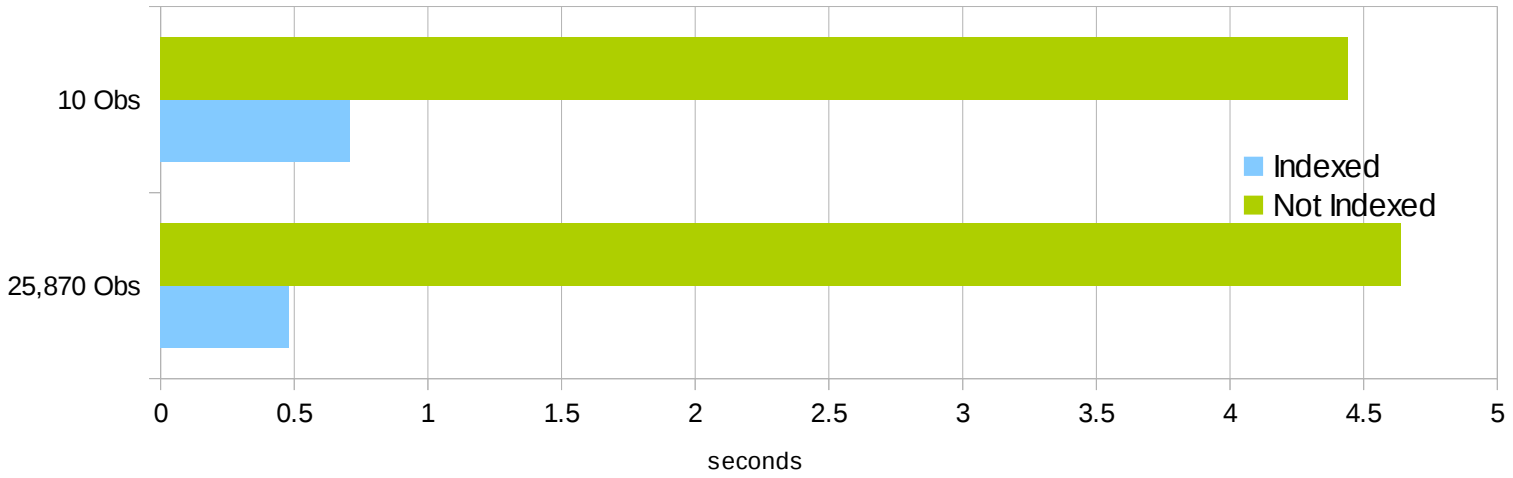


SAS: more efficient I/O: Indexes

Variable	Type	Len	2,750,000 Obs
CURR	Char	3	Indexes:
DATE	Num	7	curr
D1RATE	Num	8	curr date
D2RATE	Num	8	
D3RATE	Num	8	
INTERPOL	Num	8	
OBSOLETE	Char	1	

```
data _null_ ;
  set lib.not_indexed ;
  where ... ;
run ;
```

```
data _null_ ;
  set lib.indexed ;
  where ... ;
run ;
```

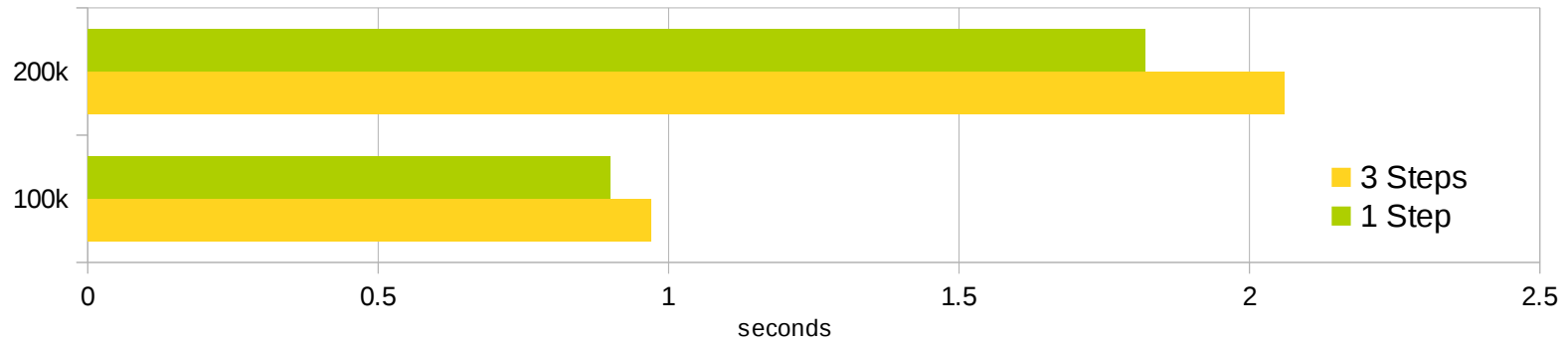


SAS: multiple data steps

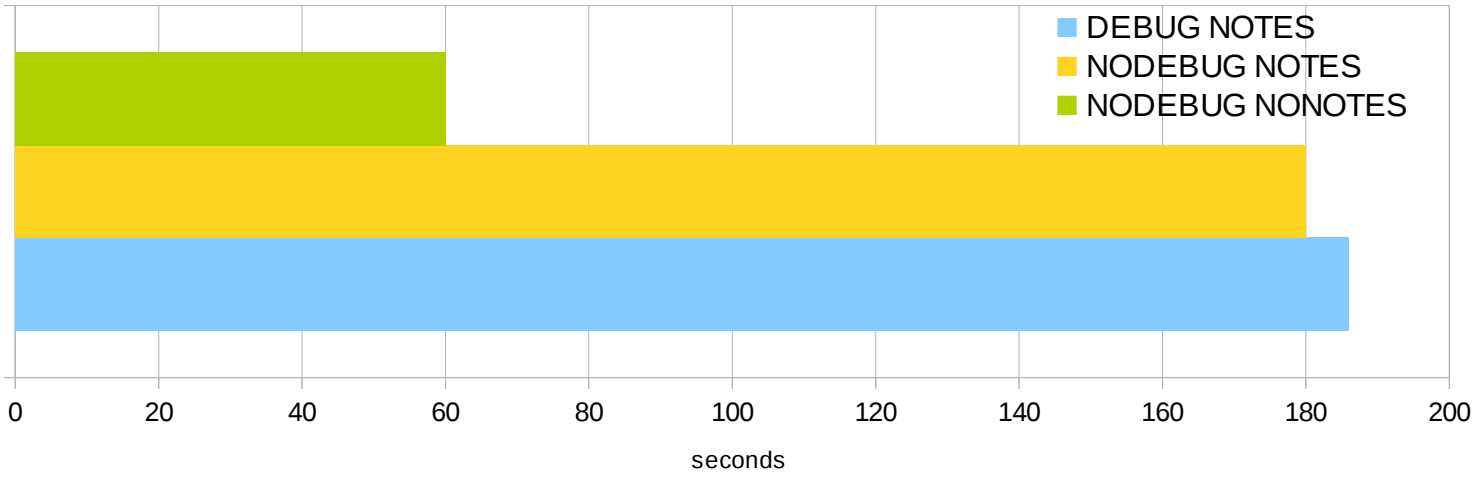
```
data tmp1 ;
   set a.opt_nstep ;
   keep pt ... size ;
   where measdt>. ;
run ;

data tmp2 ;
   set tmp1 ;
   if size in ('NA' 'UN', 'ND')
      then sizen=. ;
   else if size='<5' then sizen=5 ;
   else sizen=input(size,best12.) ;
run ;

data tmp3 ;
   set tmp2 ;
   where sizen>. ;
run ;
```

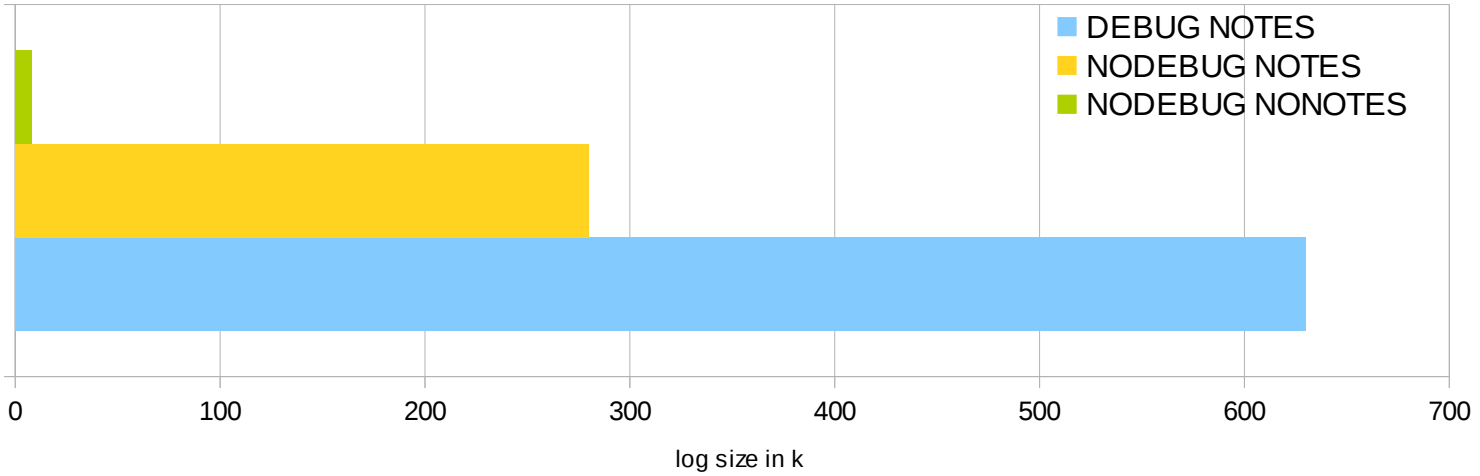
```
data tmp3 ;
   set a.opt_nstep ;
   keep pt ... size ;
   where measdt>. ;
   if size in ('NA' 'UN', 'ND')
      then sizen=. ;
   else if size='<5' then sizen=5 ;
   else sizen=input(size,best12.) ;
   if sizen>. then output ;
run ;
```

Two tests (repeated 30x):
1: 200k obs, 150k selected
2: 100k obs, 75k selected



SAS: verbosity can be expensive





SAS: Interactive use

- Results window and ODS output
  - Closing the results window produced a 10-fold increase in speed in a program using LIFETEST and PHREG
- ODS Graphics
  - If you don't want the graphs, turn ODS Graphics off

Conclusions

- there are some generally applicable rules for all circumstances
- you must know your programming language
- you must know your hardware and operating system
- measurement is critical
- 80/20 rule applies
- prevention is better than cure

Further details in PhUSE 2011 Paper