A Brief Introduction To DS2

RTSUG, 14 November 2024

Presenter
Charu Shankar
SAS Education

Author
Mark Jordan
SAS Education

Charu Shankar, SAS® Institute

With a background in computer systems management. SAS Instructor Charu Shankar engages with logic, visuals, and analogies to spark critical thinking since 2007.

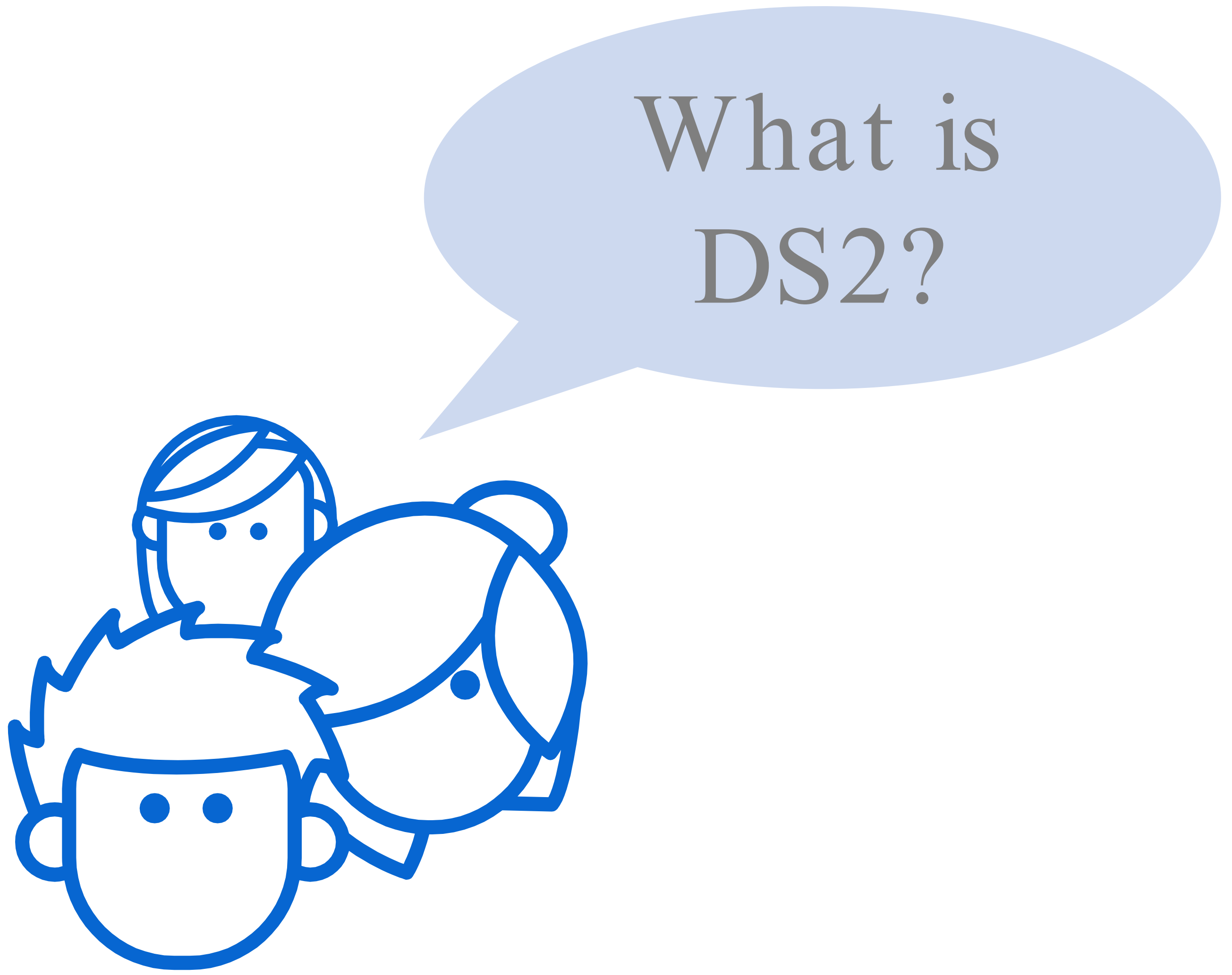
Charu curates and delivers unique content on SAS, SQL, Viya, etc. to support users in the adoption of SAS software.

When not coding, Charu teaches yoga and loves to explore Canadian trails with her husky Miko.

Agenda

- Describe the DS2 programming language
- Identify conditions where it is most appropriate to use DS2
- Compare the DS2 data block to Base SAS DATA step execution
- Understand basic DS2 Syntax
- Parallel Processing in DS2
- Q &A

What is DS2?



Advanced data manipulation language

DATA step-like syntax

Closely integrated with SQL

Convenient reusability

Structural Similarities and Differences – Data Step and DS2

Base SAS DATA Step

```
data _null_;
   Text='Hello, World!';
   put Text=;
run;
```

DS2 Data Block

```
data _null_;
   method init();
      Text='Hello, World!';
      put Text=;
   end;
enddata;
run;
```

OOPs –Object Oriented Programming

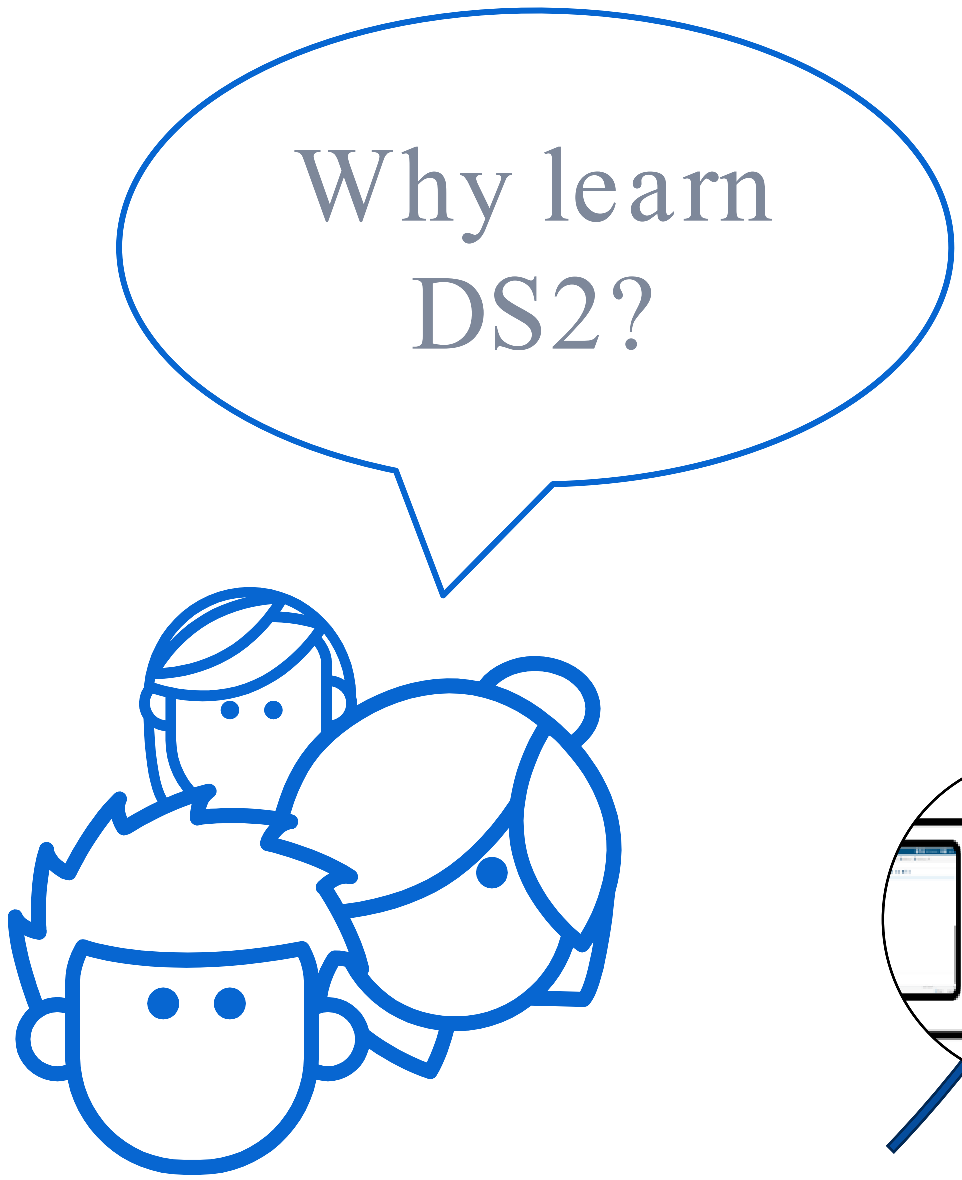
SAS DS2 is considered object-oriented because it introduces concepts such as methods, packages, and inheritance, which are key features of object-oriented programming (OOP).

Unlike the traditional SAS DATA Step, DS2 allows users to create and define reusable objects with encapsulated properties (variables) and behaviors (methods). This structure makes it possible to organize and manage complex code more efficiently, enabling modularity, reusability, and easier maintenance.

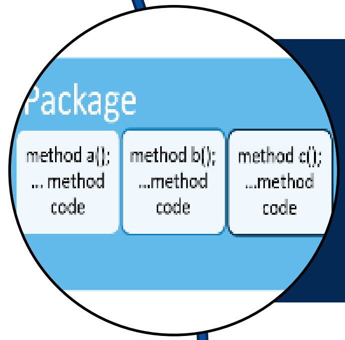
In DS2, you can define custom methods (subroutines or functions) within a DATA program block, and packages (custom-defined modules) allow for reusable code that can be called across different parts of a program.

```
proc ds2;
   data bees / overwrite=yes;
      dcl package bumblebee_class bee; /* Declare an instance of the
bumblebee_class package */
      method init();
         bee = _new_ bumblebee_class("Bombus", "pensylvanicus");
      end;
      method run();
         bee.describe();
      end;
   enddata;
run;
quit;

/* Defining the bumblebee_class package */
proc ds2;
   package bumblebee_class;
      dcl char(20) species;
      dcl char(30) scientific_name;

      /* Constructor method to initialize values */
      method bumblebee_class(char(20) sp, char(30) sci_name);
         species = sp;
         scientific_name = sci_name;
      end;

      /* Method to display bee information */
      method describe();
         put "Species: " species "Scientific Name: " scientific_name;
      end;
   endpackage;
run;
quit;
```

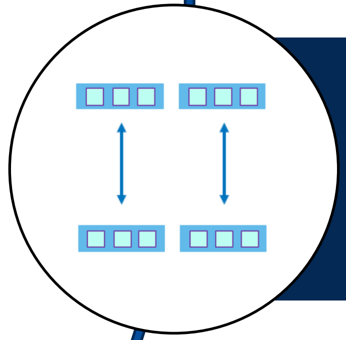
DS2 Superpowers



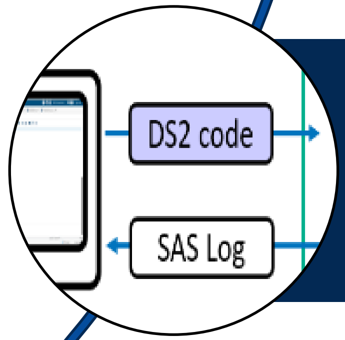
Native processing of ANSI data types


Package
method a(); ... method code
method b(); ...method code
method c(); ...method code

Simple, safe code reuse and sharing

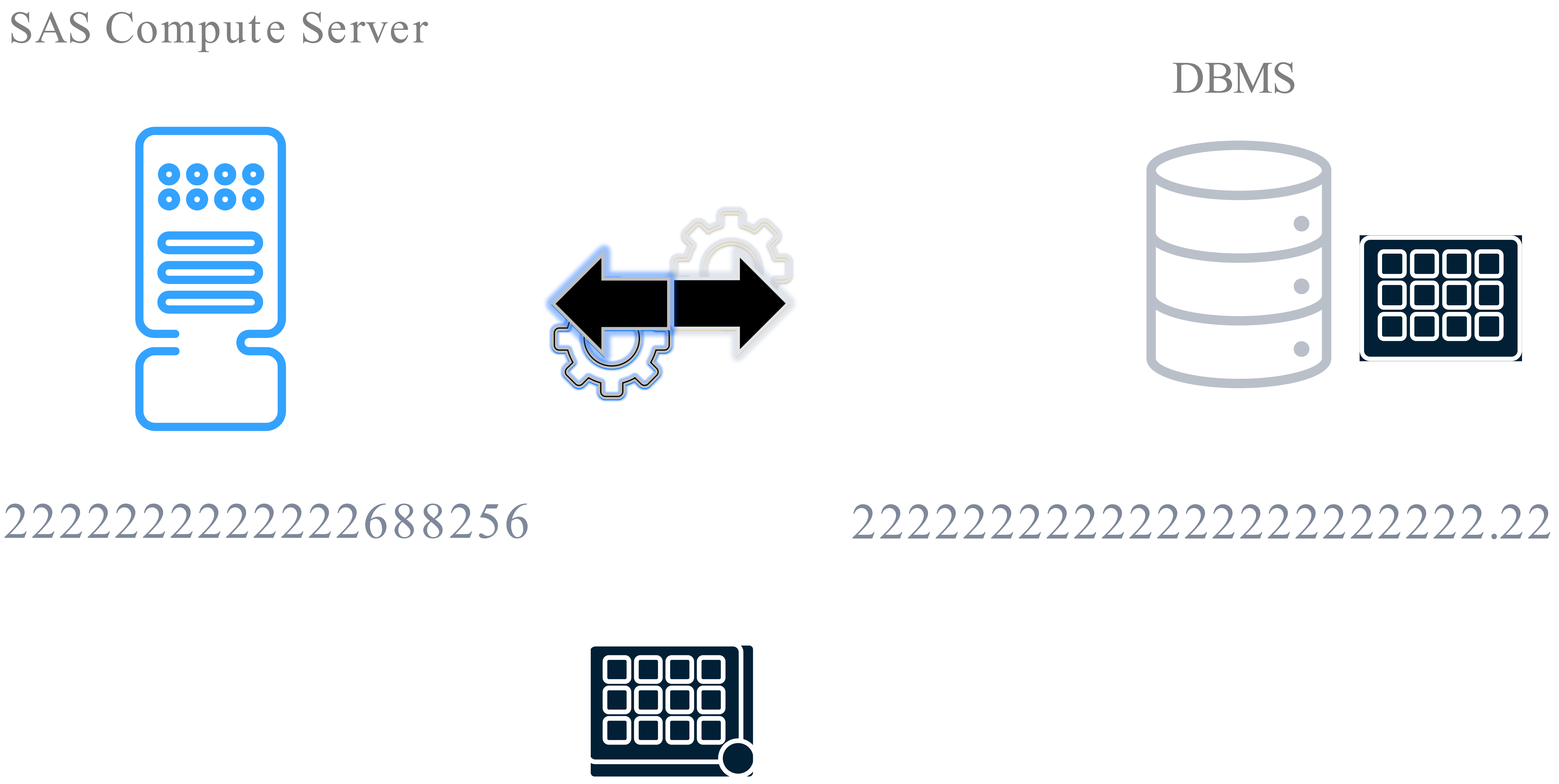


Simple, safe parallel processing

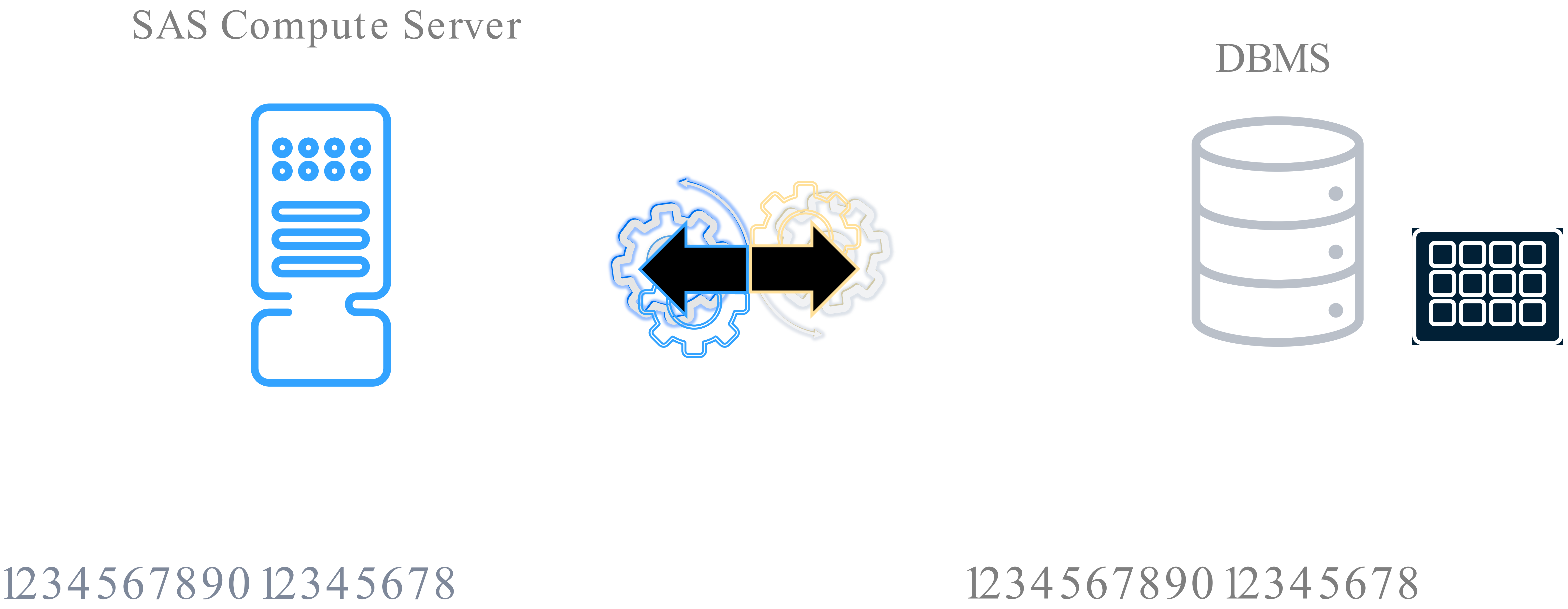


Send DS2 code to MPP platforms for execution

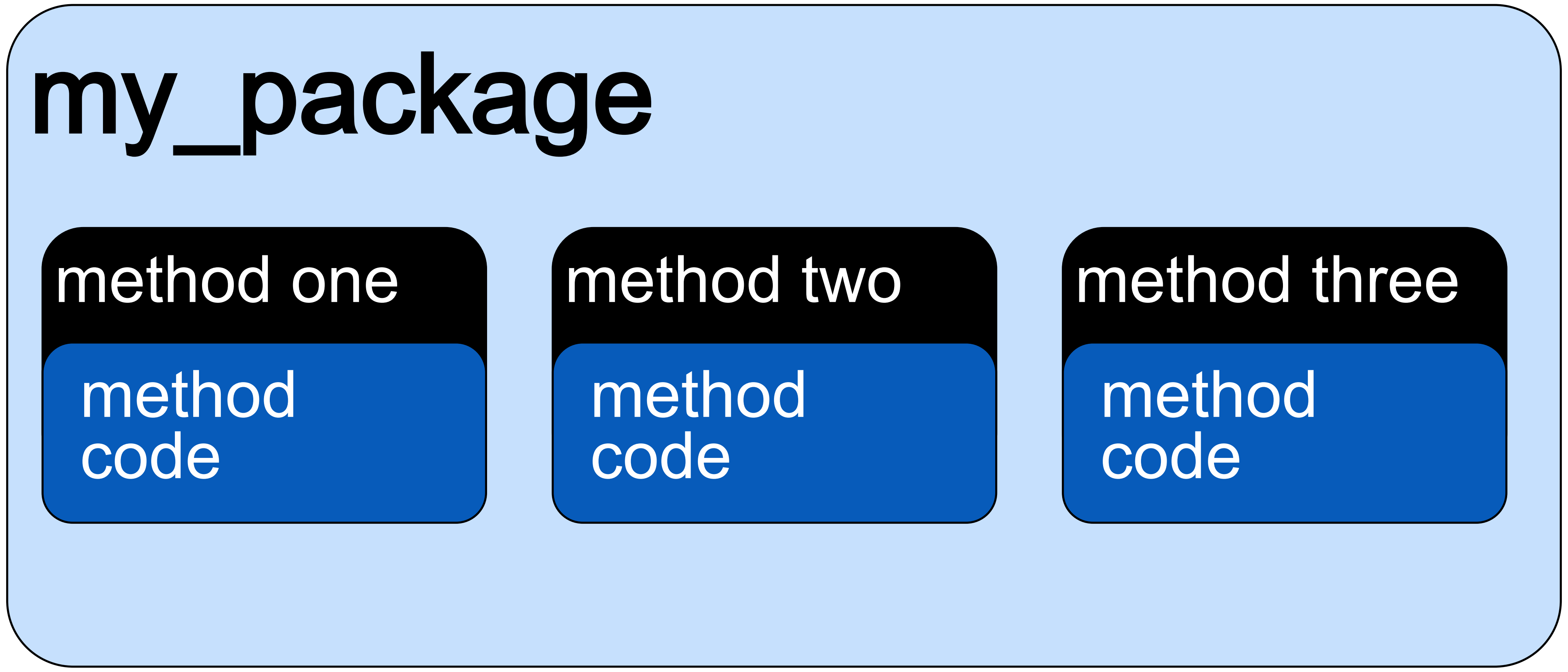
Reading Data from a Database - SAS/Access Engine Behavior



Reading Data from a Database - DS2 Behavior

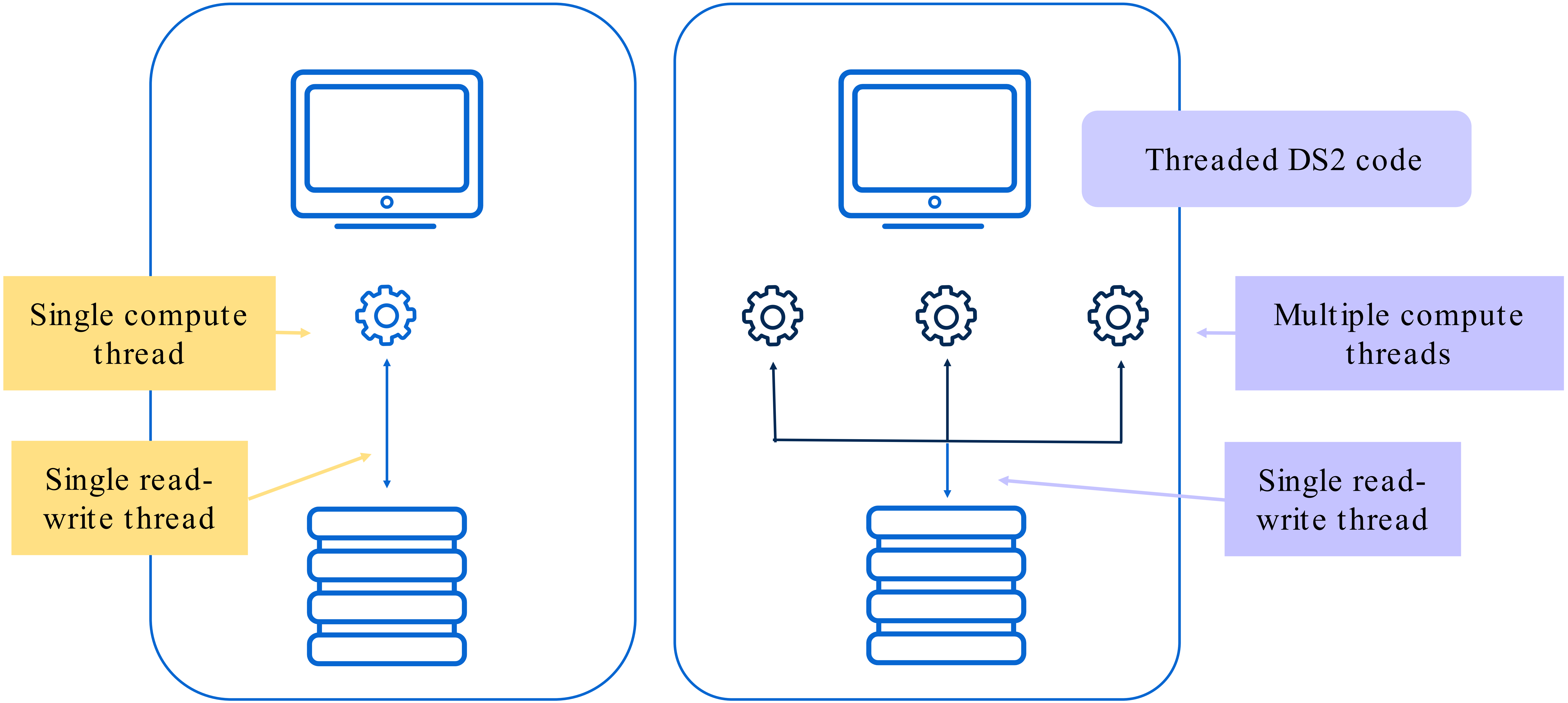


DS2- Object Oriented Programming

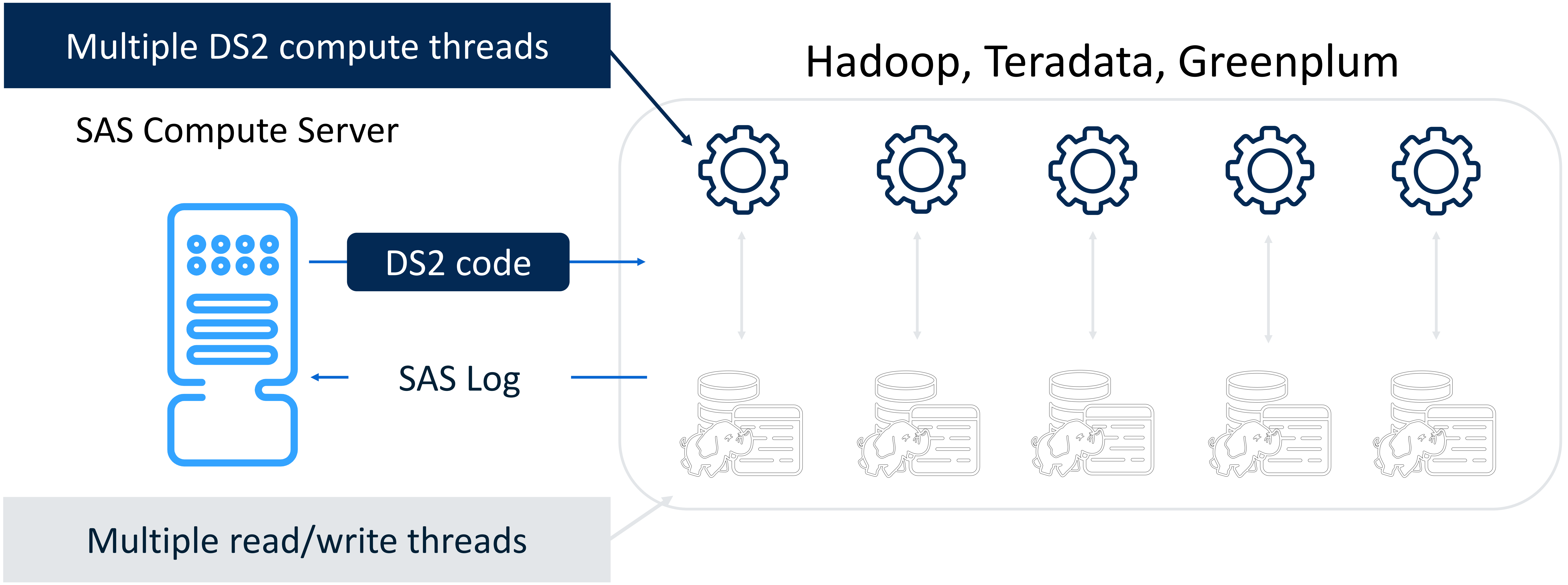


```
dcl package my_package my;
    …
A=my.two;
```

Threaded Processing – Data Step & DS2

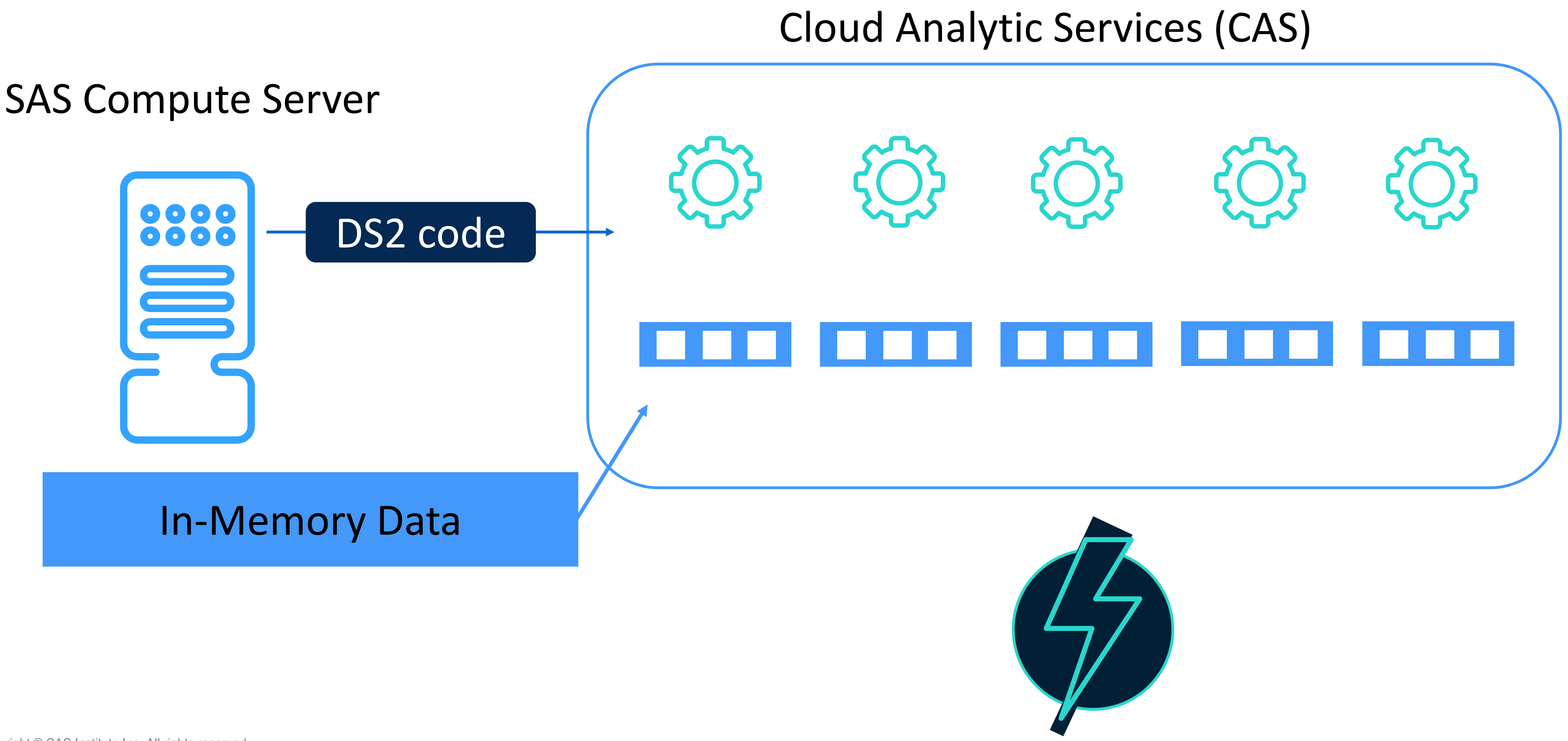


DS2 – Massively Parallel Processing System



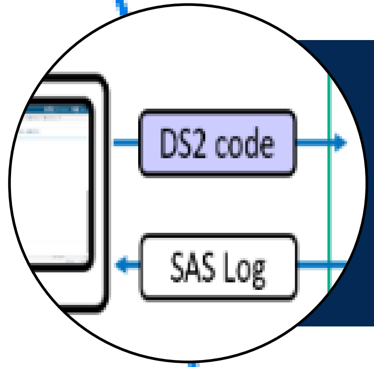
Multiple DS2 compute threads
SAS Compute Server
DS2 code
SAS Log
Hadoop, Teradata, Greenplum
Multiple read/write threads

DS2 – Cloud Analytic Services



When to Use DS2

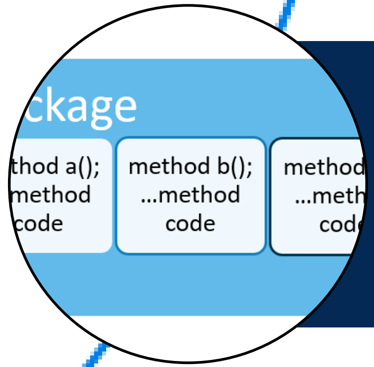
Threaded processing improves CPU-intensive performance



In-database or CAS processing is available

DECIMAL
BIGINT
VARCHAR

Working with high-precision ANSI data types



Leveraging reusable code in DS2 packages

DS2 Block Types

```
proc ds2;
package work.pgk;
   <more program statements>
endpackage;
run;
thread work.thread;
   <more program statements>
endthread;
run;
data my.table;
   <more program statements>
enddata;
run;
quit;
```

In the SAS windowing environment, program blocks without a RUN before the QUIT statement won't execute.

DS2 - Data Blocks

- begin with a DATA statement
- end with an ENDDATA statement
- require a RUN statement to execute.

```
data my.table;




enddata;
run;
```

DS2 - Methods

- Begin with a METHOD statement
- End with an END statement
- Contain all executable statements
- System or user-defined

```
data my.table;
    method myMethod(double x);
        put 'myMethod';
    end;
    method init();
        put 'INIT';
    end;
enddata;
run;
```

INIT Method

System Methods Control DATA block Flow

INIT

- Called when the block begins execution
- Each statement executed only once
- At END, automatically calls the RUN method

RUN Method

RUN

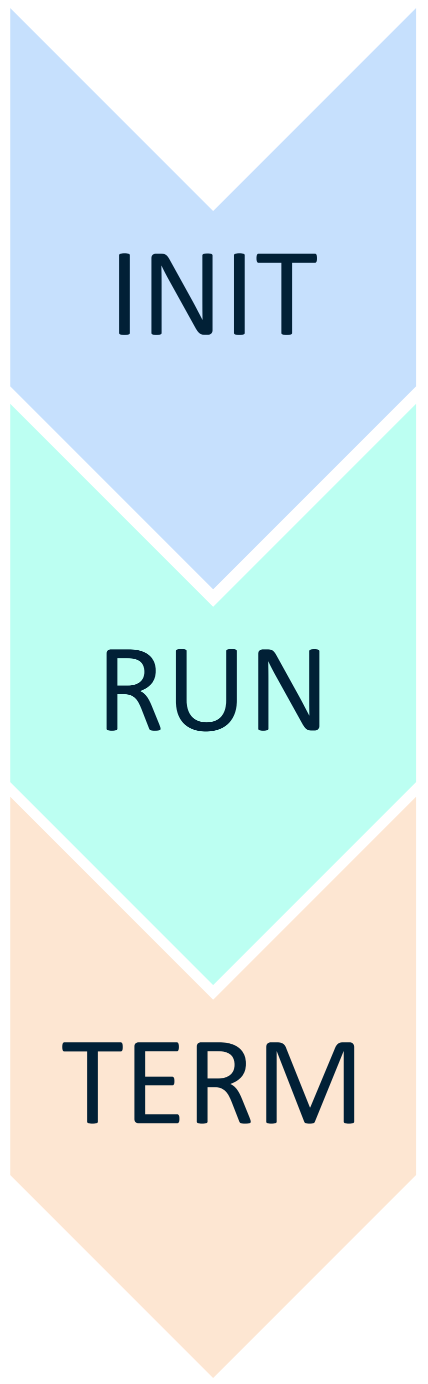
- Starts only when called by INIT
- At END, implicit OUTPUT and RETURN
- Loops until input data is exhausted
- When out of data, automatically calls TERM

TERM Method

Term

- Starts only when called by RUN
- Each statement executed only once
- END terminates program block execution

Null Method



```
data;
   method init(); } NULL Method
   end;
   method run();
      set my.table;
   end;
   method term(); } NULL Method
   end;
enddata;
run;
```

Declaration Statements

```
data;
   dcl double Amount;
   retain Amount 0;
   method run();
      dcl int year;
      set orion.banks;
      amount=1000;
      do year=1 to 5;
         amount=sum(amount,amount*rate);
      end;
   end;
enddata;
```

SAS Data Step Declarative Statements

Variable Declaration: Variables are automatically created in the DATA step when they are first referenced, but here's an example of declaring and initializing them explicitly	`data example;`
	`   length name $20 age 8;`
	`   name = 'John Doe';`
	`   age = 30;`
	`run;`

Array Declaration: Arrays group variables for easier processing.	`data example;`
	`   array scores[3] score1-score3;`
	`   do i = 1 to 3;`
	`      scores[i] = i * 10;`
	`   end;`
	`run;`

Format Declaration: The FORMAT statement defines how variables should be displayed.	`data example;`
	`   length salary 8;`
	`   salary = 50000;`
	`   format salary dollar8.;`
	`run;`

Label Declaration: Adds descriptive labels to variables for reports or displays.	`data example;`
	`   length id $5;`
	`   label id = 'Employee ID';`
	`   id = 'E123';`
	`run;`

Retain Declaration: Keeps the value of variables across iterations of the DATA step.	`data example;`
	`   retain counter 0;`
	`   counter + 1;`
	`run;`

Keep/Drop Declaration: Specifies variables to include or exclude in the output dataset.	`data example;`
	`   keep name age;`
	`   name = 'Jane Doe';`
	`   age = 25;`
	`run;`

Informat Declaration: Assigns input formats to read raw data into the SAS dataset correctly.	`data example;`
	`   informat birthdate date9.;`
	`   input birthdate $;`
	`   format birthdate date9.;`
	`datalines;`
	`15JAN1990`
	`;`
	`run;`

SAS DS2 Declarative Statements

Variable Declaration (length): In DS2, you explicitly declare all variables using dcl (declare) instead of relying on length or implicit declarations.

```
proc ds2;
   data example;
      dcl varchar(20) name;
      dcl double age;
      method run();
         name = 'John Doe';
         age = 30;
      end;
   enddata;
run;
quit;
```

Array Declaration: DS2 supports arrays, but they are defined differently. DS2 arrays are more like fixed-size collections that don’t have to be declared as array explicitly.

```
proc ds2;
   data example;
      dcl double scores[3];
      method run();
         do i = 1 to 3;
            scores[i] = i * 10;
         end;
      end;
   enddata;
run;
quit;
```

Format Declaration: In DS2,, formatting in DS2 is applied to variables within the DECLARE statement itself, as shown below:

```
proc ds2;
   data example;
      dcl double salary format=dollar8.;
      method run();
         salary = 50000;
      end;
   enddata;
run;
quit;
```

Variable Declaration

```
DECLARE|DCL data-type variable-list
<HAVING LABEL 'string' | FORMAT | INFORMAT> ;
```

```
/* Declare three DOUBLE variables formatted dollar12.2*/
dcl double Var1 Var2 Var3 having format dollar12.2;

/* Declare a high-precision fixed point numeric variable */
dcl decimal(35,5) Var1;

/* Declare a fixed-width character variable labeled 'My Text'*/
dcl char(25) Var1 having label 'My Text';
```

Variable Declaration

```
data;
   dcl double Value;
   method run();
      set orion.banks;
      do Month='JAN','FEB','MAR';
         do i=1 to 3;
            Value=10**i;
            if i > x then output;
         end;
      end;
   end;
enddata;
run;
```

```
WARNING: No DECLARE for referenced variable month;
         creating it as a global variable of type char(3).
WARNING: No DECLARE for referenced variable i;
         creating it as a global variable of type double.
WARNING: No DECLARE for referenced variable x;
         creating it as a global variable of type double.
```

ds2_d04

Variable Declaration

DATA Step

- KEEP
- DROP
- RETAIN

DS2

- KEEP
- DROP
- RETAIN

```
ERROR: Compilation error.
```

DS2 Data Block Structure

Global declarative statements

Local variable declaration

```
data orion.mythread;
   dcl double qtr1-qtr4;
   vararry double contrib[4] qtr1-qtr4;
   method run();
      dcl integer i;
      set orion.employee_donations;
      do i=1 to dim(contrib);
         put contrib[i]=;
      end;
   end;
enddata;
run;
```

Method Definition

Data Block

Iterative DO loop

All executable statements must be within a METHOD block

RUN statement causes execution

Package Blocks

DS2 packages are collections of methods and variables stored in SAS libraries.

- Simplify re-using code in DS2 data and thread blocks.
- Predefined packages ship with SAS to extend DS2 capabilities.
- User-defined packages enable easy, secure sharing of proprietary algorithms.
- Packages enable object-oriented programming for the DS2 language
  - Can accept parameters when instantiated
  - Can include user-defined constructor and destructor methods
  - Global variables declared in a package block are referred to *attributes*.
  - Attributes are private to the package instance and do not affect the PDV

Package Blocks

Local variable declarations (private to the method)

Global attribute declarations

```
package orion.mymethods;
   dcl integer PkgVar1;
   method c2f(double Tc) returns double;
      dcl integer LocVar1;
      return((Tc*9/5)+32);
   end;
   method f2c(double Tf) returns double;
      return((Tf-32)*9/5);
   end;
endpackage;
run;
```

Methods

Package

```
NOTE: Created package mymethods in
      data set orion.mymethods.
```

RUN statement causes execution

DS2 Threads

- Threads are programs stored in SAS libraries and used for parallel processing in DS2
- Similar to DATA blocks in that they
  - Enforce a more structured programming syntax
  - Require all executable code to be included in a method definition
  - Require at least explicitly written system method
  - Global variables appear in the PDV of the data block in which they are used
- Similar to DS2 packages in that they
  - can be re-used when stored in permanent SAS libraries.
  - can contain user-defined methods.
  - can accept parameters

DS2 Thread Block

Global declarative statements

Local variable declaration

```
thread orion.mythread;
  dcl double qtr1-qtr4;
  vararry double contrib[4] qtr1-qtr4;
  method run();
     dcl integer i;
     set orion.employee_donations;
     do i=1 to dim(contrib);
        put contrib[i]=;
     end;
  end;
endthread;
run;
```

Method Definition

Iterative DO loop

Thread Block

RUN statement causes execution

```
NOTE: Created thread mythread in data set orion.mythread.
```

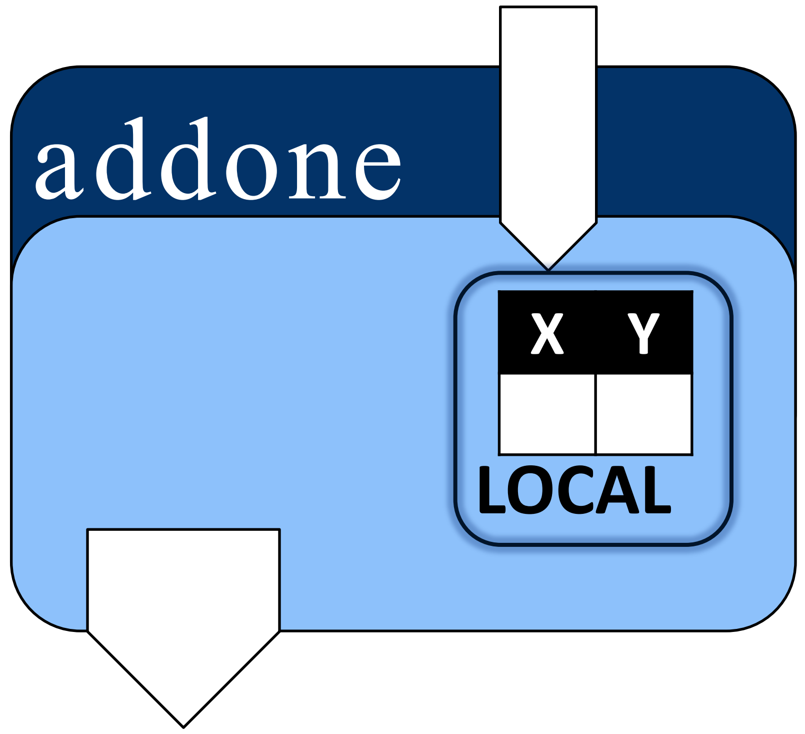
User-Defined Methods

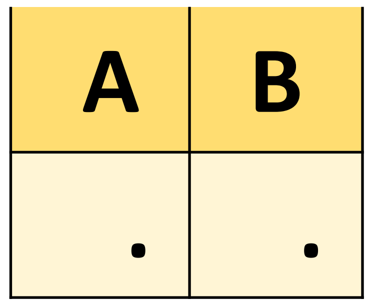
```
METHOD Name  (<IN_OUT>data-type Parameter1 …)
              ,<IN_OUT>data-type ParameterN

              <RETURNS data-type>;

       … more DS2 code …

    <RETURN value>;
END;
```

- can accept parameters
- parameters can be either standard or IN_OUT
- IN_OUT parameter values can be modified
- can return a value
- execute only when called
- can be called multiple times

Returning a value

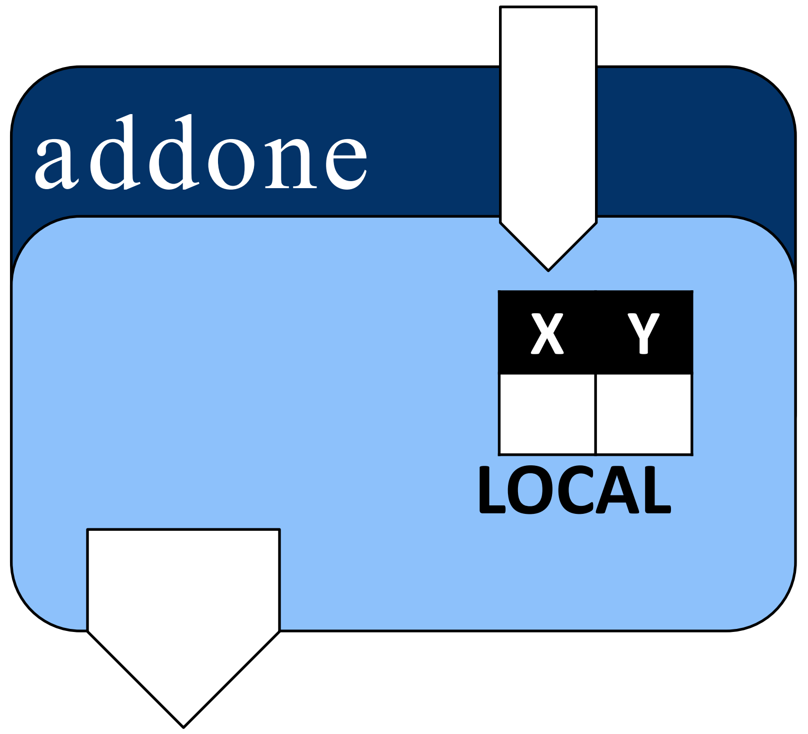
```
dcl int A B;
method addone(int x) returns int;
    dcl int y;
    y=x+1;
    return y;
end;
method init();
    B=3;
    A=addone(B-2);
end;
```





DS2 Return Statement

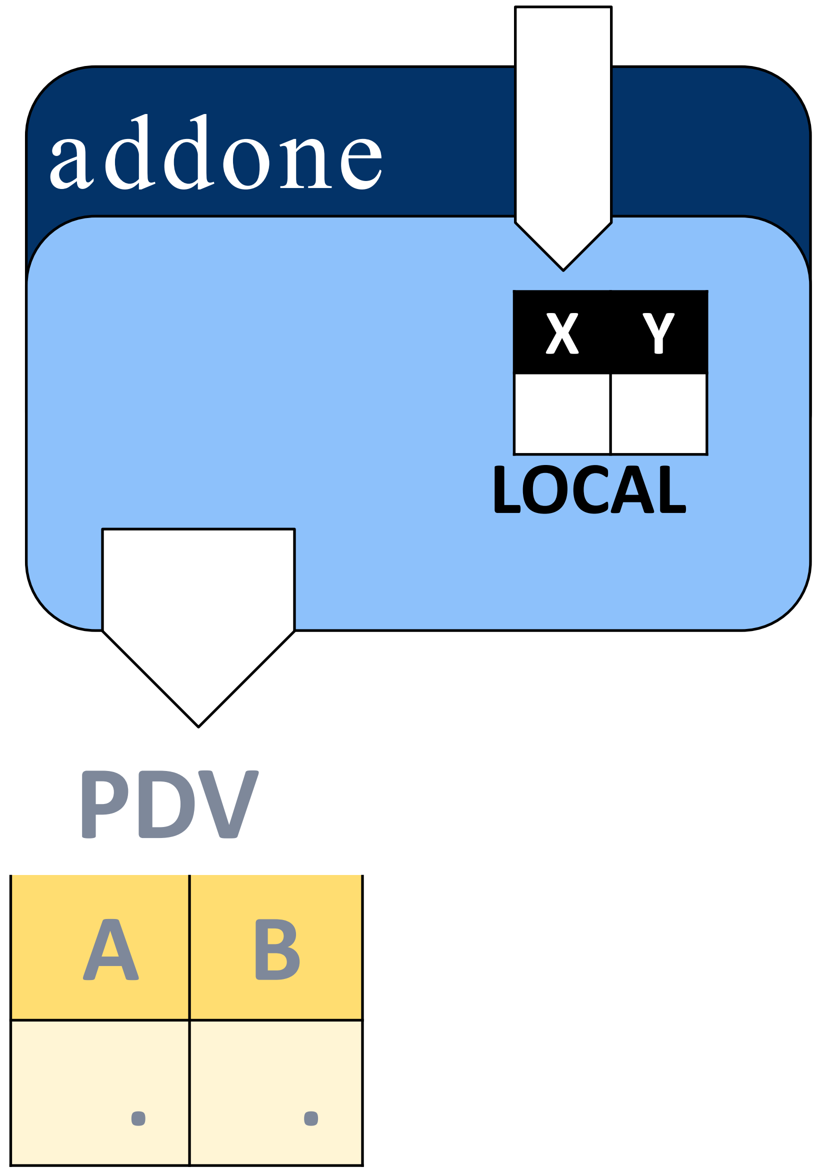
```
dcl int A B;
method addone(int x) returns int;
    dcl int y;
    y=x+1;
    return y;
end;
method init();
    B=3;
    A=addone(B-2);
end;
```



A	B
.	.

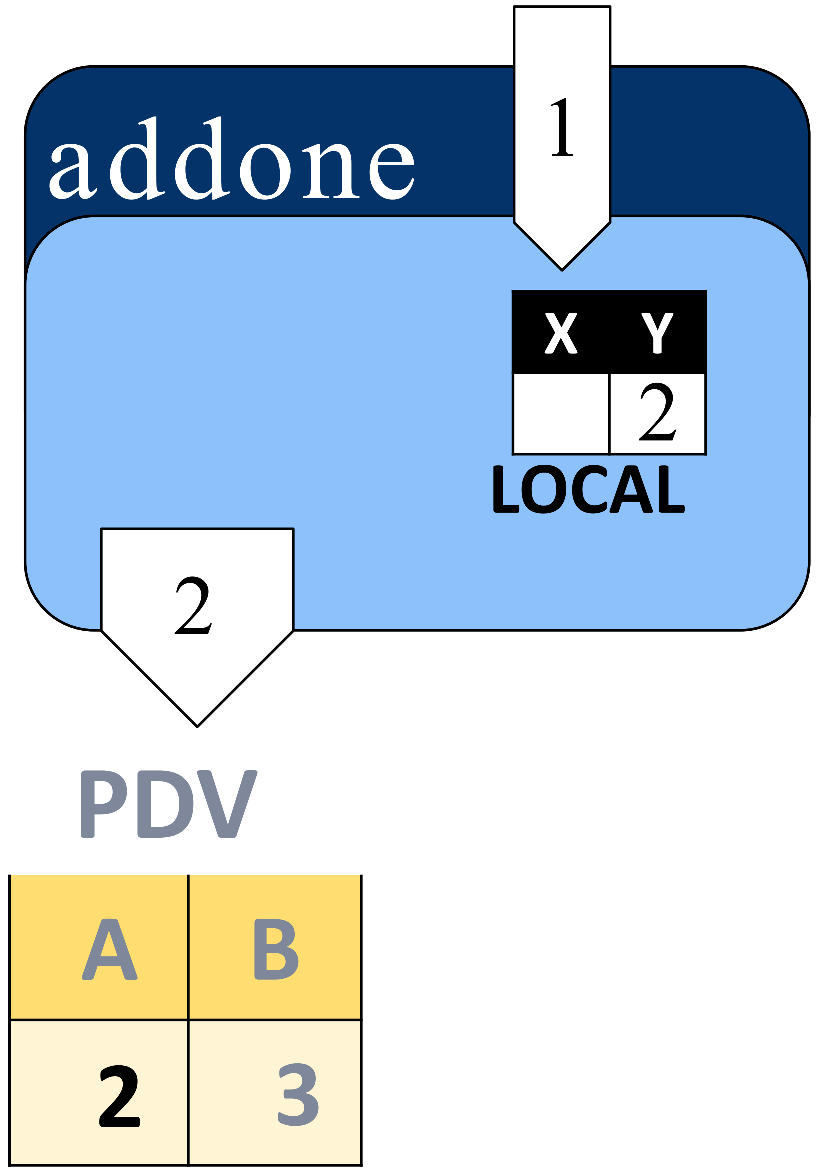
Standard Parameters

```
dcl int A B;
method addone(int x) returns int;
    dcl int y;
    y=x+1;
    return y;
end;
method init();
    B=3;
    A=addone(B-2);
end;
```



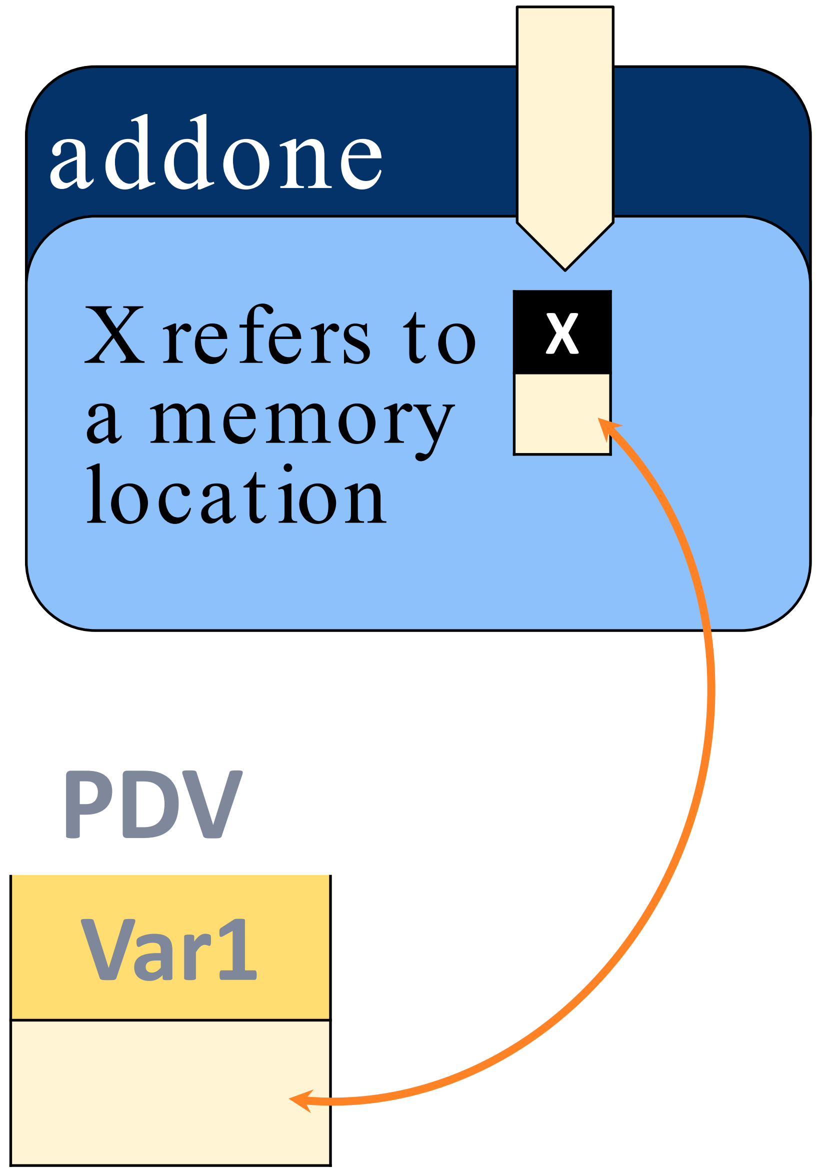
Standard Parameters

```
dcl int A B;
method addone(int x) returns int;
   dcl int y;
   y=x+1;
   return y;
end;
method init();
   B=3;
   A=addone(B-2);
end;
```



IN_OUT Parameters

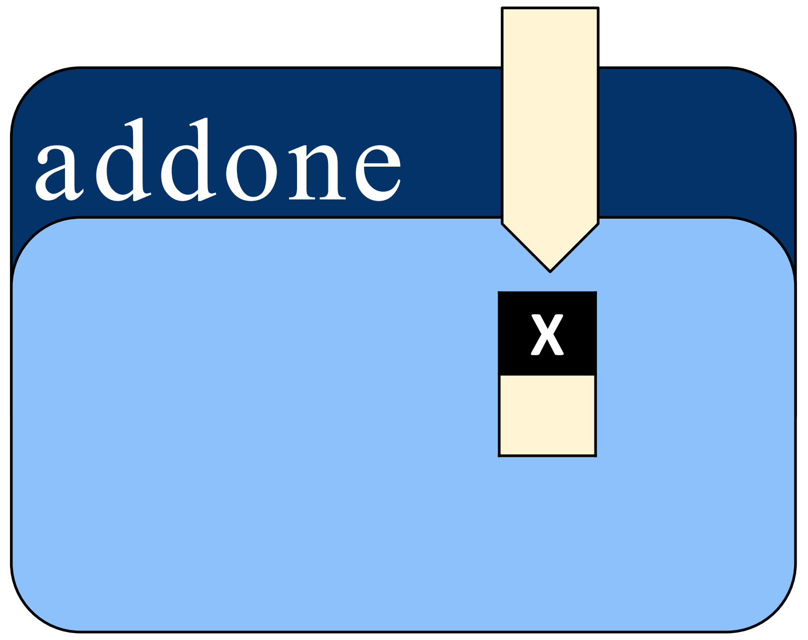
```
method addone(in_out int X);
   x=x+1;
end;
```

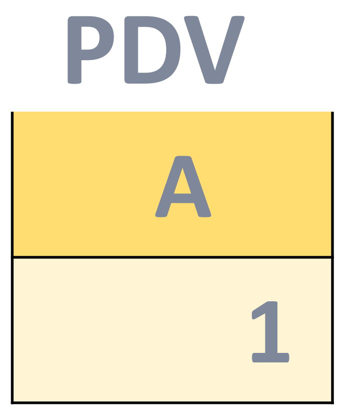


```
ERROR: Parse failed:  method addone
(in_out  >> vararray <<  int x[4]);
```

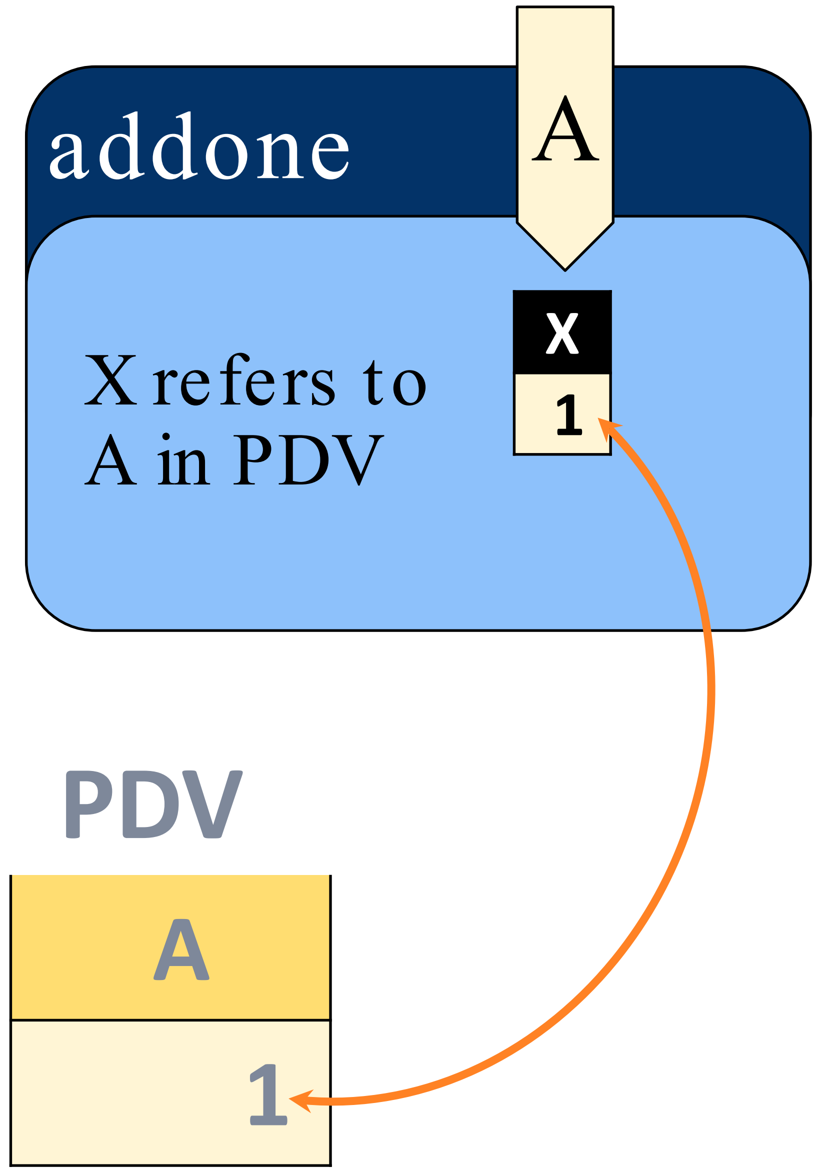
```
ERROR: In call of addone: argument 1 is 'in_out'; therefore, the type
       of the argument must be identical to the type of the method
       parameter (requires int, found double)
```

```
dcl int A;
method addone(in_out int X);
   x=x+1;
end;
method init();
   A=3-2;
   addone(A);
end;
```

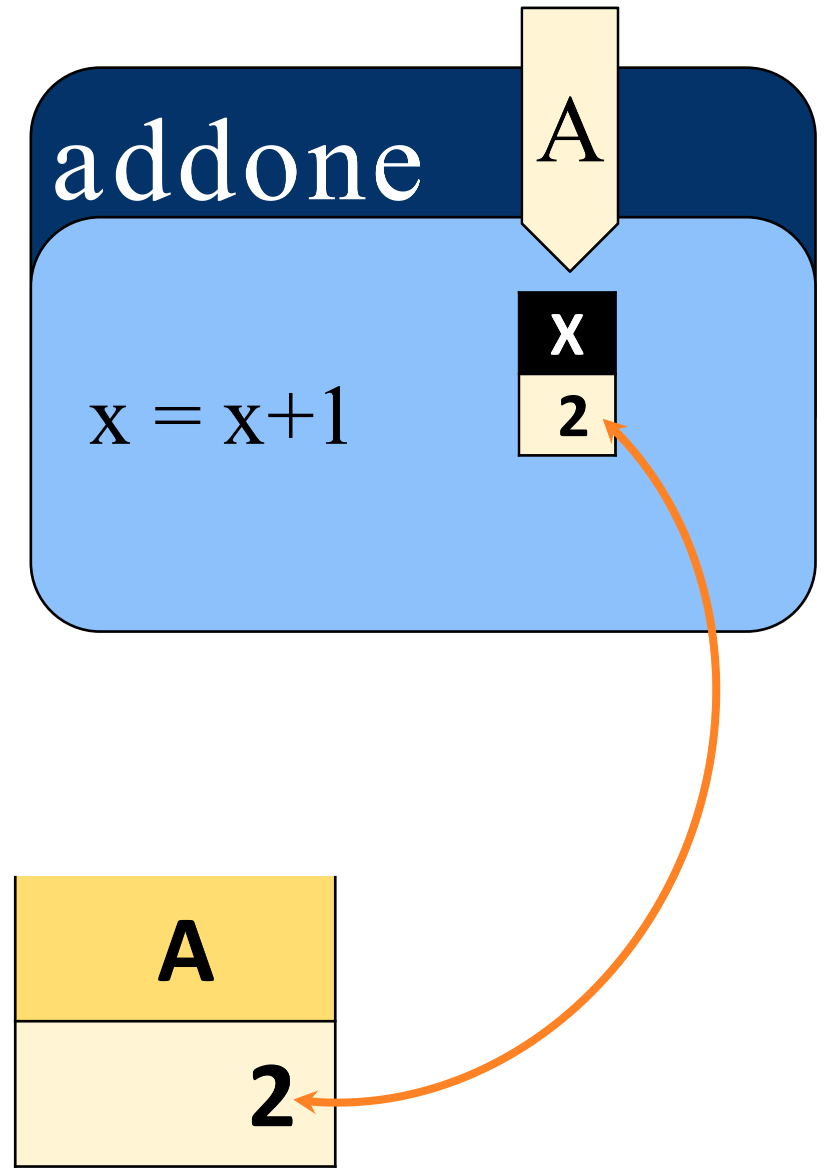




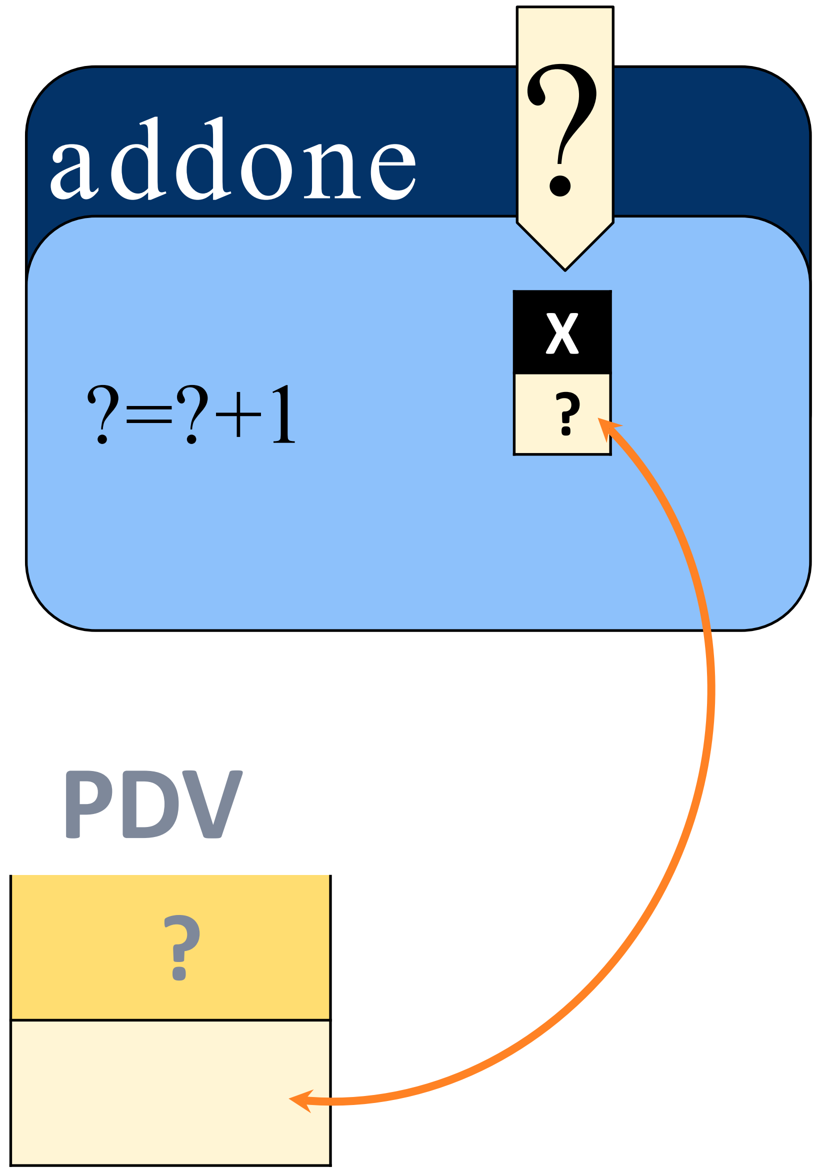
```
dcl int A;
method addone(in_out int X);
    x=x+1;
end;
method init();
    A=3-2;
    addone(A);
end;
```



```
dcl int A;
method addone(in_out int X);
   x=x+1;
end;
method init();
   A=3-2;
   addone(A);
end;
```



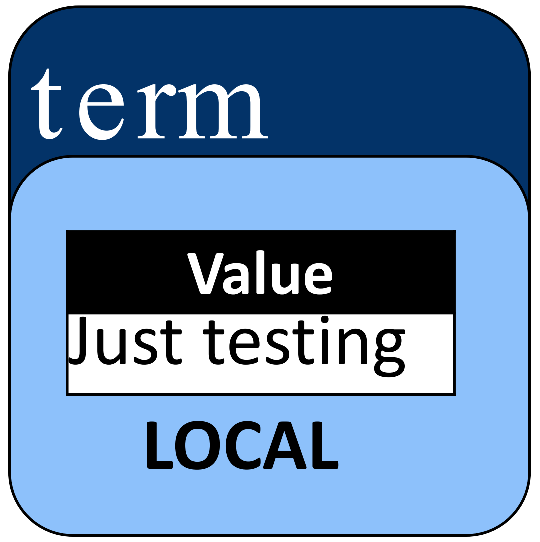
```
method addone(in_out int X);
   x=x+1;
end;
method init();
   addone(3-2);
end;
```



```
ERROR: Compilation error.
ERROR: In call of addone: argument 1 is 'in_out'; therefore, the
       argument must be a modifiable value.
ERROR: Line 22: Attempt to obtain a value from a void expression.
```

Variable Scope

```
data;
   dcl double Value;
   method init();
      Value=5000;
   end;
   method run();
      set orion.banks;
      Total=sum(value,value*rate);
   end;
   method term();
      dcl char(12) Value;
      Value='Just Testing';
      put Value=;
   end;
enddata;
run;
```



PDV

Value N 8	Name $ 29	Rate N 8	Total N 8

DATA step vs. DS2 data block

```
/* Section 1 */
if _n_ =1 then do;
   Text='**> Starting';
   put Text;
end;
```

```
/* Section 2 */

   set orion.banks end=last;
   put _all_;
```

```
/* Section 3 */
if last then do;
   Text='**> All done!';
   put Text;
end;
```

```
data _null_;
 /* Section 1 */
method init();
    Text='**> Starting';
    put Text;
 end;

/* Section 2 */
method run();
   set orion.banks;
   put _all_;
end;
/* Section 3 */
method term();
   Text='**> All done!';
   put Text;
end;

enddata;
run;
```

Converting to DATA Step DS2

PROC DSTODS2 IN=*fully-qualified-DATA-step-file-name*
OUT=*ds2-program-file-name*
OUTDIR=*path-for-output-file***;**

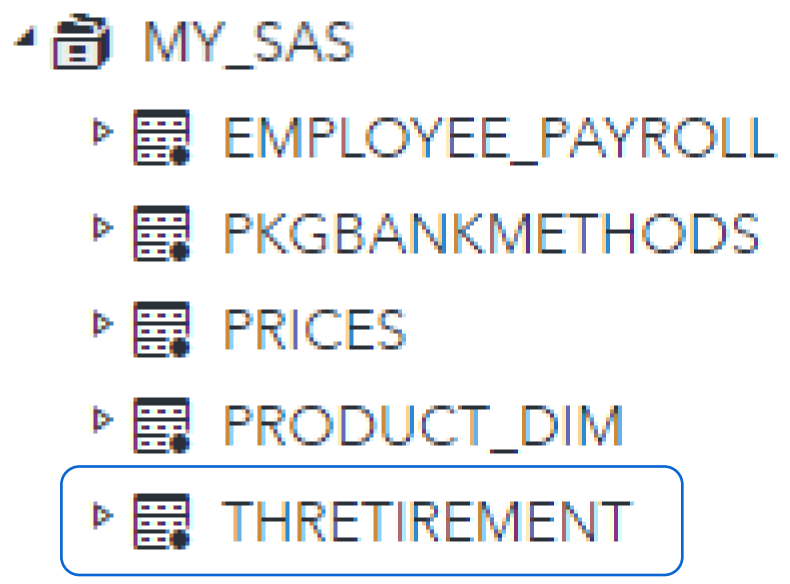
```
proc dstods2 in="&path/demos/ds203d02a.sas"
             out="ds203d02a_ds2.sas"
             outdir="&path/demos/";
run;
```

```
   set orion.banks /* END = LAST */;
```

Converting a DS2 data block to a Thread

```
thread my_sas.thRetirement;
 dcl package my_sas.pkgBankMethods bank();
    dcl double valueInitial valueFinal having format dollar12.2;
    keep Employee_ID, Employee_Name value:;
    dcl int _count;
    keep Employee_ID, Employee_Name value:;
    method run();
       dcl int duration;
       set orion.employees;
       valueInitial=Salary*.1;
       Duration=year(today())-year(employee_hire_date);
       valueFinal=bank.interest(ValueInitial,.04, Duration);
    end;
    method term();
       put '*** Thread' _threadid_ 'of' _nthreads_ 'processed'
           _count 'rows on' _hostname_;
    end;
endthread;
```

```
NOTE: Created thread thretirement in data set my_sas.thretirement.
```


MY_SAS
EMPLOYEE_PAYROLL
PKGBANKMETHODS
PRICES
PRODUCT_DIM
THRETIREMENT

Using A Thread In A Data Block

```
data;
   dcl thread my_sas.thRetirement ret;
   method run();
      set from ret;  threads=3;
   end;
enddata;
run;
```

threadid=1

Total	Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
.	.	.	.	.	.

threadid=2

Total	Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
.	.	.	.	.	.

threadid=3

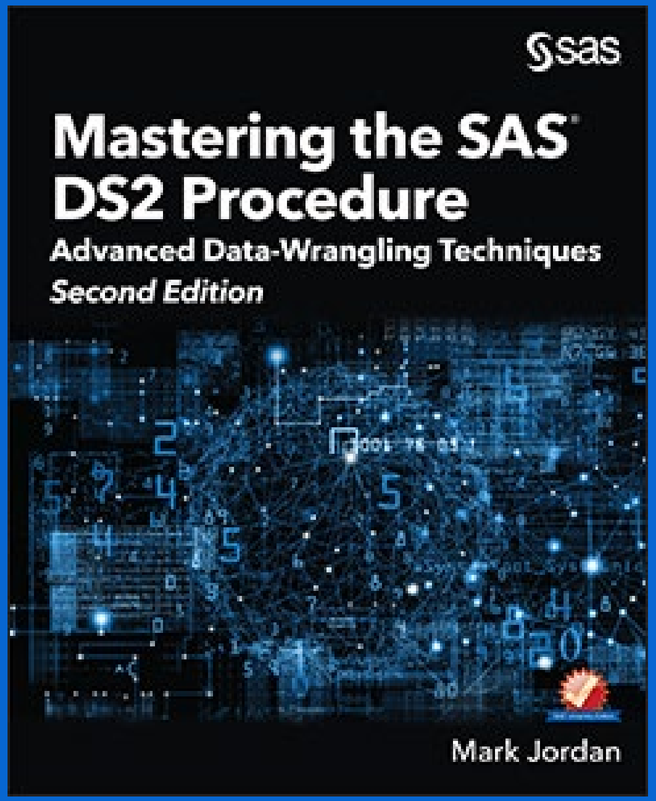
Total	Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
.	.	.	.	.	.

Using DS2 Threads

- The THREADS= option specifies the number of threads to execute in parallel.
  - Over threading can negatively affect performance.
  - &SYSNCPU is a good starting point

```
set from ret threads=&SYSNCPU;
```

	Charu Shankar	Mark Jordan
EMAIL	Charu.shankar@sas.com	Mark.Jordan@sas.com
BLOG	https://blogs.sas.com/content/author/charushankar/	http://sasjedi.tips
TWITTER	CharuYogaCan	@SASJedi
LINKEDIN	https://www.linkedin.com/in/charushankar/	https://www.linkedin.com/in/sasjedi/
AUTHOR PAGE		http://support.sas.com


sas
Mastering the SAS DS2 Procedure
Advanced Data-Wrangling Techniques
Second Edition
Mark Jordan