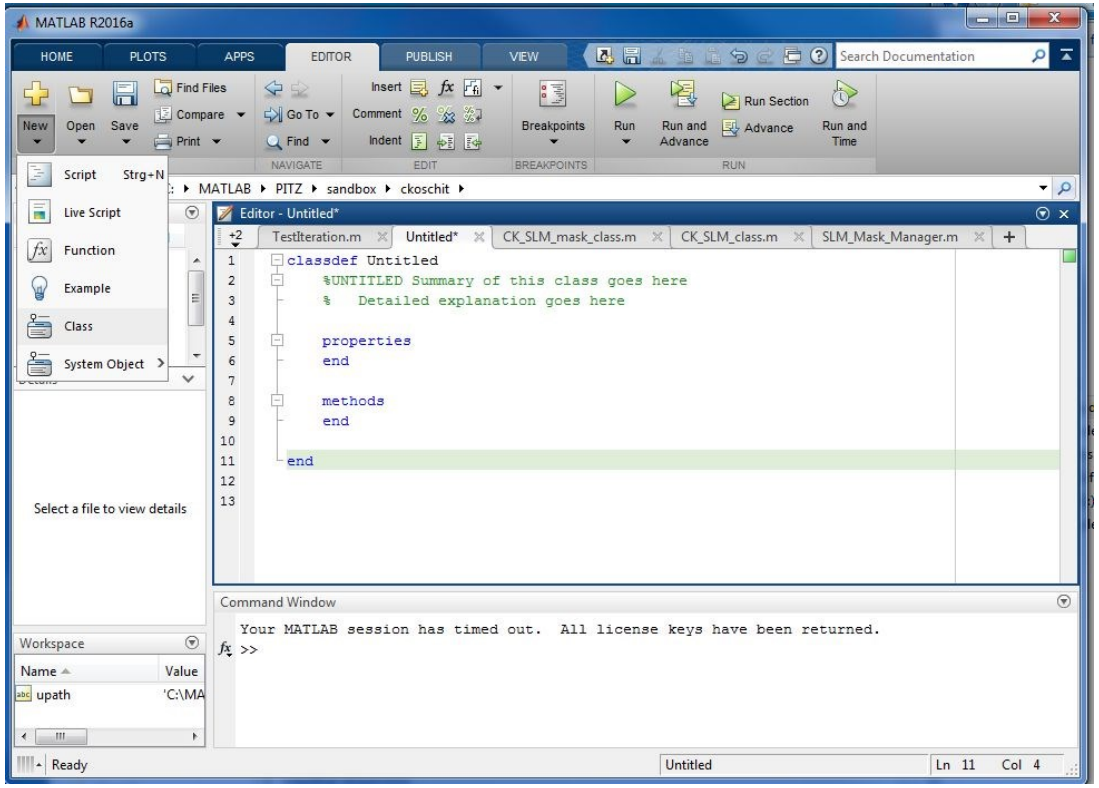
Introduction in OOP at PITZ

Improving Code Management by Object Oriented Programming

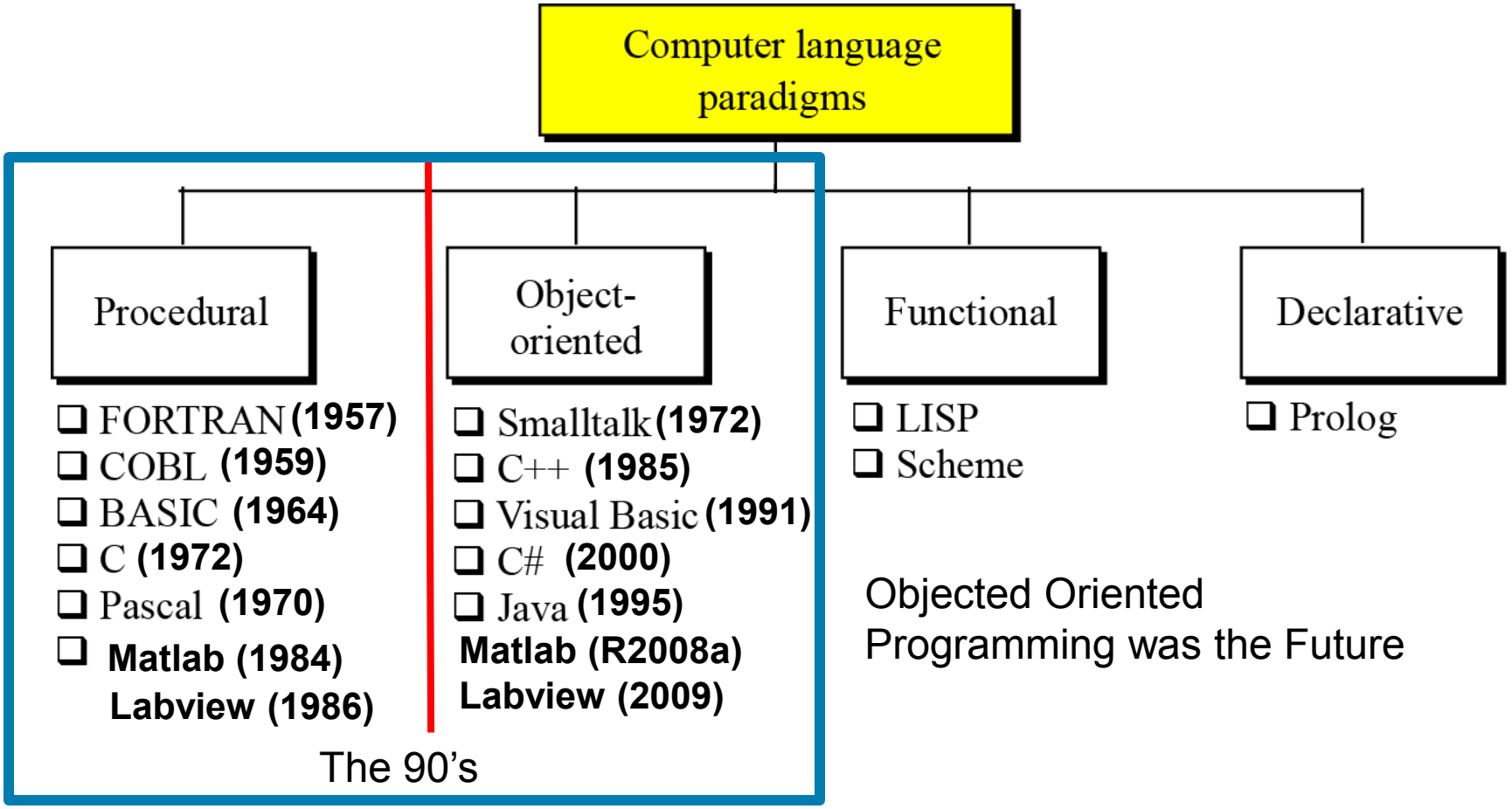


GOAL:
Reduce code duplication
Improve maintainability
and readability

HELMHOLTZ ASSOCIATION

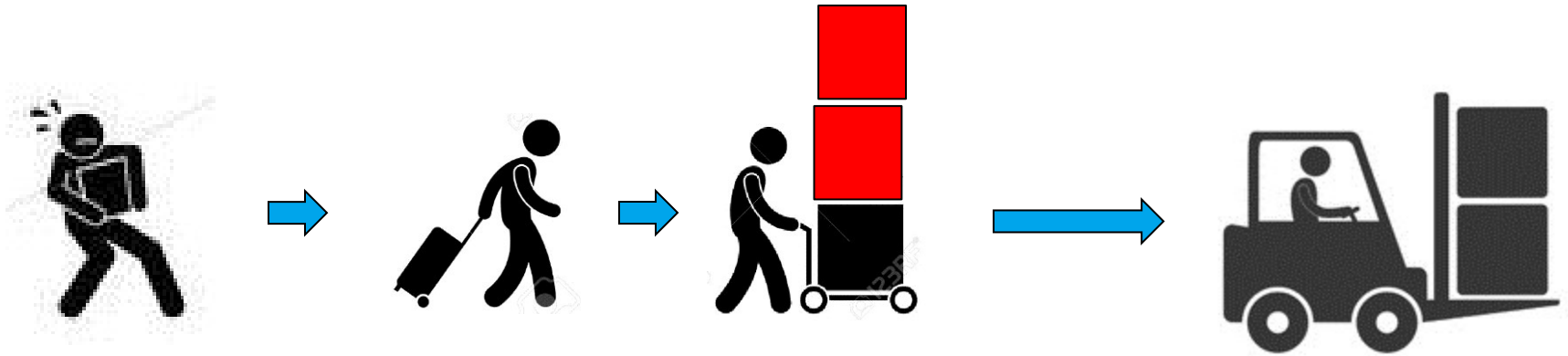
DESY

Programming Paradigms



Why don't physicist use OOP?

Physicists write Code differently

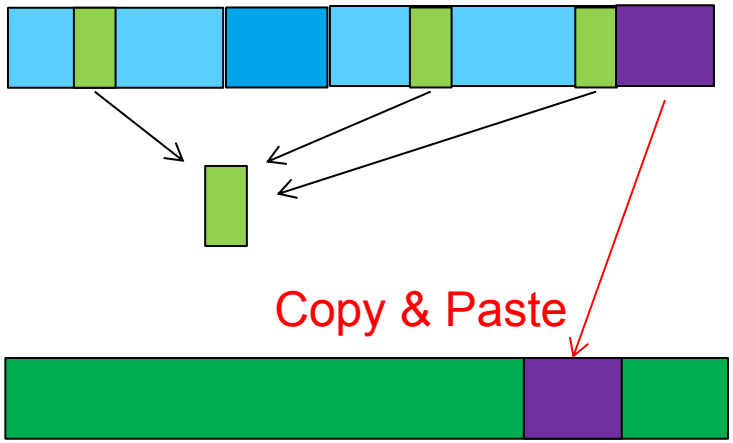


Console:
- run commands
- assign values

Script:
- Loops
- Conditional statements

Functions:
- Recursive functions
- Optional input & output

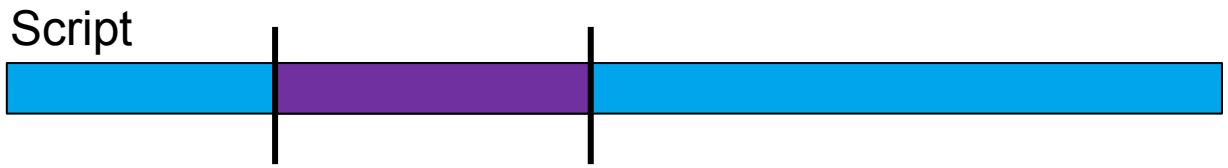
Objects:
- Inheritance
- Encapsulation
- Polymorphism



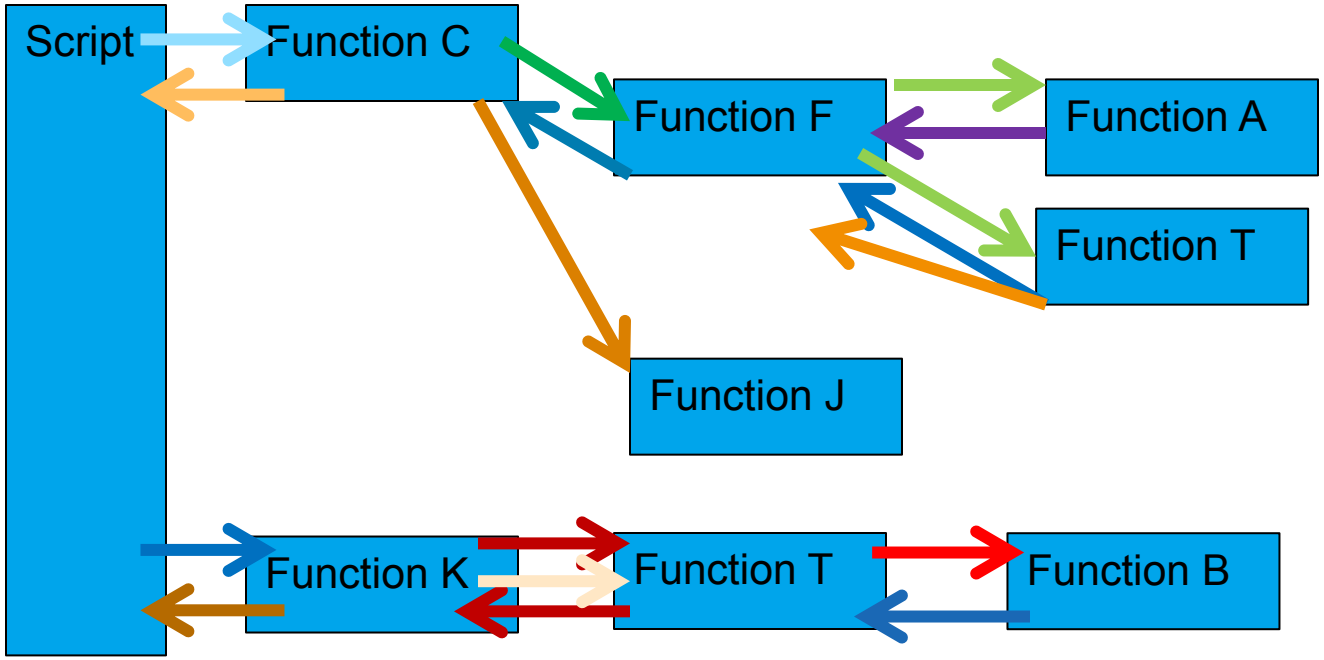
Functions/Methods/Procedures
- Should enclose any isolated task
- No code block (function, script) longer than **100 lines of code**

https://en.wikipedia.org/wiki/Unix_philosophy

Dataflow



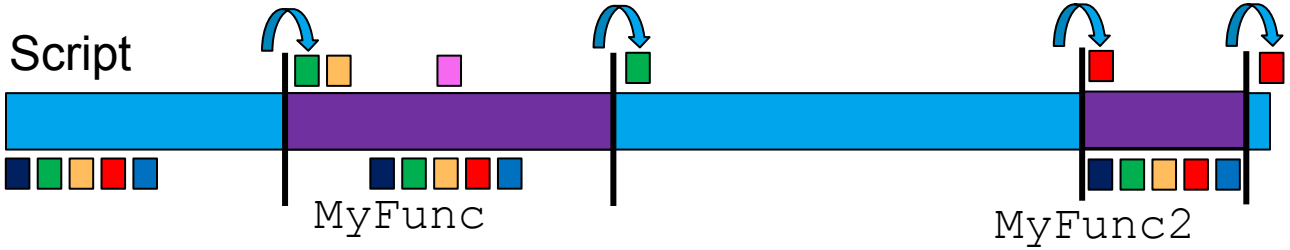

```
function argout = MyFunc(argin)
```



Data passing requires a lot of function input/ouput fields to be modified and the code where they are used

I now want to use data from A in B.
Oh No! I have to change ALL the functions (→data pipeline)

Global Variables




```
function argout = MyFunc(argin1,argin2 )
global globvars
```

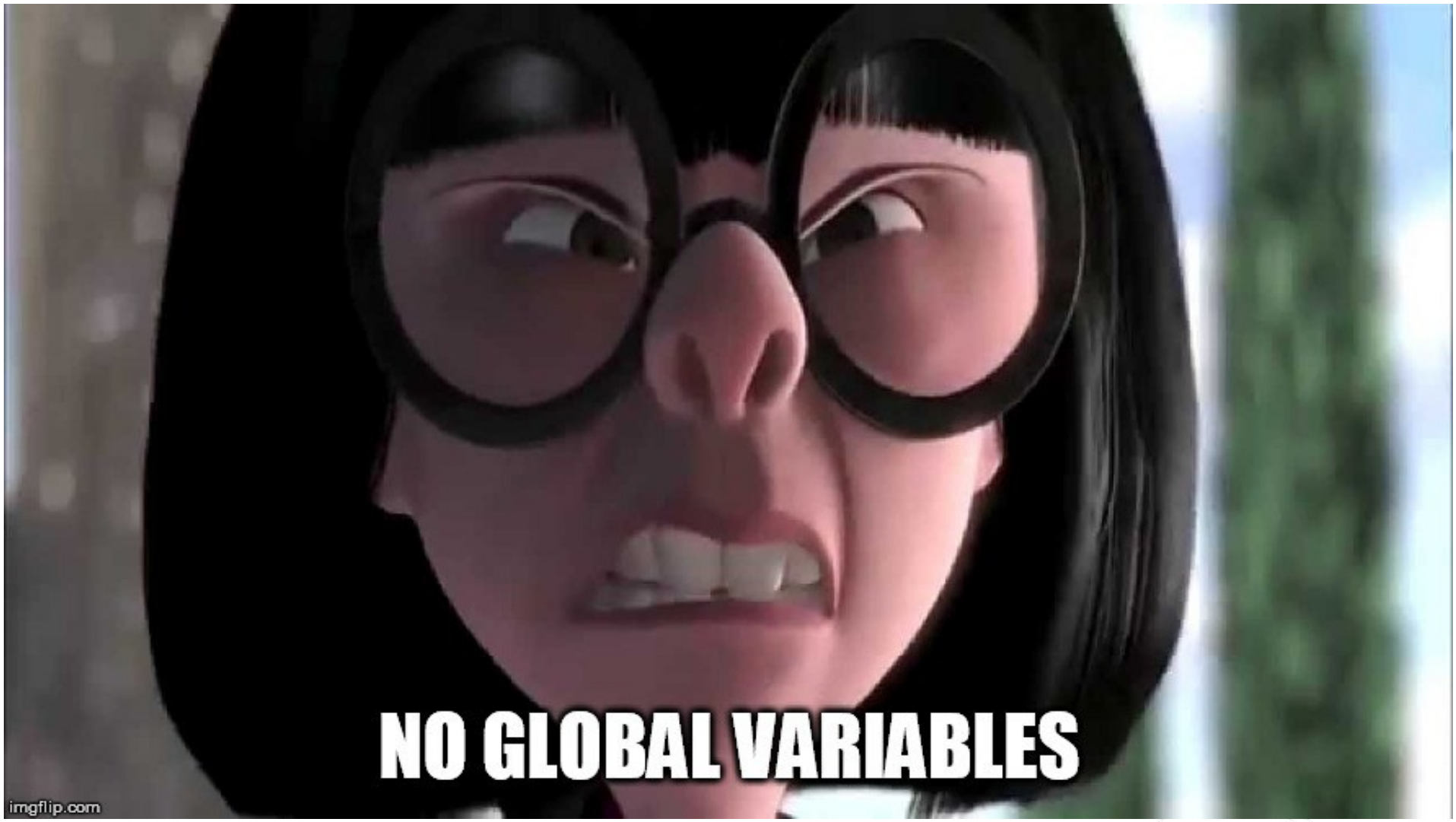
```
function argout = MyFunc2(argin)
global globvars
```

- No Access Control
- Concurrency Issues
- Namespace pollution
- Testing and Confinement

Really Bad Reasons to Use Global Variables: "I don't want to pass it around all the time."

http://wiki.c2.com/?GlobalVariablesAreBad

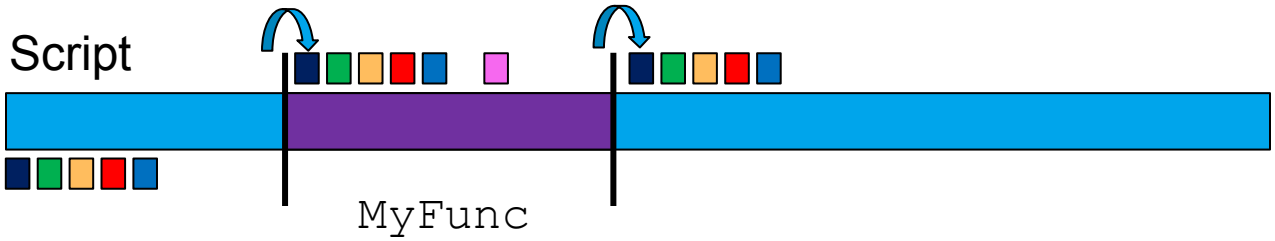
Global Variables



Really Bad Reasons to Use Global Variables: "I don't want to pass it around all the time."

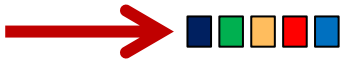
http://wiki.c2.com/?GlobalVariablesAreBad

Big Variables




```
function [bigout = MyFunc(bigin,argin2 )
```

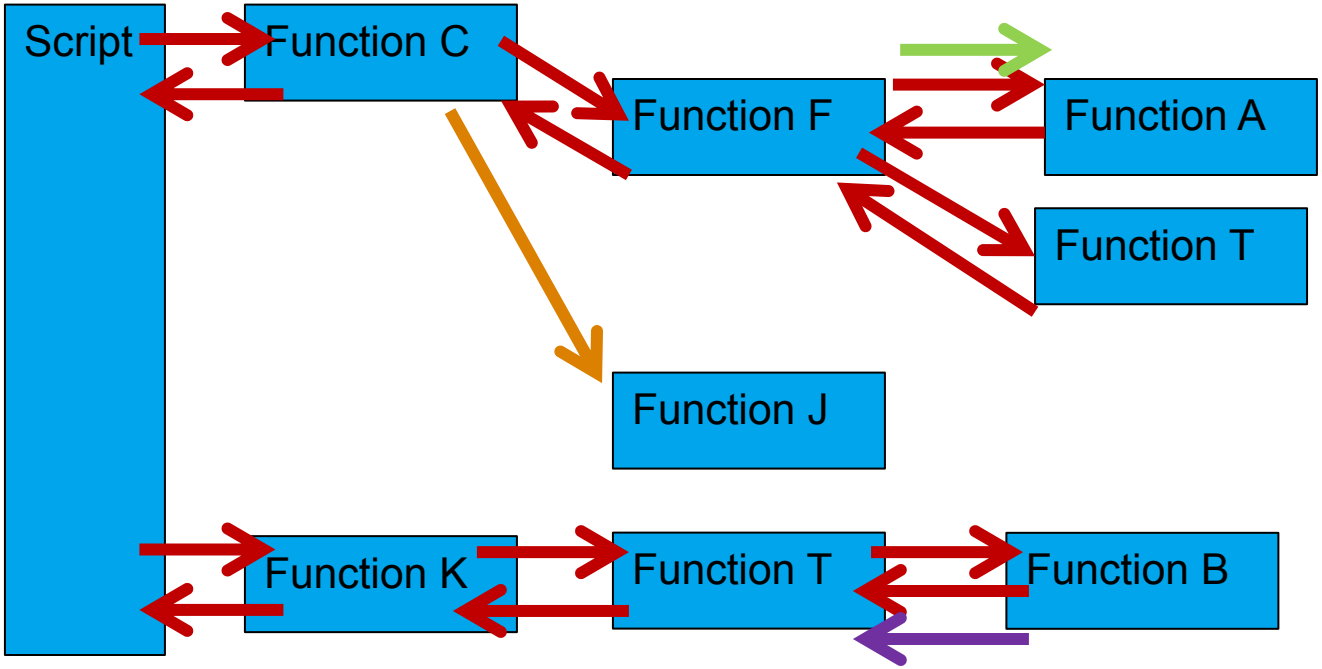
```
classdef
properties
```



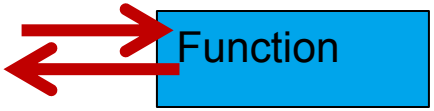
```
end

methods
```


```
end
```



These types of function are special



```
addr.IRSactR = 'PITZ.LASER/LASER_3D/SN_27000839/VALUE.RDBK'; %position read
addr.IRSactW = 'PITZ.LASER/LASER_3D/SN_27000839/VALUE.SP';   %position write
addr.IRSactA = 'PITZ.LASER/LASER_3D/SN_27000839/COMMAND';    %position act
addr.IRSactS = 'PITZ.LASER/LASER_3D/SN_27000839/STATUS';     %position status
```

Setup adresses

```
%Homing Motor
[s] = xcomm(addr.IRSactS,'connection','sync'); %check value
spos = s.data; %status bit check: homed?
if ~bitget(spos,11) %else home device
    % i=1
    tic;
    [~] = xcomm(addr.IRSactA,1,'ACCESS','W','connection','sync'); %Home stage
    disp('Go home, stage. You''re drunk.');
    pause(1);
    while ~(spos == 3080)
        [s] = xcomm(addr.IRSactS); %check value
        spos = s.data;
        pause(0.1);
    end
    disp(['Stage homed after ', num2str(toc),'s'])
else
    disp('Stage already homed.')
end
```

Check Home status

Send Homerun

Wait for Motor to arrive

```
[~] = xcomm(addr.IRSactW,x.sp(i),'ACCESS','W','connection','sync'); %set value
pause(1)
[~] = xcomm(addr.IRSactA,2,'ACCESS','W','connection','sync'); %go!
pause(1)
while ~(round(xpos,3)==round(x.sp(i),3)) || ~(spos == 3080)
    xposold = xpos;
    pause(1)
    [a] = xcomm(addr.IRSactR,'connection','sync'); %check value
    xpos = a.data;
    [s] = xcomm(addr.IRSactS,'connection','sync'); %check value
    spos = s.data;
    [b] = xcomm(addr.IRSactW,'connection','sync'); %check value
    xsp = b.data;
end
```

Send Motor

Wait for Motor to arrive

Let's build a class

```
methods
function PollStatus(obj)
            [s] = xcomm(obj.STATUSadr,'connection','sync');
            obj.mstat=s.data;
            [c] = xcomm(obj.RBKadr,'connection','sync');
            obj.rbkpos = c.data;
            [b] = xcomm(obj.SETadr,'connection','sync');
            obj.setpos=b.data;
            obj.moving=(bitget(obj.mstat,3) & bitget(obj.mstat,5));
end
function ishome=isHomed(Obj);
            [s] = xcomm(obj.STATUSadr,'connection','sync');
            spos = s.data;
            obj.ishome=bitget(spos,11);
            ishome=obj.ishome;
end

function HomeMotor(obj);
           xcomm(obj.ACTadr,1,'ACCESS','W','connection','sync');
end

function SetPosition(obj,pos)
            obj.setpos=pos;
            [~] = xcomm(obj.SETadr,pos,'ACCESS','W','connection','sync');
            pause(0.5);
            obj.GoToSetpoint;
end

function GoToSetpoint(obj)
            [~] = xcomm(obj.ACTadr,2,'ACCESS','W','connection','sync');
end

function WaitMotorArrive(obj)
            while obj.moving
                obj.PollStatus;
                pause(0.2);
            end
end

end
```

```
classdef TL_Doocs_Motor < handle
properties
        %Adresses
        doocs_adr %doocs adrress
        RBKadr    %Readback Field
        SETadr    %Set Value Field
        ACTadr    %Adress to issue Commands
        STATUSadr %Address to read Status
        name      %name derived from address
        %MotorVariables
        mstat      %Motor status bits
        setpos     %set position
        rbkpos     %readback pos
        moving=0;  %moving flag
        ishome=0;  %is homed flag
end
```

Constructor & Destructor

```
function obj=TL_Doocs_Motor(doocs_adr)
   if doocs_adr(end) ~= '/'
     doocs_adr=[doocs_adr,'/'];
   end
   obj.doocs_adr=doocs_adr;
   obj.RBKadr=[doocs_adr,'VALUE.RDBK'];
   obj.SETadr=[doocs_adr,'VALUE.SP'];
   obj.ACTadr=[doocs_adr,'COMMAND'];
   obj.STATUSadr=[doocs_adr,'STATUS'];
   tmp=strsplit(doocs_adr,'/');
   obj.name=tmp{end-1};
   obj.isHomed;
   obj.PollStatus;
end
```

Constructor creates class instance (object):

- Same name as class
- Only Method that does not take object as input
- Must have object as ouput
- Input arguments are optional
- Definition is optional

```
function delete(obj)
%Disconnect Motor ?
end
```

Destructor deletes instance:

- Must be named delete(obj)
- Has only the object as input and no ouputs
- Must not create error

Other Languages

Python:

```
class MyClass:

    proparr = []

    def __init__(self, name):
        self.name = name

    def Method(self, Value):
        self.proparr.append(Value)

    def __del__(self):
        print ("destructor")
```

C++:

```
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area () {return (width*height);}
};

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  cout << "rect area: " << rect.area() << endl;
  return 0;
}
```

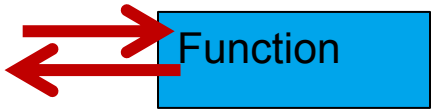
The new script

```
mot=TL_DOOCS_Motor('PITZ.LASER/LASER_3D/SN_27000839/');
if ~mot.ishome; %~mot.isHomed()
    mot.HomeMotor();
    mot.WaitMotorArrive();
end
mot.SetPosition(5);
mot.WaitMotorArrive();
```

Instantiate the object

The .dot notation is equivalent to obj = SetPosition(obj,pos) for invoking methods

- The Class definition serves as a “blueprint” for a datatype with associated methods
- The constructor creates an instance of this class
- Calling methods on this object is called invoke method
- Class definitions make code portable and can hide complexity where it is not necessary (part of Encapsulation)

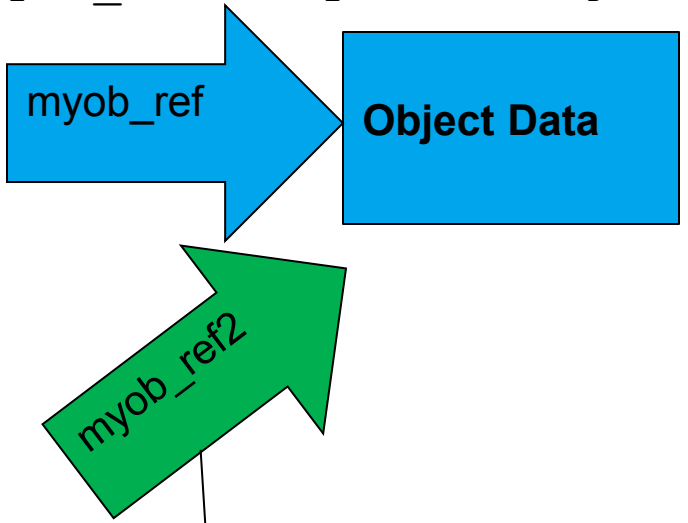


When is the object property structure passed back?

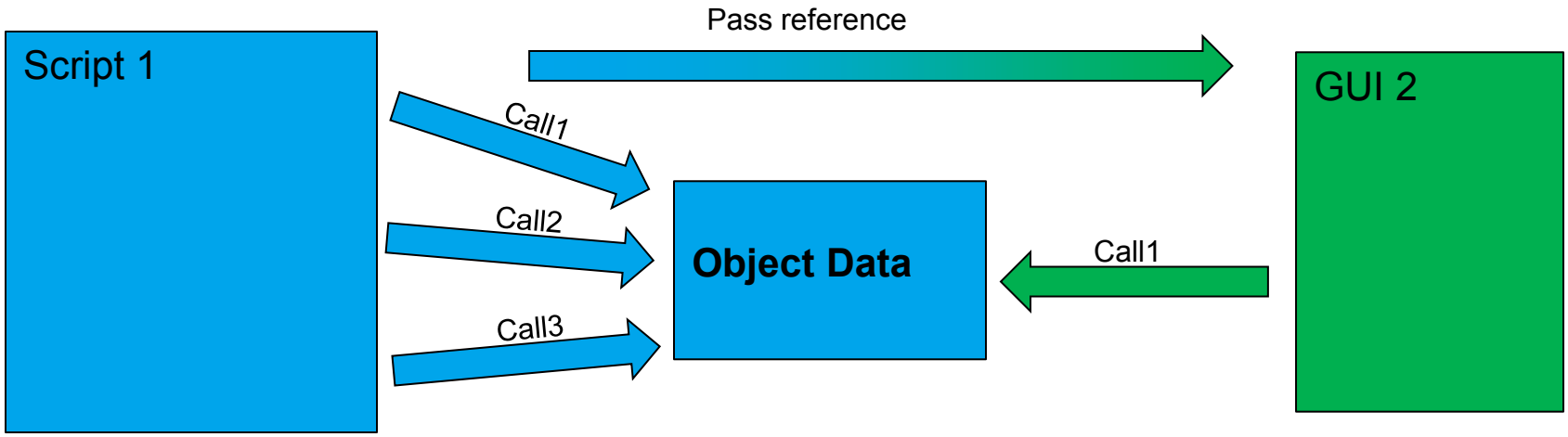
Pro Tip: Distinguish between property calls and method invoking without argument by using empty brackets

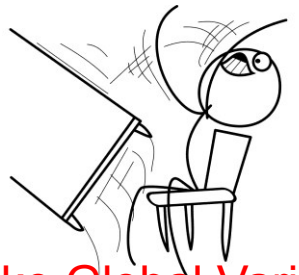
Handle classes

	Value Class	Handle Class
Declaration	`classdef Myclass`	`classdef Myclass < handle`
Constructor	`myob = Myclass(arg)`	`myob_ref = Myclass(arg)`
Copy	`myob2 = myob`	`myob_ref2 = myobref`
Function	`myob = Write(myob,msg)`	`Write(myob_ref,msg)` `myob_ref.Write(msg)`

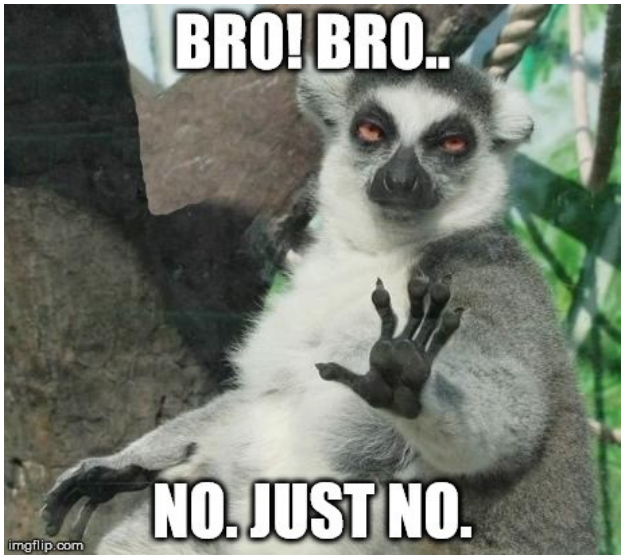


Unexpected behavior due to pointers





Just like Global Variables !!!!



Reference passing

- no explicit names → no naming conflicts
- handshake → turn off ref passing for debugging
- limited ref passes → limited number of suspects
- Lock via Set & Get Functions

Example for a Value Class

Handle Class

- Object Data contains other handles, references
- Used to synchronize (GUIs, scripts)

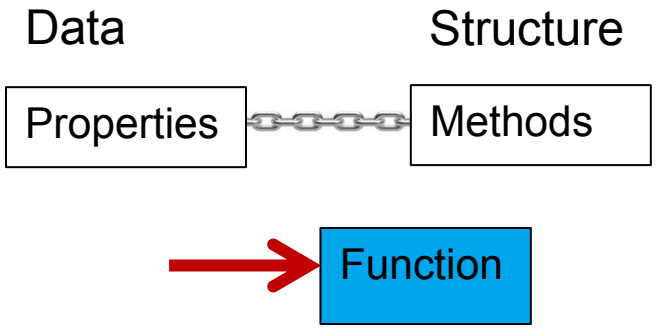
Value Class

- Object Data contains only Values
- New Instances get generated from existing Instances

```
classdef CKTriplets_class
    properties
        d1=0;
        d2=0;
        d3=0;
    end
    methods
          function obj= CKTriplets_class(d1,d2,d3)
          function obj= mtimes(obj,obj2) % obj1*obj2
          function obj= times(obj,obj2) % obj1.*obj2
          function obj= rdivide(obj,obj2) % obj1./obj2
          function obj= plus(obj,obj2) % obj1+obj2
          function obj= minus(obj,obj2) % obj1-obj2
    end
end
```

```
t1 = CKTriplets_class(4,-3,5);
t2 = CKTriplets_class(1,4,12);
t3 = (t1*t1)/t2+t2;
```

Polymorphism



No need for unique function names as type of class + function name define which function to execute

```
mot=USBMotor('USB1');
if ~mot.ishome; %~mot.isHomed()
    mot.HomeMotor();
    mot.WaitMotorArrive();
end
mot.SetPosition(5);
mot.WaitMotorArrive();
```

```
classdef USBMotor < handle
    properties
        USBAdr
        %mstat
        setpos
        rbkpos
        moving=0;
        ishome=0;
    end
    methods
        function isHomed(obj) end
        function HomeMotor(obj) end
        function WaitMotorArrive(obj) end
        function SetPosition(obj,pos) end
    end
end
```

Polymorphism allows code writing without explicit knowledge which function is executed

Summary

<u>You should consider using Objects when:</u>

- Bundled data appears in your code 
- Possibly in multitudes 
- Function headers are overloaded 
- Data passing becomes hard 
- You start eyeballing with global variables 

<u>What you should have learned today</u>

- What classes & objects are
- Understand Encapsulation, Data Pipeline & Polymorphism

A story of Boy Scouts and Yaks

The boy scout rule

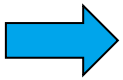
"Always leave the code you're editing a little better than you found it"

- Robert C. Martin (Uncle Bob)

{codemotion}

MADRID · NOV 27-28 · 2015

```
addr.IRSactR = 'PITZ.LASER/LASER_3D/SN_27000839/VALUE.RDBK';
%position read
addr.IRSactW = 'PITZ.LASER/LASER_3D/SN_27000839/VALUE.SP';
%position write
addr.IRSactA = 'PITZ.LASER/LASER_3D/SN_27000839/COMMAND';
%position act
addr.IRSactS = 'PITZ.LASER/LASER_3D/SN_27000839/STATUS';
%position status
```



```
obj.doocs_adr='PITZ.LASER/LASER_3D/SN_27000839/;
obj.RBKadr=[doocs_adr,'VALUE.RDBK'];
obj.SETadr=[doocs_adr,'VALUE.SP'];
obj.ACTadr=[doocs_adr,'COMMAND'];
obj.STATUSadr=[doocs_adr,'STATUS']
```


DESY

Yak shaving

Video removed for pdf version. Find here
https://www.youtube.com/watch?v=AbSehcT19u0

Yak shaving

Often enough doing one task requires an unexpected chain of other tasks to be done first. Don’t get caught up in it while boy scouting.

Find ME

The OutputManager project can be found at

https://github.com/Kritzek/OutputManager

Questions to
christian.koschitzki@desy.de