*Mc*LAB: Compiler Tools for MATLAB

Amina Aslam
Toheed Aslam
Andrew Casey
Maxime Chevalier- Boisvert
Jesse Doherty
Anton Dubrau
Rahul Garg
Maja Frydrychowicz
Nurudeen Lameed
Jun Li
Soroush Radpour
Olivier Savary

Laurie Hendren
McGill University

Leverhulme Visiting Professor
Department of Computer Science
University of Oxford

Overview

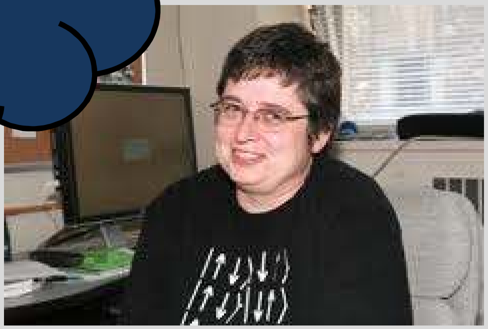
- Why MATLAB?
- Introduction to MATLAB – challenges
- Overview of the McLab tools
- Resolving names in MATLAB

Nature Article: “Why Scientific Computing does not compute” [Merali, Oct 2010]

- 38% of scientists spend at least 1/5th of their time programming.
- Codes often buggy, sometimes leading to papers being retracted. Self-taught programmers.
- Monster codes, poorly documented, poorly tested, and often used inappropriately.
- 45% say scientists spend more time programming than 5 years ago.

FORTRAN
C/C++

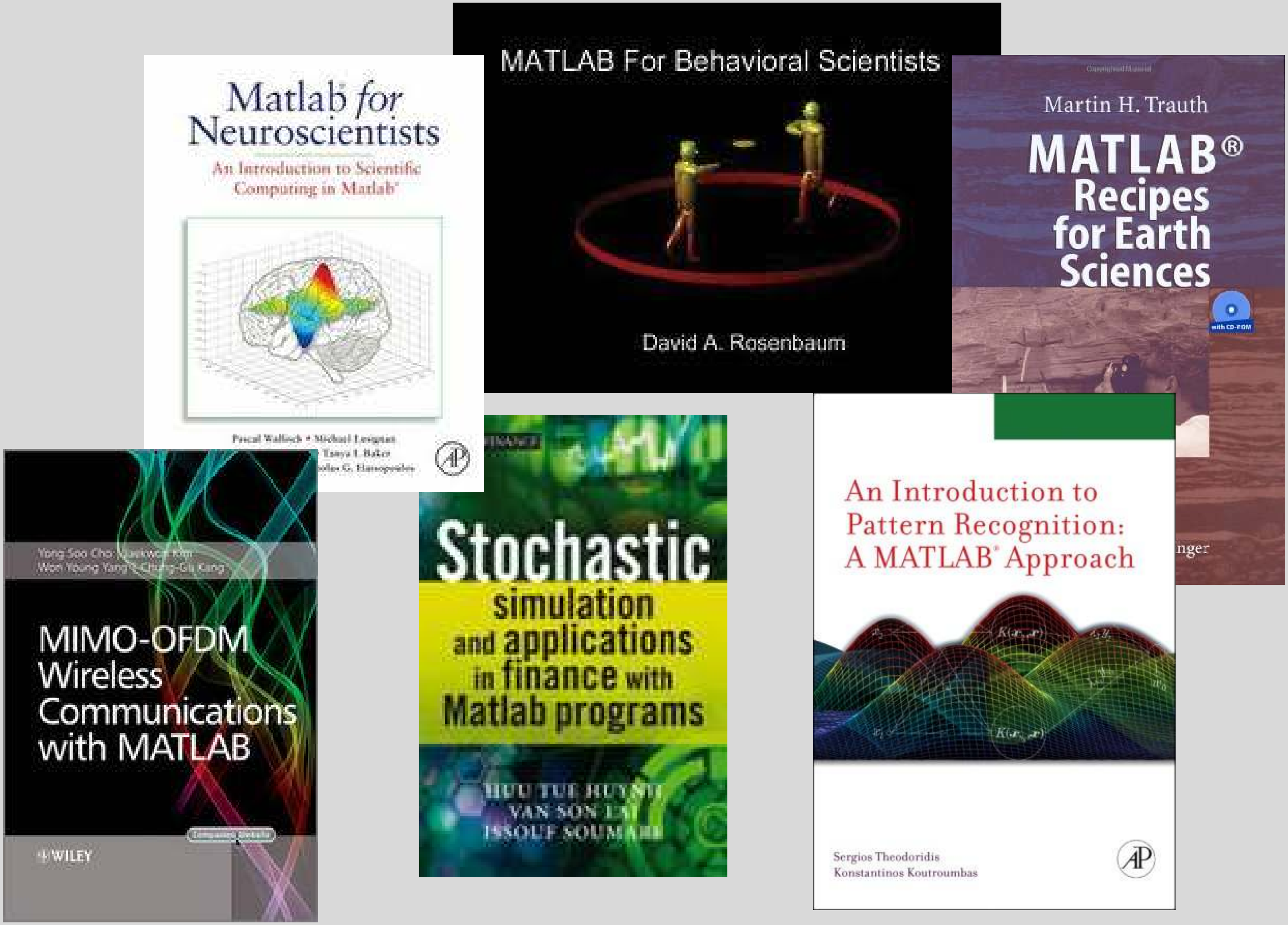
Java
AspectJ



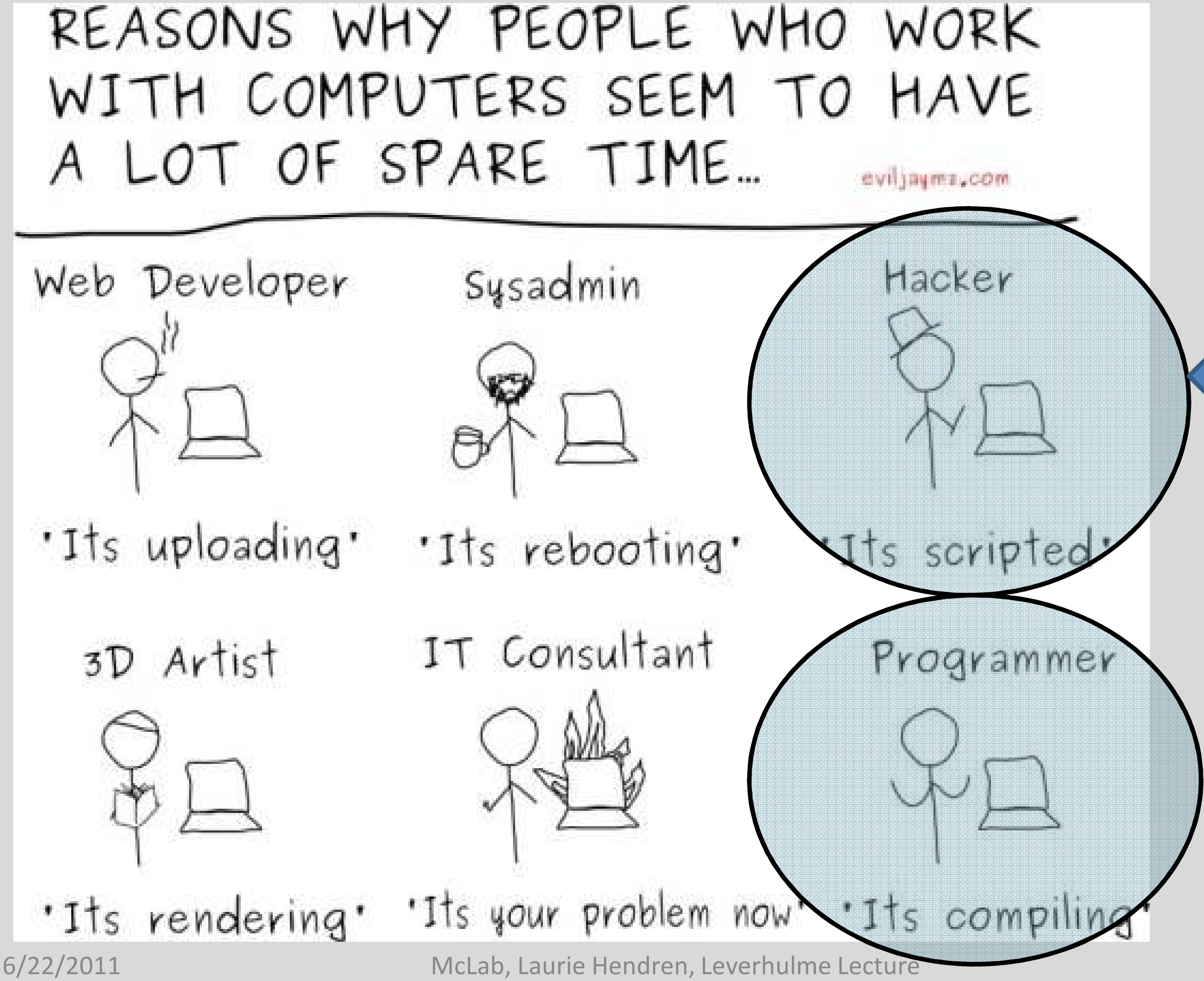
MATLAB
PERL
Python
Domain-specific

A lot of MATLAB programmers!

- Started as an interface to standard FORTRAN libraries for use by students.... but now
  - 1 million MATLAB programmers in 2004, number doubling every 1.5 to 2 years.
  - over 1200 MATLAB/Simulink books
  - used in many sciences and engineering disciplines
- Even more “unofficial” MATLAB programmers including those using free systems such as Octave or SciLab.


Matlab® for Neuroscientists
An Introduction to Scientific Computing in Matlab®
Pascal Wallisch • Michael Lusignan
Tanya I. Baker
olas G. Hatsopoulos
MATLAB For Behavioral Scientists
David A. Rosenbaum
Martin H. Trauth
MATLAB® Recipes for Earth Sciences
with CD-ROM
nger
Yong Soo Cho Jaekwon Kim
Won Young Yang Chung-Gu Kang
MIMO-OFDM Wireless Communications with MATLAB
Companion Website
WILEY
FINANCE
Stochastic simulation and applications in finance with Matlab programs
HUU TUE HUYNH
VAN SON LAI
ISSOUF SOUMARE
An Introduction to Pattern Recognition: A MATLAB® Approach
Sergios Theodoridis
Konstantinos Koutroumbas

Why do Scientists choose MATLAB?



MATLAB

FORTRAN

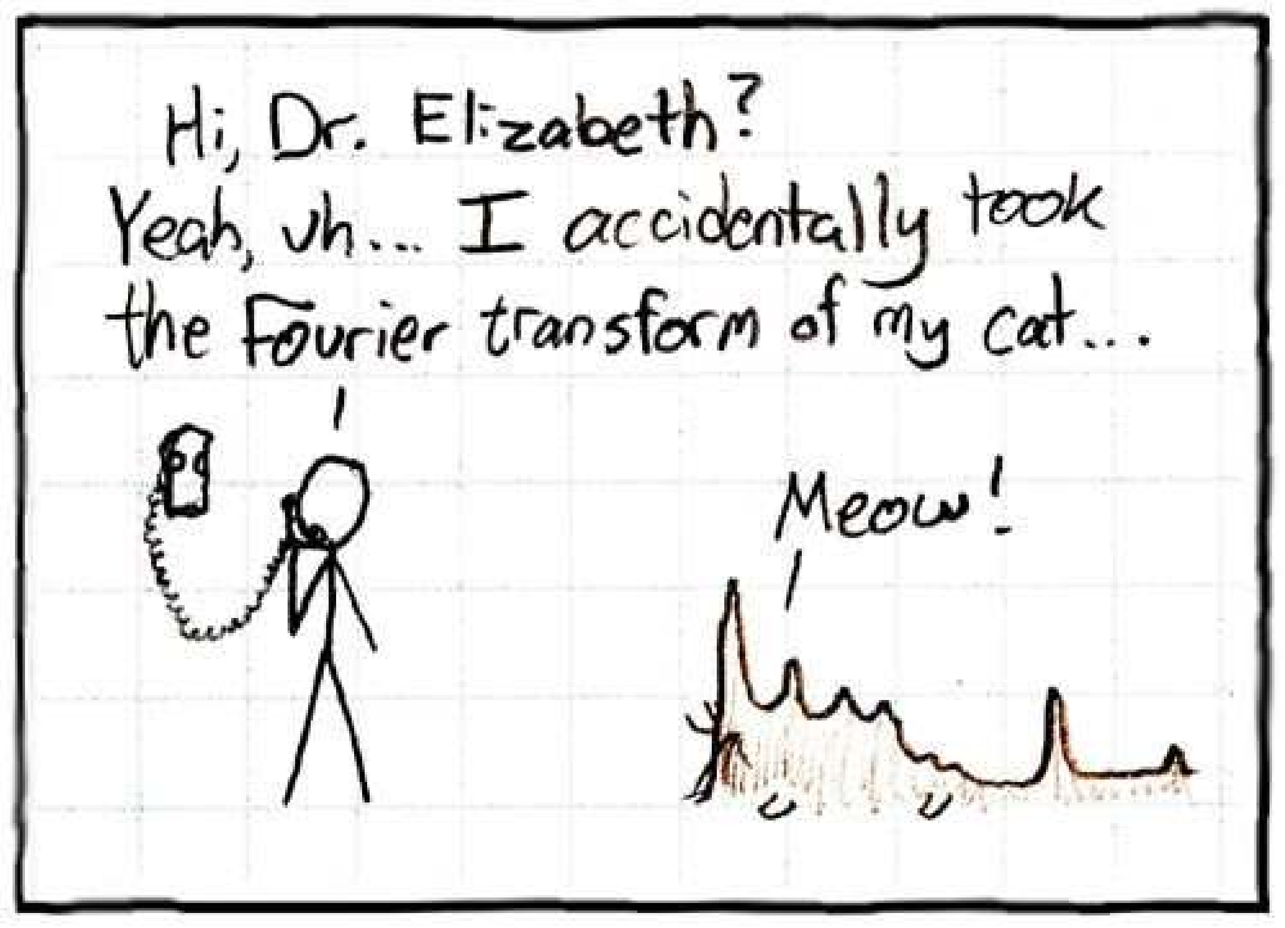
Implications of choosing a dynamic, “scripting” language like MATLAB….

Many run-time decisions …

Potentially large runtime overhead in both time and space

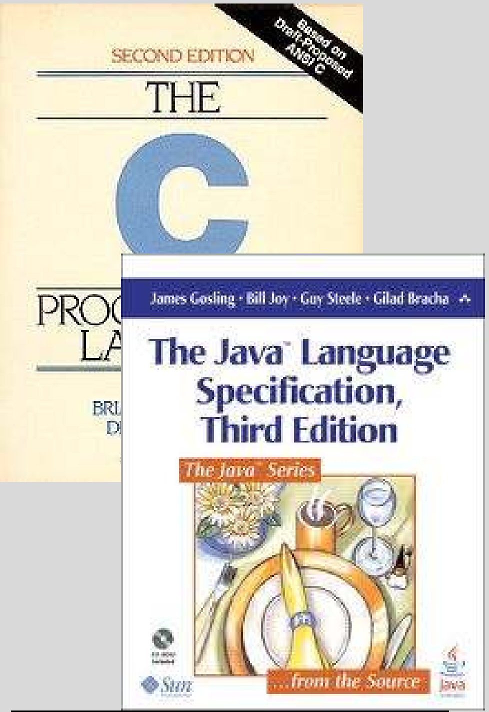
No types and “flexible” syntax

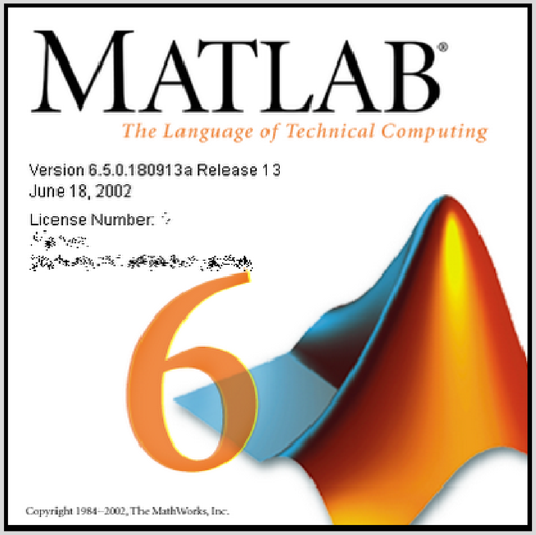
http://imgs.xkcd.com/comics/fourier.jpg


Hi, Dr. Elizabeth? Yeah, uh... I accidentally took the Fourier transform of my cat...
Meow!

Most semantic (syntactic) checks made at runtime … No static guarantees

No formal standards for MATLAB


SECOND EDITION
THE
C
PROG
LA
BRI
D
Based on Draft-Proposed ANSI C
James Gosling · Bill Joy · Guy Steele · Gilad Bracha
The Java Language Specification, Third Edition
The Java Series
...from the Source
Sun
Java

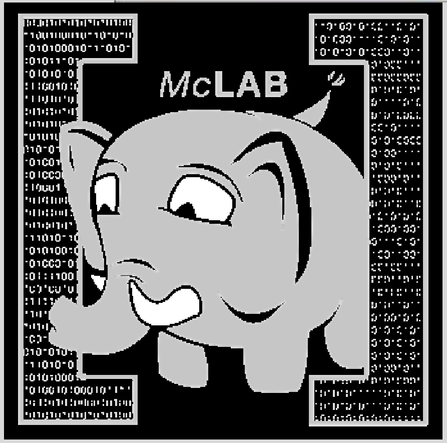


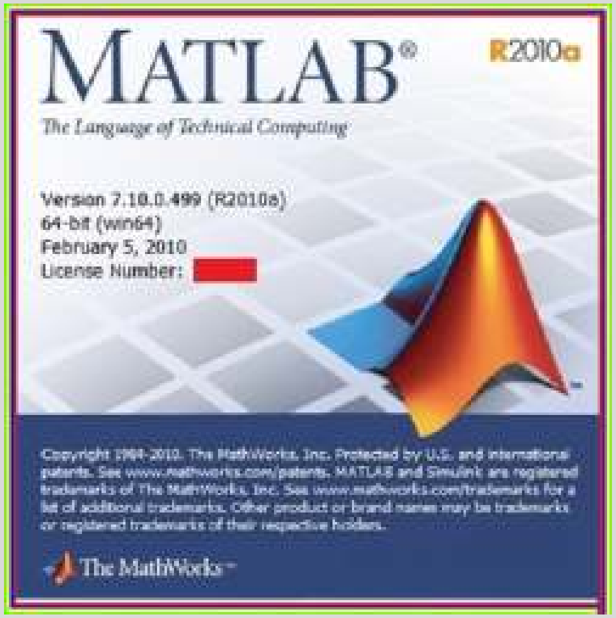
MATLAB®
The Language of Technical Computing

Version 6.5.0.180913a Release 13
June 18, 2002

License Number: [redacted]

Copyright 1984–2002, The MathWorks, Inc.





MATLAB®
R2010a
The Language of Technical Computing

Version 7.10.0.499 (R2010a)
64-bit (win64)
February 5, 2010
License Number: [redacted]

Copyright 1984-2010. The MathWorks, Inc. Protected by U.S. and international patents. See www.mathworks.com/patents. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

The MathWorks

Culture Gap

Scientists / Engineers

- Comfortable with informal descriptions and “how to” documentation.
- Don’t really care about types and scoping mechanisms, at least when developing small prototypes.
- Appreciate libraries, convenient syntax, simple tool support, and interactive development tools.

Programming Language / Compiler Researchers

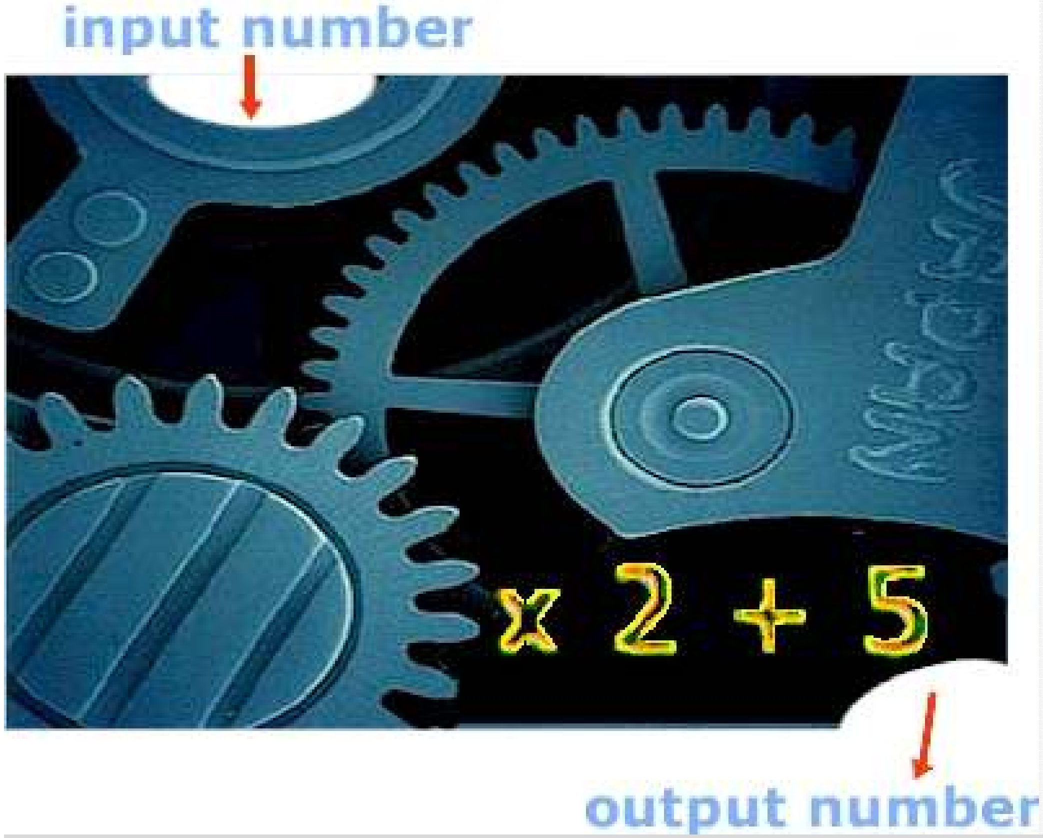
- Prefer more formal language specifications.
- Prefer well-defined types (even if dynamic) and well-defined scoping and modularization mechanisms.
- Appreciate “harder/deeper/more beautiful” programming language/compiler research problems.

Goals of the McLab Project

- Improve the understanding and documentation of the semantics of MATLAB.
- Provide front-end compiler tools suitable for MATLAB and language extensions of MATLAB.
- Provide a flow-analysis framework and a suite of analyses suitable for a wide range of compiler/soft. eng. applications.
- Provide back-ends that enable experimentation with JIT and ahead-of-time compilation.

Enable PL, Compiler and SE Researchers to work on MATLAB

Brief Introduction to MATLAB



Functions and Scripts in MATLAB

Basic Structure of a MATLAB function

```
function [ prod, sum ] = ProdSum( a, n )
  prod = 1;
  sum = 0;
  for i = 1:n
    prod = prod * a(i);
    sum = sum + a(i);
  end;
end
```

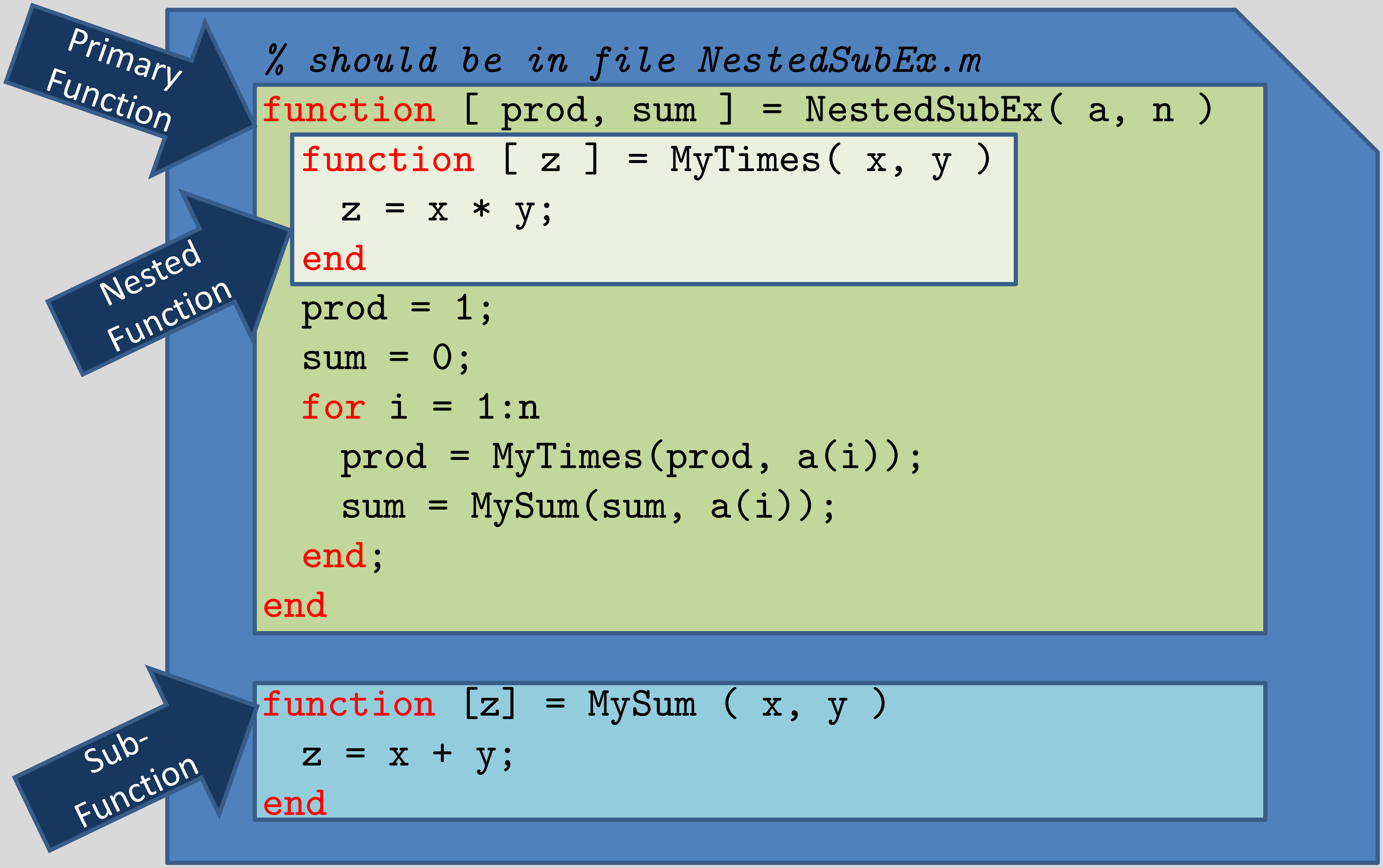
```
>> [a,b] = ProdSum([10,20,30],3)
a = 6000
b = 60

>>  ProdSum([10,20,30],2)
ans = 200

>> ProdSum(‘abc’,3)
ans =941094

>> ProdSum([97 98 99],3)
ans = 941084
```

Primary, nested and sub-functions



Primary Function

Nested Function

Sub-Function

```
% should be in file NestedSubEx.m
function [ prod, sum ] = NestedSubEx( a, n )
  function [ z ] = MyTimes( x, y )
    z = x * y;
  end
  prod = 1;
  sum = 0;
  for i = 1:n
    prod = MyTimes(prod, a(i));
    sum = MySum(sum, a(i));
  end;
end

function [z] = MySum ( x, y )
  z = x + y;
end
```

Basic Structure of a MATLAB script

```
% stored in file ProdSumScript.m
prod = 1;
sum = 0;
for i = 1:n
  prod = prod * a(i);
  sum = sum + a(i);
end;
```

```
>> clear
>> a = [10, 20, 30];
>> n = 3;
>> whos
  Name    Size    Bytes    Class
  a       1x3     24       double
  n       1x1     8        double
>> ProdSumScript()
>> whos
  Name    Size    Bytes    Class
  a       1x3     24       double
  i       1x1      8       double
  n       1x1      8       double
  prod    1x1      8       double
  sum     1x1      8       double
```

Directory Structure and Path

- Each directory can contain:
  - `.m` files (which can contain a script or functions)
  - a `private/` directory
  - a package directory of the form `+pkg/`
  - a type-specialized directory of the form `@int32/`

- At run-time:
  - current directory (implicit 1st element of path)
  - directory of last called function
  - path of directories
  - both the current directory and path can be changed at runtime (`cd` and `setpath` functions)

Function/Script Lookup Order (call in the body of a function f)

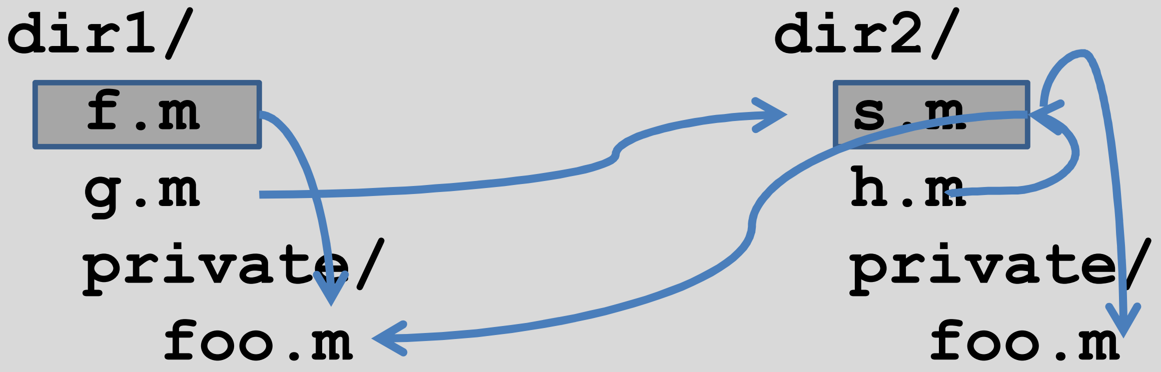
```
function f
  …
  foo(a);
  …
end
```

- Nested function (in scope of f)
- Sub-function (in same file as f)
- Function in /private sub-directory of directory containing f.
- 1st matching function, based on function name and type of first argument, looking in type-specialized directories, looking first in current directory and then along path.
- 1st matching function/script, based on function name only, looking first in current directory and then along path.

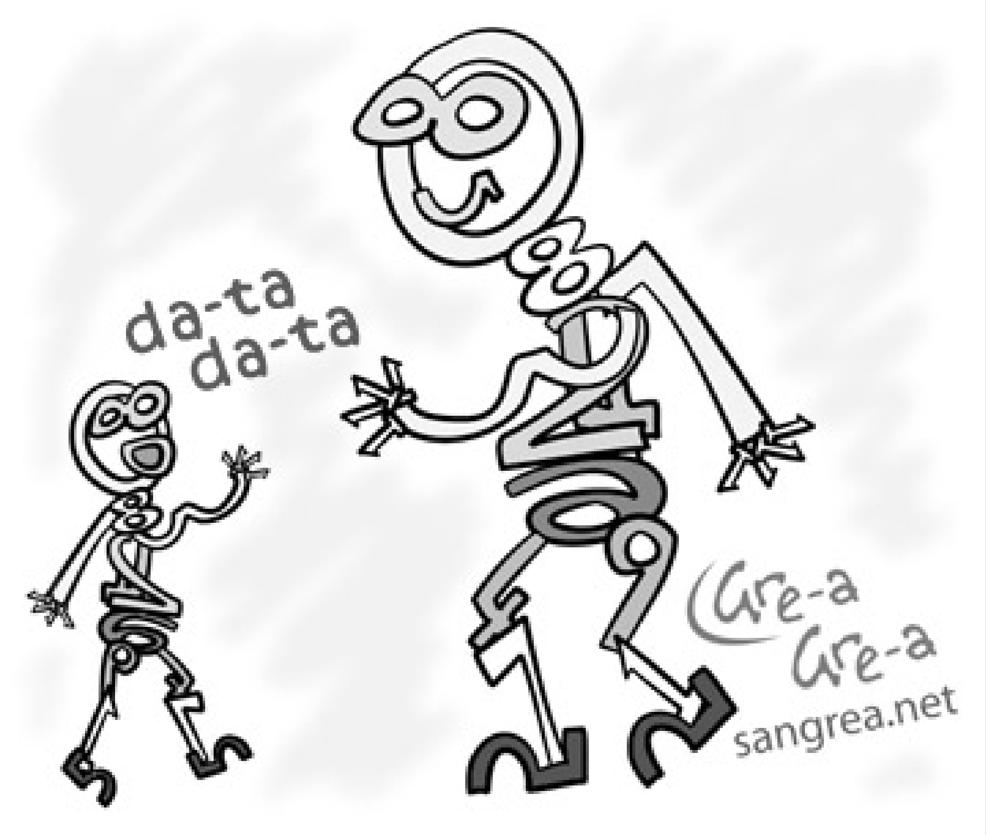
Function/Script Lookup Order (call in the body of a script s)

```
% in s.m
…
foo(a);
…
```

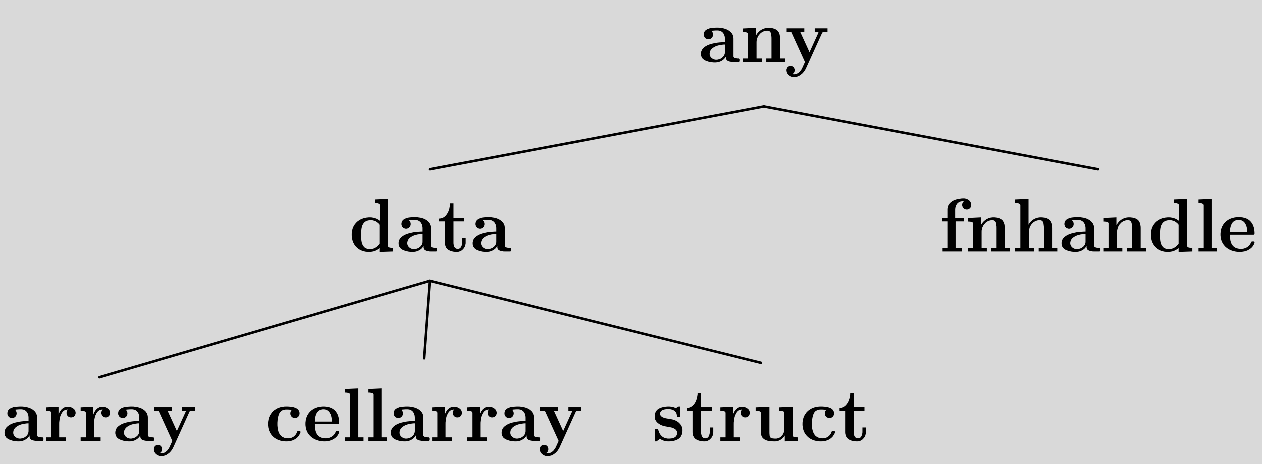
- Function in /private sub-directory of directory of last called function (not the /private sub-directory of the directory containing s).
- 1st matching function/script, based on function name, looking first in current directory and then along path.



Variables and Data in MATLAB



MATLAB types: high-level



Variables

- Variables are not explicitly declared.
- Local variables are allocated in the current workspace. Global and persistent variables in a special workspace.
- All input and output parameters are local.
- Local variables are allocated upon their first definition or via a load statement.
  - `x = ...`
  - `x(i) = ...`
  - `load (’f.mat’, ’x’)`
- Local variables can hold data with different types at different places in a function/script.

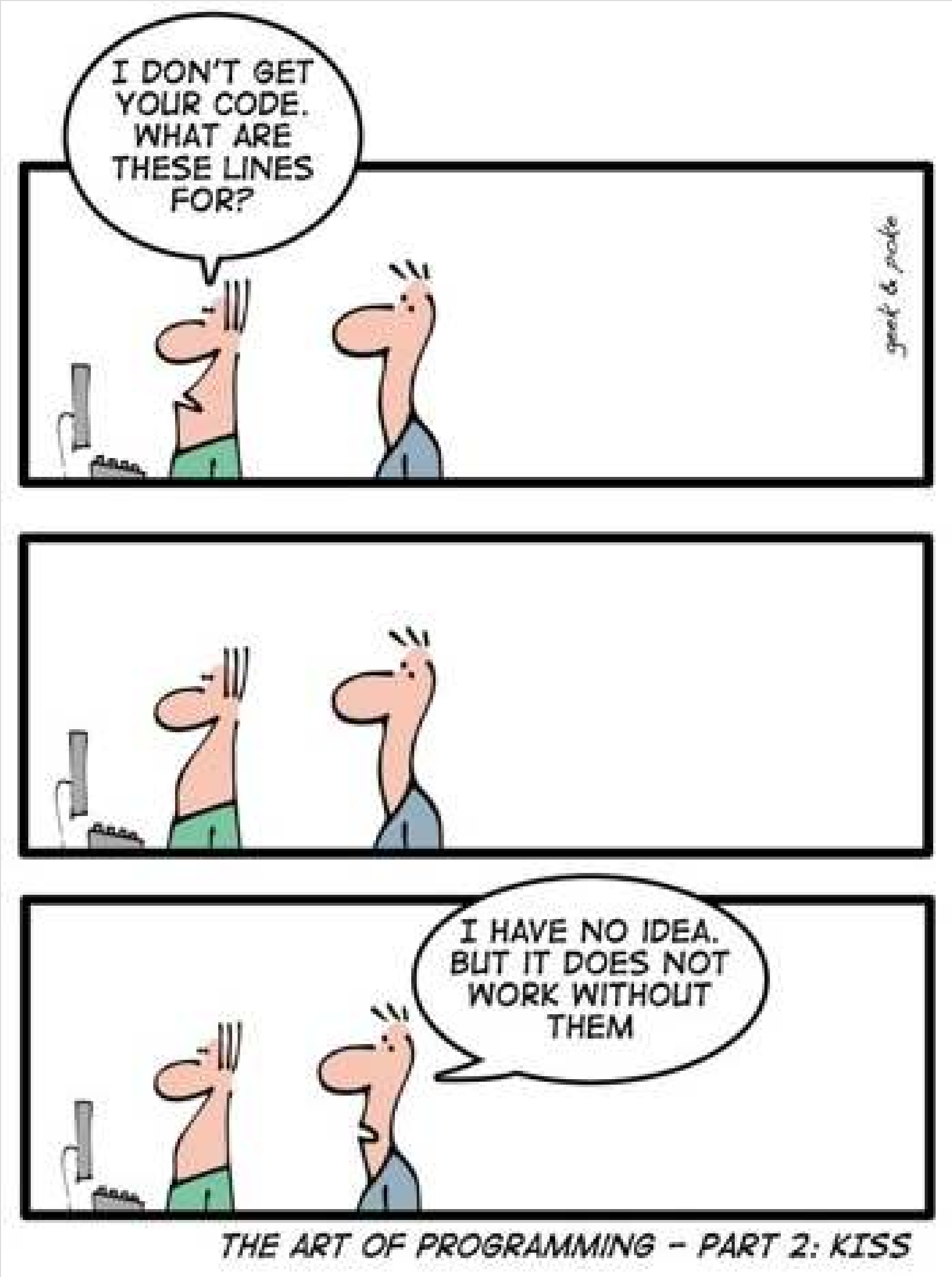
Variable Workspaces

- There is a workspace for global and persistent variables.
- There is a workspace associated with the read-eval-print loop.
- Each function call creates a new workspace (stack frame).
- A script uses the workspace of its caller (either a function workspace or the read-eval-print workspace).

Variable Lookup

- If the variable has been declared global or persistent in the function body, look it up in the global/persistent workspace.
- Otherwise, lookup in the current workspace (either the read-eval-print workspace or the top-most function call workspace).
- For nested functions, use the standard scoping mechanisms.

Other Tricky "features" in MATLAB


I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
geek & poke
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM
THE ART OF PROGRAMMING – PART 2: KISS

Irritating Front-end "Features"

- keyword **`end`** not always required at the end of a function (often missing in files with only one function).

- command syntax
  - **`length('x')`** or **`length x`**
  - **`cd('mydirname')`** or **`cd mydirname`**

- arrays can be defined with or without commas:
  [10, 20, 30] or [10 20 30]

- sometimes newlines have meaning:
  - **a = [10 20 30**
    40 50 60]; // defines a 2x3 matrix
  - **a = [10 20 30 40 50 60]; // defines a 1x6 matrix**
  - **a = [10 20 30;**
    40 50 60]; // defines a 2x3 matrix
  - **a = [10 20 30; 40 50 60]; // defines a 2x3 matrix**

“Evil” Dynamic Features

- not all input arguments required

```
function [ prod, sum ] = ProdSumNargs( a, n )
  if nargin == 1 n = 1; end;
  ...
end
```

- do not need to use all output arguments
- eval, evalin, assignin
- cd, addpath
- load

Evil Feature of the Day - Looking up an identifier

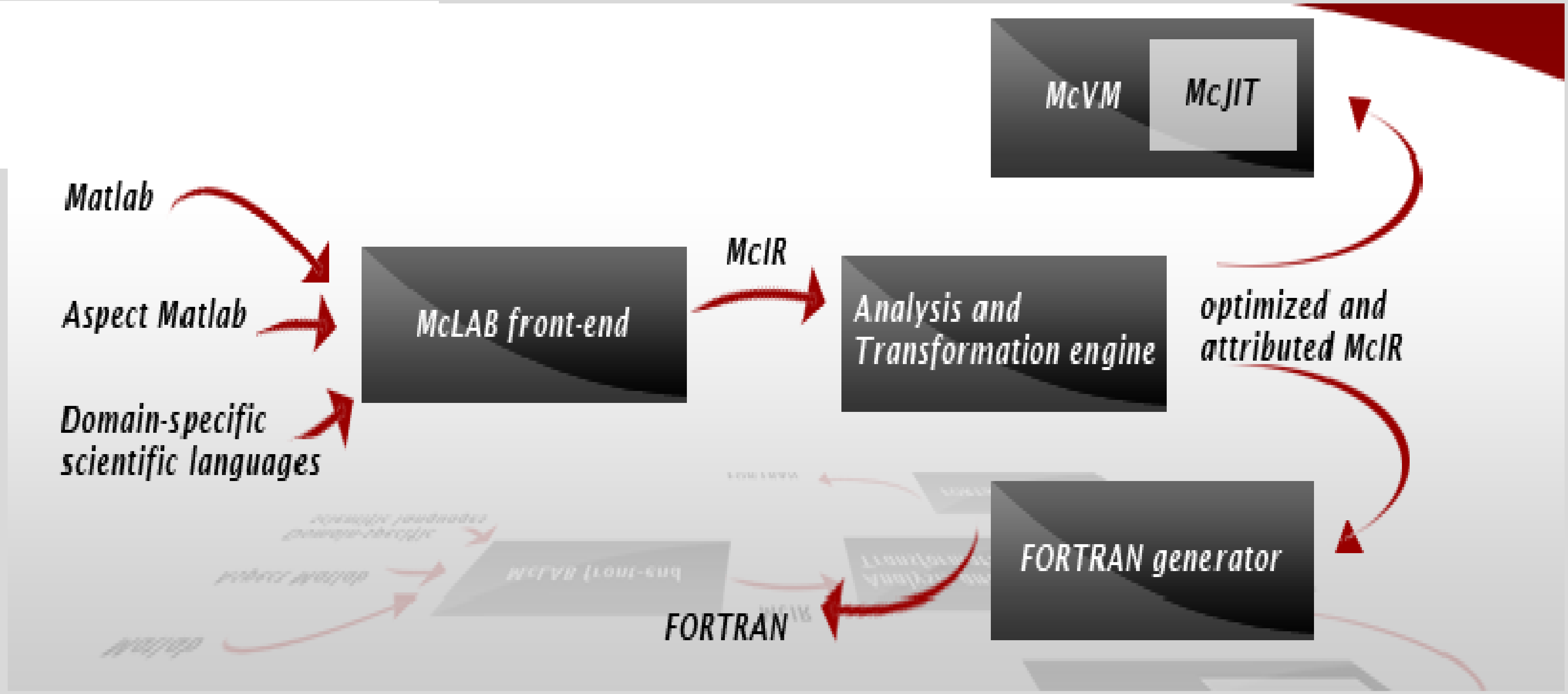
Old style general lookup - interpreter

- First lookup as a variable.
- If a variable not found, then look up as a function.

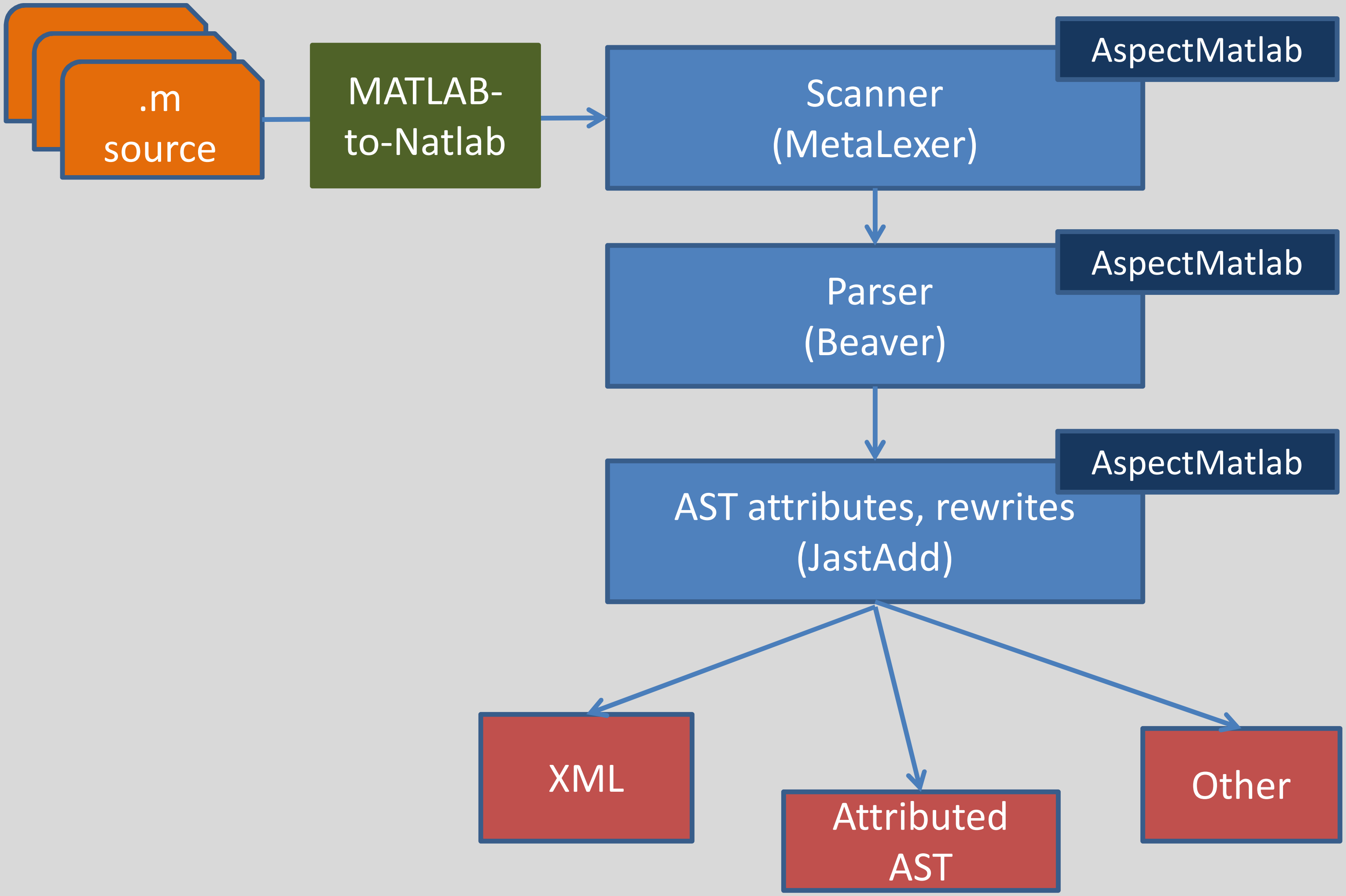
MATLAB 7 lookup - JIT

- When function/script first loaded, assign a "kind" to each identifier. VAR – only lookup as a variable, FN – only lookup as a function, ID – use the old style general lookup.
- How is the kind assignment done. What impact does it have on the semantics?

McLab – Overall Structure

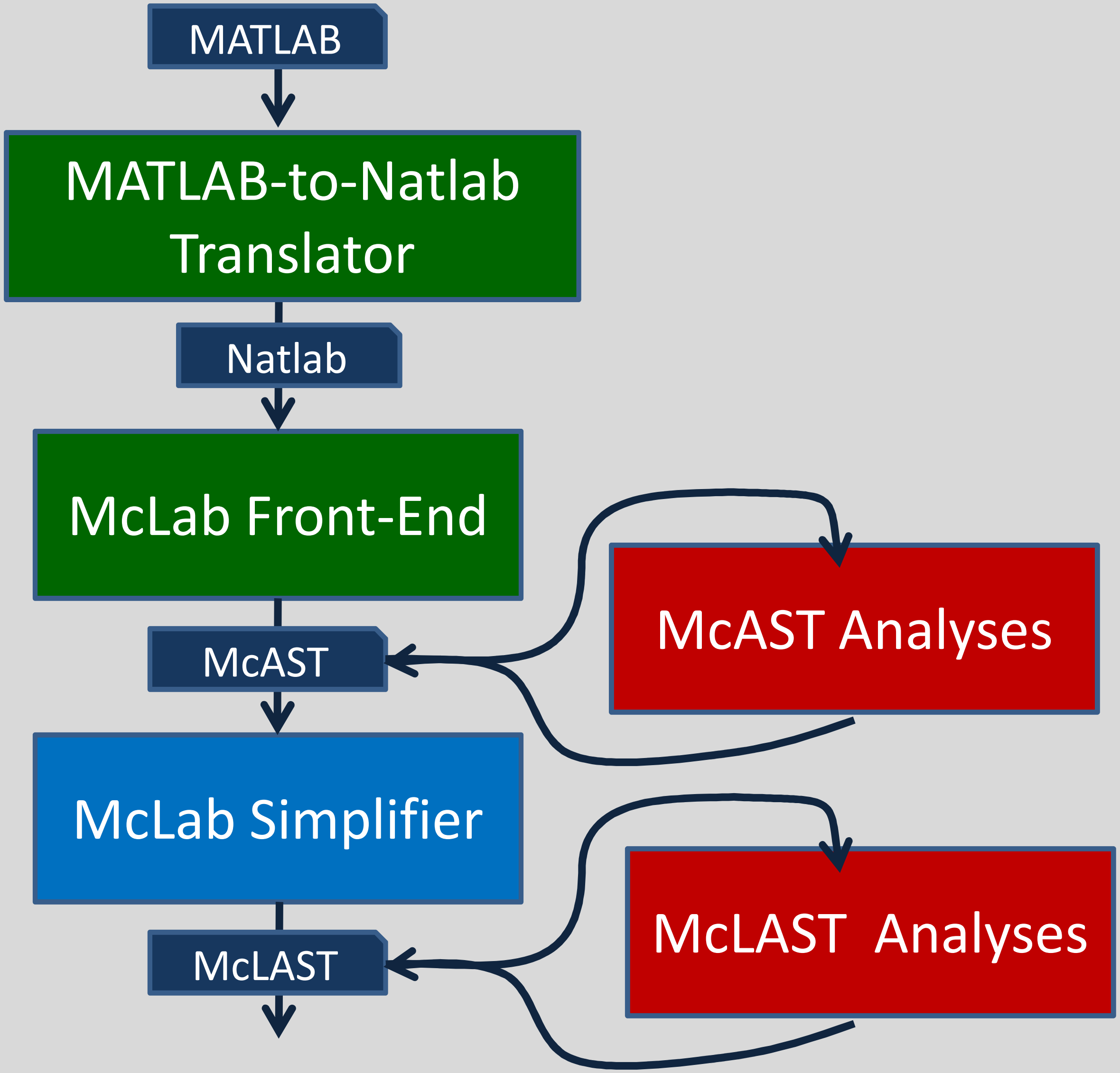

McVM
McJIT
Matlab
Aspect Matlab
Domain-specific scientific languages
McLAB front-end
McIR
Analysis and Transformation engine
optimized and attributed McIR
FORTRAN generator
FORTRAN

McLab Extensible Front-end



Analysis Engine

Analyses are written using an Analysis Framework that supports forward and backward flow analysis over McAST and McLAST.



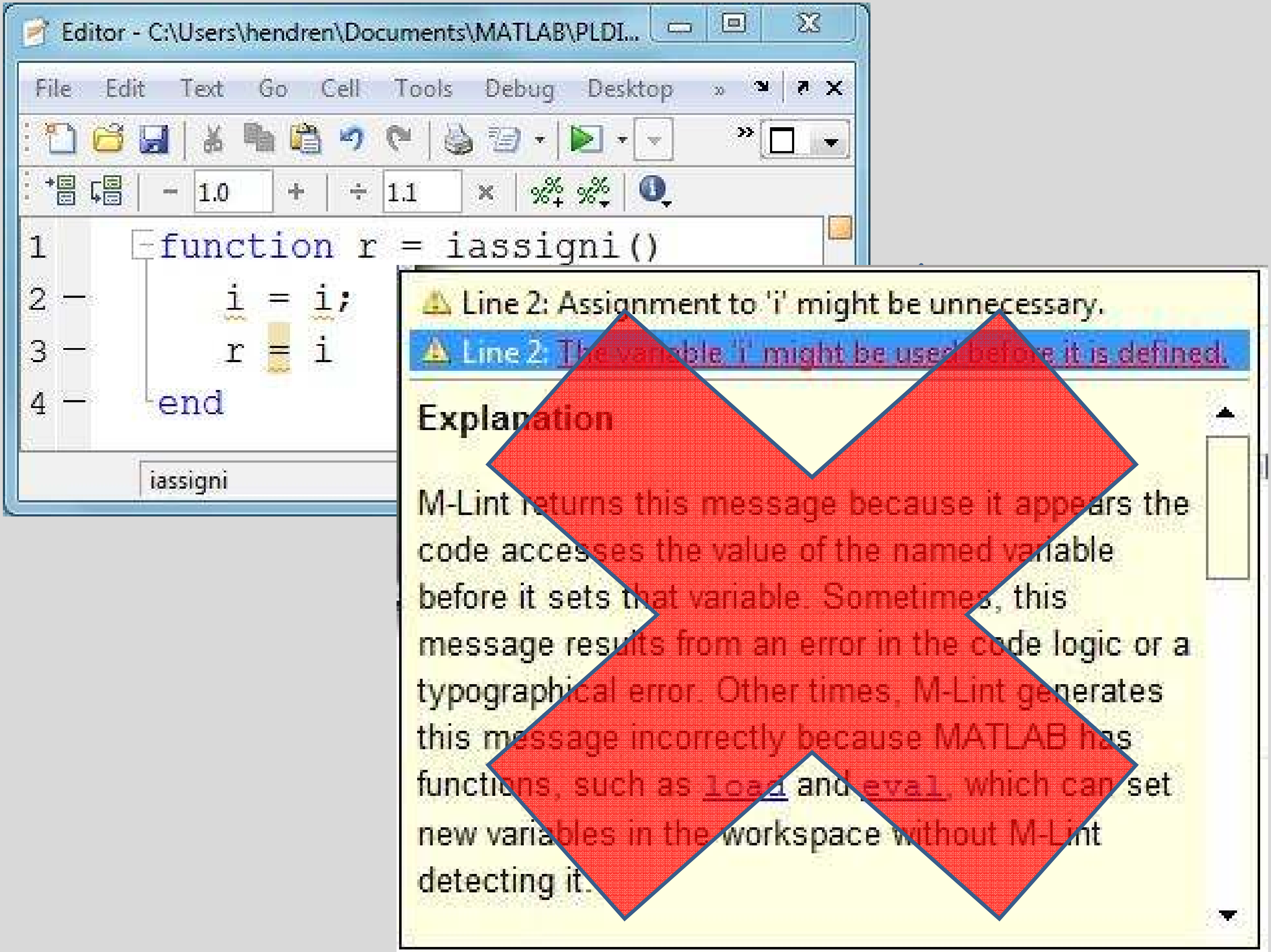
Back-ends, McVM and McFor

- **McVM**
  - A specializing virtual machine and JIT
  - Written in C++
  - Uses McLab front-end, LLVM JIT toolkit, Boehm gc, ATLAS, BLAS, LAPACK
  - Test-bed for dynamic techniques

- **McFor**
  - A MATLAB-to-FORTRAN 90 translator
  - Written in Java
  - 1st prototype showed excellent performance, but worked on smallish subset.
  - 2nd version under development
  - could potentially by used to generate code for different back-ends.

How does MATLAB resolve Names?

- No official specification
- Motivating example



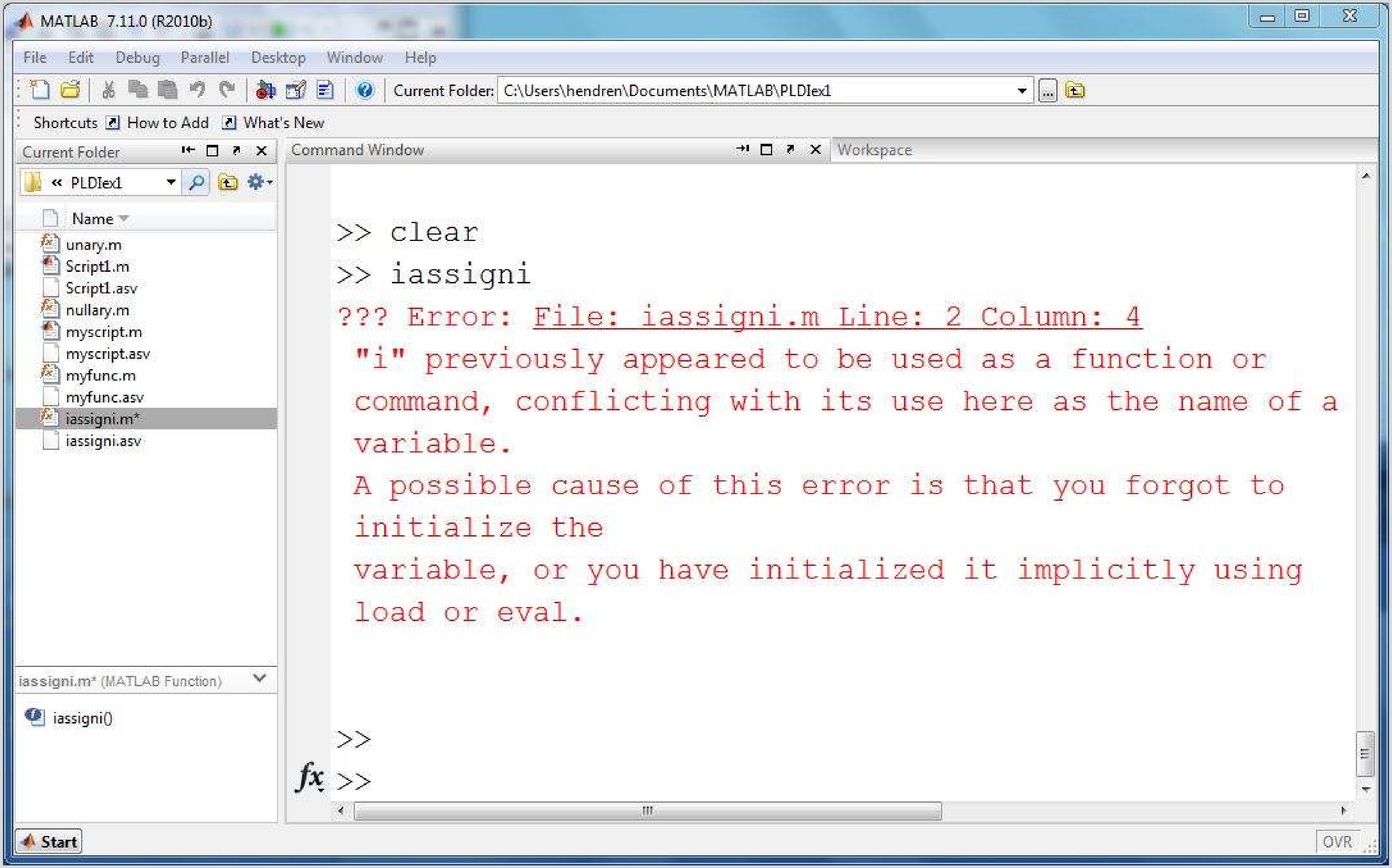
```
function r = iassigni()
    i = i;
    r = i
end
```

Line 2: Assignment to 'i' might be unnecessary.

Line 2: The variable 'i' might be used before it is defined.

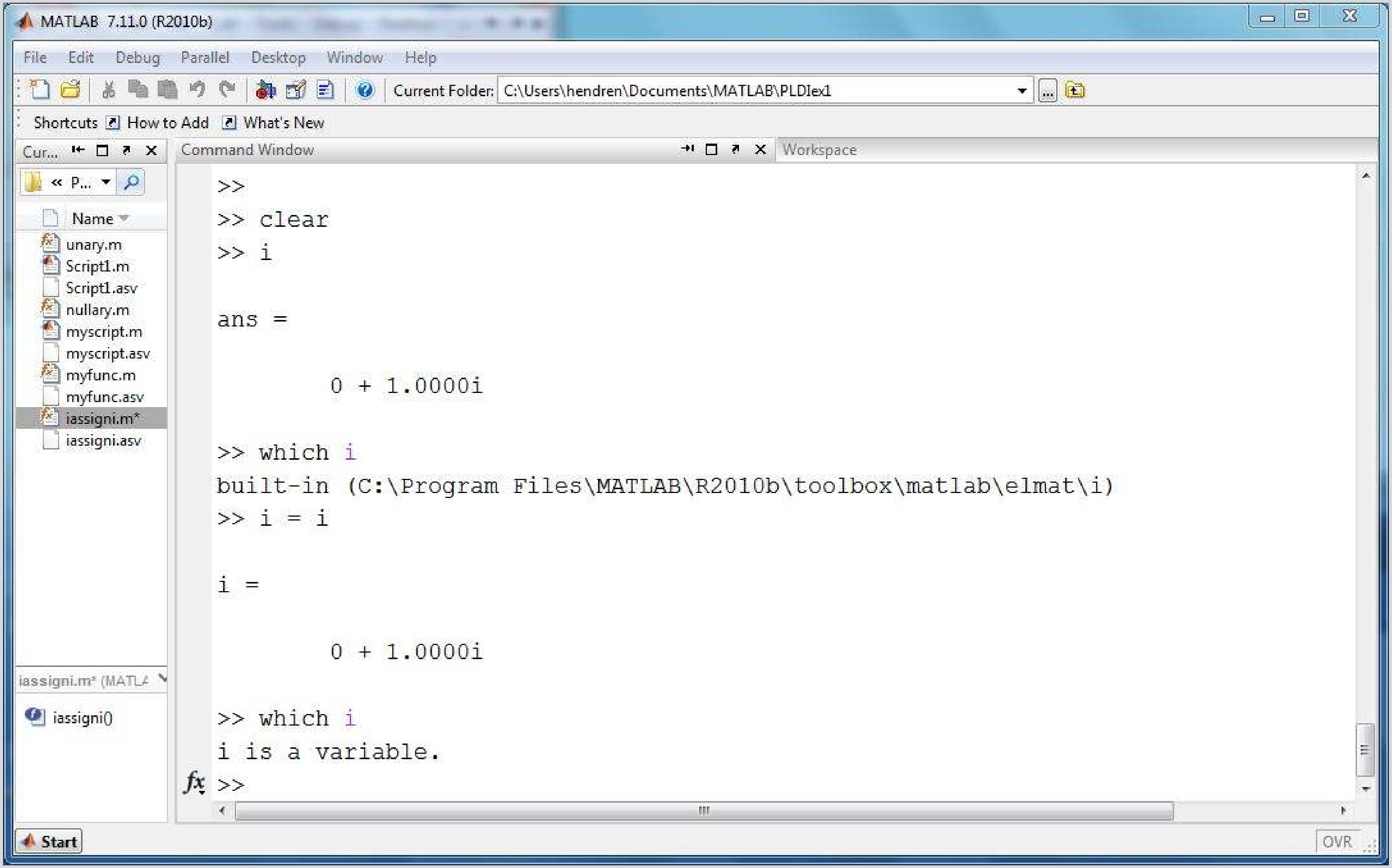
Explanation

M-Lint returns this message because it appears the code accesses the value of the named variable before it sets that variable. Sometimes, this message results from an error in the code logic or a typographical error. Other times, M-Lint generates this message incorrectly because MATLAB has functions, such as load and eval, which can set new variables in the workspace without M-Lint detecting it.



```
>> clear
>> iassigni
??? Error: File: iassigni.m Line: 2 Column: 4
 "i" previously appeared to be used as a function or
 command, conflicting with its use here as the name of a
 variable.
 A possible cause of this error is that you forgot to
 initialize the
 variable, or you have initialized it implicitly using
 load or eval.


>>
>>
```



Current Folder: C:\Users\hendren\Documents\MATLAB\PLDIex1

```
>>
>> clear
>> i

ans =

        0 + 1.0000i

>> which i
built-in (C:\Program Files\MATLAB\R2010b\toolbox\matlab\elmat\i)
>> i = i

i =

        0 + 1.0000i

>> which i
i is a variable.
>>
```

Read-Eval-Print Loop

Evil Feature of the Day - Recap

Old style general lookup - interpreter

- First lookup as a variable.
- If a variable not found, then look up as a function.

MATLAB 7 lookup - JIT

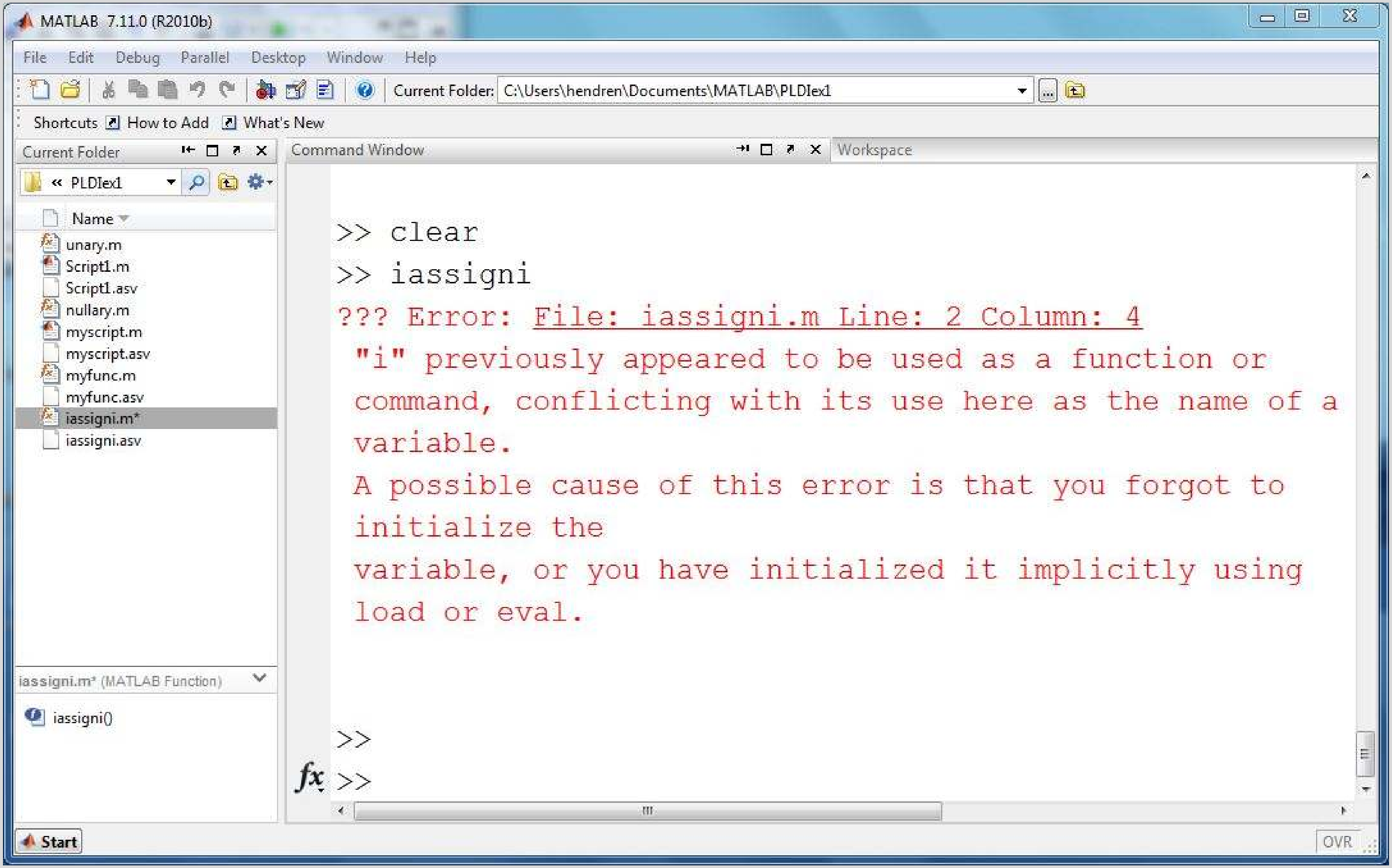
- When function/script first loaded, statically assign a "kind" to each identifier. VAR – only lookup as a variable, FN – only lookup as a function, ID – use the old style general lookup.
- Compile-time error if, within the body of a function or script, an identifier has kind VAR in one place and FN in another.

Does the kind analysis change the semantics?

Yes, in two ways!

1. New compile-time errors, so programs that would previously execute will not.
2. Different binding at run-time for some identifiers which are assigned a kind of VAR or FN.

Compile-time kind error



```
>> clear
>> iassigni
??? Error: File: iassigni.m Line: 2 Column: 4
 "i" previously appeared to be used as a function or
 command, conflicting with its use here as the name of a
 variable.
 A possible cause of this error is that you forgot to
 initialize the
 variable, or you have initialized it implicitly using
 load or eval.


>>
>>
```

Different lookup with old vs MATLAB 7 semantics

```
1 function [ r ] = KindEx( a )
2   x = a + sum(j);
3   eval('sum = ones(10);');
4   r = sum(x);
5 end
```

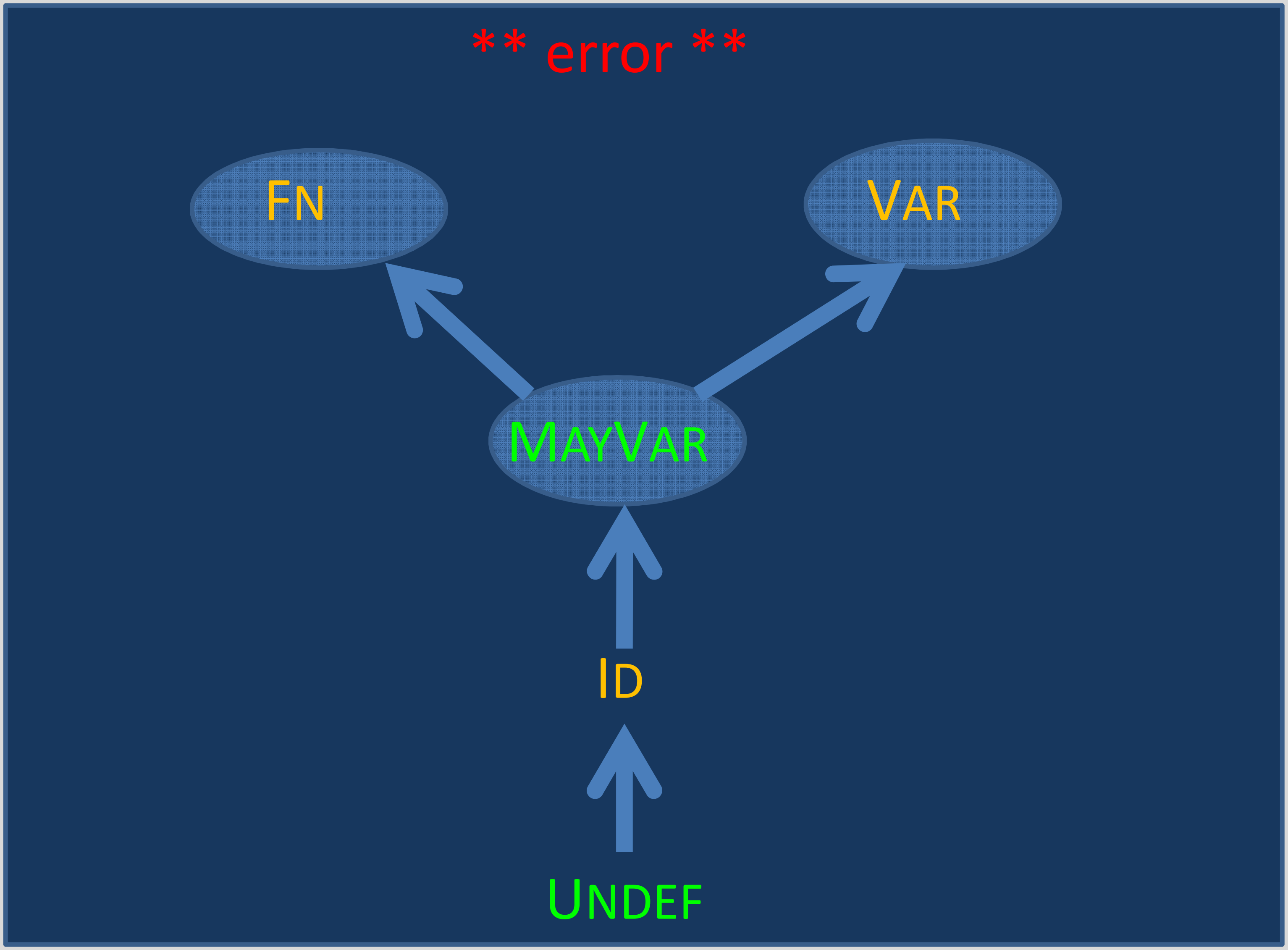
- Old interpreter semantics:
  - sum, line 2, named function
  - sum, line 4, local variable

- MATLAB 7 semantics gives a static kind of FN to sum
  - sum, line 2, named function
  - sum, line 4, named function

Our approach to the Kind Analysis Problem

- Identify that a kind analysis is needed to match MATLAB 7 semantics.
- Specify and implement a kind assignment algorithm that matches the observed behaviour of MATLAB 7. (both for functions and for scripts)
- Identify any weaknesses in the MATLAB 7 approach and suggest two more clearly defined alternatives, one flow-sensitive and one flow-insensitive.
- Determine if the alternatives could be used without significant change to the behaviour of existing MATLAB programs.

Kind Abstraction



Kind Analysis

1. Collect all identifiers used in function/script and set initial kind approximations for each identifier.
2. Traverse AST applying analysis rules to identifiers.
3. Traverse AST making final kind assignment.

Steps 1 and 3 are different for scripts and functions, step 2 uses the same rules.

Step 2: Kind Analysis Rules

Definition of identifier *x*:

$$kind[x] \leftarrow kind[x] \bowtie \textsc{Var}$$

Use of identifier *x*:

```
if ((kind[x] ∈ {Id, Undef})&exists_lib(x,lib))
    kind[x] ← Fn
else
    kind[x] ← kind[x] ⋈ Id
```

$$\text{if } ((kind[x] \in \{\textsc{Id}, \textsc{Undef}\})\&\text{exists\_lib(x,lib)})$$

$$\quad kind[x] \leftarrow \textsc{Fn}$$

$$\text{else}$$

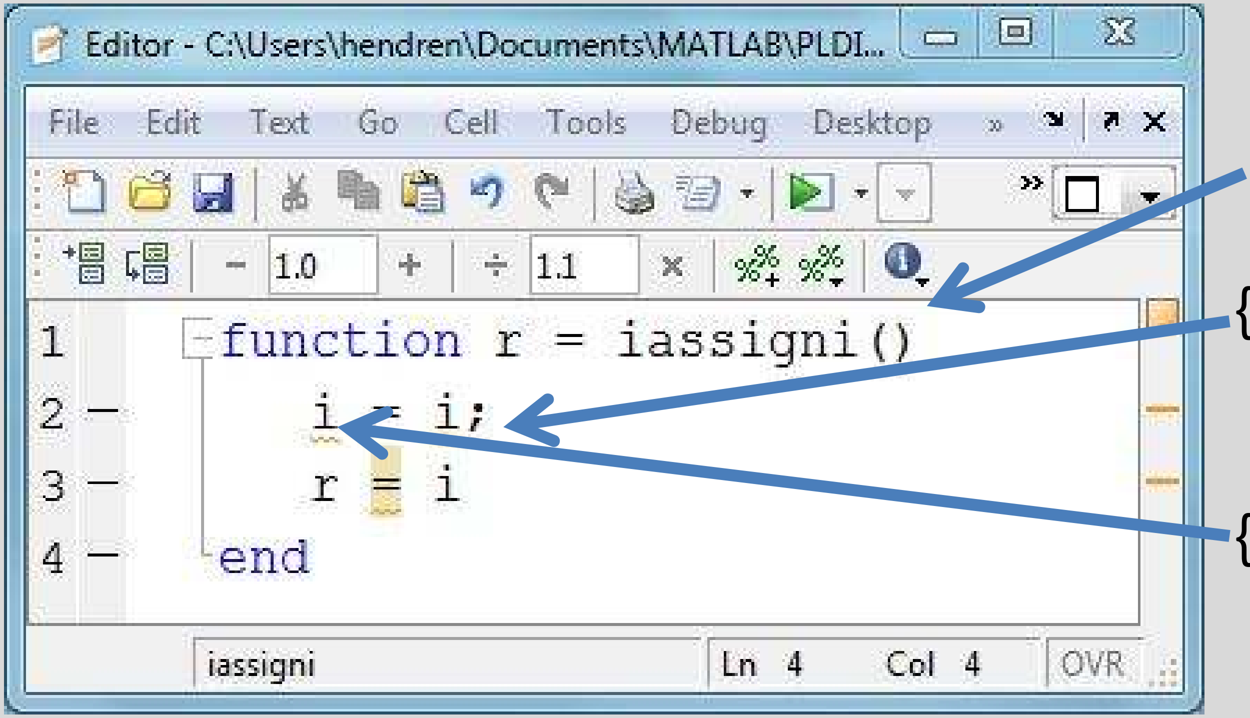
$$\quad kind[x] \leftarrow kind[x] \bowtie \textsc{Id}$$

Kind Analysis for Functions

- Initial values: input and output parameters are initialized to VAR, all other identifiers are initialized as UNDEF.

- Final values:

```
for each id occurrence in f do
   if fkind[id] in {ID, MAYVAR}
      id.kind = ID
   else    /* fkind[id] in {VAR, FN} */
      id.kind = fkind[id]
```



```
function r = iassigni()
    i = i;
    r = i
end
```

{(r,VAR),(i,UNDEF)}

{(r,VAR),(i,FN)}

{(r,VAR),(i,**error**)}

```
if (($kind[x] \in \{\textsc{Id}, \textsc{Undef}\}$)&exists_lib(x,lib))
    $kind[x] \leftarrow \textsc{Fn}$
else
    $kind[x] \leftarrow kind[x] \bowtie \textsc{Id}$
```

READ RULE

$$\boxed{kind[x] \leftarrow kind[x] \bowtie \textsc{Var}}$$

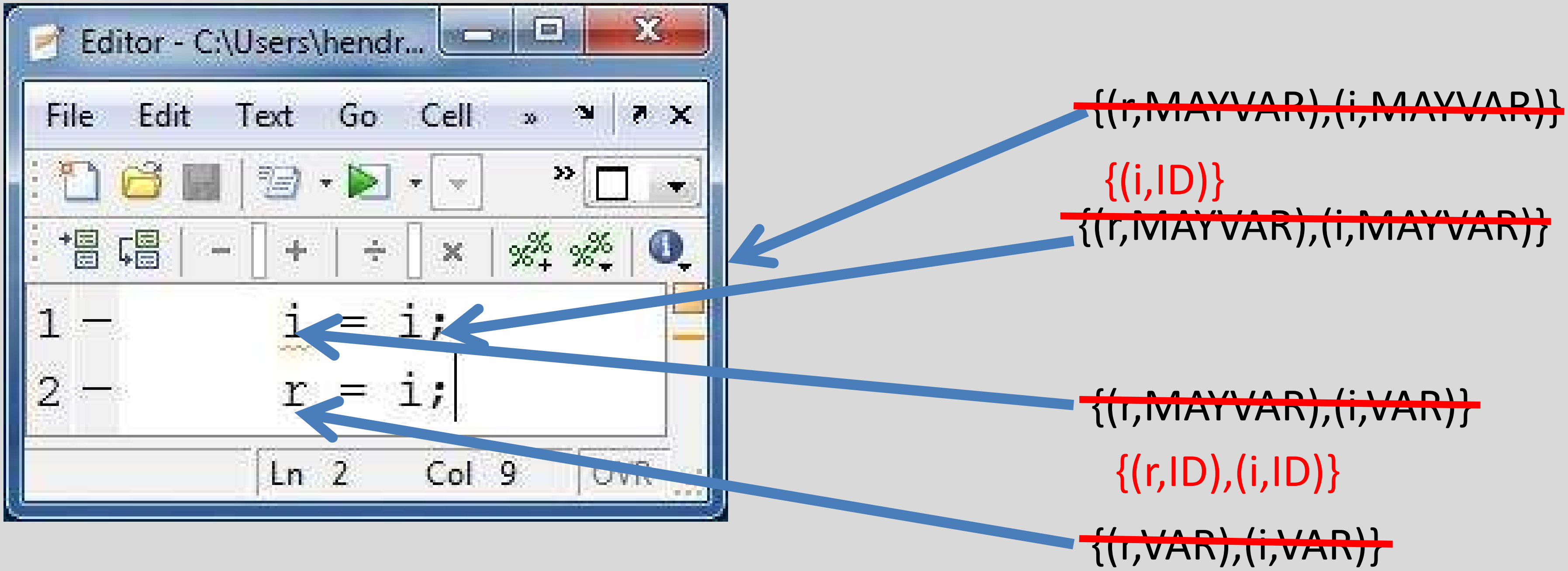
WRITE RULE

Kind Analysis for Scripts

- Initial values: all identifiers are initialized to MAYVAR
- Final values:

```
for each id occurrence in s do
   if id.kind in {VAR, MAYVAR}
     id.kind = ID
   else /* id.kind must be FN, it can't be ID or UNDEF*/
     id.kind = FN
```

- Note: most identifiers will be mapped to ID



```
i = i;
r = i;
```

~~{(r,MAYVAR),(i,MAYVAR)}~~

{(i,ID)}

~~{(r,MAYVAR),(i,MAYVAR)}~~

~~{(r,MAYVAR),(i,VAR)}~~

{(r,ID),(i,ID)}

~~{(r,VAR),(i,VAR)}~~

```
for each id occurrence in s do
  if id.kind in {VAR, MAYVAR}
    id.kind = ID
  else /* id.kind must be FN, it can't be ID or UNDEF*/
    id.kind = FN
```

```
if (($kind[x] \in \{\textsc{Id}, \textsc{Undef}\}$)&exists_lib(x,lib))
  $kind[x] \leftarrow \textsc{Fn}$
else
  $kind[x] \leftarrow kind[x] \bowtie \textsc{Id}$
```

READ RULE

$$\boxed{kind[x] \leftarrow kind[x] \bowtie \textsc{Var}}$$

WRITE RULE

Problems with MATLAB 7 kind analysis

- apparently not clearly documented, in some ways just a side-effect of a JIT implementation decision
- without a clear specification, confusing for the programmer and compiler/tool developer
- loses almost all information about variables in scripts
- some strange anomalies due to a "traversal-sensitive" analysis

Examples of Anomalies

```
if ( exp )
  ... = sum(10);     (sum,FN)
else
   sum(10) = ...;    *error*
```

```
if ( ~exp )
    sum(10) = ... ;    (sum,VAR)
else
    ... = sum(10);     (sum,VAR)
```

```
size(size(10)) = ...
(size,VAR)
        (size, VAR)
```

```
t = size(10);   (size,FN)
size(t) = ...      *error*
```

Flow-sensitive Analysis

```
if ( exp )
  ... = sum(10);     (sum,FN)
else
   sum(10) = ...;    (sum, VAR)
// merge,  *error*
```

```
size(size(10)) =
        (size,FN)
 *error*
```

- Apply a flow-sensitive analysis that merges at control-flow points.
- Consider explicit loads to be definitions -

```
load ('f.mat', 'x')
```

- Map final kinds for scripts using the same algorithm as for functions.

Flow-insensitive Analysis

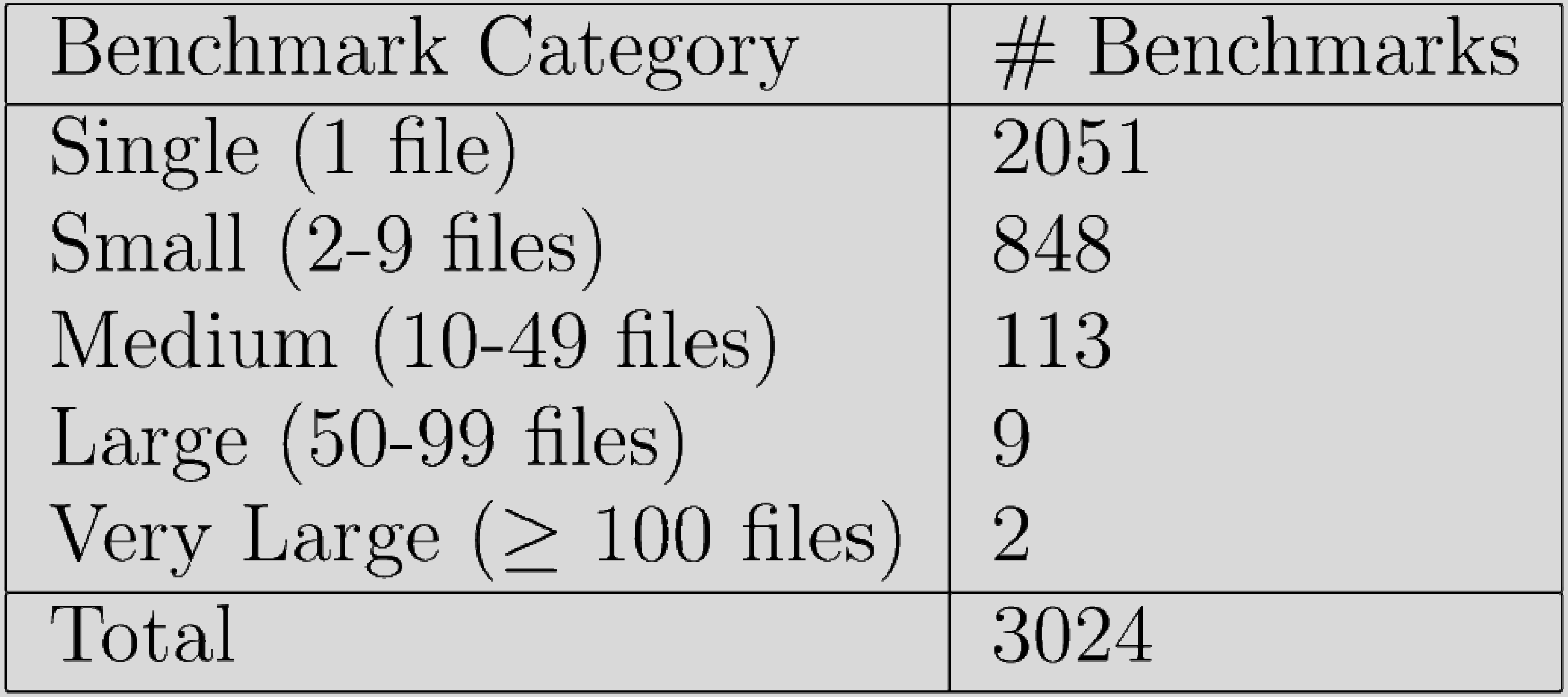
if (exp)
  … = sum(10);
else
  sum(10) = …;
(sum,VAR)
	size(size(10)) =
	    (size,VAR)

1. Assign VAR to identifiers that are defined on lhs, or declared global or persistent.
2. Assign FN to identifiers which have a handle taken or used in command syntax.
3. Assign FN to identifiers that have no assignment yet, and which are found in the library.

error if assigned both FN and VAR

Results: What is the distribution of kinds for functions/scripts in real MATLAB programs?

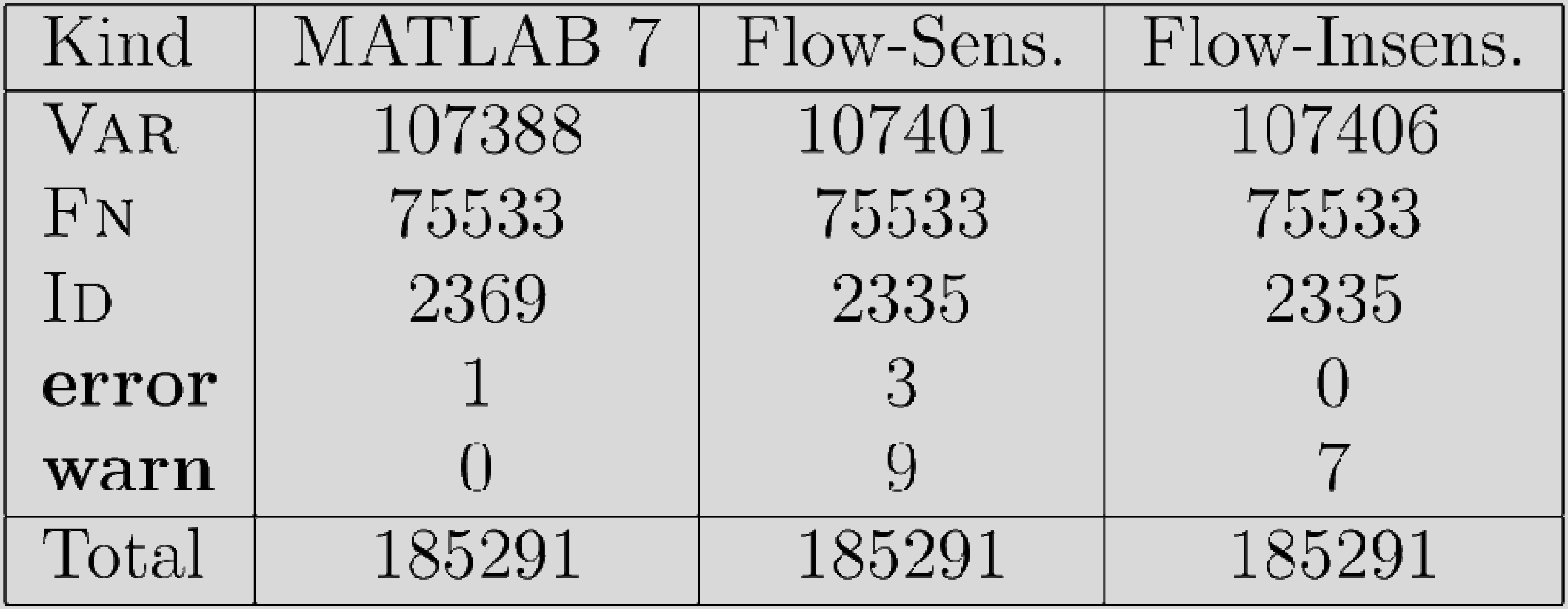
Various-sized benchmarks from a wide variety of application areas



Benchmark Category	# Benchmarks
Single (1 file)	2051
Small (2-9 files)	848
Medium (10-49 files)	113
Large (50-99 files)	9
Very Large ($\geq$ 100 files)	2
Total	3024

Send benchmarks or links to hendren@cs.mcgill.ca

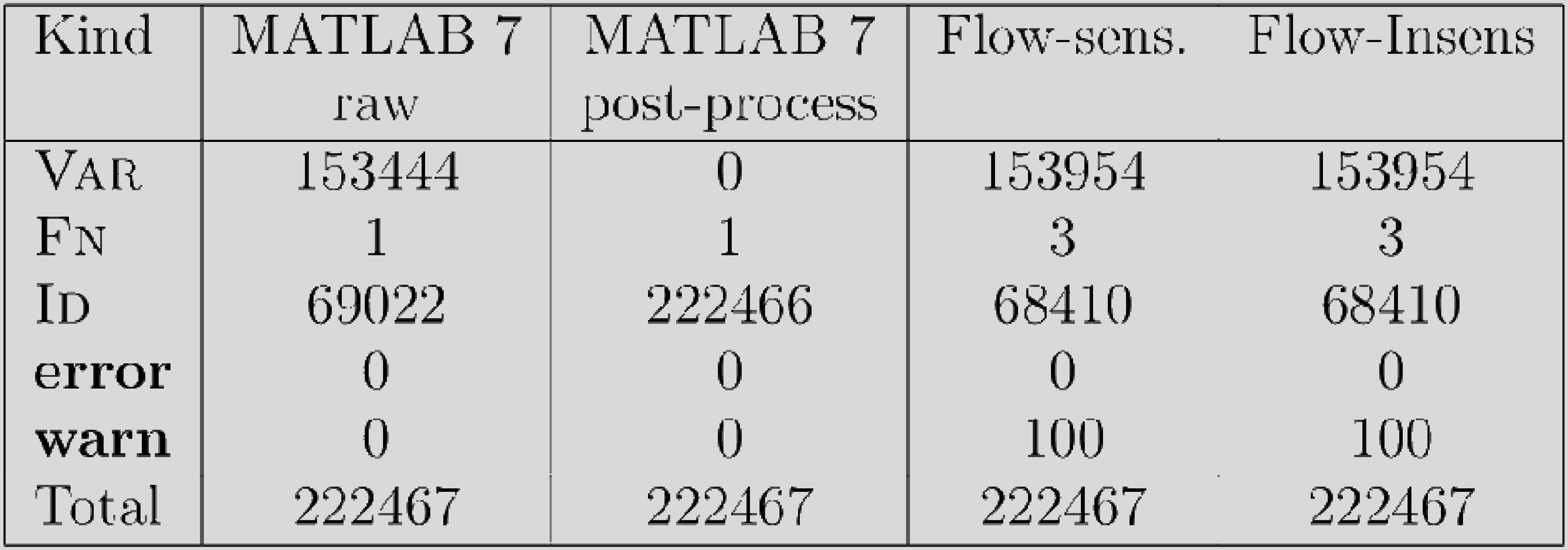
Results for Functions - number of identifiers with each Kind



Kind	MATLAB 7	Flow-Sens.	Flow-Insens.
VAR	107388	107401	107406
FN	75533	75533	75533
ID	2369	2335	2335
error	1	3	0
warn	0	9	7
Total	185291	185291	185291

11698 functions

Results for Scripts – number of identifier instances with each Kind



Kind	MATLAB 7 raw	MATLAB 7 post-process	Flow-sens.	Flow-Insens
VAR	153444	0	153954	153954
FN	1	1	3	3
ID	69022	222466	68410	68410
error	0	0	0	0
warn	0	0	100	100
Total	222467	222467	222467	222467

2035 scripts

Conclusions and Ongoing Work

- McLab is a toolkit to enable PL, compiler and SE research on MATLAB (close the gap).
- Release of three main tools: front-end/analysis framework, McVM (Virtual Machine) and McFor (MATLAB to FORTRAN) (tbd). PLDI 2011 tutorial.
- High-level: Refactoring tools for MATLAB. How to help programmers convert their programs to better structured, and more efficient codes?
- Lower-level: static compilation to Fortran90 and new dynamic techniques in McVM/McJIT.
- http://www.sable.mcgill.ca/mclab