Compiling MATLAB

Arun Chauhan
Indiana University

Collaborators

Ohio-State / OSC

Muthu Baskaran

Dave Hudak

Ashok Krishnamurthy

Rajkiran Panuganti

P. Sadayappan

Indiana University

Peter Gotschling

Andrew Lumsdaine

Pooja Malpani

Daniel McFarlin

Youngsang Shin

Rice University

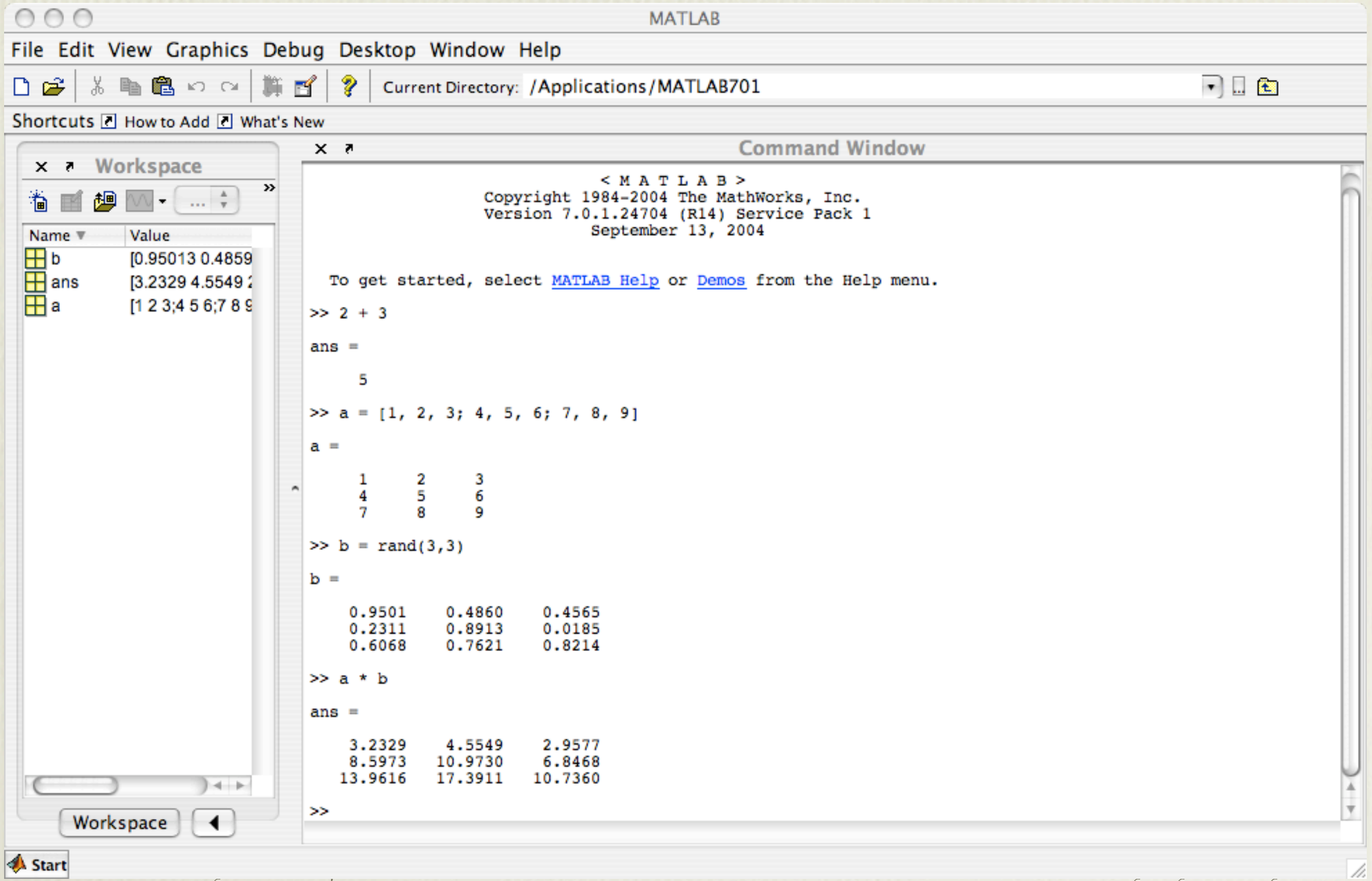
Jason Eckhardt

Ken Kennedy

Cheryl McCosh

Overview

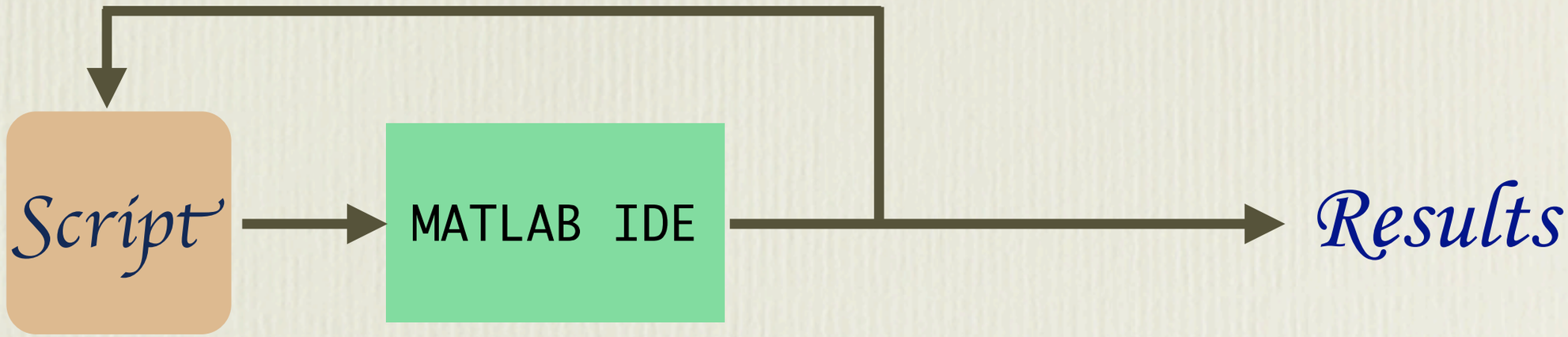
MATLAB IDE


MATLAB
File Edit View Graphics Debug Desktop Window Help
Current Directory: /Applications/MATLAB701
Shortcuts How to Add What's New
Workspace
Name Value
b [0.95013 0.4859
ans [3.2329 4.5549 2
a [1 2 3;4 5 6;7 8 9
Workspace
Command Window
< M A T L A B >
Copyright 1984-2004 The MathWorks, Inc.
Version 7.0.1.24704 (R14) Service Pack 1
September 13, 2004
To get started, select MATLAB Help or Demos from the Help menu.
>> 2 + 3
ans =
5
>> a = [1, 2, 3; 4, 5, 6; 7, 8, 9]
a =
1 2 3
4 5 6
7 8 9
>> b = rand(3,3)
b =
0.9501 0.4860 0.4565
0.2311 0.8913 0.0185
0.6068 0.7621 0.8214
>> a * b
ans =
3.2329 4.5549 2.9577
8.5973 10.9730 6.8468
13.9616 17.3911 10.7360
>>
Start

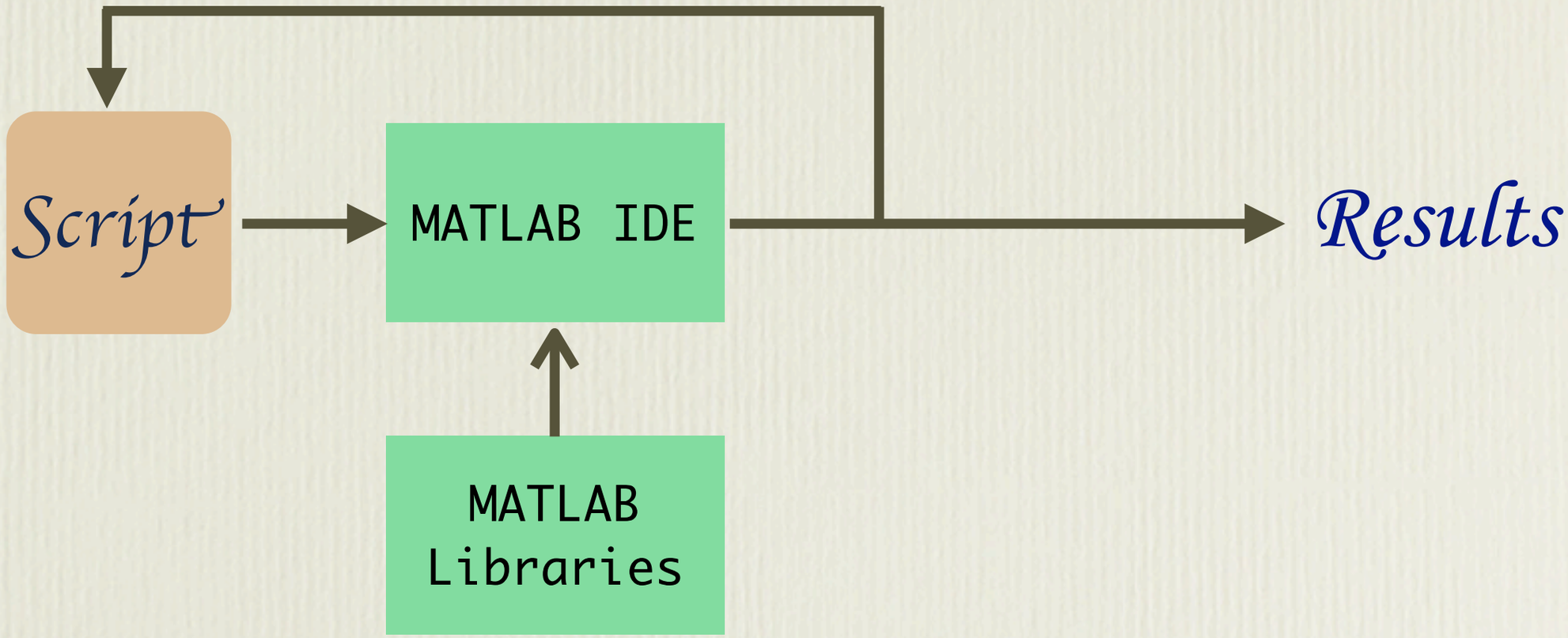
Why MATLAB?

- Ease of Use
  - writing
  - debugging
  - maintaining

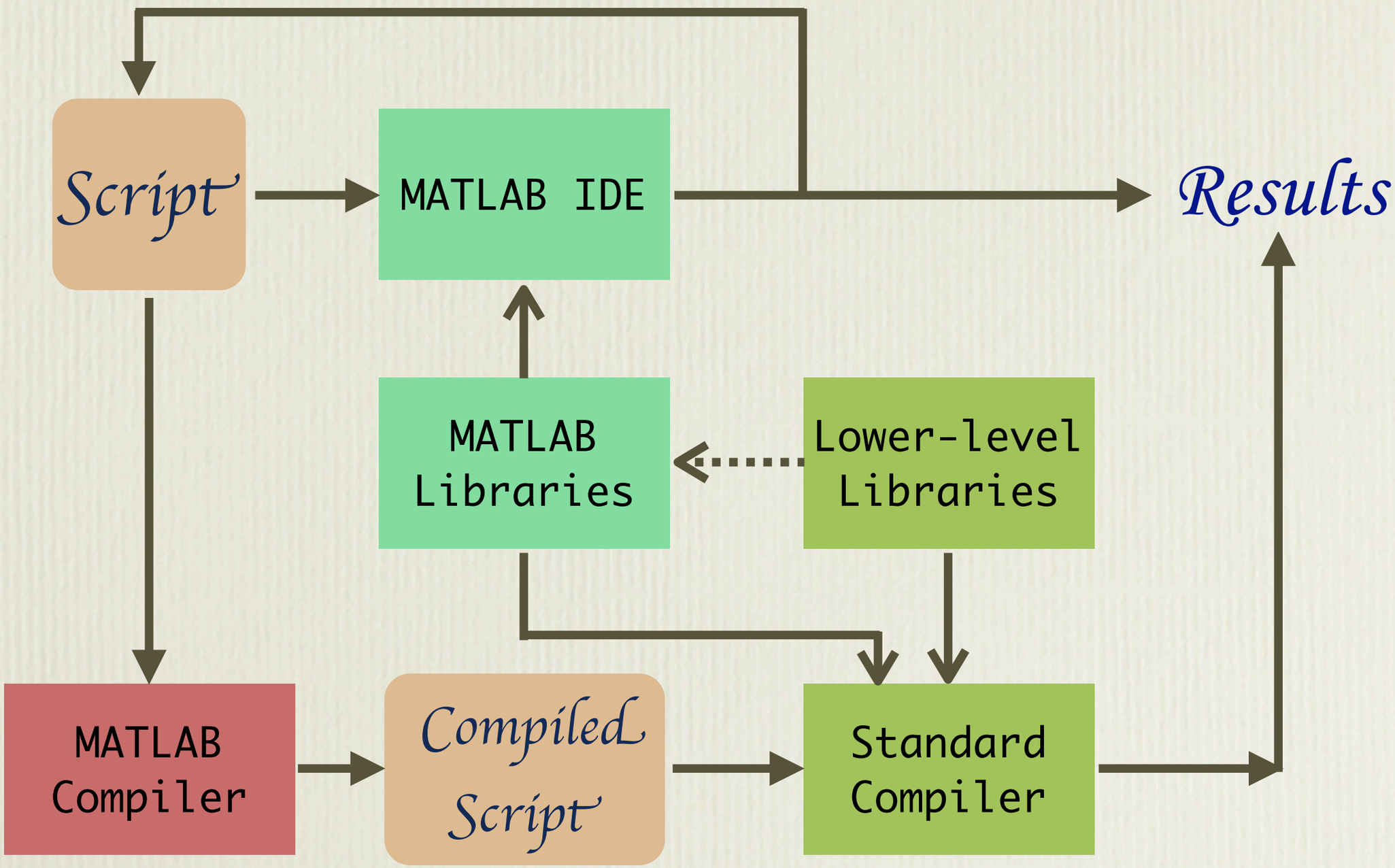
Operating Model



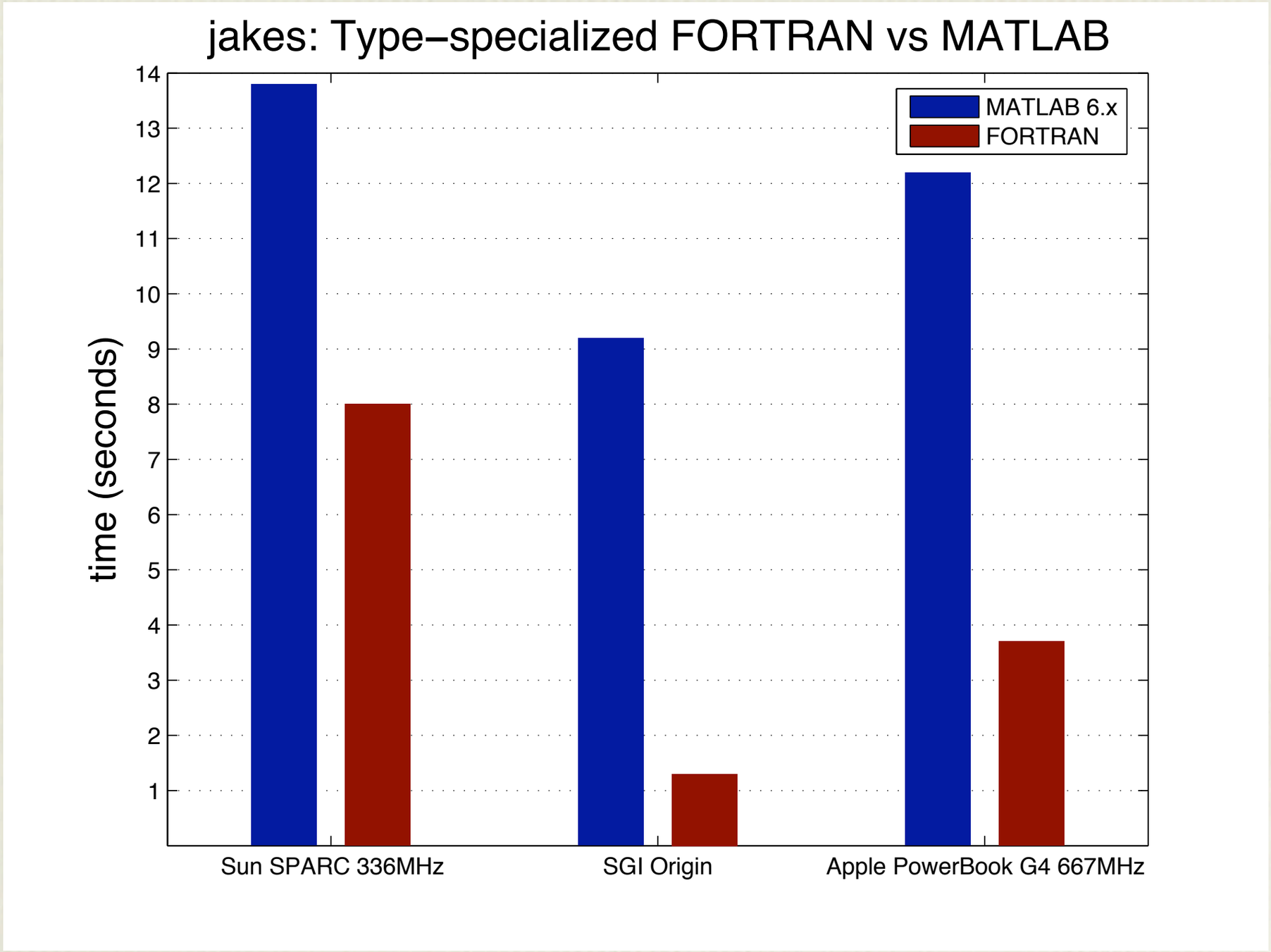
Operating Model



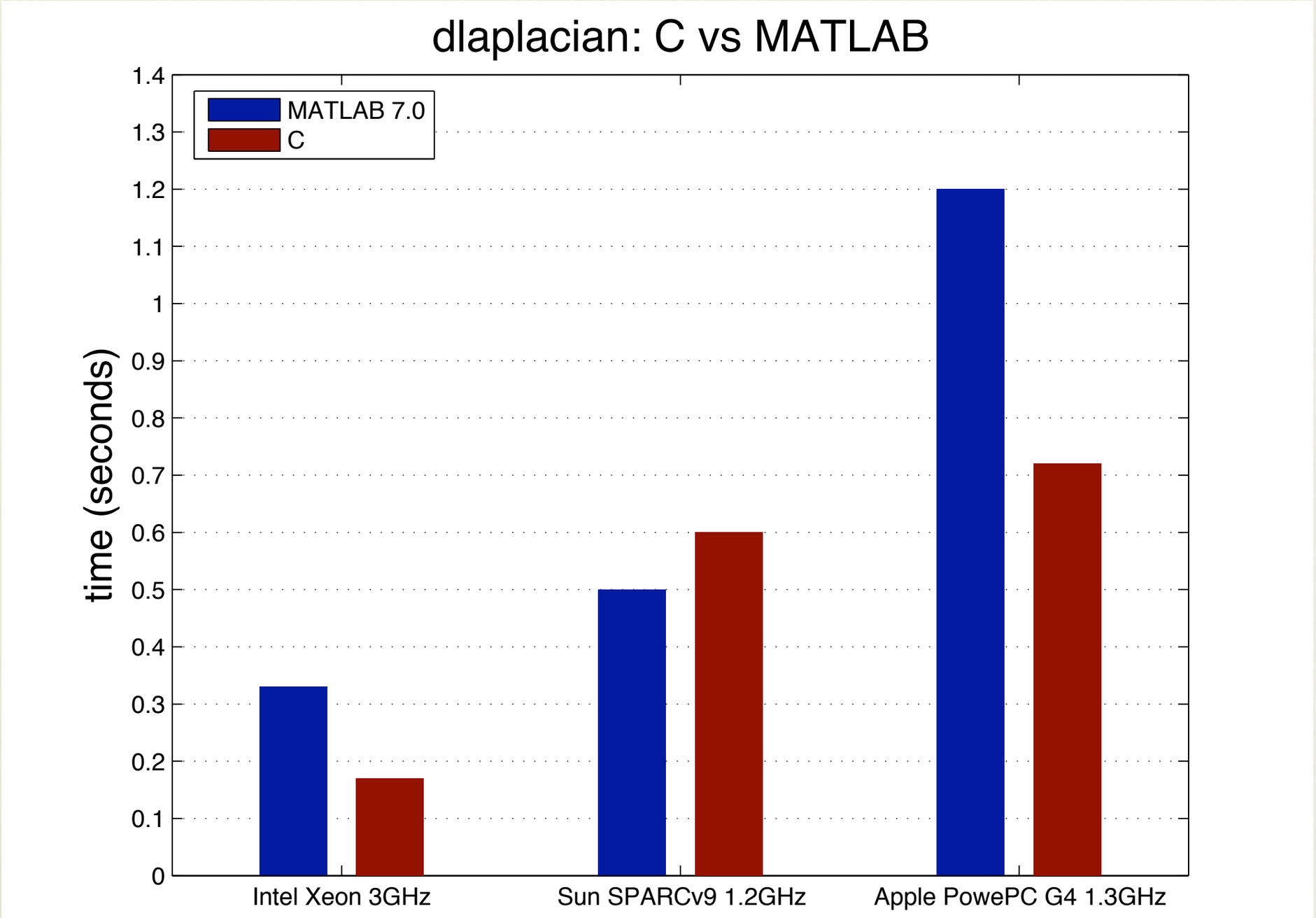
Operating Model



Lowering for Performance



Lowering for Performance (II)



Compiling and Lowering

- Lowering as the first step
  - enabled by type inference
- High-level operators enable contextual analysis
  - standard black-box technique is inadequate
- Compiler awareness of additional properties
  - annotations by library developers
  - automatic discovery

Optimizing Libraries

```
for ii = 1:200
  ...
  chan = jakes_mp1 (16500, 160, ii, num_paths);
  ...
  for snr = 2:2:20
     ...
     [s,x,ci,h,L,a,y,n0] = newcodesig (NO, I, num_paths, M, snr, chan, sig_pow_paths);
     [o1,d1,d2,d3,mf,m] = codesdhd (y, a, h, NO, Tm, Bd, M, B, n0);
     ...
  end
  ...
end
```

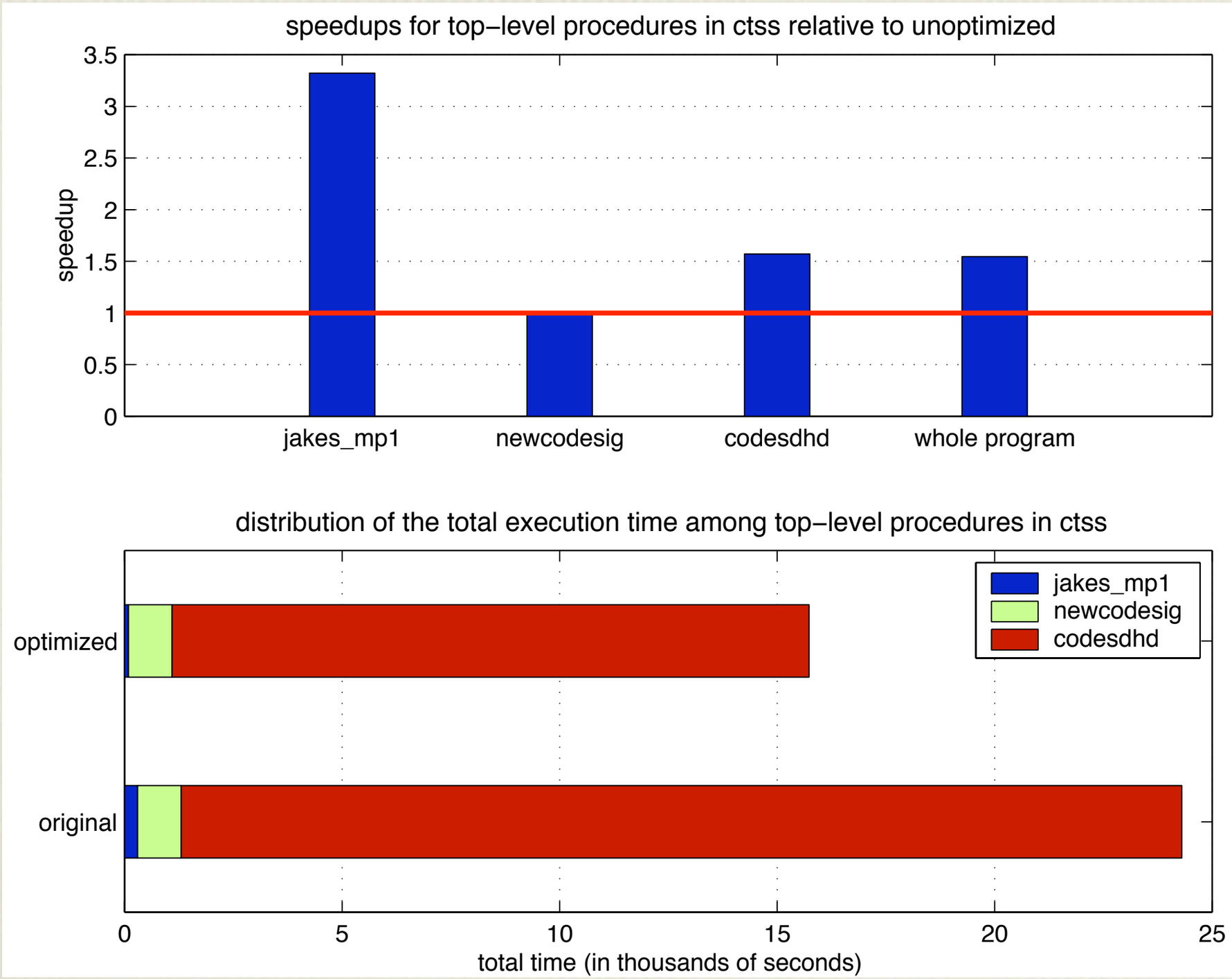
Optimizing Libraries

```
tmp = jakes_mp1_init (16500, 160, num_paths);
for ii = 1:200
  ...
  chan = jakes_mp1_iter (ii, tmp);
  ...
  ...
  for snr = 2:2:20
     ...
     [s,x,ci,h,L,a,y,n0] = newcodesig (NO, I, num_paths, M, snr, chan, sig_pow_paths);
     [o1,d1,d2,d3,mf,m] = codesdhd (y, a, h, NO, Tm, Bd, M, B, n0);
     ...
  end
  ...
end
```

Optimizing Libraries

```
tmp1 = jakes_mp1_init (16500, 160, num_paths);
[h,L,tmp2] = newcodesig_init_1 (NO, I, num_paths, M, sig_pow_paths);
[m, tmp3] = codesdhd_init (a, h, NO, Tm, Bd, M);
for ii = 1:200
  ...
  chan = jakes_mp1_iter (ii, tmp1);
  ...
  ...
  [a, tmp4] = newcodesig_init_2 (chan, tmp2);
  for snr = 2:2:20
    ...
    [s,x,ci,y,n0] = newcodesig_iter (snr, tmp4);
    ...
    [o1,d1,d2,d3,m] = codesdhd_iter (y, tmp3);
    ...
  end
  ...
end
```

Procedure Strength Reduction



Library Identities

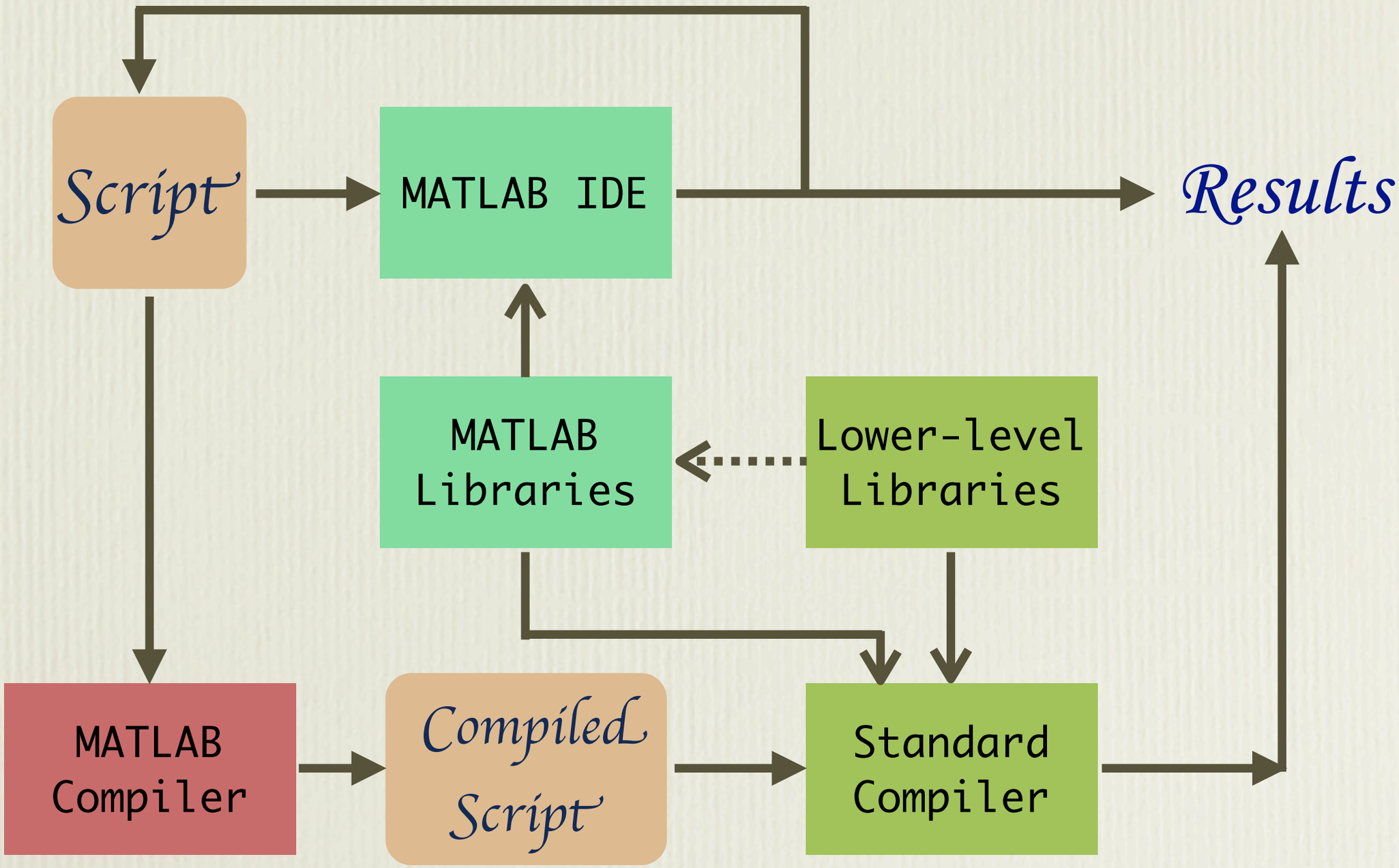
```
function [s, r, j_hist] = min_sr1 (xt, h, m, alpha)
  ...
  while ~ok
     ...
     invsr = change_form_inv (sr0, h, m, low_rp);
     big_f = change_form (xt-invsr, h, m);
     ...
     while iter_s < 3*m
        ...
        invdr0 = change_form_inv (sr0, h, m, low_rp);
        sssdr = change_form (invdr0, h, m);
        ...
     end
     ...
     invsr = change_form_inv (sr0, h, m, low_rp);
     big_f = change_form (xt-invsr, h, m);
     ...
     while iter_r < n1*n2
        ...
        invdr0 = change_form_inv (sr0, h, m, low_rp);
        sssdr = change_form (invdr0, h, m);
        ...
     end
      ...
end
```

Lessons

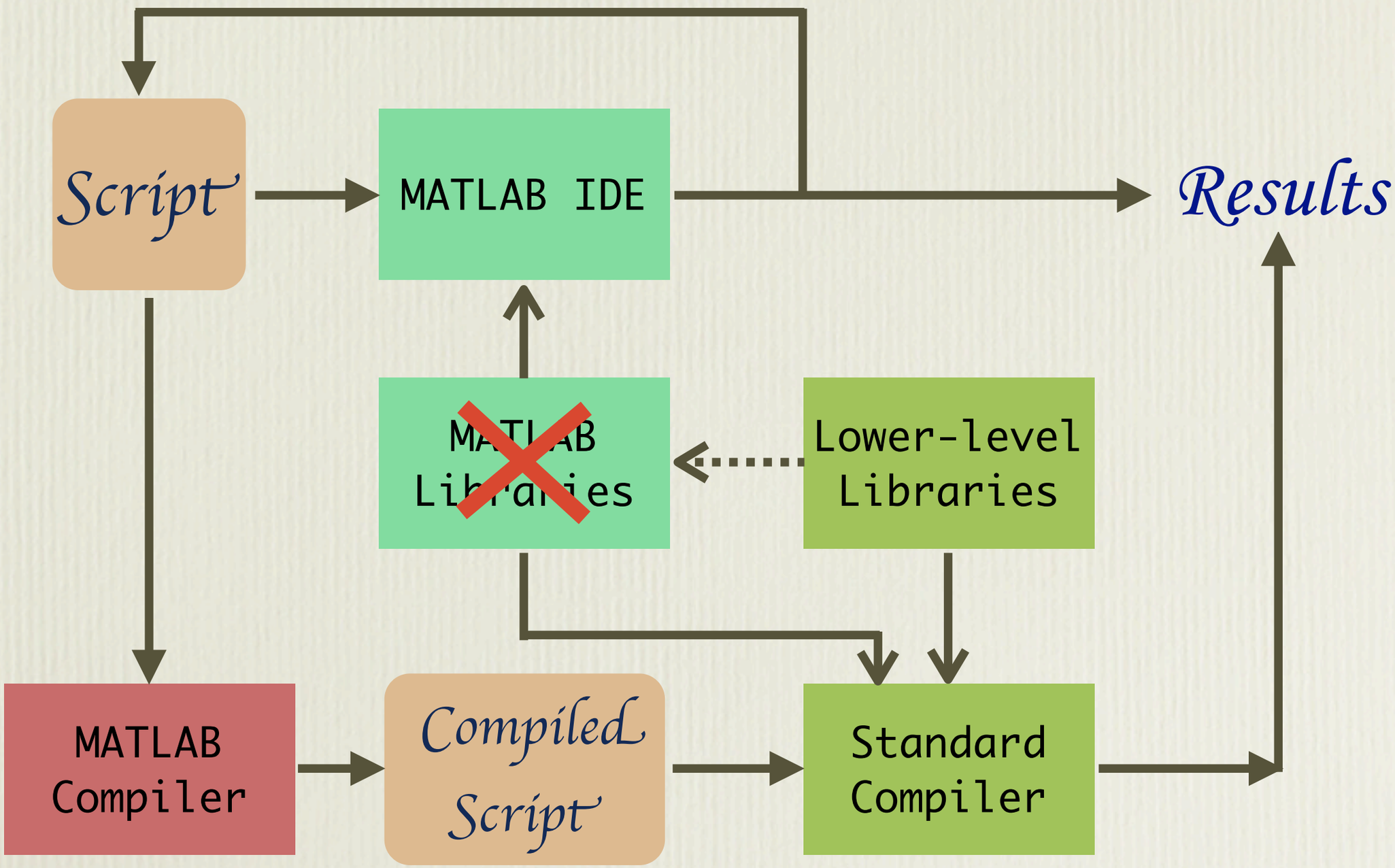
Two Types of Lessons

- Computer science
  - compilation (top-down vs bottom-up)
  - type inference strategy
- Software engineering
  - high-level compiler development

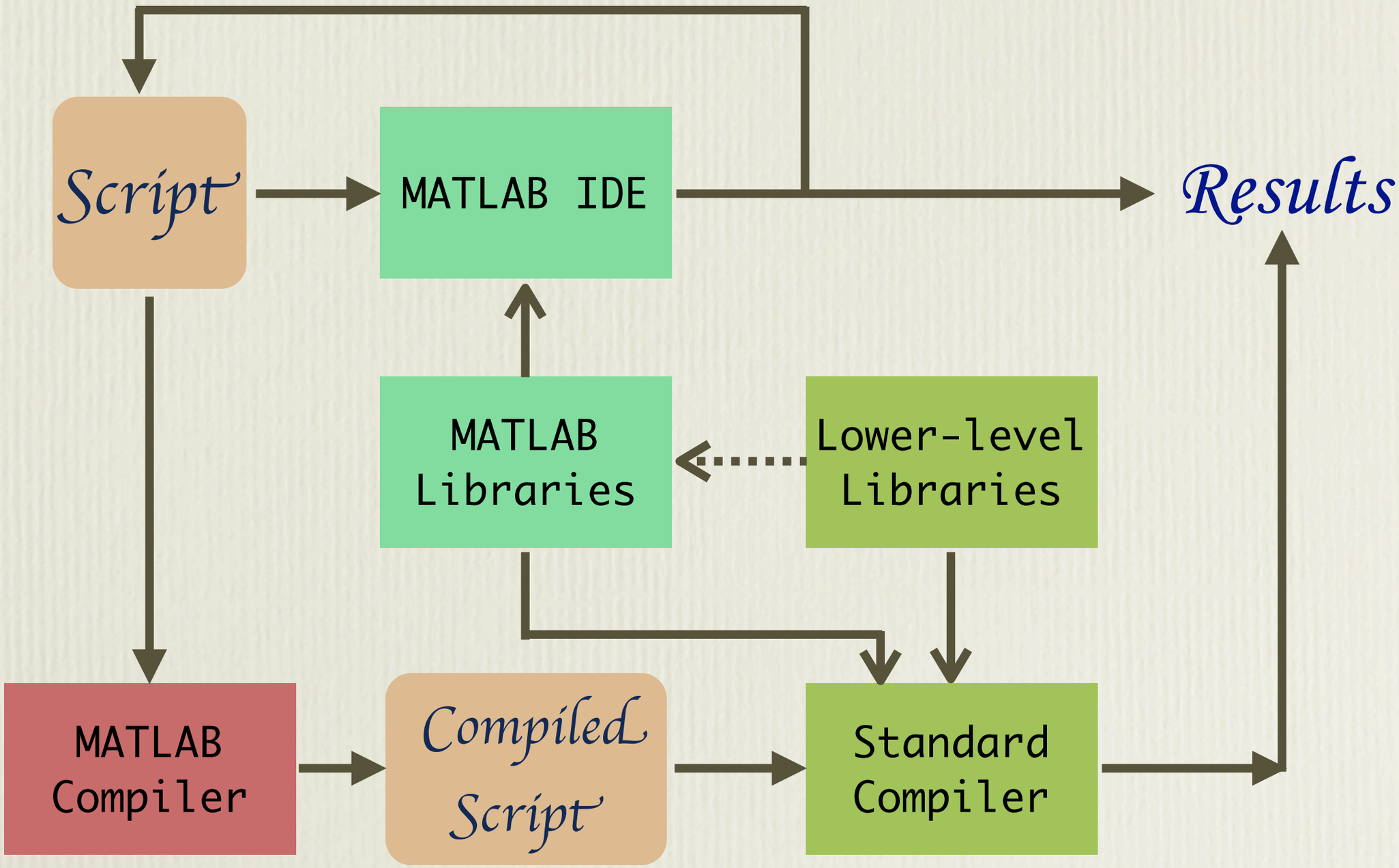
Compilation Strategy



Compilation Strategy



Compilation Strategy



Type Inference

Type = (intrinsic type, array size)

intrinsic type = integer, real, complex, etc.

array size = <d_1, d_2, d_3, ...>

- Heavily overloaded operators
  - backward propagation of very limited use
- Rich array subscripting semantics
  - arrays as subscripts
  - dynamically resizable arrays
  - "end" keyword

Subscripting in MATLAB

- Standard subscripting
  - A(10), A(j+1, i), A(B(i))
  - A(1:100)
- MATLAB subscripts: above plus ...
  - A(B), A(B,C) where B and C are arrays
  - A(end), A(end+2)
  - A(x, y) where A is a 3-D array

Type Inference: Solution

- Program transformation-based strategy
  1. Expose type disambiguation in source language
  2. Evaluate types through concrete interpretation
- Advantages
  - Availability of the power of MATLAB

```
z(1:2:end, 1:2:end, 1:2:end) = zn;
z = dlaplacian(z, [1,1/2,1/4,1/8]);
z = z(1:end-2, 1:end-2, 1:end-2);
```

```
a_0 = ParaM_compute_end_val(ParaM_size_z, 3, 1);
b_0 = ParaM_compute_end_val(ParaM_size_z, 3, 2);
c_0 = ParaM_compute_end_val(ParaM_size_z, 3, 3);
z(1:2:a_0, 1:2:b_0, 1:2:c_0) = zn;
d_0 = 1 / 2;
e_0 = 1 / 4;
f_0 = 1 / 8;
z = dlaplacian(z, [1, d_0, e_0, f_0]);
h_0 = g_0 - 2;
g_0 = ParaM_compute_end_val(ParaM_size_z, 3, 1);
j_0 = i_0 - 2;
i_0 = ParaM_compute_end_val(ParaM_size_z, 3, 2);
l_0 = k_0 - 2;
k_0 = ParaM_compute_end_val(ParaM_size_z, 3, 3);
z = z(1:h_0, 1:j_0, 1:l_0);
```

Type Inference: Solution

- Program transformation-based strategy
  1. expose type disambiguation in source language
  2. evaluate types through concrete interpretation
- Advantages
  - availability of the power of MATLAB
  - continuous spectrum of possibilities

Engineering

- C++ as the language of choice
  - wide availability
  - "portable"
  - modular (?)
- A "high-level" language for writing compilers
  - Stratego (University of Utrecht, the Netherlands)

Status and Plan

Compiler Implementation

- Moved infrastructure to Stratego
  - shared with Ohio-State, Rice, UCSB
- Robust handling of difficult subscripting cases
  - exception: cell subscripting
- Type inference
  - exception: cells, structs

Plan: Compiling MATLAB

- Implement the hybrid compilation strategy
- Identify libraries
  - C++ generics or C / Fortran
    - study undertaken by Youngsang to evaluate the performance tradeoffs
  - Octave, Andrew Lumsdaine's concepts-based
- Implement "high-level" transformations
  - annotation language

Plan: Parallelizing MATLAB

- Model: distributed data
  - distributed data ⇒ parallel operations
  - replicated data ⇒ sequential operations
- Implementation
  - dependence analysis
  - cost-based analysis of data distributions
  - library development

Thank You