Marble

Matrix Manipulation Made Manageable

Our Team

Huaxuan Gao
System Architect

Qiwen Luo
Language Guru

Yixin Pan
Super Tester

Xindi Xu
Project Manager

Matrix in Other Languages

Python:

- ✗ No built-in matrix manipulation
- ✗ No type checking
- ✓ Flexible and simple

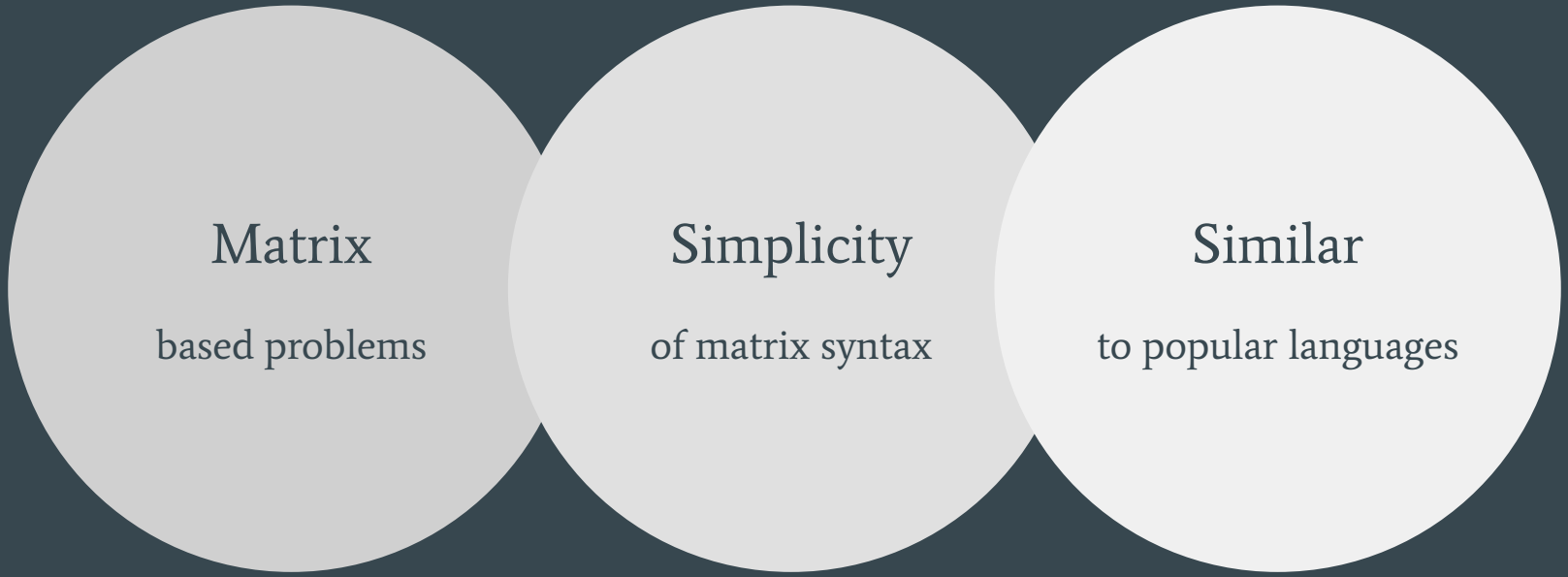
```
m = [[1,2,3], [4,5,6], [7,8,9]]
m[0][0]
numpy.matmul(m,2); # 3rd party lib
```

Matlab:

- ✓ Many built-in functions
- ✗ Unusual 1-based indexing
- ✗ GUI-based

```
m = [1 2 3; 4 5 6; 7 8 9]
m(0,0) % error
m*2
```

Motivation


Matrix
based problems
Simplicity
of matrix syntax
Similar
to popular languages

Language Overview

Data types

```
int,float,bool,matrix
```

Operators

```
+,-,*,/,%,==,!=,<,<=,>,>=
!,&&,||
```

Function Declaration & Scope

```
int function main(){...}
```

Built-in function - print

```
print,printf,printb,printmf
```

Comments

```
//, /* */
```

Variable - Declaration & Assignment

```
int gloabal_local; OR float local = 1.1;
local = 2.1; OR local += 1.1; OR local -= 1.1;
```

While & For

```
while(i < 10){...}
for(int i = 0; i < 10; i += 1){...}
```

If-else

```
if(i < 10){...} else {...}
```

Return

```
return 0;
```

Language Overview: Matrix

Declaration

```
matrix a = [1.2,2.3;3.2,4.1];

matrix z = zeros(2, 2); // [0.0,0.0;0.0.0.0]
```

Access & Assignment

```
a[0,1] // addr=sp+0*col_num+1+offset

a[0,1] = 2.1;
```

Operators

```
+,-,*(cross product)
```

Built-in functions

```
int row = rows(a);

int col = cols(a);
```

```
int function main() {
    matrix z = zeros(2,2);
    // mat +/- mat
    matrix a = [1.1, 2.2; 3.3, 4.4];
    matrix b = [5.5, 6.6; 7.7, 8.8];
    matrix c = a + b;
    matrix d = a - b;
    // int * mat mat * int
    int d = 3;
    matrix e = d * a;
    matrix f = a * d;

    // float * mat mat * float
    float g = 1.1;
    matrix h = g * a;
    matrix i = b * g;

    // mat * mat
    matrix j = a * b;
    matrix k = b * a;
    return 0;
}
```

Sample code: determinant

Variable Declaration
(Matrix Initialization)

Function Declaration
& Function Call

For Loop
& Binary Operations

Matrix Operations
& Built-in Functions

```
float function determinant(matrix a){
    int dim = rows(a);
    // Row ops on matrix a to get in upper triangle form
    for(int dia = 0; dia < dim; dia=dia+1){
        for(int r = dia + 1; r < dim; r=r+1){
            float r_scalar = a[r,dia] / a[dia,dia];
            for(int c = 0; c < dim; c=c+1){
                a[r,c] = a[r,c] - r_scalar * a[dia,c];
            }
        }
    }
    // Multiply entries on the diagonal
    float product = 1.0;
    for(int i = 0; i < dim; i = i+1){
        product = product * a[i,i];
    }
    return product;
}
int function main(){
    matrix a = [1.0, 2.0; 3.0, 4.0];
    float det_a = determinant(a);
    return det_a;
}
```

Matrix Representation (Using printmf)

1.0 2.0
3.0 4.0

Sample code: determinant

Variable Declaration
(Matrix Initialization)

Function Declaration
& Function Call

For Loop
& Binary Operations

Matrix Operations
& Built-in Functions

```
float function determinant(matrix a){
    int dim = rows(a);
    // Row ops on matrix a to get in upper triangle form
    for(int dia = 0; dia < dim; dia=dia+1){
        for(int r = dia + 1; r < dim; r=r+1){
            float r_scalar = a[r,dia] / a[dia,dia];
            for(int c = 0; c < dim; c=c+1){
                a[r,c] = a[r,c] - r_scalar * a[dia,c];
            }
        }
    }
    // Multiply entries on the diagonal
    float product = 1.0;
    for(int i = 0; i < dim; i = i+1){
        product = product * a[i,i];
    }
    return product;
}
int function main(){
    matrix a = [1.0, 2.0; 3.0, 4.0];
    float det_a = determinant(a);
    return 0;
}
```

Sample code: determinant

Variable Declaration
(Matrix Initialization)

Function Declaration
& Function Call

For Loop
& Binary Operations

Matrix Operations
& Built-in Functions

```
float function determinant(matrix a){
    int dim = rows(a);
    // Row ops on matrix a to get in upper triangle form
    for(int dia = 0; dia < dim; dia=dia+1){
        for(int r = dia + 1; r < dim; r=r+1){
            float r_scalar = a[r,dia] / a[dia,dia];
            for(int c = 0; c < dim; c=c+1){
                a[r,c] = a[r,c] - r_scalar * a[dia,c];
            }
        }
    }
    // Multiply entries on the diagonal
    float product = 1.0;
    for(int i = 0; i < dim; i = i+1){
        product = product * a[i,i];
    }
    return product;
}
int function main(){
    matrix a = [1.0, 2.0; 3.0, 4.0];
    float det_a = determinant(a);
    return 0;
}
```

Sample code: determinant

Variable Declaration
(Matrix Initialization)

Function Declaration
& Function Call

For Loop
& Binary Operations

Matrix Operations
& Built-in Functions

```
float function determinant(matrix a){
    int dim = rows(a);
    // Row ops on matrix a to get in upper triangle form
    for(int dia = 0; dia < dim; dia=dia+1){
        for(int r = dia + 1; r < dim; r=r+1){
            float r_scalar = a[r,dia] / a[dia,dia];
            for(int c = 0; c < dim; c=c+1){
                a[r,c] = a[r,c] - r_scalar * a[dia,c];
            }
        }
    }
    // Multiply entries on the diagonal
    float product = 1.0;
    for(int i = 0; i < dim; i = i+1){
        product = product * a[i,i];
    }
    return product;
}
int function main(){
    matrix a = [1.0, 2.0; 3.0, 4.0];
    float det_a = determinant(a);
    return 0;
}
```

Sample code: determinant

Variable Declaration
(Matrix Initialization)

Function Declaration
& Function Call

For Loop
& Binary Operations

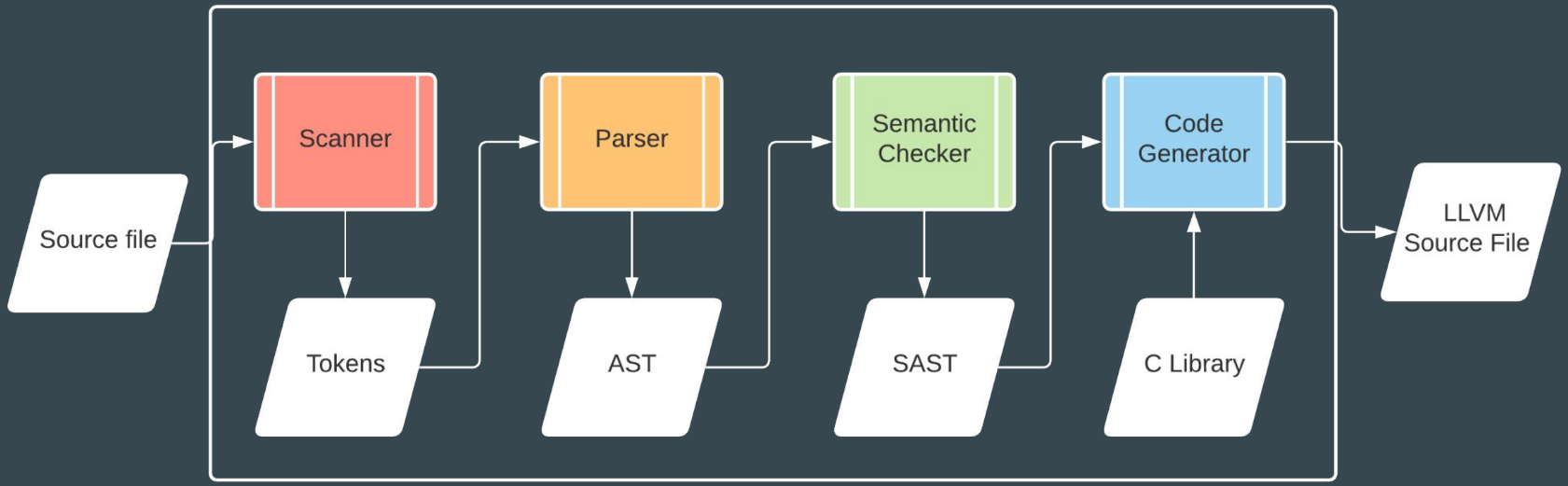
Matrix Operations
& Built-in Functions

```
float function determinant(matrix a){
    int dim = rows(a);
    // Row ops on matrix a to get in upper triangle form
    for(int dia = 0; dia < dim; dia=dia+1){
        for(int r = dia + 1; r < dim; r=r+1){
            r_scalar = a[r,dia] / a[dia,dia];
            for(int c = 0; c < dim; c=c+1){
                a[r,c] = a[r,c] - r_scalar * a[dia,c];
            }
        }
    }
    // Multiply entries on the diagonal
    float product = 1.0;
    for(int i = 0; i < dim; i = i+1){
        product = product * a[i,i];
    }
    return product;
}
int function main(){
    matrix a = [1.0, 2.0; 3.0, 4.0];
    float det_a = determinant(a);
    return 0;
}
```

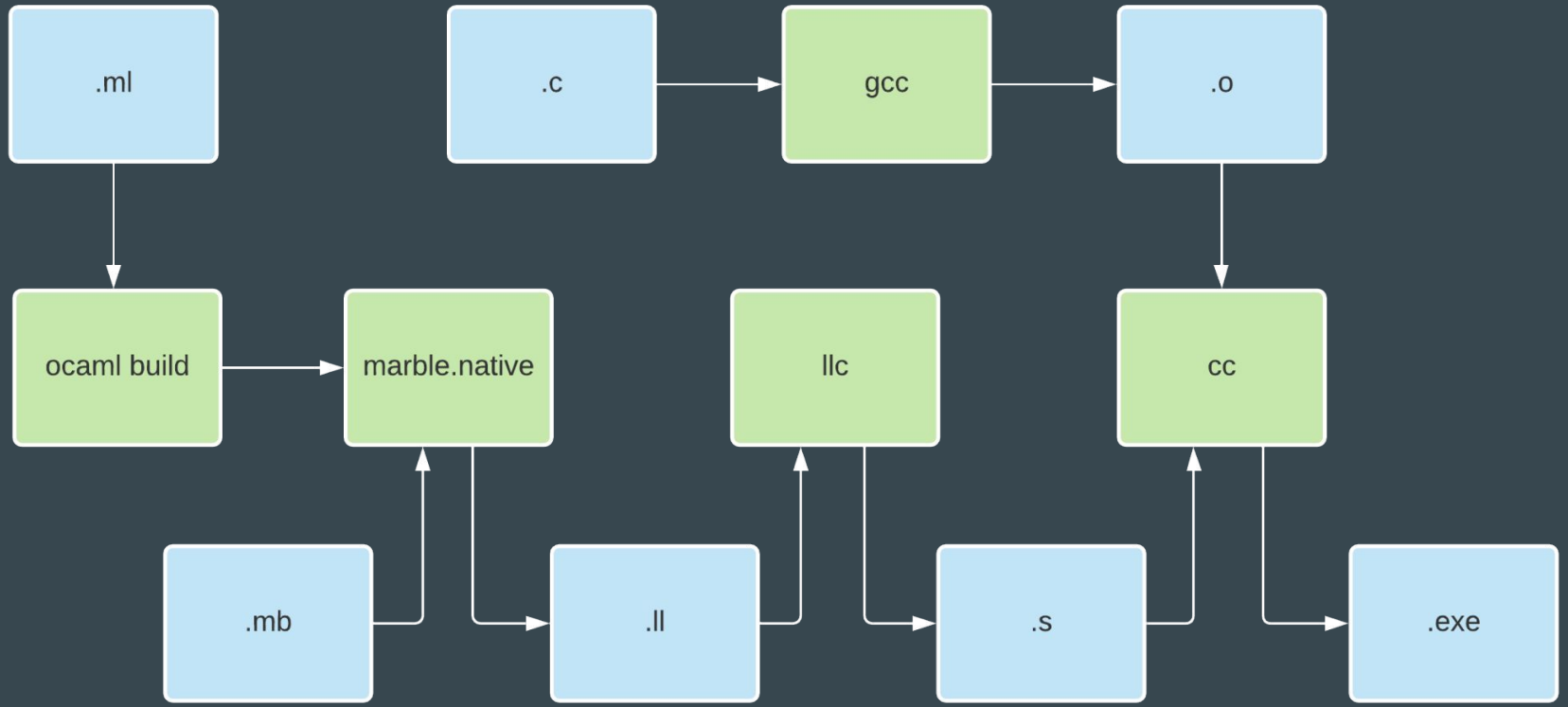
Output:

-2.0

Architecture



Execution Process



Future Work

Credits

Special thanks to:

Maxwell Levatich, our Project Advisor

Professor **Stephen A. Edwards**

And wonderful past projects: MicroC, Matrix Mania,

Demo

Questions?