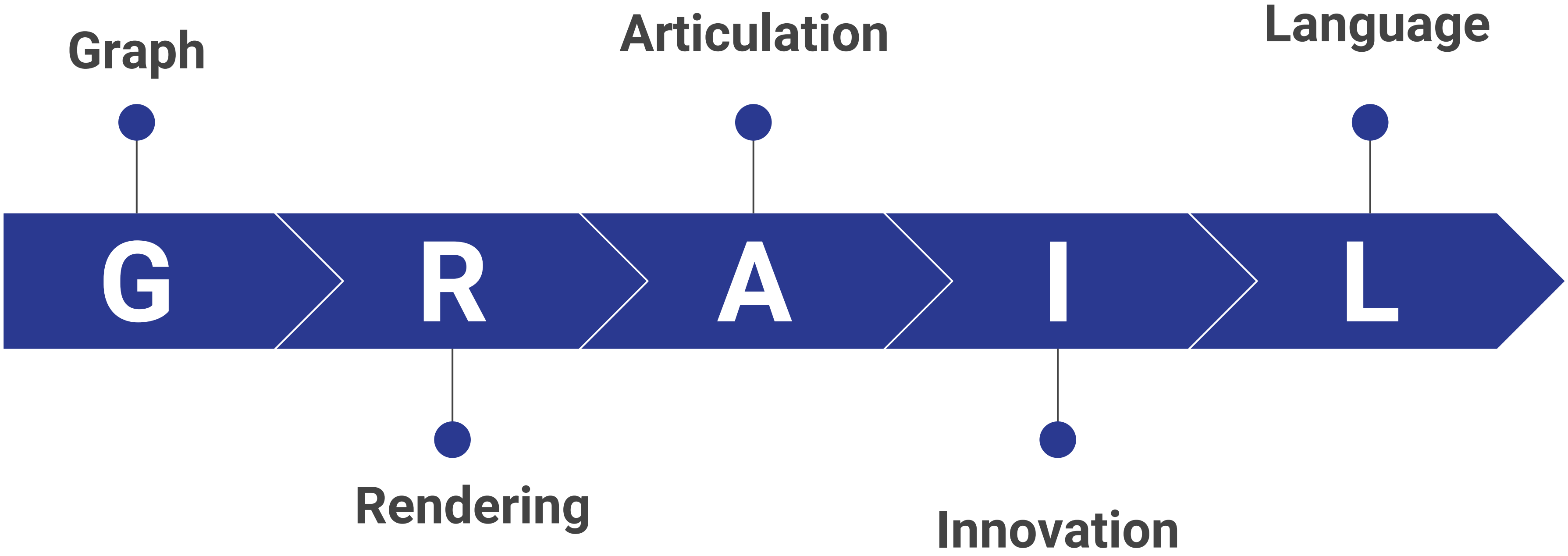

Graph
Rendering
Articulation
Innovation
Language
G
R
A
I
L

Motivation

- Represent complex graphs with syntactic simplicity
- Allow users to define flexible data-types
- Type-inference allows for concise representation of data

Key Features

Intuitive Syntax

```
c = {station: "49th St Station", line: "1", lat:39.9436, lon:75.2167, service: [0,1,1,1,1,1,1]};

d = {station: "116th St Station", line: "1", lat:39.56, lon:75.456, service: [0,1,1,1,1,1,0]};

g = (c -- d) with {distance: 1};
```

Primitive Types & Control Flows

Basic Types

Int, float, char, str, bool

Binary & Unary Ops

Arithmetic: +, -, *, /

Logical: >, >=, <, <=, =, ==

if ... else ...

if ... else if ...

while ...

for (...;...;...;) ...

for (... in ...) ...

Key Features

Graphical Data Structures & Operations

List

size(c.service);

Dot

c.station = “168th”;

Record

e = {test:1};

y = e.test;

Edge

u -- v with e

Graph

g =(a, b, c -- d) with {test:1}

display()

display(g)

Derived Types

List

Edges

Graphs

Records

Key Features

Structural Comparison

```
lance = {name: "Lancelot", quest:
"grail", colors:["blue"])};
robin ={name: "Robin", quest:
"grail", colors:["blue", "yellow"]};
lance == {name: "Lancelot", quest:
"grail", colors:["blue"]));
lance.colors == robin.colors;
```

Deep Copy

```
x = [[1, 2], [3,4]];
y .= x;
y[0] = [8, 9];
fsty = y[0];
fstx = x[0];
printint(fsty[0]);
printint(fstx[0]);
```

Key Features

Type Inference

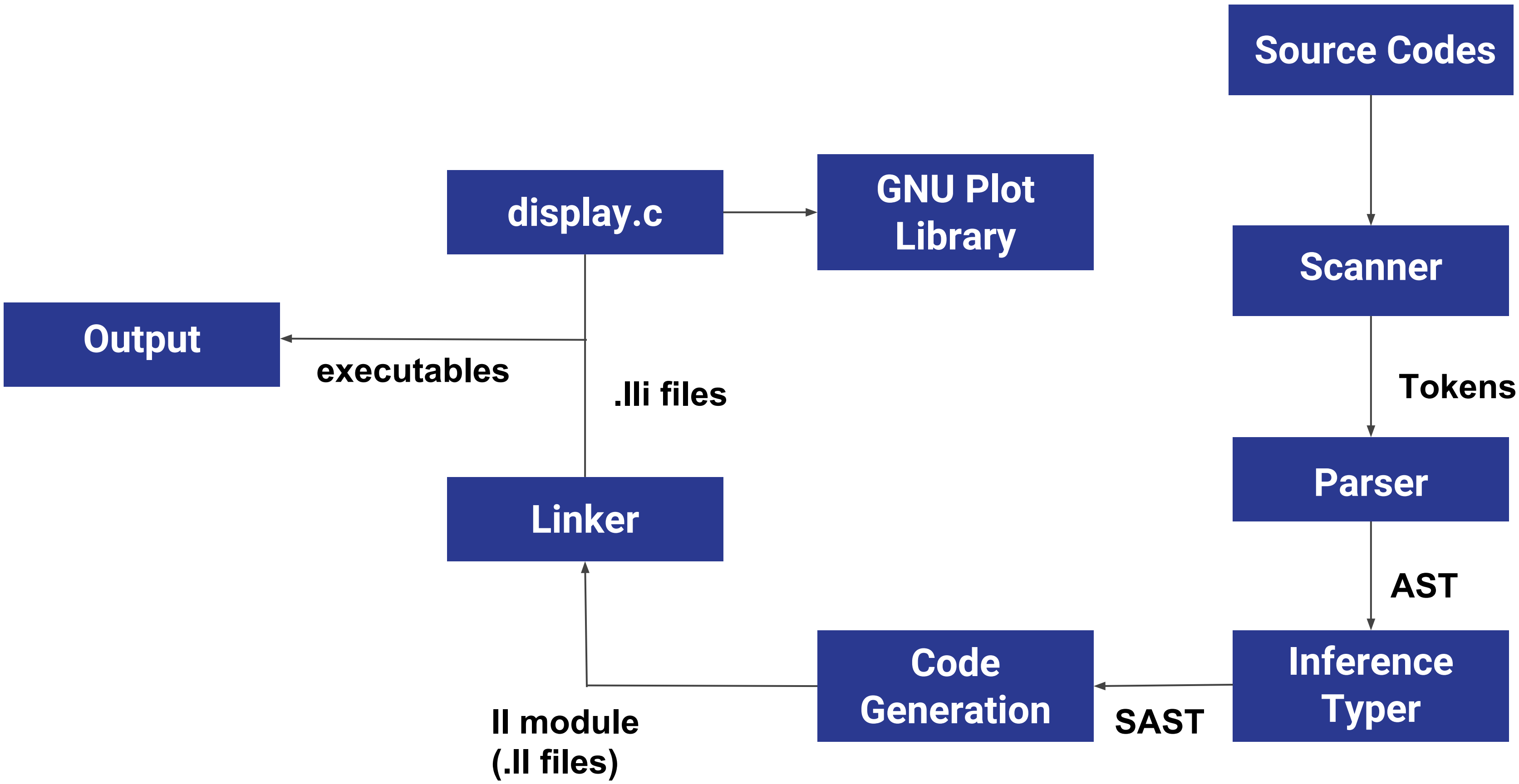
```
f(x, y){
   return x + y;
}
g(z){
  y = 0;
  if(z == "hello"){
       y = 5;
  }
  return y;
}
```

```
h(x){
  return x;
}
p(x){
  return x.school;
}
```

```
main(){
  x = f(1,2);
  y = x + g(3);
  hi = h("hi");
  z = h(3);
  a = {school: 3};
  p(a);
}
```

Implementation

Compiler Architectures



Testing

```
mkdir: test_output: File exists
-n test-comment...
OK
-n test-comparison...
OK
-n test-float-calculation...
OK
-n test-for-1...
OK
-n test-for-2...
OK
-n test-forin-1...
OK
-n test-forin-2...
OK
-n test-function-call-1...
OK
-n test-function-call-2...
OK
-n test-function-call-3...
OK
```

```
###### Testing test-while-1
./grail.native < tests/new_tests/test-while-1.gl > test-while-1.ll
/usr/local/opt/llvm/bin/llvm-link test-while-1.ll -o a.out
/usr/local/opt/llvm/bin/lli a.out > test-while-1.out
diff -b test-while-1.out tests/new_tests/test-while-1.out > test-while-1.diff
###### SUCCESS

###### Testing test-while-2
./grail.native < tests/new_tests/test-while-2.gl > test-while-2.ll
/usr/local/opt/llvm/bin/llvm-link test-while-2.ll -o a.out
/usr/local/opt/llvm/bin/lli a.out > test-while-2.out
diff -b test-while-2.out tests/new_tests/test-while-2.out > test-while-2.diff
###### SUCCESS

###### Testing fail-assign-1
./grail.native < tests/new_tests/fail-assign-1.gl 2> fail-assign-1.err >> testall.log
diff -b fail-assign-1.err tests/new_tests/fail-assign-1.err > fail-assign-1.diff
###### SUCCESS

###### Testing fail-assign-2
./grail.native < tests/new_tests/fail-assign-2.gl 2> fail-assign-2.err >> testall.log
diff -b fail-assign-2.err tests/new_tests/fail-assign-2.err > fail-assign-2.diff
###### SUCCESS

###### Testing fail-expr-1
./grail.native < tests/new_tests/fail-expr-1.gl 2> fail-expr-1.err >> testall.log
diff -b fail-expr-1.err tests/new_tests/fail-expr-1.err > fail-expr-1.diff
###### SUCCESS

###### Testing fail-expr-2
./grail.native < tests/new_tests/fail-expr-2.gl 2> fail-expr-2.err >> testall.log
diff -b fail-expr-2.err tests/new_tests/fail-expr-2.err > fail-expr-2.diff
###### SUCCESS
```

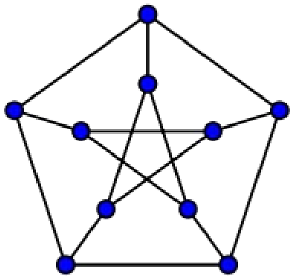
General Compiler Testing Plan

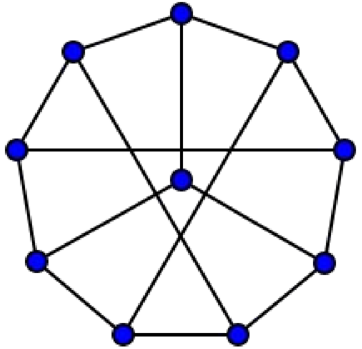
- Start from basics, like arithmetic operators, and move on to advanced features
- Feed unit test case codes for new-implemented features with expected outputs/errors
- Check for exceptions or errors
  - Unit test cases syntax correct?
  - What kind of exceptions?
  - Scanner, Parser, Typer, or Codegen?
  - Send through type-tester
- Use LLVM Interpreter
- Implemented testing programs that can get outputs from parser or typer if we feed the testers with test code files

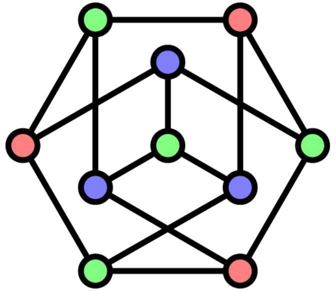
Demo

Petersen Graph

```
main()
{
    //construct the Petersen graph

    petenodes = [{key: 1}, {key: 2}, {key: 3}, {key: 4}, {key: 5}, {key: 6}, {key: 7}, {key: 8}, {key: 9}, {key: 10}];
    pete = ({key: 0}) with {weight:1};

    for(n in petenodes){
        pete &= n;
    }

    for(i = 0; i < 5; i += 1;){
        pi = petenodes[i];
        po = petenodes[i+5];
        pete .&= pi--po;
        if(i == 0){
            p2 = petenodes[2];
            p3 = petenodes[3];
            pete .&= pi -- p2;
            pete .&= pi -- p3;
        }

        if(i == 1){
            p3 = petenodes[3];
            p4 = petenodes[4];
            pete .&= pi -- p3;
            pete .&= pi -- p4;
        }

        if(i == 2){
            p4 = petenodes[4];
            pete .&= pi -- p4;
        }
    }

    for(i = 5; i < 9; i += 1;){
        pi = petenodes[i];
        pplus = petenodes[i+1];
        pete .&= pi--pplus;
    }


}
```







Thank You!