Graphene

So what's the most efficient way to spam all your Facebook friends?

Team

Adith Tekur
(System Architect/Tester)

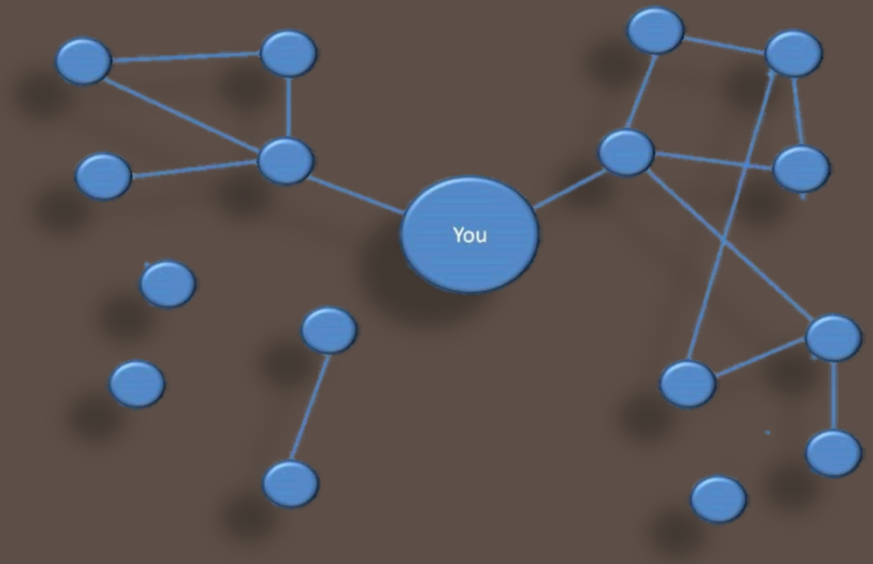
Neha Rastogi
(System Integrator)

Pooja Prakash
(Language Guru)

Sarah Panda
(Project Manager)

Sumiran Shah
(System Architect/Tester)

graphene> why();



- Web 2.0. Cloud. Big Data. Social Networking.
- Need to analyze common patterns, find trends
- Useful for advertisers, market researchers, beginner programmers who just need another language with easy to use graph interfaces

graphene> what();

- Graphene is:
  - Dynamically typed
  - Interpreted
  - Concise
  - High level
- Graphene enables users to perform common and complex graph manipulations on user defined or imported data
- Graphene features a powerful visualization library to display graphs or input them manually
- Graphene can be invoked in a REPL to perform quick, incremental operations

graphene>
sample();

```
person has name,age;
graph = input("person","data/0.edges");

def (graph) => getNodeWithMaxDegree
{
    maxnode = graph.getNodes().first();
    maxdegree = graph.getAdjacent(graph.getNodes().first()).length();
    foreach(node in graph.getNodes())
    {
        curr = graph.getAdjacent(node).length();

        if(curr > maxdegr)
        {
            maxnode = node;
            maxdegree = curr;
        }
    }
} => (maxnode);

def (graph) => GlobalReach
{
    Nodes = [];
    while(graph.getNodes().length()!=0)
    {
        current = getNodeWithMaxDegree(graph);
        Nodes.append(current);
        foreach(node in graph.getAdjacent(current))
        {
            graph -= node;
        }
        graph -= current;
    }
} => (Nodes);

g = input("person","data/0.edges");

l = GlobalReach(graph);
output().overlay(l);
```

Define a type of node of type person and read file dump

User function to find the node with max degree from a graph

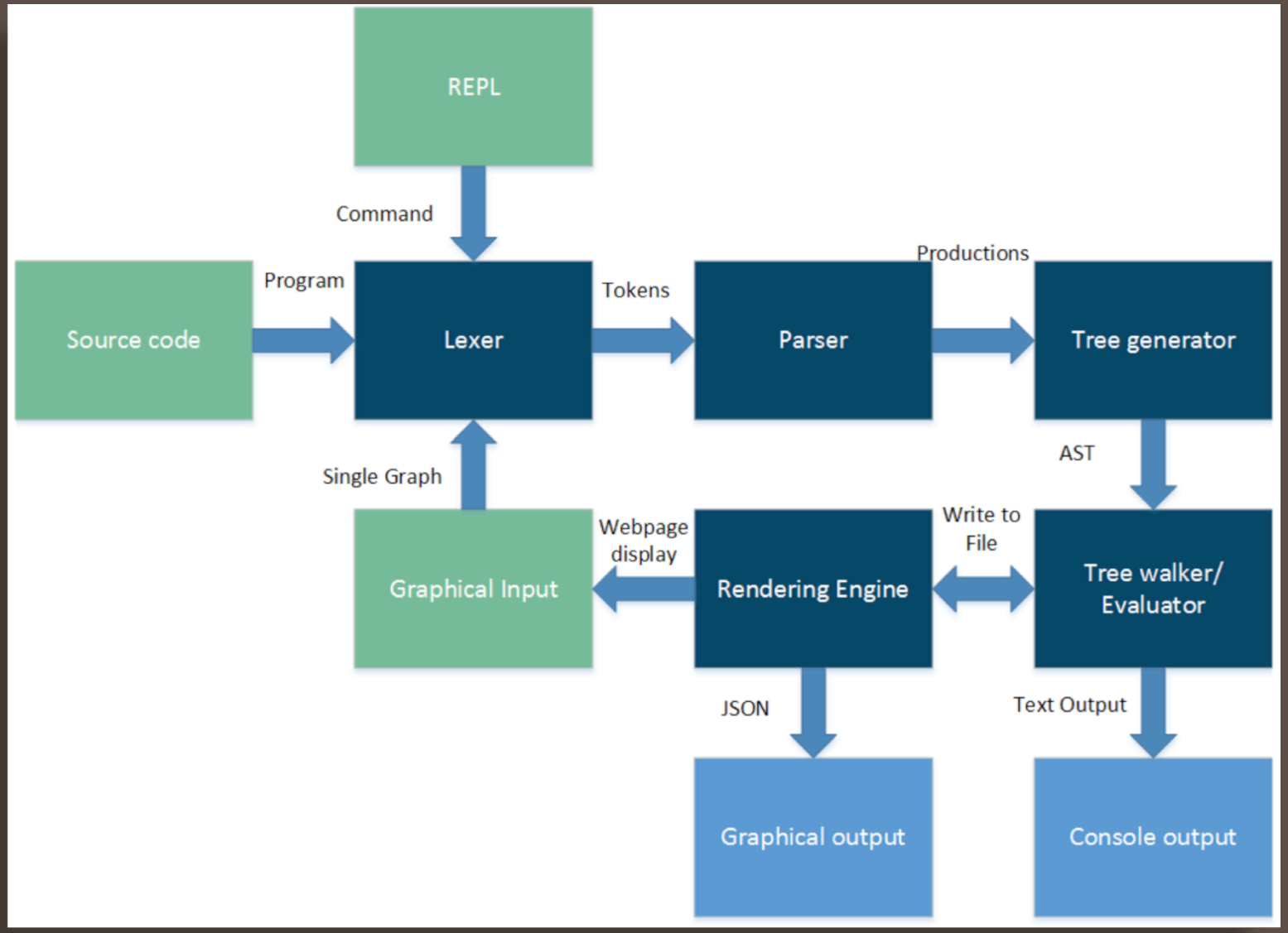
Return maxnode variable

List type

Node removal operator

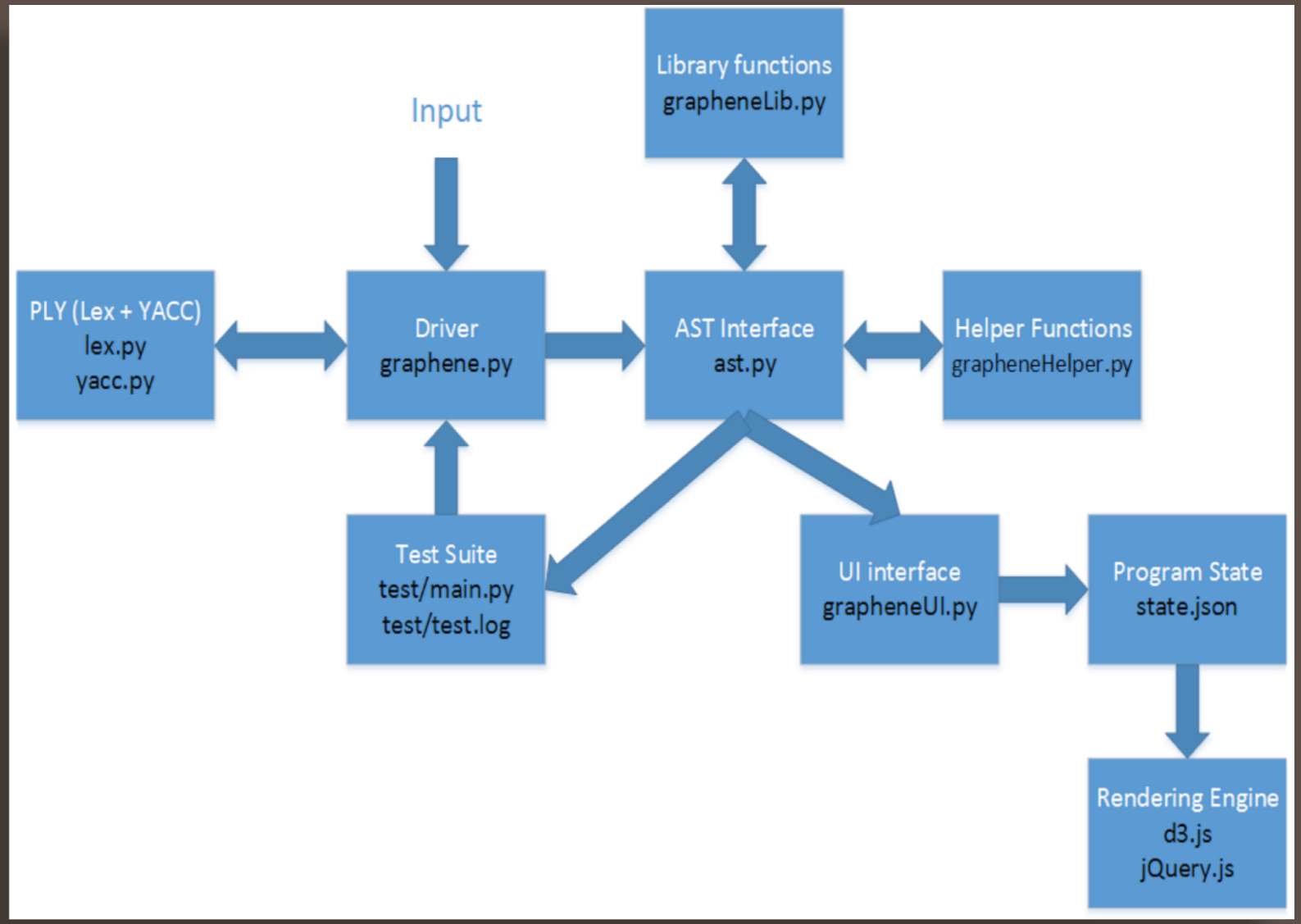
Visualize!

graphene> how();



- Flow diagram

graphene> how();



- Compiler and supporting architecture

graphene> write();

- No main function. Script is executed line by line
- Dynamically typed data
- Specialized Graph and Node data types and operations
- Features basic constructs such as if, for, foreach, while, functions
- More advanced constructs such as multiple return arguments, lambdas(first class functions), inbuilt libraries for clustering nodes, graph manipulation operators

graphene> write();

Getting your graph ready:

- Define a node type
- Create nodes
- Define a graph with edges

```
Person has name,age,city;

a=Person("A",30,"New York");
b=Person("B",31,"Seattle");
c=Person("C",30,"Boston");
d=Person("D",32,"Los Angeles");

Graph g has d{ "A" -> "B", "B" -> "C", "C" -> "B", "C" -> "D"} on "name";
```

graphene> write();

- Or maybe import from file

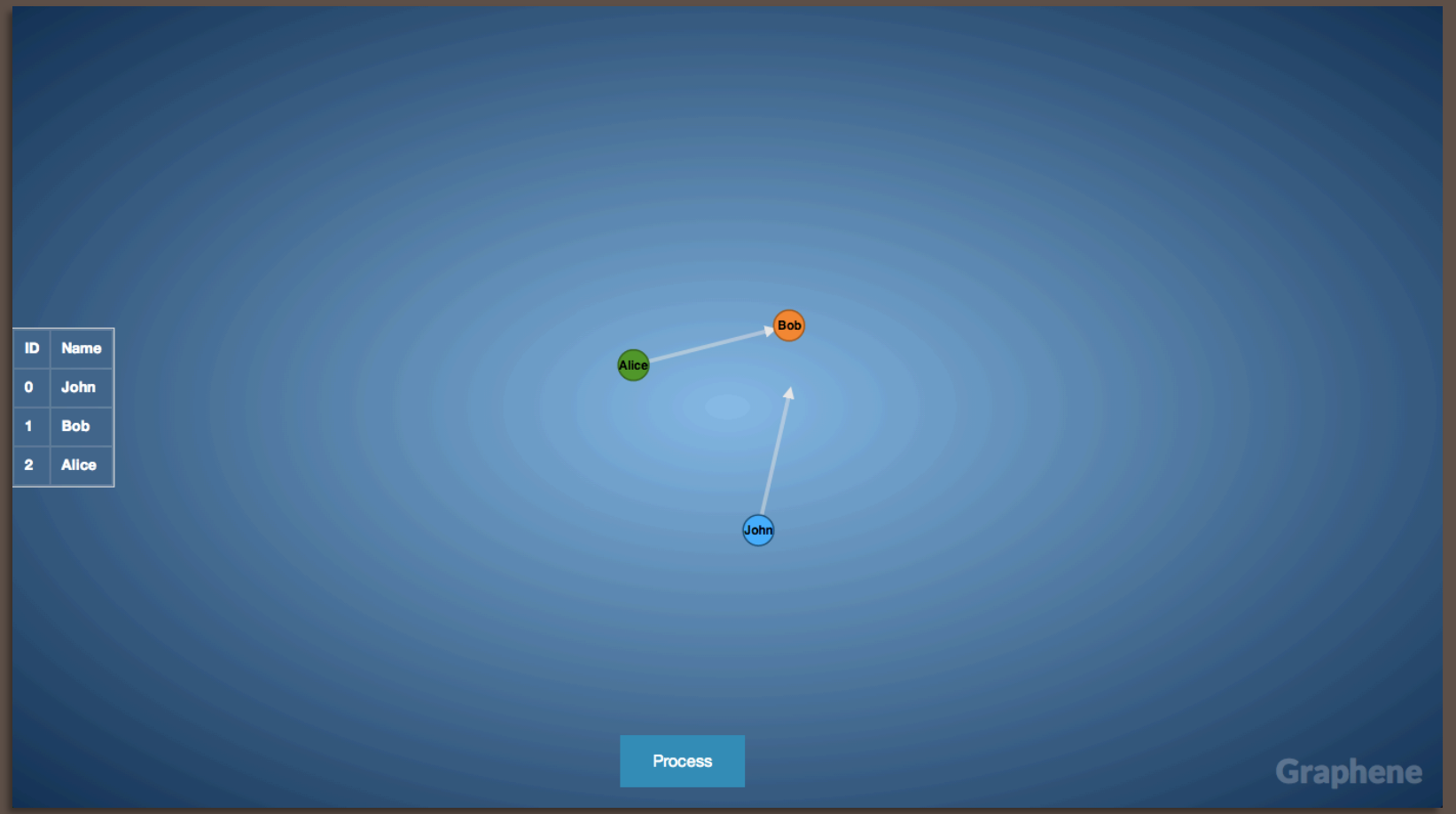
```
person has name,city,picture;
graph = input("person","data/0.edges");
```

- Or create a mesh and use edge/node operators

```
g = mesh(4, d);

g -= :1;
```

graphene>
write();

- Or, just ask the user nicely to do it for you



graphene> write();

Add edges by lookup, by id

Removing nodes

Display the program state

Display a particular graph

Overlay the output with a nodes in a particular list of nodes as highlighted

```
graphObj += "name":"Bob" -> "name":"John";
graphObj -= :1 <-> :2;

graphObj -= "name":"Alice";

output();

output(g);

output().overlay(l);
```

graphene> write();

- Node and Edge operations

```
"key":"value"

:id /*returns Node*/

nodeObj.["key"]; /*returns value for key*/
```

```
graphObj.getAdjacent(node);                          /*returns list*/
graphObj.getNonAdjacent("key":"value");              /*returns list*/
graphObj.hasNode("key":"value");                     /*returns boolean*/
graphObj.hasNode(:id);                               /*default key is ID*/
graphObj.hasEdge("key":"value", "key":"value");      /*returns boolean*/
graphObj.hasEdge(:id, :id);                          /*default key is ID*/


graphObj.inDegree("key":"value");                    /*returns number*/
graphObj.outDegree("key":"value");                   /*returns number*/
graphObj.degree("key":"value");                      /*returns number*/

graphObj -= "key":"value";                           /*returns boolean, true on success. */

graphObj.getEdges().length();                        /*Edge count*/
graphObj.getNodes().length();                        /*Node count*/
```

graphene> write();

- User defined functions

```
def (x,y) => doMath
{
    a = x + y;
    b = x - y;
    c = x * y;
    d = x / y;
} => (a,b,c,d);
```

- Lambdas

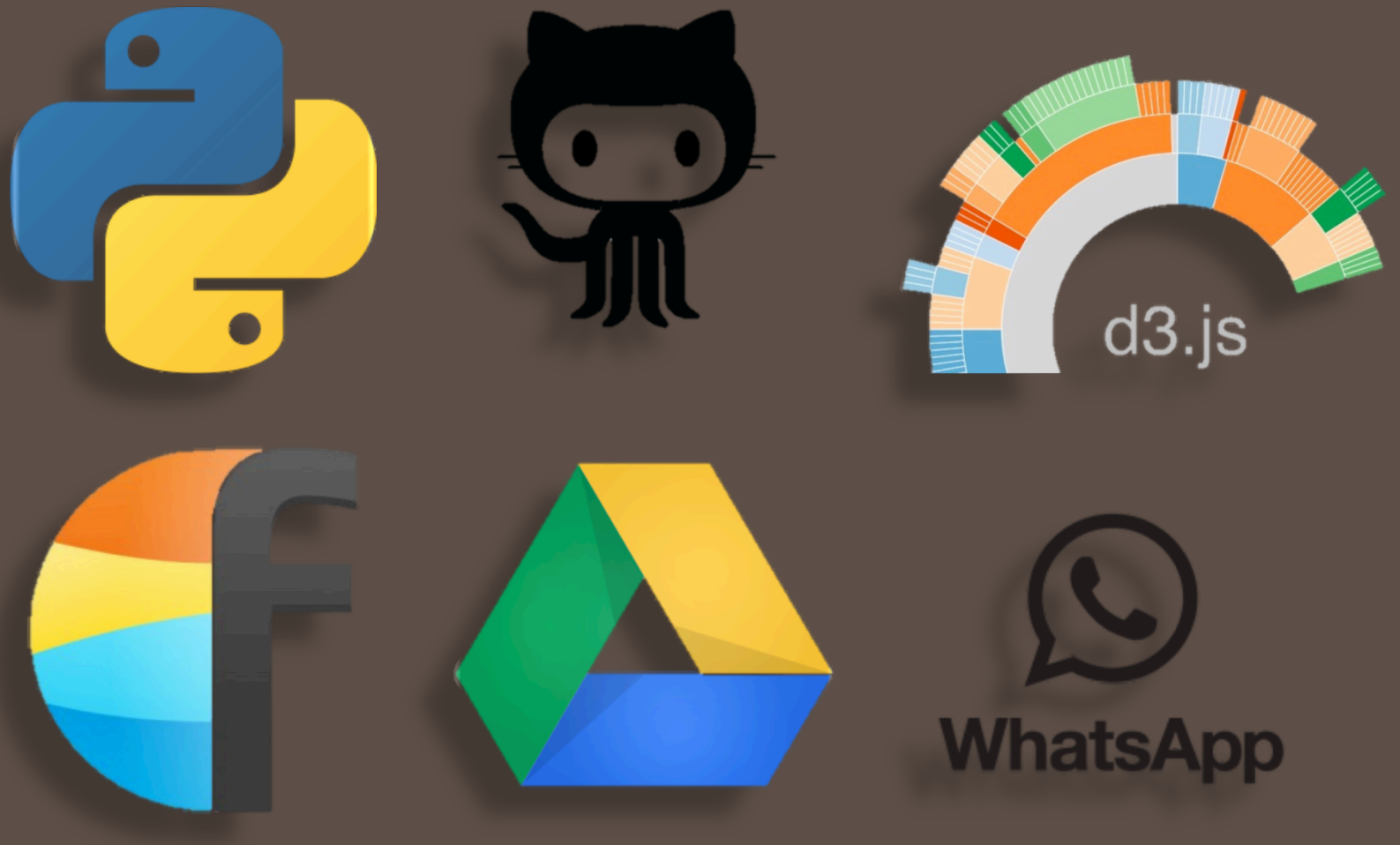
```
def (l,p) => applyLtoP
{
    result = l(p);
} => (result);

myLambda = lambda (p):{x = p*2;}=>(x);

computedValue = applyLtoP(myLambda, 5);

print(computedValue);
```

graphene>
demo();

graphene>
develop();

v/s

graphene>
cross_platform();



graphene> run();

- Bash Script to call the compiler in different modes
- Modes include debugging mode, regression testing mode.

Run with default arguments

```
$./graphene
```

Run with debugging information

```
$./graphene -d
```

Run with a file input

```
$./graphene -f hello.ene
```

Run the help manual

```
$./graphene -h
```

graphene> run();

Run the test with default arguments

```
$./graphene test
```

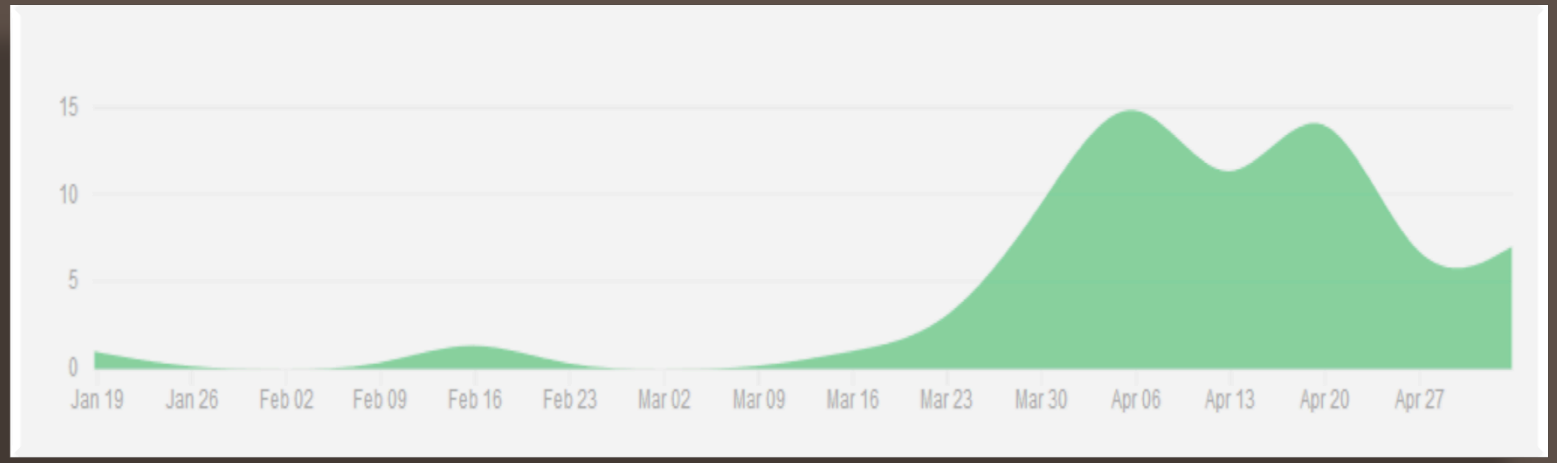
Run the tests independently

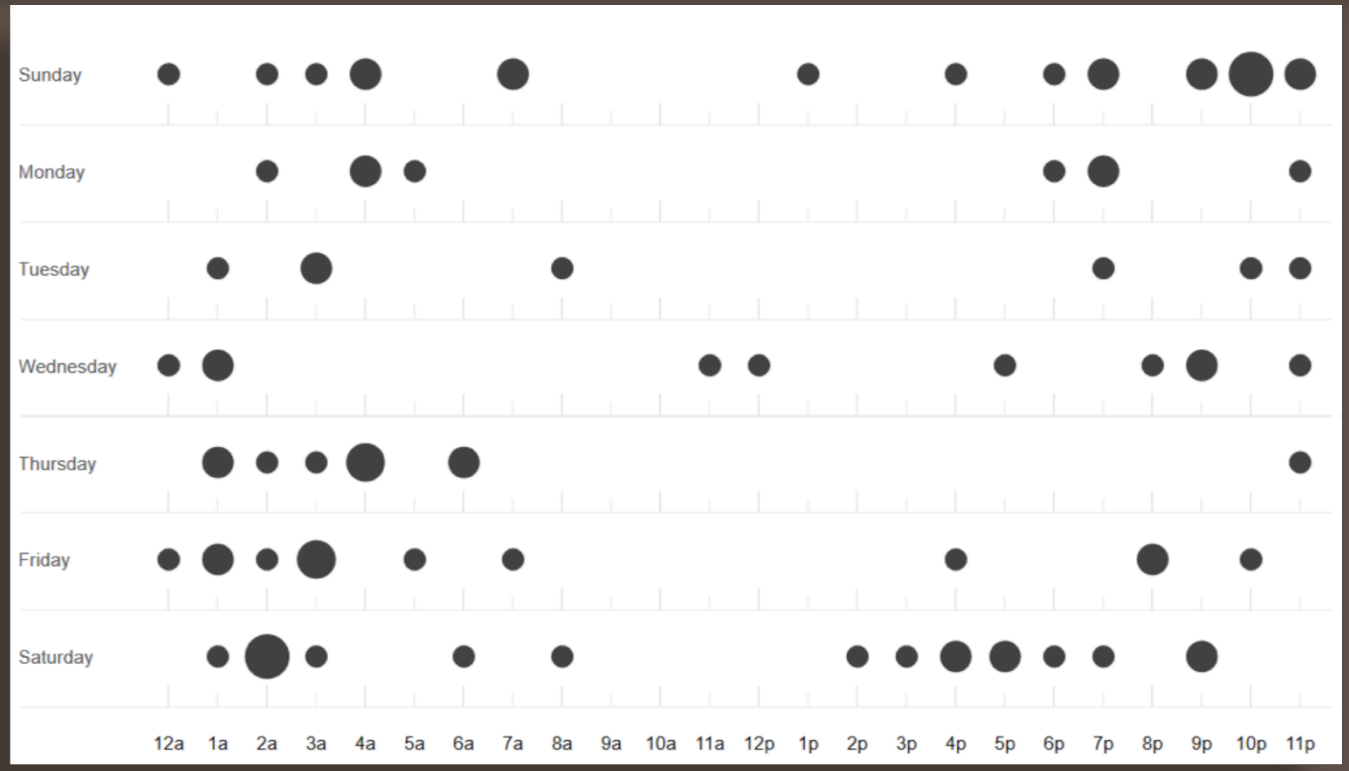
```
$./graphene test -i
```

Run test number 23

```
$./graphene test -t 23
```

graphene> manage();





graphene>
test();

graphene> test();

- Testing framework included tests which consists of the test code and expected output.
- Extensive tests to check all the basic as well as new constructs.
- Basic constructs include arithmetic, logical, control flow, lists, functions, error handling
- New constructs include lamda functions, node property, clustering.

graphene> exit();

- What worked:
  - Impeccable rigor in design,implementation and testing standards
  - Learning to be proficient at git and using it as more than a dumb remote disk
- What we could have improved
  - Better definition of scope early on
  - Better prioritization of feature implementations
- For more details:
  - https://github.com/Adith/Graphene

Graphene