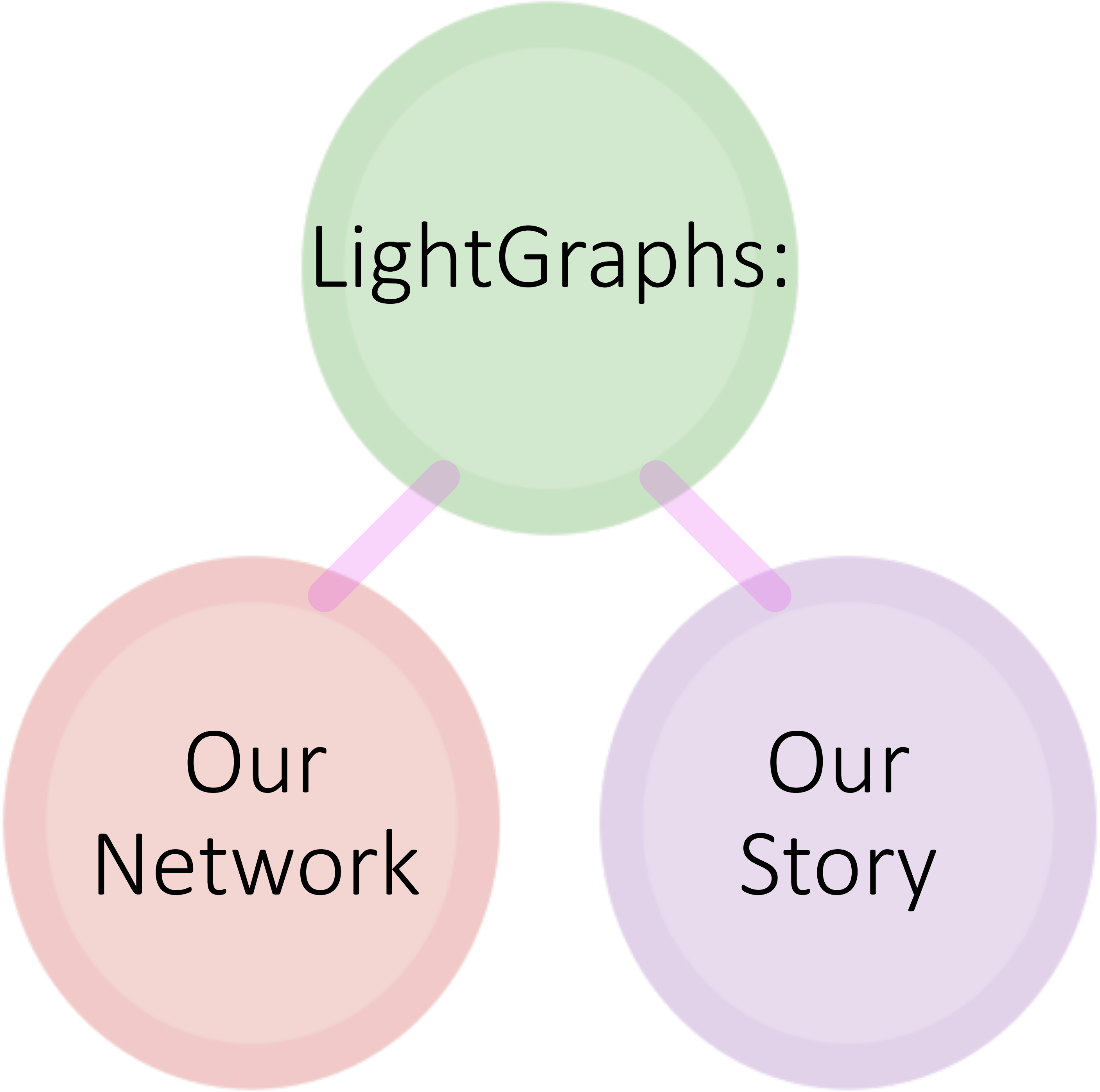
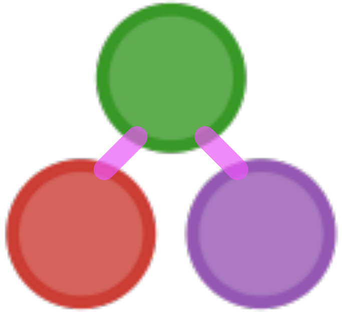


James Fairbanks, GTRI
Seth Bromberger, LLNL

About Seth

- Security researcher focused on critical infrastructure
- Looking at ways to combine graph analytics and machine learning to solve cybersecurity problems
- NOT A MATHEMATICIAN



About James

- Research Engineer focusing on online media and cybersecurity
- Looking at ways to combine graph analytics and machine learning to solve cybersecurity problems
- Used LightGraphs to study numerical accuracy requirements of spectral clustering

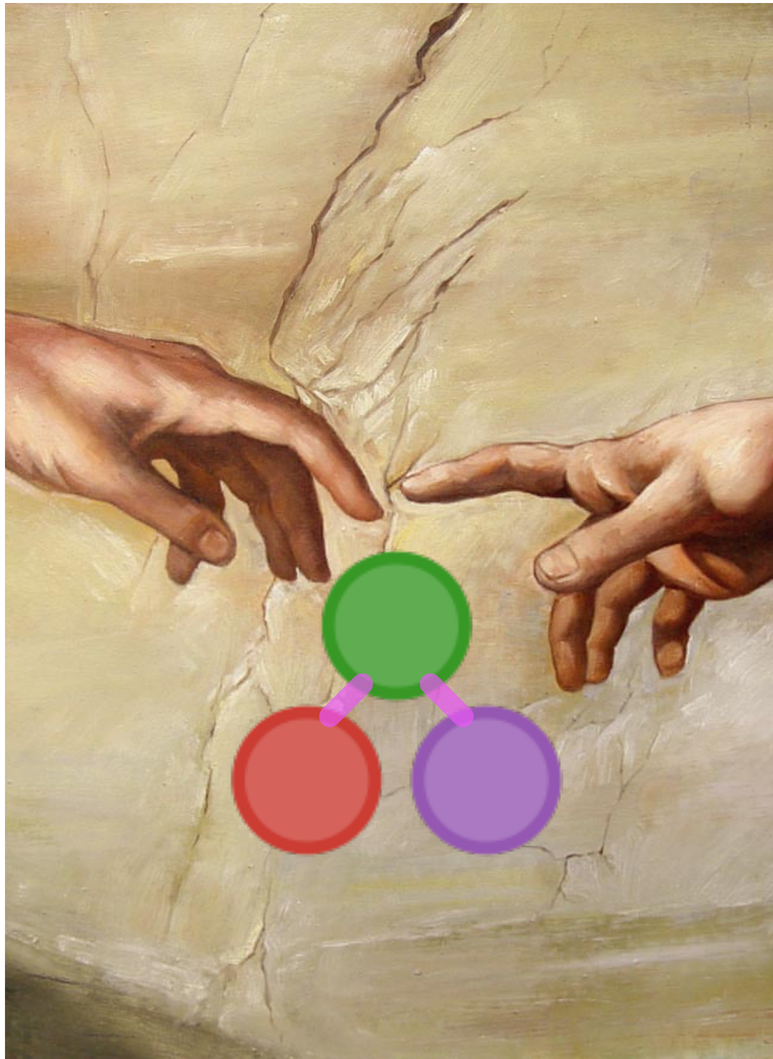
- A MATHEMATICIAN

Why Should We Care About Graphs?

- Uses of graphs in computer science:
  - Syntax Trees, Markov Chains, State Machines, Scheduling DAGs, …
- Turns out that graphs are everywhere!
- We focused on graph analysis:
  - Social media, cybersecurity, grid modeling (energy, transport, …)

In the beginning….

- Consulting for a client who wants to analyze activity logs
- Graph representation of activity solves a pressing problem
- *Graphs.jl* looks great. Let's use it!



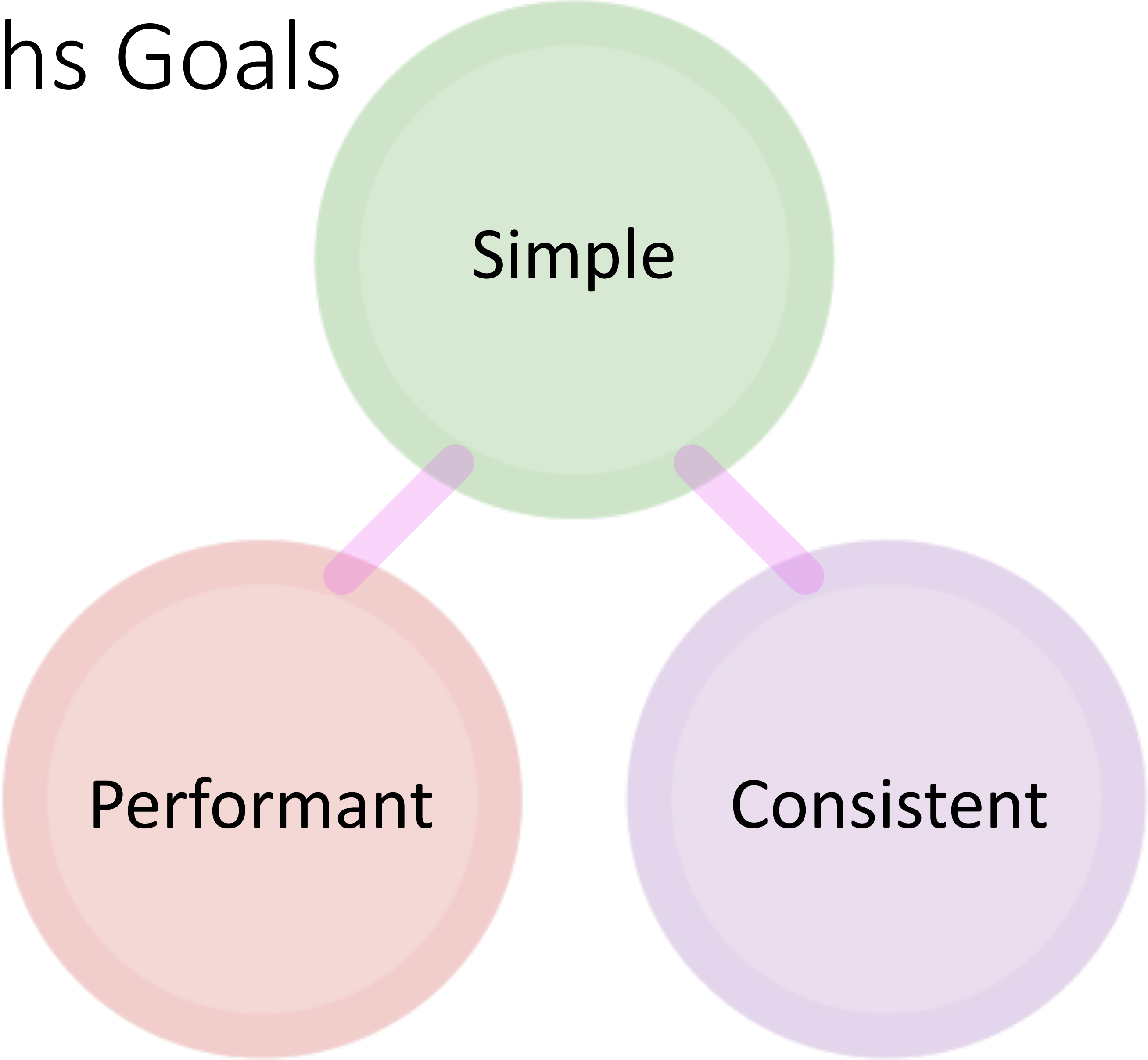
Graph Factory vs Graph Library

- Generic Interfaces
- Basic interface
- Vertex List interface
- Edge List interface
- Vertex Map interface
- Edge Map interface
- Adjacency List interface
- Incidence List interface
- Bidirectional Incidence List interface

NetworkX

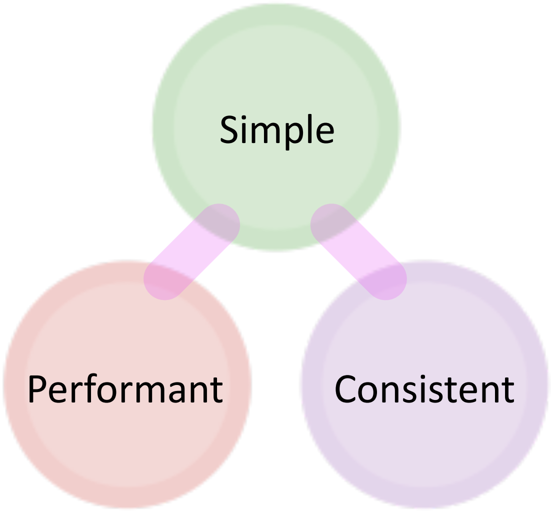
- Simple to use
- 1 language solution
- Lots of features and analysis for complex networks
- Dictionary of Dictionaries
- Just too slow

LightGraphs Goals



Design Goals

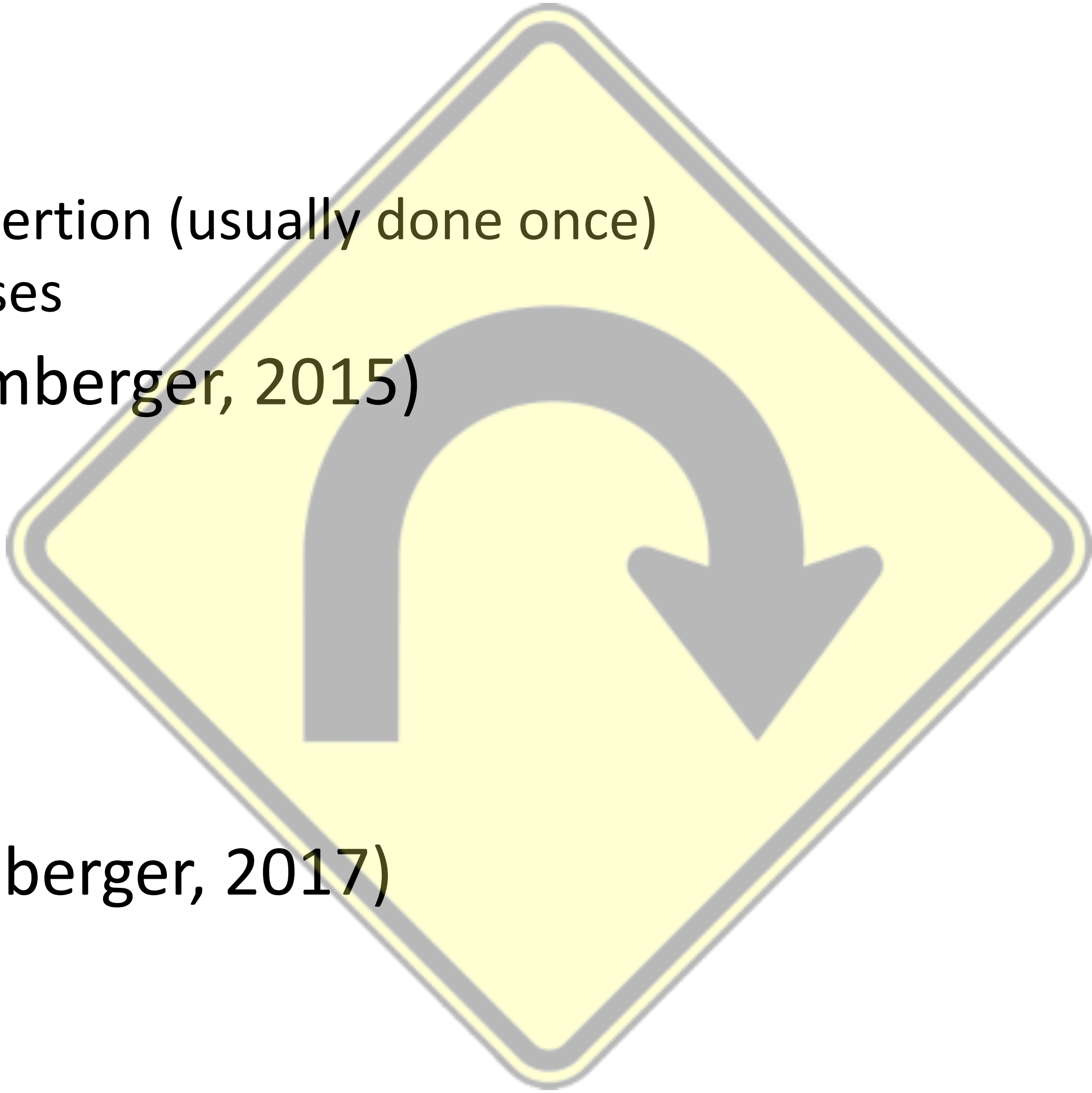
- Everything's a tradeoff
  - Adjacency lists vs Sparse Matrices vs Dense Matrices vs….
  - Vertex / Edge metadata?
  - Vertex indexing?
  - Edge sets? Edge iterators?



- Guides every decision we make.

Sometimes we change direction

- Adjacency lists: now sorted
  - Cost increase for graph creation / edge insertion (usually done once)
  - Cost advantage for all random edge accesses
- “Parameterization is the devil” (@sbromberger, 2015)
  - Complexity increase
  - But:
    - memory savings for most graphs
    - flexibility for new graph types
    - forced us to define an interface
- “Parameterize all the things!” (@sbromberger, 2017)

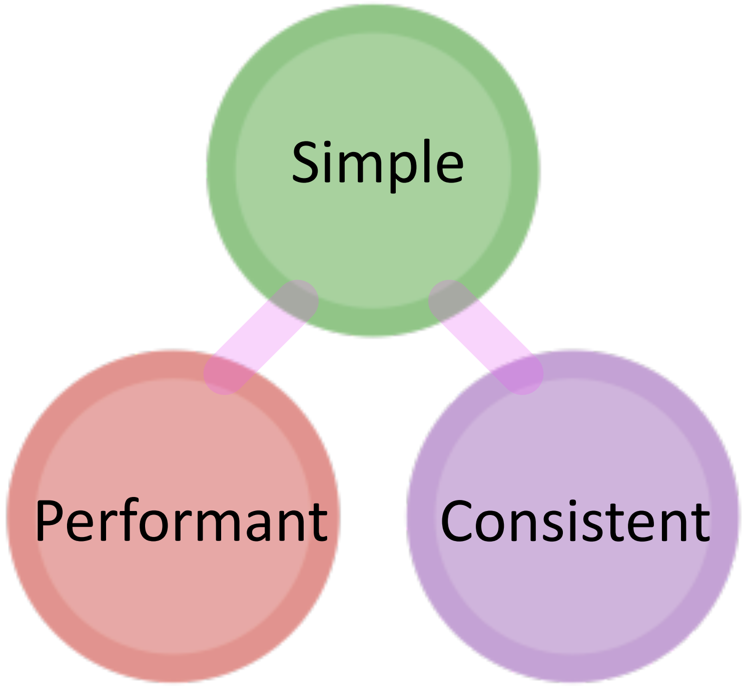


Example Design Tradeoff: Edge Sets

- Originally, we used `Set{Edge}` to provide $O(1)$ edge lookup
- $O(1)$ lookup is beneficial in some cases, but leads to
  - increased memory usage
  - slow edge insertion
- Dropping this feature halved the memory usage of graphs, at the expense of $O(\log n)$ edge lookup.
  - Users can still produce their own edge indices to accelerate lookup
  - Edge insertion is still faster, even with sorted adjacency lists

Reaping the rewards of Julian design

- We are all figuring out what idiomatic *Julian* design means together
- We take advantage of types and multiple dispatch to achieve this design



Advantages of Simplicity

- One language: easy to develop
- Fixed data structures: simple reasoning about performance
- No metadata: simple to understand and use

```
mutable struct SimpleGraph{T<:Integer} <: AbstractSimpleGraph
    ne::Int
    fadjlist::Vector{Vector{T}} # [src]: (dst, dst, dst)
end
```

Performance Benchmarks

- Graph memory: $\text{sizeof}(Int) + (|V| + 1)h + 2|E|\ \text{sizeof}(T)$
- DiGraphs: $\text{sizeof}(Int) + 2(|V| + 1)h + 2|E|\ \text{sizeof}(T)$

Test	LightGraphs	NetworkX	igraph	graph-tool
G1 = Erdos-Renyi (10k, 0.1) (s)	7.13	19	2.65	19.3
G2 = Barabassi-Albert (10k, 400) (s)	2.89	13.8	3.6	10.1
Betweenness (G2[1:3000]) (s)	4.02	DNF	6.77	3.34
Closeness (G2, s)	35.79	DNF	82	44.2
PageRank (directed G2, ms)	28.20	5 130	75.8	30.2
Local Clustering Coefficient (G2, ms)	255.53	37 400	167	270

Edge iterators use standard Julia interfaces

- We use the iterator interface `start, next, done` in order to provide an iterator over edges

```
for i in vertices(g)
    for j in neighbors(g, i)
        produce(i, j)
    end
end
```

- This leverages idiomatic Julia features to improve the readability of code.
- Encourages **“just write the loop”** programming style instead of bulk operations with optimized primitives

```
for e in edges(g)
    do work on e
end
```

GraphMatrices: Encoding Math Errors into the Type System

- For spectral graph theory you have to manage various "Graph Matrices"
  - {Combinatorial, Normalized, Stochastic, Averaging} {Adjacency, Laplacian}
- Math errors are tricky because they don't crash the code
- Compiler/Type Errors crash the code
- A "Matrix" type is too broad
- Encoding math into the type system improves code verification and validation

Types and Dispatch lead to improved generalizability

- GraphMatrices.jl was written for SparseMatrixCSC and then extended to support storing the graph as a LG graph.

- You can compute the eigenvalues of a Graph Laplacian without making a sparse matrix copy.

- Reduces memory overhead by a factor of 2

```
-type CombinatorialAdjacency{T} <: Adjacency{T}
-       A::SparseMatrix{T}
-       D::Vector{T}
 end

 function CombinatorialAdjacency{T}(A::SparseMatrix{T})
        D = vec(sum(A,1))
-       return CombinatorialAdjacency(A,D)
 end
```

```
+type CombinatorialAdjacency{T,S,V} <: Adjacency{T}
+       A::S
+       D::V
 end

 function CombinatorialAdjacency{T}(A::SparseMatrix{T})
        D = vec(sum(A,1))
+       return CombinatorialAdjacency{T,SparseMatrix{T},typeof(D)}(A,D)
 end
```

Abstraction Redux

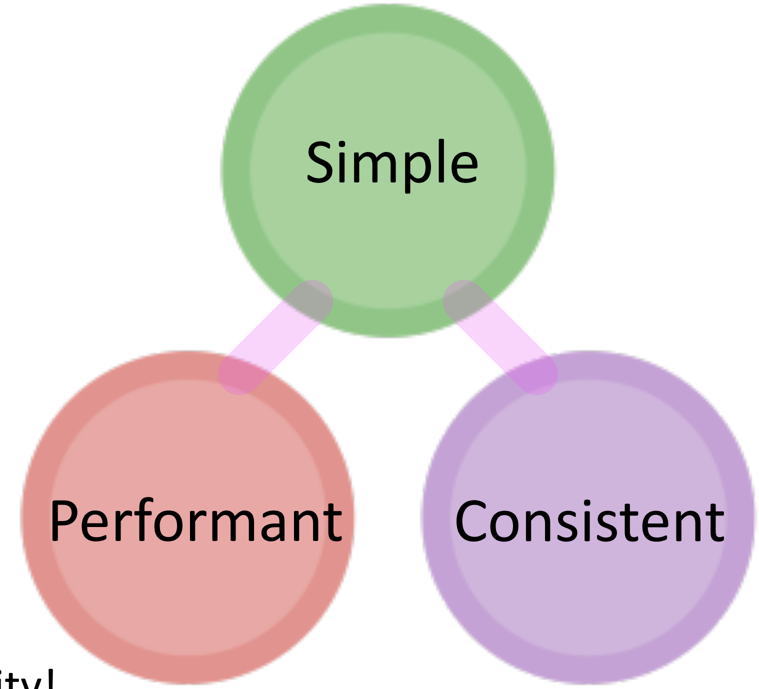
- Introduced `AbstractGraph` to allow more experimentation
- Allows graphs that store metadata inside or outside of edges
- Provides flexibility for Out-of-core / Parallel computation
- Look to DifferentialEquations.jl and JuMP for inspiration on design
- [Weighted Graphs: LightGraphs.jl/pull/663]()

GSOC 2017

- Welcome Divyansh!
- Focus on parallelizing expensive graph algorithms
- To date: betweenness centrality, closeness centrality, and Dijkstra shortest paths
- More planned

Why *you* should be using LightGraphs

- Single-language solution
- Active developer community
- Easy and fun to use



Thanks to all contributors and the whole Julia community!