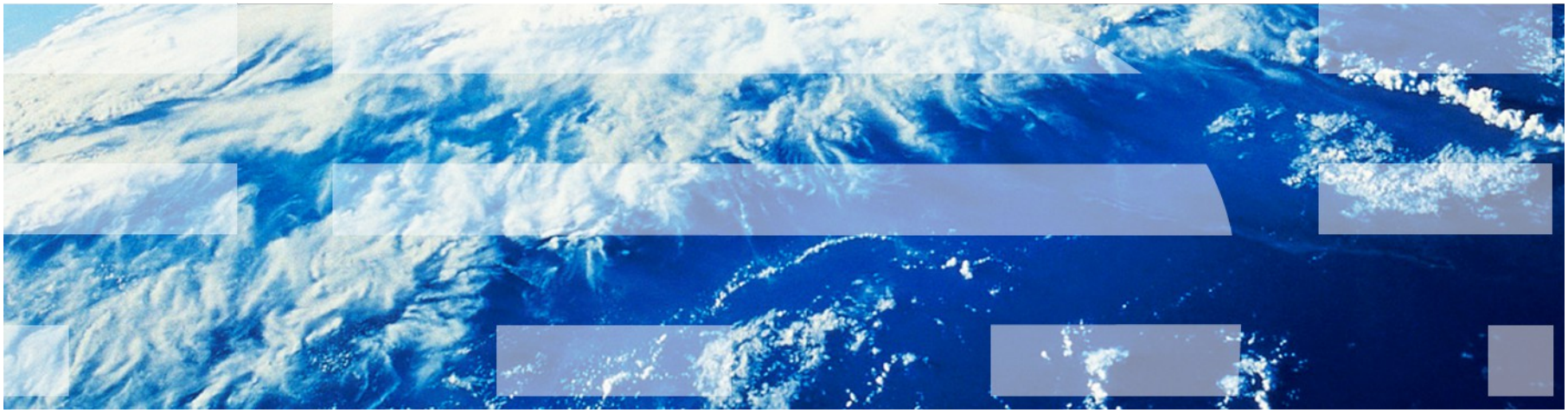
Property Graphs for Industry Solution at IBM

Yinglong Xia

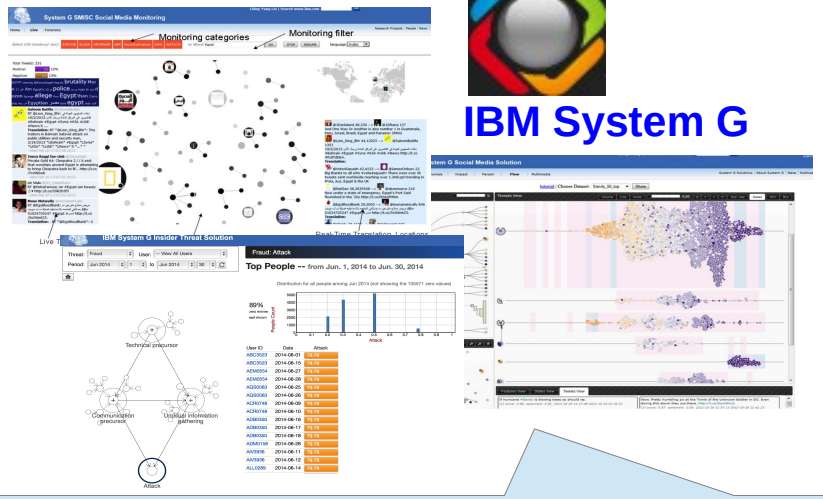
IBM T.J. Watson Research Center

November 14th, 2014

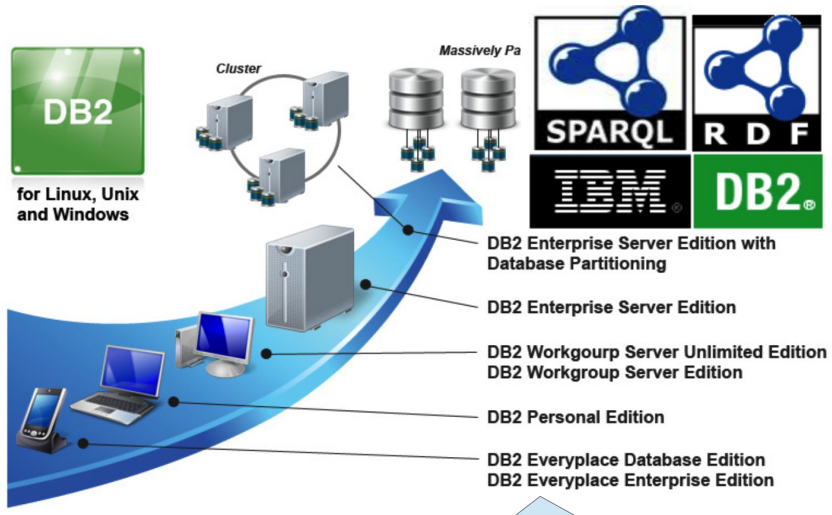


IBM Research

Glance of IBM Research's Efforts on Graph Technology



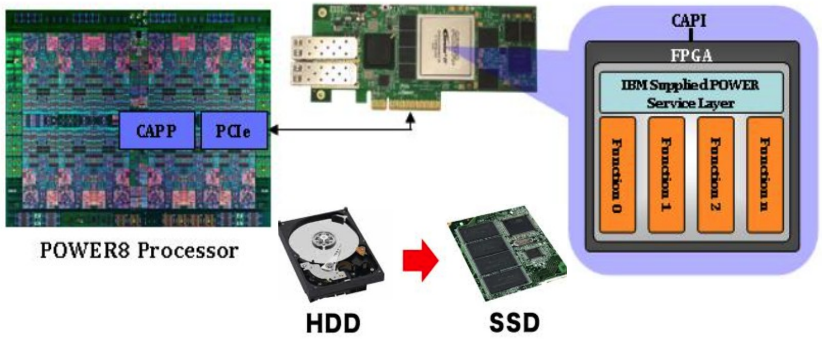
IBM System G



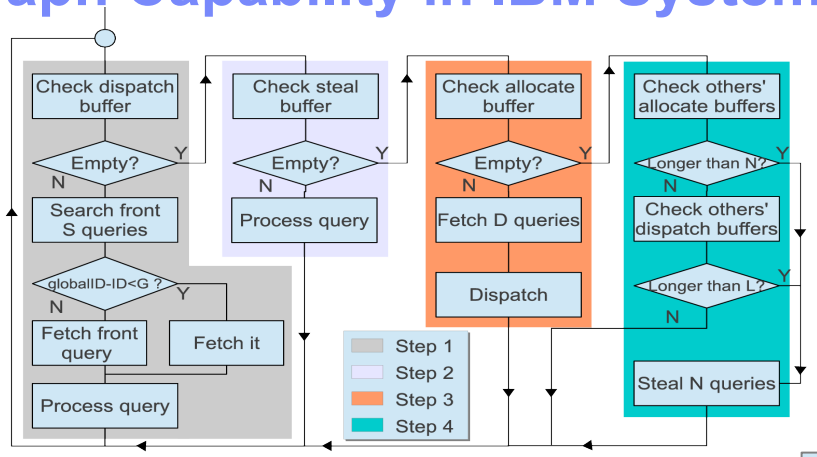
- **Industry solution drives graph research**
  - DARPA Anomaly Detection at Multiple Scales (ADAMS) project for insider threat detecting espionage, sabotage, and internal fraud
  - DARPA Social media monitoring (SMISC) for Enterprise live monitoring, thread detection, info flow analysis, and social media analysis

- **Enhancement of IM systems**
  - DB2 RDF as a triple store
  - DB2 Graph to support property graph query

- **Emerging Hardware / Accelerator**
  - CAPI for coherent interface
  - Active store and hardware accelerators



Graph Capability in IBM System G – Native Store



Graph primitives

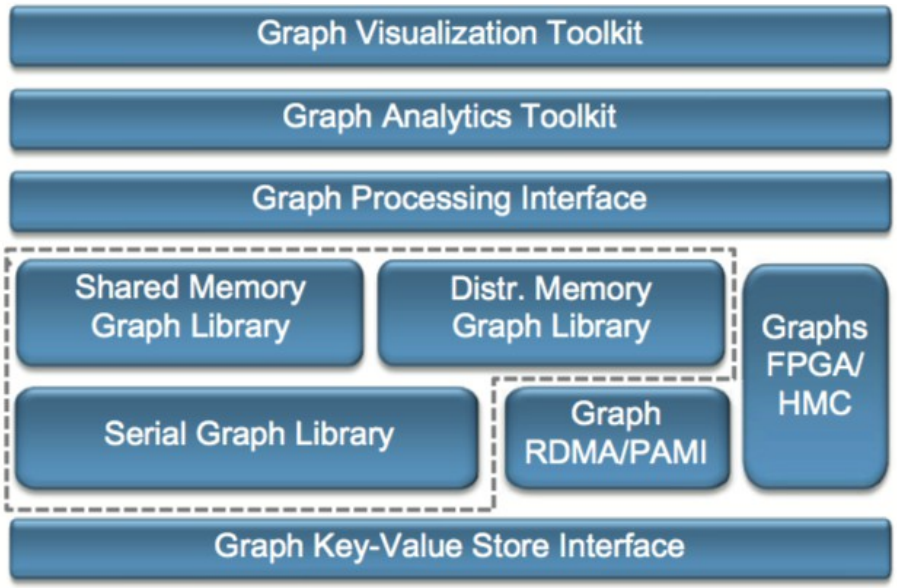
User defined property class

```
template <class VertexProperty,
          class EdgeProperty,
          class DIRECTEDNESS,
          class Traits>
class Graph {
  typedef ... vertex_descriptor;
  typedef ... edge_descriptor;
  typedef ... vertex_iterator;
  typedef ... edge_iterator;
  vertex_descriptor add_vertex(VertexProperty&);
  edge_descriptor add_edge(vertex_descriptor v1,
                           vertex_descriptor v2,
                           edge_property&);
  vertex_iterator find_vertex(vertex_descriptor);
}

typedef Graph<int,double,DIRECTED> graph1_type;

class my_vertex_property {...}
typedef Graph<my_vertex_property, double,
              UNDIRECTED> user_graph2_type;

template <class G>
process_vertex(G& g,vertex_descriptor vid){
  vertex_iterator vit = g.find_vertex(vid);
  //process vertex property
  // vit->property();
  edge_iterator eit = vit->edges_begin();
  for (;eit != vit->edges_end();++eit){
    //process edge identified by eit
    // eit->target(); eit->property();...
  }
}
```

Graph traversal

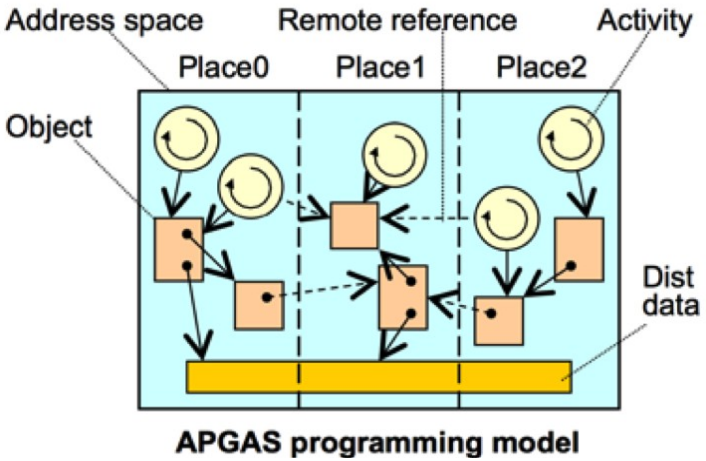
- **Native Store**
  - Performance oriented
  - Highly efficient graph access
  - Scale **up** & out

REST	SPARQL*	Gremlin
socket	TinkerPop/Blueprints	
gShell commands	JNI	
multiproperty_graph		
ibmppl_graph		
Graph KV Store		

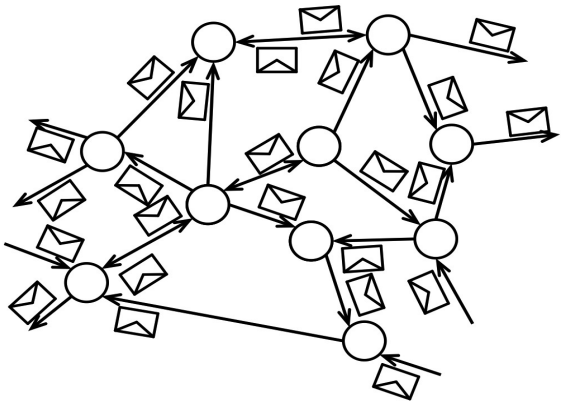


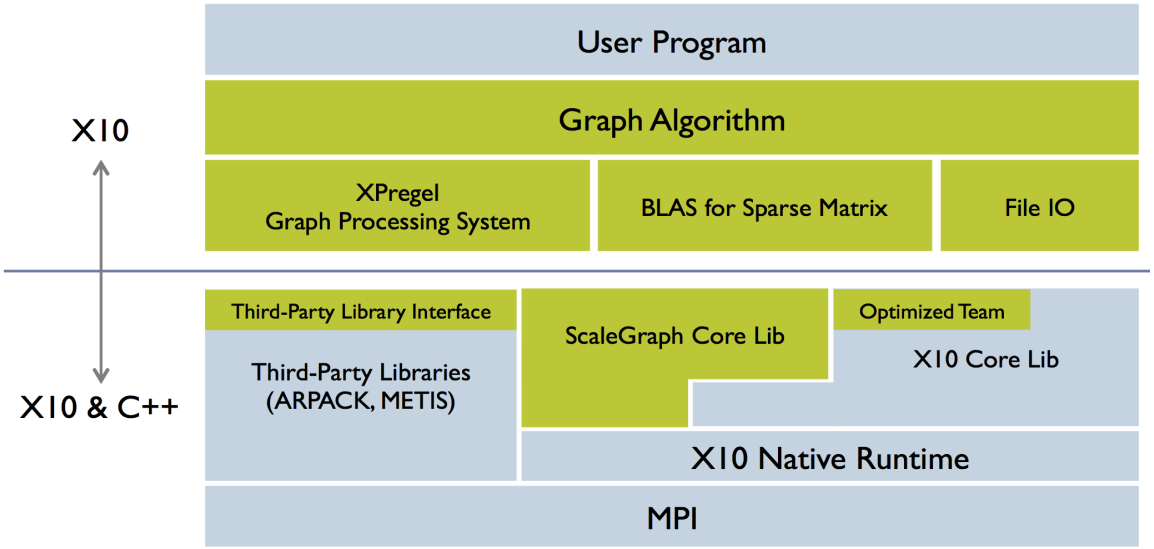
Graph Capability in IBM System G – ScaleGraph

- IBM X10 Programming Language
  - Distributed functional language
  - Aim at productivity
  - Adopt APGAS programming model
  - Run as native code or on Java
  - Open sourced
- Graph computing model
  - XPregel model for distribution execution
  - SpVM model for graph analytics
- Performance optimization
  - Native MPI for comm.
  - Avoid serialization



APGAS programming model





DB2Graph

- Building property graph store over relational DB
  - Try to translate a gremlin query into a single SQL
  - Optimize graph update operations
  - Support batch loading of property graphs

- Shredding table
  - Without index: 46 - 230 ms per property query
  - With index: 3 – 71 ms per property query
- DB2XML
  - Without index: 100 - 2000 ms per property query
  - With index: 4 – 1000 ms per property query
- **DB2JSON**
  - Without index: 247 - 262 ms per property query
  - With index: 2 – 50 ms per property query

Shredding table

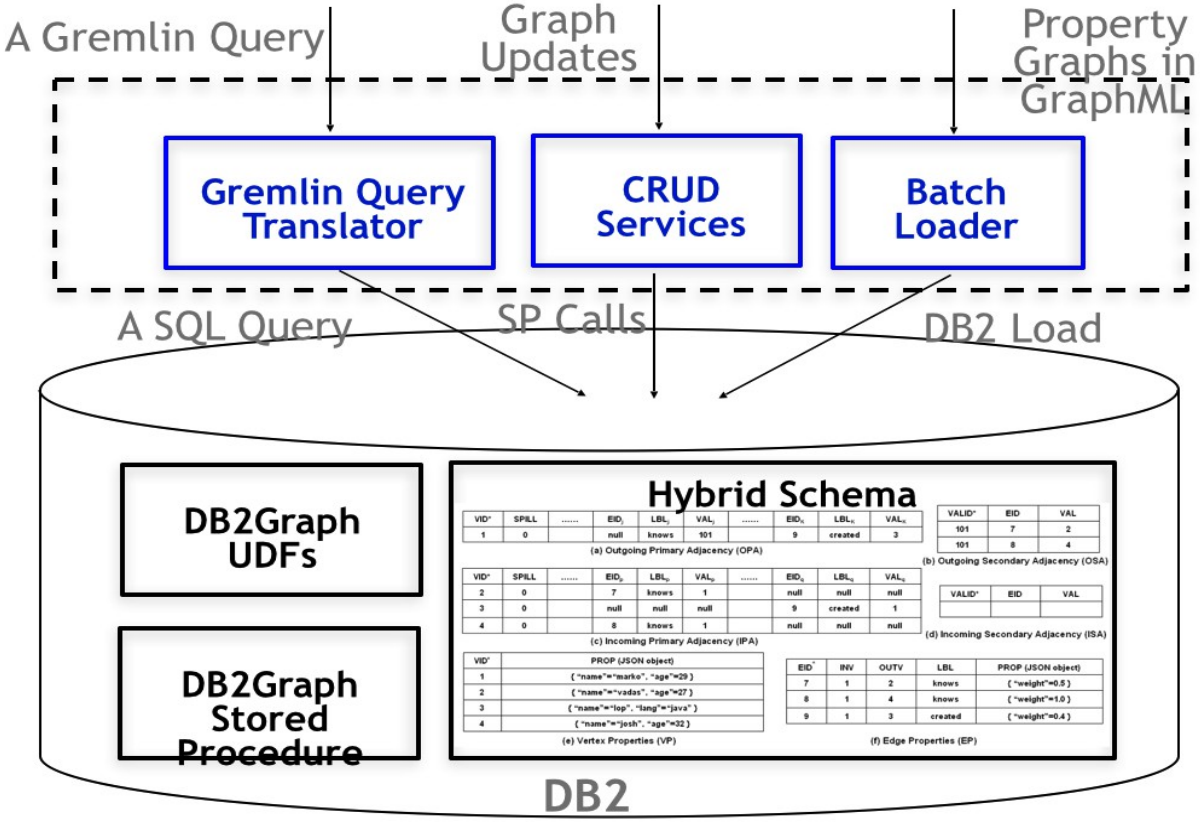
VID	.....	$PROP_i$	VAL_i	.....	$PROP_j$	VAL_j	.....
1		name	marko		age	29	

DB2XML

VID	DATA (XML)
1	<?xml version="1.0"?> <NodeProperty> <Property> <name>marko</name> </Property> <Property> <age>29</age> </Property> </NodeProperty>

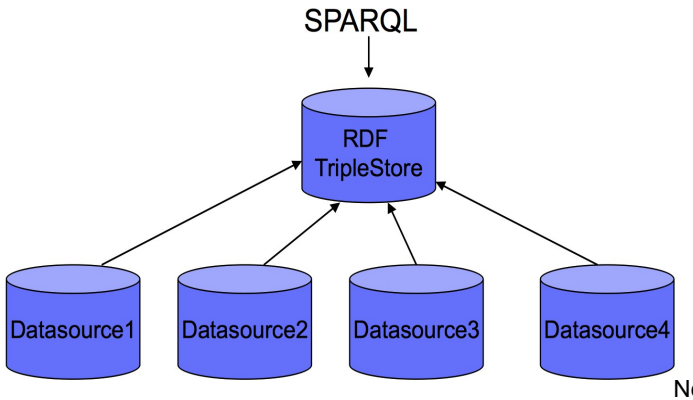
DB2JSON

VID	PROPS (JSON)
1	{ "name" : "marko", "age": 29, ... }



DB2RDF

- In DB2 LUW 10.1 we support
  - Java API's for RDF application consumers.
  - HTTP based SPARQL query
- DB2RDF – A set of user tables within a database schema that stores the RDF data
  - Direct Primary : stores the triples and the graph they belong to indexed by subject.
  - Direct Secondary : stores the RDF objects that share the same subject and predicate within a RDF graph.
  - Reverse Primary : stores the triples and the graph they belong to indexed by object.
  - Reverse Secondary : stores the RDF subjects that share the same object and predicate within a RDF graph.



Direct Primary

Subject	Graph		predicate	value1		predicate	value4		predicate10	value10		predicate4	value40
s1			pA	vA					pB	vB		pC	lid:abc
s2						pF	vF		pB	lid:cdf			

Direct Secondary

Graph	listId	Value
	lid:abc	v1
	lid:abc	v2
	lid:cdf	v3
	lid:cdf	v4

Reverse Primary

Object	Graph		predicate	sub1		predicate	sub4		predicate10	sub10		predicate4	sub40
vA			pA	s1									
vF						pF	s2						

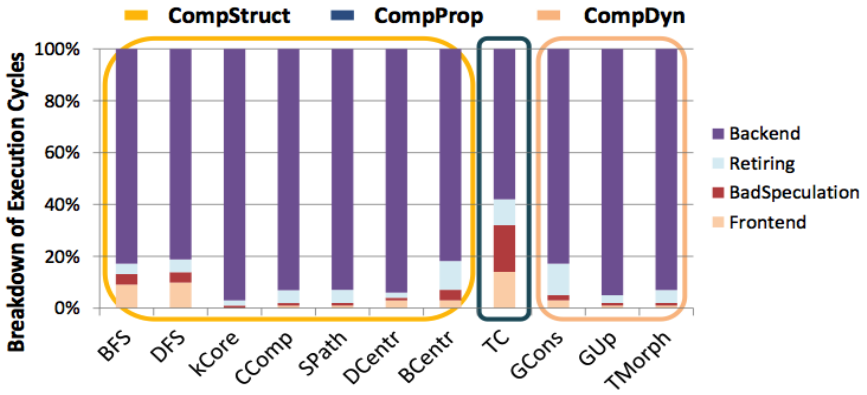
Reverse Secondary

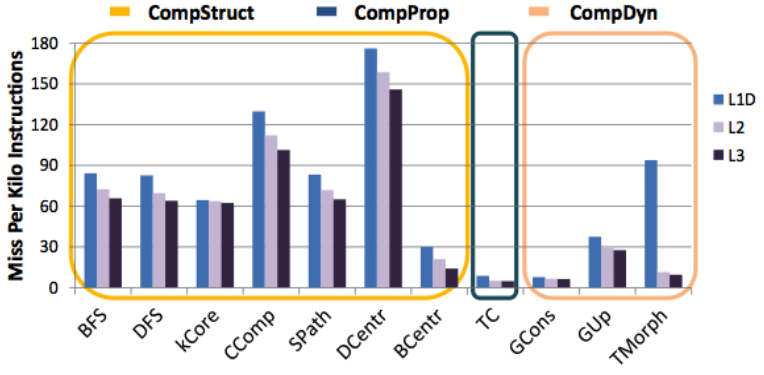
Graph	listId	Value

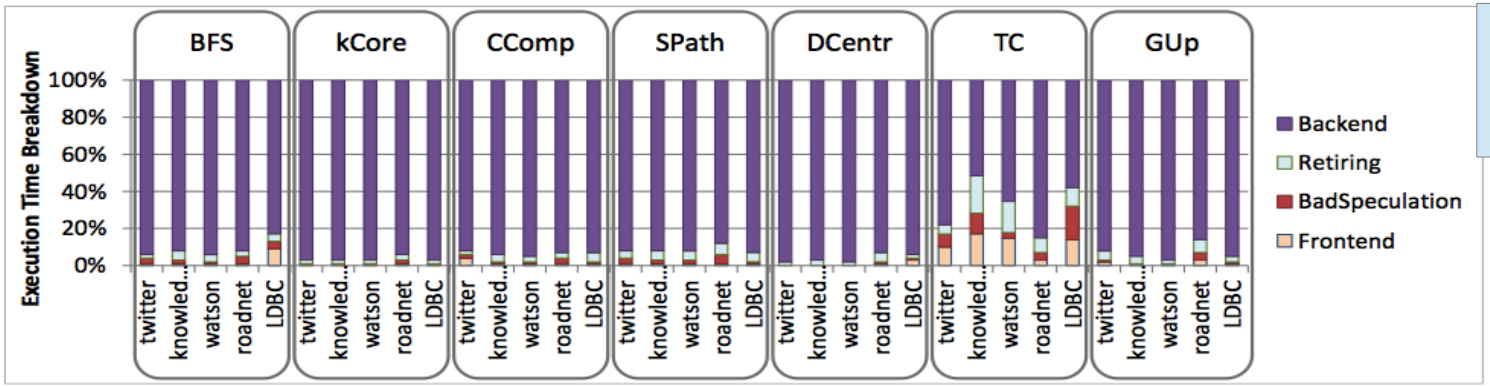
Graph Benchmark for Innovations in Hardware

Experiment Data Set	Vertex #	Edge #
LDBC Synthetic Social Graph Data Set	1K	29.7K
	10K	296.1K
	100K	2887.7K
	1000K	28.82M
Twitter Graph	13M	60M
IBM Knowledge Repo	154K	1.72M
Watson Gene Graph	2M	12.2M
CA Road Network	1.9M	2.8M

- **Graph for benchmarking H/W**
  - Traditional processor/memory design is not friendly to graph
  - Scale **up** & out limitation
  - Demanded by novel H/W design







Performance profiling and breakdown wrt various graph algorithms`

Thank you