Building *In-memory Graph Store for Fast and Concurrent Querying with Advanced Hardware Features*

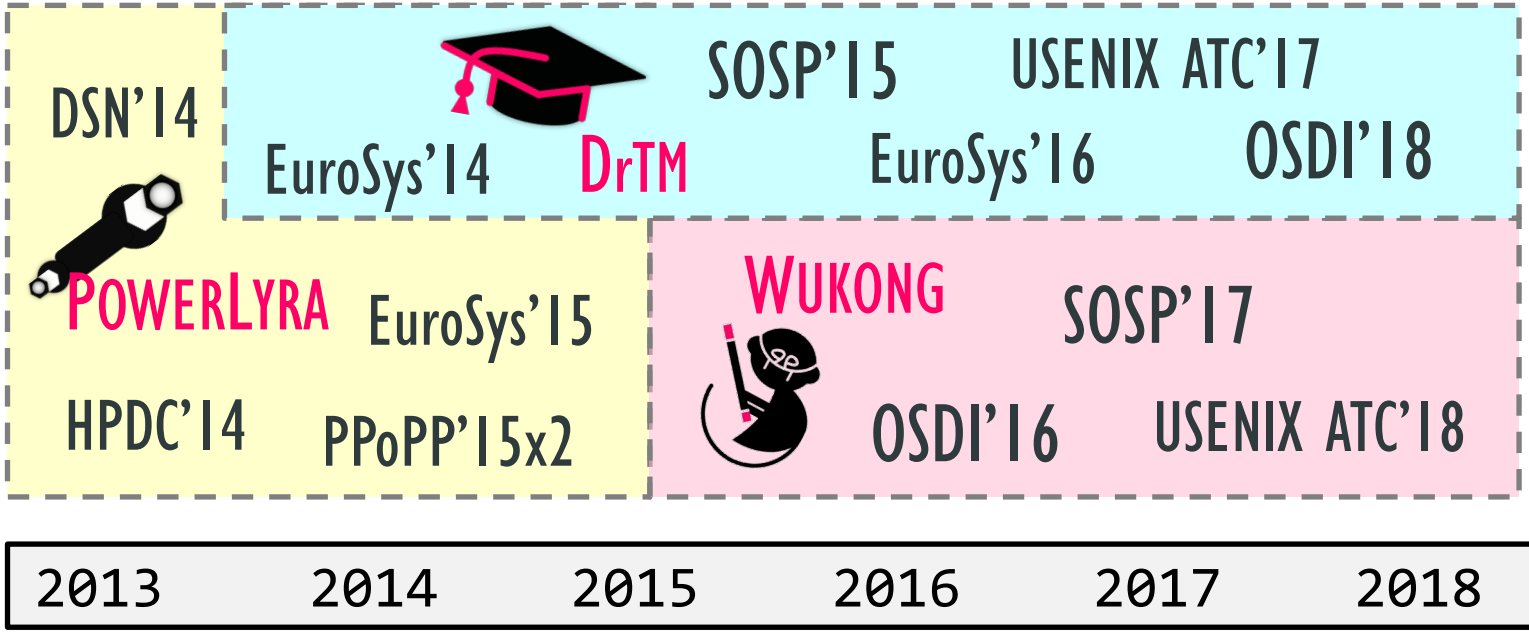
RONG CHEN

IPADS, Shanghai Jiao Tong University

IPADS Workshop, 2018

Joint work w/ *Haibo, Jiaxin, Youyang, Yunhao, Youyang, Siyuan, Chang, Ning, Jiaqi, Xiating, Xuehan, Wenhao, and Zhenhan @IPDADS*

Research in DS Group @IPADS



WUKONG Project

WUKONG

SOSP'17

OSDI'16

USENIX ATC'18

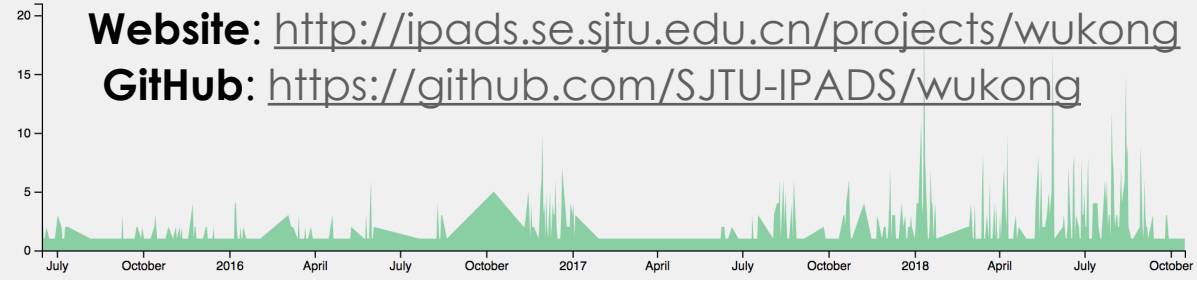
2015 2016 2017 2018

Alumnus

ACTIVE:
#Prof=1, #Students=8+

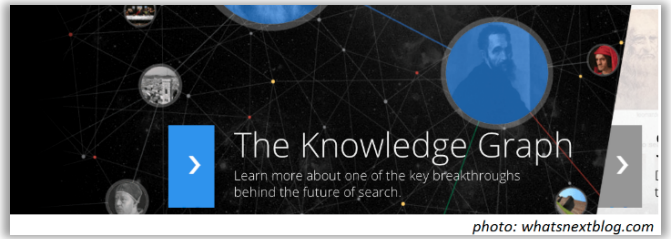
Website: http://ipads.se.sjtu.edu.cn/projects/wukong

GitHub: https://github.com/SJTU-IPADS/wukong



Graphs are Everywhere

Online ***graph query*** plays a vital role for searching, mining and reasoning linked data

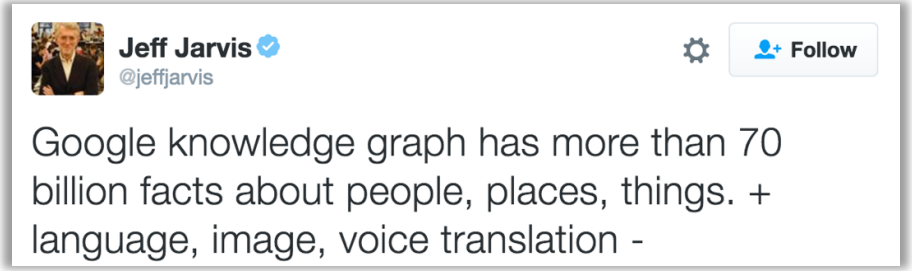


Jena

TAO

Unicorn

Knowledge Graphs and Querying



"…*Every piece of information that we crawl, index, or search is analyzed in the context of Knowledge Graph.*"

How a Database of the World's Knowledge Shapes Google's Future
MIT Technology Review, January 27, 2014

"…*Google users will able to browse through the company's '**knowledge graph**,' or its ever-expanding **database** of information **about 'entities'** – people, places and things – **the 'attributes' of those entities** and **how** different **entities are connected** to one another.*"

What Google's Search Changes Might Mean for You
Wall Street Journal, March 14, 2012

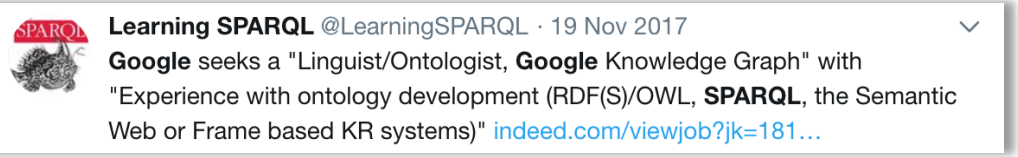
RDF and SPARQL

RDF: **R**esource **D**escription **F**ramework

- Public: *DBpedia*, *PubChemRDF*, *Bio2RDF*
- Google's *Knowledge Graph*, Bing's *Satori*, Facebook's, LinkedIn's, Yahoo's, . . .

SPARQL

- **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage



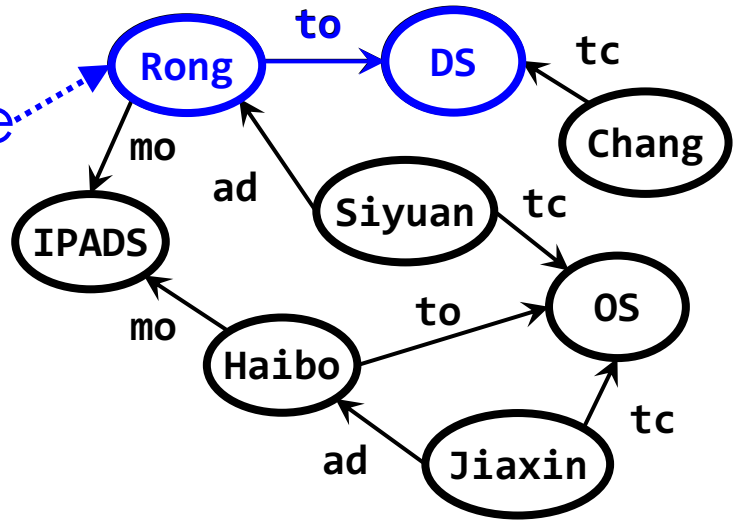
RDF and SPARQL

mo: MemberOf
ad: ADvisor
to: TeacherOf
tc: TakeCourse

RDF is a graph composed by a set of ⟨**S**ubject, **P**redicate, **O**bject⟩ triples

Rong	to	DS
Rong	mo	IPADS
Siyuan	ad	Rong
Siyuan	tc	OS
Haibo	to	OS
Haibo	mo	IPADS
Jiaxin	ad	Haibo
. . .		

triple



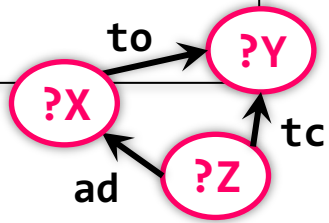
RDF and SPARQL

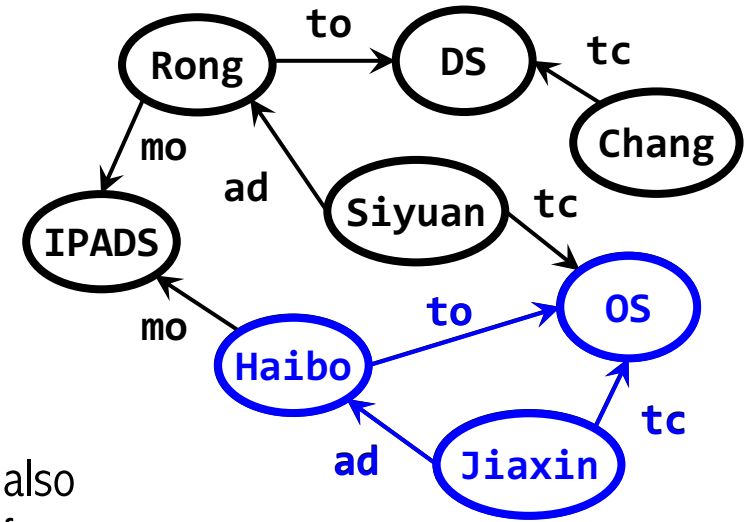
SPARQL is standard query language for RDF

```
SELECT ?X ?Y ?Z WHERE {
TP1   ?X teacherof ?Y .
TP2   ?Z takecourse ?Y .
TP3   ?Z adivsor ?X .
}
```

Triple Pattern

Variable





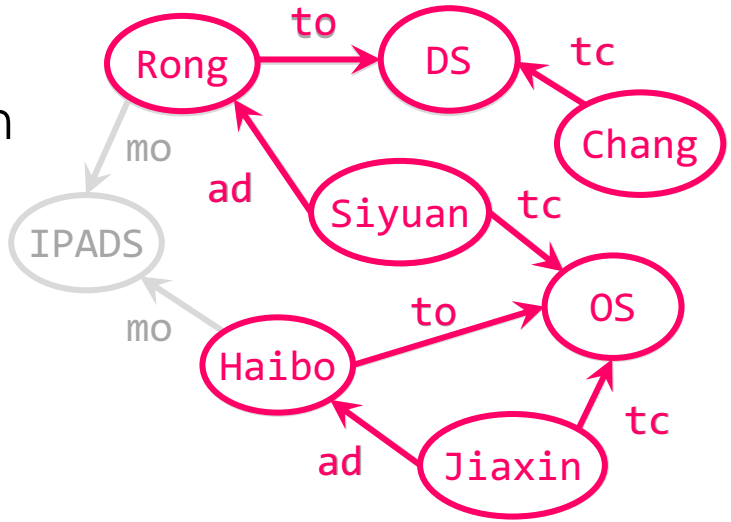
Professor (?X) advises (**ad**) student (?Z) who also takes (**tc**) a course (?Y) taught by (**tc**) the professor

Queries are Heterogeneous

Heavy Query (Q_H)

- Non-selective query
- Start from a set of vertices
- Explore a large part of graph

```
SELECT ?X ?Y ?Z WHERE {
 ?X teacherof ?Y .
 ?Z takecourse ?Y .
 ?Z adivsor ?X .      }
```



Queries are Heterogeneous

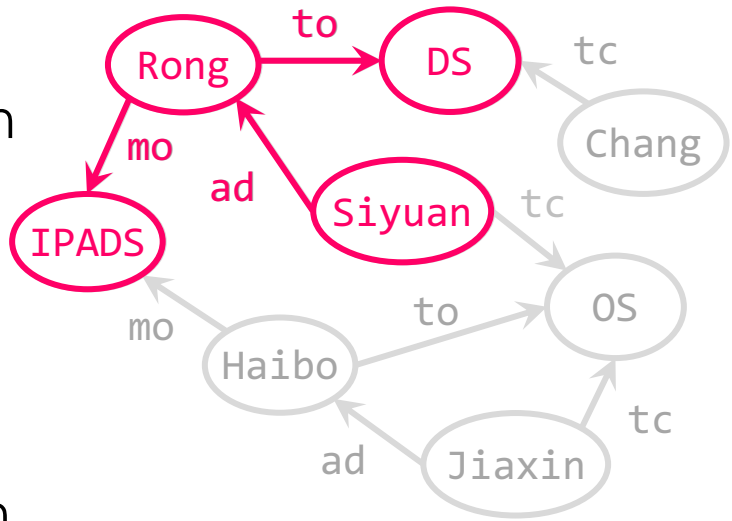
```
SELECT ?Y WHERE {
 ?X memberOf IPADS
 ?X teachOf DS .
 ?Y advisor ?X . }
```

Heavy Query (Q_H)

- Non-selective query
- Start from a set of vertices
- Explore a large part of graph

Light Query (Q_L)

- Selective query
- Start from a given vertex
- Explore a small part of graph

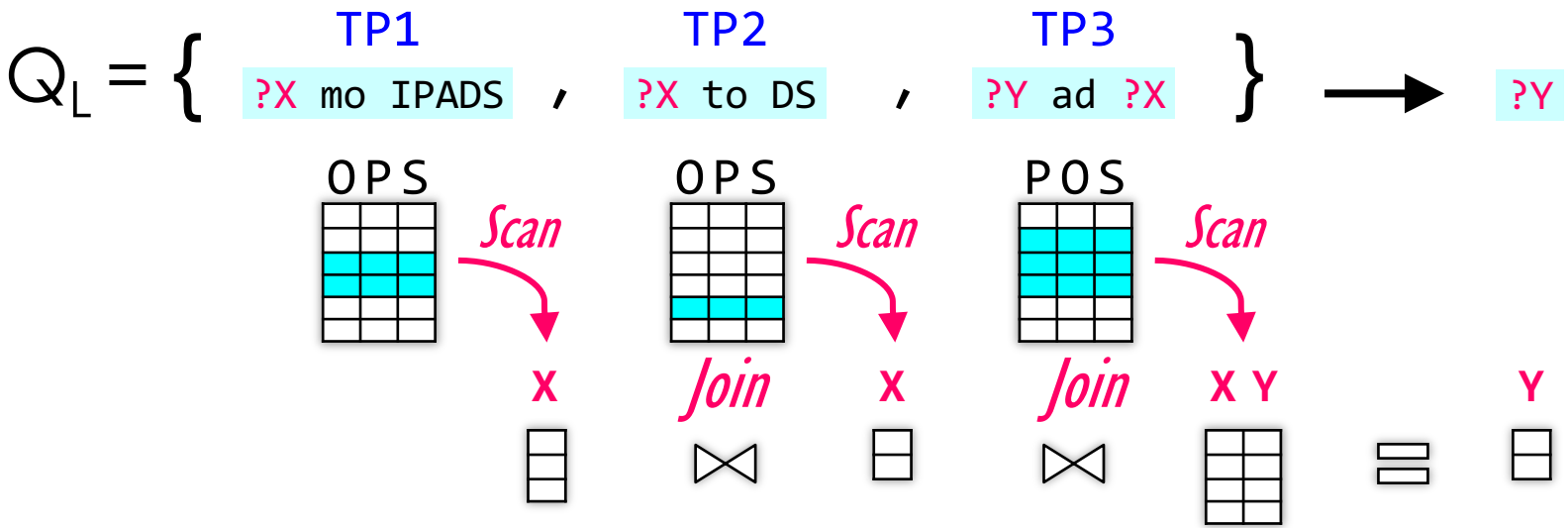


State of the Art

```
SELECT ?Y WHERE {
  TP1  ?X memberOf IPADS
  TP2  ?X teachOf DS .
  TP3  ?Y advisor ?X . }
```

Triple Store and Scan-Join

- Store RDF data as a set of triples in RDBMS

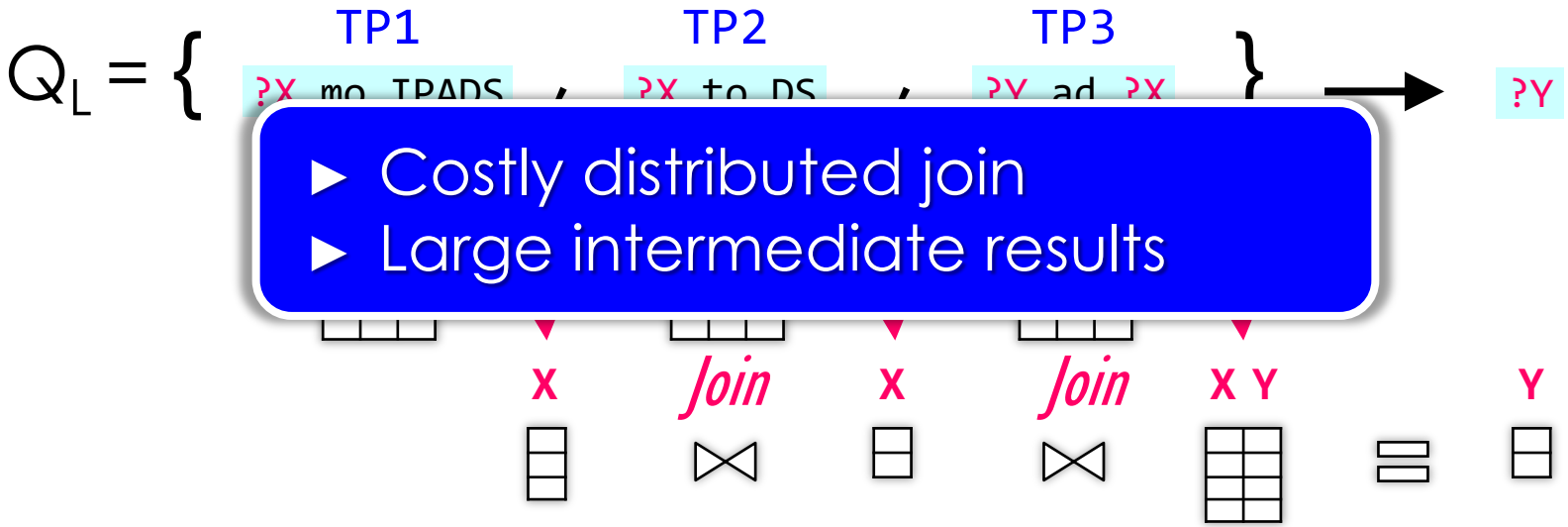


State of the Art

```
SELECT ?Y WHERE {
  ?X memberOf IPADS
  ?X teachOf DS .
  ?Y advisor ?X . }
```
TP1 TP2 TP3

Triple Store and Scan-Join

- Store RDF data as a set of triples in RDBMS



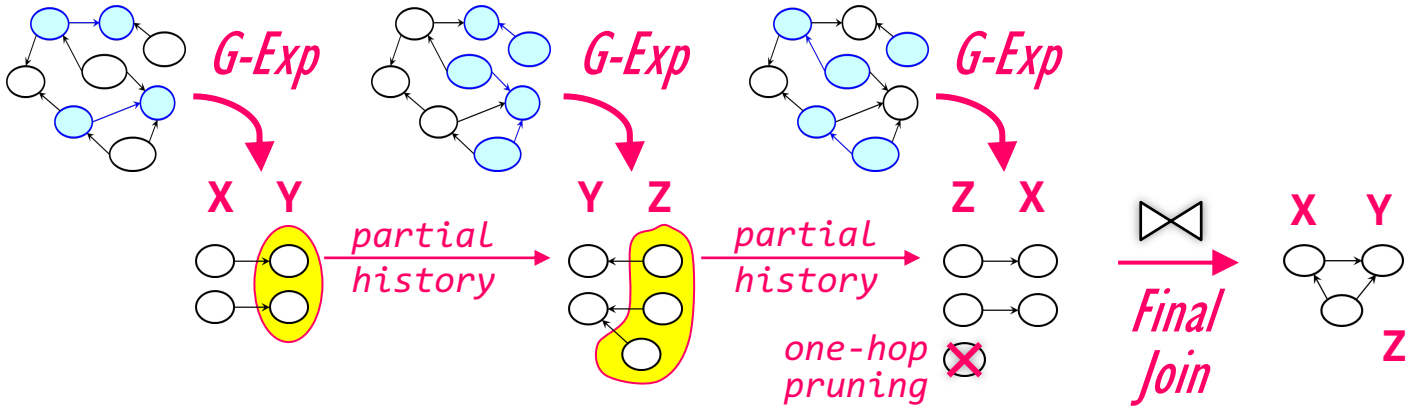
State of the Art

```
SELECT ?X ?Y ?Z WHERE {
TP1   ?X teacherof ?Y .
TP2   ?Z takecourse ?Y .
TP3   ?Z adivsor ?X .      }
```

Graph Store and Graph Exploration

- Store RDF data in a *native* graph model

Q_L = { TP1: ?X to ?Y , TP2: ?Z tc ?Y , TP3: ?Z ad ?X } → ?X ?Y ?Z



State of the Art

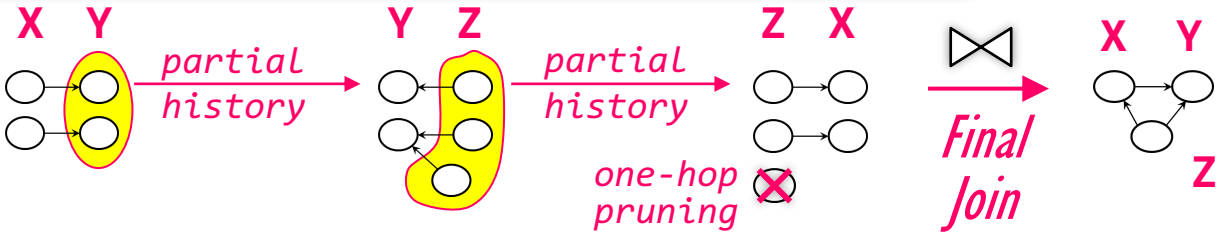
```
SELECT ?X ?Y ?Z WHERE {
TP1   ?X teacherof ?Y .
TP2   ?Z takecourse ?Y .
TP3   ?Z adivsor ?X .      }
```

Graph Store and Graph Exploration

- Store RDF data in a *native* graph model

Q_L = { TP1 , TP2 , TP3 } → ?X ?Y ?Z

- Costly final join (90%)
- Synchronized execution



Performance

LUBM #T=1.41T	CONF	CORES	Latency (ms)	
			HEAVY	LIGHT
Trinity.*RDF*	VLDB	144	12,034	5.65
TriAD	SIGMOD	96	1,920	8.72

Graph Store

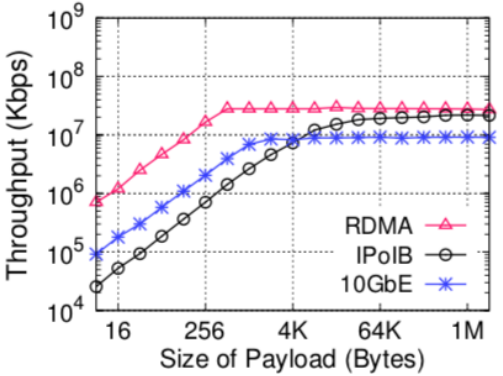
Triple Store

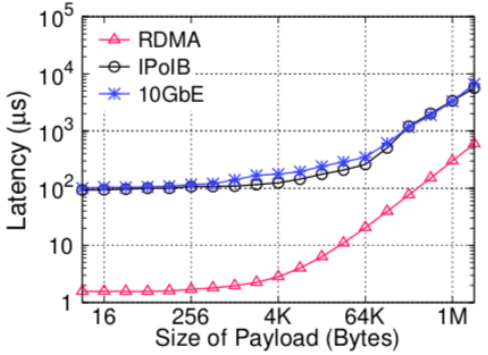
[Wukong, OSDI 2016]

RDMA: Remote Direct Memory Access

High speed, low latency,
and low CPU overhead

- Interface: IPoIB, SEND/RECV Verbs, READ/WRITE *(one-sided primitive)*
- *Bypass* OS kernel: zero copy
- Round-trip time: one-sided/~1-3µs, verb msg/~7µs, IPoIB/~100 µs





Performance

LUBM #T=1.41T	CONF	CORES	Latency (m)	
			HEAVY	LIGHT
Trinity.*RDF*	VLDB	144	12,034	5.65
TriAD	SIGMOD	96	1,920	8.72
+RDMA	40G IB	96	**+4%**	**+137%**

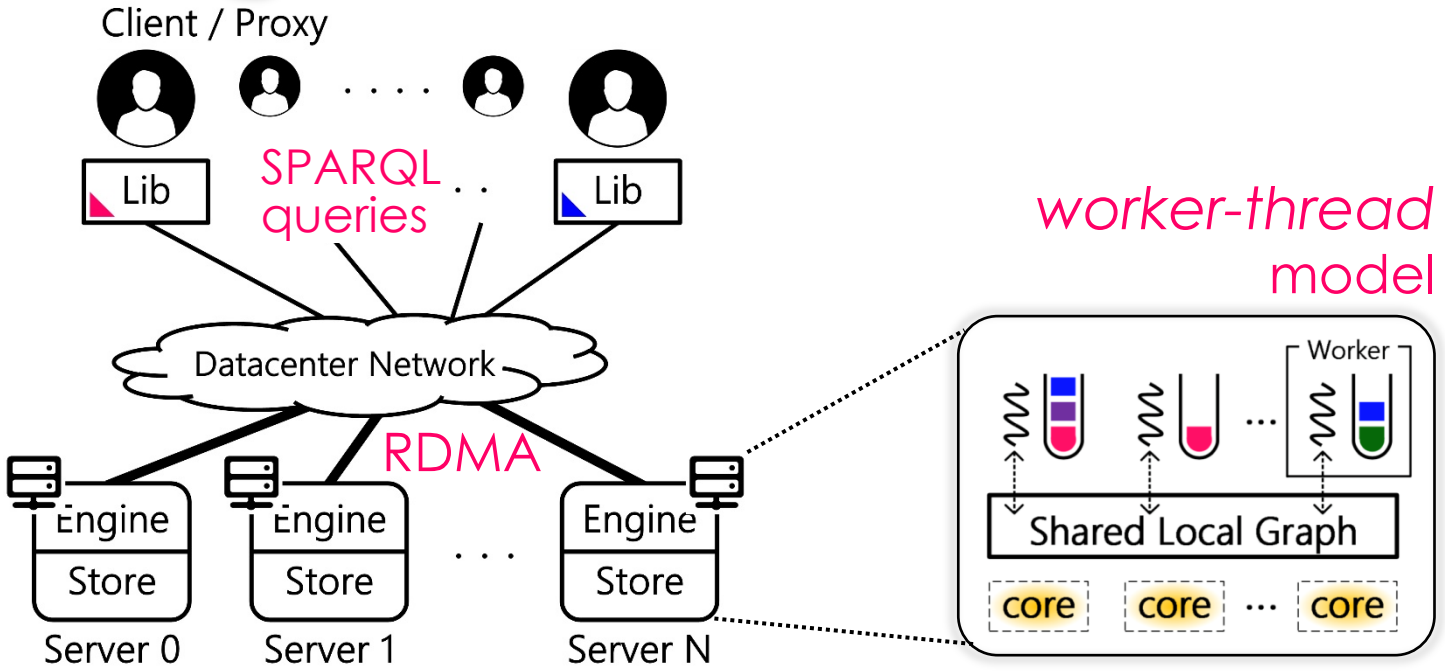
Graph Store

Triple Store

[Wukong, OSDI 2016]

System Overview

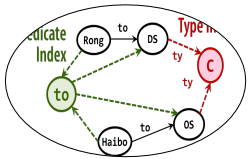
Wukong : A distributed *in-memory* graph store



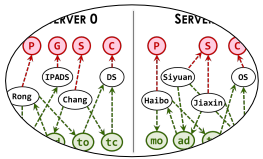
Systematic Approaches

RDMA-friendly in-memory graph store

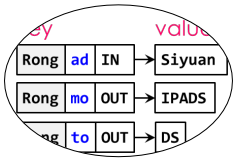
1. Model & Indexing



2. Differentiate Partitioning

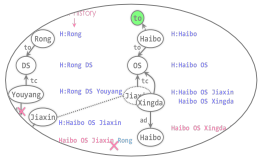


3. Vertex Decomposition

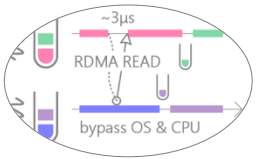


RDMA-enable graph exploration

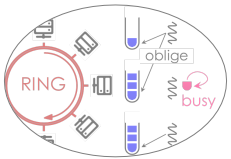
1. Full-history Pruning



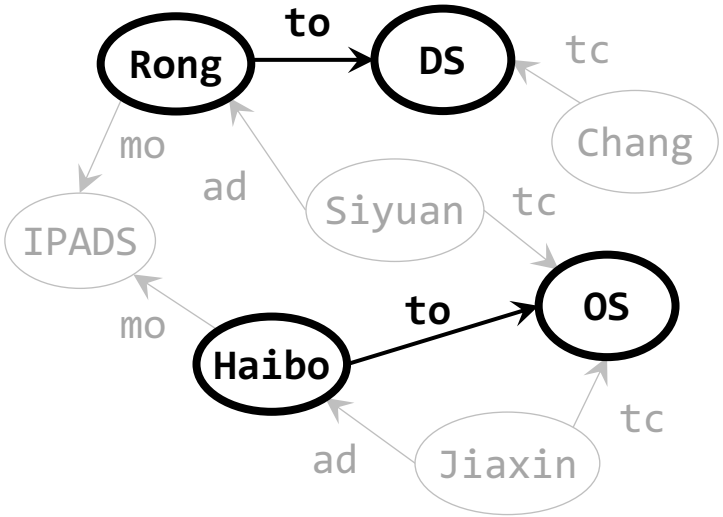
2. Data/Exec Migration



3. Worker-obliger Strategy



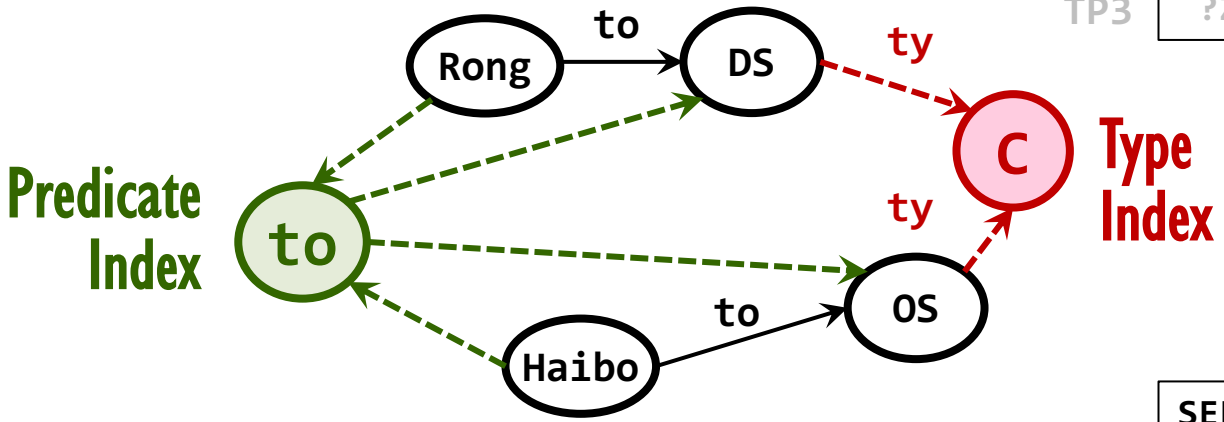
Graph Model and Indexing




```
      SELECT ?X ?Y ?Z WHERE {
TP1     ?X teacherof ?Y .
TP2     ?Z takecourse ?Y .
TP3     ?Z adivsor ?X .       }
```

```
      SELECT ?X ?Y WHERE {
TP1     ?X type Course .
TP2     ?Y teacherOf ?X . }
```

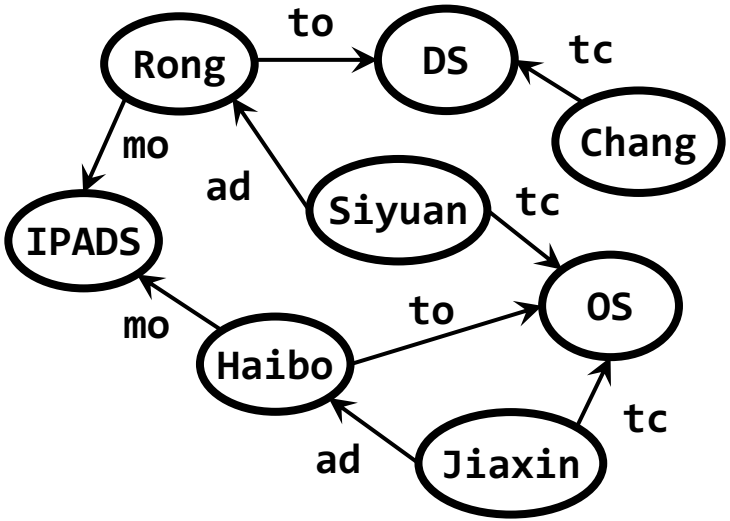
Graph Model and Indexing




```
SELECT ?X ?Y ?Z WHERE {
TP1   ?X teacherof ?Y .
TP2   ?Z takecourse ?Y .
TP3   ?Z adivsor ?X .      }
```

```
SELECT ?X ?Y WHERE {
TP1   ?X type Course .
TP2   ?Y teacherOf ?X . }
```

Differentiated (Graph) Partitioning



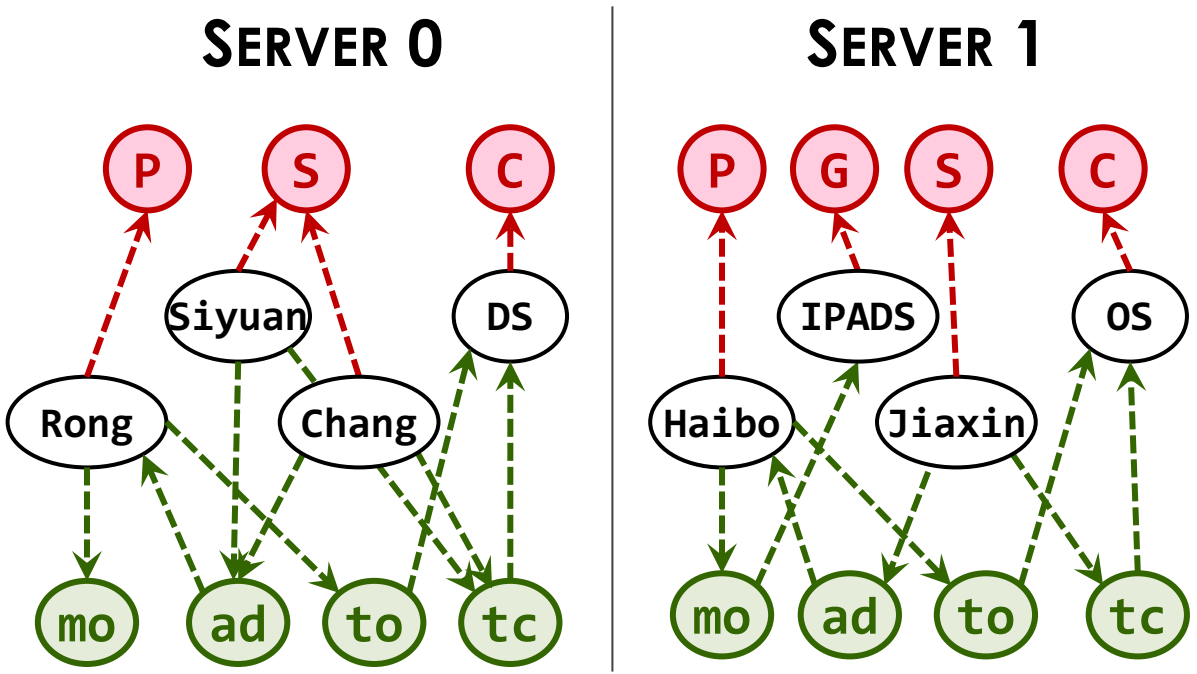
Differentiated (Graph) Partitioning

```
SELECT ?Y WHERE {
  ?X memberOf IPADS      TP1
  ?X teachOf DS .        TP2
  ?Y advisor ?X .    }   TP3
```

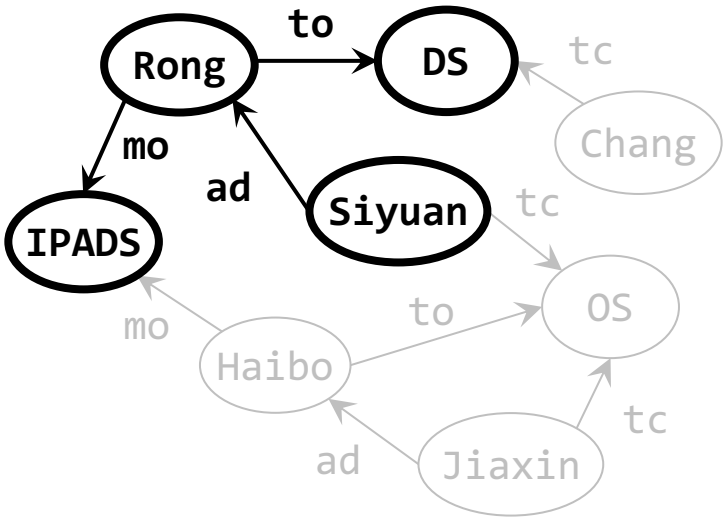
- Start from normal vertex
- Exploit locality

```
SELECT ?X ?Y WHERE {
  ?X teacherOf ?Y .      TP1
  ?Z takerCourse ?Y .    TP2
  ?Z advisor ?X .    }   TP3
```

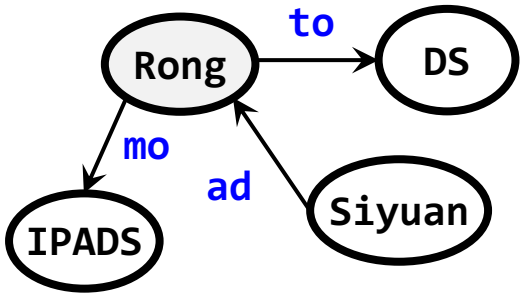
- Start from index vertex
- Exploit parallelism



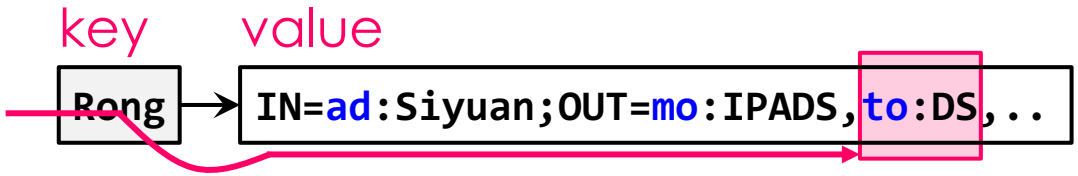
Vertex Decomposition



Vertex Decomposition



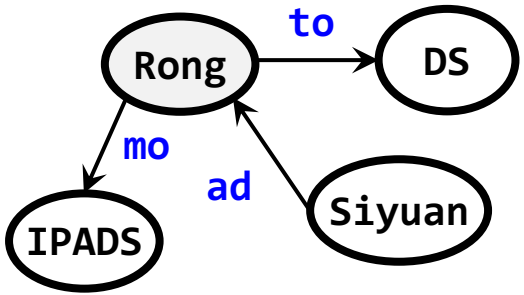
Traditional (e.g., Trinity.*RDF*)




```
SELECT ?Y WHERE {
 RONG teacherOf ?X .     TP1
 ?Y takerCourse ?X . }   TP2
```

- Inefficient lookup
- Unnecessary data transfer

Vertex Decomposition

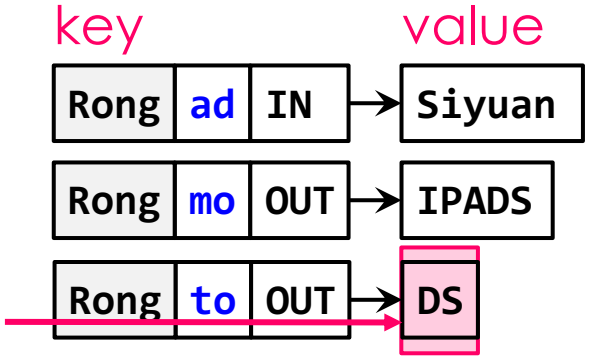



```
SELECT ?Y WHERE {
 RONG teacherOf ?X .      TP1
 ?Y takerCourse ?X . }    TP2
```

- Inefficient lookup
- Unnecessary data transfer

Traditional (e.g., Trinity.*RDF*)

Decomposition

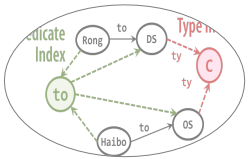


Efficient for both **local** and **remote** (RDMA) accesses

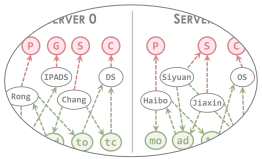
Systematic Approaches

RDMA-friendly in-memory graph store

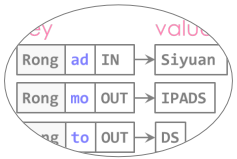
1. Model & Indexing



2. Differentiate Partitioning

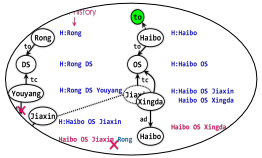


3. Vertex Decomposition

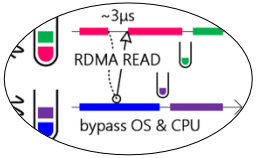


RDMA-enable graph exploration

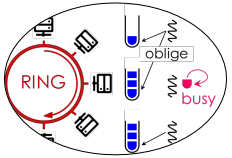
1. Full-history Pruning



2. Data/Exec Migration



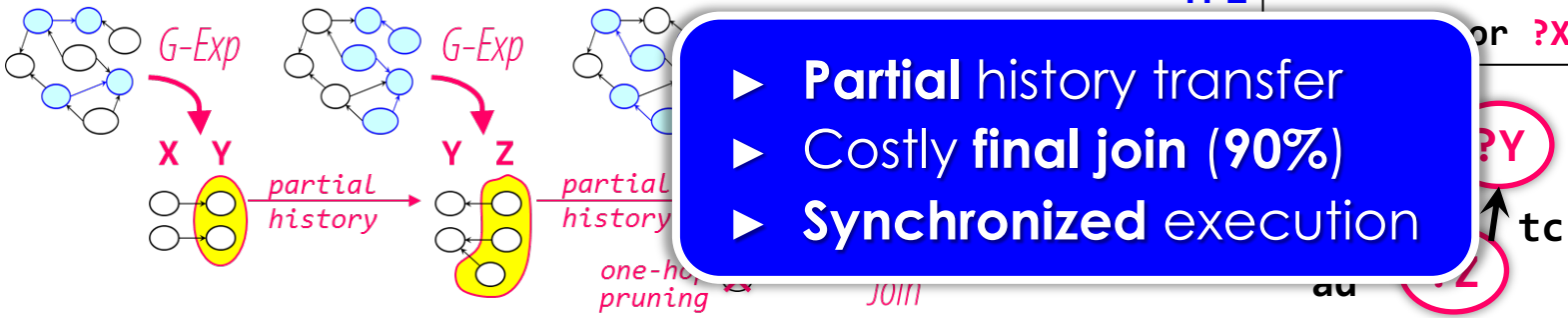
3. Worker-obliger Strategy



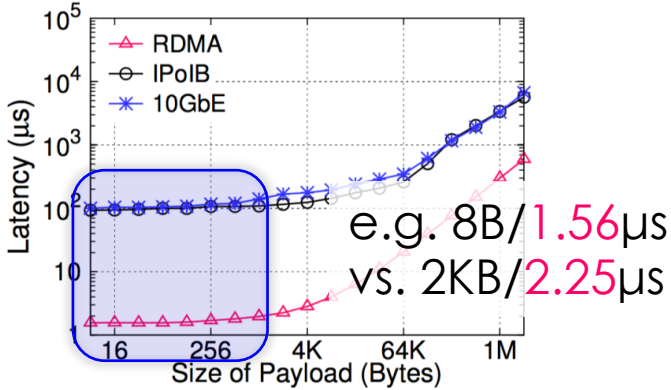
Prior Graph Exploration

```
SELECT ?X ?Y ?Z WHERE {
  ?X teacherof ?Y .
  ?Z takecourse ?Y .
  ?Z memberof ?X .      }
```

TP1
TP2



- **Partial** history transfer
- Costly **final join (90%)**
- **Synchronized** execution



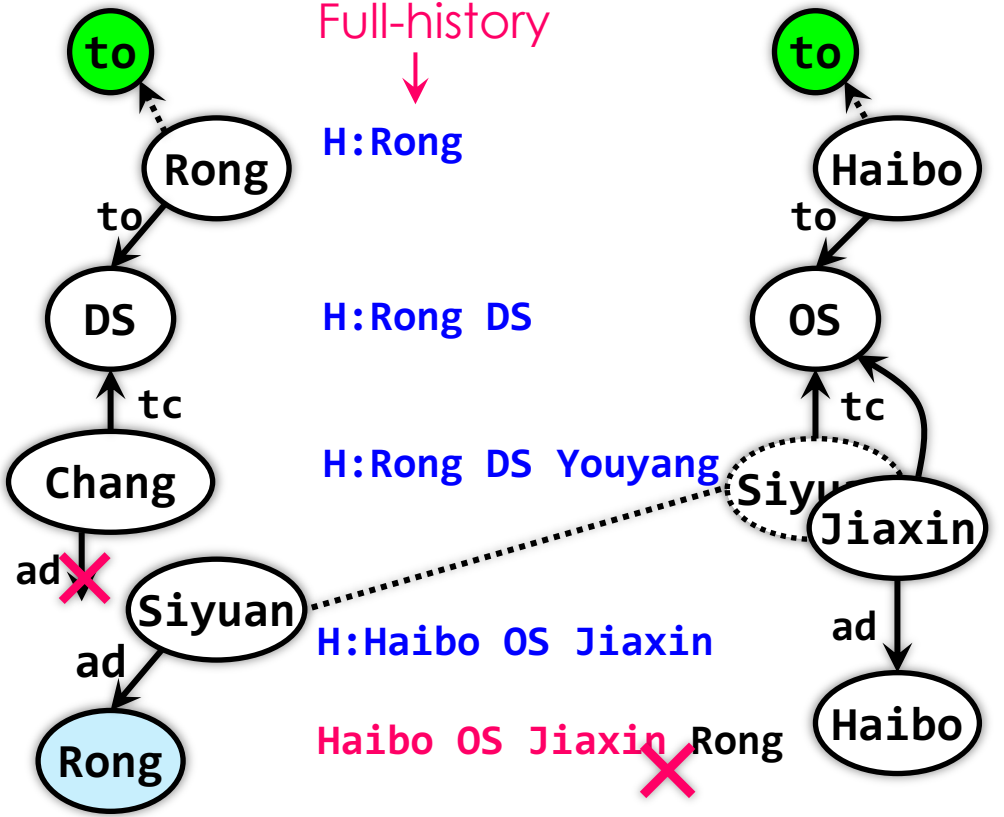
e.g. 8B/1.56µs
vs. 2KB/2.25µs

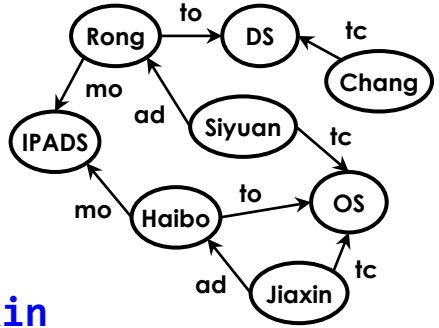
The latency of RDMA is relatively insensitive to payload sizes (~2K)

Graph exploration w/
full-history pruning

Full-History Pruning

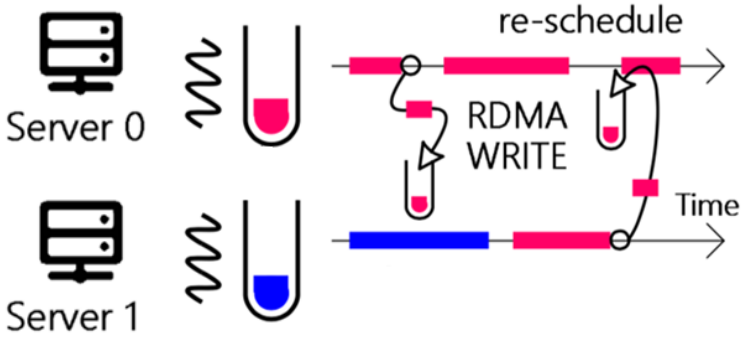
```
SELECT ?X ?Y ?Z WHERE {
 ?X teacherof ?Y .
 ?Z takecourse ?Y .
 ?Z adivsor ?X .      }
```





Migrate Execution or Data

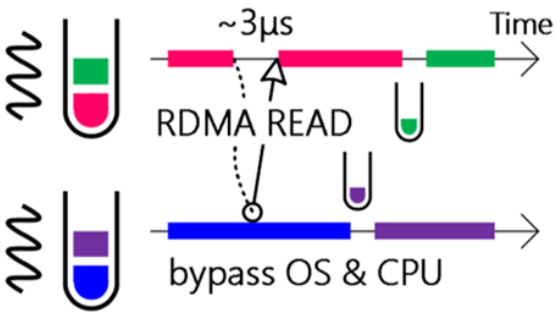
Fork-join
(migrate exec)



- Send sub-query by RDMA WRITE
- *Async* exploration w/ full-History

Exploit high parallelism

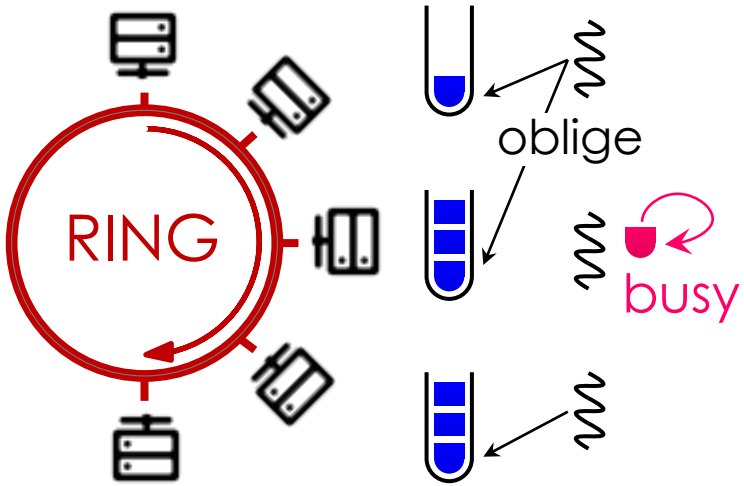
In-place
(migrate data)



- Fetch data by RDMA READ
- Bypass remote CPU & OS

Exploit low latency

Worker-obliger Strategy




```
1  int next = 1

   OBLIGER()
2    s = state[(tid+next)%N]
3    q = NULL
4    s.lock()
5    if (s.cur == tid //reentry
6     || s.end < now)
7      s.cur = tid;
8      s.end = now + T
9      next++
10     q = s.dequeue()
11   s.unlock()
12   return q
```

```
   SELF()
13   s = state[tid]
14   s.lock()
15   s.cur = tid
16   s.end = now + T
17   next = 1
18   q = s.dequeue()
19   s.unlock()
20   return q

   NEXT_QUERY()
21   if (q = OBLIGER())
22     return q
23   return SELF()
```

- ***Latency**-centric* work-stealing algorithm
- **Ring** policy: practical, effective, and easy to impl.

Performance

LUBM #T=1.41T	CONF	CORES	Latency (m)	
			HEAVY	LIGHT
Trinity.*RDF*	VLDB	144	12,034	5.65
TriAD	SIGMOD	96	1,920	8.72
+RDMA	40G IB	96	+4%	+137%*

* using only 1 core

[Wukong, OSDI 2016]

Performance

LUBM #T=1.41T	CONF	CORES	Latency (m)	
			HEAVY	LIGHT
Trinity.*RDF*	VLDB	144	12,034	5.65
TriAD	SIGMOD	96	1,920	8.72
+RDMA	40G IB	96	+4%	+137%*
WUKONG	OSDI	96	**248**	**0.40***
			7.4X	**21.8X**

* using only 1 core

[Wukong, OSDI 2016]

Performance

LUBM #T=1.41T	CONF	CORES	Latency (m)		THPT (q/s)
			HEAVY	LIGHT	
Trinity.*RDF*	VLDB	144	12,034	5.65	≈ 400
TriAD	SIGMOD	96	1,920	8.72	250
+RDMA	40G IB	96	+4%	+137%*	
WUKONG	OSDI	96	**248**	**0.40***	**269K**
			7.4X	**21.8X**	**1,076X**

* using only 1 core

[Wukong, OSDI 2016]

Conclusion: Wukong@OSDI16

New hardware technologies open opportunities

Wukong: a distributed in-memory RDF store that leverages RDMA-based graph exploration to support fast and concurrent RDF queries

Achieving orders-of-magnitude lower latency & higher throughput than prior state-of-the-art systems

Website: http://ipads.se.sjtu.edu.cn/projects/wukong

GitHub: https://github.com/SJTU-IPADS/wukong

Fast and Concurrent RDF Queries using *RDMA-assisted* GPU Graph Exploration

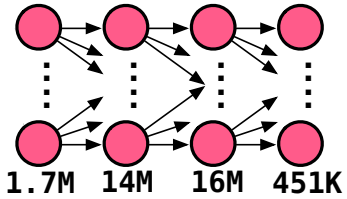
USENIX ATC '18
2018 USENIX Annual Technical Conference

Queries are Heterogeneous

Heavy Query (Q_H)

- Start from a set of vertices
- Explore a large part of graph

Q7*
390 ms

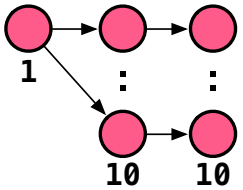


3000X

Light Query (Q_L)

- Start from a given vertex
- Explore a small part of graph

Q5*
0.13 ms

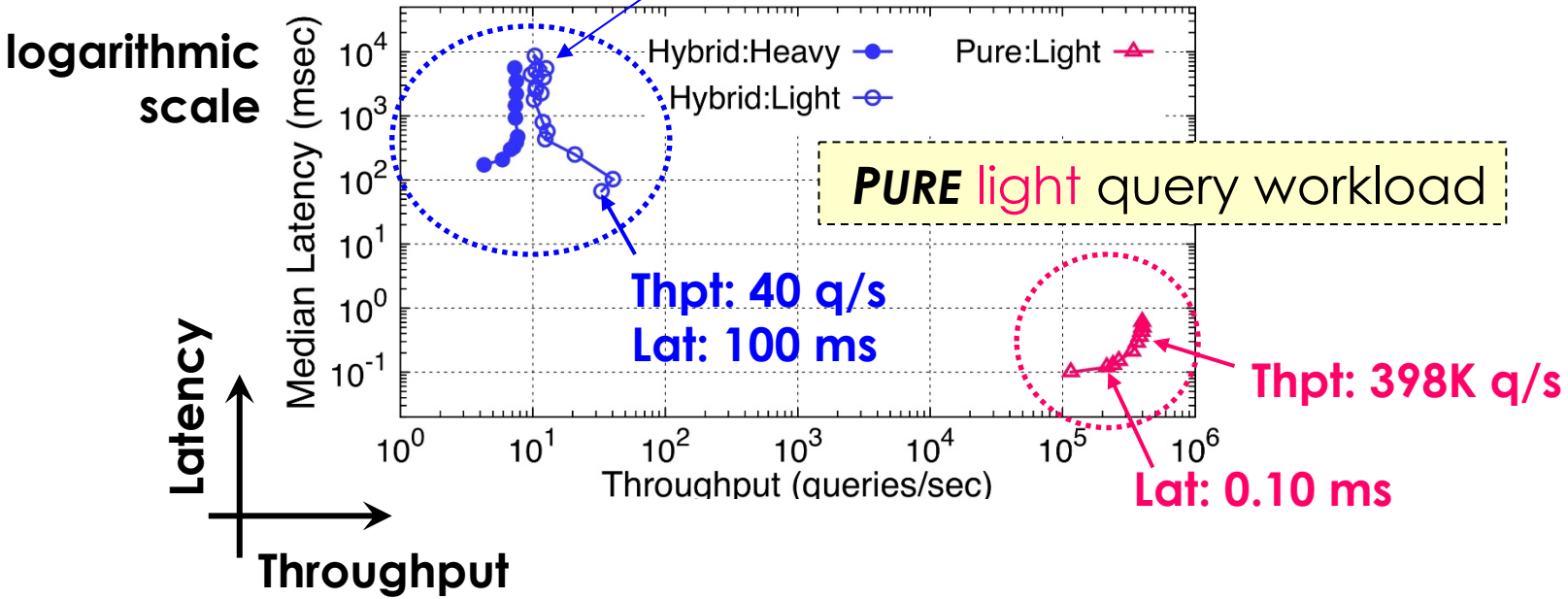


* Wukong on a 10-server cluster for LUBM-10240 dataset

Concurrent Workload

HYBRID light & heavy query workload

Thpt: 10 q/s
Lat: 8,600 ms



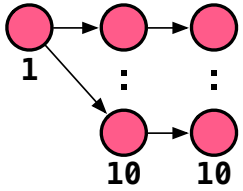
PURE light query workload

General Idea

Heterogeneous Workload

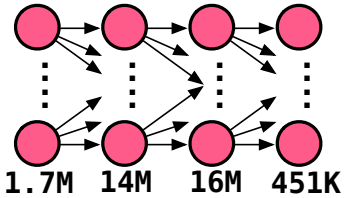
Heterogeneous Hardware

Light Query



CPU

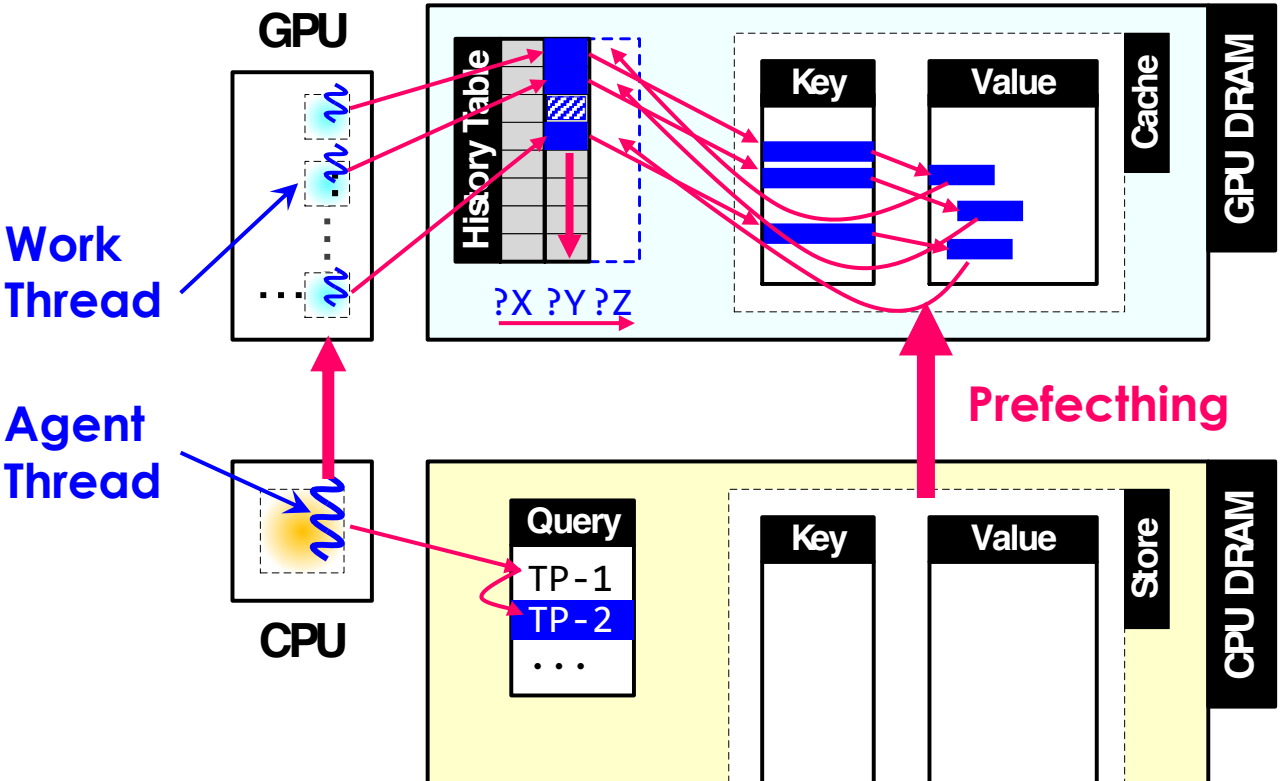
Heavy Query



GPU

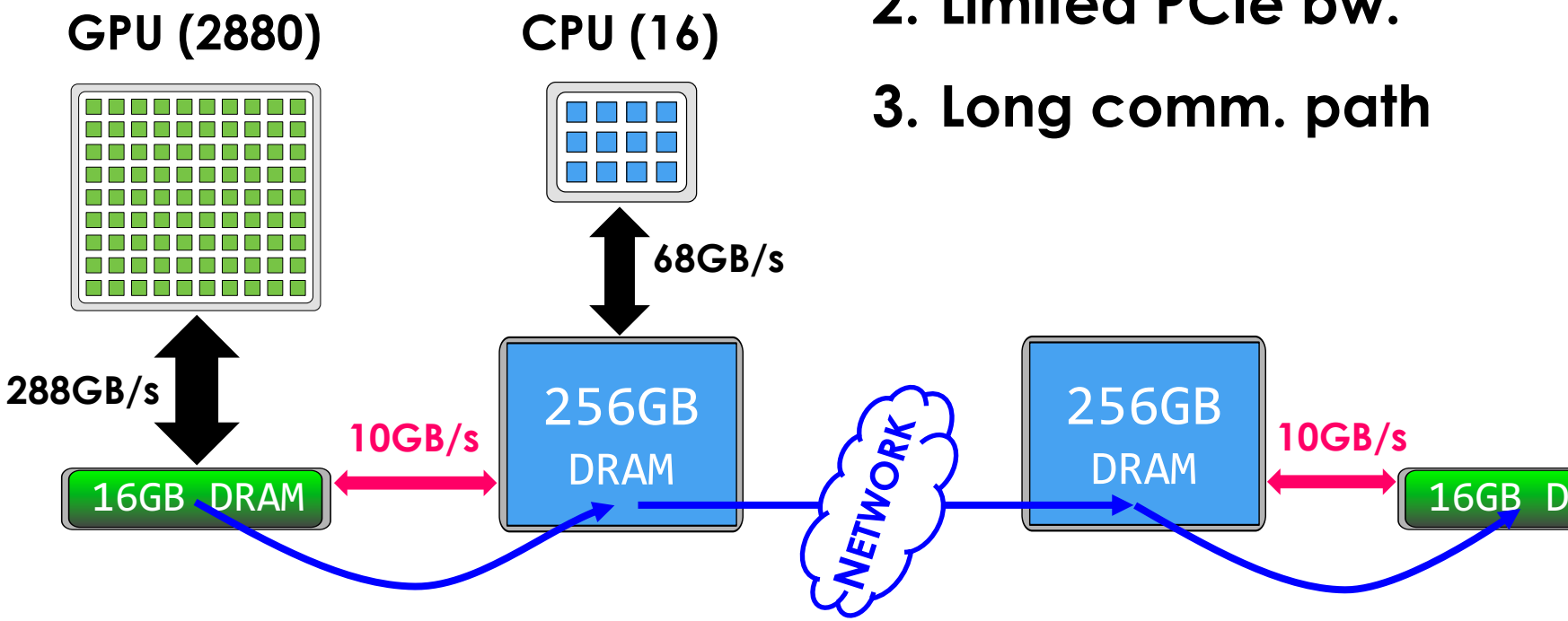
Query Execution on GPU

```
SELECT ?X ?Y ?Z WHERE {
TP1   ?X teacherof ?Y .
TP2   ?Z takecourse ?Y .
TP3   ?Z adivsor ?X .      }
```

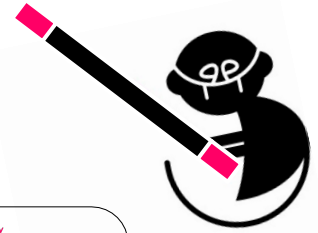


Challenges

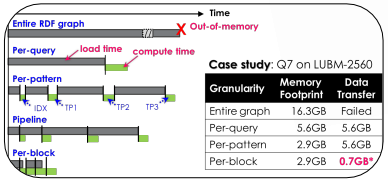
1. Small GPU memory
2. Limited PCIe bw.
3. Long comm. path



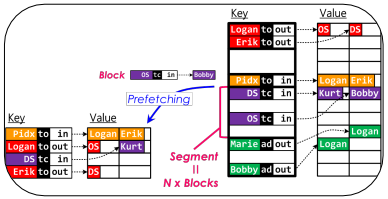
Systematic Approaches



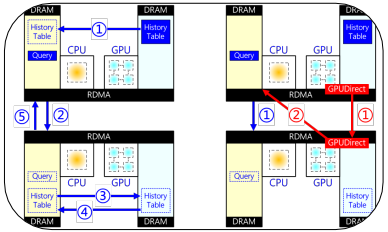
1. **Smart data prefetching**



2. **GPU-friendly key/value store**



3. ***Heterogeneous RDMA comm.***



Performance

LUBM #T=1.41T	CONF	CORES	Latency (ms)		THPT (q/s)	
			HEAVY	LIGHT	HEAVY	LIGHT
TriAD	SIGMOD	100	3,094	7.04		
WUKONG	OSDI	100	215	**0.34***	**7.7**	**40**

* using only 1 core

[Wukong+G, ATC 2018]

Performance

LUBM #T=1.41T	CONF	CORES	Latency (ms)		THPT (q/s)	
			HEAVY	LIGHT	HEAVY	LIGHT
TriAD	SIGMOD	100	3,094	7.04		
WUKONG	OSDI	100	215	**0.34***	7.7	40
WUK+G	ATC	100 10 GPU	**47**	0.38*	**45.4**	**346K**
			7.4X	**-11%**	**5.9X**	**8,650X**

* using only 1 core

[Wukong+G, ATC 2018]

Conclusion: Wukong+G@ATC18

Hardware heterogeneity opens opportunities for hybrid workloads on graph data

Wukong+G : a distributed RDF query system supports heterogeneous CPU/GPU processing for hybrid queries on graph data

Outperform prior state-of-the-art systems by more than one order of magnitude when facing hybrid workloads

Website: http://ipads.se.sjtu.edu.cn/projects/wukong

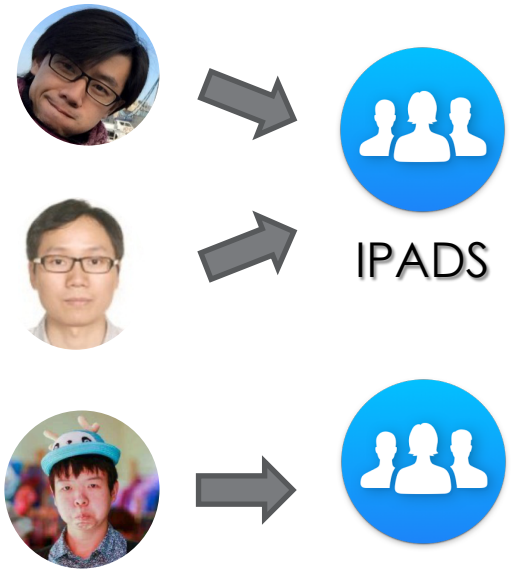
GitHub: https://github.com/SJTU-IPADS/wukong

Sub-millisecond Stateful Stream Querying over Fast-evolving Linked Data

Streaming Data and Querying

Multiple data sources are continuously generating **streaming data** in **high velocity**

Example: Social Networking

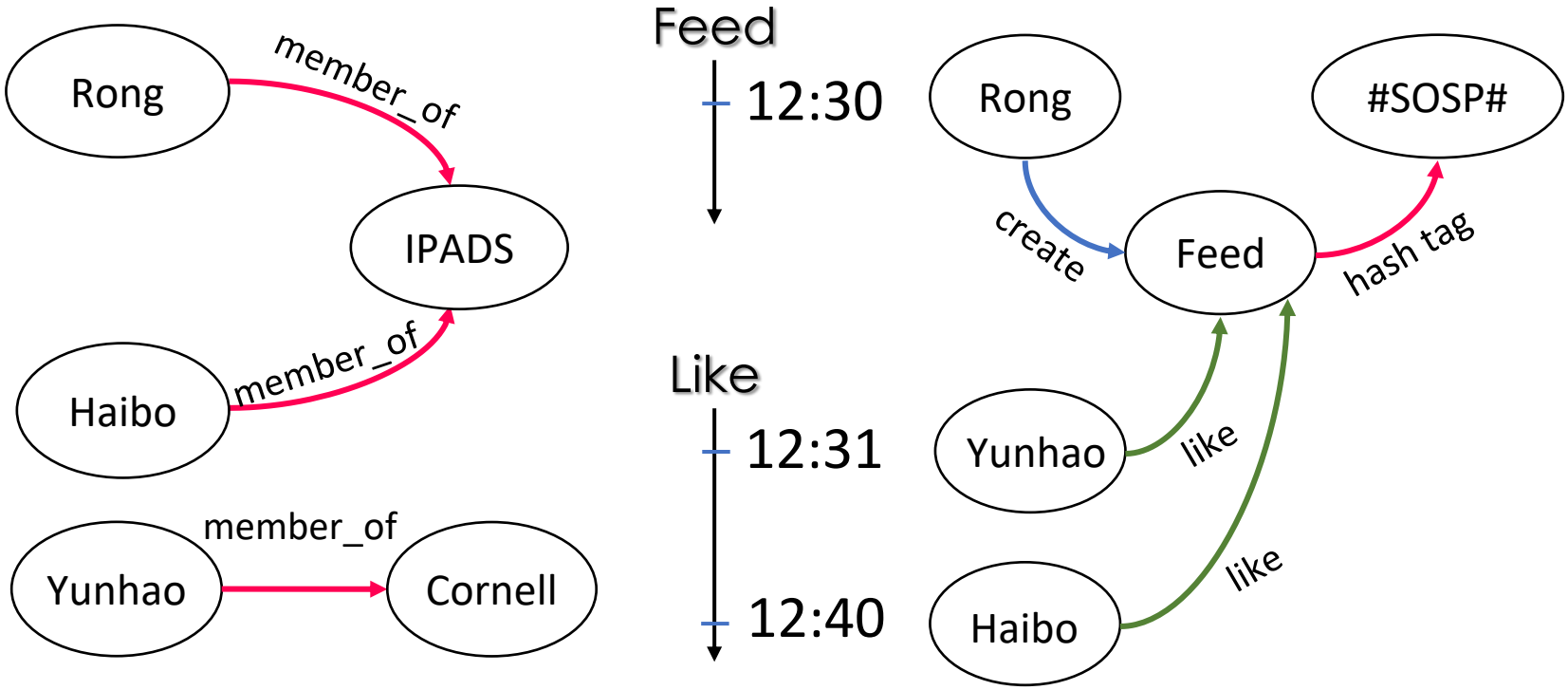


<Rong, creates, Feed>. *12:30*
<Feed, hash_tag, SOSP>. *12:30*
<Yunhao, likes, Feed>. *12:31*
<Haibo, likes, Feed>. *12:40*

Stored Data

Streaming Data

Example: Social Networking

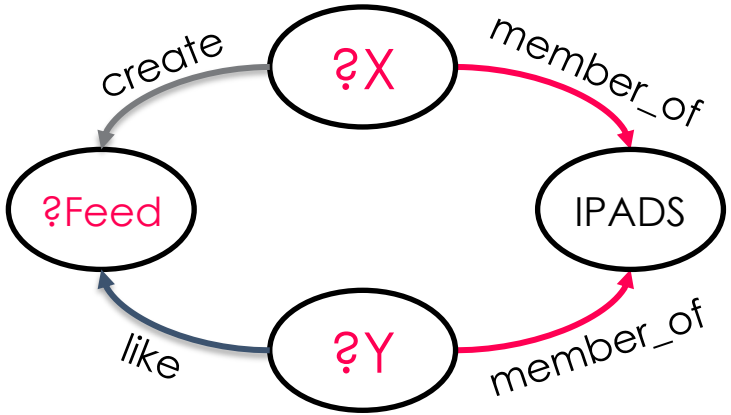


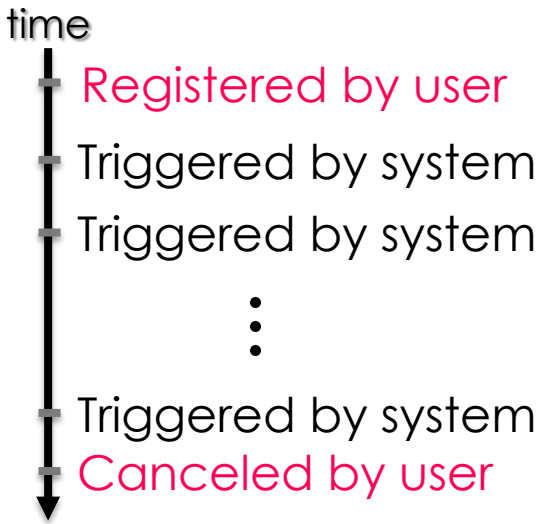
Stored Data

Streaming Data

Example: Social Networking

Last 30 minutes, which IPADS members created feeds that are liked by other IPADS members?

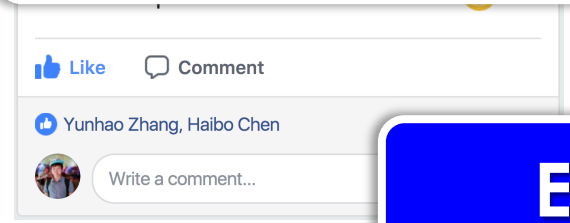




Stateful Streaming Query

A **stateful** streaming query needs to *integrate* **streaming data** with **stored data**

High Velocity



Real-time User Activity

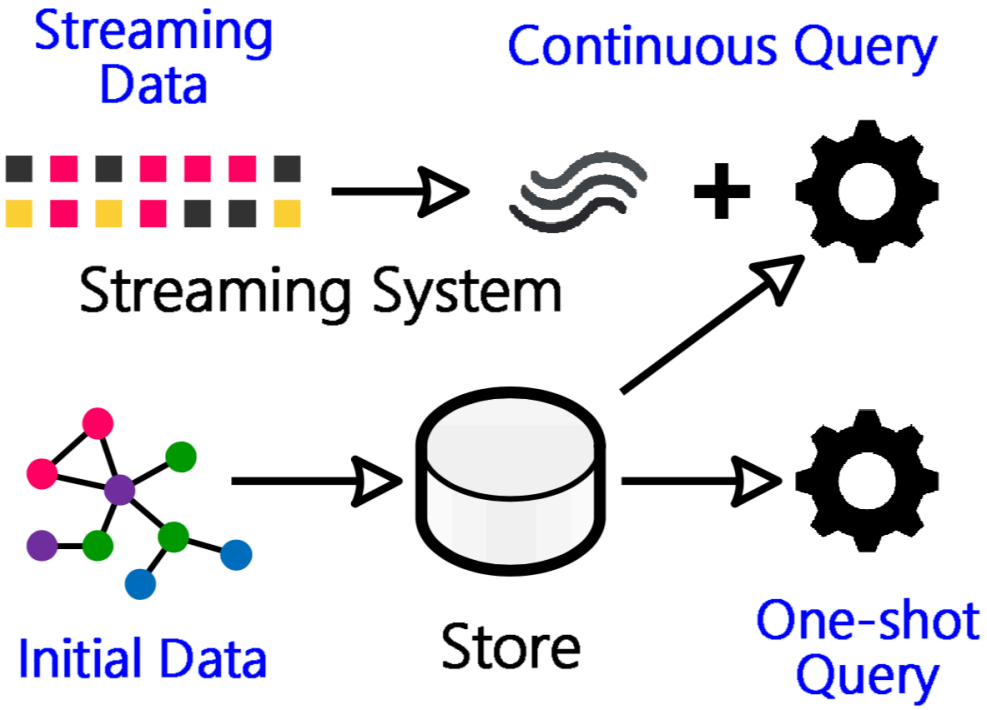
+

Large Volume

Durable Social Graph

Evolving!

Conventional Approach



Conventional Approach

1. **Cross-system Cost**

   ~40% execution time waste on data transformation and transmission

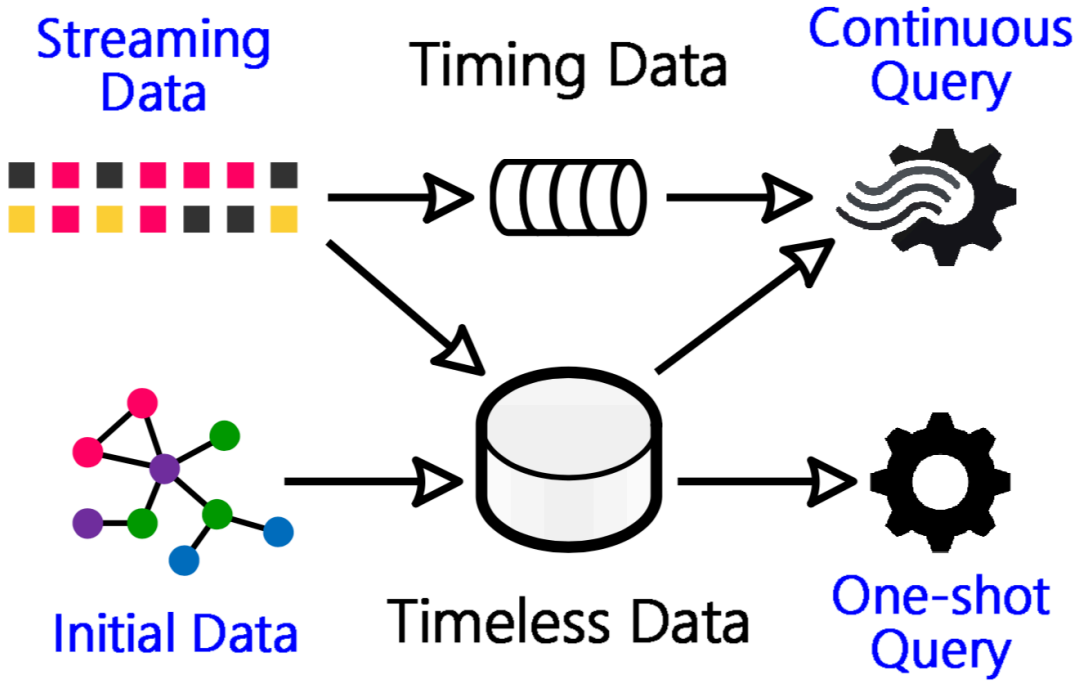
2. **Inefficient Query Plan**

   Semantic gaps between the two systems impair query optimizations

3. **Limited Scalability**

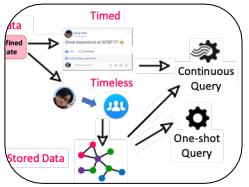
   Stream processing systems dedicate all resources to improve the performance of a single job

Integrated Design

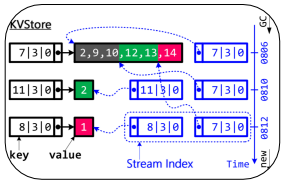


Systematic Approaches

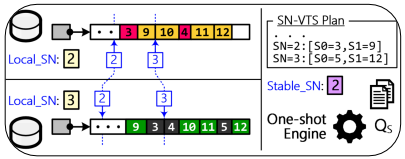
1. **Hybrid store: timed & timing**



2. **Stream index & partitioning**



3. **Bounded snapshot scalarization**



Performance

LSBENCH #T=118M R=133K/s	CORES	WUKONG STORM	CSPARQL-engine
GEO. M	**24 x1**	5.91	757

LSBENCH #T=3.75B R=133K/s	CORES	WUKONG STORM	WUKONG HERON	SPARK STREAMING
GEO. M	**24 x8**	6.29	5.85	679

[Wukong+S, SOSP 2017]

Performance

LSBENCH #T=118M R=133K/s	CORES	WUKONG STORM	CSPARQL-engine	WUKONG+S
GEO. M	24 x1	5.91 12.3X	757 1,577X	0.48ms

LSBENCH #T=3.75B R=133K/s	CORES	WUKONG STORM	WUKONG HERON	SPARK STREAMING	WUKONG+S
GEO. M	24 x8	6.29 13.7X	5.85 12.7X	679 1,476X	0.46ms

[Wukong+S, SOSP 2017]

Conclusion: Wukong+S@SOSP17

Wukong+S : distributed querying engine adopting **integrated design** for **stateful** stream queries over **fast-evolving** linked data

Achieving **sub-millisecond** latency and exceeding **one million** queries per second

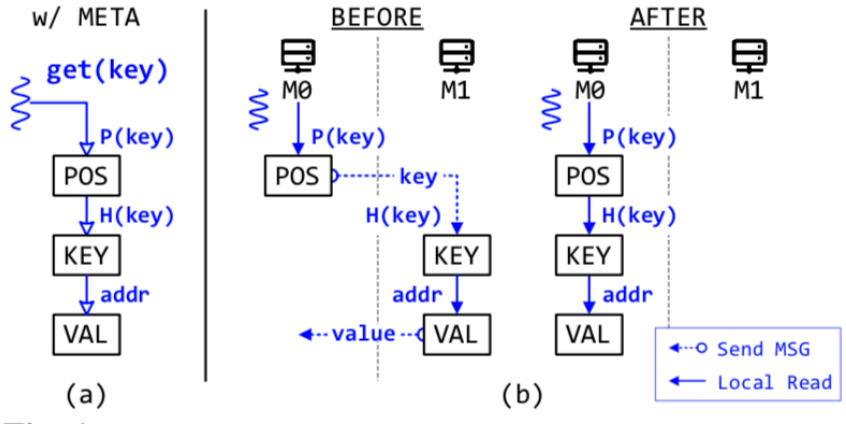
Website: http://ipads.se.sjtu.edu.cn/projects/wukong

GitHub: https://github.com/SJTU-IPADS/wukong

Current Projects on **Wukong**

- Lightweight, non-invasive migration for graph store
- Fast and accurate optimizer for graph query
- Supporting pipeline workload on linked data

Shard-based Migration



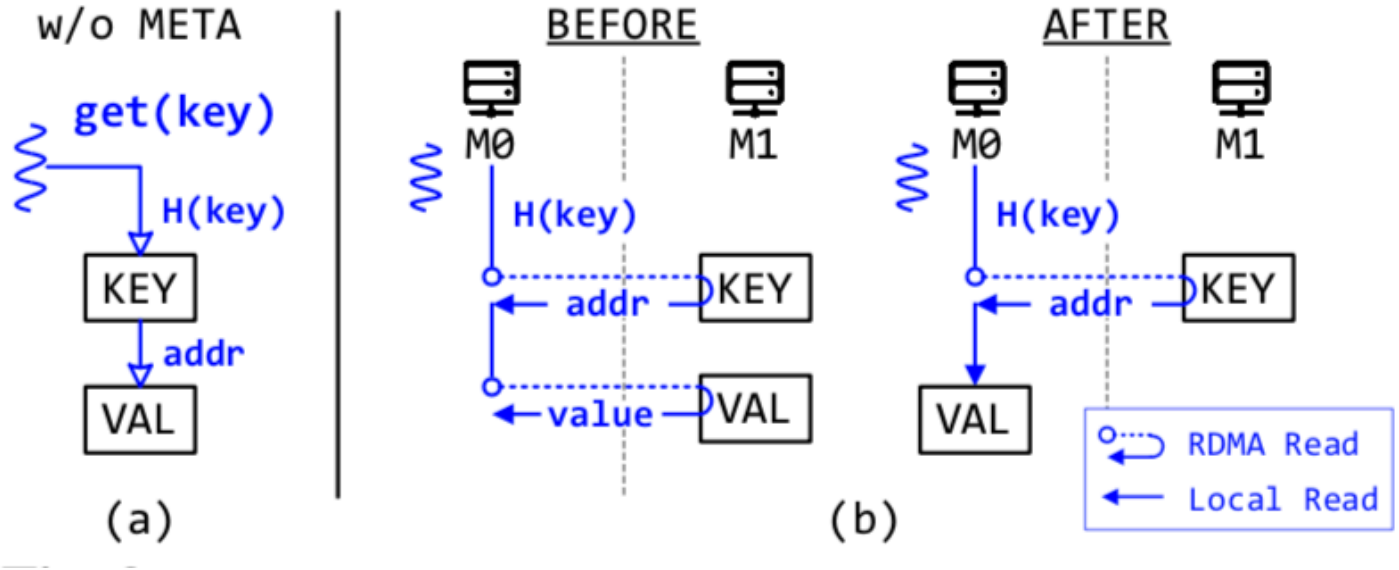
	Ideal	Shard-based	
		Before	After
Throughput (K ops/sec)	3,204	116	159
Median/50^{th} Latency (msec)	0.61	17.15	12.02
Tail/99^{th} Latency (msec)	2.45	78.60	63.50
Remote Access Rate (%)	0	87.9	68.1
Data Volume Migrated (GB)	-	-	5.xx

Dataset: Graph500

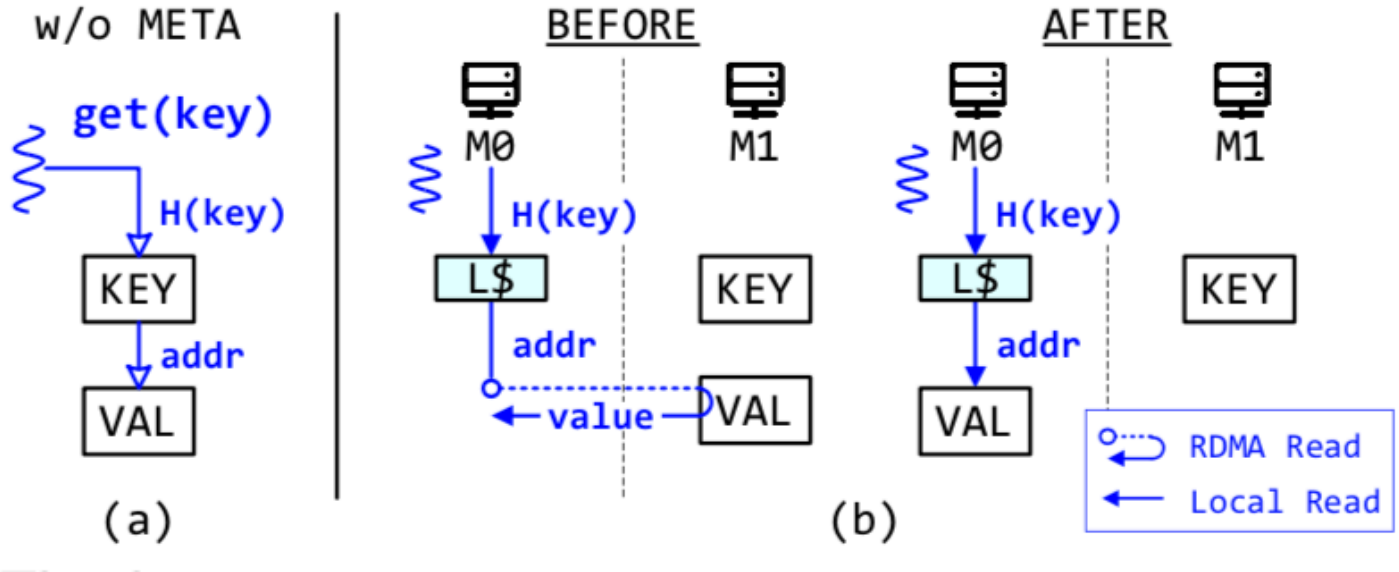
Workload: 2-hop queries

- Graph data: poor locality
- Additional metadata (POS)
- Conflict with READ & WRITE operations

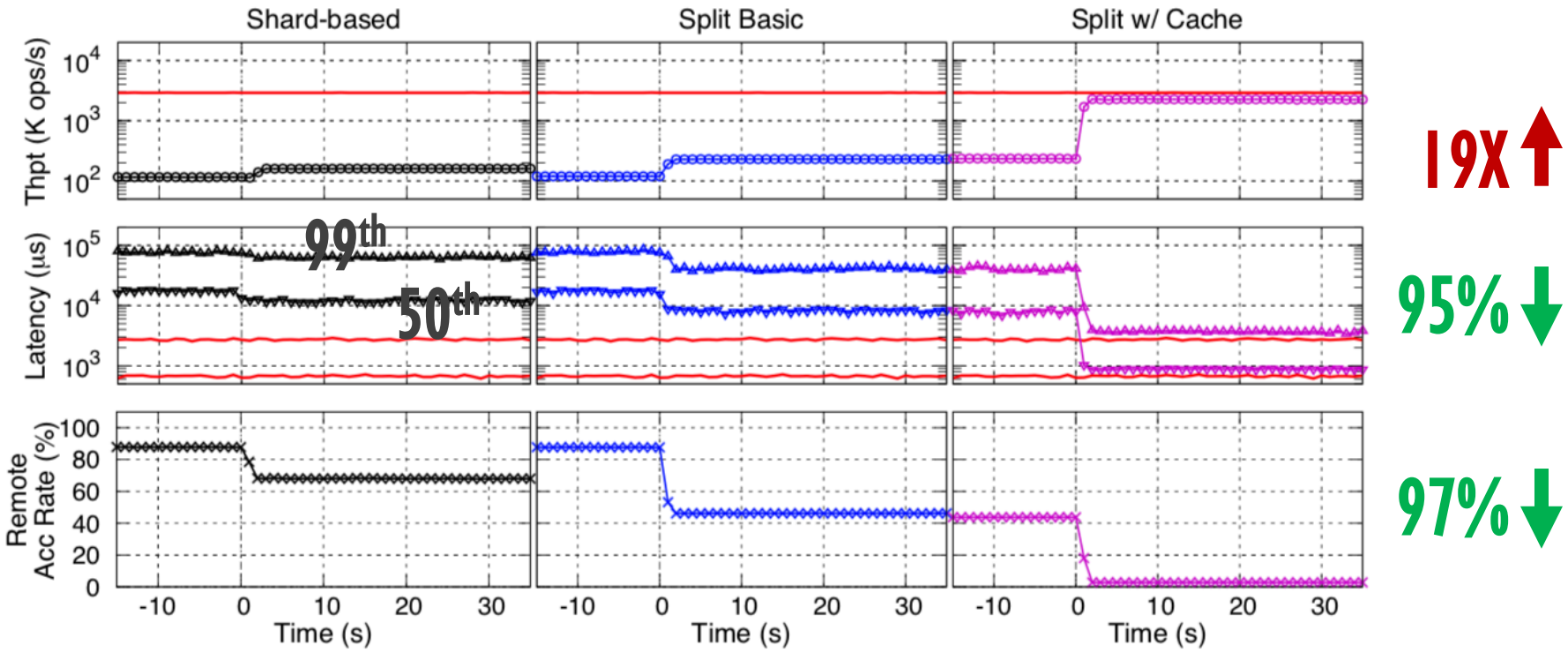
Split Live Migration



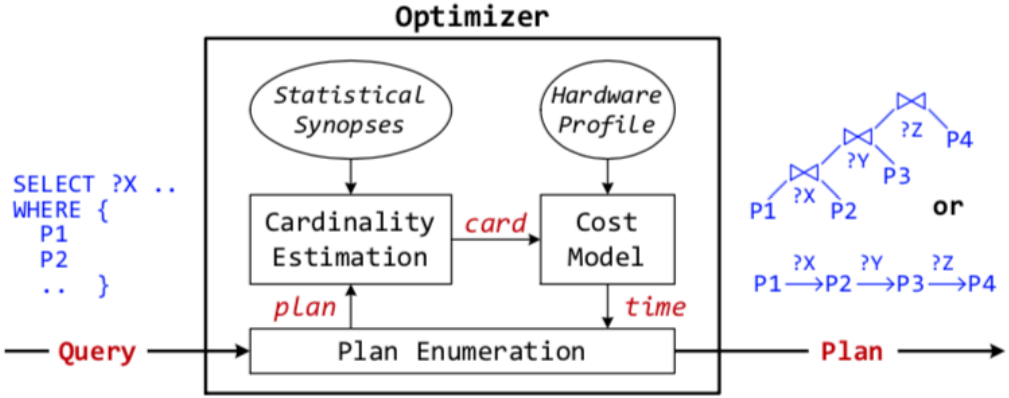
Integrate with **Location Cache**



Lightweight & Non-invasive Migration

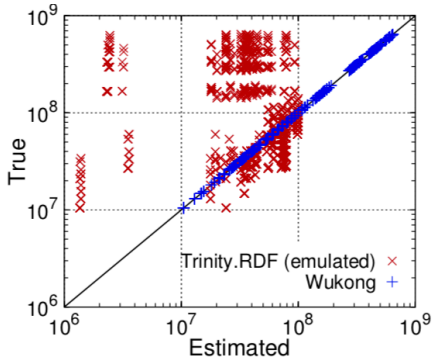


Fast and Accurate Optimizer

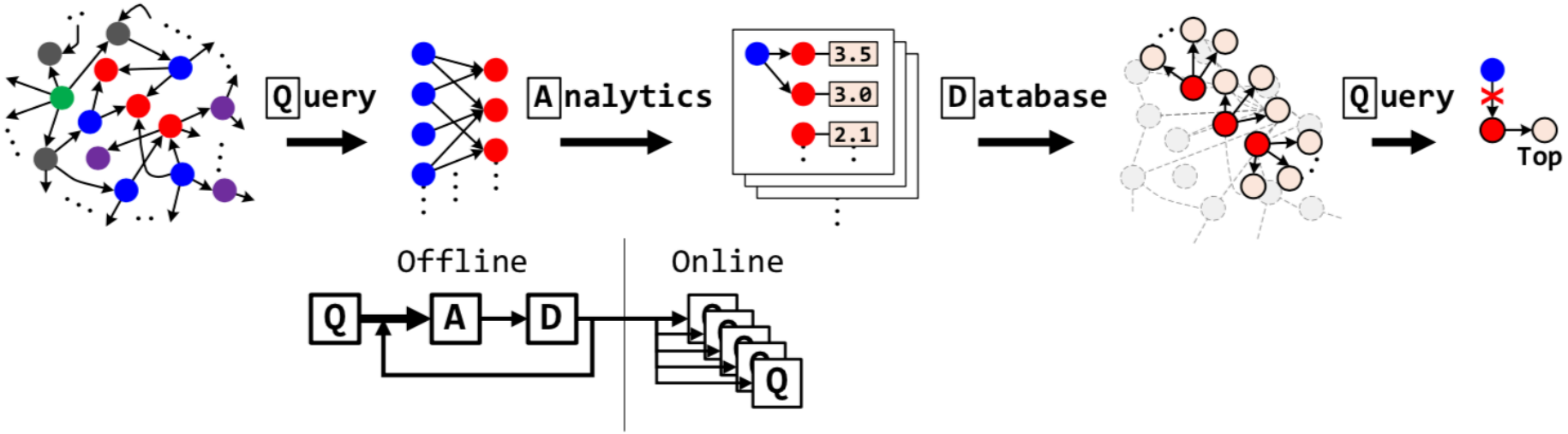


Wukong			
OPT	OSDI16	Trinity.RDF	Wukong+P
311	513	325|0.073	**311**|0.520
80	80	80|0.011	**80**|0.012
225	227	274|0.119	†(~~225~~)**0**|0.305
0.030	0.030	0.030|0.021	0.030|**0.008**
0.023	0.023	0.023|0.005	0.023|**0.005**
0.095	0.095	0.097|0.009	0.095|**0.007**
194	309	347|0.077	**195**|0.290
4.93	5.67	6.25	(~~5.31~~)2.07

1. Cardinality Estimation: Type-centric
2. Cost model: Mimic-based
3. Plan enumeration: budget-aware



Pipeline Graph Processing



Lubm-640/ 4-node /s	Query			Transform		Preprocessing	Analytics	Total
	excutution	result bring back	final aggregate and send back	maping	format			
Wukong + powerlyra	0.640	3.096	9.82		0	2.86	2.64	15.776
Wukong + Gemini	0.640	3.096	9.82	6.28	0.67	2.97	0.25	23.086
Wukong-pipeline	0.462	1.343	0		0	0.702	0.322	2.829

Thanks

Website: http://ipads.se.sjtu.edu.cn/projects/wukong

GitHub: https://github.com/SJTU-IPADS/wukong

Wukong, short for Sun Wukong, who is known as the Monkey King and is a main character in the Chinese classical novel "Journey to the West". Since Wukong is known for his extremely fast speed (21,675 kilometers in one somersault) and the ability to fork himself to do massive multi-tasking, we term our system as Wukong.

Questions