Using Property Graphs for Rich Metadata Management in HPC Systems

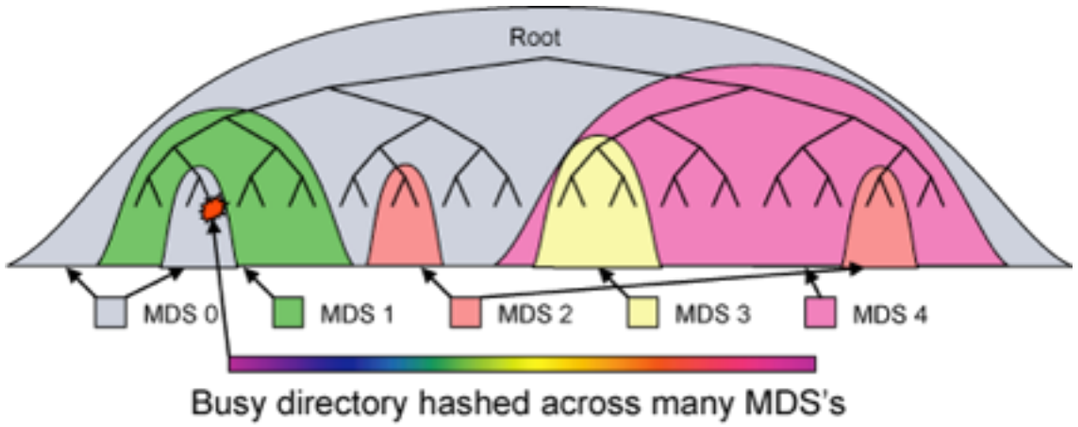
Dong Dai, Robert B. Ross, Philip Carns, Dries Kimpe, and Yong Chen

Rich Metadata in HPC

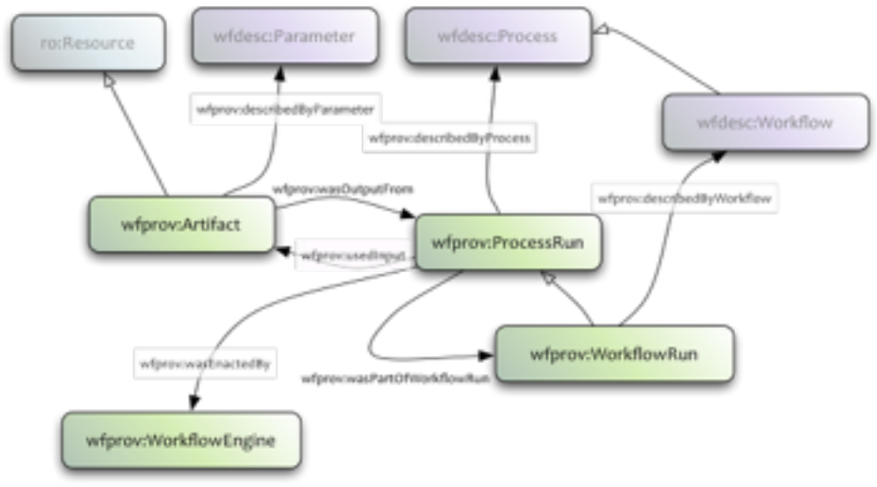
- The data used to describe other data
  - Simple Metadata
  - Rich Metadata
- HPC systems heavily rely on these metadata
  - *inode* attributes for file management
  - *location* information for directories and files stored across metadata server
  - *provenance* information partially collected and stored

Attribute	Description	Attribute	Description
ino	inode number	ctime	change time
pino	parent inode number	atime	access time
name	file name	owner	file owner
type	file or directory	group	file group
size	file size	mode	file mode
mtime	modification time		

- ***A. Leung, et. al. Magellan: A Searchable Metadata Architecture for Large-Scale File Systems***

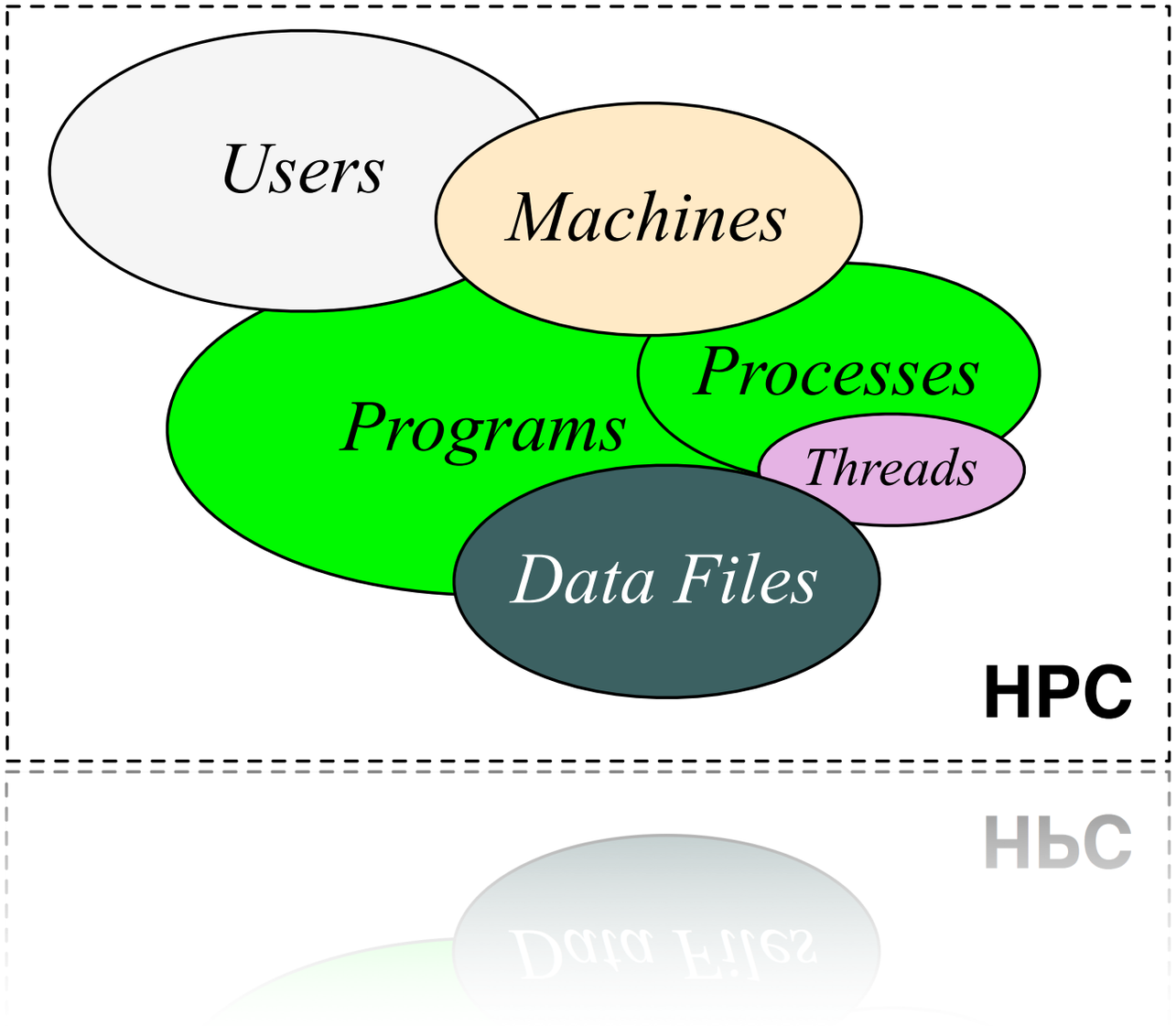


- ***S. A. Weil, et. al. Ceph: A Scalable, High-Performance Distributed File System***



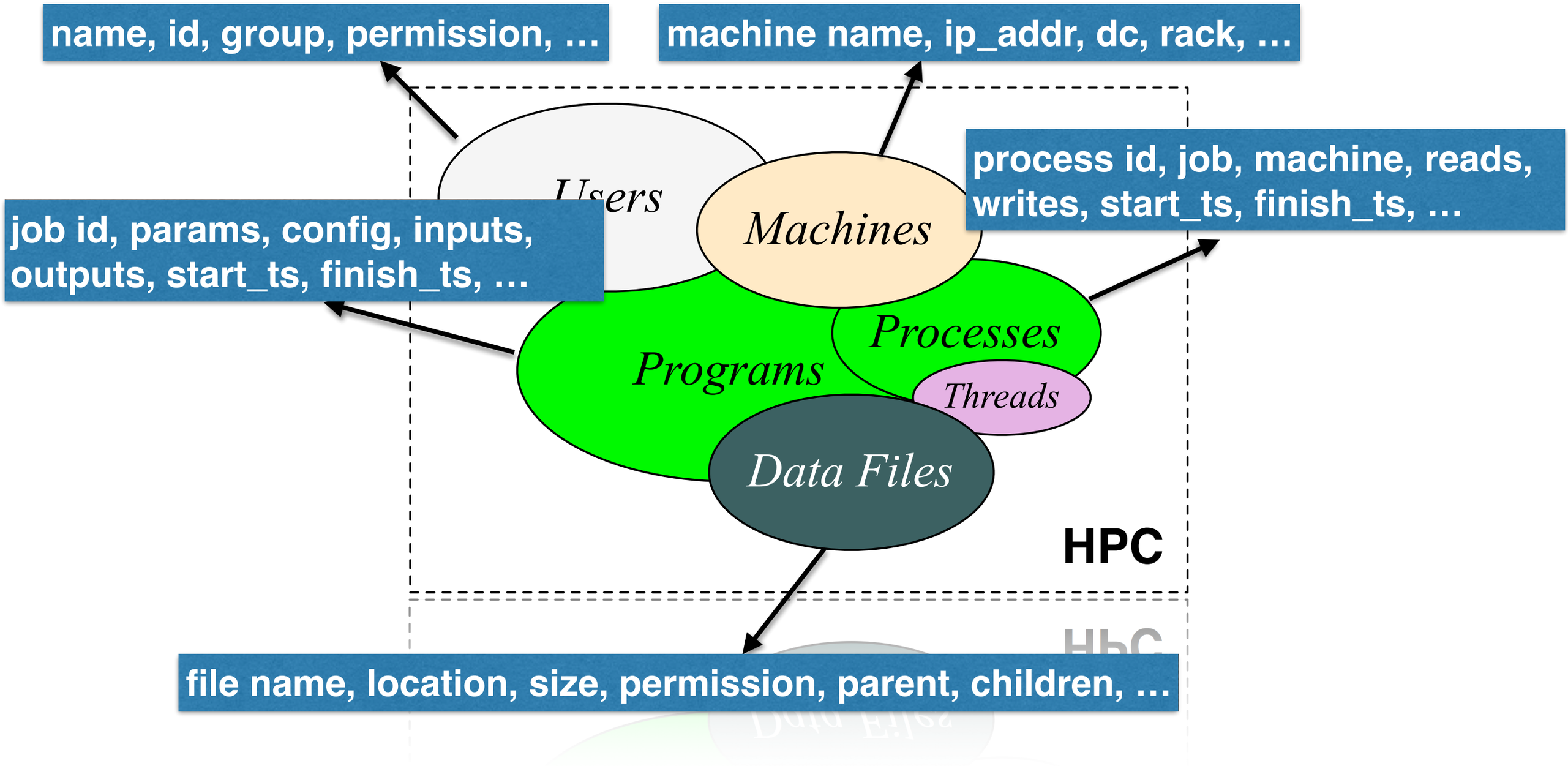
- ***Wf4Ever Research Object Model 1.0, http://wf4ever.github.io/ro/***

Rich Metadata in HPC



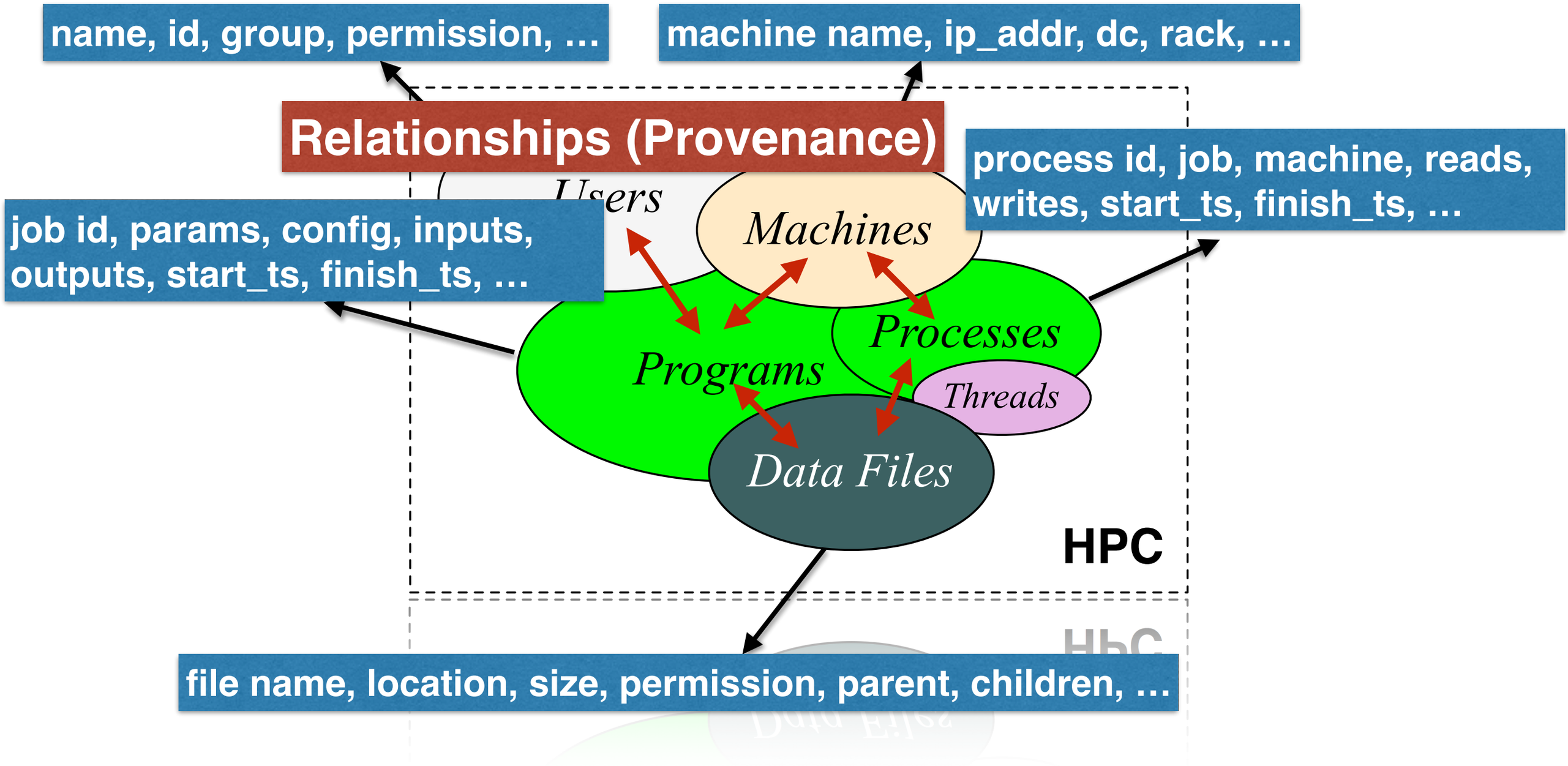
1. *Diverse metadata need to be managed;*
2. *Relationships need to be captured*

Rich Metadata in HPC



1. *Diverse metadata need to be managed;*
2. *Relationships need to be captured*

Rich Metadata in HPC



1. *Diverse metadata need to be managed;*
2. *Relationships need to be captured*

Rich Metadata Challenges

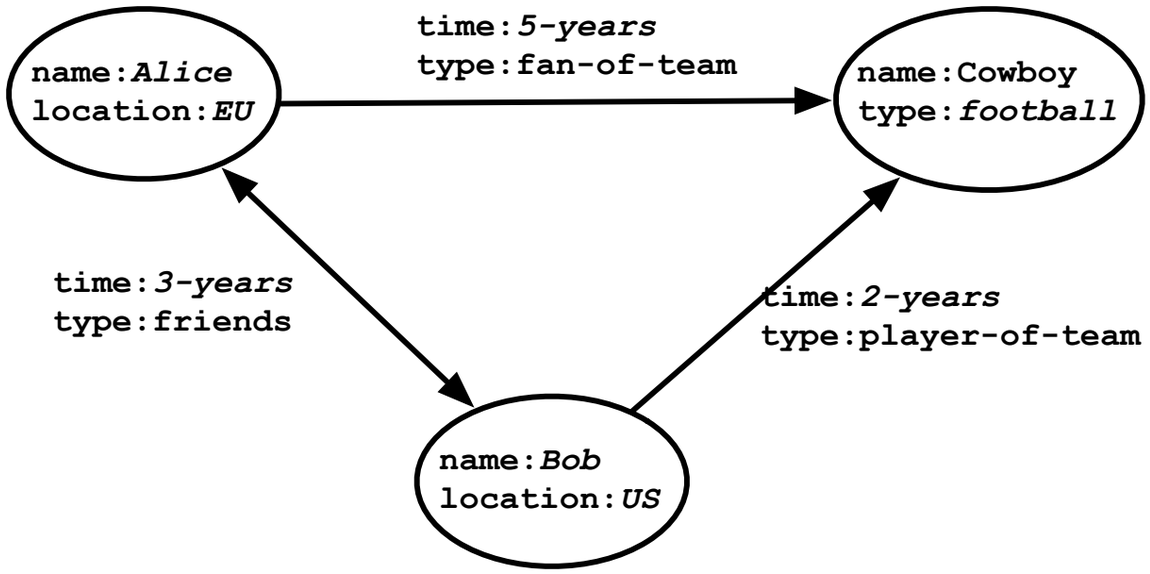
- **Metadata Integration**
  - diverse metadata should be collected from different components
  - diverse metadata should be managed in a unified way

- **Storage System Pressure**
  - large volume of metadata generated from different components
  - high concurrent insert rates from parallel applications

- **Efficient Processing and Querying**
  - some operations exist in the critical execution path of applications
  - some operations require complex query and searching

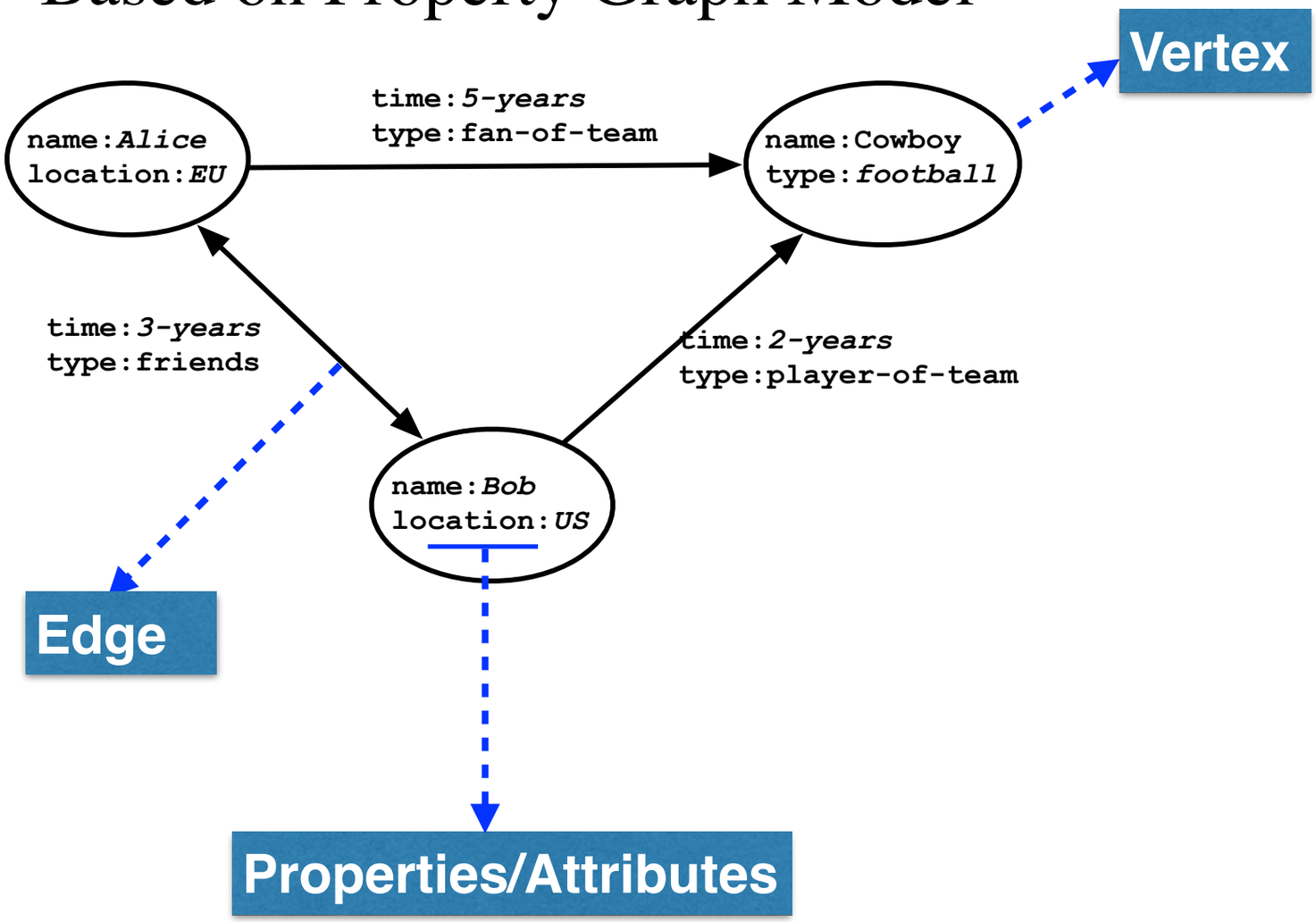
Graph-based Solution

- Based on Property Graph Model



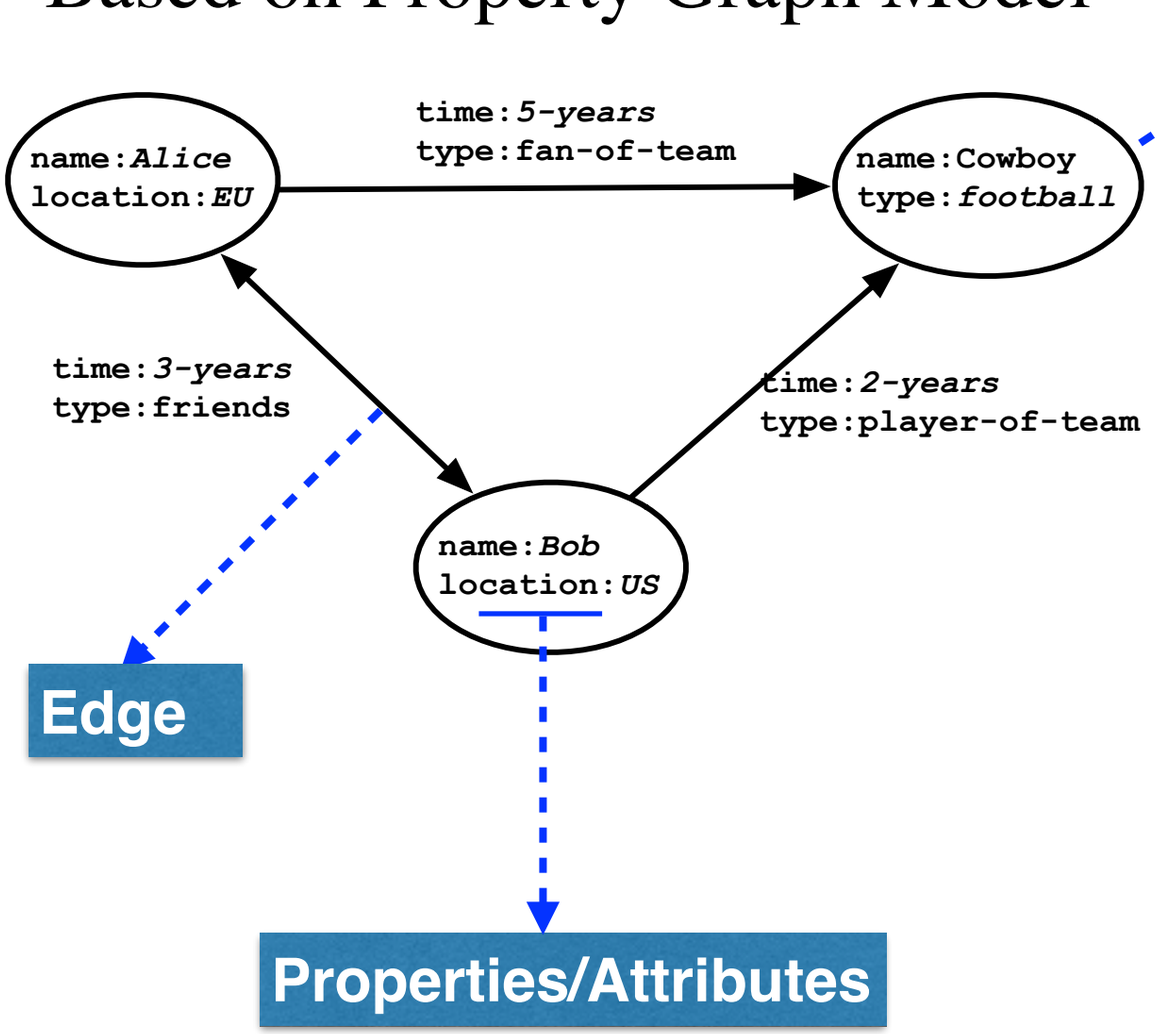
Graph-based Solution

- Based on Property Graph Model



Graph-based Solution

- Based on Property Graph Model

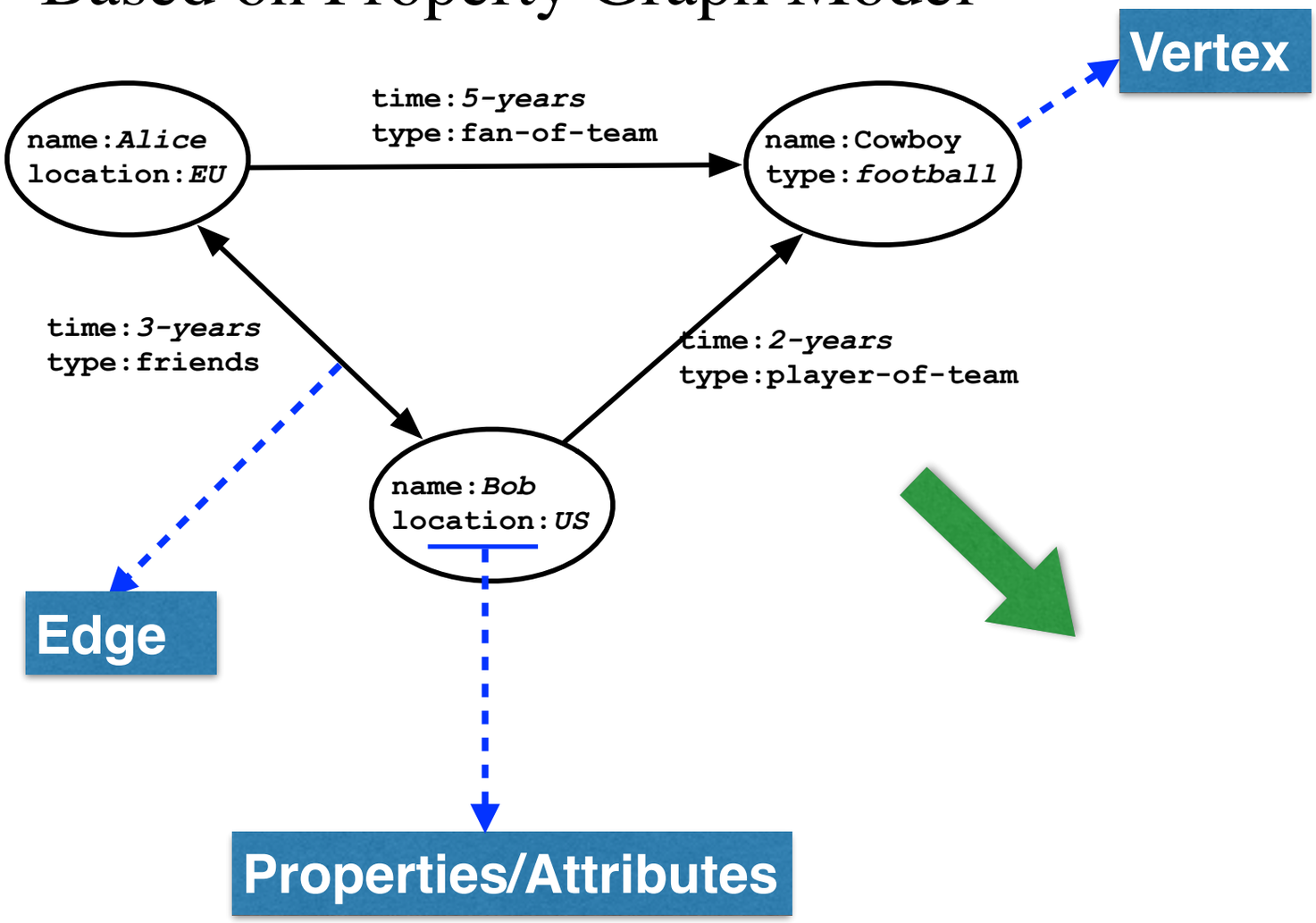


Motivation:

- **Metadata Integration**
- **Storage Pressure**
- **Graph-based Traversal**

Graph-based Solution

- Based on Property Graph Model

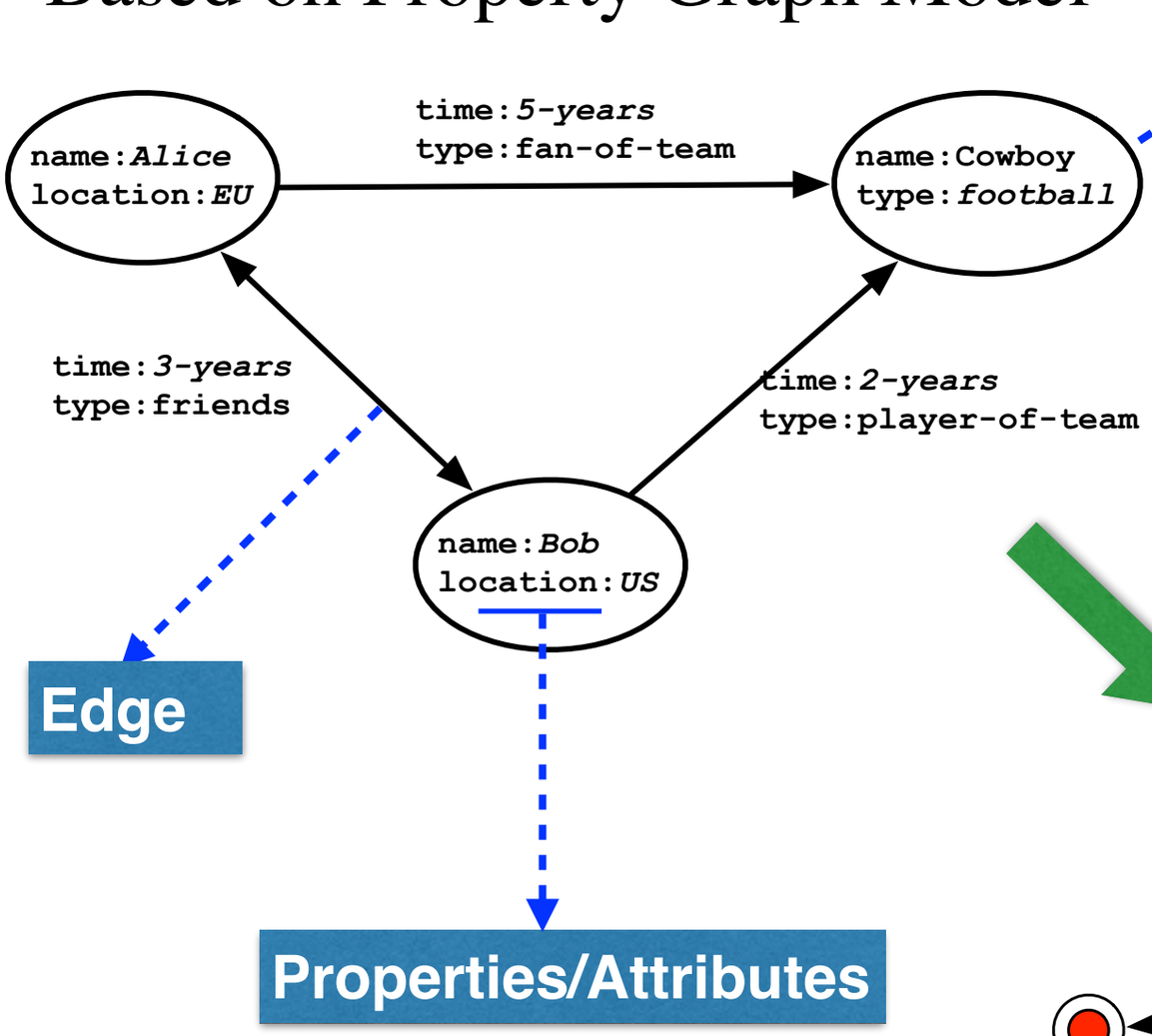


Motivation:

- **Metadata Integration**
- **Storage Pressure**
- **Graph-based Traversal**

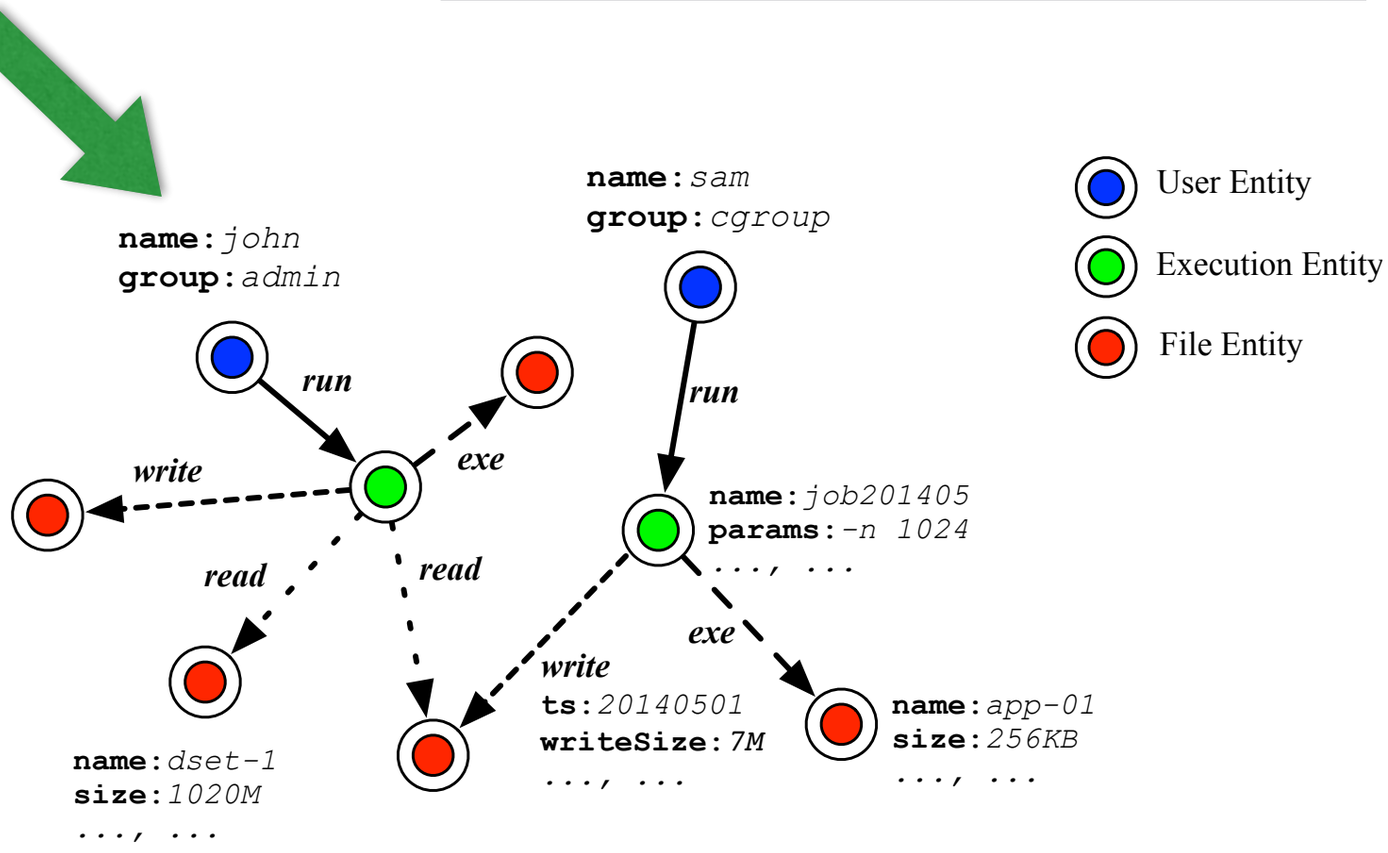
Graph-based Solution

- Based on Property Graph Model



Motivation:

- **Metadata Integration**
- **Storage Pressure**
- **Graph-based Traversal**



Map HPC Metadata to Graph

Map HPC Metadata to Graph

- **Entity => Vertex**
  - Data Object: represents the basic data unit in storage
  - Executions: represents applications including Jobs, Processes, Threads
  - User: represents real end user of a system
  - Users allowed to define their own entities

Map HPC Metadata to Graph

- **Entity => Vertex**
  - Data Object: represents the basic data unit in storage
  - Executions: represents applications including Jobs, Processes, Threads
  - User: represents real end user of a system
  - Users allowed to define their own entities
- **Relationship => Edge**
  - Relationships between different entities are mapped as edges
  - User *runs* Executions. An edge with type '*Run*' is created between them
  - Reversed relationships also are defined
  - Users allowed to define their own relationships

Map HPC Metadata to Graph

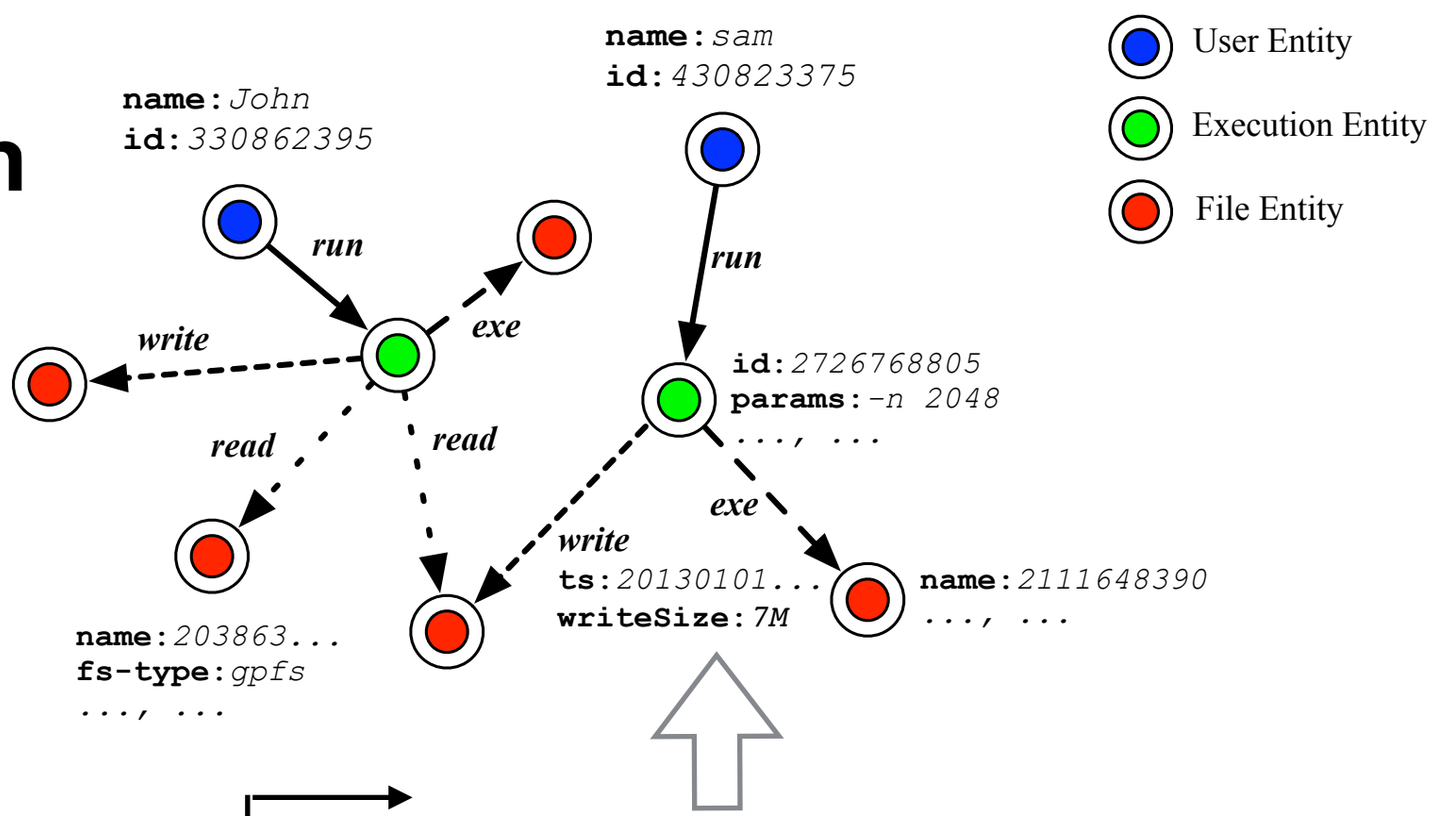
- **Entity => Vertex**
  - Data Object: represents the basic data unit in storage
  - Executions: represents applications including Jobs, Processes, Threads
  - User: represents real end user of a system
  - Users allowed to define their own entities
- **Relationship => Edge**
  - Relationships between different entities are mapped as edges
    - User *runs* Executions. An edge with type '*Run*' is created between them
    - Reversed relationships also are defined
    - Users allowed to define their own relationships
- **Attributes => Property**
  - On both Entity and Relationship
  - Stored as Key-Value pairs attached on vertices and edges

Create an Example Graph

```
# darshan log version: 2.00
# size of file statistics: 1328 bytes
# size of job statistics: 1080 bytes
# exe: 2111648390
# uid: 330862395
# jobid: 2726768805
# start_time: 1357021943
# start_time_asci: Tue Jan  1 00:32:23 2013
# end_time: 1357024059
# end_time_asci: Tue Jan  1 01:07:39 2013
# nprocs: 2048
# run time: 2117
# metadata: proj = 3028319225
# metadata: cn = 512
# metadata: jobid = 2519384738
# metadata: prev_ver = 2.00

#<rank> <file>  <counter>       <value> <name suffix>   <mount pt>      <fs type>
0       2038639327916076341     CP_INDEP_OPENS  0       ...2346818267   /intrepid-fs0   gpfs
0       2038639327916076341     CP_COLL_OPENS   0       ...2346818267   /intrepid-fs0   gpfs
0       2038639327916076341     CP_INDEP_READS  0       ...2346818267   /intrepid-fs0   gpfs
```

Darshan
HPC I/O Characterization Tool



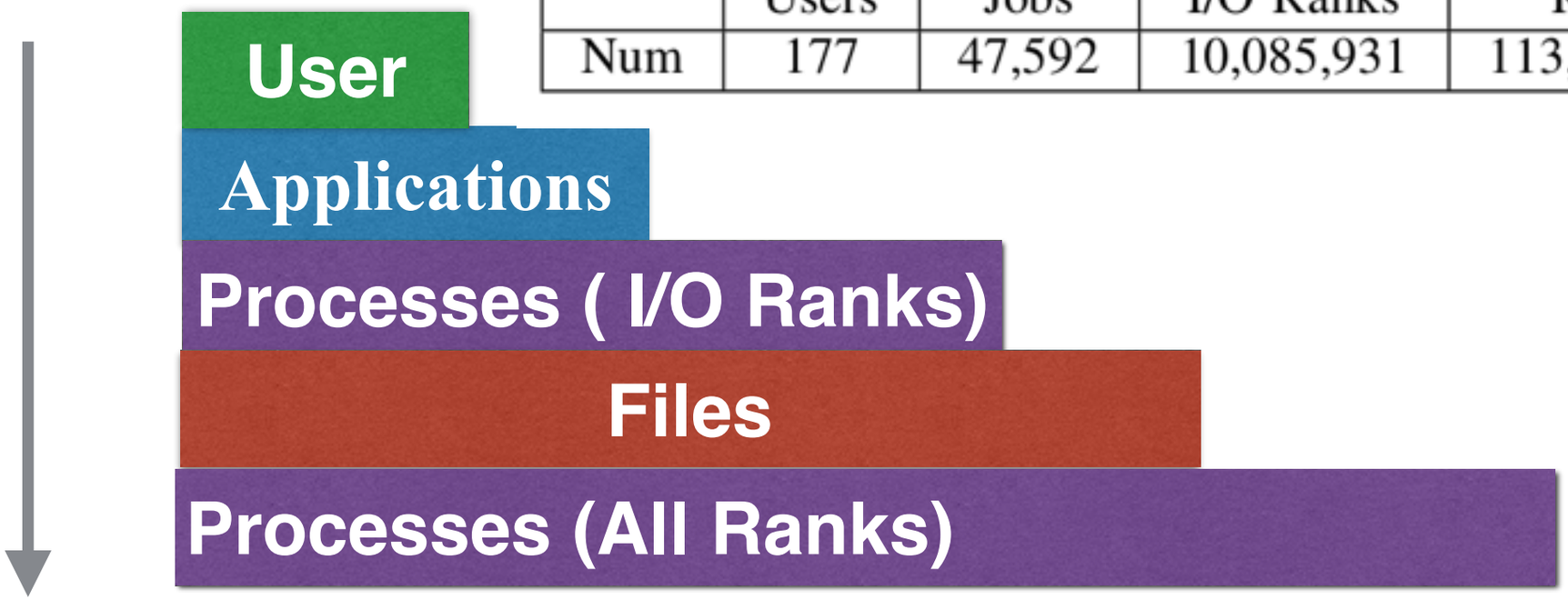
- Each log file => one Job
  - *jobid, start_time, end_time, exe*
- Each uid => one User
- All Ranks => Processes
  - *nprocs, file_access*
- File and exe => Data Object
- Synthetically create directory structure
  - *data files visited by the same execution will be placed under the same directory*
  - *directories accessed by the same user are placed under one directory*

Complete set of logs from Intrepid in 2013

42% of all core-hours consumed in 2013

Sample Graph: Size

detailed level



	Users	Jobs	I/O Ranks	Ranks	Files
Num	177	47,592	10,085,931	113,278,038	34,608,033

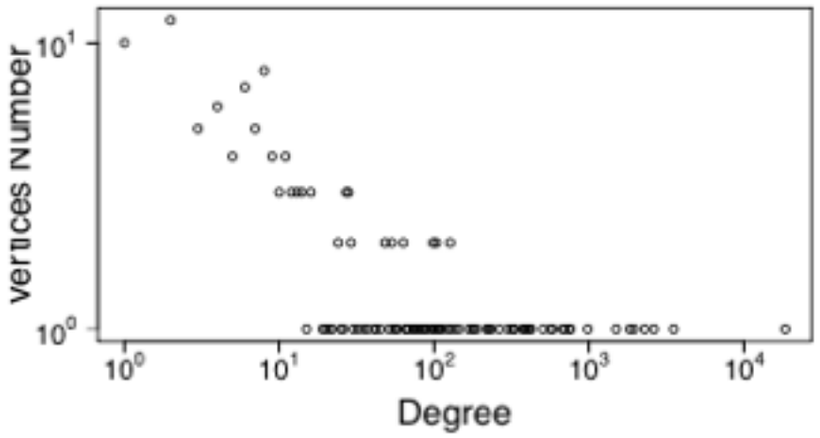
Number	Basic	With I/O Ranks	With Full Ranks	With Directory
Vertices	34.6 M	41.7 M	147.8 M	147.9 M
Edges	126.5 M	133.6 M	239.8 M	366.3 M

	Road Graph USA [1]	Twitter [2]	Facebook [3]	Web Page (2002) [4]
Vertices	24 M	645 M	1.28 B	2.1 B
Edges	29 M	81.4 B	256 B	15 B

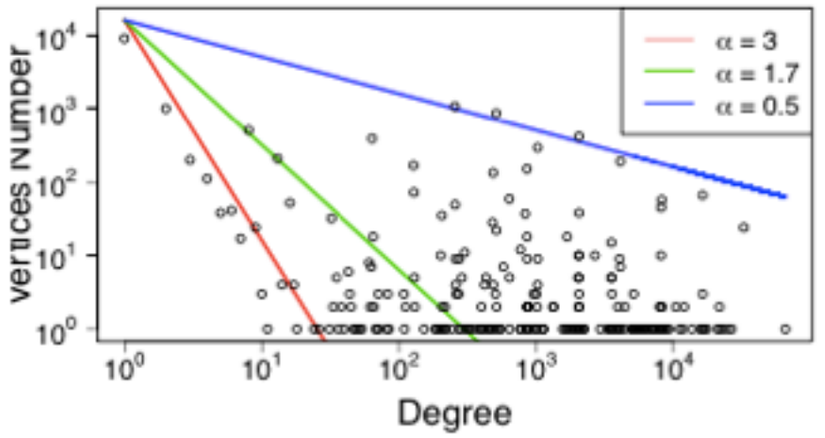
M = million; B = billion; T = trillion

Sample Graph: Structure

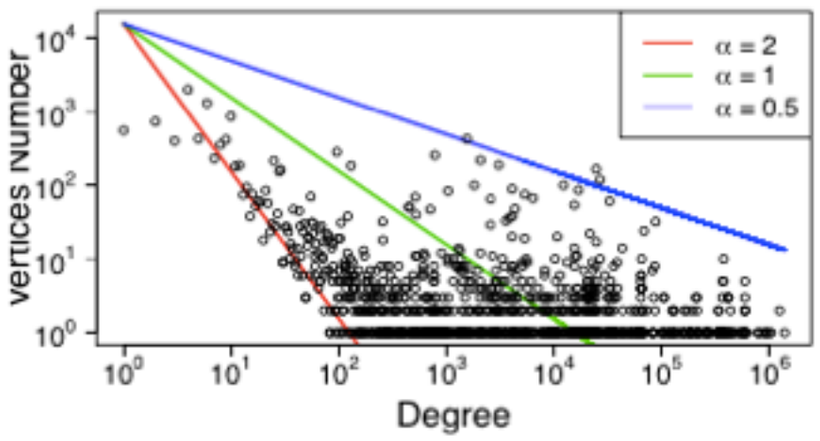
- Common Attribute
  - most entities have small degree
  - small number of entities have much **huge** degree
- Skewed power-law distribution
  - many nature graphs belong to this category
  - obey: $\mathbf{P}(d) \propto d^{-\alpha}$
  - Further investigation also confirm they fit the power-law distribution



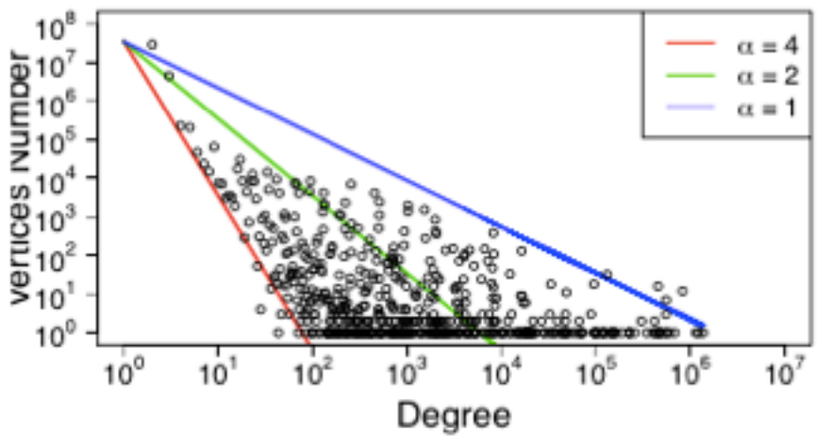
(a) User node degree



(b) Job node degree



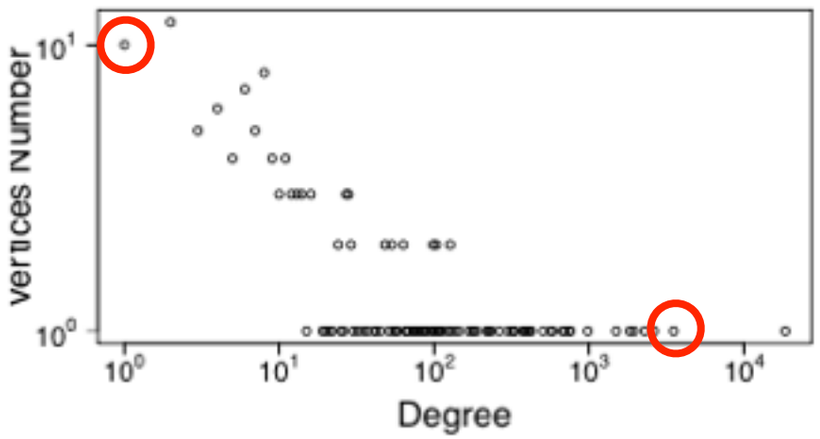
(c) Process node degree



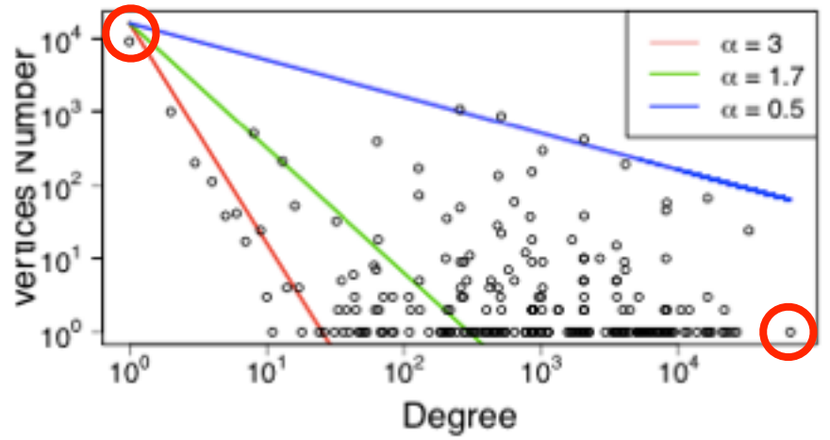
(d) File node degree

Sample Graph: Structure

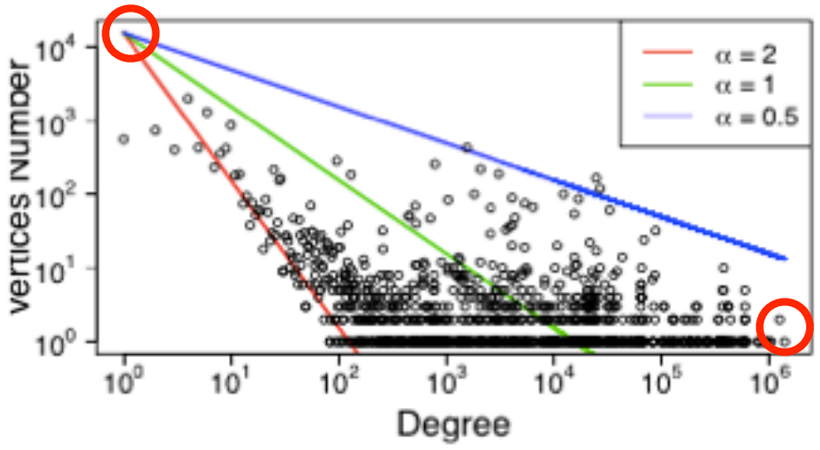
- Common Attribute
  - most entities have small degree
  - small number of entities have much **huge** degree
- Skewed power-law distribution
  - many nature graphs belong to this category
  - obey: $\mathbf{P}(d) \propto d^{-\alpha}$
  - Further investigation also confirm they fit the power-law distribution



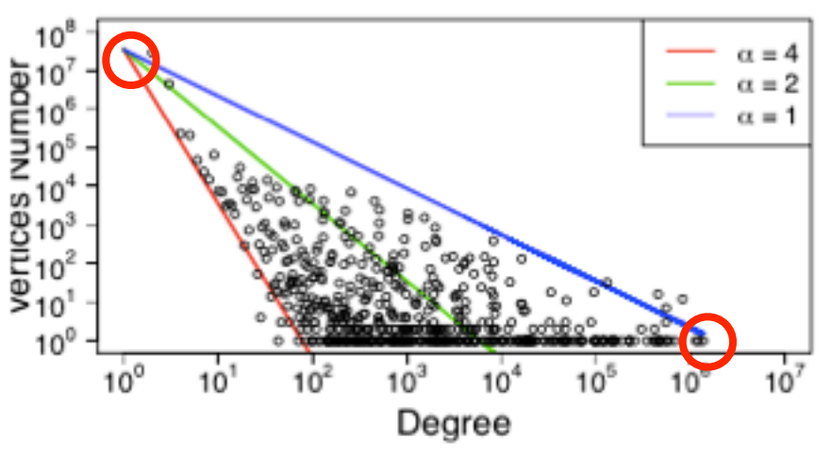
(a) User node degree



(b) Job node degree



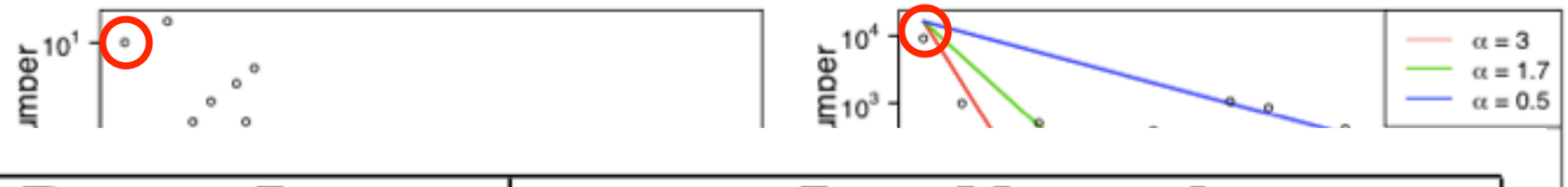
(c) Process node degree



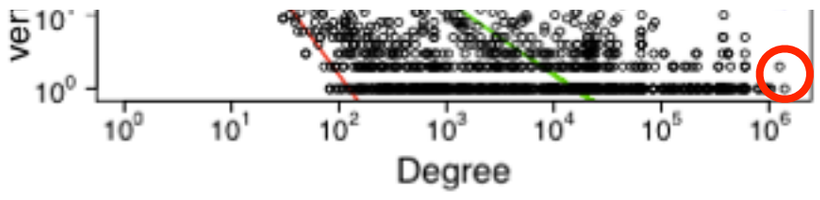
(d) File node degree

Sample Graph: Structure

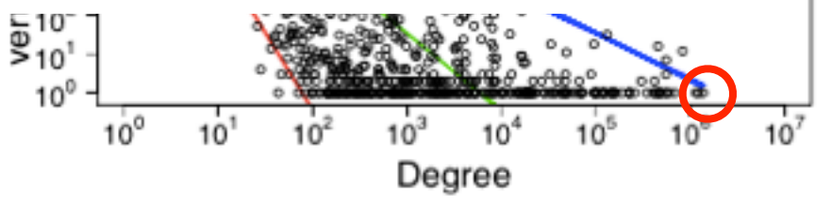
- Common Attribute
  - most entities have small
  - obey: $P(d) \propto d$
  - Further investigation also confirm they fit the power-law distribution



	Power-Law Params.		Log-Normal Params.		
	x_{min}	α	x_{min}	μ	σ
User Degree	53	1.713	2	2.963	2.301
Process Degree	1537	1.751	74	7.798	1.825
File Degree	2	4.796	1	0.7833	0.3253
Power-Law	7	1.952	3	-17.937	4.869



(c) Process node degree



(d) File node degree

Operations on the Graph: Namespace Traversal

- **Hierarchical Namespace Traversal**

  - Present logical layout of data sets to users
    - traditional POSIX-style tree-structure directory

  - The metadata graph already contains
    - *belongs/contains* relationships between Data Objects vertices
    - directory can be considered as Data Object entity too

- ***locate files by given path***
  1. locate the root directory in the graph
  2. repeatedly travel through *contains* edges from *directory* vertices to directory or files vertices

Locate -> Traversal -> Filter -> Traversal

Operations on the Graph: Data Audit

- **Data Audit**
  - The metadata graph already contains
    - *run* relationships between Users and Executions
    - *read/write* relationships between Executions and Data Objects
    - additional *attributes* are also recorded with these relationships
  - ***locate files accessed by a specific user in a given time frame***
    1. locate the given user in the graph
    2. travel through *run* edges from User to Execution
    3. filter execution based on the time frame
    4. travel through *read* edges from Executions to Data Objects

Locate -> Traversal -> Filter -> Traversal

Operations on the Graph: Provenance Search

Provenance Challenge

- **Provenance Support**
  - Wide range of use cases
    - data sharing, reproducibility, work-flow
  - The metadata graph already contains
    - *Relationships* between different entities
    - User-defined attributes and relationships

#8 Problems: Given a *fMRI* workflow with multiple stages processing.

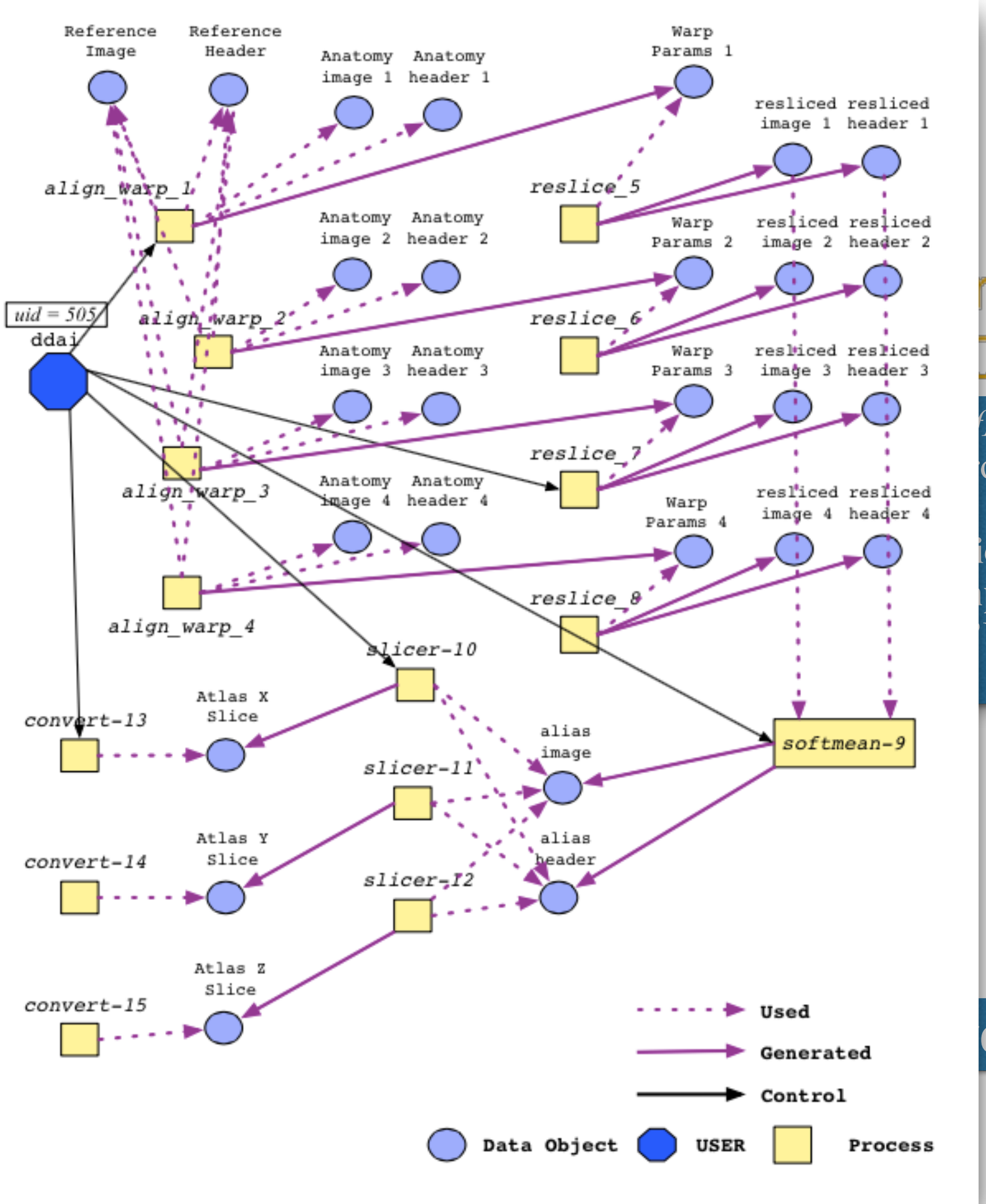
Try to find the **Execution whose model is 'AlignWarp' and inputs have annotation ['center':'UChicago']**

- ***#8 in the first Provenance Challenge***
  1. Use graph to abstract the workflow executions
  2. Search all Executions with model "AlignWarp"
  3. Travel through *read* edges to Data Objects entities
  4. Filter based on property 'center' ('UChicago')

Search Attributes -> Traversal -> Filter -> Traversal

Operati

- **Provenance**
  - Wide range
    - data sha
  - The metada
    - *Relatio*
    - User-de
  - ***#8 in the fi***
    1. Use grap
    2. Search a
    3. Travel th
    4. Filter bas



rovenance
Challenge

fMRI workflow
ocessing.

ion whose model
puts have
'UChicago']

Se **ersal**

TEXAS TECH UNIVERSITY

Argonne NATIONAL LABORATORY

Operations on the Graph: Provenance Search

Provenance Challenge

- **Provenance Support**
  - Wide range of use cases
    - data sharing, reproducibility, work-flow
  - The metadata graph already contains
    - *Relationships* between different entities
    - User-defined attributes and relationships

#8 Problems: Given a *fMRI* workflow with multiple stages processing.

Try to find the **Execution whose model is 'AlignWarp' and inputs have annotation ['center':'UChicago']**

- ***#8 in the first Provenance Challenge***
  1. Use graph to abstract the workflow executions
  2. Search all Executions with model "AlignWarp"
  3. Travel through *read* edges to Data Objects entities
  4. Filter based on property 'center' ('UChicago')

Search Attributes -> Traversal -> Filter -> Traversal

Requirements

Read

- **Search/Locate, Travel, Filter Pattern**
  - Search graph vertices and edges by their attributes => **Indexing**
  - Fast locate vertices and edges by global ID => **Partitioning**
  - Efficient multi-step traversal in large graph => **Traversal Speed**
  - Customized filter function during traversal => **Filtering**

Write

- **HPC Environment**
  - **High Volume:** rich metadata are actually ‘big data’
  - **Lots of Clients:** millions of cores generate metadata concurrently
  - **High Contention:** clients modify the same vertex or edge at the same time
    - Creating files under the same directory
    - All applications read/write the same file

On-going Work

Existing Graph Infrastructure

Graph Databases

sones

InfiniteGraph
Powered by Objectivity

*dex

Neo4j

OrientDB

RDF

bigdata

AllegroGraph

HYPERGRAPHDB

TITAN

Graph Processing Frameworks

APACHE GIRAPH

Google Pregel

GraphLab

GraphX

X-Stream

Proposed Solutions

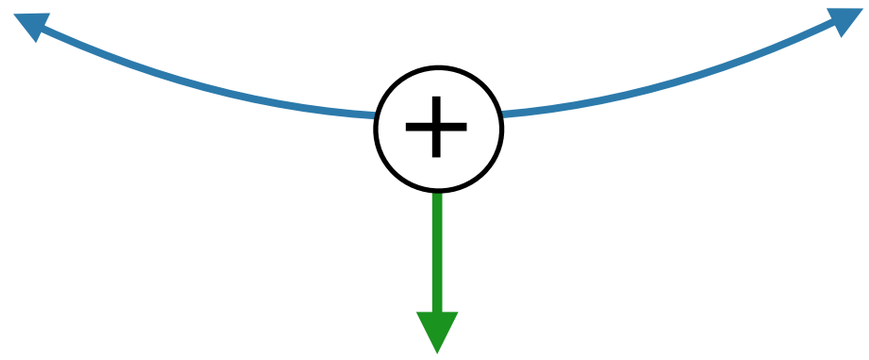
On-going Work

Basic Needs

- Property Graph Model
- Distributed Writes/Reads
- User-defined Indexing
- Graph Traversal

Performance Requirements

- High Contention Writes
- Efficient Graph Traversal
- Consider the Graph Structure



Proposed Solution

Existing Graph Infrastructure (Titan + Cassandra)

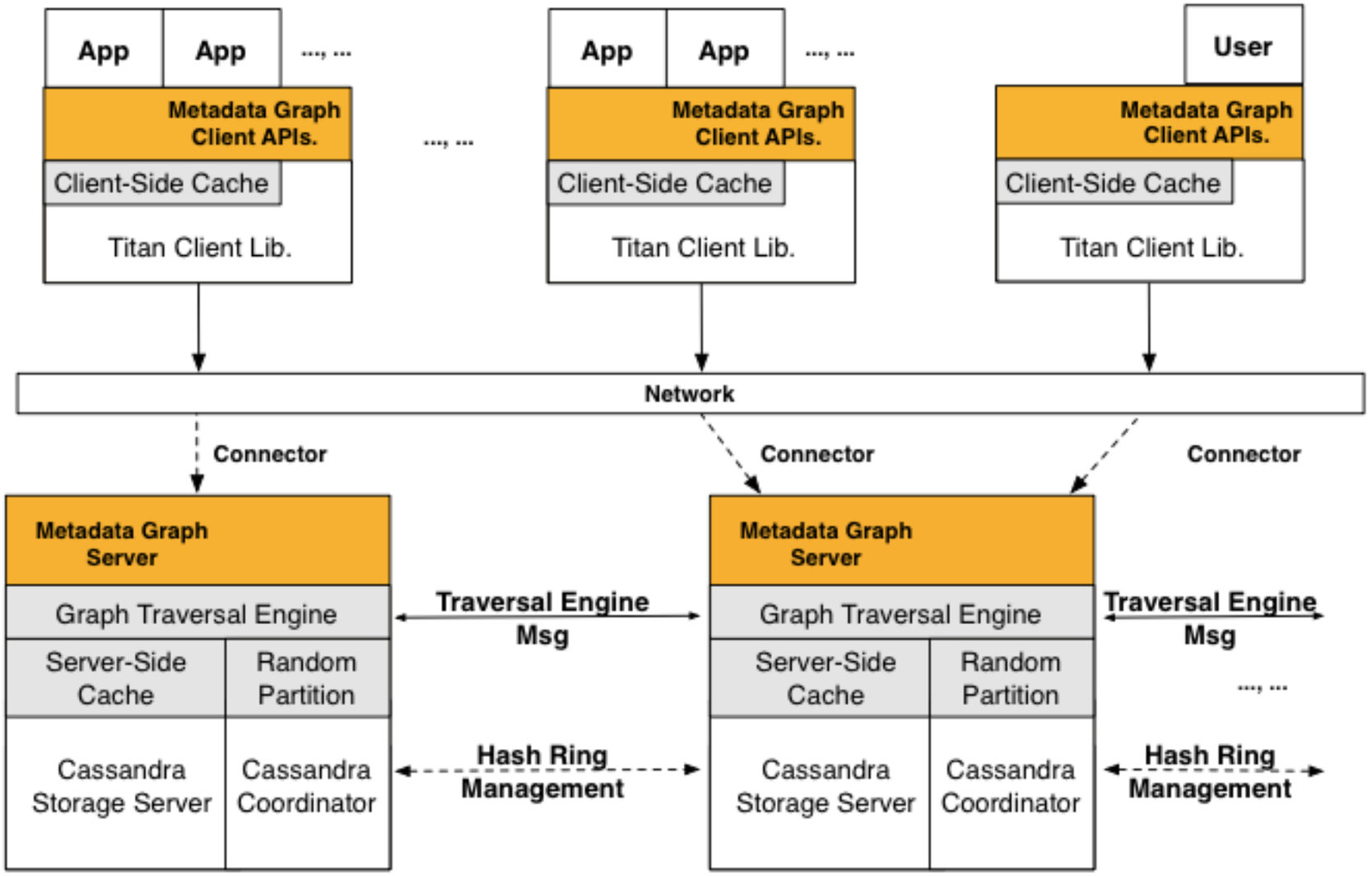
+

- Burst Write Partition Strategy
- Fast Server-side Traversal
- Caching strategy for power-law like graphs

On-going Work

Prototyped Graph Infrastructure

- Table-based Abstraction
- SQL-Like APIs
- Caching Strategy
- Partition For Burst Write
- Fast Server-side Traversal



Conclusion

- We observed that a property graph representation seems to be a good match for rich metadata in HPC storage
- We generated an example metadata property graph using access data from a real, large-scale system over a year period
- We observed properties of this graph, compared it to graphs in other contexts, and identified some challenges for processing these graphs for HPC metadata storage

References

[1] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, The Shortest Path Problem: Ninth DIMACS Implementation Challenge. American Mathematical Soc., 2009, vol. 74.

[2] "Twitter Statistics," http://www.statisticbrain.com/twitter-statistics/.

[3] A. Ching, "Giraph: Production-Grade Graph ProcessingInfrastructure for Trillion Edge Graphs," in ATPESC, ser.ATPESC '14, 2014.

[4] J.-L. Guillaume, M. Latapy et al., "The Web Graph: anOverview," in Actes d'ALGOTEL'02, 2002.

[5] A. Leung, I. Adams, and E. L. Miller, "Magellan: A Searchable Metadata Architecture for Large-Scale File Systems," *University of California, Santa Cruz, Tech. Rep. UCSC-SSRC-09-07*, 2009.

[6] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. My- ers *et al.*, "The Open Provenance Model Core Specifi- cation (v1. 1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[7] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A Ccalable, High-Performance Distributed File System," in Proceedings of the 7th symposium on Operating Systems Design and Implemen- tation. USENIX Association, 2006, pp. 307–320.

Thanks & Questions