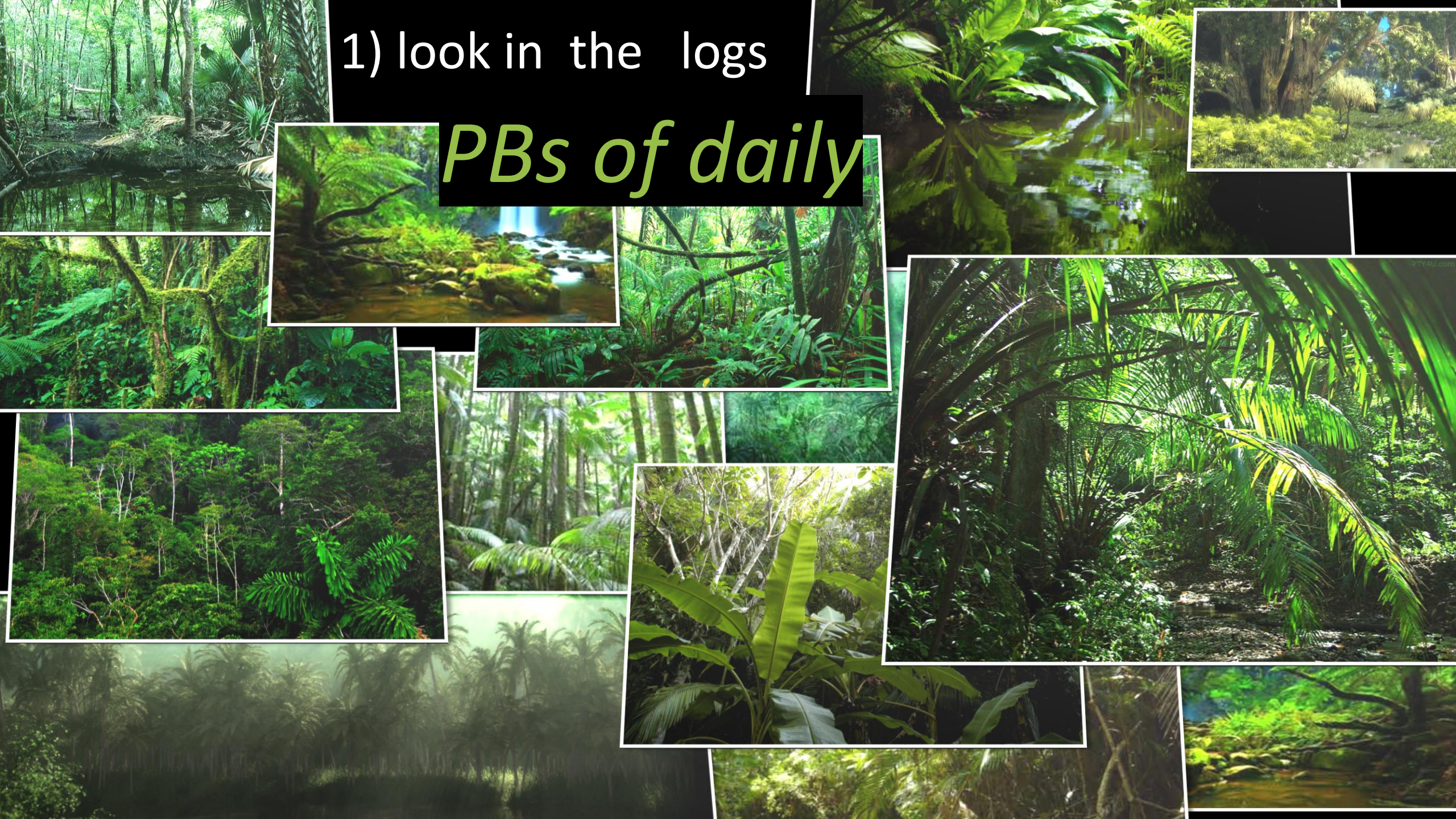
Dependency Driven Analytics

a Compass for Uncharted Data Oceans/Jungles

Ruslan Mavlyutov, Carlo Curino, Boris Asipov, Phil Cudre-Mauroux

The production job *“JobA”* failed…
impact? debug? re-run?


1) look in the logs
PBs of daily

2) ask local experts
(they know “how” to look)

But don’t bother them too much…

The Problem

Focused analyses of massive, loosely structured, evolving data has prohibitive cognitive and computational costs.

The Problem

Focused analyses of massive, loosely structured, evolving data has prohibitive cognitive and computational costs.

Cost of understanding raw data

Cost of processing raw data

A better vantage point?

Dependency Driven Analytics (DDA)

DDA today

DDA vision

- Derive a *dependency graph (DG)* from raw data

The *DG* serve as:

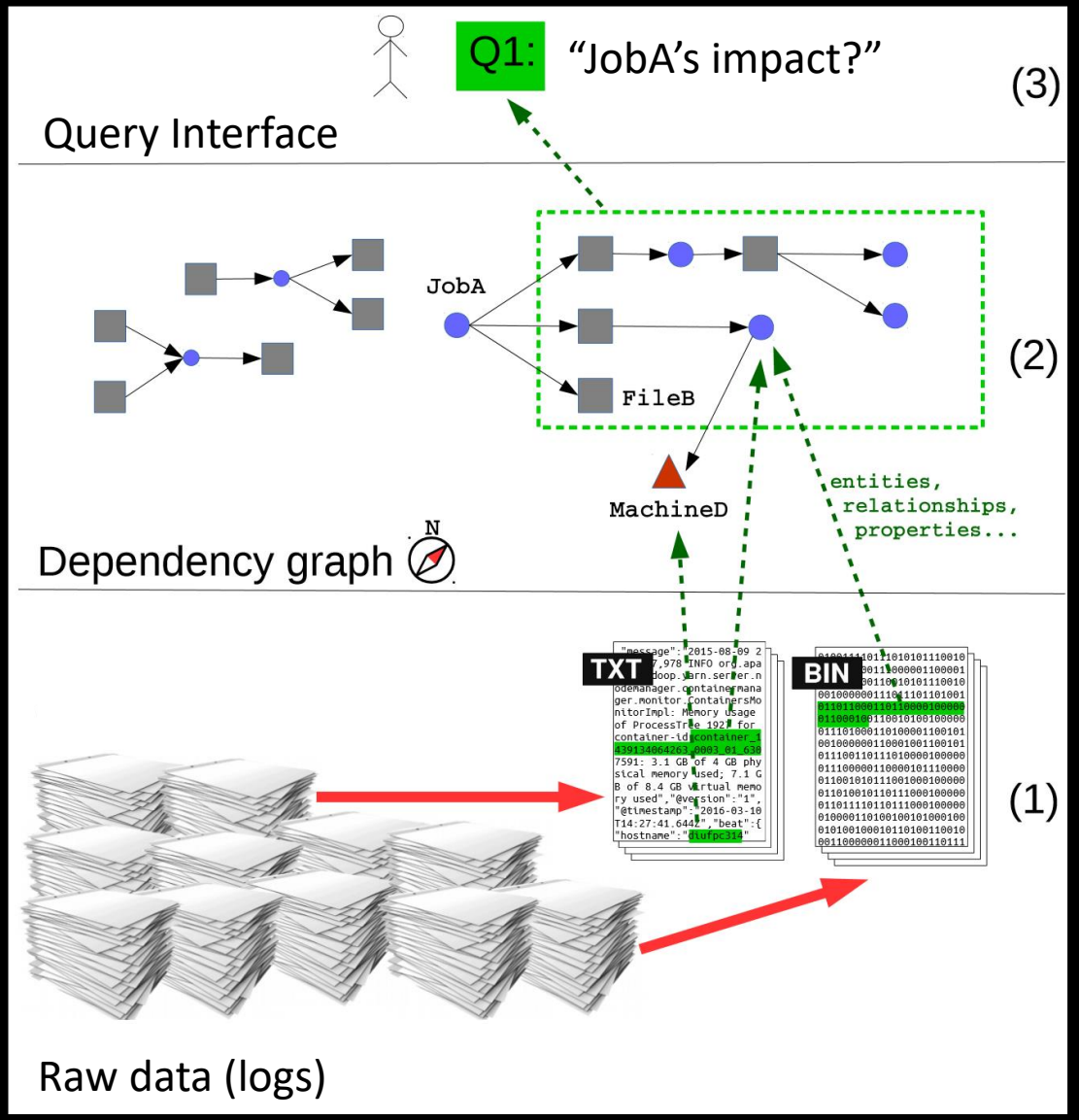
- *Conceptual Map*, and
- *Sparse Index* for the raw data

- *Automation*
- *Language-integration*
- *Real-time*
- *…*

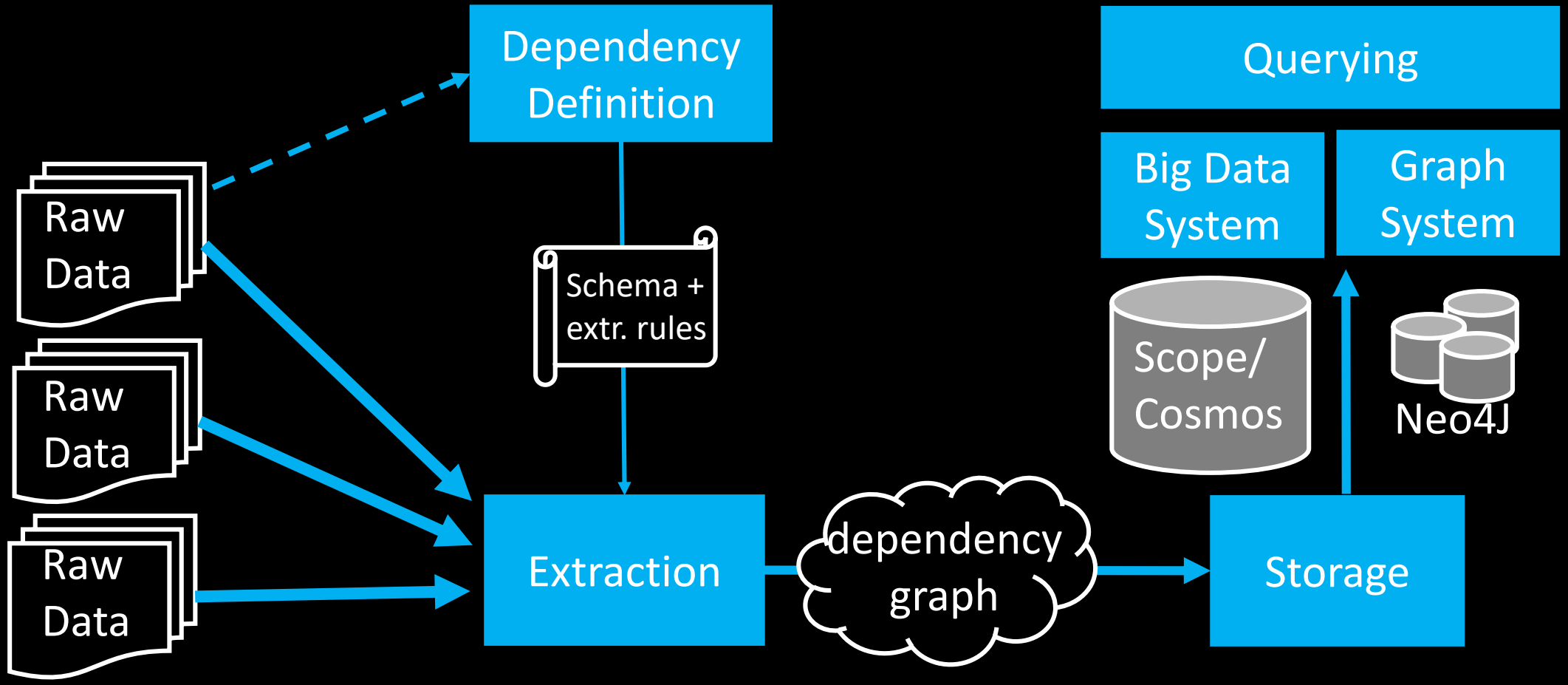
DDA: infrastructure logs “incarnation”

- The DG stores:
  provenance + telemetry

- NODES: jobs / files / machines / tasks / …
- EDGES: job-reads-file, task-runs-on-machine
- PROPERTIES: timestamps / resources usage / …



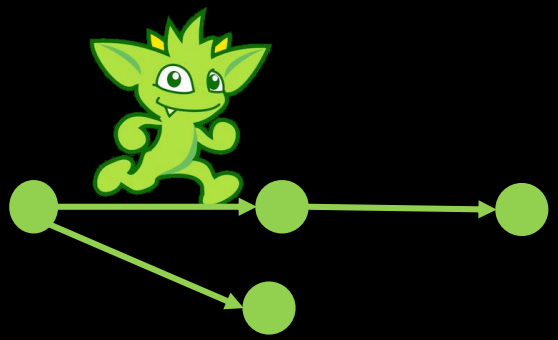
Current implementation

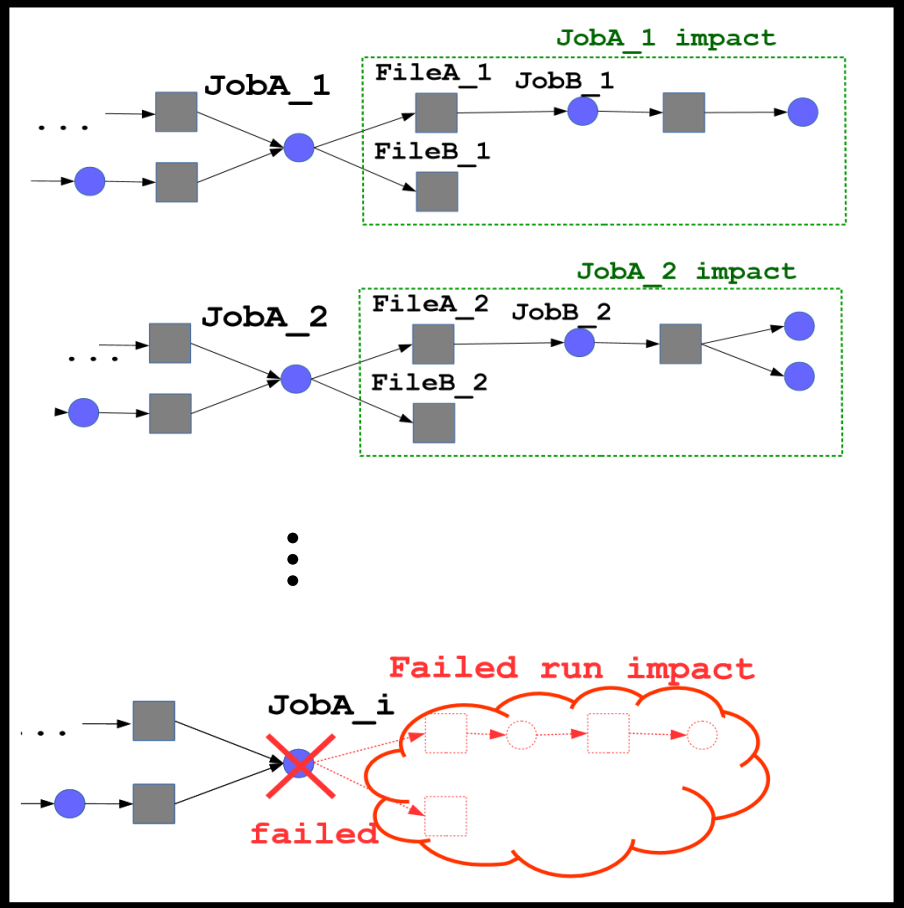


Extract *"jobs processing hours"*

```
extStart =  EXTRACT * FROM "ProcStarted_%Y%m%d.log"
            USING EventExtractor("ProcStarted");

startData = SELECT ProcessGuid AS ProcessId,
                   CurrentTimeStamp.Value AS StartTime,
                   JobGuid AS JobId
            FROM  extStart
            WHERE ProcessGuid != null AND JobGuid != null AND
                  CurrentTimeStamp.HasValue;

…

procH = SELECT endData.JobId,
                SUM((End - Start).TotalMs)/1000/3600 AS procHours,
        FROM startData INNER JOIN endData ON startData.ProcessId ==
             endData.ProcessId AND startData.JobId == endData.JobId
        GROUP BY JobId;

OUTPUT (SELECT JobId, procHours FROM procH) TO "processingHours.csv";
```

Example: “Measure JobA’s impact”






```
graph.traversal().V()
    .has("JobTemplateName","JobA_*")
    .local(
        emit().repeat(out()).times(100)
       .hasLabel("job").dedup()
       .values(“procHours").sum()
    ).mean()
```

DDA: Initial Experiments

Improvements of up to:

- 7x less LoC*
- 700x less run-time
- > 50,000x less CPU-time
- > 800x less I/O

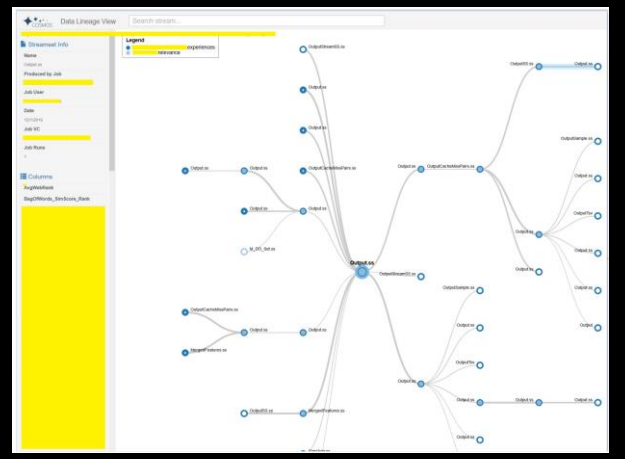
Q3: "Average Count of inputs per Job"			
	Raw	Graph on DFS	Neo4J
LoC	50	25	7
Run-time	58min	11min	4.9sec
CPU-time	563h	25h	<40sec
IOs	18.6TB	61GB	<24GB (RAM)

Q4: "Ratio of recurring vs Ad-hoc Jobs"			
	Raw	Graph on DFS	Neo4J
LoC	22	21	3
Run-time	8min	24min	10.9sec
CPU-time	41h	14h	<90sec
IOs	966GB	638MB	<24GB

* Heavy under-representation of hardness of baseline

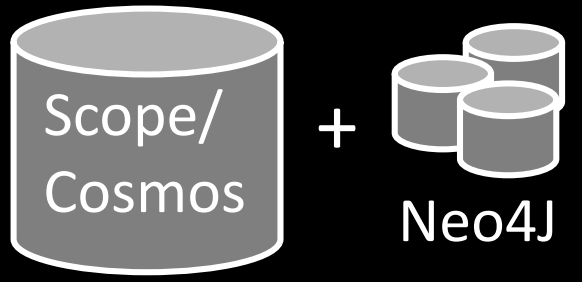
Not all queries are as easy…

Simple search/browsing → UI (keyword search)



Local or agg. queries on telemetry / provenance → Graph queries on DG (i.e., covering index)

Complex/AdHoc queries (e.g., debugging) → Mix of DG and raw data querying (clumsy today)



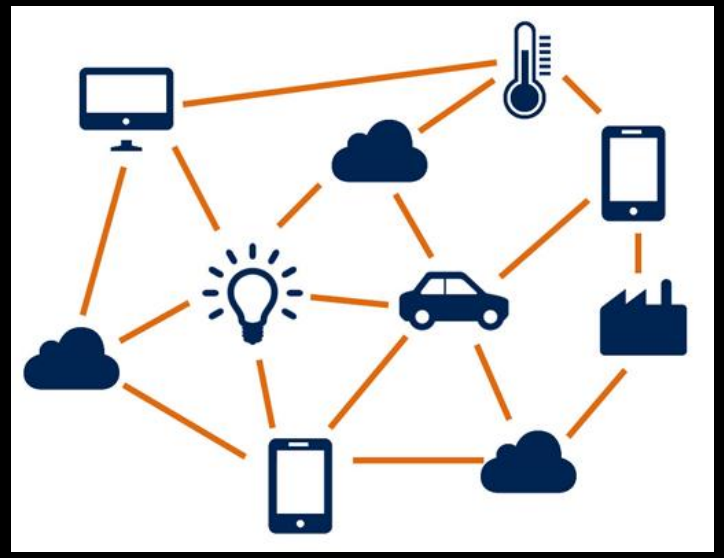
DDA: open challenges

- Automatically *“map”* the raw data
- Real-time log ingestion at scale
- Scale-out graph management
  - Leverage specialized graph structures
- Integrated language for graph+relational+unstructured

Scope

Infrastructure logs

Internet of Things



Enterprise Search

...

Conclusions

Problem:

- Focused analyses of massive, loosely structured, evolving data has prohibitive costs

DDA solution:

- Extract a *Dependency Graph (DG)* → *conceptual map + sparse index*
- Current impl. leverages existing BigData/Graph tech

Open challenges:

- automation / real-time / scalable graph tech / integrated language