Signal Processing on Databases

Jeremy Kepner

Lecture 0: Introduction

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

This work is sponsored by the Department of the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

Acknowledgements

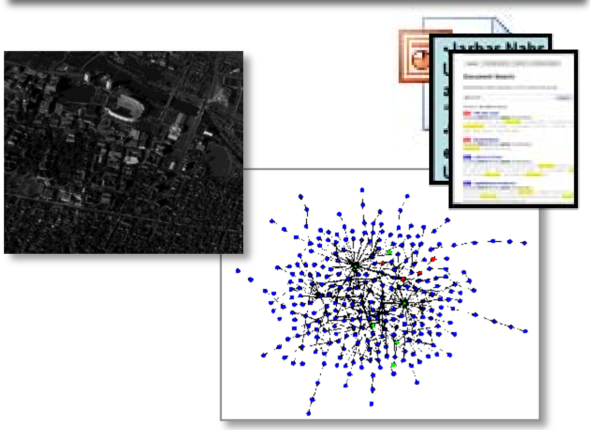
- Nicholas Arcolano
- Michelle Beard
- Nadya Bliss
- Josh Haines
- Matthew Schmidt
- Ben Miller
- Benjamin O'Gwynn
- Tamara Yu
- Bill Arcand
- Bill Bergeron
- David Bestor
- Chansup Byun
- Matt Hubbell
- Pete Michaleas
- Julie Mullen
- Andy Prout
- Albert Reuther
- Tony Rosa
- Charles Yee
- Dylan Hutchinson

Outline

- Introduction
- Course Outline
- Example Implementation
- Summary

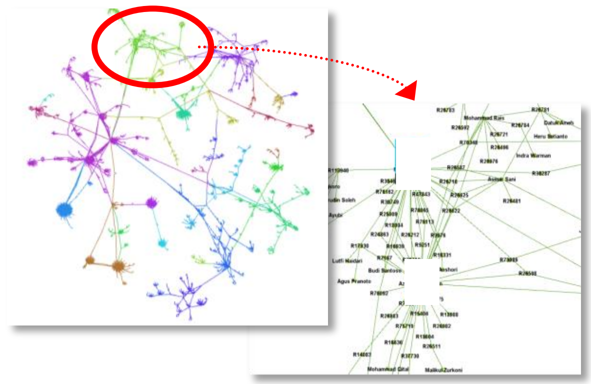
Example Applications of Graph Analytics

ISR



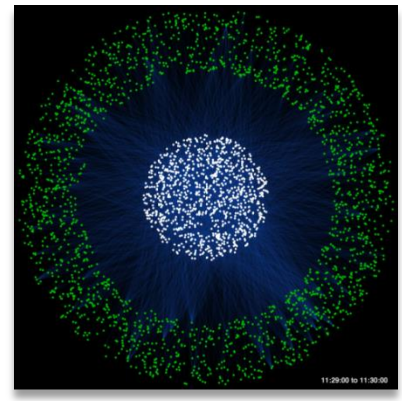
- Graphs represent entities and relationships detected through multi-INT sources
- 1,000s – 1,000,000s tracks and locations
- GOAL: Identify anomalous patterns of life

Social



- Graphs represent relationships between individuals or documents
- 10,000s – 10,000,000s individual and interactions
- GOAL: Identify hidden social networks

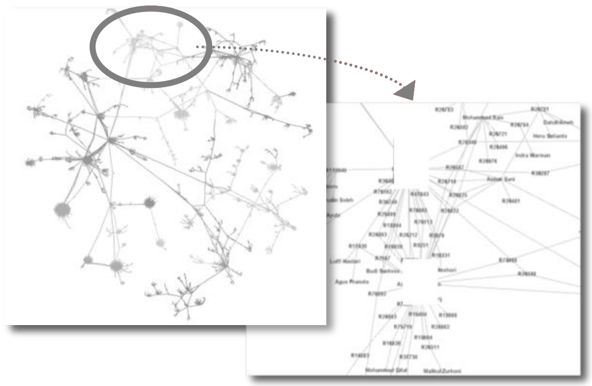
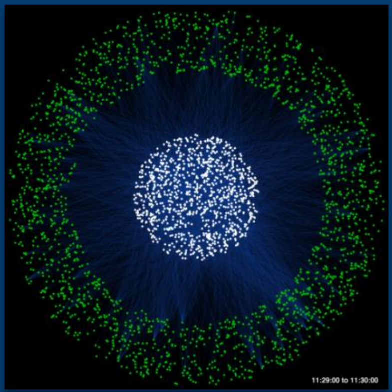
Cyber



- Graphs represent communication patterns of computers on a network
- 1,000,000s – 1,000,000,000s network events
- GOAL: Detect cyber attacks or malicious software

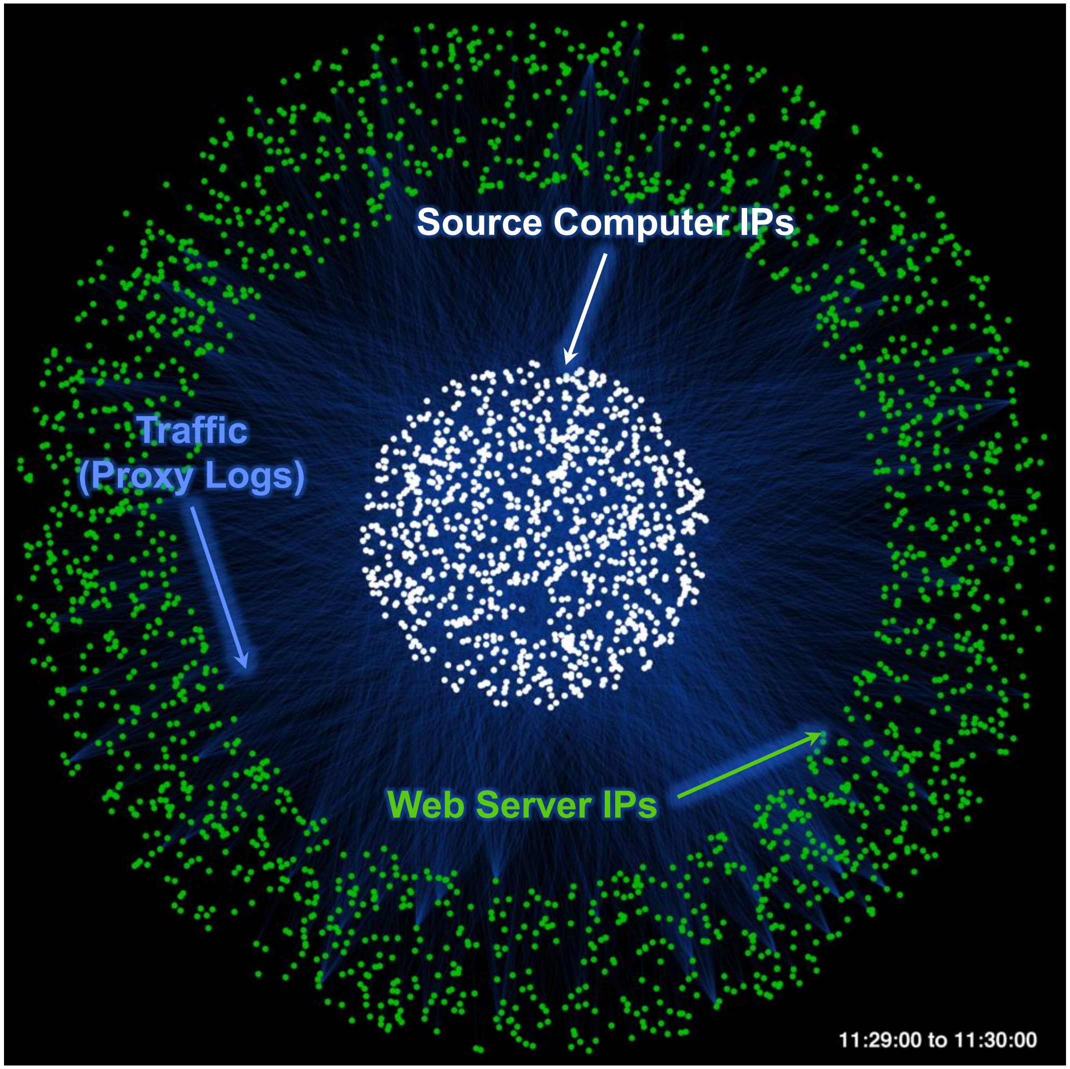
- Cross-Mission Challenge: Detection of subtle patterns in massive multi-source noisy datasets

Example Applications of Graph Analytics

ISR	Social	Cyber
		
• Graphs represent entities and relationships detected through multi-INT sources • 1,000s – 1,000,000s tracks and locations • GOAL: Identify anomalous patterns of life	• Graphs represent relationships between individuals or documents • 10,000s – 10,000,000s individual and interactions • GOAL: Identify hidden social networks	• Graphs represent communication patterns of computers on a network • 1,000,000s – 1,000,000,000s network events • GOAL: Detect cyber attacks or malicious software

- **Cross-Mission Challenge: Detection of subtle patterns in massive multi-source noisy datasets**

Example: Web Traffic Graph



Graph Statistics

- 90 minutes worth of traffic
- 1 frame = 1 minute of traffic
- Number of source computers: 4,063
- Number of web servers: 16,397
- Number of logs: 4,344,148

Malicious Activity Statistics

- Number of infected IPs: 1
- Number of event logs: 16,000
- % infected traffic: 0.37%
- Existing tools did not detect event
- Detection took *10 days* and required manual log inspection

Challenge: Activity signature is typically a weak signal

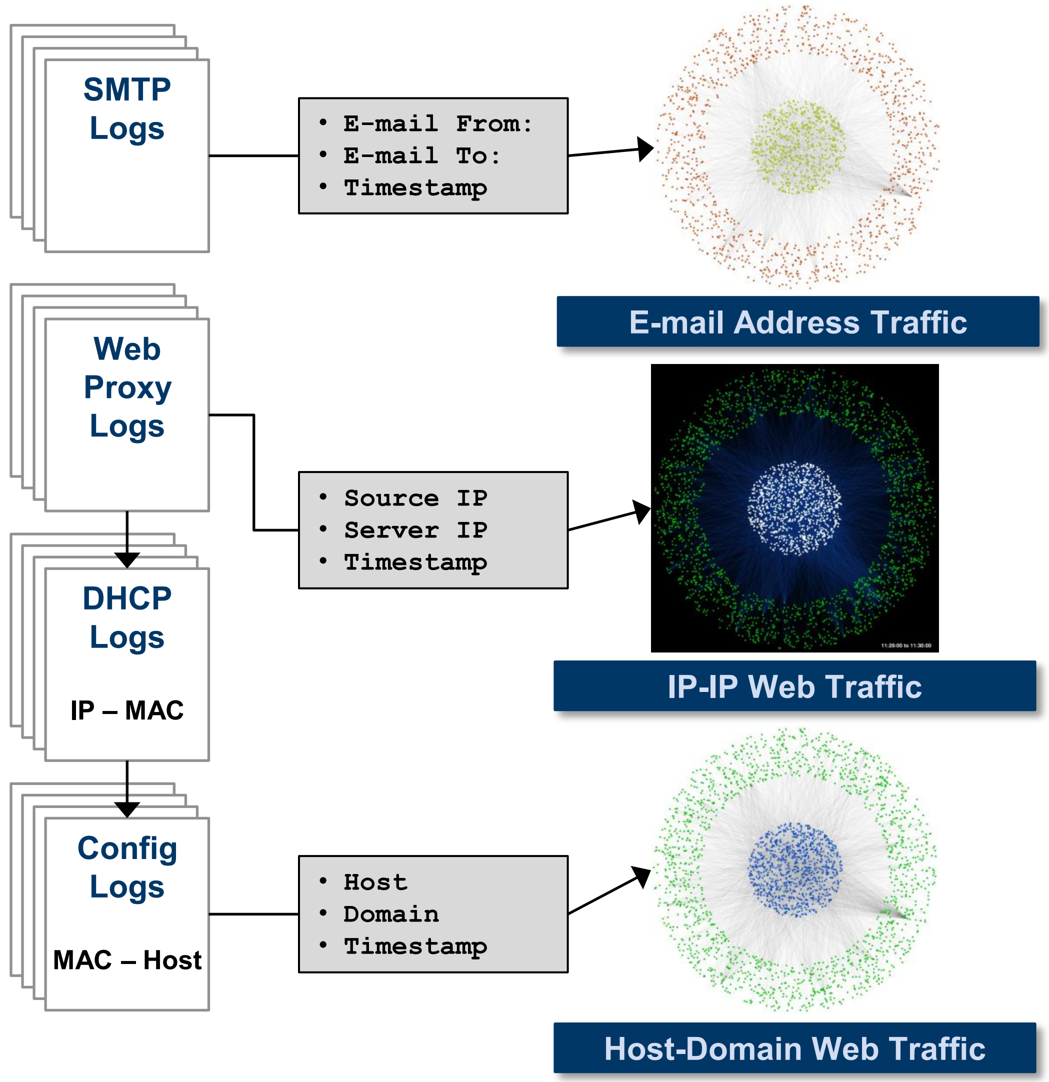
Big Data Challenge: Data Representation

Data Sources

- Raw data sources are rarely stored in a graph format
- Data is often derived from multiple collection points

Graph Construction

- Many different graphs can be built from a single data source
- Constructing a single graph may require many sources
- Building multi-graphs requires that entities be normalized



Challenge: Raw data source representations do not enable the efficient construction of graphs of interest

Technology Stack

Graph Analytics	**Applicability** • Cyber, COIN, ISR, Bioinformatics **Resiliency** • Uncertainty in data and observation
High Level Languages	**Scalability** • Parallel language support **Programmability** • Automated performance optimization
Distributed Storage and Indexing	**Portability** • Bindings to multiple databases **Elasticity** • Virtual machine development
High Performance Processing	**Performance** • Novel instruction set architectures **Efficiency** • Specialized circuitry and communication

Outline

- Introduction
- Course Outline
- Example Implementation
- Summary

The MIT Formula

Theory

- Academics
- Departments
  - EECS, Math, Physics, …
- Mathematics
- Algorithms
- Software

+

Experiment

- Research
- Laboratories
  - Lincoln, CSAIL, Media, …
- Measurement
- Data
- Bytes

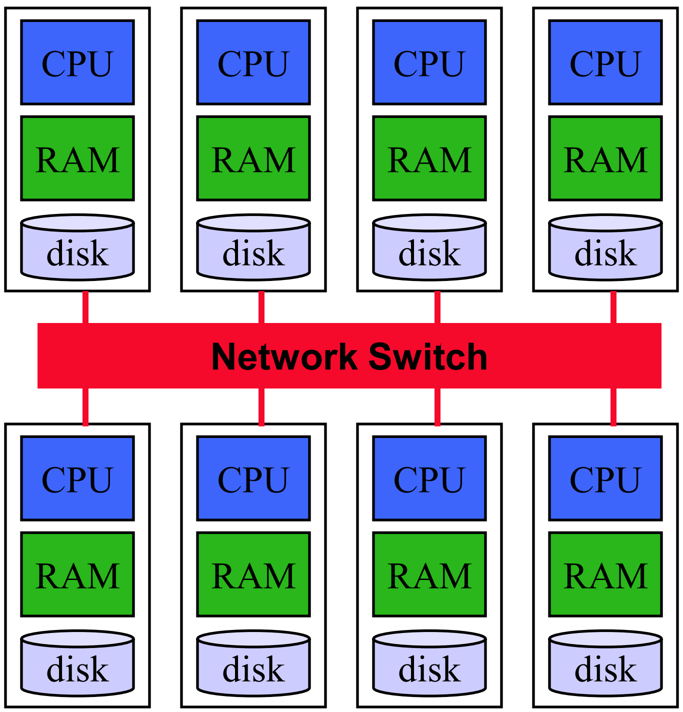
=

Discovery

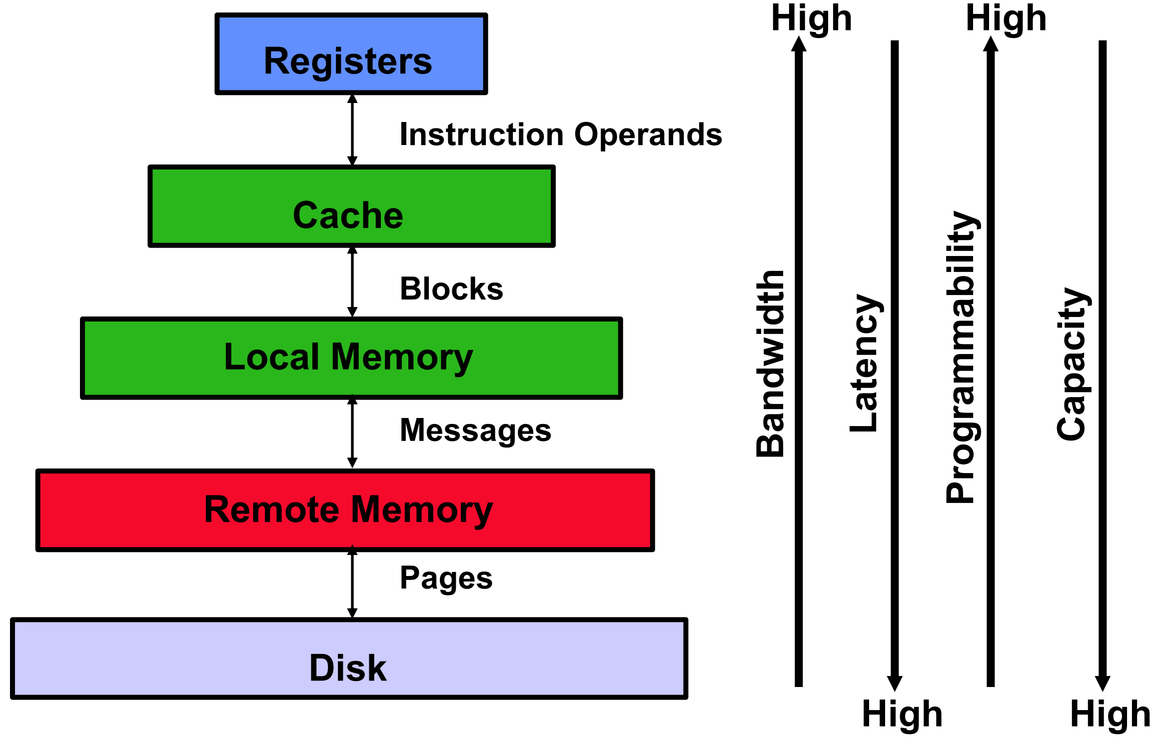
- MIT is the #1 Science and Engineering University on Earth
- A simple formula for success permeates all of MIT
- Implementing this formula often reduces to software and bytes

Software and Bytes Live on Parallel Computers

Parallel Architecture

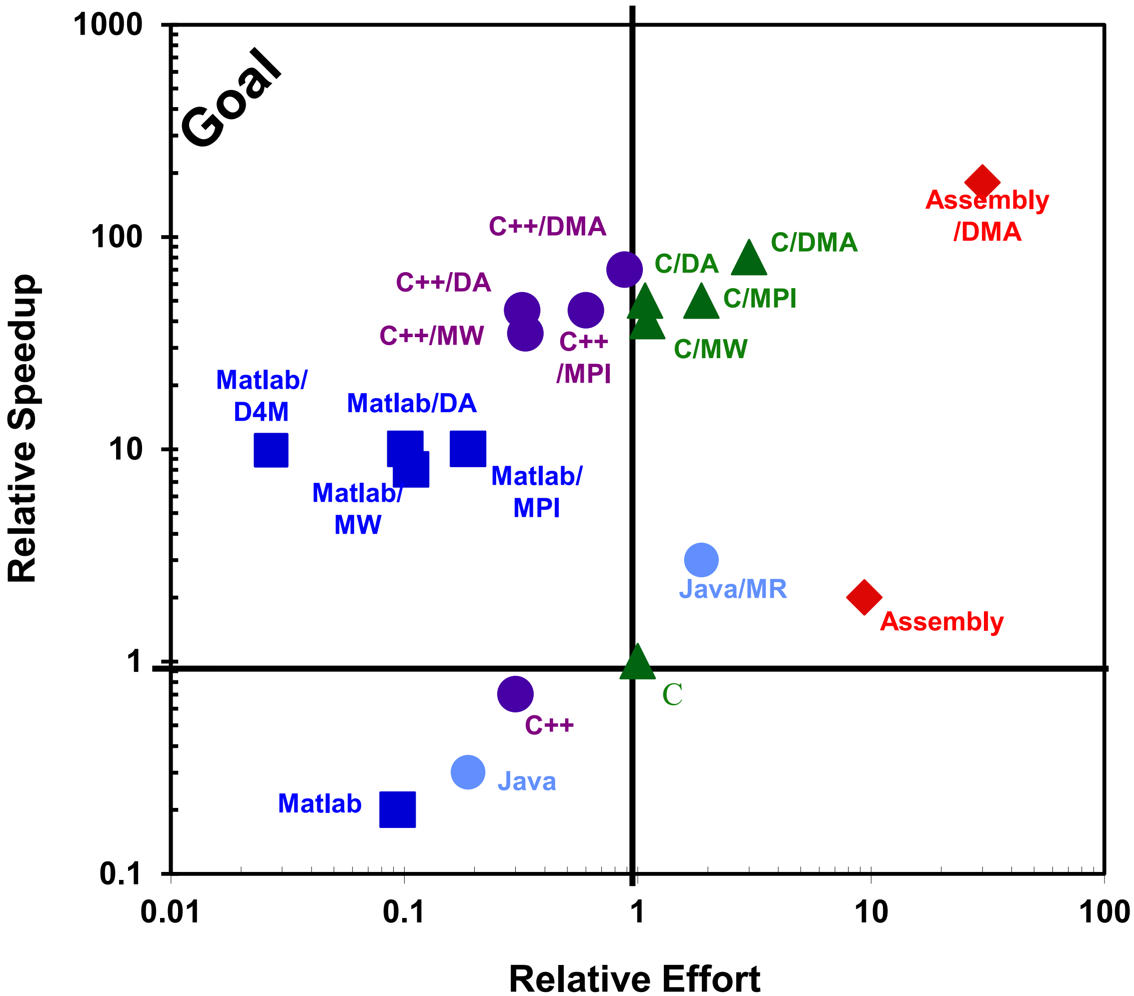


Memory Hierarchy



- Nearly all modern computers are Von Neumann architectures with multi-level memory hierarchies
- The architecture selects the algorithms and data that run well on it

Software Performance vs. Parallel Programmer Effort

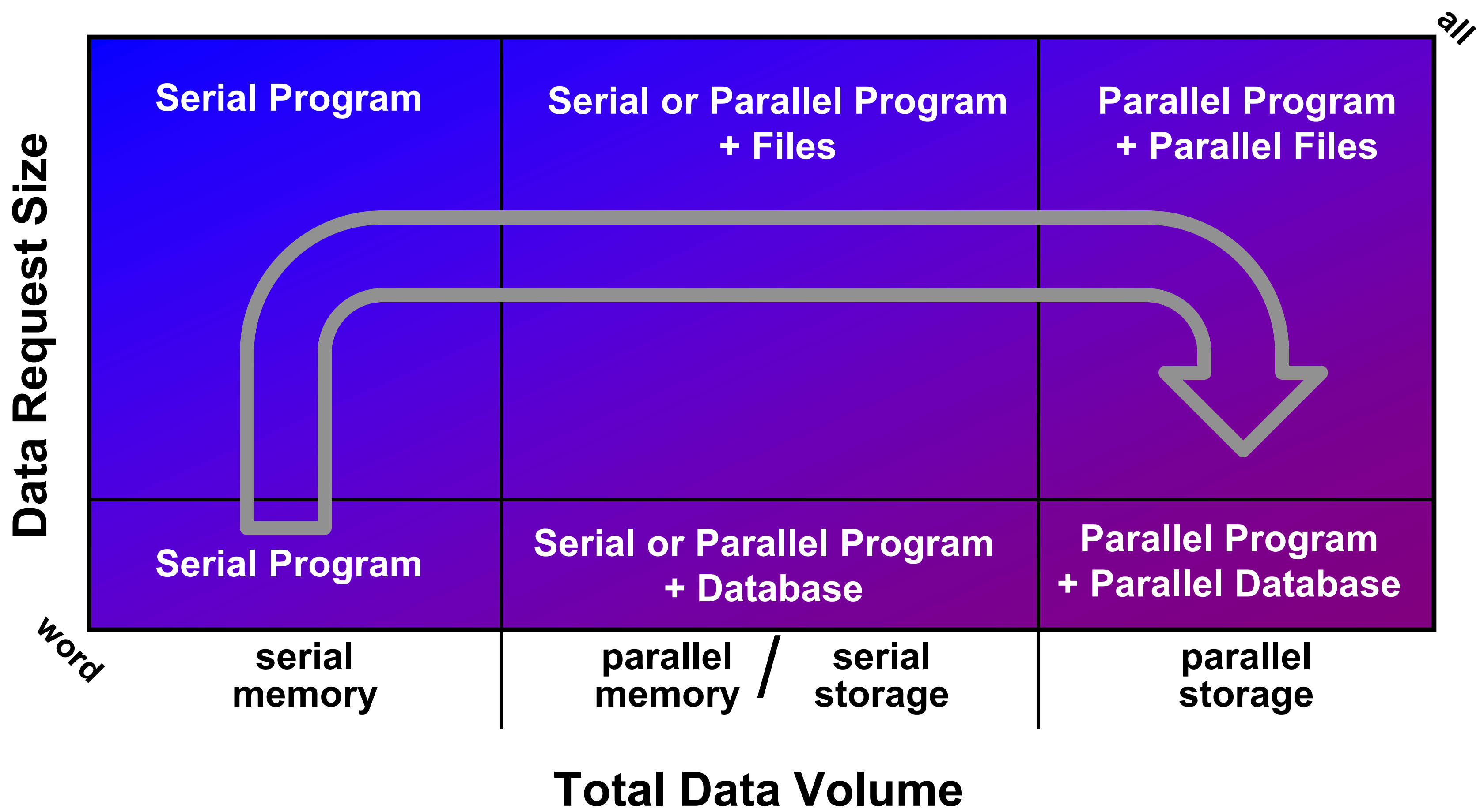


Parallel Programming Models

DMA	= Direct Memory Access
MPI	= Message Passing
DA	= Distributed Arrays
MW	= Manager/Worker
MR	= Map/Reduce
D4M	= Dynamic Distributed Dimensional Data Model

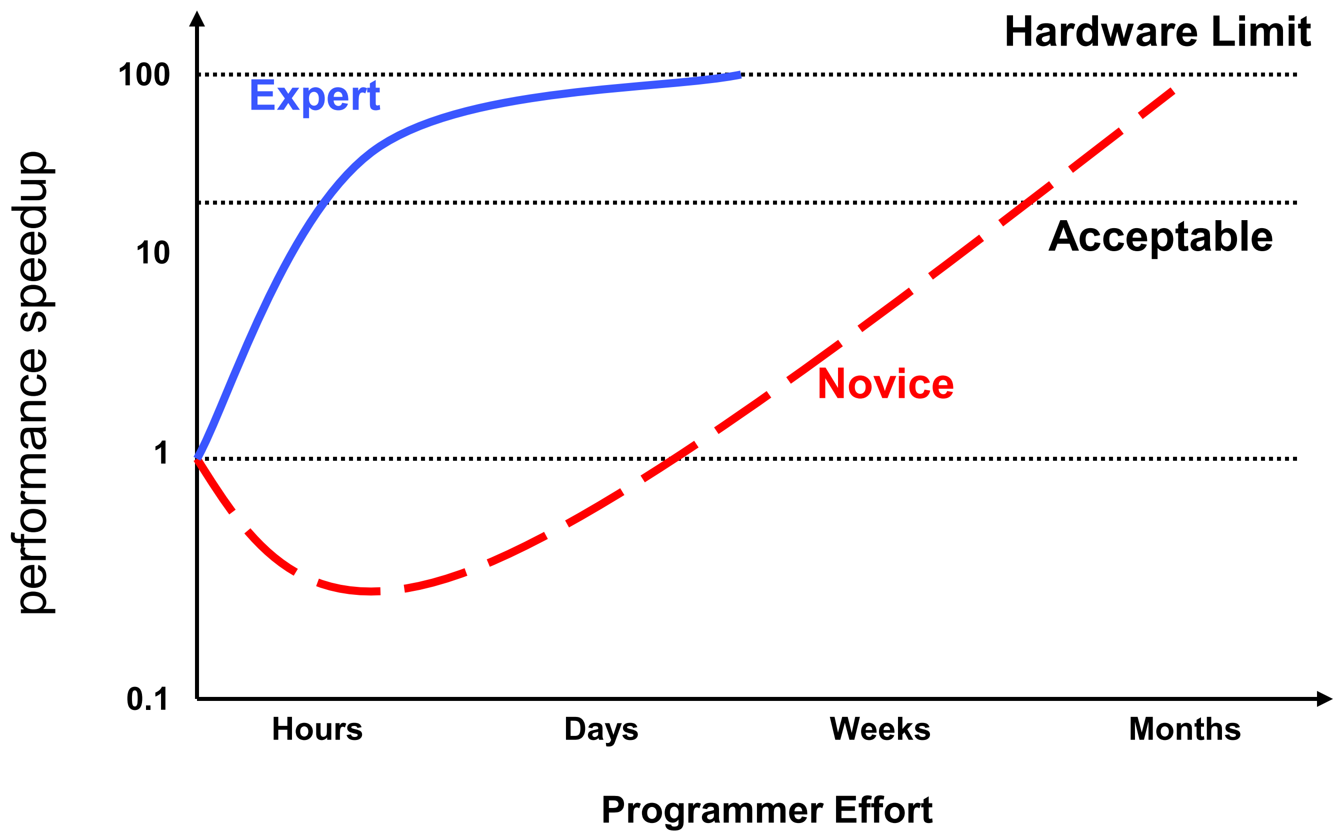
- **Goal: Software that does a lot with the least effort**

Data Use Cases



- Data volume and data request size determine best approach
- Always want to start with the simplest and move to the most complex

The Fast Path



- **The class teaches the highest performance and lowest effort software techniques that are currently known**

Key Course Concepts

- **Bigger definition of a graph**
  - How to move beyond random, undirected, unweighted graphs to power-law, directed, multi-hyper graphs
- **Bigger definition of linear algebra**
  - How to move beyond real numbers to doing math with words and strings
- **Bigger definition of processing**
  - How to move beyond map/reduce to distributed arrays programming

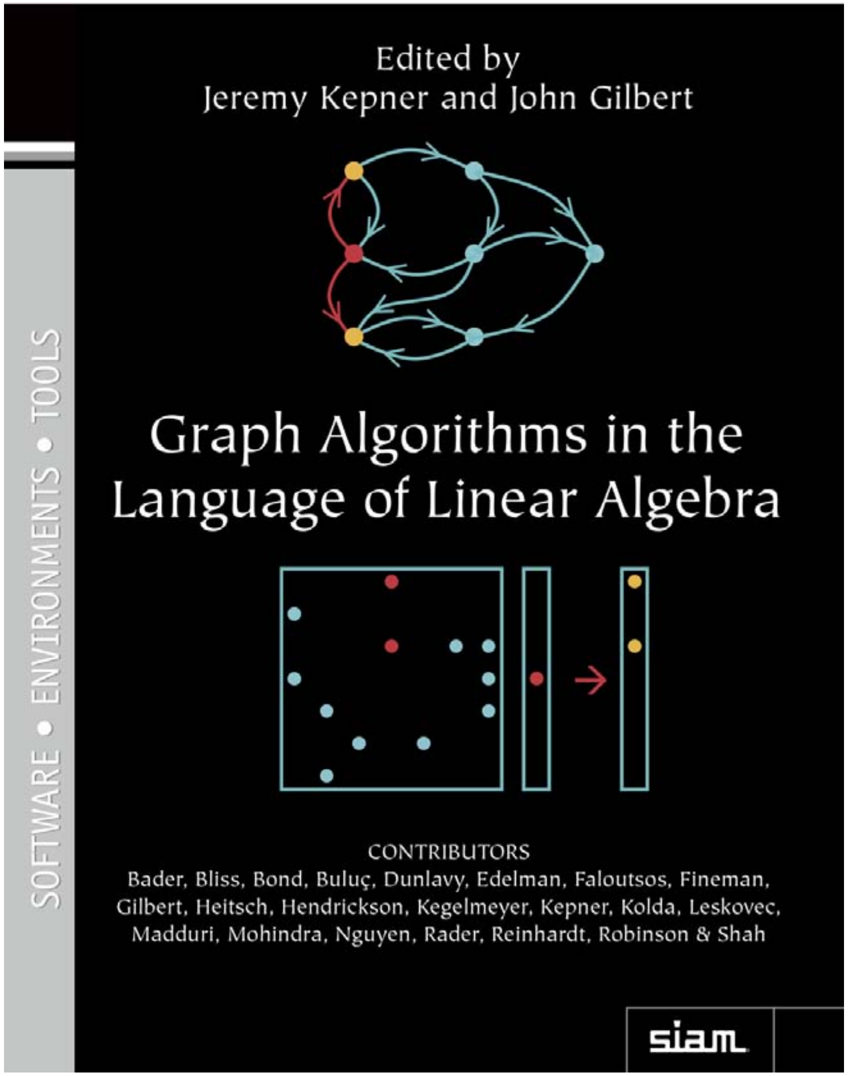
> - These abstract concepts are the foundation for high performance signal processing on large unstructured data sets

Course Outline

- **Introduction**
  - Review course goals and structure
- **Using Associative Arrays**
  - Schemas, incidence matrices, and directed multi-hyper graphs
- **Group Theory**
  - Extending linear algebra to words using fuzzy algebra
- **Entity Analysis in Unstructured Data**
  - Reading and parsing unstructured data
- **Analysis of Structured Data**
  - Graph traversal queries
- **Power Law Data**
  - Models and fitting
- **Cross Correlation**
  - Sequence data, computing degree distributions, and finding matches
- **Parallel Processing**
  - Kronecker graphs, parallel data generation and computation
- **Databases**
  - Relational, triple store, and exploded schemas

References

- **Book: "Graph Algorithms in the Language of Linear Algebra"**
- **Editors: Kepner (MIT-LL) and Gilbert (UCSB)**
- **Contributors:**
  - **Bader (Ga Tech)**
  - **Bliss (MIT-LL)**
  - **Bond (MIT-LL)**
  - **Dunlavy (Sandia)**
  - **Faloutsos (CMU)**
  - **Fineman (CMU)**
  - **Gilbert (USCB)**
  - **Heitsch (Ga Tech)**
  - **Hendrickson (Sandia)**
  - **Kegelmeyer (Sandia)**
  - **Kepner (MIT-LL)**
  - **Kolda (Sandia)**
  - **Leskovec (CMU)**
  - **Madduri (Ga Tech)**
  - **Mohindra (MIT-LL)**
  - **Nguyen (MIT)**
  - **Radar (MIT-LL)**
  - **Reinhardt (Microsoft)**
  - **Robinson (MIT-LL)**
  - **Shah (USCB)**



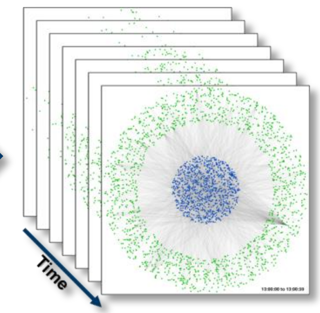
Outline

- Introduction
- Course Outline
- Example Implementation
- Summary

Constructing Graph Representations of Raw Data Source

Raw Data

Vertex and Edge Lists



(1) Parse edge and vertex information from raw data

Developed Once Per Data Source Per Graph

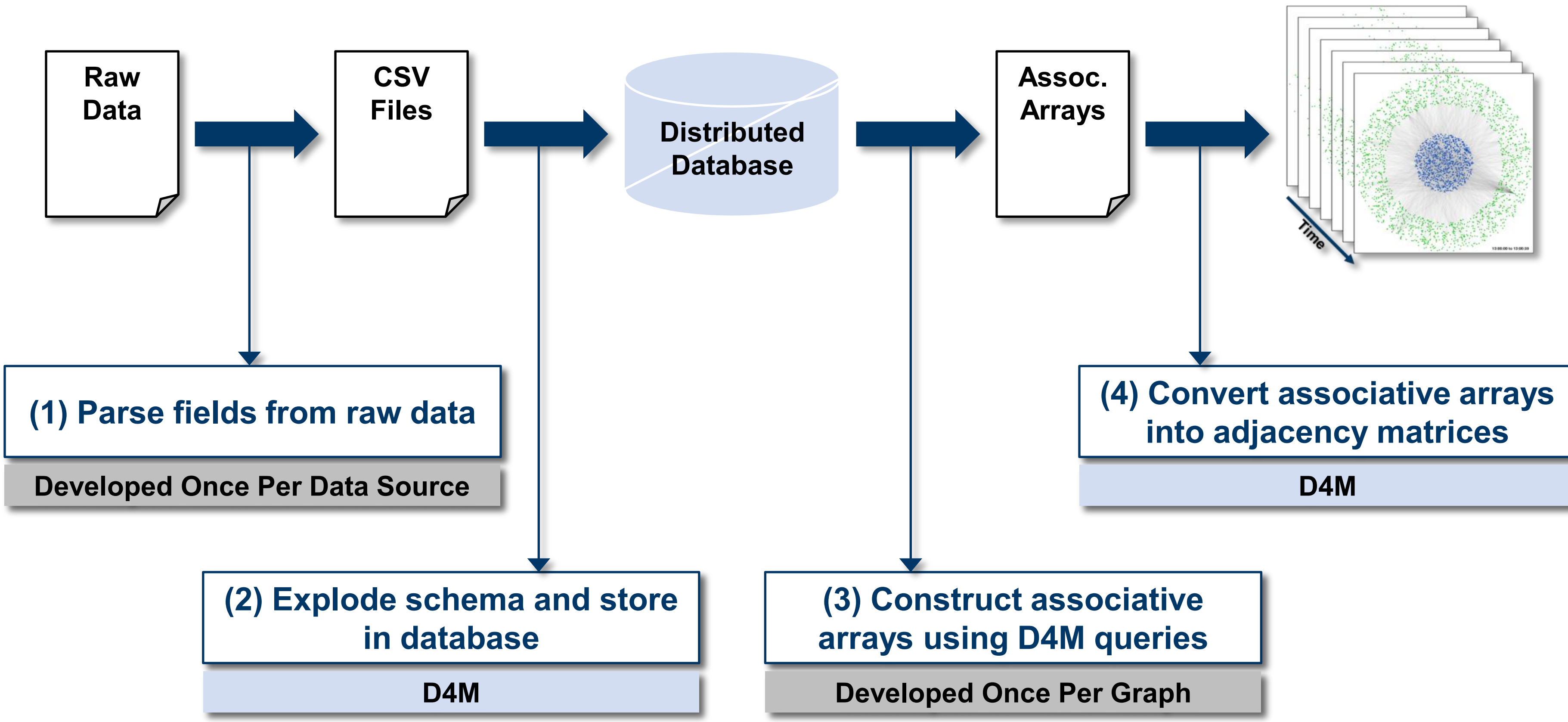
(2) Convert edge lists into adjacency matrices

Developed Once

- Raw data sources can contain information about multiple types of relations between entities
- The process of constructing a graph representation is specific to both the data source and the relationships represented by the graph

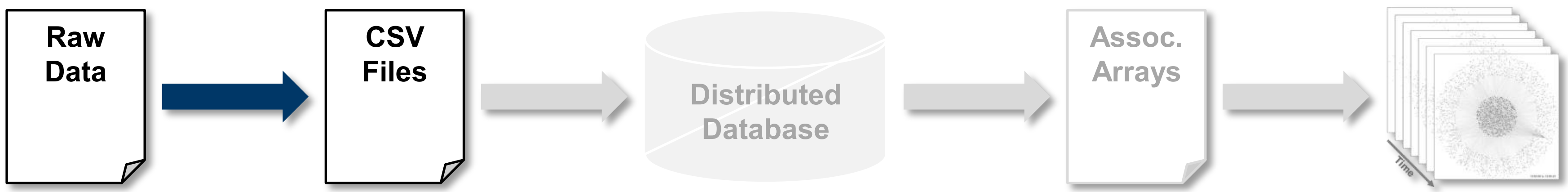
- The development time of parsing and graph construction algorithms can overwhelm the runtime of the algorithm

Graph Construction Using D4M



- **D4M provides needed flexibility in the construction of large-scale, dynamic graphs at different resolutions and scopes**

Graph Construction Using D4M: Parsing Raw Data Into Dense Tables



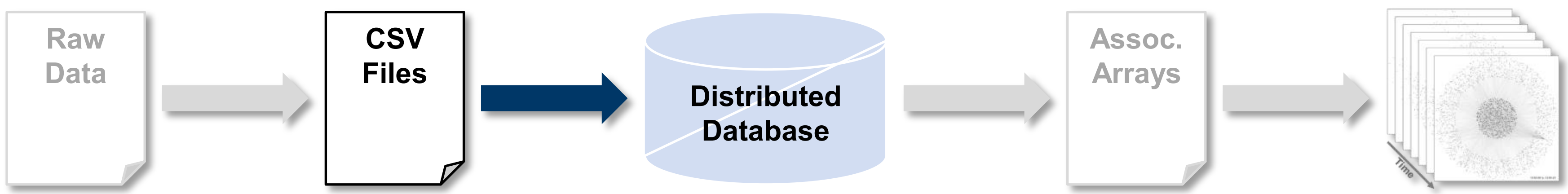
Proxy Logs

```
128.0.0.1 208.29.69.138 "-" [10/May/2011:09:52:53] "GET http://www.thedailybeast.com/ HTTP/1.1" 200
1024 8192 "http://www.theatlantic.com/" "Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.13)
Gecko/20101209 CentOS/3.6-2.el5.centos Firefox/3.6.13" "bl" - "text/html" "MITLAB" 0.523 "-"
Neutral TCP_MISS
192.168.1.1 157.166.255.18 "-" [12/May/2011:13:24:11] "GET http://www.cnn.com/ HTTP/1.1" 335 256
10296 "-" "Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.13) Gecko/20101209 CentOS/3.6-
2.e15.centos Firefox/3.6.13" "bu" - "text/html" "MITLAB" 0.784 "-" Neutral TCP_MISS
...
```

Dense Table

log_id	src_ip	server_ip	time_stamp	req_line	...
001	128.0.0.1	208.29.69.138	10/May/2011:09:52:53	GET http://www.thedailybeast.com/ HTTP/1.1	…
002	192.168.1.2	157.166.255.18	12/May/2011:13:24:11	GET http://www.cnn.com/ HTTP/1.1	…
003	128.0.0.1	74.125.224.72	13/May/2011:11:05:12	GET http://www.google.com/ HTTP/1.1	…
⋮	⋮	⋮	⋮	⋮	

Graph Construction Using D4M: Explode Schema



Dense Table

log_id	src_ip	server_ip
001	128.0.0.1	208.29.69.138
002	192.168.1.2	157.166.255.18
003	128.0.0.1	74.125.224.72

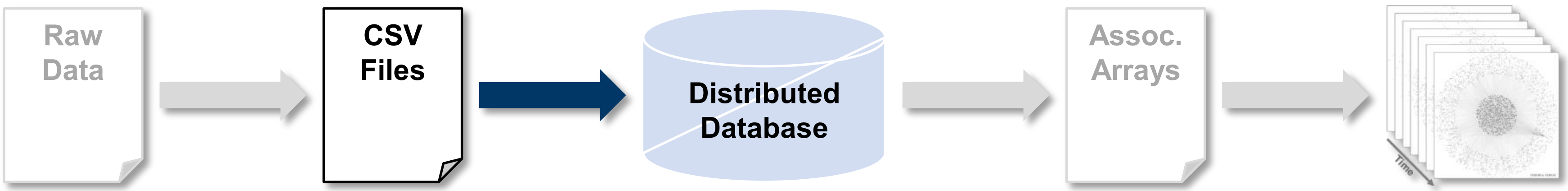
Use as row indices

Create columns for each unique type/value pair

	src_ip|128.0.0.1	src_ip|192.168.1.2	server_ip|157.166.255.18	server_ip|208.29.69.138	server_ip|74.125.224.72
log_id|001	1	0	0	1	0
log_id|002	0	1	1	0	0
log_id|003	1	0	0	0	1

Exploded Table

Graph Construction Using D4M: Storing Exploded Data as Triples



Exploded Table

	src_ip|128.0.0.1	src_ip|192.168.1.2	server_ip|157.166.255.18	server_ip|208.29.69.138	server_ip|74.125.224.72
log_id|001	1	0	0	1	0
log_id|002	0	1	1	0	0
log_id|003	1	0	0	0	1

D4M stores the triple data representing both the exploded table and its transpose

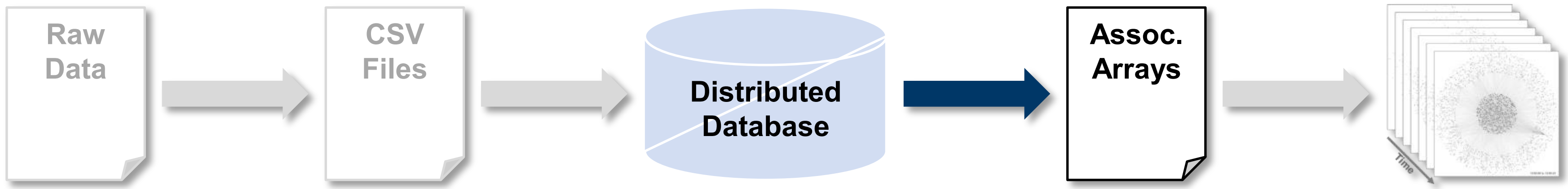
Table Triples

Row	Column	Value
log_id|001	src_ip|128.0.0.1	1
log_id|001	server_ip|208.29.69.138	1
log_id|002	src_ip|192.168.1.2	1
log_id|002	server_ip|157.166.255.18	1
log_id|003	src_ip|128.0.0.1	1
log_id|003	server_ip|74.125.224.72	1

Table Transpose Triples

Row	Column	Value
server_ip|157.166.255.18	log_id|002	1
server_ip|208.29.69.138	log_id|001	1
server_ip|74.125.224.72	log_id|003	1
src_ip|128.0.0.1	log_id|001	1
src_ip|128.0.0.1	log_id|003	1
src_ip|192.168.1.2	log_id|002	1

Graph Construction Using D4M: Construct Associative Arrays

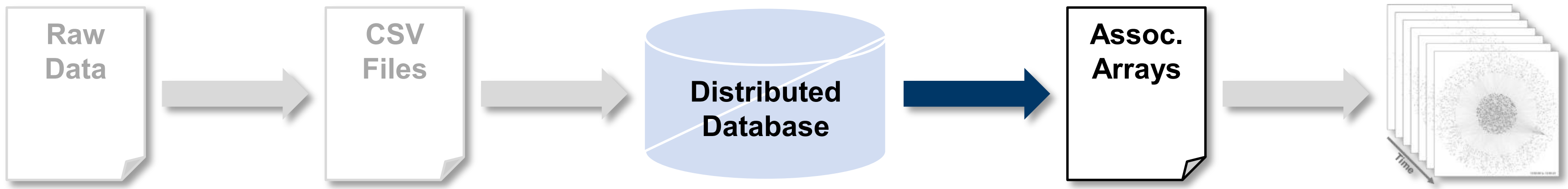


D4M Query #1

```
keys = T(:,'time_stamp|10/May/2011:00:00:00',:, ...
            'time_stamp|13/May/2011:23:59:59',);
```

```
('log_id|001','time_stamp|11/May/2011:09:52:53',1)
('log_id|002','time_stamp|12/May/2011:13:24:11',1)
('log_id|003','time_stamp|13/May/2011:11:05:12',1)
  ...
```

Graph Construction Using D4M: Construct Associative Arrays



D4M Query #1

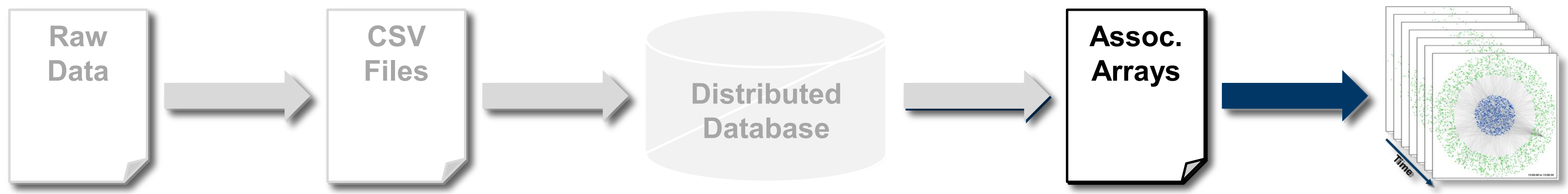
```
keys = T(:,'time_stamp|10/May/2011:00:00:00',:, ...
         'time_stamp|13/May/2011:23:59:59',);
```

D4M Query #2

```
data = T(Row(keys), :);
```

```
('log_id|001','server_ip|208.29.69.138',1)
('log_id|001','src_ip|128.0.0.1',1)
('log_id|001','time_stamp|11/May/2011:09:52:53',1)
  ...
('log_id|002','server_ip|157.166.255.18',1)
('log_id|002','src_ip|192.168.1.2',1)
('log_id|002','time_stamp|12/May/2011:13:24:11',1)
  ...
('log_id|003','server_ip|74.125.224.72',1)
('log_id|003','src_ip|128.0.0.1',1)
('log_id|003','time_stamp|13/May/2011:11:05:12',1)
  ...
```

Graph Construction Using D4M: Construct Associative Arrays



D4M Query #1

```
keys = T(:,'time_stamp|10/May/2011:00:00:00',:, ...
           'time_stamp|13/May/2011:23:59:59',);
```

D4M Query #2

```
data = T(Row(keys), :);
```

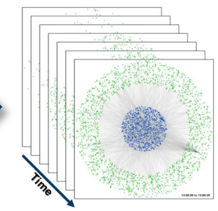
Associative Array Algebra

```
G = data(:,'src_ip|*').' * data(:,'server_ip|*');
```

```
('src_ip|128.0.0.1','server_ip|208.29.69.138',1)
('src_ip|128.0.0.1','server_ip|74.125.224.72',1)
('src_ip|192.168.1.2','server_ip|157.166.255.18',1)
  ...
```

Graph Construction Using D4M: Construct Associative Arrays

Raw Data → CSV Files → Distributed Database → Assoc. Arrays →



D4M Query #1

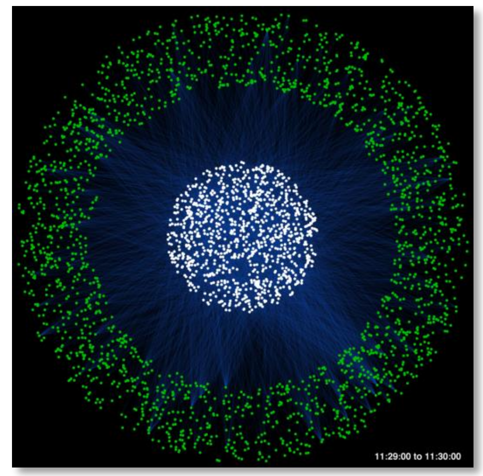
```
keys = T(:,'time_stamp|10/May/2011:00:00:00',:, ...
           'time_stamp|13/May/2011:23:59:59',);
```

D4M Query #2

```
data = T(Row(keys), :);
```

Associative Array Algebra

```
G = data(:,'src_ip|*').' * data(:,'server_ip|*');
```

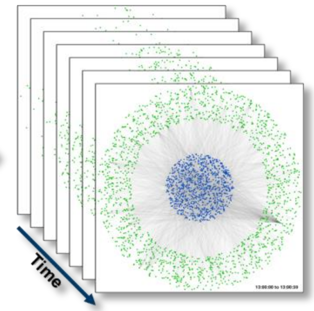



```
Adj(G);
```

- **Graphs can be constructed with minimal effort using D4M queries and associative array algebra**

Constructing Graph Representation of One Week's Worth of Proxy Data

Raw Data → CSV Files → Distributed Database → Assoc. Arrays →



Parsing and ingestion done in parallel using 128 processors

Completed in less than 2 hours

Graph data generated in parallel using 7 processors (one per day)

Completed in 3 hours

- Ingested ~130 million proxy log records resulting in ~4.5 billion triples
- Constructed 604,800 secondwise source IP to server IP graphs
- Constructing graphs with different vertex types could be done without re-parsing or re-ingesting data

- Utilizing D4M could allow analysis to be run in nearly real-time (dependent on raw data availability)

Summary

- Big data is found across a wide range of areas
  - Document analysis
  - Computer network analysis
  - DNA Sequencing
- Currently there is a gap in big data analysis tools for algorithm developers
- D4M fills this gap by providing algorithm developers composable associative arrays that admit linear algebraic manipulation

Example Code and Assignment

- **Example code**
  - D4Muser_share/Examples/1Intro/1AssocIntro

- **Assignment**
  - Test your LLGrid account and D4M
  - Copy the D4Muser_share/Examples to your LL Grid home directory
  - Verify that you can run the above examples
    - Start Matlab
    - CD to your copy of the example
    - Run the Examples