Scalable Data Structures for Machine Learning

Carlos Guestrin
University of Washington
Dato, Inc.

http://www.istc-cc.cmu.edu/

Intel Science & Technology
Center for Cloud Computing

The GraphLab journey

GraphLab Carnegie Mellon

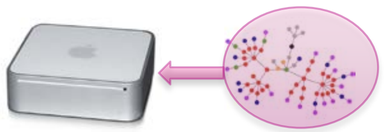
Low et al. [UAI 2010]

- Distributed system for graph computation
- Challenges scaling to huge graphs

GraphLab

Gonzalez et al. [OSDI 2012]

- New graph computation model
- Highly scalable to huge natural graphs



GraphChi

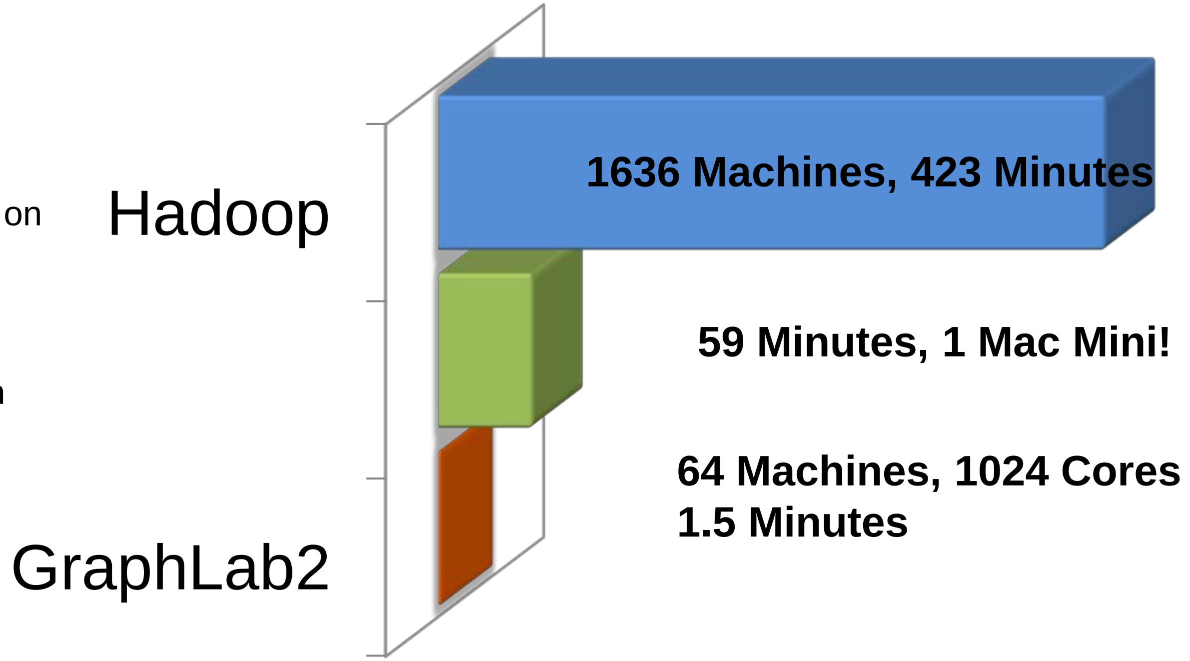
Kyrola et al. [OSDI 2012]

- Designed for best single machine scaling
- Optimized out-of-core computation

Triangle Counting on Twitter Graph

40M Users
1.2B Edges

Total: 34.8 billion triangles



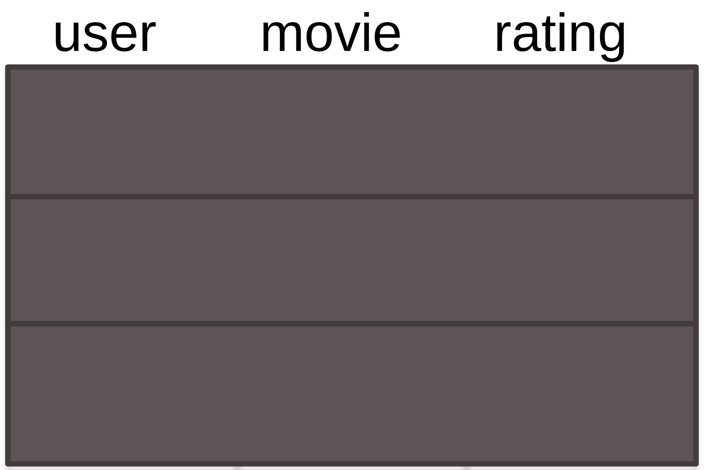
Many systems

- Assume unbounded resources
- **Optimize for scale**

- Limited scalability on single machine / small cluster
- Really painful to build intelligent applications…

SFrame: Scalable data frame for ML

Data frames



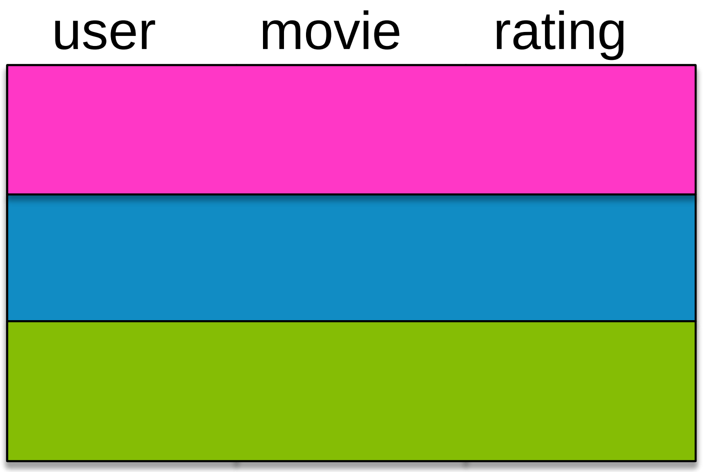
When you choose a
data frame,
have your application in mind

SFrame is
optimized for ML

ML has specific
data access patterns,
we make them fast, really fast

(Columnar transformations,
creating new features, iterations,…)

Data is usually rows…



But, data engineering typically column transformations…

Feature engineering is columnar

Normalizes the feature x:

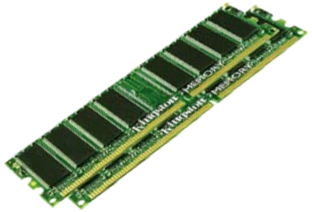
```
sf['rating'] = sf['rating'] / sf['rating'].sum()
```

Sequential operations happen over one or a few columns, not rows of data (certain algorithms, e.g., SGD, operate or rows, won't cover today, but can be addressed in framework)

Opportunity for Out-of-Core ML

Opportunity for big data on 1 machine

Fast, but significantly limits data size



Capacity	10 TB	1 TB	0.1 TB
Throughput	0.1 GB/s	0.5 GB/s	1 GB/s

For sequential reads only!
Random access very slow

GraphChi early example
SFrame data frame for ML

Out-of-core ML opportunity is huge

Usual design → Lots of random access → Slow

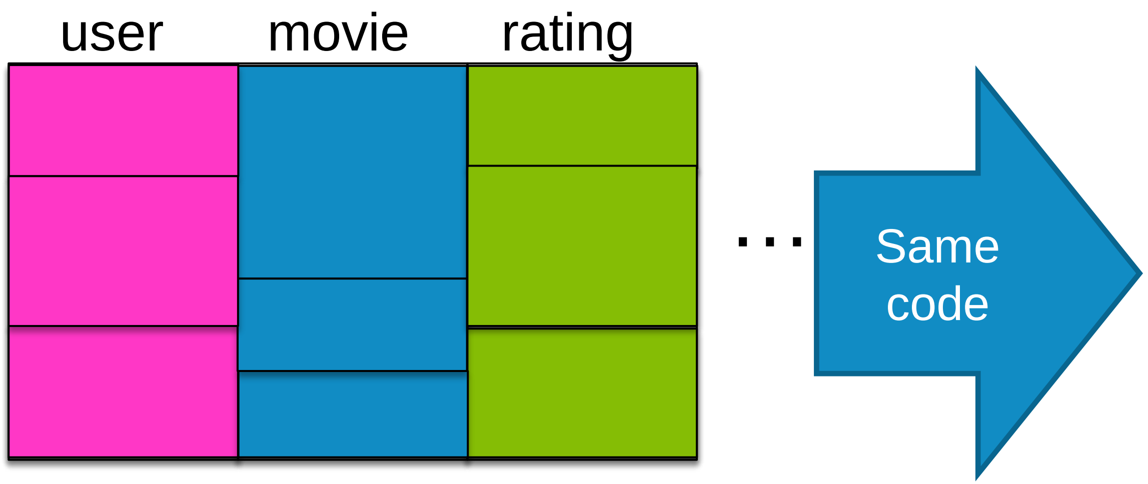
Design to maximize sequential access for ML algo patterns

SFrame: Scalable data frame optimized for ML

Never run out of memory

Sharded, compressed, out-of-core, columnar

Arbitrary lambda transformations, joins,… from Python

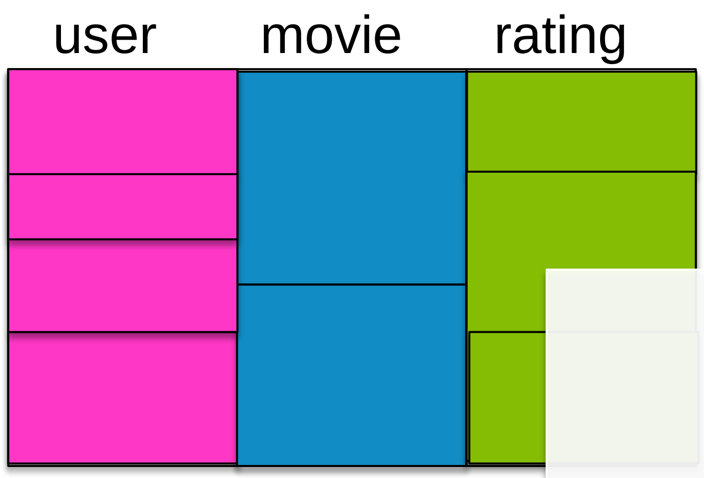


Large data on one machine?

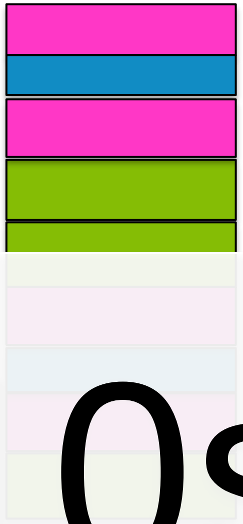
Limited RAM ➔ Must use disk

(out-of-core computation)

SFrame columnar encoding



SFrame File



Type aware compression:

- Variable Bit length Encode
- Frame Of Reference Encode
- ZigZag Encode
- Delta / Delta ZigZag Encode
- Dictionary Encode
- General Purpose LZ4

10s

Netflix Dataset,
99M rows, 3 columns, ints
1.4GB raw
289MB gzip compressed

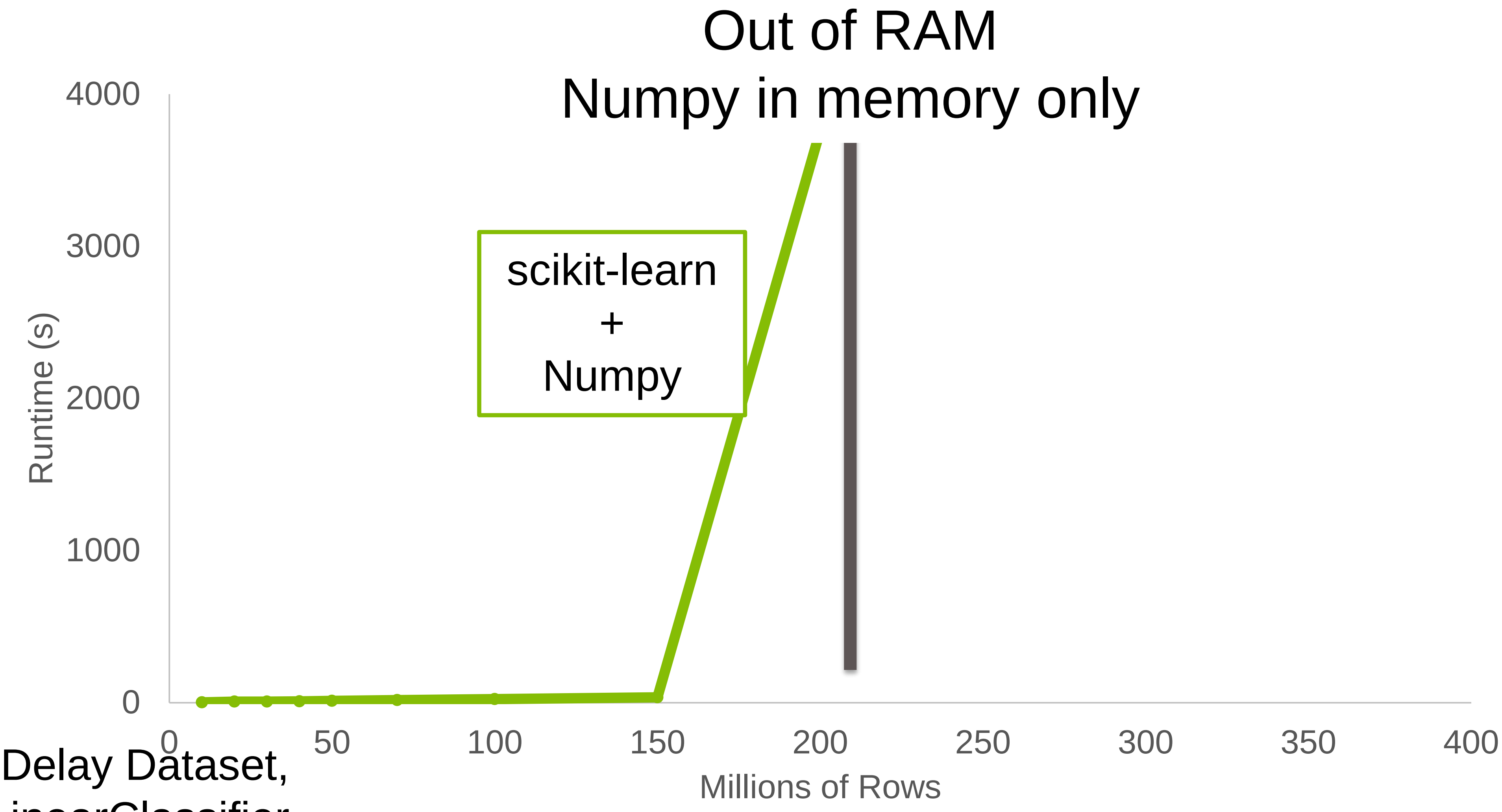
User	→	176 MB	14.2 bits/int
Movie	→	**257 KB**	0.02 bits/int
Rating	→	47 MB	3.8 bits/int

Total → **223MB**

Demo: 10TBs of data on one machine!

SFrame ❤︎ all ML

scikit-learn is awesome, but...



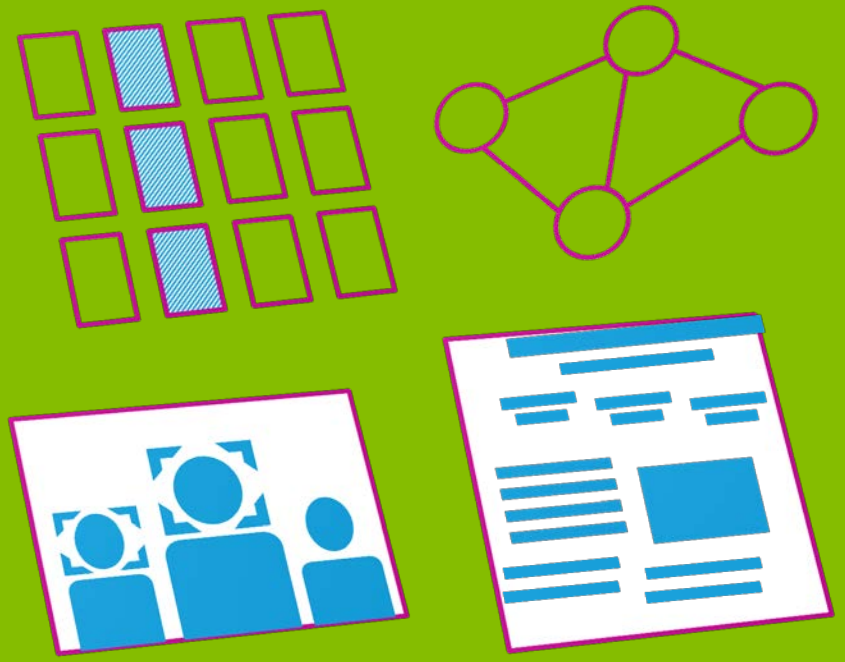
Airline Delay Dataset,
SGDLinearClassifier

Demo: 10TBs of data on one machine *redux*

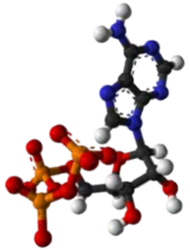
Numpy Automatically Backed by Sframes → Scale many Python packages (scikit-learn, scipy,...)

```
import graphlab.numpy
Scalable numpy activation successful
```

ML is not just about tables



Social Media **Science** **Advertising** **Web**



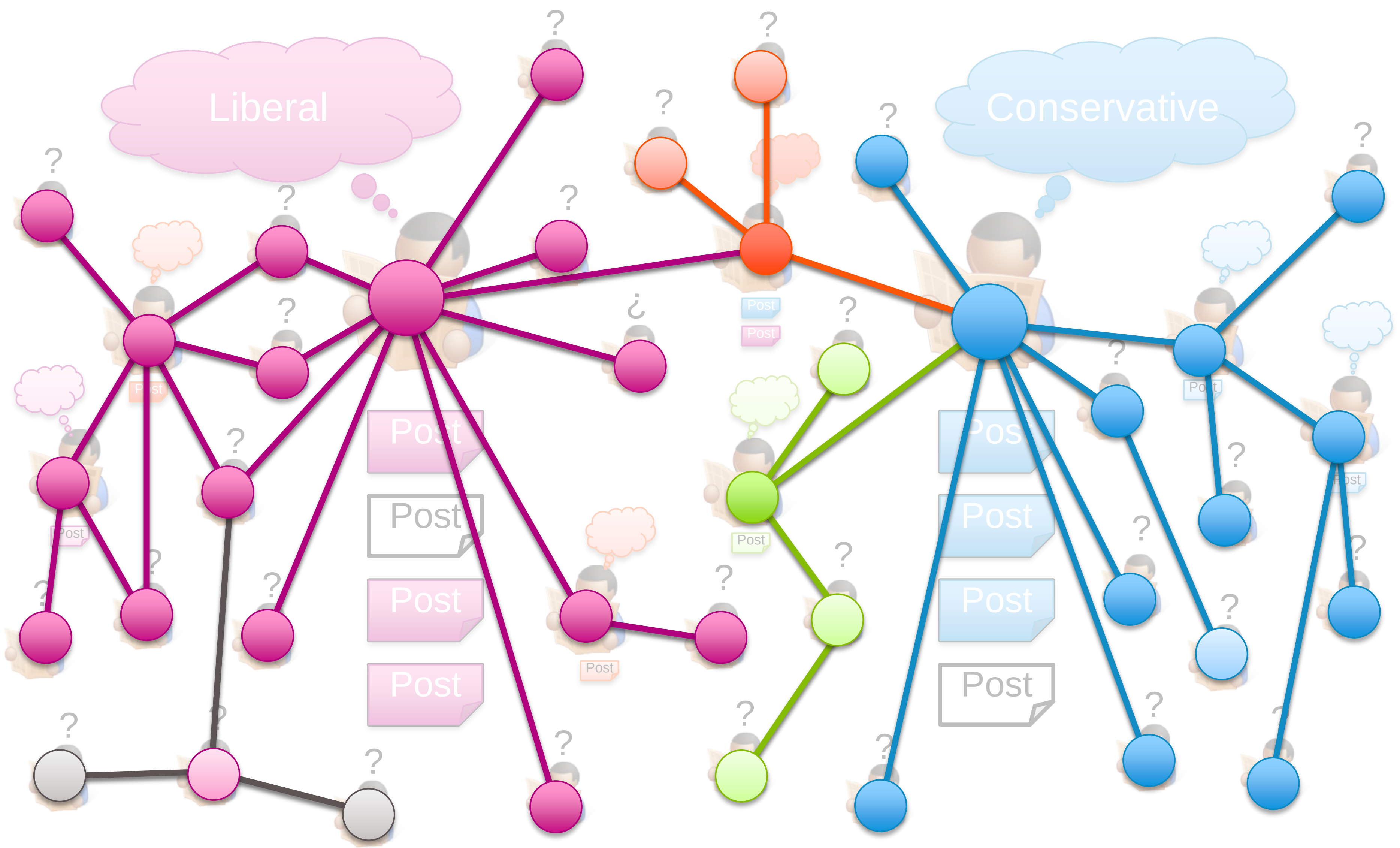
- **Graphs** encode the **relationships** between:

People Products Ideas
Facts Interests

- **Big: trillions** of **vertices** and **edges** and rich metadata
  - Facebook (10/2012): 1B users, 144B friendships
  - Twitter (2011): 15B follower edges

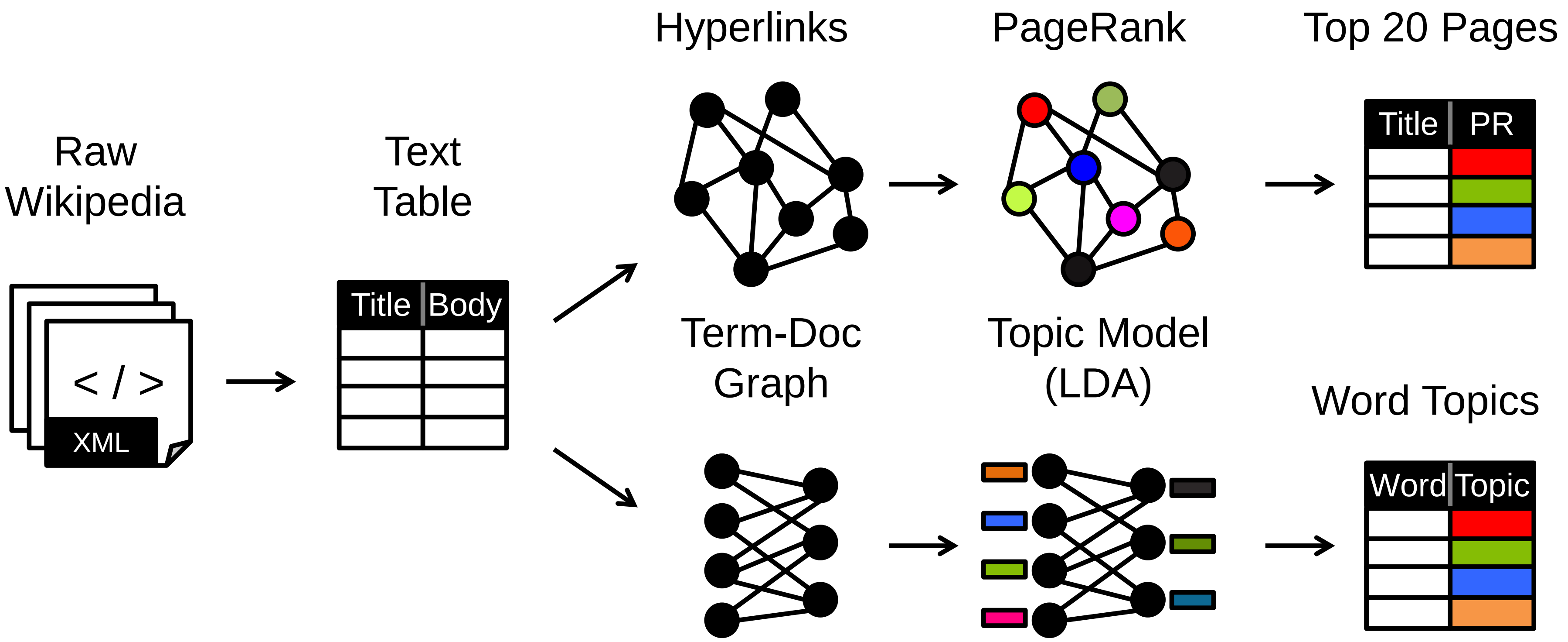
For example…

Example: Estimating political bias

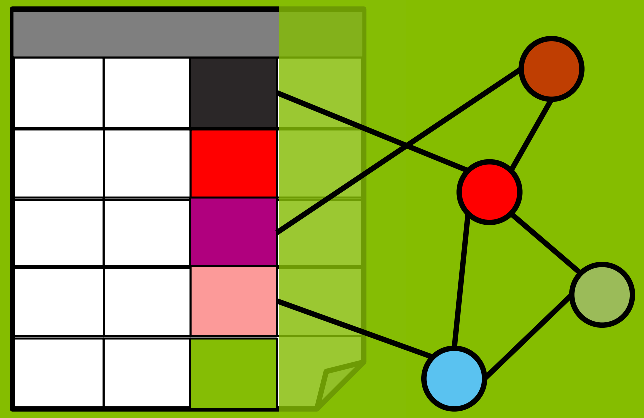


But, ML is about all data types…

ML pipelines combine multiple data types



Integrating tables and graphs



SGraph

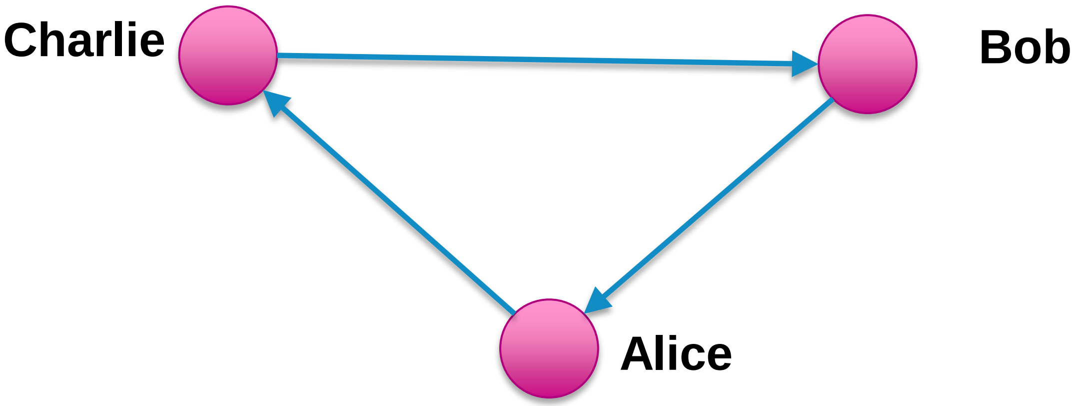
Graph processing & analytics

Neighborhoods, paths, graph algos, community detection, label propagation, ML on graphs, viz, …

Backed by SFrame

Out-of-core & scalable

Basic graph representation



Vertex Table

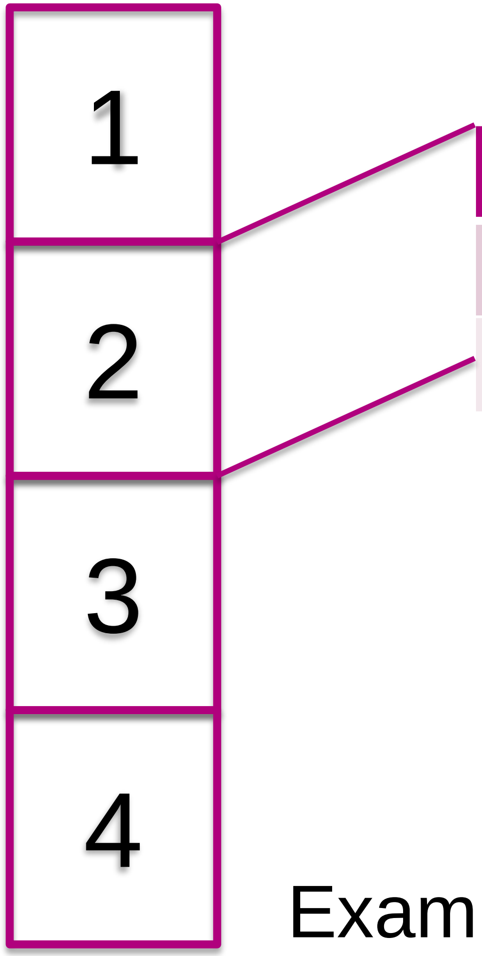
__id	Address	ZipCode
Alice	…	98105
Bob	…	98102
Charlie	…	98103

Edge Table

__src_id	__dst_id	Message
Alice	Bob	“hello”
Bob	Charlie	“world”
Charlie	Alice	“moof”

SGraph vertex data layout

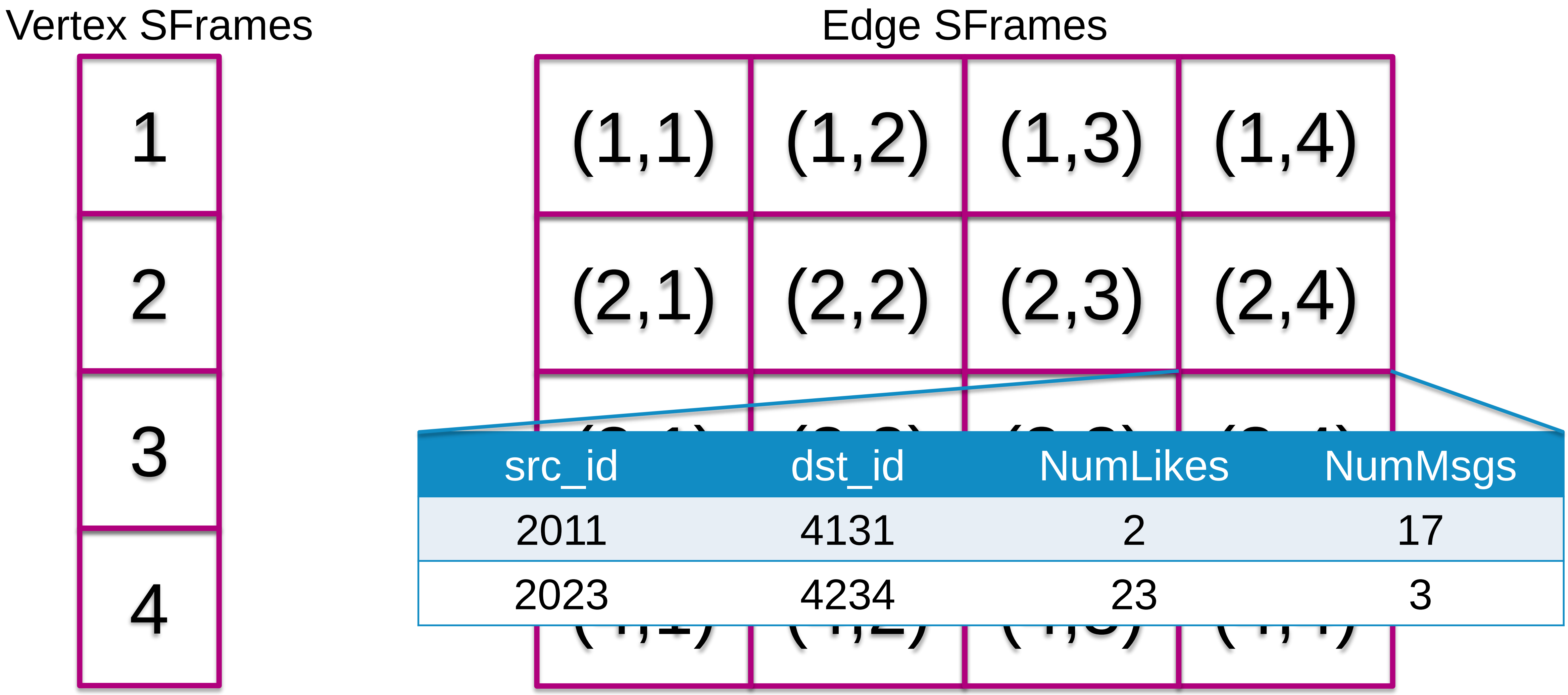
Vertex SFrames



__id	Name	Address	ZipCode
1011	John	…	98105
2131	Jack	…	98102

Example: vertices partitioned into p = 4 SFrames

SGraph edge data layout



src_id	dst_id	NumLikes	NumMsgs
2011	4131	2	17
2023	4234	23	3

Edges partitioned into p^2 = 16 SFrames

Deep integration of SFrames and SGraphs

- Seamless interaction between graph data and table data

SGraph vertex data can be viewed as tables

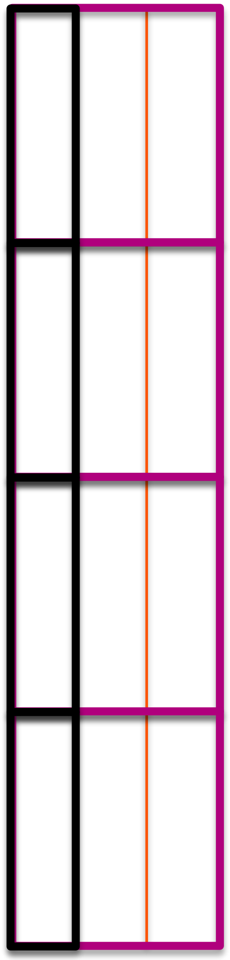
```
g = SGraph(…)
g.vertices[‘large_pagerank’] = g.vertices[‘pagerank’]>100
```

SGraph edge data can be viewed as tables

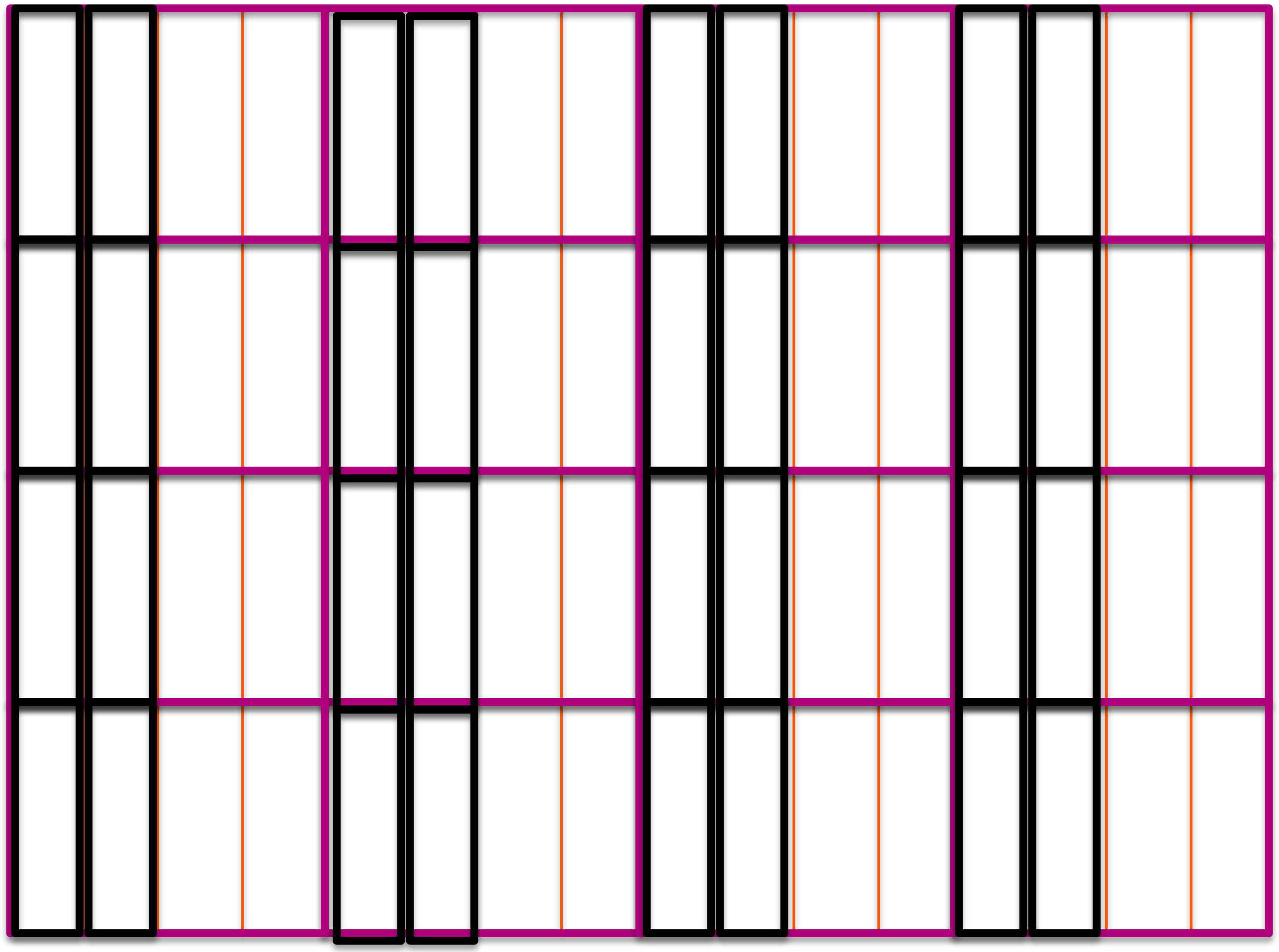
```
g = SGraph(…)
g.edges[‘normalized_ratings’] = g.edges[‘ratings’]/g.edges[‘ratings’].mean()
```

SGraph columnar → Selection is easy

Vertex SFrames



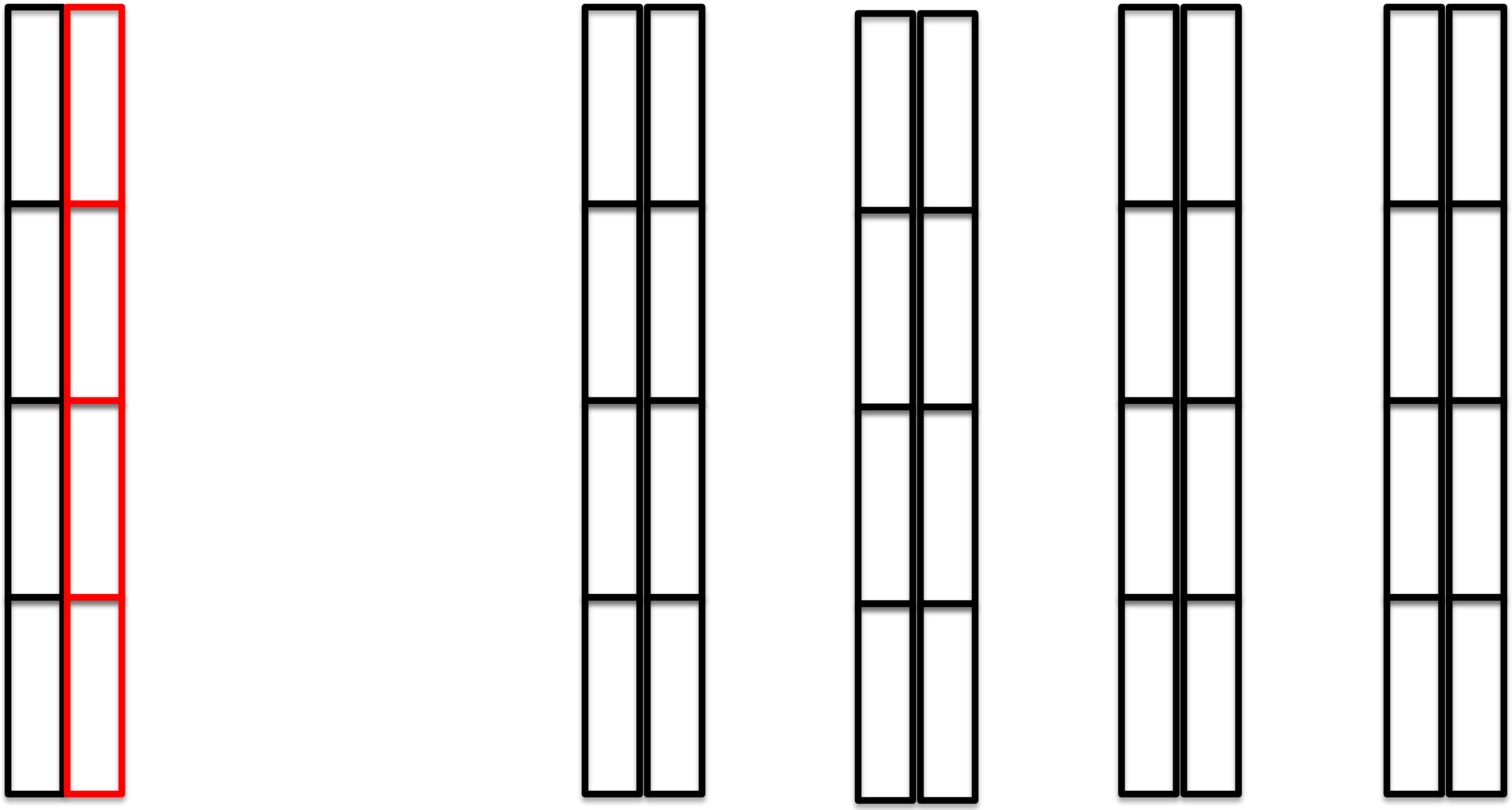
Edge SFrames



SGraph columnar ➔ Adding features is easy

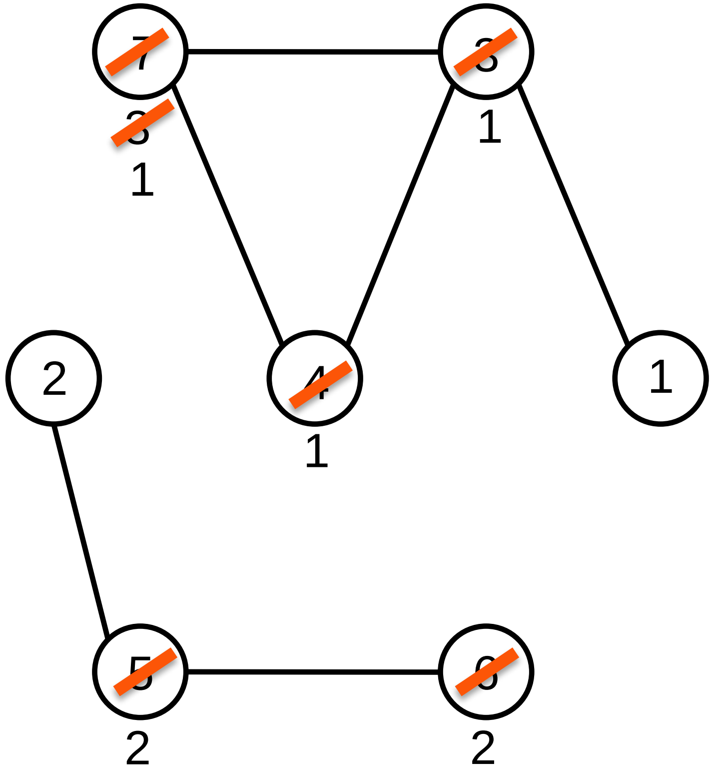
Vertex SFrames

Edge SFrames



Performing computations on graphs

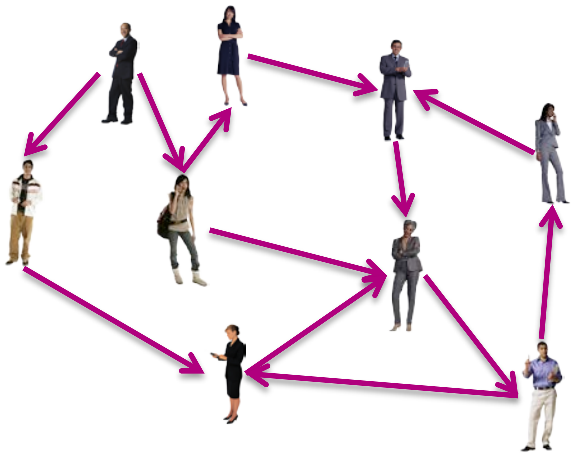
Distributed connected components algorithm



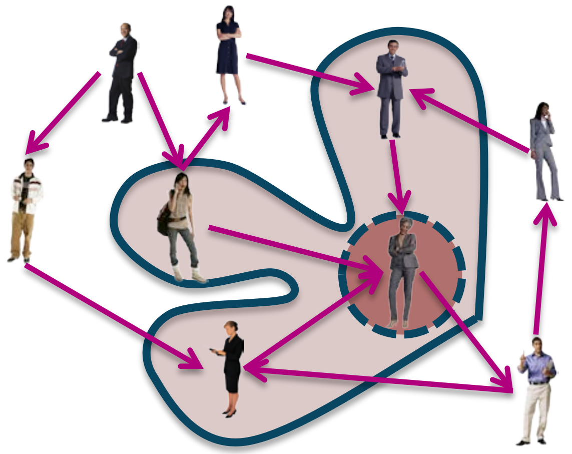
- Initialize: Assign vertex id as component
- Iterate:
  - My id is the minimum of my neighborhood

Properties of graph-parallel algorithms

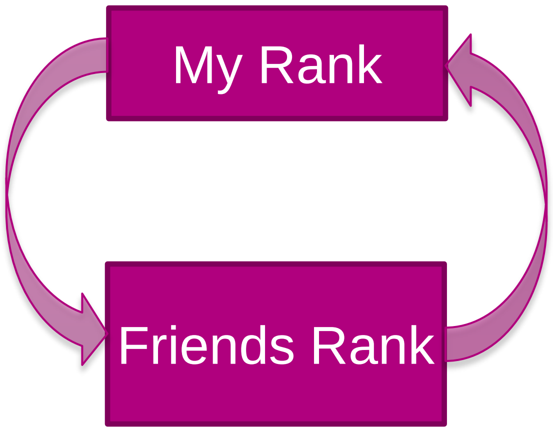
Dependency Graph



Local Updates



Iterative Computation



Graphical Models
Gibbs Sampling
Belief Propagation
Variational Inf.

Collaborative Filtering
Item-item similarity
Tensor Factorization

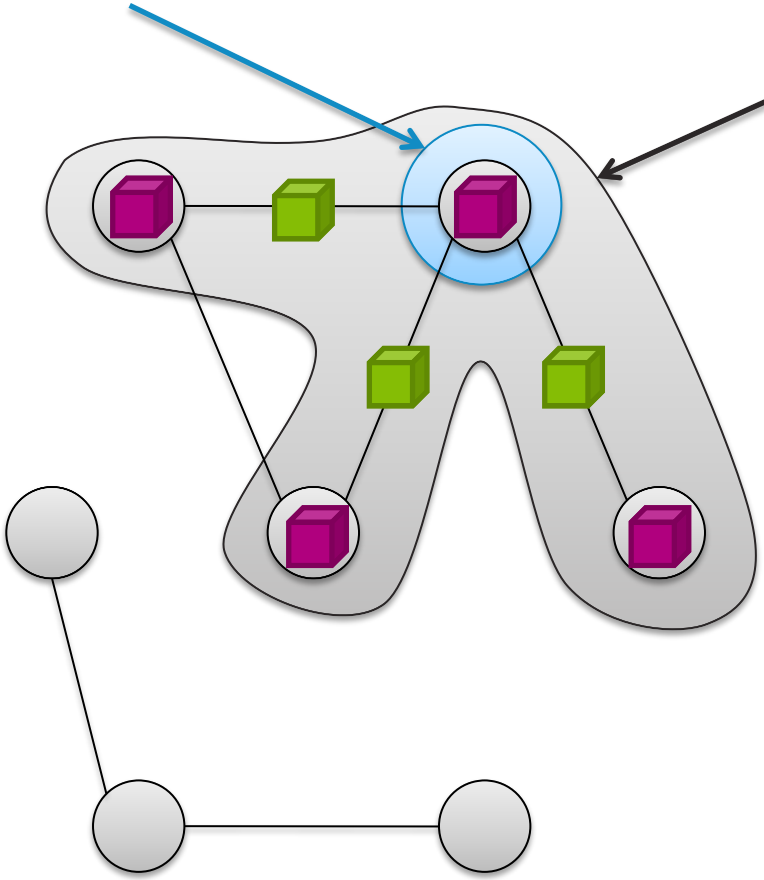
Semi-Supervised Learning
Label Propagation
CoEM

Data-Mining
PageRank
Triangle Counting

Graph-parallel programming abstractions

Vertex programs [Low et al. '10]

User-defined program: applied to
vertex transforms data in scope of vertex



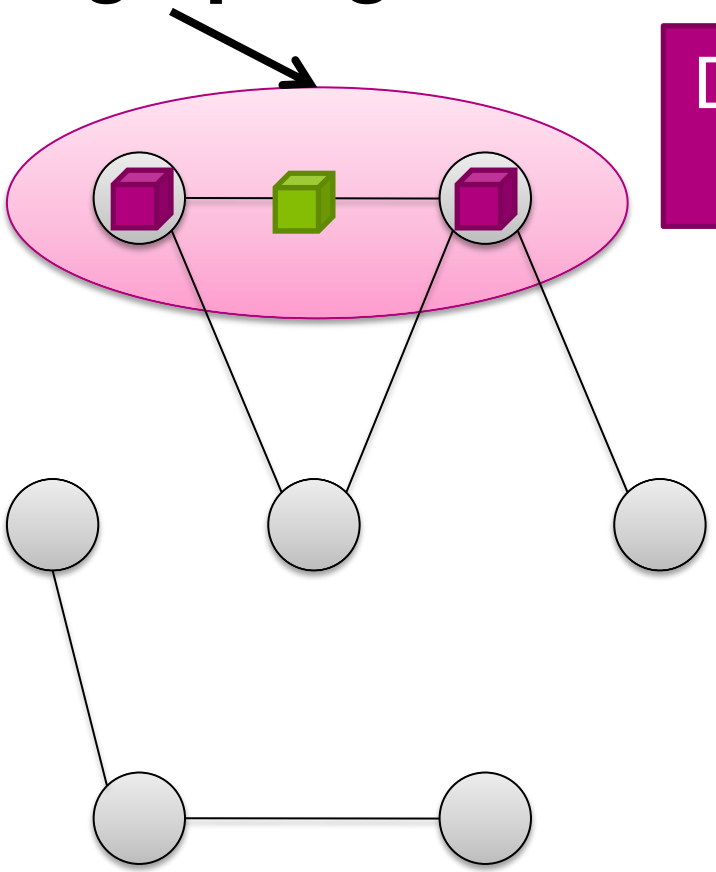
Vertex programs simple, but exhibit significant performance challenges in out-of-core & distributed settings

```
connected_components(vertex, neighbors){

  // Compute minimum component
  min_component = min(vertex['component'],
                                          components of
neighbors)

  // Update vertex component
  vertex['component'] = min_component

}
```

Triple_apply:

simple, highly-parallelizable graph processing abstraction

Edge programs not vertex programs!



Distributed implementation is much simpler & more efficient than vertex programs

```
connected_comp_triple_apply(src, dst, edge){

  // Compute minimum component
  min_component = min(src['component'],

       dst['component'])
  // Update both vertices
 src['component'] = min_component
 dst['component'] = min_component

}
```

Optimizing triple_apply execution

Triple_apply needs vertex data for src and dst vertices

Vertex SFrames

1
2
3
4

Edge SFrames

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

Naïve traversal over edges

Vertex SFrames

1
2
3
4

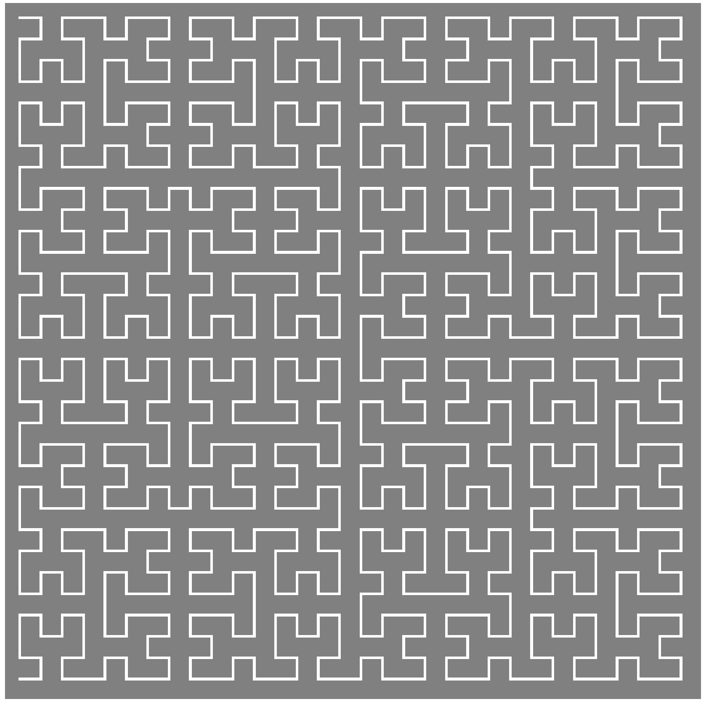
Edge SFrames

(1,1)	(1,2)	(1,3)	(1,4)
(4,1)	(4,2)	(4,3)	(4,4)

Significant IO cost
No cache locality

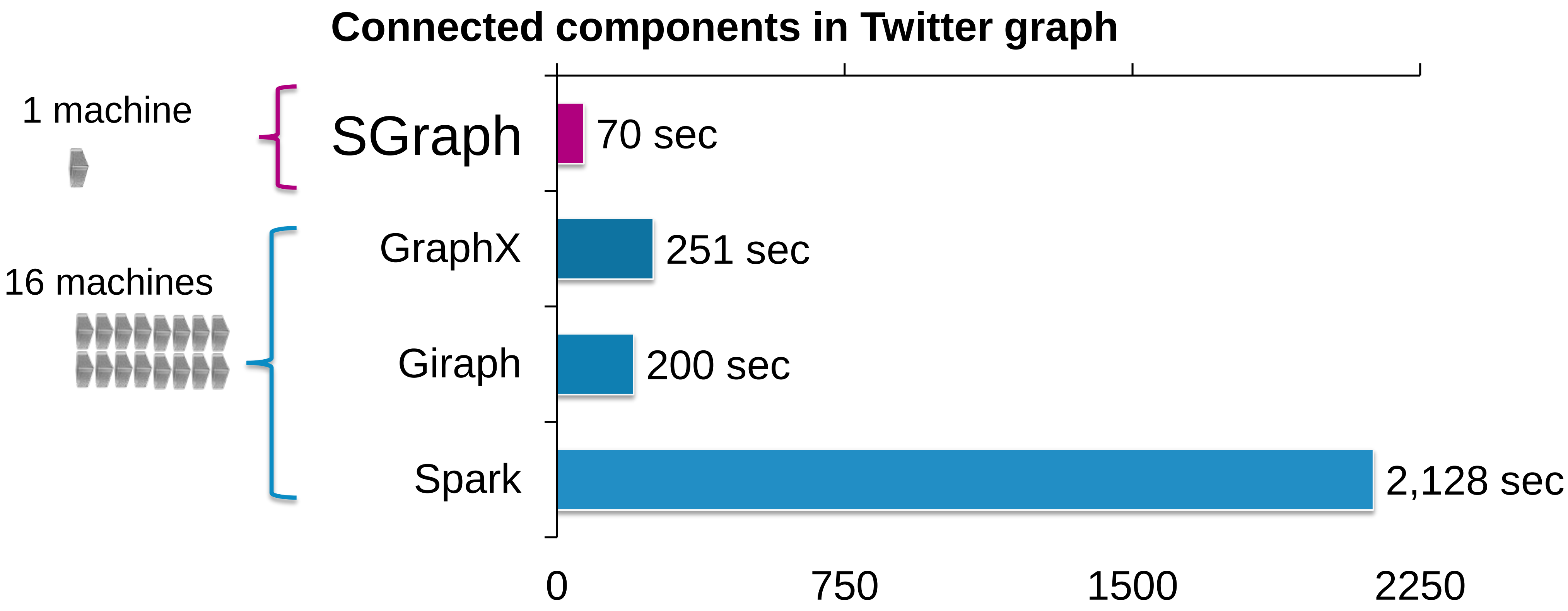
Need walk ordering minimizing loading-unloading

- Efficient option: Hilbert space-filling curves
  - Minimum loads of vertex data
  - Preserves locality ➔ great cache behavior



SGraph: performance

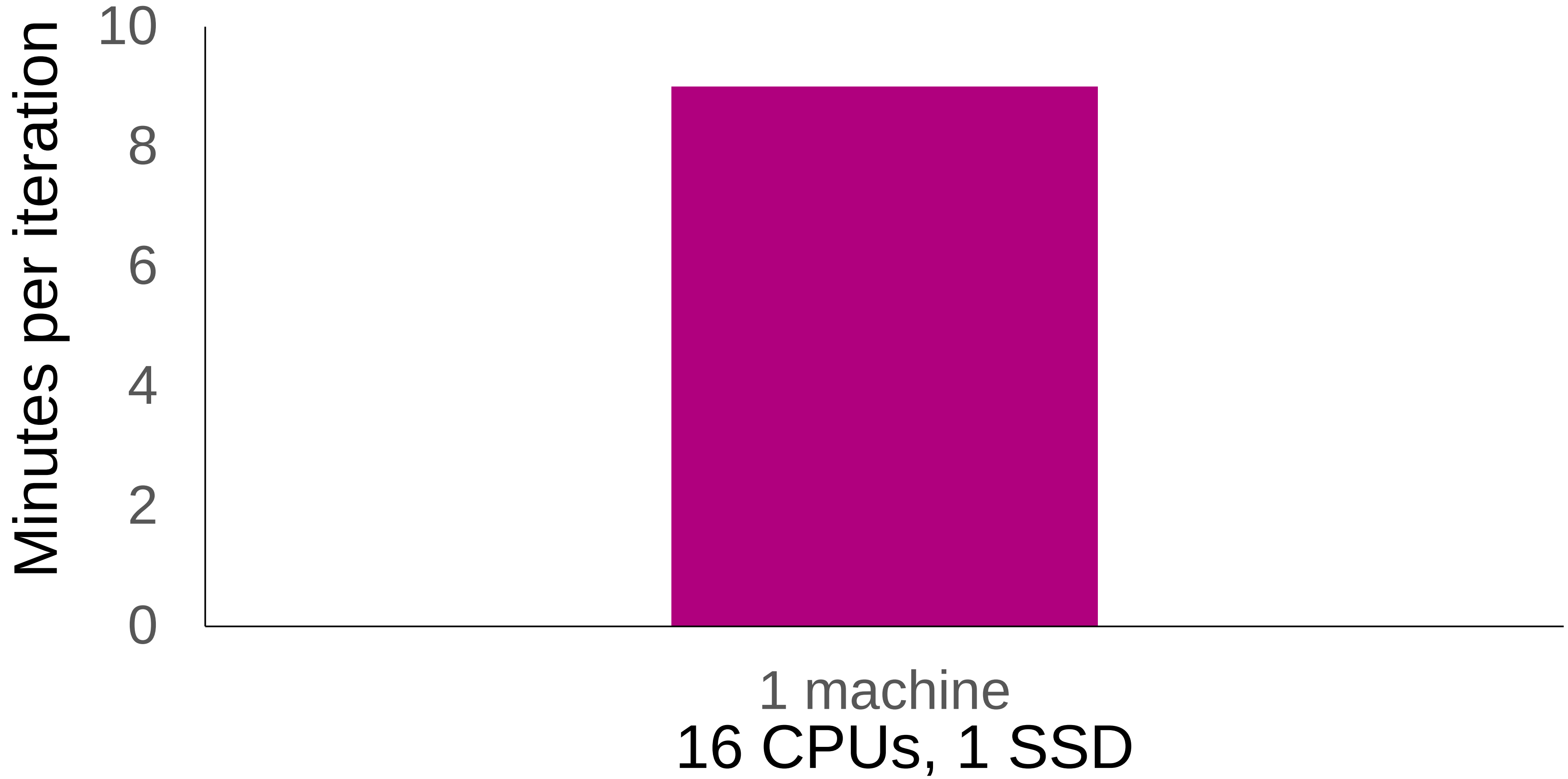
Performance of SGraph



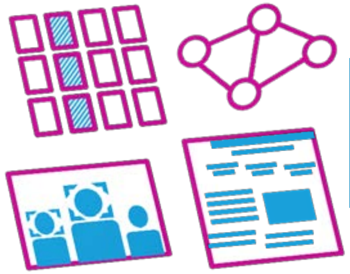
Source(s): [Gonzalez et. al. (OSDI 2014)]
Twitter: 41 million Nodes, 1.4 billion Edges

Pagerank on Common Crawl Graph

3.5 billion Nodes and 128 billion Edges



SFrame/SGraph Summary



Tables, graphs, text, images

SFrame & SGraph

Open-source ❤

BSD license

More than 10,000 downloads

Optimized out-of-core computation for ML

High Performance

1 machine can handle:
TBs of data
100s Billions of edges

Optimized for ML

. Columnar transformation
. Create features
. Iterators
. Filter, join, group-by, aggregate
. User-defined functions
. Easily extended through SDK

Dato

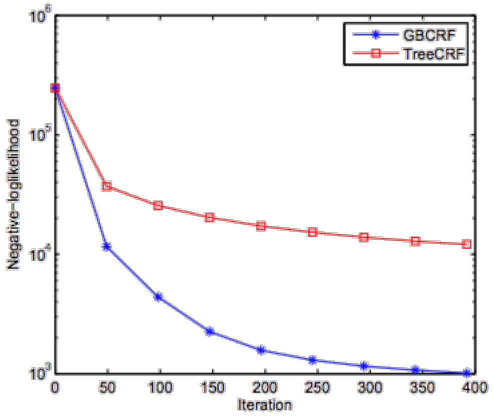
Create Intelligence™

A tech-transfer update…

(Not ISTC IP)

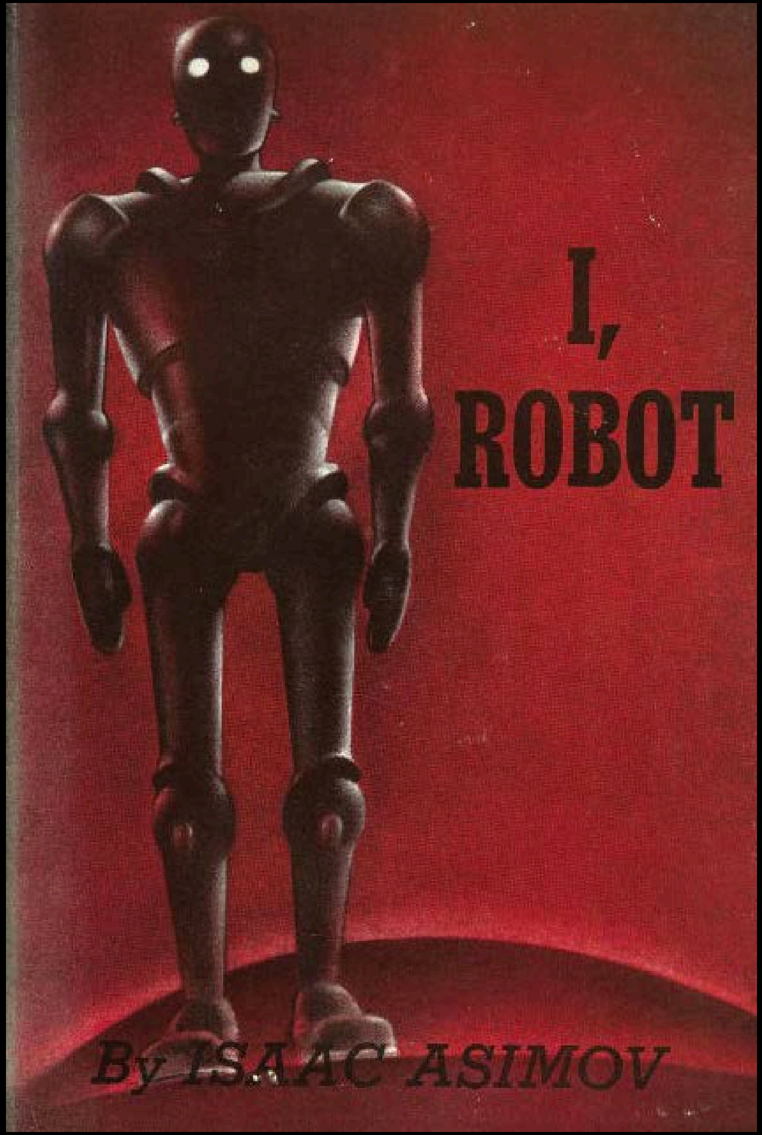
The ML pipeline circa 2013

Data → ML Algorithm → My curve is better than your curve → Write a paper



Neural Information Processing Systems Foundation

ICML


I,
ROBOT
By ISAAC ASIMOV

Retail

PageRank
Search

Coupons

Movie Distribution

Networking

Disruptive companies differentiated by

INTELLIGENT APPLICATIONS

using

Machine Learning

Music

Campaigning

Adsense
Advertising

Real Estate

Human Resources

Legal Advice

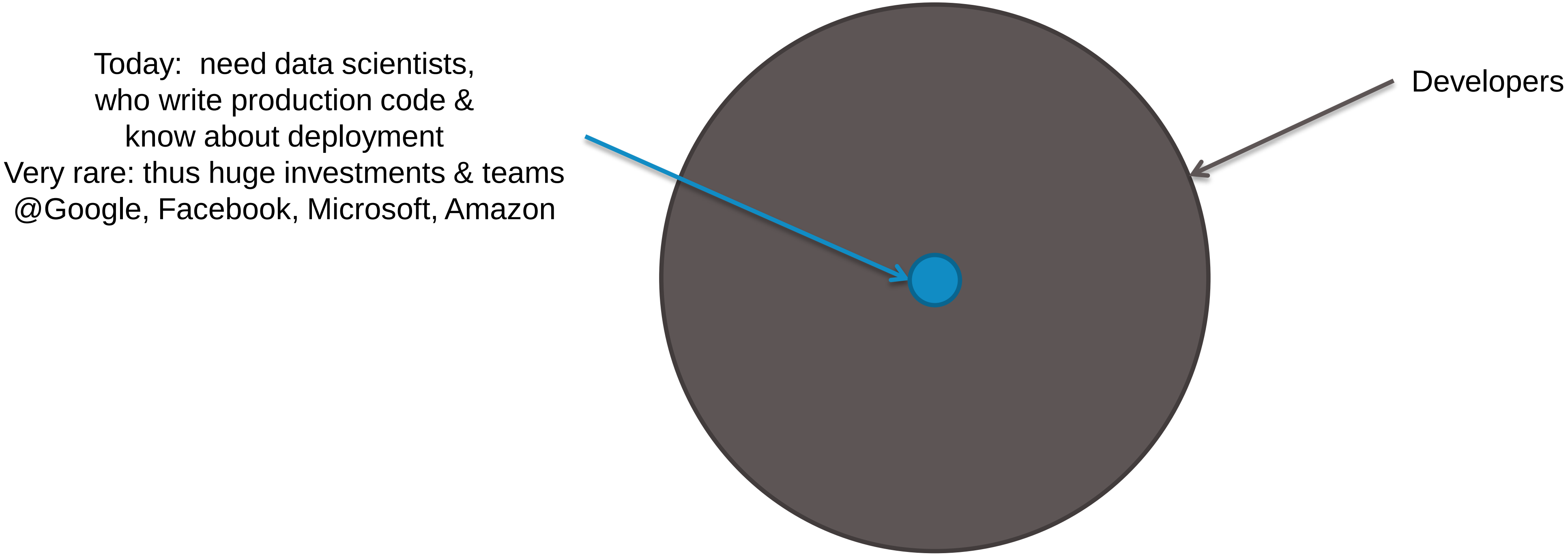
Dating

UBER
Taxis

CRM

Wearables

In 5 years, every successful app will be intelligent



Promise will only come to be if we

change how ML is done

Dato’s mission is to

accelerate the creation of intelligent applications

by making

sophisticated machine learning

as easy as

“Hello world!”

Demo of an intelligent application made easy

Since last year…

Dato customers…

Sophisticated machine learning made easy

Create Intelligence Accelerants

High-level
ML toolkits

AutoML

tune params, model selection,…
➔
so you can focus on
creative parts

Reusable
features

transferrable feature engineering
➔
accuracy with less data &
less effort

High-level ML toolkits
get started with 4 lines of code, then modify, blend, add yours…

Recommender	Image search	Sentiment analysis	Data matching	Auto tagging	Churn predictor

```
import graphlab as gl
data = gl.SFrame.read_csv('my_data.csv')
model = gl.recommender.create(data,
                 user_id='user',
                    item_id='movie’,
                    target='rating')
recommendations = model.recommend(k=5)
```

User segmentation	Data completion
Summarization	…

GraphLab Create includes easy to use, deep learning on multi-GPUs

```
graphlab.deeplearning.create(data,target=label')
```

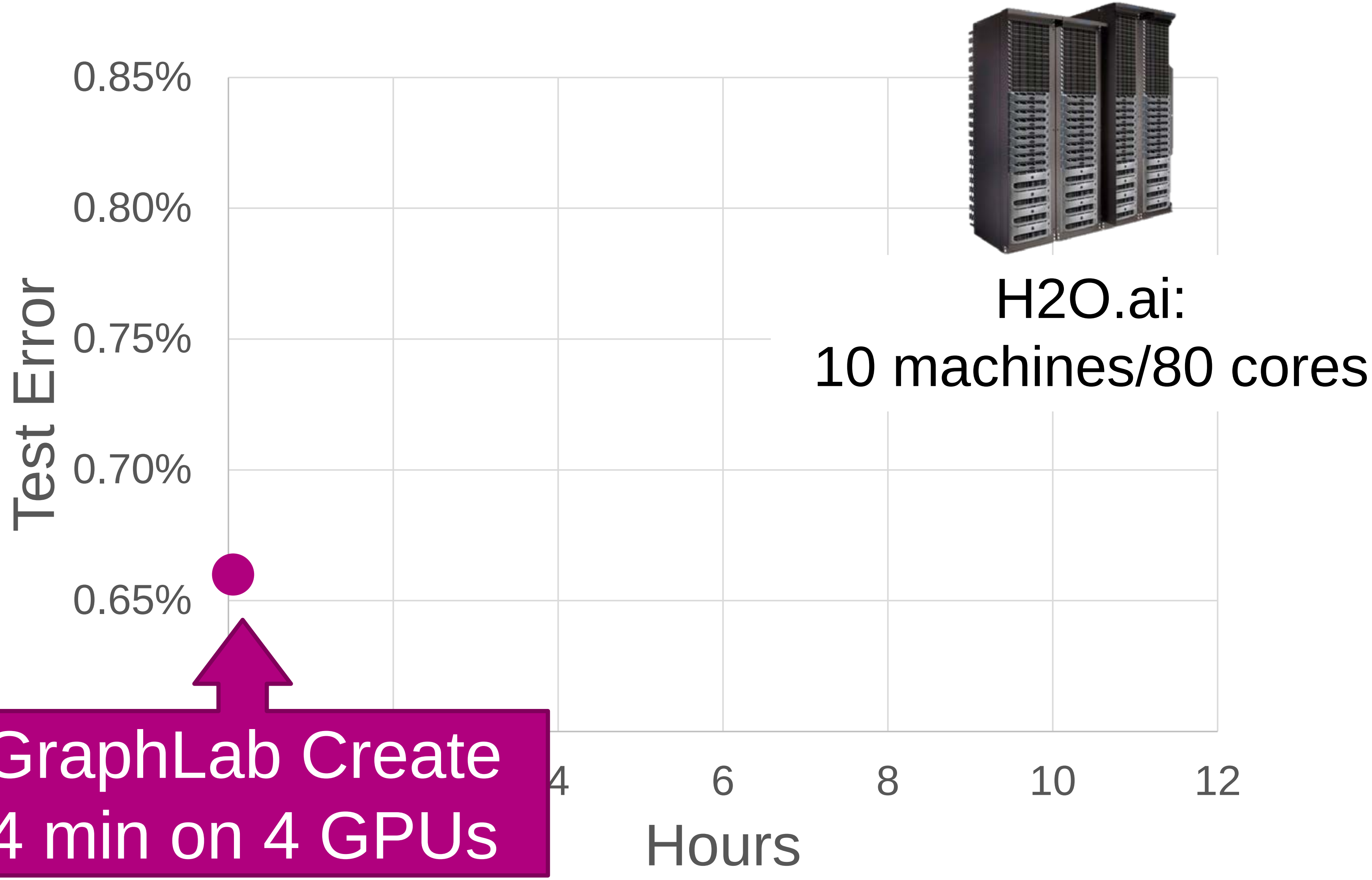
Deep learning in 1 line of code

You can also open the box and add your own layers

Average Pooling Layer	Rectified Linear Layer
Convolution Layer	Sigmoid Layer
Dropout Layer	SoftMax Layer
Flatten Layer	SoftPlus Layer
Full Connection Layer	Sum Pooling Layer
Max Pooling Layer	Tanh Layer

Deep learning tutorial tomorrow, 4pm!

Digit recognition benchmark



Sophisticated machine learning made distributed

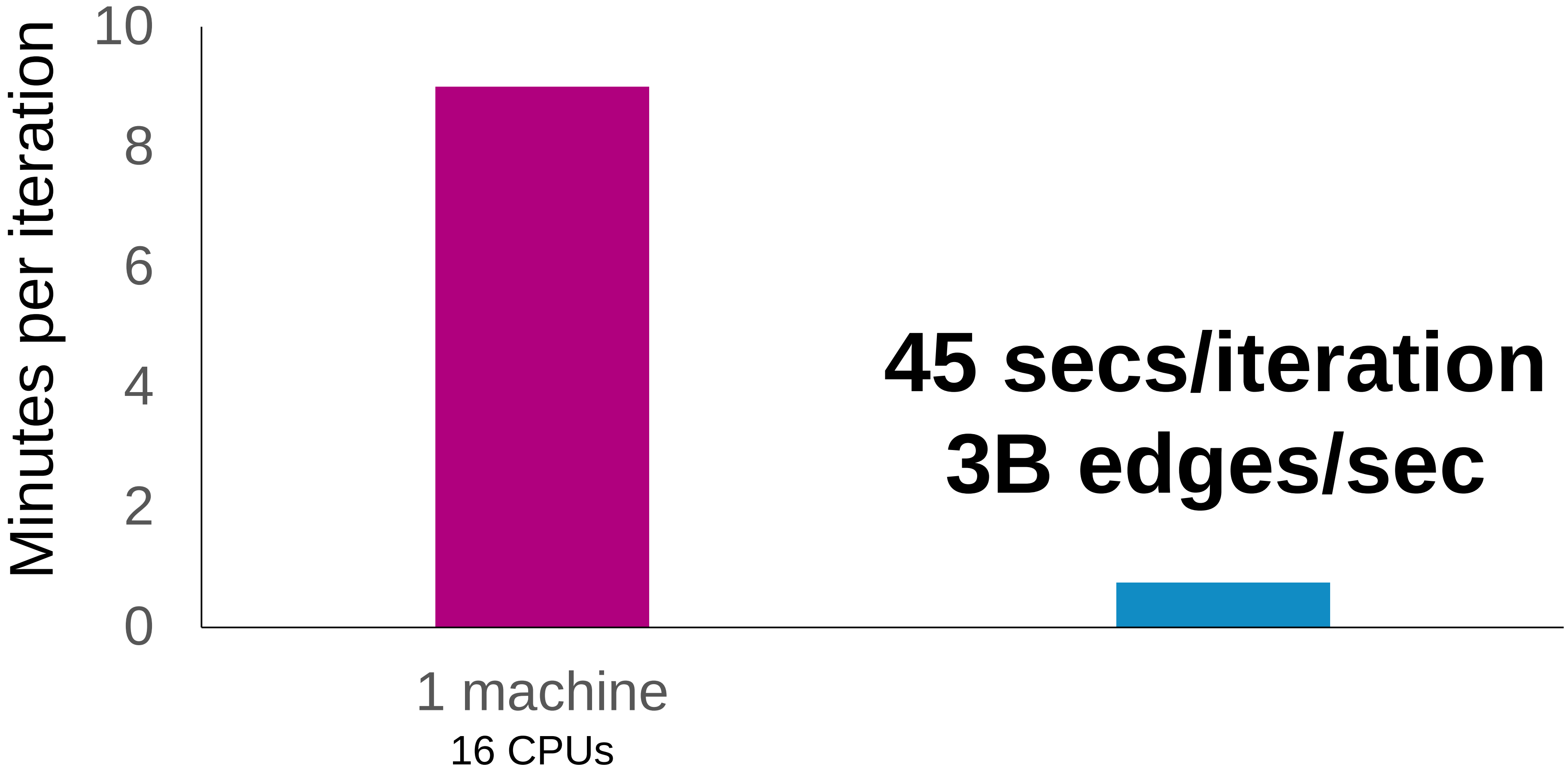
Create Intelligence on Huge Data

Distributed machine learning

Your big data infrastructure
(cloud, hadoop, spark,..)

Pagerank on Common Crawl Graph

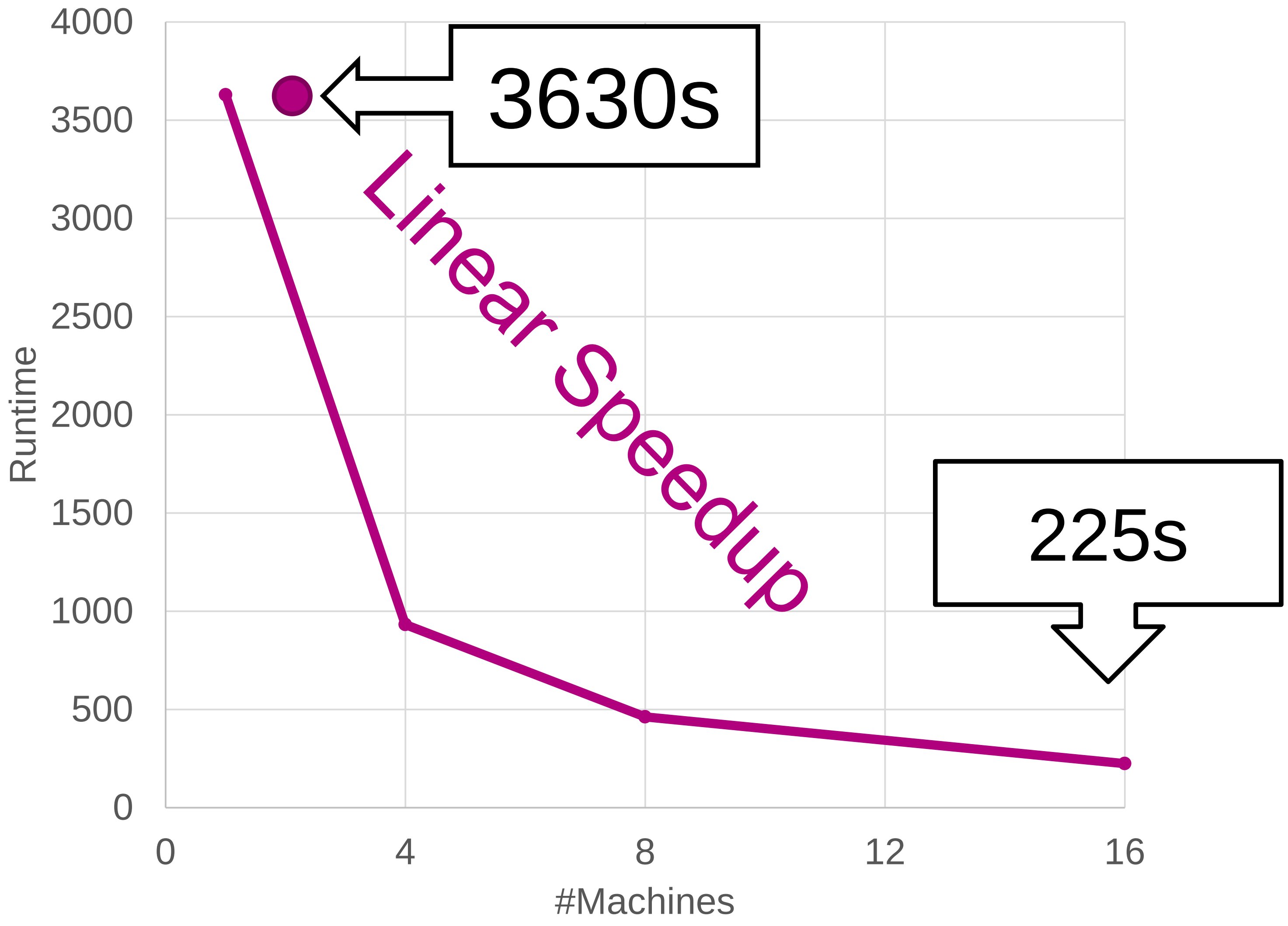
3.5 billion Nodes and 128 billion Edges



Criteo Terabyte Click Prediction

4.4 Billion Rows
13 Features

½ TB of data



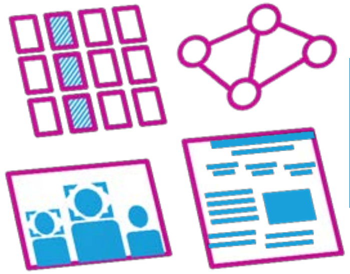
Same code, distributed ML

```
c = gl.deploy.spark_cluster.load('hdfs://…')
```

Single machine ML code

```
gl.set_distributed_execution_environment(c)
import graphlab as gl
data = gl.SFrame.read_csv('s3://…')
model = gl.classifier.create(data,
                             target='click')
```

SFrame/Sgraph summary



Tables, graphs, text, images

SFrame & SGraph

Open-source ❤

BSD license

More than 10,000 downloads

Optimized out-of-core computation for ML

High Performance

1 machine can handle:
TBs of data
100s Billions of edges

Optimized for ML

. Columnar transformation
. Create features
. Iterators
. Filter, join, group-by, aggregate
. User-defined functions
. Easily extended through SDK