The Technology Behind GraphLab Create

Yucheng Low, PhD
Chief Architect

GraphLab

GraphLab Philosophy

Users-First Architecture

User

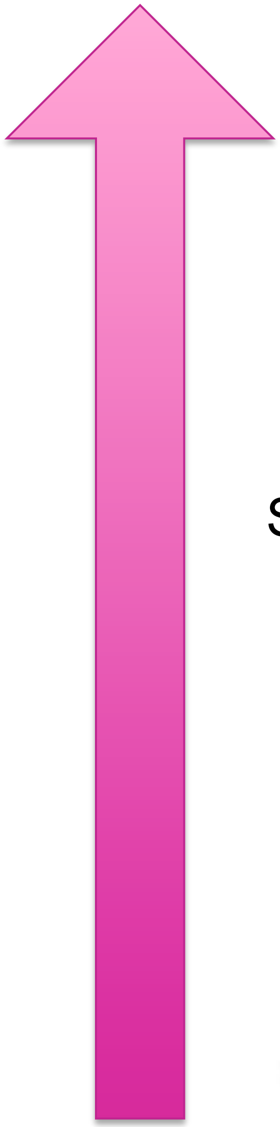
Architecture

Systems

User

Architecture

Systems



Systems-First Architectures

Systems define constraints.
Optimize for performance.

PowerGraph

User

Architecture

Systems

Users-First Architectures

Users define constraints.
Optimize for user interaction.

What is a Users-First Architecture for Data Science?

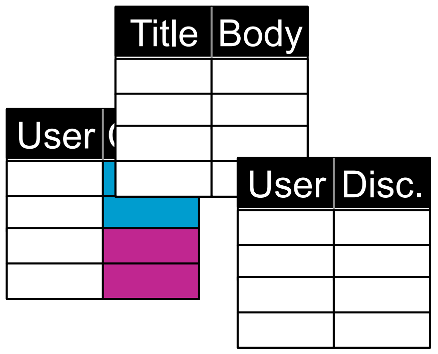
SFrame and SGraph

Built *by* data scientists,

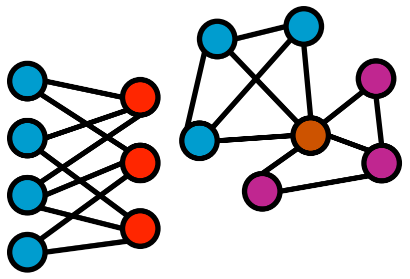
for data scientists.

Building on decades of database and systems research.

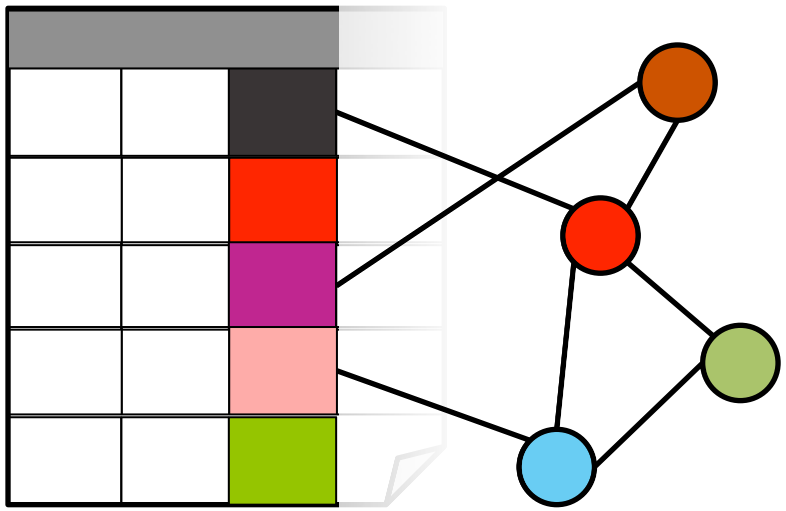
SFrame: Scalable Tabular Data Manipulation



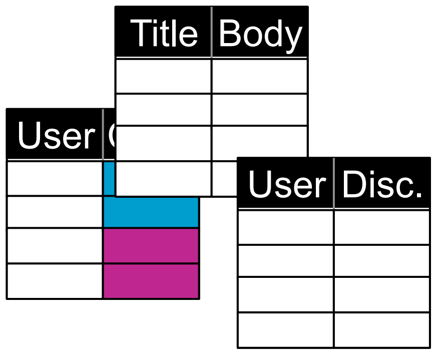
SGraph: Scalable Graph Manipulation



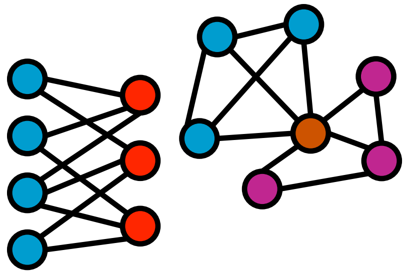
Enabling users to easily and efficiently translate between both representations to get the best of both worlds.



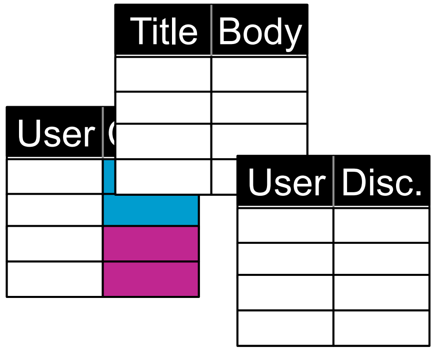
SFrame: Scalable Tabular Data Manipulation



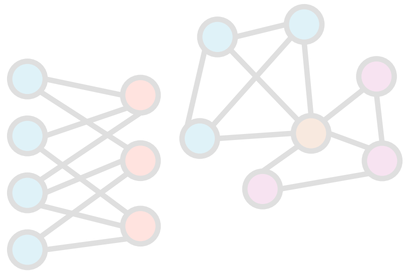
SGraph: Scalable Graph Manipulation



SFrame: Scalable Tabular Data Manipulation



SGraph: Scalable Graph Manipulation



SFrame Design

Pain Point #1: Resource Limits

Jobs fail because:

- Machine run out of memory
- Did not set Java Heap Size correctly
- Resource Configuration X needs to be bigger.

SFrame Design

- **Graceful Degradation as 1st principle**
  - Always Works

Pain Point #2: Too Strict or Too Weak Schemas

We want strong schema types.
We also want weak schema types.
Missing Values

SFrame Design

- **Graceful Degradation as 1st principle**
  - Always Works
- **Rich Datatypes**
  - Strong schema types: int, double, string...
  - Weak schema types: list, dictionary

SFrame Design

- **Graceful Degradation as 1st principle**
  - Always Works
- **Rich Datatypes**
  - Strong schema types: int, double, string...
  - Weak schema types: list, dictionary

Pain Point #3: Feature Manipulation

Difficult or costly to inspect existing features and create new features.
Hard to perform data exploration.

SFrame Design

- **Graceful Degradation as 1st principle**
  - Always Works
- **Rich Datatypes**
  - Strong schema types: int, double, string...
  - Weak schema types: list, dictionary
- **Columnar Architecture**
  - Easy feature engineering + Vectorized feature operations.
  - Immutable columns + Lazy evaluation
  - Statistics + visualization + sketches

SFrame Python API Example

Make a little SFrame of 1 column and 5 values:

```
sf = gl.SFrame({‘x’:[1,2,3,4,5]})
```

Normalizes the column x:

```
sf[‘x’] = sf[‘x’] / sf[‘x’].sum()
```

Uses a python lambda to create a new column:

```
sf[‘x-squared’] = sf[‘x’].apply(lambda x: x*x)
```

Create a new column using a vectorized operator:

```
sf[‘x-cubed’] = sf[‘x-squared’] * sf[‘x’]
```

Create a new SFrame taking only 2 of the columns:

```
sf2 = sf[[‘x’,’x-squared’]]
```

SFrame Querying

Supports most typical SQL SELECT operations using a Pythonic syntax.

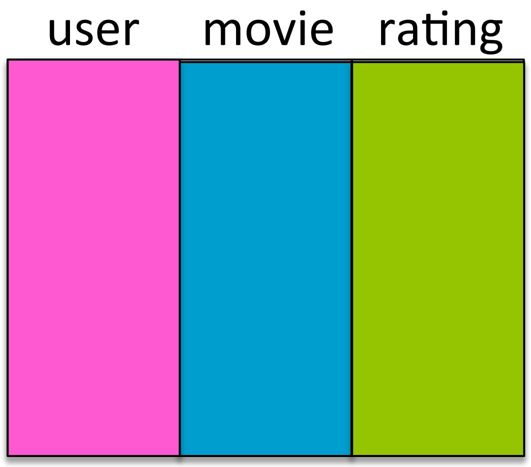
SQL

```
SELECT Book.title AS title, COUNT(*) AS authors
    FROM Book
    JOIN Book_author ON Book.isbn = Book_author.isbn
    GROUP BY Book.title;
```

SFrame Python

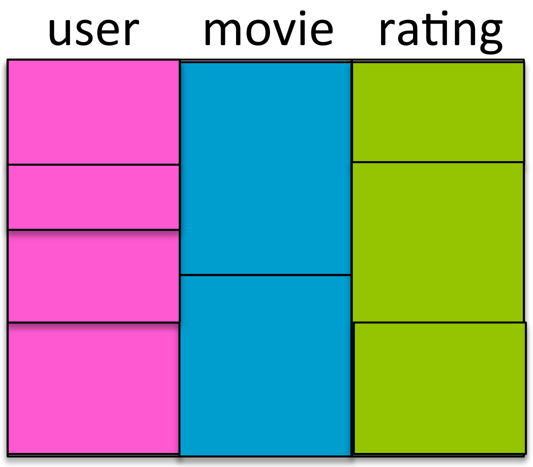
```
Book.join(Book_author, on=‘isbn’)
    .groupby(‘title’, {‘authors’:gl.aggregate.COUNT})
```

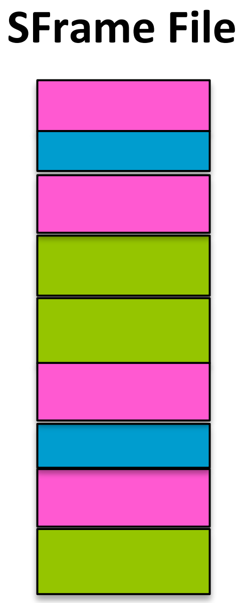
SFrame Columnar Encoding



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

SFrame Columnar Encoding



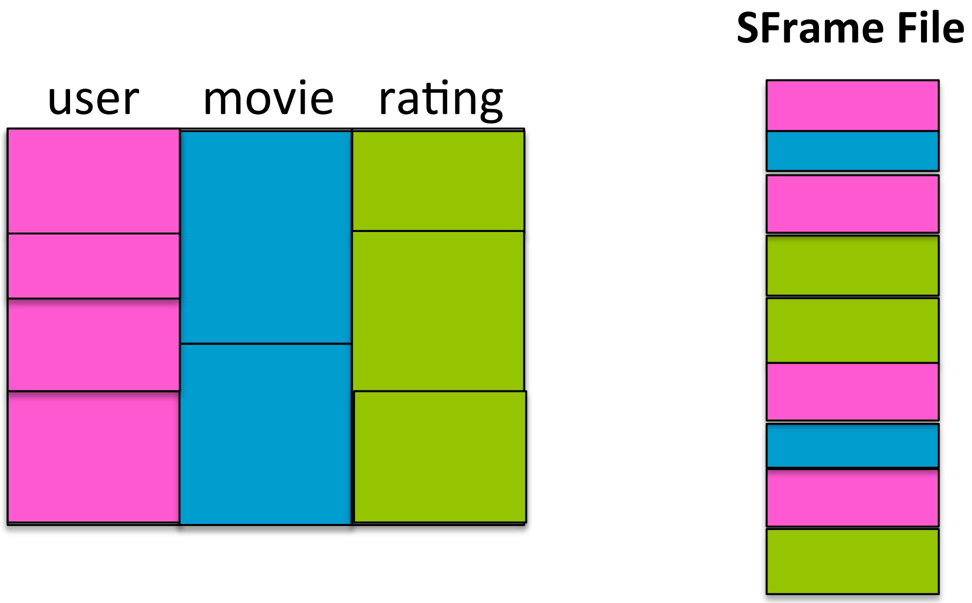


Type aware compression:

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

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

SFrame Columnar Encoding

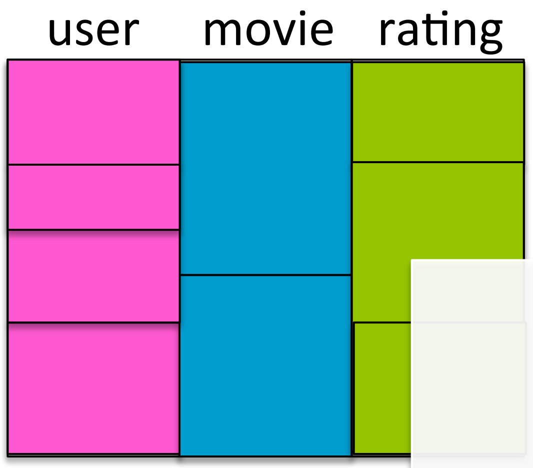


Type aware compression:
- Variable Bit length Encode
- Frame Of Reference Encode
- ZigZag Encode
- Delta / Delta ZigZag Encode
- Dictionary Encode
- General Purpose LZ4

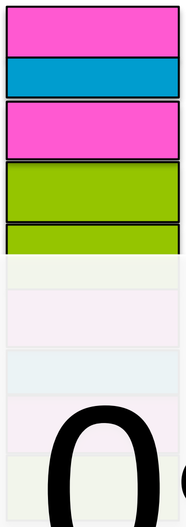
Netflix Dataset,
99M rows, 3 columns, ints
1.4GB raw
289MB gzip compressed

User	→	176 MB	14.2 bits/int
Movie	→	**257 KB**	0.02 bits/int
Rating	→	47 MB	3.8 bits/int
Total	→	**223MB**	

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

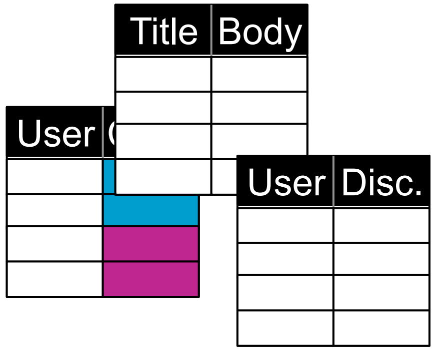
Total → **223MB**

SFrames Distributed

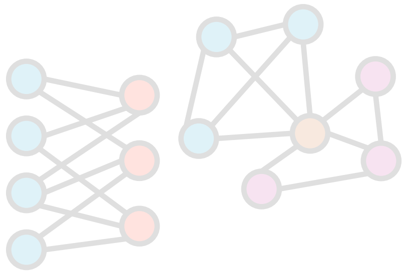
The choice of distributed or local execution is a question of query optimization.

- Distributed Dataflow
- Columnar Query Optimizations
- Communicate columnar compressed blocks rather than row tuples.

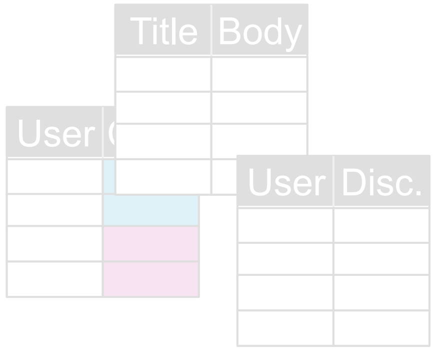
SFrame: Scalable Tabular Data Manipulation



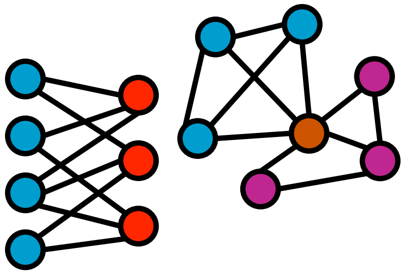
SGraph: Scalable Graph Manipulation



SFrame: Scalable Tabular Data Manipulation



SGraph: Scalable Graph Manipulation



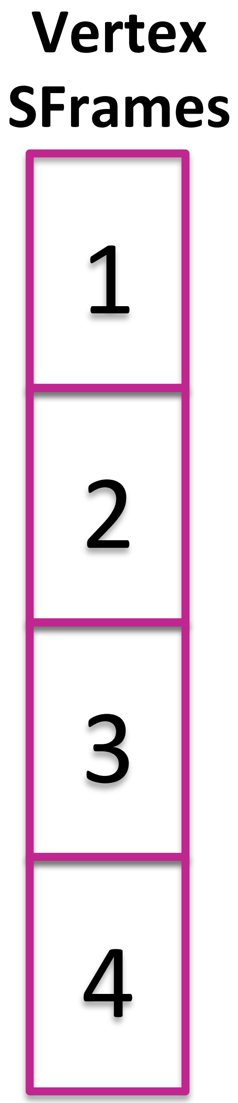
SGraph

- **SFrame backed** graph representation. Inherits SFrame properties.
  - Data types, External Memory, Columnar, compression, etc.

- Layout optimized for batch **external memory computation**.

SGraph Layout

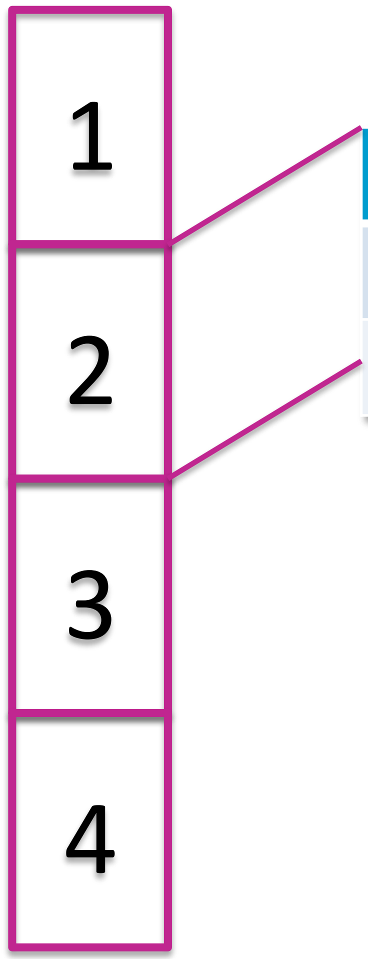
Vertices Partitioned into p = 4 SFrames.



SGraph Layout

Vertices Partitioned into p = 4 SFrames.

Vertex SFrames



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

SGraph Layout

Edges partitioned into p^2 = 16 SFrames.

Vertex SFrames

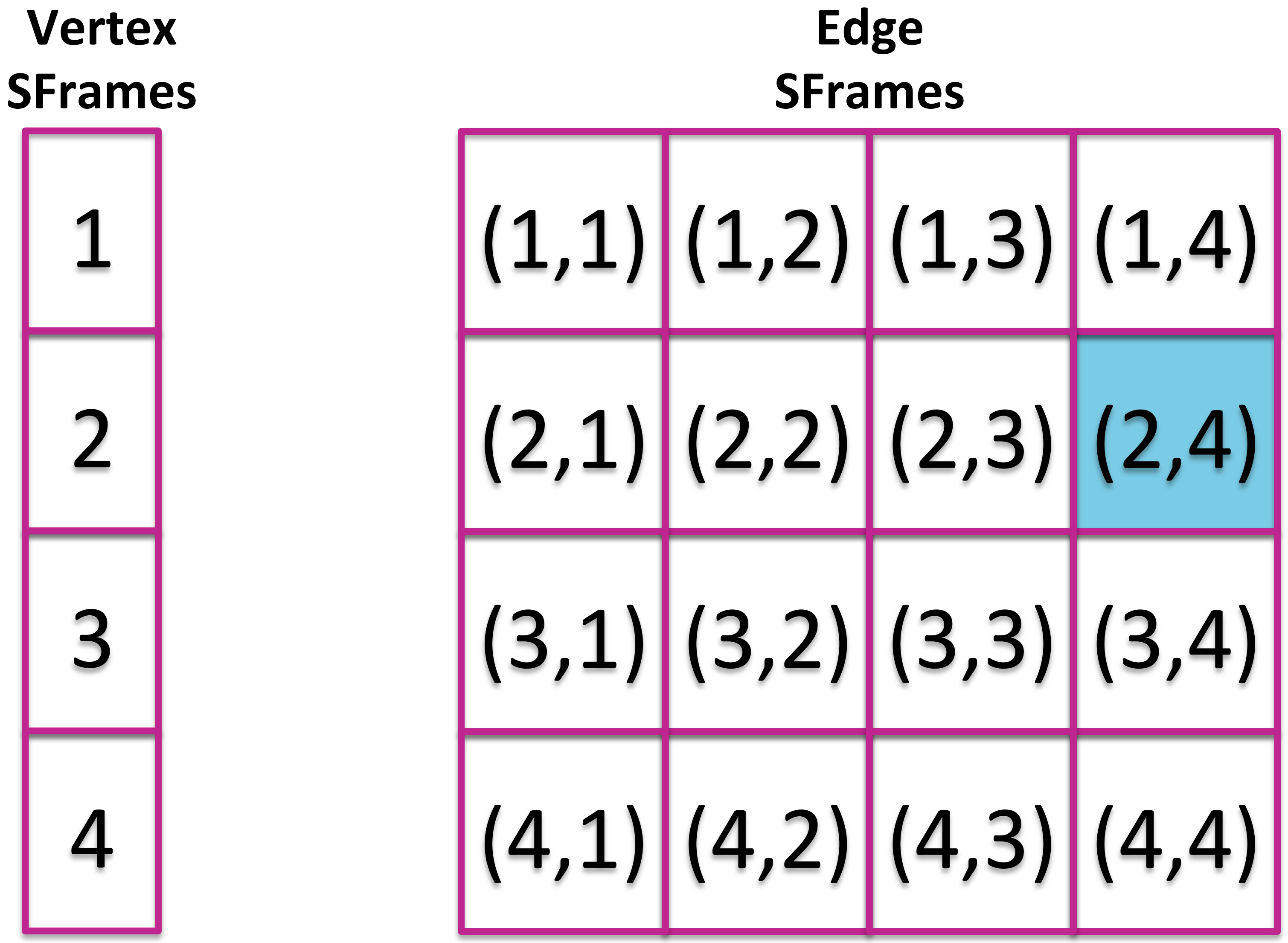
1
2
3
4

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)

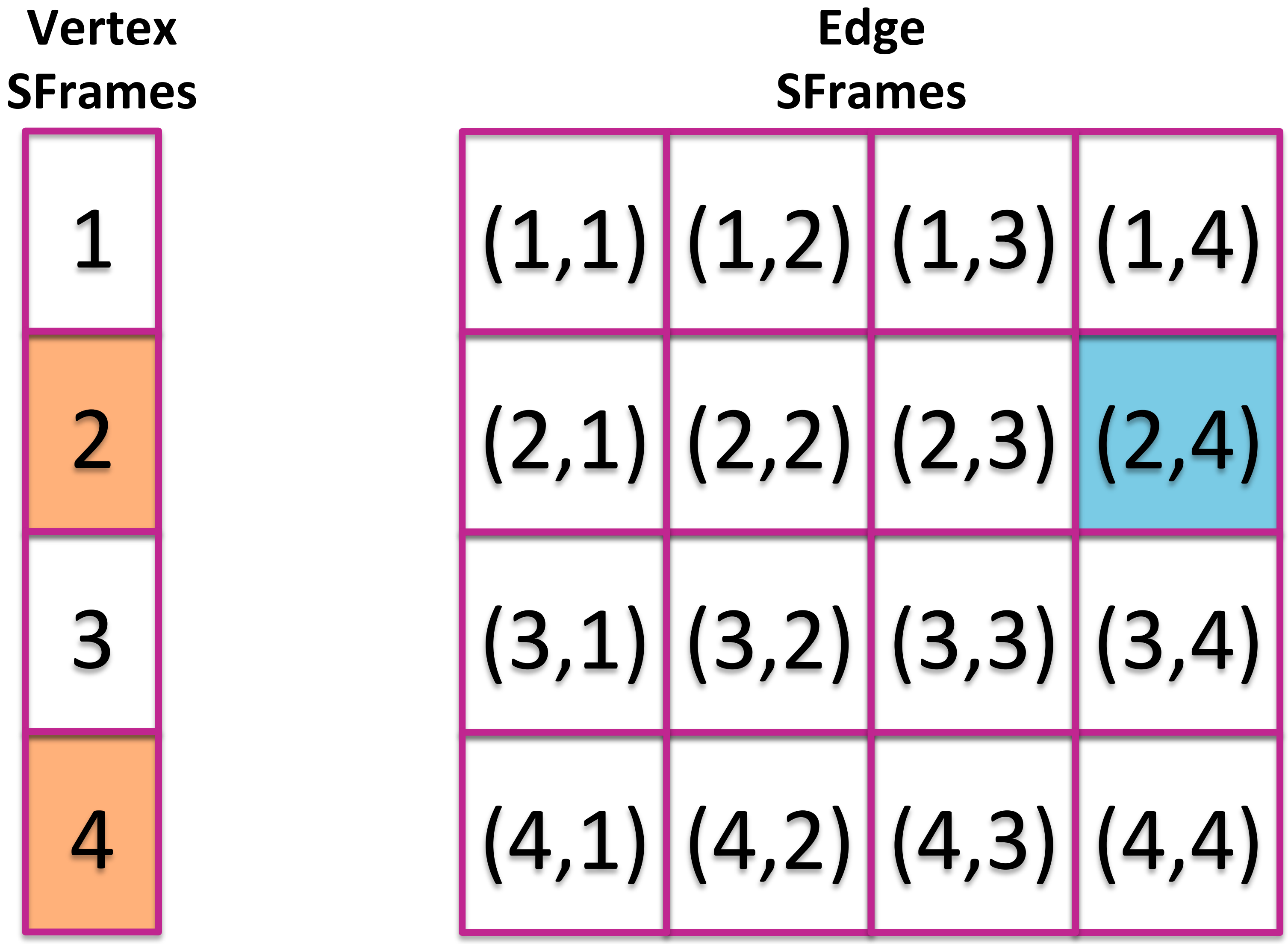
SGraph Layout

Edges partitioned into p^2 = 16 SFrames.

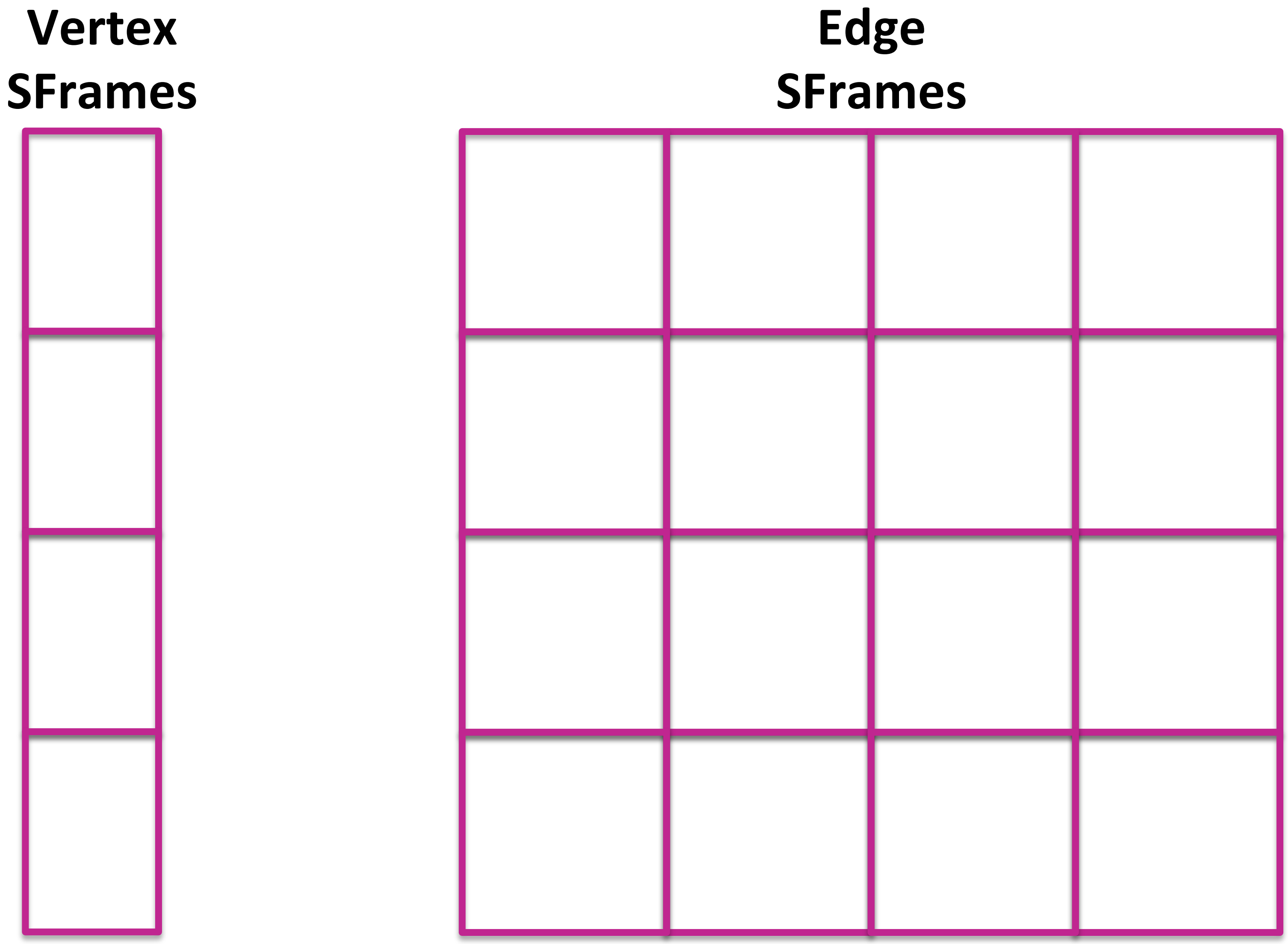


SGraph Layout

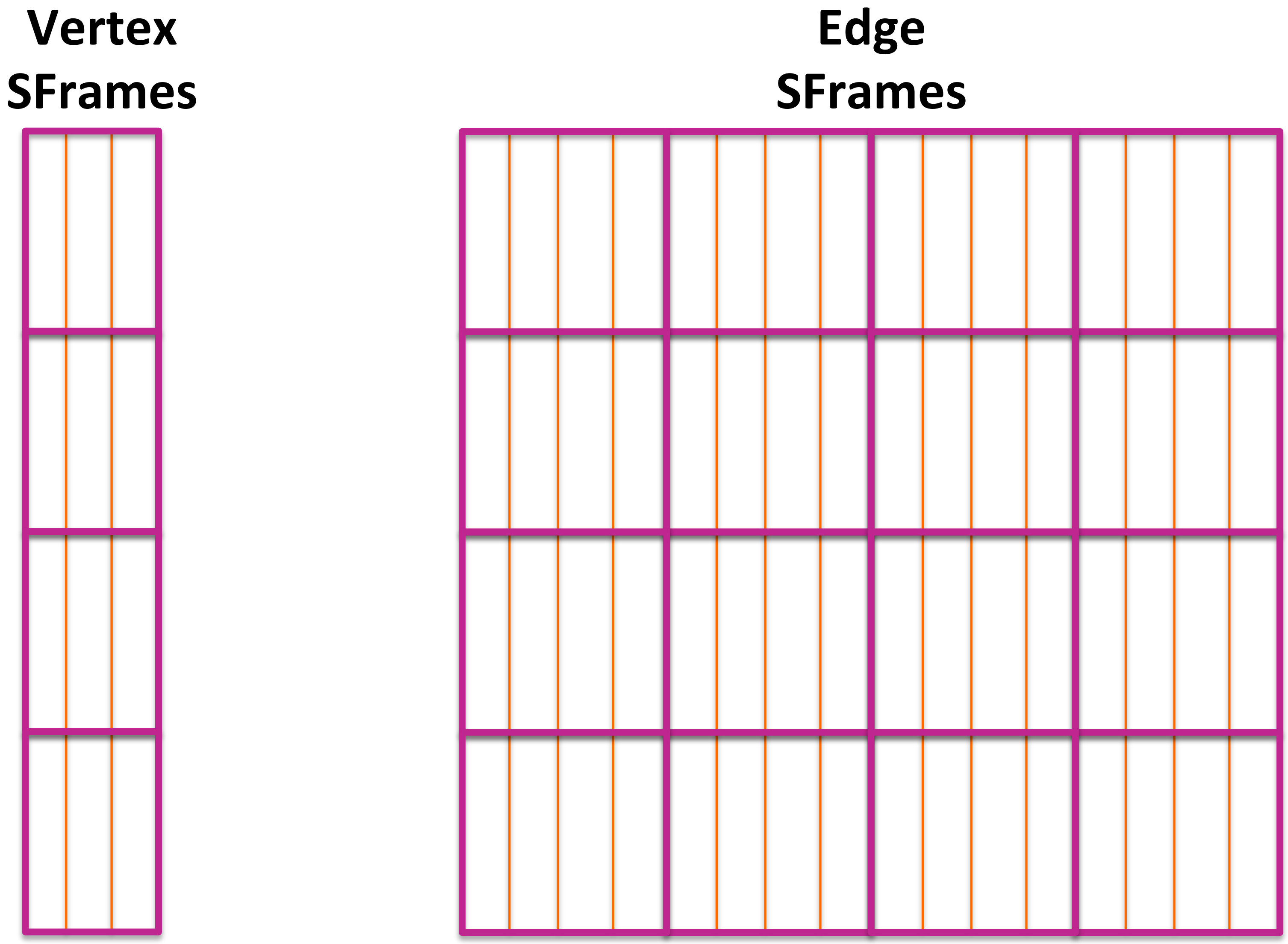
Edges partitioned into p^2 = 16 SFrames.



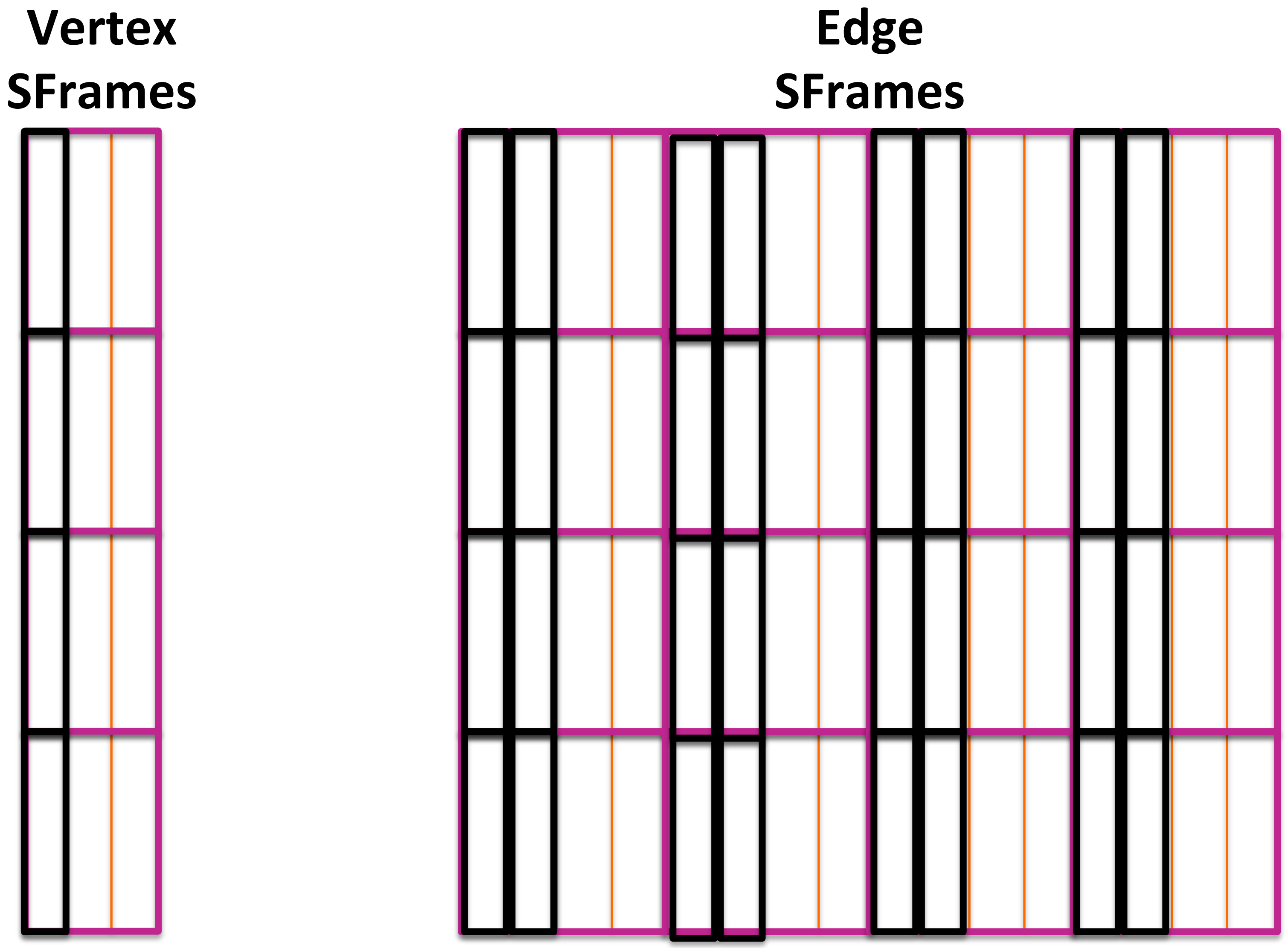
SGraph Layout



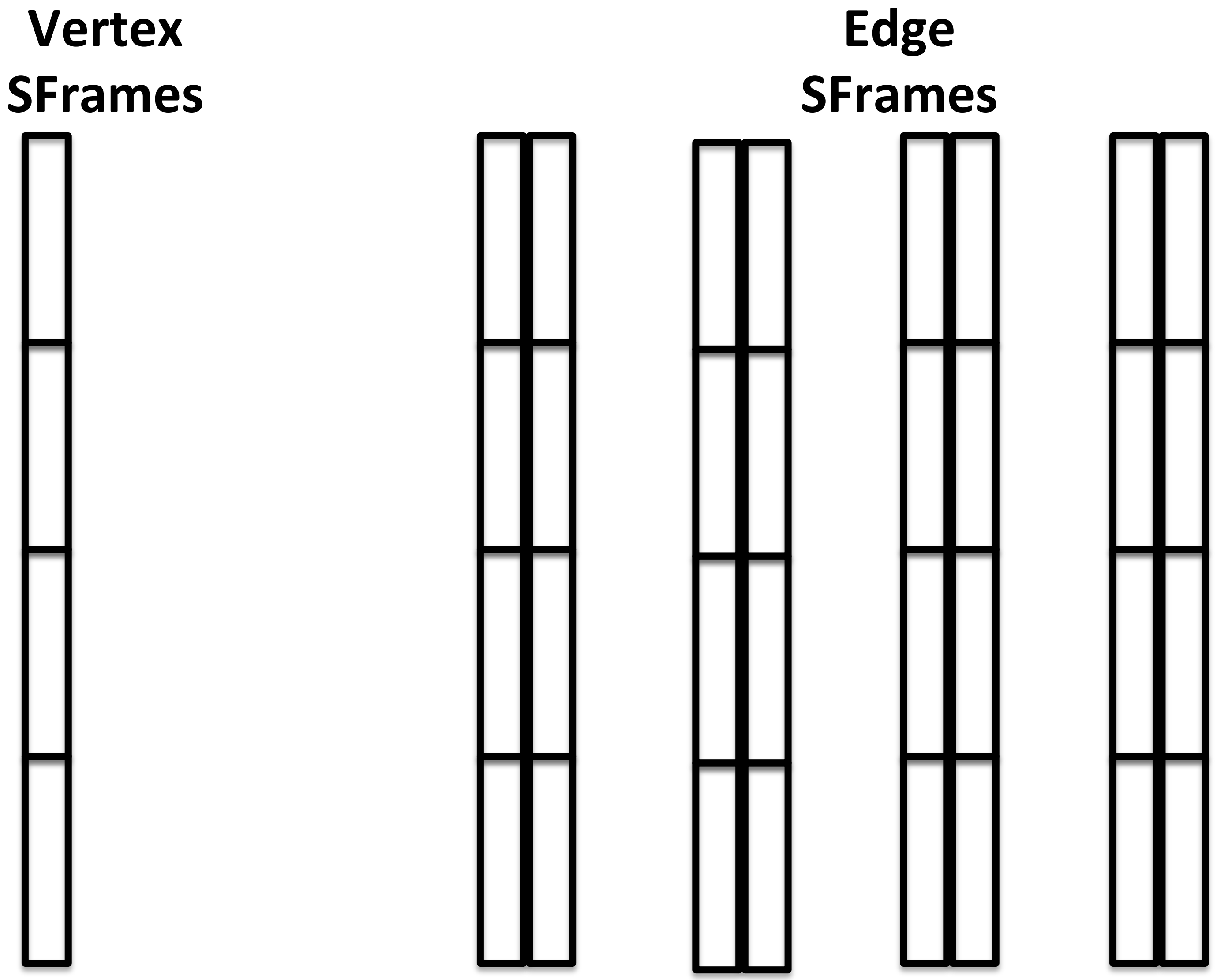
SGraph Layout



SGraph Layout



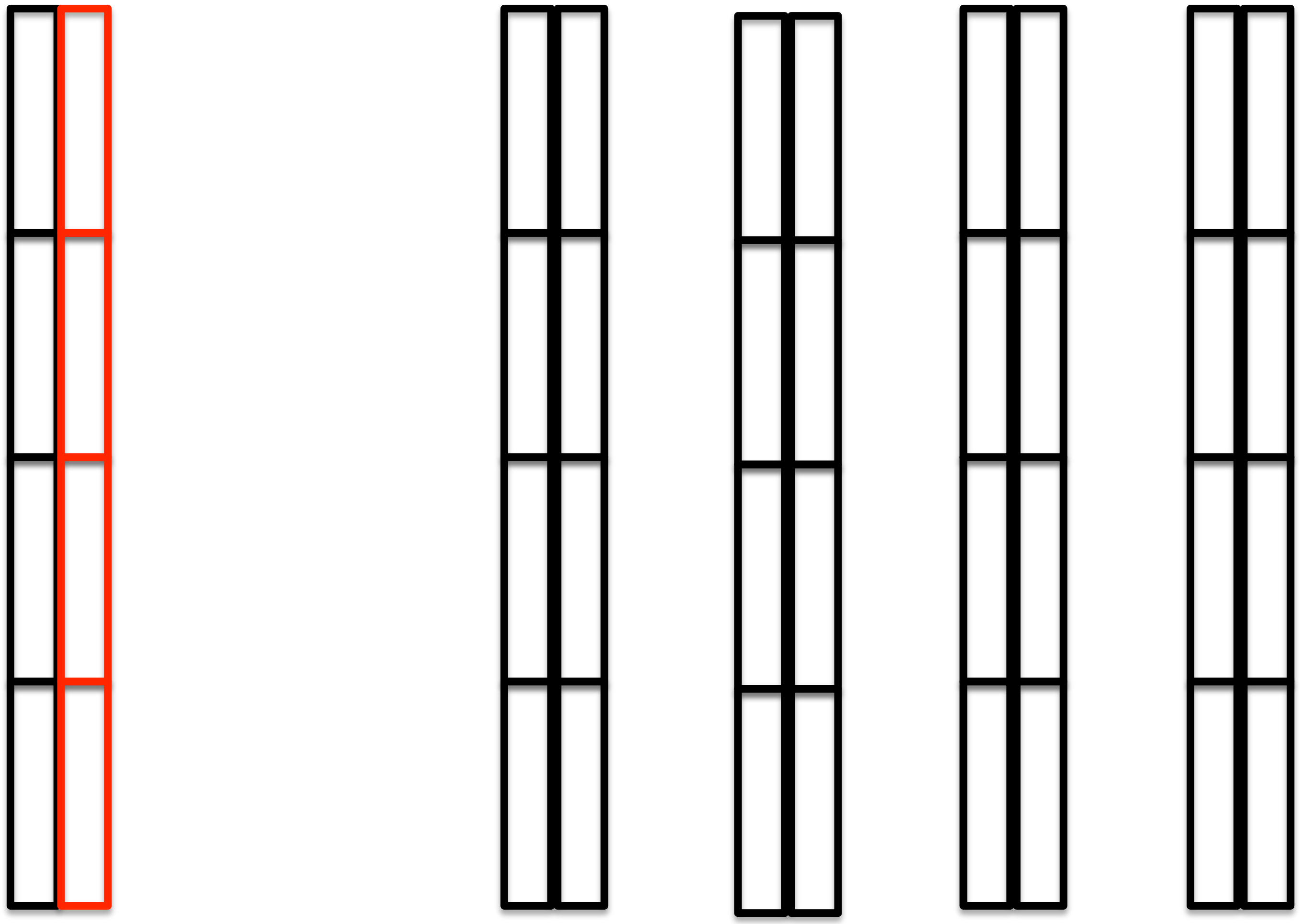
SGraph Layout



SGraph Layout

Vertex SFrames

Edge SFrames

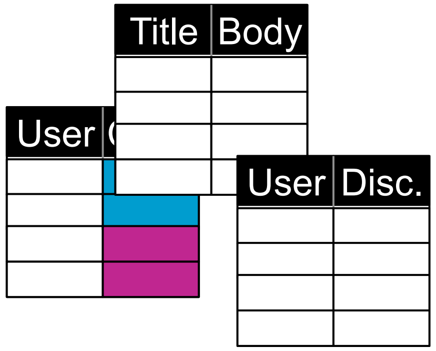


Deep Integration of SFrames and SGraphs

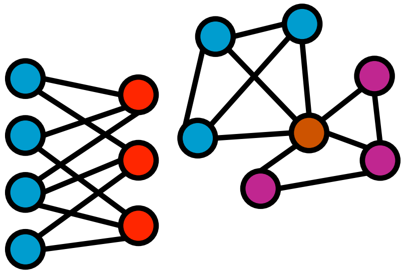
- Seamless interaction between graph data and table data.
- Queries can be performed easily across graph and tables.

Demo

SFrame: Scalable Tabular Data Manipulation



SGraph: Scalable Graph Manipulation



User-first architecture.

Built *by* data scientists,

for data scientists.