Graph and Search at Linkedin

Swee Lim

SIGIR Graph Search and Beyond

SIGIR 2015

About Linkedin

Our Vision

Create economic opportunity for every member of the global workforce

Our Mission

Connect the world's *professionals* and make them more productive and successful

The Economic Graph

Identity

Member's professional profile of record

Network

Connect, follow, employment, education, …

Entities

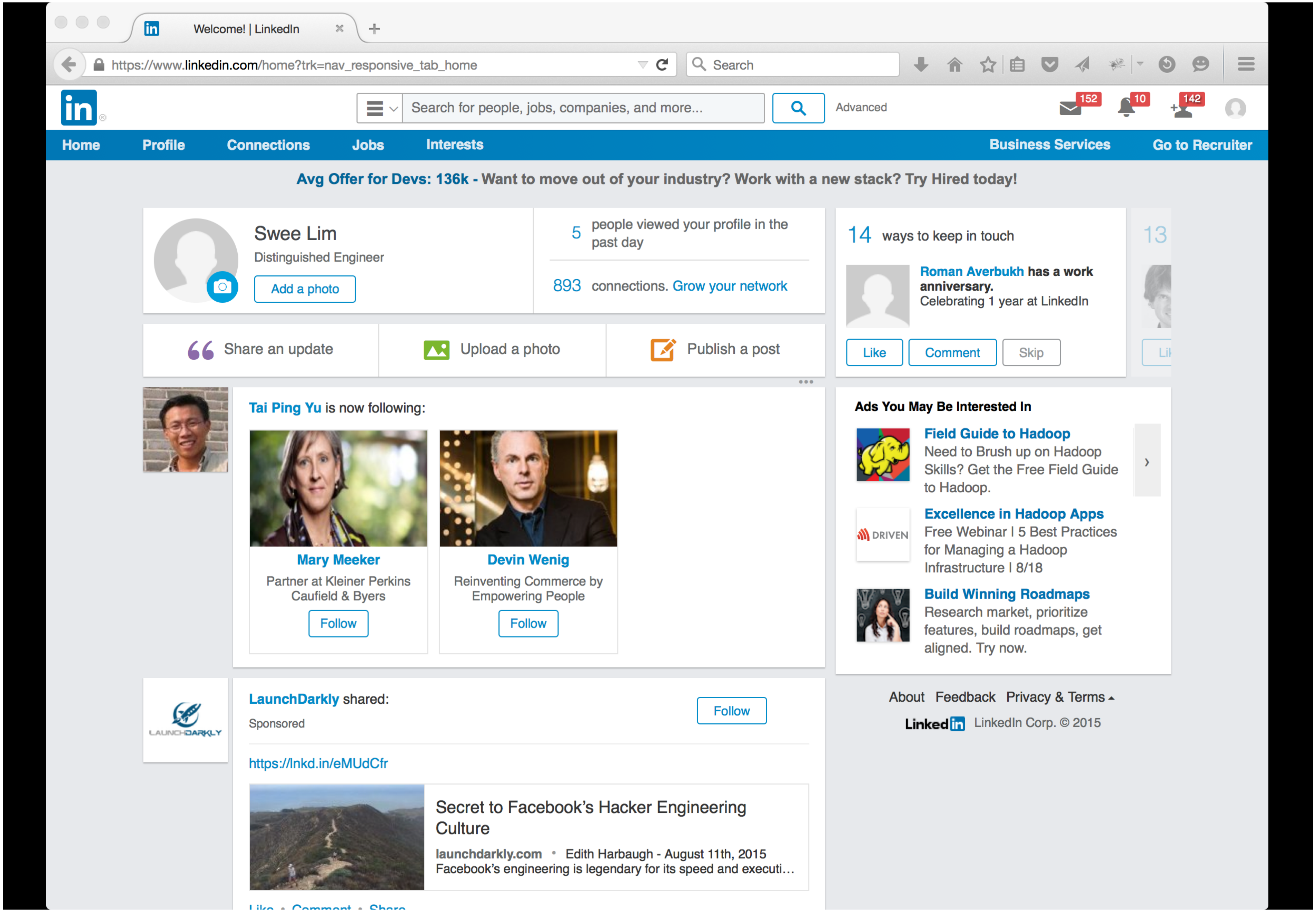
Companies, Schools, Jobs, Skills, Articles, Locations, …

For our members

Discover, Learn, Find and to be Found

For our customers

Hire, Market, and Sell


Welcome! | LinkedIn
https://www.linkedin.com/home?trk=nav_responsive_tab_home
Search
Search for people, jobs, companies, and more...
Advanced
152
10
142
Home
Profile
Connections
Jobs
Interests
Business Services
Go to Recruiter
Avg Offer for Devs: 136k - Want to move out of your industry? Work with a new stack? Try Hired today!
Swee Lim
Distinguished Engineer
Add a photo
5 people viewed your profile in the past day
893 connections. Grow your network
14 ways to keep in touch
Roman Averbukh has a work anniversary.
Celebrating 1 year at LinkedIn
Like
Comment
Skip
13
Share an update
Upload a photo
Publish a post
Tai Ping Yu is now following:
Mary Meeker
Partner at Kleiner Perkins Caufield & Byers
Follow
Devin Wenig
Reinventing Commerce by Empowering People
Follow
Ads You May Be Interested In
Field Guide to Hadoop
Need to Brush up on Hadoop Skills? Get the Free Field Guide to Hadoop.
Excellence in Hadoop Apps
Free Webinar I 5 Best Practices for Managing a Hadoop Infrastructure I 8/18
Build Winning Roadmaps
Research market, prioritize features, build roadmaps, get aligned. Try now.
About Feedback Privacy & Terms
LinkedIn Corp. © 2015
LaunchDarkly shared:
Sponsored
Follow
https://lnkd.in/eMUdCfr
Secret to Facebook's Hacker Engineering Culture
launchdarkly.com • Edith Harbaugh - August 11th, 2015
Facebook's engineering is legendary for its speed and executi…

How do we use the graph?

Graph is mostly implicit

It affects almost everything you see,
e.g. feed, search, names, profiles

Online

- Most pages make multiple calls to the “online” graph
- For dynamic content, such as feed, search, profile (name) visibility

Offline

- Available in offline systems such as Hadoop tables
- For more “static” content, such as recommendations, such as *People You May Know (PYMK)*

Interesting Economic Graph Queries
(answered online)

What to Pay Attention To

“The 10 most commonly followed entities by people in the industries of my most recent 2 employers and my second-degree network”

Database Tribes

“People who are connected and have worked on the same project at two or more jobs at least one of which in the database industry”

Marketing Jobs in Energy

“Senior marketing job postings at Bay Area companies relevant to the term ‘energy’ aggregated by month for the past year”

First-degree interconnections

“All interconnections between members of a person’s first degree network”

What do these queries have in common?

Deep, complex join structure

Large fan-out

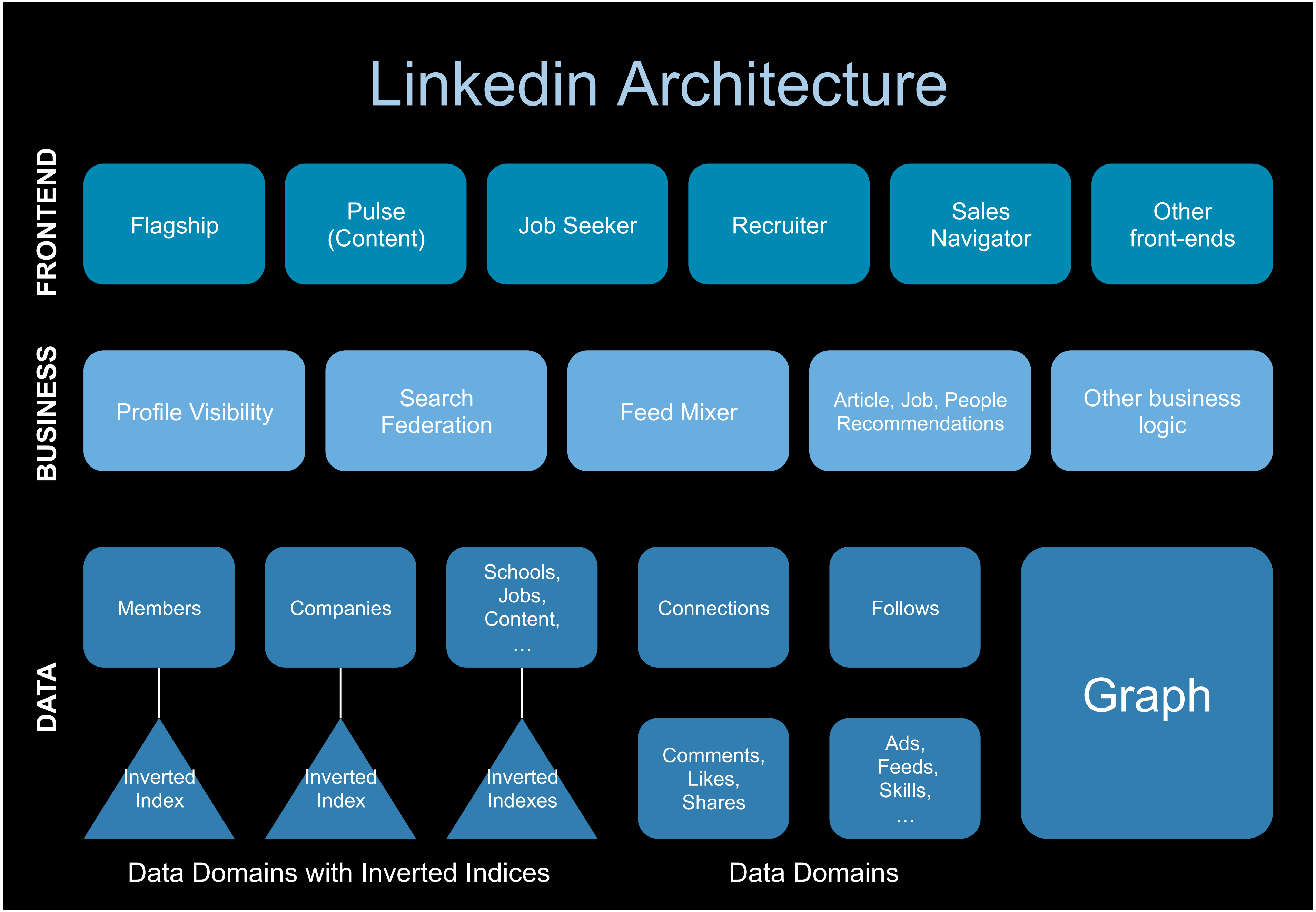
(Richard Branson has millions of followers)

Skew

(Most have fewer followers)

What do we need?

Fast and efficient joins

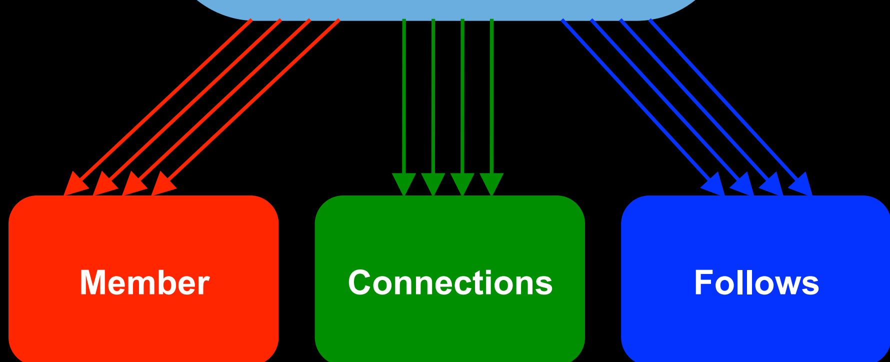

Linkedin Architecture
FRONTEND
Flagship
Pulse (Content)
Job Seeker
Recruiter
Sales Navigator
Other front-ends
BUSINESS
Profile Visibility
Search Federation
Feed Mixer
Article, Job, People Recommendations
Other business logic
DATA
Members
Companies
Schools, Jobs, Content, …
Connections
Follows
Graph
Inverted Index
Inverted Index
Inverted Indexes
Comments, Likes, Shares
Ads, Feeds, Skills, …
Data Domains with Inverted Indices
Data Domains

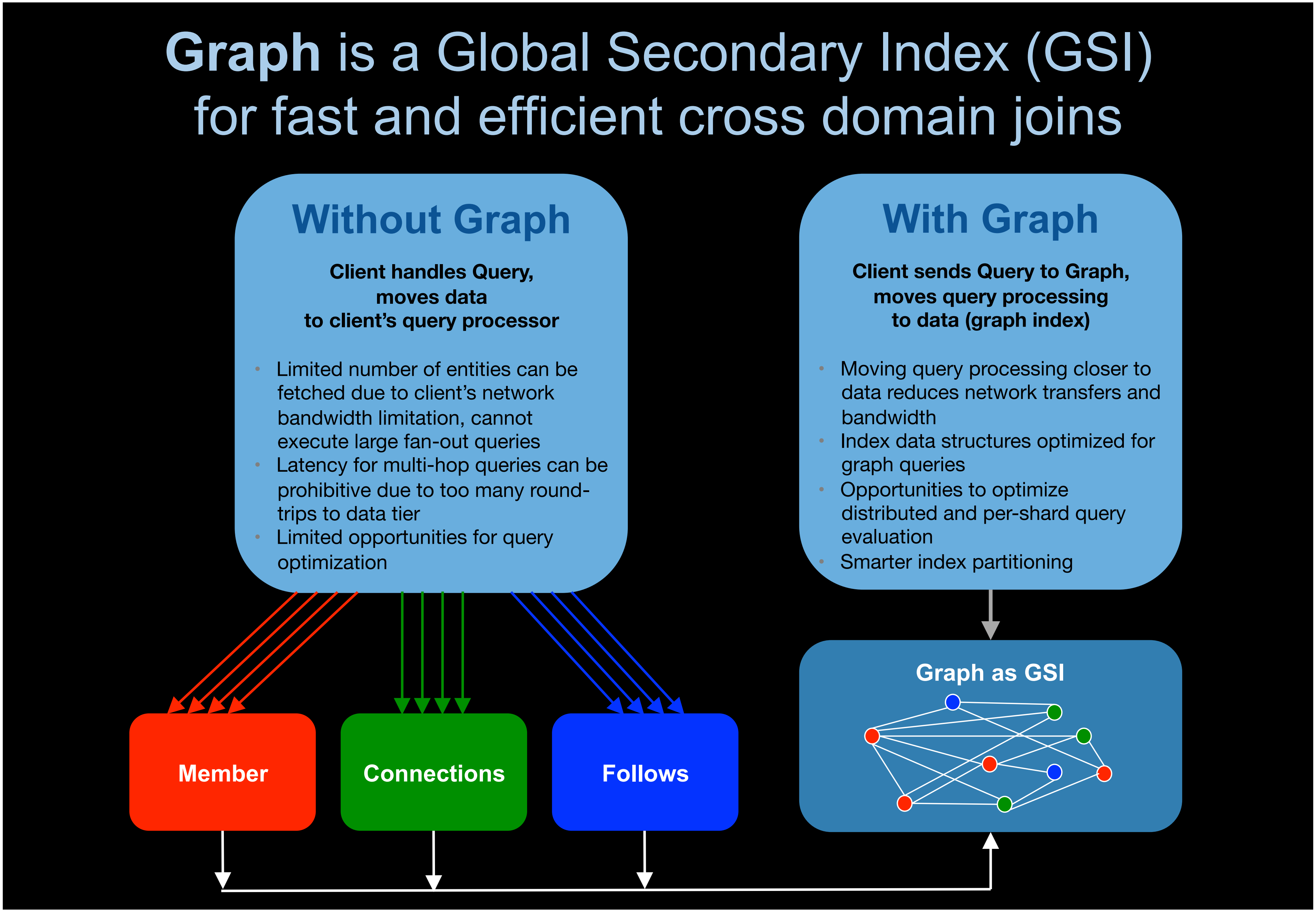
Why do we need Graph?

Without Graph

Client handles Query, moves data to client's query processor

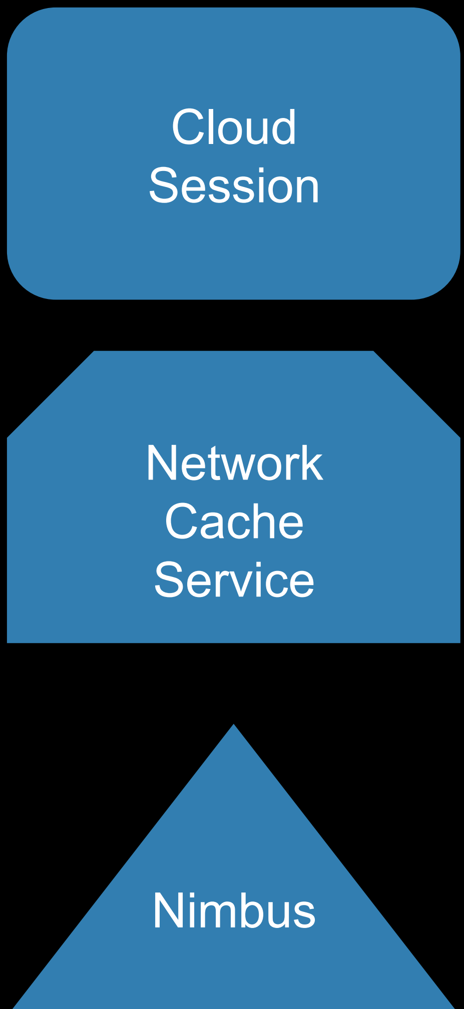
- Limited number of entities can be fetched due to client's network bandwidth limitation, cannot execute large fan-out queries
- Latency for multi-hop queries can be prohibitive due to too many round-trips to data tier
- Limited opportunities for query optimization





Graph is a Global Secondary Index (GSI)
for fast and efficient cross domain joins
Without Graph
Client handles Query,
moves data
to client's query processor
• Limited number of entities can be fetched due to client's network bandwidth limitation, cannot execute large fan-out queries
• Latency for multi-hop queries can be prohibitive due to too many round-trips to data tier
• Limited opportunities for query optimization
With Graph
Client sends Query to Graph,
moves query processing
to data (graph index)
• Moving query processing closer to data reduces network transfers and bandwidth
• Index data structures optimized for graph queries
• Opportunities to optimize distributed and per-shard query evaluation
• Smarter index partitioning
Member
Connections
Follows
Graph as GSI

Current 3^{rd} Generation Graph (~5 years old)



- Provides API end-point called by clients
- Specific operations for 1^{st} degree, 2^{nd} degree, network sizes, common entities, set operations, paths
- General queries using GQL highly restricted

- Extensive caching based on understanding of data for expensive queries, can be stale
- Member's 2^{nd} degree connections
- Network sizes > 1^{st} degree
- Influencer follower counts (e.g. Richard Branson)

- Term partitioned by source of relationship
- Sorted adjacency list (like an inverted index)
- Optimized to return 1^{st} degree connections
- Example : Member connected to Member
  P3 : { 8 => 10, 42 } { 42 => 8, 77 }
  P7 : { 10 => 8 , 33 } { 77 => 42 }

Why build next generation Graph? Limitations of current generation Graph

- Initially only supported member to member connections, generalized later to support more node and edge types
- Optimized for current high volume queries, 1st degree operations
- Fixed number of bytes allocated to edge properties, fixed number and size of properties (no strings)
- No node properties
- Source and destination node types fixed for each edge type because of sorted adjacency list, e.g. cannot have generic member follow member, company, school (currently 3 different edge types)
- Cannot natively support more than 2-way relationships, e.g. member endorsed member for skill
- Common entities is not efficient due to term based partitioning scheme
- Query language and evaluation under developed, e.g. no composition, not declarative, no planning
- Old implementation assumptions, e.g. sizes of adjacency lists (fan-out for member to member connections much smaller than Richard Branson’s followers)

Liquid : our next generation graph

Enable use cases not previously possible
or efficient to execute in current system

N-way relationships   Fast-joins   Rich properties

Democratize adding and querying Graph data

No-cost schema evolution

Graph-oriented query language

Liquid Key Desirable Properties

All relations are first class

O(k) navigation
(required for fast joins)

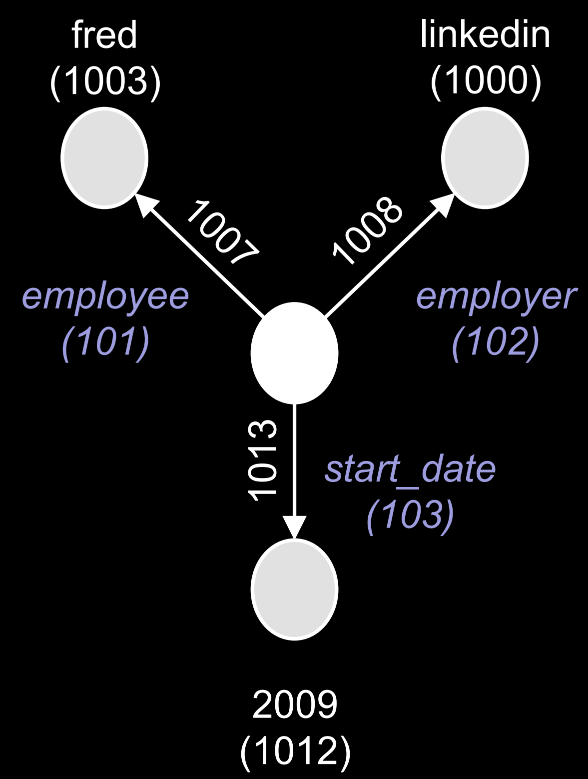
O(k) schema evolution
(easy to add and evolve a live system)

Graph oriented query language

Representing a Graph as a log of Nodes and Edges

Predicates

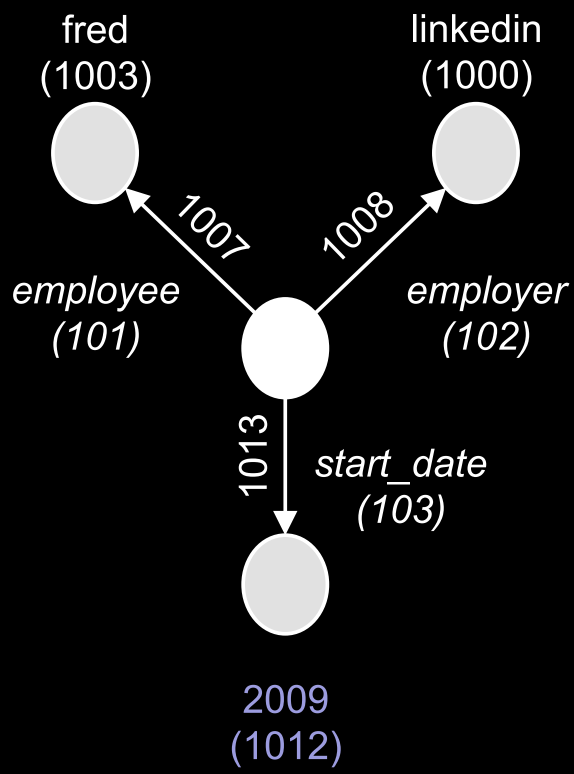
```
100:  {"name"}
101:  {"employee"}
102:  {"employer"}
103:  {"start_date"}
...
1000:  {"linkedin"}
1001:  {"LinkedIn Corporation"}
1002:  {A sub: 1000 pred: 100 obj: 1001}
1003:  {"fred"}
1004:  {"Fred M'Bogo"}
1005:  {A sub: 1003 pred: 100 obj: 1004}
1006:  {}
1007:  {A sub: 1006 pred: 101 obj: 1003}
1008:  {A sub: 1006 pred: 102 obj: 1000}
1009:  {"2008"}
1010:  {A sub: 1006 pred: 103 obj: 1009}
1011:  {D sub: 1006 pred: 103 obj: 1009}
1012:  {"2009"}
1013:  {A sub: 1006 pred: 103 obj: 1012}
```



Representing a Graph as a log of Nodes and Edges

Values

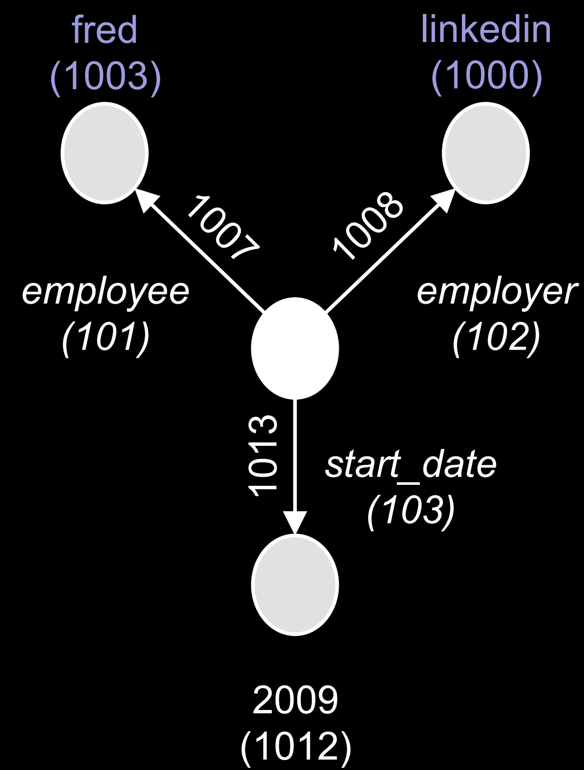
```
100: {"name"}
101: {"employee"}
102: {"employer"}
103: {"start_date"}
...
1000: {"linkedin"}
1001: {"LinkedIn Corporation"}
1002: {A sub: 1000 pred: 100 obj: 1001}
1003: {"fred"}
1004: {"Fred M'Bogo"}
1005: {A sub: 1003 pred: 100 obj: 1004}
1006: {}
1007: {A sub: 1006 pred: 101 obj: 1003}
1008: {A sub: 1006 pred: 102 obj: 1000}
1009: {"2008"}
1010: {A sub: 1006 pred: 103 obj: 1009}
1011: {D sub: 1006 pred: 103 obj: 1009}
1012: {"2009"}
1013: {A sub: 1006 pred: 103 obj: 1012}
```



Representing a Graph as a log of Nodes and Edges

Entities

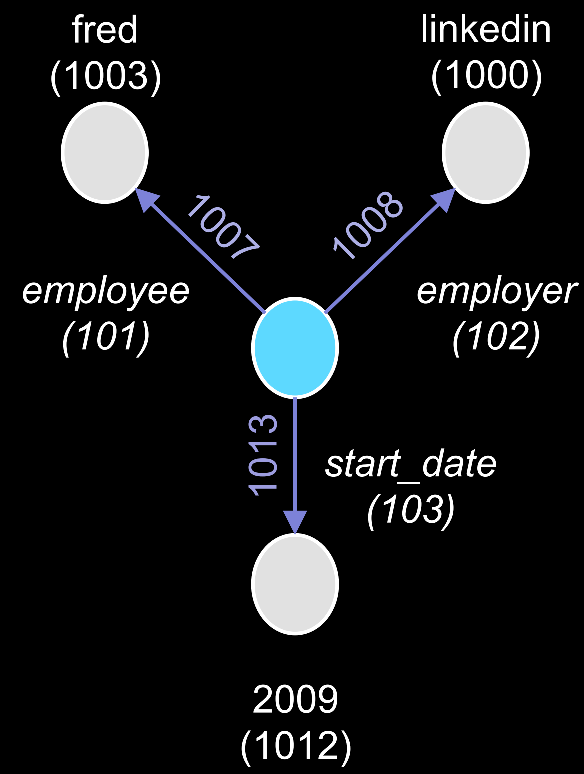
```
100: {“name”}
101: {"employee"}
102: {"employer"}
103: {"start_date"}
...
1000: {"linkedin"}
1001: {"LinkedIn Corporation"}
1002: {A sub: 1000 pred: 100 obj: 1001}
1003: {"fred"}
1004: {"Fred M'Bogo"}
1005: {A sub: 1003 pred: 100 obj: 1004}
1006: {}
1007: {A sub: 1006 pred: 101 obj: 1003}
1008: {A sub: 1006 pred: 102 obj: 1000}
1009: {"2008"}
1010: {A sub: 1006 pred: 103 obj: 1009}
1011: {D sub: 1006 pred: 103 obj: 1009}
1012: {"2009"}
1013: {A sub: 1006 pred: 103 obj: 1012}
```



Representing a Graph as a log of Nodes and Edges

Relationships (subject, predicate, object)

```
100: {"name"}
101: {"employee"}
102: {"employer"}
103: {"start_date"}
...
1000: {"linkedin"}
1001: {"LinkedIn Corporation"}
1002: {A sub: 1000 pred: 100 obj: 1001}
1003: {"fred"}
1004: {"Fred M'Bogo"}
1005: {A sub: 1003 pred: 100 obj: 1004}
1006: {}
1007: {A sub: 1006 pred: 101 obj: 1003}
1008: {A sub: 1006 pred: 102 obj: 1000}
1009: {"2008"}
1010: {A sub: 1006 pred: 103 obj: 1009}
1011: {D sub: 1006 pred: 103 obj: 1009}
1012: {"2009"}
1013: {A sub: 1006 pred: 103 obj: 1012}
```



Liquid Inverted Indexing

for O(k) Navigation

```
100: {"name"}
101: {"employee"}
102: {"employer"}
103: {"start_date"}
...
1000: {"linkedin"}
1001: {"LinkedIn Corporation"}
1002: {A sub: 1000 pred: 100 obj: 1001}
1003: {"fred"}
1004: {val: "Fred M'Bogo"}
1005: {A sub: 1003 pred: 100 obj: 1004}
1006: {}
1007: {A sub: 1006 pred: 101 obj: 1003}
1008: {A sub: 1006 pred: 102 obj: 1000}
1009: {"2008"}
1010: {A sub: 1006 pred: 103 obj: 1009}
1011: {D sub: 1006 pred: 103 obj: 1009}
1012: {"2009"}
1013: {A sub: 1006 pred: 103 obj: 1012}
```

S index

subject	count	predicate/object
1003	1	1005 {p:100 o:1004}
1006	5	1007 {p:101 o:1003}, 1008 {p:102 o:1000}, 1010 {p:103 o:1009}, 1011 {p:103 o:1009}, 1013 {p:103 o:1012}

+

P (predicate), O (object) indices

as hash tables in memory

Liquid Inverted Indexing

for O(k) Navigation

```
100: {“name”}
101: {"employee"}
102: {"employer"}
103: {"start_date"}
...
1000: {"linkedin"}
1001: {"LinkedIn Corporation"}
1002: {A sub: 1000 pred: 100 obj: 1001}
1003: {"fred"}
1004: {val: "Fred M'Bogo"}
1005: {A sub: 1003 pred: 100 obj: 1004}
1006: {}
1007: {A sub: 1006 pred: 101 obj: 1003}
1008: {A sub: 1006 pred: 102 obj: 1000}
1009: {"2008"}
1010: {A sub: 1006 pred: 103 obj: 1009}
1011: {D sub: 1006 pred: 103 obj: 1009}
1012: {"2009"}
1013: {A sub: 1006 pred: 103 obj: 1012}
```

SP index

subject/ predicate	count	object
{s:1003 p:100}	1	1005 {o:1004}
{s:1006 p:101}	1	1007 {o:1003}
{s:1006 p:102}	1	1008 {o:1000}
{s:1006 p:103}	3	1010 {o:1009}, 1011 {o:1009}, 1013 {o:1012}

+ OP and SPO indices

Prologin (Datalog) Query Language

```
Edge("e1", "employee", "fred").
Edge("e1", "employer", "linkedin").
Edge("e1", "start_date", "2009").


Employment(p, c, d) :-
  Edge(e, "employee", p),
  Edge(e, "employer", c),
  Edge(e, "start_date", d).

Employment("fred", "linkedin", "2009").


Employment("fred", "linkedin", _)?
```

```
Employment(_, "linkedin", "2009")?
Employment(_, _, "2009")?
Employment(_, "linkedin", _)?


Like(a, b) :-
  Edge(a, "like", b).

Like("e1", "a1").

EmployeeLiked(c, l) :-
  Employment(e, c, _),
  Like(e, l).

EmployeeLiked("linkedin", _)?
EmployeeLiked(_, "a1")?
EmployeeLiked("linkedin", "a1")?
```

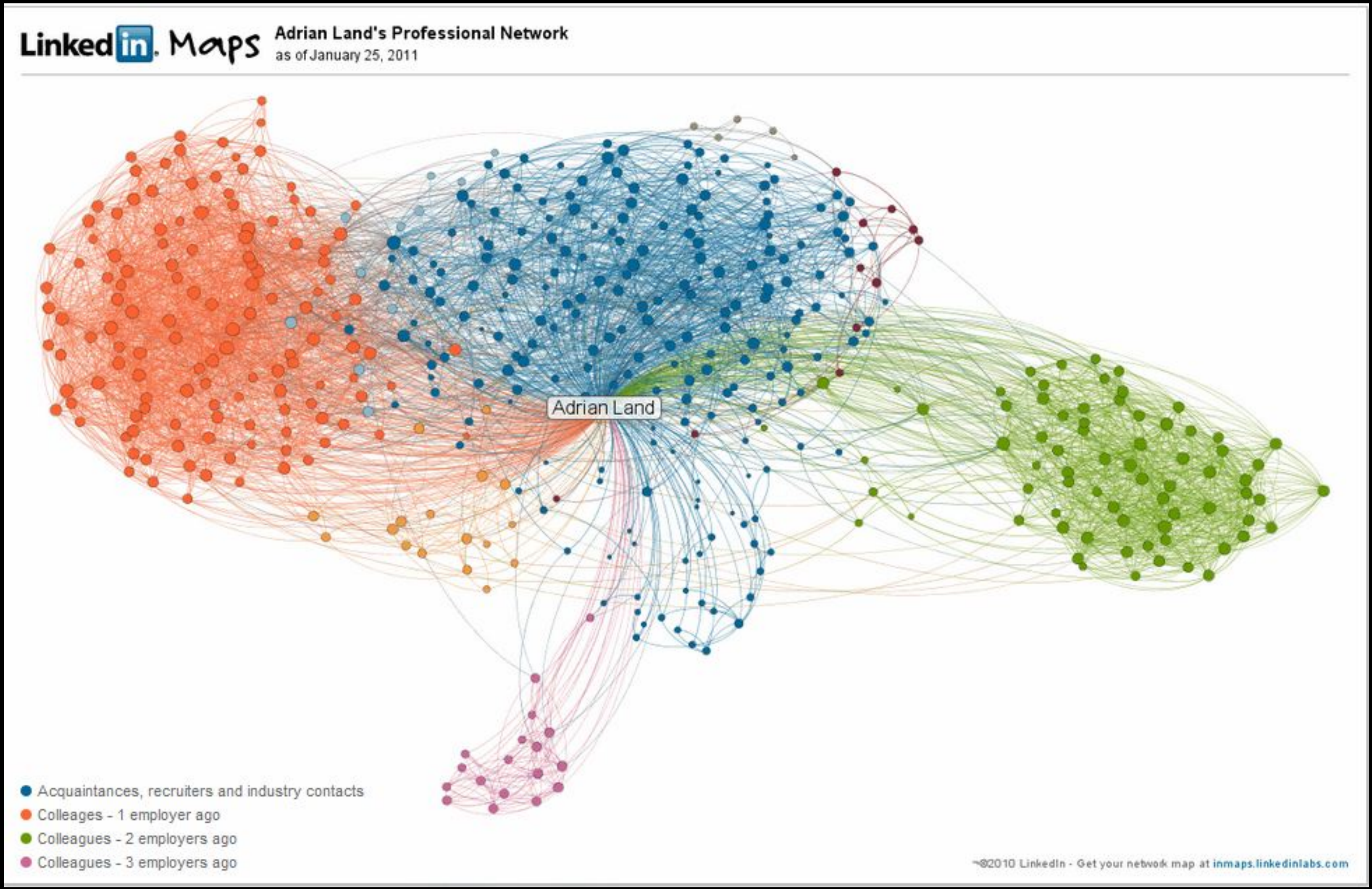
Datalog as core
option to add other bindings
such as SQL

Query Evaluation

Dynamic cost-based

Skew Aware

Community Sharding



Community Sharding
(initial thoughts)

Streaming Graph Partitioning for Large Distributed Graphs

"Linear Deterministic Greedy" is competitive with METIS (current best offline algorithm), particularly so when the number of partitions is small, < 100

35% increase in PageRank performance relative to random

Liquid advantages:

1. We're not actually streaming
2. Special handling (random) for large fan-outs
3. Small number of partitions

Distributed Query Evaluation

(initial thoughts)

Each node is a Liquid instance

Federated query evaluation

- optimize for single node win
- if lose:
- build small database
- accumulate partial results from shards, D round trips
- issue final query against small database

Search at Linkedin

Already covered in SIRIP yesterday

- Multiple verticals – people, jobs, companies, groups
- Query intent - small set of likely intents, much easier to guess
- Architecture - Conventional doc-sharded inverted index
- Graph influence on retrieval
  - Added 1st degree to people index
  - 2nd degree comes from Graph

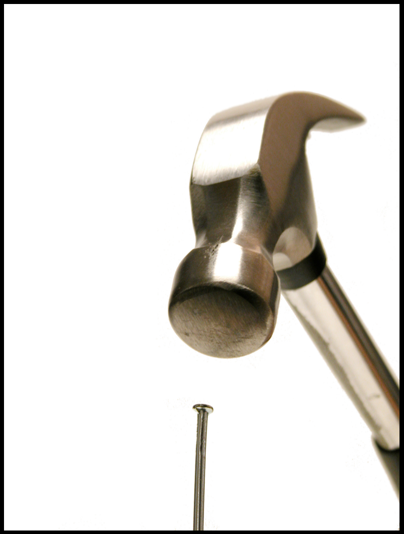
Should Graph and Search converge?

- Graph provides full and precise results, focus on traditional database query optimization (joins, multiple index structures)
- Search provides best effort results focus on relevance, traditional IR techniques

- A single Graph index for multiple domains (members, companies, jobs, schools, skills)
- A Search index per domain

- Graph N-way relations are 1st class
- Search 2-way relations are 1st class
- How about pre-materializing N-way relations as 2-way relations?
  Which combinations of 2 dimensions to materialize?
  Lists as payload, e.g. member endorsed member => list of skills

Likely Direction



Leverage best of what each system does best

Create query language and evaluator that leverages best of both


ANY
QUESTIONS
?