THE STIG PROJECT

NoSQL R&D at Tagged

presented by

Jason Lucas
Architect of Scalable Infrastructure

TAGGED™

The Stig Project

GOALS

Facilitate the Developer

- Decrease the burden
  - Provide a single path to data.
  - Create a uniform representation available to multiple application languages.
  - Reduce the need for “defensive programming”
- Enforce consistency
  - Re-introduce atomic transactions
  - Control assumptions with assertions
- Promote correctness
  - Provide a more robust data representation
  - Support unit testing
- Offer power in simplicity
  - Offer a robust expression language
  - Describe effects rather than details of distribution
- Above all else:
  - “I want to feel like I'm doing a good job.”

Scale Like Crazy

- Use a distributed architecture.
  - Shard data over multiple machines.
  - Use commodity hardware.
  - Scale as linearly as possible.
  - Use replicas to speed average access.
- Move queries to data.
  - Decompose queries by separating areas of concern.
  - Farm sub-queries to the shards which hold the relevant data.
  - Use comprehensions instead of realizations wherever possible.
- Build for the web.
  - Provide durable sessions.
  - Allow clients to disconnect and reconnect at will.
  - Continue running in the background.
- Increase concurrency.
  - Break large objects down into smaller ones.
  - Escrow deltas around fields which are partitioned or contentious.
  - Use assertions instead of locks to permit interleaving of operations.

Without Driving Ops Crazy

- Be highly available
  - Replicate storage across multiple machines
  - Shift responsibilities between machines transparently to compensate for machine faults
  - Bring machines back into service transparently when they become available
- Tolerate partitioning
  - Fall back transparently to lower levels of service during a partitioning event
  - Reconcile the database automatically when partitions rejoin
- Simplify maintenance
  - Tolerate unreliable hardware
  - Make software upgrades easy to manage
  - Be flexible with regard to physical topology
  - Make system status, performance, and capacity easy to measure and comprehend
  - Degrade gracefully under load
  - To the greatest degree possible, make the system maintain itself

Exceed Expectations

- Enable previously unthinkable features
  - Don’t include histories in your schemas; the database keeps histories
  - Design apps with real-time, multi-user communications; database sessions are “chatty”
  - Feel free to compute Erdős Numbers or routes to Kevin Bacon
  - Test for the existence of interesting data states in constant time, not log time
- Decrease development cycle time
  - Build working apps on your desktop; the database can be simulated
  - Evolve your schema at will; the database doesn’t make a distinction between data and metadata
  - Use any language you like; the database looks the same from all clients

The Stig Project

CONCEPTS

Representing Graphs

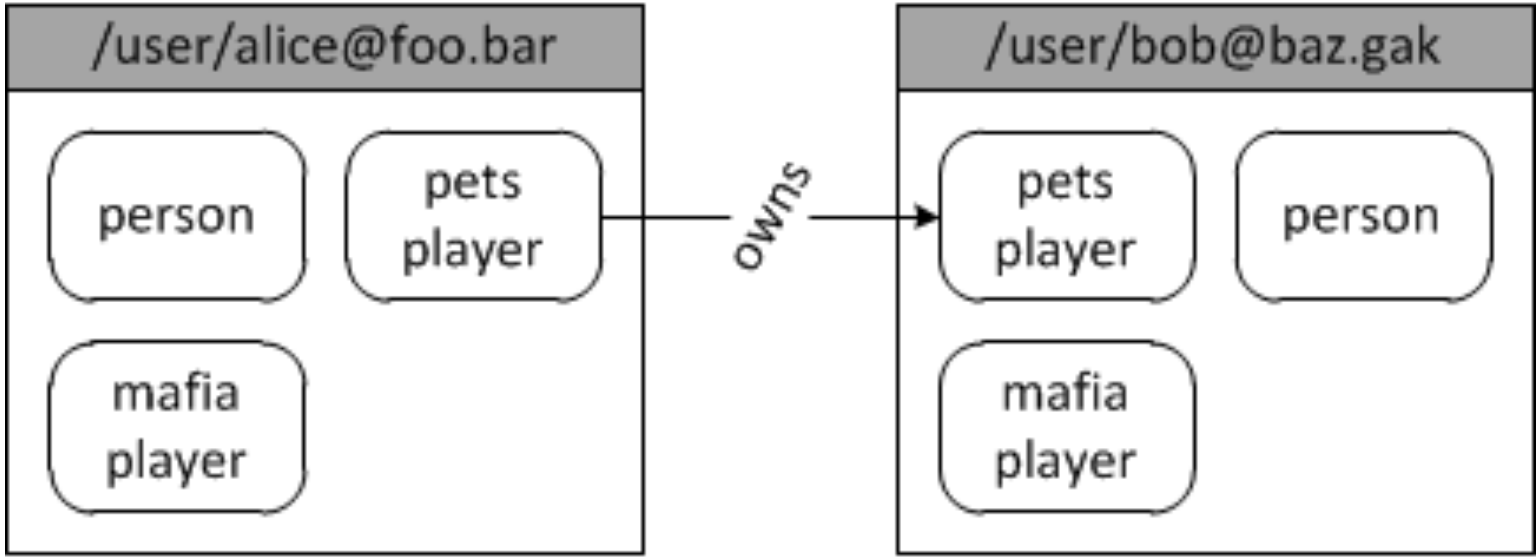
SQL & NoSQL

- Graphs in Tables
  - Walks spread outward in waves
  - Self-joins proliferate
- Graphs Key-Value Stores
  - Generally node-centric
  - Edges are denormalized conjugate sets
  - Non-transactional multi-set is deadly
- Graphs in XML Stores
  - Floating chunk syndrome
  - Worst of both worlds
- Graphs in Doc & Graph Stores
  - Typeless at nodes
  - Interned at nodes

Stig

- Sharded Edge Lists
  - An index is a table and vice-versa
  - An edge exists or it doesn't
- Complex Data in Nodes
  - Objects stay objects
  - Type is enforced
- Nodes Live at Locations
  - A location address is a tuple
  - Tuples work like a directory structure
  - Schemas evolve by adding neighbor nodes
  - Pivots go from one graph to another

Locations, Nodes, & Edges



Deconstructing Commits

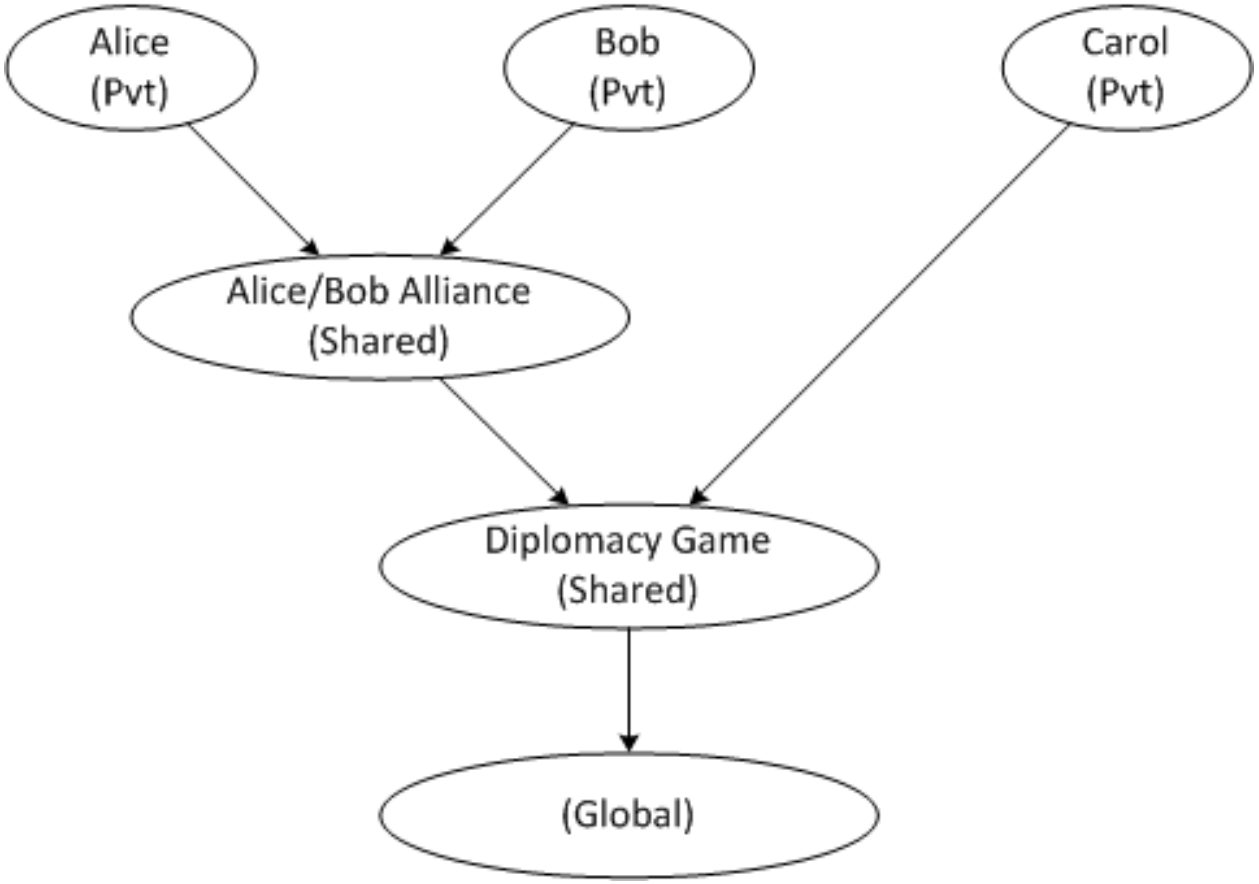
SQL: Commits

- Two States
  - Uncommitted: only me
  - Committed: everybody else
  - One sandbox per connection
- Variable Isolation
  - High isolation limits concurrency
  - Low isolation hard to cope with
- Two Guarantees
  - Written to disk
  - Ephemeral
- Some NoSQL Options
  - No transactional integrity
  - Post-hoc reconciliation

Stig: Points of View

- Private
  - Only me, but I get as many as I want
  - Maybe ephemeral
- Shared
  - Restricted scope, rapid communication
  - Maybe ephemeral
- Global
  - A singleton, same as commit
- Guarantees
  - Self-consistent
  - Replicated in data center
  - Written to disks
  - Replicated to other data centers

Points of View in Diplomacy



Making Time Flow

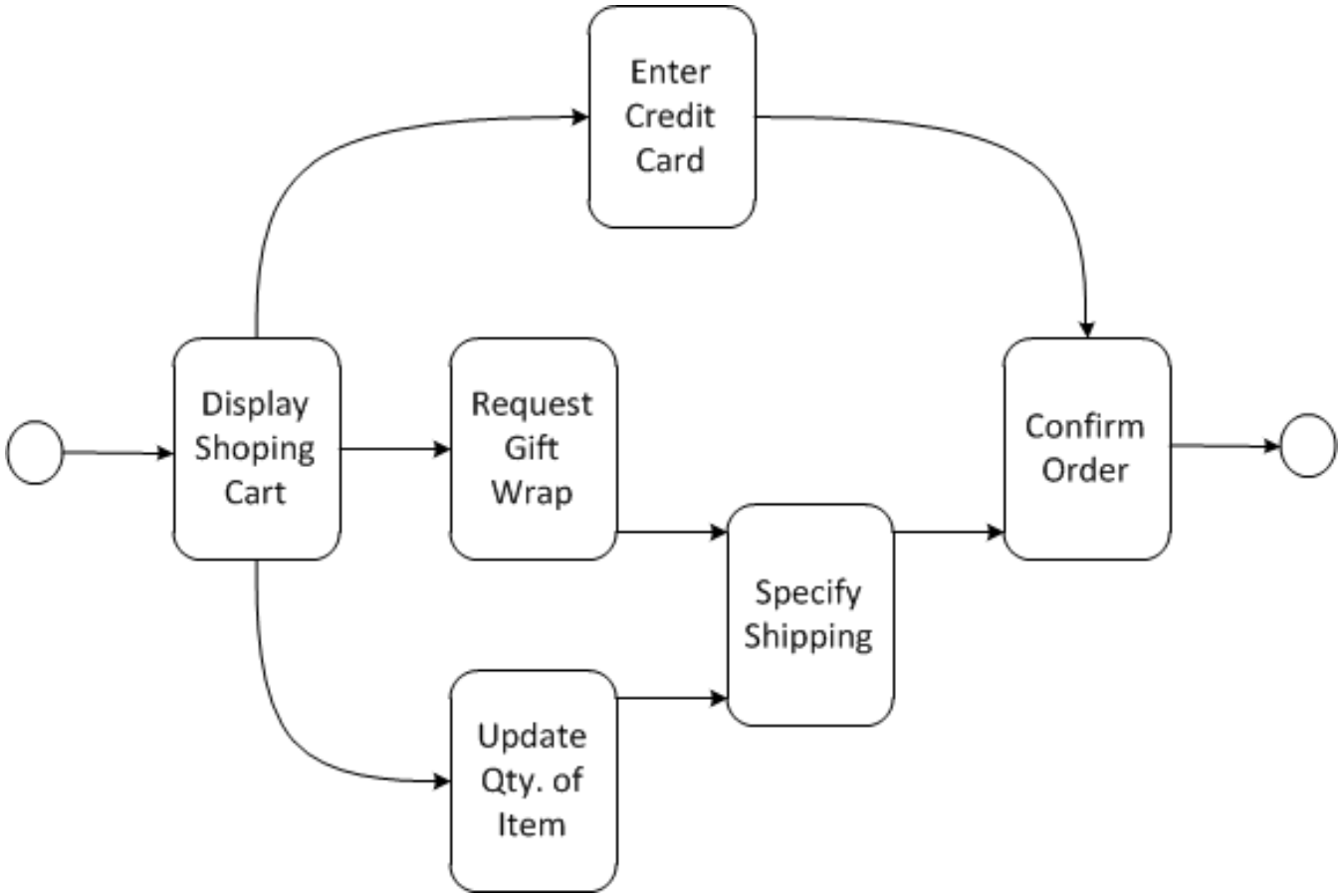
SQL: Clocking & Locking

- Time Flows Naturally
  - System clock is ok
- Execution Time ≈ Query Time
  - A query made after an update will see the results of the update because time flow is linear
  - The order of events is definite
- Locks Enforce Consistency
  - Updates block each other
- MVCC in Lieu of Locks
  - Reads are writes
  - Collisions are rollbacks

Stig: Causality

- Time is Uncertain
  - Distributed machines cannot rely on their system clocks
- Declared Dependencies
  - Each query declares its predecessors, so causality is a graph
  - The order of events is unknowable, but any topological sort of the graph is ok
- Assertions Enforce Consistency
  - MVCC facilitates time travel
  - Query: seek a time in the past at which assertions are true
  - Update: seek a time in the future at which assertions are still true

Checkout Time



Finding Meaning

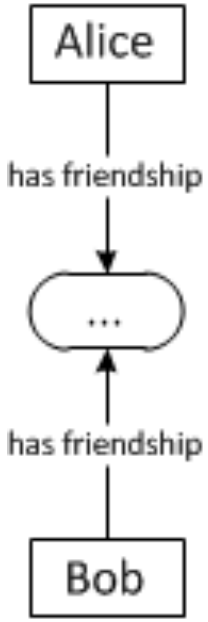
SQL: Projections

- Tables & Views
  - Tables store the base data
  - Views collect data from tables and other views
  - Views often present performance bottlenecks
- Analysis Belongs to Data Definition
  - Adding or changing a view or index is a schema change
  - Programmers must work with DBAs, limiting individual initiative
  - Changes have the potential to degrade the data service as a whole

Stig: Inference

- Asserted & Inferred Edges
  - Asserted edges store the base data
  - Inferred edges collect data from asserted and inferred edges
  - Inference is distributed, on-going, and subject to time-travel
- Analysis Belongs to Program Definition
  - Inference rules aren't "special"
  - Programmers can invent as they like
  - Scope of risk is limited

Inferring Friends & Stalkers




```
<a, 'is friend of' b>
  if <a, 'has friendship', x>
  and <b, 'has friendship', x>
  and a is not b;

<a, 'is stalking' b>
  if <a, 'is friend of', b>
  and a.age >= 18
  and b.age < 18;
```

Query Language

- Language
  - Purely functional, lazily evaluated, and strictly typed
  - Prolog-like notation for describing walks across the graph
- Composability Rules
  - Comprehensions of sequences form the foundation
  - Transformations of sequences (map, reduce, filter, zip, etc.) are the building blocks
- Distributed Evaluation Rocks
  - Queries are broken down and sent to the servers where they need to be
  - Evaluation occurs in parallel
- Compiled & Stored
  - Queries compile down to machine code and get stored in the graph itself
  - Stored programs are subject to on-going analysis
  - Programs can call each other
- Library-Driven
  - Language fundamentals support construction of libraries
  - We can emulate other languages, such as LINQ and Python
- Clients
  - Currently PHP, Java, Python, and C/C++
  - We can also serve HTTP directly

Finding Mutual Friends

```
// Mutual friends function definition:
mutualFriends x y = solve mutualFriend: [<x, 'is friend of',
mutualFriend>; <y, 'is friend of', mutualFriend>];

// Mutual friends query:
mutualFriends 'd74386a3-8529-4d4e-9052-9dd09cf644c6'
'225618ce-0778-4e63-87df-063dd87aa7ea'

// Outputs:
[
 { mutualFriend =
   { location = 'ac457d2d-e6b7-485e-bc6d-8a8df64ff06c';
     user = { name = "John Doe"; emailAddress = "johndoe@example.com" }
   }
 }, { mutualFriend =
   { location = 'c85fa863-e6a5-4366-a04b-4f8aa4c56dd8';
     user = { name = "Jane Doe"; emailAddress = "janedoe@example.com" }
   }
 }, // etc.
]
```

Our Source & Doors are Open

- About our Code
  - Written in C++0x and Haskell, with Python for tools
  - Entirely unit-test driven and designed for easy adoption
- About the Stig Team
  - Four full-time engineers with backgrounds in compilers, databases, distributed systems, and AI
- About Tagged
  - #3 in social networking and growing
  - Located in downtown San Francisco, voted a top-10 place to work
  - Funded on our own revenue, answer only to our users and each other
- Why Open Source?
  - We want to give back
  - We benefit first and most
  - Competitive advantage would be temporary anyway
  - Knowing it's open keeps us on our toes
  - There's more to do than we can do ourselves
  - We attract the kind of people we want to work with
- Contact Me
  - Jason Lucas
    Architect of Scalable Infrastructure
  - jlucas@tagged.com

TAGGED™