HyperGraphDB

Motivation, Architecture and Applications

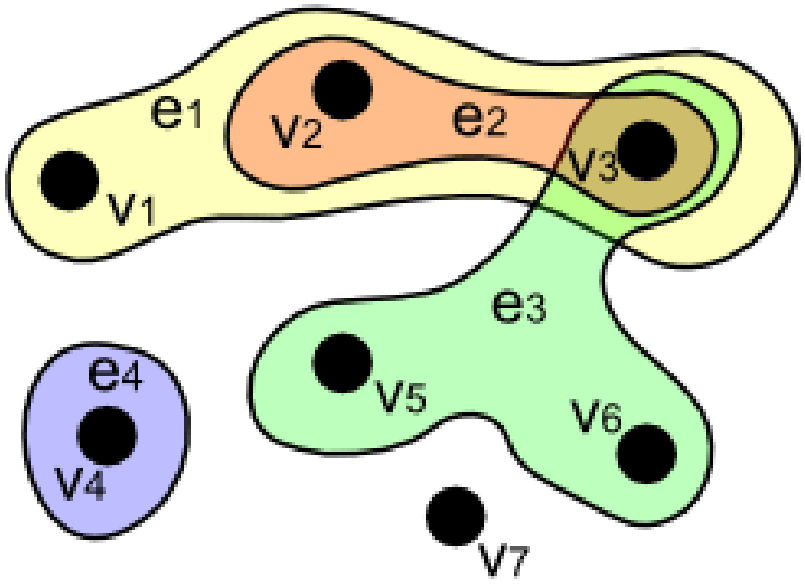
By Borislav Iordanov, Kobrix Software, Inc. www.kobrix.com

NoSQL Live in Boston, March 11, 2010

From Graph to HyperGraphs

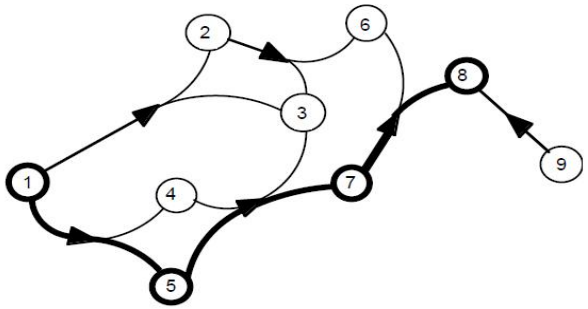
- In a graph $G=(V, E)$, V is a set and E a set of pairs $e=\{v_1, v_2\}$ from V.
- Take e to be any subset $\{v_1,\ldots, v_n\}$ of V and you get a hypergraph.
- Study: set combinatorics, Claude Berge
- Applications: machine learning, database indexing and optimization, SAT

An Undirected Hypergraph

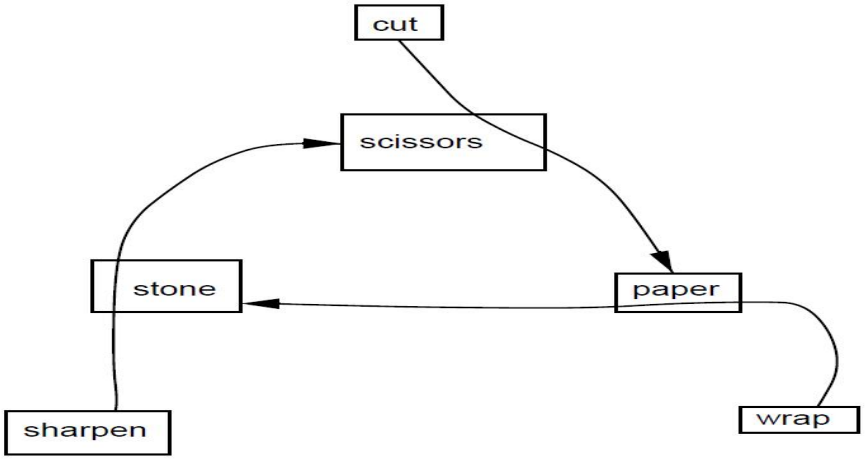


Directed Hypergraphs

- Type 1: partition a hyperedge into a head and a tail



- Type 2: make the hyperedge a tuple



HyperGraphDB Model

- Type 2 directed edges by default
- Support of Type 1 directed edges
- Undirected edges – maybe in the future
- V (nodes) + E (edges) = A (atoms)
  - Care about set theoretic foundation? Look at Peter Aczel *Non-well-founded sets*
- Invented by Dr. Ben Goertzel et al. for an AGI system.

Benefits

- All representations automatically reified
- Higher-order logic
- Recursive buildup of structures:
  - levels of abstraction
  - contextualization
- N-ary relations lead to more compact and natural representations
  - Example: between(New Jersey, New York, Boston)

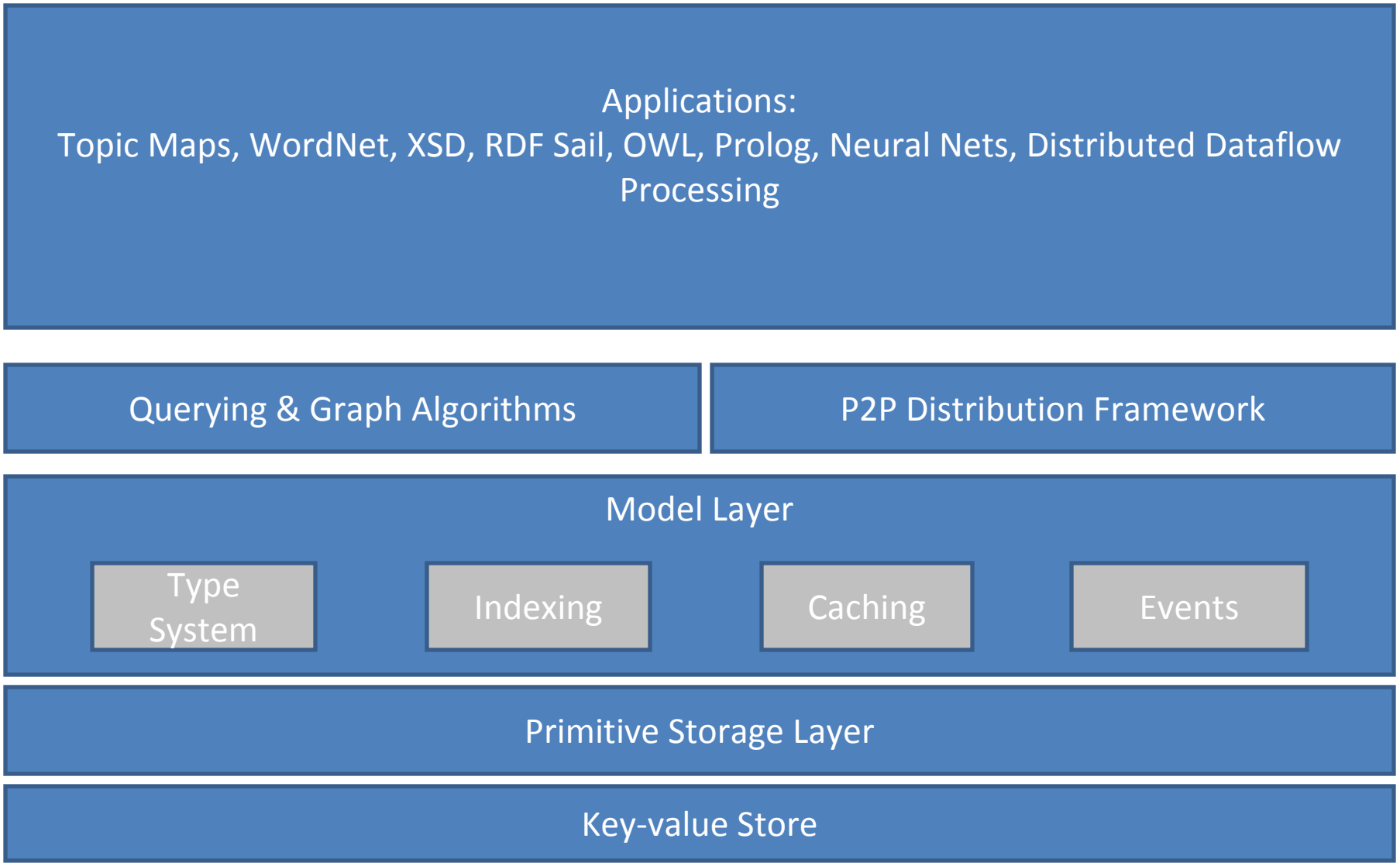
Applications

- Artificial Intelligence:
  - Ben Goertzel et al. – OpenCog AI (http://www.opencog.org)
  - Harold Boley and Directed Recursive Labelnode Hypergraphs, circa 1977
- Freebase is a 4-unform hypergraph
- Relational algebra: HyperGraphDB is essentially a dynamic schema relational DB.
- Computational Biology, see “Hypergraphs and Cellular Networks” by Steffen Klamt et al.

More Applications

- RDF – triple stores are conceptually hypergraphs
- Named RDF – quadruple stores, hypergraphs too
- General (type 2) directed hypergraphs proposed to the RDF community as well.
- Efficient RDF representation: System ∏
- Topic Maps – perfect marriage
- OO Database

Architecture



HyperGraphDB Concepts

- Atom – links 0 or more other atoms, carries strongly typed value
- Value – arbitrary, doesn't participate in linkage
- Type – a special kind of atom that manages value storage and interpretation
- Target Set – the set of atoms that an atom points to.
- Arity – size of the target set
- Incidence Set – the set of atoms pointing to an atom.
- Node – an atom with arity 0 (doesn't point to anything)
- Link – an atom with arity > 0 (points to something)

Storage Architecture

- Two layers – primitive and model layer
- Primitive Layer – a low-level graph of identities and raw data
- Mode layer – a layout for representing typed hypergraph atoms

Primitive Layer

- A graph of identities and raw, byte[] data
- LinkStore

  ID -> [ID, …., ID]
- DataStore

  ID -> byte[]

  Current IDs are type 4 UUID

Model Layer

- Formalizes layout of primitives:

  AtomID -> [TypeID, ValueID, TargetID, ..., TargetID]

  TypeID := AtomID

  TargetID := AtomID

  ValueID -> [ID, …, ID] | byte[]

- A set of predefined indices:

  IncidenceIndex: AtomID -> SortedSet<AtomID>

  TypeIndex: TypeID -> SortedSet<AtomID>

  ValueIndex: ValueID -> SortedSet<AtomID>

Type System: Why types?

- Meaningful interpretation of data.
- Ensure integrity and consistency.
- Customized storage model
- They are a dynamic database schema

Type System : Types Are Atoms

- So an application domain model is directly represented, augmented and programmed against
- So new ones can be dynamically added
- So type constructors (= types of types) will cover any type systems of any programming language
- Reflectivity is good any way you look at it

Typing Bootstrap:Predefined Types

- Stored at database creation time
- … more added any time later
- Handle primitive types such as numbers, strings etc.
- Other standard types: lists, maps etc.
- Type constructors for structured records (e.g. Java beans), strongly type relationships and more.
- Any domain/application specific custom type implementations

Working with HyperGraphDB

- Embedded, Java-based (C++ version planned)
- Store any object as an atom value and create arbitrary n-ary relationships b/w any objects
- The object structure is a graph at runtime, but it can be represented in many ways:
  1. As a serialized blob.
  2. As a primitive value graph.
  3. As an atom graph.

Indexing

- Associate indexes with atom types – then indexing is automatic
- Out-of-box + custom indexing possible
- Out-of-box implementations:
  - object property -> atom
  - target -> atom
  - target -> another target
  - target tuple -> atom
  - multikey: compose any of the above

Querying

- Traversals – API for standard graph traversals. Hyper-traversals by jumping levels
- Constrained Atom Sets (SQL style) – API to retrieve sets of atoms based on constraining conditional expressions.
- (Vaporware) Graph patterns - a new comprehensive query language, coming up, looking for help to do it!

Distribution

- Build on ACL (Agent Communication Language) foundation
- Pluggable presence&communication layer – XMPP (default), JXTA (available) or your own
- Nested workflows framework for agent (i.e. DB instance) conversations
- Primitive conversations such as subgraph transfer available
- Eventually consistent replication at model layer level.

API Highlights

- HGHandle – universal reference to atoms
- HyperGraph – a database instance
  - add, remove, replace, define atoms
  - access to high-level objects: type system, transaction, primitive store etc.
- HGTypeSystem – manage types, mapping b/w HGDB types and Java classes etc.
- HGQuery.hg – create and execute queries

More API Highlights

- HGEventManager – track and/or prevent every database atom operation
- HGTraversal – breadth-first, depth-first, user-defined adjancy lists
- HGIndexManager –arbitrary indexing of atoms
- HGStore – primitive storage layer
- HGIndex – use the key-value store directly
- HyperGraphPeer/ActivityManager – manage P2P activities

API –Interfaces

- HGLink

  getArity(), getTargetAt(i)
- HGAtomType

  make runtime object from storage

  add, remove value from storage

  subsumes(X, Y)
- HGAtomPredicate/HGQueryCondition

  Plug into querying facilities

API – More Interfaces

- HGALGenerator – generate a list of atoms adjacent to a given atom
- HGIndexer - index an atom by an arbitrary key derived somehow from it
- Activity/FSMActivity – implement P2P conversation workflow

Thank you ☺

Next: Seco?