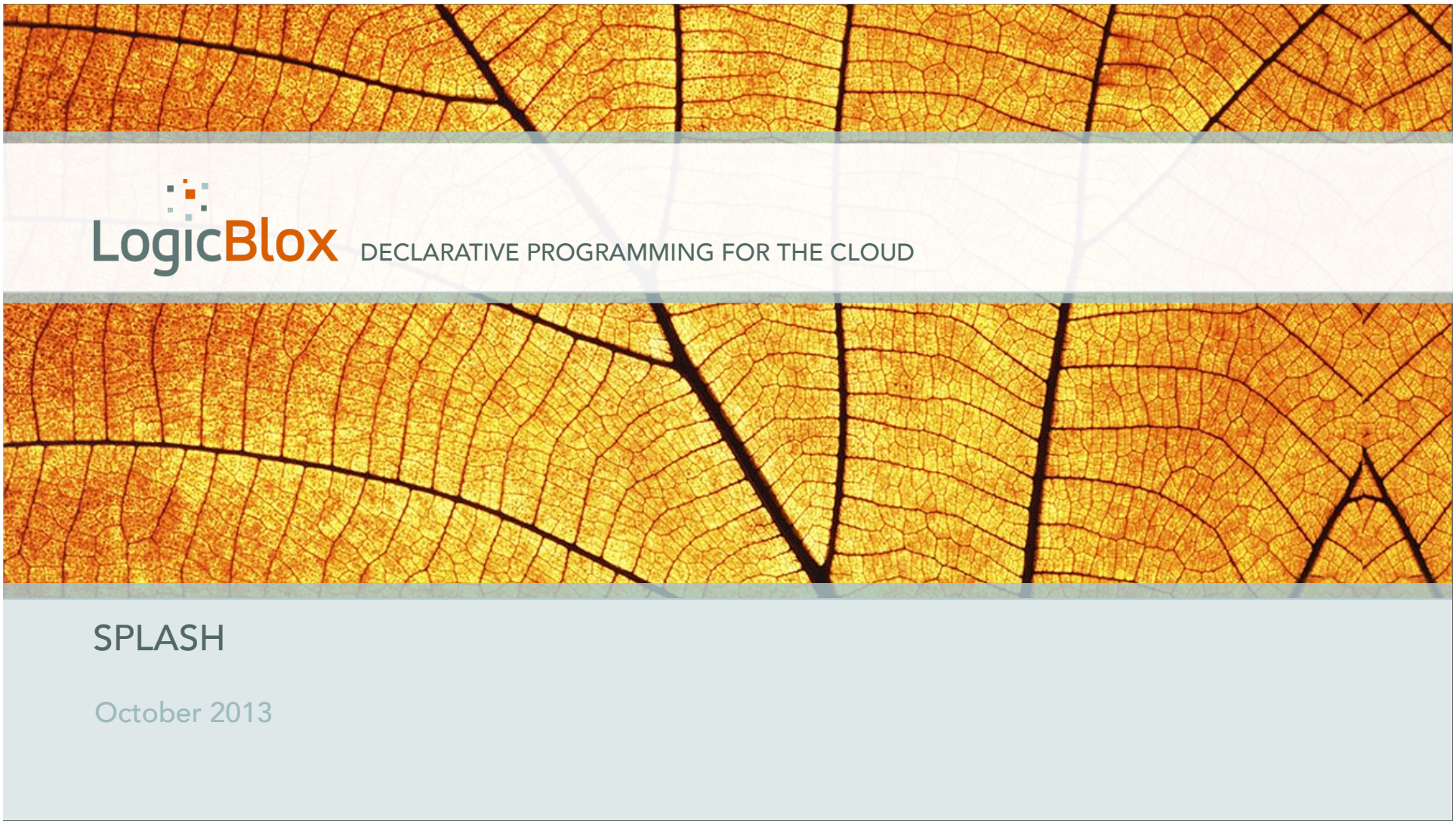


SPLASH

October 2013

WHAT DO YOU GET WHEN YOU CROSS A PL WITH A DB?

WHAT DO YOU GET WHEN YOU CROSS A PL WITH A DB?

(bear with me for a little while and I will tell you)


OUR DOMAIN: BUSINESS

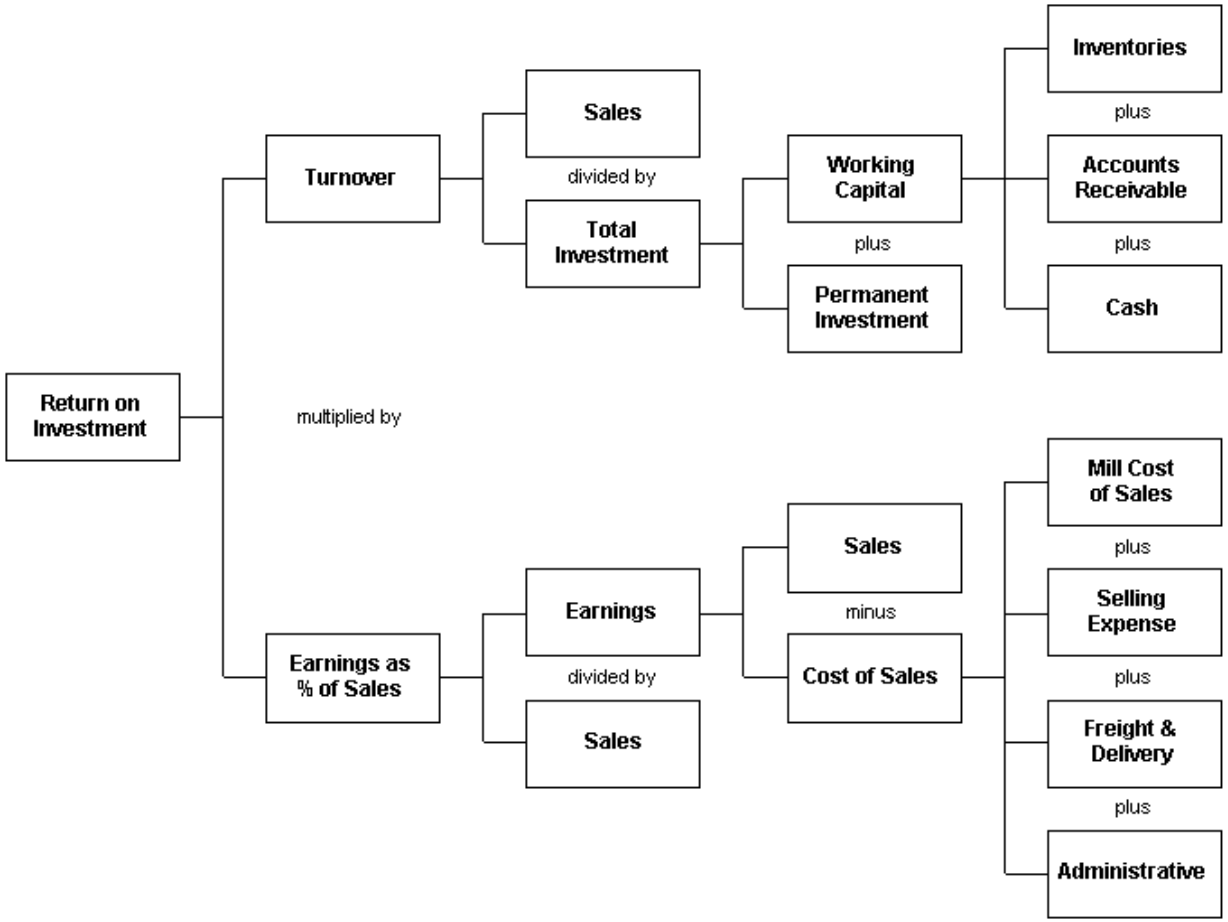
LEST YOU THINK THAT BUSINESS IS BENEATH YOU

"Being good in business is the most fascinating kind of art. Making money is art and working is art and good business is the best art" – Andy Warhol

COMPLEXITY IN BUSINESS

- Businesses are big systems with overwhelming complexity, variability, and detail
- To manage the complexity of business we build simplifying models:
  - Financial models
  - Supply network models
  - Consumer demand models
  - Price elasticity models
  - Risk models
- Typical models are limited/small/coarse/lo-fidelity
  - Spreadsheets: typically 1000's of little spreadsheets
  - Enterprise systems: you lose the forest for the trees
- The most useful models are the ones that can be built and maintained by domain experts who can improve them over time

MODELING A BUSINESS – RETURN ON INVESTMENT



MODELING A BUSINESS – RETURN ON INVESTMENT

Profit = Sales - Cost
Sales= UnitsSold * UnitPrice
UnitsSold = min[UnitDemand, UnitsOnHand]
UnitDemand = StatisticalFunctionOf[InternalFactors, ExternalFactors]
    ExternalFactors: economy, competition, weather, suppliers, clients, etc.
    InternalFactors: price, promotion, labor, product attributes
UnitsOnHand = MathematicalFunctionOf[…]

Constraints must be satisfied

- Investment cannot exceed CashOnHand + CashWeCanBorrow
- Workers can only work 40 hours per week
- Government Regulations
- A given product has to have the same price through all channels
- Can't sell more than what we have

Models ultimately have to apply to P products, S stores+sites, T time periods, E employees, C customers, K competitors, etc.

Hence Big Data and Big Computation from declarative models

BUILDING A STATISTICAL MODEL OF DEMAND

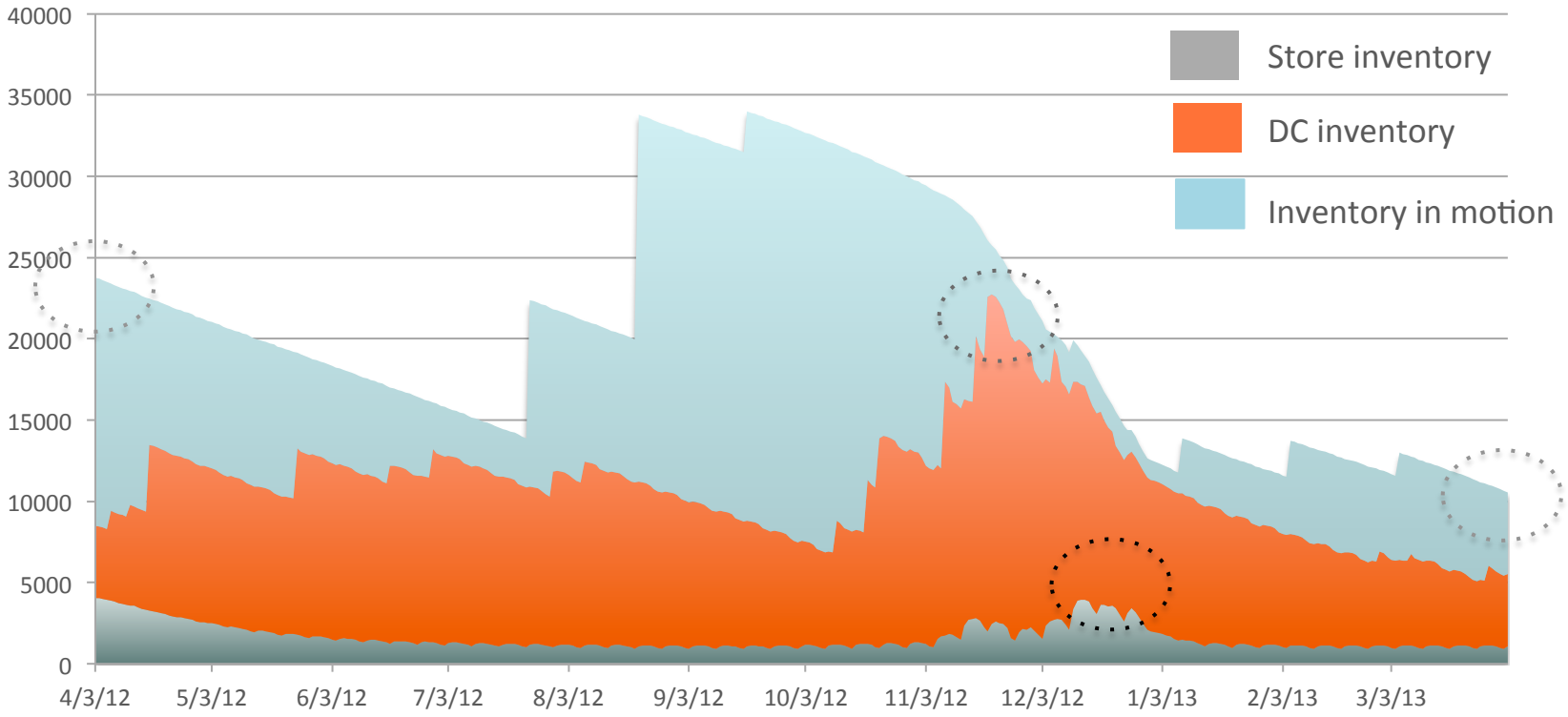
- Size of the problem
  - 234 Billion weekly forecasts (500K skus X 9,000 stores X 52 weeks)
  - Many TB's of data
  - 3,000 computing cores elastically provisioned
- Forecast accuracy
  - Measured 25% to 50% reduction in MAPE
  - The harder the problem the better the improvement
  - Measured reduction of bias in forecasts
- Benefit Quantification
  - $96M to $149M from inventory reductions alone

BUILDING A MATHEMATICAL MODEL OF A SUPPLY CHAIN NETWORK

- Retailer X problem size:
  - 8,000 active SKUs
  - 100 stores
  - Modeling 435 days
  - Number of inventory variables (black arcs) alone is:
    8,000 * 100 * 435 ~ 350 million variables
- Retailer Y problem size:
  - 40,000 SKUs
  - 1,500 stores
  - Number of inventory variables alone is:
    40,000 * 1,500 * 435 ~ 26 billion variables

- 1,000 lines for the entire model
  - Actual constraints and equations < 400

BUILDING A MATHEMATICAL MODEL OF A SUPPLY CHAIN NETWORK



POP CULTURE IS CATCHING ON!!!

July 14, 2013

The Meh List

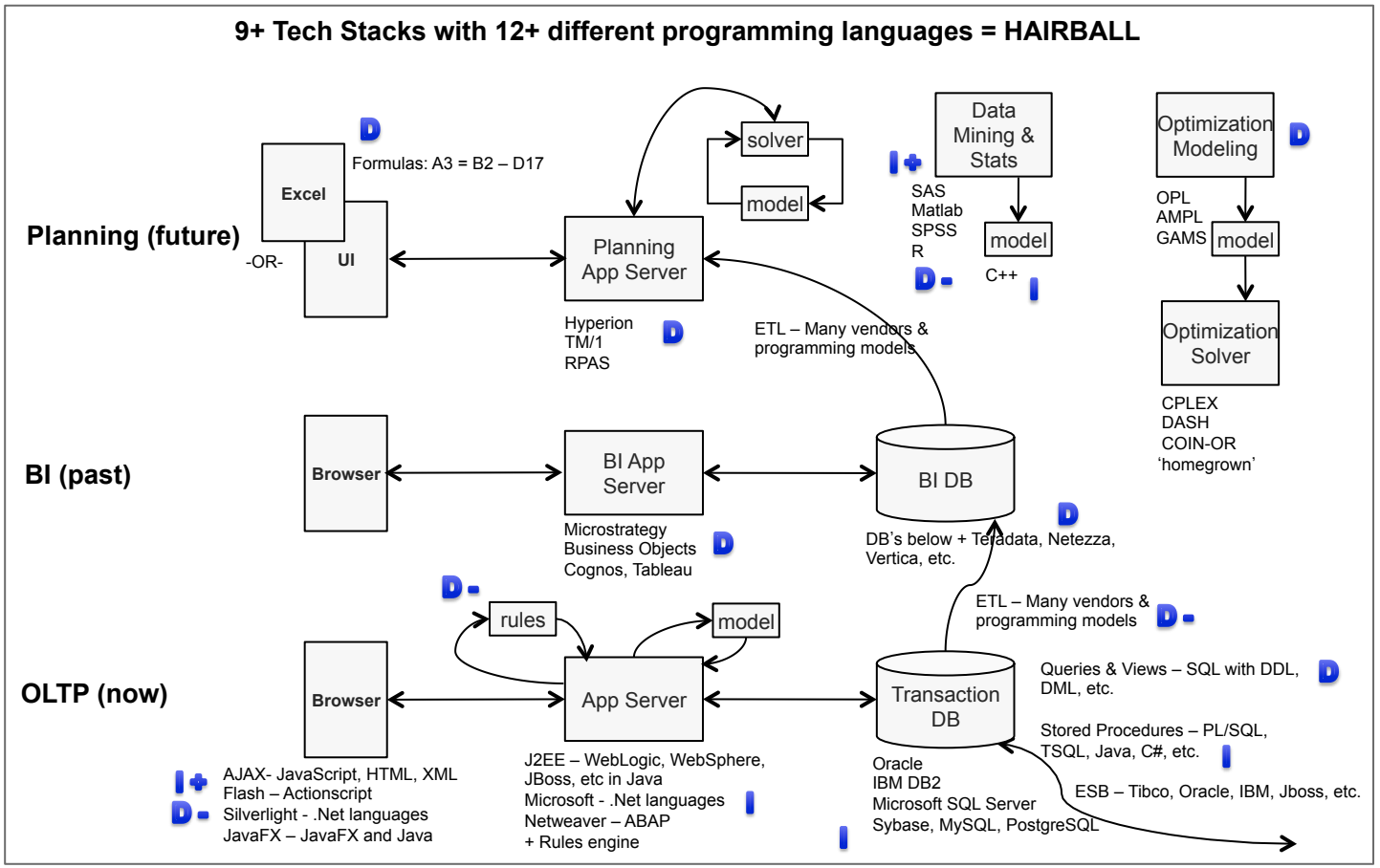
By CHARLES DUHIGG

1. S.P.F. 4
2. Straight wedding announcements
3. Medium-size data
4. Generations L through R
5. Three-quarter marathons
6. Getting your grill on
7. Gold
8. The Man of Aluminum
9. Ecuadorean trade agreements
10. Imperative programming languages
11. Interactive Maps*
12. Using social media to get other people to do your job

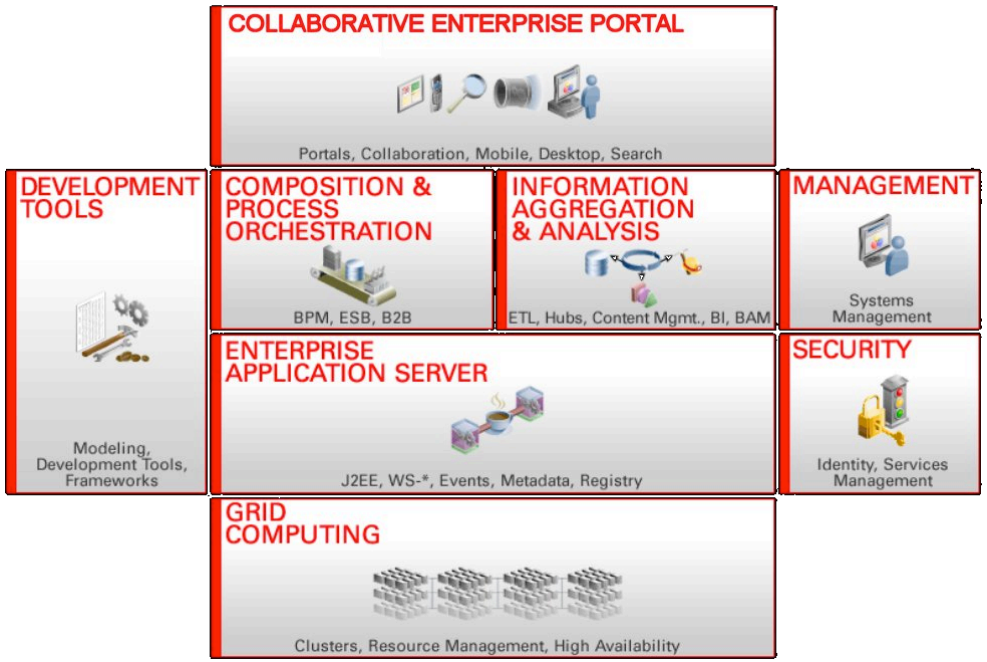
Additional reporting by complete strangers on Facebook.
**Submitted by @ErikFlannigan on Twitter via #mehlist*

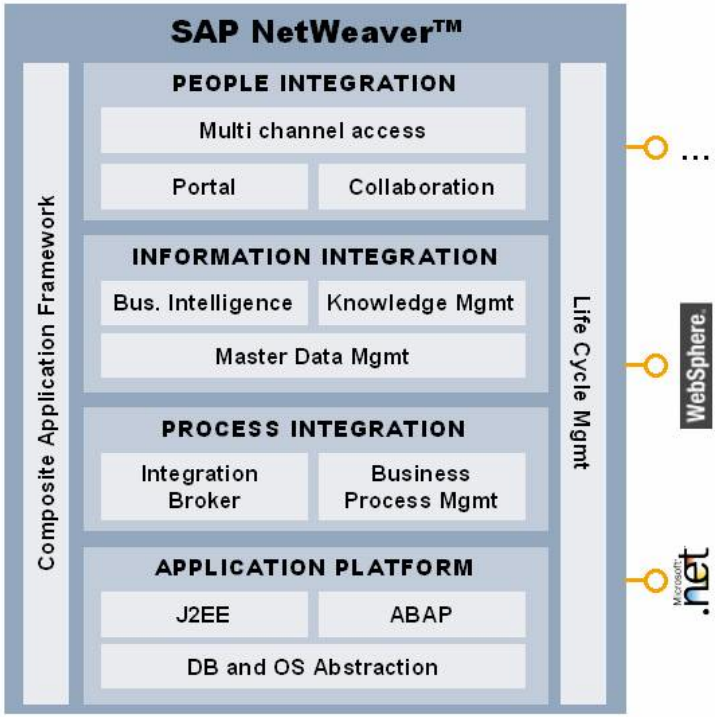

ENTERPRISE IT 101

BEST PRACTICE ENTERPRISE TECH STACK

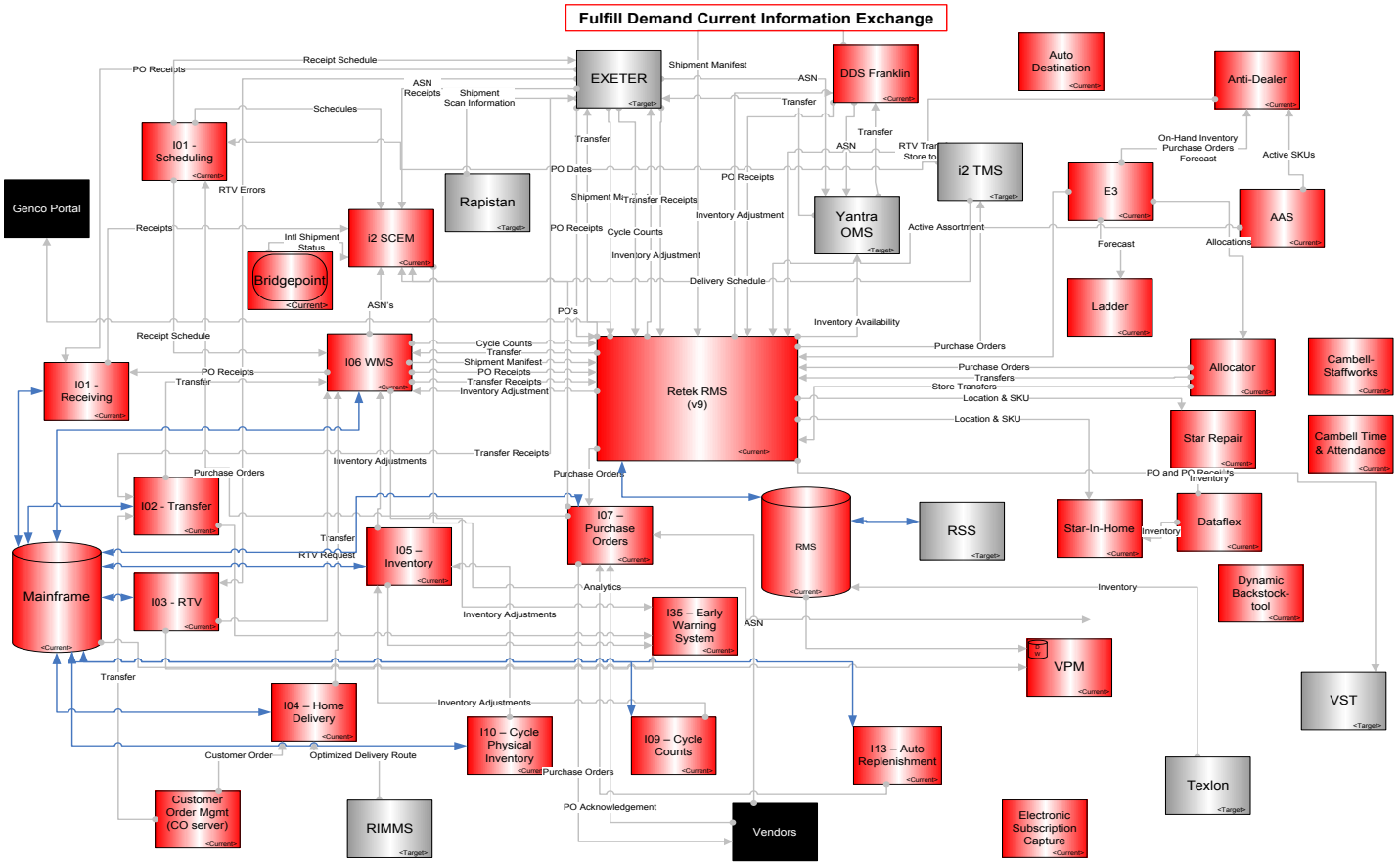


ORACLE AND SAP





IT LANDSCAPE – A REAL ENTRPRISE (2% of footprint)




ENTERPRISE IT 201
Enter the Cloud

SUPERCOMPUTING FOR THE REST OF US

Top500 List - November 2011

R_{max} and R_{peak} values are in TFlops. For more details about other fields, check the TOP500 description.

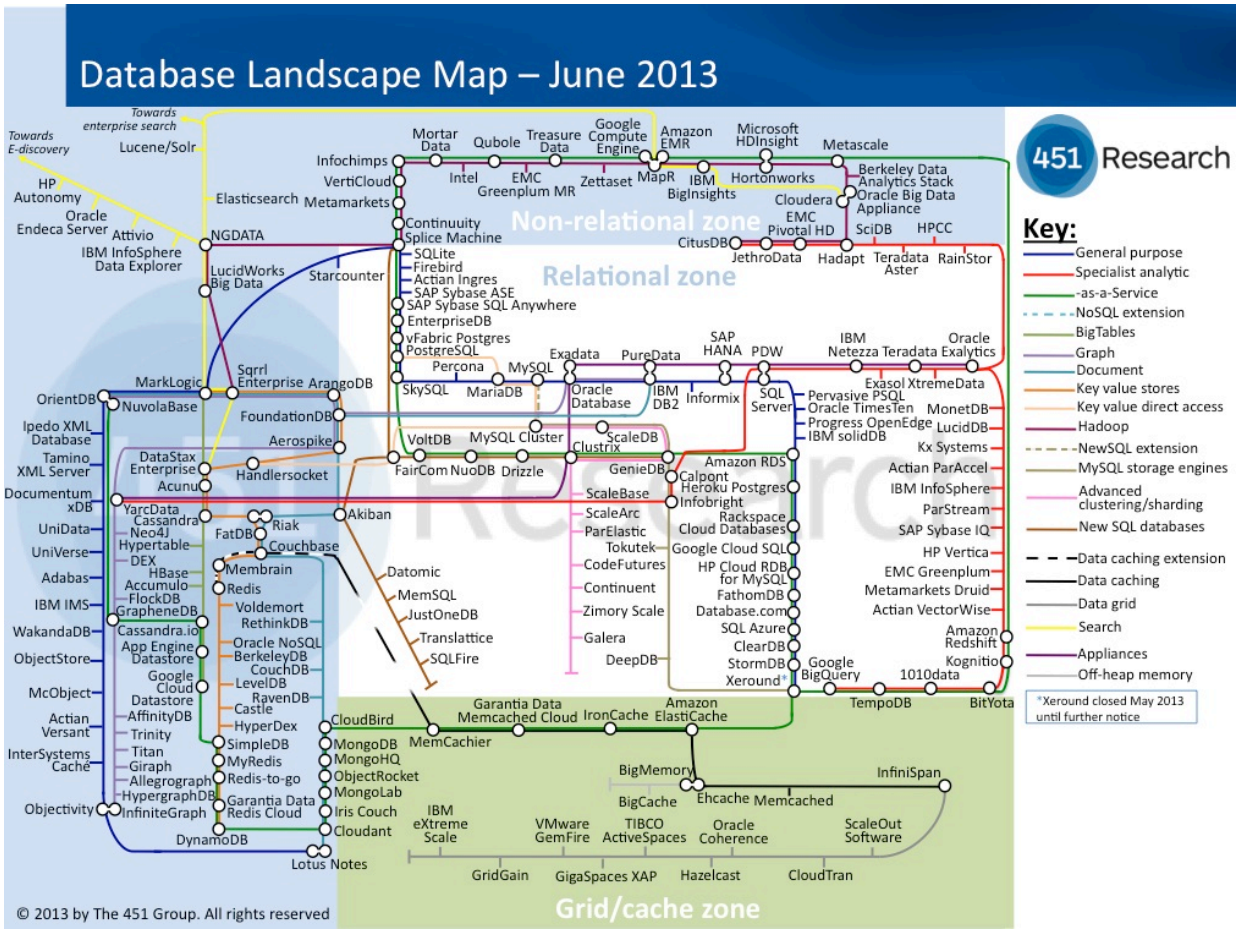
Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
2	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT	186368	2566.0	4701.0	4040
3	DOE/SC/Oak Ridge National Laboratory United States	**Jaguar** - Cray XT5-HE Opteron 6-core 2.6 GHz Cray Inc.	224162	1759.0	2331.0	6950
4	National Supercomputi in Shenzhen (NSCS) China					
5	GSIC Center, Tokyo Ins Technology Japan					
6	DOE/NNSA/LANL/SNL United States					

41	Calcul Canada/Calcul Québec/Université de Sherbrooke Canada	Rackable C2112-4G3 Cluster, Opteron 12 Core 2.10 GHz, Infiniband QDR SGI	37728	240.3	316.9	
42	Amazon Web Services United States	**Amazon EC2 Cluster Compute Instances** - Amazon EC2 Cluster, Xeon 8C 2.60GHz, 10G Ethernet Self-made	17024	240.1	354.1	
43	Grand Equipement National de Calcul Intensif - Centre Informatique National de l'Enseignement Suprieur (GENCI-CINES) France	**Jade** - SGI Altix ICE 8200EX, Xeon E5472 3.0/X5560 2.8 GHz SGI	23040	237.8	267.9	1064
44	KTH - Royal Institute of Technology Sweden	**Lindgren** - Cray XE6, Opteron 12 Core 2.10 GHz, Custom Cray Inc.	36384	237.2	305.6	

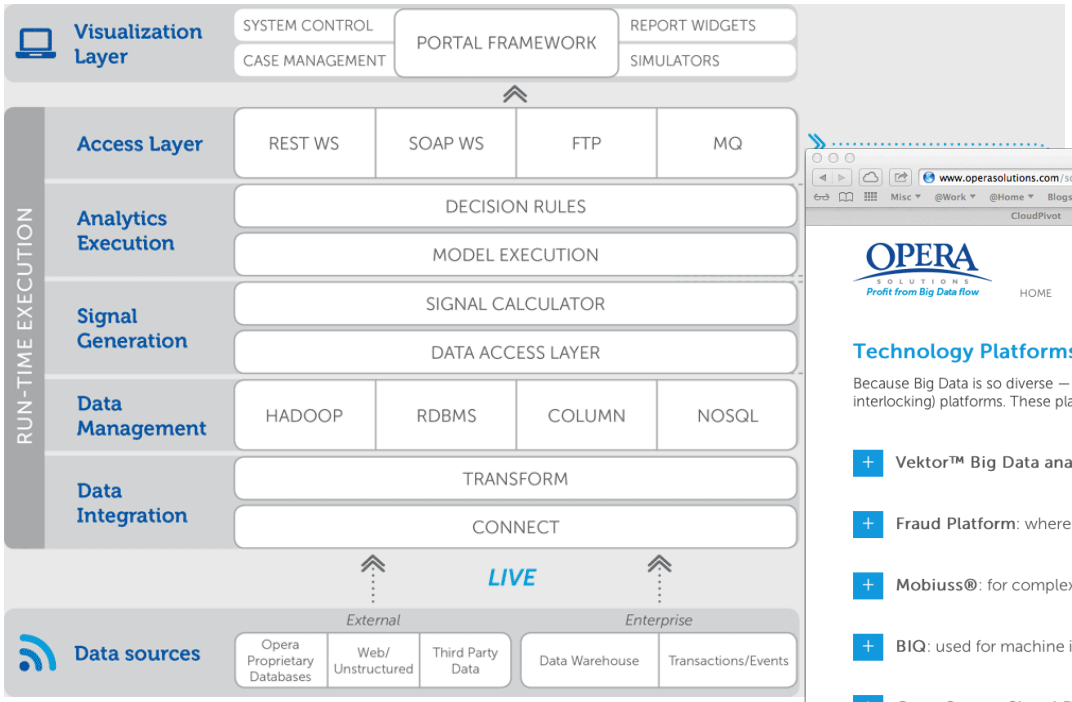
THE CLOUD HAS FREED US FROM TYRANNY

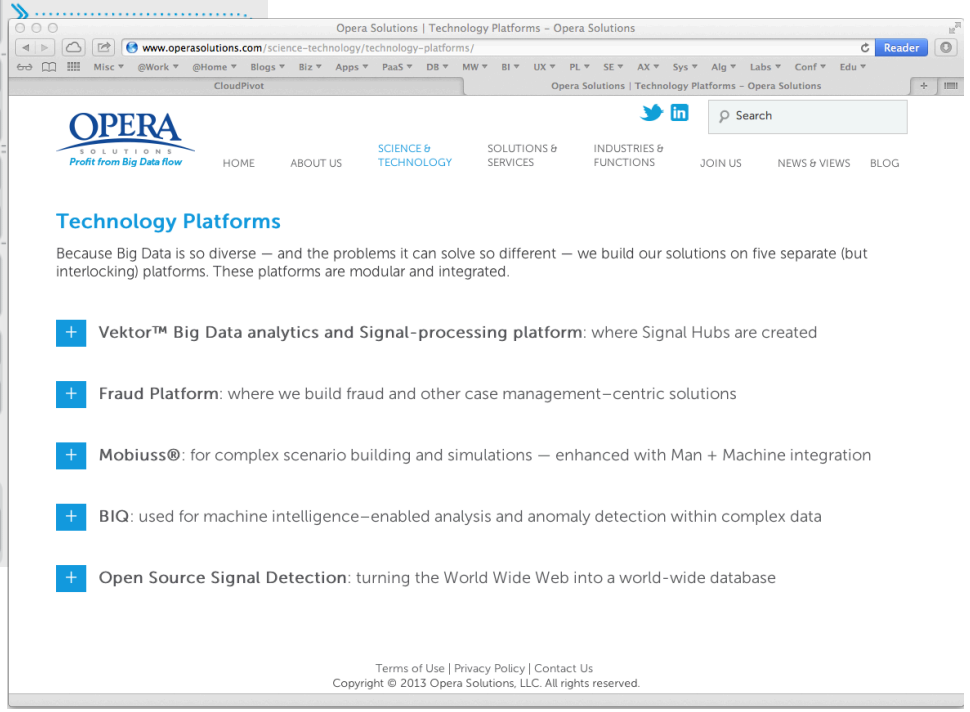
- Cloud has been a catalyst for breathtaking change and innovation in infrastructure
- Infrastructure evolving at very rapid rate
- The on-premise stack is no longer sacrosanct
- Several communities being formed
  - NoSQL – scalability at the expense of queries and ACID support
  - In-memory – speed in exchange for full mem hierarchy support
  - Column Store – mem hierarchy optimization, bad for TRX's
  - NewSQL – elasticity for transactional workloads
- Some are trying to create a new category of hybrid database:
  - OLTP+ OLAP = OLTAP (but still not other interesting categories)
  - Use in-memory "feature" to help achieve performance goals – but expensive.
  - Combine column and row store technology under the covers – hairball on the inside
  - Programming model still a hairball – e.g. HANA: SQLscript, SQL, MDX, RDL, R, ABAP, etc.

AND REPLACED IT WITH THE TYRANNY OF CHOICE



HAIRBALL 2.0


Visualization Layer
SYSTEM CONTROL
CASE MANAGEMENT
PORTAL FRAMEWORK
REPORT WIDGETS
SIMULATORS
RUN-TIME EXECUTION
Access Layer
REST WS
SOAP WS
FTP
MQ
Analytics Execution
DECISION RULES
MODEL EXECUTION
Signal Generation
SIGNAL CALCULATOR
DATA ACCESS LAYER
Data Management
HADOOP
RDBMS
COLUMN
NOSQL
Data Integration
TRANSFORM
CONNECT
LIVE
External
Enterprise
Data sources
Opera Proprietary Databases
Web/ Unstructured
Third Party Data
Data Warehouse
Transactions/Events


Opera Solutions | Technology Platforms – Opera Solutions
www.operasolutions.com/science-technology/technology-platforms/
Reader
CloudPivot
Opera Solutions | Technology Platforms – Opera Solutions
OPERA SOLUTIONS
Profit from Big Data flow
Search
HOME
ABOUT US
SCIENCE & TECHNOLOGY
SOLUTIONS & SERVICES
INDUSTRIES & FUNCTIONS
JOIN US
NEWS & VIEWS
BLOG
Technology Platforms
Because Big Data is so diverse — and the problems it can solve so different — we build our solutions on five separate (but interlocking) platforms. These platforms are modular and integrated.
Vektor™ Big Data analytics and Signal-processing platform: where Signal Hubs are created
Fraud Platform: where we build fraud and other case management–centric solutions
Mobiuss®: for complex scenario building and simulations — enhanced with Man + Machine integration
BIQ: used for machine intelligence–enabled analysis and anomaly detection within complex data
Open Source Signal Detection: turning the World Wide Web into a world-wide database
Terms of Use | Privacy Policy | Contact Us
Copyright © 2013 Opera Solutions, LLC. All rights reserved.

THE HAIRBALL IN (IN)ACTION

Contractors See Weeks of Work on Health Site

Published: October 20, 2013 | 718 Comments

According to one specialist, the Web site contains about 500 million lines of software code. By comparison, a large bank's computer system is typically about one-fifth that size.

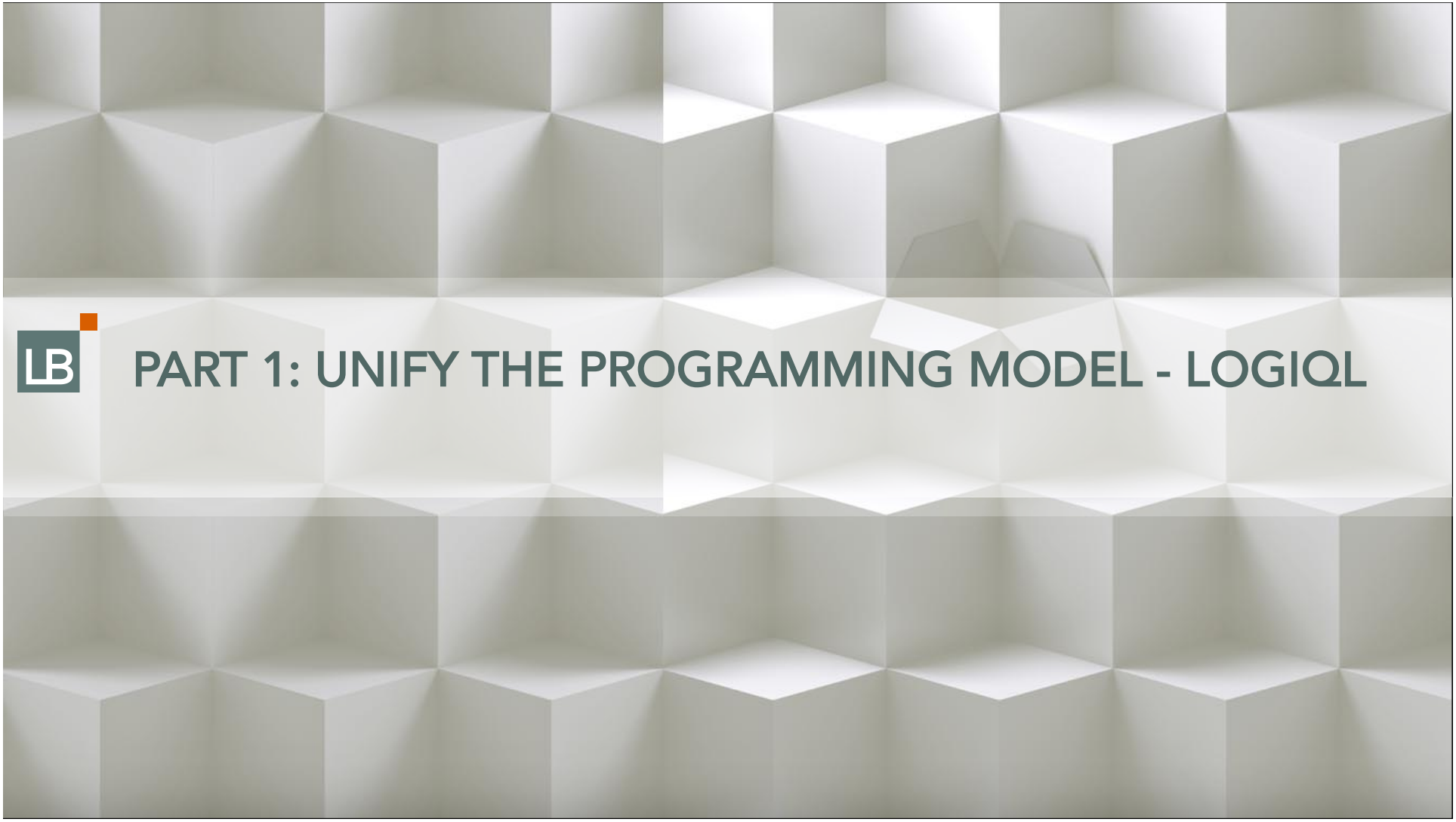
One specialist said that as many as five million lines of software code may need to be rewritten before the Web site runs properly.

PEP TALK

We are here to invent the future

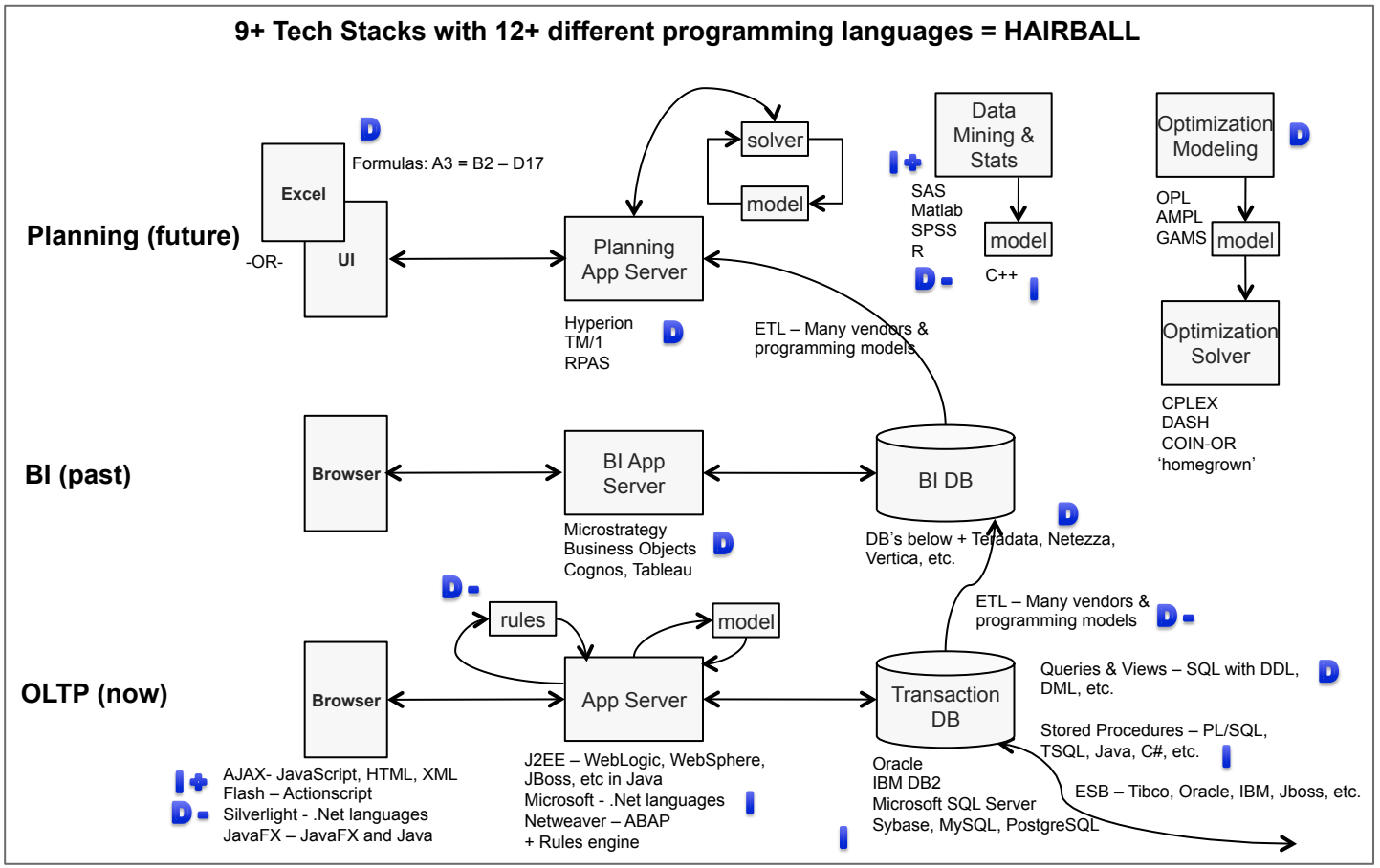
IN 2007 THE SMARTPHONE UNIFIED CONSUMER DEVICES

CAN WE BUILD A "SMART" DATABASE TO DO THE SAME FOR THE HAIBALL?



PART 1: UNIFY THE PROGRAMMING MODEL - LOGIQL

BEST PRACTICE ENTERPRISE TECH STACK



LOGIQL IS BASED ON DATALOG

- Usability
  - Declarative by birth and first order by the grace of god
    - So is Excel – passing functions to functions is too hard for most
    - Order and Cons() considered harmful – first order and first normal form
  - Skins – give people the syntax they like (including diagrams)
  - Automatic incremental computation (support live programming tools)
- Expressivity
  - Guaranteed expressivity with “controllable” power
- Safety
  - Turing completeness considered harmful
  - ACID with efficient full serializability
- Performance
  - Memory hierarchy friendly
  - Parallelizable fragment
  - Semantics independent of a specific evaluator/solver
  - Jujutsu – let’s use the power of the machine against itself. Declarativity makes it possible to use the power of the cloud at compile time to go faster

Tyson Condie @tcondie 6 Mar

And I quote Alan Kay "This [Bloom] is the way people will program computers in the future" #bloom

Expand

Core LogiQL

- Derivation Rules
  - Define how things are calculated
- Integrity Constraints ← Key Language Feature
  - Define the set of values possible
- Event-Condition-Action Rules ← Key Language Feature
  - Define state change

DERIVATION RULES

LogiQL

```
grandparent(x, y) <- parent(x, t), parent(t, y).

ancestor(x, y) <- parent(x, y).
ancestor(x, y) <- ancestor (x, t), ancestor(t, y).

abs[x] = x  <- x >= 0.
abs[x] = -x <- x < 0.

pi[] = 3.0f <- !pp[]=_ .
pp[] = pi[] + sin[pi[]] .
pi[] = pp[] .
```

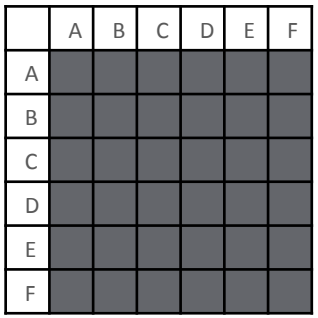
DERIVATION RULES

LogiQL - PageRank

```
d[] = 0.85f.
tolerance[] = 0.01f.

pr[p] = r       <- r = 1.0f /  node_count[], node(p), !pr[p] = _.
pr[p] = r       <- r = (1.0f - d[]) + (d[] * sum[p]),   abs[r - pr[p]] > tolerance[].
pr[p] = pr[p] <- r = (1.0f - d[]) + (d[] * sum[p]), !(abs[r - pr[p]] > tolerance[]).
pr[p] = pr[p] <- !sum[p] = _.

sum[n] = t
    <- agg<<t = total(r)>>
      edge(p, n),
      r = pr[p] / out_count[p].
```

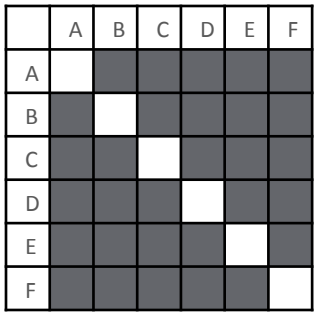
INTEGRITY CONSTRAINTS

```
person(x) ->.                // a,b,c,d,e,f
man(x) -> person(x).         // a,b,c
old(x) -> person(x).         // b,d,f
```



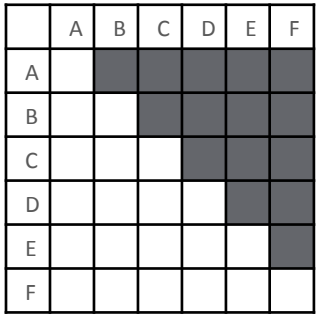

```
parent(x, y) ->
    person(x), person(y)
```

2^36 possibilities: ~64 billion




```
parent2(x, y) ->
    person(x), person(y),
    x != y
```

2^30 possibilities: ~1 billion




```
parent3(x, y) ->
    person(x), person(y),
    y > x
```

2^15 possibilities: ~32 thousand

EVENT-CONDITION-ACTION RULES

LogiQL

```
^my_state[x]=y+1 <- +some_event(x), -some_other_event(x), my_state@prev[x]=y.
```

+, - : operator that detects that an event happened (+ is insertion, - is deletion)
@prev : let us reflect over the state of the world before the event
^ : operator that allows update to the state of the world after the event

Implicit Frame Rule:

```
R(x) <- +R(x).
R(x) <- R@prev(x), !-R(x).
```

Updates are therefore not destructive: “John is a good” yesterday “John is bad” today.

EVENT-CONDITION-ACTION RULES

LogiQL

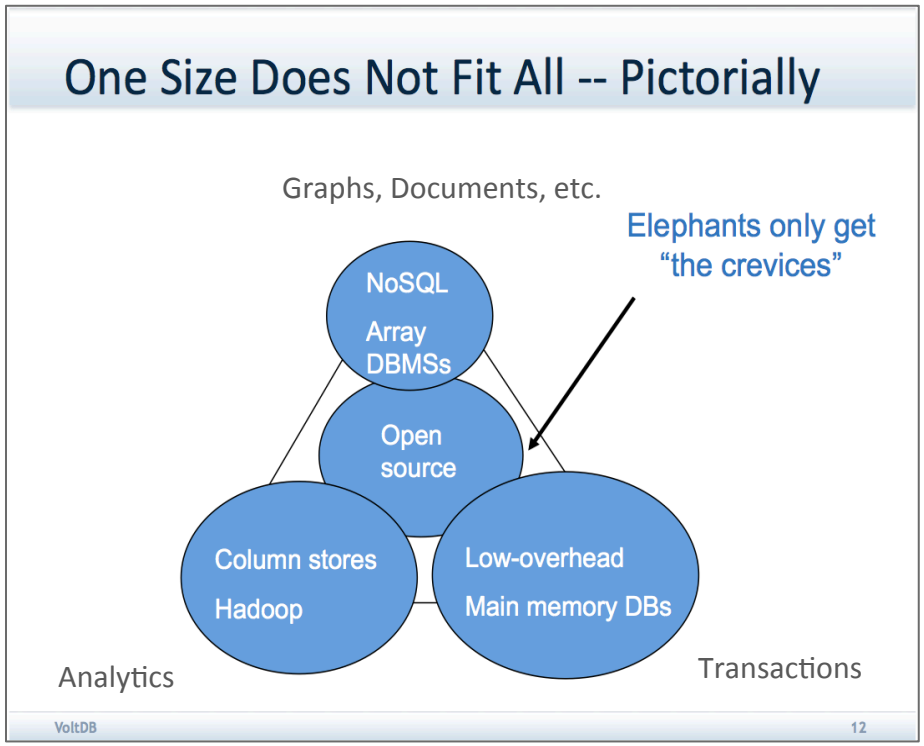
```
checking[i] = a -> identity(i), account(a).
savings[i] = a -> identity(i), account(a).

balance[a] = b -> account(a), decimal(b).

transfer(a1, a2, v) -> account(a1), account(a2), decimal(v).
lang:pulse(`transfer).

^balance[a2] = balance@prev[a2] + v,
^balance[a1] = balance@prev[a1] - v
  <-
  +transfer(a1, a2, v).

checking[i] = a1, savings[i] = a2 -> balance[a1] + balance[a2] > 0.0.
```

SUPPORT FOR PROGRAMMING IN THE LARGE

- Modules - How to organize code, create interfaces
  - LogiQL applications can contain up to 10,000's of rules
  - Really good compared to millions of LOC, but still need organization
- Meta - How to reuse rules
  - LogiQL rules are reusable across data. But they need to be reusable across predicates.
  - Related to type-genericity, high-order functions
- Layers - How to compose large components to build product variants
  - Based on GenVoca work in variability management / product line support
- Types
  - Inferring types from derivation rules and from integrity constraints
- Verification
  - How does the cloud change what we can do at compile time?
- Automatic test data generation


PART 2: UNIFY THE BACK END

THE ARGUMENT FOR SPECIALIZATION



…we are heading toward a world with at least 5 (and probably more) specialized engines and the death of the 'one size fits all' legacy systems.

-The End of an Architectural Era (It's Time for a Complete Rewrite), 2007

SPECIALIZATION JUSTIFIED BY 10X to 100X BETTER PERFORMANCE

#3: One Size Does NOT Fit All

- OSFA is an old technology with hundreds of bags hanging off it
- It breaks 100% of the time when under load
- Load = size or speed or complexity
- Load is increasing at a startling rate
- Purpose-built will exceed by 10x to 100x
- History has not been completely written yet…but let's look at VoltDB as an example

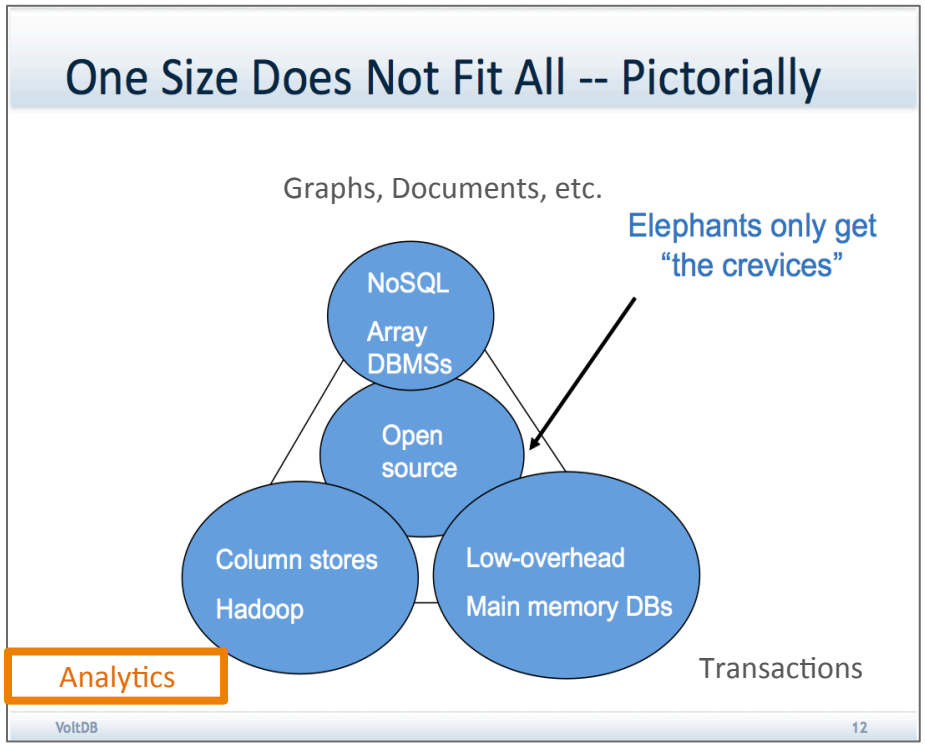
> *"…specialized systems can each be a factor of 50 faster than the single 'one size fits all' system…A factor of 50 is nothing to sneeze at."*
>
> *-My Top 10 Assertions About Data Warehouses, 2010*

WE AGREE!!!

Specialized databases are only worth it if
10X to 100X better


LB
ANALYTICS

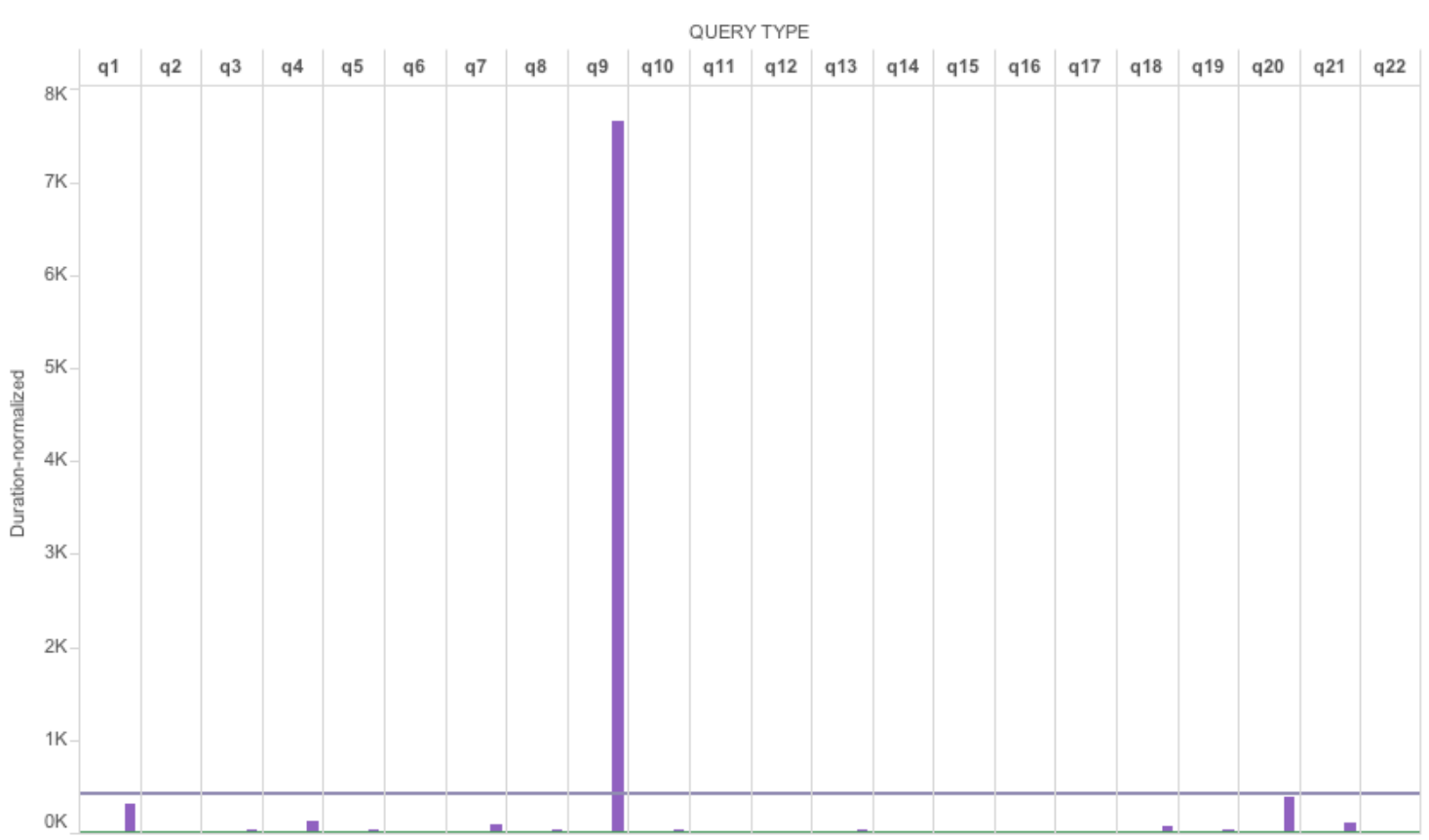
THE ARGUMENT FOR SPECIALIZATION



"…specialized systems can each be a factor of 50 faster than the single "one size fits all" system…A factor of 50 is nothing to sneeze at."

-Stonebraker's Top 10 Assertions About Data Warehouses, 2010

TPC-H: FOR OLDSQL, IN MEMORY COLUMN, AND COLUMN

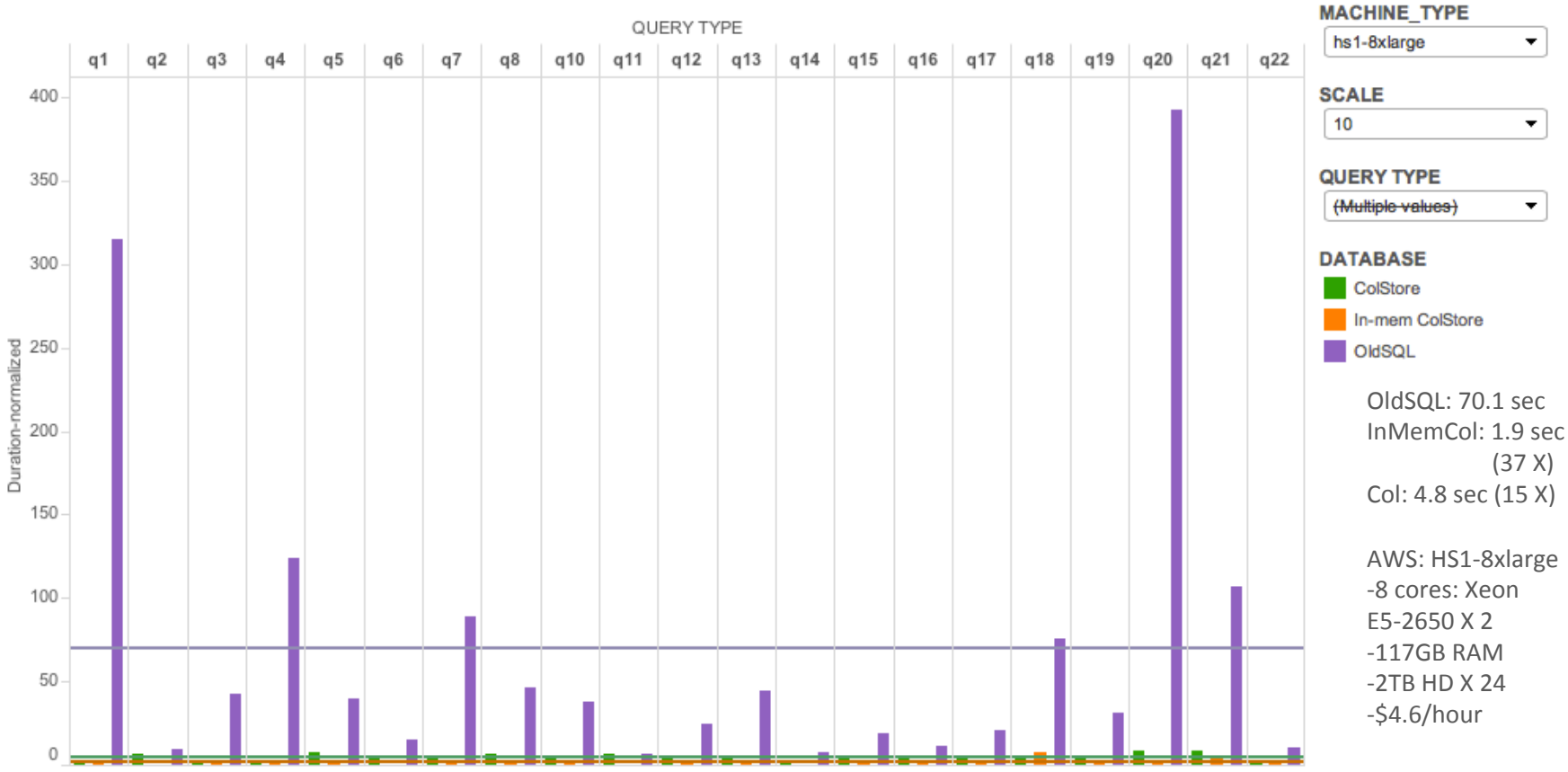


OldSQL: 415.1 sec
InMemCol: 2.2 sec (188 X)
Col: 5.1 sec (81 X)

Turned off nested loop join in the OldSQL DB to get q21 to finish.

OldSQL DB doesn't parallelize queries

TPC-H: OLDSQL, IN MEMORY COLUMN, AND COLUMN (no q9)

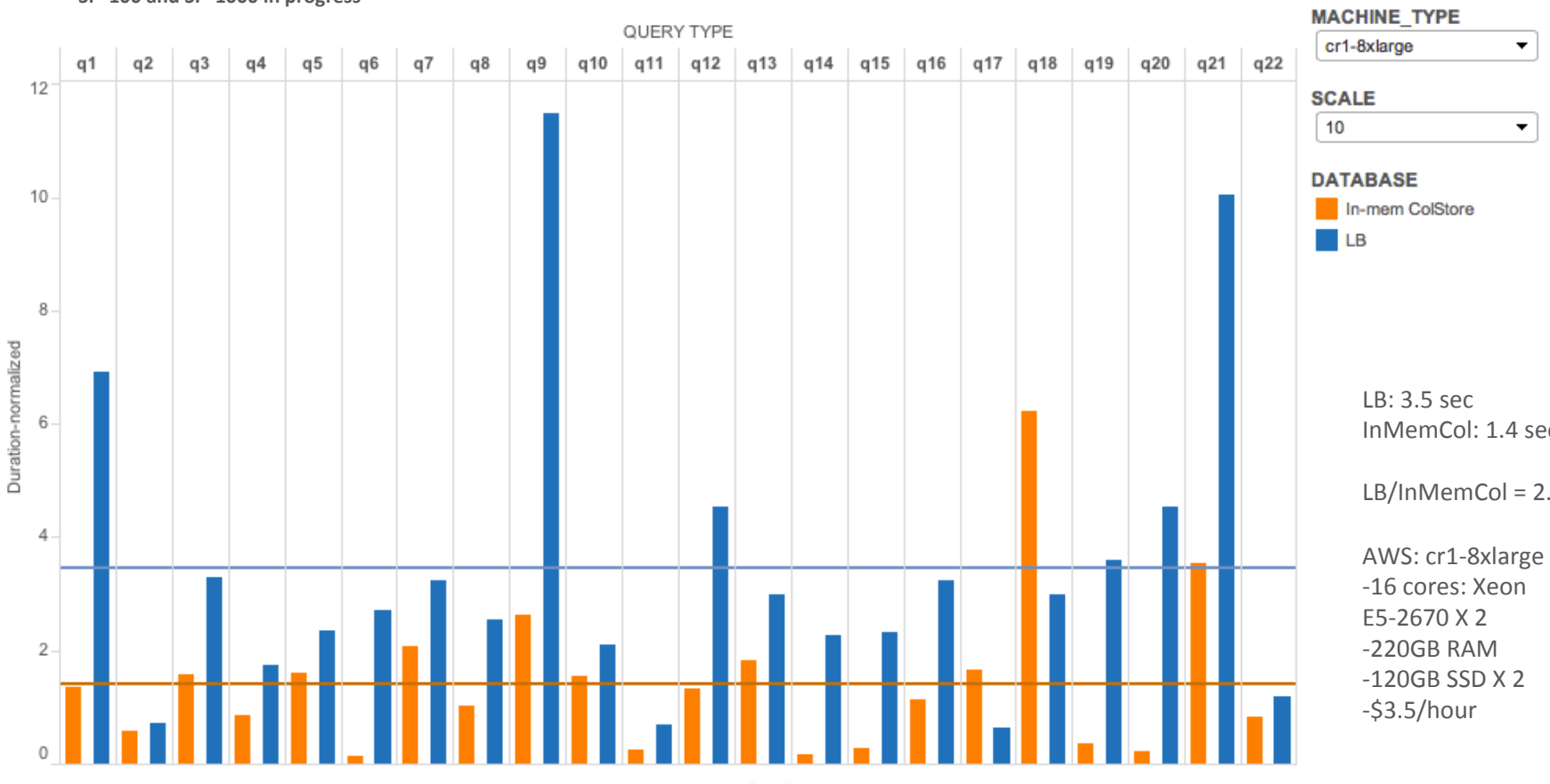


OldSQL: 70.1 sec
InMemCol: 1.9 sec (37 X)
Col: 4.8 sec (15 X)

AWS: HS1-8xlarge
-8 cores: Xeon E5-2650 X 2
-117GB RAM
-2TB HD X 24
-$4.6/hour

TPC-H: LB, IN MEMORY COLUMN

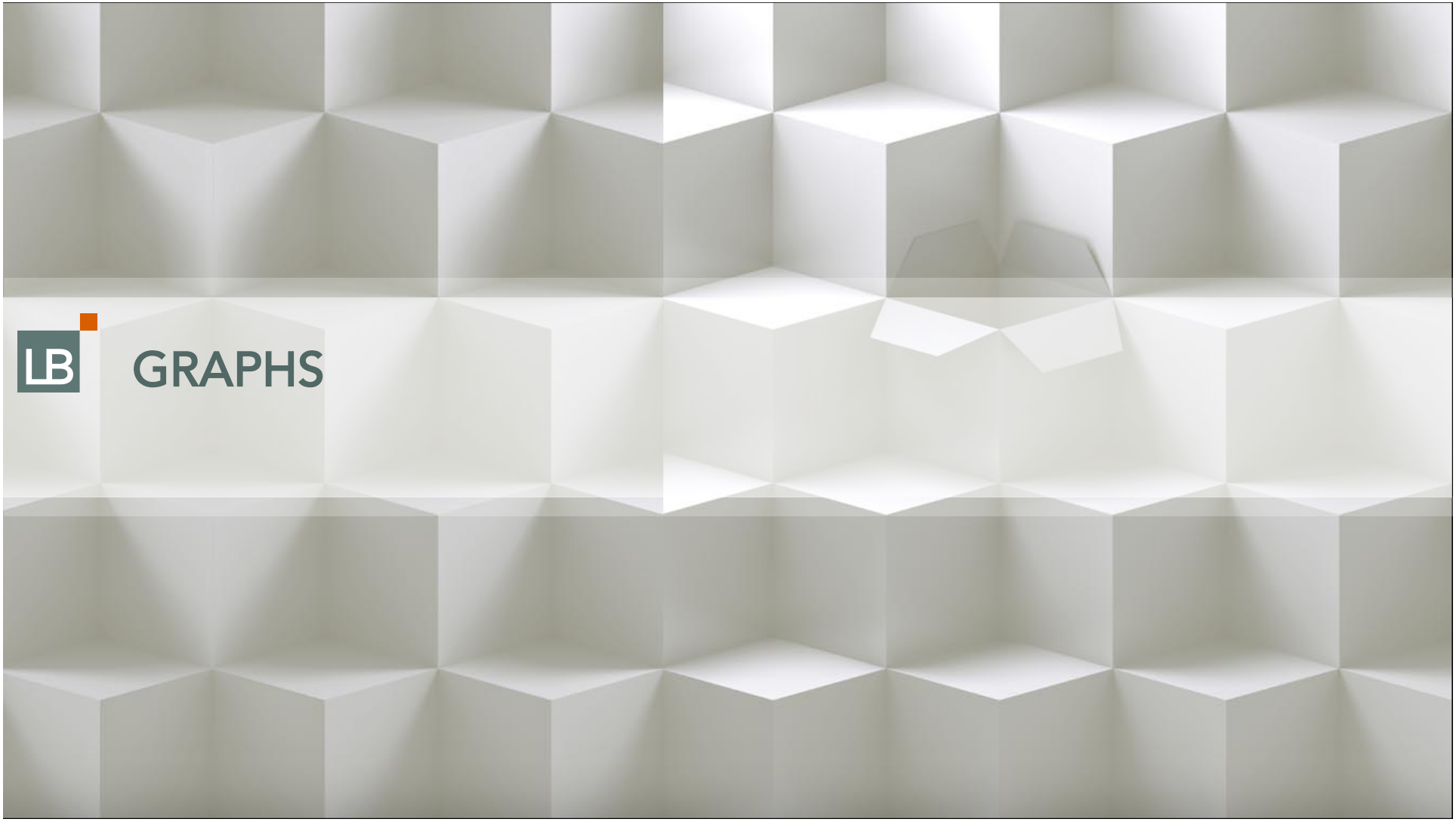
SF=100 and SF=1000 in progress



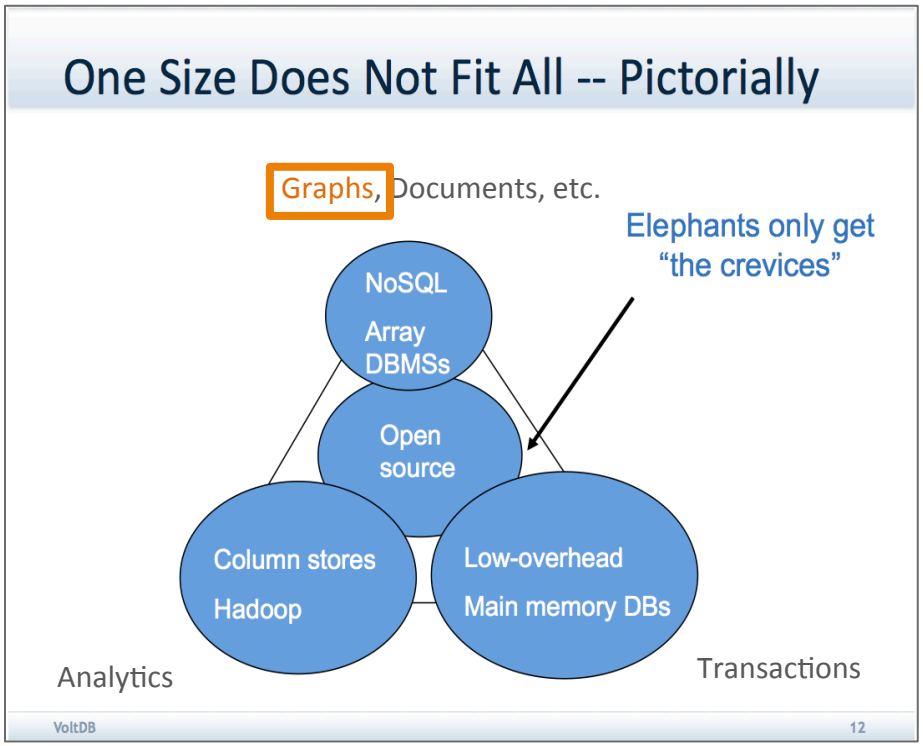
LB: 3.5 sec
InMemCol: 1.4 sec

LB/InMemCol = 2.5 X

AWS: cr1-8xlarge
-16 cores: Xeon E5-2670 X 2
-220GB RAM
-120GB SSD X 2
-$3.5/hour


LB
GRAPHS

THE ARGUMENT FOR SPECIALIZATION



"…specialized systems can each be a factor of 50 faster than the single "one size fits all" system…A factor of 50 is nothing to sneeze at."

-Stonebraker's Top 10 Assertions About Data Warehouses, 2010

MOTIVATION FOR GRAPHS AND NETWORKS

- Useful for wide variety of problems
  - Social graphs
  - Supply chain networks
  - Communication networks
  - Financial transaction networks
  - Gene regulatory networks
  - Disease transmission networks
  - Ecological food networks
  - Sensor networks
  - Semantic Web
  - Program analysis (call-graph analysis)
- Growing interest in methods for analyzing such graph and network data for anomaly detection, vulnerability prediction, scientific discovery, and assessing the potential impact of interventions
- Over 20 specialized graph database and graph engine solutions: e.g. Neo4j, DEX, Virtuoso,Titan, HypergraphDB, AllegroGraph, Graphlab, Pregel, Giraph, GraphX,…

THE ARGUMENT FOR GRAPH DATABASES

- Queries that start with a node and go to nearby nodes shouldn't take time proportional to the size of the graph

- Conclusion:
  - We must use pointer based navigational techniques to traverse the graph
  - We have to learn to "think like a vertex"

- Similar to network databases in the 1970's and Object Databases in the 1980's and 1990's.

WE HEAR YOU JEFF!!!

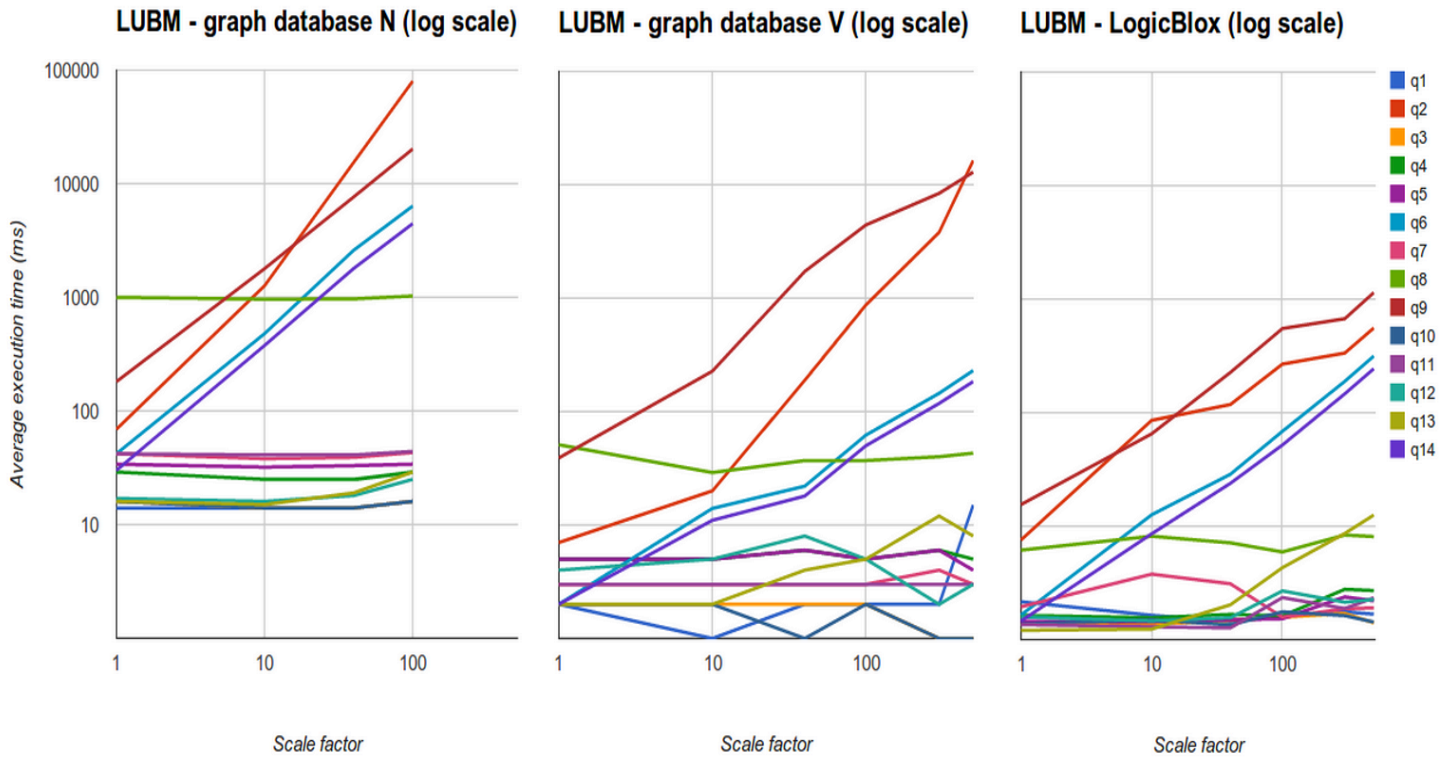
Jeffrey Ullman

Shared publicly - Sep 12, 2013

I was back at the closing workshop for XLDB this morning. I wanted to hear the discussion of graph databases, but they basically had trouble understanding that graphs are really relations, and unless you need to do recursions, the same technology as is used for relations works.

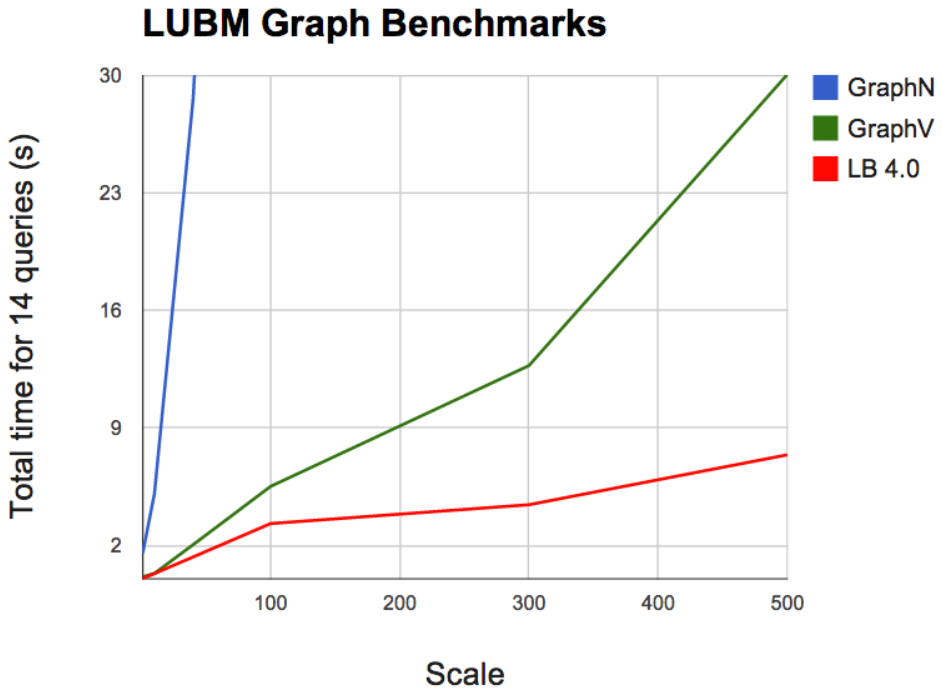
LEHIGH UNIVERSITY GRAPH BENCHMARK (LUBM)

4 cores, 16GB RAM

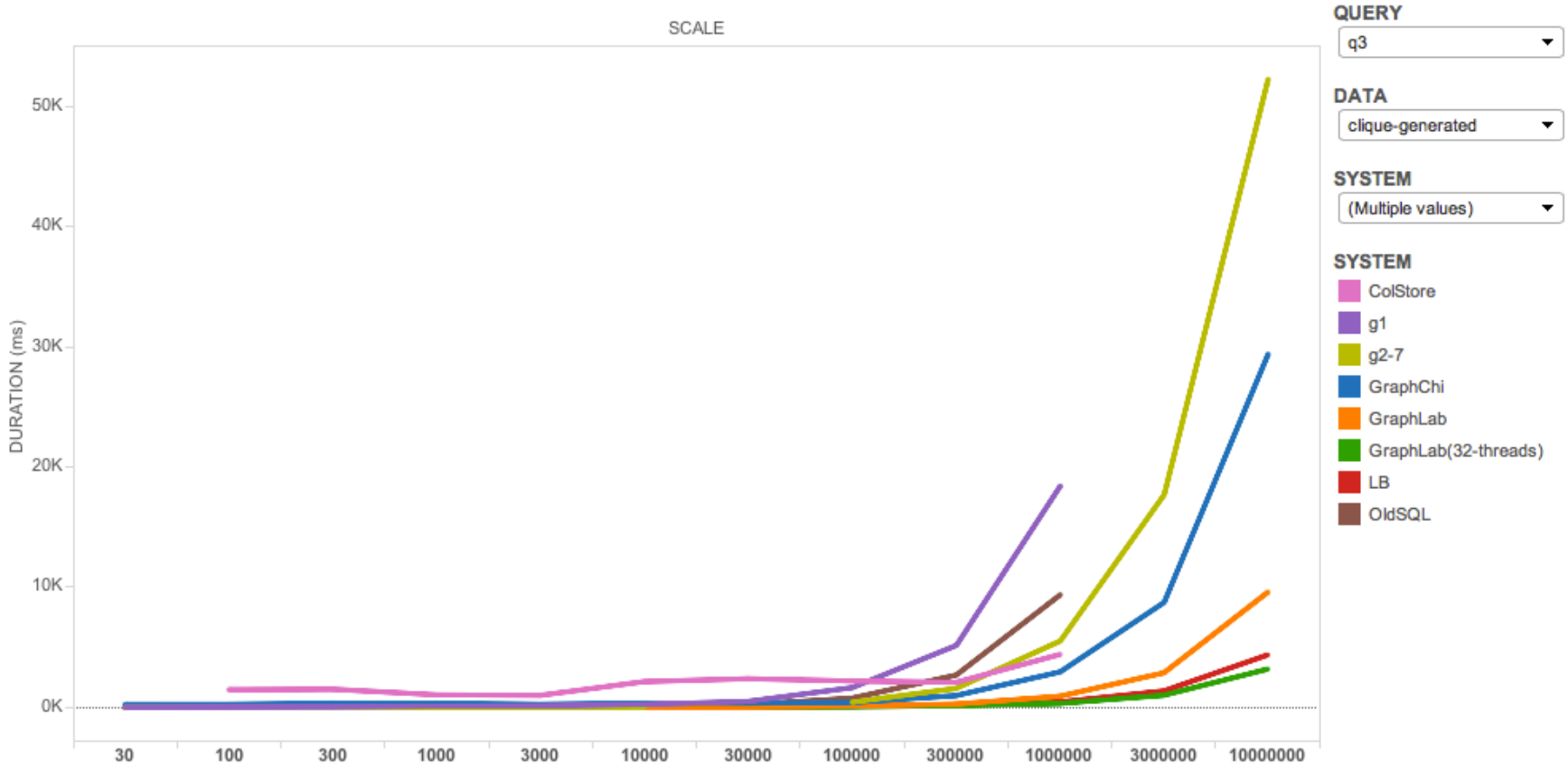


LEHIGH UNIVERSITY GRAPH BENCHMARK (LUBM)

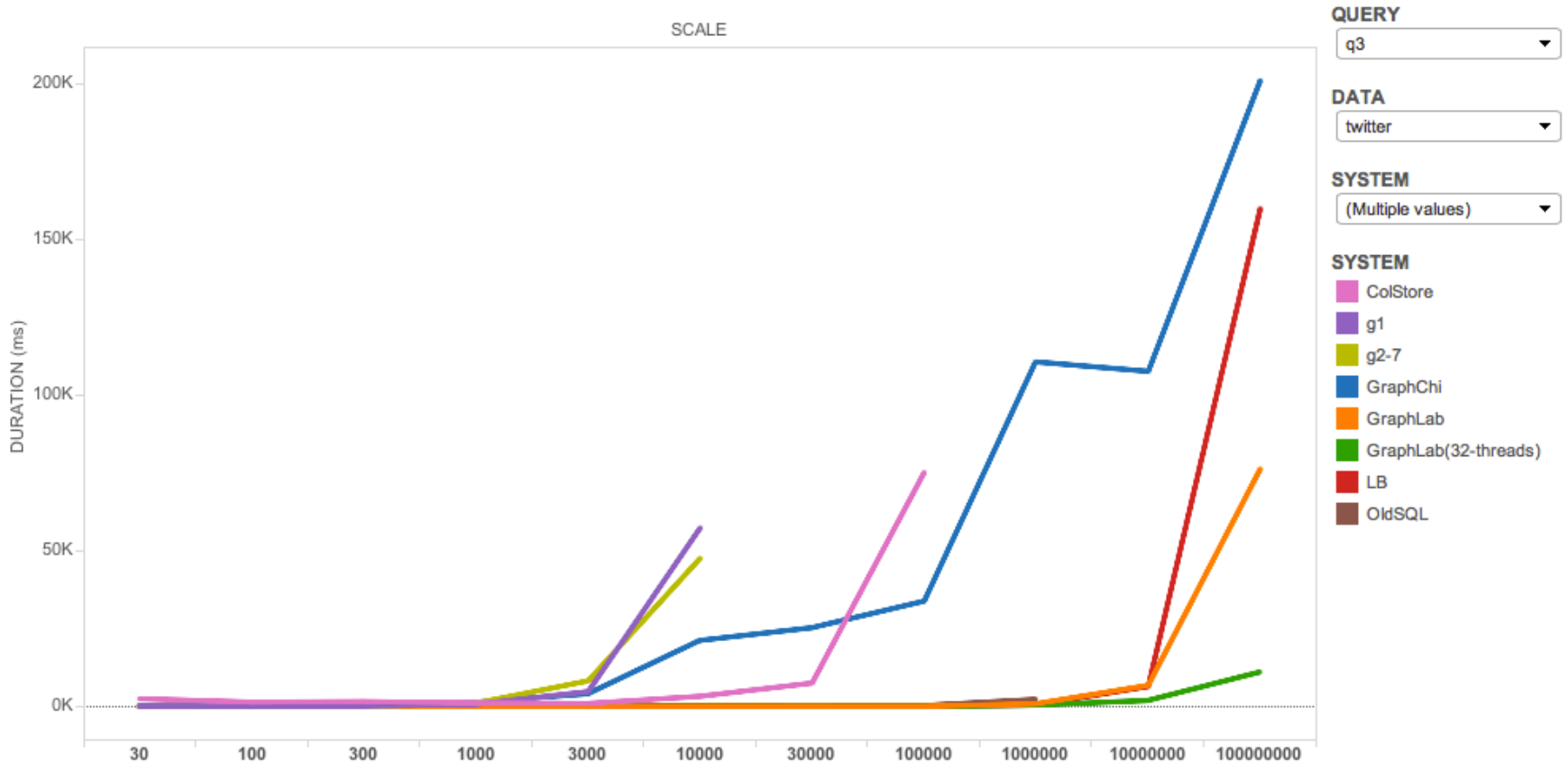
Aggregate results across 14 LUBM queries, 4 cores, 16GB RAM



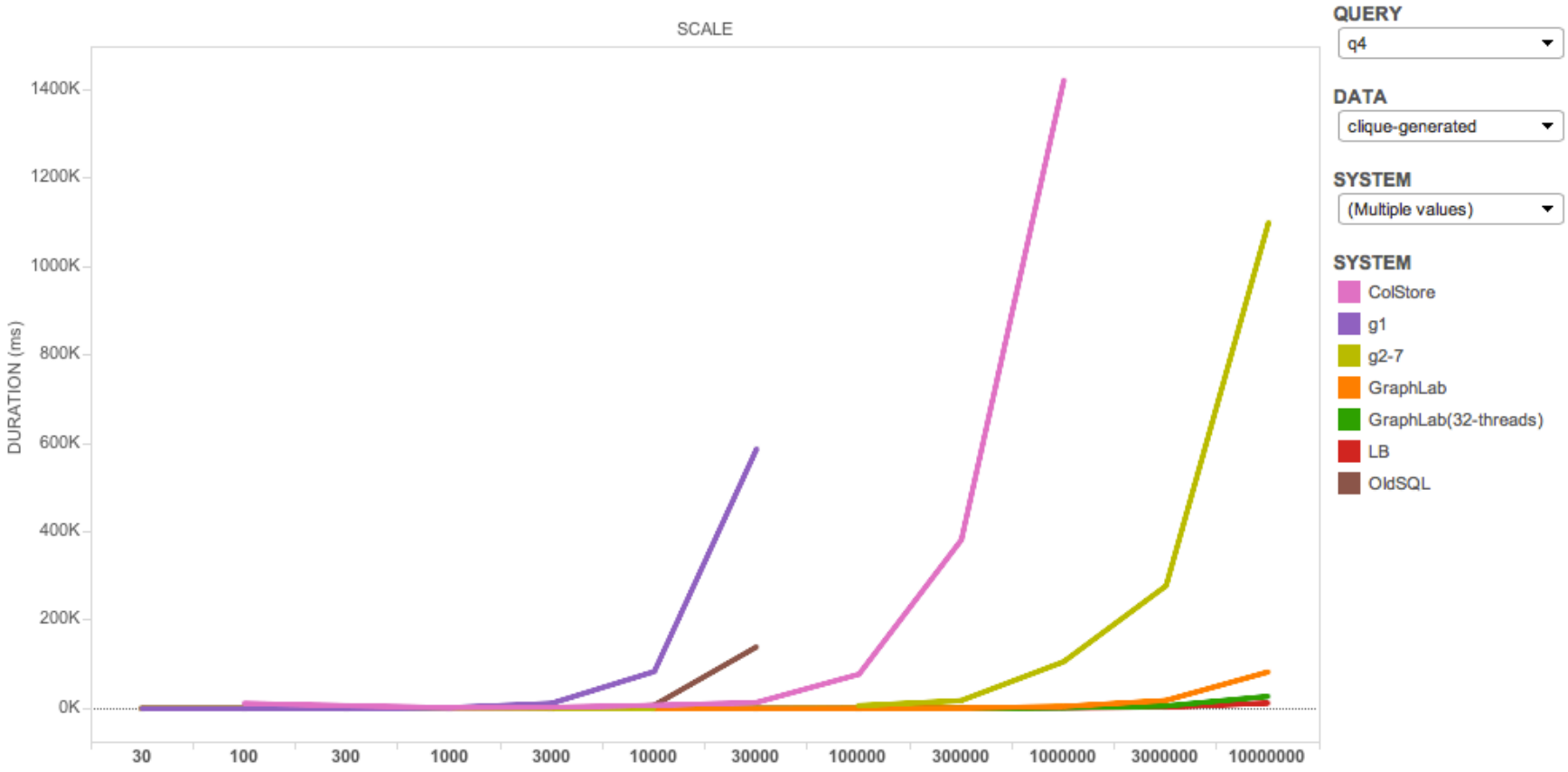
3-CLIQUE (TRIANGLE) QUERIES ON SYNTHETIC DATA



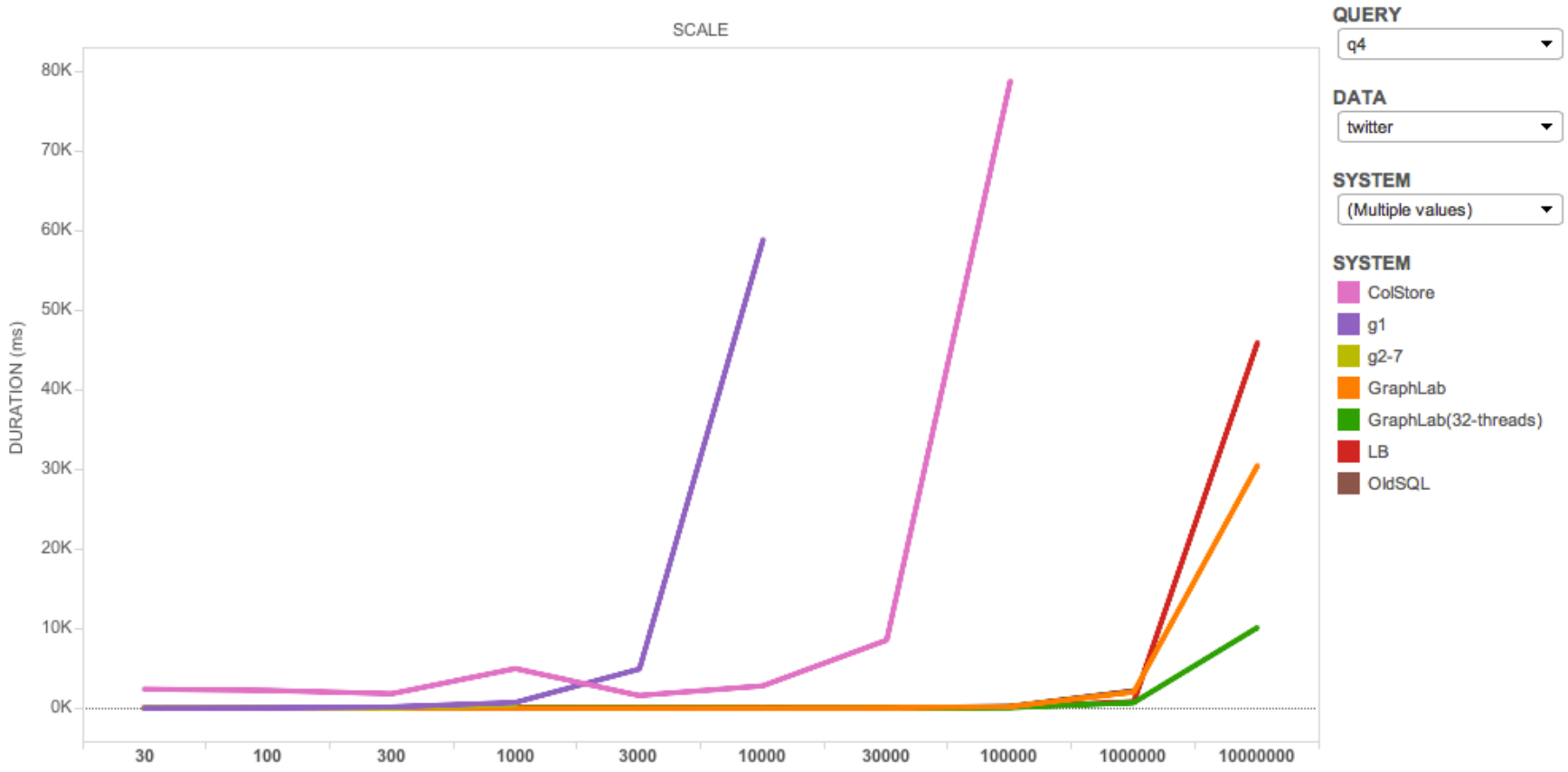
3-CLIQUE (TRIANGLE) QUERIES ON REAL DATA



4-CLIQUE QUERIES ON SYNTHETIC DATA



4-CLIQUE QUERIES ON REAL DATA



Clique Computation

LogiQL – 3 Clique

```
cliques(a,b,c)
  <-
  edge(a,b),
  edge(a,c),
  edge(b,c),
  a < b < c.
```

LogiQL – 4 Clique

```
cliques(a,b,c,d)
  <-
  edge(a,b),
  edge(a,c),
  edge(a,d),
  edge(b,c),
  edge(b,d),
  edge(c,d),
  a < b < c < d.
```

3 Clique (Triangle) Computation

SQL (naïve query – doesn't perform well)

```
select distinct v1.x as x,v2.x as y, v3.x as
w
   from edge as v1,edge as v2,edge as v3
   where
        v1.y = v2.x
    and v2.y = v3.x
    and exists(
      select 1 from edge as vv1 where
vv1.x = v1.x and vv1.y = v3.x)
    and v1.x < v2.x
    and v2.x < v3.x;
```

SPARQL

```
sparql PREFIX g: <http://logicblox.com/
graph>
   SELECT DISTINCT ?av ?bv ?cv FROM <
$database>
   WHERE {
     ?a g:edge ?b .
     ?a g:edge ?c .
     ?b g:edge ?c .
     ?a g:value ?av .
     ?b g:value ?bv .
     ?c g:value ?cv .
     FILTER (xsd:int(?av) < xsd:int(?bv) and
           xsd:int(?bv) < xsd:int(?cv))
   };
```

3 Clique (Triangle) Computation

GraphLab – C++

```
class triangle_count :
      public graphlab::ivertex_program<graph_type, set_union_gather> {
public:
  bool do_not_scatter;

  // Gather on all edges
  edge_dir_type gather_edges(icontext_type& context, const vertex_type& vertex) const {
    return graphlab::ALL_EDGES;
  }

  /*
   * For each edge, figure out the ID of the "other" vertex
   * and accumulate a set of the neighborhood vertex IDs.
   */
  gather_type gather(icontext_type& context, const vertex_type& vertex, edge_type& edge)
const {
    set_union_gather gather;
    graphlab::vertex_id_type otherid = edge.target().id() == vertex.id() ?
                          edge.source().id() : edge.target().id();

    size_t other_nbrs = (edge.target().id() == vertex.id()) ?
        (edge.source().num_in_edges() + edge.source().num_out_edges()):
        (edge.target().num_in_edges() + edge.target().num_out_edges());

    size_t my_nbrs = vertex.num_in_edges() + vertex.num_out_edges();

    if (PER_VERTEX_COUNT || (other_nbrs > my_nbrs) || (other_nbrs == my_nbrs && otherid >
vertex.id())) {
      gather.v = otherid;
    }
    return gather;
  }
```

GraphLab – C++ (cont.)

```
  /*
   * the gather result now contains the vertex IDs in the neighborhood.
   * store it on the vertex.
   */
  void apply(icontext_type& context, vertex_type& vertex, const gather_type& neighborhood) {
    do_not_scatter = false;
    if (neighborhood.vid_vec.size() == 0) {
      // neighborhood set may be empty or has only 1 element
      vertex.data().vid_set.clear();
      if (neighborhood.v != (graphlab::vertex_id_type(-1))) {
        vertex.data().vid_set.vid_vec.push_back(neighborhood.v);
      }
    }
    else {
      vertex.data().vid_set.assign(neighborhood.vid_vec);
    }
    do_not_scatter = vertex.data().vid_set.size() == 0;
  }

  /*
   * Scatter over all edges to compute the intersection.
   * I only need to touch each edge once, so if I scatter just on the
   * out edges, that is sufficient.
   */
  edge_dir_type scatter_edges(icontext_type& context, const vertex_type& vertex) const {
    if (do_not_scatter) return graphlab::NO_EDGES;
    else return graphlab::OUT_EDGES;
  }


  /*
   * For each edge, count the intersection of the neighborhood of the
   * adjacent vertices. This is the number of triangles this edge is involved
   * in.
   */
  void scatter(icontext_type& context, const vertex_type& vertex, edge_type& edge) const {
    const vertex_data_type& srclist = edge.source().data();
    const vertex_data_type& targetlist = edge.target().data();
    if (targetlist.vid_set.size() < srclist.vid_set.size()) {
      edge.data() += count_set_intersect(targetlist.vid_set, srclist.vid_set);
    }
    else {
      edge.data() += count_set_intersect(srclist.vid_set, targetlist.vid_set);
    }
  }
};
```

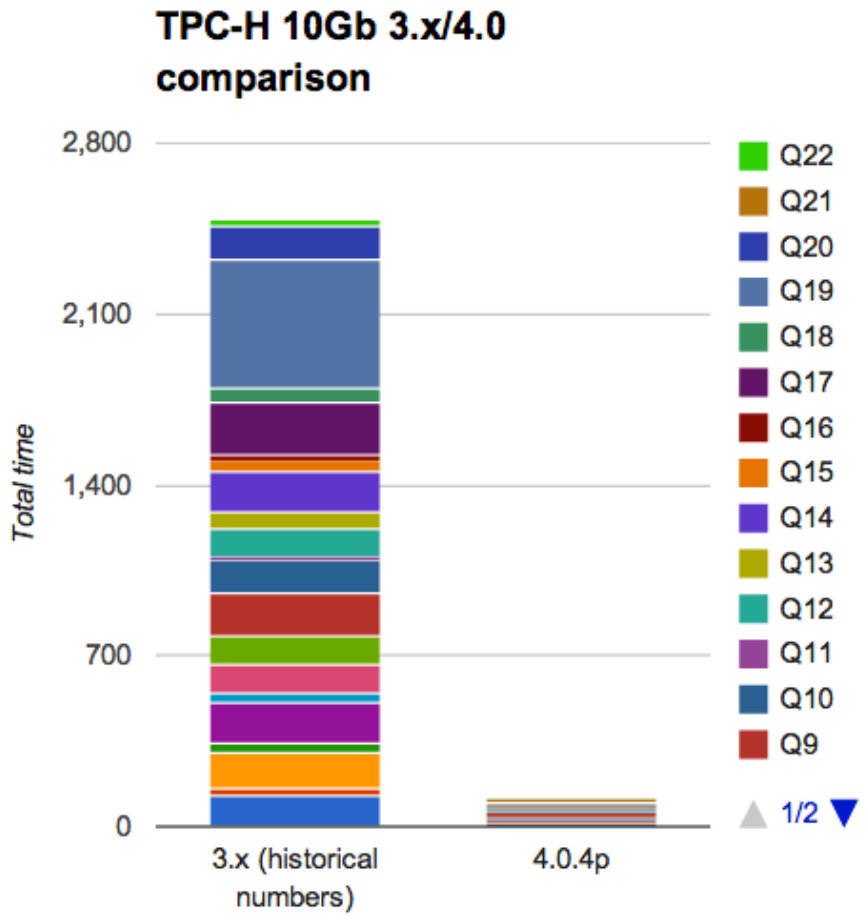
Screaming Fast Program Analysis

Context sensitive program analysis – querying the call graph

- "Exception Analysis and Points-To Analysis - Better Together"(ISSTA'09)
- "Strictly Declarative Specification of Sophisticated Points-to Analyses"(OOPSLA'09)
- "Pick Your Contexts Well – Understanding Object-Sensitivity"(POPL'11)
- "Efficient and Effective Handling of Exceptions in Java Points-To Analysis" (CC'13)
- Hybrid Context Sensitivity for Points-To Analysis" (PLDI'13)
- "Set-based pre-processing for points-to analysis" (OOPSLA'13)

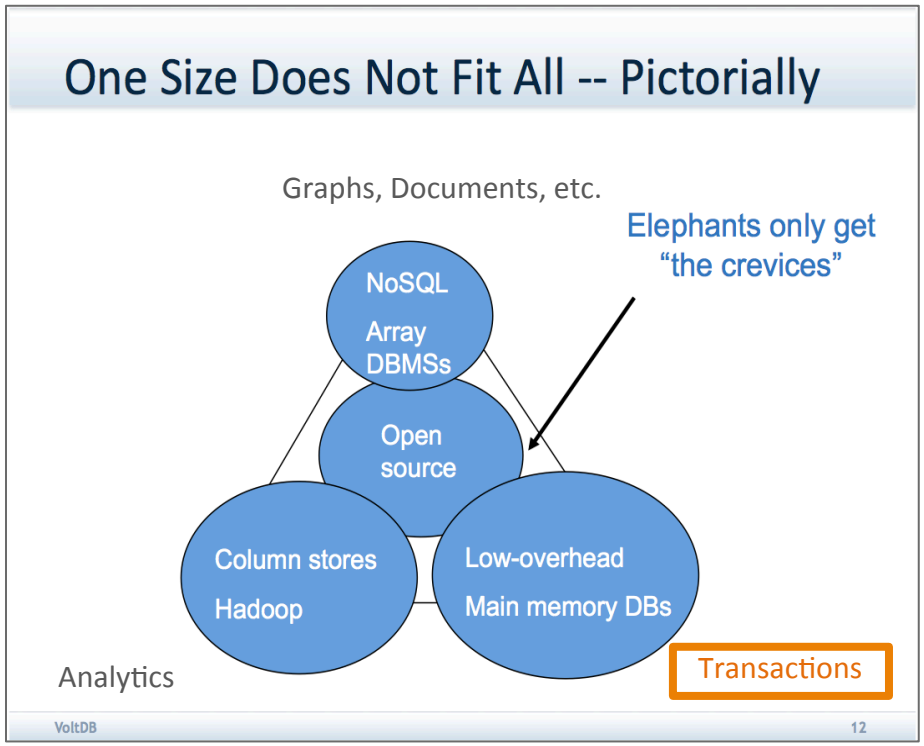
All work today done on v3.x of LB which lacks many of the features needed to achieve the performance we've seen for analytics & graphs

PROGRAM ANALYSIS WORK TODATE DONE ON LB 3.X




LB
TRANSACTIONS

THE ARGUMENT FOR SPECIALIZATION

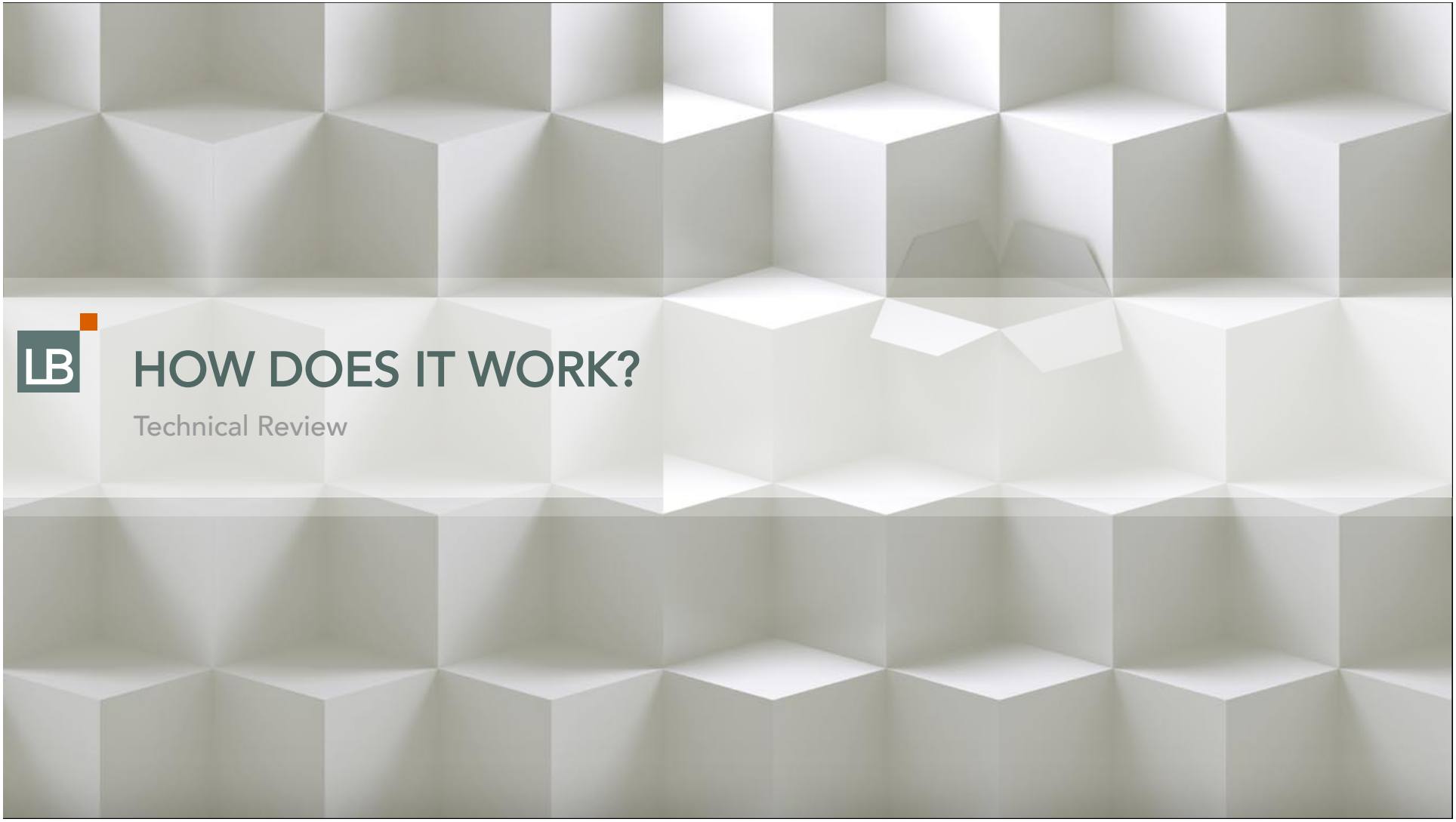


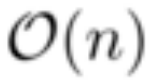
"…specialized systems can each be a factor of 50 faster than the single "one size fits all" system…A factor of 50 is nothing to sneeze at."

-Stonebraker's Top 10 Assertions About Data Warehouses, 2010

LOGICBLOX FOR OLTP

- Fully ACID, fully SERIALIZABLE
- Micro benchmarks measure 1,000 to 10,000 trx/sec/core
- Each transaction starts by branching a version of the database:
  - Very fast: currently 80,000 branches per cpu core per second
  - Changes made in one branch (one transaction) have no visibility outside the transaction: perfect isolation
- No locks for read or **write** transactions
  - Read-only transactions are perfectly scalable: do not need to be concerned about changes occurring in other branches.
    - throughput = cpus X (single-cpu-tps)
  - LogiQL-based handling of transaction conflicts – less principled solutions have to use locks or limit write queries. LogicBlox does neither.
- TPC-C benchmark forthcoming


HOW DOES IT WORK?
Technical Review



Computer Science @CompSciFact 28 Sep

"Computer science is now about systems. It hasn't been about algorithms since the 1960's." -- Alan Kay #hlf13

Expand

PHILOSOPHY: BRAINS BEFORE BRAWN

- Algorithmic scalability
  - New worst-case optimal join algorithm
  - Adaptive domain decomposition for parallelization
  - Incremental maintenance proportional to trace edit distance
- Data structures
  - Compression: close to info-theoretic limit in some cases
  - I/O minimization, cache consciousness
  - Persistent data structures for strongest consistency guarantees, full serializability, built-in auditability
- Unified, declarative programming model
  - For complete freedom to evaluate the model in the most efficient way possible.
  - Cost estimation
    - concentration inequalities for estimates: relative error → 0 (asymptotically almost surely)
  - Query optimizer
    - finds plans with cost at most $(1 + \varepsilon) \cdot \mathrm{OPT}$, where OPT is the optimal (estimated) cost, and choice of ε
- Brute Force
  - Distribution across thousands of cores; in-memory; GPUs

A SMART JOIN ALGORITHM

- Leapfrog Triejoin: A Simple Worst-Case Optimal Join Algorithm
  T. Veldhuizen 2012,
  http://arxiv.org/abs/1210.0481

- Really a "select-project-union-difference-join" algorithm,
  i.e. ∃1 queries

- Proven worst-case optimal
  in the sense of (Ngo, Porat, Re, Rudra, PODS'12 – Best Paper)

Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm

Todd L. Veldhuizen
LogicBlox Inc.
Two Midtown Plaza
1349 West Peachtree Street NW
Suite 1880, Atlanta GA 30309
tveldhui@{logicblox.com,acm.org}

ABSTRACT

Recent years have seen exciting developments in join algorithms. In 2008, Atserias, Grohe and Marx (henceforth AGM) proved a tight bound on the maximum result size of a full conjunctive query, given constraints on the input relation sizes. In 2012, Ngo, Porat, Ré and Rudra (henceforth NPRR) devised a groundbreaking join algorithm with worst-case running time proportional to the AGM bound [9]. Our commercial Datalog system LogicBlox employs a novel join algorithm, *leapfrog triejoin*, which compared conspicuously well to the NPRR algorithm in preliminary benchmarks. This spurred us to investigate its complexity. In this paper we establish that leapfrog triejoin is also worst-case optimal, up to a log factor, in the sense of NPRR. Moreover, leapfrog triejoin achieves worst-case optimality for finer-grained classes, for example, those defined by constraints on the size of projections of the input relations. We show that leapfrog triejoin is asymptotically faster than NPRR for some such classes. On a practical note, leapfrog triejoin can be implemented using conventional data structures such as B-trees, and extends naturally to $\exists_1$ queries. We believe our algorithm offers a useful addition to the existing toolbox of join algorithms, being easy to absorb, simple to implement, and having a concise optimality proof.

General Terms

Algorithms,Theory

1. INTRODUCTION

Join processing is a fundamental and extremely well-studied problem in database systems. Many useful queries can be formulated as one or more *full conjunctive queries*.

A full conjunctive query is a conjunctive query [4] with no projections, i.e., every variable in the body appears in the head [1]. As a running example we use the query defined by the Datalog rule:

$$Q(a,b,c) \leftarrow R(a,b), S(b,c), T(a,c). \qquad (1)$$

(For intuition: if $R = S = T$, then Q finds triangles.)

Given constraints on the sizes of the input relations such as $|R| \leq n$, $|S| \leq n$, $|T| \leq n$, what is the maximum possible size of the query $|Q|$? This question has practical import, since a tight bound $|Q| \leq Q^*$ implies an $\Omega(Q^*)$ worst-case running time for algorithms answering such queries.

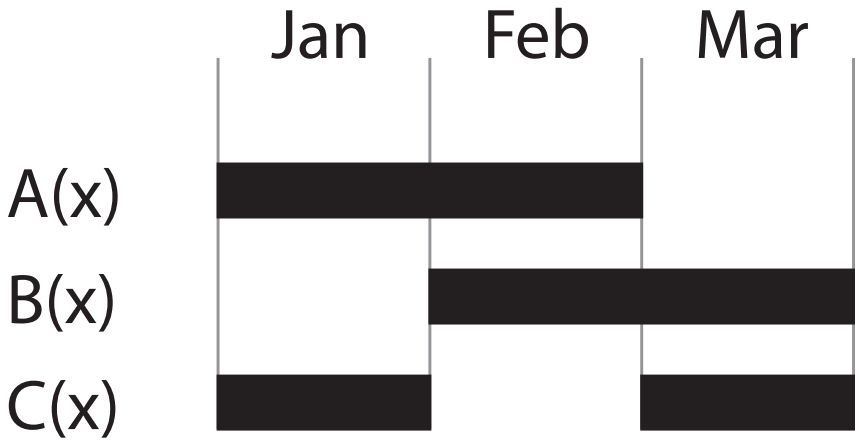
Atserias, Grohe and Marx (henceforth AGM [3]) established a tight bound on the size of Q: the *fractional edge cover bound* (Section 2.2). For the case where $|R| = |S| = |T| = n$, the fractional cover bound yields $|Q| \leq n^{3/2}$. In earlier work, Grohe and Marx [7] gave an algorithm with running time $O(|Q^*|^2 f(n))$, where $f(n)$ is a polynomial determined by the fractional cover bound. In 2012, Ngo, Porat, Ré and Rudra (henceforth NPRR [9]) devised a ground-breaking algorithm with worst-case running time $O(Q^*)$, matching the AGM bound. The algorithm is non-trivial, and its implementation and analysis depend on some fairly deep machinery also developed in the paper.

The NPRR algorithm was brought to our attention by Dung Nguyen, a summer intern at LogicBlox, who implemented it experimentally in our commercial Datalog system. His preliminary benchmarks suggested that our own join algorithm performed dramatically better than NPRR for at least some test problems [10]. LogicBlox uses a novel and hitherto proprietary join algorithm we call *leapfrog triejoin*. These benchmark results motivated us to analyze our algorithm, in light of the breakthroughs of NPRR.

Conventional join implementations employ a stable of join operators (see e.g. [6]) which are composed in a tree to produce the query result; this tree is prescribed by a *query plan* produced by the optimizer. The query plan often relies on materialization of intermediate results. In contrast, leapfrog triejoin joins all input relations simultaneously without materializing any intermediate

INTUITION BEHIND LFTJ

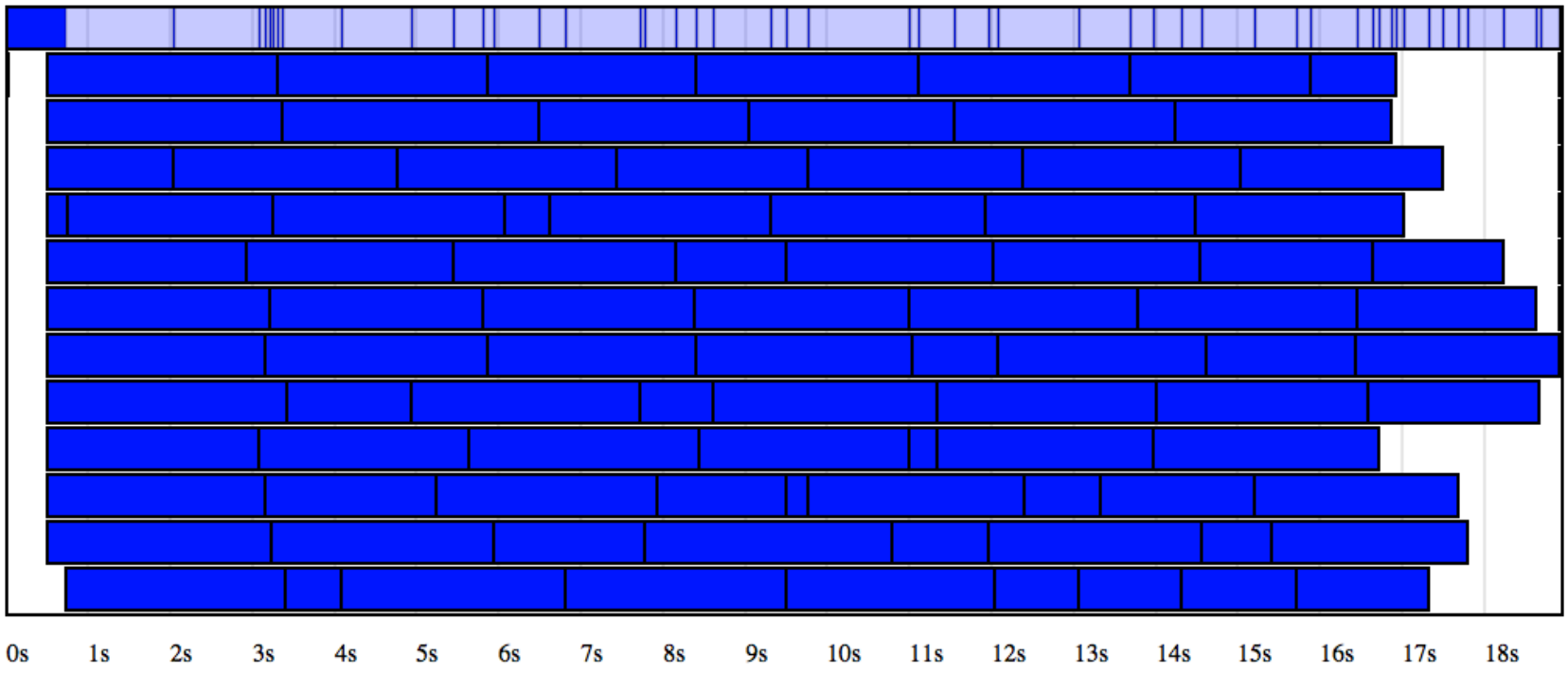
- Why are simultaneous joins more efficient than pairwise joins?



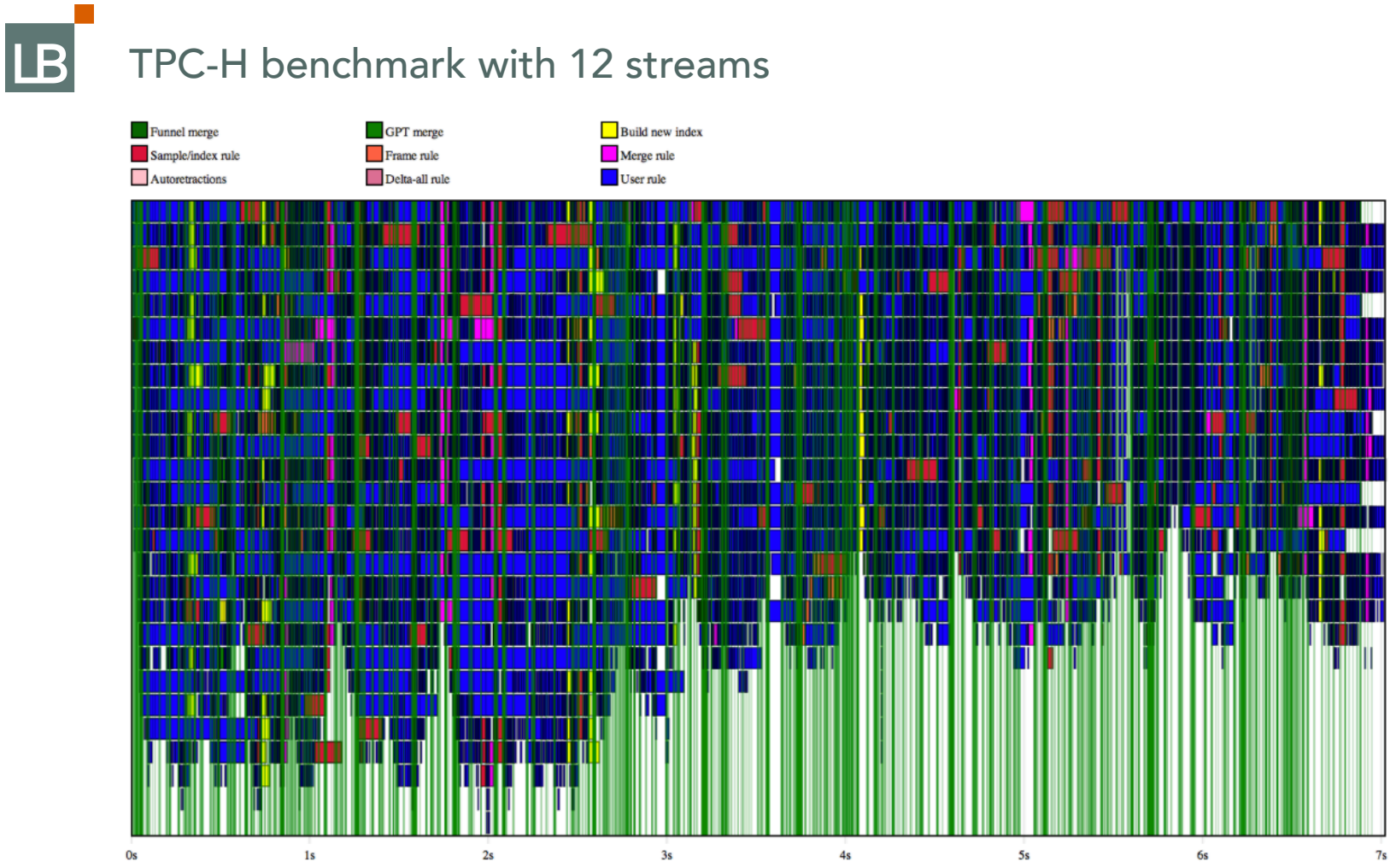
- Suppose A has 1M records for Jan and 1M for Feb, B has 1M for Feb and 1M for Mar, etc. Any pairwise join generates 500K results. Simultaneous join ➔ no results, examine < 10 records.

A SMARTER WAY TO PARALLELIZE QUERIES

- Paper in progress
- Fixed domain decomposition (sharding), having N jobs for N cpus ⇒ poor load balancing
- Instead:
  - For each query, a custom decomposition in which each subdomain will require the same anticipated amount of work
  - Partitioning policy is automatic, dynamic, and adaptive
  - Have k*N jobs, where k is large:
    - scheduling/packing are NP-complete problems
    - fitting boxes into a shipping container ⇒ hard
    - pouring sand into a dumptruck ⇒ dead easy
    - sand, not boxes



TPC-H query on 12 cores



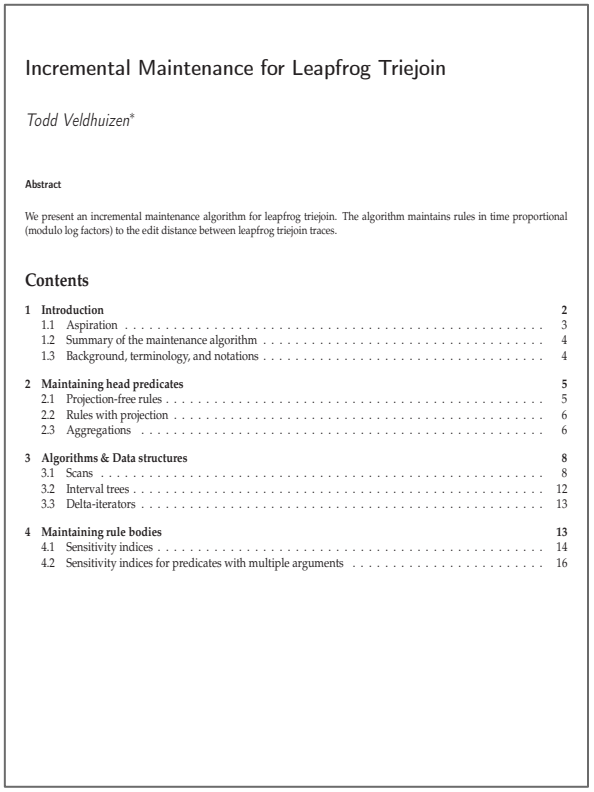
TPC-H benchmark with 12 streams

SMARTER INCREMENTAL MAINTENANCE

- Incremental Maintenance for Leapfrog Triejoin, T. Veldhuizen 2013, http://arxiv.org/abs/1303.5313

- Replaced our implementation of classic **Count** and **DRed** algorithms [Gupta+ 93]
  - Count for non-recursive predicates
  - DRed ("delete and rederive") for recursive predicates

- Guarantees that the work done is proportional to the trace edit distance between the before and after



Incremental Maintenance for Leapfrog Triejoin

*Todd Veldhuizen**

Abstract

We present an incremental maintenance algorithm for leapfrog triejoin. The algorithm maintains rules in time proportional (modulo log factors) to the edit distance between leapfrog triejoin traces.

Contents

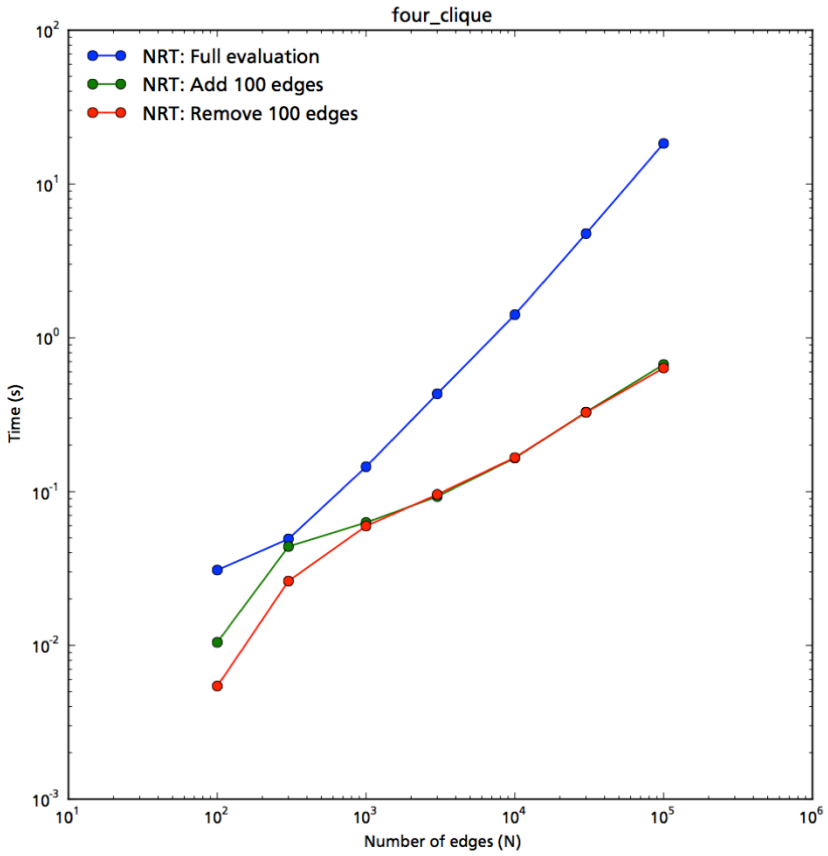
1 Introduction 2
1.1 Aspiration . . . 3
1.2 Summary of the maintenance algorithm . . . 4
1.3 Background, terminology, and notations . . . 4

2 Maintaining head predicates 5
2.1 Projection-free rules . . . 5
2.2 Rules with projection . . . 6
2.3 Aggregations . . . 6

3 Algorithms & Data structures 8
3.1 Scans . . . 8
3.2 Interval trees . . . 12
3.3 Delta-iterators . . . 13

4 Maintaining rule bodies 13
4.1 Sensitivity indices . . . 14
4.2 Sensitivity indices for predicates with multiple arguments . . . 16

INCREMENTAL MAINTENANCE OF 4-CLIQUE

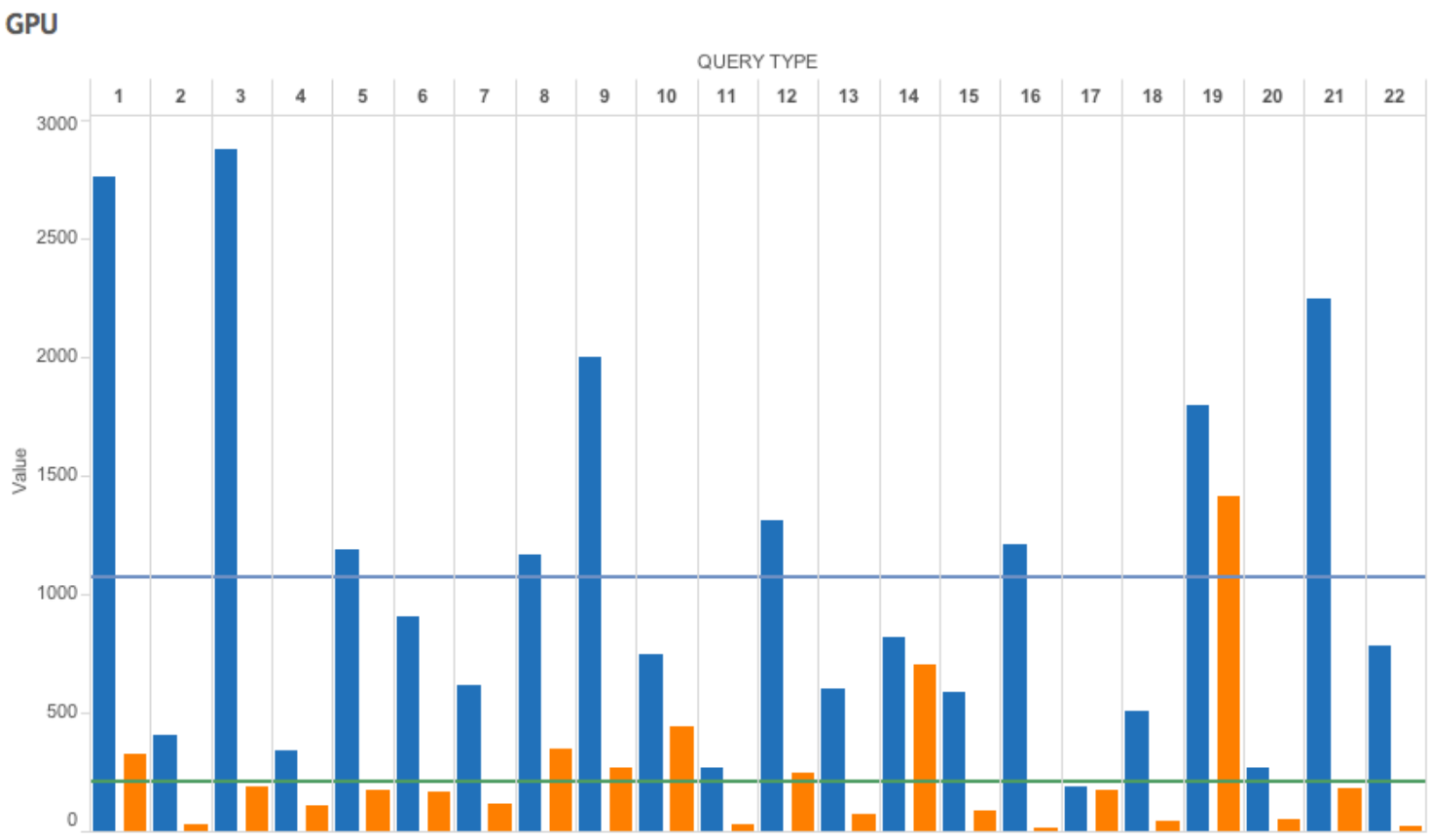


RESEARCH & ACADEMIC COLLABORATIONS

LOGICBLOX FOR LANGUAGE AND SYSTEMS RESEARCH

- LB source available for community to experiment with
  - New language features
    - E.g. probabilistic programming
  - Join Algorithms
    - E.g. MINESWEEPER – a new instance optimal algorithm
  - Incremental maintenance algorithms
    - E.g. DBTOASTER
  - Query costing
  - Query optimization
  - Query parallelization
  - Alternative hardware configurations (e.g. GPU or FPGA)
    - E.g. REDFOX
  - Transaction support
- Benchmarking framework
  - Add new benchmarks
  - Add new databases and systems
  - Add new data sets

EXAMPLE: LOGIQL ON GPUS



SF=1
GPU 4.5X faster
GPU 13.5X per $

Nvidia Geoforce GTX Titan
-2688 CUDA cores (4.5sp TF)
-6GB RAM
-PCI 3.0
-$1000

AWS: cr1-8xlarge
-16 cores: Xeon E5-2670 X 2
-220GB RAM
-$1556 X2

Collaboration with GT & NVidia

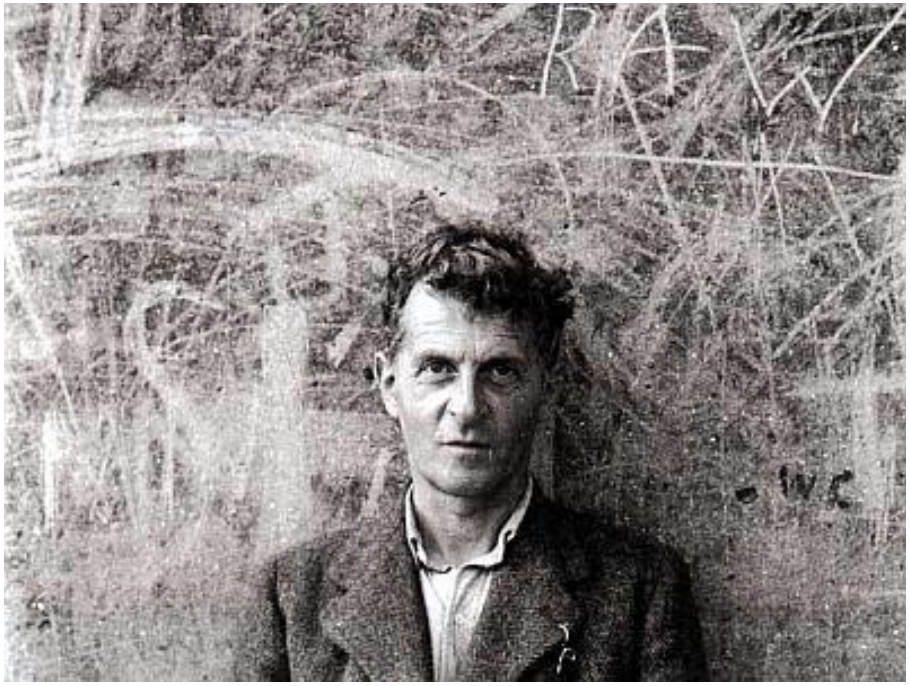
APPLICATIONS OF LB

- Number of researchers to accept academic license: 60
  - Last 10 came from: Osaka, Athens, Zurich, Oracle Labs, Stanford, Michigan State, Wuhan, Waterloo, Indian Institute of Science, IMDEA
- Program Analysis
- Graph based applications
  - Social
  - Fraud
- Simulation
- Provenance
- Network analysis
- Security
- Entity resolution

LB FOR EDUCATION

DEMO

“The more ambitious plan may have more chances of success” - George Polya, How To Solve It, 1973



“Don't get involved in partial problems, but always take flight to where there is a free view over the whole single great problem, even if this view is still not a clear one” – Ludwig Wittgenstein

WHAT DO YOU GET WHEN YOU CROSS A PL WITH A DB?

- A programming languages that
  - Supports out-of-core computations (in the era of Big Data)
  - Automatically parallelizes across many cores and across many machines
  - Provides support for concurrent user access to data
  - Provide automatic incrementalization of all programs
  - Provides a purely declarative programming model that allows end-users to interact with the program
- A databases that
  - Has an expressive query language that captures important complexity classes
  - Supports programming in the large
  - Compiles queries so that they can run at native speed
- And this PLDB will
  - Have automatic and transparent support for reactive and live programming
  - Support elastic resource addition and removal (in the era of the Cloud)
  - Take advantage of heterogeneous hardware resources: CPUs, GPUs, FPGAs
  - Support declarative statistical and mathematical model building (aka probabilistic programming)


THANK YOU.