Centrum Wiskunde & Informatica

Benchmarking
Graph Data Management Systems

EDBT Summer School 2015

Peter Boncz

boncz@cwi.nl

1. ***LDBC Social Network Benchmark***

   Tuesday: LDBC & SNB introduction

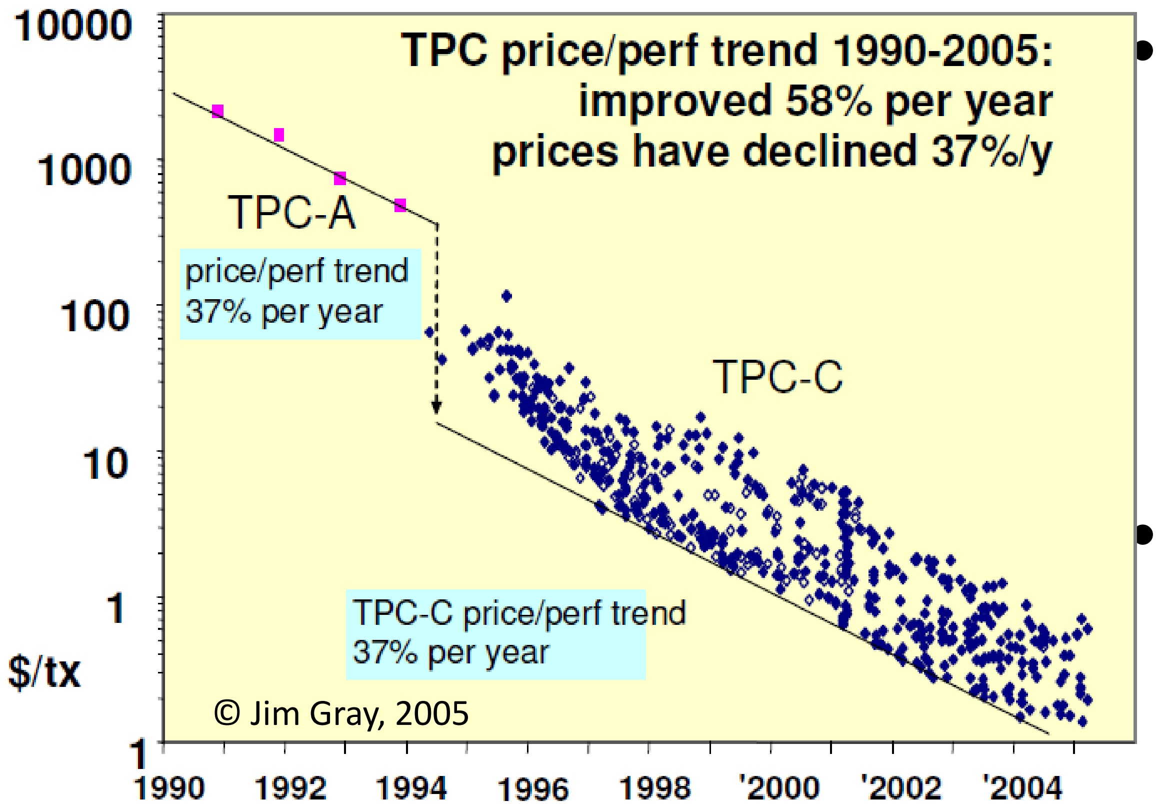
   Friday: SNB in depth

2. **SNB Programming Challenge** www.cwi.nl/~boncz/snb-challenge

   Tuesday: what it is about & hardware properties & tips

   Friday: the solution space & winners

Why Benchmarking?

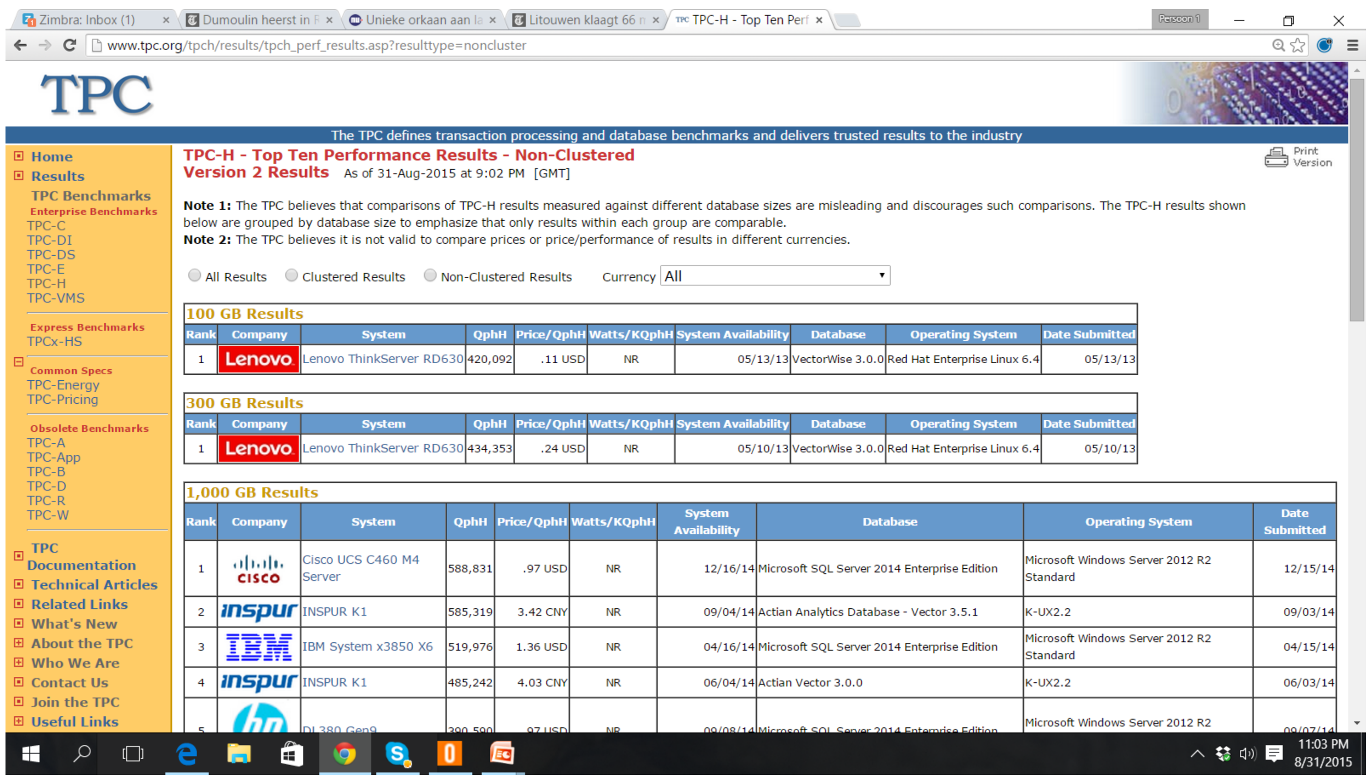


- make competing products comparable
- accelerate progress, make technology viable

What is the LDBC?

Linked Data Benchmark Council = LDBC

- Industry entity similar to TPC (www.tpc.org)
- Focusing on graph and RDF store benchmarking

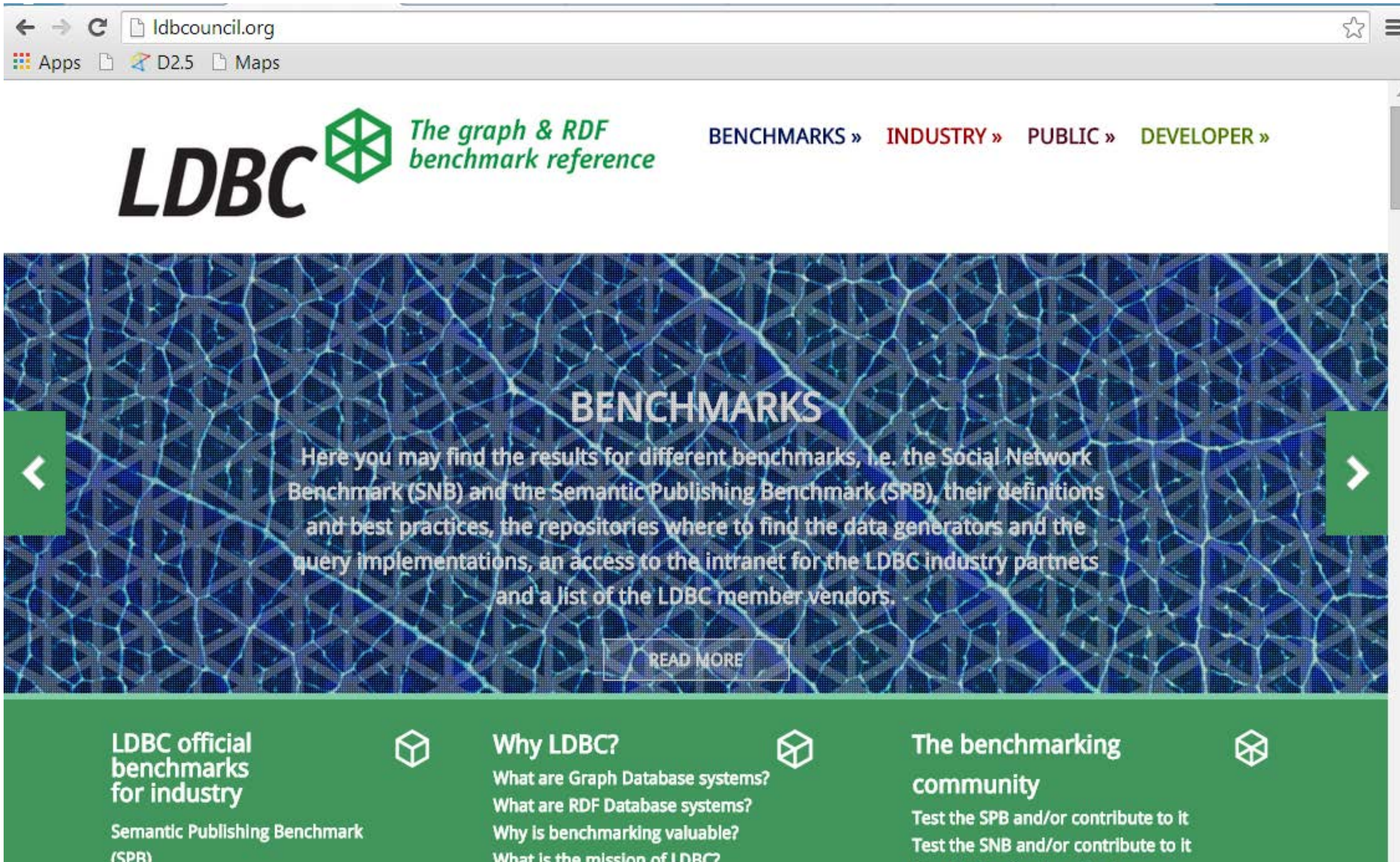


LDBC Organization (non-profit)

"sponsors"

+ non-profit members (FORTH, STI2) & personal members
+ Task Forces, volunteers developing benchmarks
+ TUC: Technical User Community (6 workshops, 36 graph and RDF user case studies, 12 vendor presentations)

ldbcouncil.org



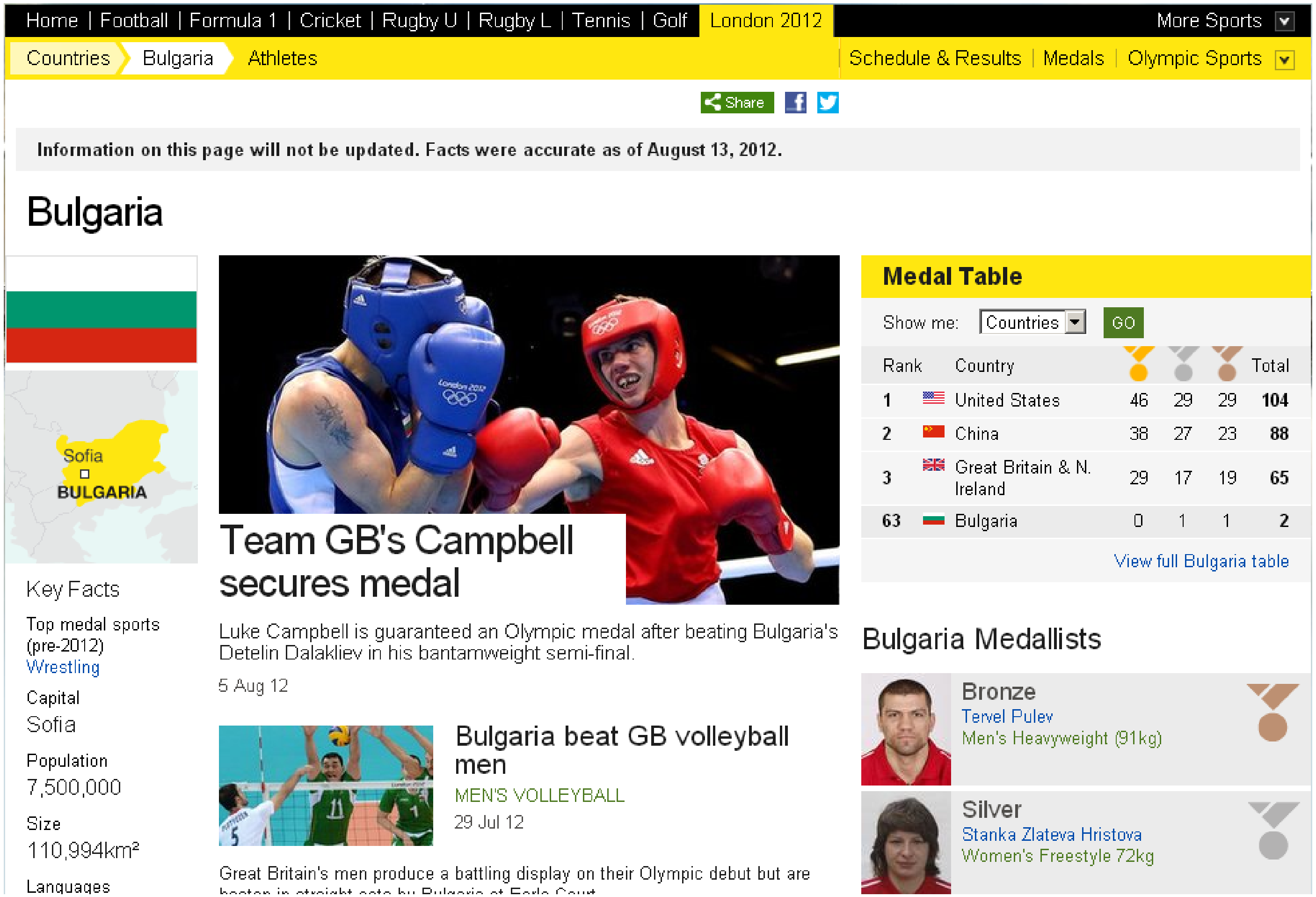
What does a benchmark consist of?

- Four main elements:
  - *data schema*: defines the structure of the data
  - *workloads*: defines the set of operations to perform
  - *performance metrics*: used to measure (quantitatively) the performance of the systems
  - *execution rules*: defined to assure that the results from different executions of the benchmark are valid and comparable
- Software as Open Source (GitHub)
  - data generator, query drivers, validation tools, ...

LDBC Task Forces

- Semantic Publishing Benchmark Task Force
  - Develops industry-grade RDF benchmark
- Social Network Benchmark Task Force
  - Develops benchmark for graph data management systems
  - Broad coverage: three workloads
- Graph Analytics Task Force
  - Spin-off from the SNB task force
- Graph Query Language Task Force
  - Not strictly about benchmarking
  - Studies features of graph database query languages

Semantic Publishing Benchmark (SPB)



SPB scope

- The scenario involves a media/ publisher organization that maintains semantic metadata about its Journalistic assets (articles, photos, videos, papers, books, etc), also called Creative Works
- The Semantic Publishing Benchmark simulates:
  - Consumption of RDF metadata (Creative Works)
  - Updates of RDF metadata, related to Annotations
- Aims to be an industrially mature RDF database benchmark (SPARQL1.1, some reasoning, text and GIS queries, backup&restore)

Social Network Benchmark (SNB)

- Intuitive: everybody knows what a SN is
  - Facebook, Twitter, LinkedIn, …
- SNs can be easily represented as a graph
  - Entities are the nodes (Person, Group, Tag, Post, ...)
  - Relationships are the edges (Friend, Likes, Follows, …)
- Different scales: from small to very large SNs
  - Up to billions of nodes and edges
- Multiple query needs:
  - interactive, analytical, transactional
- Multiple types of uses:
  - marketing, recommendation, social interactions, fraud detection, ...

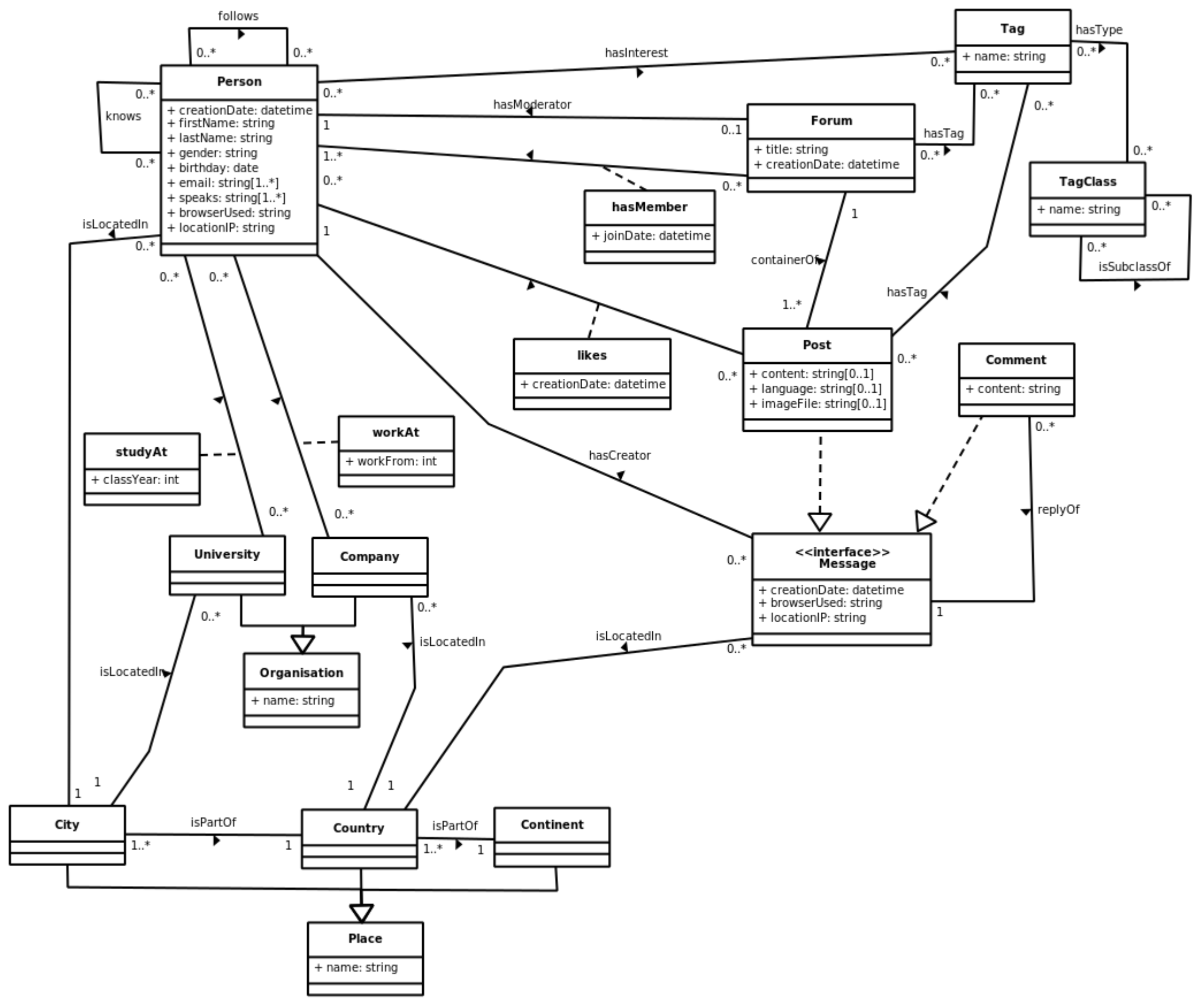
Audience

- For **developers** facing graph processing tasks
  - recognizable scenario to compare merits of different products and technologies
- For **vendors** of graph database technology
  - checklist of features and performance characteristics
- For **researchers**, both industrial and academic
  - challenges in multiple choke-point areas such as graph query optimization and (distributed) graph analysis

Data Schema

- Specified in UML for portability
  - Classes
  - associations between classes
  - Attributes for classes and associations
- Some of the relationships represent dimensions
  - Time (Y,QT,Month,Day)
  - Geography (Continent,Country,Place)
- Data Formats
  - CSV
  - RDF (Turtle + N3)

Social Network Benchmark: schema



Benchmark Workloads

- **Interactive**: tests throughput running short queries while consistently handling concurrent updates
  - *Show all photos posted by my friends that I was tagged in*

More details will follow in the second lecture

- **Business Intelligence**: consists of complex structured queries for analyzing online behavior
  - *Influential people the topic of open source development?*

Draft queries available on ldbcouncil.org website (deliverable D2.2.4) & github

- **Graph Analytics**: tests the functionality and scalability on most of the data as a single operation
  - *PageRank, Shortest Path(s), Community Detection*

LDBC

GRADES2015 “Graphalytics: A Big Data Benchmark for Graph-Processing Platforms ” - Mihai Capota, Tim Hegeman, Alexandru Iosup(TU Delft); Arnau Prat (UPC), Orri Erling (OpenLink Technologies), Peter Boncz (CWI)

Interactive (On-line) Workload

- Test online ACID features and scalability
- The system under test is expected to run in a steady state, providing durable storage
- Updates are typically small
- Updates will conflict a small percentage of the time
- Queries typically touch a small fraction of the database

CWI Centrum Wiskunde & Informatica

SNB Interactive Workload

Q1. *Extract description of friends with a given name* Given a person's `firstName`, return up to 20 people with the same first name, sorted by increasing distance (max 3) from a given `person`, and for people within the same distance sorted by last name. Results should include the list of workplaces and places of study.

Q2. *Find the newest 20 posts and comments from your friends.* Given a start `Person`, find (most recent) Posts and Comments from all of that Person's friends, that were created before (and including) a given `Date`. Return the top 20 Posts/Comments, and the Person that created each of them. Sort results descending by creation date, and then ascending by Post identifier.

Q3. *Friends within 2 steps that have recently traveled to countries X and Y.* Find friends and friends of friends of a given `Person` who have made a post or a comment in the foreign `CountryX` and `CountryY` within a specified period of `DurationInDays` after a `startDate`. Return top 20 Persons, sorted descending by total number of posts.

Q4. *New Topics.* Given a start `Person`, find the top 10 most popular Tags (by total number of posts with the tag) that are attached to Posts that were created by that Person's friends. Only include Tags that were attached to Posts created within a given `time interval`, and that were never attached to Posts created before this interval.

Q5. *New groups.* Given a start `Person`, find the top 20 Forums which that Person's friends and friends of friends became members of after a given `Date`. Sort results descending by the number of Posts in each Forum that were created by any of these Persons.

find the other Tags that occur together with this Tag on Posts that were created by start Person's friends and friends of friends. Return top 10 Tags, sorted descending by the count of Posts that were created by these Persons, which contain both this Tag and the given Tag.

Q7. *Recent likes.* For the specified `Person` get the most recent likes of any of the person's posts, and the latency between the corresponding post and the like. Flag Likes from outside the direct connections. Return top 20 Likes, ordered descending by creation date of the like.

Q7. *Recent likes.* For the specified `Person` get the most recent likes of any of the person's posts, and the latency between the corresponding post and the like. Flag Likes from outside the direct connections. Return top 20 Likes, ordered descending by creation date of the like.

Q8. *Most recent replies.* This query retrieves the 20 most recent reply comments to all the posts and comments of `Person`, ordered descending by creation date.

Q9. *Latest Posts.* Find the most recent 20 posts and comments from all friends, or friends-of-friends of `Person`, but created before a `Date`. Return posts, their creators and creation dates, sort descending by creation date.

Q10. *Friend recommendation.* Find a friend of a friend who posts much about the interests of `Person` and little about topics that are not in the interests of the user. The search is restricted by the candidate's `horoscopeSign`. Returns 10 Persons for whom the difference between the total number of their posts about the interests of the specified user and the total number of their posts that are not in the interests of the user, is as large as possible. Sort the result descending by this difference.

Q11. *Job referral.* Find a friend of the specified `Person`, or a friend of her friend (excluding the specified person), who has long worked in a company in a specified `Country`. Sort ascending by start date, and then ascending by person identifier. Top 10 result should be shown.

Q12. *Expert Search.* Find friends of a `Person` who have replied the most to posts with a tag in a given `TagCategory`. Count the number of these reply Comments, and collect the Tags that were attached to the Posts they replied to. Return top 20 persons, sorted descending by number of replies.

Q13. *Single shortest path.* Given `PersonX` and `PersonY`, find the shortest path between them in the subgraph induced by the Knows relationships. Return the length of this path.

Q14. *Weighted paths.* Given `PersonX` and `PersonY`, find all weighted paths of the shortest length between them in the subgraph induced by the Knows relationship. The weight of the path takes into consideration amount of Posts/Comments exchanged.

Example: Q5 - SPARQL

```
select ?group count (*)
where {
   {select distinct ?fr
    where {
        {%Person% snvoc:knows ?fr.} union
        {%Person% snvoc:knows ?fr2.
         ?fr2 snvoc:knows ?fr. filter (?fr != %Person%)}
    }
   } .
   ?group snvoc:hasMember ?mem . ?mem snvoc:hasPerson ?fr .
   ?mem snvoc:joinDate ?date . filter (?date >= "%Date0%"^^xsd:date) .
   ?post snvoc:hasCreator ?fr . ?group snvoc:containerOf ?post
}
group by ?group
order by desc(2) ?group
limit 20
```

Example: Q5 - Cypher

```
MATCH (person:Person)-[:KNOWS*1..2]-(friend:Person)
WHERE person.id={person_id}
MATCH (friend)<-[membership:HAS_MEMBER]-(forum:Forum)
WHERE membership.joinDate>{join_date}
MATCH (friend)<-[:HAS_CREATOR]-(comment:Comment)
WHERE (comment)-[:REPLY_OF*0..]->(:Comment)-[:REPLY_OF]->(:Post)<-
   [:CONTAINER_OF]-(forum)
RETURN forum.title AS forum, count(comment) AS commentCount
ORDER BY commentCount DESC
MATCH (person:Person)-[:KNOWS*1..2]-(friend:Person)
WHERE person.id={person_id}
MATCH (friend)<-[membership:HAS_MEMBER]-(forum:Forum)
WHERE membership.joinDate>{join_date}
MATCH (friend)<-[:HAS_CREATOR]-(post:Post)<-[:CONTAINER_OF]-(forum)
RETURN forum.title AS forum, count(post) AS postCount
ORDER BY postCount DESC
```

Example: Q5 - Sparksee

```
v.setLongVoid(personId);
long personOID = graph.findObject(personId, v);
Objects friends = graph.neighbors(personOID, knows, EdgesDirection.Outgoing);
Objects allFriends = graph.neighbors(friends, knows, EdgesDirection.Outgoing);
allFriends.union(friends);
allFriends.remove(personOID);
friends.close();
Objects members = graph.explode(allFriends, hasMember, EdgesDirection.Ingoing);
v.setTimestampVoid(date);
Objects candidate = graph.select(joinDate, Condition.GreaterEqual, v, members);
Objects finalSelection = graph.tails(candidate);
candidate.close();
members.close();
Objects posts = graph.neighbors(allFriends, hasCreator, EdgesDirection.Ingoing);
ObjectsIterator iterator = finalSelection.iterator();
while (iterator.hasNext()) {
      long oid = iterator.next();
      Container c = new Container();
      Objects postsGroup = graph.neighbors(oid, containerOf, EdgesDirection.Outgoing);
      Objects moderators = graph.neighbors(oid, hasModerator, EdgesDirection.Outgoing);
      long moderatorOid = moderators.any();
      moderators.close();
      Objects postsModerator = graph.neighbors(moderatorOid, hasCreator, EdgesDirection.Ingoing);
      postsGroup.difference(postsModerator);
      postsModerator.close();
      postsGroup.intersection(posts);
      long count = postsGroup.size();
      if (count > 0) {
        graph.getAttribute(oid, forumId, v);
        c.row[0] = db.getForumURI(v.getLong());
        c.compare2 = String.valueOf(v.getLong());
        c.row[1] = String.valueOf(count);
        c.compare = count;
        results.add(c);
      }
      postsGroup.close()
}
```

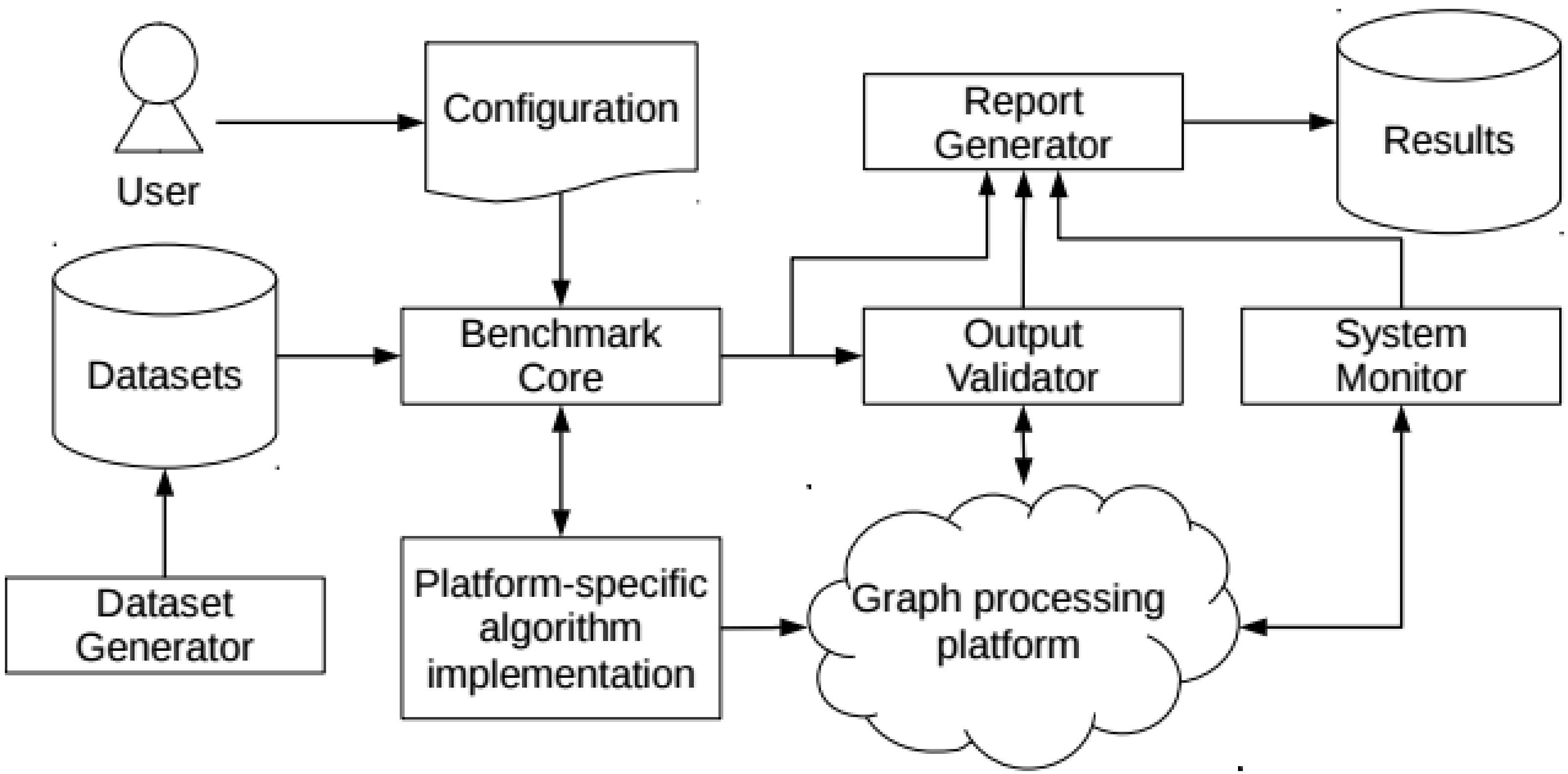
Business Intelligence Workload

- The workload stresses query execution and optimization
- Queries typically touch a large fraction of the data
- The queries are concurrent with trickle load
- The queries touch more data as the database grows

Graph Analytics Workload (Graphalytics)

- The analytics is done on most of the data in the graph as a single operation
- The analysis itself produces large intermediate results
- The analysis transactional: no need for isolation from possible concurrent updates

Graphalytics Architecture



Graphalytics Algorithms

- general statistics (STATS)
  - counts the numbers of vertices and edges in the graph and computes the mean local clustering coefficients
- breadth-first search (BFS)
  - traverses the graph starting from a seed vertex, visiting first all the neighbors of a vertex before moving to the neighbors of the neighbors.
- connected components (CONN) algorithm
  - determines for each vertex the connected component it belongs to.
- community detection (CD) algorithm
  - detects groups of nodes that are connected to each other stronger than they are connected to the rest of the graph
- graph evolution (EVO)
  - predicts the evolution of the graph according to the “forest fire” model

Systems

- **Graph database** systems
  - e.g. Neo4j, InfiniteGraph, DEX, Titan
- **Graph programming frameworks**
  - e.g. Giraph, Signal/Collect, Graphlab, Green Marl, Grappa
- **RDF** database systems
  - e.g. OWLIM, Virtuoso, BigData, Jena TDB, Stardog, Allegrograph
- **Relational** database systems
  - e.g. Postgres, MySQL, Oracle, DB2, SQLServer, Virtuoso, MonetDB, Vectorwise, Vertica
- **noSQL** database systems
  - e.g. HBase, REDIS, MongoDB, CouchDB, or even MapReduce systems like Hadoop and Pig

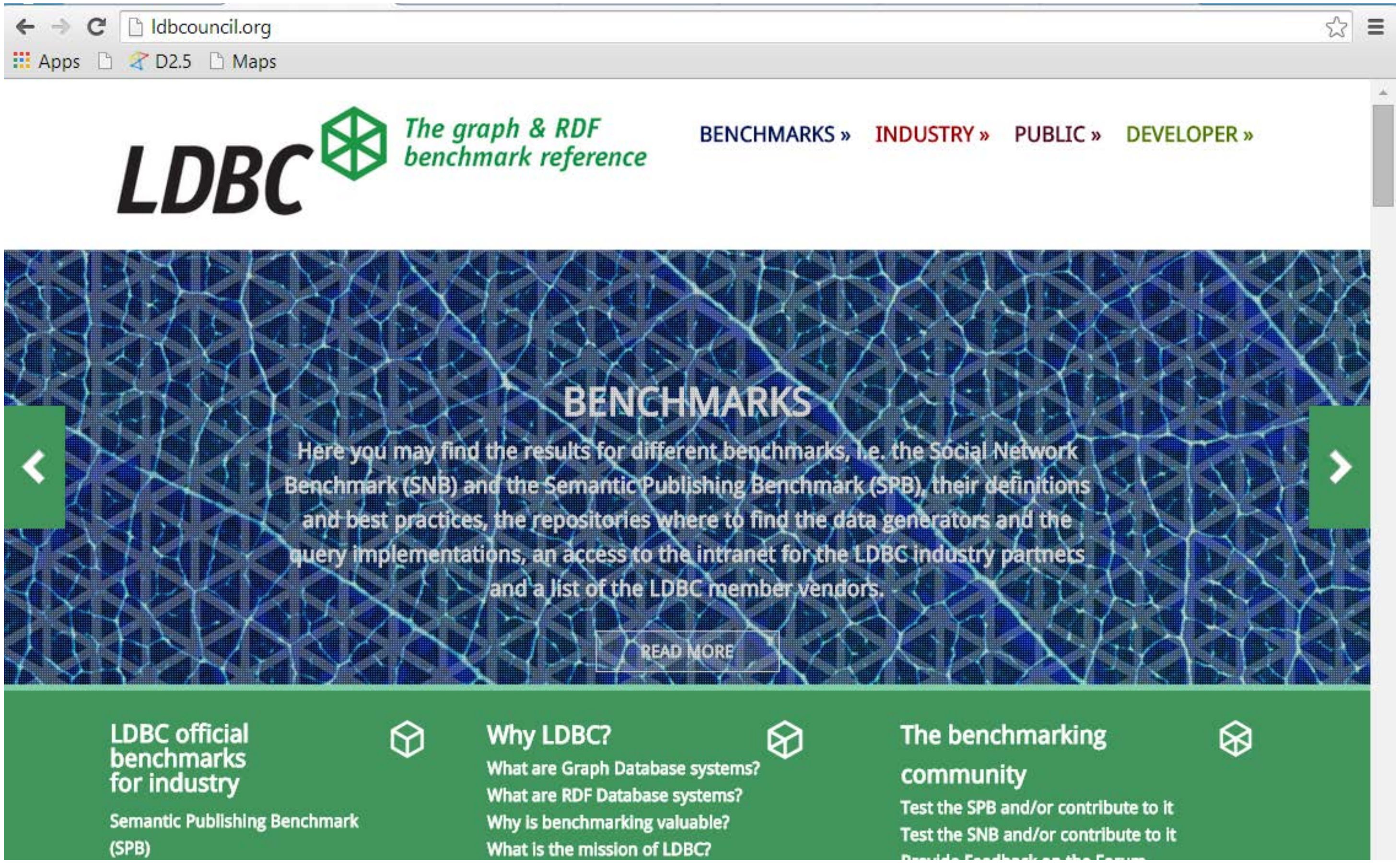
Workloads by system

System	Interactive	Business Intelligence	Graph Analytics
Graph databases	Yes	Yes	Maybe
Graph programming frameworks	-	Yes	Yes
RDF databases	Yes	Yes	-
Relational databases	Yes	Yes	Maybe, by keeping state in temporary tables, and using the functional features of PL-SQL
NoSQL Key-value	Maybe	Maybe	-
NoSQL MapReduce	-	Maybe	Yes

More Information

http://www.ldbcouncil.org **http://github.com/ldbc**

Blogs
Specifications
Early Result FDRs
Videos of TUC talks
Developer info
Code, Issue Tracking



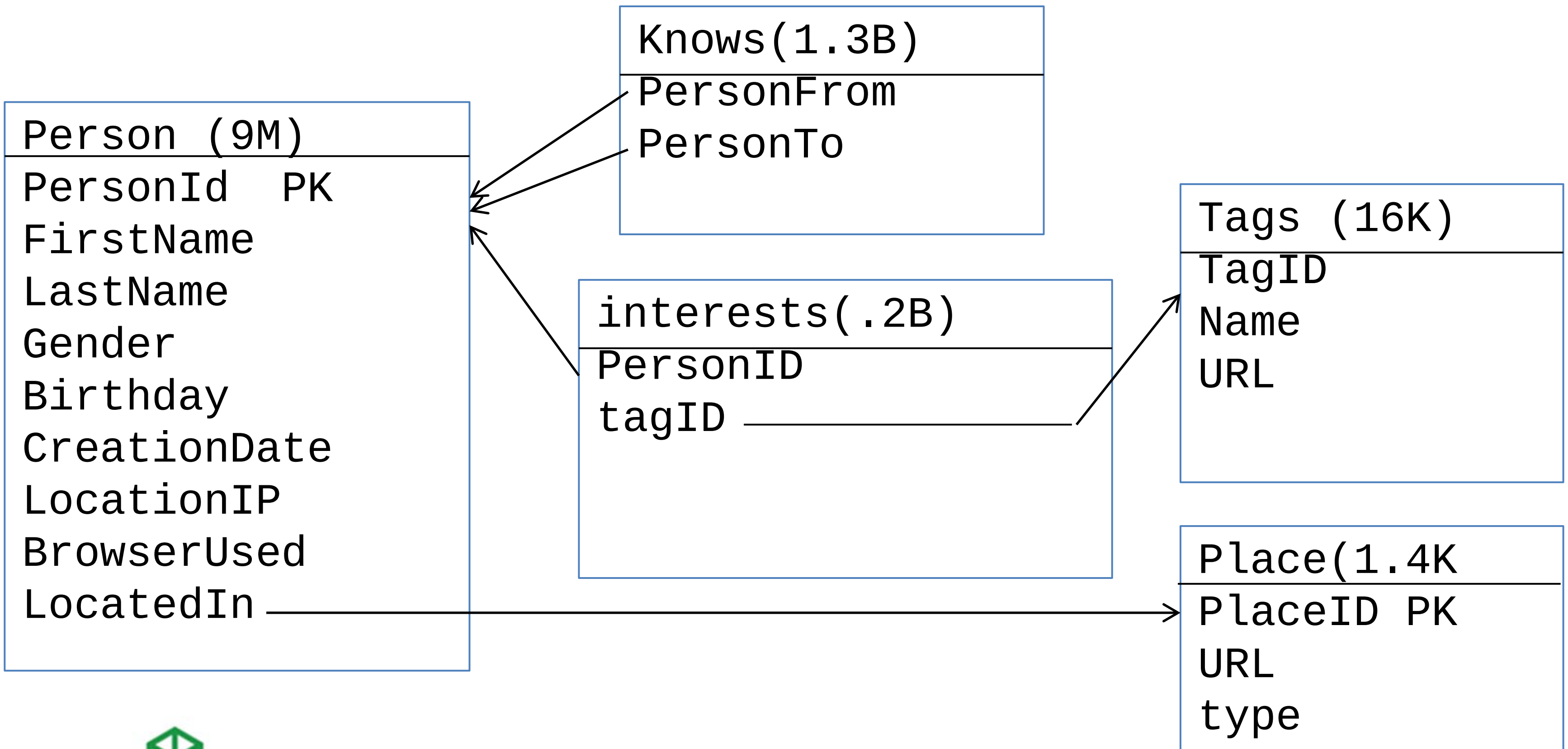
SNB Challenge: Querying a Social Graph

LDBC SNB Data generator

- Synthetic dataset available in different scale factors
  - SF100 🡐 for quick testing
  - SF3000 🡐 the real deal
- Very complex graph
  - Power laws (e.g. degree)
  - Huge Connected Component
  - Small diameter
  - Data correlations
    Chinese have more Chinese names
  - Structure correlations
    Chinese have more Chinese friends

CSV file schema

- See: http://wikistats.ins.cwi.nl/lsde-data/practical_1
- Counts for sf3000 (total 37GB)



The Query

- The marketeers of a social network have been data mining the musical preferences of their users. They have built statistical models which predict given an interest in say artists A2 and A3, that the person would also like A1 (i.e. rules of the form: A2 and A3 ➔ A1). Now, they are commercially exploiting this knowledge by selling targeted ads to the management of artists who, in turn, want to sell concert tickets to the public but in the process also want to expand their artists' fanbase.
- The ad is a suggestion for people who already are interested in A1 to buy concert tickets of artist A1 (with a discount!) as a birthday present for a friend ("who we know will love it" - the social network says) who lives in the same city, who is not yet interested in A1 yet, but is interested in other artists A2, A3 and A4 that the data mining model predicts to be correlated with A1.

The Query

For all persons P :

- *who have their birthday on or in between D1..D2*
- *who do not like A1 yet*

  we give a score of

  - *1 for liking any of the artists A2, A3 and A4 and*
  - *0 if not*

  the final score, the sum, hence is a number between 0 and 3.

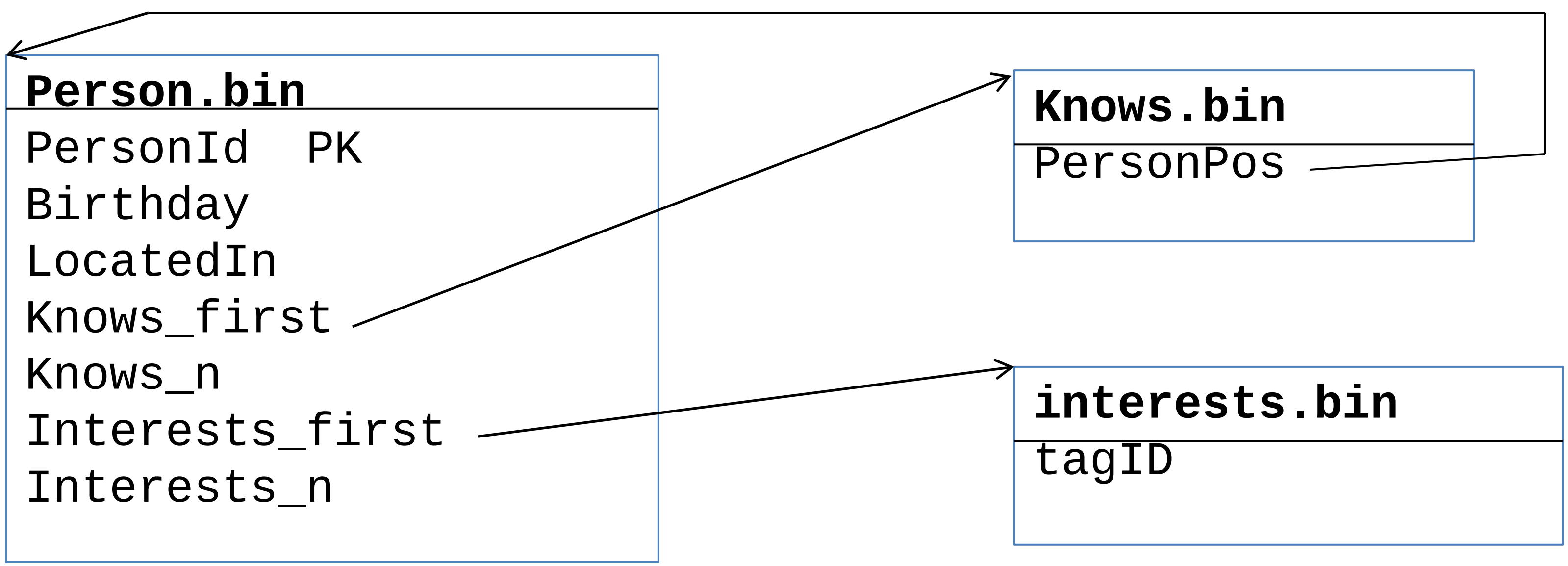
  Further, we look for friends F:

  - *Where P and F who know each other mutually*
  - *Where P and F live in the same city and*
  - *Where F already likes A1*

    The answer of the query is a table (score, P, F) with only scores > 0

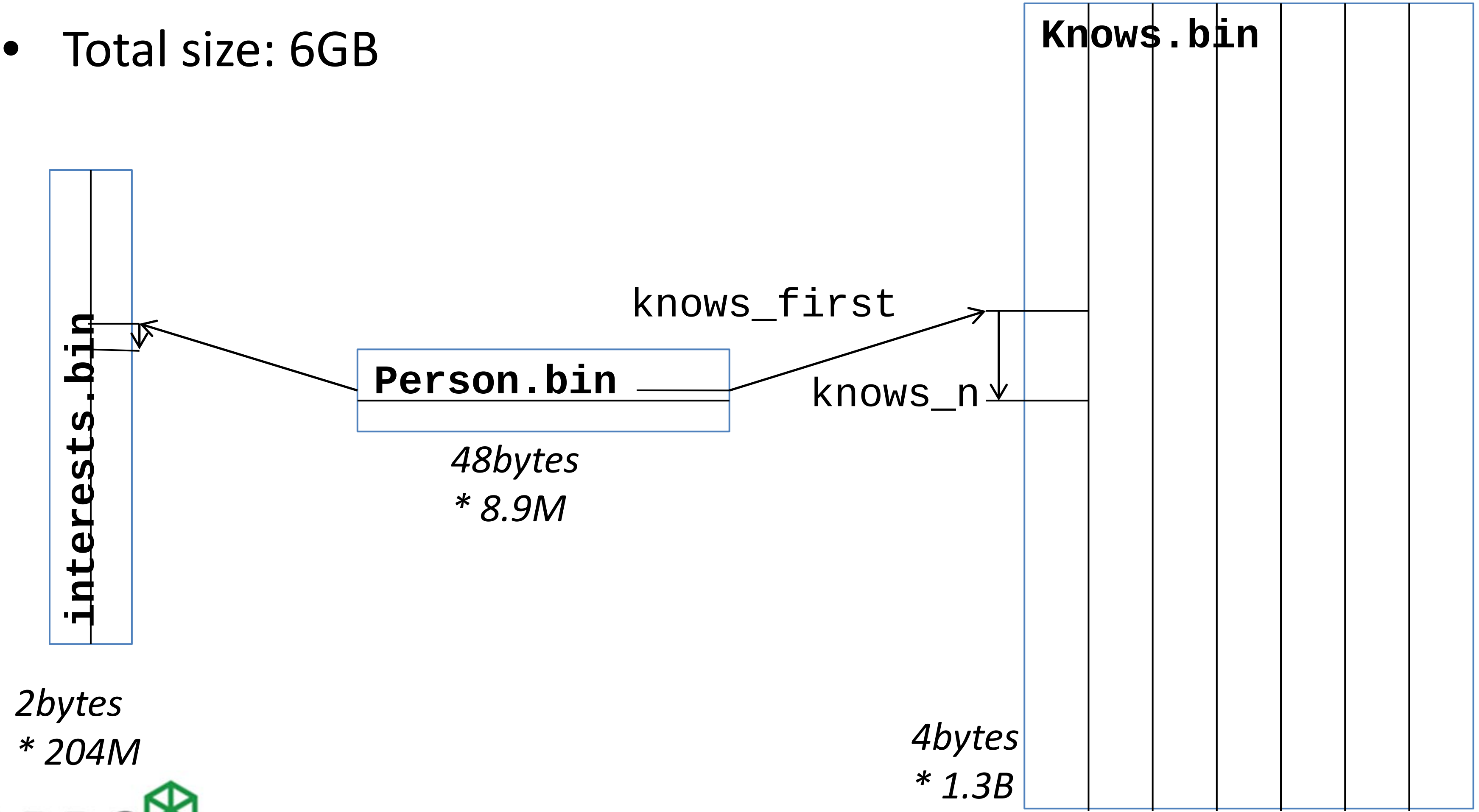
Binary files

- Created by “loader” program in example github repo
- Total size: 6GB



What it looks like

- Created by “loader” program in example github repo
- Total size: 6GB



The Naïve Implementation

The "cruncher" program

Go through the persons P sequentially

- *counting how many of the artists A2,A3,A4 are liked as the score*

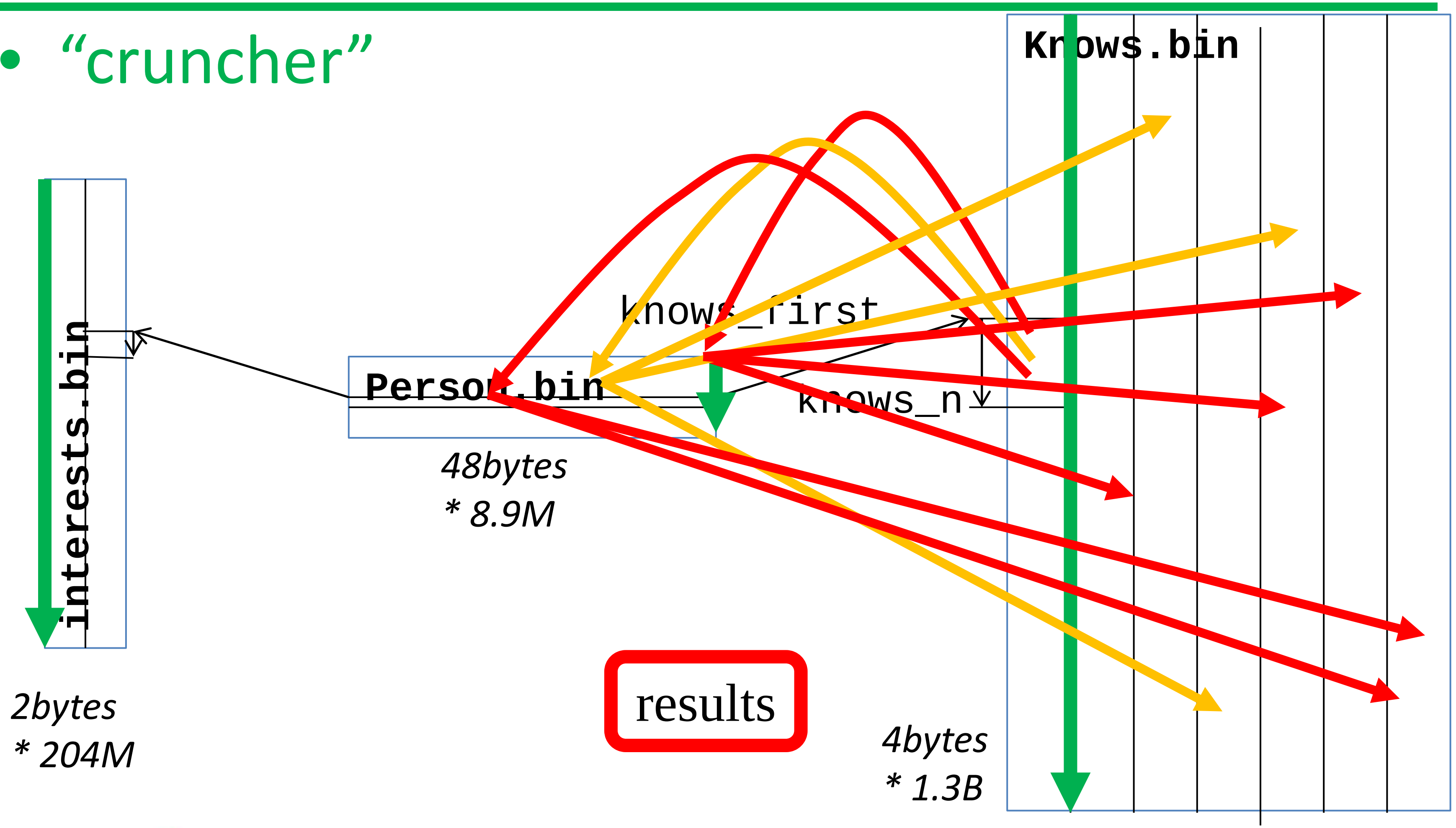
  for those with score>0:
  - *visit all persons F known to P.*

    For each F:
    - *checks on equal location*
    - *check whether F already likes A1*
    - *check whether F also knows P*

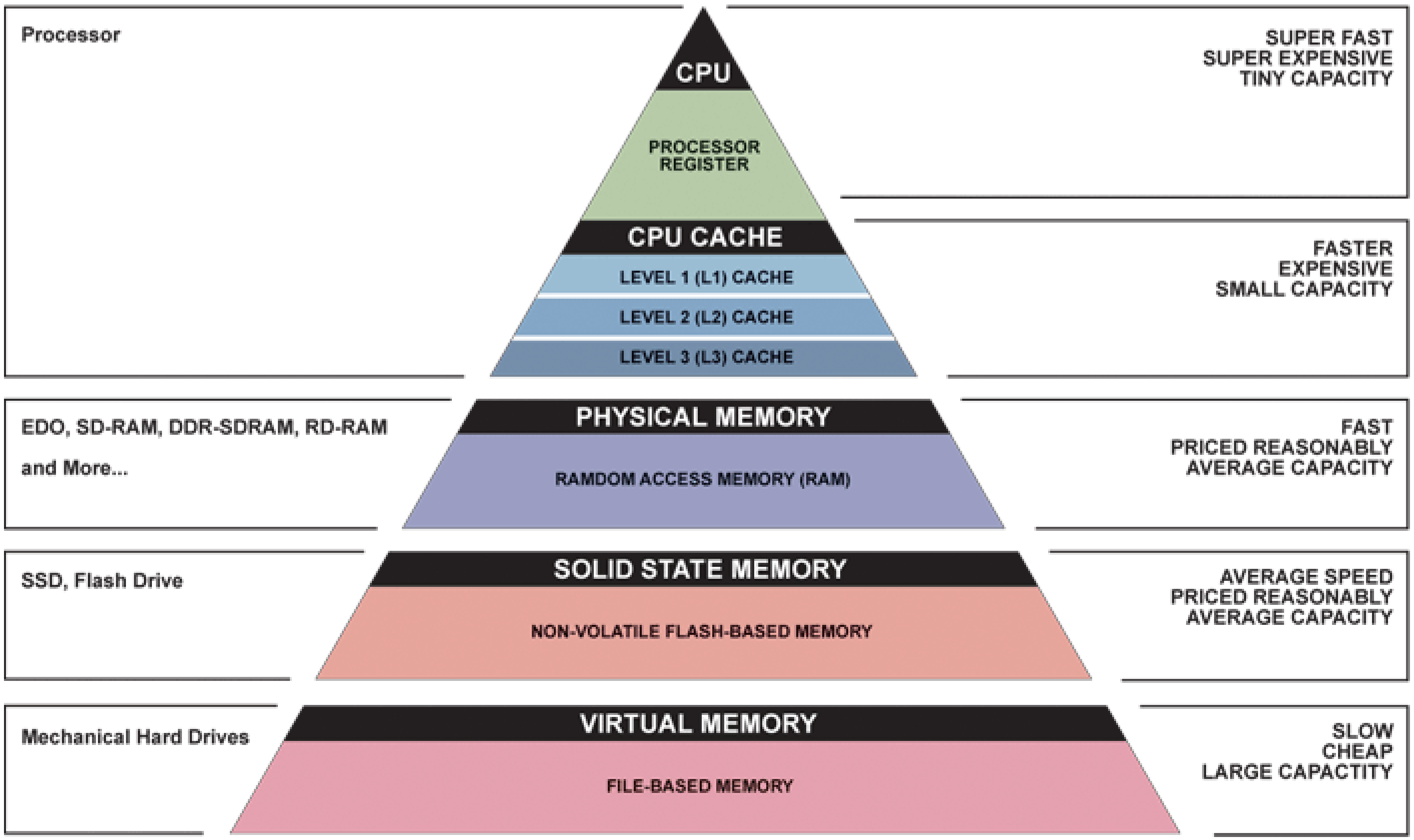
    if all this succeeds (score,P,F) is added to a result table.

Naïve Query Implementation

- "cruncher"

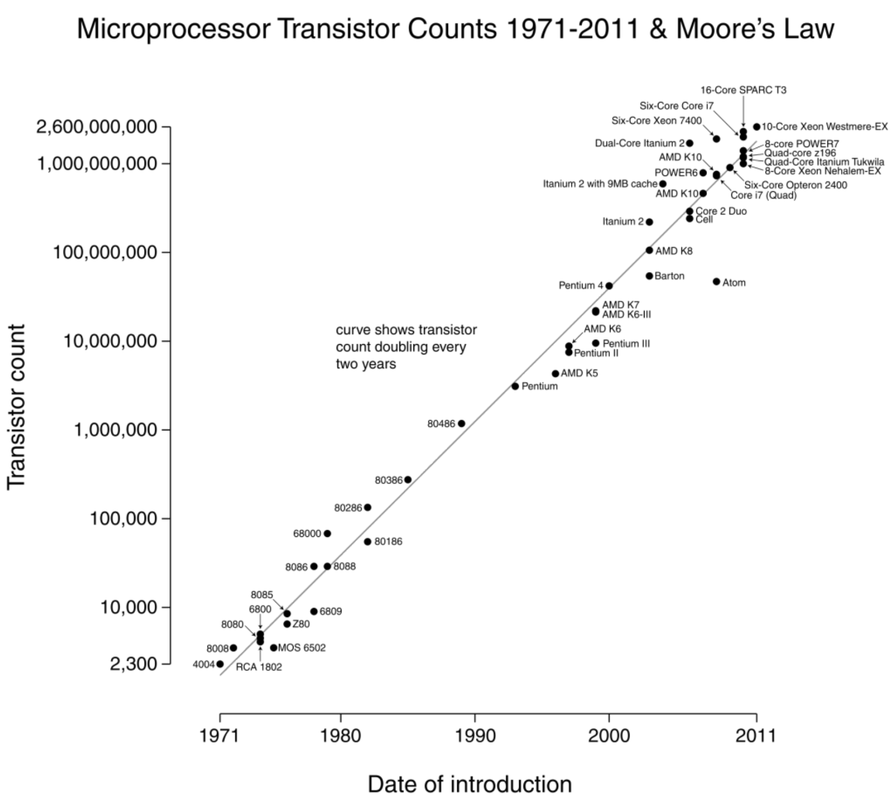


Memory Hierarchy

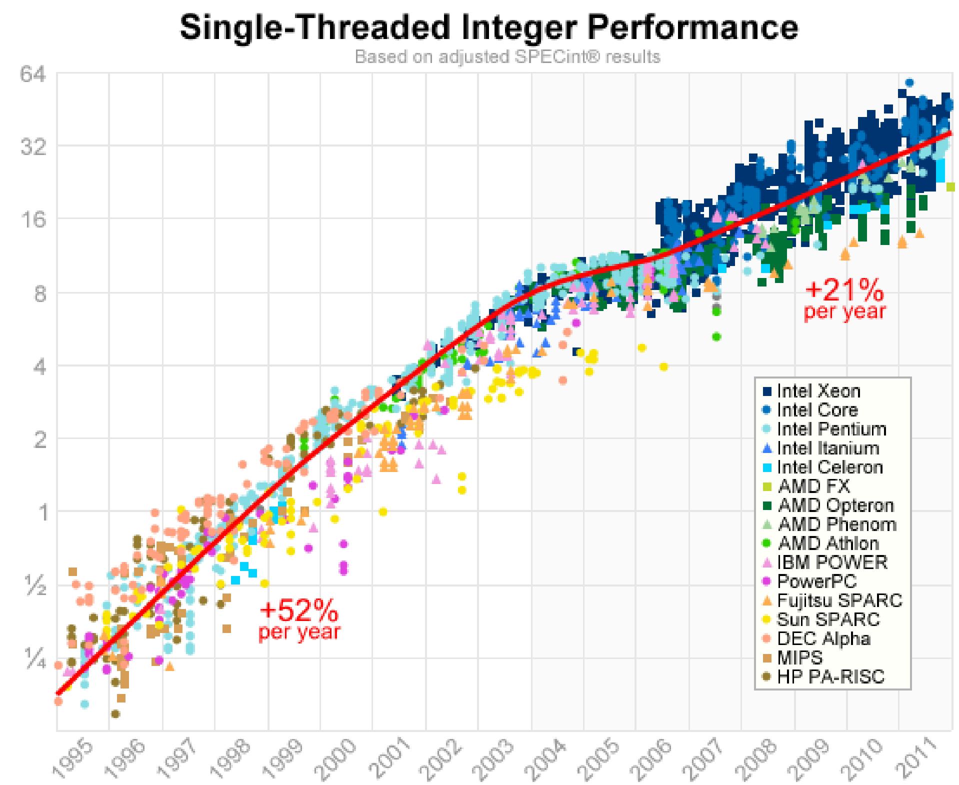


▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Hardware Progress

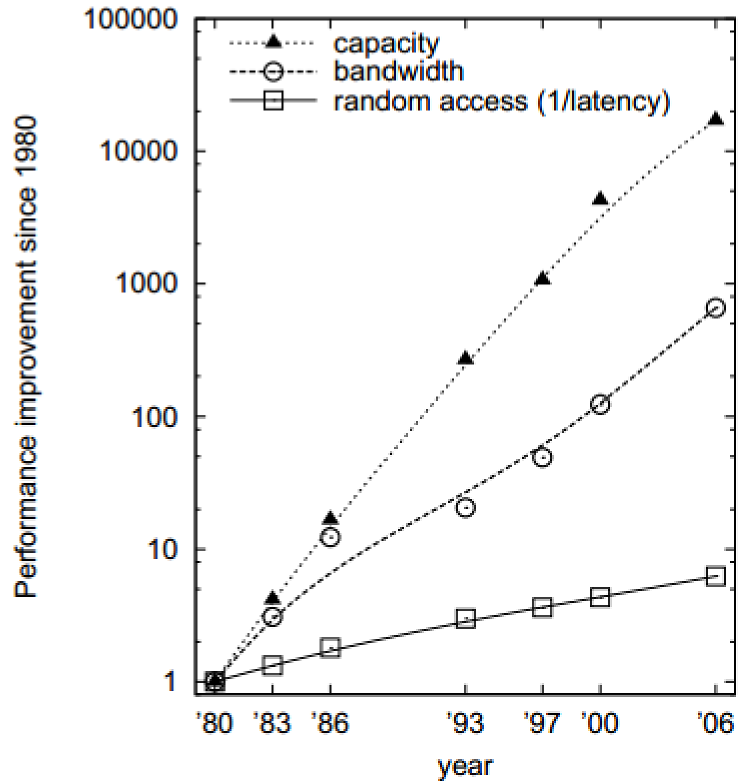


Transistors

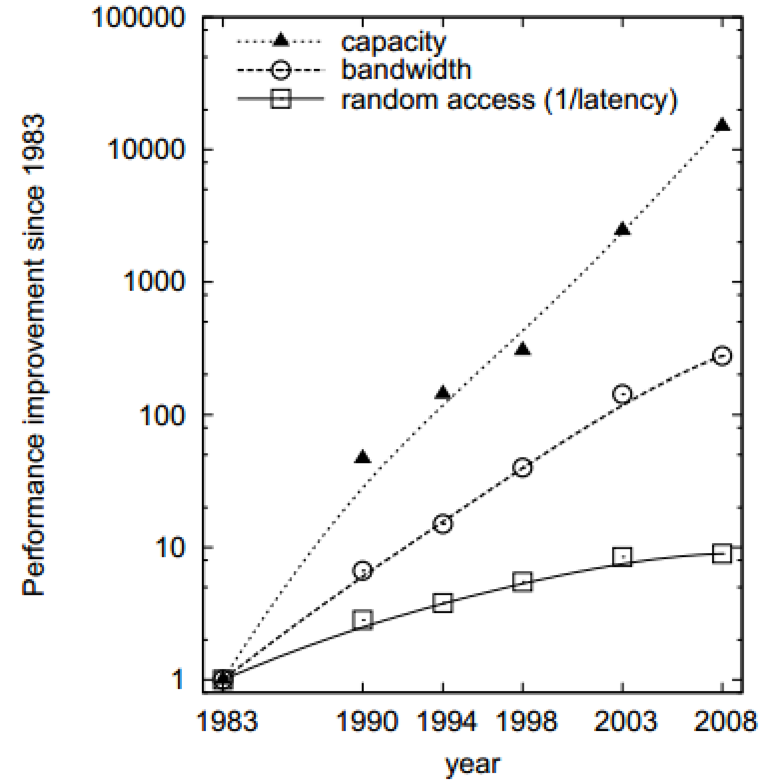


CPU performance

RAM,Disk Improvement Over the Years

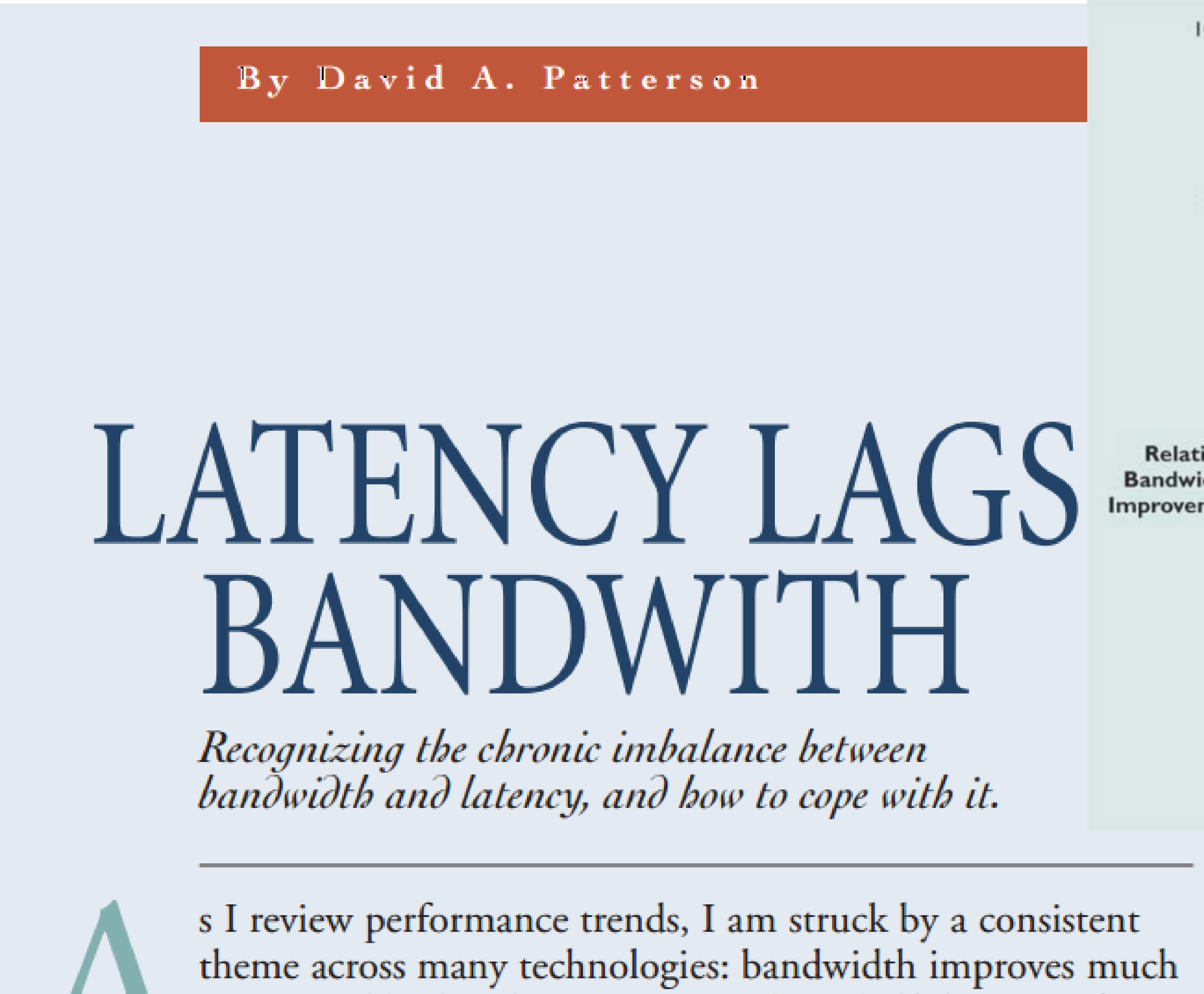


RAM



Magnetic Disk

Latency Lags Bandwidth

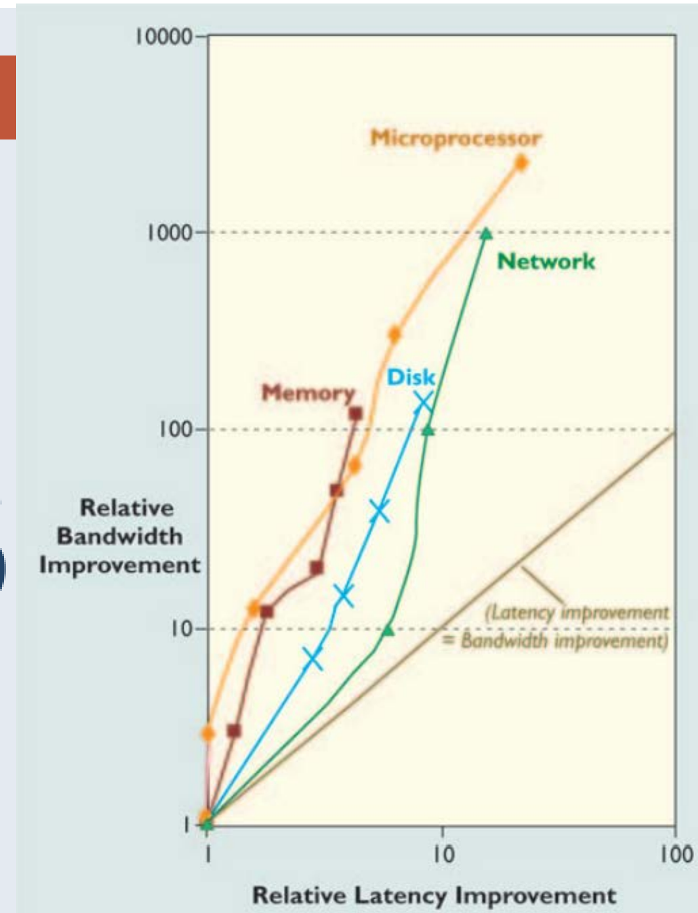


By David A. Patterson

LATENCY LAGS BANDWITH

Recognizing the chronic imbalance between bandwidth and latency, and how to cope with it.

As I review performance trends, I am struck by a consistent theme across many technologies: bandwidth improves much



Geeks on Latency

jboner / **latency.txt**

Created on 31 May 2012

Latency Numbers Every Programmer Should Know

latency.txt

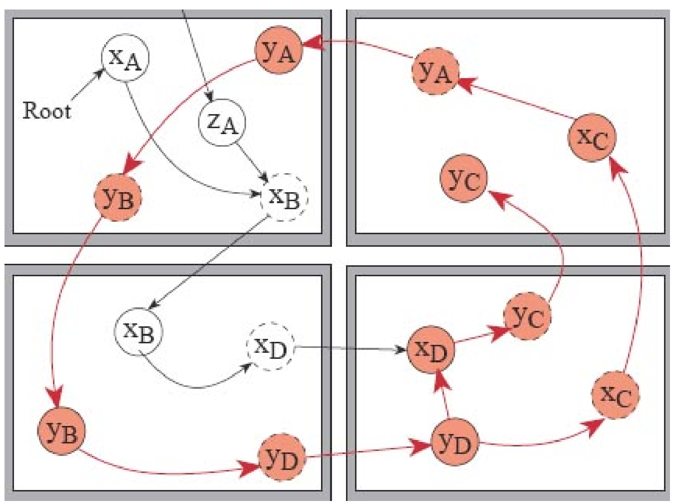
```
Latency Comparison Numbers
--------------------------
L1 cache reference                            0.5 ns
Branch mispredict                             5   ns
L2 cache reference                            7   ns                 14x L1 cache
Mutex lock/unlock                            25   ns
Main memory reference                       100   ns                 20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy              3,000   ns
Send 1K bytes over 1 Gbps network        10,000   ns    0.01 ms
Read 4K randomly from SSD*              150,000   ns    0.15 ms
Read 1 MB sequentially from memory      250,000   ns    0.25 ms
Round trip within same datacenter       500,000   ns    0.5  ms
Read 1 MB sequentially from SSD*      1,000,000   ns    1    ms  4X memory
Disk seek                            10,000,000   ns   10    ms  20x datacenter roundtrip
Read 1 MB sequentially from disk     20,000,000   ns   20    ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA     150,000,000   ns  150    ms

Notes
-----
1 ns = 10-9 seconds
1 ms = 10-3 seconds
* Assuming ~1GB/sec SSD
```

Sequential Access Hides Latency

- Sequential RAM access
  - CPU prefetching: multiple consecutive cache lines being requested concurrently
- Sequential Magnetic Disk Access
  - Disk head moved once
  - Data is streamed as the disk spins under the head
- Sequential Network Access
  - Full network packets
  - Multiple packets in transit concurrently

Consequences For Algorithms

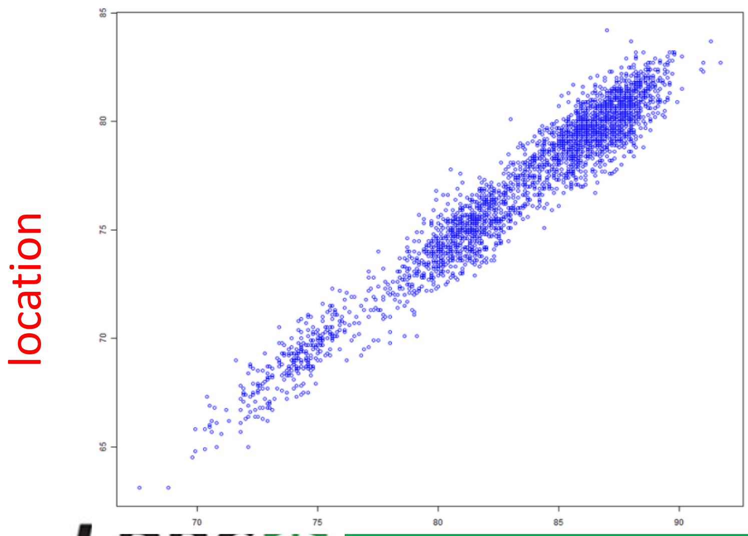
- Analyze the main data structures
  - How big are they?
    - Are they bigger than RAM?
    - Are they bigger than CPU cache (a few MB)?
  - How are they laid out in memory or on disk?
    - One area, multiple areas?

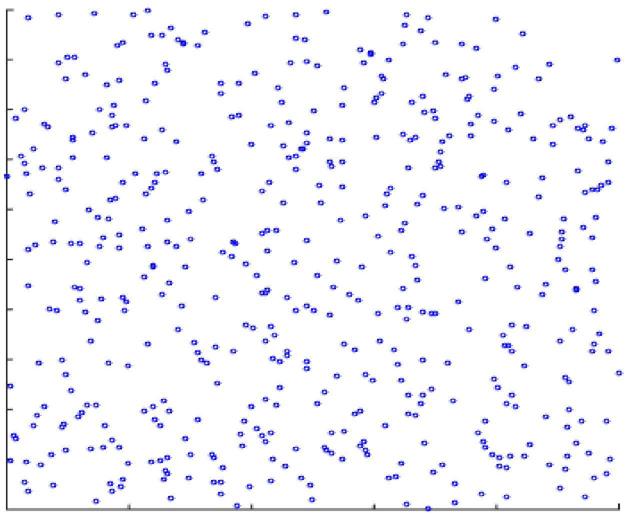


Java Object Data Structure
vs
memory pages (or cache lines)

Consequences For Algorithms

- Analyze your access patterns
  - Sequential: you're OK
  - Random: it better fit in cache!
    - What is the access granularity?
    - Is there temporal locality? Is there spatial locality?





Improving Bad Access Patterns

- Minimize Random Memory Access
  - Apply filters first. Less accesses is better.
- Denormalize the Schema
  - Remove joins/lookups, add looked up stuff to the table (but.. makes it bigger)
- Trade Random Access For Sequential Access
  - perform a 100K random key lookups in a large table
    ➔ put 100K keys in a hash table, then
    scan table and lookup keys in hash table
- Try to make the randomly accessed region smaller
  - Remove unused data from the structure
  - Apply data compression
  - Cluster or Partition the data (improve locality) …hard for social graphs

Naïve Query Implementation

- "cruncher"