```
date(); // 1509332838, seconds since midnight 1-1-1970,
// October 30, 2017

// Creating a presentation for Alan Labouseur
bigtable = “Bigtable: A Distributed Storage System for
Structured Data”;
comparison = “A Comparison of Approaches to
Large-Scale Data Analysis”;
stonebraker = “One Size Fits All - An Idea Whose
Time Has Come and Gone (2005)”
this = summarize(bigtable, comparison, stonebraker);
print(name); // “Tom Magnusson”
print(authors(bigtable));
// “Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A.
// Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber”
print(authors(comparison));
// “Andre Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. Dewitt,
// Samuel Madden, Michael Stonebraker”
print(authors(stonebraker));
// “Michael Stonebraker, Ugur Cetintemel”
```

# mainIdeas(bigtable);

- Widely applicable, scalable, high performance, high availability **data store***
- ***not quite a database**
  - Different interface
  - No relational model! (Oh my Codd!)
    - Dynamic schema
- Emphasis on distribution and client specific optimizations
- Focus on availability and consistency over speed
- Has
  - a unique conceptual model
  - An API for clients to interface with
- Built on top of other Google infrastructure such as Chubby and Google File System
- As of 2006 ran over **60 production Google apps**, such as Google * (**Earth, Finance, Analytics**)

## bigtable.implementationDetails

- Model
  - “Sparse, distributed, persistent multidimensional sorted map” (2)
  - **(row: String, column: String, time:int64) → String**
  - Values in a (row, column) have many versions, each version timestamped
  - **Column Families**: column key groupings, prefixed like *family:colname*
  - **Tablet**: small groups of rows that are similar to each other, unit of distribution, makes reads of that kind of data faster
- API
  - **RowMutation** for writes, **Scanner** for reads
  - MapReduce can run on it (not necessary though)
- Building Blocks
  - **Google File System** for logs and data files
    - **SSTable** as file format, broken into **blocks**, index of blocks at end of file
  - Distributed Lock service called **Chubby**
    - Coordinates distribution model: one **master server**, many **tablet servers**
    - Tablets grow and split across tablet servers, master takes care of failures
- **Compaction**: serialize most recent operations into an SSTable (memory to disk)
- **Locality group**: many column families into group, efficient access
- **Compression**: Custom client compression on block level, surprisingly good
- Use **caching** to optimize reads, **bloom filters** to reduce disk accesses

## this = analysis(bigtable);

## this.pros();

- **Performant**: strong if Google's using it on many production applications.
  - Makes use of many available optimization patterns, including caching, bloom filters, transaction log partitioning.
- **Understandable conceptual model**: intuitively understand **tablet** (a small table) and **column families** (columns that are related to each other)
- **Scalable**: cluster-wide reads/writes per second increased in almost every metric when adding new servers, even up to 500 servers.
- **Failure tolerant**: tablet servers in a cluster can die, that server's tablets can be reconstructed via log files in a reasonably short amount of time.

## this.cons();

- **Coupled**: Bigtable is coupled to the underlying systems such as Google File System and Chubby.
- **Underlying systems are problematic**: the authors made it seem like Chubby gave them problems, especially in reliability. Google File System occasionally gave them I/O bottlenecks.
- **No multi-row transactions**: Clients are required to batch a bunch of single-row requests together; this seems less performant than it could be (optimizing for multi-row requests), though the authors address this concern.
- **No mention of SQL support**: SQL doesn't fit the Bigtable model.

## mainIdeas(comparison);

- **MapReduce**: Key-Value data is 1. Mapped into intermediate form 2. Reduced (combined, aggregated, etc.) in some way to a final output. Its open source implementation is Hadoop.
- **Distributed DBMS**: the classic relational model working across many nodes in a cluster.
- Compare **MapReduce** (Hadoop) with **distributed DBMS** (DBMS-X and Vertica); which is better?
- **Distributed DBMS beat Hadoop performance-wise**: Hadoop performed poorly once loaded with data, compared to the distributed DBMS which were significantly faster in many metrics when manipulating data.
- **Hadoop was easier to configure**: it was faster and easier to get Hadoop up and running than the DBMSs.
- **Hadoop was harder to manipulate**: Users could write simple SQL queries as if on a non-distributed DBMS; Hadoop required users to pay attention to MapReduce specific minutiae
- **Hadoop is more failure-tolerant**: though this failure tolerance comes at an indeterminate performance cost
- **Both are acquiring new tools**: MapReduce have abstractions like Pig and Hive. Distributed DBMS are continuing to grow large third party tools and continue to benefit from SQL's simplicity

## comparison.implementationDetails

- Performed several tests
  - **Grep test**: searching for three letter string in lots of rows of random data; the three letter string is found once-per 10,000 rows
    - **Load times**: Hadoop remained consistently fast, despite adding nodes to the cluster; the DBMSs appeared to be quite slow
    - **Task Execution**: Hadoop was slower, especially as more nodes were added to a cluster, DBMS were much faster
      - Hadoop took a long time to get all of its functions operating across a cluster, indicative of how slow it is
  - **HTML Data test**: store HTML and data about a file, similar to web crawler db
    - **Load times**: Hadoop was pretty fast, DBMSs not so much
    - **Selection Task**: find a url above a certain pagerank
      - Hadoop was slow, RDBMs were faster
    - **Aggregation Task**: calculate ad revenue from pages
      - Hadoop requires Map and Reduce, was significantly slower than RDBMS
    - **Join Task**: Compute complex query on one table, joined together by complex query on another table
      - Whoa Hadoop was slow, required way too much processing
    - **UDF Task**: perform part of PageRank calculation
      - Hadoop was comparable to DBMSs

## analysis(comparison);

- **Fairness**: the authors did their best to give MapReduce (Hadoop) a fair shot at completing comparable tasks. They offered explanations for the differences in querying the data in Hadoop vs DBMSs.
    - The comparison between DBMSs is more valid than that between those and Hadoop; the DBMSs executed identical queries, while authors had to translate those queries into Hadoop, which leaves room for significant differences (a developer might implement a “query” in a different way to achieve the same thing).
- **Hadoop seems attractive because of quick set up**: the relational model can often scare newcomers because it requires thought to set up the correct way; key value store in Hadoop is simple and easy to load (but has significant performance costs)
- **This paper would be a great advertisement for Vertica**: it seemed Vertica performed the best and required the least set up headaches compared to DBMS-X.
- **The paper lacks DBMS diversity**: I would bet the implementation details of distributed DBMS systems would differ widely, which could impact performance. The paper was telling us more about DBMS-X and Vertica than distributed DBMSs in general.

## compare(bigtable, comparison);

- **Bigtable was about a store system**: it was a paper detailing the implementation of an existing system and the design decisions that went into it.
- **The Comparison paper was about… comparing**: it compared three systems, two of which were distributed DBMSs, the other was Hadoop, an open-source implementation of MapReduce.
- **Both had performance metrics**
  - The comparison paper's thesis was that Hadoop (MapReduce) is slower than distributed DBMSs, and most of its performance graphs displayed just that. Performance was the main feature of the paper.
  - Bigtable focused on its implementation from a conceptual standpoint, and used the performance metrics to support the "smart thinking" employed in the system's design. It focused more on consistency and availability than performance, though it is performant.

## mainIdeas(stonebraker);

- **Michael Stonbraker and Ugur Cetintemel's paper won the 10 Year Test of Time award**: they predicted the demise of the relational model in 2005; Stonebraker talks about how it has come true 10 years later
- 1970-2000: Relational DBMS can answer any question
- 2005: Stonebreaker notes that models different from the relational model are sprouting
- 2015: "One size fits none"
- **Data Warehouses**: column stores
- **Transaction processing (OLTP)**: main memory
- **NoSQL**: No standards, key-value, json stores, etc.
- **Analytics**: data scientist replace business analysts, move towards data science tools (rather than SQL), no clear winner (but arrays are important)
- **Streaming**: OLTP engines (main memory), S-Store
- **Graph Analytics**: No clear winner, simulate in a variety of models (none are relational)
- Lots of opportunity for innovation
- "Elephants" that were selling the relational model will have to transition
- SAP vs Oracle: SAP will do something to thwart Oracle efforts

## bigtable.advantages();

- **Google tested performance**: I would trust that this store works because Google's implemented it on many of its production applications and vouches for it.
- **Generalized**: though Stonebraker said that one-size-fits none, Bigtable seems to fly in the face of that. It supports Google Earth location data, Google Analytics webpage data, and Google Finance stock data. Those are some diverse data.
- **Scalable**: Google is a paragon of data volume, so if it works for trillions of rows of data, it probably works for anyone's needs.
- **Not relational**: Stonebraker said the technology is dying; Google adapted early (around the time of the original Stonebraker paper).

## bigtable.disadvantages();

- **Generalized**: Any application that specializes in storing a particular type of data would probably outperform Bigtable as Stonebraker alluded to.
- **Bigtable is proprietary**: Stonebraker mentioned the potential for innovation right now; though Google's published a paper about its implementation, the actual source code is proprietary, which slows overall innovation in this space.
- **Not relational**: Lots of existing Database Administrators know SQL, which enables them to interact with the data. Bigtable does not support SQL, and is thus missing out on lots of potential quality DBAs.