Storage, Processing and Reliability on Google's Distributed Systems

Dario Freni, Google
dariofreni@google.com

Outline

"Google's mission is to organise the world's information and make it universally accessible and useful."

- Organizing information
  - Protocol buffers
  - Google File System
  - Bigtable
  - Spanner/F1
- Processing information
  - MapReduce
  - Flume
  - MillWheel
- Serving information
  - High Availability

Organizing information

Google's systems are storing enormous amount of data, examples:

- web index!
- cache archive of web pages
- all YouTube videos
- Maps and related information
- GMail data
- Picasa / Google+ photos

Storage systems

Over the years many different storage technologies have been developed at Google.

- Code and implementations proprietary, BUT
- Results are often shared with the research community
  - many open source alternatives available

http://research.google.com/

Example names:

- Chubby
- Google File System
- Bigtable
- Megastore
- Colossus
- Blobstore
- Spanner
- F1

Protocol buffers

- Structured representation of data
- Message definition is compiled into native classes
  - most languages supported
- Why not JSON/XML/Whatever?
  - Efficiency
  - Compactness
  - Safer wrt backwards compatibility

Example message definition:

```
message WebPage {
  required string url = 1;
  required string html = 2;
  repeated string keywords = 3;
  optional bool visited = 4;
}
```

Google

Protocol buffers

- Structured representation of data
- Message definition is compiled into native classes
  - most languages supported
- Why not JSON/XML/Whatever?
  - Efficiency
  - Compactness
  - Safer wrt backwards compatibility

Example message definition:

```
message WebPage {
  required string url = 1;
  required string html = 2;
  repeated string keywords = 3;
  optional bool visited = 4;
}
```

OPEN SOURCE

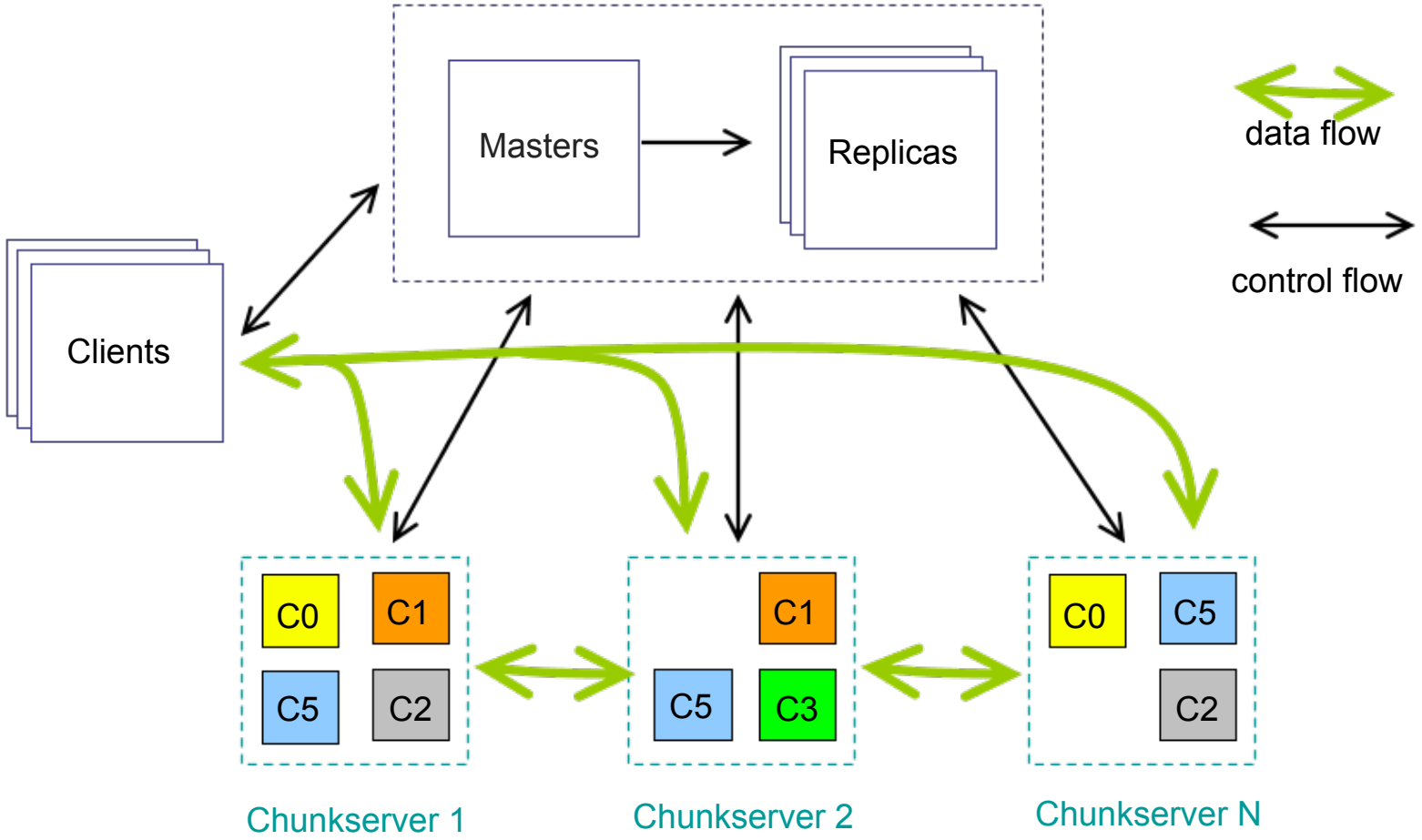
(just search "Protocol Buffers" on Google to find it)

GFS - Google File System

"A massively distributed and fault tolerant
file system that efficiently stores and retrieves data"

- Distributed storage filesystem
- Provides basic POSIX calls for files
  - open(), read(), write(), close()
- Allows multiple client to access simultaneously
- Improves availability of the data by storing multiple copies in different machines

GFS - Architecture



GFS - Colossus

The original implementation of GFS is now > 10 years old

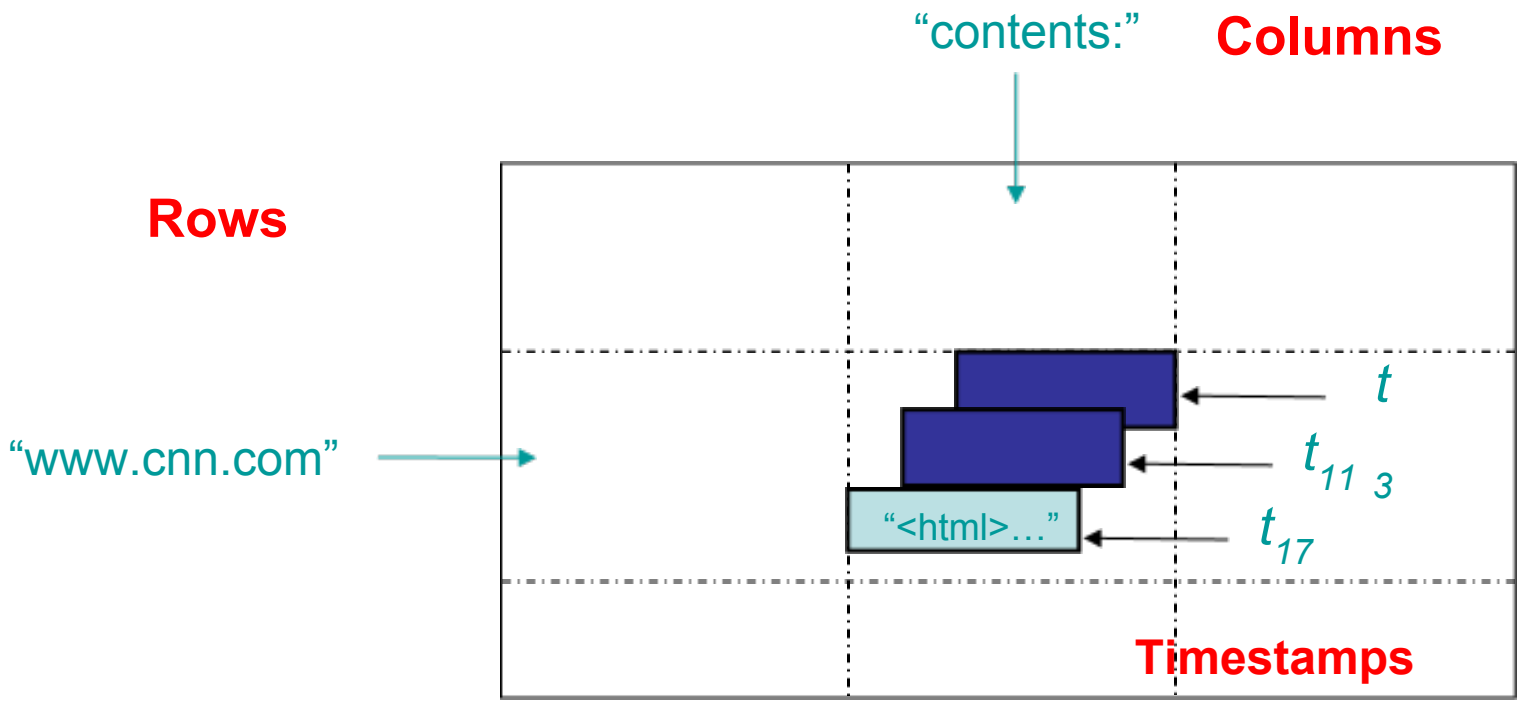
To accommodate new requirements, we currently use a new technology code named Colossus:

- Uses Bigtable technology to store metadata
  - (will explain Bigtable in a bit)
- Increases the file size limits
- Improves the latency
- Decrease storage usage
  - Redundancy can be achieved by using **Reed-Solomon** encodings
    - (Google "Reed-Solomon" for more details)

Bigtable

- Distributed, multi-dimensional sorted map
- Designed to store billions of rows
- Scale across thousands of machines
  - in one datacenter
- Provide data replication
  - across multiple datacenters
- Built on top of **GFS**

Bigtable


"contents:"
Columns
Rows
"www.cnn.com"
"<html>…"
t
t11 3
t17
Timestamps

Bigtable - Tablets

- Large tables broken into **tablets** at row boundaries
  - Tablet holds contiguous range of rows
  - Aim for ~100MB to 200MB of data per tablet
- Serving machine responsible for ~100 tablets
  - Fast recovery:
    - 100 machines each pick up 1 tablet from failed machine
  - Fine-grained load balancing:
    - Migrate tablets away from overloaded machine
    - Master makes load-balancing decisions

From Bigtable to Spanner

Bigtable good, but…

- does not support transactions: atomicity is only guaranteed at the row-level
  - replicated Bigtables are only eventually consistent
- does not support relational tables: for application requiring complex, evolving schemas, Bigtable can be difficult to use.

Other systems were built on top of Bigtable trying to fix this:

- e.g.: Megastore for supporting transactions and replication. There is only so much you can do without adding too much complexity.

Spanner

A new system, named **Spanner**, has been studied and developed for **years** to provide stronger guarantees than Bigtable wrt transactions and global replication.

Distributed multiversion database

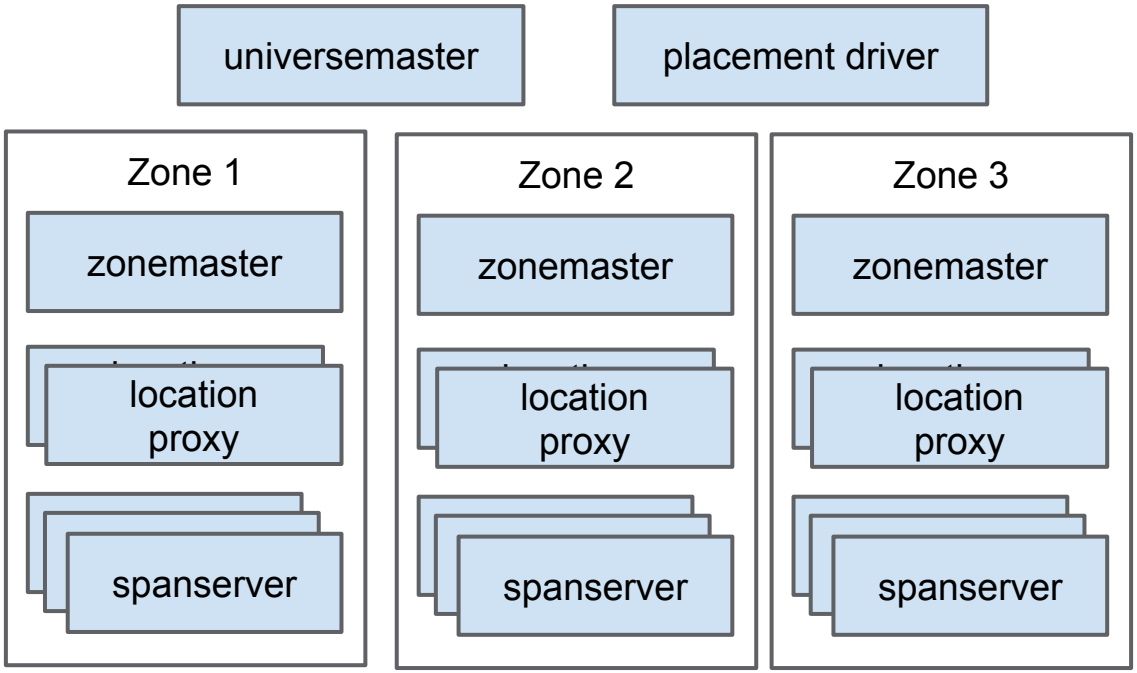
- General-purpose transactions (ACID)
- SQL query language
- Schematized tables
- Semi-relational data model

Spanner

"Designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows"

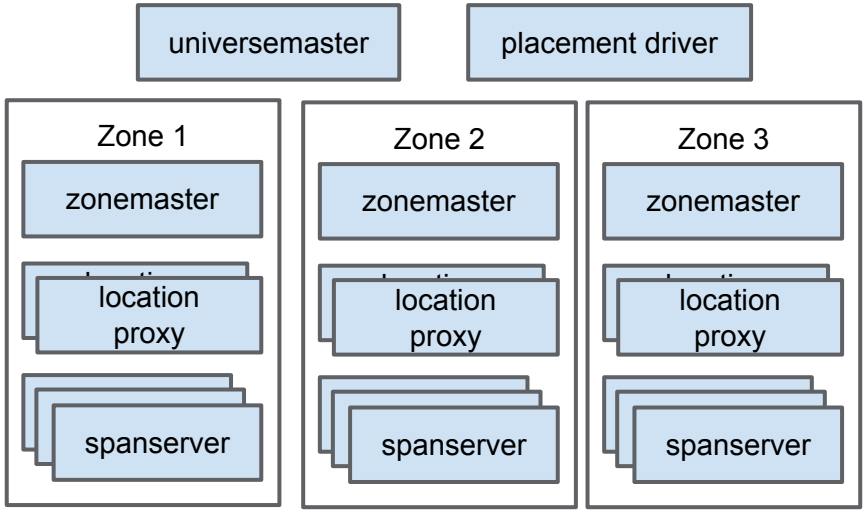
- Like Bigtable, data is versioned
  - the timestamp of each version is the commit time
- Global distribution of data is configurable
  - e.g.: data can automatically be moved closer to its most frequent users
- Provide strong consistency guarantees
  - external consistency of reads and writes
  - global consistent reads across the database at a timestamp
    - useful e.g. for backups, MapReduce executions, atomic schema changes

Spanner - Implementation



Spanner - Implementation

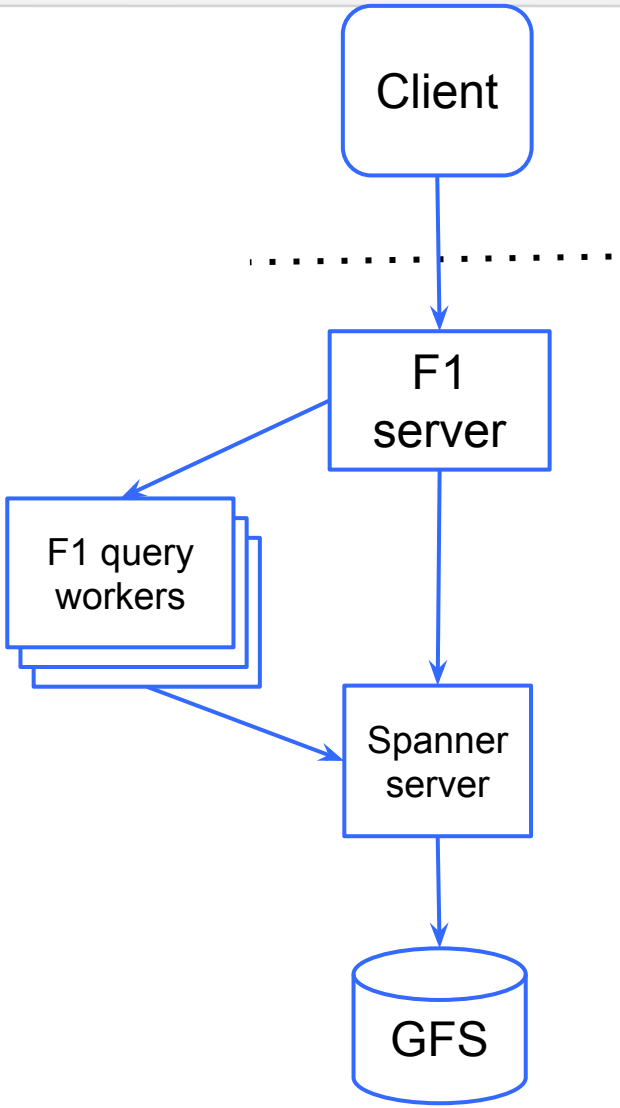
- Each spanserver contains 100-1000 *tablets*.
- Tablets are replicated across datacenters (zones)
- Distributed locking happens at a tablet-level
  - At each transaction the commit timestamp is confirmed across zones using Paxos.
  - This is made possible by a particular "uncertain" time representation (TrueTime).



F1

Features

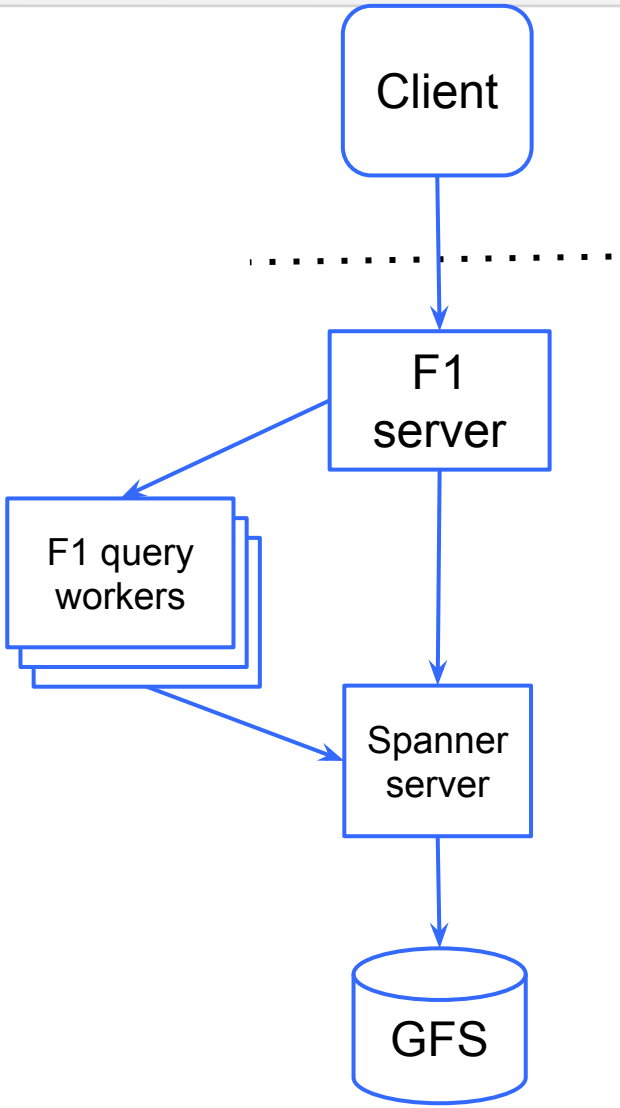
- Relational schema
  - Consistent indexes
  - Extensions for hierarchy and rich data types
    - Protocol buffers!
  - Non-blocking schema changes
- Multiple interfaces
  - SQL, key/value R/W, MapReduce
- Change notifications



F1

Architecture

- Sharded Spanner servers
  - data on GFS and in memory
- Stateless F1 server
- Worker pools for distributed SQL execution



F1

Column data types are mostly Protocol Buffers

- Stored like blobs in Spanner
- SQL syntax extensions for reading nested fields
- Coarser schema with fewer tables - inlined objects instead

Why useful?

- Protocol Buffers pervasive at Google → no impedance mismatch
- Simplified schema and code - apps use the same objects
  - Don't need foreign keys or joins if data is inlined

Google

F1 - SQL on Protocol Buffers

```
SELECT CustomerId, Whitelist
FROM Customer
```

CustomerId	Whitelist

123	feature {
	  feature_id: 18
	  status: ENABLED
	}
	feature {
	  feature_id: 269
	  status: ENABLED
	}
	feature {
	  feature_id: 302
	  status: ENABLED
	}

```
SELECT CustomerId, f.*
FROM Customer c
PROTO JOIN c.Whitelist.feature f
WHERE f.feature_id IN (269, 302)
  AND f.status = 'ENABLED'
```

CustomerId	feature_id	status
123	**269**	**ENABLED**
123	302	ENABLED

Google

Have a break!

See you in 10 minutes!

MapReduce

- Programming model for processing and generating large data sets
- Can be applied to a large variety of problems
- Allows for parallel computation of problems of the following form:

map: (k1,v1) → list(k2,v2)
reduce: (k2,list(v2)) → list(v2)

- Where k1 and v1 is an input key-value map, and v2 is the desired output

MapReduce

Example: counting number of occurrences of each word in a large set of documents.

Map function emits a symbolic "1" per each word

Reduce gets called once per each unique word, with the list of values associated to it.

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator
values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
    Emit(AsString(result));
```

MapReduce

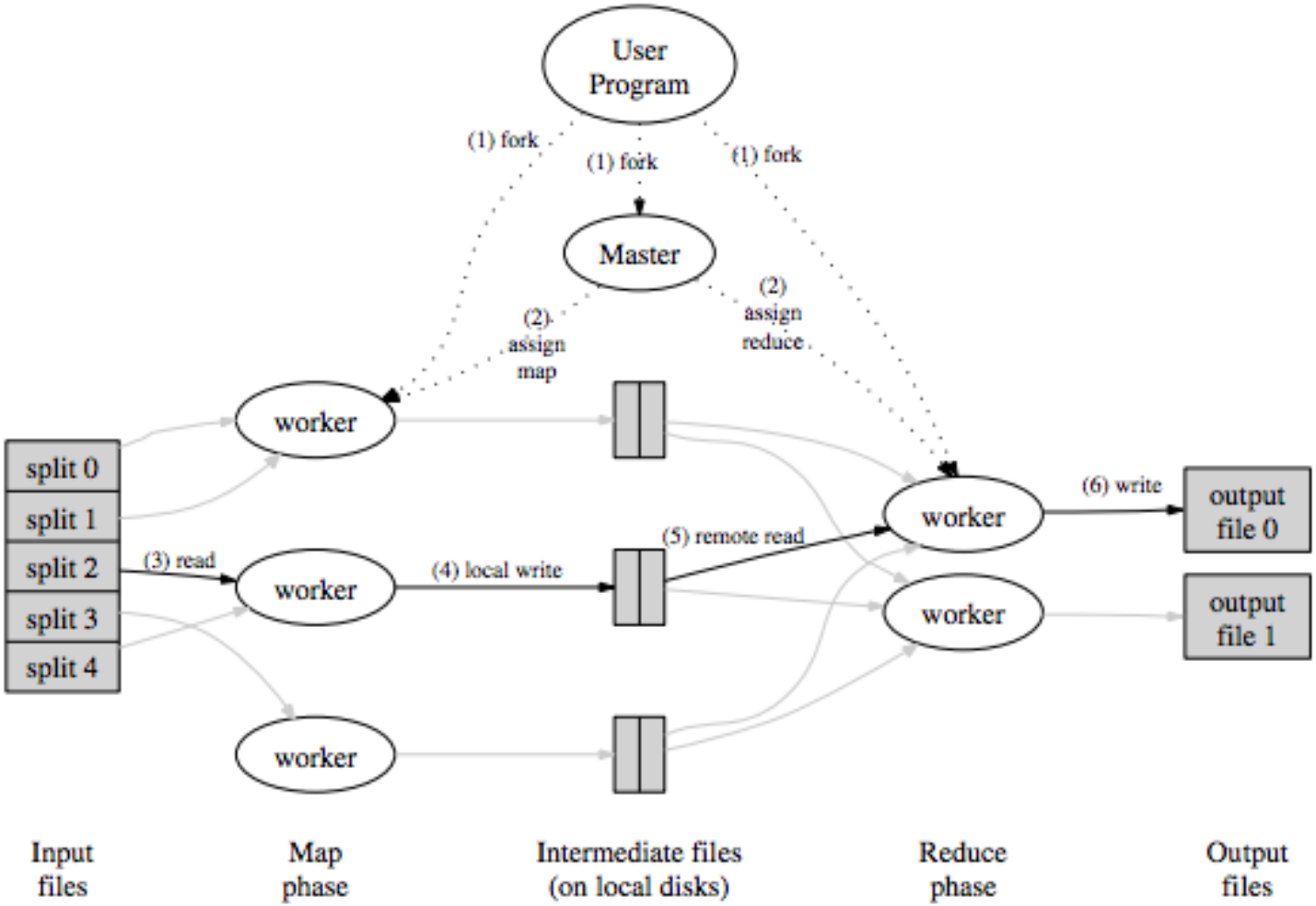
The system performs an important extra step between Map and Reduce, informally called Shuffle:

map: (k1,v1) → list(k2,v2)
shuffle: list(k2,v2) → (k2,list(v2))
reduce: (k2,list(v2)) → list(v2)

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator
values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
    Emit(AsString(result));
```

Google

MapReduce - Architecture



MapReduce

Plenty of possible applications, simple examples:

- Distributed Grep
  - Map-only, emit results if a keyword is found
- Count URL clicks
  - Input is weblogs, Map emits <URL, 1>, Reduce emits <URL, count>
- Reverse Web-Link graph
  - Map scans all the *target* links from each *source* page
    - emits <target, source>
  - Reduce concatenates the list of all the sources
    - emits <target, list(source)>

You can concatenate multiple MapReduces for doing more complex algorithms

- but it can start to become unmaintanable

FlumeJava

- API + Library
  - Java library for writing data-parallel pipelines
  - Classes for (possibly huge) immutable collections
  - Methods for data-parallel operations
  - Builds execution graph implicitly via "lazy execution"
- Optimizer
  - Fuses data-parallel operations, forms MapReduces
- Executor
  - Runs optimized execution graph
  - Garbage collection, task parallelism, fault tolerance, monitoring

FlumeJava - Example: TopWords

```
PCollection<String> lines =
  readTextFile("...");
PCollection<String> words =
  lines.parallelDo(
    new ExtractWordsFn());
PTable<String,Long> wordCounts =
  words.count();
PTable<String,Long> topWords =
  wordCounts.top(
    new WordOrderFn(), 1000);
PCollection<String> formattedOutput=
  topWords.parallelDo(
    new FormatFn());
formattedOutput.writeToTextFile
("...");

FlumeJava.run();
```

```
class FormatFn extends
MapFn<Pair<String,Long>,String>() {
  public String map(
    Pair<String, Long> pair) {
    return pair.getFirst() +
      ": " + pair.getSecond();
  }
}
```

FlumeJava - Example: TopWords

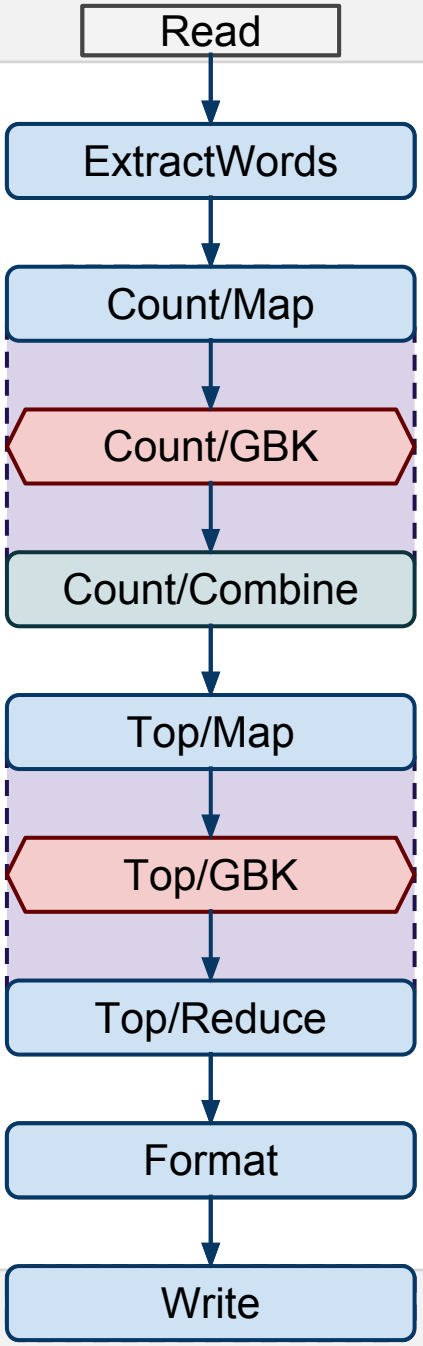
```
readTextFile(...)
  .parallelDo(new ExtractWordsFn())
  .count()
  .top(new WordOrderFn(), 1000)
  .parallelDo(new FormatFn())
  .writeToTextFile(...);

FlumeJava.run();
```

```
class FormatFn extends
MapFn<Pair<String,Long>,String>() {
  public String map(
    Pair<String, Long> pair) {
    return pair.getFirst() +
      ": " + pair.getSecond();
  }
}
```

FlumeJava - Execution Plan and Optimizer

- Represent the deferred computation as a graph

- Basic operation primitives:
  - ParallelDo (DoFn)
  - GroupByKey
  - CombineValues (CombFn)
  - Flatten



Google

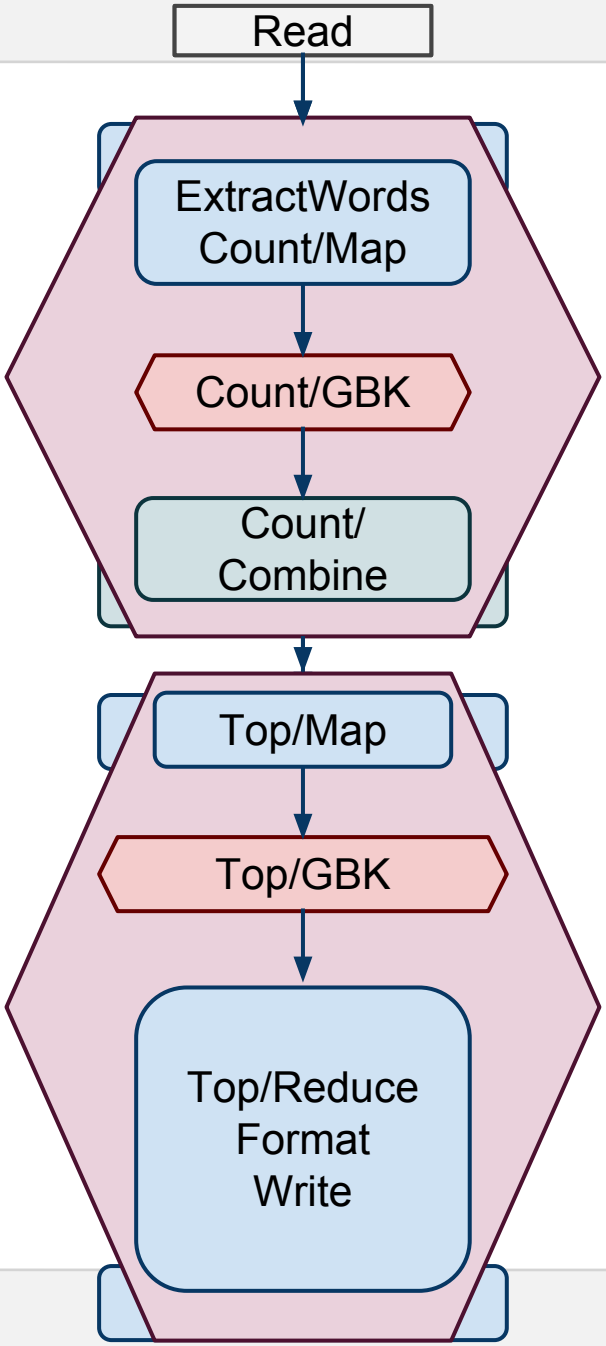
FlumeJava - Execution Plan and Optimizer

Fuse trees of parallelDo into one

- producer-consumer
- co-consumer ("siblings")
- eliminate now-unused intermediate PCollections

Form MapReduces

- pDo* + gbk + cv + pDo* --> MapShuffleCombineReduce (MSCR)
- Multi-mapper, multi-reducer, multi-output



MillWheel

Flume and MapReduce are suitable for processing data in **batch**.

- This means that the only way to process new input data is to reprocess all the input dataset.

MillWheel is a system to process large **streams** of data

- Millions of events per second

Low-latency analysis is becoming increasingly important

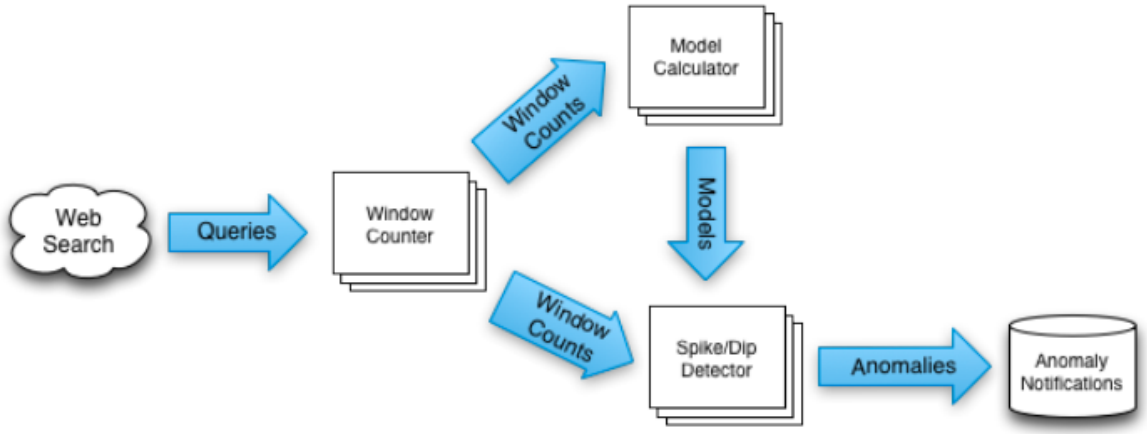
- Breaking news articles
- Intrusion detection
- Quickly shutting down spammers

Many custom, problem-specific solutions were being built.
MillWheel aims to build a general framework.

MillWheel

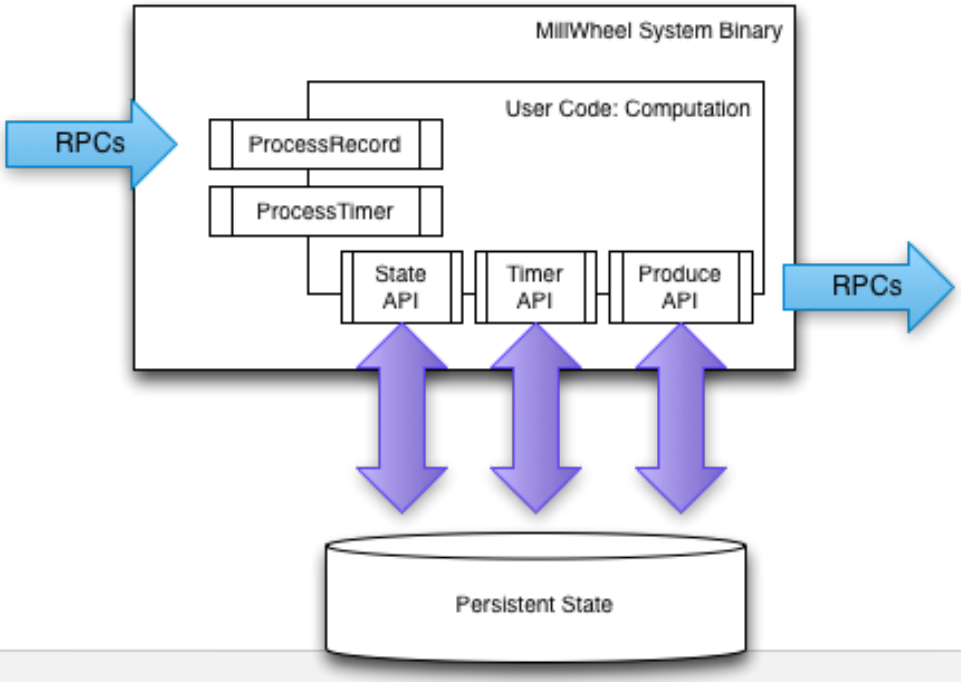
MillWheel is a framework for creating streaming analysis systems. It provides a programming model and an underlying execution system.

- User-defined code runs in *Computations*
- Computations are connected by *Streams*
- Records flow along edges as (key, value, timestamps) tuples



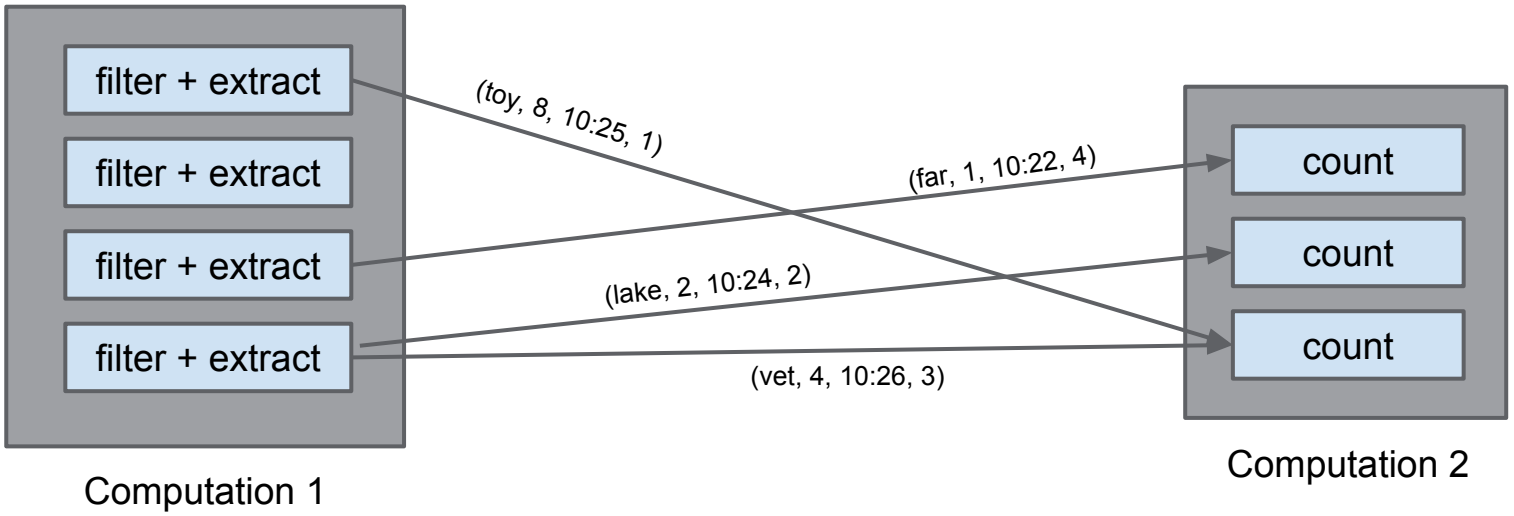
MillWheel

Processing is **per key**. Computations can access and mutate **state** for the current record's key, set and process **timers**, and **produce** records.



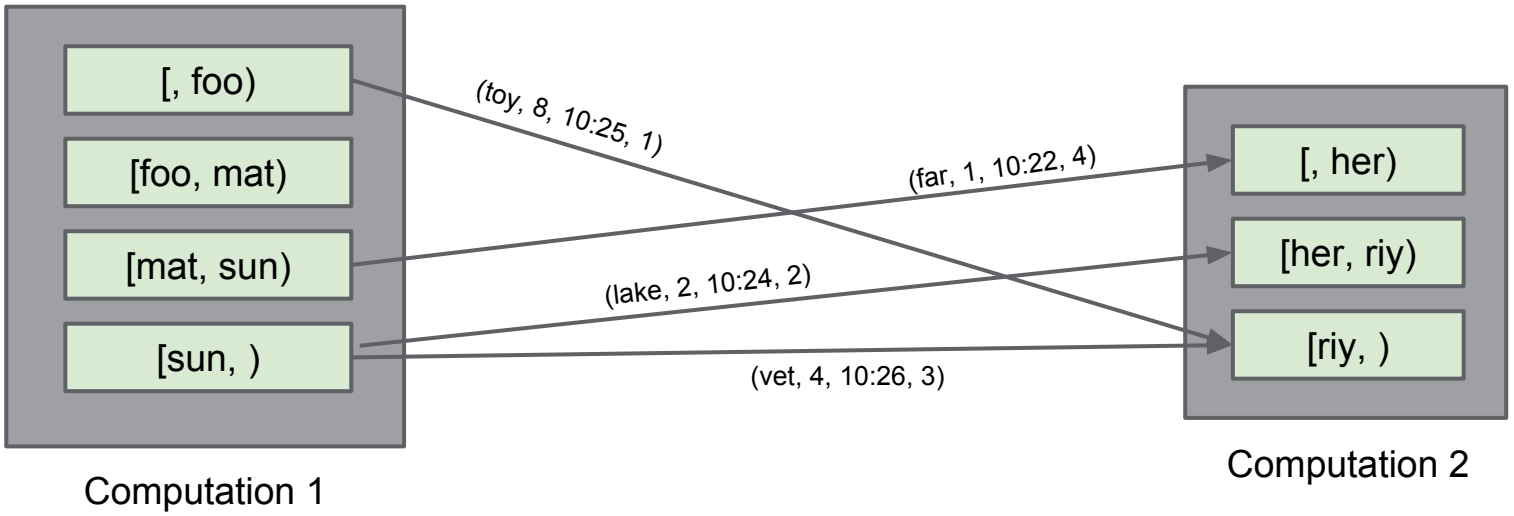
MillWheel

Each computation in the graph represents processing a **keyspace** (might be several user functions). Edges are **shuffle paths**, and **(key, value, timestamp, sequence number)** quadruples flow along then. Timestamps are determined by data sources, not by MillWheel.



MillWheel

Each computation is **range sharded** across many workers. A single key is managed by a single worker at a time, allowing us to consistently update per-key persistent state.



Available on Google Cloud!

MapReduce framework already available in App Engine

- Java and Python supported
  - Search for "MapReduce App Engine" to find more.

Flume + MillWheel is offered as Google Cloud Dataflow product:

- You can download and play with the SDK
  - https://github.com/GoogleCloudPlatform/DataflowJavaSDK
- You can request access for using it in your Google Compute Engine instances

Reliability - machines aren't perfect

Our services need to stay up 24/7. Reasons are obvious:

- it is what users expect!

But this stuff runs on many servers, and servers can break. Servers **will** break. Example:

- Power supplies **M**ean **T**ime **B**etween **F**ailures is 100000 hours
  - 10000 machines mean one PSU will fail every 10 hours

- It's not a matter of **if**, it's just a matter of **when**.

Reliability - machines aren't perfect (2)

How can we make sure the service stays up despite machine failures?

Software must be design to cope with failures:

- It must be capable of running on several machines and fail over gracefully if a machine breaks
- It's desirable to run N+2 instances of the software in production, to increase fault-tolerance

Reliability - people aren't perfect either

Software bugs will happen. How can we minimize the impact?

Extensive testing, e.g. when making changes to the code

- unit testing
- integration testing

Before rolling out the code

- regression testing
- staging environment

Reliability - people aren't perfect either (2)

Gradual roll-out of the new releases to production, for example:

- update one machine, check if looks good
- update all machines in one datacenter, check if they look good
- repeat for all datacenters

At this scale, it is unfeasible to do this manually.

This and similar processes require sophisticated **automation**.

Challenging problem: how can you do the "check if looks good" step with 100% confidence when rolling out a new version of Search?

Reliability - What happens when something goes wrong?

Monitoring systems are in place and (hopefully) generate alerts when something goes wrong

Alerts go to one person on-call

- Critical services are managed by Site Reliability Engineers
- They have a 24/7 on-call rotation which usually includes teams from different continents to avoid being on-call at night
- Each engineer is usually on-call 10-15% of their time, depending on the number of people in the rotation

Reliability - What happens when something goes wrong? (2)

"OMG panic what should I do!"

SREs are trained to face the most difficult situations:

- they have in-depth knowledge of the services they are on-call for
- they have amazing troubleshooting skills
- they have been trained to handle incidents
  - more people can be contacted if deemed necessary
  - everyone helps when it's about resolving production issues


JOIN US!

Technical Internships

- The hiring for 2015 interns is concluded. We will start looking for 2016 interns in September 2015. Check out google.com/jobs/students for more information.
- Apply online at google.com/jobs/students and submit your CV and transcripts
- You apply for a role, not a specific project
- Internships take place year-round
- Start/end dates are flexible
- Are at least three months of full-time work
- Paid
- You can intern in Google offices globally
- We do not offer Master Thesis internships

A Software Engineering Intern at Google:

- Must be currently pursuing a BS, MS or PhD in Computer Science or a similar technical field. Must start and end before they graduate
- Must have experience in systems software or algorithms
- Must have excellent programming skills (C++, Java, or Python)
- Must have knowledge of UNIX/Linux or Windows environments and APIs

Google™

DO COOL THINGS THAT MATTER

New Grad Roles

You should apply if you:

- are in your final year of higher education or out of university less than 12 months
- apply for positions globally throughout the year online at www.google.com/jobs/students
- We have a lot of openings in Europe (Zurich, London) and in Mountain View, California

What's the format of an interview?

- Step 1: Short introduction
- Step 2: Main part of the interview / technical assessment
- Step 3: Your chance to ask questions

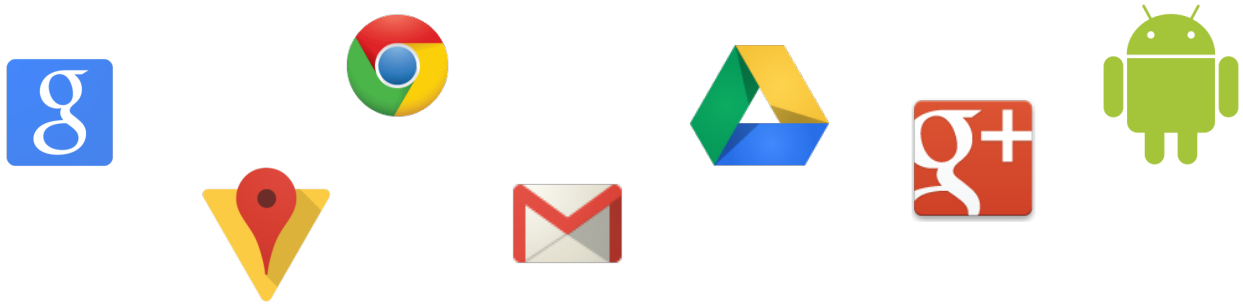
- There won't be puzzle questions - they don't reflect your problem solving/coding/design abilities!
- Questions will be in-depth. We want to see how you think about complicated problems.
- The right answer would be nice but it is not necessary – your thought process is more important

What skills are we looking to assess?

- Language syntax, idioms, performance issues
- Algorithms and data structures
- Analytical skills
- Sound design
- Communication skills

MORE INFORMATION

g.co/techstudentsEMEA



Thanks!