The age of rename() is over*

Steve Loughran

December 2018

(*mostly)

Me: stevel@apache.org

- Hadoop committer ~10 years
- Object storage
- The lower bits of your stack traces belong to me

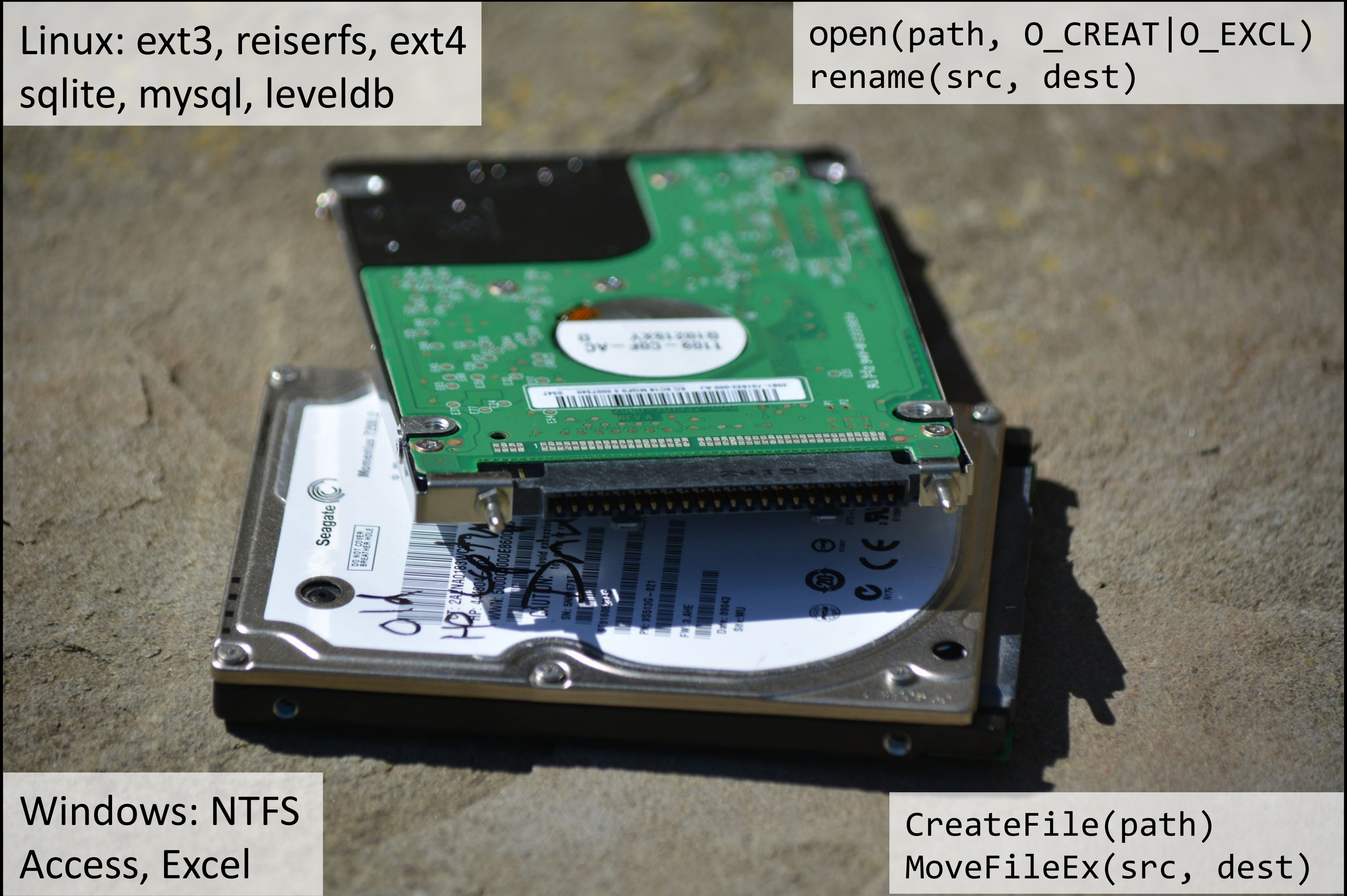
Why do we save data?

- Publish data for other applications
- Process data published by other applications
- Persist state for future instances of an application
- Work with more data than fits into RAM
- Share data with other instances of same application
- Save things people care about & want to get back

FAT8
dBASE II & Lotus 1-2-3
int 21h

Linux: ext3, reiserfs, ext4
sqlite, mysql, leveldb

```
open(path, O_CREAT|O_EXCL)
rename(src, dest)
```



Windows: NTFS
Access, Excel

```
CreateFile(path)
MoveFileEx(src, dest)
```

Facebook Prineville Datacentre
1+ Exabyte on HDFS + cold store
Hadoop, Hive, Spark, Kafka, ...

Model and APIs

Model #1: File-System

Directories and files

Posix with stream metaphor

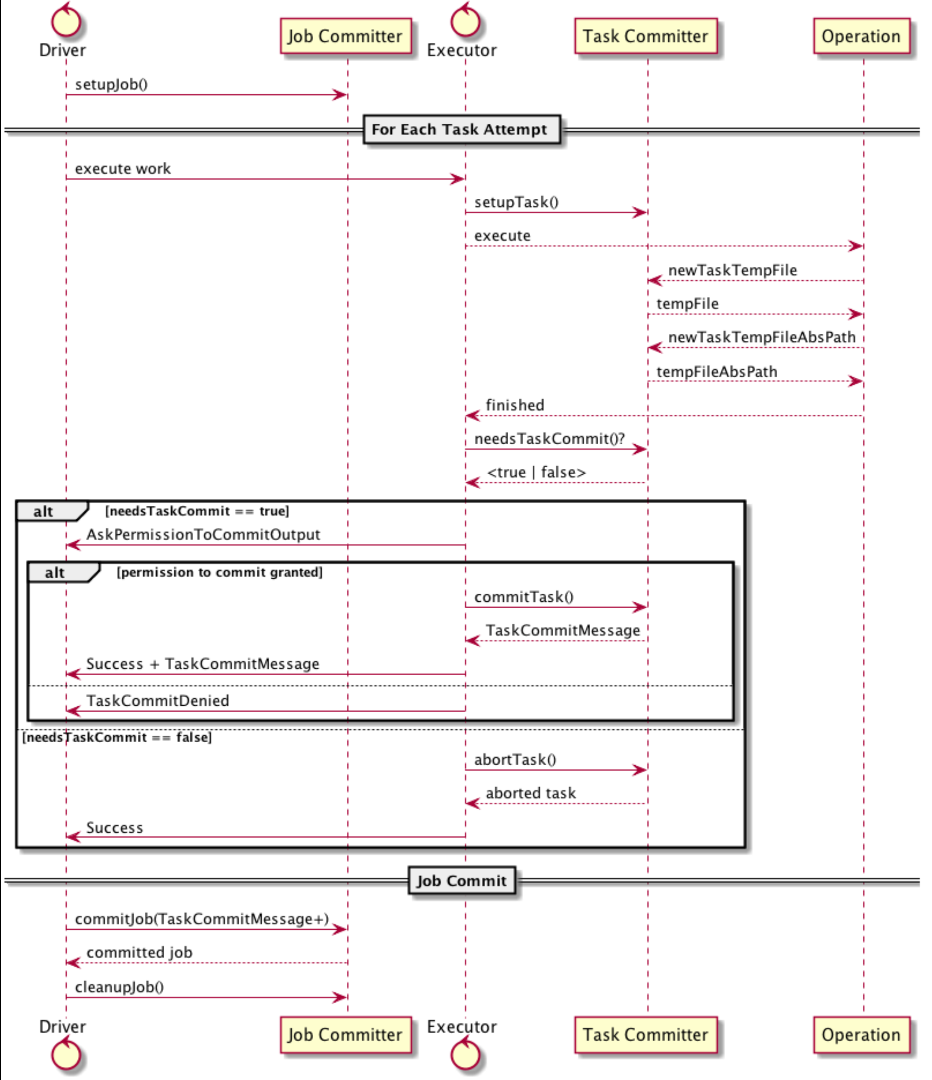
Posix: hierarchical (distributed?) filesystems

`org.apache.hadoop.fs.FileSystem`

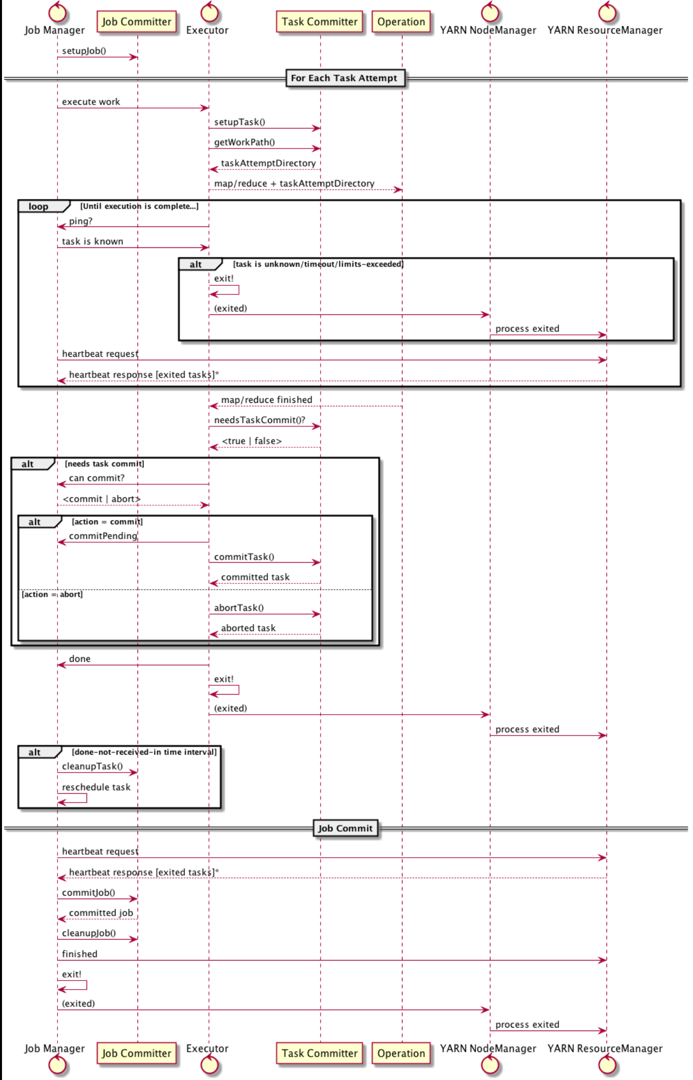
hdfs wasb s3a swift adl gcs

```scala
val work = new Path("s3a://stevel-frankfurt/work")
val fs = work.getFileSystem(new Configuration())
val task00 = new Path(work, "_temp/task00")
fs.mkdirs(task00)
val out = fs.create(new Path(task00, "part-00"), false)
out.writeChars("hello")
out.close();
fs.listStatus(task00).foreach(stat =>
  fs.rename(stat.getPath, work)
)
```

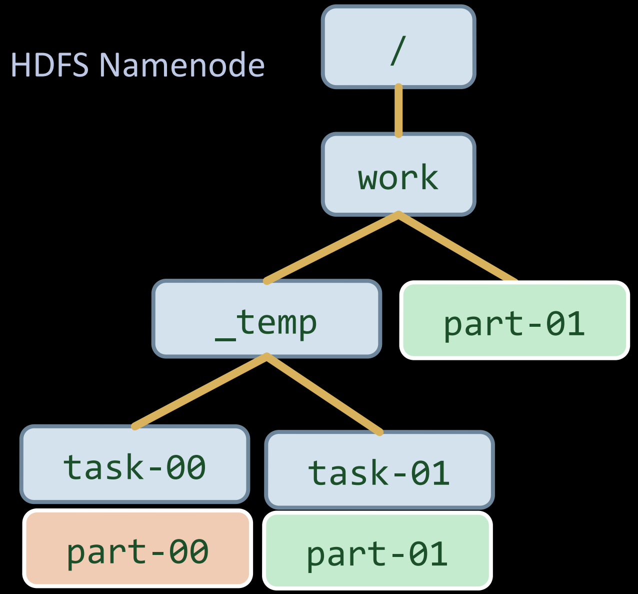
Apache Spark Commit Protocol

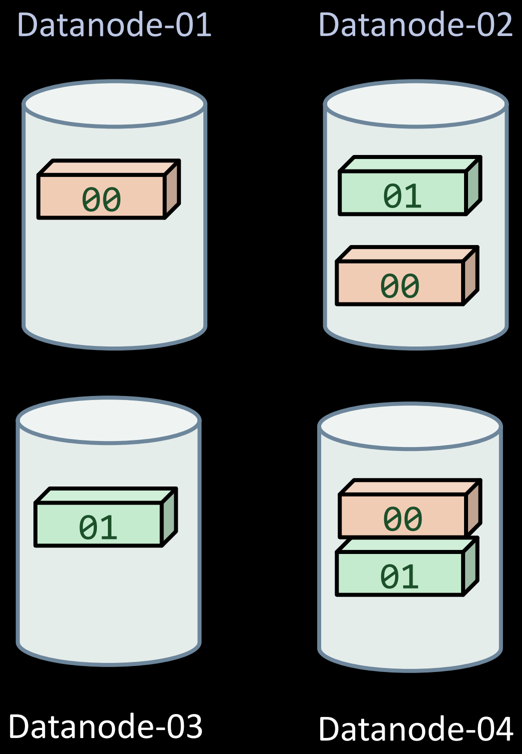


MapReduce Commit Protocol



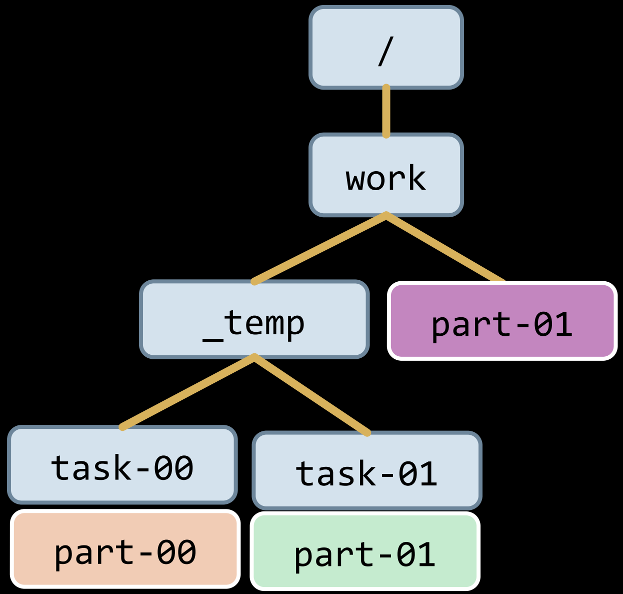
rename() gives us O(1) atomic task commits

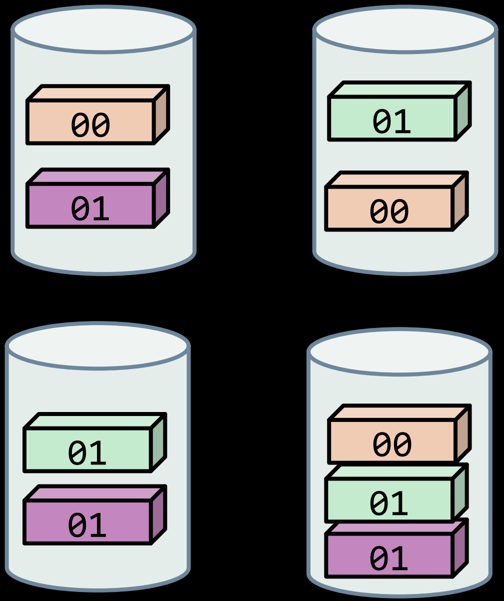





```
rename("/work/_temp/task00/*", "/work")
```

Amazon S3 doesn't have a rename()



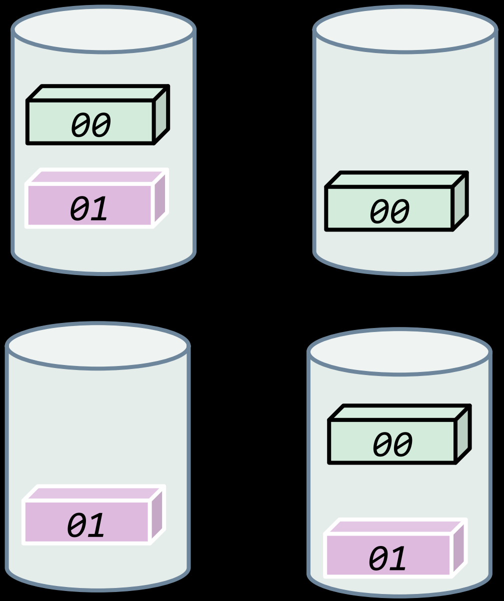



```
LIST /work/_temp/task-01/*
COPY /work/_temp/task-01/part-01 /work/part-01
DELETE /work/_temp/task-01/part-01
```

Fix: end the metaphor

work/part-01

work/part-00



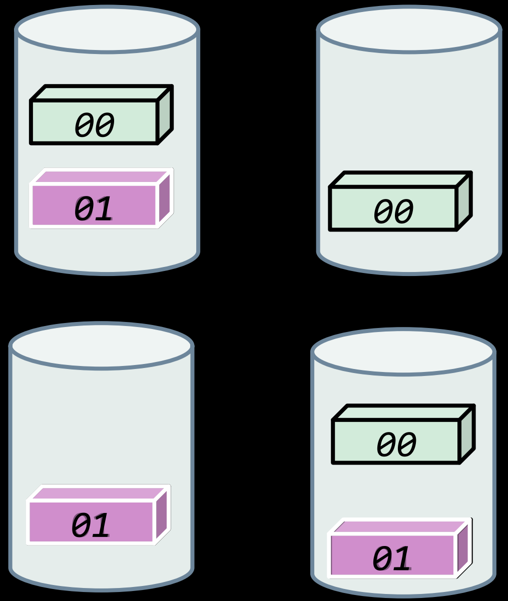

```
POST /work/part-01?uploads => UploadID

POST /work/part01?uploadId=UploadID&partNumber=01
POST /work/part01?uploadId=UploadID&partNumber=02
POST /work/part01?uploadId=UploadID&partNumber=03
```

job manager selectively completes the uploads

work/part-01

work/part-00

```
POST /work/part-01?uploadId=UploadID
<CompleteMultipartUpload>
 <Part>
  <PartNumber>01</PartNumber><ETag>44a3</ETag>
  <PartNumber>02</PartNumber><ETag>29cb</ETag>
  <PartNumber>03</PartNumber><ETag>1aac</ETag>
 </Part>
</CompleteMultipartUpload>
```



S3A O(1) zero-rename commit demo!

What else to rethink?

- s/Directories/r/patterns/
- `seek()+read()` sequences ==> Scatter/Gather IO
- Blocking operations ==> async block reads executed out of order
- TODO: How to work with Eventually Consistent data?

HADOOP-11867

Model and API #2: memory

SSD via SATA
SSD via NVMe/M.2
Future NVM technologies

```c
typedef struct record_struct {
  int field1, field2;
  long next;
} record;

int fd = open("/shared/dbase", O_CREAT | O_EXCL);
record* data = (record*) mmap(NULL, 8192,
  PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

(*data).field1 += 5;
data->field2 = data->field1;

msync(record, sizeof(record), MS_SYNC | MS_INVALIDATE);
```

```c
typedef struct record_struct {
  int field1, field2;
  record_struct* next;
} record;

int fd = open("/shared/dbase");
record* data = (record*) pmem_map(fd);

// lock ?

(*data).field1 += 5;
data->field2 = data->field1;

// commit ?
```

NVM moves the commit problem into memory I/O

- How to split internal state into persistent and transient?
- When is data saved to NVM? (cache flushed, sync in memory buffers, ...)
- How to co-ordinate shared R/W access over RDMA?
- Versioning?
- How do we write apps for a world where rebooting doesn't reset our state?

Model #3: SQL

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain opera-

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

A History and Evaluation of System R

Donald D. Chamberlin
Morton M. Astrahan
Michael W. Blasgen
James N. Gray
W. Frank King
Bruce G. Lindsay
Raymond Lorie
James W. Mehl
Thomas G. Price
Franco Putzolu
Patricia Griffiths Selinger
Mario Schkolnick
Donald R. Slutz
Irving L. Traiger
Bradford W. Wade
Robert A. Yost

IBM Research Laboratory
San Jose, California

SUMMARY: System R, an experimental database system, was constructed to demonstrate that the usability advantages of the relational data model can be realized in a system with the complete function and high performance required for everyday production use. This paper describes the three principal phases of the System R project and discusses some of the lessons learned from System R about the design of relational systems and database systems in general.

1. Introduction

Throughout the history of information storage in computers, one of the most readily observable trends has been the focus on data independence. C.J. Date [27] defined data independence as "immunity of applications to change in storage structure and access strategy." Modern database systems offer data independence by providing a high-level user interface through which users deal with the information content of their data, rather than the various bits,

S3 Select: SQL in the object store

```
hadoop s3guard select -header use  -limit 20 \
 s3a://hwdev-steve-ireland-new/steve/boris.csv \
 "SELECT * FROM S3OBJECT s WHERE \
 endstation_name = 'Bayswater Road: Hyde Park'"
```

```
"9347774","1506","5903","04/01/2012 00:32","153","Bayswater Road:
Hyde Park","04/01/2012 00:06","151","Chepstow Villas: Notting
Hill"
"9348043","607","5365","04/01/2012 01:09","153","Bayswater Road:
Hyde Park","04/01/2012 00:59","153","Bayswater Road: Hyde Park"
"9363308","708","5167","04/01/2012 06:30","153","Bayswater Road:
Hyde Park","04/01/2012 06:18","404","Palace Gate: Kensington
Gardens"
...
```

Summary:
As storage changes so must the models

- Object stores address scale & HA —at cost of semantics and performance
- Non-Volatile Memory is the other radical change
- Posix metaphor isn't suited to either —*what next?*
- SQL makes all this someone else's problem
  (leaving only O/R mapping, transaction isolation...)

Questions?