# Programming in the Cloud

Babak Behzad
Faraz Faghri

## ... Big Data

Everybody fancies:

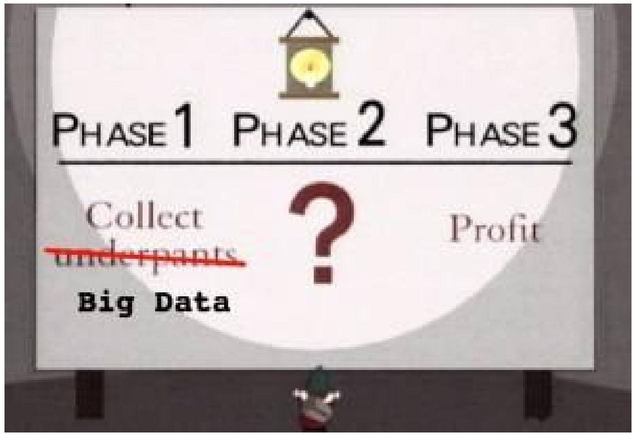
- data analysis
- log processing
- machine learning
- spam detection

...

But ...

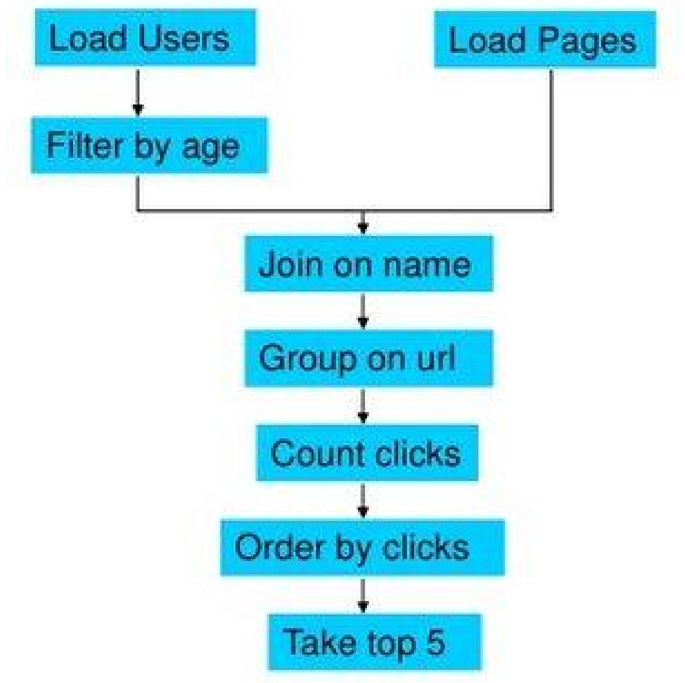
- unstructured data
- complex infrastructure
- MapReduce coding is time-consuming
- and long

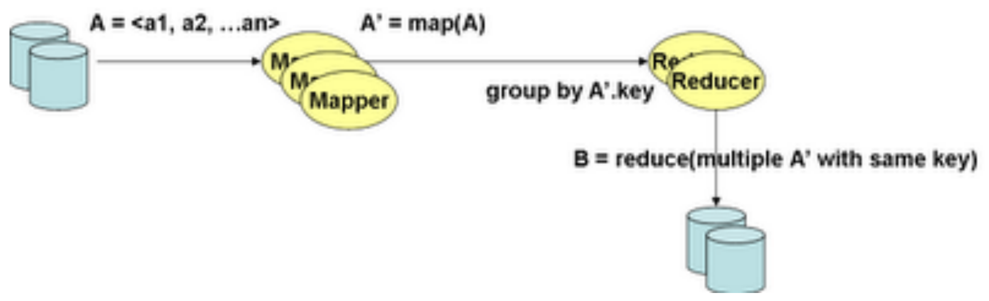
...



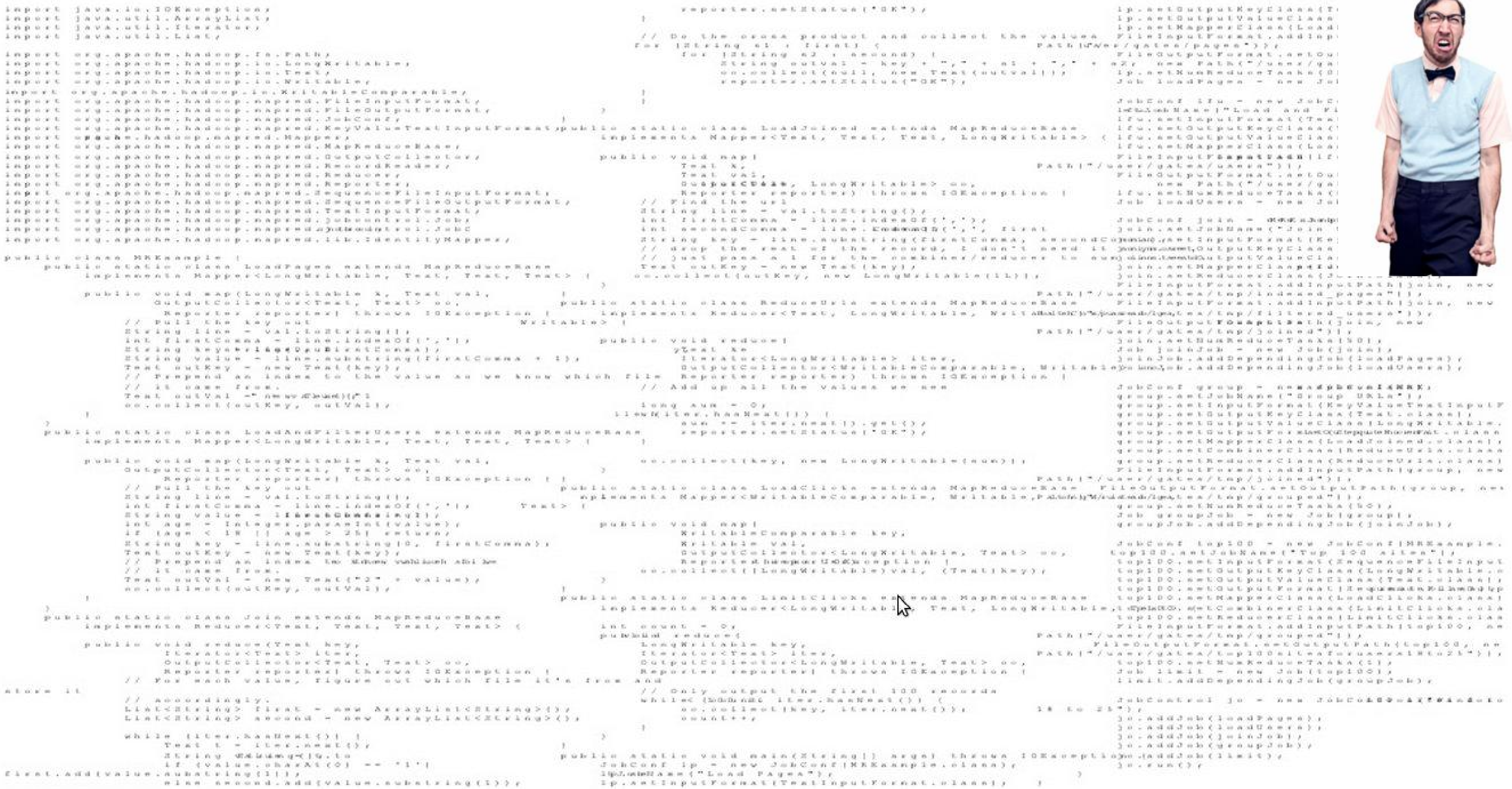
# Data analyst asks ...

- what are the top 5 most visited pages by users aged 18-25?





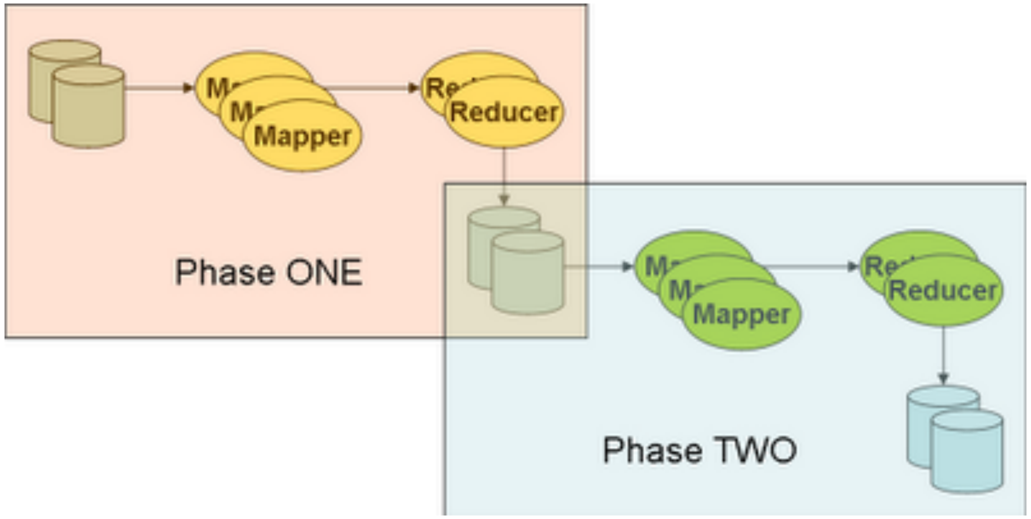
# MapReduce solution ...



```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                 Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                 Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));

                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first
Comma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);

        lp.setOutputKeyClass(T
        lp.setOutputValueClass
        lp.setMapperClass(Load
        FileInputFormat.addInp
Path("/user/gates/pages"));
        FileOutputFormat.setOu
            new Path("/user/ga
        lp.setNumReduceTasks(0
        Job loadPages = new Jo

        JobConf lfu = new JobC
        lfu.setJobName("Load and Fi
        lfu.setInputFormat(Tex
        lfu.setOutputKeyClass(
        lfu.setOutputValueClas
        lfu.setMapperClass(Loa
        FileInputFormat.addI
Path("/user/gates/users"));
        FileOutputFormat.setOu
            new Path("/user/ga
        lfu.setNumReduceTasks(
        Job loadUsers = new Jo

        JobConf join = new JobConf(MRExample
        join.setJobName("Join
        join.setInputFormat(Ke
        join.setOutputKeyClass
        join.setOutputValueCla
        join.setMapperClass(Id
        join.setReducerClass(J
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputF
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.
        group.setOutputFormat(SequenceFileOutputFormat.class
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class
        group.setReducerClass(ReduceUrls.class)
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInput
        top100.setOutputKeyClass(LongWritable.c
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutp
        top100.setMapperClass(LoadClicks.class)
        top100.setCombinerClass(LimitClicks.cla
        top100.setReducerClass(LimitClicks.clas
        FileInputFormat.addInputPath(top100, ne
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, ne
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

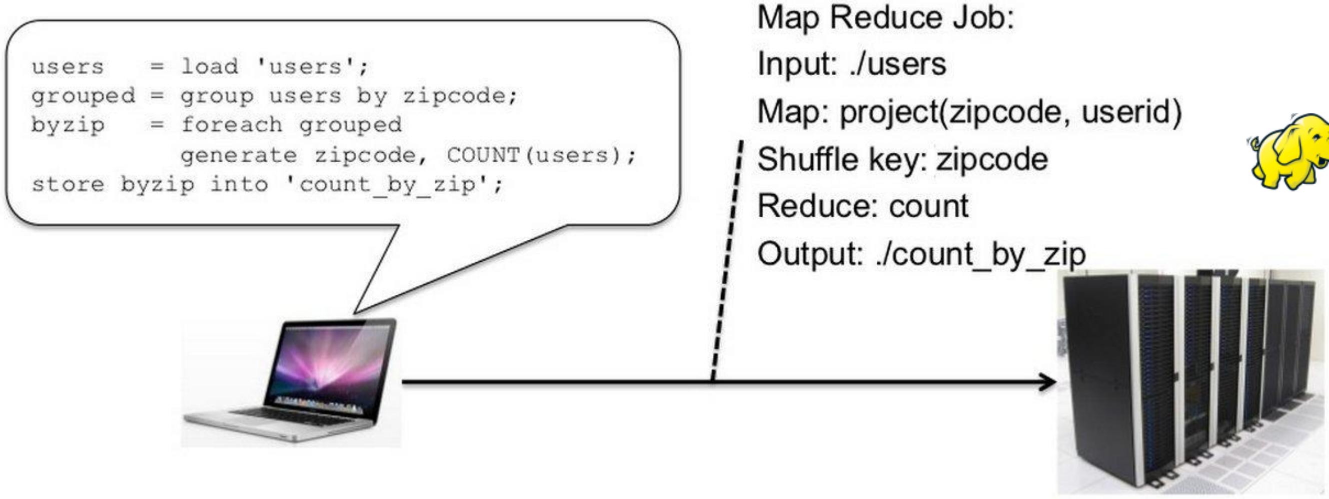
# Desired solution ...

- Desirable to have a higher level declarative language that describes the parallel data processing model.
- SQLish query where the user specifies the "what" and leaves the "how" to the underlying processing engine.



## Pig and Hive: 2 ways of doing 1 thing ...

- Both generate Map-Reduce jobs from a query written in a higher level language.
- User doesn't need to deal with Hadoop complexity.
- Lives on client machine, nothing to install on cluster.



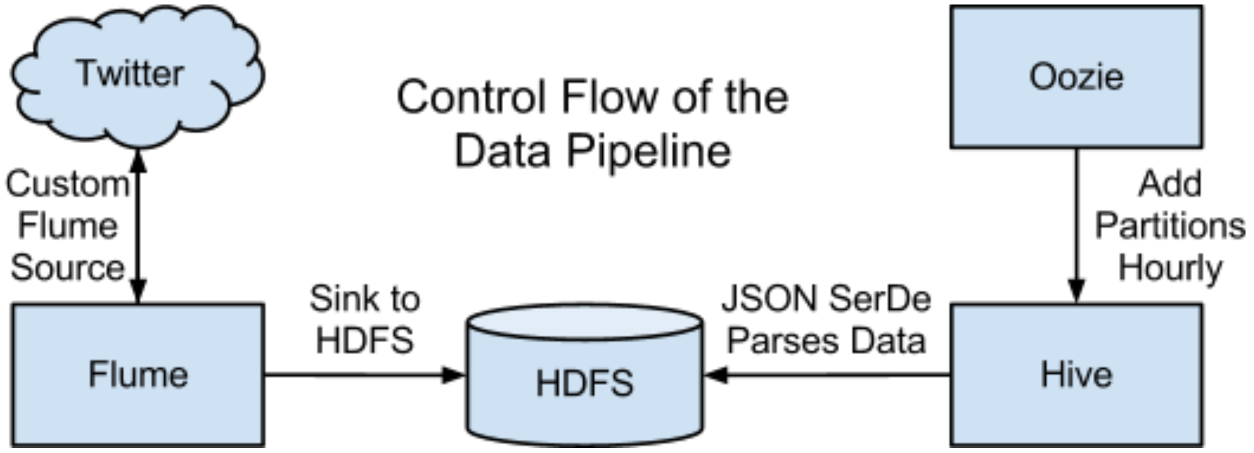
```
users   = load 'users';
grouped = group users by zipcode;
byzip   = foreach grouped
          generate zipcode, COUNT(users);
store byzip into 'count_by_zip';
```

Map Reduce Job:
Input: ./users
Map: project(zipcode, userid)
Shuffle key: zipcode
Reduce: count
Output: ./count_by_zip

## Pig and Hive example ...

| | **Procedural data-flow language** | **Declarative SQLish language** |
|---|---|---|
| **Count** | - mytable = GROUP mytable ALL;<br>- mytable = FOREACH mytable GENERATE COUNT (mytable);<br>- DUMP mytable; | - SELECT COUNT(*) FROM mytable; |
| **Count Distinct Number** | - mytable_col1 = GROUP mytable BY col1;<br>- mytable_col1_count= FOREACH mytable_col1 {<br>        col_2 = DISTINCT mytable.col_2;<br>        GENERATE group, COUNT(col_2);<br> };<br>- DUMP mytable_col1_count; | - SELECT COUNT(DISTINCT col1) FROM mytable; |
| **Join** | - mytable = JOIN mytable BY col1, othertable BY col1;<br>- DUMP mytable; | - SELECT * FROM mytable INNER JOIN othertable ON mytable.col1 = othertable.col1; |

# Let's analyze tweets ...



- **Twitter Streaming API** outputs **tweets** in a **JSON format**

- **Flume**: distributed service for efficiently collecting, aggregating, and moving large amounts of log data.
- **Hive** provides a query interface which can be used to query data that resides in HDFS.

- **SerDe** stands for Serializer and Deserializer, which are interfaces that tell Hive how it should translate the data into something that Hive can process.

- Apache Oozie is a workflow coordination system, an extremely flexible system for designing **job workflows**, which can be scheduled **scheduled to run** based on a set of criteria.

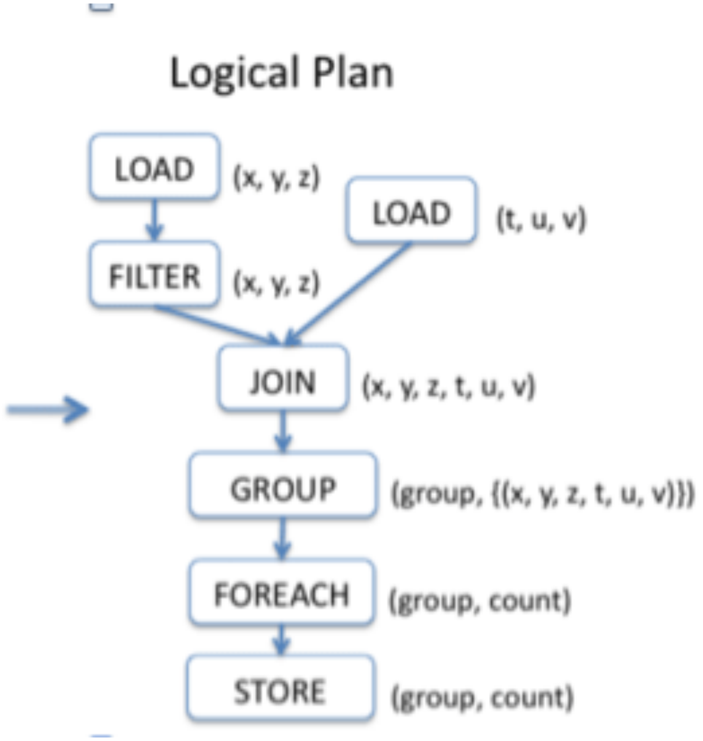
# Pig ...

pig.jar:

- •parses
- •checks
- •optimizes
- •plans execution
- •submits jar to Hadoop
- •monitors job progress

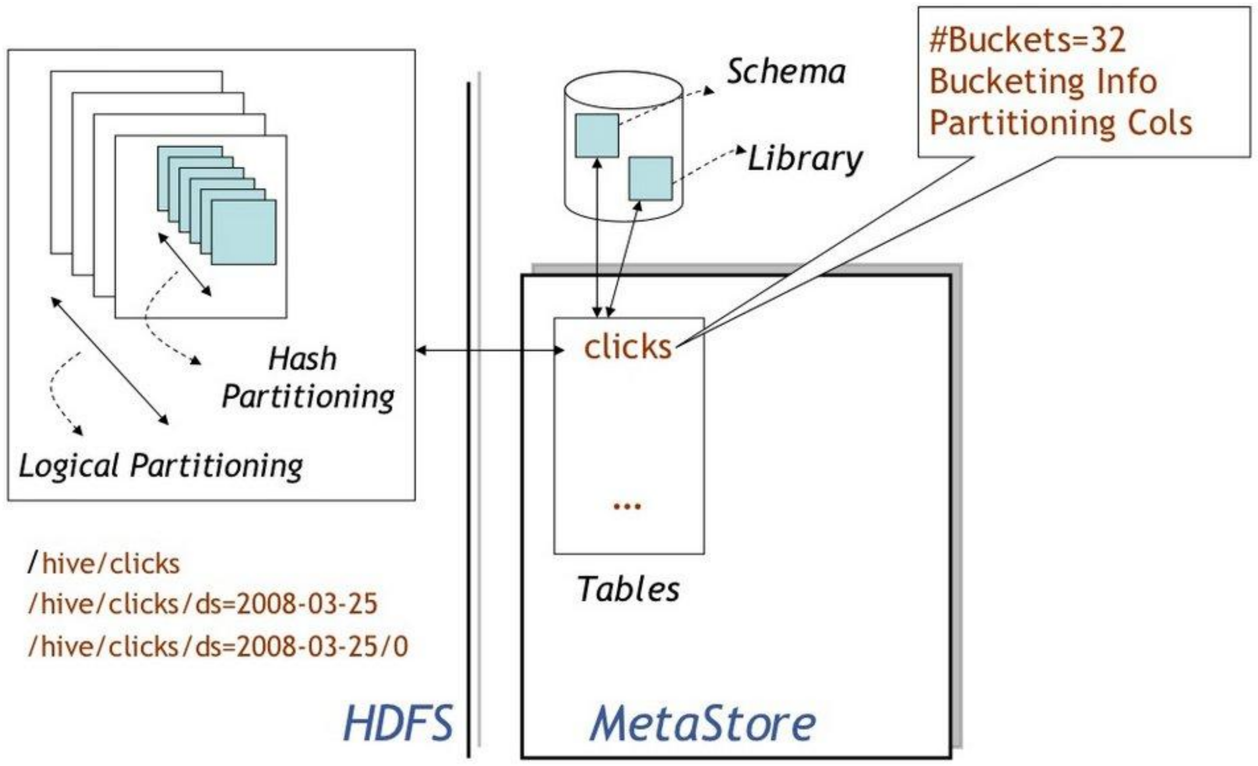
Pig Latin

```
A = LOAD 'file1' AS (x, y, z);
B = LOAD 'file2' AS (t, u, v);
C = FILTER A by y > 0;
D = JOIN C BY x, B BY u;
E = GROUP D BY z;
F = FOREACH E GENERATE
      group, COUNT(D);
STORE F INTO 'output';
```



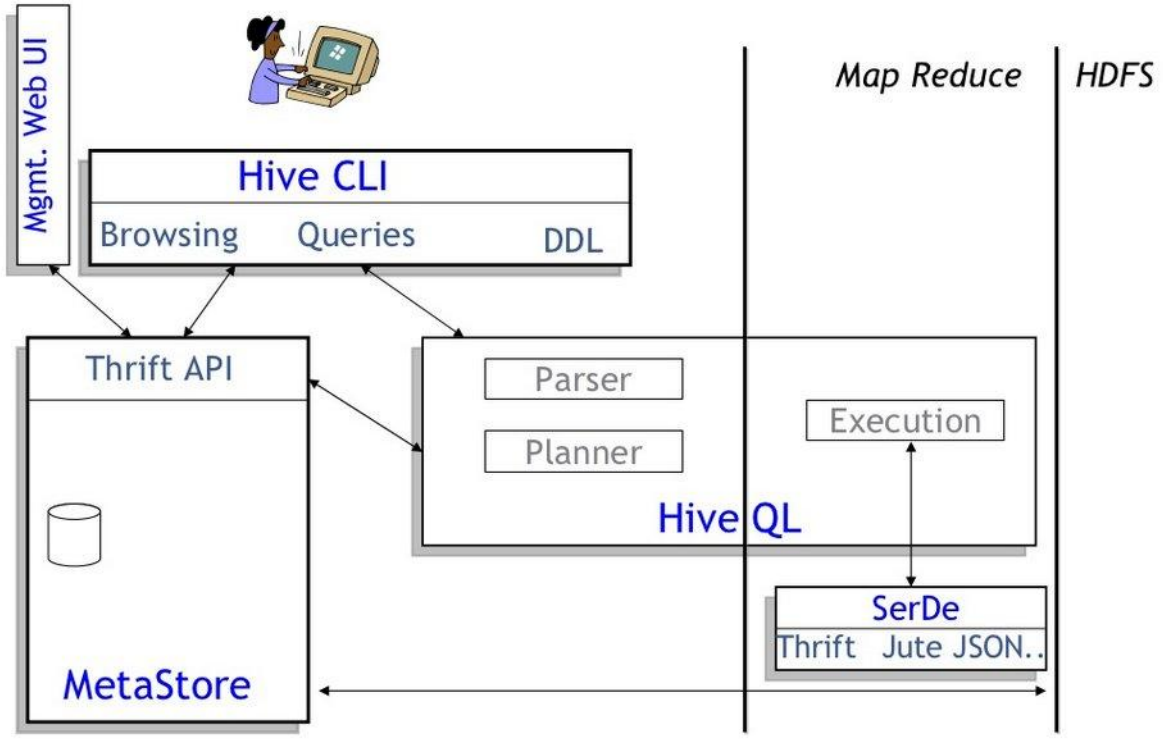
# Hive ...

- **Tables**: Each table has a corresponding directory on HDFS.
- **Partitions**: distribution of data within sub-directories of a table directory.
- **Buckets**: files in partition dir, based on the hash of a column in the table.



# Hive ...

- Apache **Thrift** software framework, for scalable cross-language services development
- **DDL**:Hive Data Definition Language: create table, show, alter ...



## User Defined Function ...

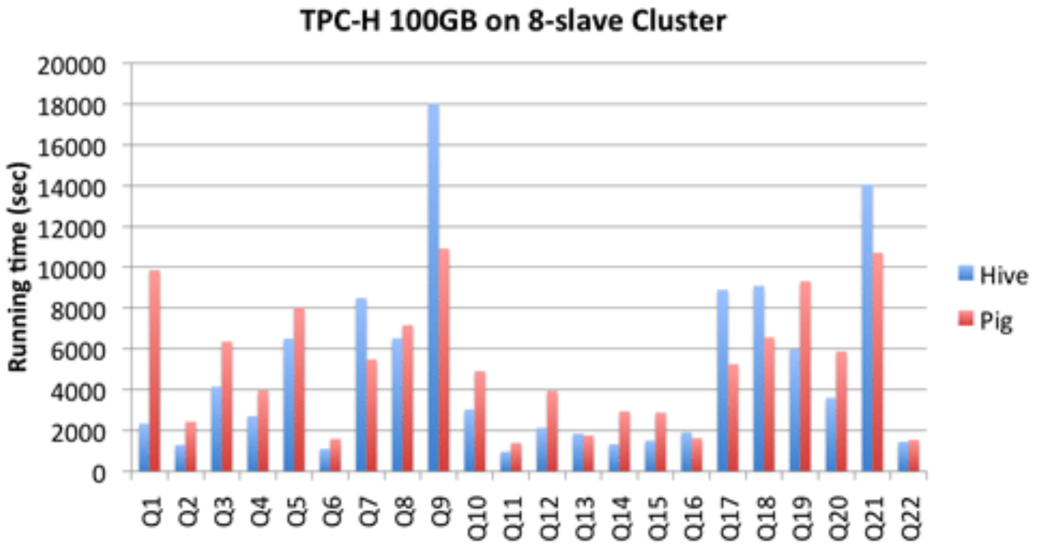
- way to do an operation on a field or fields
- called from within a pig script
- b = FOREACH a GENERATE **foo**(color)

- Implementation: just think about a small problem
- Many collaborators: Linkedin DataFU, Twitter Elephant-bird, etc.

# Performance ...

Twitter : Typically a Pig script is 5% of the code of native MapReduce written in about 5% of the time. However, queries typically take between 110-150% the time to execute that a native MapReduce job would have taken.

Hortonworks benchmark:



## Final word ...

- Not a database, read-only operations.
- Load data you tell how the data is organized, deserialize the data.
- They work with Hadoop and HDFS.

Piazza

- Reliability
- Popularity

# MegaPipe: A New Programming Interface for Scalable Network I/O

OSDI'12

# Cloud ...

- Imagine a typical Web 2.0 data-center (e.g. facebook, Twitter, Wikipedia, Youtube, etc.)

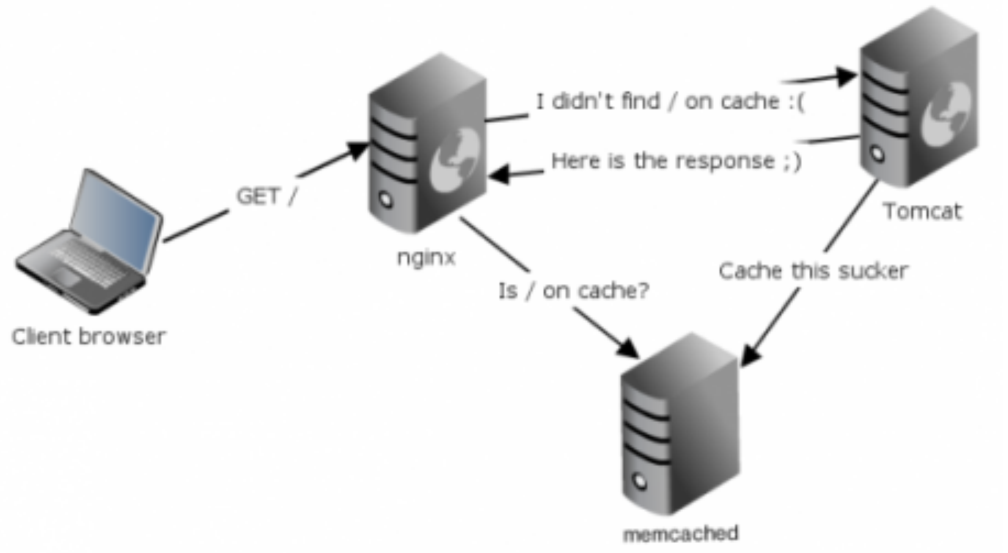
- web applications access the database to retrieve and update user data

- However, the database can become a bottleneck.

- ... so a cache is used to store the results of database queries.

- cache itself must be highly scalable

- Memcached is an example of a scale-out system: adding additional servers to a Memcached cluster linearly increases the capacity of the cache.



# Early Motivation

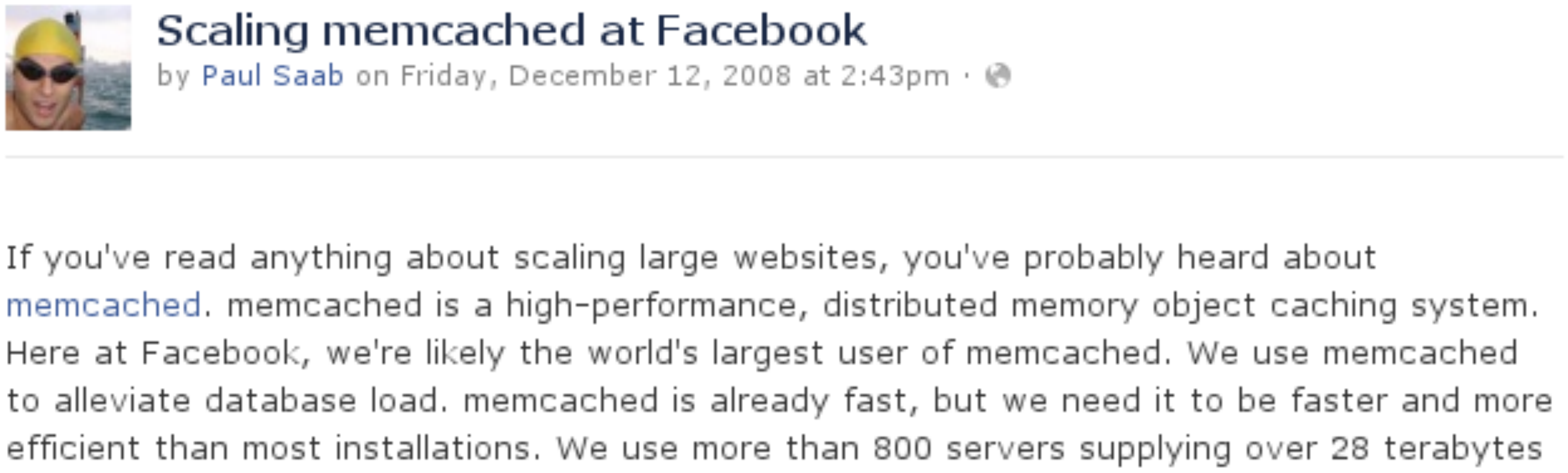
```
function get_foo(int userid) {
    data = db_select("SELECT * FROM users WHERE userid = ?", userid);
    return data;
}
```

**Memcached:**

- **provides a giant distributed hash table.**
- **When the table is full: LRU order.**
- **Applications**
  - **first check memcached (RAM)**
  - **then fetch from slower backing store (DB, Disk)**

```
function get_foo(int userid) {
    /* first try the cache */
    data = memcached_fetch("userrow:" + userid);
    if (!data) {
        /* not found : request database */
        data = db_select("SELECT * FROM users WHERE userid = ?", userid);
        /* then store in cache until next get */
        memcached_add("userrow:" + userid, data);
    }
    return data;
}
```

## Early Motivation



Scaling memcached at Facebook

by Paul Saab on Friday, December 12, 2008 at 2:43pm ·

If you've read anything about scaling large websites, you've probably heard about memcached. memcached is a high-performance, distributed memory object caching system. Here at Facebook, we're likely the world's largest user of memcached. We use memcached to alleviate database load. memcached is already fast, but we need it to be faster and more efficient than most installations. We use more than 800 servers supplying over 28 terabytes

**1. Per-connection buffer to read, write data: a lot of memory => a per thread shared connection buffer pool**

**2. Considerable lock contention over UDP socket locks => one reply socket per thread**

**3. One core get saturated, doing network interrupt handling => distribute network processing evenly across cores.**

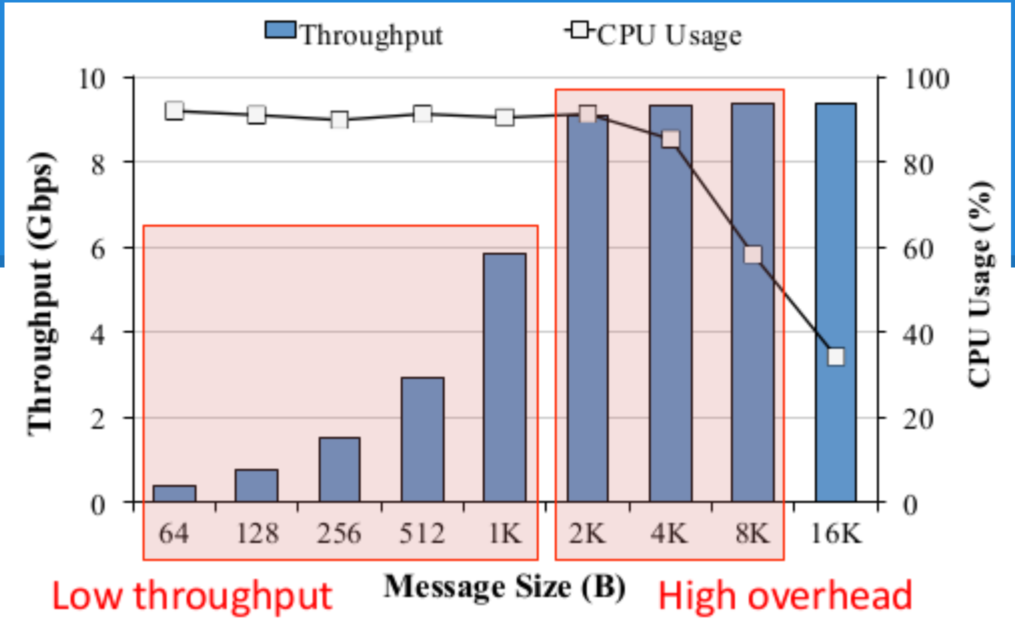
**4. Their machines: 8-core. A big contention on the lock that protects each network device's transmit queue => better algorithm with less locking**

## Motivation

1. Bulk-transfer workload
   - One way, large data transfer
     - Video streaming, HDFS
   - Cheap
2. Message-oriented workload
   - Short connections or small messages
     - HTTP, RPC, DB, key-value stores, ...
   - CPU-intensive

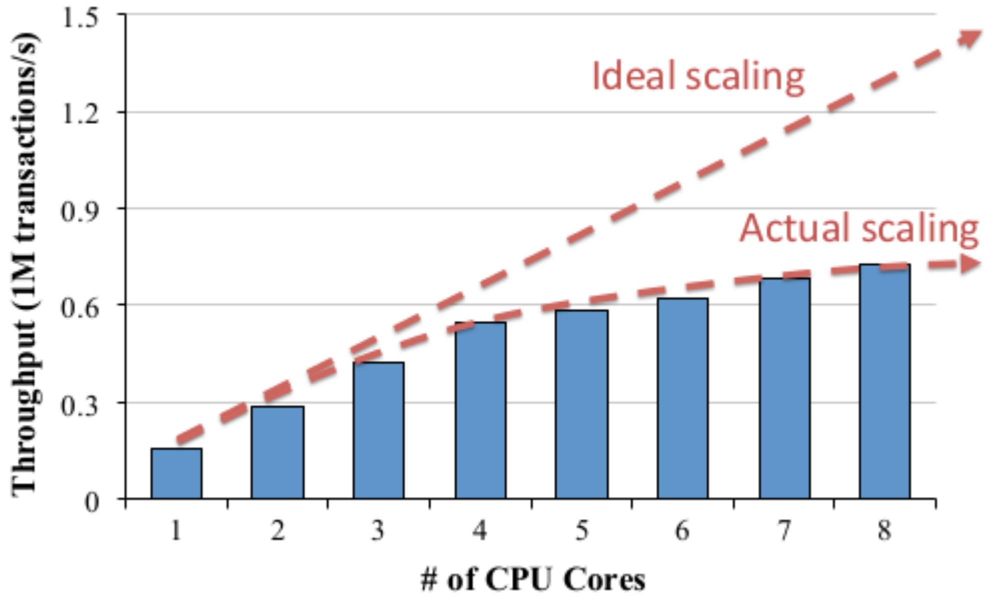
Existing network API on multi-core systems

- Inefficient for "message-oriented" workloads
- Hard time to scale to high connection rate



Low throughput of small messages

Not efficient use of multi-cores



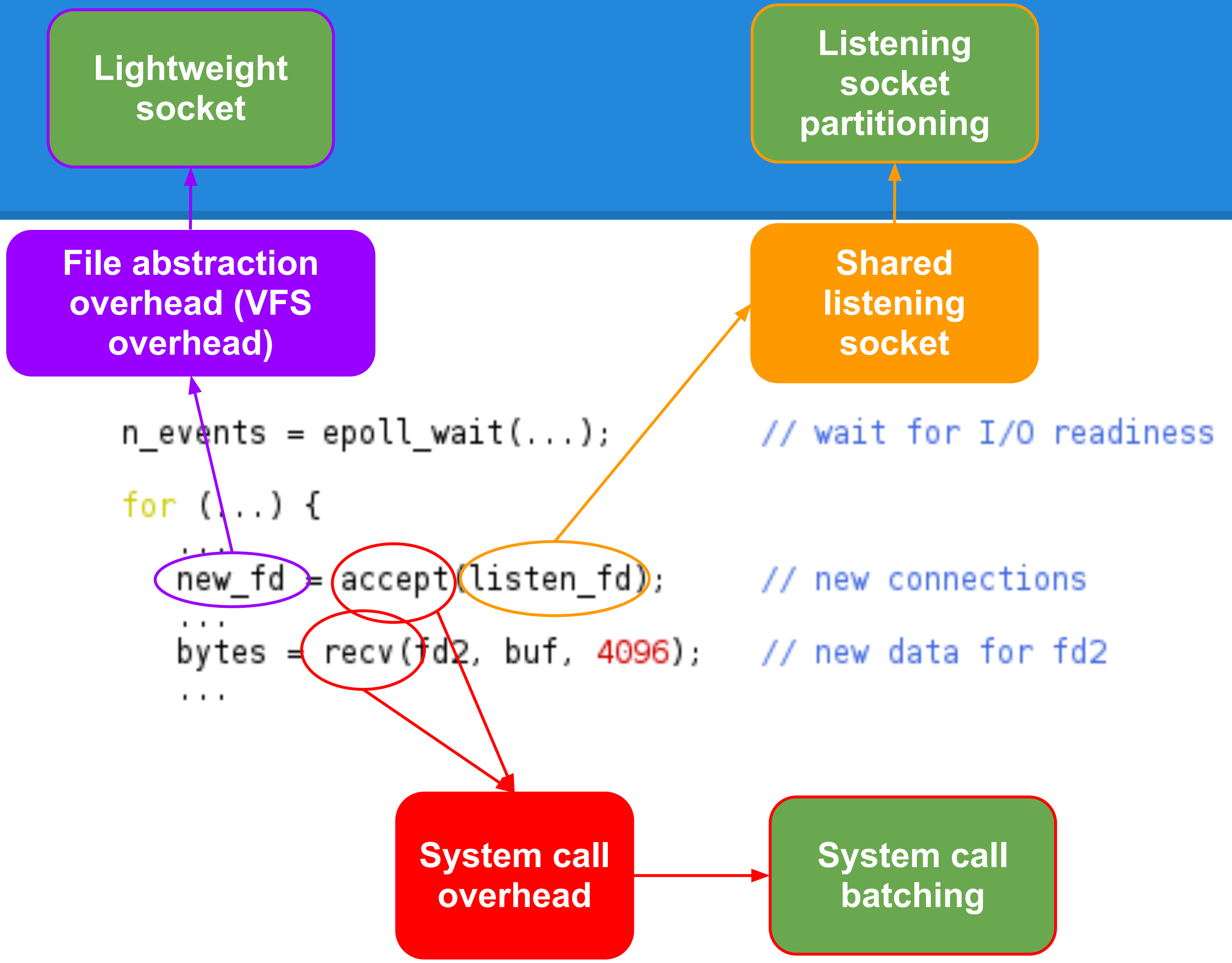
## Intro

- Networking: Central role in modern applications

- MegaPipe: A new API for efficient, scalable network I/O.Especially I/O intensive workloads

- More on Operating System, not Distributed Systems

- The core abstraction: a channel
  - a per-core, bi-directional pipe between the kernel and



```
n_events = epoll_wait(...);          // wait for I/O readiness

for (...) {
  ...
  new_fd = accept(listen_fd);        // new connections
  ...
  bytes = recv(fd2, buf, 4096);      // new data for fd2
  ...
```

## Architectural Overview

- MegaPipe involves both user-space library and Linux kernel modifications.

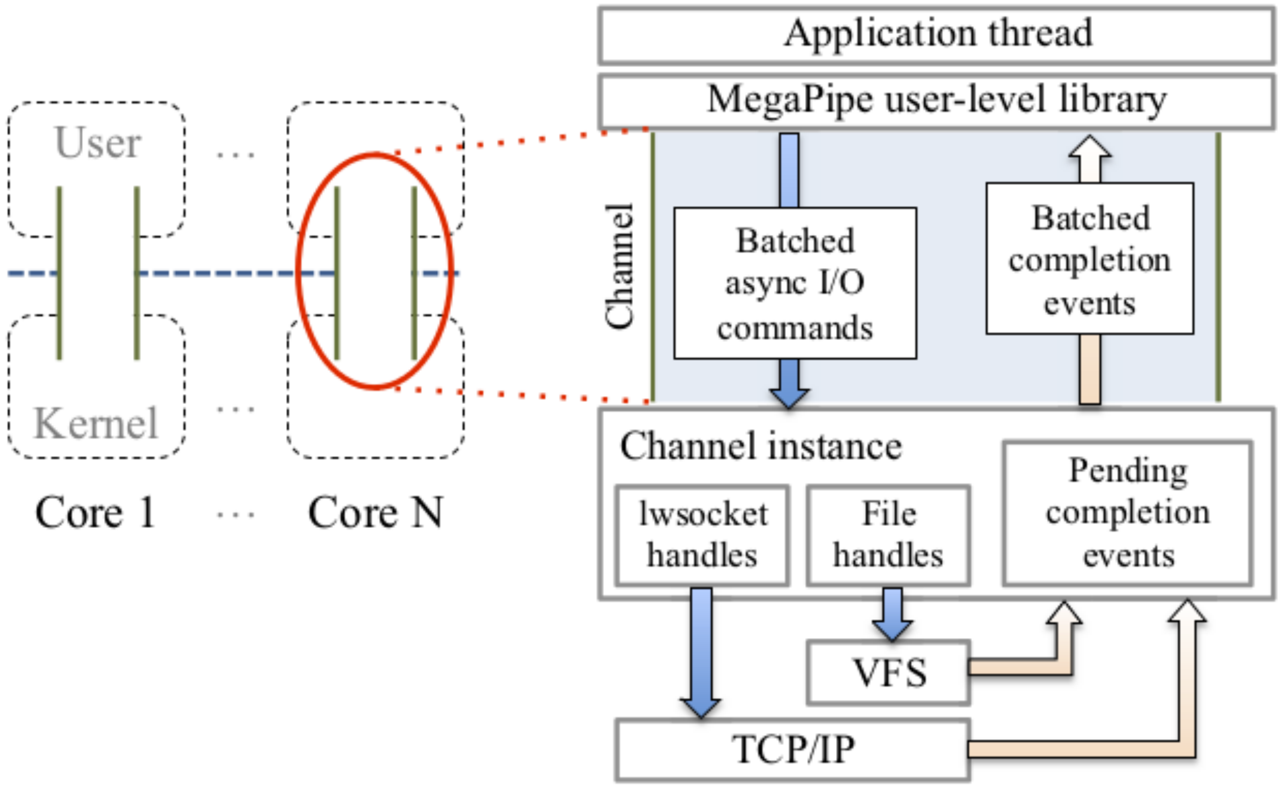
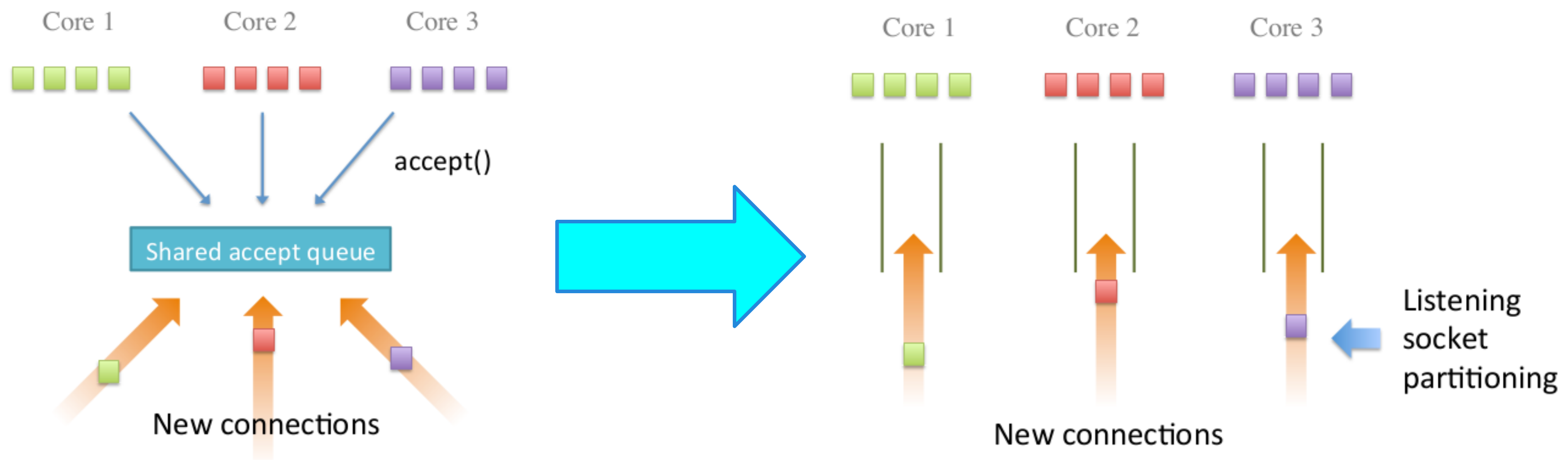

User
Kernel
Core 1
Core N
Application thread
MegaPipe user-level library
Channel
Batched async I/O commands
Batched completion events
Channel instance
lwsocket handles
File handles
Pending completion events
VFS
TCP/IP

**Figure 2:** MegaPipe architecture

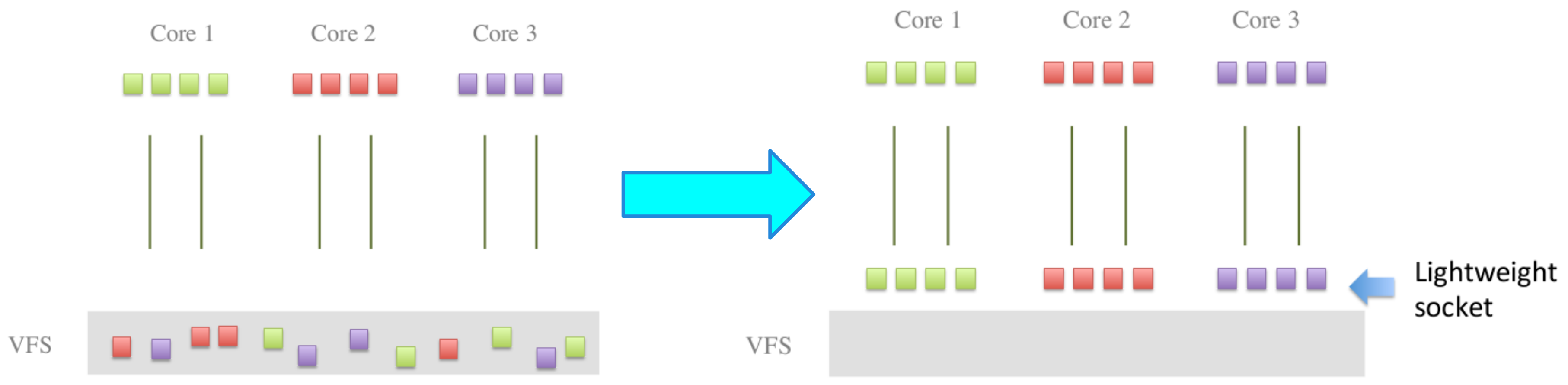
## Listening Socket Partitioning

- When an application thread associates a listening socket to a channel, MegaPipe spawns a separate listening socket.



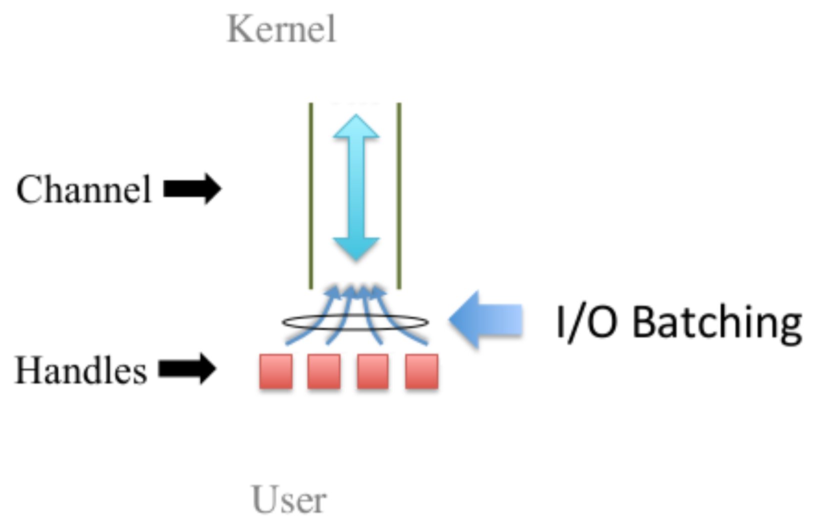
## lwsocket: Lightweight Socket

- In UNIX, many different types of open files:
  - disk files, sockets, pipes, devices
- This is an overkill for network sockets:
  - Sockets are rarely shared
  - Sockets are ephemeral
- Lwsocket: Do not create a file instance

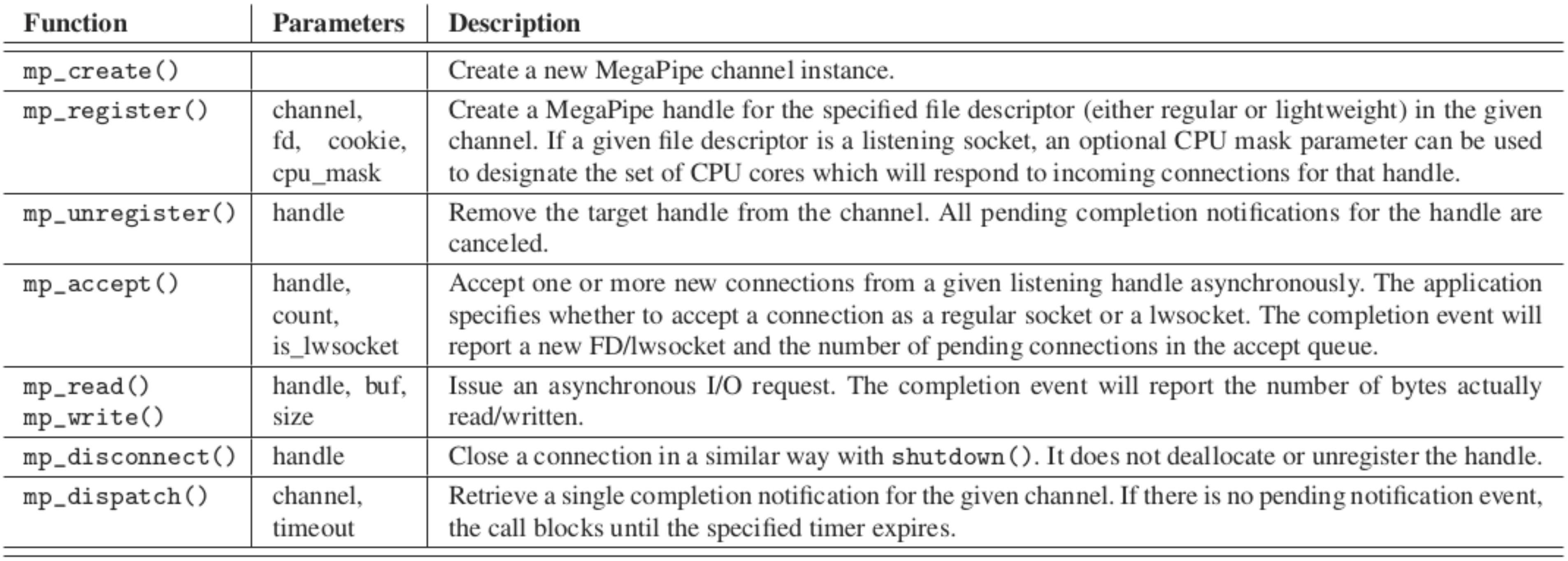


# System Call Batching

- System calls:
  - Cost of mode switching => Expensive
  - Negative effect on cache locality => Inefficient
- To amortize system call costs, MegaPipe:
  - Batches multiple I/O requests and their completion notifications into a single system call.



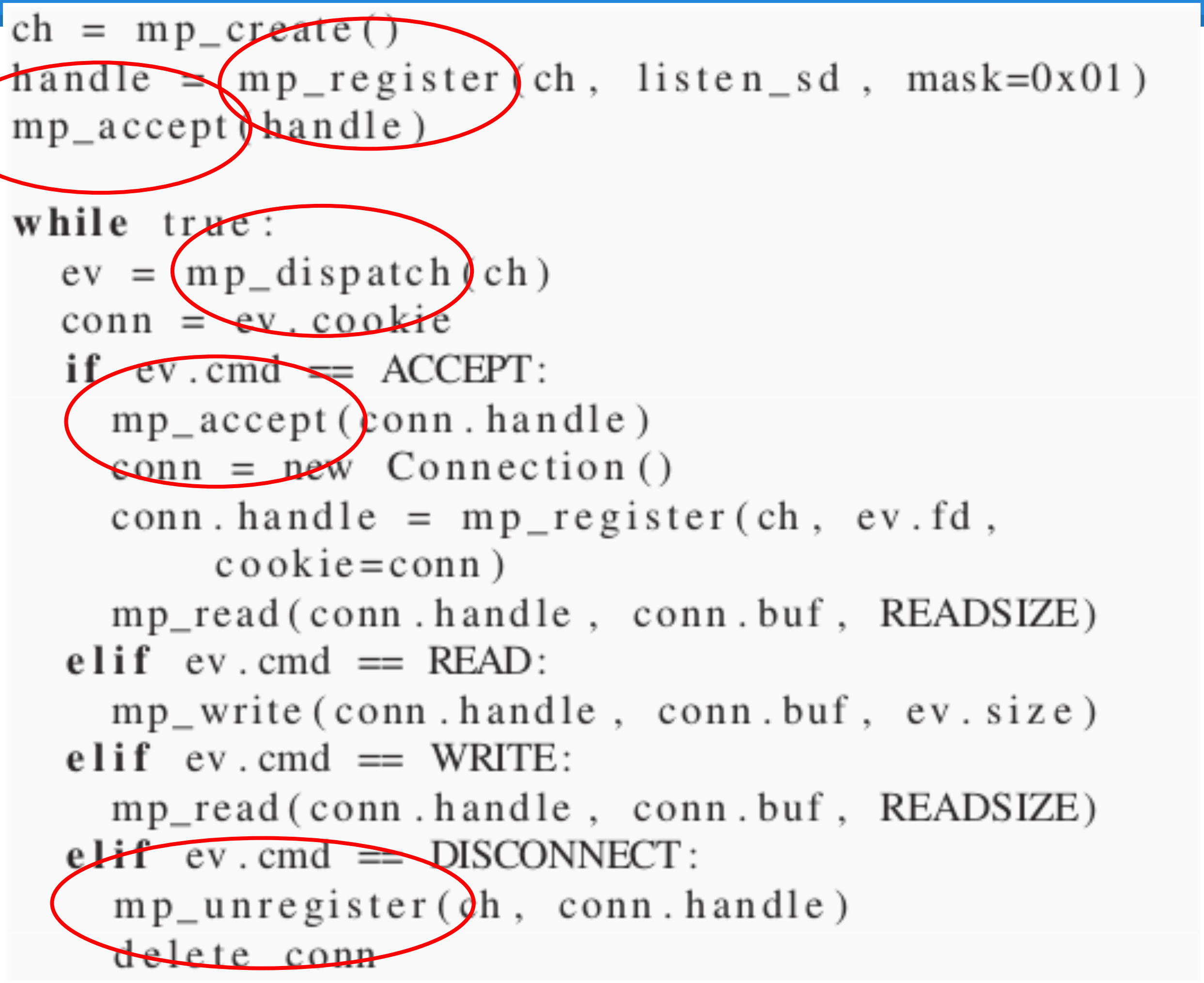
# API



| Function | Parameters | Description |
|---|---|---|
| `mp_create()` | | Create a new MegaPipe channel instance. |
| `mp_register()` | channel, fd, cookie, cpu_mask | Create a MegaPipe handle for the specified file descriptor (either regular or lightweight) in the given channel. If a given file descriptor is a listening socket, an optional CPU mask parameter can be used to designate the set of CPU cores which will respond to incoming connections for that handle. |
| `mp_unregister()` | handle | Remove the target handle from the channel. All pending completion notifications for the handle are canceled. |
| `mp_accept()` | handle, count, is_lwsocket | Accept one or more new connections from a given listening handle asynchronously. The application specifies whether to accept a connection as a regular socket or a lwsocket. The completion event will report a new FD/lwsocket and the number of pending connections in the accept queue. |
| `mp_read()` `mp_write()` | handle, buf, size | Issue an asynchronous I/O request. The completion event will report the number of bytes actually read/written. |
| `mp_disconnect()` | handle | Close a connection in a similar way with `shutdown()`. It does not deallocate or unregister the handle. |
| `mp_dispatch()` | channel, timeout | Retrieve a single completion notification for the given channel. If there is no pending notification event, the call blocks until the specified timer expires. |

**Table 1:** MegaPipe API functions (not exhaustive).

# Ping-pong server pseudocode



```
ch = mp_create()
handle = mp_register(ch, listen_sd, mask=0x01)
mp_accept(handle)

while true:
  ev = mp_dispatch(ch)
  conn = ev.cookie
  if ev.cmd == ACCEPT:
    mp_accept(conn.handle)
    conn = new Connection()
    conn.handle = mp_register(ch, ev.fd,
          cookie=conn)
    mp_read(conn.handle, conn.buf, READSIZE)
  elif ev.cmd == READ:
    mp_write(conn.handle, conn.buf, ev.size)
  elif ev.cmd == WRITE:
    mp_read(conn.handle, conn.buf, READSIZE)
  elif ev.cmd == DISCONNECT:
    mp_unregister(ch, conn.handle)
    delete conn
```

**Listing 1:** Pseudocode for ping-pong server event loop

## Microbenchmarks: Performance

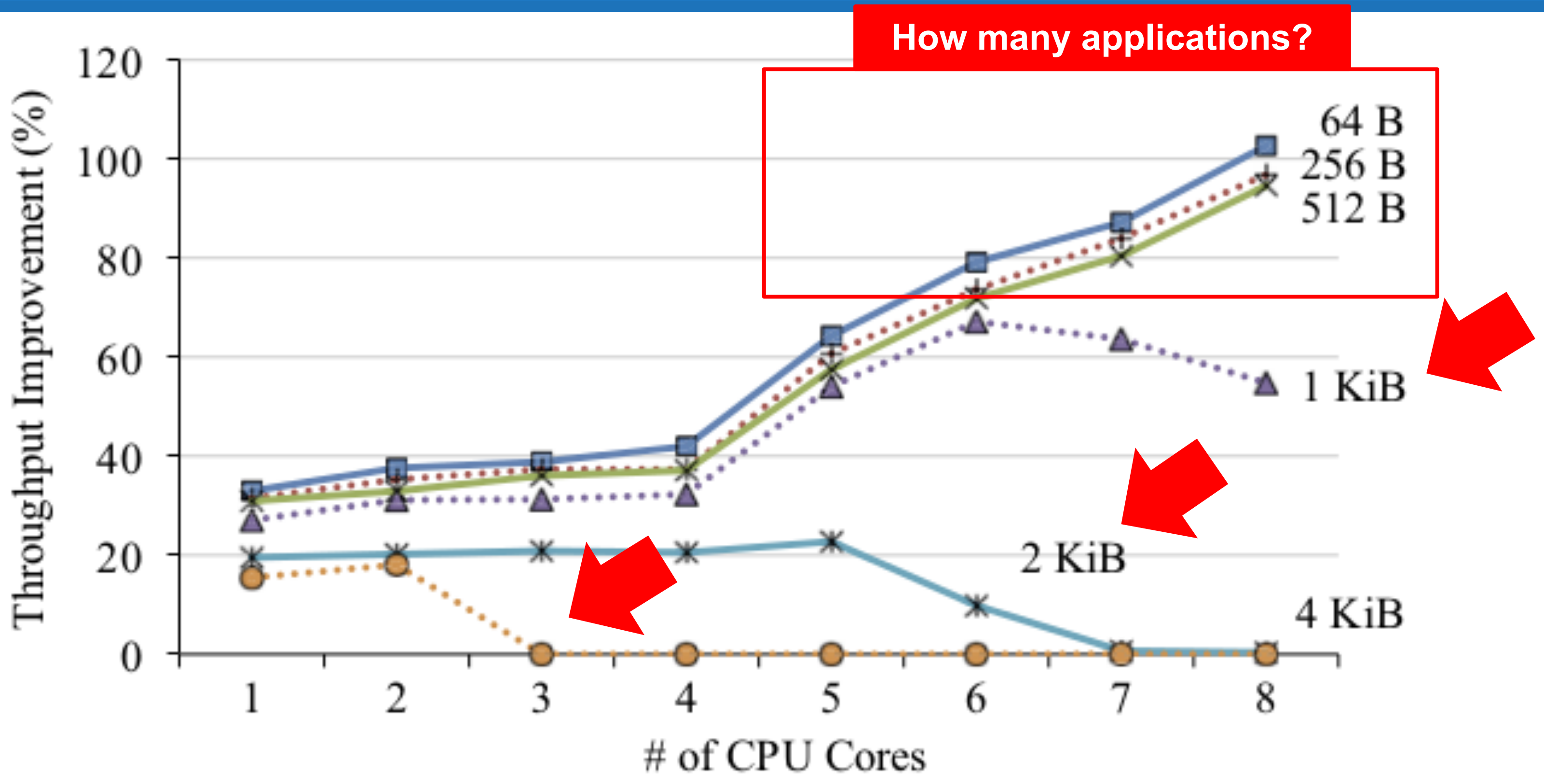


**Figure 4:** Relative performance improvement for varying message sizes over a range of CPU cores.

## Microbenchmarks: Multi-core scalability

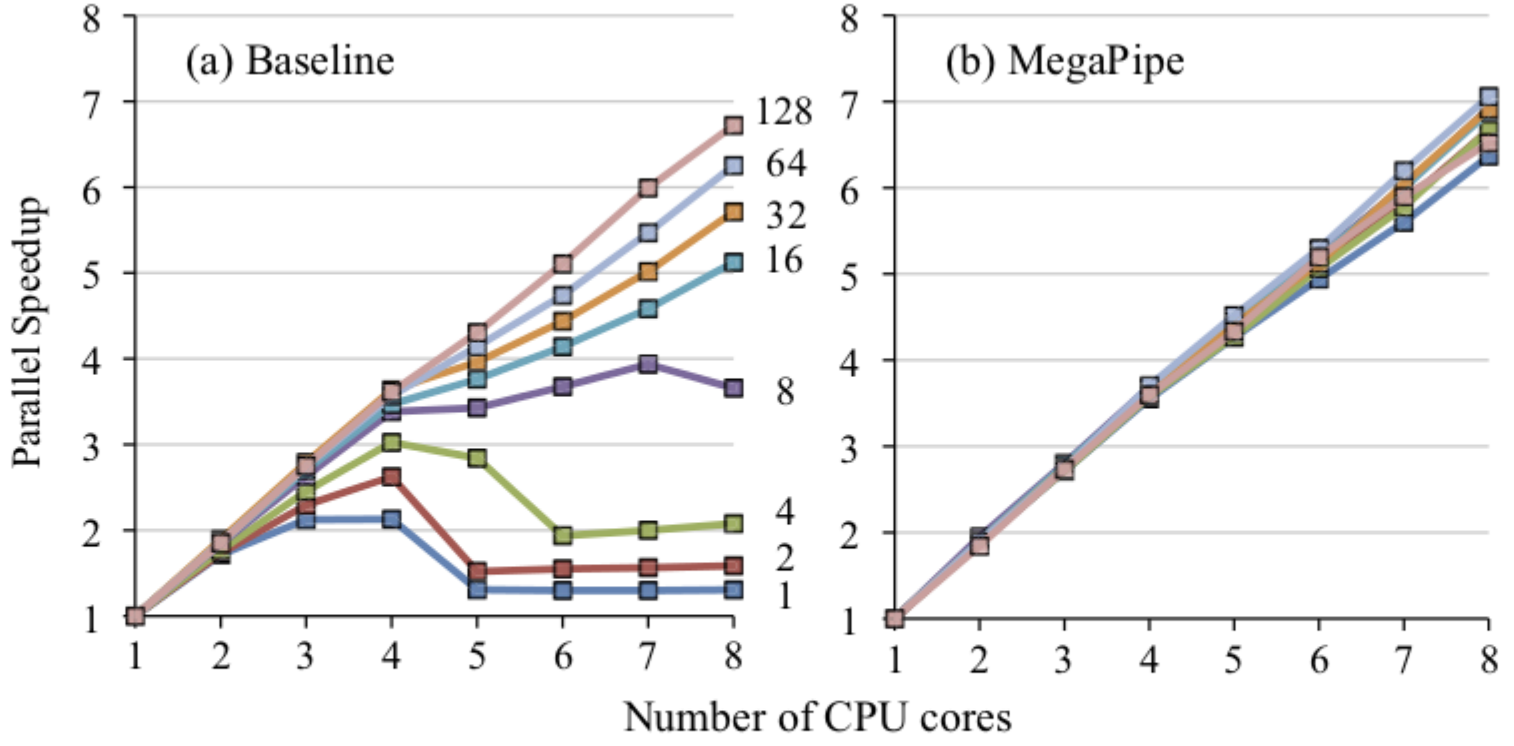


**Figure 3:** Comparison of parallel speedup for varying numbers of transactions per connection (labeled) over a range of CPU cores (x-axis) with 64 B messages.

# Macrobenchmarks: memchached

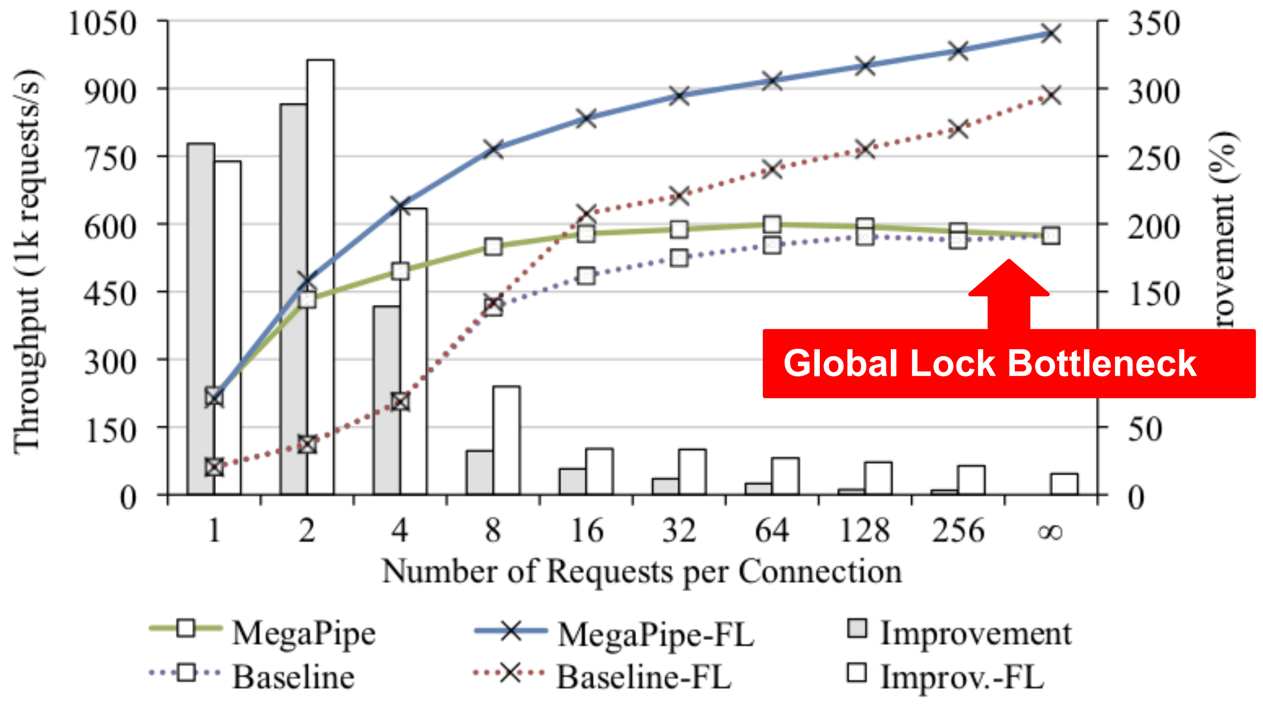


**Figure 5:** memcached throughput comparison with eight cores, by varying the number of requests per connection. ∞ indicates persistent connections. Lines with "X" markers (-FL) represent fine-grained-lock-only versions.

# MacroBenchmarks: nginx

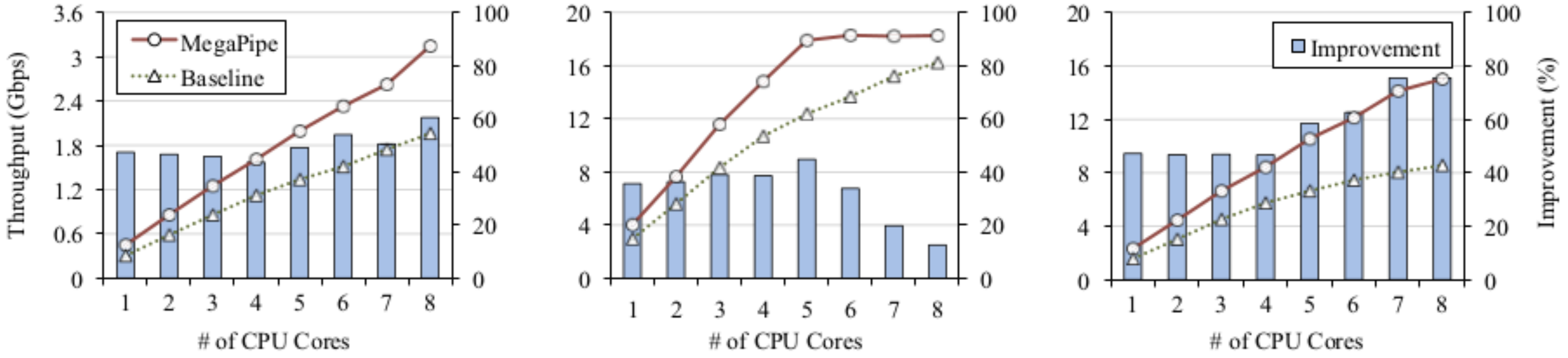


**Figure 7:** Evaluation of nginx throughput for the (a) SpecWeb, (b) Yahoo, and (c) Yahoo/2 workloads.

## Conclusion

- Short connections or small messages:
    - High CPUoverhead
    - Poorly scaling with multi-core CPUs

- MegaPipe
    - Key abstraction: per--core channel
    - Enabling three optimization opportunities:
        - Batching, partitioning, lwsocket

## Thoughts/criticism

- **Multi-core Era!**
- **Legacy Code:** The main question is, are people willing to change their code to this new API?
- **Evaluation?** Why not more real-world applications?

## Discussion Points

- **From Piazza:**
  - Compared to HW solutions (infiniband)
  - Delay of batching?
  - Will it be as great as it is right now for VMs?
  - All the API is asynchronous!
  - "applications should control sharing"
- **More?**
  - Let's discuss here in the class...

## Backup Slides

## Implementation

- MegaPipe consists of two parts:
  - Kernel Module: MP-K
  - User-level Library: MP-L
  - Interaction: Through a special device: /dev/megapipe
- MP-K
  - Maintains set of handles for both FDs and lwsockets in a red-black tree for each channel.
  - Depending on the file type, it handles async I/O.
- Upon receiving batched I/O commands from MP-L through a channel, MP-K checks if they can be processed immediately.
  - If yes, process it and issue a completion notification
  - If no, bookkeep, register a callback to the socket or declare epoll interest for other file types.