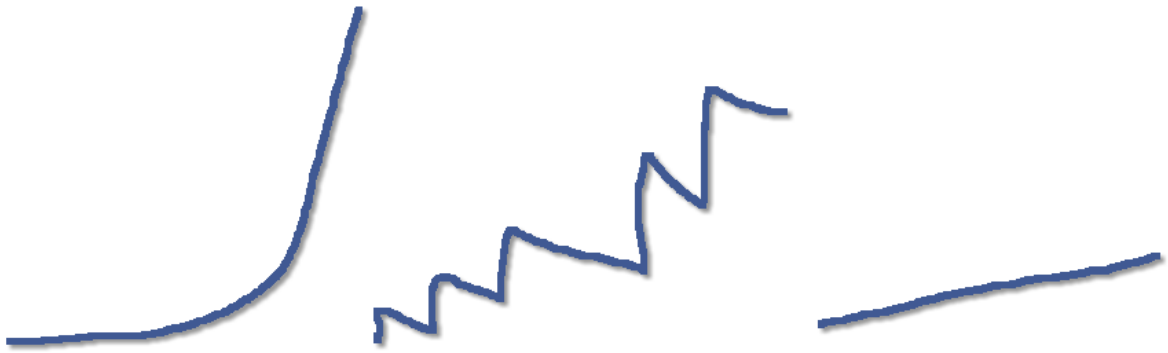
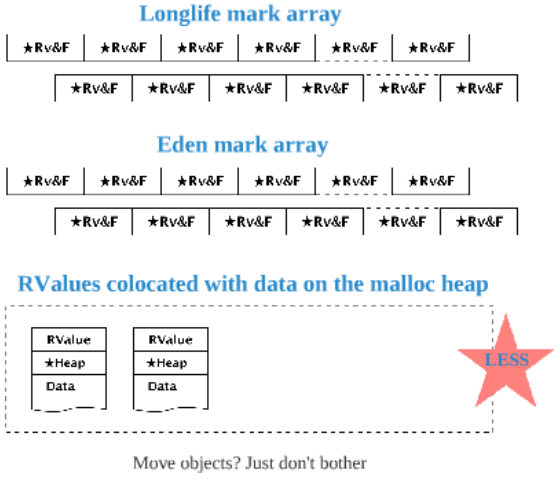
Twitter's Growth

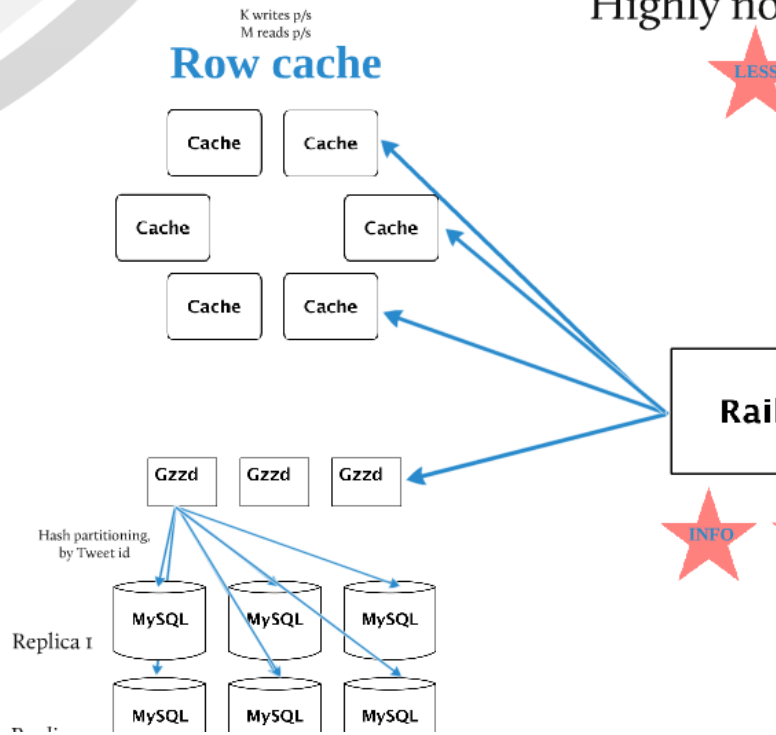


Performance Engineering at Twitter

Evan Weaver
@evan

Future Ruby



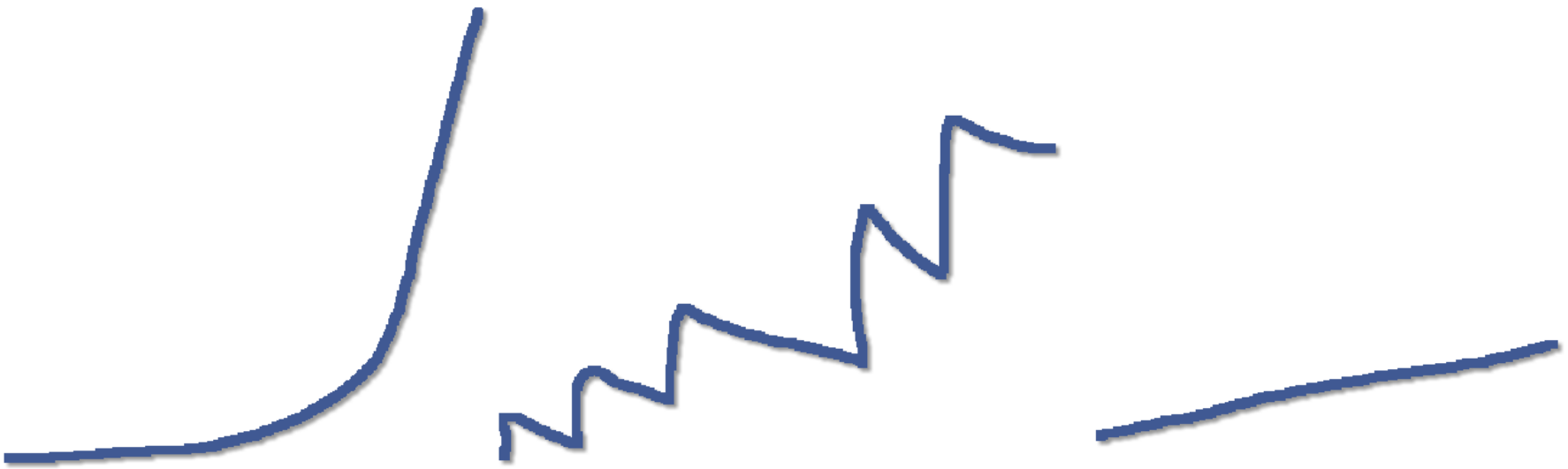


Twitter's Growth

Usage

FE throughput

Hardware



formance Engineering at T

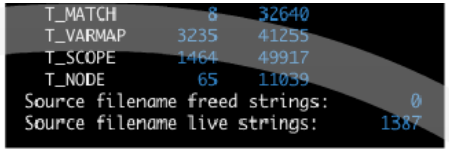
Evan Weaver

K writes p/s

Just do less


LESS


MEM
RValue
RValue
RValue
RValue
RValue
Offsite dat
HTTP request/response ratio
120
100



Focus on memory

Access data explicitly and in bulk

Let the code make informed decisions

Replica 1

Replica 2

Performance Engineering at

Evan Weaver
@evan

Just do less

LESS

Ruby 1.8

MEM

Single Ruby heap space

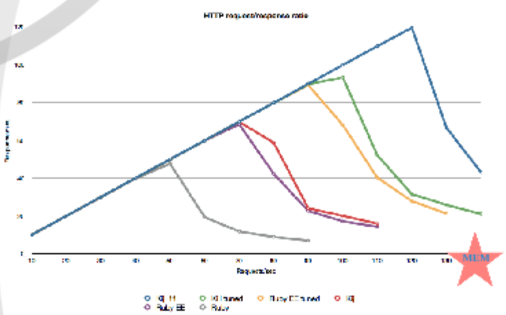
RValue RValue RValue RValue RValue

Malloc heap

Offsite data (Strings, Arrays, Hashes)

Pays the mark and sweep cost for unfreeable objects again and again

Collects frequently, just in case



Kiji 0.11

MEM LESS

Longlife Ruby heap

RValue RValue RValue RValue RValue RValue

Mark barrier set

Eden Ruby heap

RValue RValue RValue RValue RValue RValue

Malloc heap

Offsite data (Strings, Arrays, Hashes)

Longlife context:
AST compilation, constant assignment, singleton assignment

Collects when fixed-size Ruby heaps are full

MEM

Future Ruby

MEM INFO

Longlife mark array

Eden mark array

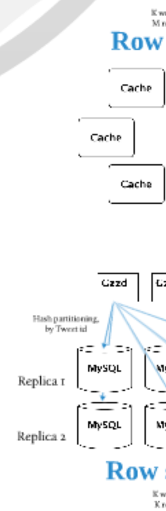
RValues colocated with data on the malloc heap

LESS

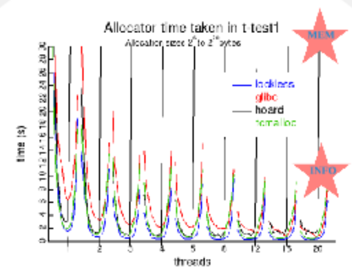
Move objects? Just don't bother

Collects based on RSS pressure

⟵ Ruby GC



Tweet

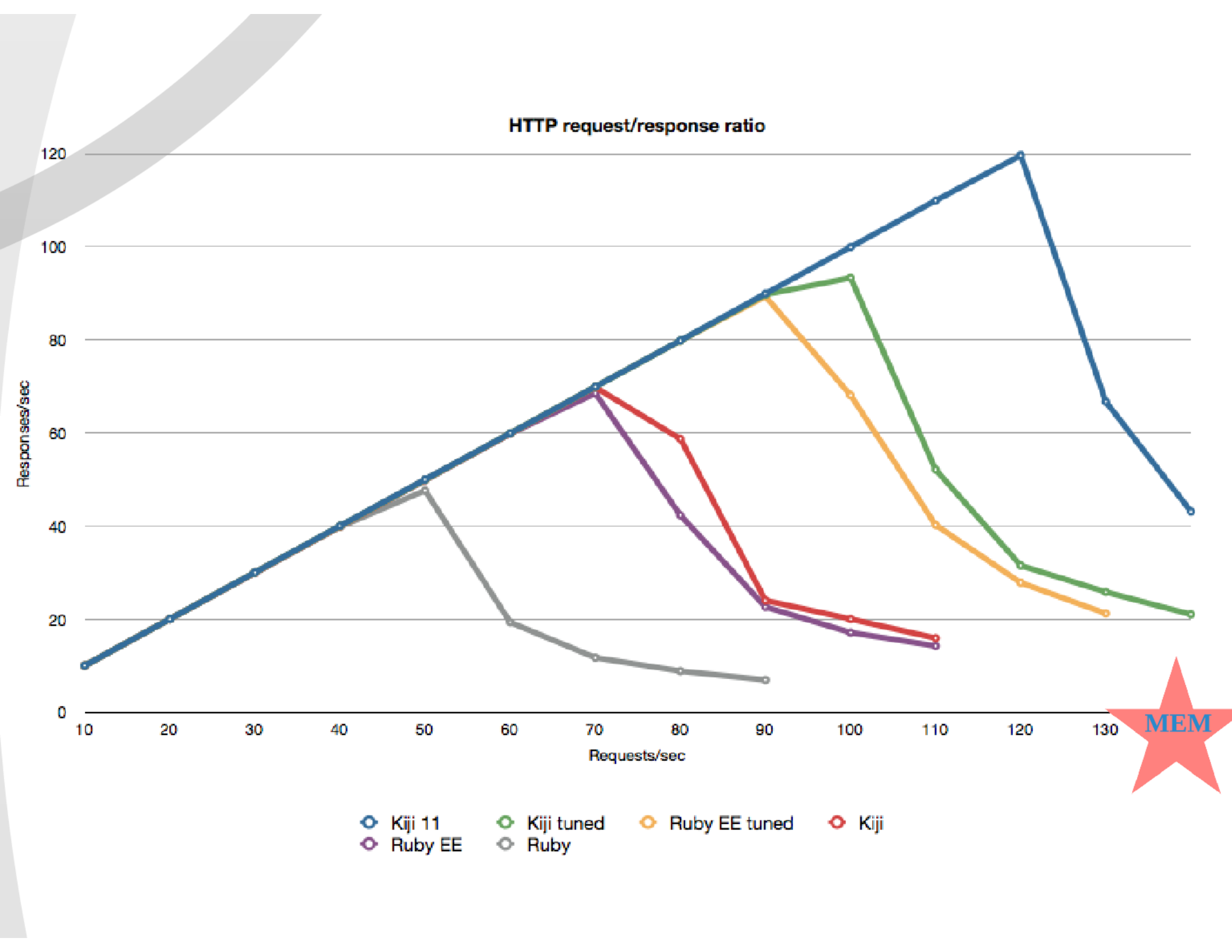


TCMalloc for unthreaded throughput,
JEMalloc otherwise

Malloc

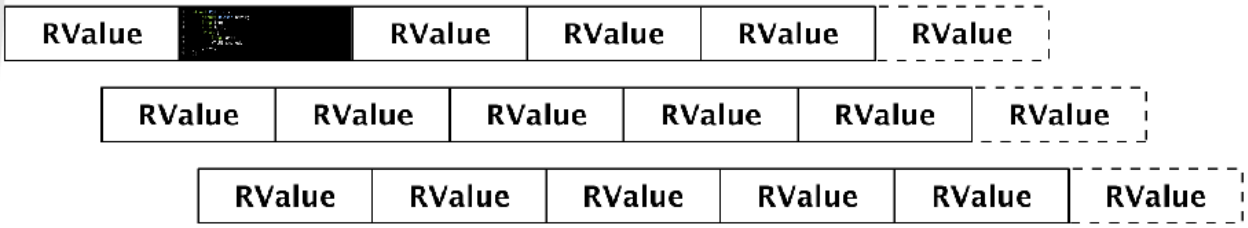
Events are
serve

Conc


HTTP request/response ratio
120
100
80
60
40
20
0
Responses/sec
10
20
30
40
50
60
70
80
90
100
110
120
130
Requests/sec
Kiji 11
Kiji tuned
Ruby EE tuned
Kiji
Ruby EE
Ruby
MEM

Ruby 1.8

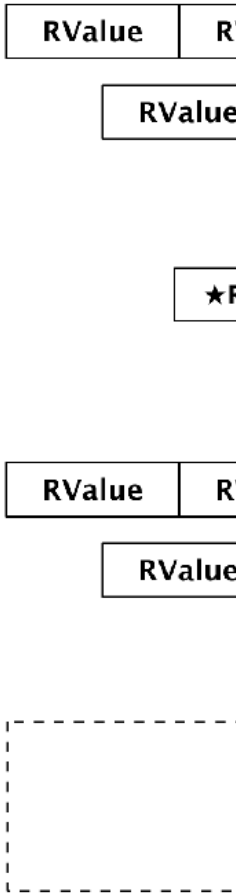
Single Ruby heap space



Malloc heap

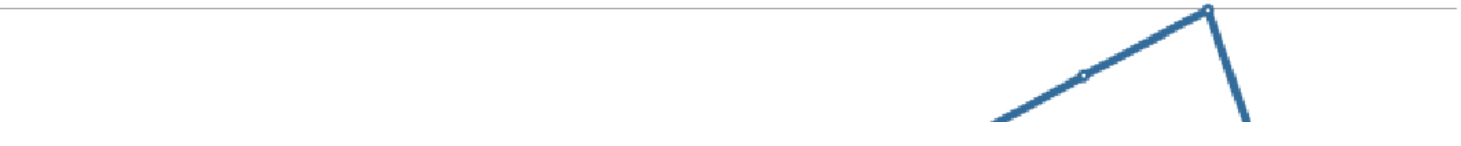
Pays the mark and sweep cost for
unfreeable objects again and again

Collects frequently, just in case



MEM

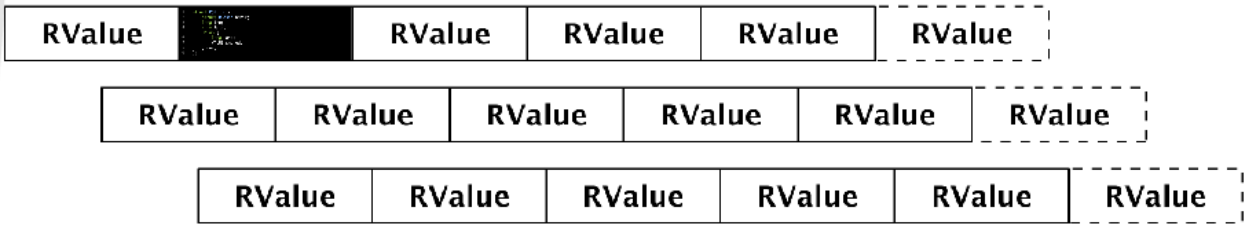
HTTP request/response ratio



```
1   struct RString {
2       struct RBasic basic;
3       long len;
4       char *ptr;
5       union {
6           long capa;
7           VALUE shared;
8       } aux;
9   };
```

Ruby 1.8

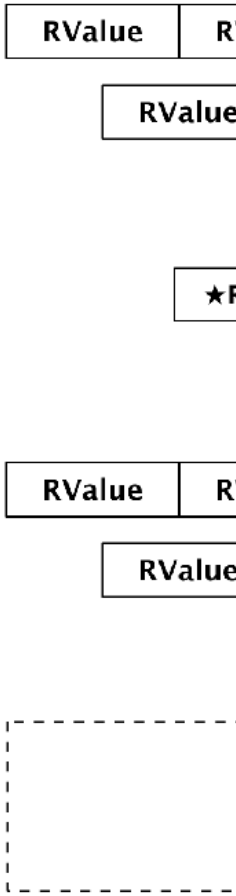
Single Ruby heap space



Malloc heap

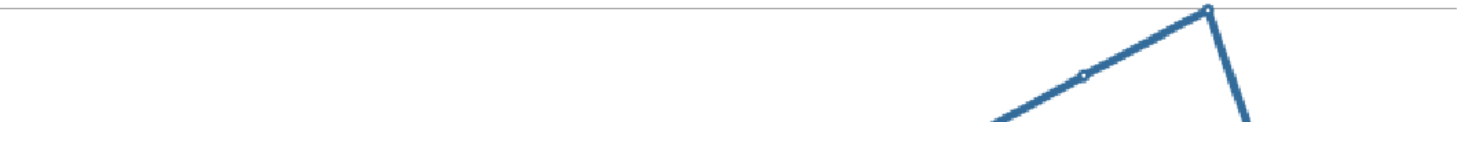
Pays the mark and sweep cost for
unfreeable objects again and again

Collects frequently, just in case



MEM

HTTP request/response ratio



Perfor

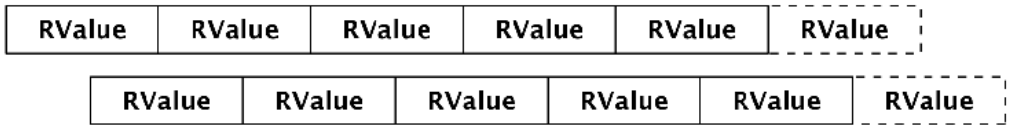
.8

MEM

Kiji o.II

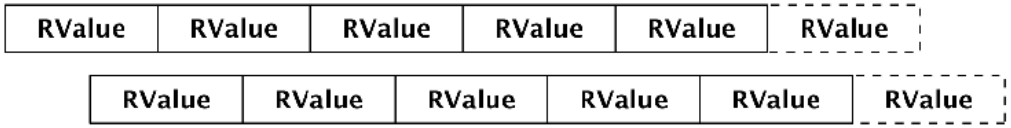
LESS

Longlife Ruby heap



Mark barrier set

Eden Ruby heap



Malloc heap

Longlife context:
AST compilation, constant assignment, singleton assignment

Collects when fixed-size Ruby heaps are full

MEM

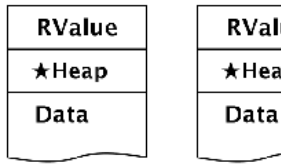
Futu

MEM

Lon

Ec

RValues colocat



Move

```
Longlife collection (after 6 edens)
Objects moved to longlife:     12098
Remembered set kept:               12014
Remembered set recycled:               3
Longlife heaps in heapspace:          44
Longlife empty heaps:                  0
Longlife total slots:            1441766
Longlife already free slots:       53822
Longlife finalized free slots:         0
Longlife live objects:           1362492
Longlife freed objects:            25452
Longlife objects summary:
   Type            Live     Freed
   T_STRING      295608     24950
   T_NODE       1066884       502
Eden collection
```

```
Eden heaps in heapspace:                64
Eden empty heaps:                        0
Eden total slots:                  2097115
Eden already free slots:            371168
Eden finalized free slots:           16481
Eden live objects:                  155539
Eden freed objects:                1553927
Eden objects summary:
  Type          Live     Freed
  T_OBJECT     39659     11249
  T_CLASS       7765       602
  T_ICLASS      3062      1907
  T_MODULE      1053         0
  T_FLOAT        302     11005
  T_STRING     37769    771351
  T_REGEXP      3769      8649
  T_ARRAY      19208 460182
  T_HASH       13081     84353
  T_STRUCT     18788     63913
  T_BIGNUM        14      5857
  T_FILE          26         0
  T_DATA        6271         8
  T_MATCH          8     32640
```

Future Ruby

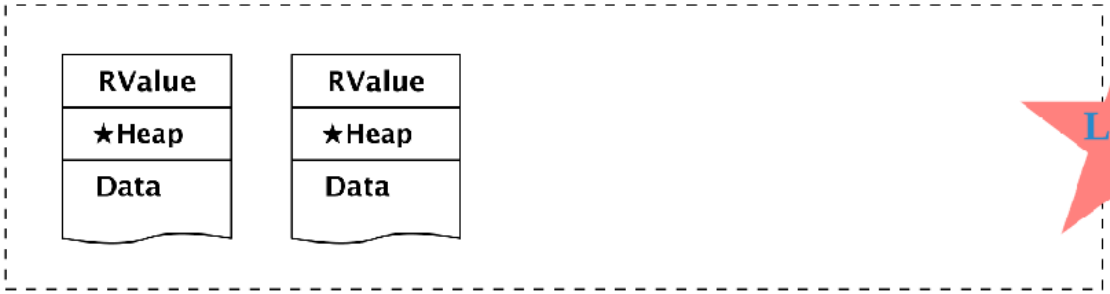
MEM

INFO

Longlife mark array

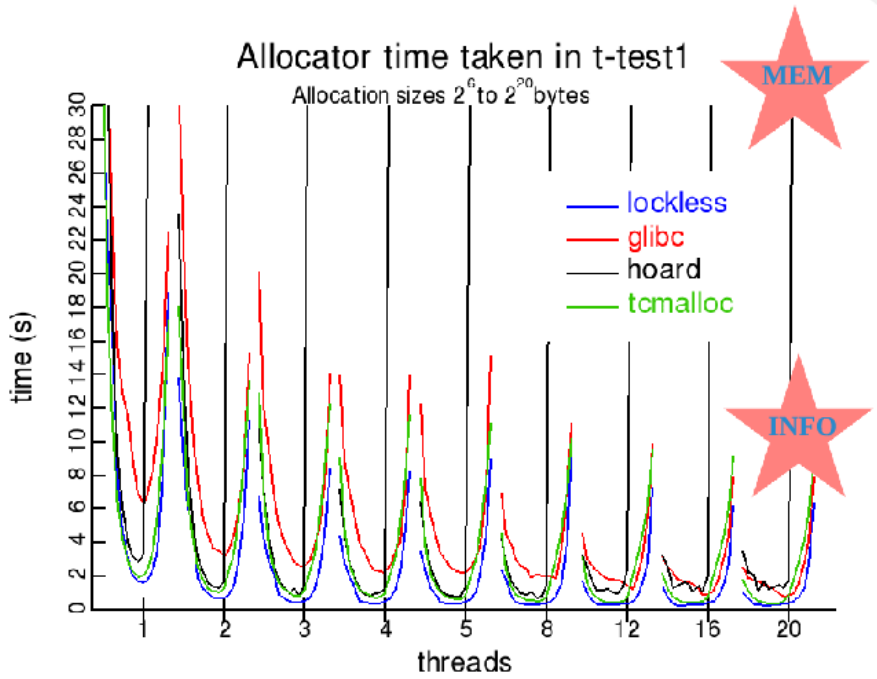
Eden mark array

RValues colocated with data on the malloc heap



Move objects? Just don't bother

Collects based on RSS pressure

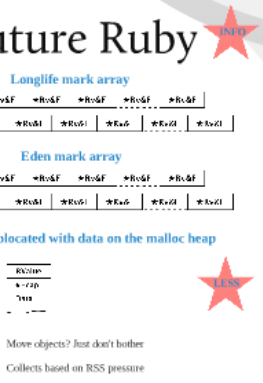


TCMalloc for unthreaded throughput,
JEMalloc otherwise

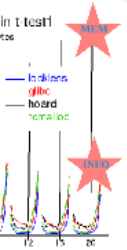
Malloc

rmance Engineering at Twitter

Evan Weaver
@evan

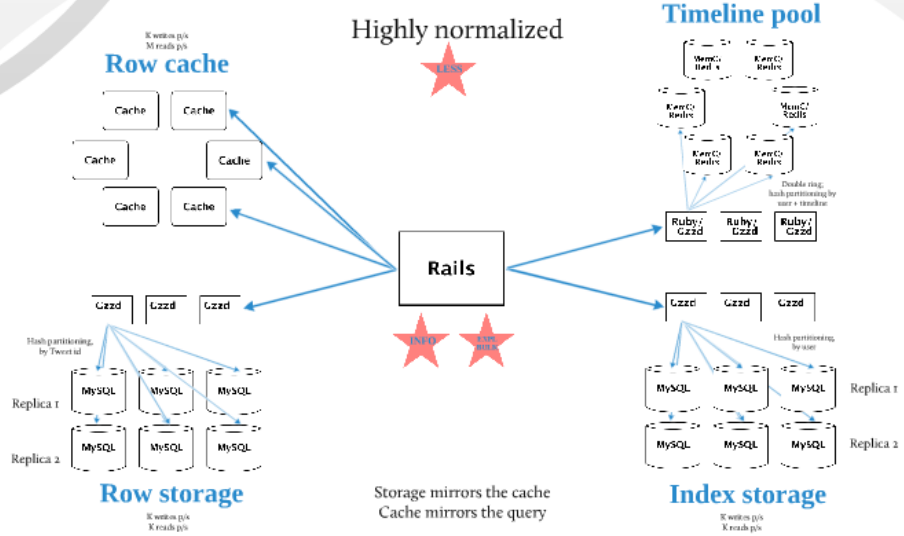


y GC

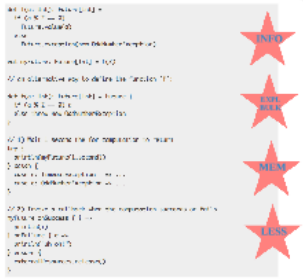


ded throughput,
herwise

oc



Tweet Storage ⟶



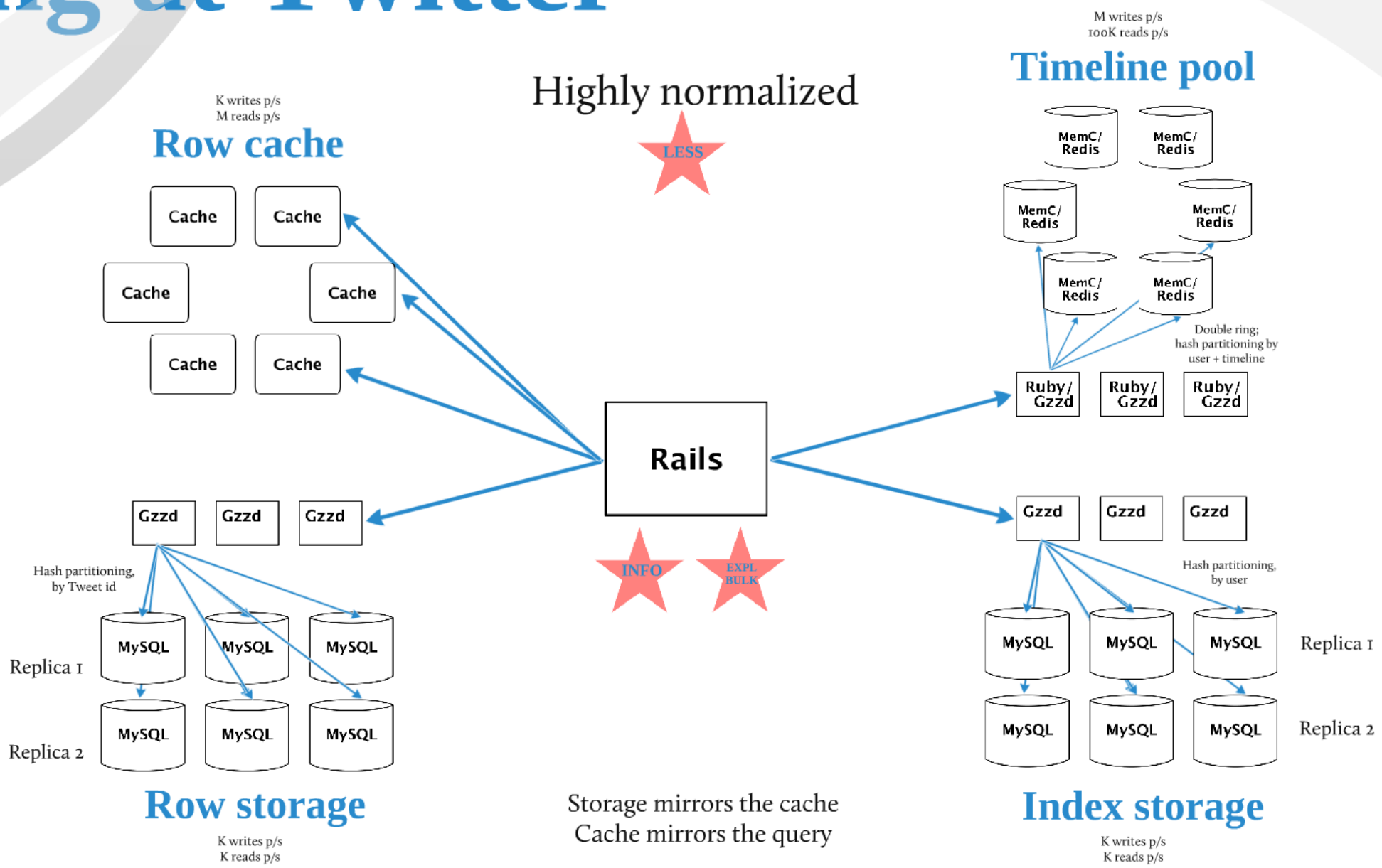
Events are the right model for server workloads

Concurrency

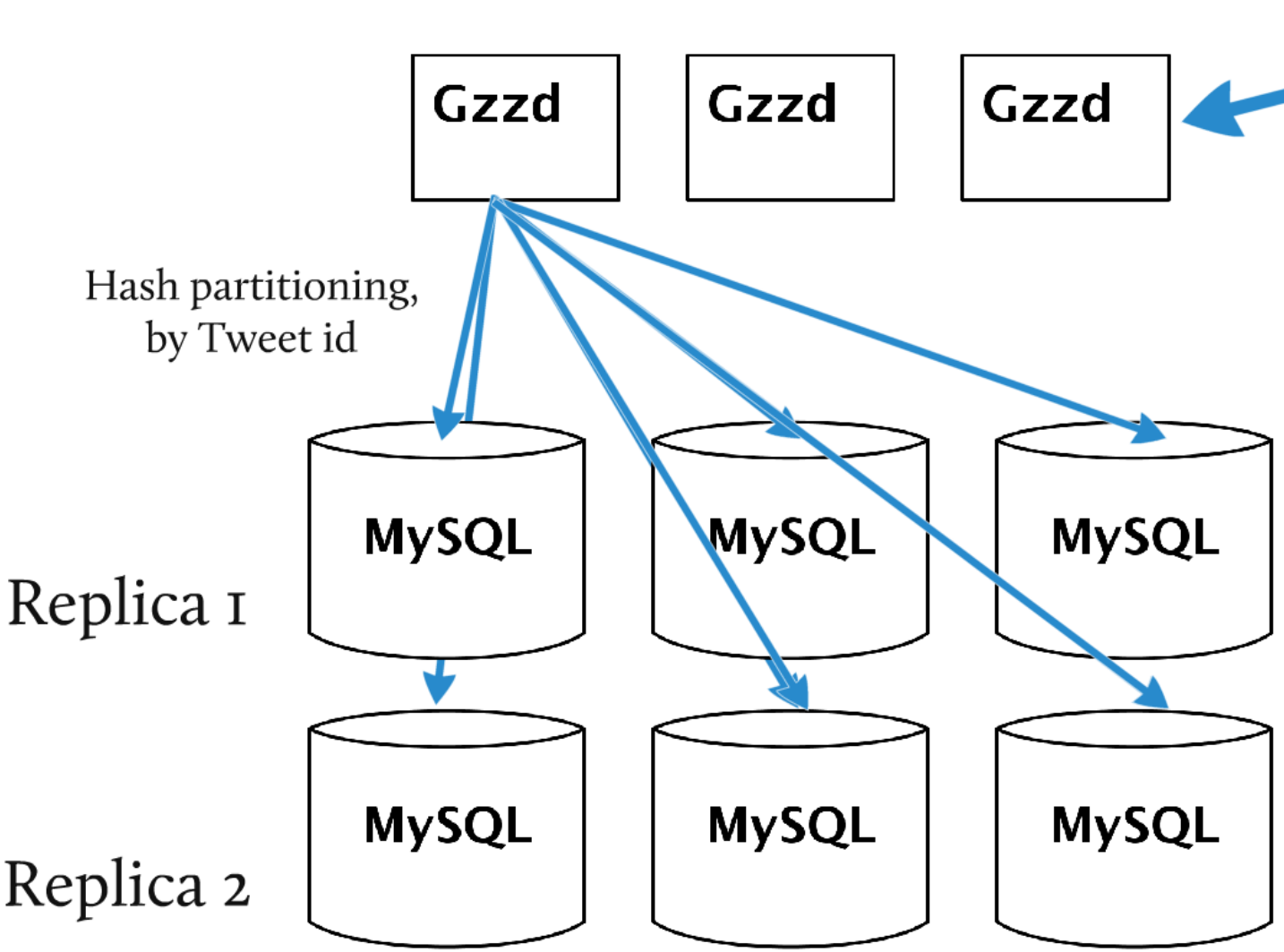
Repository pattern

Let the coc
make inform
decisions

ing at Twitter

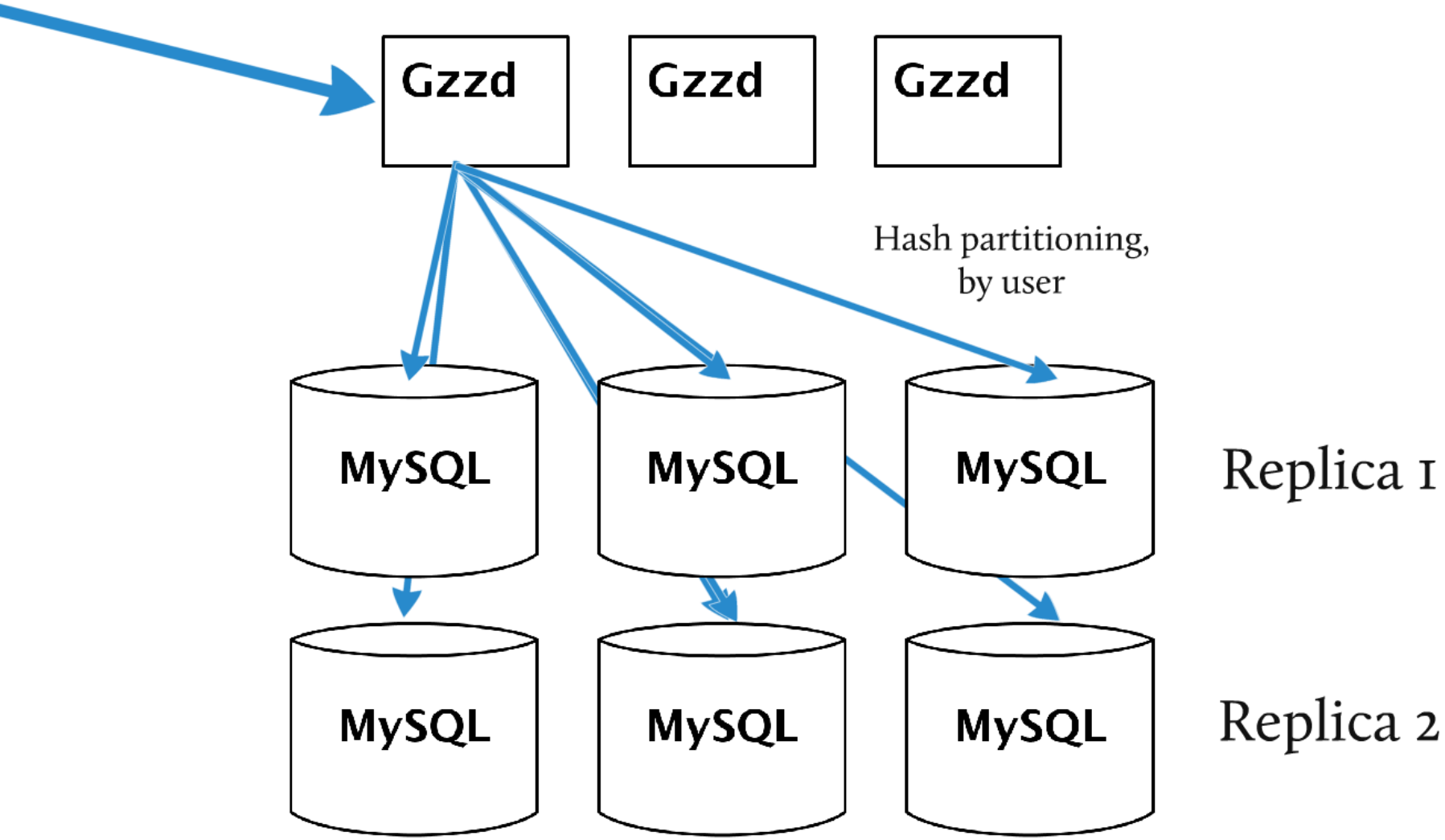


Tweet Storage ⟶



Row storage

K writes p/s
K reads p/s

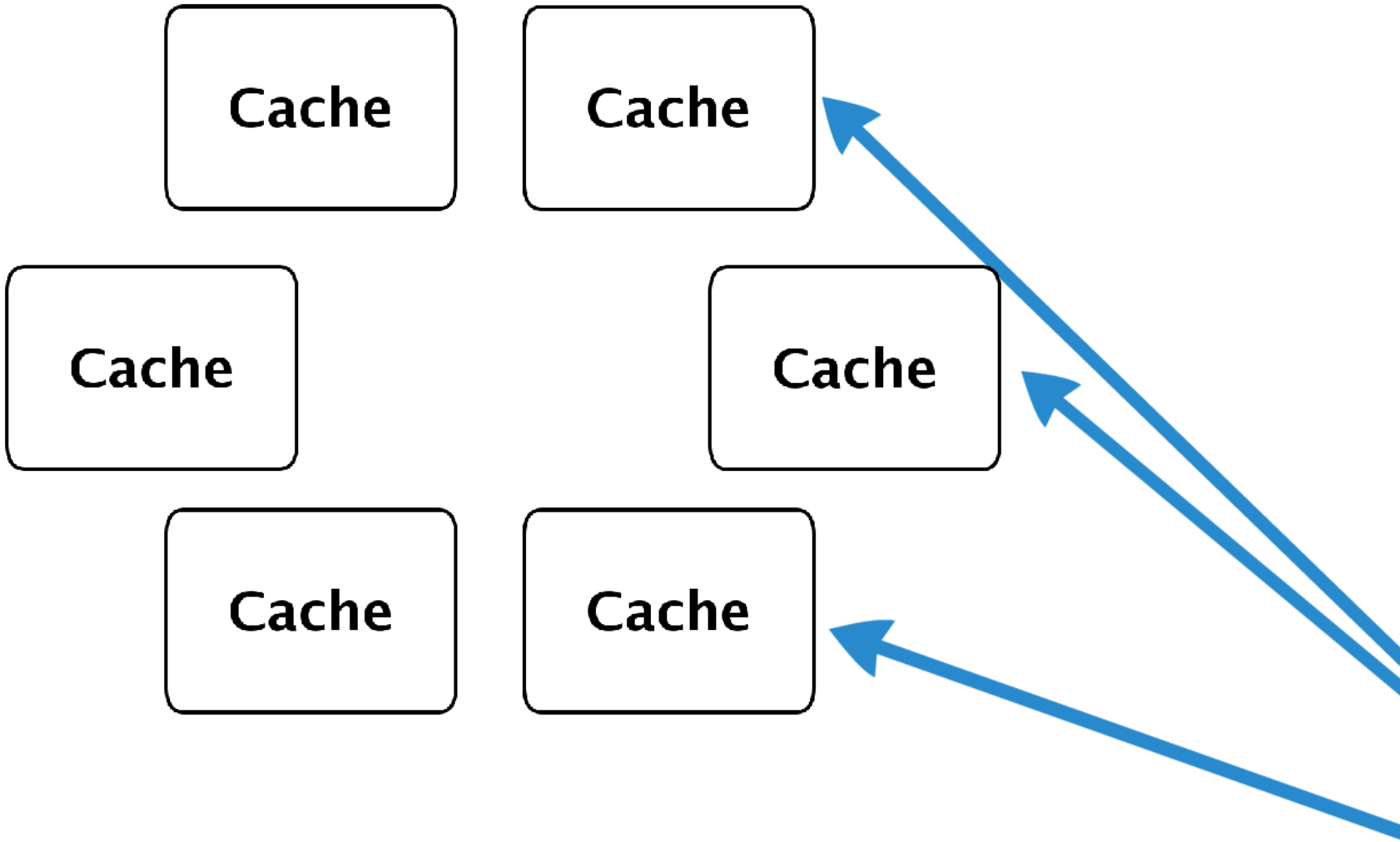


Index storage

K writes p/s
K reads p/s

K writes p/s
M reads p/s

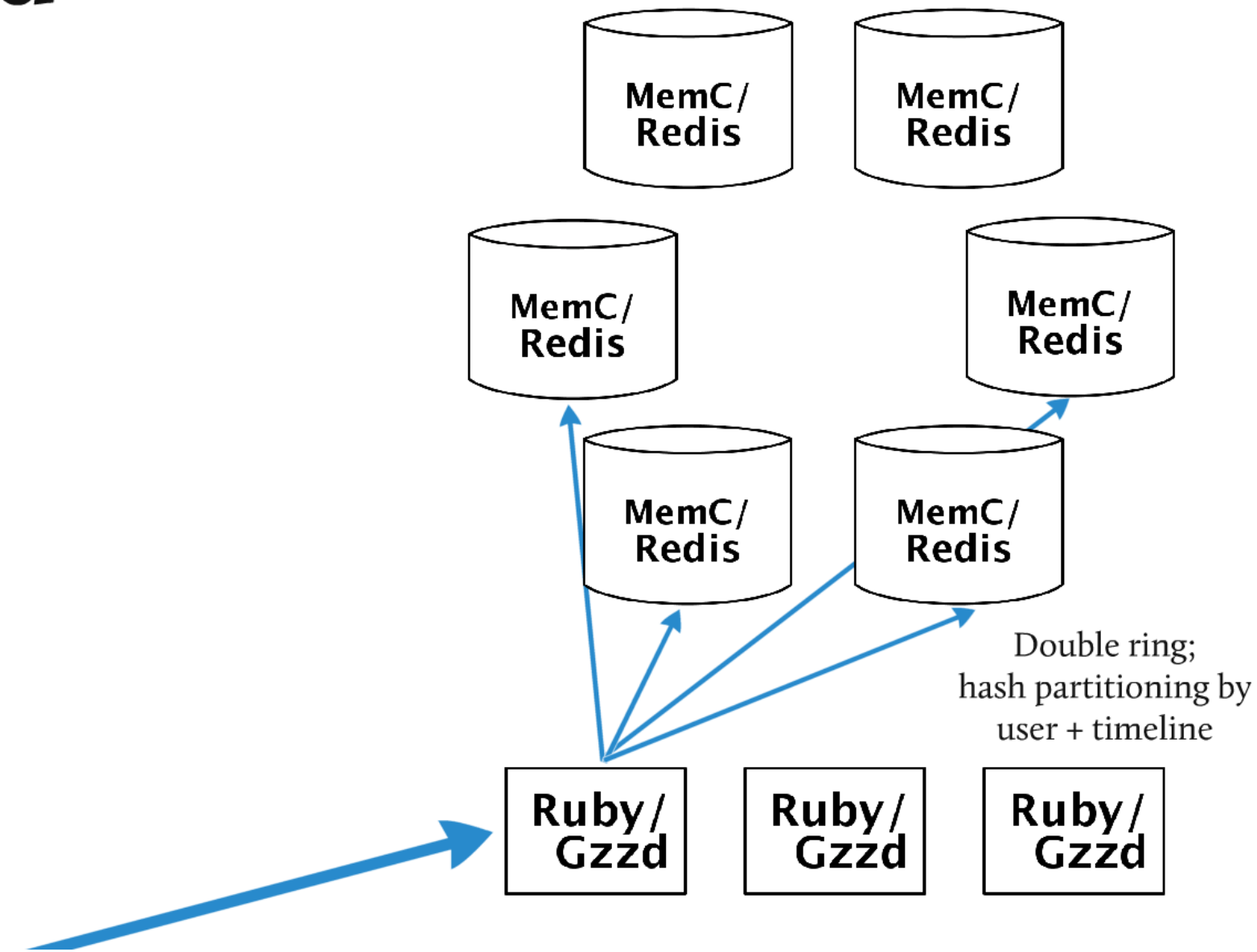
Row cache



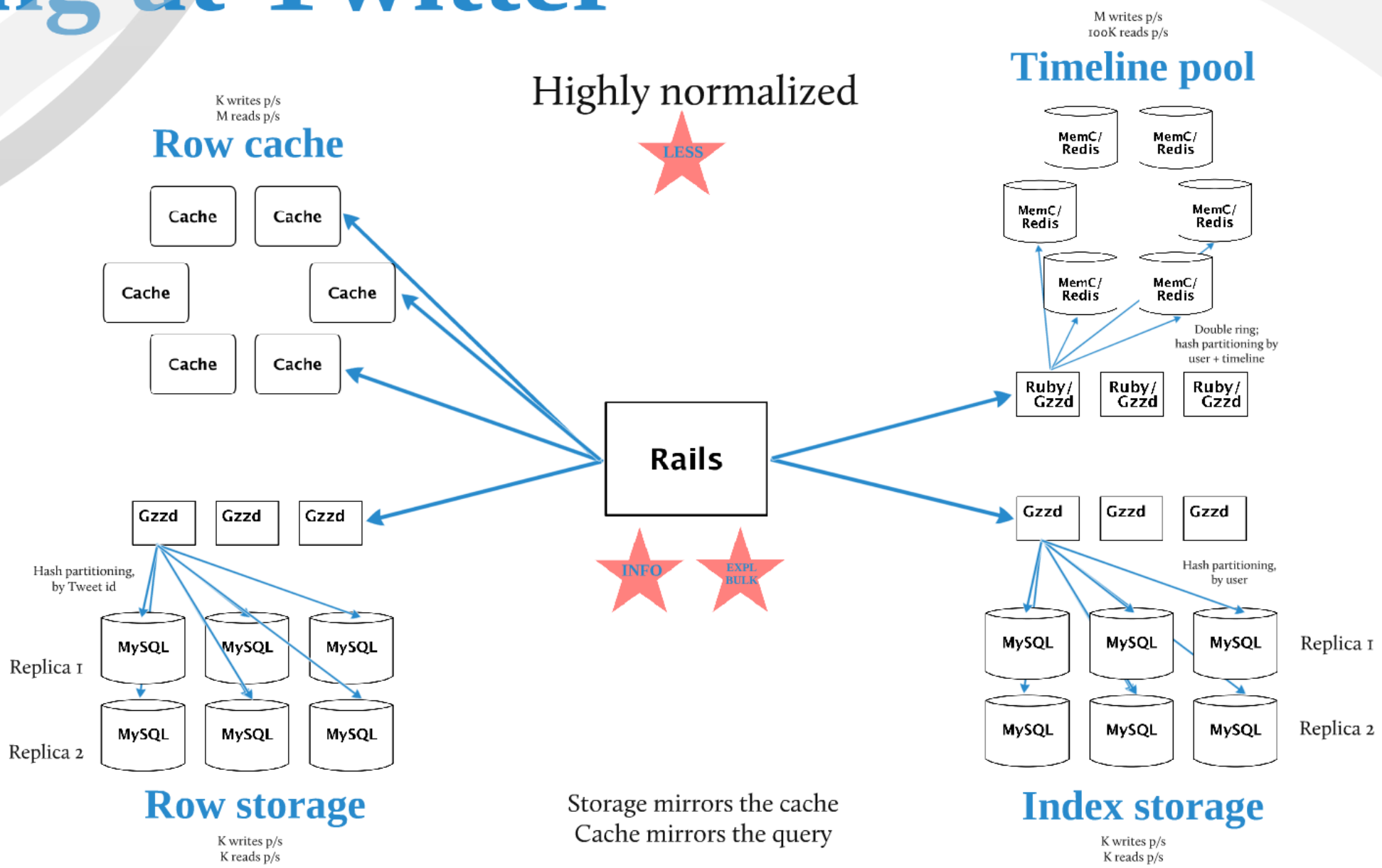
d

M writes p/s
100K reads p/s

Timeline pool



ing at Twitter



Tweet Storage ⟶

Repository pattern

```ruby
module UnitedRecord

  # The store is the heart of United Record. To write your own
  # store all you have to do is implement this simple interface.
  # By implementing this interface you will be able to leverage
  # the composition stores and quickly be able to build
  # complicated storage topologies.

  class Store

    def multi_get(keys, options = {})

    def multi_get_by_index(index, values, options = {})

    def insert(key, data)

    def update(key, data)

    def delete(key)

  end

end
```

```ruby
module UnitedRecord
  class CachingStore < UnitedRecord::Store
    attr_reader :cache_store, :persistent_store, :index

    def initialize(cache_store, persistent_store, options = {})
      @cache_store = cache_store
      @persistent_store = persistent_store
    end

    def multi_get(keys, options = {})
      cached_values = options[:cache] ? @cache_store.multi_get(keys) : {}
      cached_keys = cached_values.keys
      uncached_keys = keys - cached_values.keys
      db_values = {}
      if options[:persistent] && uncached_keys.length > 0
        db_values = @persistent_store.multi_get(uncached_keys).each do |k, v|
          v ||= UnitedRecord::Tombstone
          @cache_store.insert(k, v) if options[:read_through]
        end

        if options[:read_through]
          db_keys = db_values.keys
          nonexistent_keys = keys - cached_keys - db_keys
          nonexistent_keys.each do |k|
            @cache_store.insert(k, UnitedRecord::Tombstone)
          end
        end
      end
      cached_values.merge(db_values).delete_if{|h,k| k.nil? || k == UnitedRecord::Tombstone}
    end

    ...
```

```
def f(a: Int): Future[Int] =
  if (a % 2 == 0)
    Future.value(a)
  else
    Future.exception(new OddNumberException)

val myFuture: Future[Int] = f(2)

// an alternative way to define the function `f`:

def f(a: Int): Future[Int] = Future {
  if (a % 2 == 0) a
  else throw new OddNumberException
}

// 1) Wait 1 second the for computation to return
try {
  println(myFuture(1.second))
} catch {
  case e: TimeoutException   => ...
  case e: OddNumberException => ...
}

// 2) Invoke a callback when the computation succeeds or fails
myFuture onSuccess { i =>
  println(i)
} onFailure { e =>
  println("uh oh!")
} ensure {
  externalResources.release()
}
```

Events are the right model for server workloads

Concurrency

Re

```
1  modul
2
3    # T
4    # s
5    # b
6    # t
7    # c
8
9    cla
10
11     d
12
13     d
14
15     d
16
17     r
18
19     d
20
21   end
22
23 end
```

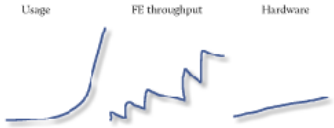
```
1  module Un
2    class C
3      attr_
4
5      def i
6        @co
7        @pe
8      end
9
10     def m
11       cac
12       cac
13       unc
14       sh_
15       if
16         d
17
18
19         e
20
21         i
22
23
24
25
26
27       e
28     end
29     cac
30   end
31
32   ...
```

```
def f(a: Int): Future[Int] =
  if (a % 2 == 0)
    Future.value(a)
  else
    Future.exception(new OddNumberException)

val myFuture: Future[Int] = f(2)

// an alternative way to define the function `f`:

def f(a: Int): Future[Int] = Future {
  if (a % 2 == 0) a
  else throw new OddNumberException
}

// 1) Wait 1 second the for computation to return
try {
  println(myFuture(1.second))
} catch {
  case e: TimeoutException   => ...
  case e: OddNumberException => ...
}

// 2) Invoke a callback when the computation succeeds or fails
myFuture onSuccess { i =>
  println(i)
} onFailure { e =>
  println("uh oh!")
} ensure {
  externalResources.release()
}
```

Performance Engineering at Twitter

Evan Weaver
@evan

Twitter's Growth



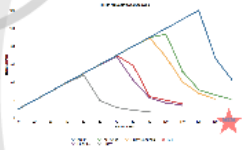
Just do less

LESS

Let the code make informed decisions

INFO

Ruby GC

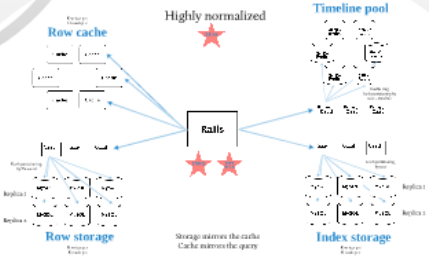


Ruby 1.8

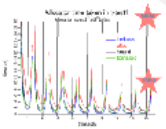
Kiji 0.11

Future Ruby

Tweet Storage



Malloc

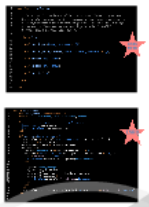


TCMalloc for unthreaded throughput, JEMalloc otherwise

Concurrency

Events are the right model for server workloads

Repository pattern



Focus on memory

MEM

Access data explicitly and in bulk

EXPL BULK

QCon

杭州站 · 2011年10月20日~22日

www.qconhangzhou.com（6月启动）

QCon北京站官方网站和资料下载

www.qconbeijing.com

全球企业开发大会

THE ANNUAL INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE