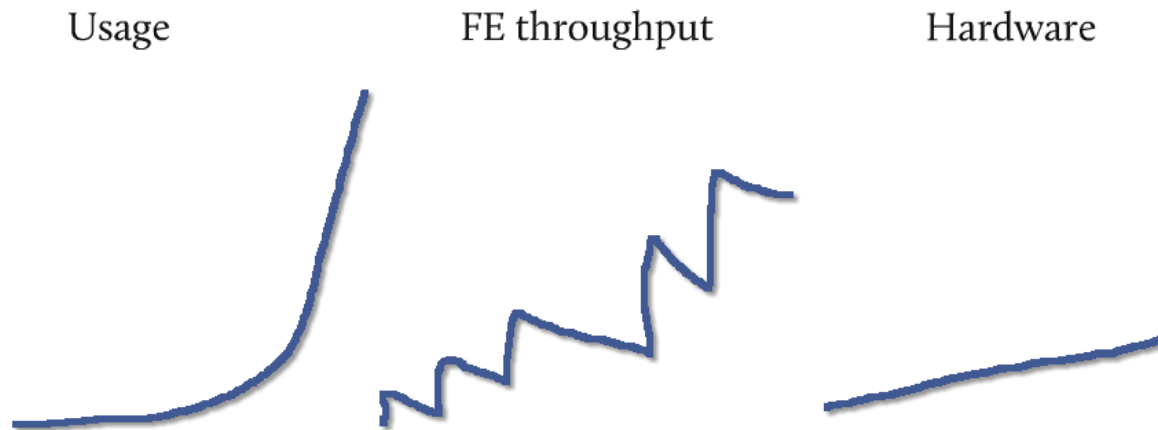


Twitter's Growth



Performance Engineering at Twitter

Evan Weaver
@evan

Future Ruby

Longlife mark array

★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F
★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F

Eden mark array

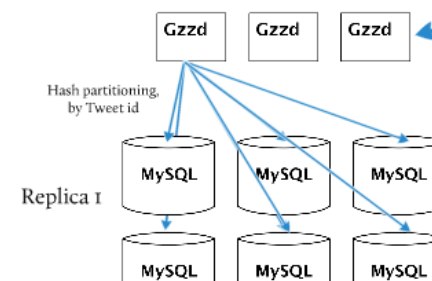
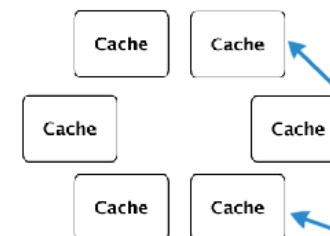
★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F
★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F	★Rv&F

RValues colocated with data on the malloc heap

RValue	RValue
*Heap	*Heap
Data	Data

Move objects? Just don't bother

Row cache



Highly no

LESS

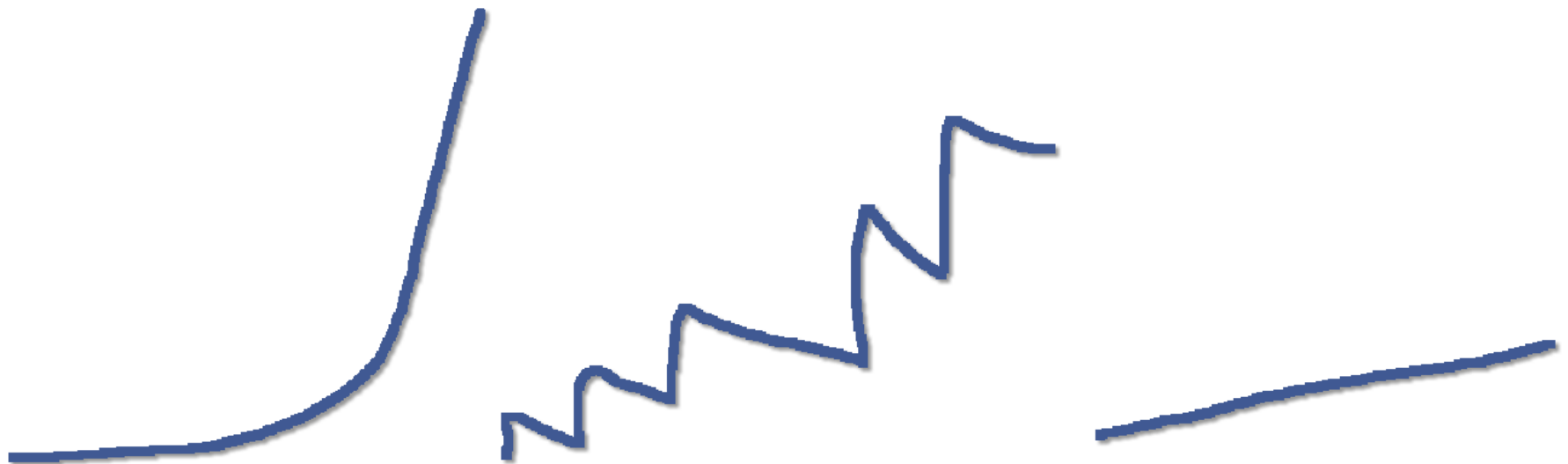
INFO

Twitter's Growth

Usage

FE throughput

Hardware



Performance Engineering at Twitter

Evan Weaver

Just do less



Ru

Singl

RValue		RValue
--------	--	--------

RValue	RValue
--------	--------

RValue	R
--------	---

Offsite dat

Pays th
unfreed

Collec

HTTP request/response ratio

120

100



```
T_MATCH      8      32640
T_VARMAP    3235    41255
T_SCOPE     1464    49917
T_NODE       65     11039
Source filename freed strings: 0
Source filename live strings: 1387
```

Focus on memory



```
27 end
28 end
29 cached values.merge(do values.delete {if a.k k.nil? || k == DeletedRecord::Timestamp}
30 {v} v)
31
32
```

Access data explicitly and in bulk



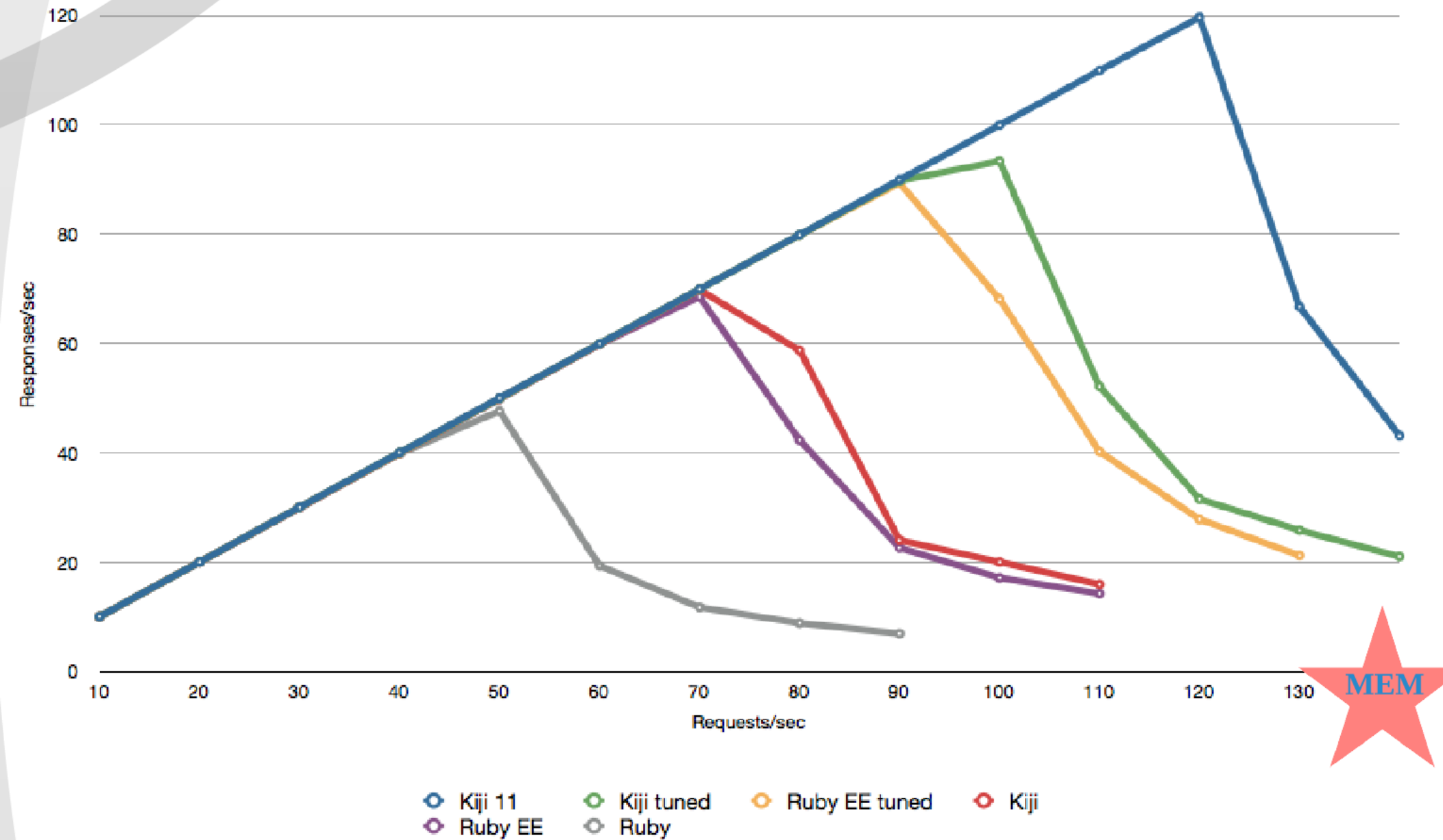
**Let the code
make informed
decisions**



Replica 1

Replica 2

HTTP request/response ratio

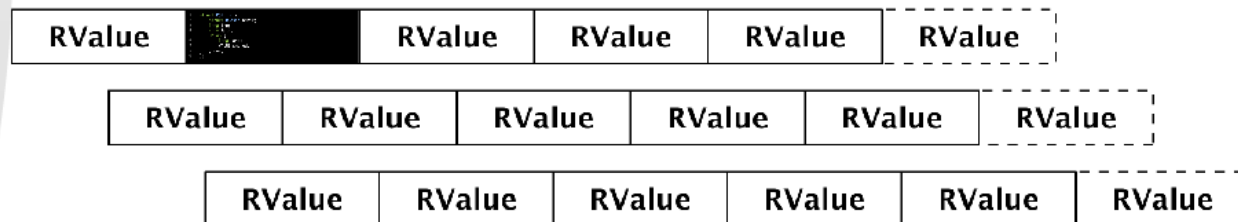


MEM



Ruby 1.8

Single Ruby heap space



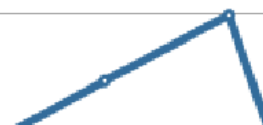
Malloc heap



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

Collects frequently, just in case

HTTP request/response ratio



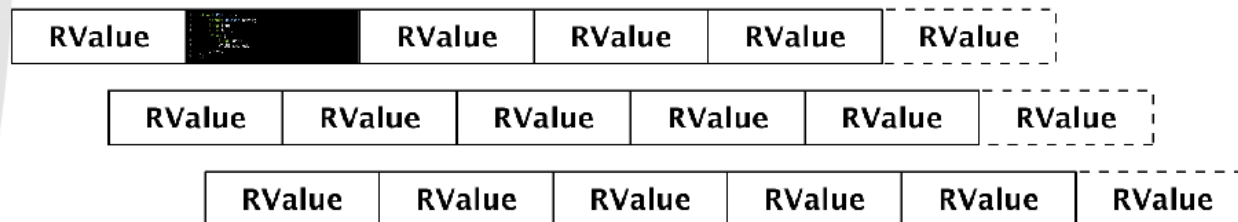
MEM

```
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



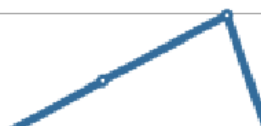
Malloc heap



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

Collects frequently, just in case

HTTP request/response ratio



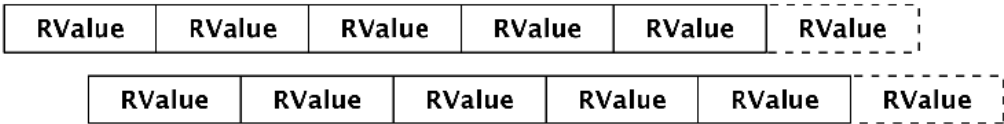
MEM



Kiji 0.11



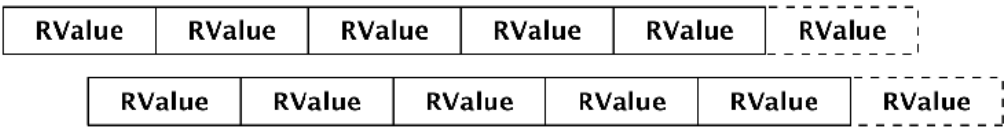
Longlife Ruby heap



Mark barrier set



Eden Ruby heap



Malloc heap



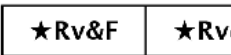
Longlife context:
AST compilation, constant
assignment, singleton assignment

Collects when fixed-size Ruby
heaps are full



Futu

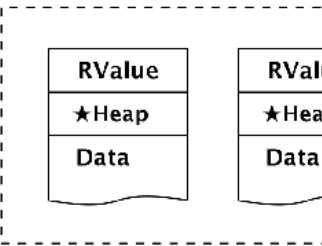
Long



Ed



RValues colocat



Move

```
Longlife collection (after 6 edens)
Objects moved to longlife:      12098
Remembered set kept:            12014
Remembered set recycled:         3
Longlife heaps in heap space:    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 collection:

Eden heaps in heap space: 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

MEM

LESS



Future Ruby



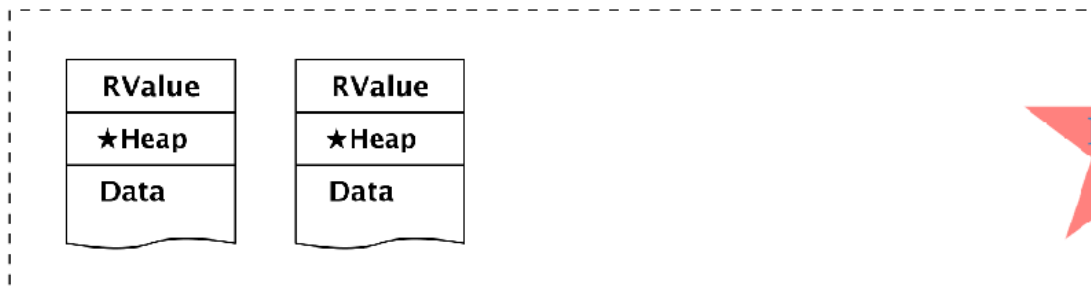
Longlife mark array



Eden mark array

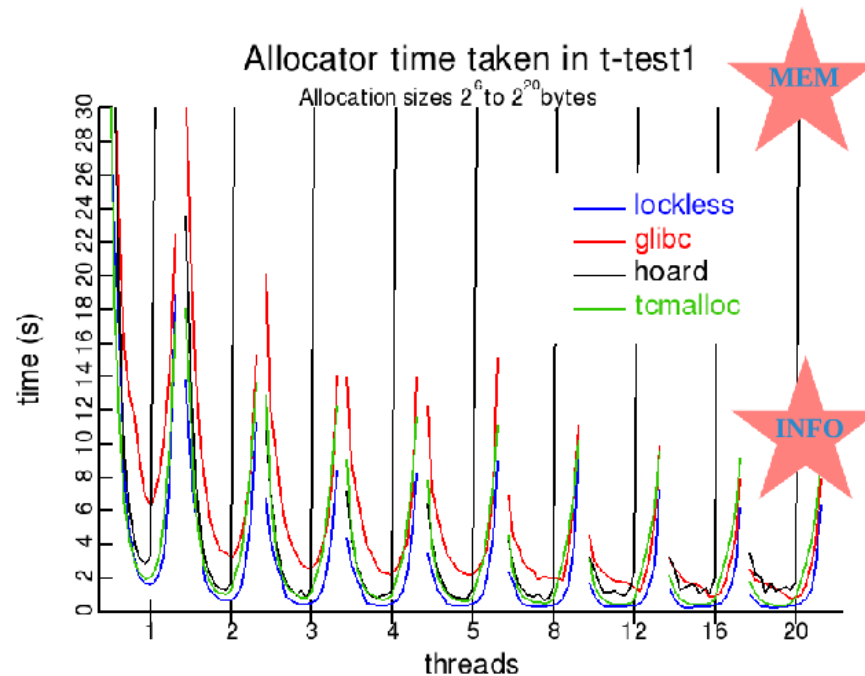


RValues colocated with data on the malloc heap



Move objects? Just don't bother

Collects based on RSS pressure



TCMalloc for unthreaded throughput,
JEMalloc otherwise

Malloc

Performance Engineering at Twitter

Evan Weaver
@evan

Introduce Ruby

Longlife mark array

Eden mark array

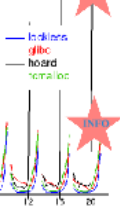
Allocated with data on the malloc heap

Move objects? Just don't bother

Collects based on RSS pressure

GC

In 1 test1



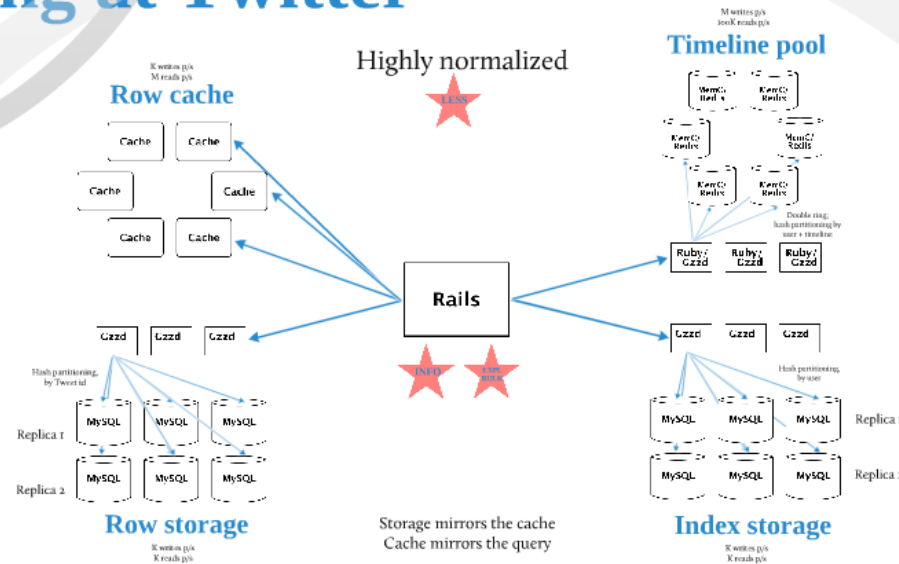
ded throughput,
otherwise

OC

Tweet Storage

Events are the right model for
server workloads

Concurrency



Let the code make informed decisions

Repository pattern

```
module Repository
  def create_tweet(tweet)
    # Create a new tweet in the database
    # and return the tweet object
  end

  def find_tweet(tweet_id)
    # Find a tweet by its ID
    # and return the tweet object
  end

  def update_tweet(tweet)
    # Update a tweet in the database
    # and return the tweet object
  end

  def delete_tweet(tweet_id)
    # Delete a tweet by its ID
    # and return the tweet object
  end
end
```

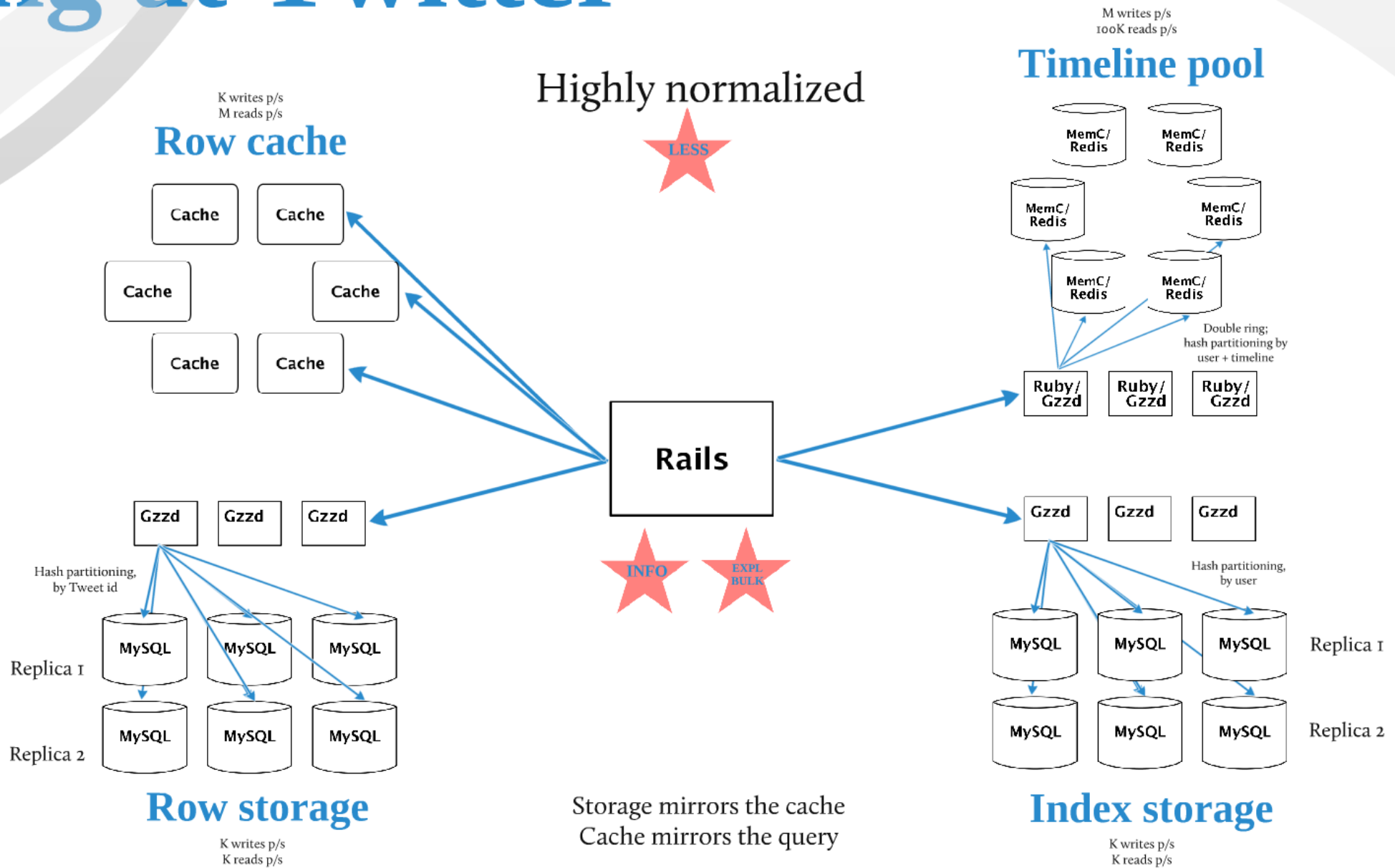
```
module Repository
  def create_tweet(tweet)
    # Create a new tweet in the database
    # and return the tweet object
  end

  def find_tweet(tweet_id)
    # Find a tweet by its ID
    # and return the tweet object
  end

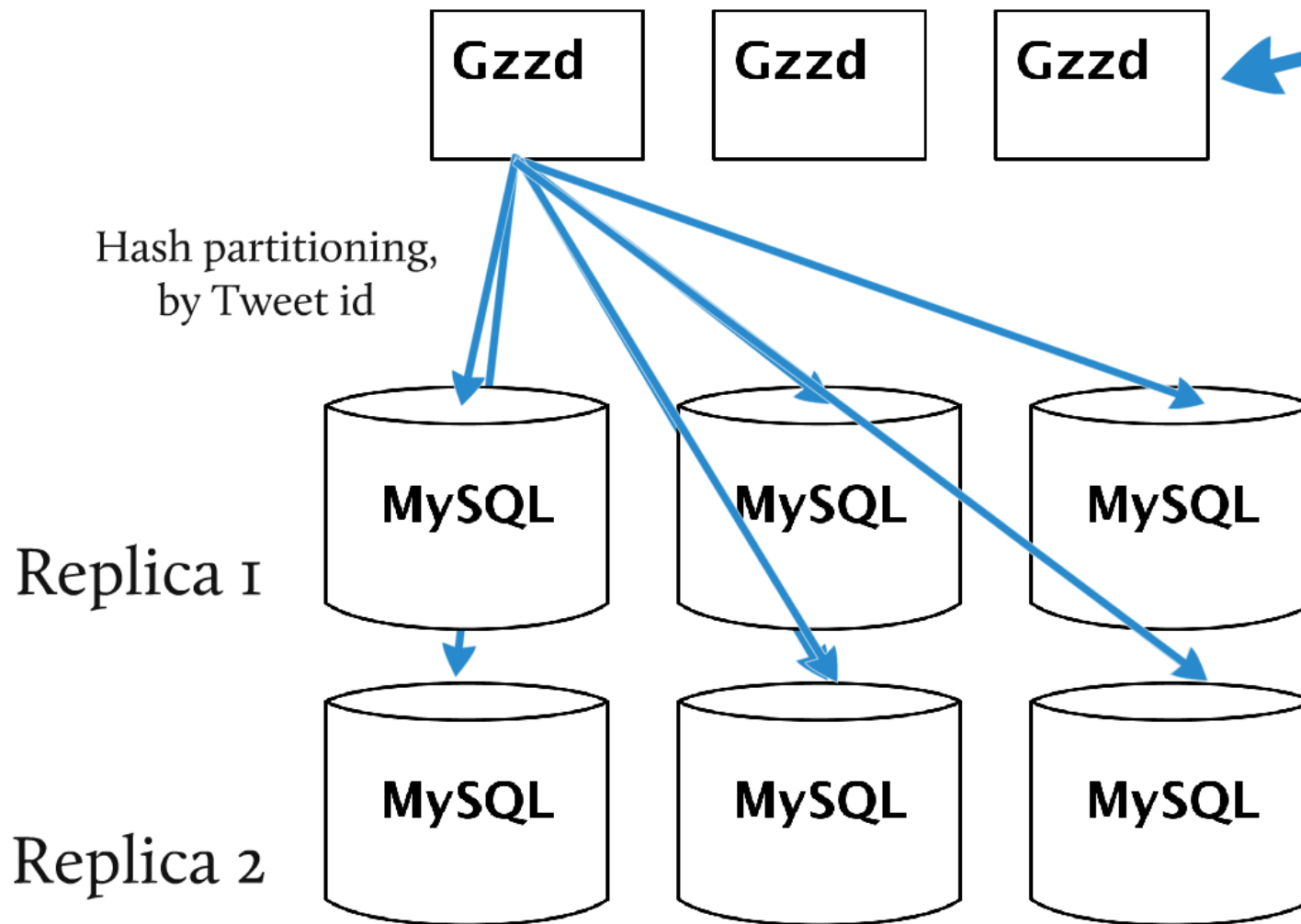
  def update_tweet(tweet)
    # Update a tweet in the database
    # and return the tweet object
  end

  def delete_tweet(tweet_id)
    # Delete a tweet by its ID
    # and return the tweet object
  end
end
```

ing at Twitter

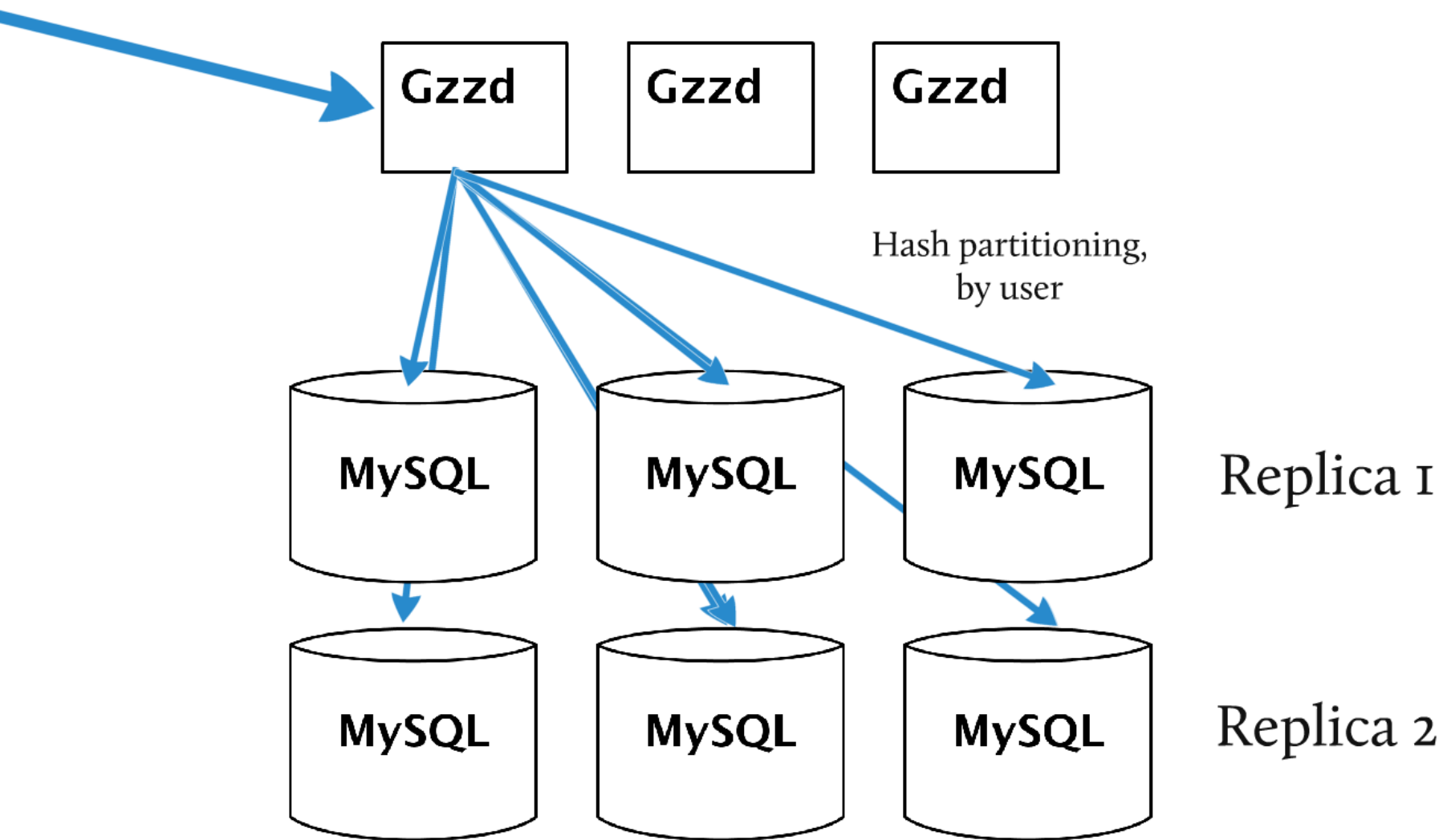


Tweet Storage →



Row storage

K writes p/s
K reads p/s

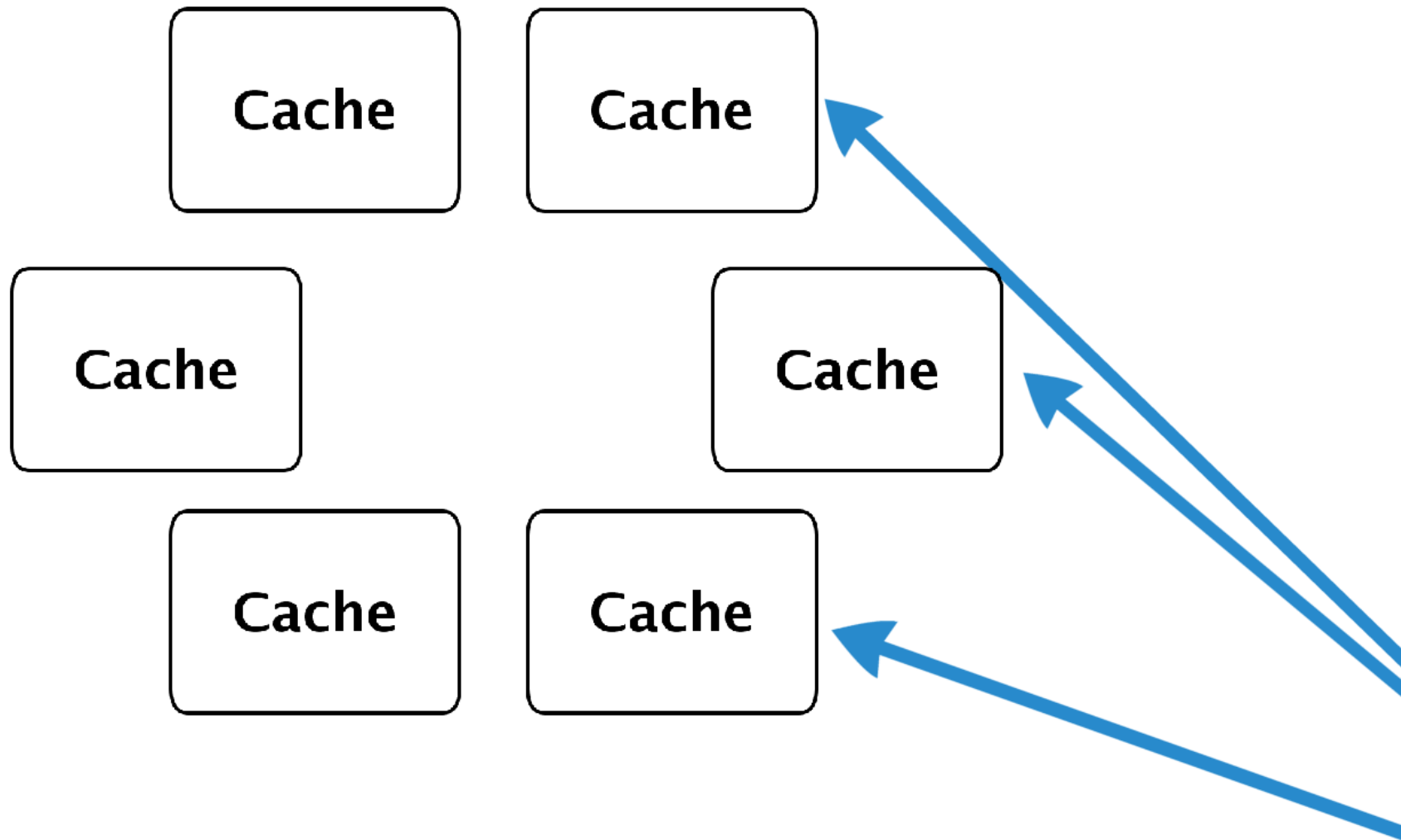


Index storage

K writes p/s
K reads p/s

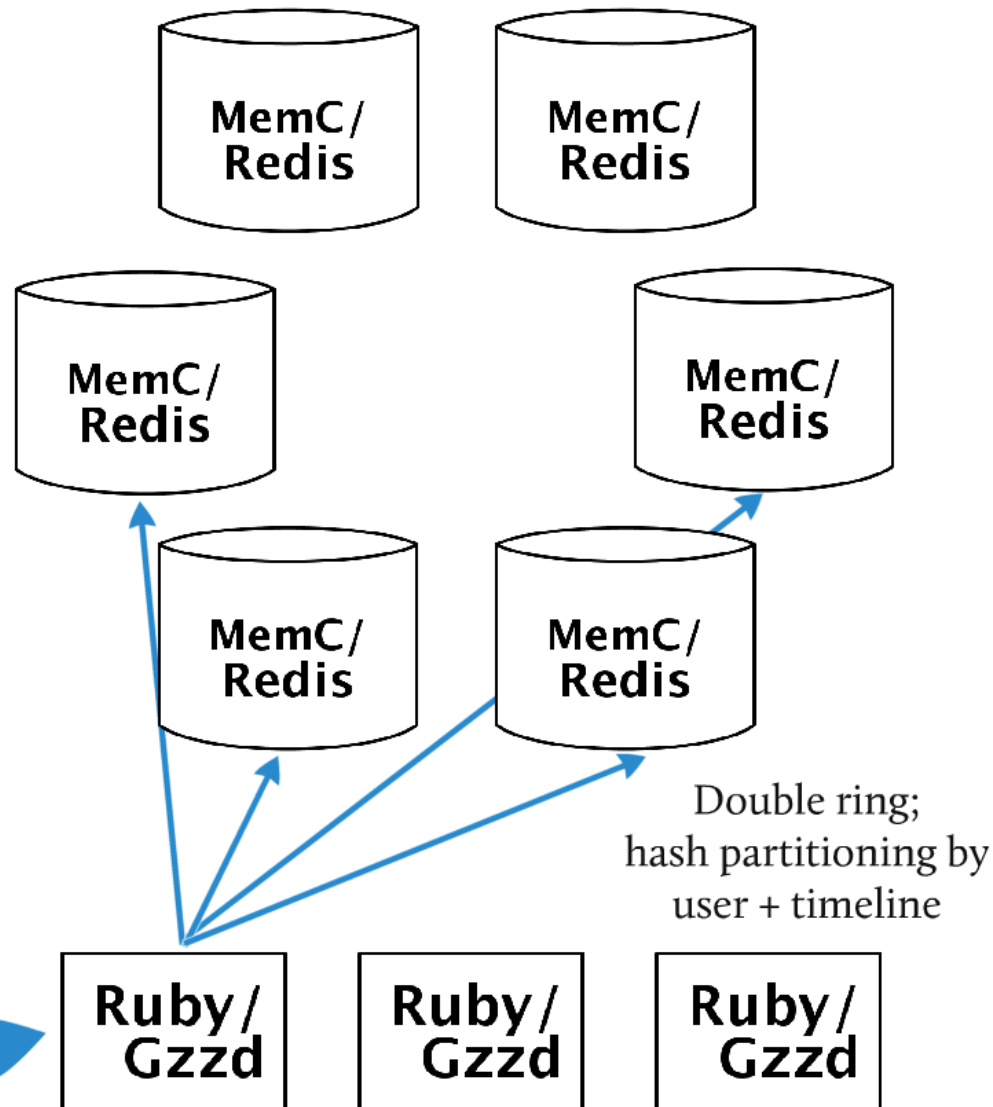
K writes p/s
M reads p/s

Row cache

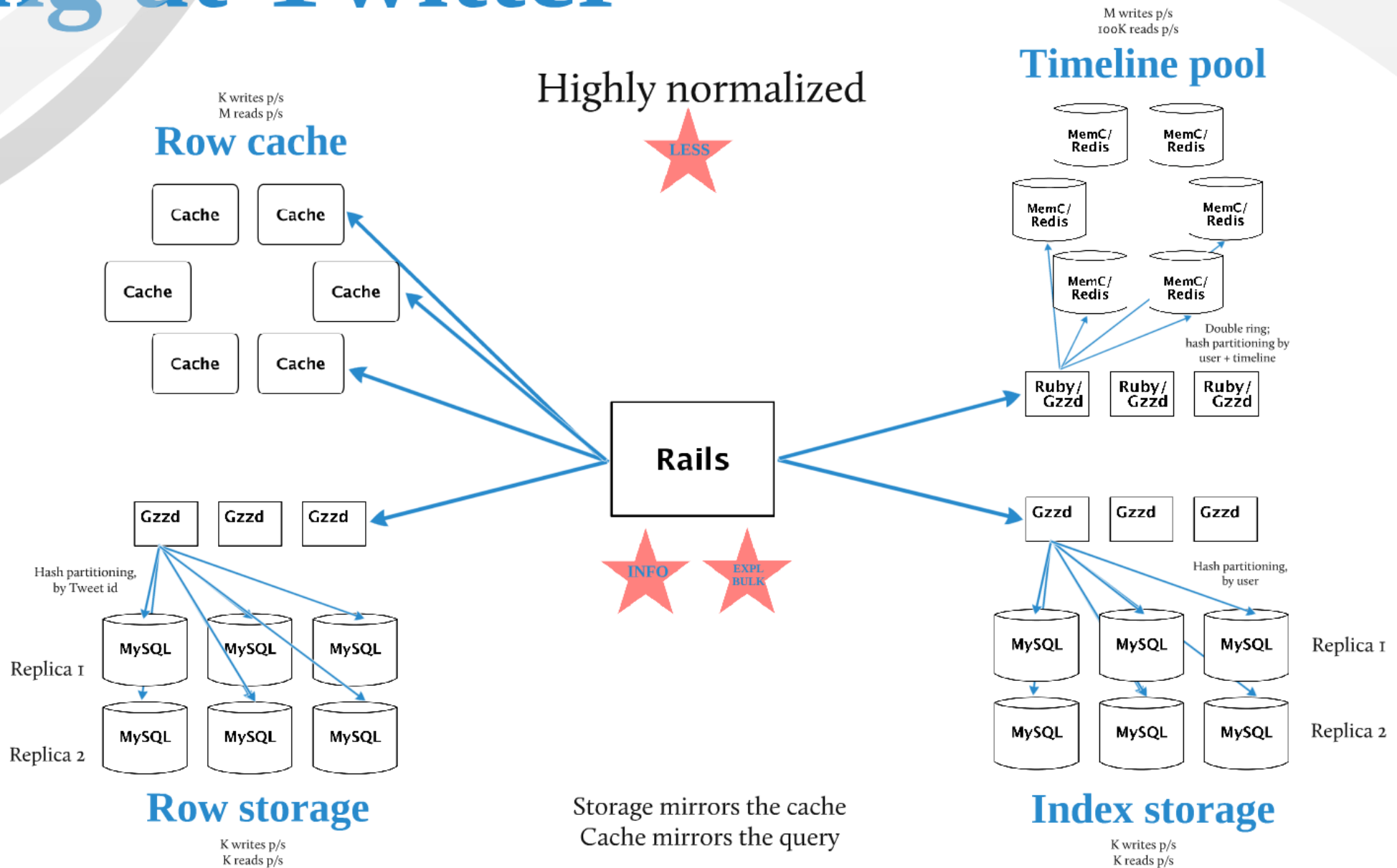


M writes p/s
100K reads p/s

Timeline pool



ing at Twitter



Tweet Storage →

Repository pattern

```
1  module UnitedRecord
2
3      # The store is the heart of United Record. To write your own
4      # store all you have to do is implement this simple interface.
5      # By implementing this interface you will be able to leverage
6      # the composition stores and quickly be able to build
7      # complicated storage topologies.
8
9      class Store
10
11          def multi_get(keys, options = {})
12
13          def multi_get_by_index(index, values, options = {})
14
15          def insert(key, data)
16
17          def update(key, data)
18
19          def delete(key)
20
21      end
22
23  end
```



EXPL
BULK


```

1  module UnitedRecord
2    class CachingStore < UnitedRecord::Store
3      attr_reader :cache_store, :persistent_store, :index
4
5      def initialize(cache_store, persistent_store, options = {})
6        @cache_store = cache_store
7        @persistent_store = persistent_store
8      end
9
10     def multi_get(keys, options = {})
11       cached_values = options[:cache] ? @cache_store.multi_get(keys) : {}
12       cached_keys = cached_values.keys
13       uncached_keys = keys - cached_keys
14       db_values = {}
15       if options[:persistent] && uncached_keys.length > 0
16         db_values = @persistent_store.multi_get(uncached_keys).each do |k, v|
17           v ||= UnitedRecord::Tombstone
18           @cache_store.insert(k, v) if options[:read_through]
19         end
20
21         if options[:read_through]
22           db_keys = db_values.keys
23           nonexistent_keys = keys - cached_keys - db_keys
24           nonexistent_keys.each do |k|
25             @cache_store.insert(k, UnitedRecord::Tombstone)
26           end
27         end
28       end
29       cached_values.merge(db_values).delete_if{|h,k| k.nil? || k == UnitedRecord::Tombstone}
30     end
31
32     ...

```



INFO

```

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("un oh!")
} ensure {
  externalResources.release()
}

```

INFO

EXPL
BULK

MEM

LESS

Events are the right model for
server workloads

Concurrency

Rep

```

1 module
2
3   #
4   #
5   #
6   #
7   #
8
9   class
10
11   a
12
13   a
14
15   a
16
17   a
18
19   a
20
21   end
22
23   end

```

```

1 module Un
2   class C
3   attr_
4
5   def u
6   @ca
7   @pe
8   end
9
10  def
11  coe
12  coe
13  unc
14  dh_
15  if
16  d
17
18
19
20
21
22
23
24
25
26
27
28
29
30
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()  
}
```



INFO



EXPL
BULK



MEM



LESS

Evan Weaver
@evan



Single tasky loop space

$W_0 = \text{[redacted]}$ \rightarrow W_1 \rightarrow W_2 \rightarrow W_3 \rightarrow W_4

$\xrightarrow{\text{red}} W_0 \xrightarrow{\text{red}} W_1 \xrightarrow{\text{red}} W_2 \xrightarrow{\text{red}} W_3 \xrightarrow{\text{red}} W_4$

Multi-task loop

$\mathcal{L}^2(\mathbb{R}^n, \mathcal{A}_n) \rightarrow \mathcal{L}^2(\mathbb{R}^n, \mathcal{A}_n)$ \rightarrow $\mathcal{L}^2(\mathbb{R}^n, \mathcal{A}_n)$

Why this loop and how to use it for
 (a) Fourier analysis (b) signal processing

It's done frequently, you know

[illegible]

Longitudinal mark array

$$\begin{array}{ccccccc} \dots & x_{1,t} & \dots & x_{2,t} & \dots & x_{n,t} & \dots \\ \hline \dots & x_{1,t+1} & \dots & x_{2,t+1} & \dots & x_{n,t+1} & \dots \end{array}$$

Index mark array

$$\begin{array}{ccccccc} \dots & x_{1,t} & \dots & x_{2,t} & \dots & x_{n,t} & \dots \\ \hline \dots & x_{1,t+1} & \dots & x_{2,t+1} & \dots & x_{n,t+1} & \dots \end{array}$$

RV shows collected with data on the index long

t	$x_{1,t}$	$x_{2,t}$	$x_{n,t}$
1	$x_{1,1}$	$x_{2,1}$	$x_{n,1}$
2	$x_{1,2}$	$x_{2,2}$	$x_{n,2}$
3	$x_{1,3}$	$x_{2,3}$	$x_{n,3}$
4	$x_{1,4}$	$x_{2,4}$	$x_{n,4}$
5	$x_{1,5}$	$x_{2,5}$	$x_{n,5}$
6	$x_{1,6}$	$x_{2,6}$	$x_{n,6}$
7	$x_{1,7}$	$x_{2,7}$	$x_{n,7}$
8	$x_{1,8}$	$x_{2,8}$	$x_{n,8}$
9	$x_{1,9}$	$x_{2,9}$	$x_{n,9}$
10	$x_{1,10}$	$x_{2,10}$	$x_{n,10}$

When clipped? Are dead losses

Collected on RV platform

-Ruby GC

Tweet Storage→



The diagram illustrates the memory layout of a 2D array. The top part shows a 2x2 grid of 4x4 blocks, with blue arrows indicating row-major traversal. The bottom part shows a 4x4 grid of 4x4 blocks, with blue arrows indicating column-major traversal. Labels include 'Row cache', 'Row storage', and 'Column-major'.

Diagram illustrating the Index storage structure. The diagram shows a hierarchical tree of nodes. The root node is labeled "Index". It has three children: "leaf", "leaf", and "leaf". The first "leaf" node has three children: "leaf", "leaf", and "leaf". The second "leaf" node has three children: "leaf", "leaf", and "leaf". The third "leaf" node has three children: "leaf", "leaf", and "leaf". The diagram also shows a "leaf" node with three children: "leaf", "leaf", and "leaf". The diagram is labeled "Index storage" at the bottom.



Malloc

Concurrency





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