



Falcon - built for speed

Ann Harrison
Kevin Lewis



If it's so fast, why isn't it done yet?

Talk overview

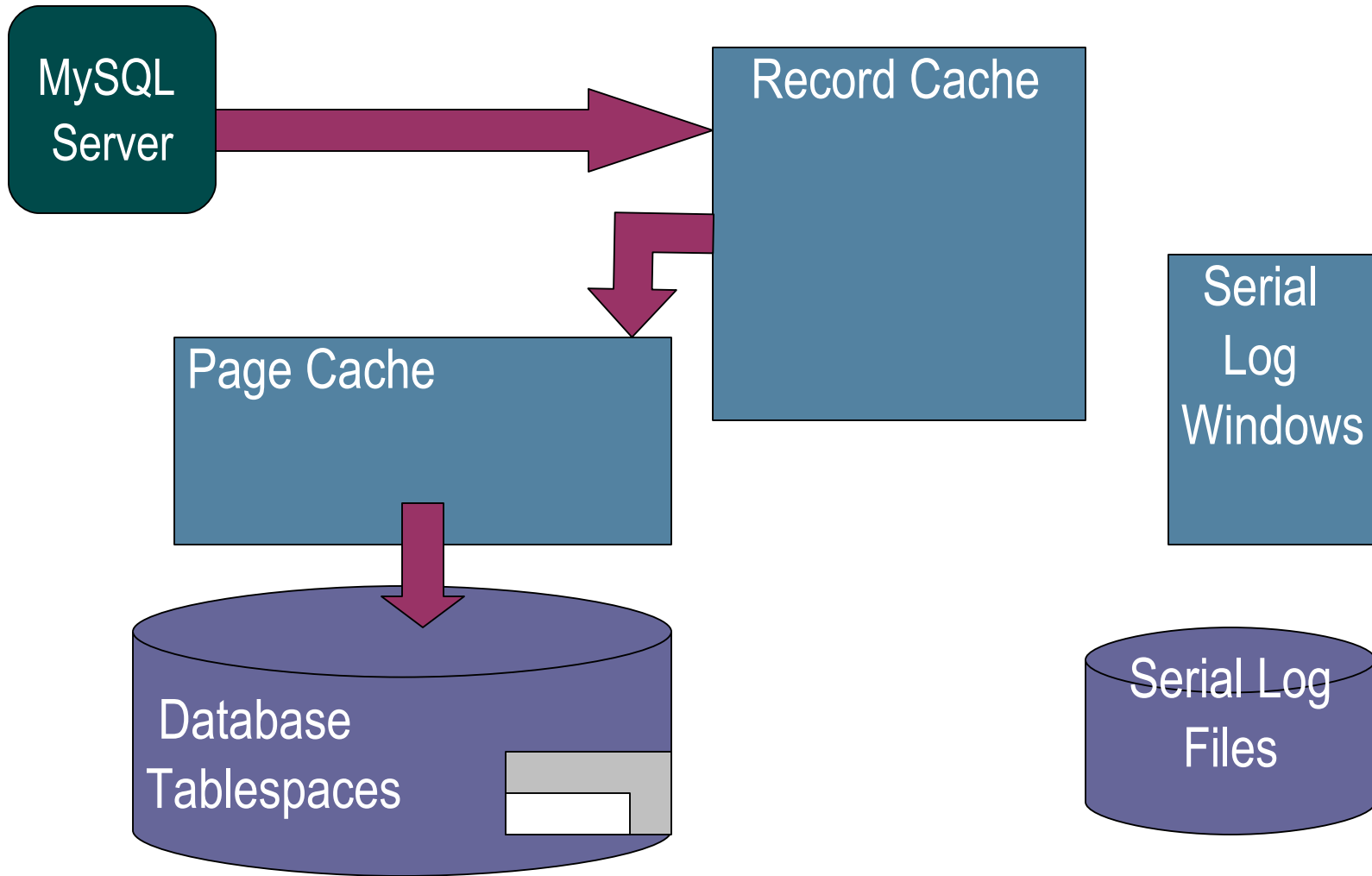
Falcon at a glance

Project history

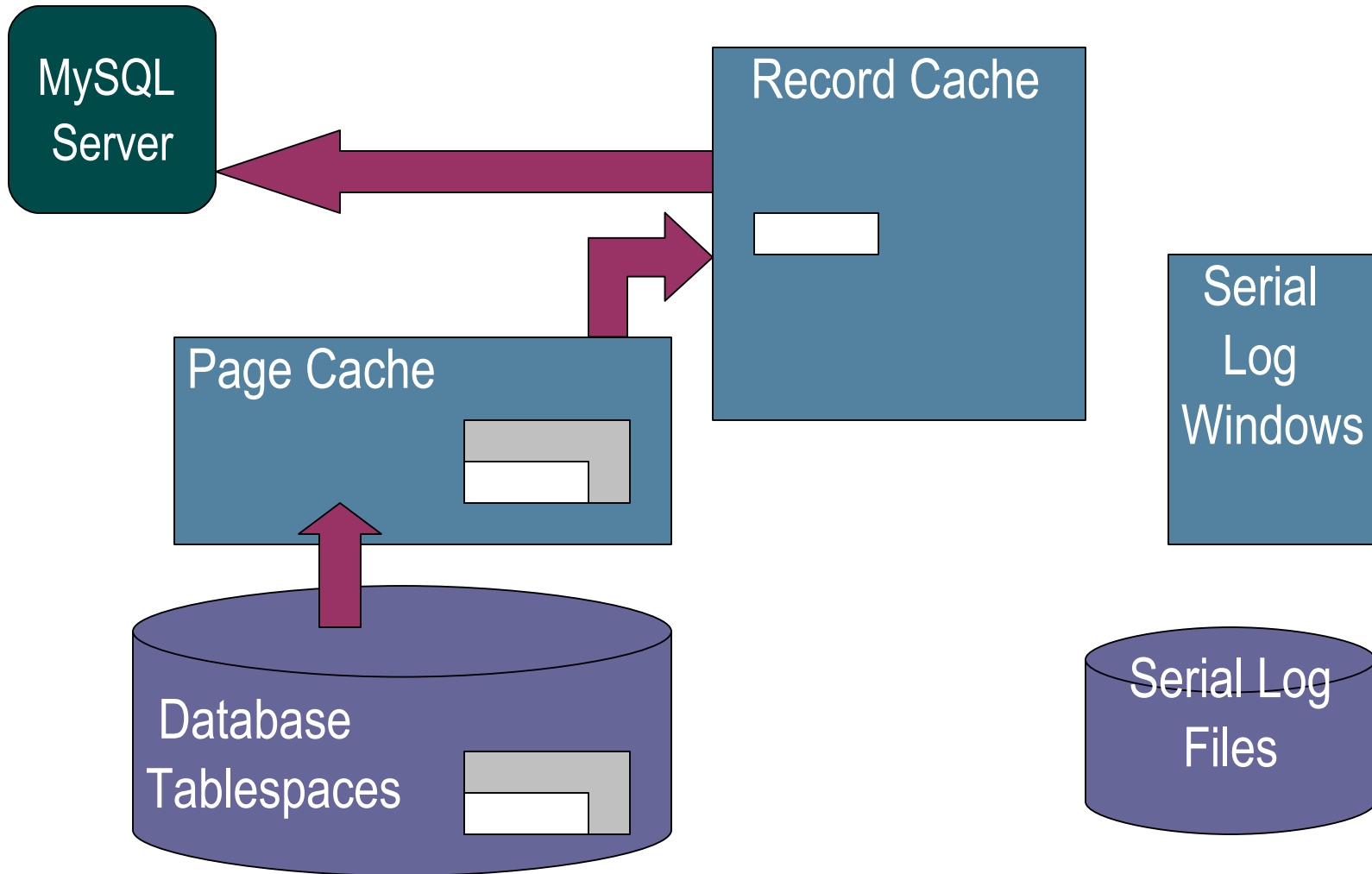
Multi-threading for the database developer

Cycle locking

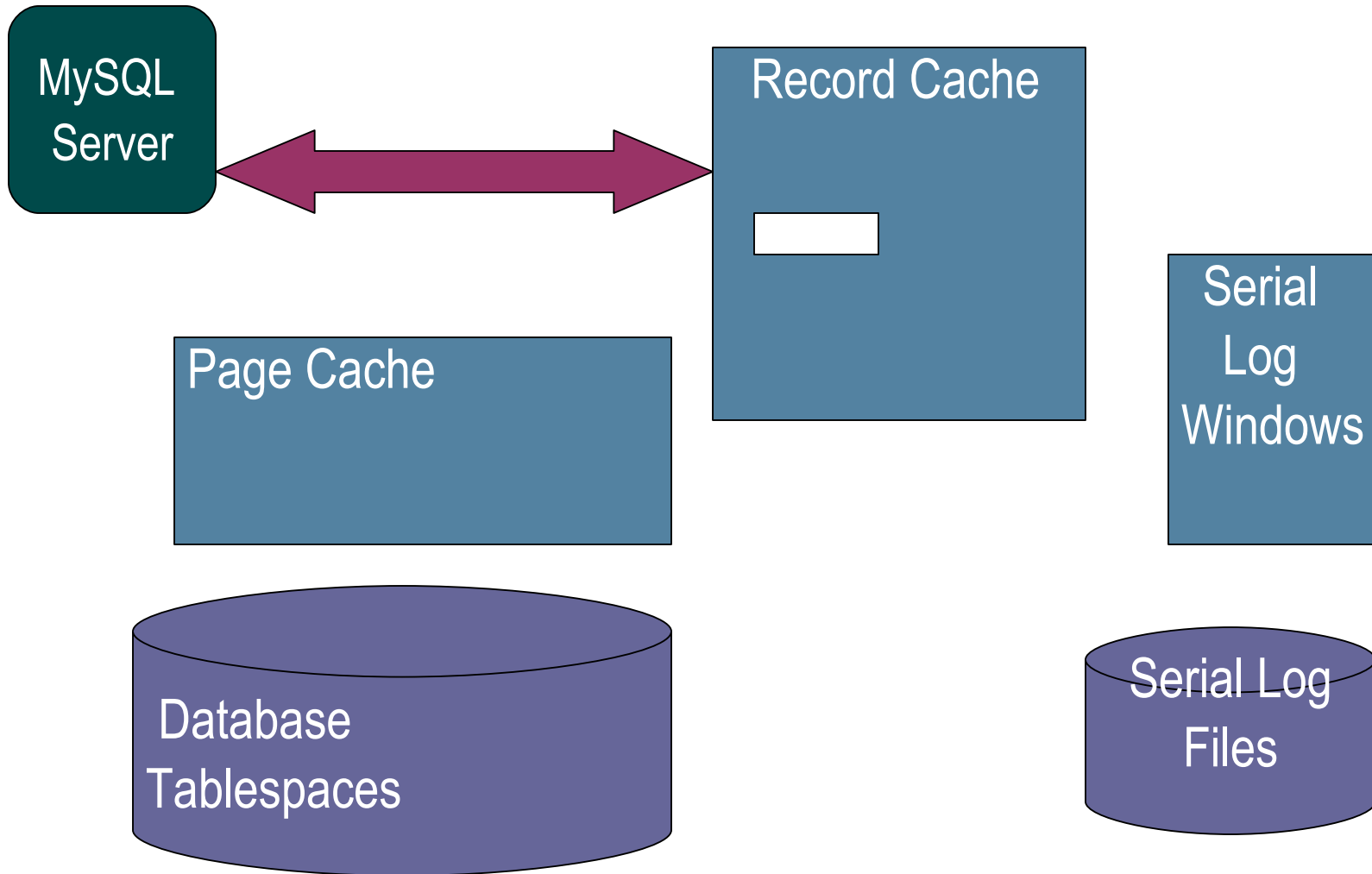
Falcon at a glance – read first record



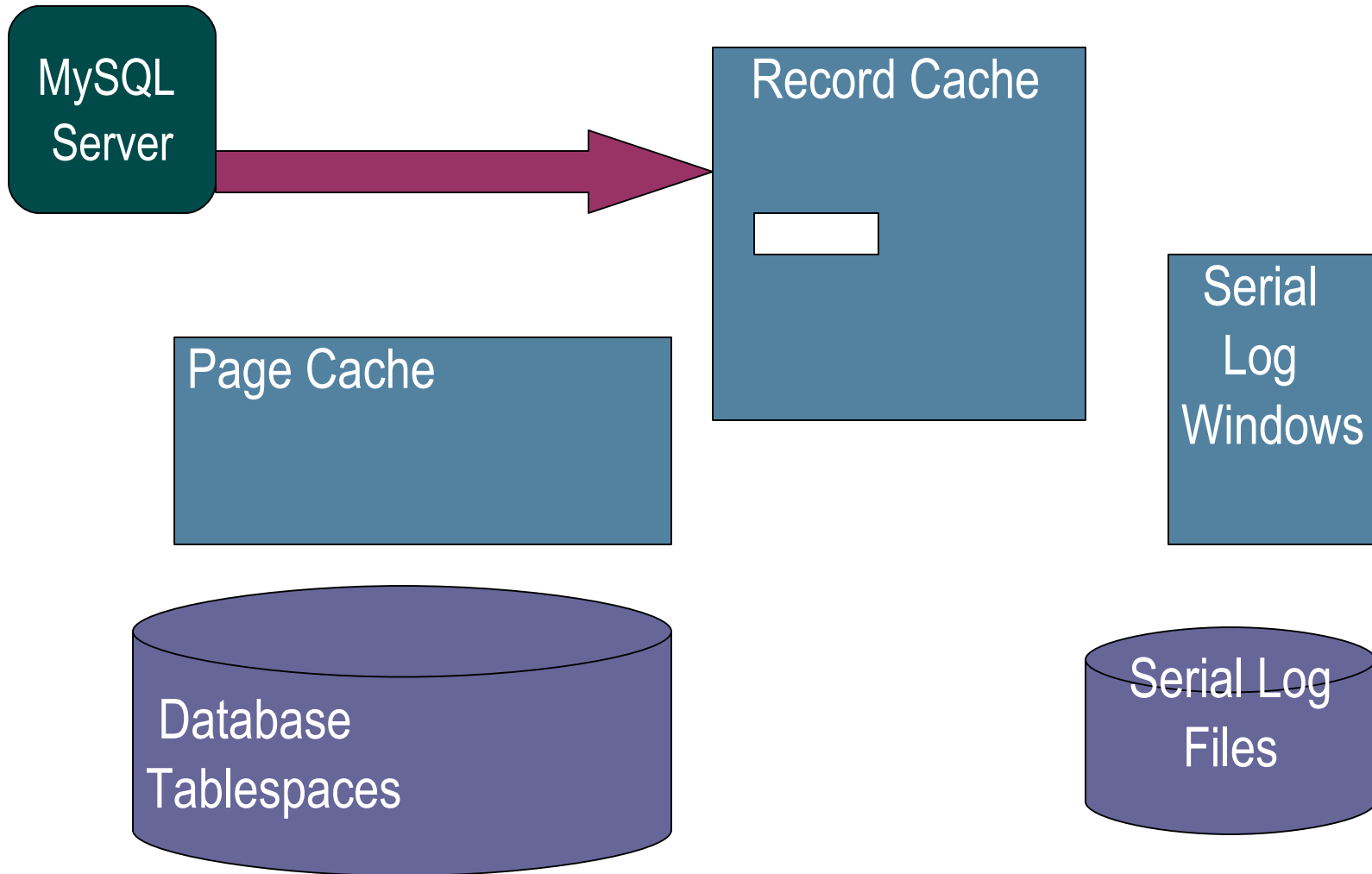
Falcon at a glance – read complete



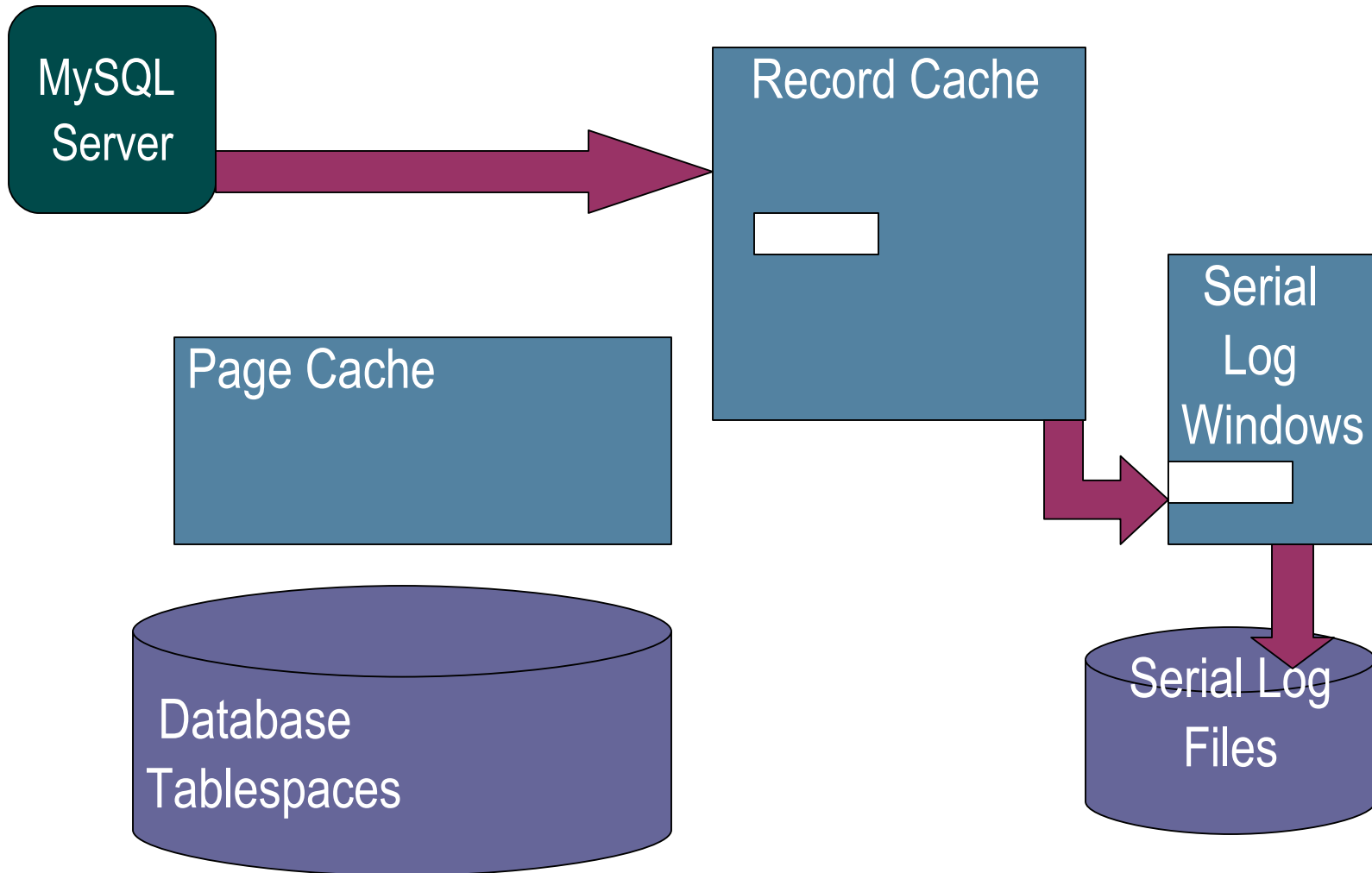
Falcon at a glance – read again



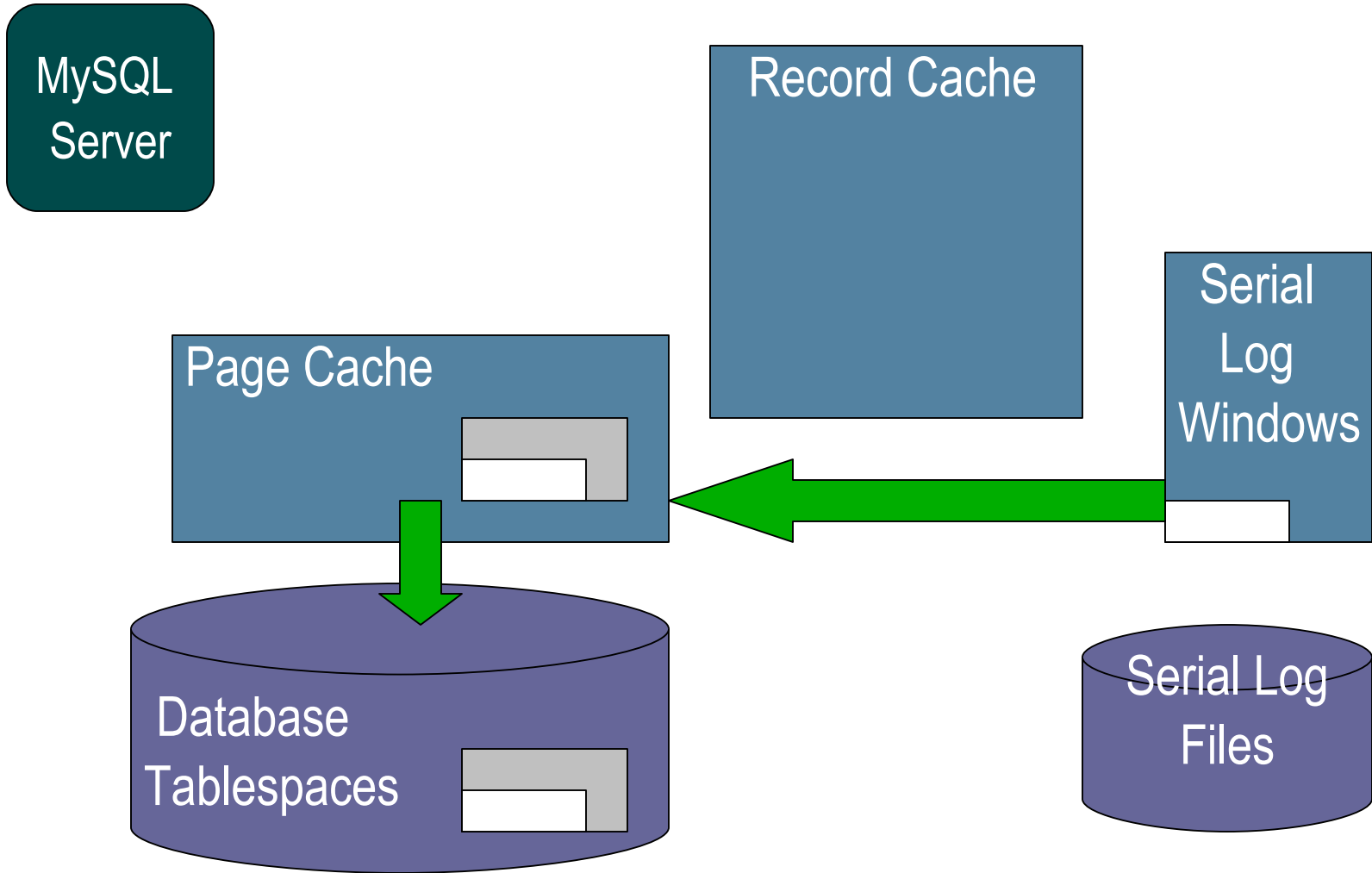
Falcon at a glance – write new record



Falcon at a glance – commit



Falcon at a glance – write complete



Falcon history

Origin

Transactional SQL Engine for Web App Environment

Bought by MySQL in 2006

MVCC

Consistent Read

Versions control write access

Memory only – no steal

Indexes and data separate

Data encoded on disk and in memory

Fine grained multi-threading

Falcon Goals circa 2006

Exploit large memory for more than just a bigger cache

Use threads and processors for data migration

Eliminate tradeoffs, minimize tuning

Scale gracefully to very heavy loads

Support web applications

Web application characteristics

Large archive of data

Smaller active set

High read:write ratio

Uneven, bursty activity

What we did instead

Enforce limit on record cache size

Respond to simple atypical loads

- Autocommit single record access

- Repeat “insert ... select”

- Single pass read of large data set

Challenge InnoDB on DBT2

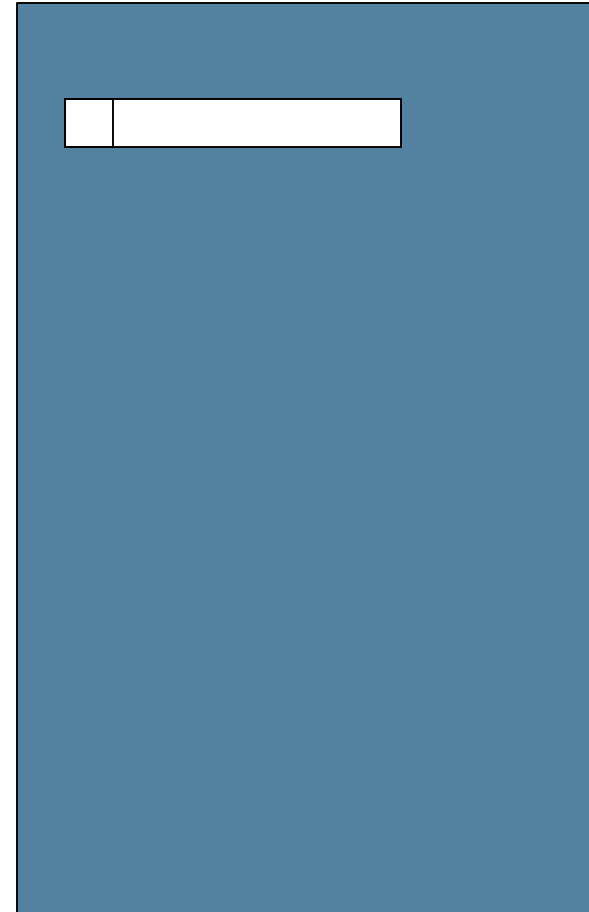
- Large working set

- Continuous heavy load

Hired the world's most vicious test designer

Record Cache

Record Cache contains:
Committed records with no versions

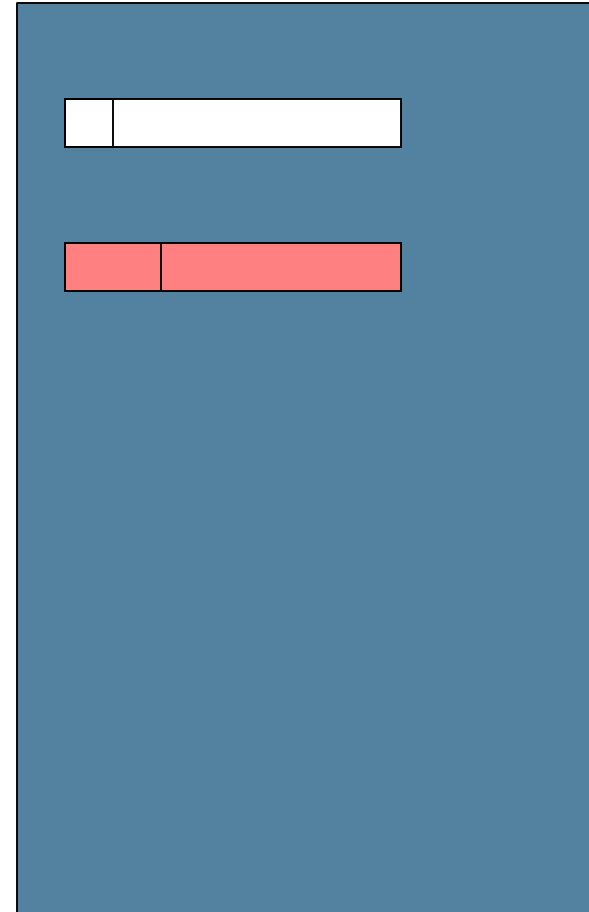


Record Cache

Record Cache contains:

Committed records with no versions

New, uncommitted records



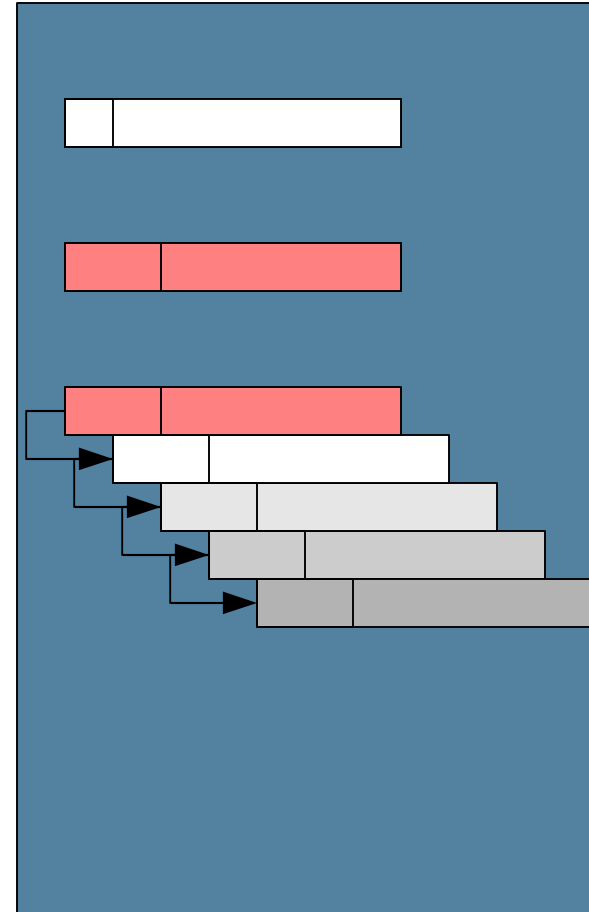
Record Cache

Record Cache contains:

Committed records with no versions

New, uncommitted records

Records with multiple versions



Record Cache cleanup – step 1

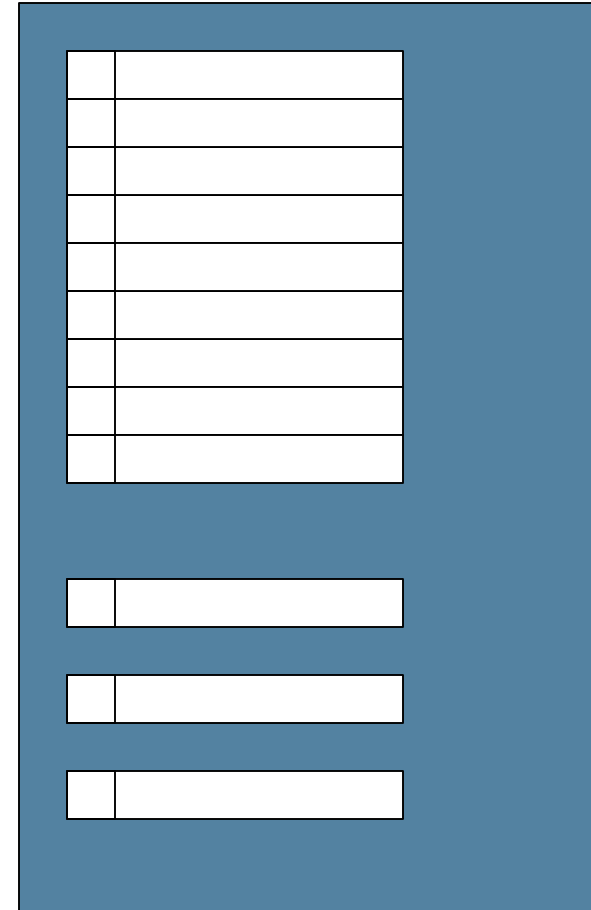
Cleanup old committed single version records

Scavenger

Runs on schedule or demand

Removes oldest mature records

Settable limits – start and stop

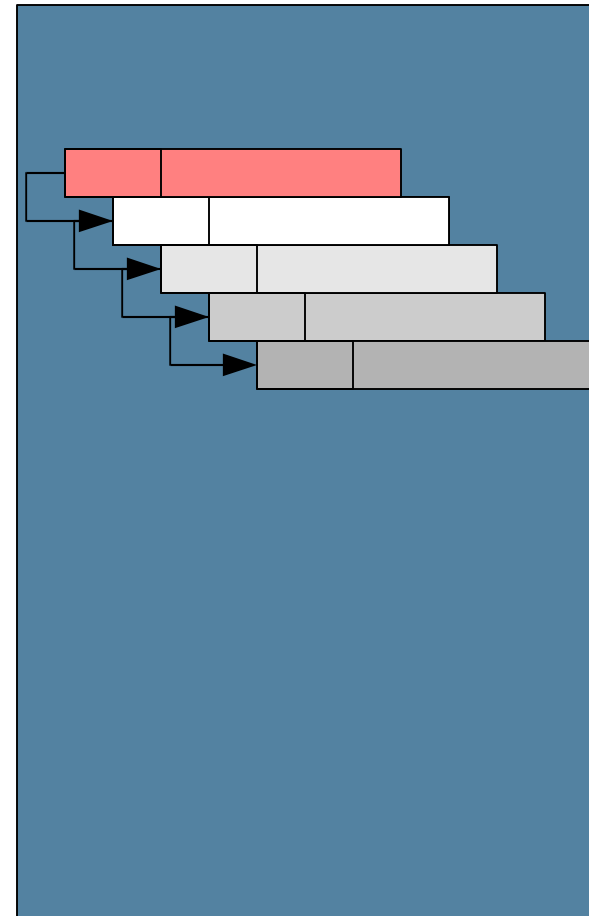


Record Cache Cleanup – step 2

Clean out record versions too old to be useful

Prune

Remove old, unneeded versions



Record Cache Cleanup – step 3

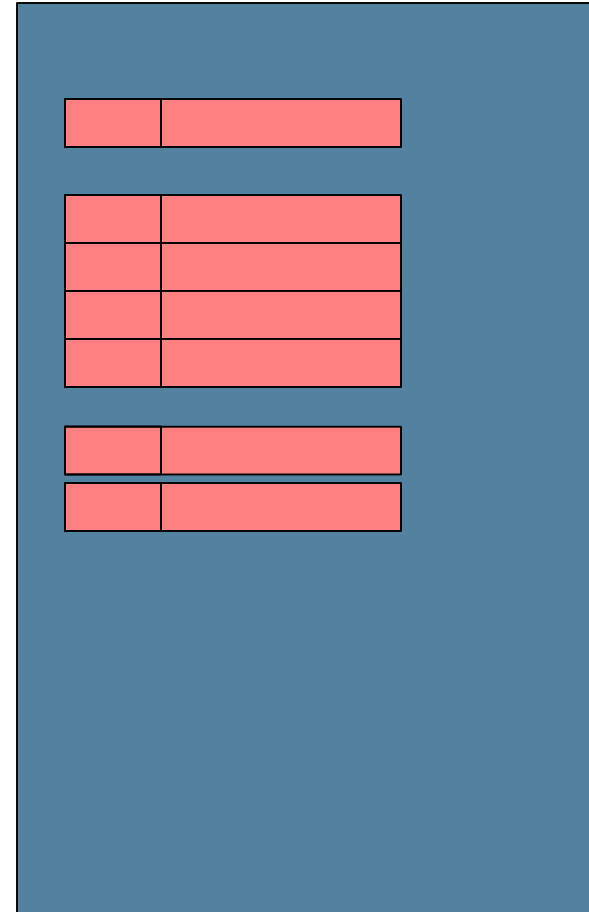
Clean up a cache full of new records

Chill

Copy new record data to log

Done by transaction thread

Settable start size



Record Cache Cleanup – step 4

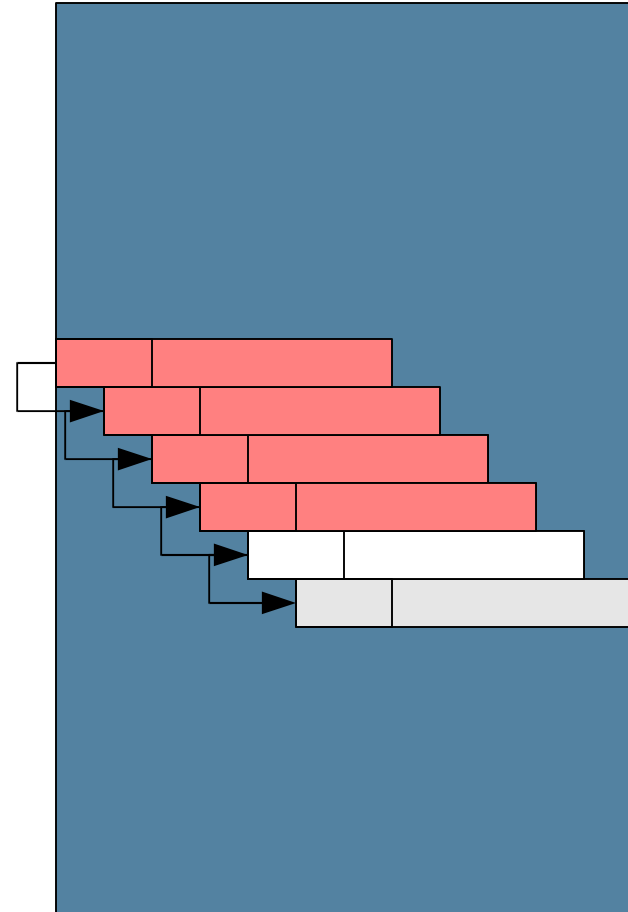
Clean up multiple versions of a single record created by a single transaction

Remove intermediate versions

Created by a single transaction

Rolled back to save point

Repeated updates



Record Cache Cleanup – step 5

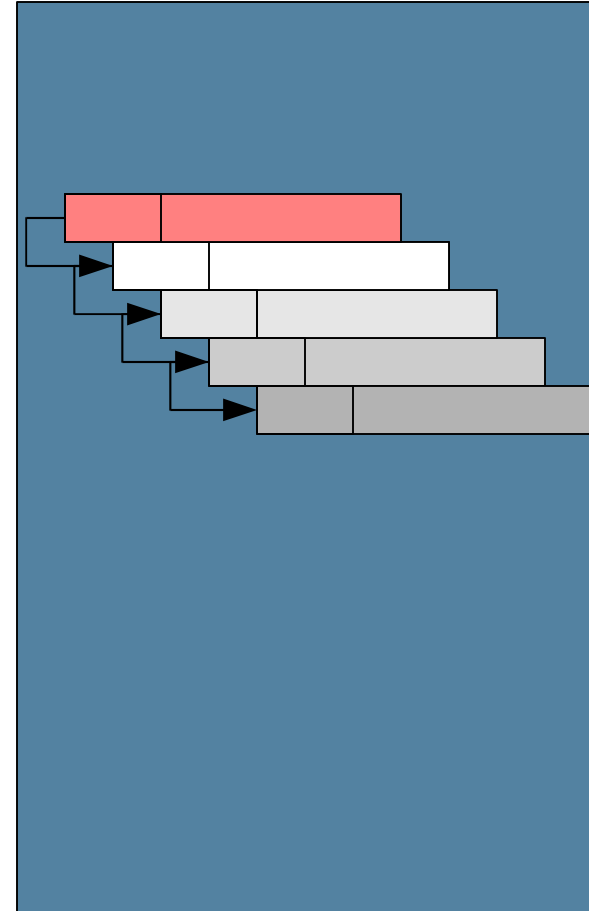
Clean up records with multiple versions, still potentially visible

Backlog

Copy entire record tree to disk

Expensive

Not yet working



Simple, atypical loads

Challenge:

- Autocommit single record access

 - Record cache is useless

 - Record encoding is useless

 - Transaction creation / destruction is too expensive

Response:

- Reuse read only transactions

Result:

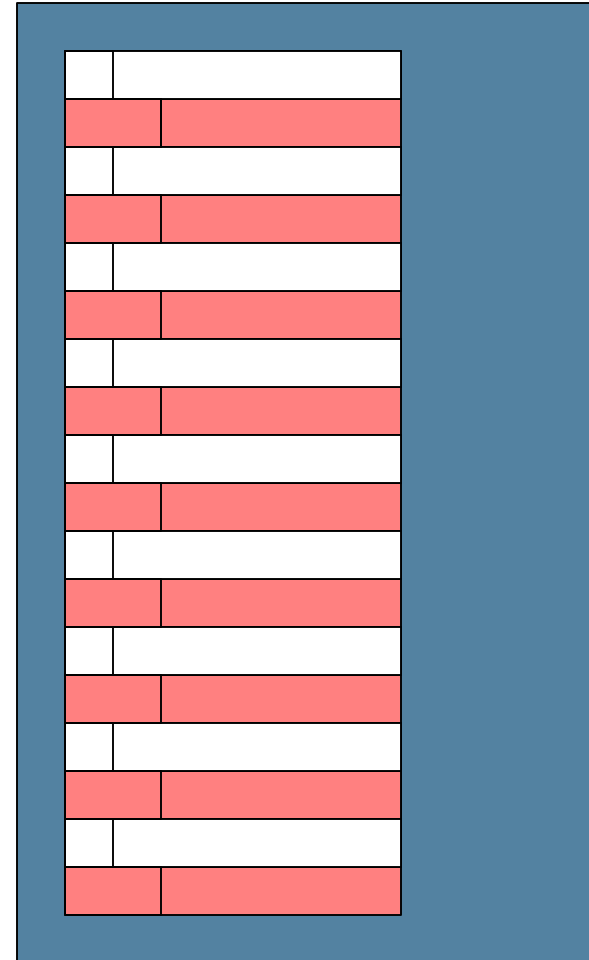
- Multi-threaded bookkeeping nightmare

Simple, atypical loads

Challenge:

Repeat “insert ... select...”

Fill cache with old and new records



Simple, atypical loads

Challenge:

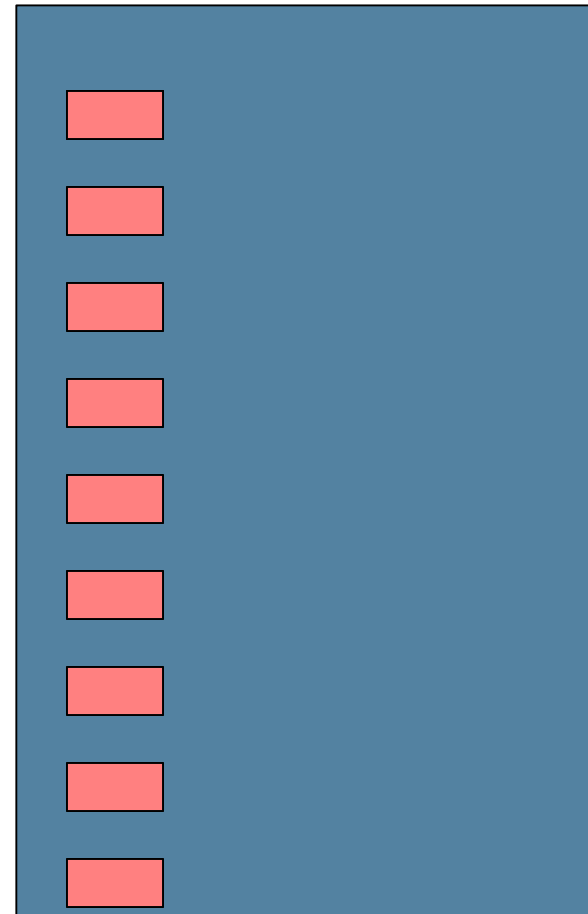
Repeat “insert ... select...”

Fill cache with old and new records

First solution

Scavenge old records

Chill new record data



Simple, atypical loads

Challenge:

Repeat “insert ... select...”

Fill cache with old and new records

First solution

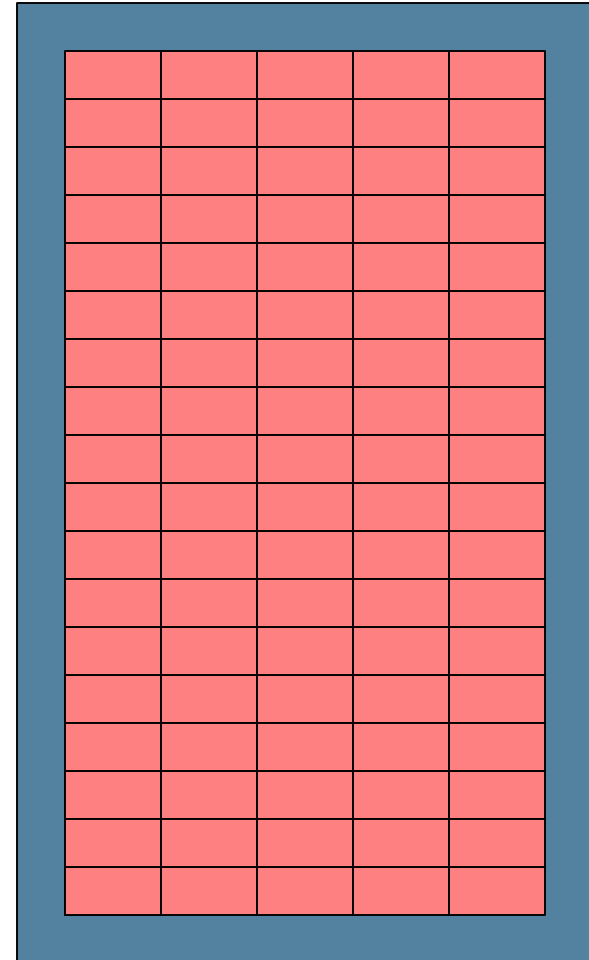
Scavenge old records

Chill new records

Second solution

Move the records headers out

Also helps index creation



Simple, atypical loads

Single pass read of large data set

Read more records than

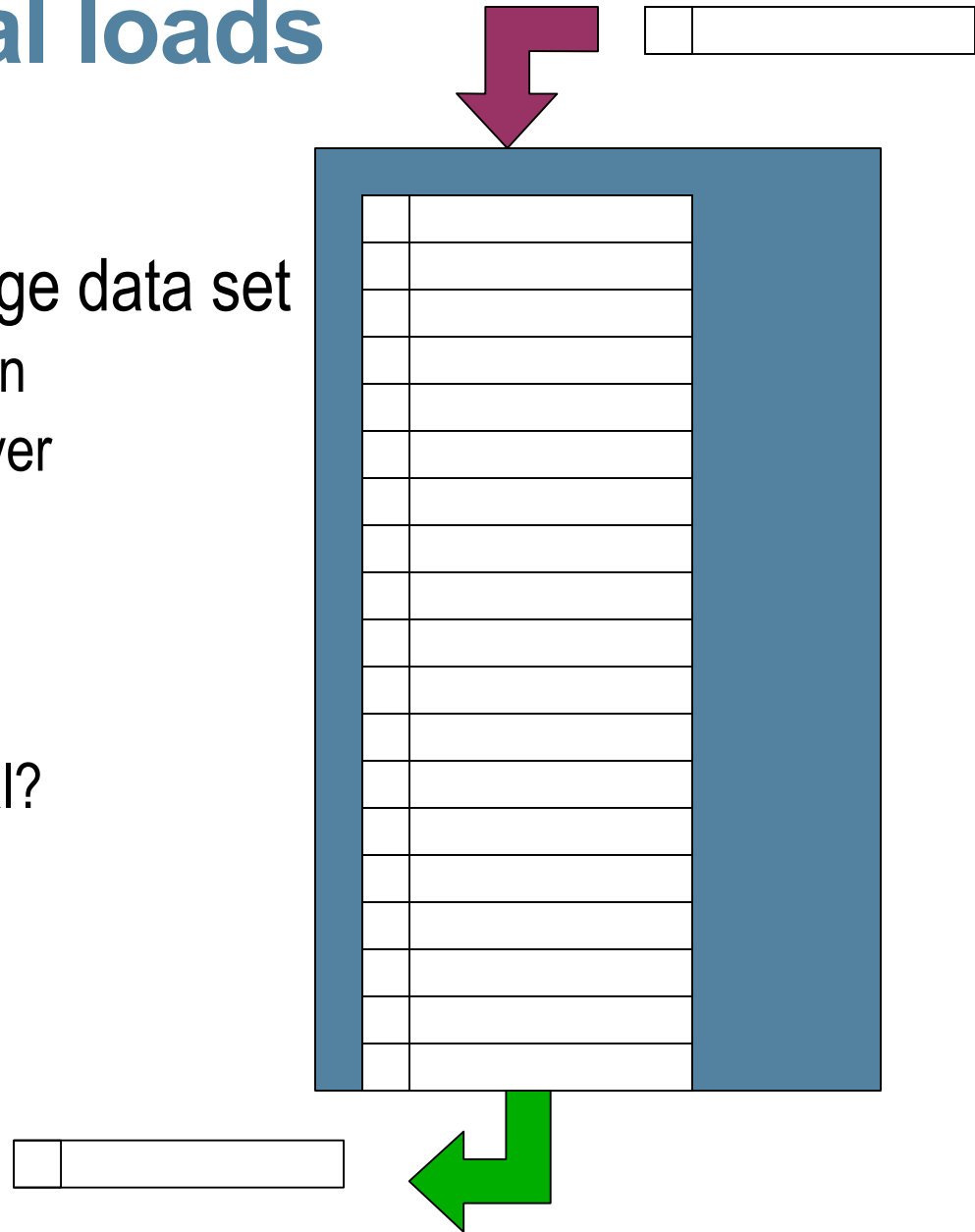
Read them over and over

Caches are useless

Encoding is overhead

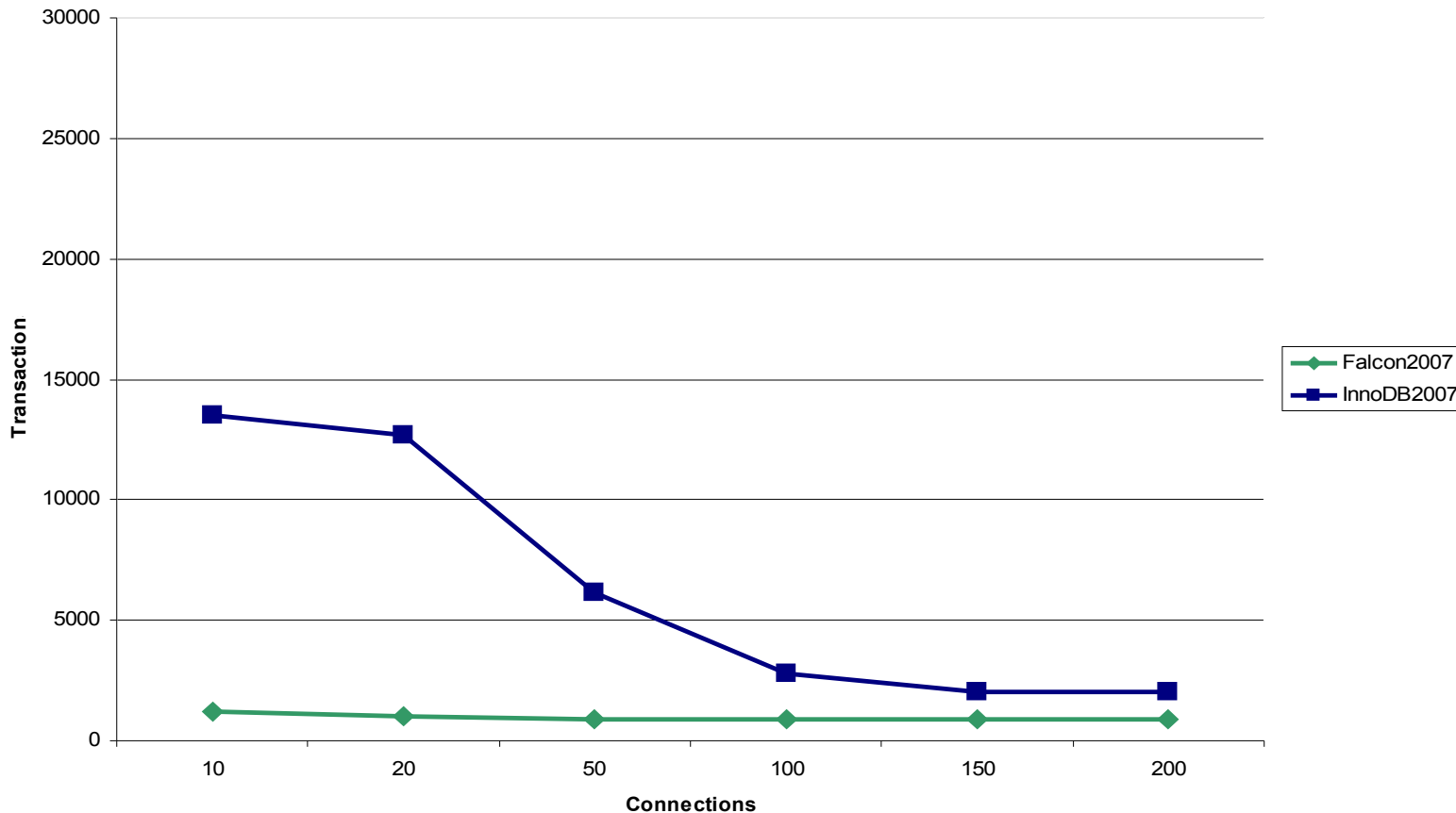
Response:

Make encoding optional?



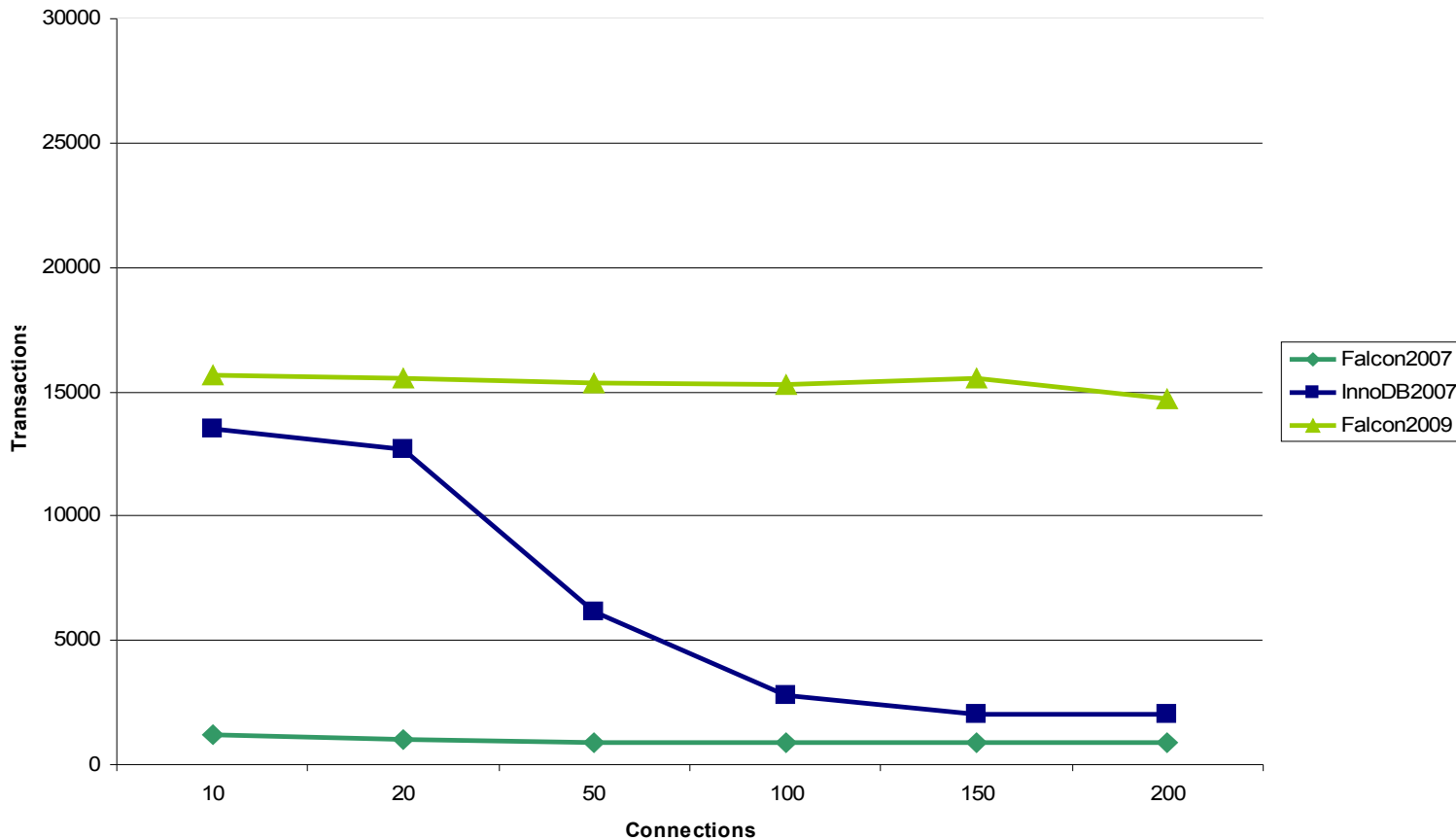
Challenge InnoDB on DBT2

Initial results were not encouraging (2007)



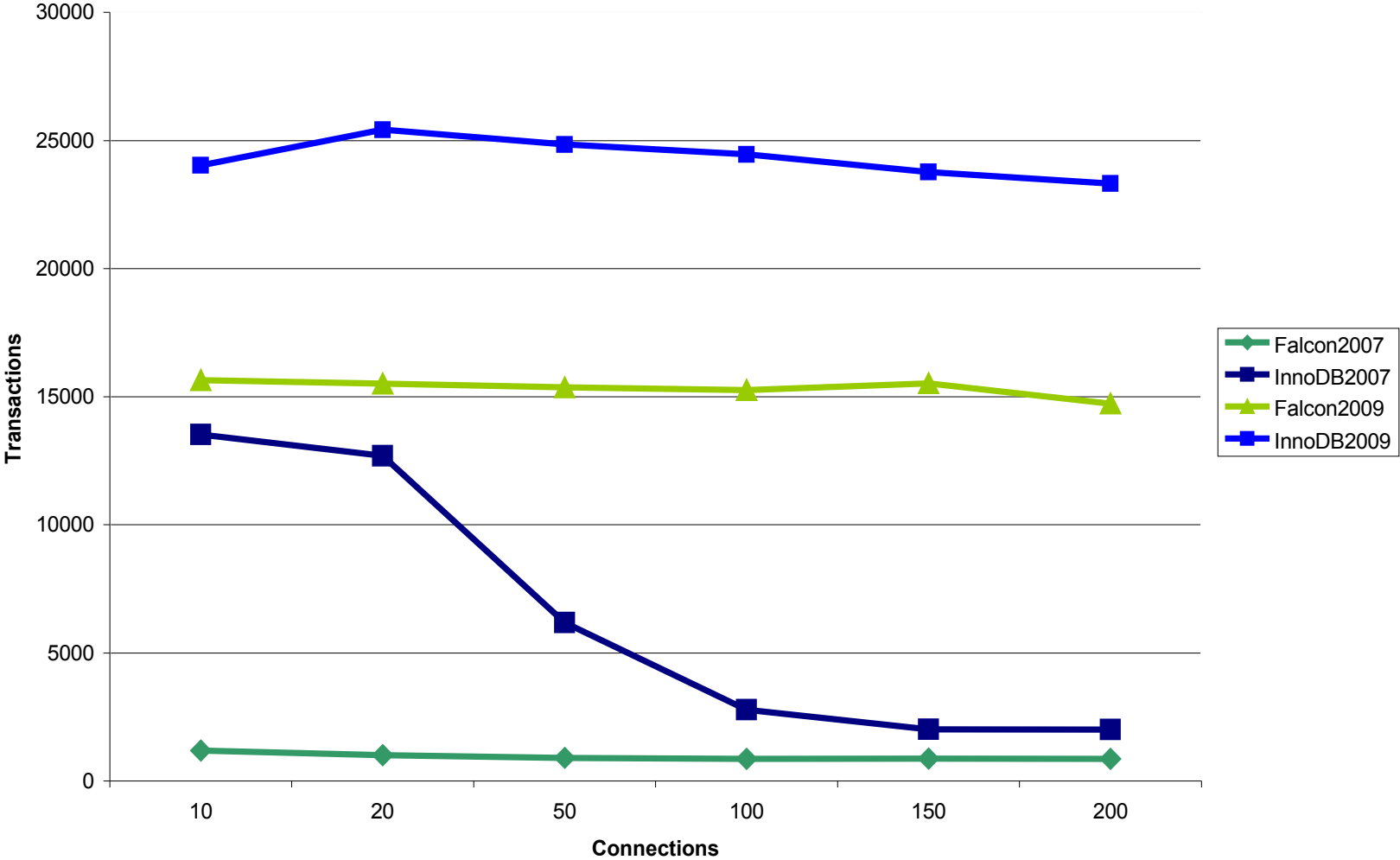
Challenge InnoDB on DBT2

But Falcon has improved a lot since April 2007

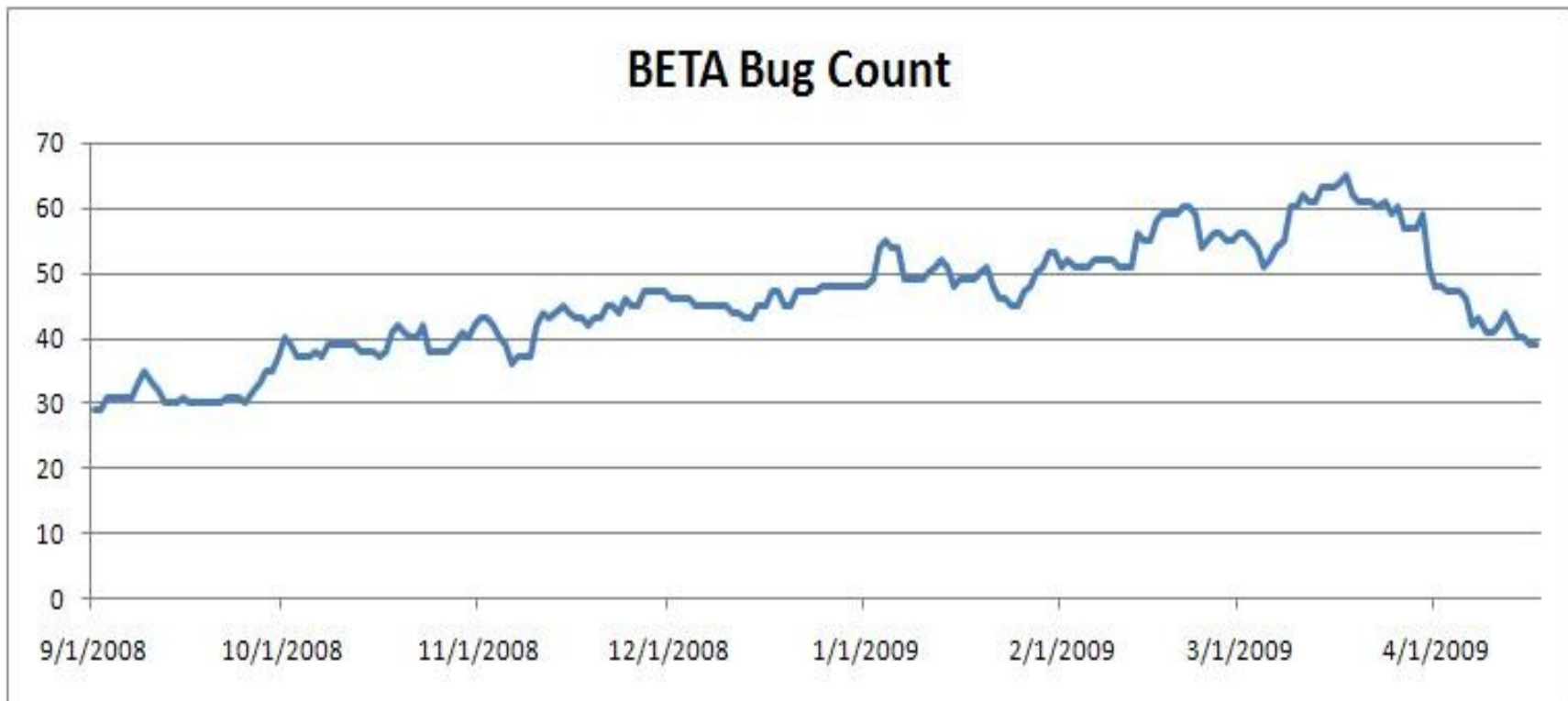


Challenge InnoDB on DBT2

So did InnoDB



Bug trends



Multi-threading

Databases are a natural fit for multi-threading

- Connections

- Gophers

- Scavenger

- Disk reader/writer

Except for shared structures

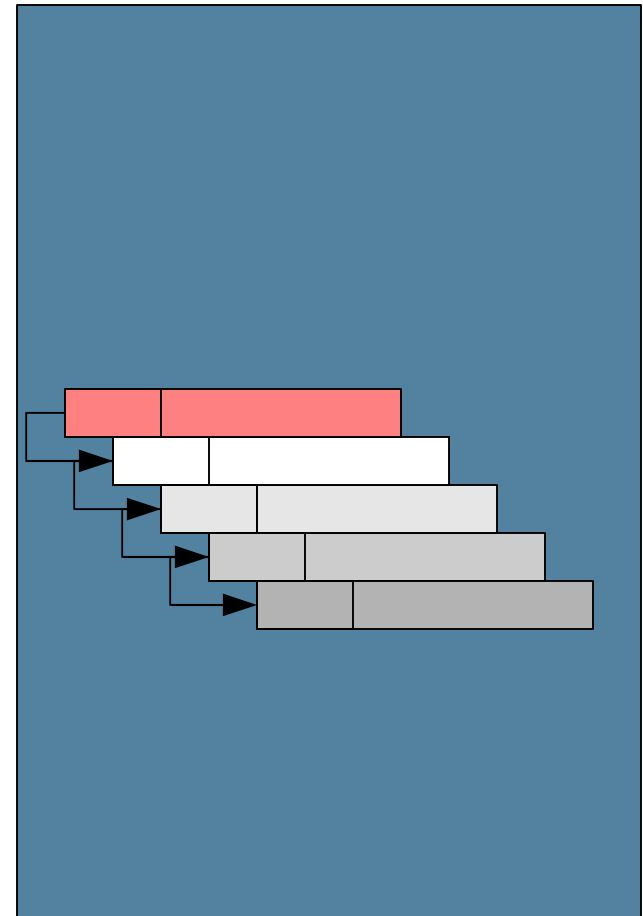
Locking blocks parallel operations

Challenge – sharing without locking

Multi-threading

Non-locking operation

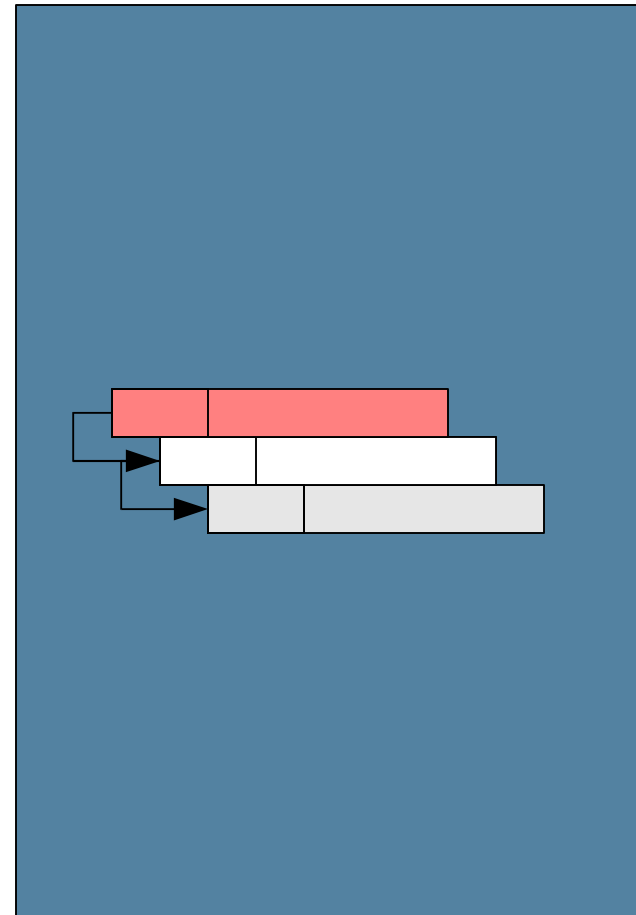
Purge old record versions



Multi-threading

Non-locking operation

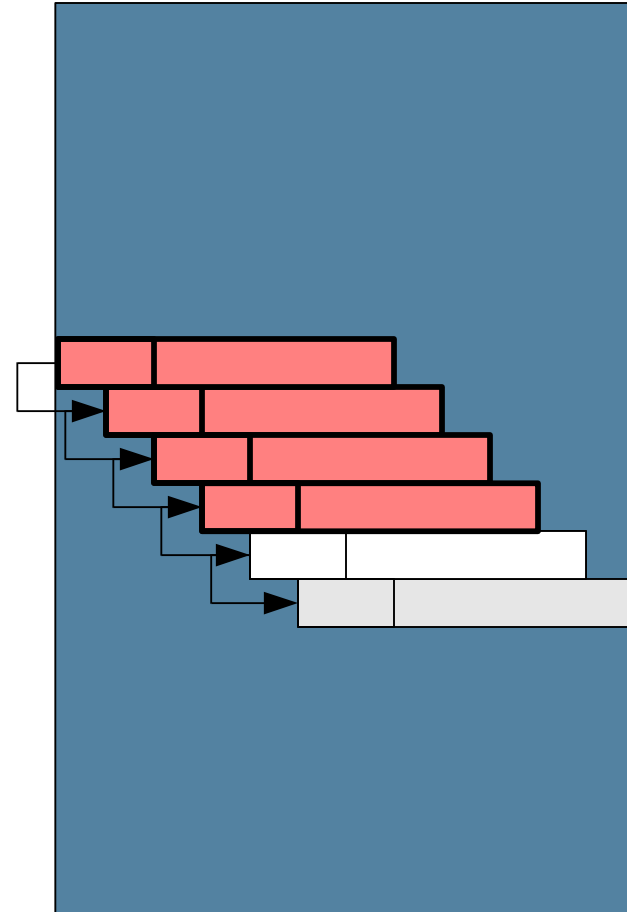
Purge old record versions



Multi-threading

Locking operation

Remove intermediate versions

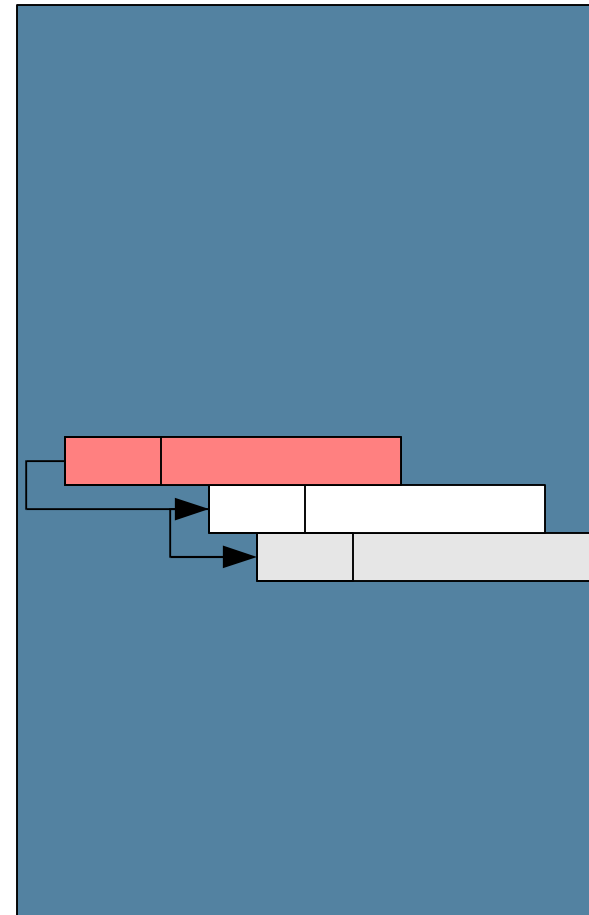


Multi-threading

Locking operation

Remove intermediate versions

What granularity of lock?



Multi-threading – Lock granularity

One per record:

Too many interlocked instructions

One per record group:

Thread reading one record prevents scavenging of another

No answer is right – more options?

Cycle locking – read record chain

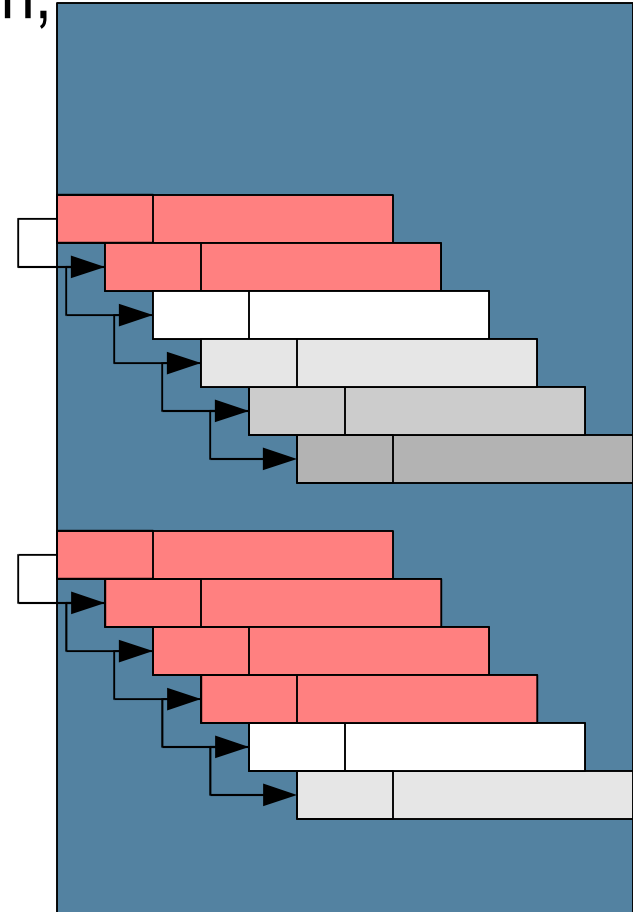
Before starting to read a record chain, get a shared lock on a “cycle”

Cycle 1 = 3 shared	Cycle 2 inactive
-----------------------	---------------------

Transaction A

Transaction B

Transaction C



Cycle locking – clean a record chain

Before starting to read a record chain,
get a shared lock on a “cycle”

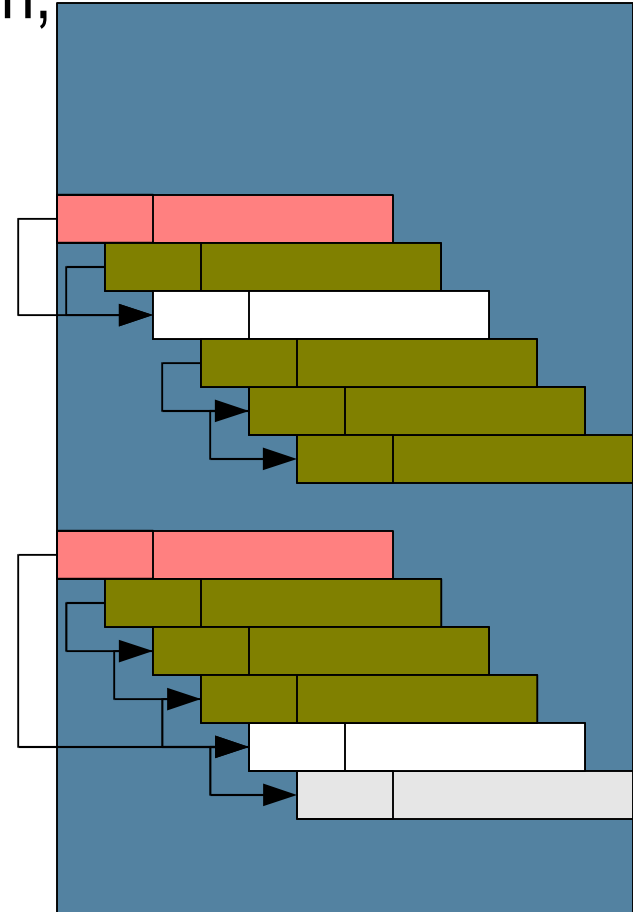
Cycle 1 = 4 shared	Cycle 2 inactive
-----------------------	---------------------

Transaction A active in Cycle 1

Transaction B active in Cycle 1

Transaction C active in Cycle 1

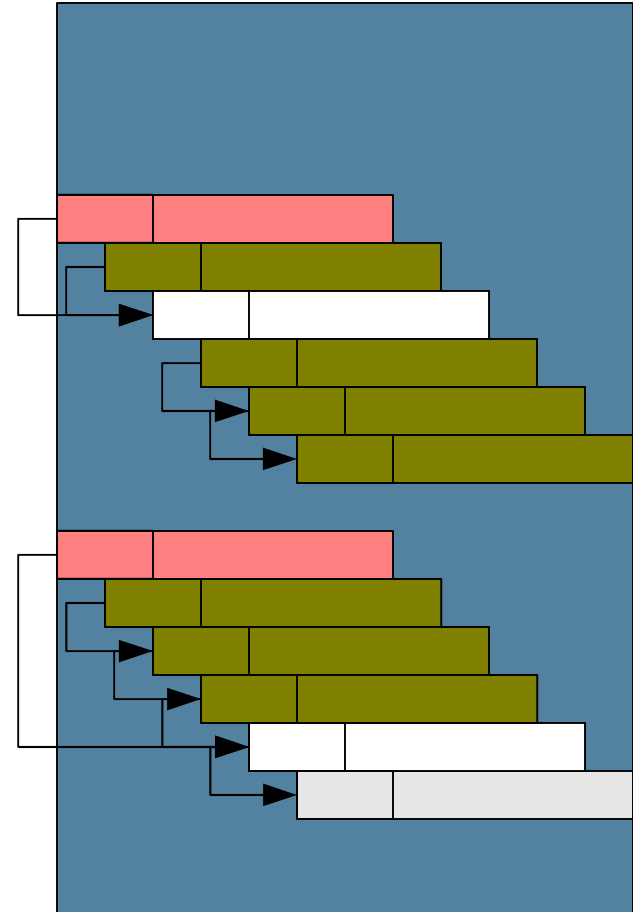
Scavenger unlinks versions
from record chain and links them
to a “to be deleted” list.



Cycle locking – records relinked

Cycle 1 = 1 shared	Cycle 2 inactive
-----------------------	---------------------

Transaction A releases lock
 Transaction B releases lock
 Transaction C still active
 Scavenger releases lock



Cycle locking – swap cycles

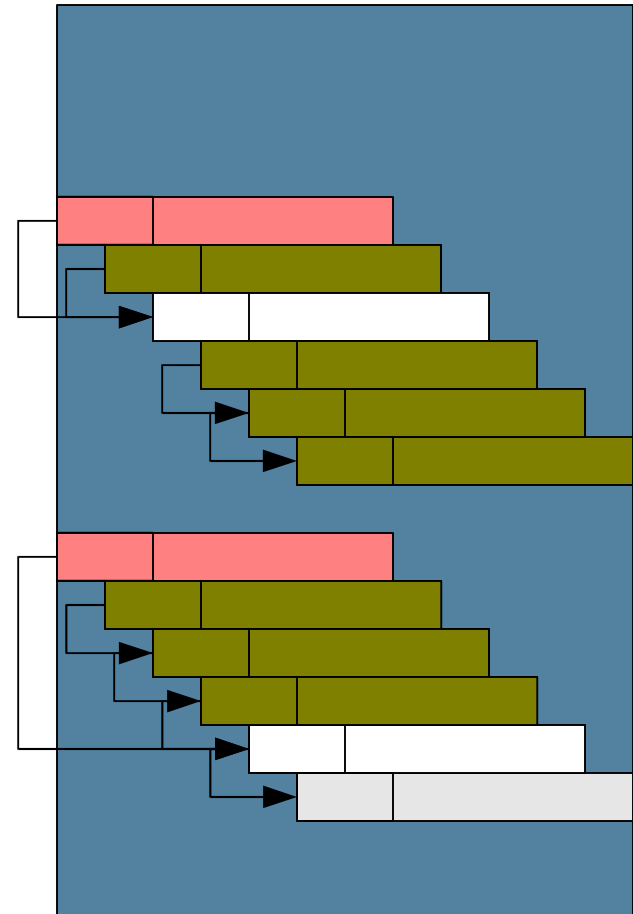
New access locks cycle 2

Cycle 1 = 1 shared	Cycle 2 = 1 shared
-----------------------	-----------------------

Transaction C holds Cycle 1 lock

Cycle Manager requests exclusive on Cycle 1 (pumps cycle)

Transaction A acquires Cycle 2 lock



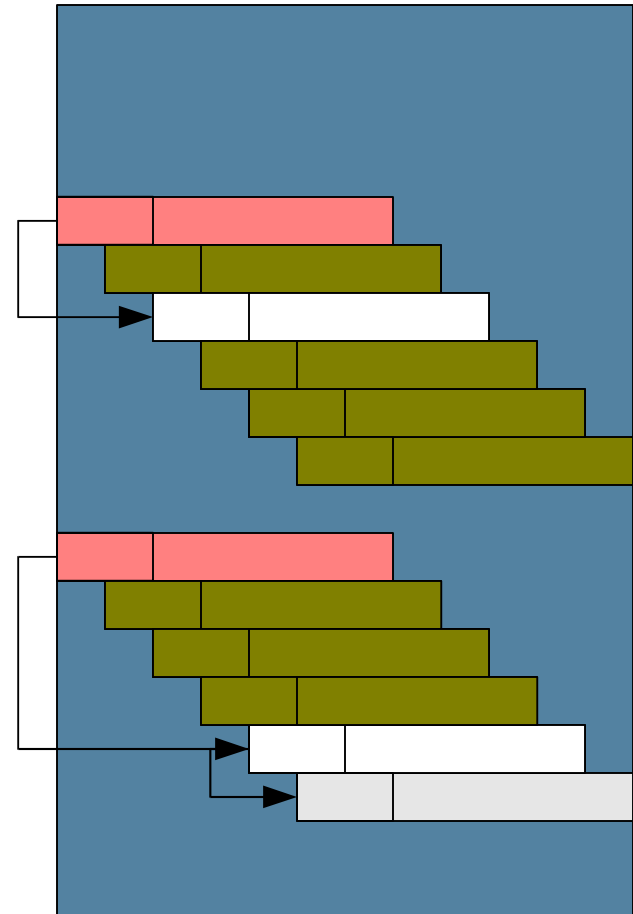
Cycle locking – cleanup phase

Cycle 1 = 0	Cycle 2 = 2
shared	shared
exclusive	

Transaction C releases lock

Transaction B acquires Cycle 2 lock

Cycle manager exclusive Cycle 1



Cycle locking – cleanup complete

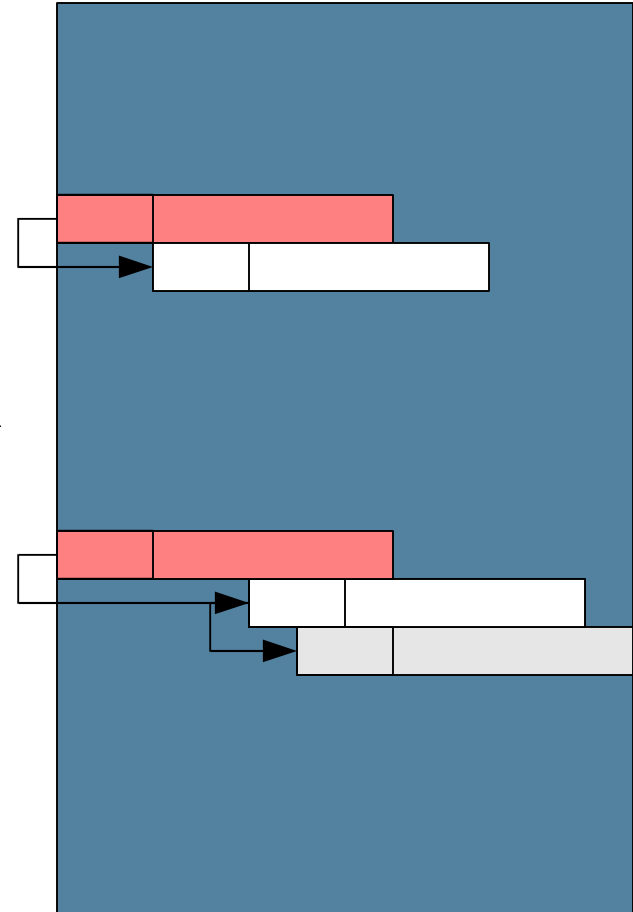
Cycle 1 exclusive	Cycle 2 = 2 shared
----------------------	-----------------------

Transaction C acquires Cycle 2 lock

Cycle manager exclusive Cycle 1

Remove unlinked, unloved, old versions

When cleanup is done, Cycle manager releases cycle 1



Questions