



# ***MySQL Performance & Scalability***

**Dimitri KRAVTCHUK**

**Benchmark Team  
Paris Sun Solution Center**



# Before we start...

- Few words about SSC :-)
- Paris  $\Leftarrow$  10Mbit, 20ms latency  $\Rightarrow$  LLG



# SSC Locations

- North America
  - > USA: Hillsboro, Broomfield, McLean, Menlo Park
- Latin America
  - > Sao Paulo, Brazil; Ft. Lauderdale, Florida; Mexico City, Mexico
- Europe
  - > Edinburgh, Frankfurt, Madrid, Manchester, Milan, Munich, Paris, Walldorf
- Asia
  - > Bangalore, India; Beijing, China; Hong Kong; Seoul, Korea; Singapore; Taipei, Taiwan; Tokyo, Japan;
- Pacific
  - > Sydney, Australia



# Sun Solution Center Is Near You



ASSCs in BLUE  
SSCs in BLACK

## United States

- San Francisco Bay Area, CA
- Hillsboro, OR
- Broomfield, CO
- Mc Lean, VA
- Chicago, IL - Diamond Management
- Plano, TX - EDS
- College Park, MD - Univ of Maryland
- Pittsburgh, PA - Deloitte Consulting

## Latin America

- Ft. Lauderdale, FL, USA
- Mexico City, Mexico
- Sao Paulo, Brazil

## Europe / Middle East / Africa

- Edinburgh, Scotland, UK
- Manchester, UK
- Warrington, UK - Avnet
- Paris, France
- Frankfurt, Germany
- Munich, Germany
- Walldorf, Germany
- Milan, Italy
- Madrid, Spain
- Göteborg, Sweden - Inserve Technology
- Helsinki, Finland - ArrowECS
- Tallin, Estonia - Microlink
- UAE - Tech Access

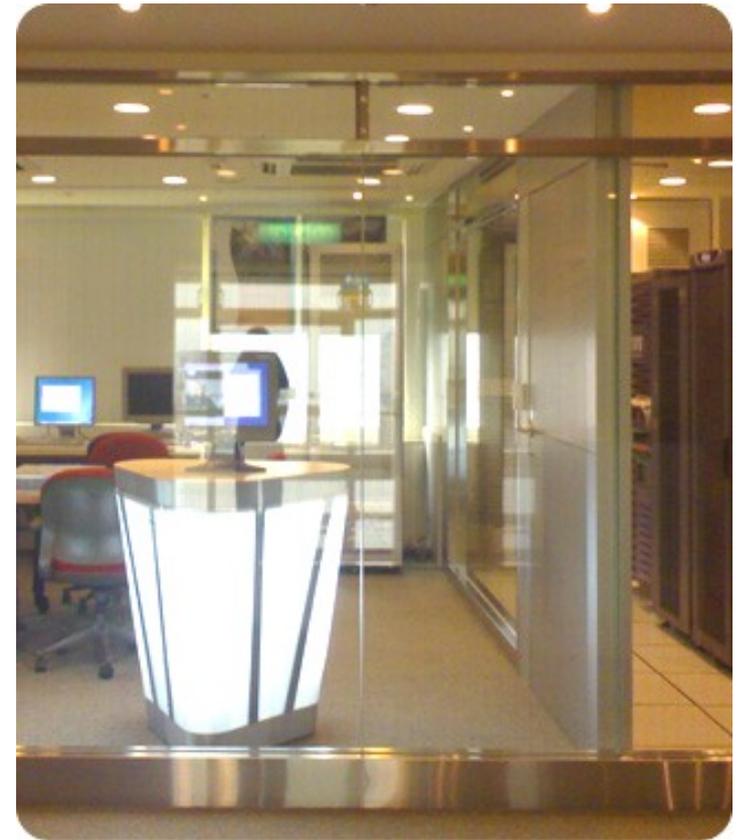
## Asia Pacific

- Bangalore, India
- Bangalore, India - Wipro
- Beijing, China
- Hong Kong, China
- Shenyang, China - Neusoft
- Seoul, Korea
- Singapore
- Singapore - Ingram Micro
- Sydney, Australia
- Sydney, Australia - Express Data
- Tokyo, Japan

# Sun Solution Center

## Benchmark and Performance Characterization

- Architecture design
- High-end performance and scalability (servers, storage)
- Performance characterization
- Competitive benchmarks
- Internal product BU benchmarks
- Performance tuning
- Customer/Partner benchmarks
- Customer briefings



# Sun Solution Center

## Partner Solution Center

- Architecture design and validation
- Portfolio management and solutions offerings
- Customer/Partner Proof-of-Concepts
- End-to-end software development for live customers
- Industry solutions development and showcase
- Building of horizontal/biz solutions (eg: IdM, Security ... etc.)
- Business innovation and compliance (SOX, HIPA ... etc.)
- Demos, solution showcase

# To know more

<http://www.sun.com/solutioncenters>

## Test for success.

We assembled the best team in the industry to assess unique business solutions.



Overview

Services

Locations

Get Started

At a Glance | Welcome Letter | FAQs



"Most of our customers share two characteristics; they believe in the power of the community to solve challenging problems, and they believe that technology is a competitive differentiator for their business. The Sun Solution Centers bring together state-of-the-art technology and expertise in simulated environments where our clients can envision, build, and test innovative business solutions." Jonathan Schwartz, President and CEO.

### What can Sun Solution Centers do for you?



The goal of the Centers is to minimize your risk, justify your expense, and shorten time to deployment of your new business solutions by providing the tools you need to 'test before you invest'. We do this by offering Sun and Sun partner access to in-depth expertise in technologies, industries, and applications in collaborative, state-of-the-art

### Working with Sun Solution Centers

#### » How to Get Started

Considering a new business solution? Interested in exploring in-depth what the power of Sun can do for you? Get started by contacting your Sun Account Manager or Systems Engineer. They can initiate the process by discussing your needs with you and then requesting an engagement with the Sun Solution Center.

### This Month's Top 5 Requested Services

- Finance Industry POC
- Telco POC
- SAP Sizing
- Customer Workshop
- HPC Performance Consulting

» See all Services



Authorized Sun Solution Centers

Find out where they are.

# Agenda

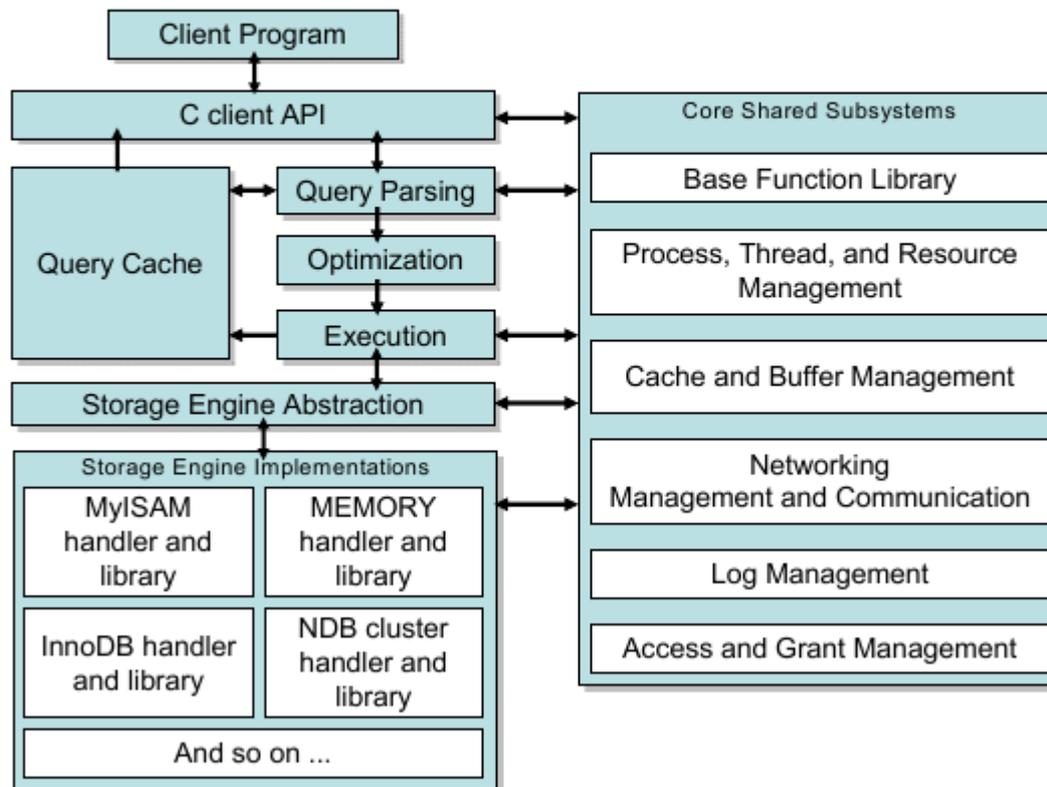
- MySQL History / milestones
- MySQL Architecture Overview
- Storage Engines
- Bottlenecks
- Performance
- InnoDB
- MySQL 5.4
- Tuning / Monitoring
- MySQL & ZFS

# MySQL History

- 1994 - Michael (Monty) Widenius & David Axmark
- 1995 – first release
- 2000 – MySQL 3.23
  - > 2001 – v.3.23.34a + InnoDB !
- 2002 – MySQL 4.0 (unions)
- 2004 – MySQL 4.1 (prep. statements, sub-queries)
- 2005 – MySQL 5.0 (trigg., views, cursors, XA, stor.proc.)
  - > Oracle + InnoDB...
- Nov.2008 – MySQL 5.1 (partitions, row-level replication)
  - > Feb.2008: Sun + MySQL
- Apr.2009 – MySQL 5.4 : Performance! :-)

# MySQL Design

## MySQL Architecture Overview (High Level)



# MySQL Design

- Multi-Threaded database !
  - > Fast context switch
  - > Simplified data access
  - > Concurrency?..
- Storage Engines
  - > CREATE TABLE ... ENGINE=<NAME\_OF\_ENGINE>
  - > ALTER TABLE ... ENGINE=<NAME\_OF\_ENGINE>

# MySQL Storage Engines

- MyISAM (default engine, owned by Sun/MySQL)
- InnoDB (transactional, owned by Innobase/ORACLE)
- NDB (Cluster(!), owned by Sun/MySQL)
- Falcon (transactional, owned by Sun/MySQL)
- PBXT (transactional, owned by PrimeBase)
- XtraDB (transactional, InnoDB fork by Percona)
- MEMORY (in memory data, Sun/MySQL)
- etc...

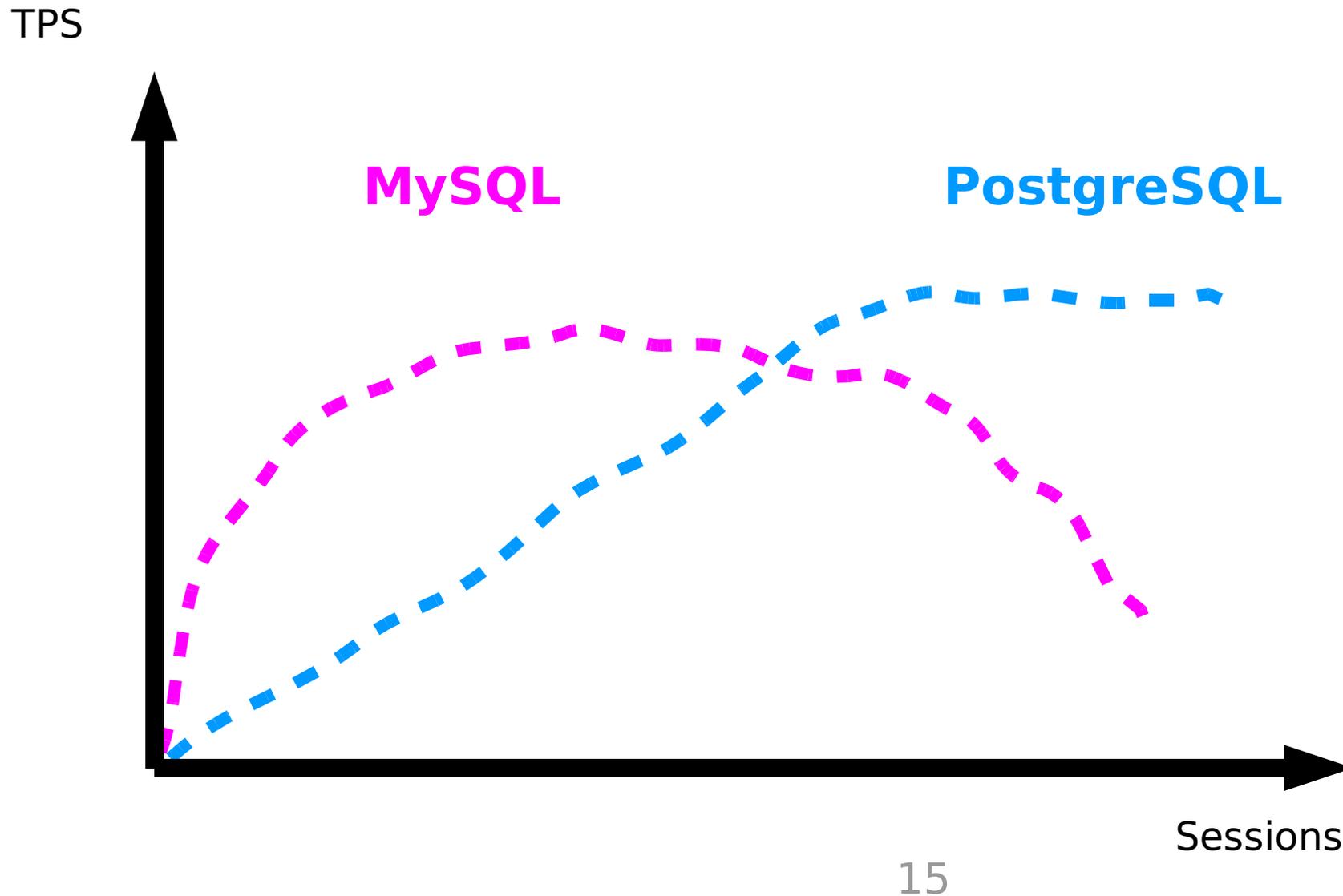
# MyISAM Engine

- Non-transactional! / No fast recovery! :-)
- Cache
  - > Index only
  - > Data => FS cache
  - > mysql> flush tables;
- Single Writer @Table
  - > Main bottleneck! => single writer
  - > Solutions: delayed inserts, low priority
- Query plan
  - > Index forcing may be necessary (hint)
- Maria Engine: next step for MyISAM

# MySQL with MyISAM Success

- Full Text search queries!
- SELECT count(\*) ... :-))
- Extremely SIMPLE!
  - > my.conf => configuration parameters
  - > mysql.server start / stop
  - > Database => directory
  - > Table => directory/Table.MYD, Table.MYI, Table.frm
  - > \$ cp Base1/Table.\* /other/mysql/Base2
- Data binary compatibility! (ex: reports via NFS)
- Replication ready!
- Very FAST! (until some limit :-))

# MyISAM vs PostgreSQL (in 2000)

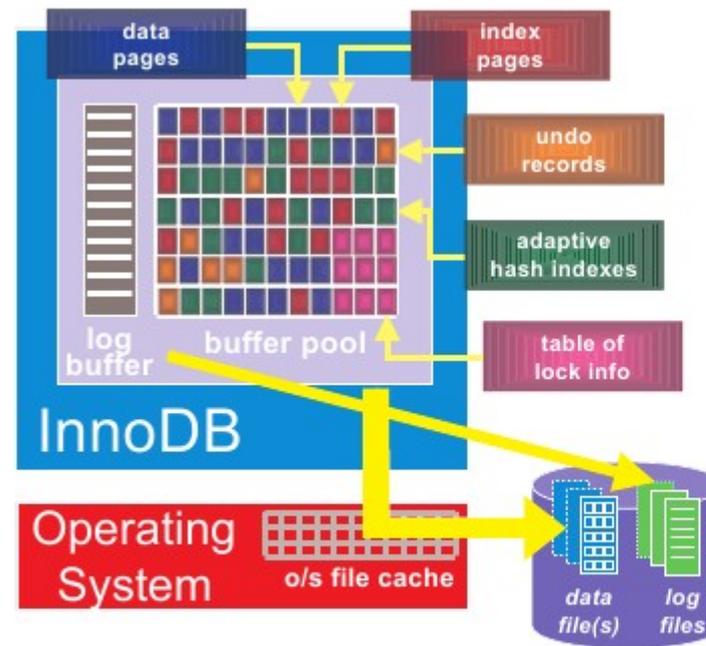


# InnoDB Engine

- Row-level locking
- True transactions / UNDO
- Auto recovery
- Double write / Checksums
- Tablespaces or File per Table option
- Buffer pool
- Multi-threaded
- The fastest transactional disk-based MySQL Storage Engine!

# InnoDB Design

- Index-only read
- Fuzzy Checkpoint
- Group Commit
- Log flush policy
- Threads:
  - > User sessions
  - > Master
  - > Read Ahead
  - > Page Writer
  - > Log Writer
- Performance

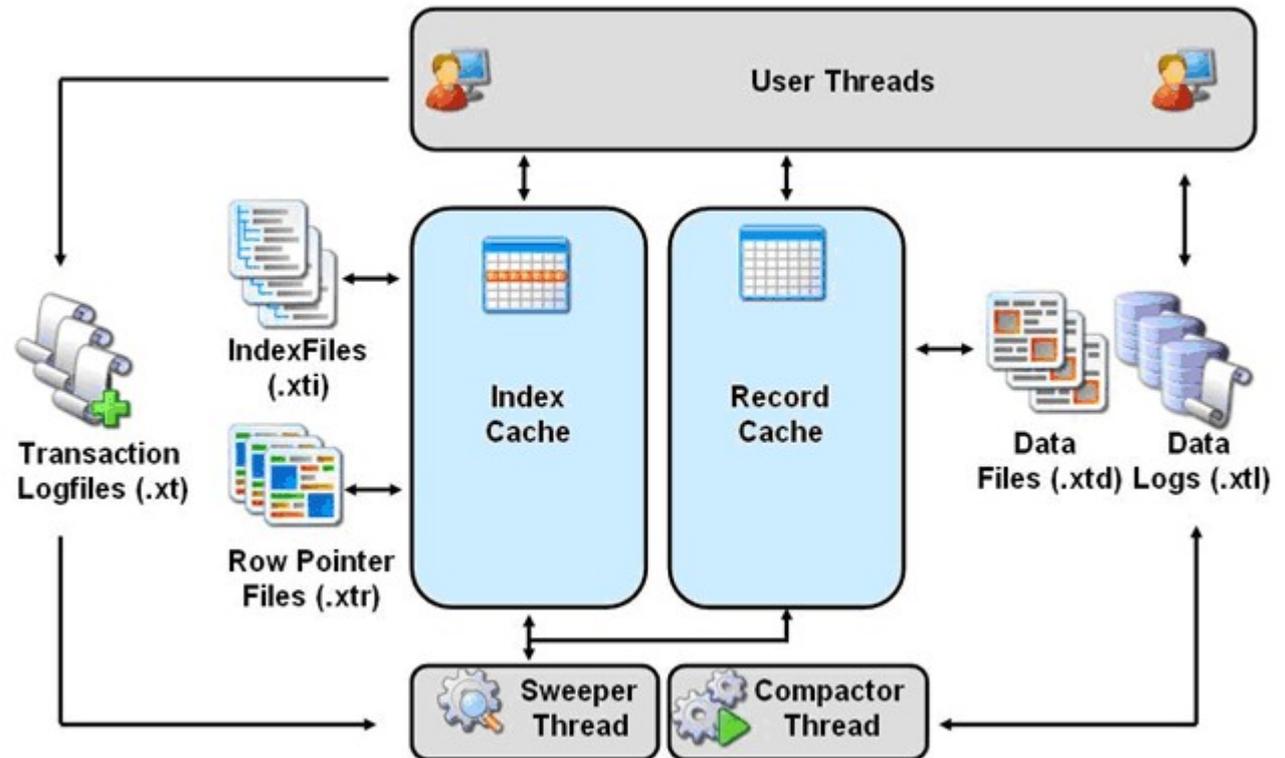


- When data & index pages are read, they are cached for re-use, and are replaced when least-recently-used
- InnoDB on-the-fly creates adaptive hash indexes depending on query pattern (more on this later)
- On updates, row-level locking information is maintained in an efficient bit-map
- Undo info, used to reverse a transaction's changes, is cached in memory, later written to sys tablespace
- Compact representation of changes is kept in log buffer. On commit, log records are written to log file (but no buffer pool pages need to be written)
- Eventually, data, index and undo pages are flushed to data files

INNOBASE

# PBXT Engine

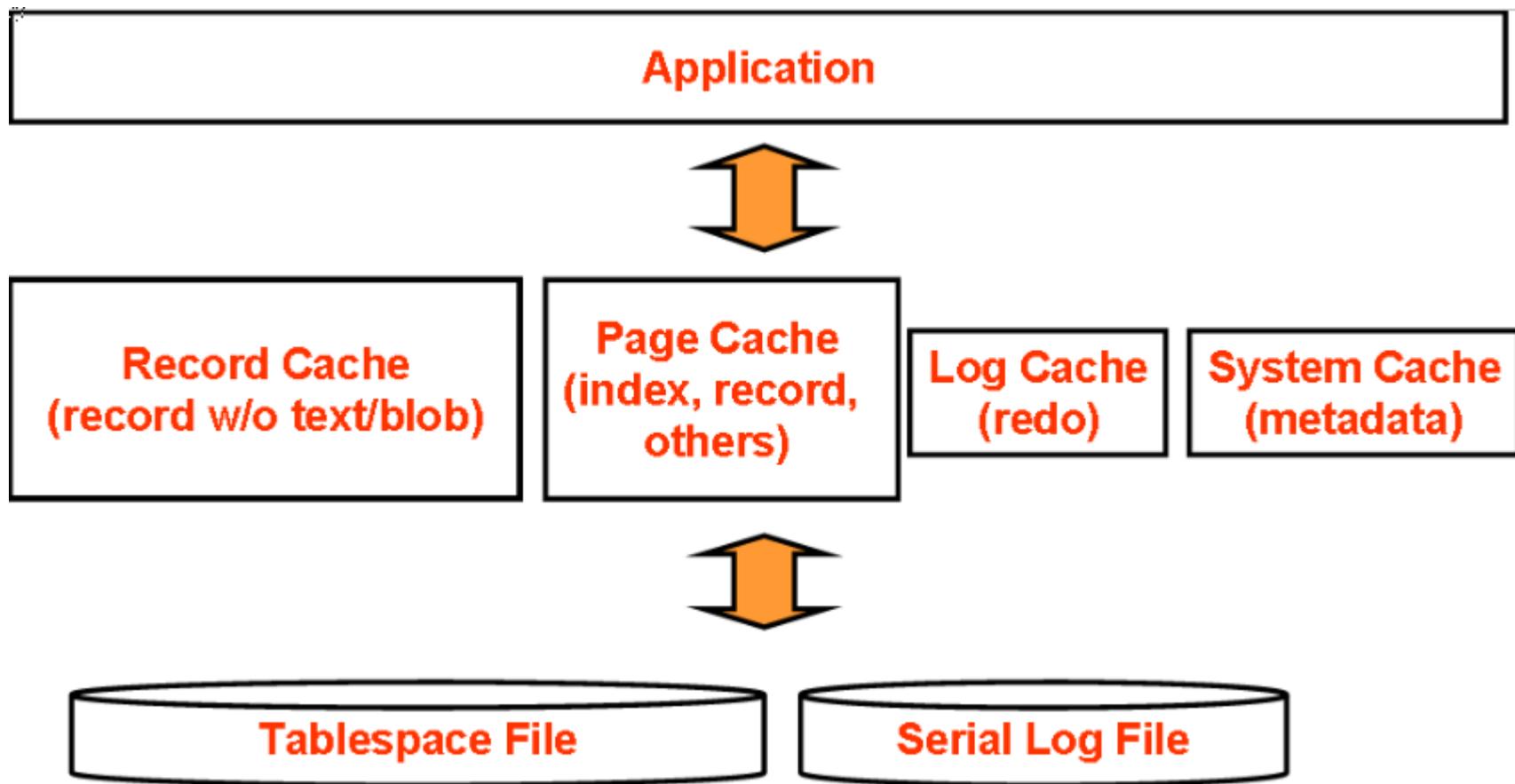
- Developed by PrimeBase
- Very promising!
- But let's see...



<http://dev.mysql.com/tech-resources/articles/pbxt-storage-engine.html>

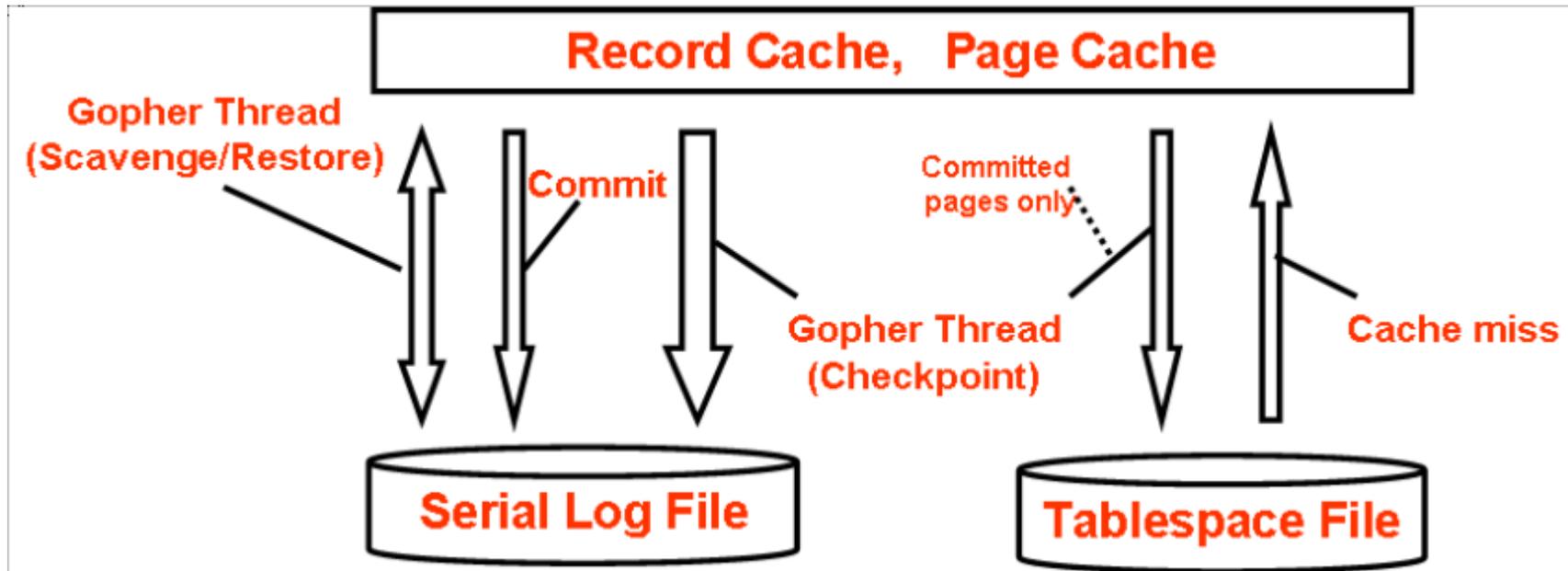
# Falcon Engine

- <http://dev.mysql.com/tech-resources/articles/falcon-in-depth.html>



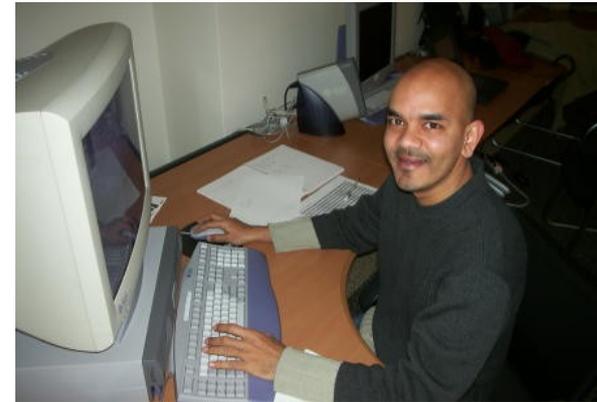
# Falcon Engine (2)

- Performance is much lower than others...
- Core dumps..



# NDB Cluster - History

- Feb.2002, Ericsson Benchmark
- Goal: 1M TPS
  - > Yes, **one million /sec** ...
- Alzato Team:
  - > Mohan PAKKURTI
  - > Johan ANDERSSON
  - > Mikael RONSTROM (not arrived yet)



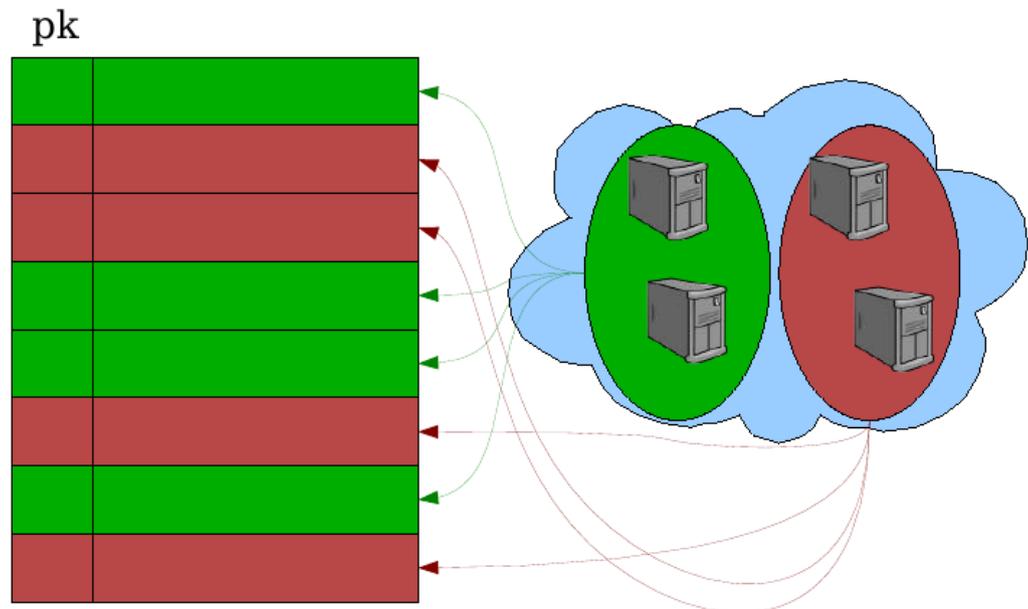
# Ericsson constraints

- Database server cannot be powered off!.. Never!
- Memory-based Database
- Expected H/W:
  - > Sun E25K, 72CPU 900Mhz, 288GB RAM
  - > 8 Storage arrays (T3), ~1TB data
- Expected performance:
  - > 1M TPS read-only
  - > 200K TPS mixed (updates)

# NDB Design

- Shared nothing nodes
- All data always synchronously replicated (2 or more times, two phase commit)
- Changes are flushed to disk on checkpoint

SELECT \* from t1



# Benchmark, Week #1

- Nothing works...

# Benchmark, Week #2

- NDB is running! (finally)
- But.. crashing all the time..
- From few tests which worked normally we may see the main bottleneck is the NDB *interconnect!*
- Moving to SHM model (*shared memory communications*)
- I'm tracing the source code and adding my own *performance stats counters* to monitor the application live with `dim_STAT`
- Debugging continues...

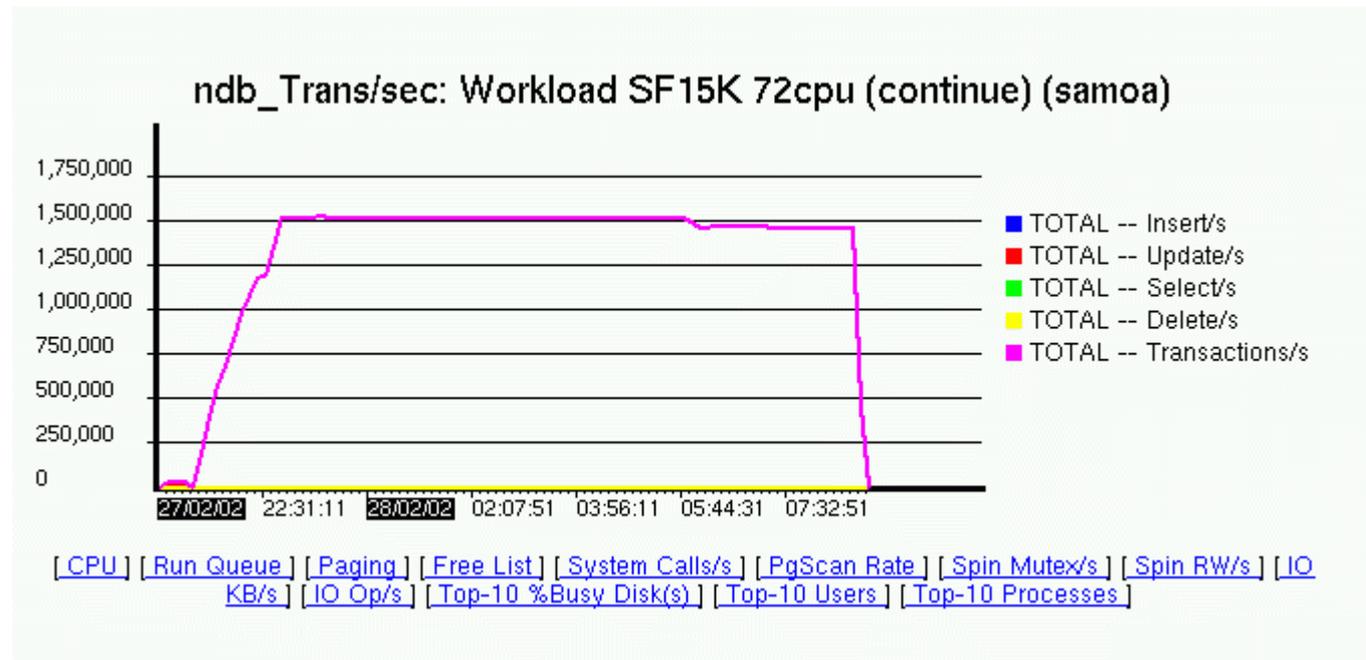
# Benchmark, Week #3

- Mikael arrived! (and managers..)
- Design Discussions again and again..
- 1. Analyzing bottlenecks..
- 2. Improving the code
- 3. Probe test
- 4. GOTO 1



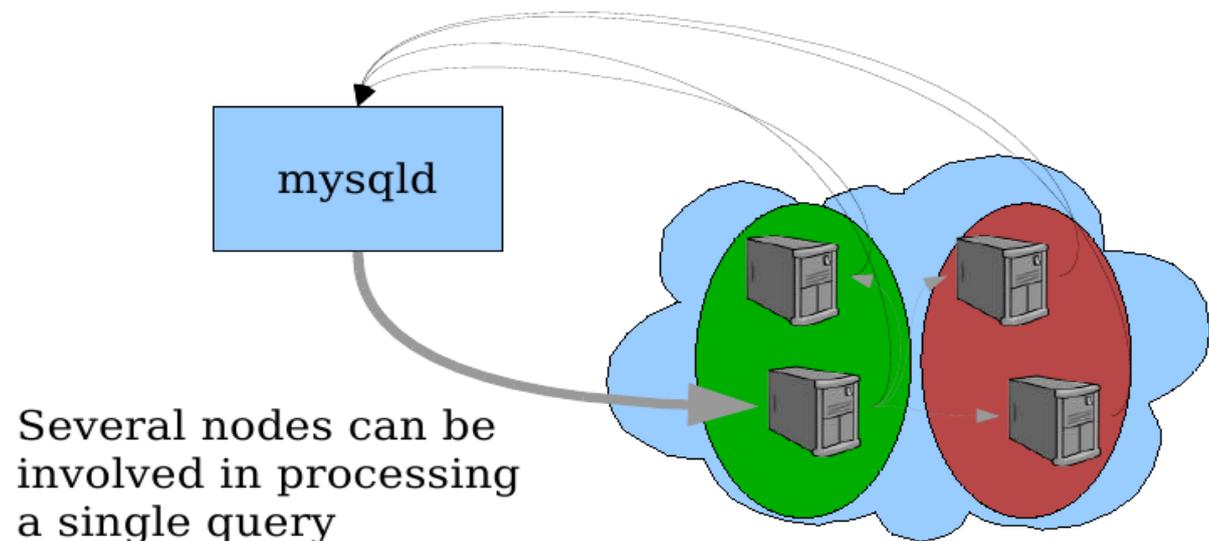
# Benchmark, Finally...

- Final results on 32CPU:
  - > 1.5M TPS on read-only!
  - > 340K TPS on updates!
- What about 64CPU?..
  - > ... :-)



# NDB Cluster & MySQL

- Oct.2003 – MySQL \$\$ Alzato
- Fully integrated within MySQL
- NDB API



Parallelism=Better Performance

# MySQL Performance (general)

- Choose the right Engine for each of your table/database
  - > Read-Only / Text search => MyISAM
  - > Read+Write / Transactions => InnoDB
  - > Short/Small Transactions + DB fits in RAM => NDB
- Tune / Optimize your queries
- Once scalability limit is reached => go for Distributed:
  - > Sharding
  - > Master / Slave(s) => role-based workload
  - > Any other similar :-)
- Used by: Google, Facebook, Amazon, USA elections(!)..
- Scalability = Main Performance Problem!...

# MySQL Performance @2007

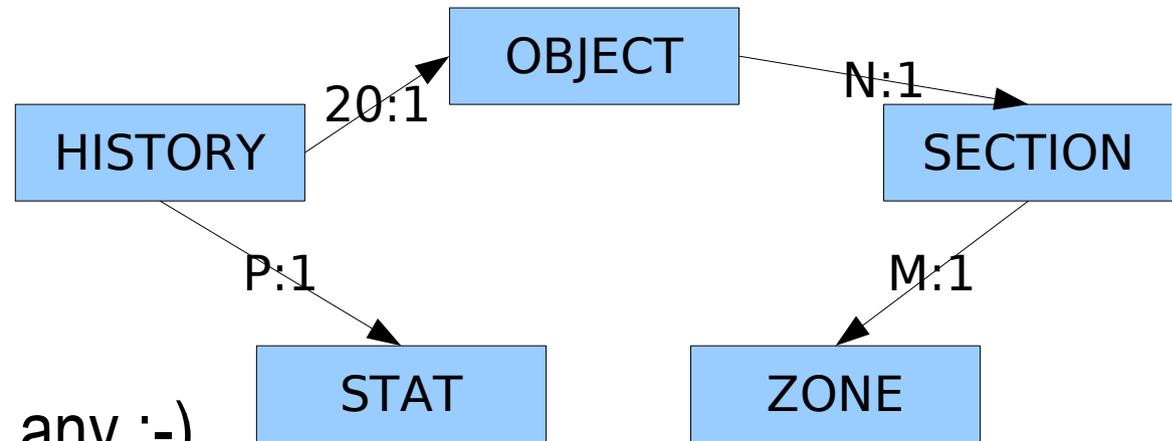
- Real customers start to come in Benchmark Center :-)
- We start to see real customers using MySQL for financial operations! (I mean real money transactions are stored in databases, not blogs/ forums/ web pages :-))
- Customers expected to stress their databases to validate them for production workload..
- Let me present you dbSTRESS :-)

# db\_STRESS Benchmark

- Based on real customer's workload (OLTP)

- Database schema:

- > STAT: 1000 rows
- > SECTION: 100 rows
- > ZONE: 10 rows
- > OBJECT: 1M, 10M, ... any :-)
- > HISTORY: x20 number of OBJECTs



- Transactions:

- > READ [and WRITE]
- > Configured by Read/Write ratio (depending on goal workload)
- > NOTE: may be easily modified for another schema/workload!

# db\_STRESS Transactions

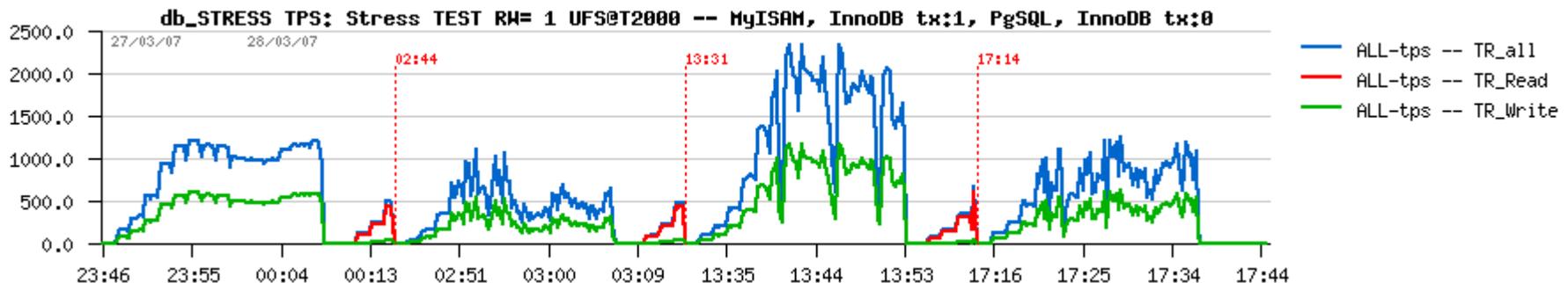
- READ – by random object ID:
  - > SEL1: select join OBJECT-SECTION-ZONE by object ID
  - > SEL2: select join HISTORY-STAT data by object ID
- WRITE – by found object ID
  - > DEL: delete one HISTORY record by object ID
  - > INS: insert one HISTORY record for object ID
  - > UPD: update one HISTORY record by object ID
- Options:
  - > Read / Write ratio: 1, 5, 10, ... (reads per write)
  - > UPDATE-only=0/1 (not executing DELETE & INSERT)
- Live TPS + resp.time stats for each user session!

# db\_STRESS Scenario examples

- Stress scenario:
  - > Goal: max TPS and +/- stable TPS level after max
  - > Think time: 0 sec (full stress)
  - > Concurrent Sessions: 1, 2, 4, ..., 256 (increasing step by step)
  - > Ratio Write / Read: 0, 1, 10, 1000 (0 means read-only)
- 1600 users scenario:
  - > Goal: stable 1 TPS for each session, all sessions should stay "alive"
  - > Think time: 1 sec (cool)
  - > Concurrent Sessions: 50, 100, 200, 400, 600, 800, 1000, 1200, 1400, 1600 (increasing step by step)
  - > Ratio Write / Read: 0, 1, 10 (0 means read-only)

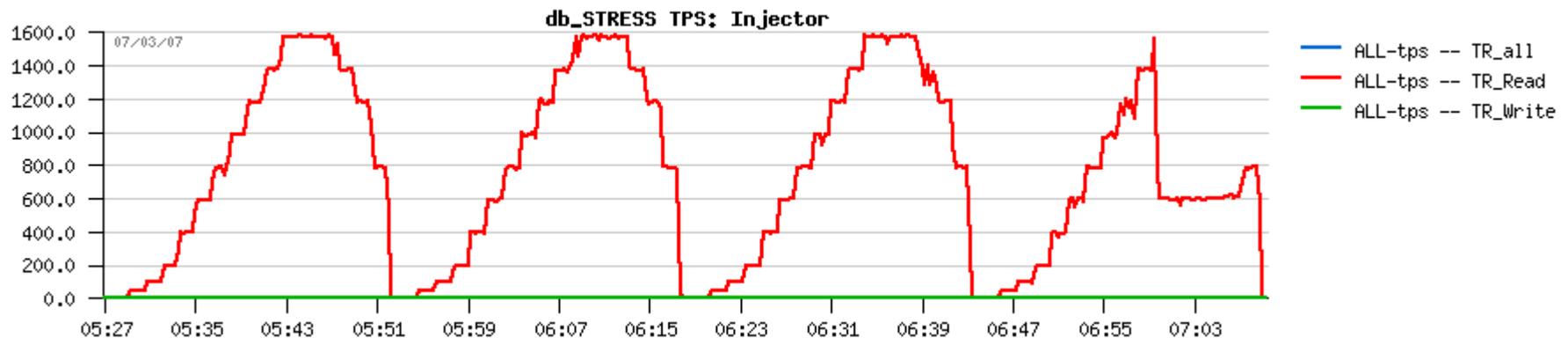
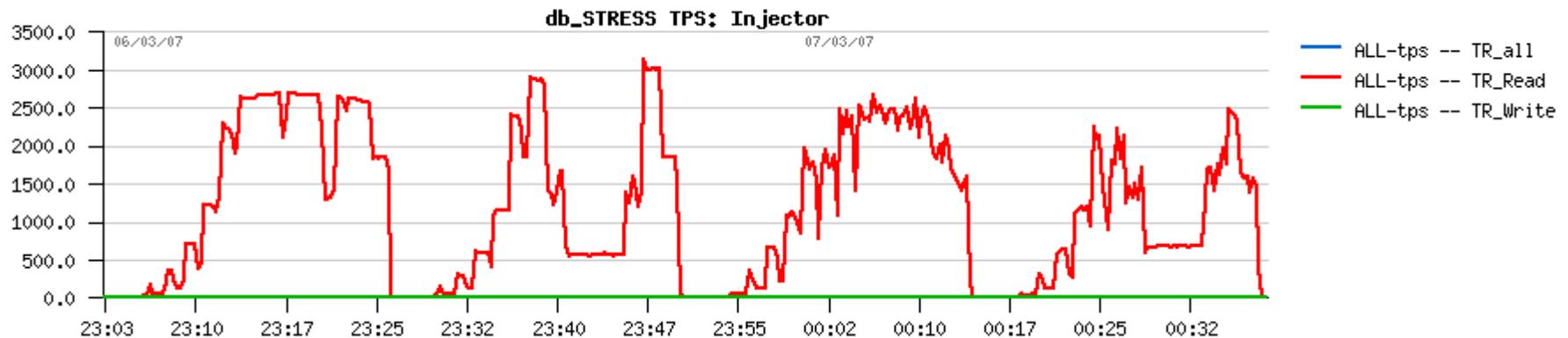
# MySQL @dbSTRESS in 2007

- Unstable results, x2 times slower vs PostgreSQL
- Key InnoDB parameters:
  - > innodb\_thread\_concurrency = 16
  - > innodb\_flush\_log\_at\_tx\_commit = 0/1/2
  - > + MT malloc



# MySQL Read-Only in 2007

- Huge amount of locks!
- Optimal H/W platform: 2 – 4CPU... : - (



# 2008 Year: Sun + MySQL

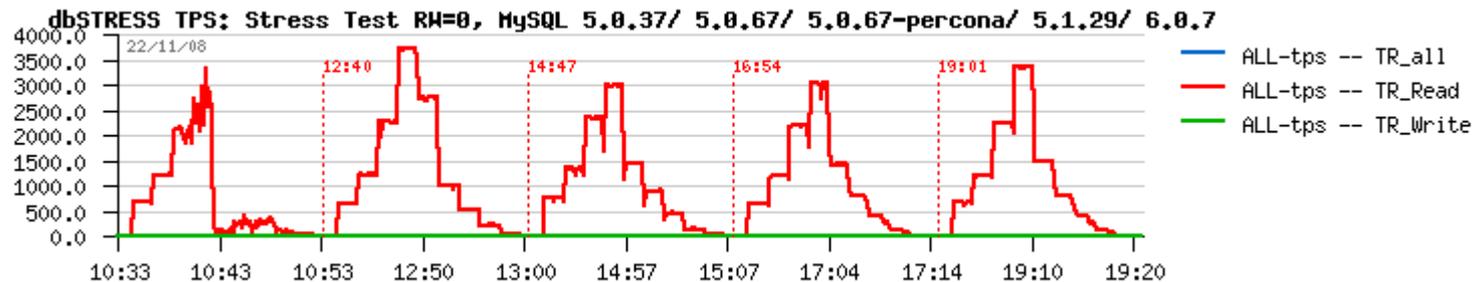
- Common projects
- Common benchmarks
- Discussions, discussions, discussions... :-)
- PAE Team => improve MySQL performance
  - > Lead by Allan PACKER
  - > From MySQL: Mikael Ronström (NDB father :-))
  - > MySQL Performance Virtual Team + Core Team

# MySQL / InnoDB @2008

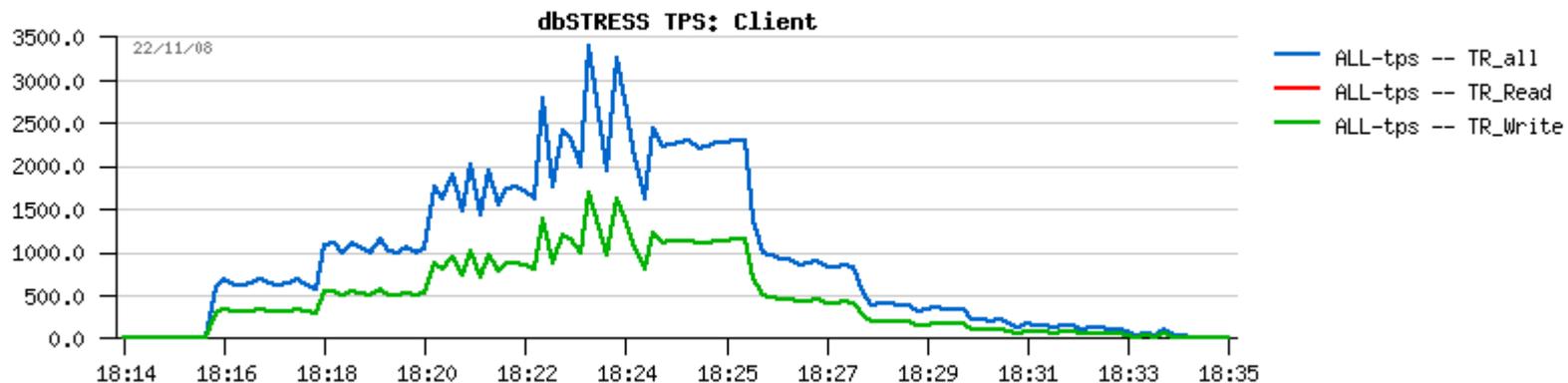
- Suggested settings:
  - > table\_cache = 8000 (too small by default (avoid LOCK\_open))
  - > innodb\_buffer\_pool\_size = 16000M (as big as possible)
  - > innodb\_thread\_concurrency = 0
  - > + MT malloc is the must!
- Google patches for MySQL
- Percona patched MySQL version (+ support)

# Performance @dbSTRESS in 2008

- “Stress” Test on all versions:
  - > 5.0.37, 5.0.67, Percona-5.0.67, 5.1.29, 6.0.7

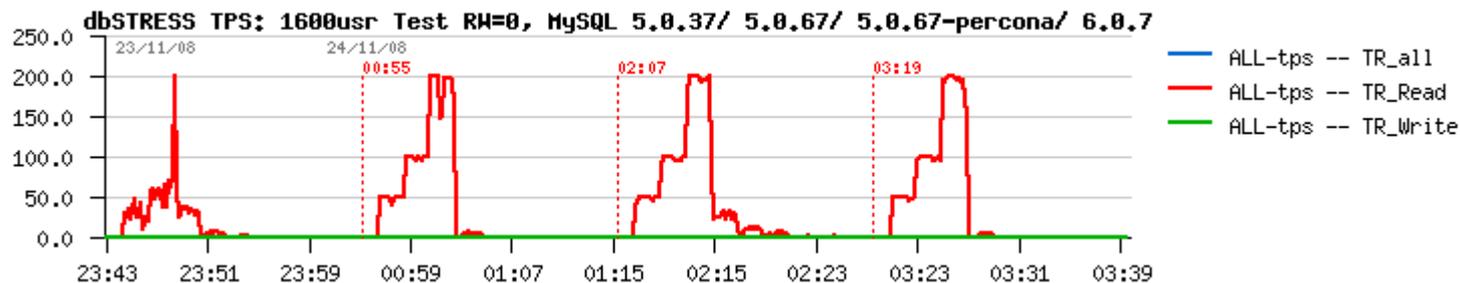


> 5.1.29



# Performance @dbSTRESS in 2008

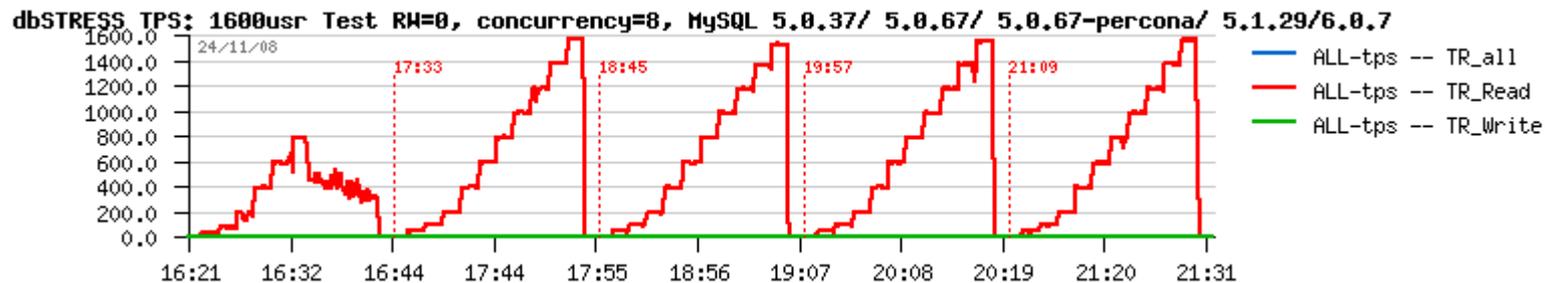
- “1600 users” (1 sec think time) Test on all versions  
 > 5.0.37, 5.0.67, Percona-5.0.67, 6.0.7



What is going wrong???

# Performance @dbSTRESS in 2008

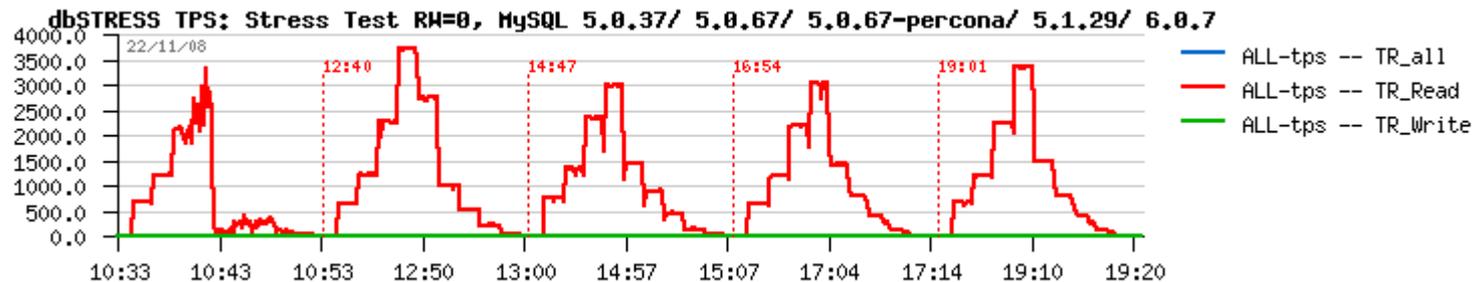
- “1600 users” Test on all versions:
  - > 5.0.37, 5.0.67, Percona-5.0.67, 5.1.29, 6.0.7
  - > innodb\_thread\_concurrency = 8



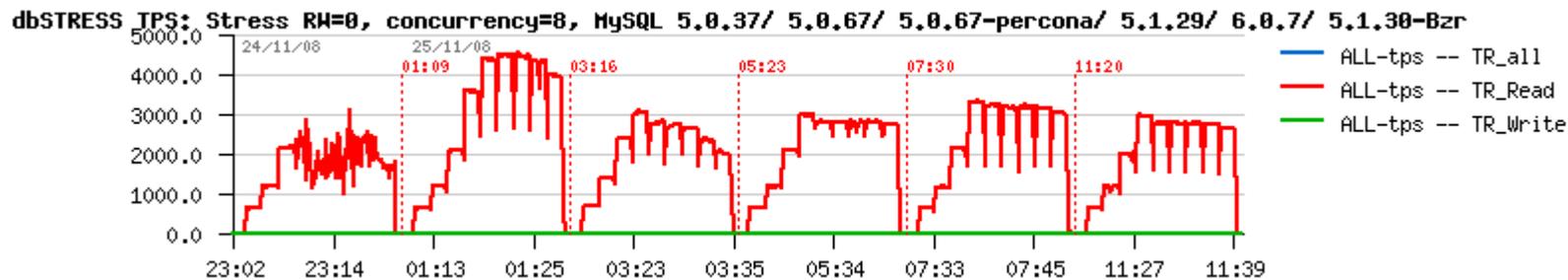
- > Much better, no? ;-)
- > And what about the “Full Stress” test now?..

# Performance @dbSTRESS in 2008

- “Full Stress” on: 5.0.37, 5.0.67, Per-5.0.67, 5.1.29, 6.0.7
  - > With innodb\_thread\_concurrency = 0 :

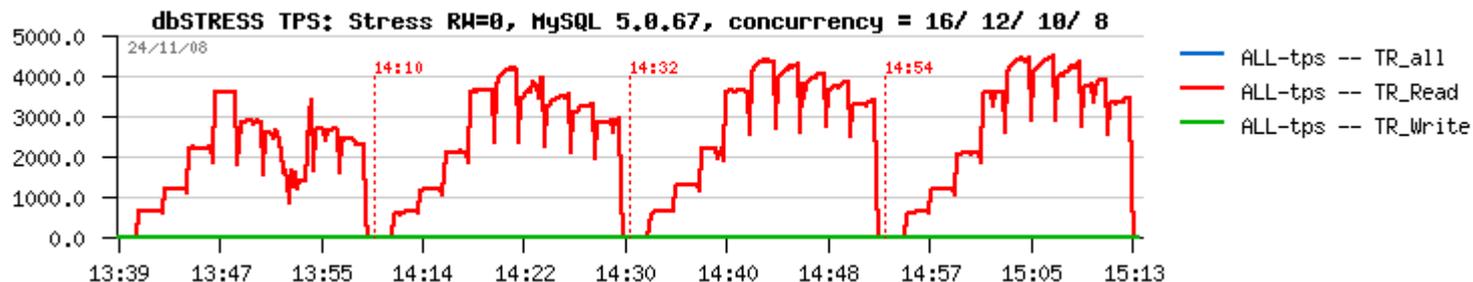


- > With innodb\_thread\_concurrency = 8 :



# Optimal thread concurrency

- “Stress” Test on MySQL 5.0.67
  - > innodb\_thread\_concurrency = 16 / 12 / 10 / 8



# InnoDB concurrency internals

- Thread queue
- FIFO order
- Entry tickets
  - > Activated thread (session) receives N entry tickets
  - > Tickets decreased on each row method call
  - > No more tickets:  
=> GOTO the Thread queue



# Concurrency management

## High locking!!

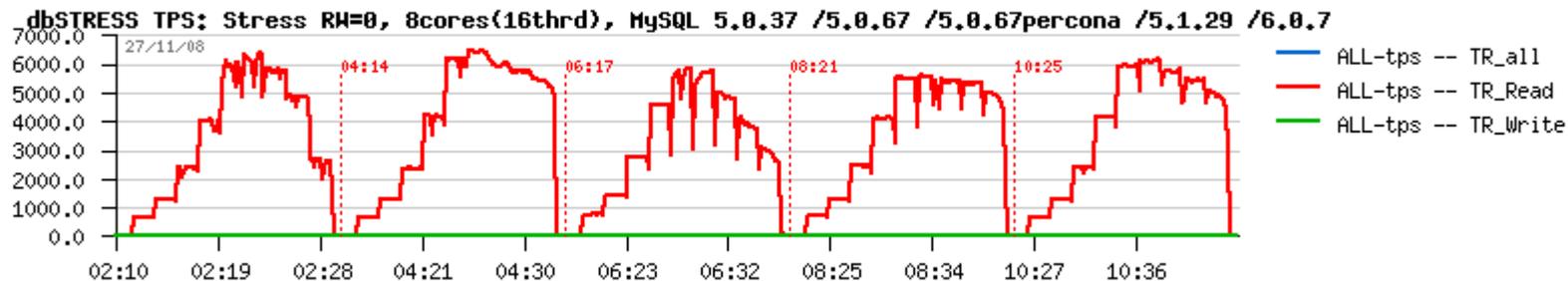
Count	nsec	Lock	Caller
225554	299695	mysql`srv_conc_mutex	mysql`srv_conc_enter_innodb+0x34
203038	300416	mysql`srv_conc_mutex	mysql`srv_conc_force_exit_innodb+0x38
32597	326522	mysql`srv_conc_mutex	mysql`srv_conc_enter_innodb+0x34
21289	319810	mysql`srv_conc_mutex	mysql`srv_conc_force_exit_innodb+0x38
8243	773016	mysql`srv_conc_mutex	mysql`srv_conc_enter_innodb+0x100
10829	357416	mysql`srv_conc_mutex	mysql`srv_conc_enter_innodb+0x34
11598	330907	mysql`srv_conc_mutex	mysql`srv_conc_enter_innodb+0x34
10641	323978	mysql`srv_conc_mutex	mysql`srv_conc_force_exit_innodb+0x38
9832	306689	mysql`srv_conc_mutex	mysql`srv_conc_force_exit_innodb+0x38
18309	161278	0x1006da540	mysql`os_mutex_enter+0x4
...			

# Concurrency tuning

- High locks => high system time on all CPU!
  - > What if we reduce the number of attributed CPU?..

# Concurrency tuning

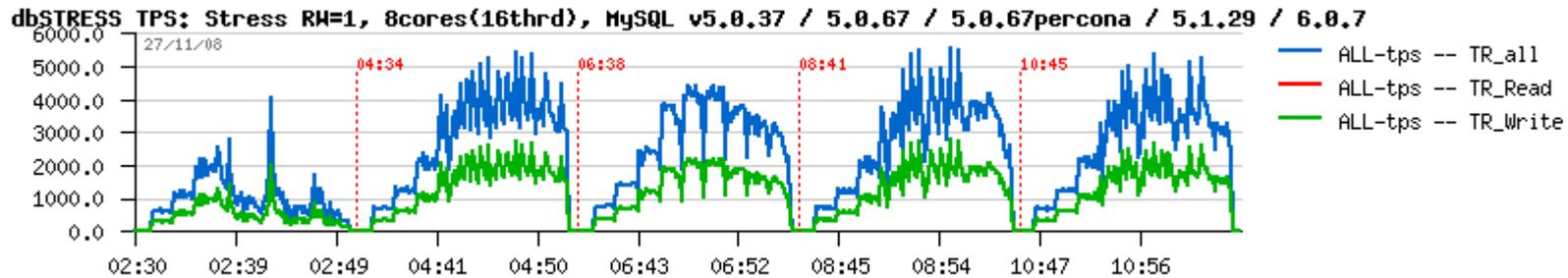
- High locks => high system time on all CPU!
  - > What if we reduce the number of attributed CPU?..
  - > CPU => 8 cores
  - > innodb\_thread\_concurrency = 16



- > 6500(!) TPS vs 4500 before

# Read+Write & concurrency

- “Stress” Test RW=1
  - > CPU: 8 cores
  - > innodb\_thread\_concurrency = 16



- > NOTE: still a performance drop on 256 users...

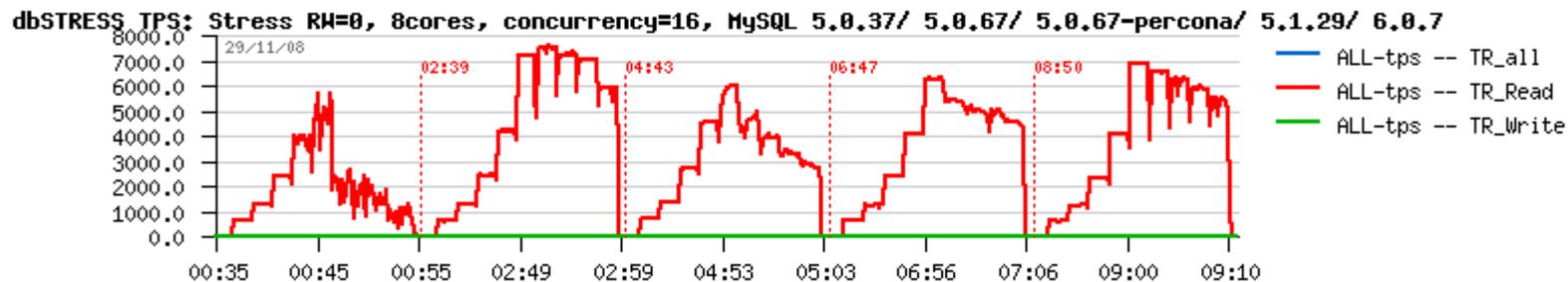
# Optimal CPU / concurrency ratio

- Focus on 256 users
  - > SPARC64 CPU has 2 hardware threads per core

CPU Cores /Threads	InnoDB Concurrency	CPU Usr%	CPU Sys%	TPS
8 /8	4	40%	12%	3,000
8 /8	8	66%	17%	4,500
8 /8	12	77%	20%	5,100
8 /8	16	78%	21%	5,000
8 /16	16	66%	23%	6,000 (!)
8 /16	20	67%	26%	5,300
8 /16	24	66%	28%	4,700
16 /16	16	56%	20%	4,300

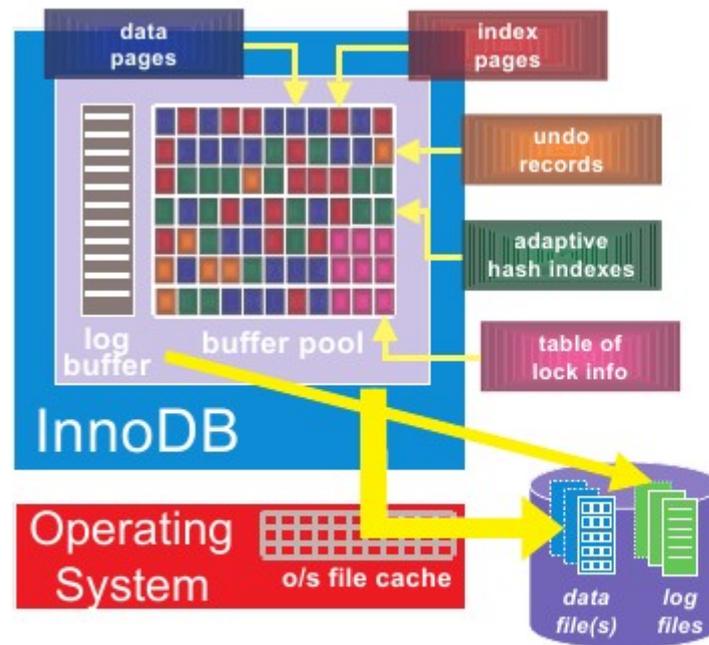
# Final results @2008

- “Stress” Test Read-Only
  - > CPU: 8 cores (16 hardware threads)
  - > innodb\_thread\_concurrency = 16
- 7500 TPS!



# InnoDB Performance Issues

- Locks...
- I/O capacity
- Checkpoint
- Google patches
- Percona
- Sun
- Oracle
- ...



- When data & index pages are read, they are cached for re-use, and are replaced when least-recently-used
- InnoDB on-the-fly creates adaptive hash indexes depending on query pattern (more on this later)
- On updates, row-level locking information is maintained in an efficient bit-map
- Undo info, used to reverse a transaction's changes, is cached in memory, later written to sys tbspace
- Compact representation of changes is kept in log buffer. On commit, log records are written to log file (but no buffer pool pages need to be written)
- Eventually, data, index and undo pages are flushed to data files

INNOBASE

# InnoDB Design: Memory

- Default malloc() is usually not MT-oriented
- Concurrent memory allocation freeze threads
- Important part of MySQL is written on C++
  - > Creation of any variable calls malloc() automatically in C++ !
- Use MT-oriented solutions:
  - > HOARD (was the first and most fast before)
  - > Libraries libmtmalloc / libumem @Solaris
  - > Library libtcmalloc @Linux

# InnoDB Design: Locks

- Initially – exclusive mutexes (home made)
- Google patch => Read/Write locks
- Recently => GCC atomics (as well SunStudio too)
- Still many hot locks..
  - > Hottest: LOCK\_open
  - > Then: kernel\_mutex
- Still many hot areas:
  - > Sync array
  - > Various buffers / lists

# InnoDB Design: I/O level

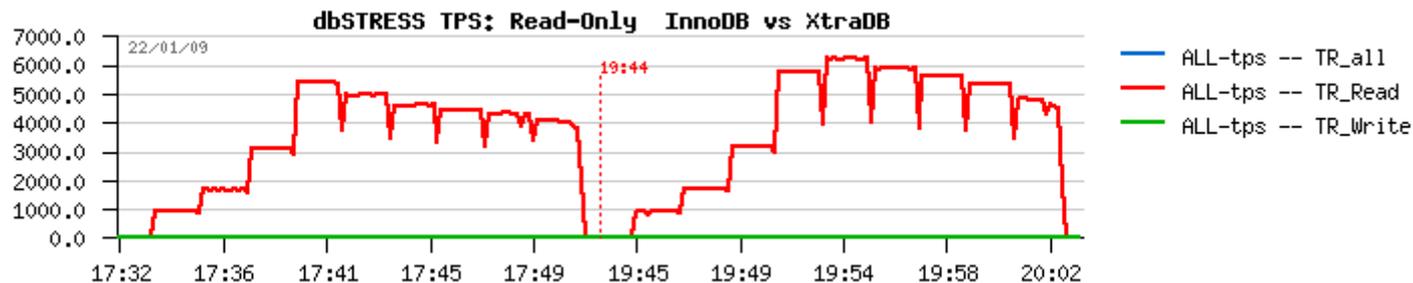
- Initially:
  - > limited to 100 writes/sec...
  - > 1 Read-Ahead thread
  - > 1 Page-Write thread
- Available solutions (patches):
  - > I/O capacity option!
  - > Multiple Read-Ahead threads
  - > Multiple Page-Write threads
  - > Adaptive Checkpoint

# December 2008: XtraDB !

- Announced in Dec.2008
- Based on InnoDB plugin + patches (Percona.com)
- Adaptive checkpoint
- I/O capacity
- I/O threads
- Light locks implementation
- More details: [mysqlperformanceblog.com](http://mysqlperformanceblog.com)

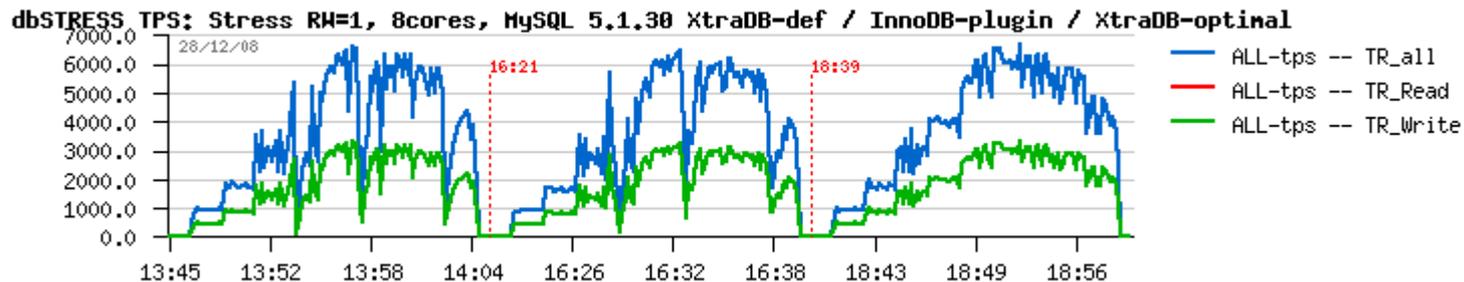
# XtraDB @dbSTRESS (in 2008)

- “Full Stress” Read-Only
  - > MySQL 5.1.30 + InnoDB-plugin / XtraDB
  - > innodb\_thread\_concurrency = 16
  - > 8 cores



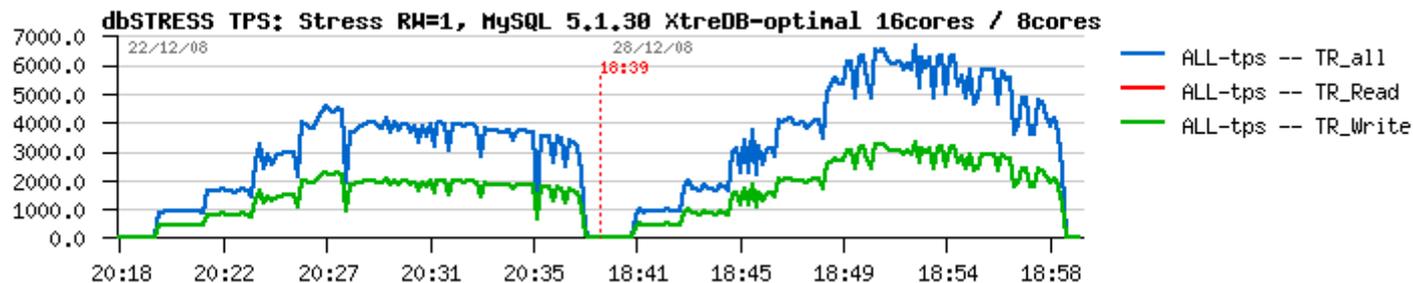
# XtraDB @dbSTRESS (in 2008)

- “Full Stress” Read+Write
  - > MySQL 5.1.30 + XtraDB-def / InnoDB-plugin / XtraDB-optimal
  - > innodb\_thread\_concurrency = 16
  - > 8 cores



# XtraDB @dbSTRESS (in 2008)

- “Full Stress” Read+Write
  - > MySQL 5.1.30 + XtraDB-optimal
  - > innodb\_thread\_concurrency = 16
  - > 16 cores vs 8 cores

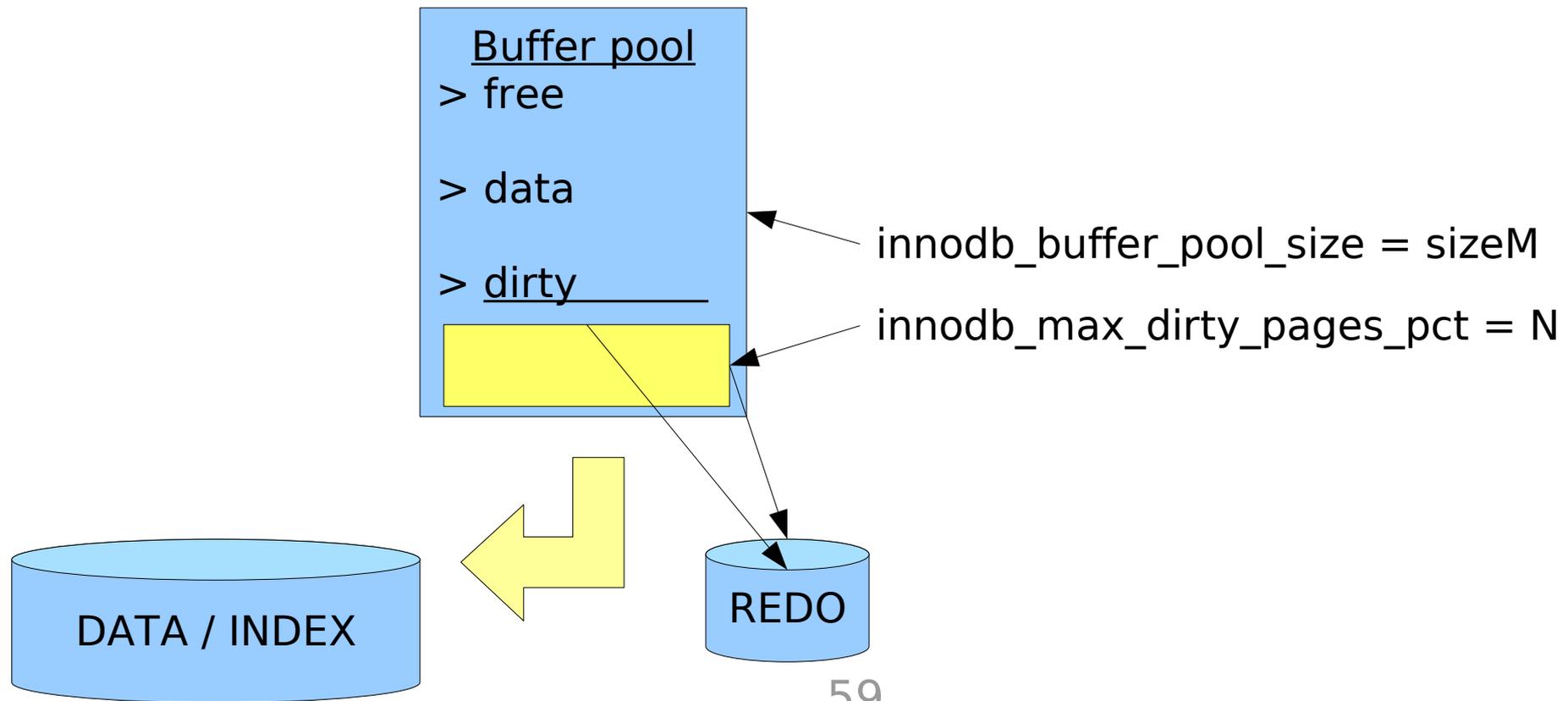


First Bummer...

**BUM!... :-)**

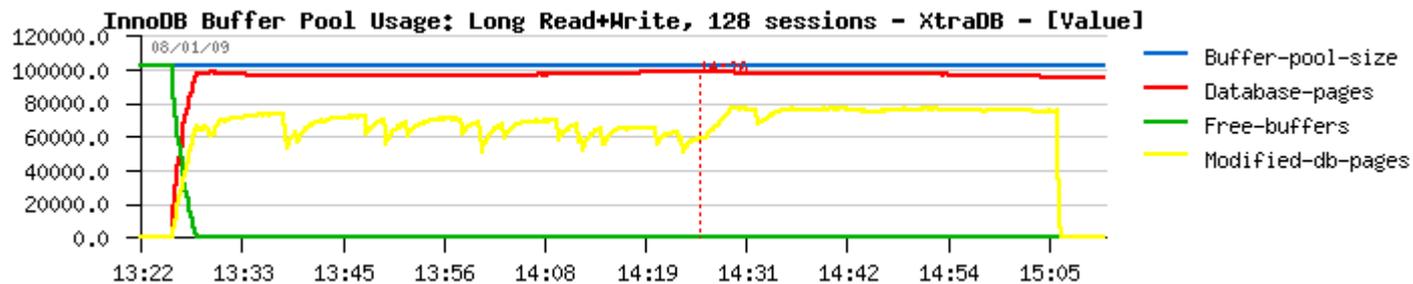
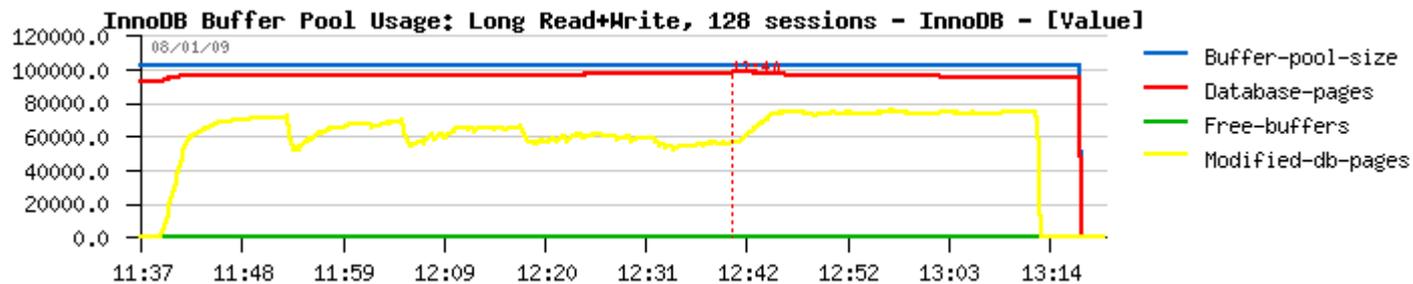
# InnoDB internals: Buffer pool

- Small explanation before to go further..
  - > SQL> show innodb status \G



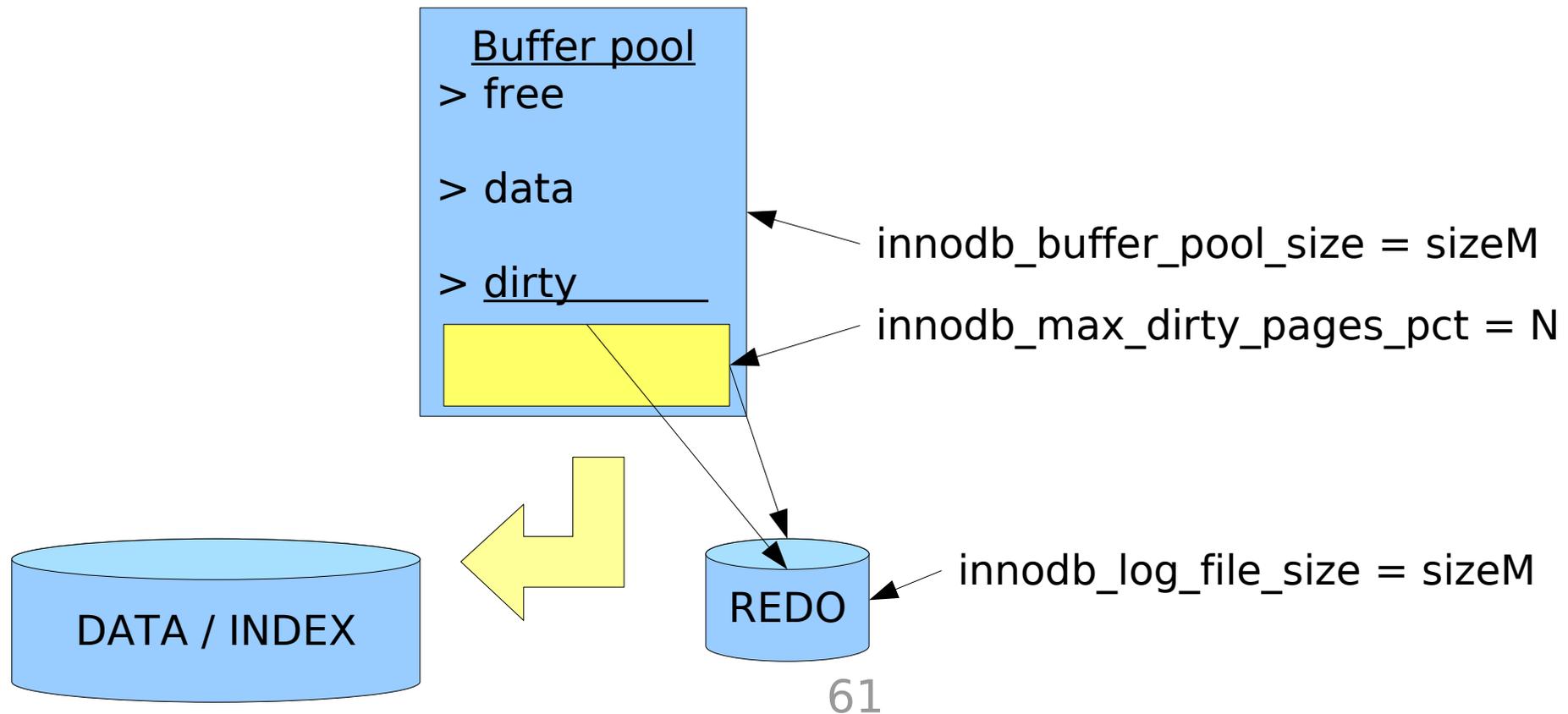
# Long Read+Write Test (Jan.2009)

- Long Read+Write Test: 128 sessions during 1 hour
  - > What's the state of the buffer pool?..
  - > **NOTE:** innodb\_max\_dirty\_pages\_pct=15



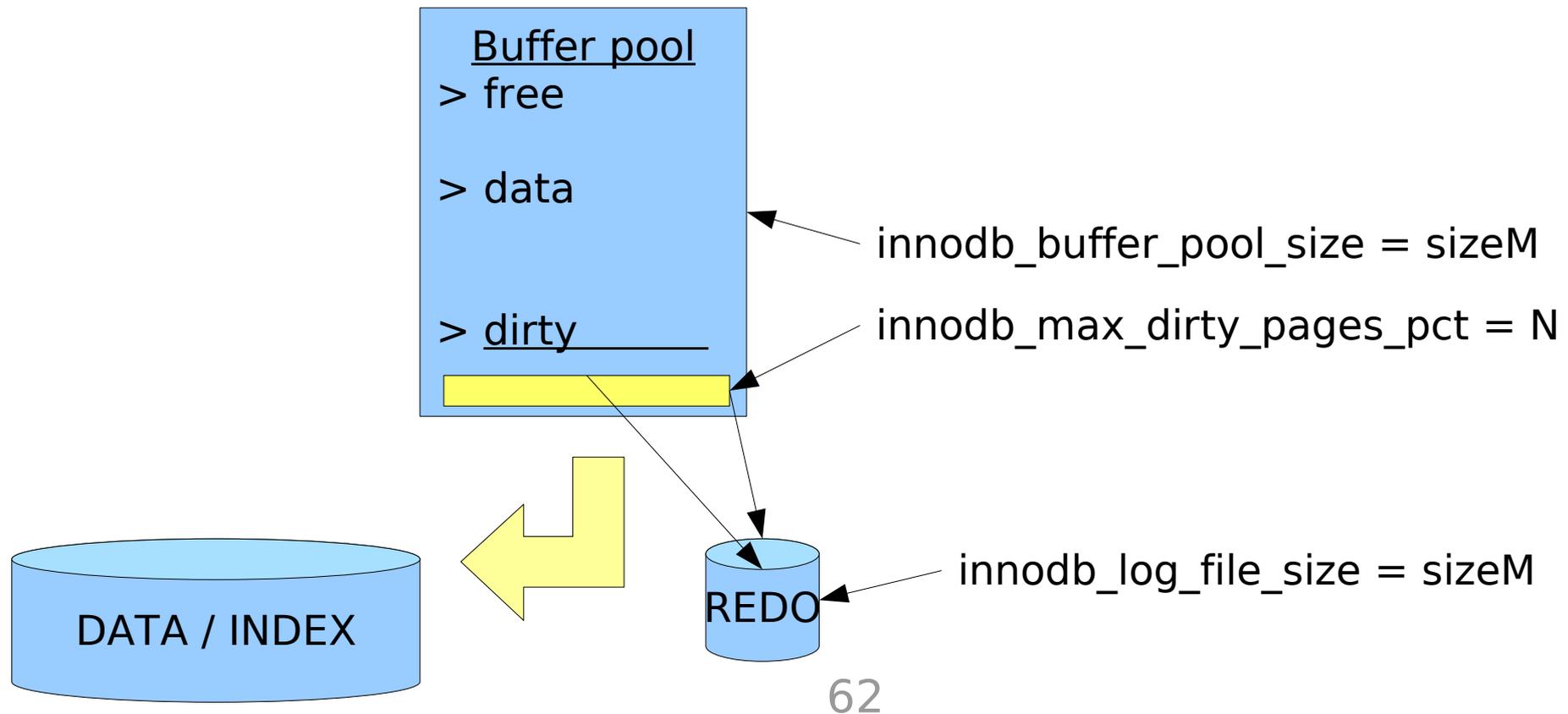
# Dirty pages & Redo log

- Seems dirty pages limit is completely ignored here...
  - > Workaround: reduce redo log size!



# Dirty pages & Redo log

- Seems dirty pages limit is completely ignored here...
  - > Workaround: reduce the size of redo log !
  - > innodb\_log\_file\_size: 500M => 128M

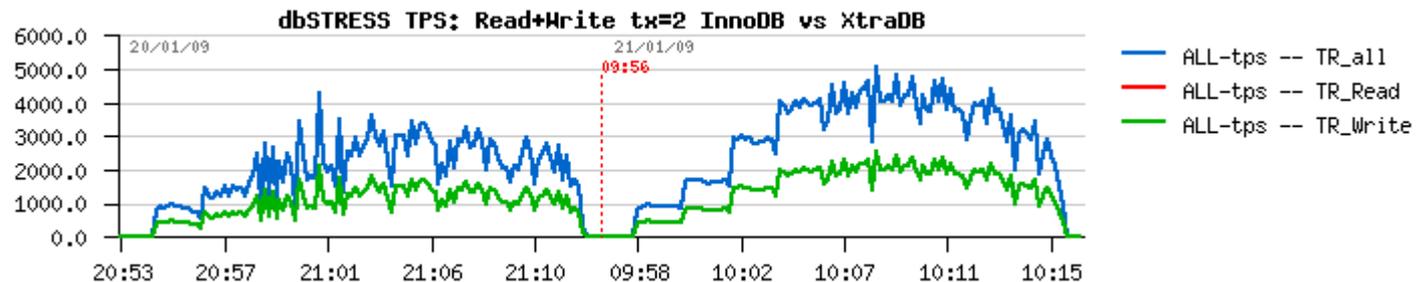


# InnoDB / XtraDB with 128M redo

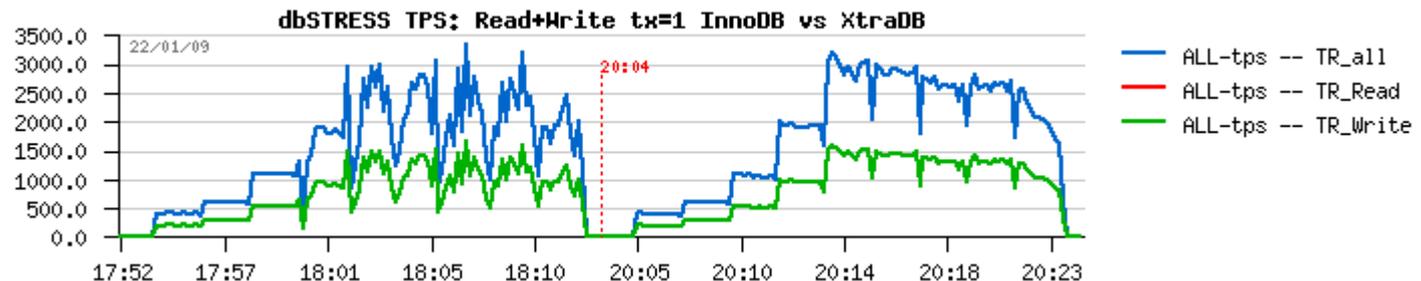
- “Stress” Read+Write: InnoDB vs XtraDB

- > innodb\_log\_file\_size = 128M

- > innodb\_flush\_log\_at\_trx\_commit = 2

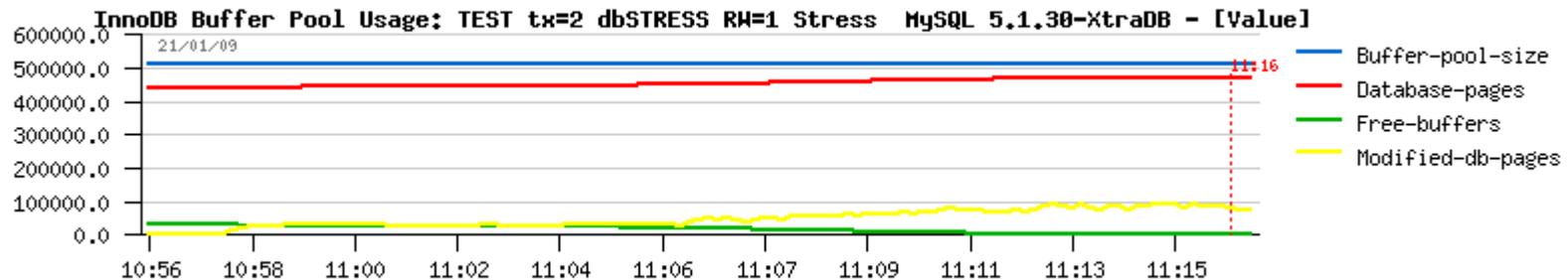


- > innodb\_flush\_log\_at\_trx\_commit = 1



# Buffer @XtraDB with 128M redo

- “Stress” Read+Write: XtraDB
  - > innodb\_log\_file\_size = 128M
  - > innodb\_flush\_log\_at\_trx\_commit = 2



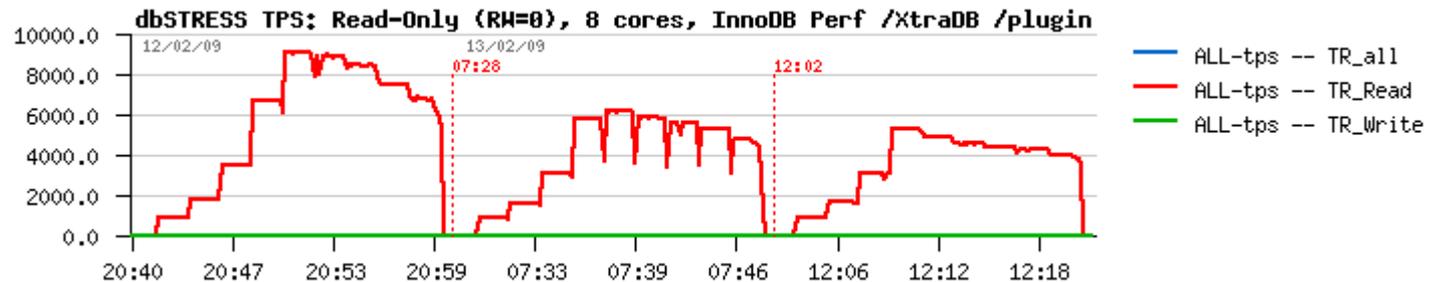
- NOTE: since then during all tests 128M redo was used!

# MySQL Performance version (Feb)

- Internal “probe” version => performance-oriented
- Improvements:
  - > Sun patches applied
  - > Google patches adopted
  - > Atomic operations (GCC / SunStudio)
  - > I/O capacity
  - > I/O read & write threads
  - > Timer based concurrency model !!!
    - > No more thread queue
    - > Very light locking
    - > Threads are sleeping few ms on wait
- Working version expected for April's MySQL UC2009

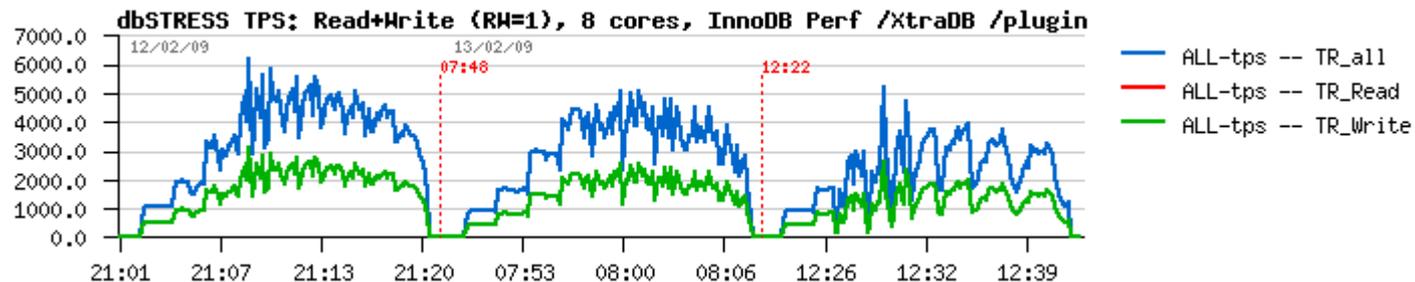
# Performance version @dbSTRESS

- “Stress” Read-Only
  - > CPU: 8 cores
  - > Buffer pool: 12G
  - > innodb\_thread\_concurrency = 16
  - > “Perf” vs XtraDB vs InnoDB plugin:



# Performance version @dbSTRESS

- “Stress” Read+Write
  - > CPU: 8 cores
  - > Buffer pool: 12G
  - > innodb\_thread\_concurrency = 16
  - > “Perf” vs XtraDB vs InnoDB plugin:

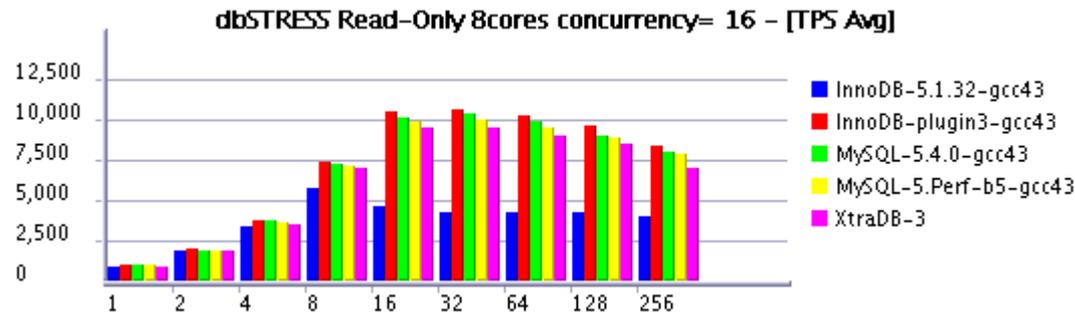


# InnoDB plugin-1.0.3 (Mar.2009)

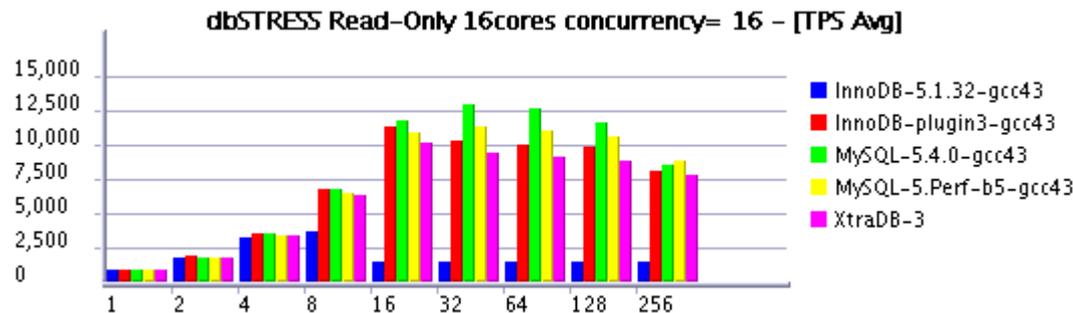
- Performance improvements!
  - > Google RW-locks adoption => scalability!
  - > Use system malloc()
  - > Dynamic On/Off insert buffering
  - > Dynamic On/Off hash indexes
  - > Bug fixes
  - > etc..

# MySQL 5.4 (Apr.2009)

- Performance + Scalability improvement!
  - > Read-Only on 8 cores:

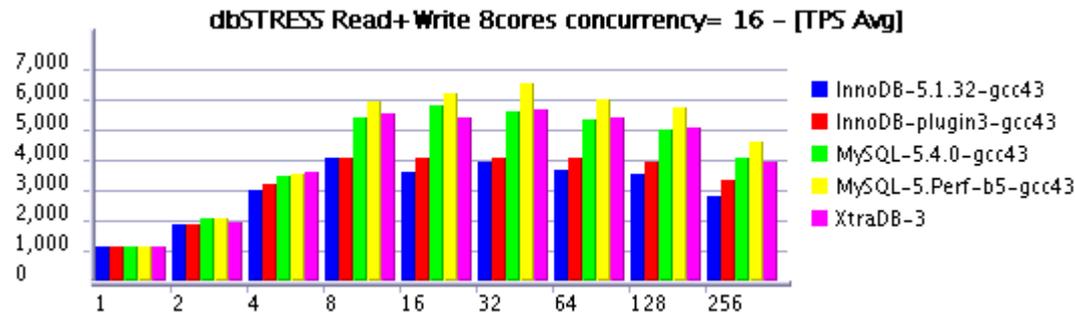


- > Read-Only on 16 cores:

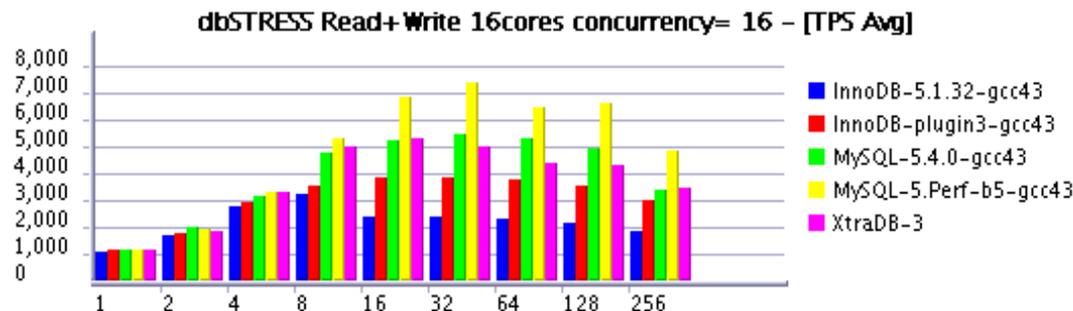


# MySQL 5.4 (Apr.2009)

- Performance + Scalability improvement!
  - > Read+Write on 8 cores:



- > Read+Write on 16 cores:



# MySQL 5.4 (Apr.2009)

- Performance + Scalability improvement!
- On my tests:
  - > For the first time moving from 8 to 16 cores improves performance !!! :-)
  - > For the first time MySQL out-performing PostgreSQL !!!
- Next probe “Performance build” runs even faster!

# InnoDB plugin @Sep.2009

- Plugin-1.0.4 (Aug.2009)
  - > I/O capacity
  - > I/O read / write threads
  - > Adaptive flushing
  - > Read Ahead control configuration option
  - > Group Commit fixed!!!
  - > Bug fixes..

# XtraDB @Sep.2009

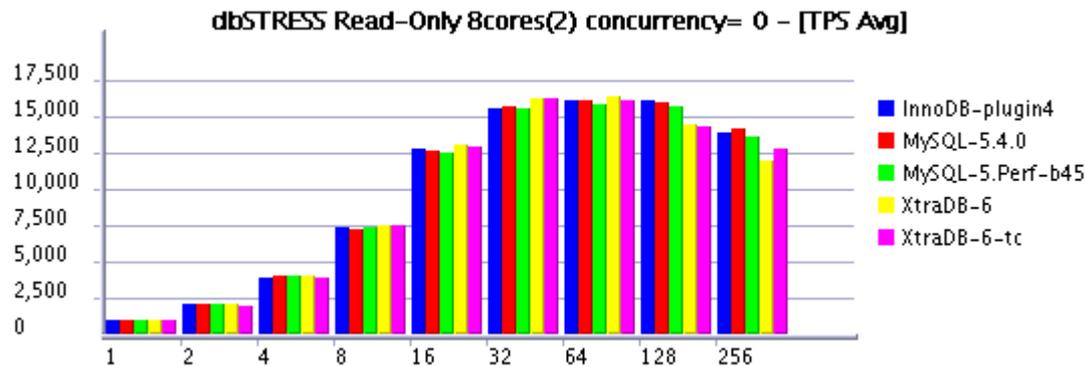
- Adopted most of improvements from MySQL 5.4
- Aligned with InnoDB plugin & benefits all its features
- Some XtraDB unique features:
  - > Fast recovery (x10 speed-up) !
  - > Import / Export of InnoDB tables !
  - > Fixes from Google v4 patch
  - > Limit for data dictionary size
- Xtrabackup => hot backups ready !

# Check your glasses first! :-)

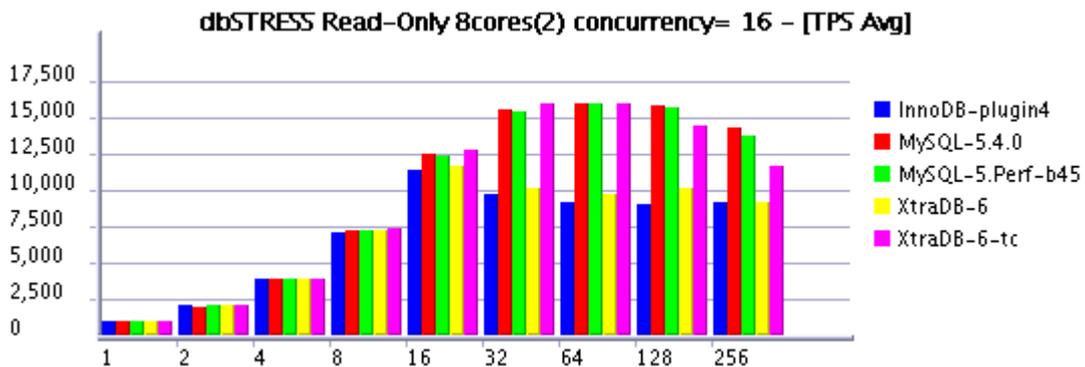


# dbSTRESS results @Sep.2009

- Read-Only on 8 cores
  - > innodb\_thread\_concurrency = 0

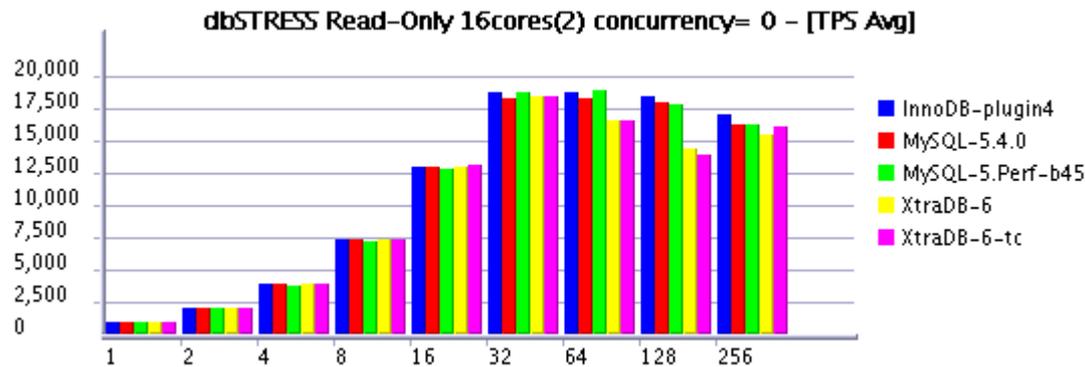


- > innodb\_thread\_concurrency = 16

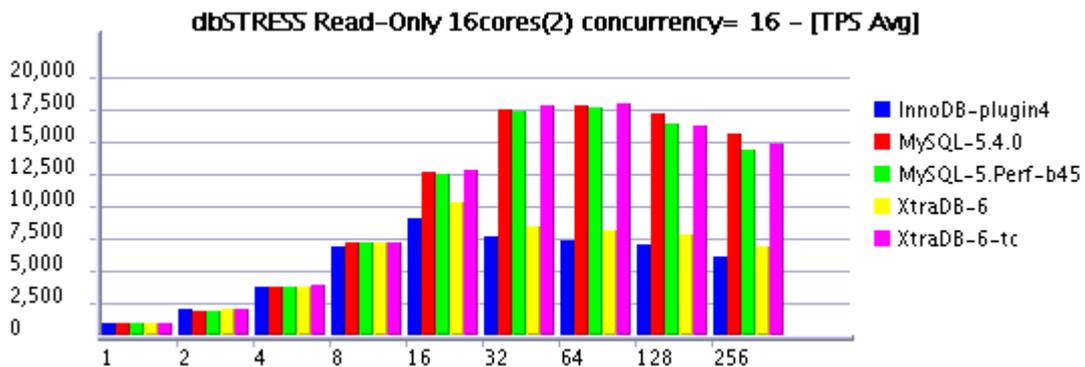


# dbSTRESS results @Sep.2009

- Read-Only on 16 cores
  - > innodb\_thread\_concurrency = 0

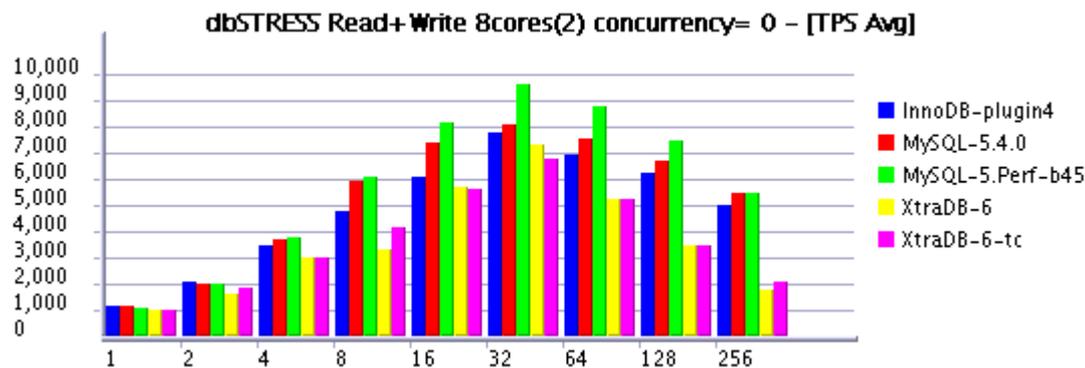


- > innodb\_thread\_concurrency = 16

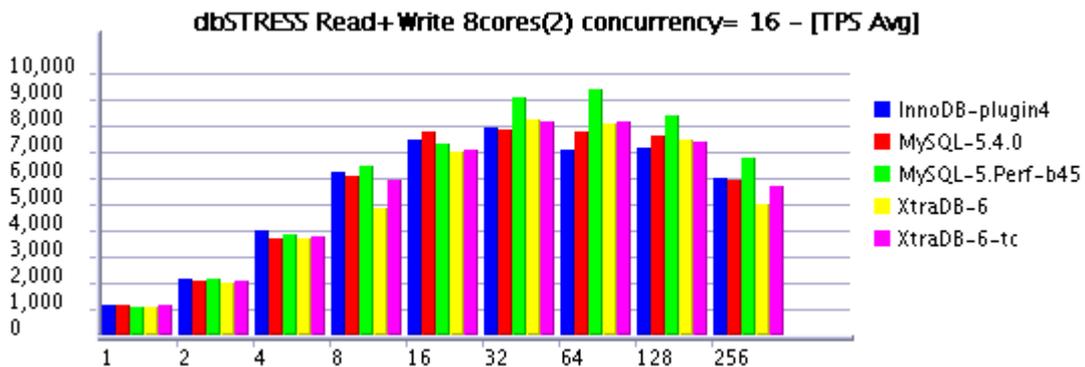


# dbSTRESS results @Sep.2009

- Read+Write on 8 cores
  - > innodb\_thread\_concurrency = 0

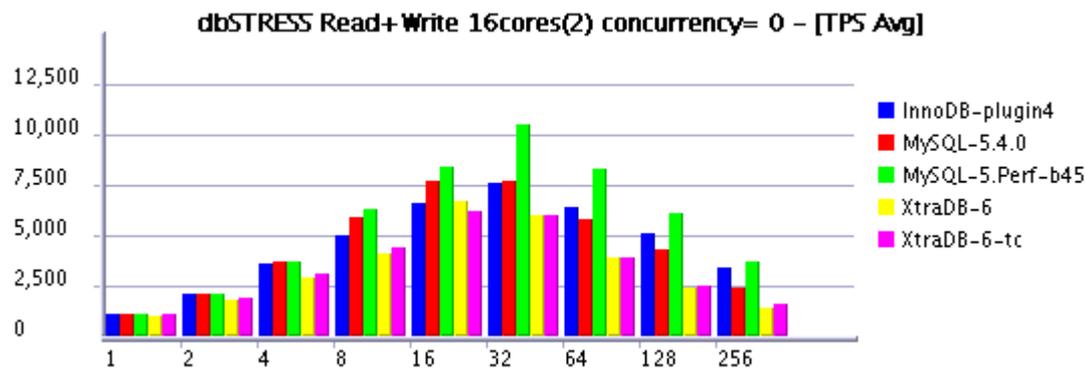


- > innodb\_thread\_concurrency = 16

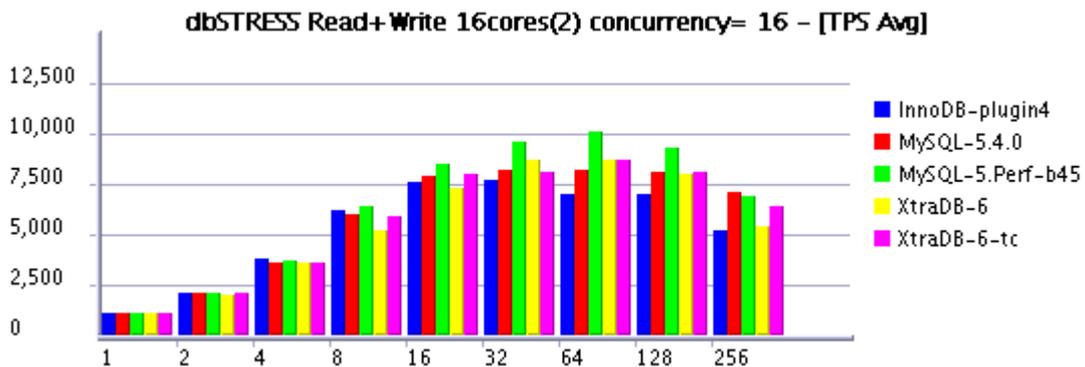


# dbSTRESS results @Sep.2009

- Read+Write on 16 cores
  - > innodb\_thread\_concurrency = 0

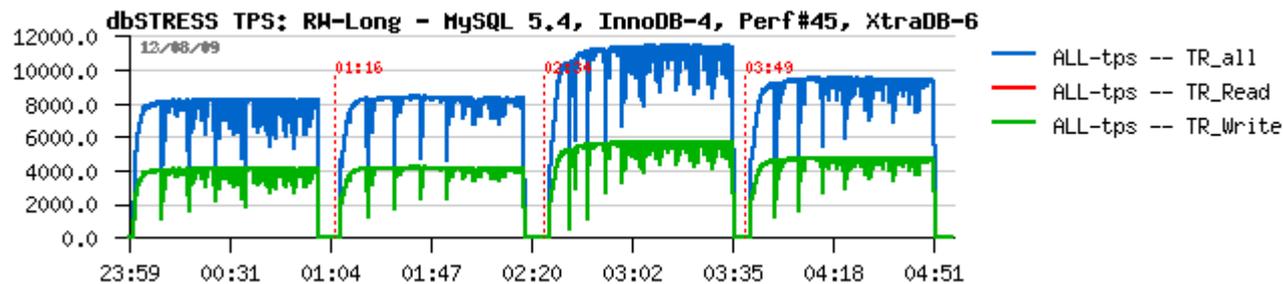


- > innodb\_thread\_concurrency = 16



# dbSTRESS results @Sep.2009

- Long Read+Write test on 16 cores
  - > innodb\_thread\_concurrency = 16



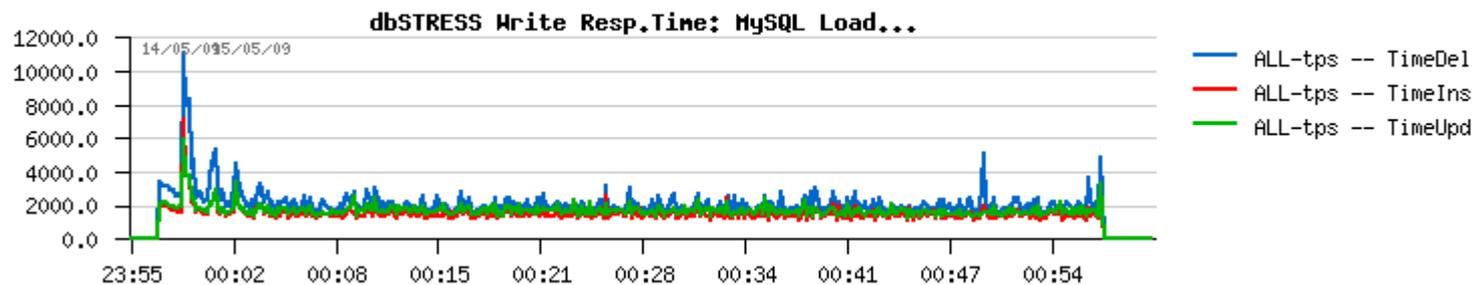
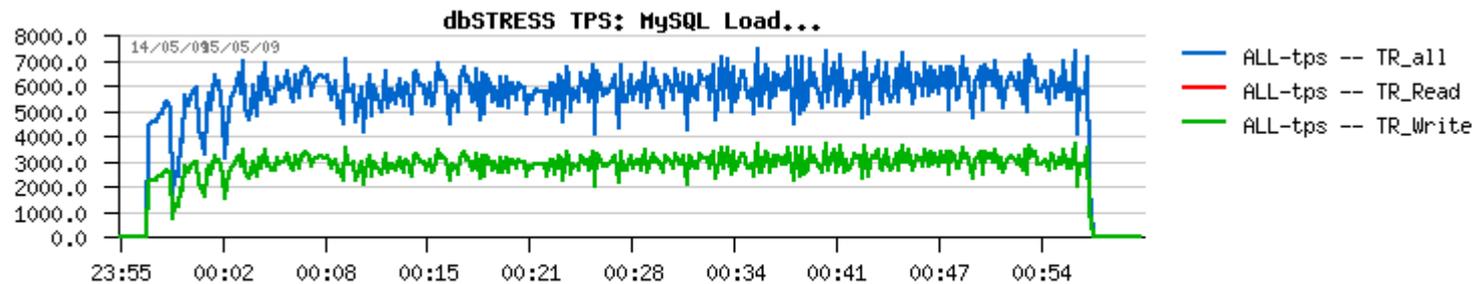
- Current hottest locks:
  - > LOCK\_open
  - > kernel\_mutex
- Work continues.. ;-)

## Work in pipe... (Apr.2009 - now)

- Redo log size impact
- Dirty pages limit
- Purge loop
- Purge lag
- Ahead flushing
- Purge thread
- etc...

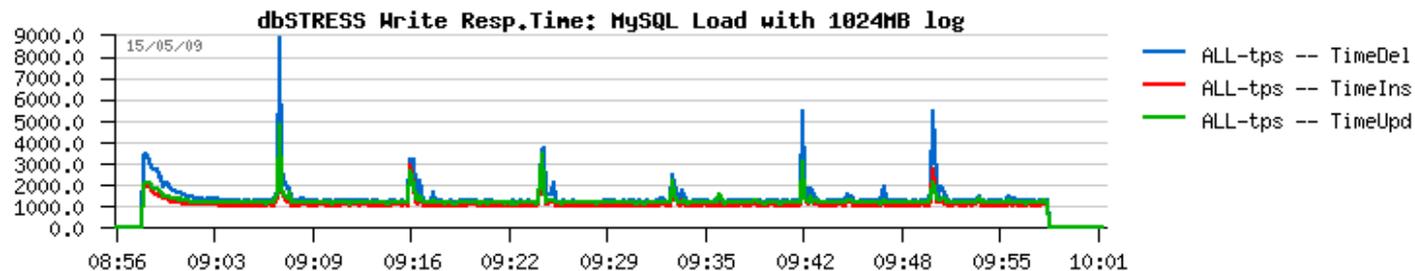
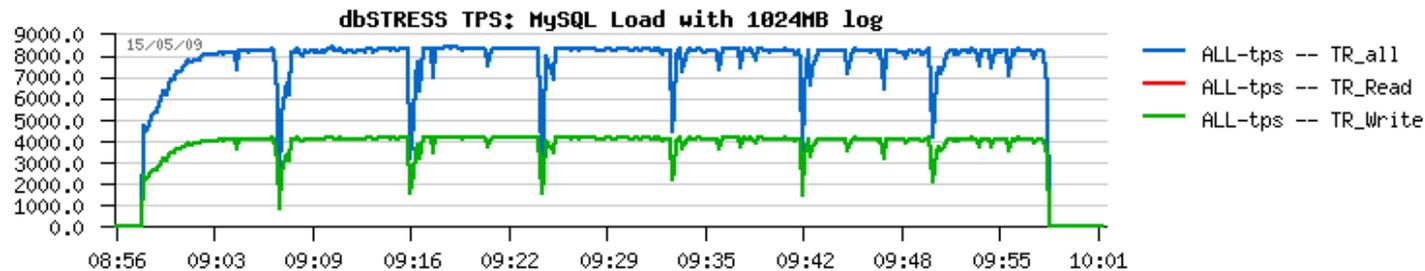
# InnoDB: Redo log size impact

- Long “Stress” Read+Write:
  - > Redo log = 128M



# InnoDB: Redo log size impact

- Long “Stress” Read+Write:
  - > Redo log = 1024M

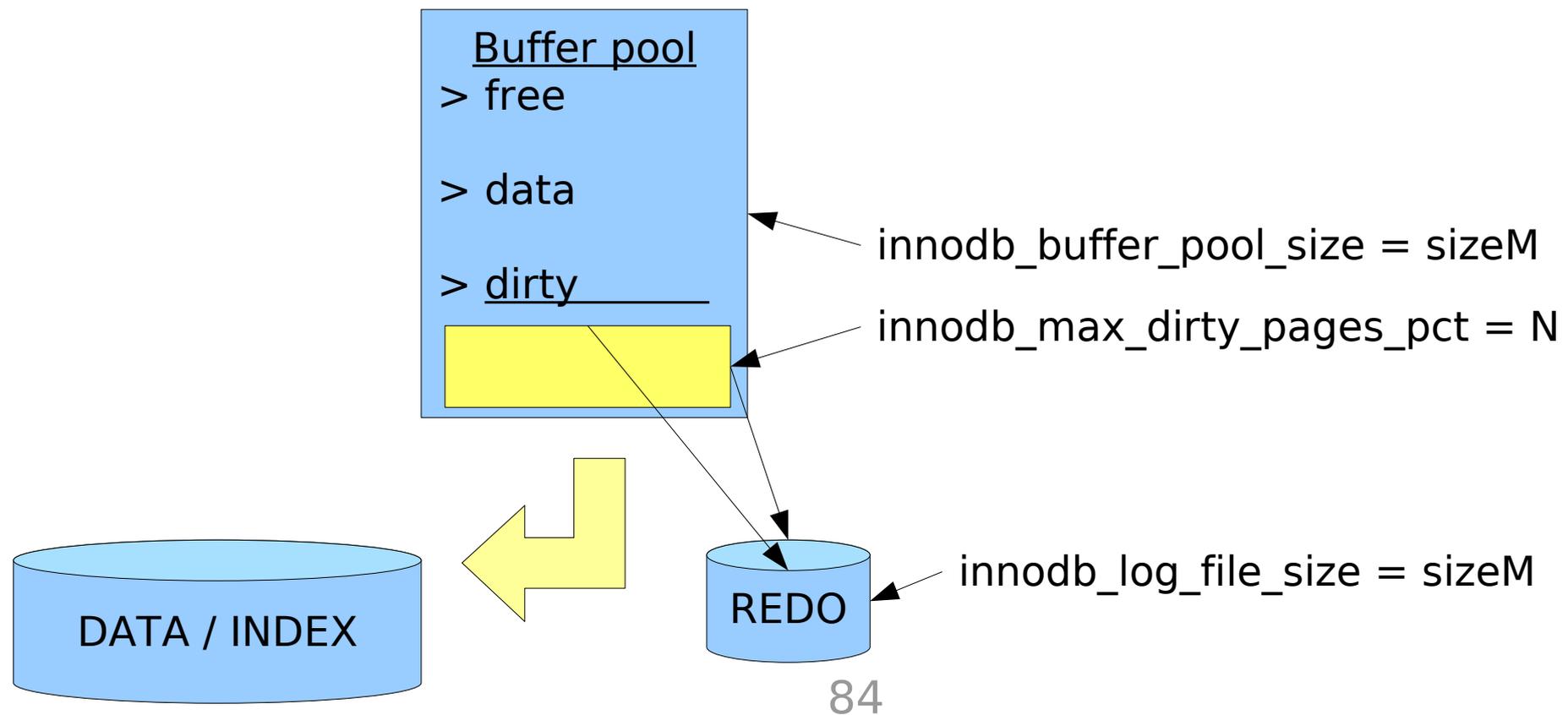


# InnoDB: Redo log size impact

- Long “Stress” Read+Write:
  - > Redo log = 1024M
  - > 30% Performance improvement!!! (vs 128M)
- But...
  - > Why such kind of “furious flushing” ?..
  - > What about dirty pages?..
  - > What about recovery time?..

# InnoDB: Dirty pages limit (cont.)

- Completely ignored on high workload
- Redo log size!
- So, why?..



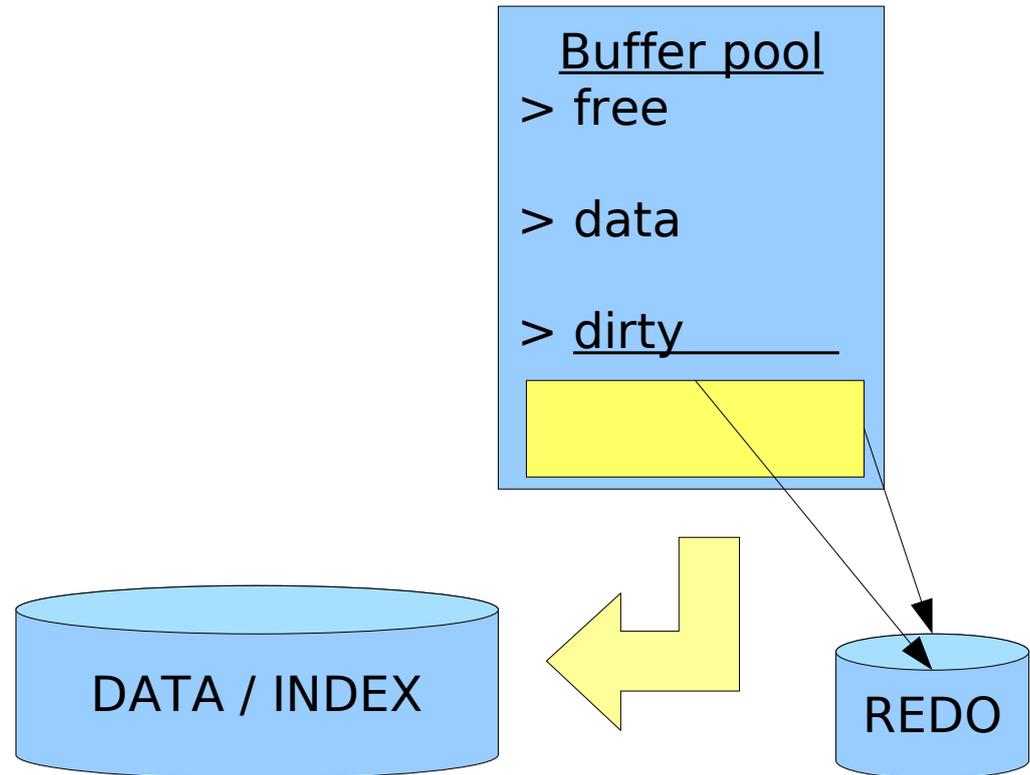
# InnoDB: Dirty pages limit (cont.)

- Analyzing Master thread code:

## Master Thread

```

loop: //Main loop
...
if( dirty pct > limit)
  flush_batch( 100% IO);
...
do {
  pages= trx_purge();
  if( 1sec passed ) flush_log();
} while (pages);
...
goto loop;
  
```



# InnoDB: Dirty pages limit (cont.)

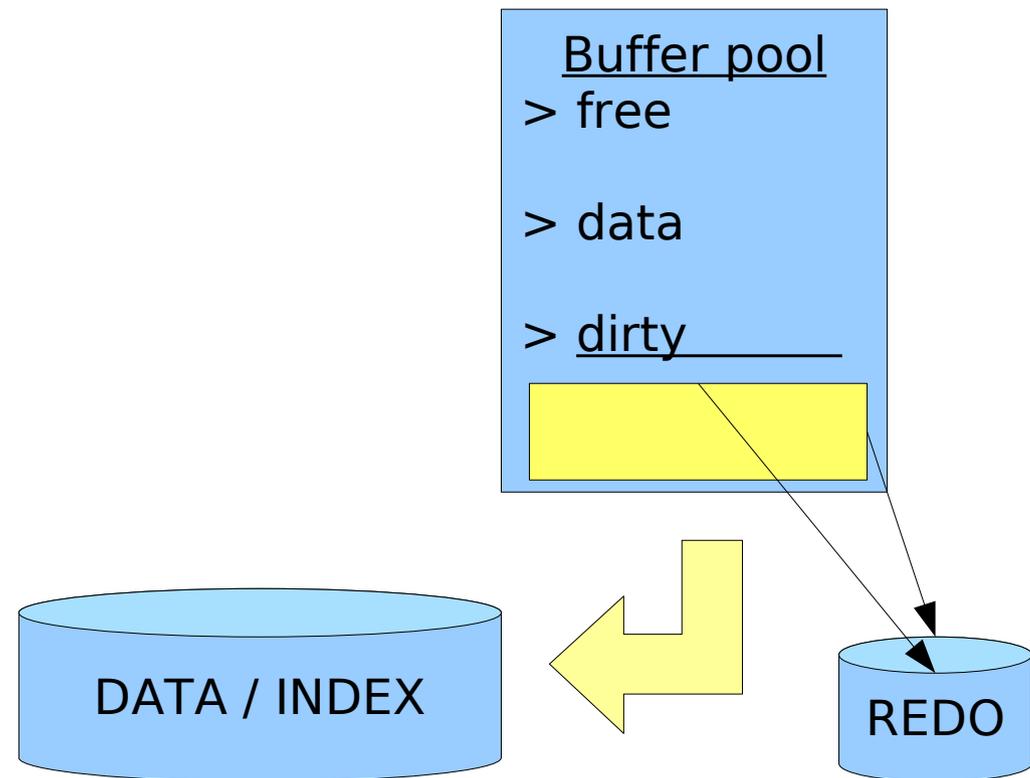
- Master thread is never leaving the purge loop!..

## Master Thread

```

loop: //Main loop
...
if( dirty pct > limit)
    flush_batch( 100% IO);
...
do {
    pages= trx_purge();
    if( 1sec passed ) flush_log();
} while (pages);
...
goto loop;

```



# InnoDB: Purge lag option

- May be used to reduce lagging between purge operations and database workload..
- How it works?..
  - > innodb\_max\_purge\_lag = N (0 means disabled (default))
  - > When History list length out-passing N => short sleep delays are added into DML queries (similar to I/O throttle on FS)
- Why lagging is dangerous?..
  - > DELETE operation is only marking the row as deleted
  - > The row is still need to be purged to be really removed and free occupied space
  - > High INSERT / DELETE activity within a 10MB table may grow it over 10GB of “dead” data (see MySQL doc)..

# InnoDB: Purge lag test

- Long “Stress” Read+Write
  - > 32 sessions
  - > 16 cores
- Testing:
  - > innodb\_max\_purge\_lag = 0
  - > innodb\_max\_purge\_lag = 10000
  - > innodb\_max\_purge\_lag = 1000000
  - > Observations: => NO DIFFERENCE!!!...
- What is going wrong?...

# InnoDB: Purge Lag code

- Problem #1:
  - > Checking if there is any old consistent view:

```
if (srv_max_purge_lag > 0 &&  
    ! UT_LIST_GET_LAST( trx_sys->view_list ) )
```

- > But why it make a problem to delay DML statement???...
- Problem #2:
  - > Min delay is 5ms
  - > Max delay is ... (very big :-)) => DANGER!!
- Fix: remove consistent view condition & max delay 50ms!

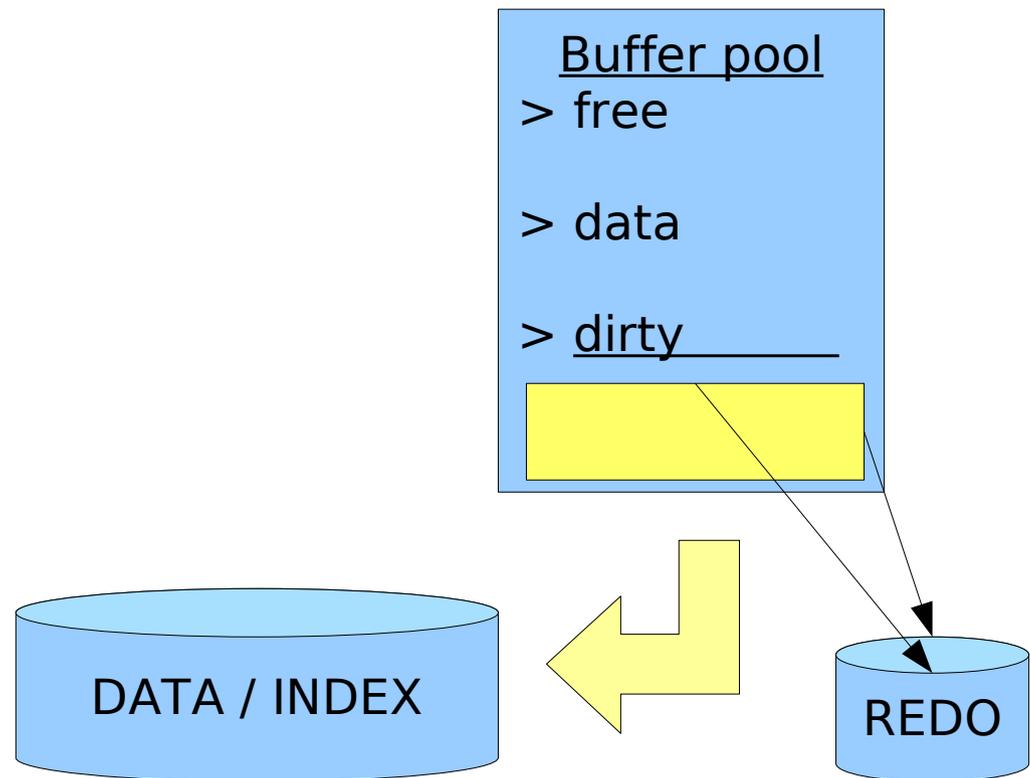
# Re-Testing with Purge lag fix

- But Master thread is still never leaving the purge loop!..

## Master Thread

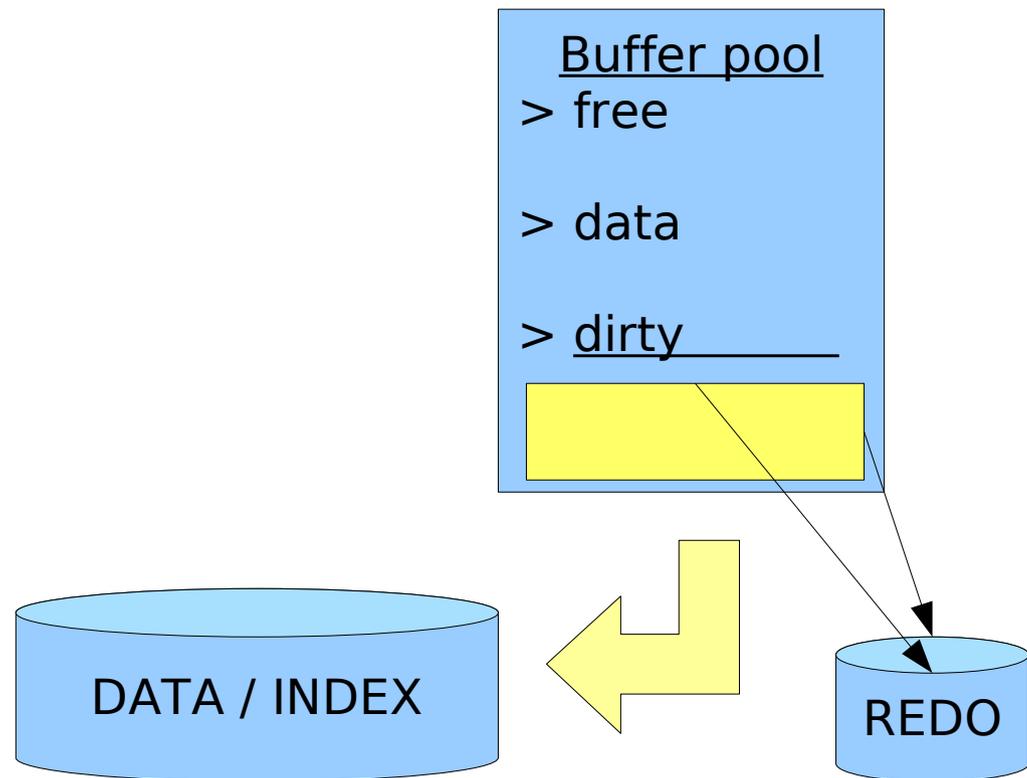
```

loop: //Main loop
...
if( dirty pct > limit)
  flush_batch( 100% IO);
...
do {
  pages= trx_purge();
  if( 1sec passed ) flush_log();
} while (pages);
...
goto loop;
  
```



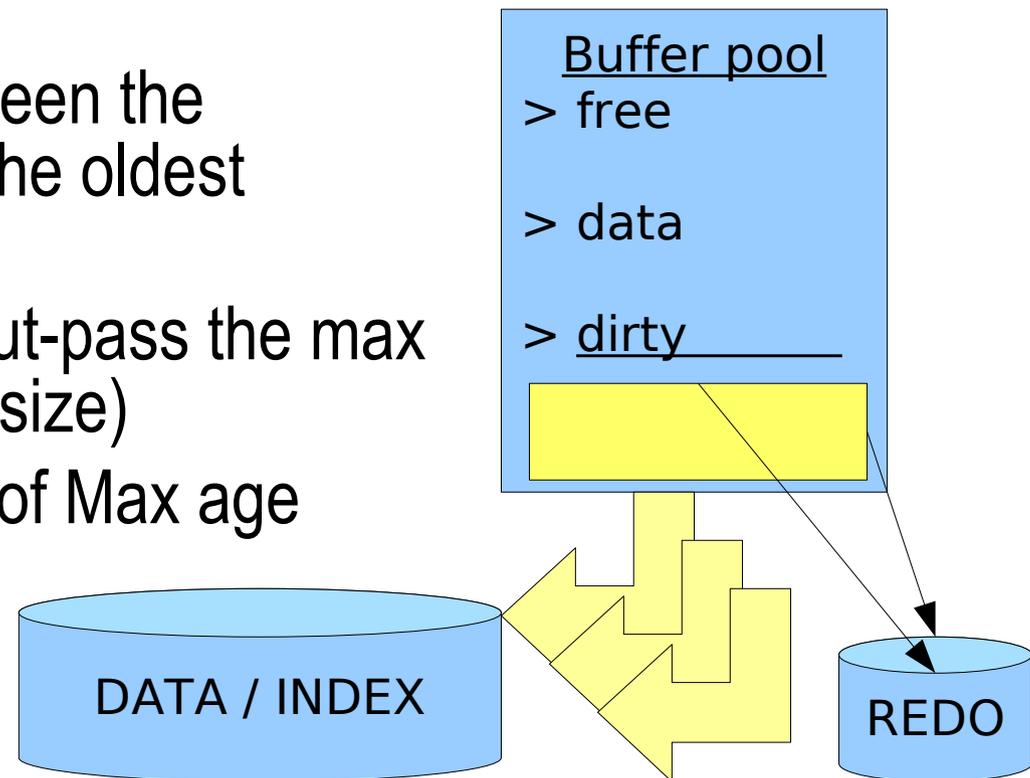
# Re-Testing with Purge lag fix

- If Master thread is still never leaving the purge loop...
  - > So, who is flushing dirty pages?..



# Re-Testing with Purge lag fix

- If Master thread is still never leaving the purge loop...
  - > So, who is flushing dirty pages?..
- Redo log constraints:
  - > Cyclic, need free space
  - > Checkpoint age: diff between the current LSN in redo and the oldest dirty page LSN
  - > Checkpoint age cannot out-pass the max checkpoint age (redo log size)
  - > If Checkpoint age  $\geq 7/8$  of Max age  $\Rightarrow$  Flush ALL dirty!



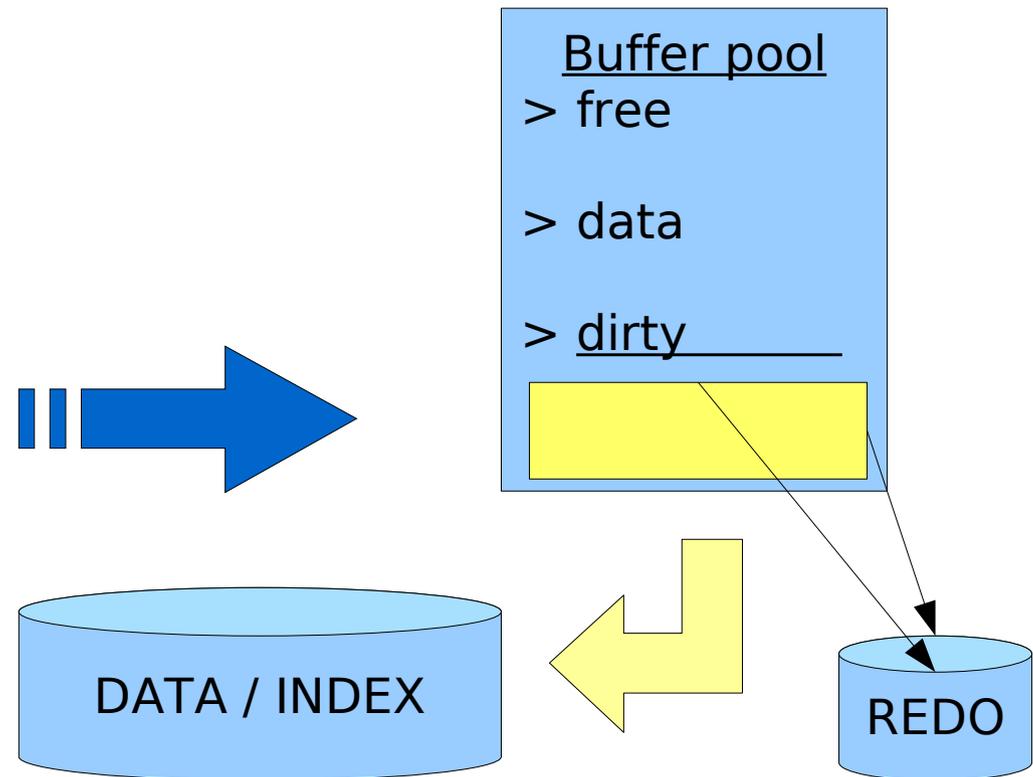
# Ahead Flushing is needed!

- Master thread should flush whatever happens!..

## Master Thread

```

loop: //Main loop
...
if( dirty pct > limit)
  flush_batch( 100% IO);
...
do {
  pages= trx_purge();
  if( 1sec passed ) flush_log();
  if( ... ) flush_batch( N% IO );
} while (pages);
...
goto loop;
  
```



# Purge loop...

- Involve flush from the purge loop is only partial solution
  - > Other maintenance tasks will be missed within a code!
  - > BTW, “adaptive” checkpoint is not working here as its code is never reached too!
- Tim Cook: but if it's so, why not to split purge processing into a separated thread?..
  - > Vince Carbone => the initial patch
  - > Dimitri => moved his changes into the patch

# Separated Purge Thread

- Having separated Purge Thread is absolutely great!..

## Master Thread

```

loop: //Main loop
...
sleep( 1 );
...
if( dirty pct > limit)
    flush_batch( 100% IO);
...
flush_log();
...
goto loop;

```

## Purge Thread

```

loop:
sleep( ... );
do { pages=
    trx_purge();
} while (pages);
goto loop;

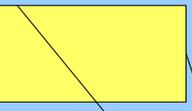
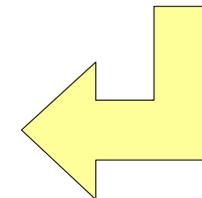
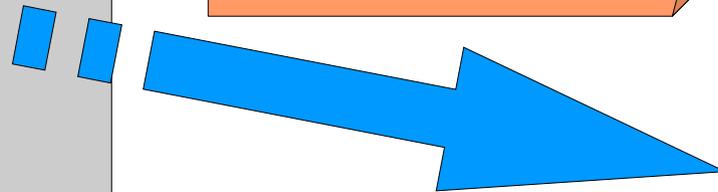
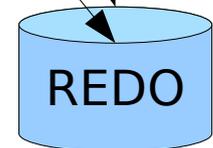
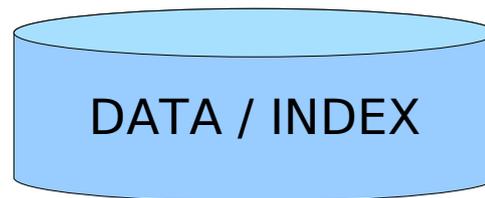
```

## Buffer pool

```

> free
> data
> dirty

```

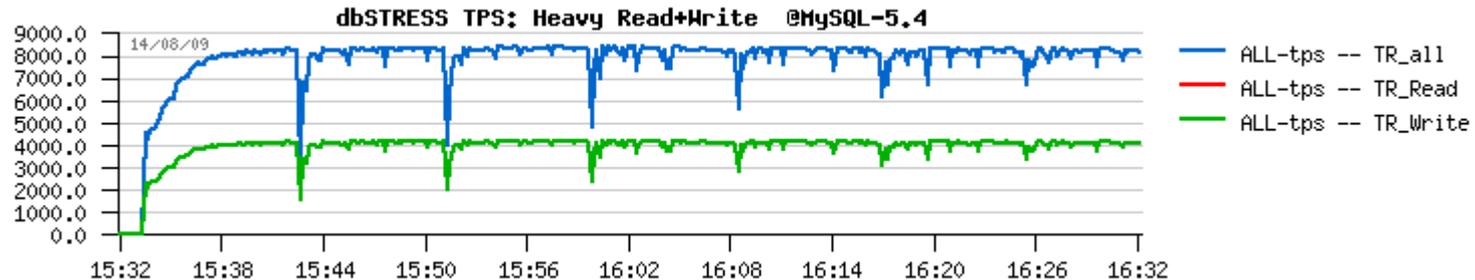


# InnoDB: Ahead Flushing

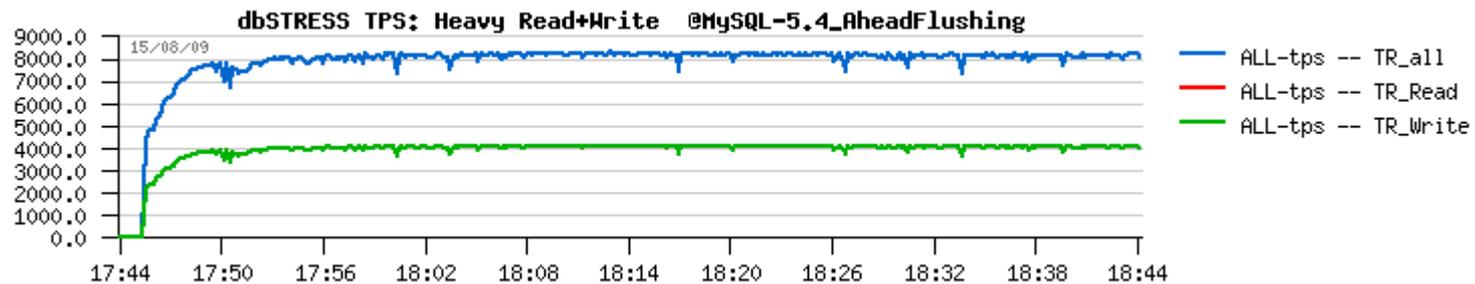
- Why Ahead Flushing?..
  - > Avoid to reach the Max Checkpoint age
  - > May also be achieved by limiting a percentage of dirty pages in the buffer pool (it'll work now :-)), but:
    - > Painful to calculate / adapt settings for each workload...
    - > No direct dependency between dirty pages amount and checkpoint age / redo log size..
    - > Flushing only with 100% of I/O capacity may be too much aggressive
- Available solutions:
  - > Percona: Adaptive Checkpoint
  - > Innobase: Adaptive Flushing
  - > Dimitri: Ahead Flushing

# Performance with Purge + Ahead

- Long “Stress” Read+Write:
  - > MySQL 5.4 default:



- > MySQL 5.4 + purge & ahead fix



# MySQL Optimization

- Tuning
- Query plan
- Query profiler
- Monitoring
- DTrace

# Setting your my.conf

- MyISAM
  - > flush\_time = Nsec
  - > key\_buffer\_size
  - > DML priority
- InnoDB
  - > Concurrency
  - > Pool / Log buffer size
  - > Redo Log size
  - > I/O capacity / threads
  - > Commit = 1 / 2 / 0
  - > Flush method
  - > Checksums? etc.

```
[mysqld]
max_connections = 2000
flush_time = 300
table_open_cache=2000

# myisam
key_buffer_size=20M
low_priority_updates=1

# innodb
innodb_file_per_table
innodb_buffer_pool_size=16000M
innodb_additional_mem_pool_size=5M
innodb_log_buffer_size=8M
innodb_log_file_size=128M
innodb_log_files_in_group=2
innodb_thread_concurrency=16
innodb_read_io_threads=8
innodb_write_io_threads=8
innodb_flush_log_at_trx_commit=2
innodb_flush_method= O_DIRECT
innodb_io_capacity=2000

#debug
##log-slow-queries=/tmp/mysql_slow.log
```

# Check your settings! :-)

- SQL>
  - > Show variables;
  - > Show variables like '%inno%';

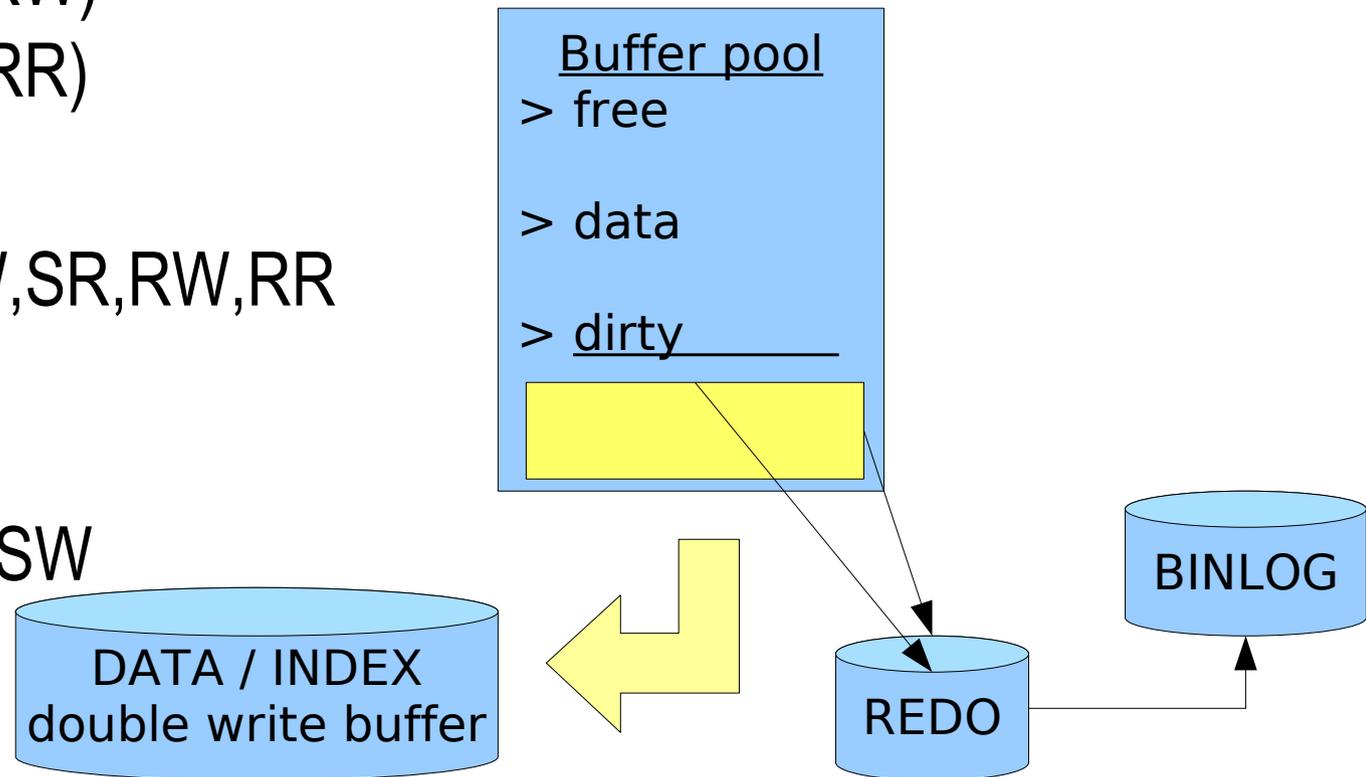
```
mysql> show variables like 'inno%';
+-----+-----+
| Variable_name          | Value                |
+-----+-----+
| innodb_adaptive_hash_index | ON                  |
| innodb_additional_mem_pool_size | 5242880             |
| innodb_autoextend_increment | 64                  |
| innodb_autoinc_lock_mode   | 1                   |
| innodb_buffer_pool_size    | 67108864            |
| innodb_checksums           | ON                  |
| innodb_commit_concurrency  | 0                   |
| innodb_concurrency_tickets | 500                 |
| innodb_data_file_path      | ibdata1:10M:autoextend |
| innodb_data_home_dir      |                     |
| innodb_doublewrite         | ON                  |
| innodb_extra_dirty_writes  | ON                  |
| innodb_fast_shutdown       | 1                   |
| innodb_file_per_table      | ON                  |
| ...                        |                     |
```

# MySQL Query Cache

- Performance improvement: may be very huge!
- Performance degradation: may be very huge too! :-)
- If you want to use => limit it to 10-20MB
- Monitor your cache hit to be sure it's benefit!
- Otherwise: disable it in my.conf:
  - > `query_cache_size = 0`

# InnoDB and I/O Performance

- Keep in mind the nature of I/O operation!
  - > Sequential Write (SW)
  - > Sequential Read (SR)
  - > Random Write (RW)
  - > Random Read (RR)
- InnoDB
  - > Data files  $\leq$  SW,SR,RW,RR
  - > Redo log  $\leq$  SW
  - > Bin log  $\leq$  SW
  - > Double write  $\leq$  SW

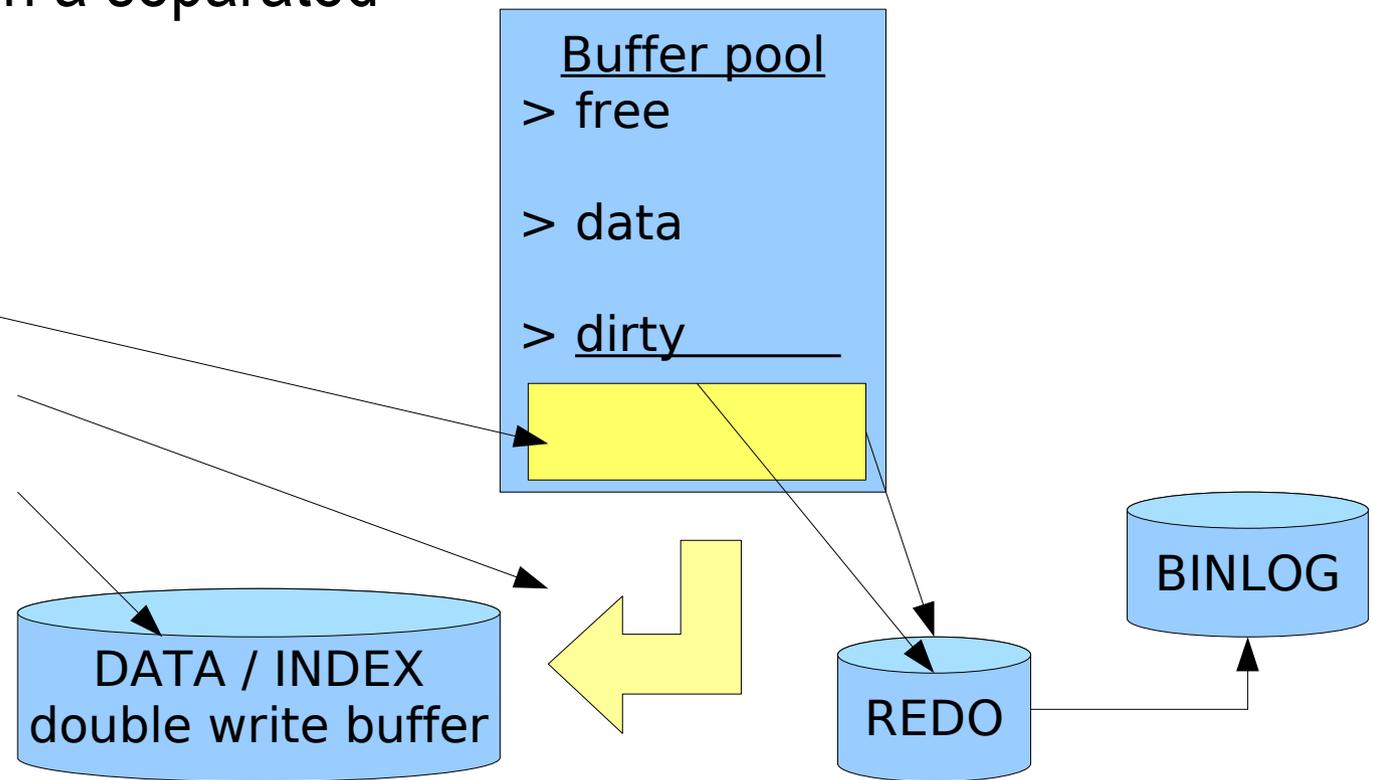


# InnoDB and I/O Performance

- Avoid a hot-mix of I/O operations!
  - > Random Read (RR) <= most painful & costly!!!
  - > Place REDO on different LUNs/disks
  - > Place BINLOG on a separated storage array!

- I/O Settings

- > I/O capacity
- > I/O write threads
- > I/O read threads



# InnoDB: Doublewrite buffer

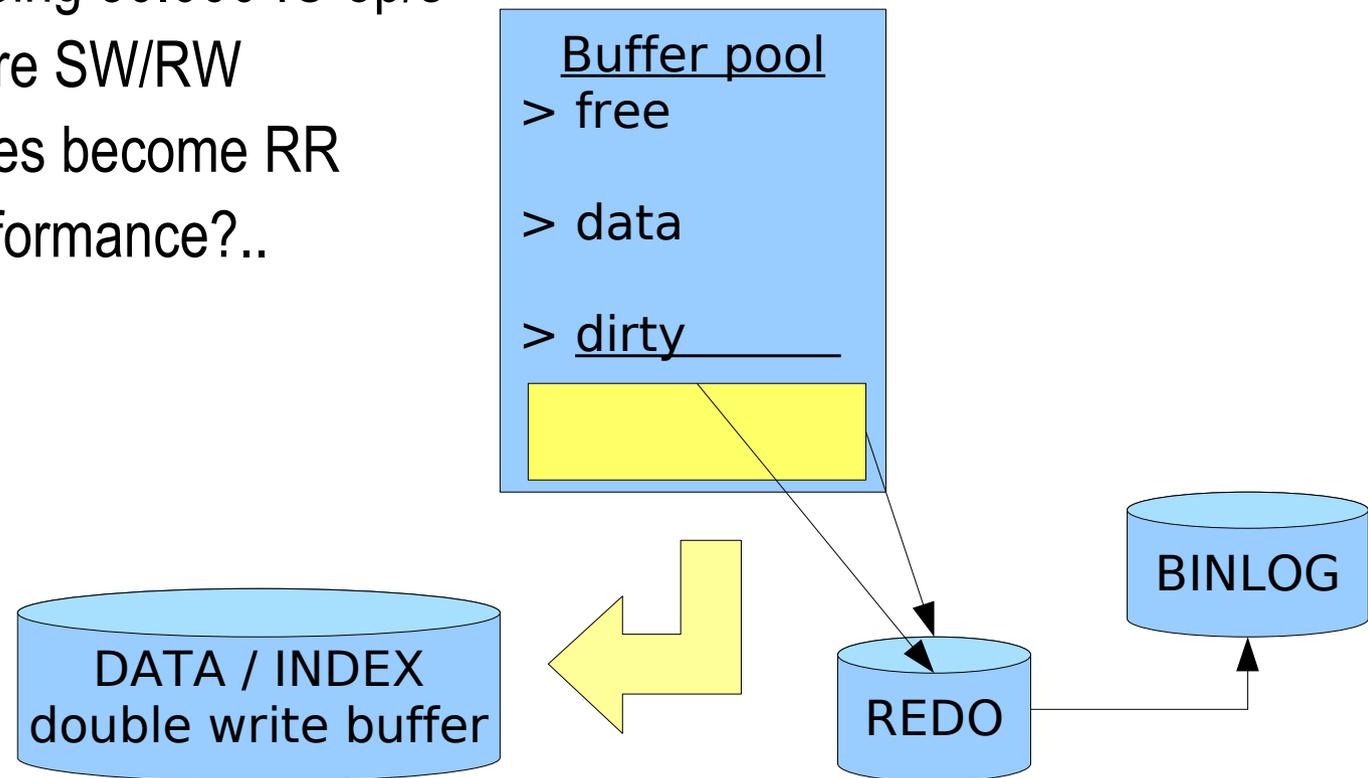
- Protecting from partially written pages
  - > Data first written into Doublewrite buffer (sys.tablespace)
  - > Then flushed to the datafiles
  - > On recovery: if partially written page discovered => use its image from doublewrite buffer
- What is the cost?..
  - > Doublewrite I/O is sequential, so should be fast
  - > Writes will do less sync calls:
    - > Instead of sync on every page write
    - > Sync once on doublewrite buffer write
    - > Then once on the datafile(s) for the same chunk of pages

# Doublewrite: Real Performance

- Usually:
  - > performance remains the same (or better)
  - > + recovery guarantee!
- In some cases:
  - > Up to 30% performance degradation...
  - > Why?...

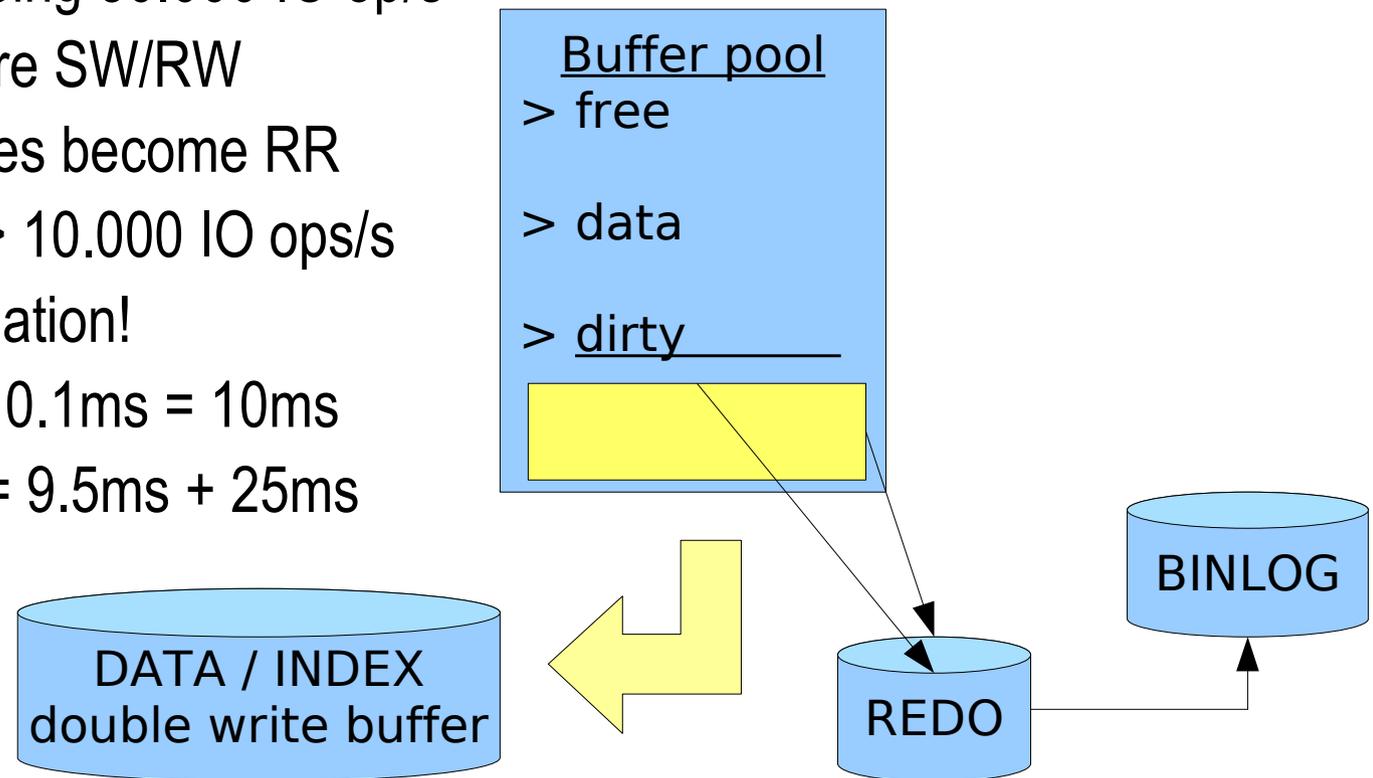
# Doublewrite: I/O dependence

- Random Reads are killing!
  - > RR = 5ms wait per operation
  - > Example:
    - > Application is doing 30.000 IO op/s
    - > All operations are SW/RW
    - > Now 5% of Writes become RR
    - > What about performance?..



# Doublewrite: I/O dependence

- Random Reads are killing!
  - > RR = 5ms wait per operation
  - > Example:
    - > Application is doing 30.000 IO op/s
    - > All operations are SW/RW
    - > Now 5% of Writes become RR
    - > Performance => 10.000 IO ops/s
    - > x3 times degradation!
    - > 100 SW = 100 x 0.1ms = 10ms
    - > 95 SW + 5 RR = 9.5ms + 25ms

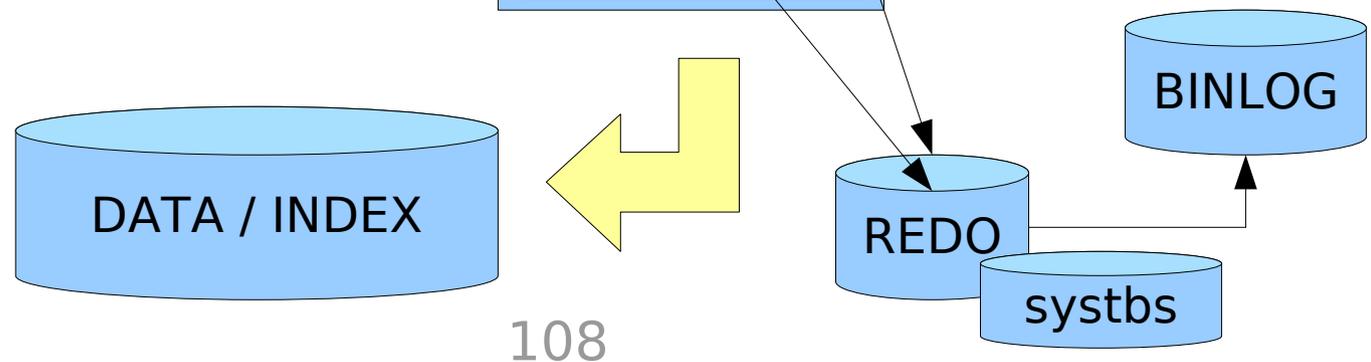


# Doublewrite: I/O dependence

- Workaround: move doublewrite buffer on REDO disks
  - > Have to set `innodb_file_per_table` initially for DB
  - > Move system tablespace on REDO disks

```
$ mv /DATA/ibdata1 /LOG  
$ ln -s /LOG/ibdata1 /DATA
```

```
Buffer pool  
> free  
> data  
> dirty
```



# MySQL Monitoring

- SQL> status;
- SQL> show status;
- SQL> show processlist;
- SQL> show innodb status \G
- etc...

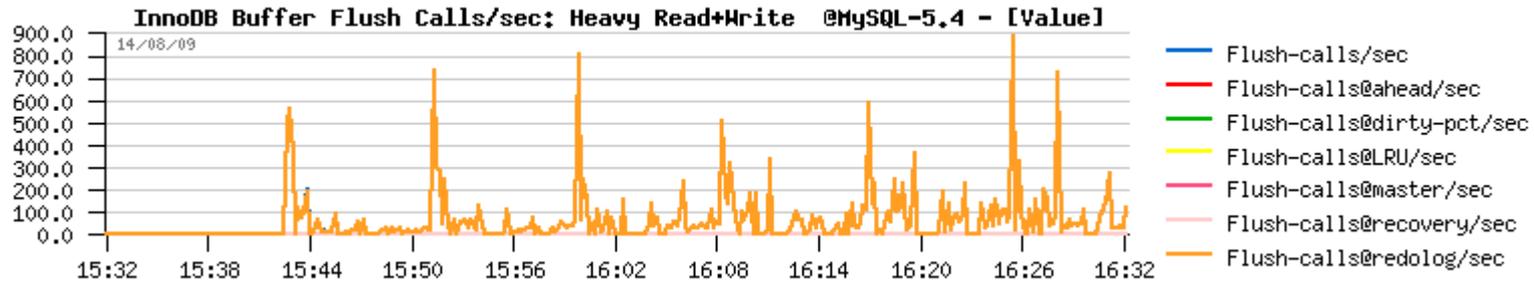
# InnoDB: Instrumentation

- Currently Available:
  - > SQL> show innodb status \G
  - > DTrace probes
- Extended “innodb status”:
  - > Google & Percona patches
  - > Dimitri:

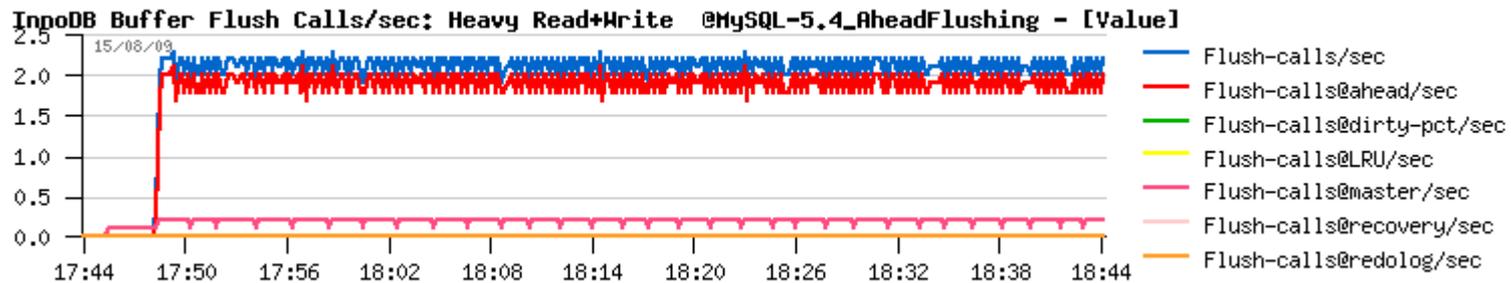
```
mysql> show innodb status \G
...
-----
PURGE STATUS
-----
History len: 2
DML Delay: 0 max: 0 delayed-queries: 0
Purge calls: 15 exec: 1 sleeps: 15 purged-pages: 2
-----
BUFFER FLUSH
-----
Flush calls: 350 exec: 33 flushed-pages: 13250
Called-by recovery: 0 redolog: 325 LRU: 0 master: 20 ahead: 0 dirty-pct: 5
```

# InnoDB: Instrumentation Example

- Flush Calls/sec by Caller
  - > MySQL 5.4 default:



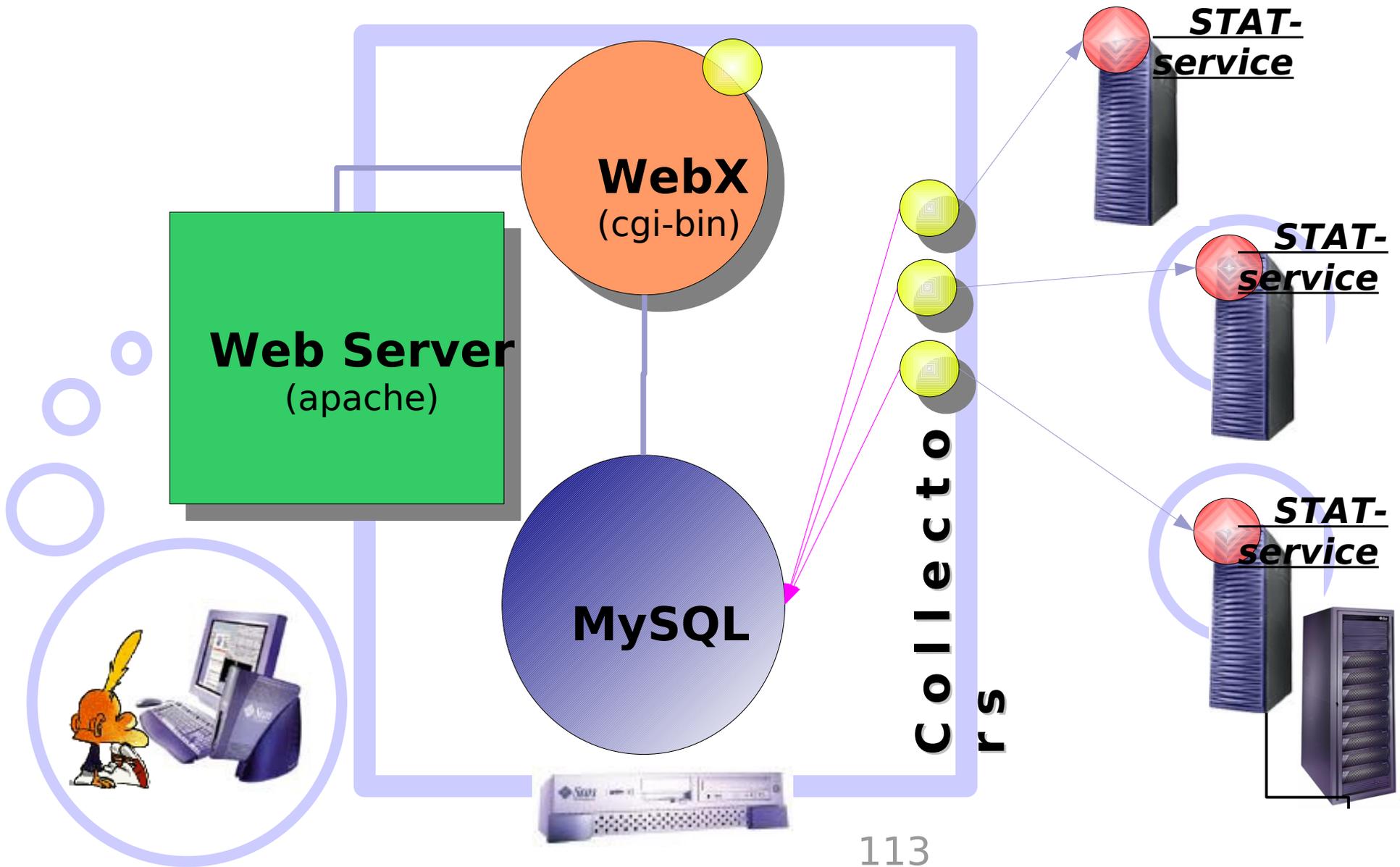
- > MySQL 5.4 + purge & ahead fix:



# Monitoring Tools

- MySQL Enterprise Monitoring
- Cacti
- dim\_STAT (<http://dimitrik.free.fr>)
  - > All System load stats (CPU, I/O, Network, RAM, Processes,...)
  - > Manly for Solaris & Linux, but any other UNIX too :-)
  - > Add-Ons for Oracle, MySQL, PostgreSQL, Java, etc.
  - > MySQL Add-Ons:
    - > MysqlSTAT : all available data from “show status”
    - > MysqlLOAD : compact data, multi-host monitoring oriented
    - > InnodbSTAT : most important data from “show innodb status”
    - > InnodbIOSTAT : Dtrace wrapper monitoring InnoDB I/O activity
  - > And any other you want to add! :-)

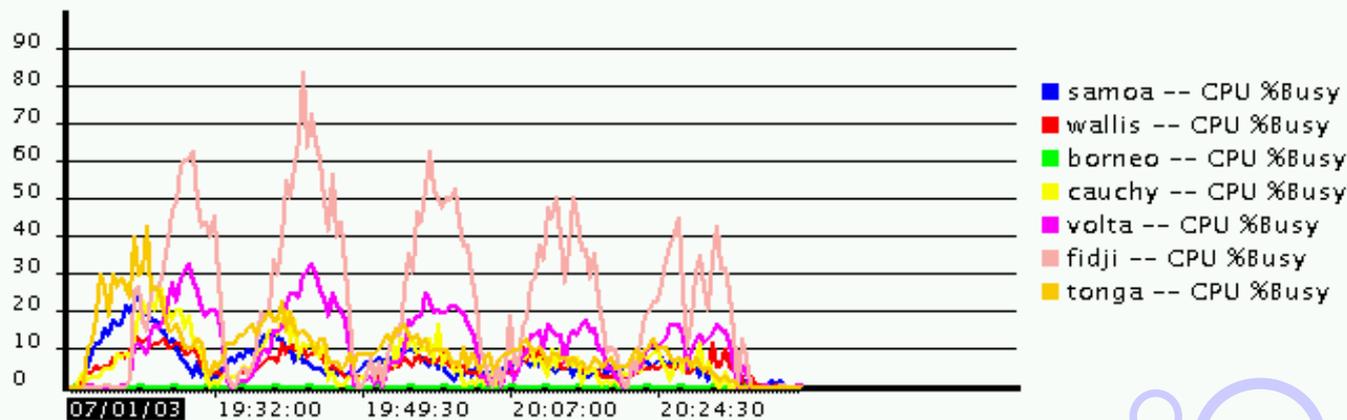
# dim\_STAT Architecture Overview



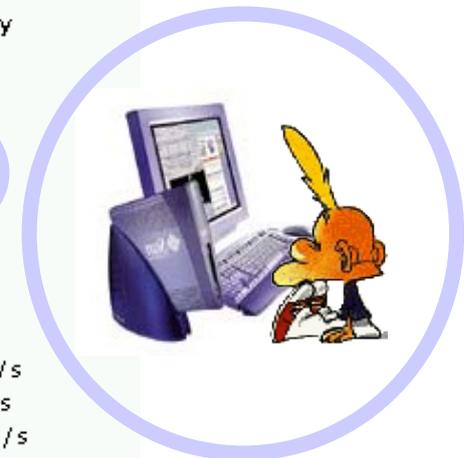
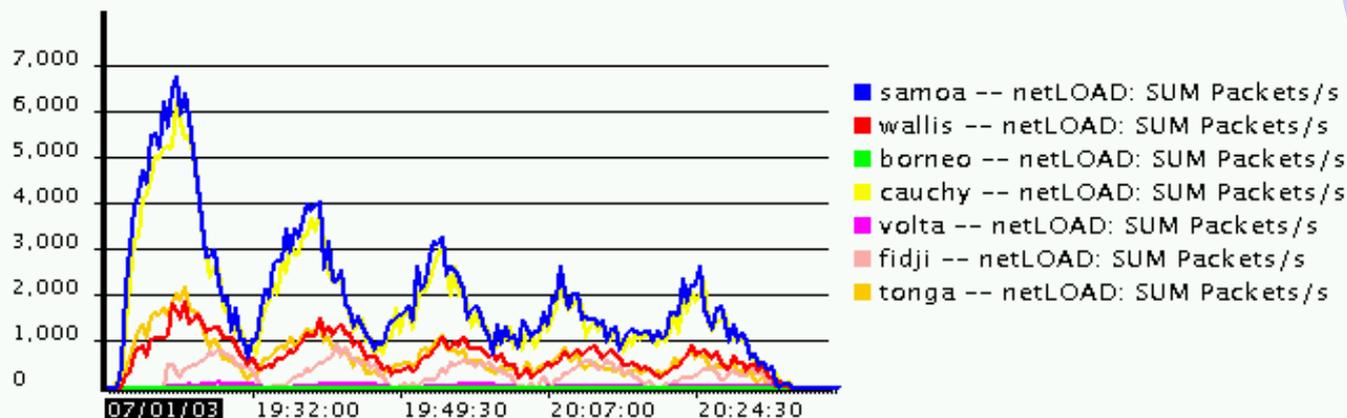
# Example: Multi-host Analyzing...

dim\_STAT Multi-Host Analyzer

CPU %Busy



netLOAD: SUM Packets/s



# MySQL Query Plan

- SQL> explain SELECT ... ;

```
mysql> explain select * from dim_STAT;
```

table	type	possible_keys	key	key_len	ref	rows	Extra
dim_STAT	ALL	NULL	NULL	NULL	NULL	19	

1 row in set (0.00 sec)

```
mysql> explain select * from dim_STAT where id = 1;
```

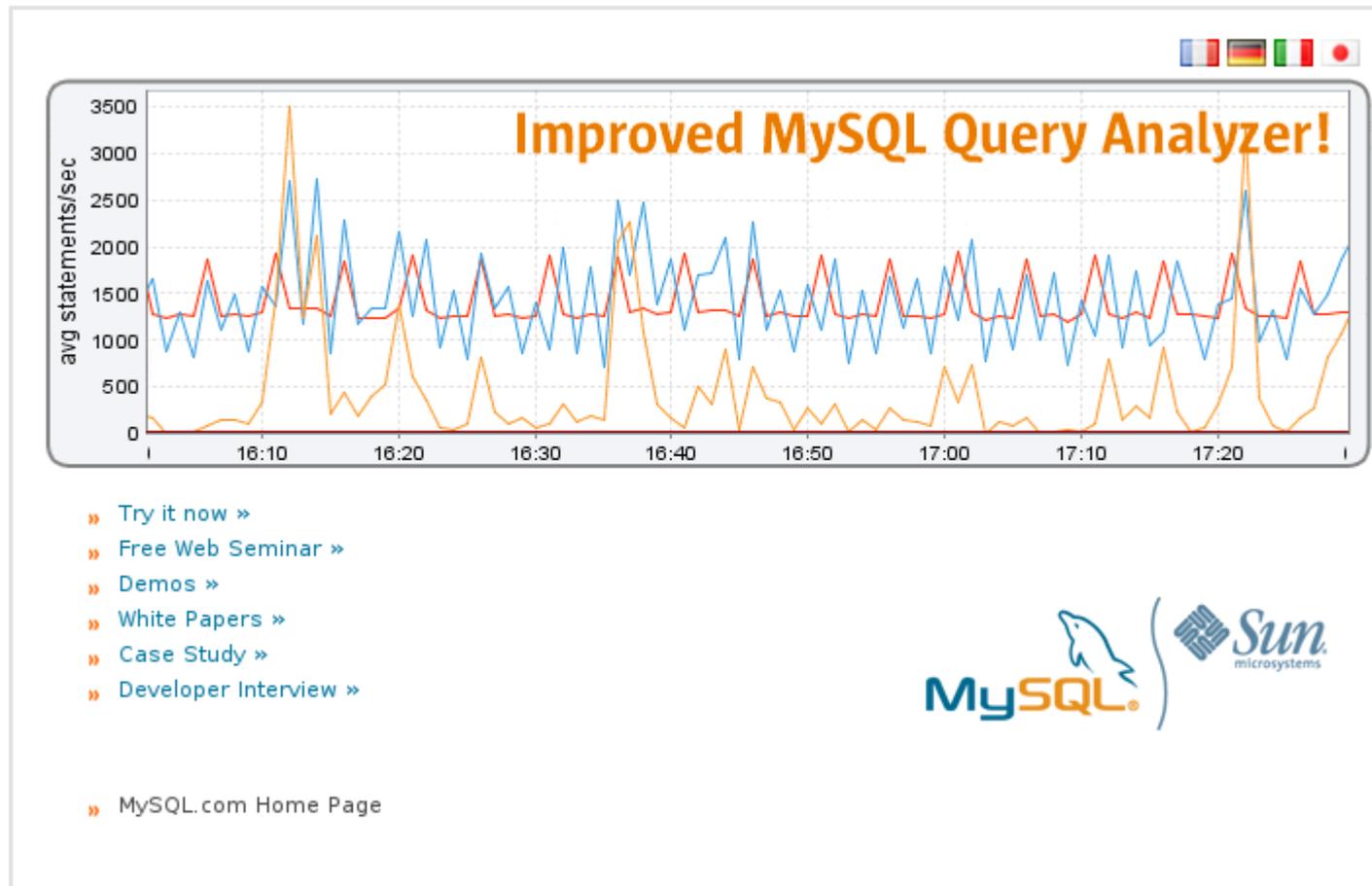
table	type	possible_keys	key	key_len	ref	rows	Extra
dim_STAT	const	dim_STAT_ID	dim_STAT_ID	4	const	1	

1 row in set (0.00 sec)

```
mysql>
```

# MySQL Query Analyzer

- Absolutely great tool!



# MySQL Query Profiler

- Available since MySQL 5.1
  - > Developed by MySQL community!
- How it works:
  - > SQL> set profiling = 1;
  - > SQL> query1 ;
  - > SQL> ...
  - > SQL> set profiling = 0;
  - > SQL> show profiles;
  - > SQL> show profile for query N;

# Query Profiler example

- Note: may profile only your current session!

```
mysql> set profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from toto;
...

mysql> set profiling = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> show profiles;
+-----+-----+-----+
| Query_ID | Duration | Query |
+-----+-----+-----+
| 1 | 0.00047300 | select * from toto |
+-----+-----+-----+
1 row in set (0.00 sec)
```

# Query Profiler example (2)

- Note: may profile only your current session!

```
mysql> show profile for query 1;
+-----+-----+
| Status          | Duration |
+-----+-----+
| starting        | 0.000098 |
| Opening tables  | 0.000024 |
| System lock     | 0.000010 |
| Table lock      | 0.000017 |
| init            | 0.000032 |
| optimizing       | 0.000011 |
| statistics      | 0.000021 |
| preparing       | 0.000019 |
| executing        | 0.000007 |
| Sending data    | 0.000119 |
| end             | 0.000009 |
| query end       | 0.000006 |
| freeing items   | 0.000086 |
| logging slow query | 0.000007 |
| cleaning up     | 0.000007 |
+-----+-----+
15 rows in set (0.00 sec)
```

# Query Profiler example2

- Coming from customer's production server:

```
mysql> show profile for query 1;
+-----+-----+
| Status                | Duration |
+-----+-----+
| starting              | 0.000125 |
| Opening tables        | 0.000015 |
| System lock           | 0.000003 |
| Table lock            | 0.000008 |
| init                  | 0.000070 |
| optimizing             | 0.000016 |
| statistics            | 0.000224 |
| preparing             | 0.000015 |
| Creating tmp table    | 0.000046 |
| executing              | 0.000002 |
| Copying to tmp table | 0.833941 |
| Sorting result        | 0.000243 |
| Sending data          | 0.000927 |
| end                   | 0.000003 |
| removing tmp table    | 0.000008 |
...

```

# MySQL & DTrace

- Everybody can!!!
- Switch to another presentation & prove it! :-)

# MySQL & ZFS

- ZFS Core Features:
  - > Data Integrity
  - > Everything is checksummed
  - > Immense capacity
    - > 128 bit filesystem
    - > Max size of a file is  $2^{64}$  bytes
    - > Each directory can hold  $2^{48}$  files
  - > Simple administration
    - > zpool & zfs are the only two commands you need to know
  - > Performance (workload depending)

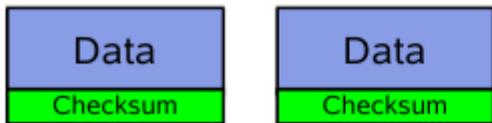
# ZFS Design

- Pooled Storage
  - > Common pool from which filesystems are allocated
- End-to-end data integrity
  - > Historically thought to be expensive, not really
  - > Alternative is unacceptable
- Everything is transactional
  - > Always consistent on disk
  - > Removes almost all constraints on IO order
  - > Think database transactions
- Copy on Write
  - > Never overwrite live data
  - > On-disk state is always consistent (never need fsck)

# End-to-end Checksums

## Disk Block Checksums

- Checksum stored with data block
- Any self-consistent block will pass
- Can't even detect stray writes
- Inherent FS/volume interface limitation

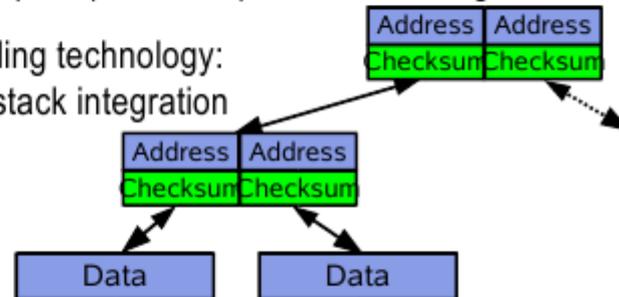


Only validates the media

✓	Bit rot
✗	Phantom writes
✗	Misdirected reads and writes
✗	DMA parity errors
✗	Driver bugs
✗	Accidental overwrite

## ZFS Checksum Trees

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire pool (block tree) is self-validating
- Enabling technology:  
ZFS stack integration

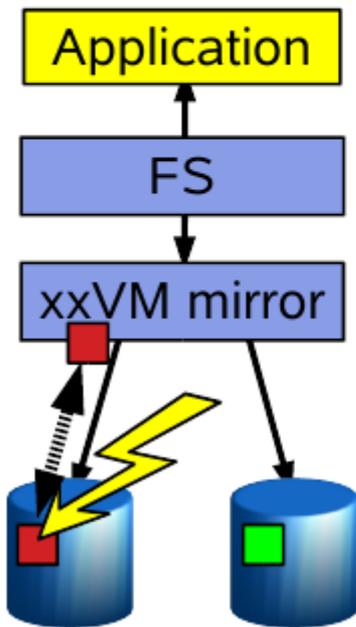


Validates the entire I/O path

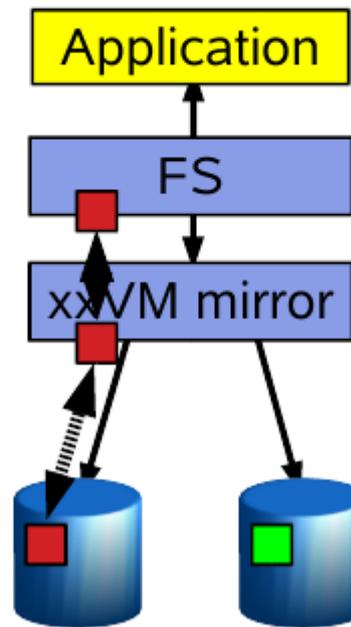
✓	Bit rot
✓	Phantom writes
✓	Misdirected reads and writes
✓	DMA parity errors
✓	Driver bugs
✓	Accidental overwrite

# Traditional Mirroring

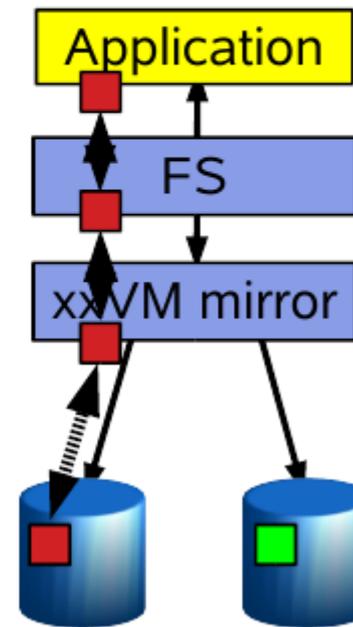
1. Application issues a read. Mirror reads the first disk, which has a corrupt block.  
It can't tell.



2. Volume manager passes bad block up to filesystem.  
If it's a metadata block, the filesystem panics. If not...

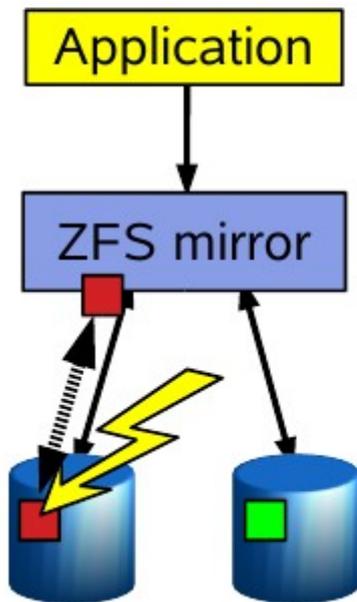


3. Filesystem returns bad data to the application. If the data is modified, both good & bad mirror copies will then be corrupted.

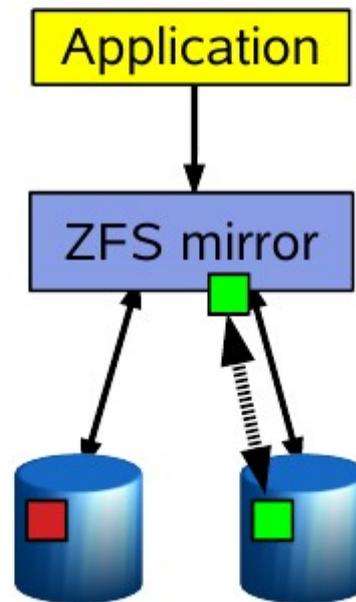


# ZFS Mirroring

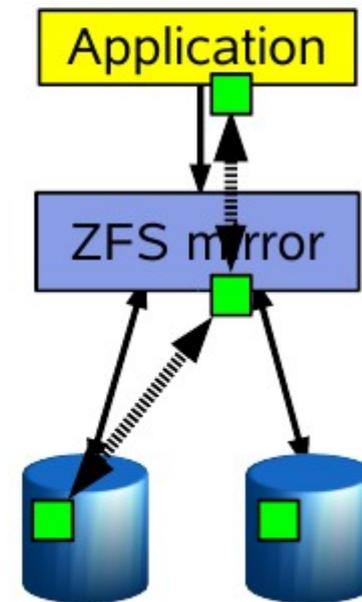
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



2. ZFS tries the second disk. Checksum indicates that the block is good.



3. ZFS returns good data to the application and repairs the damaged block.



# ZFS & MySQL Tuning

- Limit ZFS arc size from the beginning
  - > Use InnoDB cache rather FS
  - > Cache only metadata
    - > zfs set primarycache=metadata fs/db
- Align ZFS record size to DB block size!
  - > zfs set recordsize=16k fs/db <== do it before DB creation!!
  - > Use different pools for data, redo & binlog!
- Turn off ZFS prefetch:
  - > set zfs:zfs\_prefetch\_disable = 1
- If storage arrays - turn off cache flush:
  - > set zfs:zfs\_nocacheflush = 1

# ZFS as MySQL Hot Backup

- SQL> flush tables with read lock;
- ZFS snapshot (less 1 sec)
- SQL> unlock tables;
- Backup your data from the snapshot now and take your time :-))

# Q & A

- All details about presented stuff you may find on:

<http://dimitrik.free.fr>