Firebird Tour 2017: Performance

Vlad Khorsun, Firebird Project

About Firebird Tour 2017

- Firebird Tour 2017 is organized by Firebird Project, IBSurgeon and IBPhoenix, and devoted to Firebird Performance.
- The Platinum sponsor is Moscow Exchange
- Tour's locations and dates:
  - October 3, 2017 – Prague, Czech Republic
  - October 5, 2017 – Bad Sassendorf, Germany
  - November 3, 2017 – Moscow, Russia

MOSCOW EXCHANGE

- Platinum Sponsor
- Sponsor of
  - «Firebird 2.5 SQL Language Reference»
  - «Firebird 3.0 SQL Language Reference»
  - «Firebird 3.0 Developer Guide»
  - «Firebird 3.0 Operations Guide»
- Sponsor of Firebird 2017 Tour seminars
- www.moex.com

PLATINUM

IBSurgeon

HQBird

Advanced Firebird for Big Databases

- Replication, Recovery and Optimization for Firebird since 2002
- Platinum Sponsor of Firebird Foundation
- Based in Moscow, Russia

www.ib-aid.com

Performance related changes in FB3

- Engine internals
  - SMP-friendly Super Server
  - Hash Join algorithm
  - Improved nbackup synchronization
  - Reduced number of Pointer Page fetches
  - Delayed Header and\or TIP Page writes for read-only transactions

Performance related changes in FB3

- ODS12
  - Primary and Secondary Data Pages
  - Flag Swept for Data and Pointer Pages
  - Allocate and free Data Pages by extents
    - Extent is a group of 8 consecutive pages
  - SCN Pages for physical backup

Benchmarks

- Test machine
  - CPU: Intel i4770, 4/8 cores
  - HDD: 2 SATA drives in RAID 1 (mirror)
  - SSD: Samsung 850 Pro, SATA
  - RAM: 16 GB
  - OS: Windows 7

Benchmarks

- Test database
  - Generated by TPCC load tool
  - Physical parameters
    - 8 KB page size
    - 8.75 GB
    - Forced Writes = ON
  - Restored from the file copy before each test run

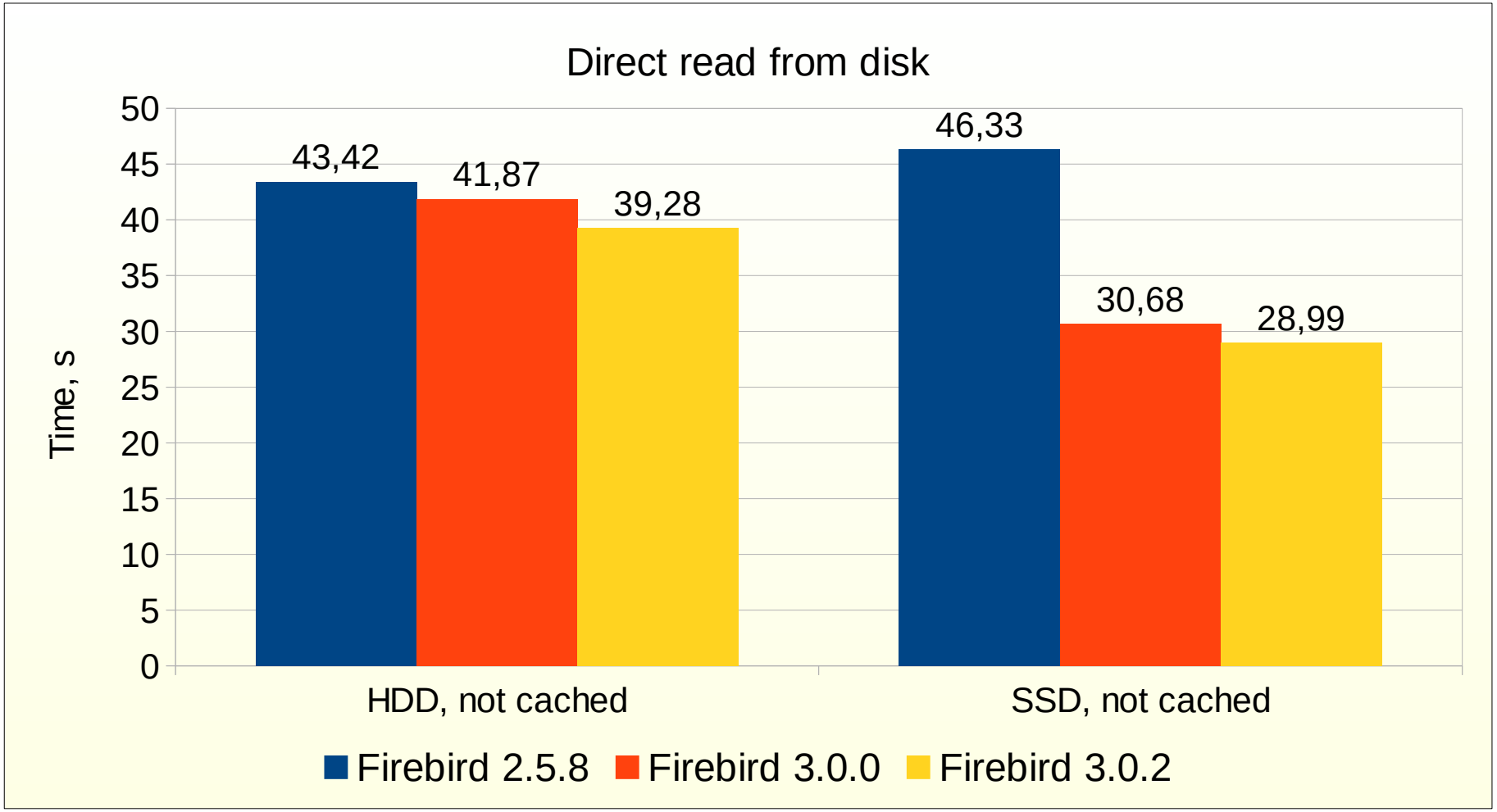
Table	Rows	Data Pages
CUSTOMER	3`000`000	235`808
DISTRICT	1`000	24
ITEM	100`000	1`520
STOCK	10`000`000	421`528
WAREHOUSE	100	2

Read-only, single table

- Test Query:
  - `SELECT COUNT(*) FROM STOCK`
- Page cache
  - 500`000 buffers
- Caching effect
  - Direct read from disk, not cached by OS nor by Firebird
    - HDD
    - SSD
  - Cached by OS, Firebird cache is empty
    - all reads are from OS cache, no read from physical disk
  - Fully cached by Firebird
    - no reads at all

Read-only, single table

- Test Query:
  - `SELECT COUNT(*) FROM STOCK`

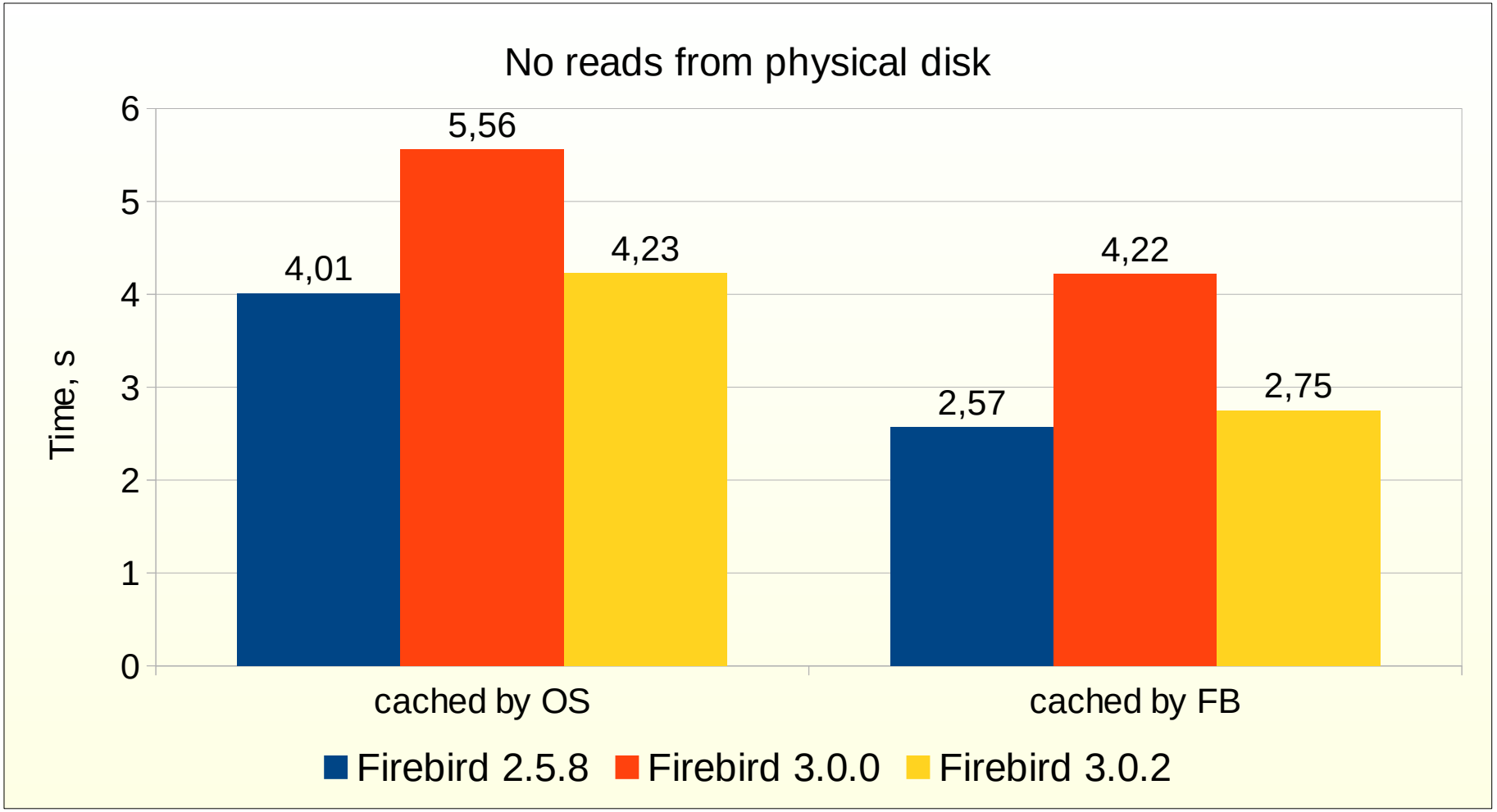


Read-only, single table

- Direct read from disk
  - HDD
    - Almost identical speed while FB3 is a bit faster than FB 2.5
      - Due to extents allocation?
  - SSD
    - FB 2.5 slower than on HDD!
      - It requires extra investigation
    - FB 3 30% faster on SSD
      - As expected

Read-only, single table

- Test Query:
  - `SELECT COUNT(*) FROM STOCK`

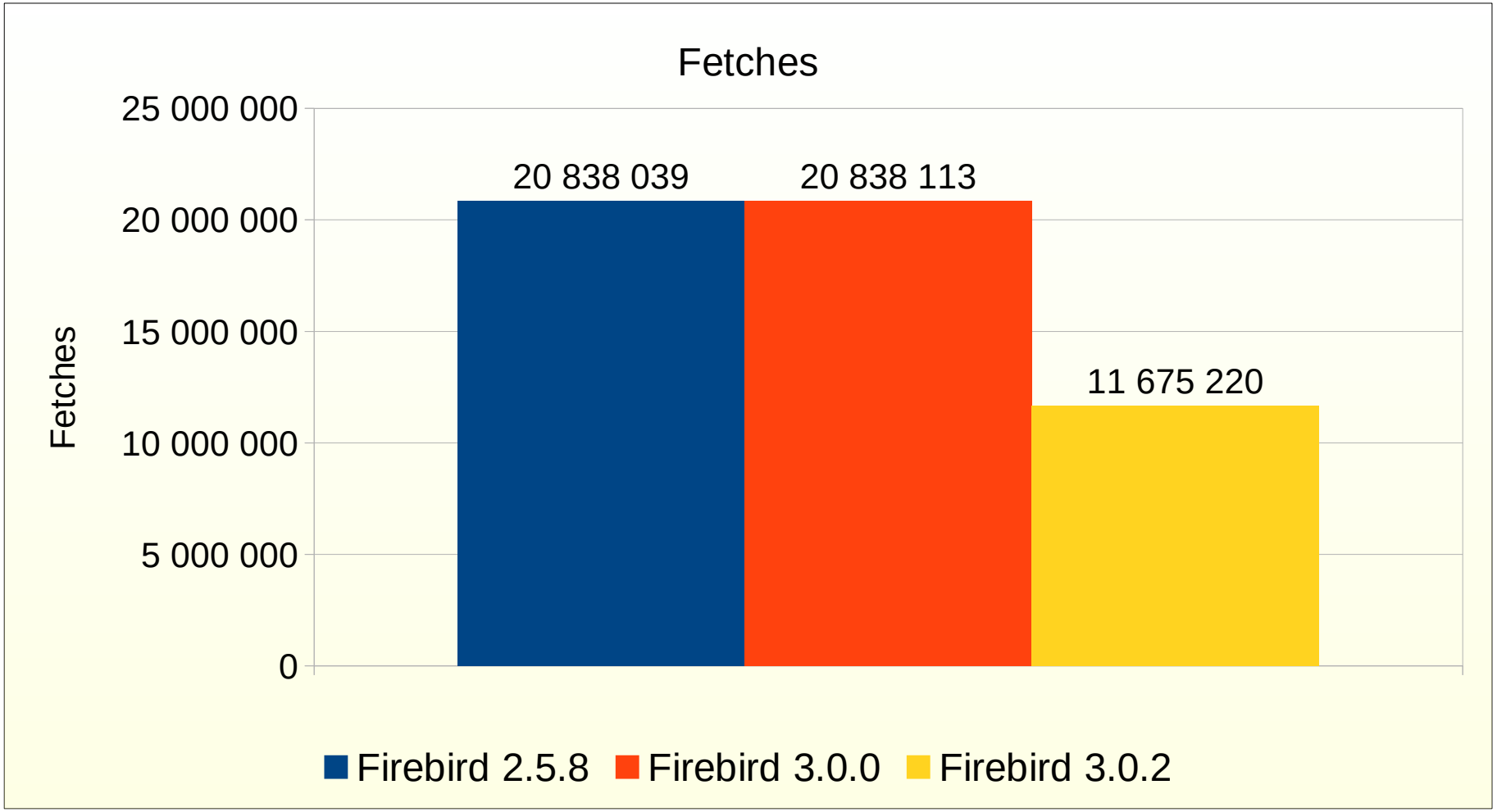


Read-only, single table

- Firebird 3.0.0 is much slower than Firebird 2.5.8
  - Minimal cost of page access in Super Server before Firebird 3 is very cheap : just increment and decrement
  - Minimal cost of page access in Super Server in Firebird 3 is much larger : four interlocked operations
- Firebird 3.0.2 is almost as fast as Firebird 2.5.8
  - Small internal cache of physical numbers of Data Pages
    - Allows to avoid most accesses of Pointer Pages
    - Reduces number of page accesses (fetches) almost two times

Read-only, single table

- Test Query:
  - `SELECT COUNT(*) FROM STOCK`



Read-only, LOOP JOIN

- Query

```
SELECT W.W_NAME, SUM(S.S_QUANTITY),
       SUM(I.I_PRICE * S.S_QUANTITY)
  FROM WAREHOUSE W JOIN STOCK S
            ON W.W_ID = S.S_W_ID
       JOIN ITEM I
            ON S.S_I_ID = I.I_ID
GROUP BY W.W_NAME
```

- PLAN

```
SORT (JOIN (
         W NATURAL,
         S INDEX (STOCK_PK),
         I INDEX (ITEM_PK)))
```

- Result set
  - 100 rows

Read-only, LOOP JOIN

Execution statistics

	Firebird 2.5.8	Firebird 3.0.2	Firebird 2.5.8	Firebird 3.0.2
Buffers	50 000	50 000	500 000	500 000
Fetches	70 015 390	60 413 829	70 015 390	60 413 828
Reads	10 188 968	10 300 878	0	0
Time	74,99	83,53	35,93	38,55

Per-table statistics

Table	Indexed	Non-indexed
ITEM	10 000 000	0
STOCK	10 000 000	0
WAREHOUSE	0	100

Read-only, LOOP JOIN

- Firebird 3 still 7-11% slower than Firebird 2.5
- Firebird 3 makes a bit less fetches but not enough less to catch up Firebird 2.5
- Table STOCK read all 10M records
  - Using index – far not efficient
- Table ITEMS read all 100K records 100 times!
  - Using index – far not efficient
- Plan and performance is very far from optimal

- Try to disable index access to avoid LOOP join?

Read-only, MERGE JOIN

- Query

```
SELECT W.W_NAME, SUM(S.S_QUANTITY),
       SUM(I.I_PRICE * S.S_QUANTITY)
  FROM WAREHOUSE W JOIN STOCK S
            ON W.W_ID +0 = S.S_W_ID +0
       JOIN ITEM I
            ON S.S_I_ID = I.I_ID +0
GROUP BY W.W_NAME
```

- Plan, Firebird 2.5.8

```
SORT (MERGE (
        SORT (MERGE (
                 SORT (S NATURAL),
                 SORT (I NATURAL))
              ),
        SORT (W NATURAL))
      )
```


Read-only, MERGE JOIN

- MERGE JOIN algorithm
  - Requires both input datasources to be sorted on the join columns
  - Sort both inputs
  - Join sorted data within one pass

Read-only, HASH JOIN

- Query

```
SELECT W.W_NAME, SUM(S.S_QUANTITY),
       SUM(I.I_PRICE * S.S_QUANTITY)
  FROM WAREHOUSE W JOIN STOCK S
            ON W.W_ID +0 = S.S_W_ID +0
       JOIN ITEM I
            ON S.S_I_ID = I.I_ID +0
GROUP BY W.W_NAME
```

- Plan, Firebird 3.0.2

```
SORT (
   HASH (
      HASH (S NATURAL, I NATURAL),
      W NATURAL
   )
)
```


Read-only, HASH JOIN

- HASH JOIN algorithm
  - Set larger datasource as left input, smaller datasource as right input
  - Build hash table for the right (smaller) datasource
  - Scan left datasource in natural order and probe hash table for matching rows

Read-only, MERGE vs HASH

Execution statistics

	Firebird 2.5.8	Firebird 3.0.2	Firebird 2.5.8	Firebird 3.0.2
Buffers	50 000	50 000	500 000	500 000
Fetches	21 040 678	12 027 694	21 040 678	12 027 694
Reads	420 343	423 272	0	0
Time	29,48	23,02	28,27	21,54

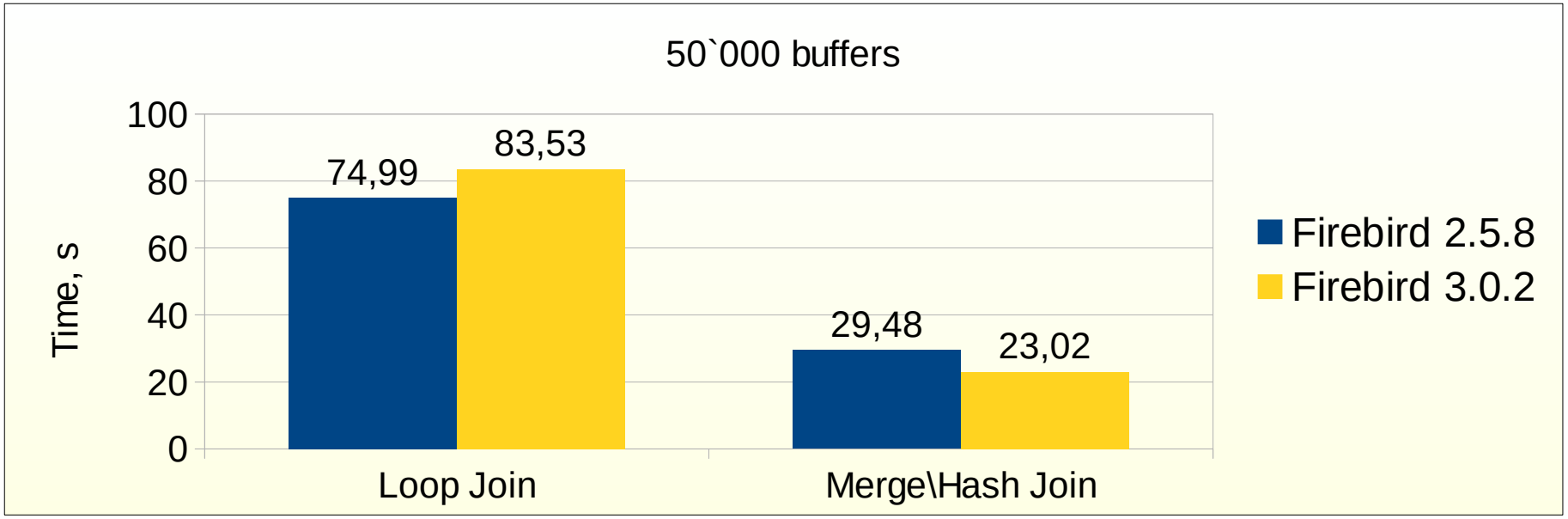
Per-table statistics

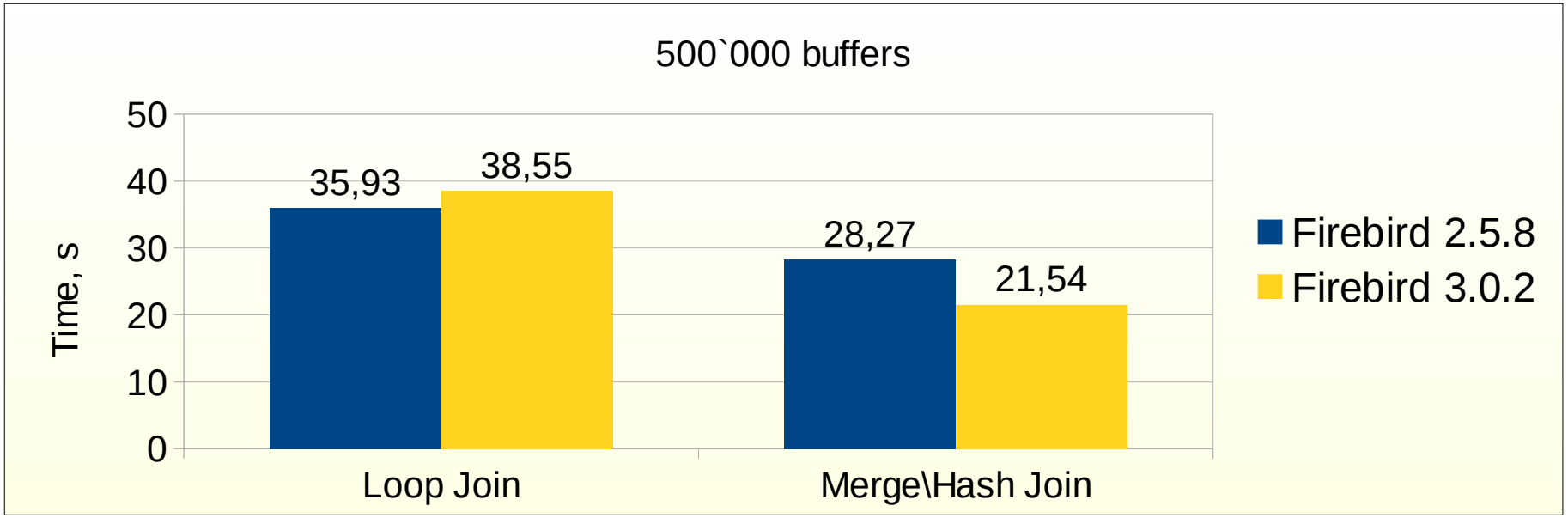
Table	Indexed	Non-indexed
ITEM	0	100 000
STOCK	0	10 000 000
WAREHOUSE	0	100

Read-only, MERGE vs HASH

- Firebird 2.5, MERGE vs LOOP
  - >23 times less reads (420K vs 10M)
  - 3.5 times less fetches (21M vs 70M)
- Firebird 3, HASH vs LOOP
  - >23 times less reads (423K vs 10M)
  - 5 times less fetches (12M vs 60M)
- Both versions
  - Much faster
  - Almost independent on Firebird cache size
- Firebird 3 now >22% faster than Firebird 2.5

Read-only, LOOP vs MERGE vs HASH





Read-only, multi-threaded

- Multi-threaded read-only benchmark
  - Table STOCK have 10'000'000 records
    - 100 warehouses, 100'000 items in each
  - Each reader reads random items in own warehouse
  - We don't test network subsystem
    - client application executes procedure, which selects a number of random items (100 by default) of the same warehouse

```
SELECT * FROM BENCH_1 (:W_ID, 100)
```

  - We don't test IO subsystem
    - before each test series we ensure whole table STOCK is fully cached by filesystem
  - We do test Firebird scalability

Read-only, multi-threaded

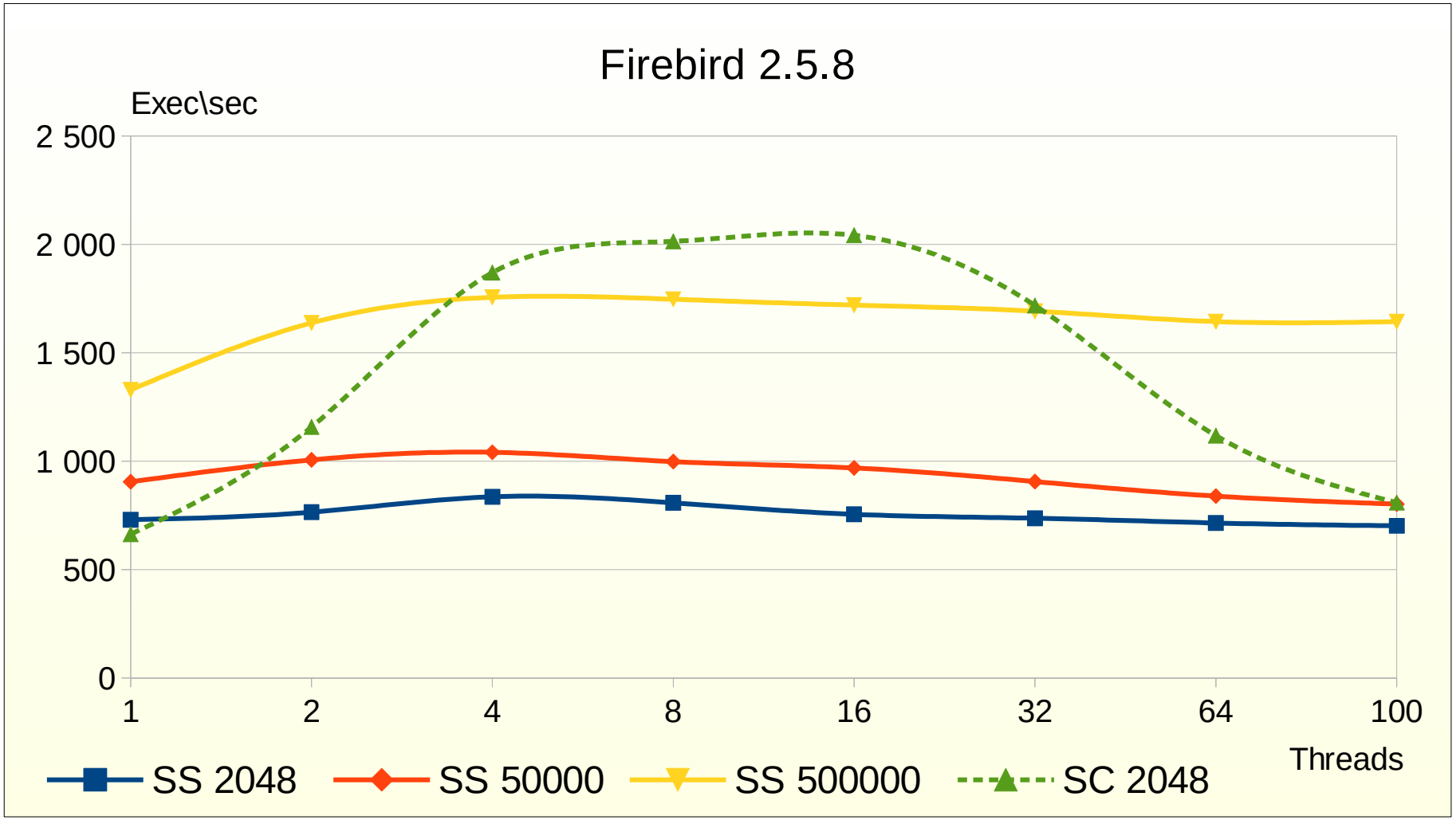
```
CREATE OR ALTER PROCEDURE BENCH_1 (
  W_ID INTEGER,
  NUM  INTEGER)
RETURNS (
  I_ID INTEGER)
AS
DECLARE I1 INTEGER;
BEGIN
  WHILE (NUM > 0) DO
  BEGIN
    I1 = RAND() * (100000 - 1);

    SELECT S.S_I_ID
      FROM STOCK S
     WHERE S.S_W_ID = :W_ID AND S.S_I_ID = :I1
      INTO :I_ID;

    SUSPEND;

    NUM = NUM - 1;
  END
END
```


Read-only, multi-threaded

- Multi-threaded read-only benchmark
  - Firebird versions
    - 2.5.8 base version for comparison
    - 3.0.0 first release of v3, have some perf problems
    - 3.0.2 current release of v3, have some improvements
  - Super Server
    - default cache size (2 048)
    - medium cache size (50 000)
    - huge cache size (500 000)
  - Classic
    - big cache size (2 048)
    - SuperClassic for Firebird 2.5
    - Classic mode for Firebird 3

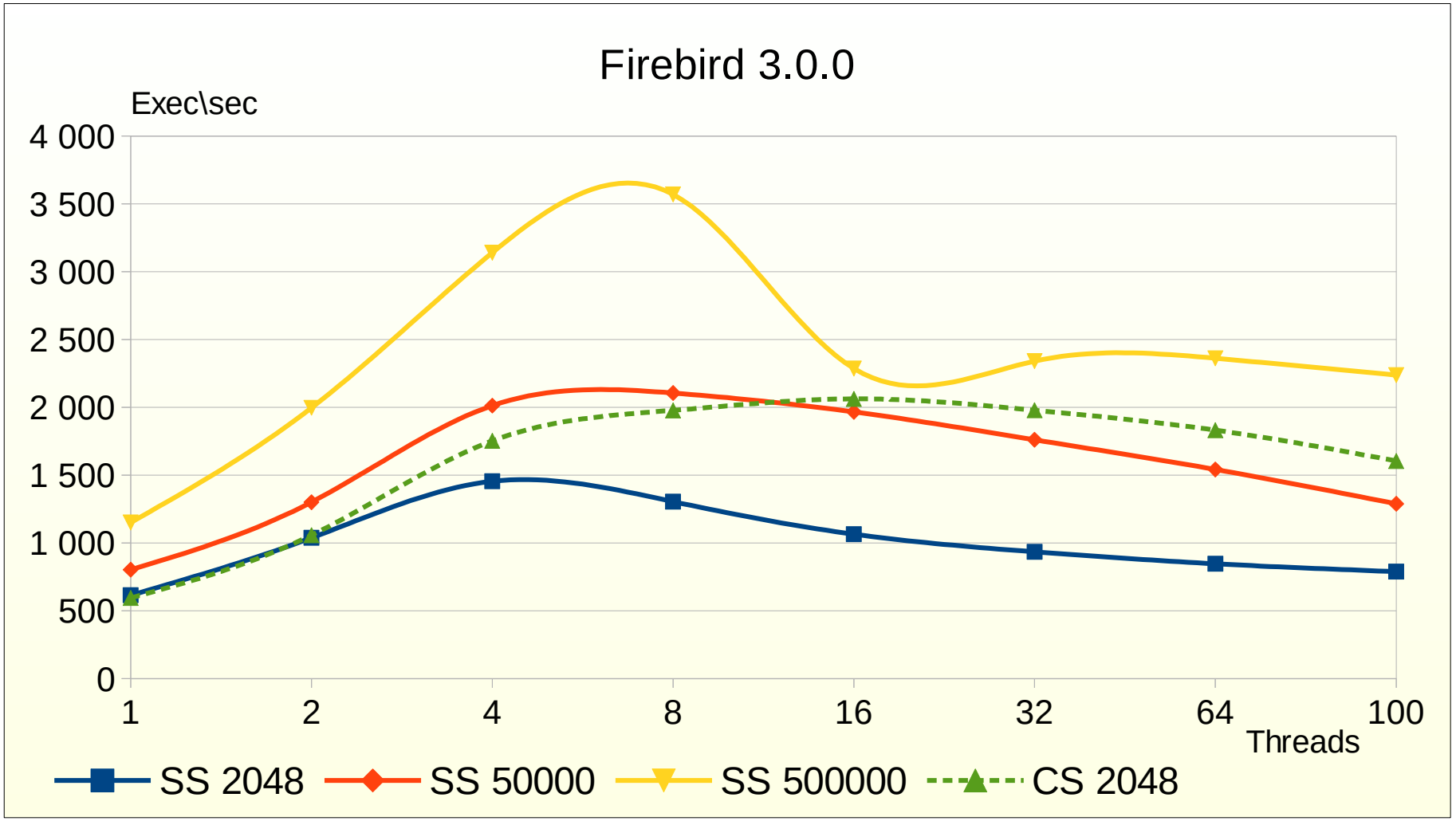
Read-only, multi-threaded

- Different modes within same version



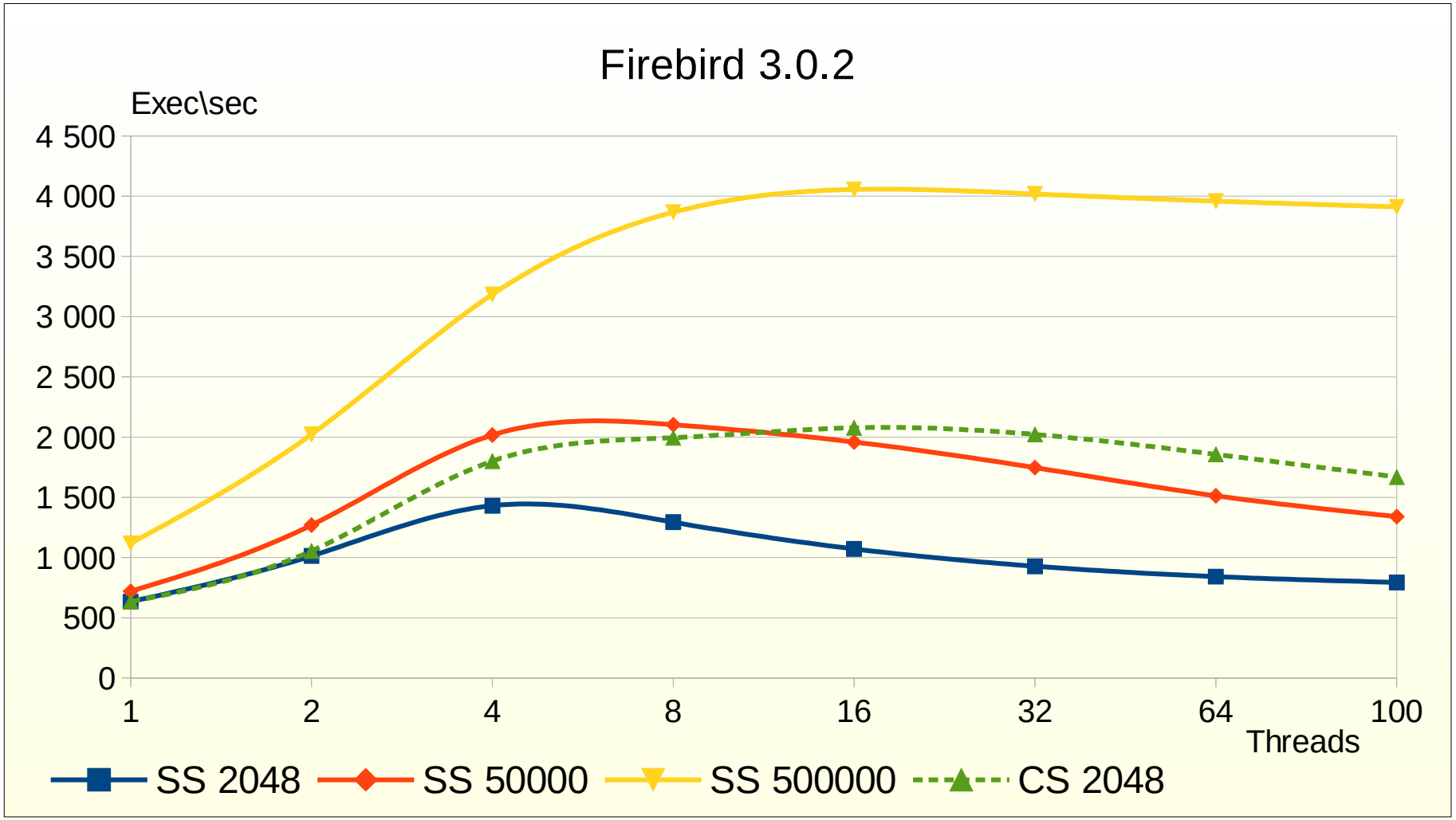
Read-only, multi-threaded

- Different modes within same version



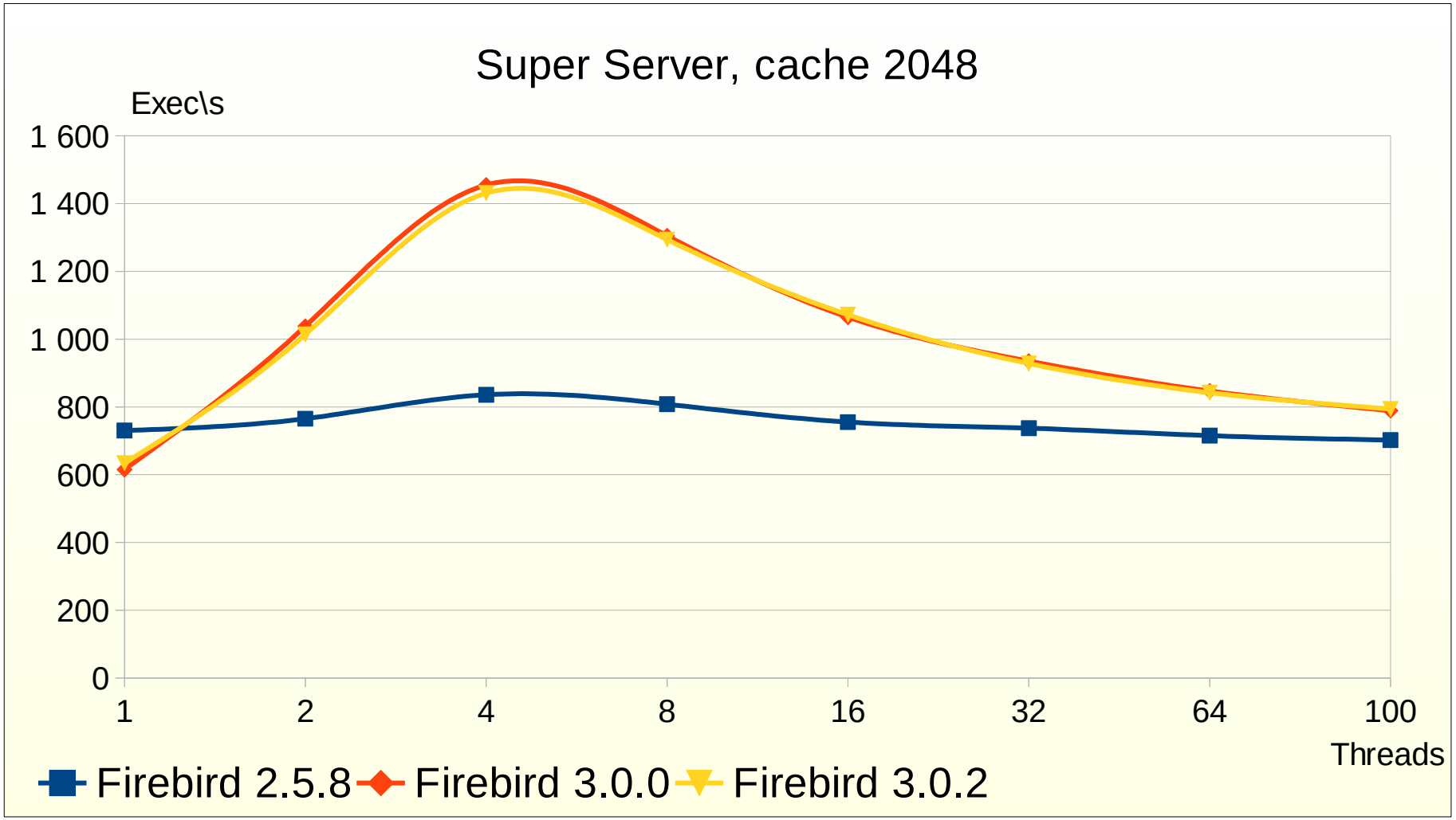
Read-only, multi-threaded

- Different modes within same version



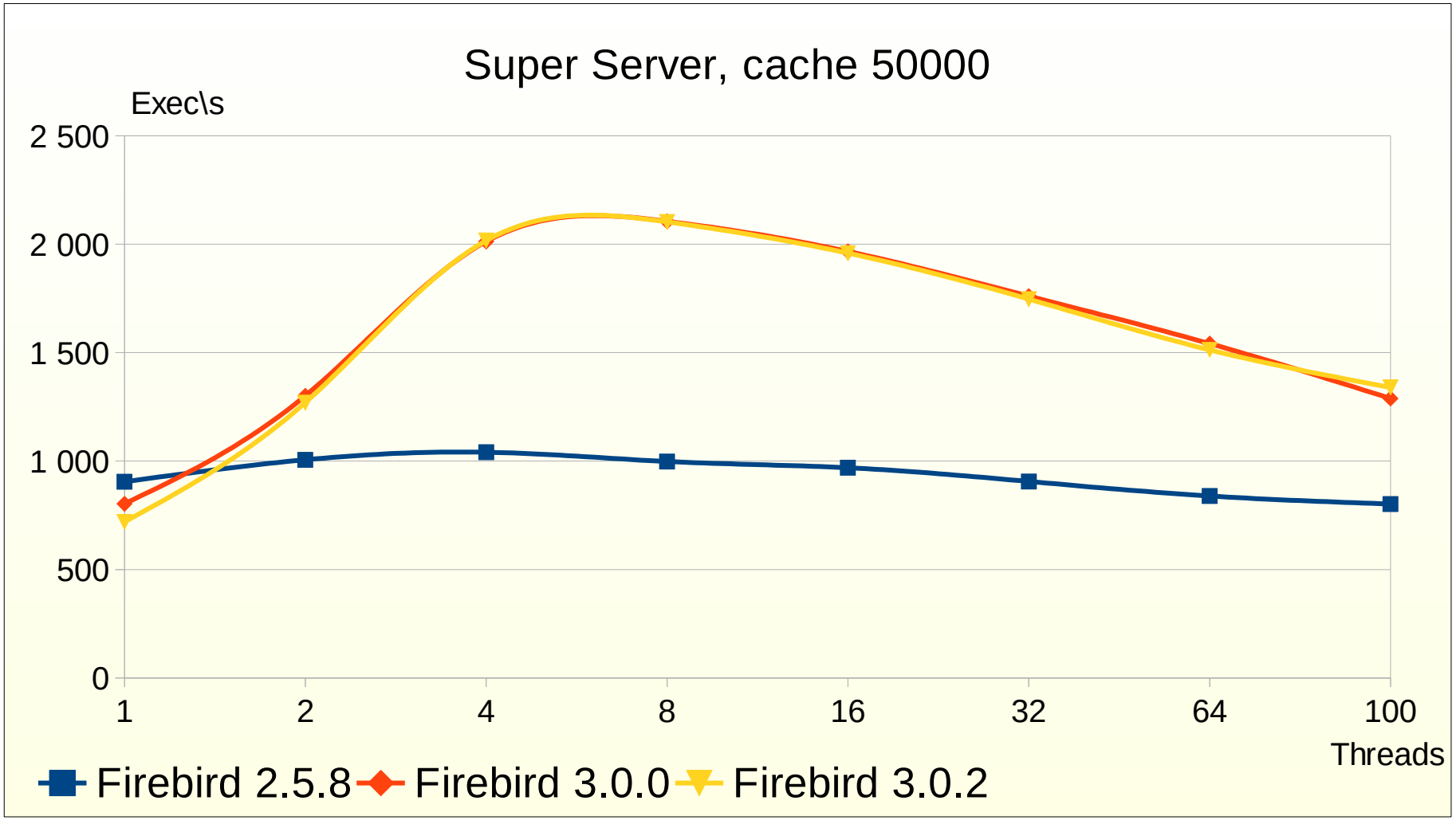
Read-only, multi-threaded

- Different versions, same mode



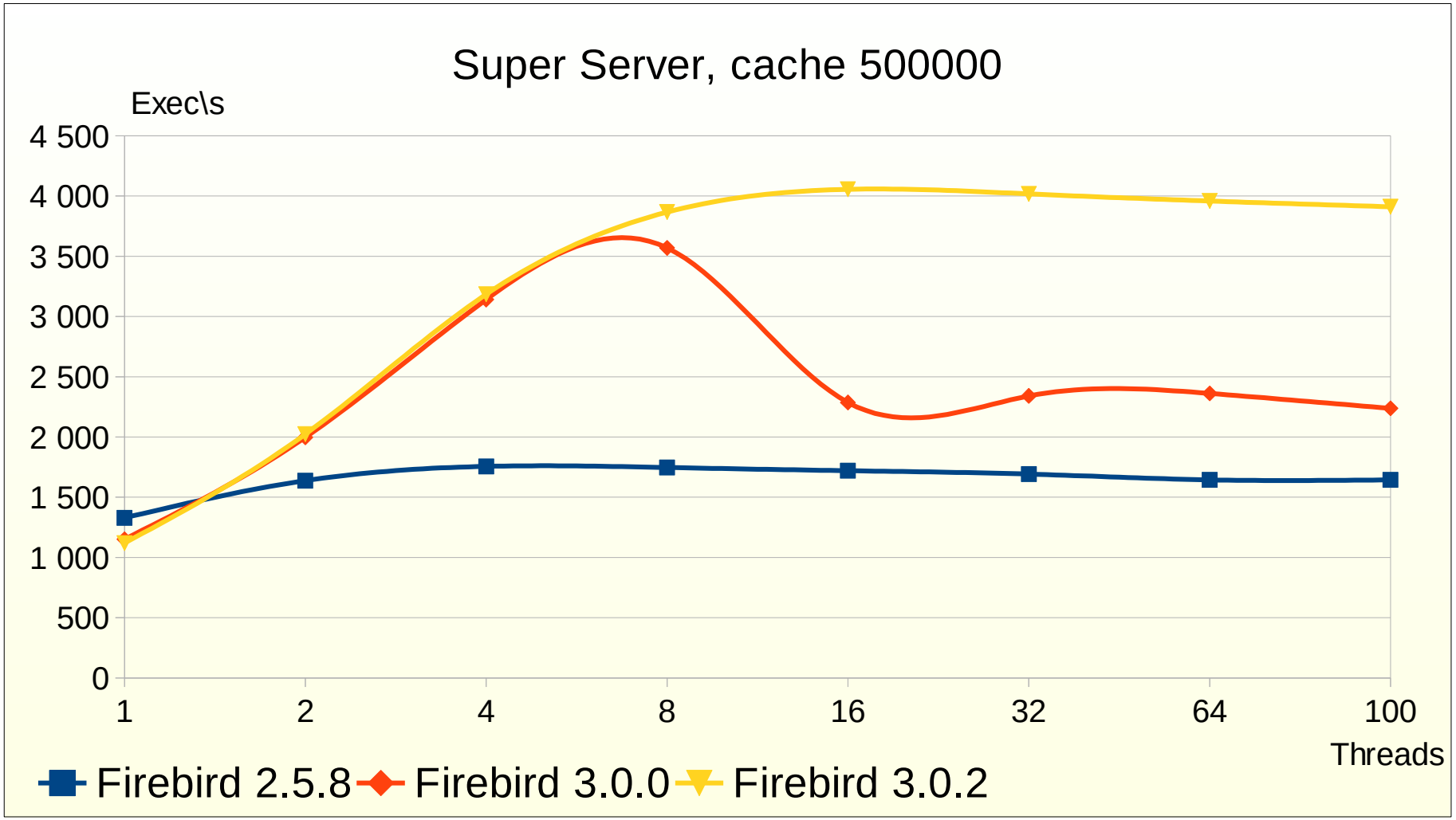
Read-only, multi-threaded

- Different versions, same mode



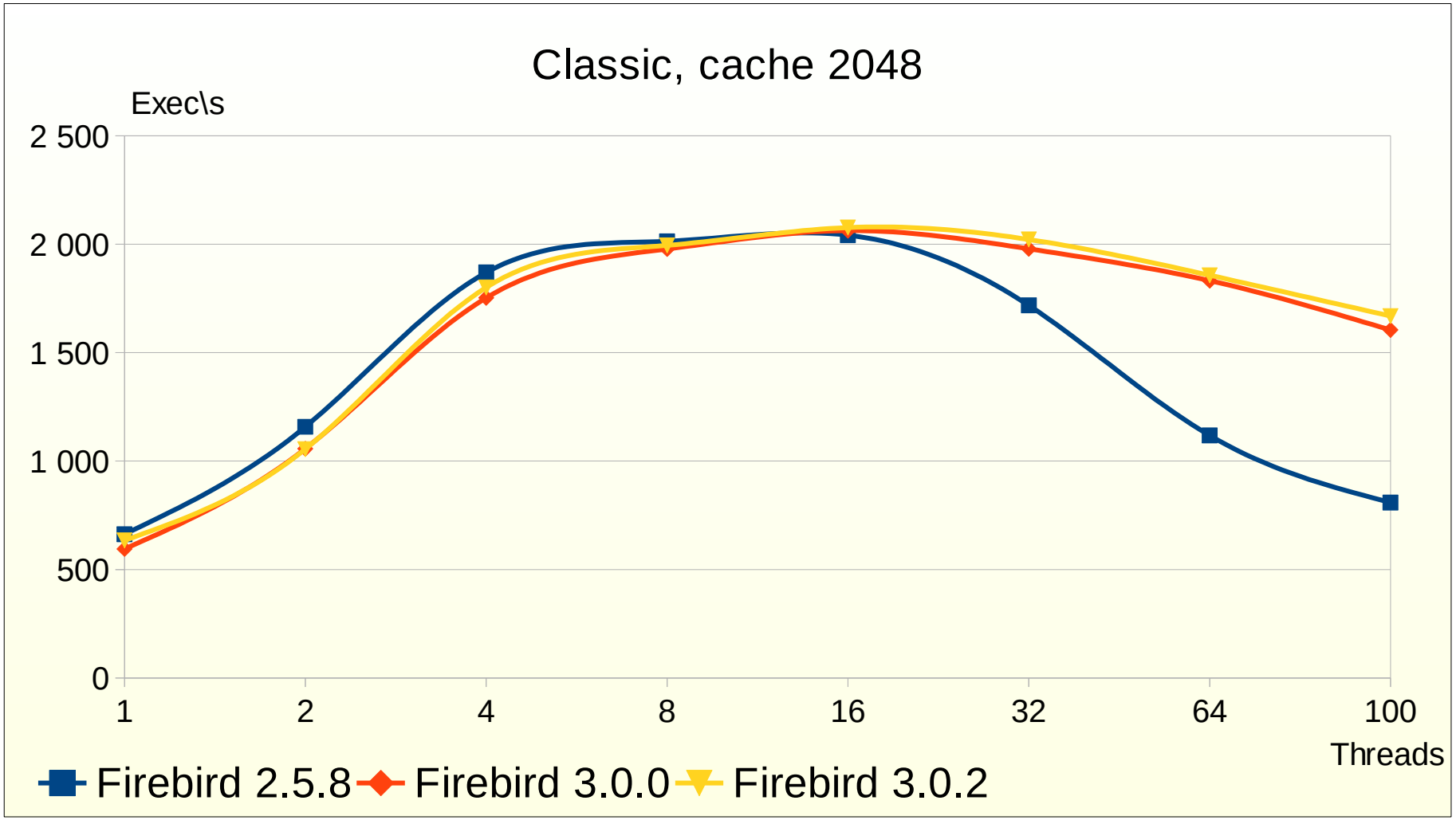
Read-only, multi-threaded

- Different versions, same mode



Read-only, multi-threaded

- Different versions, same mode



Read-only, multi-threaded

- Super Classic 2.5.8 scales well up to 8 threads, keeps the performance at 16 threads and then performance drops
- Classic v3 scales to the almost same level as 2.5.8 but degrades in less degree
- Super v3 with small and medium cache run a bit worse than Classic v3
- Super 3.0.0 with huge cache scales up to 8 threads, then performance drops significantly
- Super 3.0.2 with huge cache scales well up to 8 threads, slightly adds at 16 threads and then performance drops less than by 4% (for 100 threads)
- Super 3.0.2 with huge cache works two times faster than Classic

TPCC benchmark

- TPCC database
  - Generated by TPCC load tool
    - 100 warehouses
    - Standard scaling factor
  - Physical parameters
    - 8 KB page size
    - 8.75 GB
    - Forced Writes = ON
  - Restored from the file copy before each test run

TPCC benchmark

- TPCC test parameters
  - Terminals: 1, 2, 4, 8, 16, 32, 64, 100
    - Each terminal uses own connection
    - Each terminal works within own thread
  - Whole test time: 35 min
    - Warm time: 5 min
    - Steady time: 30 min
  - Measurements: every 10 sec

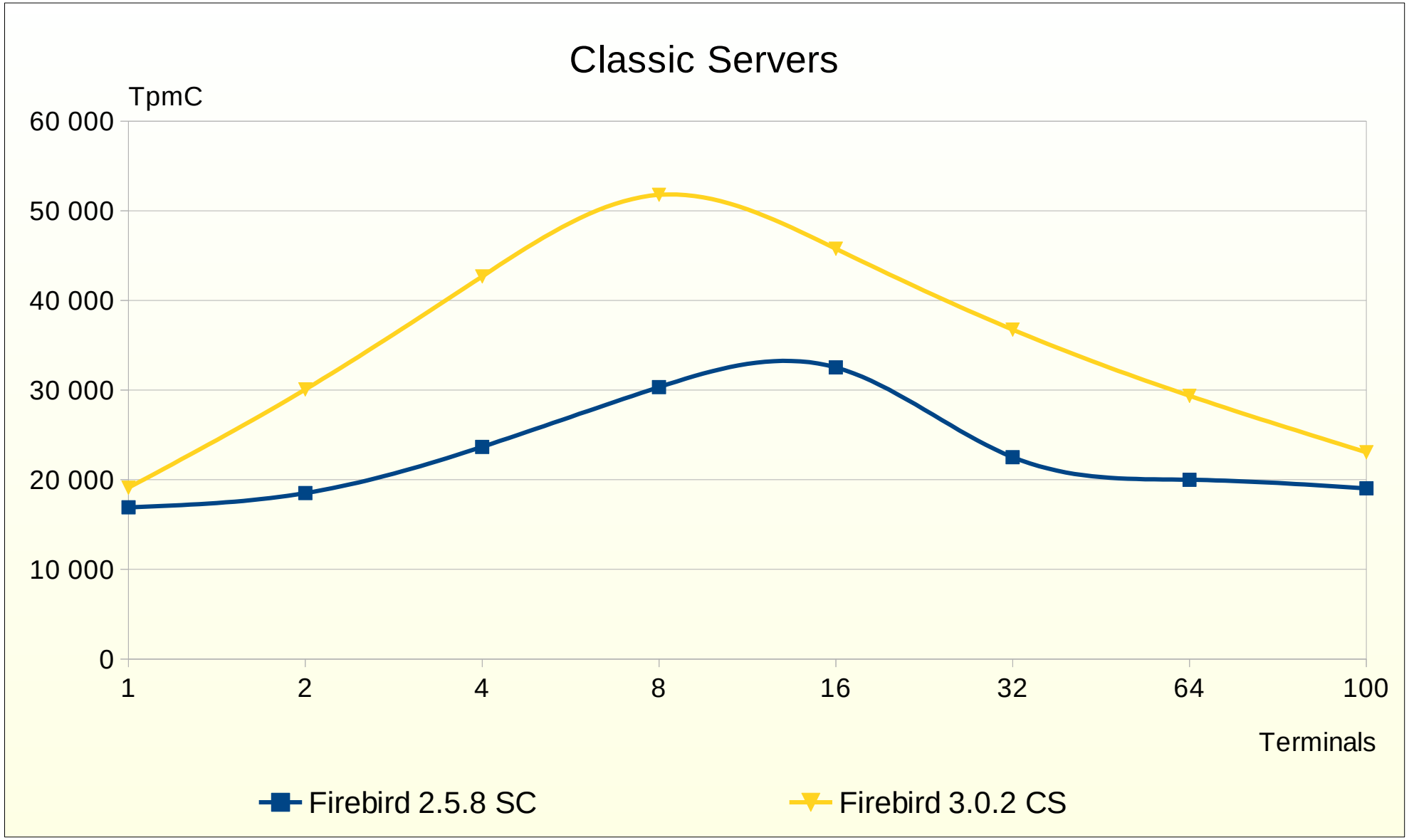
TPCC benchmark

- What Firebird architectures to test?
  - Firebird 2.5.8 – base for comparison, choose better one
    - Super Server
      - no, it doesn’t scale well
    - Classic Server or Super Classic
      - Super Classic is faster
  - Firebird 3.0.2
    - Super
      - yes, sure?
    - Classic
      - yes, lets compare it with 2.5 Classic Server
    - SuperClassic
      - no, it have no benefits in v3 as it was in v2.5

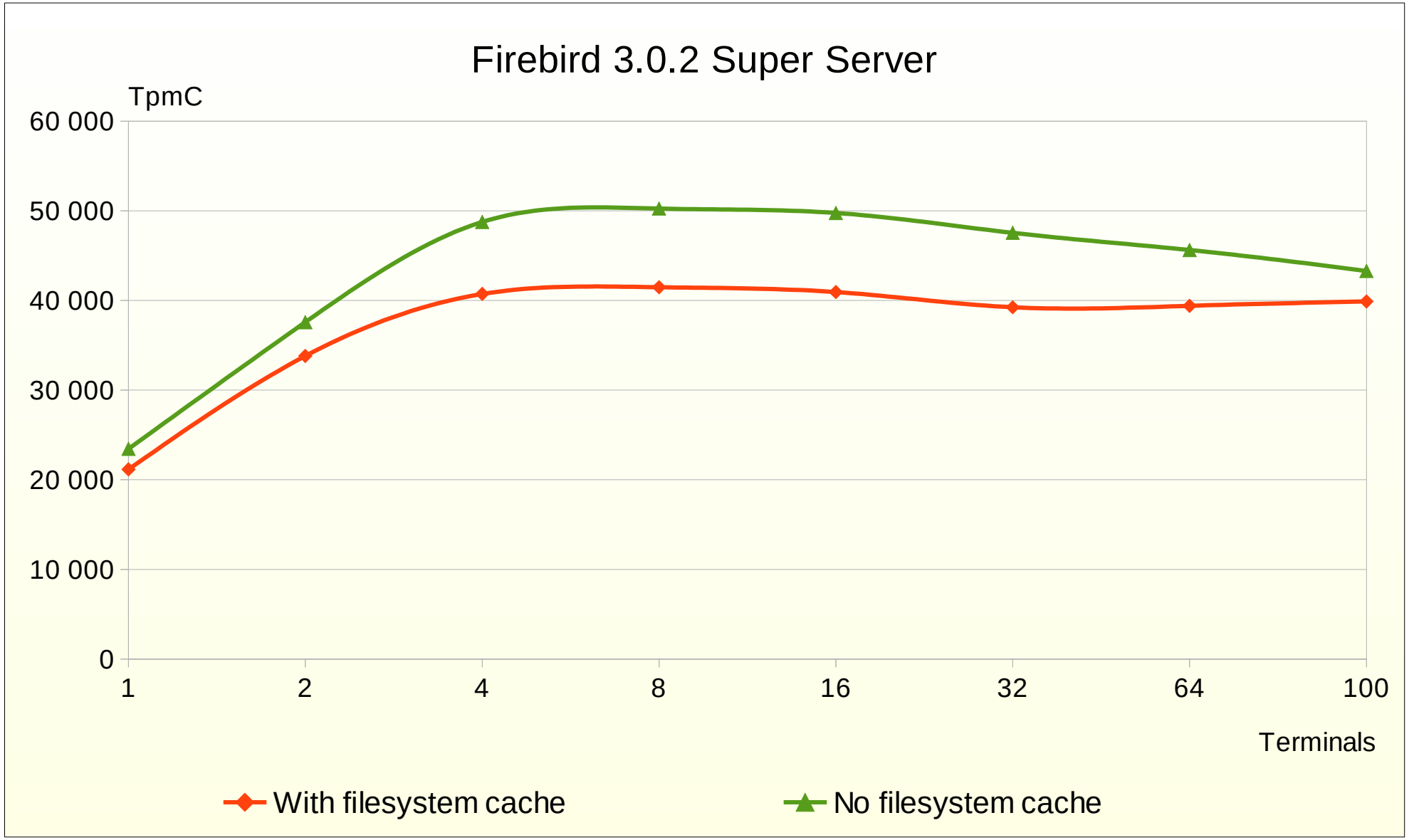
TPCC benchmark

- Firebird configuration
  - Local protocol (XNET)
  - Super Classic 2.5.8 and Classic Server 3.0.2
    - Page cache = 1 024
  - Super Server 3.0.2
    - Page cache = 524 288 (4GB)
  - FileSystemCacheThreshold = 1 073 741 824 (8TB)
    - *My mistake, planned to set it to 1 048 576 pages (8GB), but…*
      … it not changes final results
  - TempCacheLimit = 1 073 741 824 (1GB)
  - LockHashSlots = 22 111
  - GCPolicy = cooperative
    - cooperative is best choice for TPCC load

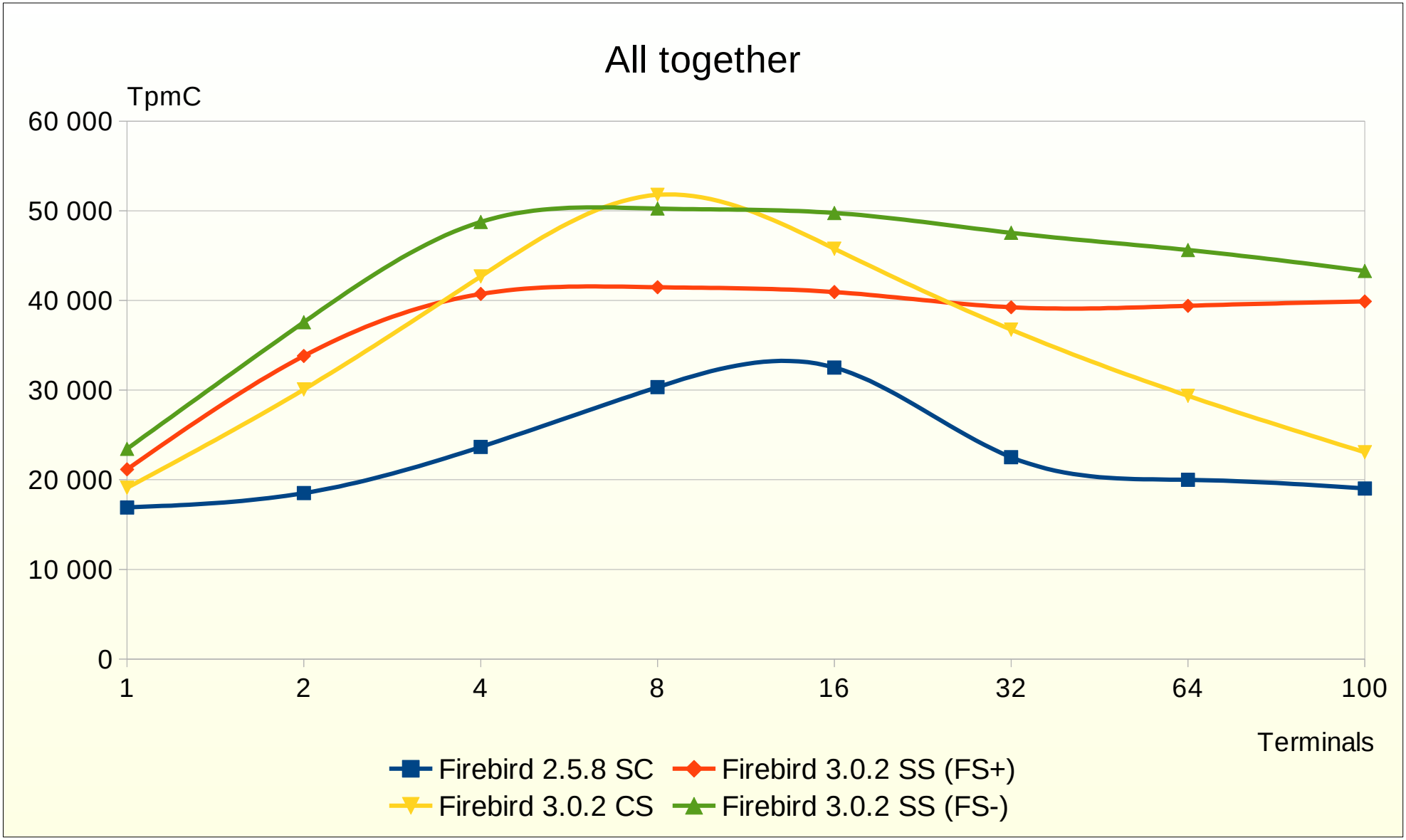
TPCC, scaling



TPCC, scaling



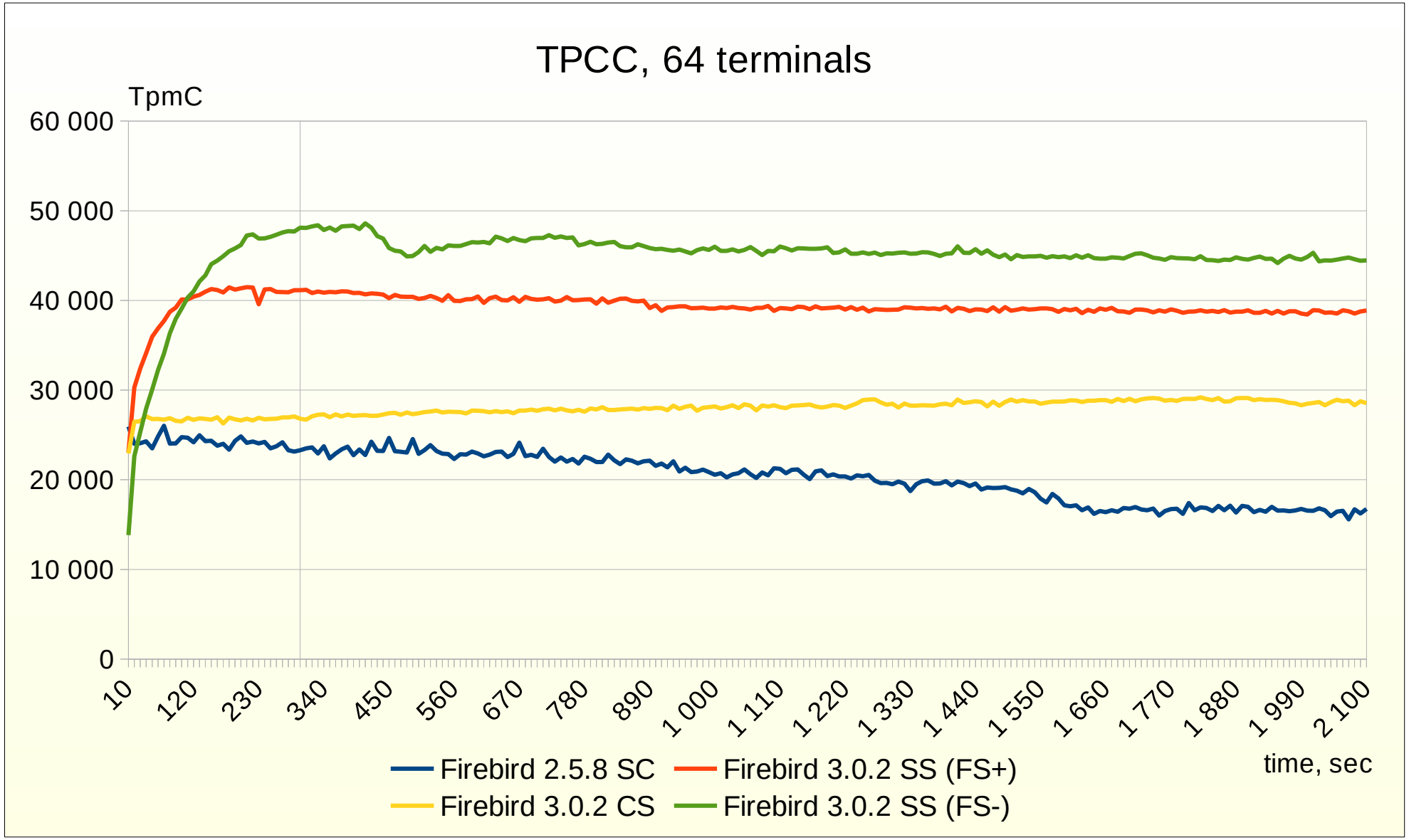
TPCC, scaling



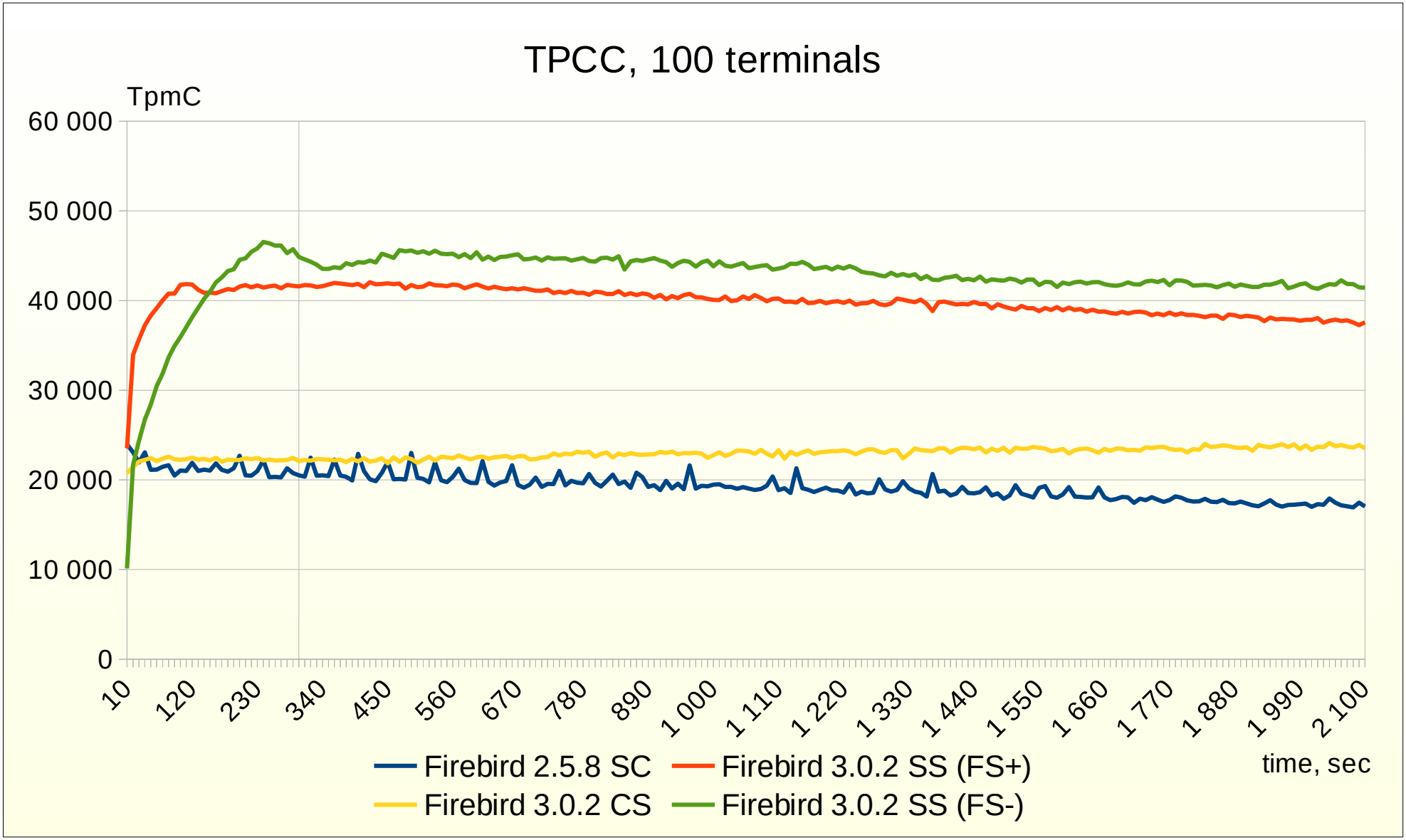
TPCC

- Scalability is very important factor, but not the only one
- Good database engine should also
  - Avoid performance degradation over time
  - Deliver stable performance (with small or no fluctuations)
  - Ensure equal performance for every connection with the same queries
  - etc

TPCC, load over time



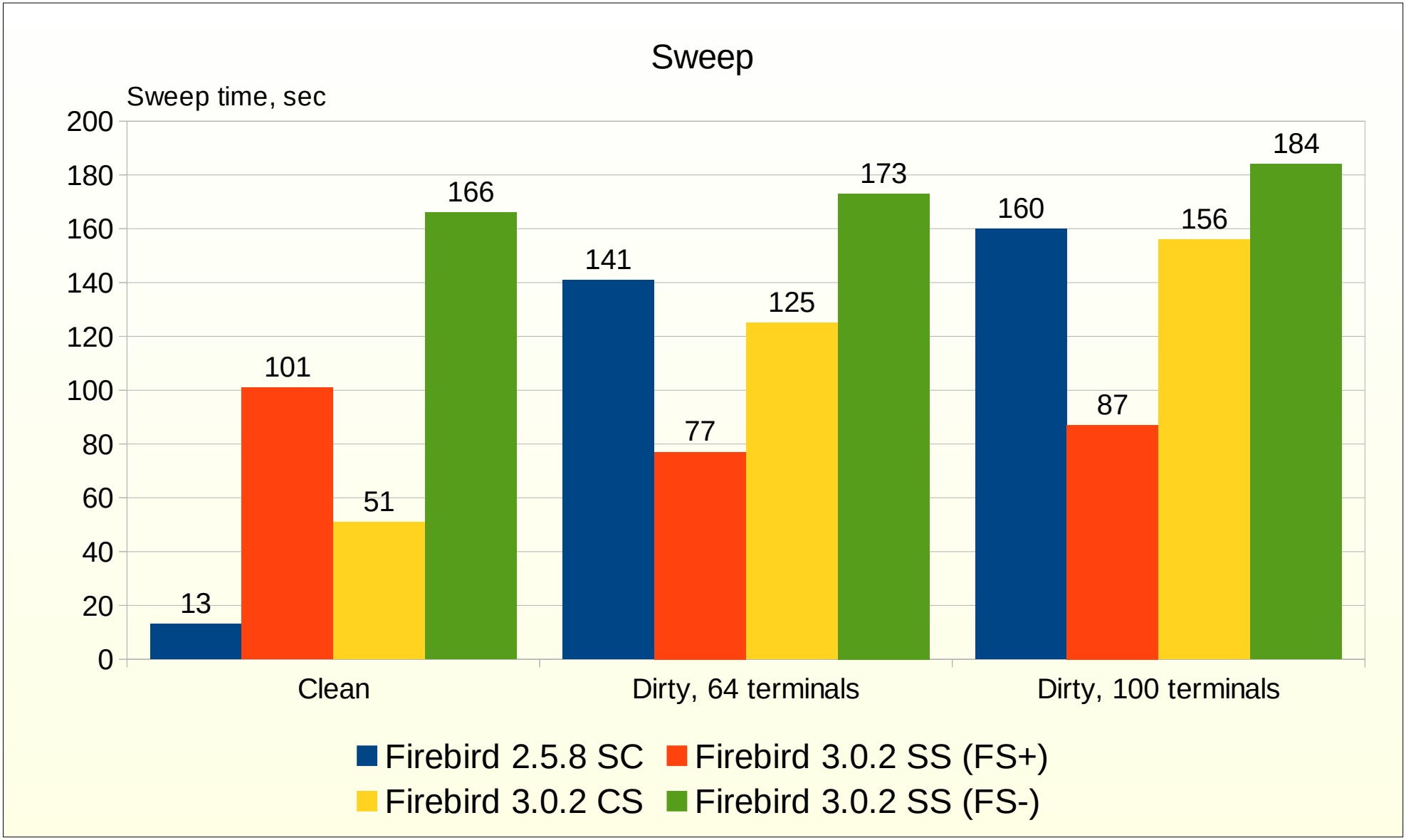
TPCC, load over time



TPCC and Sweep

- Test A
  - Goal: evaluate “pure” sweep performance
    - Run sweep on clean database
    - Run TPCC test
    - Run sweep on “dirty” database

TPCC and Sweep



TPCC and Sweep

- Why sweep in Firebird 3 is much slower than Firebird 2.5 on clean database?
  - Firebird 2.5 just read database, no writes happens
    - Database file is fully cached by OS
  - Firebird 3 set ‘swept’ flag on every Data Page when run first time after data is loaded
    - After ‘swept’ flag is set next sweep run in near zero time

TPCC and Sweep

- Why v3 in Super mode sweeps clean database two times longer than v3 in Classic mode?
  - When Firebird allocates page cache (4GB) OS take some memory from fully cached database file.
    Thus Firebird needs to read pages from disk, not from the file system cache.
  - When database cache was reset to default 2048 pages, Super run sweep in the same 50 seconds as Classic
  - Flash algorithm works sub-optimal with a huge cache used in Super mode
    - To be fixed in Firebird 3.0.3
    - Quick fix reduced sweep time from 110 to 80 sec

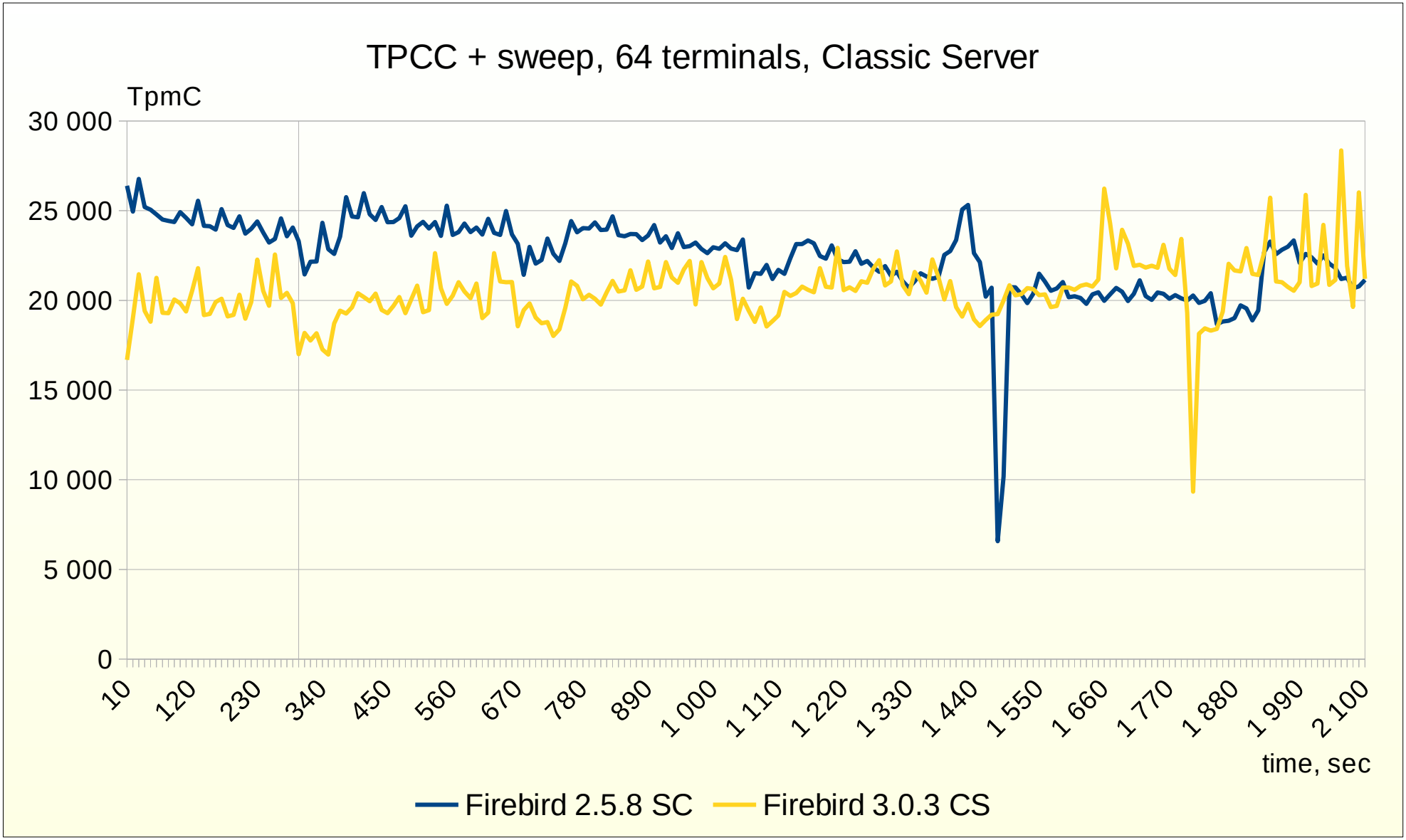
TPCC and Sweep

- Test B
  - Goal: evaluate engine and sweep performance with high load
    - Run sweep on clean database
    - Run TPCC test
      - Run sweep every 5 minutes
    - Run sweep on “dirty” database

TPCC and Sweep

- Firebird 3.0.2
  - Problems found
    - Sweep under load in Classic mode is very slow, looks like lock starvation problem.
    - Quick fix is ready, but issue requires more investigations.
    - Test B uses Firebird 3.0.3 in Classic mode with quick fix.
      - With fix it runs a bit slower

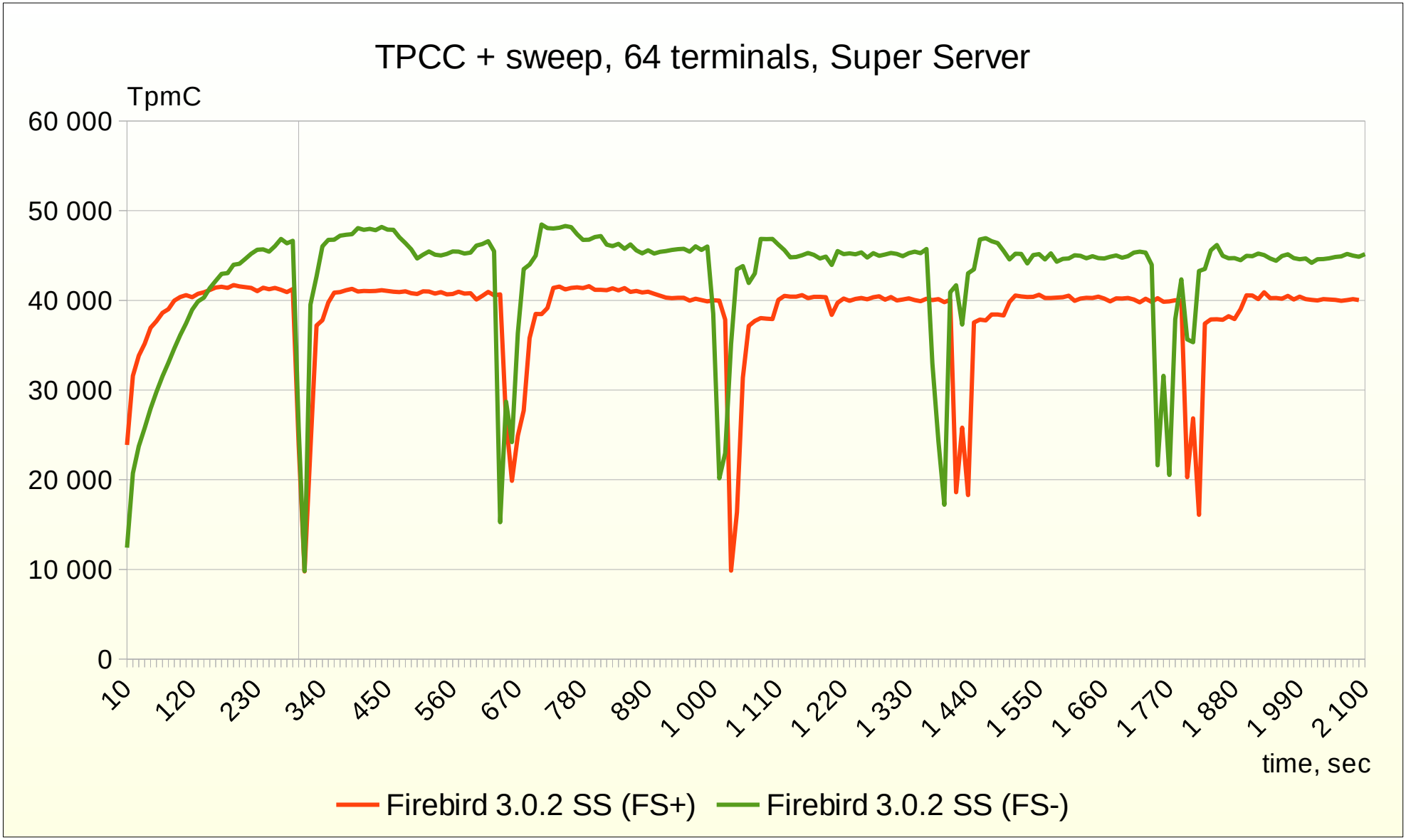
TPCC and sweep, load over time



TPCC and sweep, load over time

- Firebird in Classic mode
  - Affected slightly by the sweep
  - Performance drops about 10% and restored after the sweep

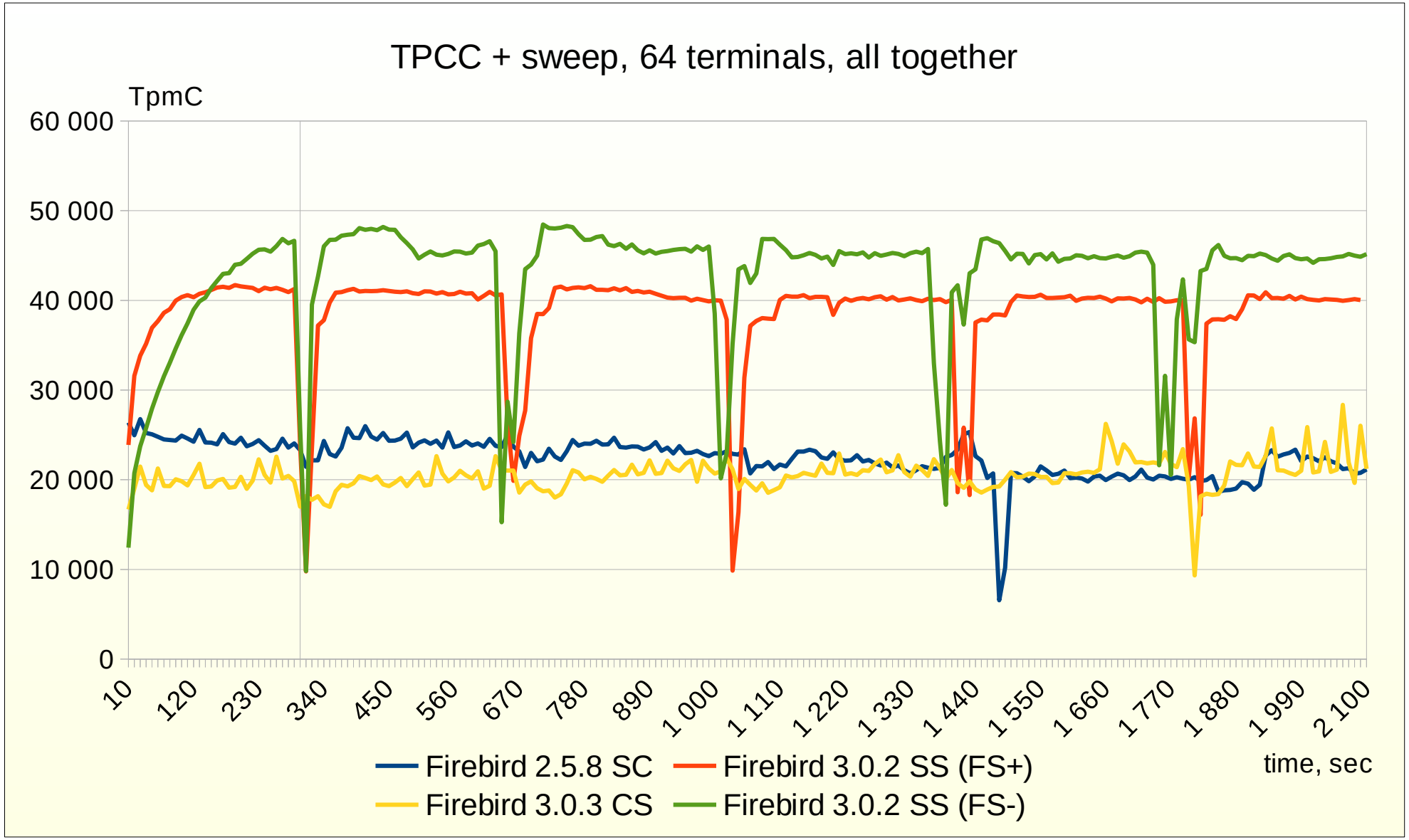
TPCC and sweep, load over time



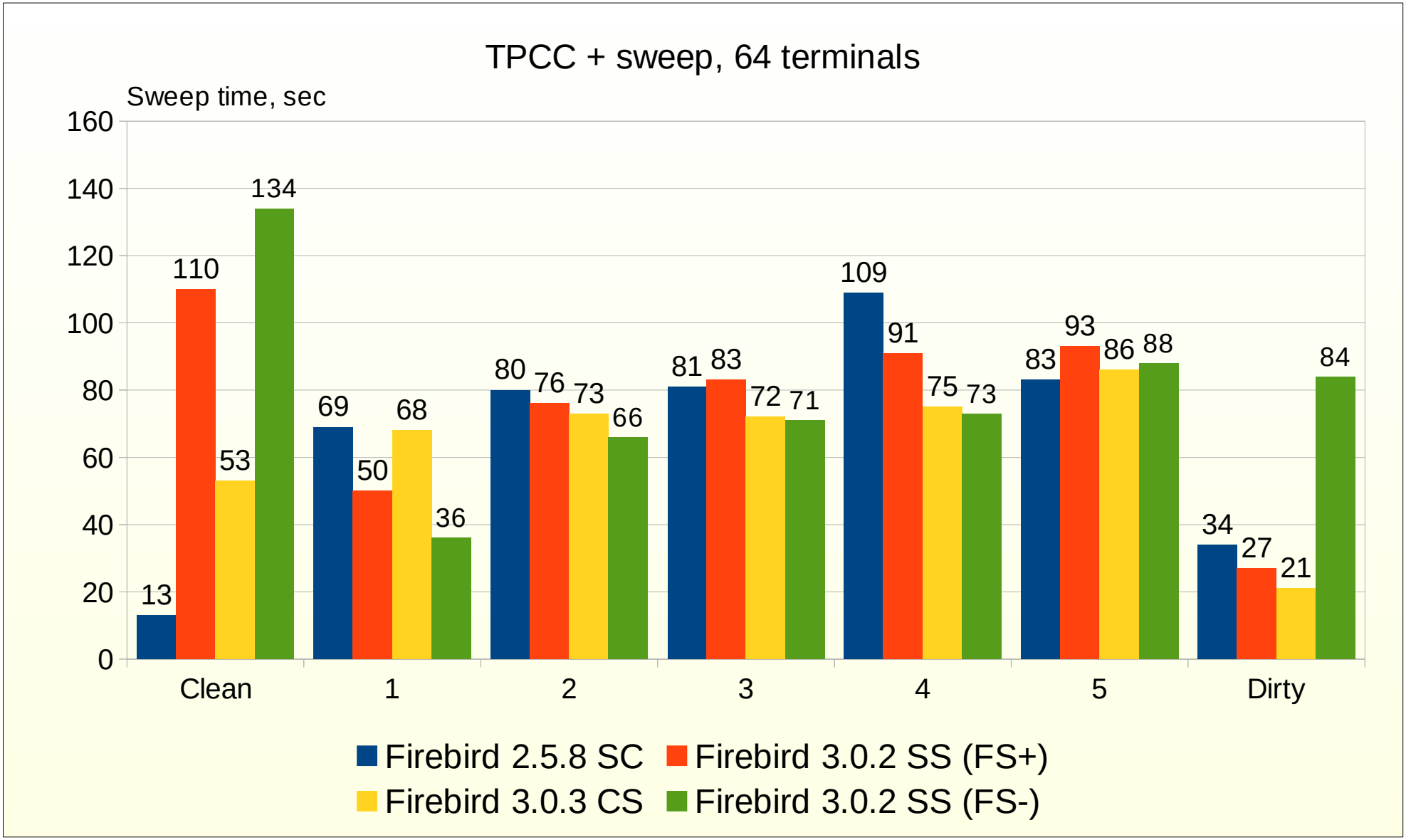
TPCC and sweep, load over time

- Firebird in Super mode
  - Performance drops significantly when sweep starts
    - Should be investigated
  - Performance restored relatively quickly (20-30 sec), much before sweep end (50-90 sec)

TPCC and sweep, load over time



TPCC and sweep, load over time



TPCC and sweep, load over time

- Firebird 3 run first sweep pass slow as it requires to mark all Data Pages as swept
- On heavy loaded system sweep run faster in Firebird 3
- Super mode with huge own cache and file system support
  - sweep time greater than Classic mode, but note – Super make >70% more transactions and produces more work for sweep to do
- Super mode with huge own cache and with no file system support
  - Runs faster under load
  - Slow when Firebird cache is empty
    - It is expected

TPCC and nbackup

- Test
  - Goal: evaluate engine and nbackup performance with high load
    - Run TPCC test
      - After warm-up period run nbackup level 0
      - Every 5 minutes run nbackup and increment backup level
  - All backup files are stored on separate from main database disk drive

TPCC and nbackup

- File system cache
  - nbackup utility reads database file itself, i.e. not using Firebird page cache
  - Switch -DIRECT
    - ON – don’t use file system cache, default on Windows
    - OFF – use file system cache, default on Linux
  - When cached file opens without buffering, Windows removes it from file system cache immediately
  - With default settings, nbackup usage will remove database file from file system cache (on Windows)

TPCC and nbackup

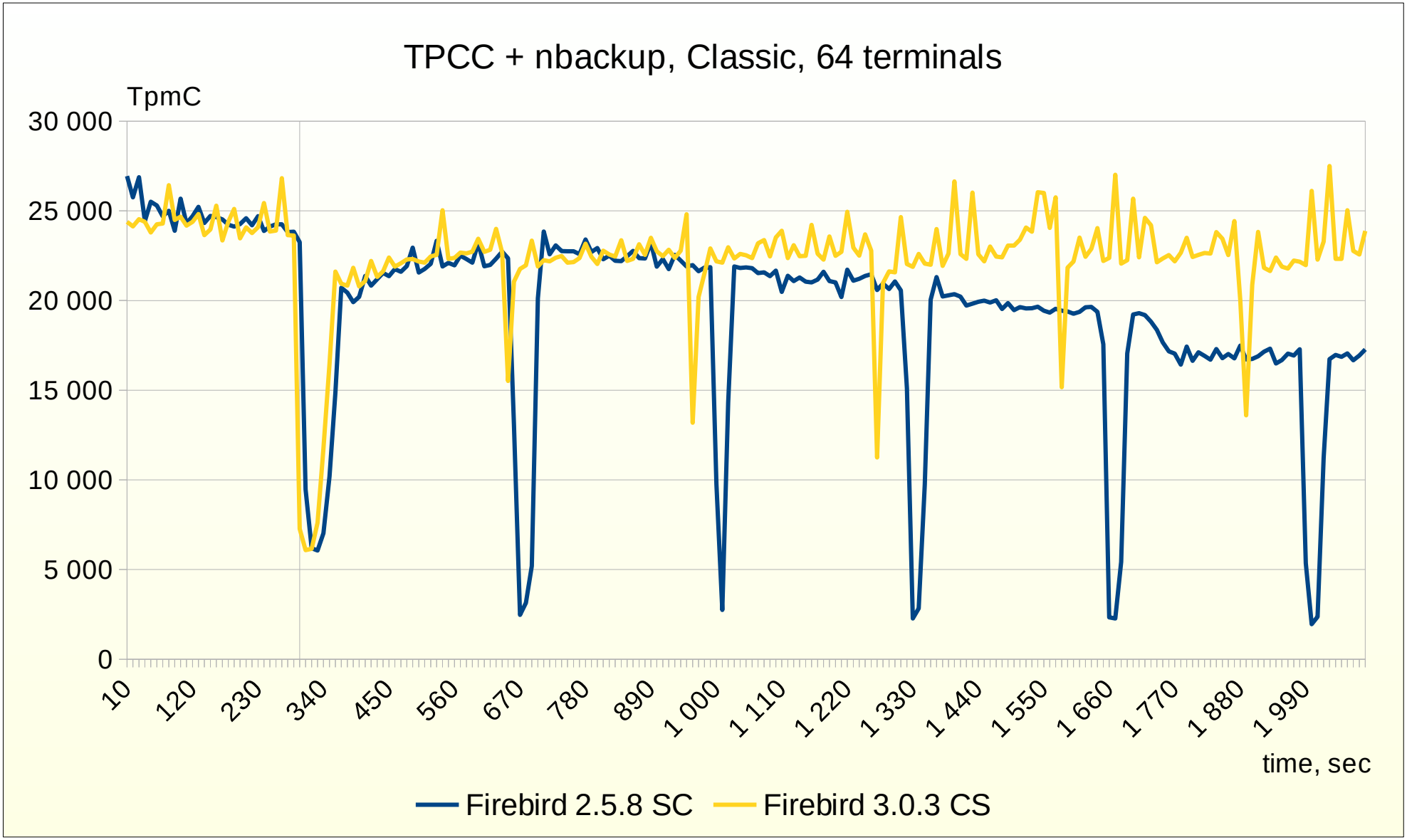
- File system cache
- We run nbackup with
  - -DIRECT OFF – for Classics and for Super, when it run with support of file system cache
  - -DIRECT ON – for Super, when it run with no support of file system cache

	Firebird 2.5.8 SC	Firebird 3.0.3 SS (FS+)	Firebird 3.0.3 CS	Firebird 3.0.3 SS (FS-)
File system cache	used	used	used	not used
-DIRECT	OFF	OFF	OFF	ON

TPCC and nbackup

- Firebird 3.0.2
  - Problems found and fixed
    - CORE-5613: SuperServer could hung when changing physical backup state under high load
    - CORE-5614: Physical backup merge stage could run too long, especially with huge page cache
  - TPCC and nbackup tests done using current snapshot build of Firebird 3.0.3

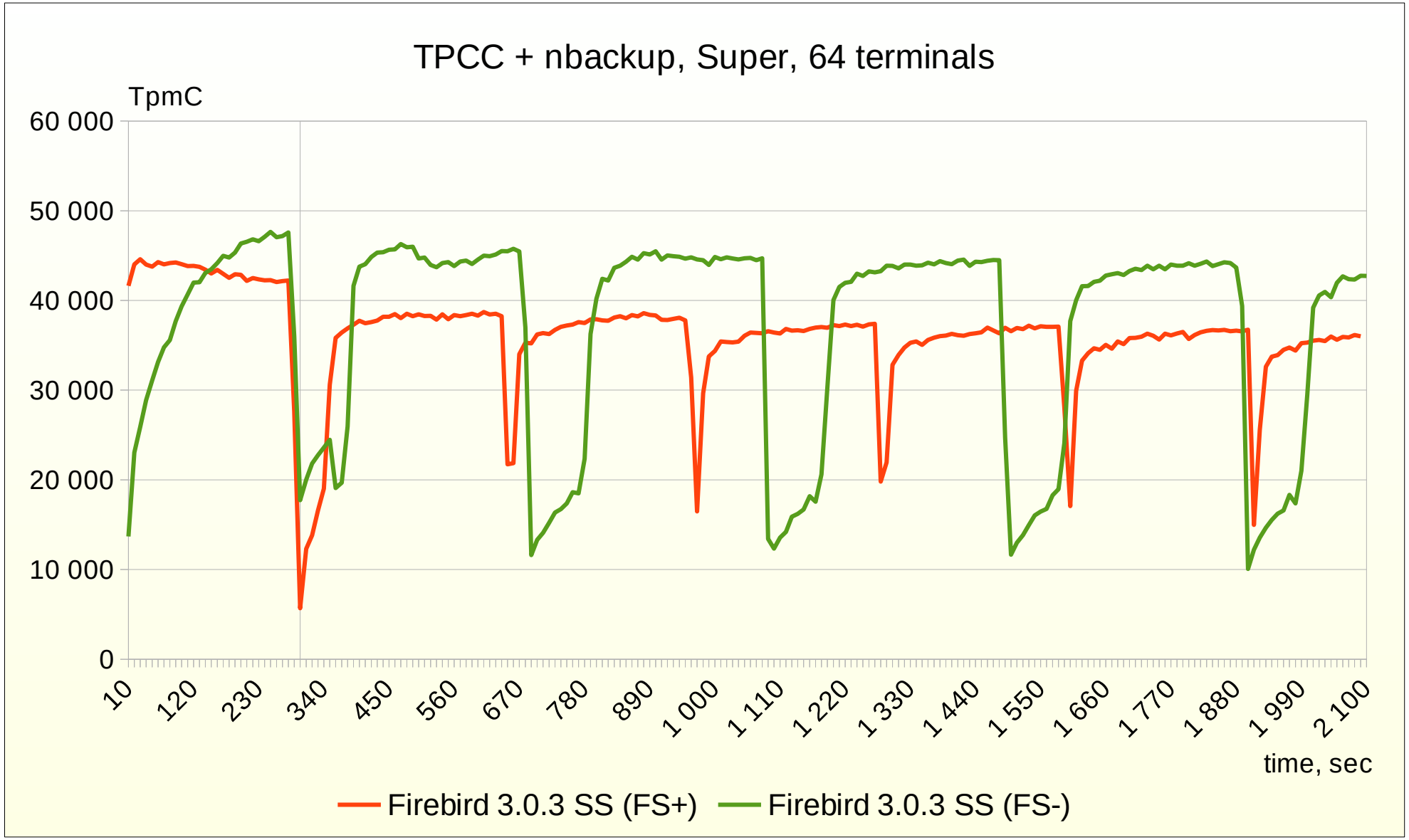
TPCC and nbackup



TPCC and nbackup

- Classic v2.5 vs v3
  - Firebird performance drops significantly when physical backup run
  - Firebird 3 suffers in much less degree than Firebird 2.5

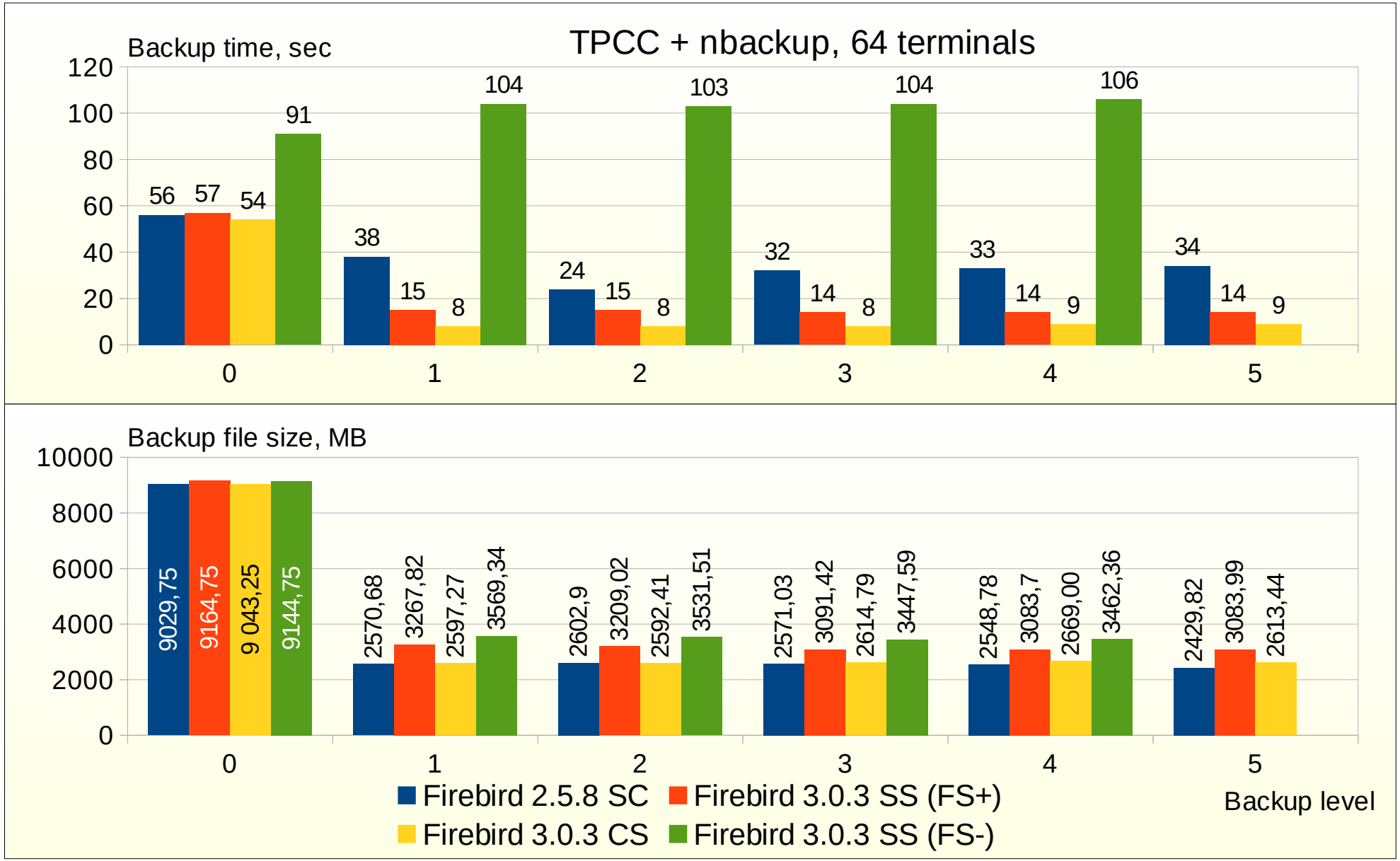
TPCC and nbackup



TPCC and nbackup

- Super v3: with\without file system cache
  - File system cache support helps physical backup to run faster
  - With file system cache performance drops less
  - Even without file system cache performance is better than in Classic v2.5

TPCC and nbackup



Firebird 4

- Performance features in Firebird 4
  - New Batch API
  - Pool for external connections of EXECUTE STATEMENT
    - Implementation available in HQbird
  - Server-side cache of SQL statements
    - Implementation available in HQbird
  - Garbage collection of intermediate record versions

Firebird 4

- New Batch API
  - Send single query with multiply set of params
    - Including BLOB's
- Expected benefits
  - Radical less number of network roundtrips
- Usage
  - Import tools
  - gbak restore
- Current availability
  - Separate branch 'batch' in Firebird source tree on GitHub

Firebird 4

- External connections pool
  - EXECUTE STATEMENT …
    ON EXTERNAL DATA SOURCE
  - Put unused external connection into pool
  - Lookup for the new external connection at pool
    - Check for
      - connection string, user name, role, password
        - Hash used for fast search
      - connection liveness
        - Broken connection silently removed from pool

Firebird 4

- External connections pool, properties
  - Scope : single pool instance per server process
    - Super and SuperClassic modes :
      all local connections uses same pool of external connections
      despite of local database
    - Classic mode :
      each Firebird process contains own pool
  - Common for all external databases
    - To be discussed

Firebird 4

- External connections pool, properties
  - Limited number of inactive (unused) external connections
    - Use LRU algorithm when pool is full
    - Limit could be changed by DBA
  - Limited lifetime of inactive external connection
    - Free inactive external connections after specified time
    - Lifetime could be changed by DBA

Firebird 4

- External connections pool, management
  - Manage pool properties using new DDL-like statement
    - ALTER EXTERNAL CONNECTIONS POOL
      - SET SIZE
      - SET LIFETIME
      - CLEAR ALL
      - CLEAR OLDEST
  - *It could be changed in release version?*

Firebird 4

- External connections pool, management
  - Query pool state with RDB$GET_CONTEXT()
    - Namespace : SYSTEM
    - Variables :
      - EXT_CONN_POOL_SIZE
      - EXT_CONN_POOL_IDLE_COUNT
      - EXT_CONN_POOL_ACTIVE_COUNT
      - EXT_CONN_POOL_LIFETIME
  - Support in monitoring tables to be discussed

Firebird 4

- External connections pool
- Benefits
  - Radical less connection time when using EXECUTE STATEMENT with external databases
  - Much lower network and CPU load
- Usage
  - Any application intensively works with external queries
- Current availability
  - HQBird 2.5
  - Firebird 4 (soon)

Firebird 4

- Server-side cache of SQL statements
  - Put unused SQL statement into statements cache
    - isc_dsql_free_statement(..., DSQL_drop)
    - Release of last reference on IStatement or IResultSet
  - Lookup cache when preparing new SQL statement
    - Explicit prepare
      - isc_dsql_prepare()
      - IAttachment::prepare()
    - Implicit prepare
      - isc_dsql_execute_immediate()
      - IAttachment::execute(), IAttachment::openCursor()
    - Check for
      - Hash of SQL statement text
      - SQL statement text itself

Firebird 4

- SQL statements cache, properties
  - Scope
    - Attachment
      - Each attachment have its own cache of SQL statements
  - Limits
    - Number of statements in cache
      - Set using firebird.conf
      - *Could be changed in Firebird release version?*
    - DML statements only
      - No sense to cache DDL statements

Firebird 4

- SQL statements cache
  - LRU algorithm defines which statement to keep in cache
    - Freed by application statement put into head of LRU list
    - If cache already full, statement from the tail of the LRU list is removed and released
    - It allows to keep in cache statements that really often used by application
  - To be discussed before include into Firebird:
    - Control memory usage not number of statements?
    - Manage limits in run-time
    - Support in monitoring tables

Firebird 4

- SQL statements cache
- Benefits
  - ‘Instant’ time of prepare of most often used SQL statements
  - Much lower network and CPU load
- Use cases
  - Application that not re-used prepared statements by itself
  - Application that can’t use prepared statements
    - Connections pool at client side often makes it impossible
      - Place to improve connections pool at client side?
- Current availability
  - HQBird 3

Firebird 4

- Garbage collection of intermediate record versions
  - Allows to clean up “inner” versions at version chain
- Benefits
  - Keep record versions chains short, despite of long running transactions
  - Performance stability less affected by apps with “bad transactions management”
- Usage
  - All kind of applications
- Current availability
  - Branch “read_consistency” at redsoftbiz/firebird fork:

    https://github.com/redsoftbiz/firebird.git
  - Could be included into v4 beta1

THANK YOU FOR ATTENTION

Questions?

Firebird official web site

Firebird tracker

hvlad@users.sf.net