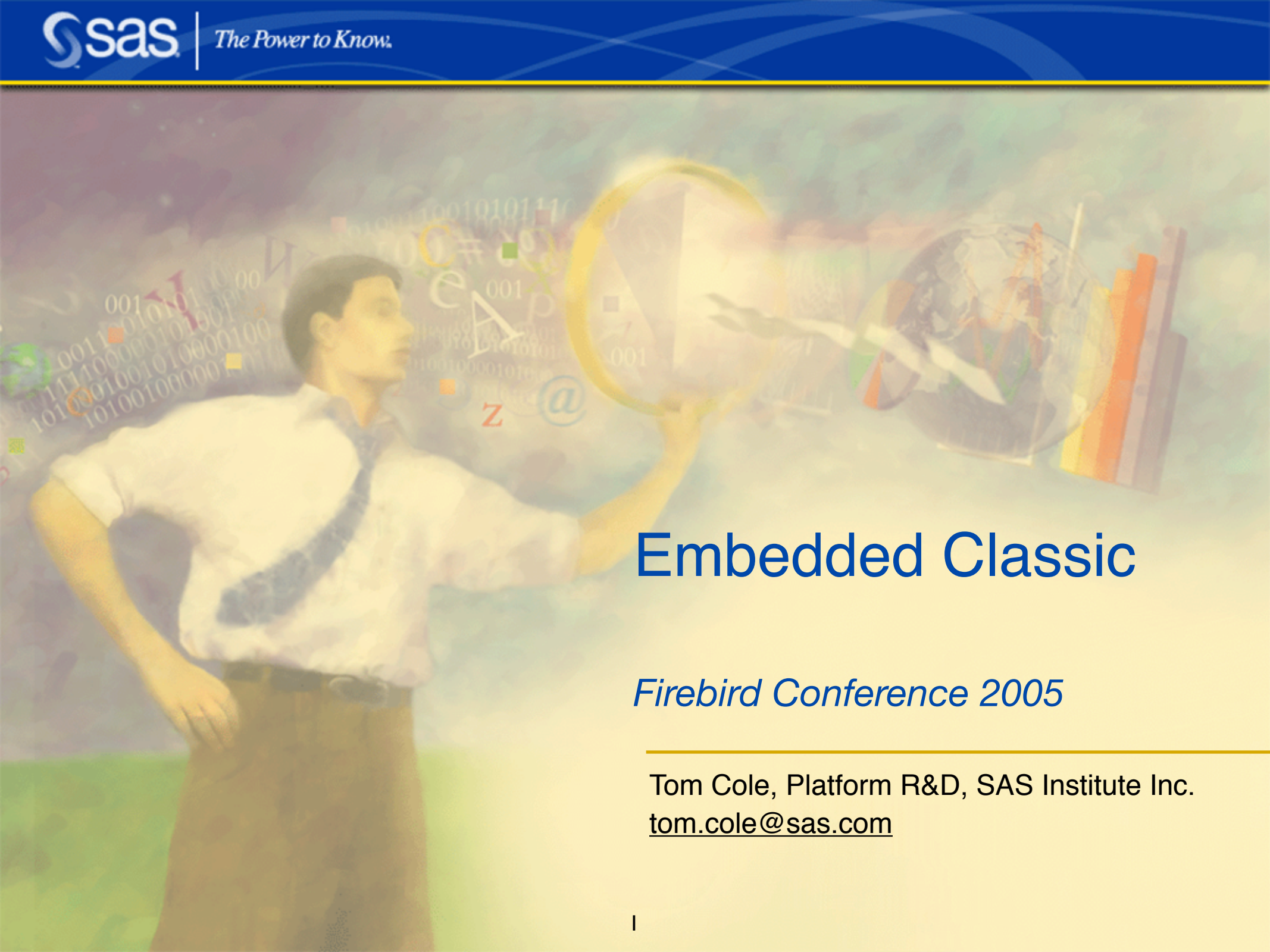

Embedded Classic

Firebird Conference 2005

Tom Cole, Platform R&D, SAS Institute Inc.
tom.cole@sas.com

Embedded Classic?

- Yes, I know. Bad title.
- It should have been called “A Vulcan Implementation Modeled On Firebird Classic, Running in an Embedded Mode.”
- This paper is about design trade-offs in how SAS Institute is using Vulcan to support *scalability*.

Firebird Modes

- Classic - Each connection forks a new process for each connection to the database. Each connection has it's own cache. Database sharing accomplished with file system lock semantics.
- Super Server - Each connection has a separate thread. All connections share the same cache. Explicit thread scheduling allows threads to run when it is "safe" to do so. Database cannot be shared across processes.

Firebird Modes

- Client/Server mode supports both Classic and Super Server models.
- Embedded mode only supports the Super Server model.
- SAS' interest is in using Firebird/Vulcan in embedded mode to support our Table Services/Table Server.

Two Caching Modes

- Vulcan was designed to unify the Firebird modes into a single consistent model.
- The Vulcan code base as originally developed by Jim Starkey did not have “caching modes” in the build styles.
- SAS added the SHARED_CACHE symbol which lets us build the product one of two ways to support different degrees of sharing and locking.

SHARED_CACHE (Vulcan Default)

- SHARED_CACHE is the default Vulcan build mode.
- The threading is most like Super Server, in that each connection runs on a separate thread.
- All threads share a single cache.
- No explicit thread scheduling; uses native threads.
- Lock Manager modified to support both in-process threads and connections from other processes.

Challenges with SHARED_CACHE

- Locking performance issues on SMP machines. Even though locks are done in few instructions, they happen so often that intra-core synchronization stalls the SMP system too often.
- This impacted SMP scalability dramatically.
- For example, in best-case read-only tests, we could only get 1.8x improvement over Firebird 1.5 on a 4-way system.

Challenges with SHARED_CACHE

- We were never able to achieve long-running stability. We require that the server be able to run for a week under characteristic loads, and the Vulcan code base would not run more than a day.
- We believe this is not necessarily inherent in the nature of the cache sharing and locking, but was a major factor in exploring alternative strategies.
- The growth of the lock table was not thread-safe in that it could be unmapped when a thread was waiting on it. This is a side-effect of some platform implementations of memory mapped storage.

(NO) SHARED_CACHE

- Most closely resembles Classic, in that each connection runs on a separate thread, but has it's own cache, lock manager, etc.
- Conscious trade-off between memory footprint and cache coherency issues versus lock contention
- Performance tests show Vulcan compiled in this mode runs greater than 4x faster than Firebird 1.5 on a 4-way SMP system.
- Vulcan stability tests run > 1 week without failure (we essentially have to stop them to do other work).

More on Tradeoffs

- Cache size per connection must be significantly smaller (we are using 75k currently).
- This affects footprint; a 1MB cache is too large for thousands of connections.
- Depending on usage, a page can often exist in more than one thread's cache
- As a result, maintaining cache coherency is very inefficient for large caches * many clients.

Our Usage Patterns

- Most of our use of Vulcan is largely data-base read operations.
- Write operations occur in bursts for short intervals.
- This makes memory/locking tradeoff practical.
- When writes do occur, performance *can* be slower than it would be with SHARED_CACHE because of the cache invalidation issue.

More on the Lock Manager

- Connections are equal regardless of whether they are inter-thread or inter-process connections.
- Each connection has a separate instance of the lock manager. Each instance has a watcher thread paired with it to support AST-style operations.
- The watcher is synched to the connection by an AST lock. This acts like the thread scheduler in Classic, but only between these two threads.

More on the Lock Manager

- Eliminated lock table growth, and replaced it with lock table extensions. Each extension is fixed in size.
- Segments can be added in a thread-safe manner when needed.
- REL pointer to ABS location calculation slightly more complex to handle multiple extensions of the table.
- Since memory is never unmapped, all reloading of pointers after waits has been removed.

More on the Lock Manager

- This also allows us to use multiple lock files - one per database - instead of a single global lock file.
- Reduces activity to the lock table.
- Reduced system-level contention for connections to different databases accessing same pages of lock table.
- By default lock tables are created with ".LCK" suffix to database name.

More on the Lock Manager

- This lock manager mechanism has proven reliable on virtually all our supported platforms.
- 32-bit and 64-bit Windows on x86, IA64, AMD
- 64-bit HP-UX, Solaris, AIX
- 32-bit and 64-bit Linux on x86, IA64, AMD
- IBM390 USS
- Currently working on Open VMS for Itanium

Other Sharing Issues

- SAS requires that multiple threads be able to use the same connection (with different active statements, etc.)
- However, our requirements allow us to serialize such access, we use AST-style locking for connections to support the multiple threads. Collisions are rare.
- With less sharing between threads, thread-local memory allocations are more useful. Added a mode to memory allocator for this in cases where we know the memory is thread-private.

Customer Profiles - Why It Matters

- Typical customer is running 4-8 way CPU
- Typical customer is running >= 16GB memory
- Typical customer reads far more data than writes.
- Elapsed time windows are immovable resource constraint for enterprise class customers.
- For these customers, spending memory to gain time is a good trade.

Some Performance Data...

16 threads doing reads on a 4-way CPU

	CPU USAGE	MEMORY	ELAPSED TIME
FB 1.5*	25%	8 MB	36 sec
VULCAN: SC	99%	18 M	19 sec
VULCAN: LC	99%	29 M	7 sec

* Firebird 1.5 runs used option to constrain tests to a single CPU due to poor performance on SMP machines with Super Server model.

Future of (NO) SHARED_CACHE?

- "firebird.h" was pushed this summer to SourceForge that can be used to select SHARED_CACHE compilation modes, as well as modifications described in this paper.
- We believe that ultimately, Vulcan's default mode of more aggressive cache sharing will be the more desirable mode. There is still work to be done in the Lock Manager to make this possible.
- We think that disabling SHARED_CACHE is a *bridging strategy* that lets us ship a product based on Vulcan sooner that matches our customer profile.

Q&A?

The Power to Know.®