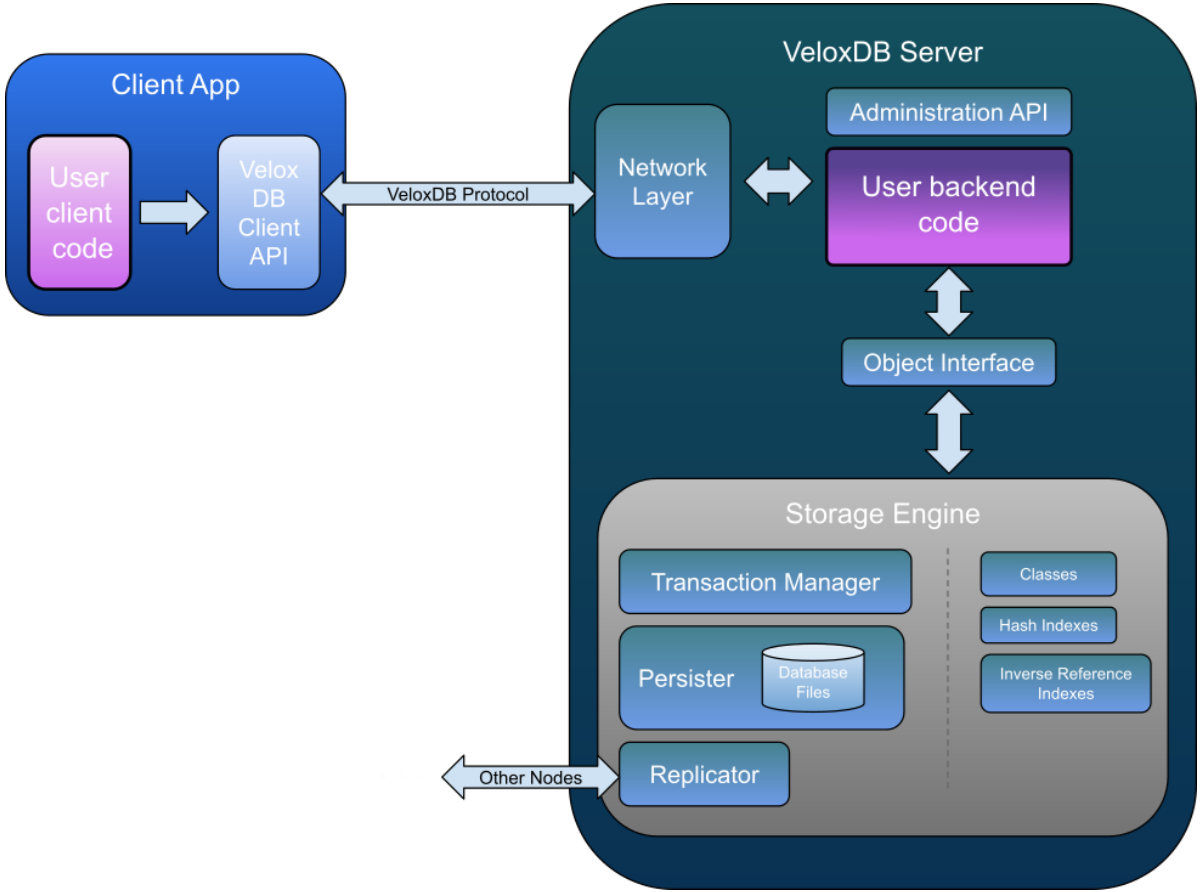
VeloxDB - A Locally Grown Database

Ivan Savu

- Members:
  - Nikola Morača
  - Denis Balog
  - Ivan Savu

- Fast (2.5 million trans/second on AWS c6a.8xlarge)
- ACID
- Object Oriented (ORM as first class citizen)
- .NET based
- Cross platform
- Open Source (MIT license)
- High Availability
- Read scale out
- In - Memory


Client App
User client code
Velox DB Client API
VeloxDB Protocol
VeloxDB Server
Network Layer
Administration API
User backend code
Object Interface
Storage Engine
Transaction Manager
Persister
Database Files
Replicator
Other Nodes
Classes
Hash Indexes
Inverse Reference Indexes

- User provides model as C# abstract classes
- VeloxDB implements abstract classes with code generation at runtime
- Each class has a pointer to unmanaged data.
- Getters are implemented by directly reading data
- Setters accumulate changes for writing to disk and replication

Example model:

```
[DatabaseClass]
public abstract class Post : DatabaseObject
{
    [DatabaseProperty]
    public abstract string Title { get; set; }

    [DatabaseProperty]
    public abstract string Content { get; set; }
}
```

Example database operation:

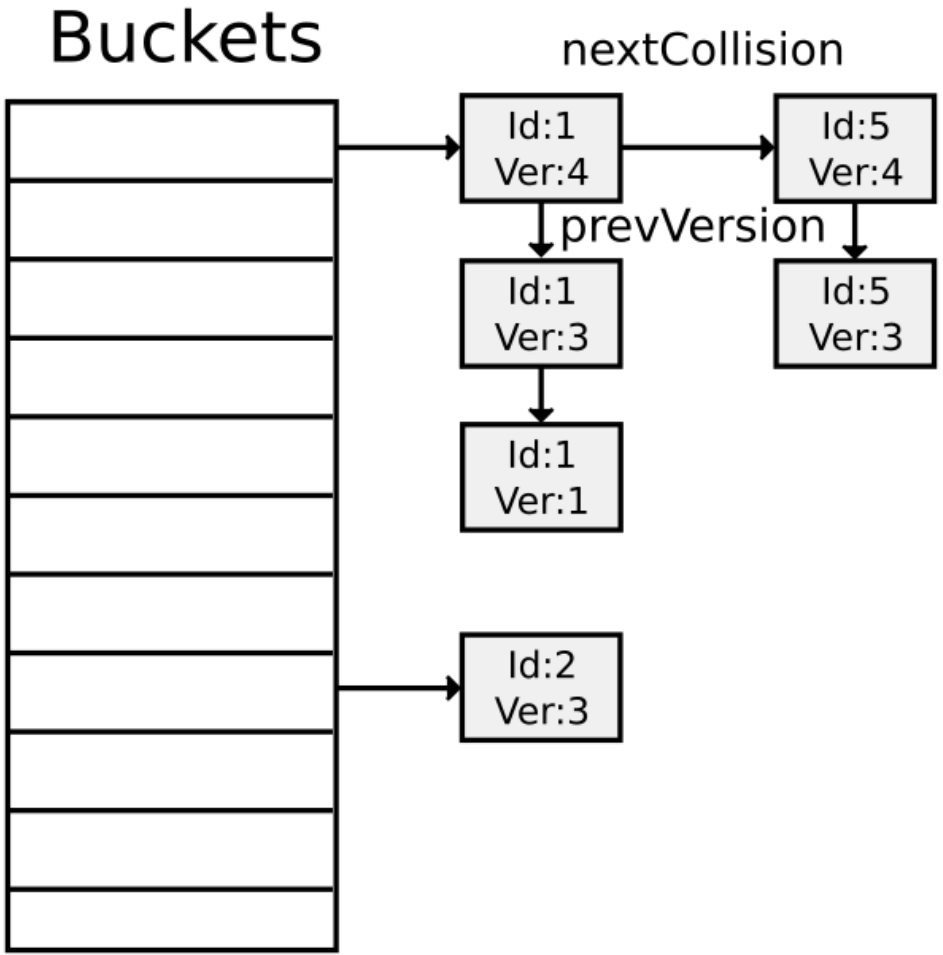
```
[DbAPIOperation]
public long CreatePost(ObjectModel om, PostDTO
post)
{
    Post newPost = om.CreateObject<Post>();
    newPost.Title = post.Title;
```

- Complexity
- Performance problems
- N+1 query problem
- Incomplete abstraction
- Difficult to debug

- Uses Multi Version Concurrency Control (MVCC) for transaction management.
- Handles updates by treating them as inserts with new versions.
- Stores multiple versions of the same object (Multiversioning).

T1: AddBook("The Great Gatsby", "F. Scott Fitzegarld")
T2: AddBook("To Kill a Mockingbird", "Harper Lee")
T3: AddBook("One Hundred Years of Solitude", "Gabriel Garcia Marquez")
T4: UpdateBookAuthor(1, "F. Scott Fitzegarld")

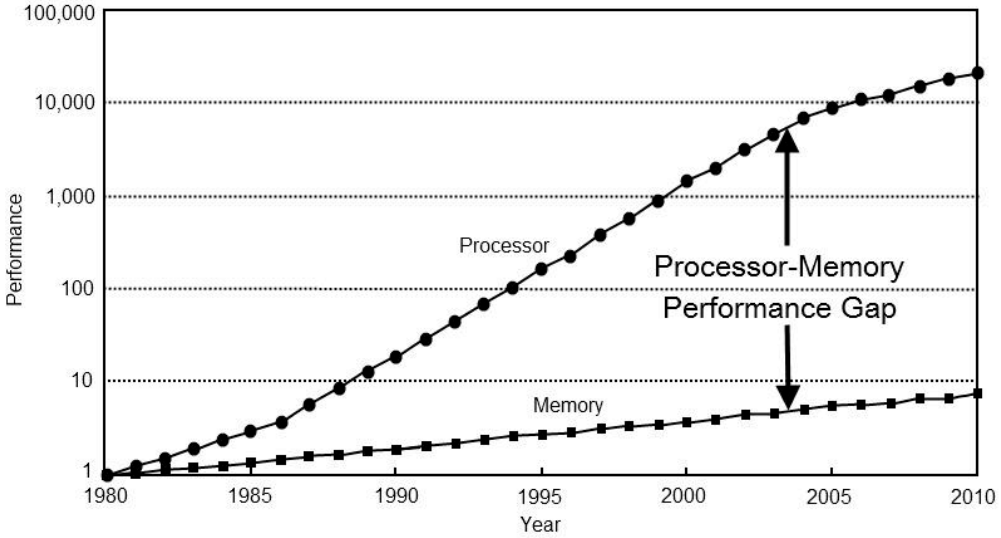
Id	Title	Author	Version
1	The Great Gatsby	F. Scott Fitzegarld	1
1	The Great Gatsby	F. Scott Fitzgerald	4
2	To Kill a Mockingbird	Harper Lee	2
3	One Hundred Years of Solitude	Gabriel Garcia Marquez	3

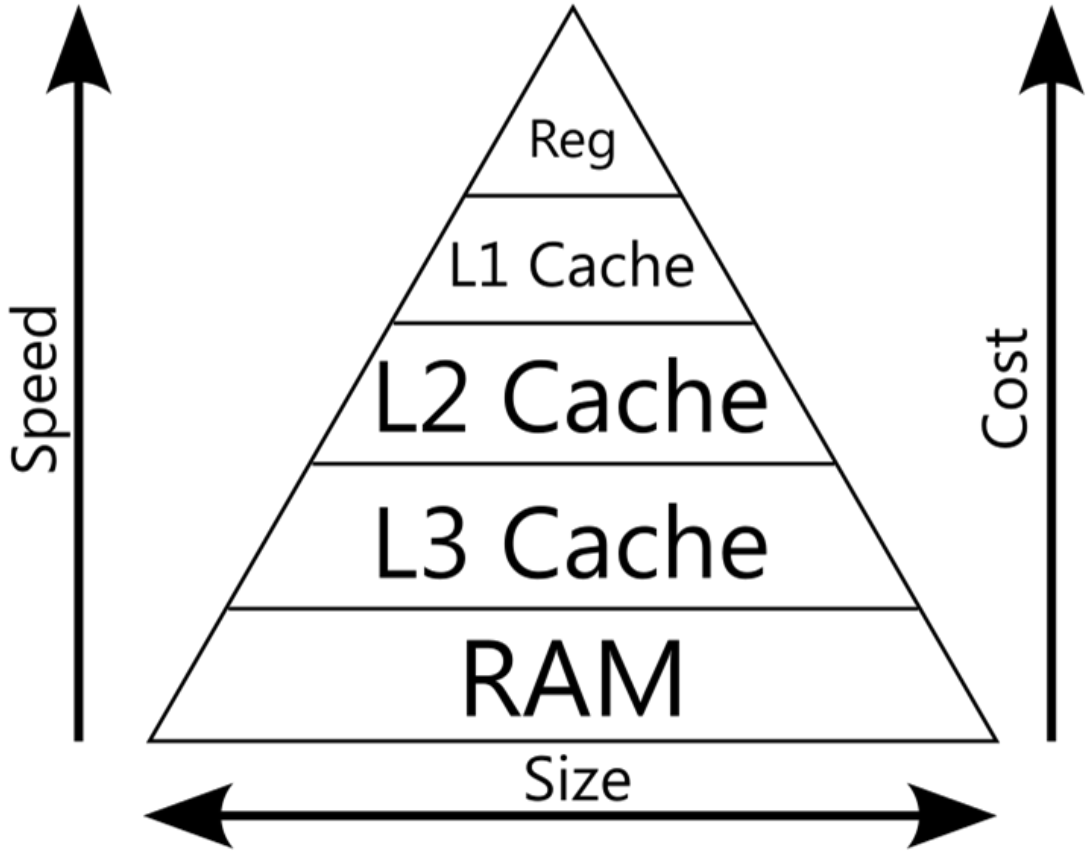

Buckets
nextCollision
Id:1
Ver:4
Id:5
Ver:4
prevVersion
Id:1
Ver:3
Id:5
Ver:3
Id:1
Ver:1
Id:2
Ver:3

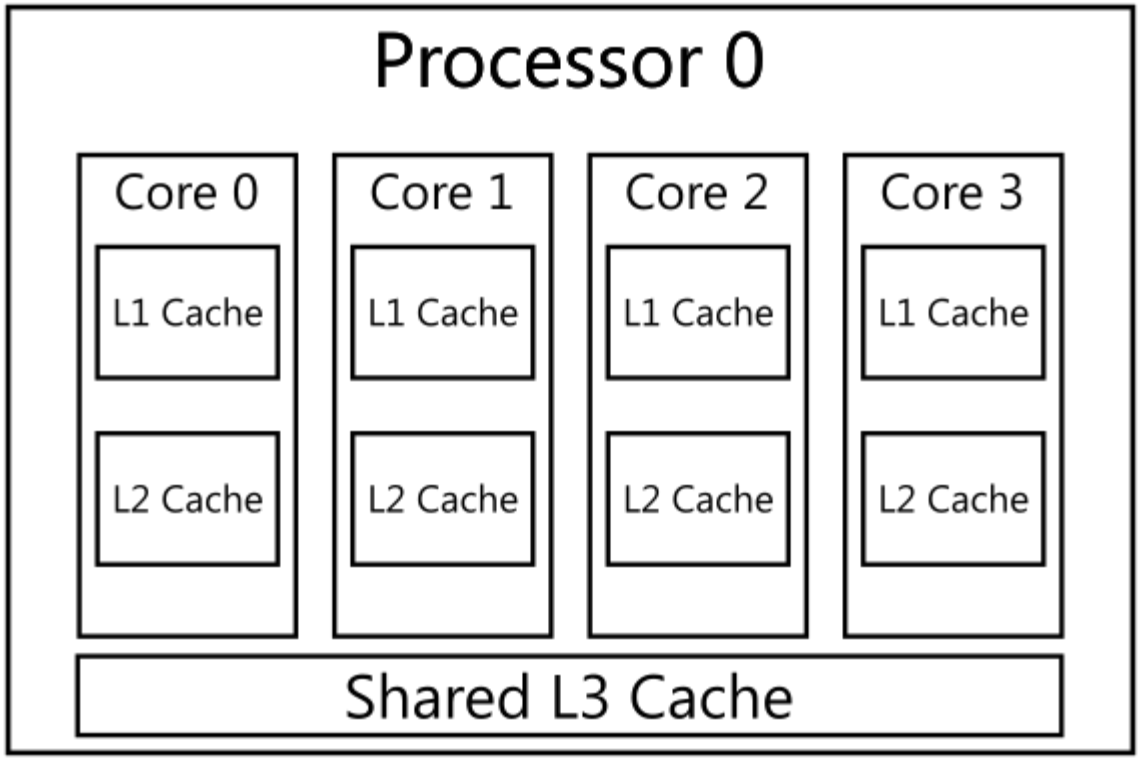
- We use very fine grained locks - employing a significant number of locks
- .NET locks are objects, each object has 24-byte overhead
- Spinlock
  - 1 bit needed
  - Suitable for rare contentions
  - Pure userspace

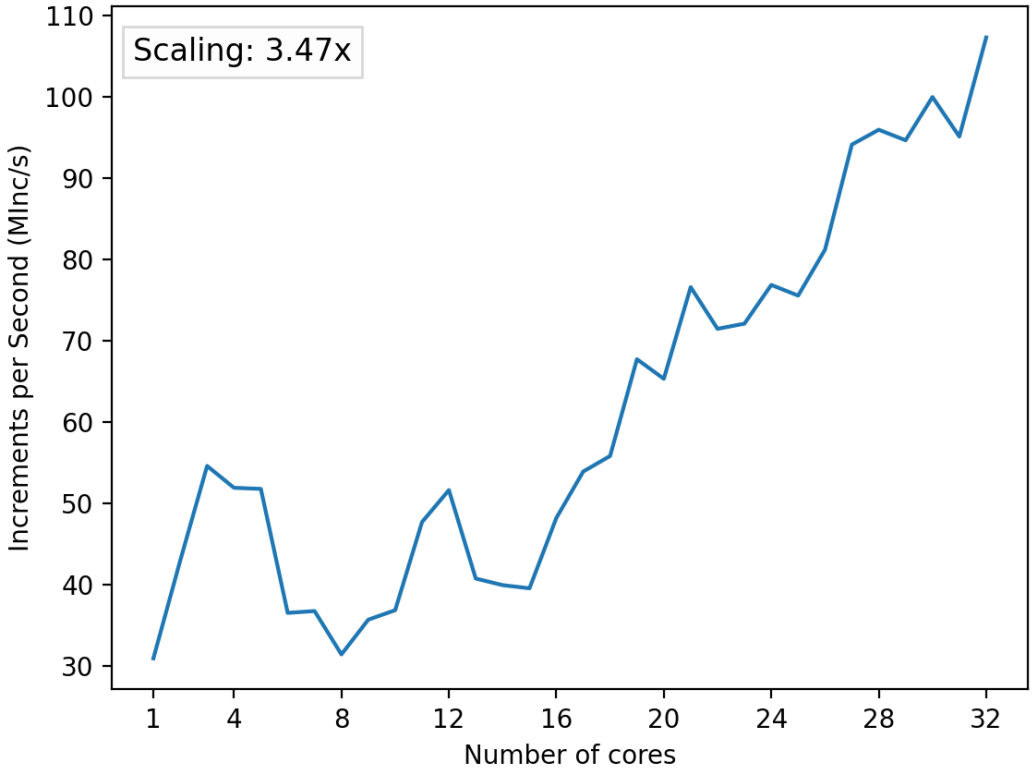
```
void EnterLock()
{
   while(true)
        observedState = state;

        if (GetBit(observedState, LOCK_BIT) != LOCKED)
        {
             // Calculate lock state with lock bit set
            lockedState = SetBit(observedState, LOCK_BIT, LOCKED)
            if (CompareAndExchange(ref state, lockedState, observedState) == observedState):
                return // Lock acquired
        }
}
```

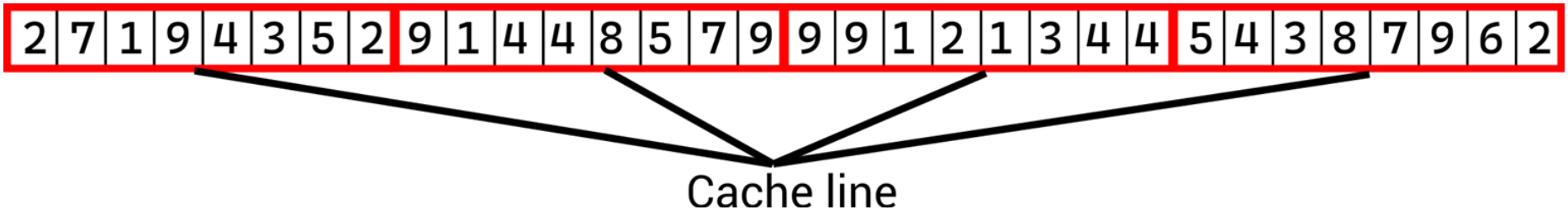
- CPUs are improving at faster rate than memory
- Bottlenecks:
  - Bandwidth
  - Latency

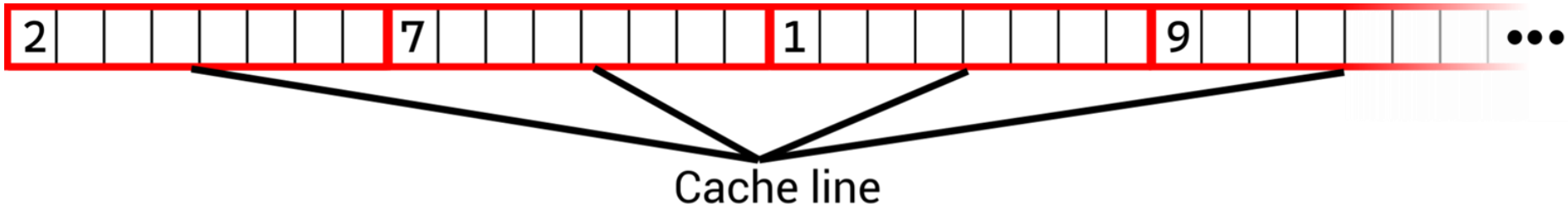


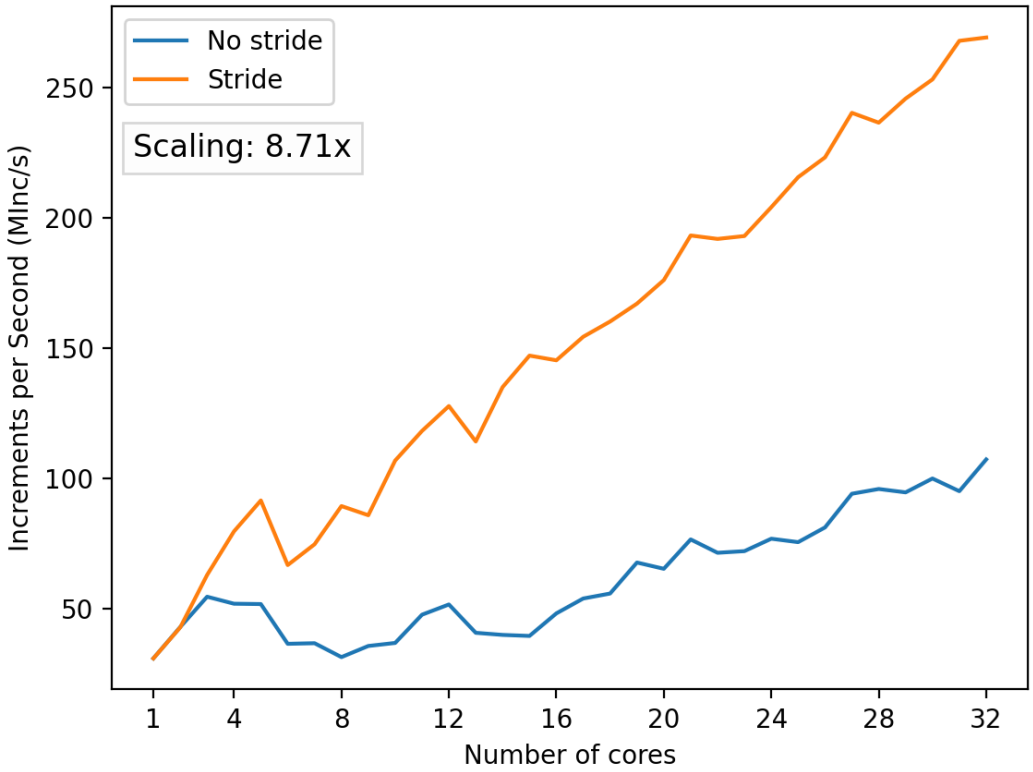

Reg
L1 Cache
L2 Cache
L3 Cache
RAM
Speed
Cost
Size

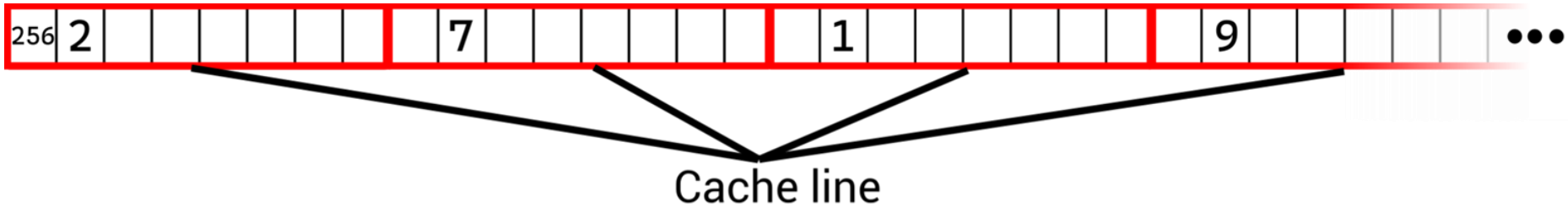

Processor 0
Core 0
L1 Cache
L2 Cache
Core 1
L1 Cache
L2 Cache
Core 2
L1 Cache
L2 Cache
Core 3
L1 Cache
L2 Cache
Shared L3 Cache

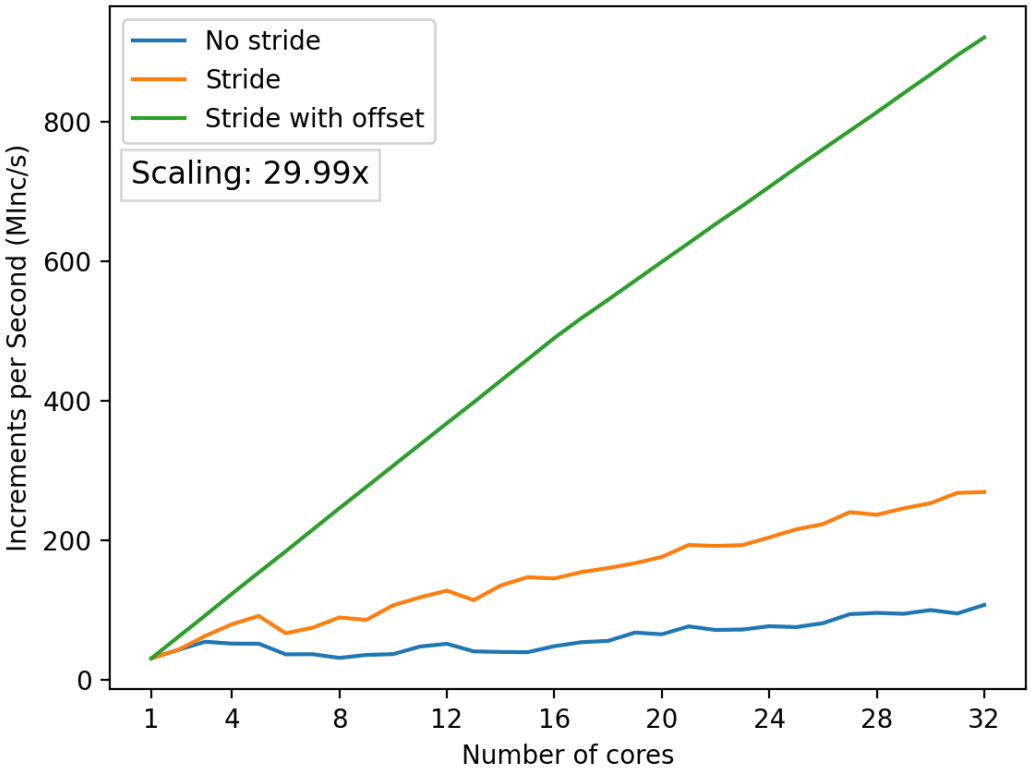

Scaling: 3.47x
Increments per Second (MInc/s)
110
100
90
80
70
60
50
40
30
1
4
8
12
16
20
24
28
32
Number of cores


2 7 1 9 4 3 5 2 9 1 4 4 8 5 7 9 9 9 1 2 1 3 4 4 5 4 3 8 7 9 6 2
Cache line

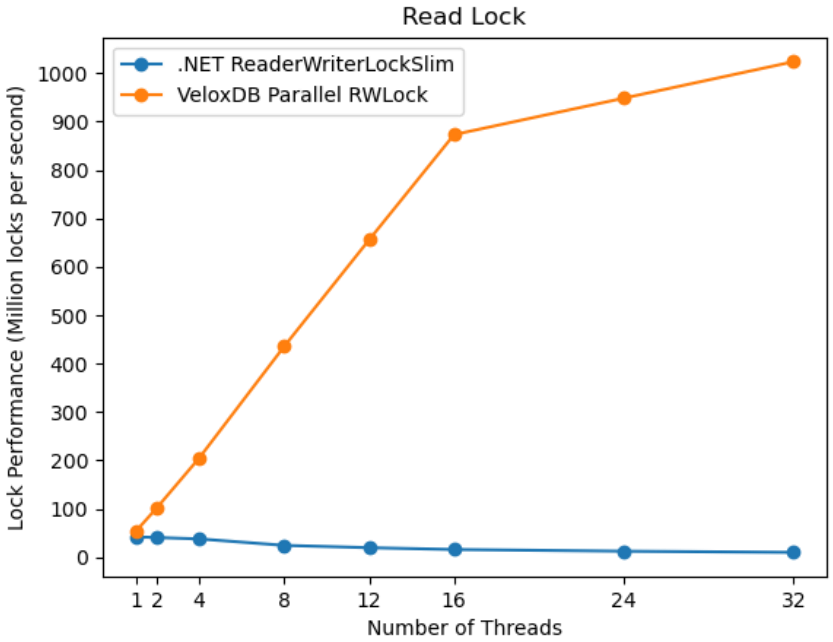

2
7
1
9
Cache line

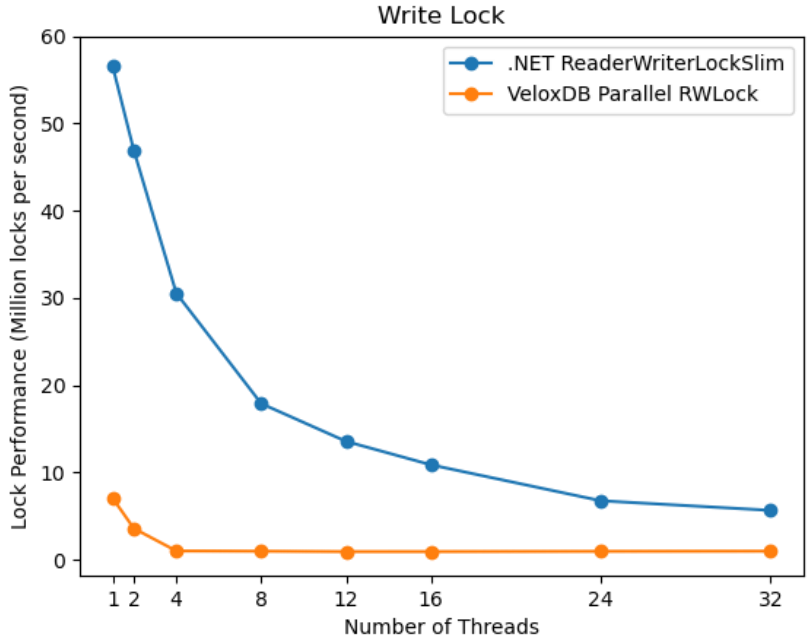

No stride
Stride
Scaling: 8.71x
Increments per Second (MInc/s)
250
200
150
100
50
1
4
8
12
16
20
24
28
32
Number of cores


256
2
7
1
9
Cache line


No stride
Stride
Stride with offset
Scaling: 29.99x
Increments per Second (MInc/s)
0
200
400
600
800
1
4
8
12
16
20
24
28
32
Number of cores

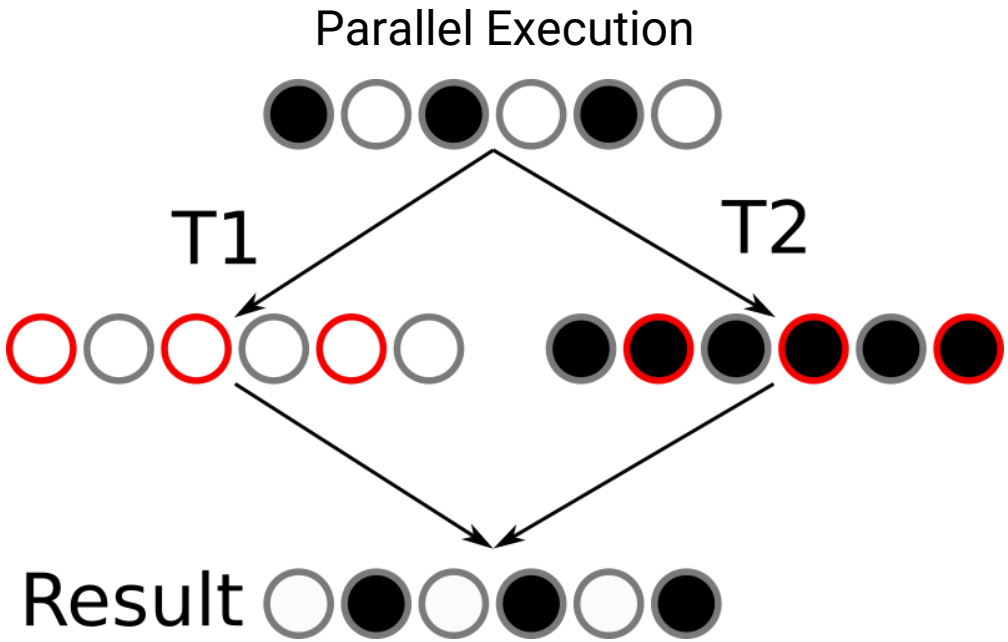
- Very fast read locking
- No contentions when reading
- Slower write
- Each CPU gets its own lock, on separate cache-line
- Read lock acquires only per-cpu lock
- Writes acquire all locks


Read Lock
.NET ReaderWriterLockSlim
VeloxDB Parallel RWLock
Lock Performance (Million locks per second)
0
100
200
300
400
500
600
700
800
900
1000
1 2 4 8 12 16 24 32
Number of Threads


Write Lock
.NET ReaderWriterLockSlim
VeloxDB Parallel RWLock
Lock Performance (Million locks per second)
0
10
20
30
40
50
60
1 2 4 8 12 16 24 32
Number of Threads

- Strict Serializability - Transactions view the database as if they are the only ones executing
- Transactions are executed in parallel
- Optimistic Concurrency Control
- Conflict - When transactions cannot be executed concurrently.
  - Write-Write conflict - Two transactions trying to modify the same object
  - Read-Write conflict - A transaction trying to write data that has been read by another transaction

- Table with black/white balls
- T1 - Scans the table and flips black to white
- T2 - Scans the table and flips white to black


Parallel Execution
T1
T2
Result

- Multiversion Concurrency Control (MVCC)
- Objects must keep track of every transaction that reads them
- Lists are expensive
- Small 8-byte structure that can hold 3 transaction IDs and a count
- If more than 3 transactions read the same object, a dedicated list is created

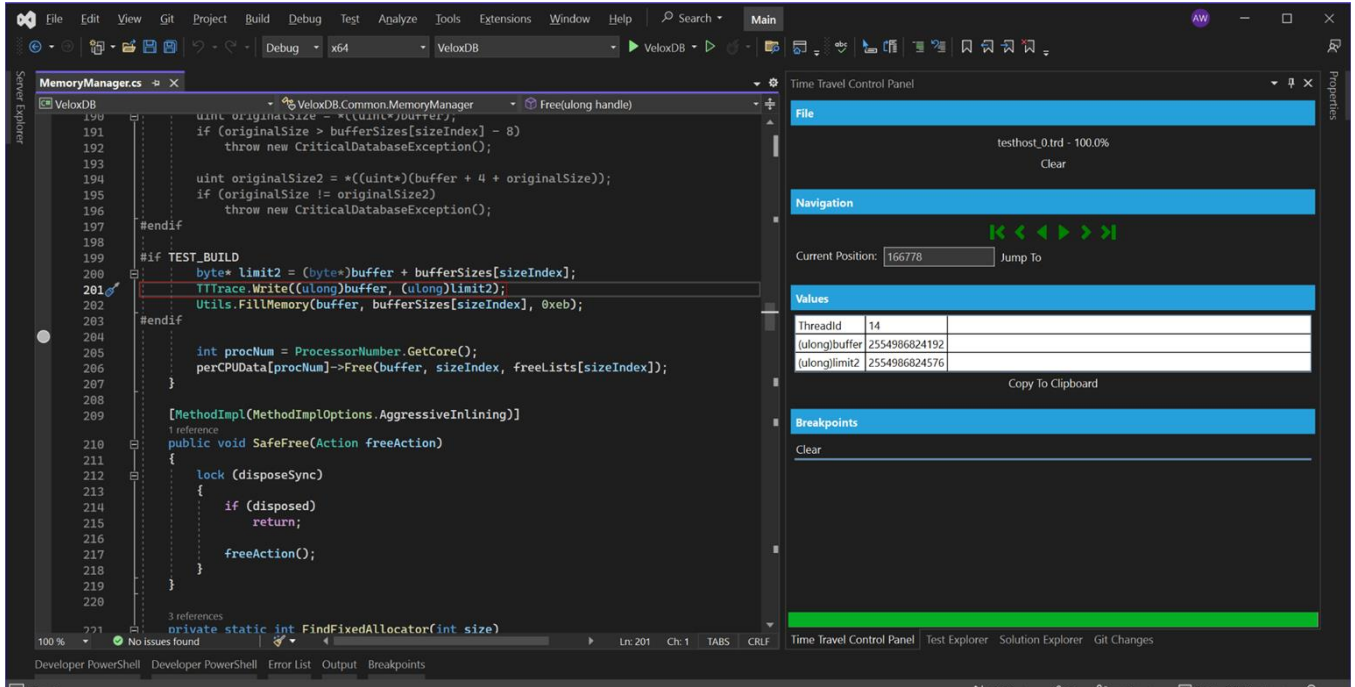
- Better understanding of data lifecycle
  - Data stored in database is going to be long lived
  - Fewer distinct types
  - Easier to detect what data can be discarded
- Reduces pressure on the .NET GC

- Each table has its own memory manager
- Each memory manager is composed of smaller per-cpu memory managers
- Fixed width allocators
- Freed objects are linked in free object list for reuse

- It’s important to verify that database behaves correctly
- Large body of tests (unit, component, end to end)
- Most useful - “stress” tests
  - Concurrent
  - Semantic Fuzzing
  - Verified using Reference database

- Challenging bugs (concurrent and memory corruption)
- Hard to reproduce
- We built our own time travel solution

```
1 reference
public void CreateObjectDiff(ClassObject* obj, ulong commonVersion, ChangesetWriter writer,
    IdSet partnerIds, List<long> otherIds, GenerateAlignDelegate alignDelegate)
{
    TTTrace.Write(TraceId, ClassDesc.Id, obj->id, obj->version, obj->IsDeleted, commonVersion);

    if (obj->IsDeleted)
        return;

    bool sbyMissing = partnerIds != null && !partnerIds.Contains(obj->id);
    if (obj->version > commonVersion || sbyMissing)
    {
        // If the standby deleted the object (due to split brain scenario) we need to pack an entire object which
        // will be achieved with the commonVersion being set to zero (this forces strings and blobs to be packed as well).
        TTTrace.Write(TraceId, ClassDesc.Id, obj->id, obj->version, commonVersion, sbyMissing);
        ulong version = sbyMissing ? 0 : commonVersion;
        alignDelegate(obj, stringStorage, blobStorage, writer, version);
        writer.LastValueWritten();
    }
    else
    {       You, 14 months ago • Initial commit
        TTTrace.Write(TraceId, ClassDesc.Id, obj->id, obj->version, commonVersion);
        otherIds ??= new List<long>();
        otherIds.Add(obj->id);
    }
}
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
Main
Debug
x64
VeloxDB
MemoryManager.cs
VeloxDB
VeloxDB.Common.MemoryManager
Free(ulong handle)
if (originalSize > bufferSizes[sizeIndex] - 8)
throw new CriticalDatabaseException();
uint originalSize2 = *((uint*)(buffer + 4 + originalSize));
if (originalSize != originalSize2)
throw new CriticalDatabaseException();
#endif
#if TEST_BUILD
byte* limit2 = (byte*)buffer + bufferSizes[sizeIndex];
TTTrace.Write((ulong)buffer, (ulong)limit2);
Utils.FillMemory(buffer, bufferSizes[sizeIndex], 0xeb);
#endif
int procNum = ProcessorNumber.GetCore();
perCPUData[procNum]->Free(buffer, sizeIndex, freeLists[sizeIndex]);
}
[MethodImpl(MethodImplOptions.AggressiveInlining)]
1 reference
public void SafeFree(Action freeAction)
{
lock (disposeSync)
{
if (disposed)
return;
freeAction();
}
}
3 references
private static int FindFixedAllocator(int size)
100 %
No issues found
Ln: 201
Ch: 1
TABS
CRLF
Time Travel Control Panel
File
testhost_0.trd - 100.0%
Clear
Navigation
Current Position:
166778
Jump To
Values
ThreadId
14
(ulong)buffer
2554986824192
(ulong)limit2
2554986824576
Copy To Clipboard
Breakpoints
Clear
Time Travel Control Panel
Test Explorer
Solution Explorer
Git Changes
Developer PowerShell
Developer PowerShell
Error List
Output
Breakpoints
Server Explorer
Properties

Questions