Dragged Kicking and Screaming: Source Multicore

Tom Leonard, Valve
9 March 2007

VALVE

Multicore

- Most significant development since consumer 3D

Multicore

- Most significant development since consumer 3D
- Explicit parallelism
  - Hardware problem becoming software problem will require new techniques

Introduction

- The decisions faced with multiple cores
- How we are approaching multiple cores
- Algorithms and paradigms

VALVE

Goals

- Integrate multicore across Valve's business
  - Expose to game programmers, licensees and MOD authors

Goals

- Integrate multicore across Valve's business
- Scale to cores without recompile

Goals

- Integrate multicore across Valve's business
- Scale to cores without recompile
- Create value beyond framerate
  - Apply cores to new gameplay

Challenges

- Games want maximal CPU utilization
- Games are inherently serial
- Decades of experience in single threaded optimization
- Millions of lines of code written for single threading

Strategies

- Threading model
- Threading framework

Threading Models

- Fine grained threading
- Coarse threading
- Hybrid threading

VALVE

Diving In

- Client
  - User input
  - Rendering
  - Graphics simulation
- Server
  - AI
  - Physics
  - Game logic

Diving In

- Experiment: run client and server each on own core

VALVE

Diving In

- Experiment: run client and server each on own core
- Benefits: forced to confront systems that are not thread safe or not thread efficient

Discoveries

- Problem: shared data access
  - Global data
  - Static data (optimizations/function local state)
  - Singleton objects

Discoveries

- Problem: shared data access
- Thread safety is easy!

Discoveries

- Problem: shared data access
- Thread safety is easy!
  - Slap on a mutex/critical section

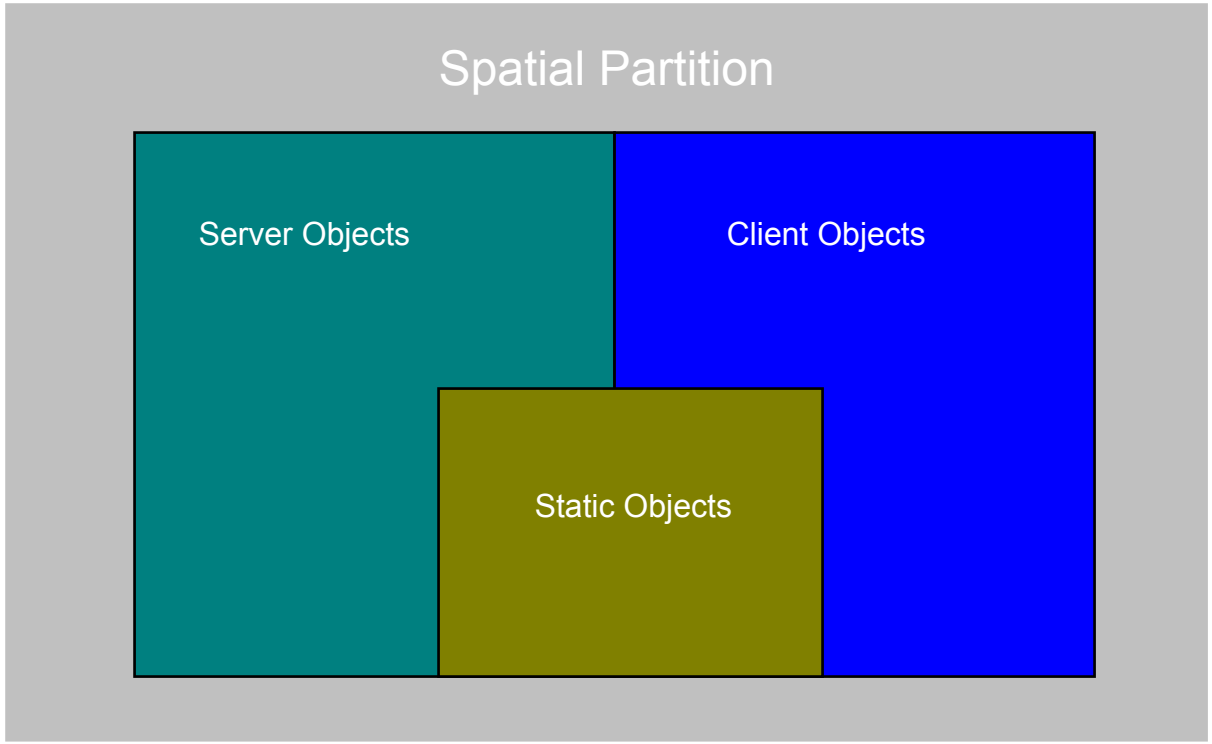
Discoveries

- Problem: shared data access
- *Bad* thread safety is easy!
  - Slap on a mutex/critical section
  - The simple thing is the worst thing
    - Mutexes are terrible
      - Excessive waits
      - Error prone
      - Fail to scale
  - Establish slow but stable baseline

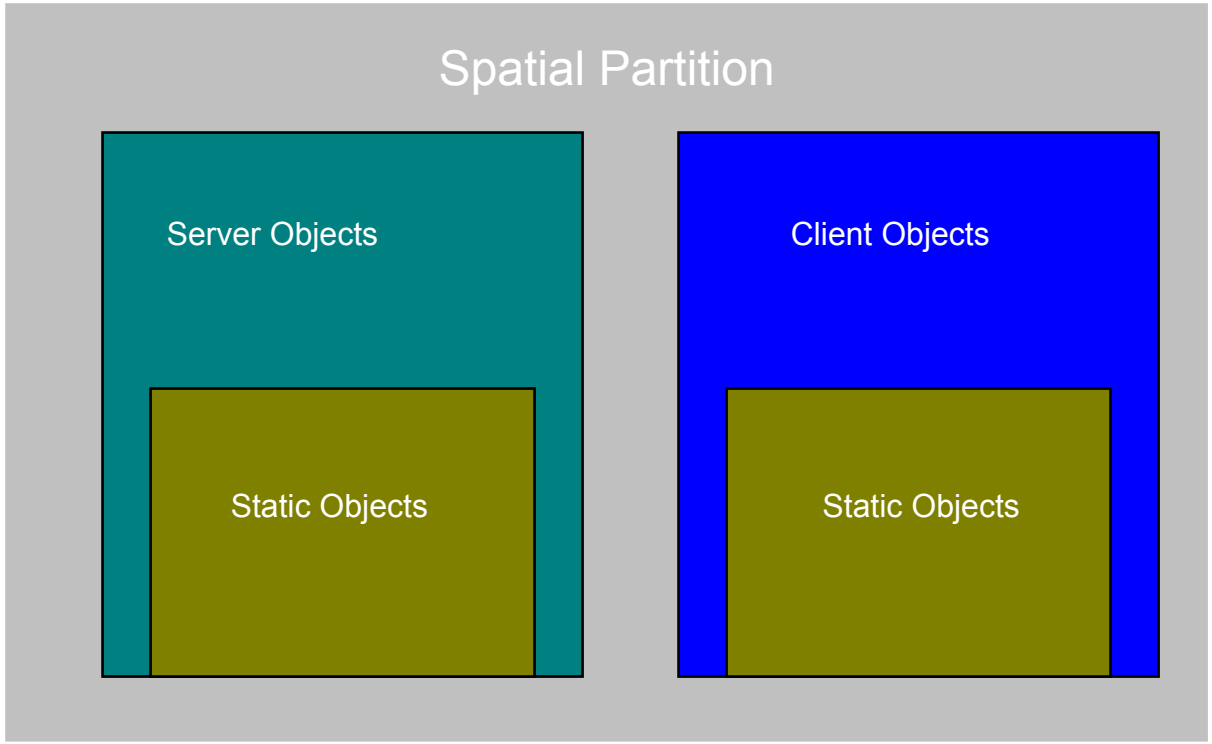
Discoveries

- Efficient thread safety
  - No synchronization (“wait-free”)
    - Each thread has a private copy of all the data needed to perform operation:
      - Threads working on independent problems
      - Replace globals with thread private data
      - Reorient to pipeline
    - Example: Source “Spatial Partition”

Discoveries



Discoveries



Discoveries

- Efficient thread safety
  - No synchronization (“wait-free”)
  - Better synchronization tools, techniques
    - Analyze data access
    - Example: symbol table using read/write lock
  - Decouple using queued function calls

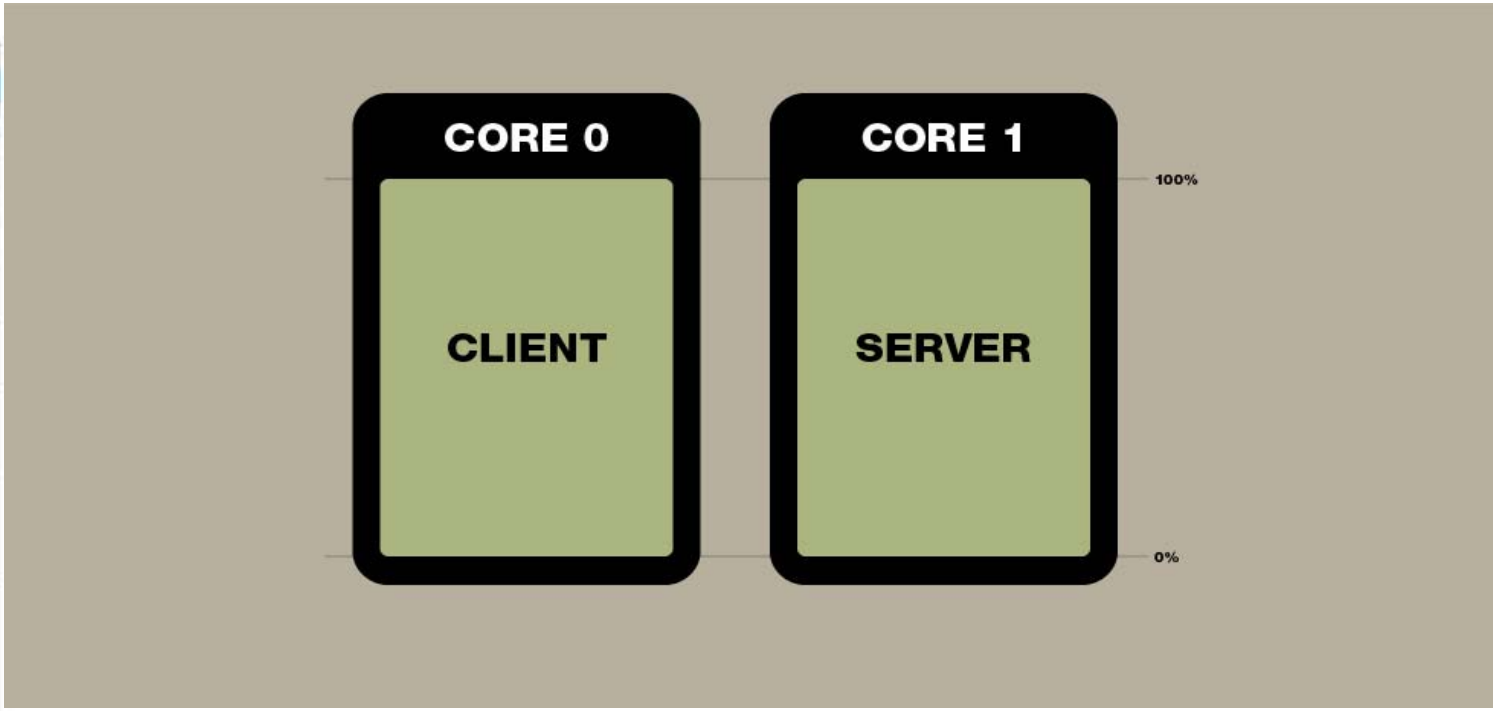
Discoveries

- What if you can't eliminate contention over shared resources?

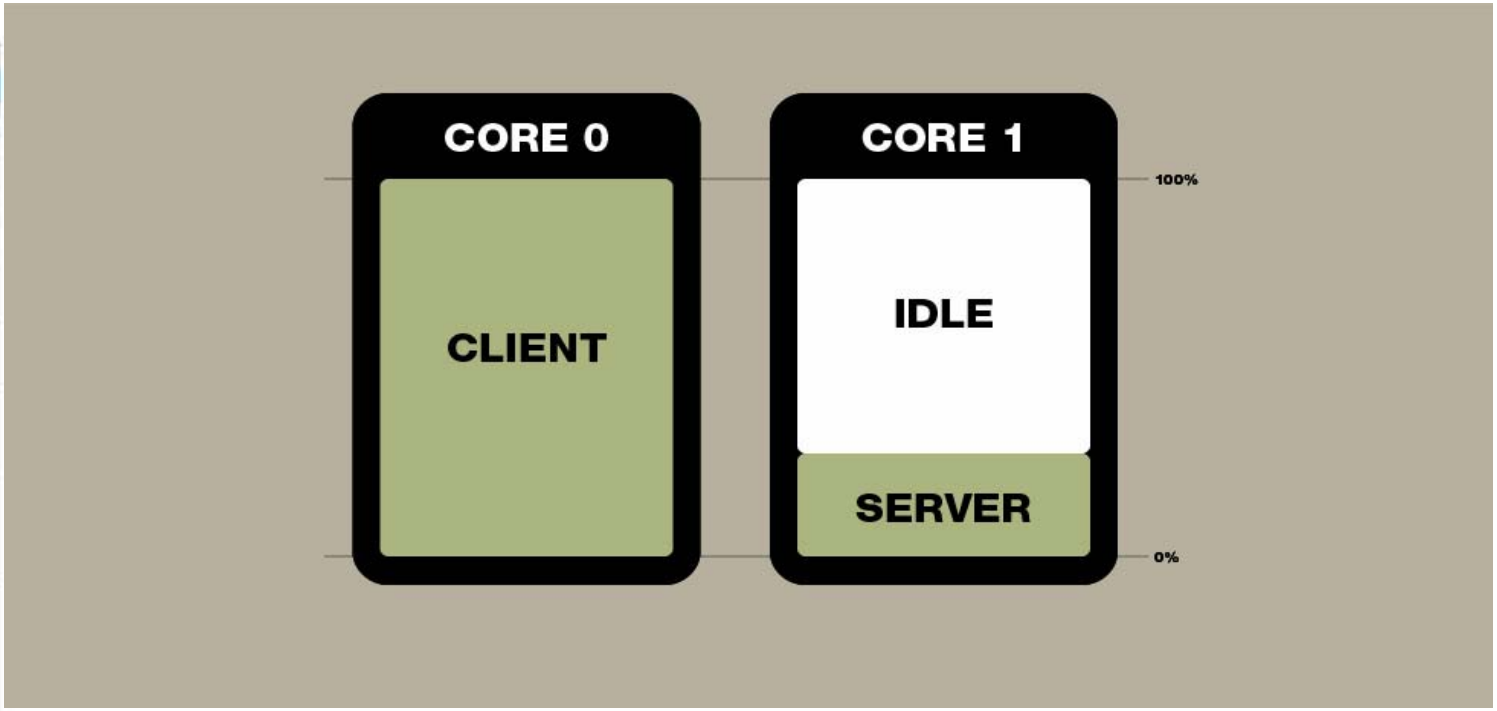
Results

- Can approach 2x in contrived maps

Results



Results



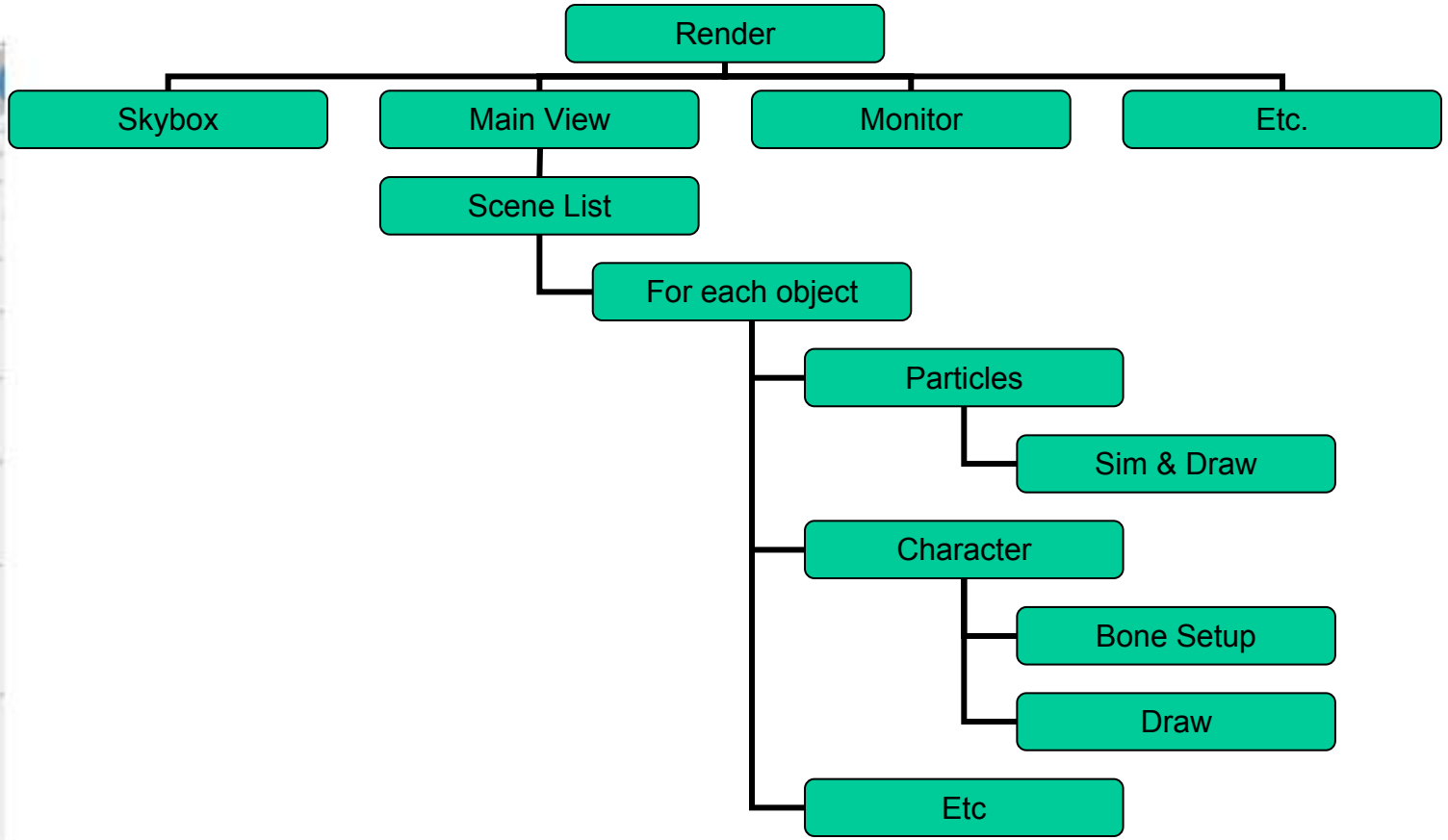
Results

- Can approach 2x in contrived maps
- More like 1.2x in real single player
- Applicable to 360 Team Fortress 2

Hybrid threading

- Use the appropriate tool for the job
  - Some systems on cores (e.g. sound)
  - Some systems split internally in a coarse manner
  - Split expensive iterations across cores fine grained
  - Queue some work to run when a core goes idle
- Need strong tools
- Maximal core utilization

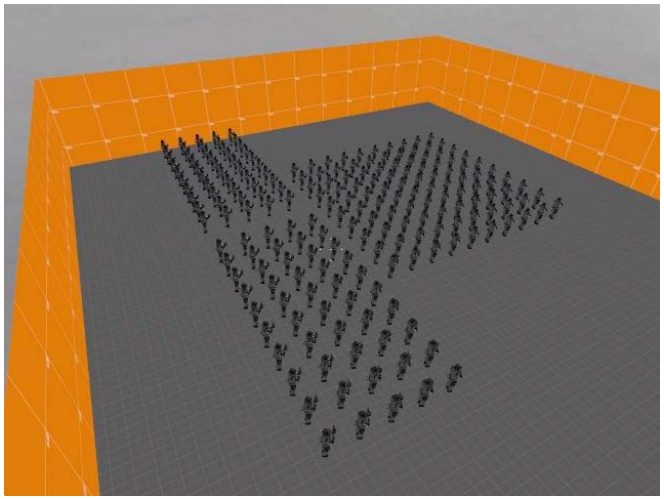
Hybrid threading: Rendering



Hybrid threading: Rendering

- Problems
  - Per-view scene construction limits opportunity
  - Arbitrary object type order
  - Arbitrary code execution
- Simulation and Rendering interleaved
  - Lazy calculation optimizations

Hybrid threading: Rendering



- Iterative Transition: Skeletal Animation
  - Parallelize lazy calculation triggers
  - Refactor bone setup into single pass per view
  - Refactor into single pass for all views
  - Same pattern for other CPU-intensive stages

Hybrid threading: Rendering

- Revised pipeline
  - Construct scene rendering lists for multiple scenes in parallel (e.g., the world and its reflection in water)
  - Overlap graphics simulation
  - Compute character bone transformations for all characters in all scenes in parallel
  - Allow multiple threads to draw in parallel
  - Serialize drawing operations on another core

Threading Tools

- Implementing Hybrid Threading

- Programmers solve game development problems, not threading problems
- Empower all programmers to leverage cores

- Operating system: too low level
- Compiler extensions (OpenMP): too opaque
- Tailored tools: correct abstraction

Tailored tools: Game Threading Infrastructure

- Custom work management system
  - Intuitive for programmers
  - Focus on keeping cores busy
  - Thread pool: N-1 threads for N cores
  - Support hybrid threading
    - Function threading
    - Array parallelism
    - Queued and immediate execution

Tailored tools: Game Threading Infrastructure

- Goal: make system easy to use, hard to mess up
- Example: compiler generated functors
  - Uses templates to package up functions and data, point of call looks very similar
  - Call arrives on other end as if called normally
  - Saves time, reduces error, encourages experimentation

Tailored tools: Game Threading Infrastructure

- One-off push to another core

```
if ( !IsEngineThreaded() )
    _Host_RunFrame_Server( numticks );
else
    ThreadExecute( _Host_RunFrame_Server, numticks );
```

Tailored tools: Game Threading Infrastructure

- Parallel loop

```
void ProcessPSystem( CParticleEffect *pEffect );

ParallelProcess( particlesToSimulate.Base(),
                 particlesToSimulate.Count(),
                 ProcessPSystem );
```

Tailored tools: Game Threading Infrastructure

- Queue up a bunch of work items, wait for them to complete

```
BeginExecuteParallel();
ExecuteParallel( g_pParticleSystem,
                 &CParticleSystem::Update, time );
ExecuteParallel( &UpdateRopes, time );
EndExecuteParallel();
```

- Low level APIs for the brave

Contention

- What if you can't eliminate contention over shared resources?
- Example: Allocator
  - Heavily used
  - Multiple pools of fixed sized blocks with a custom spin lock mutex per-pool
  - Mutex limiting scale
  - Didn't want per-thread allocators

Contention

- Lock-free algorithms
  - No thread can block system regardless of scheduling or state
  - Under the hood of all services and data structures
  - Relies on atomic write instructions, “compare-and-swap”

Contention

```
bool CompareAndSwap(int *pDest, int newValue, int oldValue)
{
    Lock( pDest );
    bool success = false;
    if ( *pDest == oldValue )
    {
        *pDest = newValue;
        success = true;
    }
    Unlock( pDest );
    return success;
}
```

Contention

```
bool CompareAndSwap(int *pDest, int newValue, int oldValue)
{
    __asm
    {
        mov eax,oldValue
        mov ecx,pDest
        mov edx,newValue
        lock cmpxchg [ecx],edx
        mov eax,0
        setz al
    }
}
```

Contention

- Use lock-free algorithm in allocator
  - Replace mutex and traditional free list per-pool with a lock-free list per-pool
  - Windows API/XDK SList

Lock-free example: singly linked list

- Compare-and-swap
  - “If head is equal to what I think it is, assign with my new head”
  - ABA Problem: is it the same head?
  - Use a serial number as a discriminating field

Lock-free example: singly linked list

```
class CSList
{
public:
    CSList()
    void Push( SListNode_t *pNode );
    SListNode_t *Pop();
    SListNode_t *Detach();
    int Count() const;
private:
    SListHead_t m_Head;
};
```

Lock-free example: singly linked list

```
struct SListNode_t
{
    SListNode_t *pNext;
};

union SListHead_t
{
    struct Value_t
    {
        SListNode_t *pNext;
        int16 iDepth;
        int16 iSequence;
    } value;
    int64 value64;
};
```

Lock-free example: singly linked list

```
Void Push( SListNode_t *pNode )
{
    SListHead_t oldHead, newHead;
    for (;;)
    {
        oldHead.value64 = m_Head.value64;
        newHead.value.iDepth = oldHead.value.iDepth + 1;
        newHead.value.iSequence = oldHead.value.iSequence + 1;
        newHead.value.Next = pNode;
        pNode->pNext = oldHead.value.pNext;
        if ( ThreadInterlockedAssignIf64( &m_Head.value64,
                  newHead.value64, oldHead.value64 ) )
        {
            return;
        }
    }
}
```

Lock-free example: singly linked list

- Lock-free list exceptionally useful
  - Keep pools of context structures when impractical to give every thread a context
  - Efficiently gather results of a parallel process for later handling
  - Build up lists of data to operate on using Push(), then use Detach() (a.k.a “Flush”) to grab the data in another thread in a single operation

Example

```
extern Vector trace_start;
extern Vector trace_end;
// etc...
struct cbrush_t
{
    int             contents;
    unsigned short  numsides;
    unsigned short  firstbrushside;
    int             checkcount;  // to avoid repeated testings
};

//////////////////////////////

void BeginTrace()
{
    g_CModelMutex.Lock();
    ++s_nCheckCount;
}
```

Example

```
struct TraceInfo_t
{
    Vector m_start;
    Vector m_end;
    // etc...
    CVisitBitVec m_BrushVisits;
};

CTraceInfoPool g_TraceInfoPool;

TraceInfo_t *BeginTrace()
{
    TraceInfo_t *pTraceInfo;
    if ( !g_TraceInfoPool.PopItem( &pTraceInfo ) )
        pTraceInfo = new TraceInfo_t;

    return pTraceInfo;
}
```

Lock-free algorithms

- Thread pool work distribution queue
  - Derived from HL2 asynchronous I/O queue
  - Designed for one provider, one consumer
  - Simple prioritized queue with mutex
  - Arbitrary priority
  - One queue for all threads

Lock-free algorithms

- Solutions
  - Use lock-free queue (Fober, et. al.)
  - Rework interface to fixed priorities, one queue per-priority
    - *Interfaces critical*
  - Queues per core in addition to a shared queue
  - Use atomic operations to get “ticket”, actual work done may differ

Lock-free algorithms

- Locks permit a stable reality
- Lock-free permits reality to change instruction to instruction
- Leverage inference rather than locks to know part of the system is stable

- Wait-free is always better

Looking Forward

- Why so much up-front investment?

Looking Forward

- Why so much up-front investment?
  - Steam
    - Communicate with customers
    - Tap markets not available via retail
  - Dramatic change is underway
    - Core count double every 18 months
    - CPU/GPU/PPU/AIPU/etc not the future
    - Many homogeneous cores
    - Division of computing power a software problem

Call to action

- Build or acquire strong tools, new techniques
- Embrace lock-free mechanisms to move work and data to and from wait-free code
- Prepare for decomposition of features over many cores
- Use accessible solutions to empower all programmers, not just systems programmers
- Support even higher level threading framed in terms of game problems

VALVE

Summary

- Started with a stable but bad threading
- Iteratively eliminated bad cases using variety of techniques, usually lock-free
- During iterations, expanded toolset to meet newly discovered needs
- Focused on ease-of-use for other programmers
- Now being applied by others at higher levels

- In Source SDK this summer
- Contact: tom_gdc@valvesoftware.com

VALVE