GAME ENGINE USING C++11

JASON JURECKA

HELLO & WELCOME

Disclaimer: I work at Blizzard but not representing a Blizzard product.

ABOUT ME...

- 12+YEAR VETERAN
- 10+ TITLES/PROJECTS CONTRIBUTED TO.
- HAVE THE RESPONSIBILITIES OF A SWISS ARMY KNIFE.

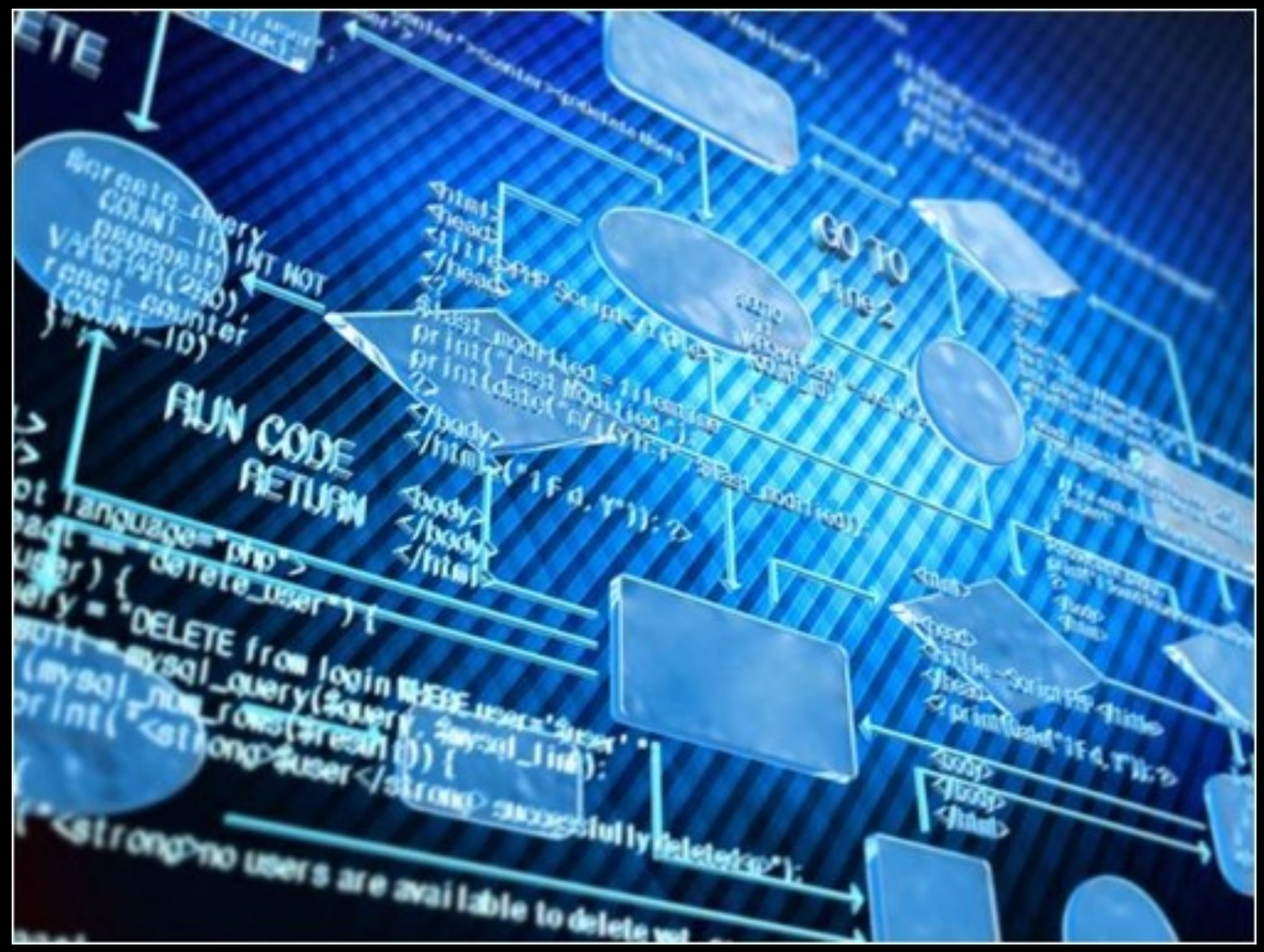
TABLE OF CONTENTS SUMMARY

PROJECT MENTALITY

- Mass Concurrency
- Leverage std:: over roll my own
- Keep things simple
- DO NOT get sucked into ‘C’ style patterns
- DO NOT focus on latest graphics
- Work for 60fps - frame ~ 16ms
- Learn/Grow

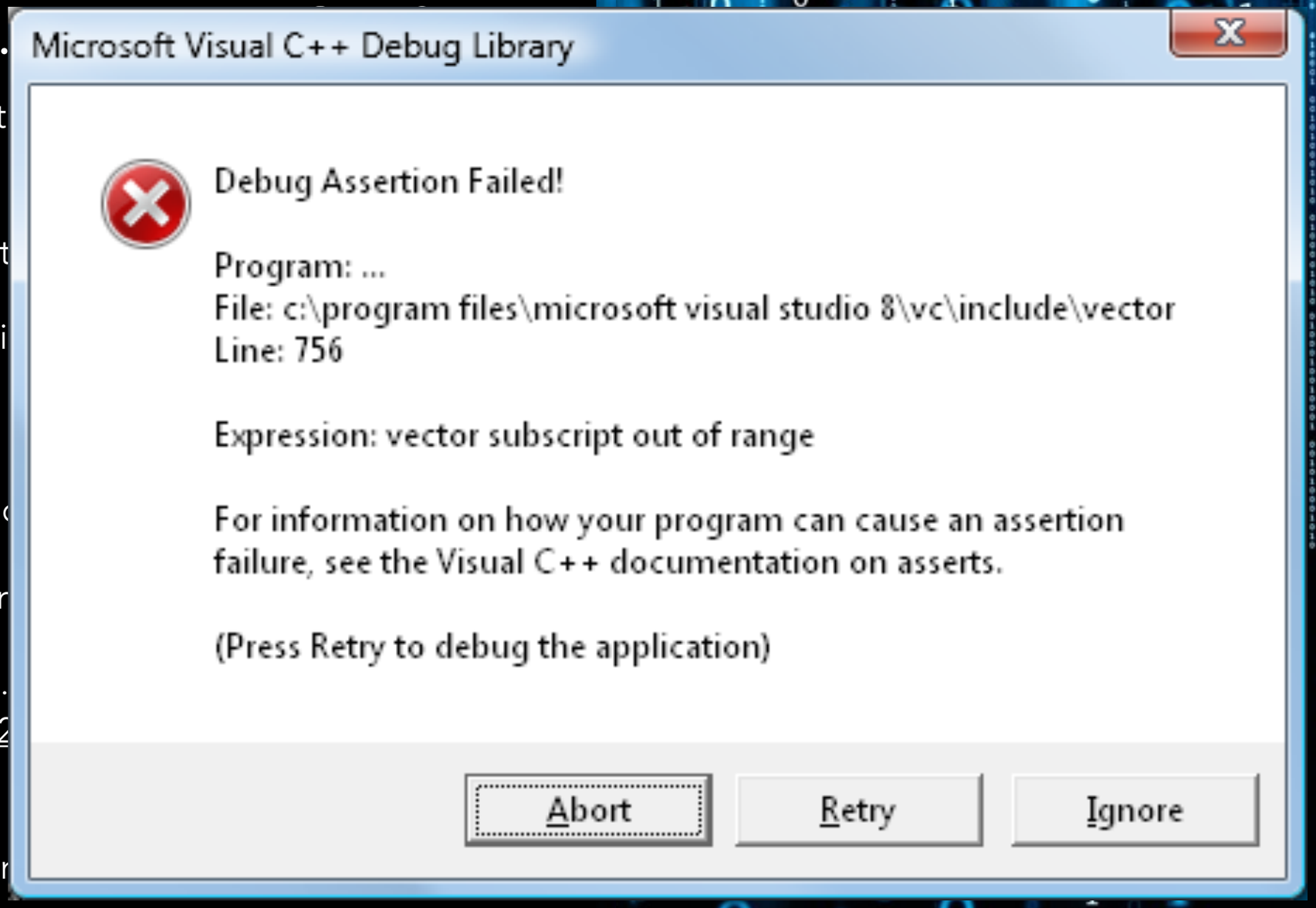
WHAT IS THE MAIN JOB OF A GAME ENGINE?

- Interface with Hardware API's (graphics, sound, network, input…)
- Data creation/loading
- Data interpretation
- Data Presentation



GAME ENGINES DON'T USE STD MUCH...

- C++98 Replacement
  years why change").
- Hardware implement
- Containers + std::stri
  - Slow Iteration
  - Slow manipulati
  - Memory usage r
  - Flat Containers .
    std.org/jtc1/sc22
    p0038r0.html
- Code size add for ter
  - https://www.youtube.com/watch?v=p6NxZ5Qnzfo



Why Not? (Cont)

- Throwing of exceptions
  - Many game code bases turn them off all together
    - Limit's 3rd Party library usage
    - Some Libraries use exceptions as execution flow
  - QA is supposed to find issues during testing phase
- We can patch to fix found things in the wild
  - It is a game not a cardiac bypass machine/moon rover
- Cost Explained link http://h-deb.clg.qc.ca/Sujets/Developpement/Exceptions-Costs.html

WHY NOT? (CONT)

- Not all games are created equal
  - Different concerns if RTS vs FPS vs RPG
  - Same with open world/level based/side-scroller
  - Platforms restrictions (XBOX/PS vs PC/MAC)
    - Console dedicated threads or smaller thread pool
  - PC variant hardware configerations
  - Mobile devices (android/ios)
  - Web browser based limits (HTML5, flash)

HOW DOES C++11 HELP US? - DATA CREATION/LOADING

- Compile time resource versioning using constexpr
- Compile time type validation using static_assert
- Compile time serialize/deserialize function lists

COMPILE TIME RESOURCE VERSIONING USING CONSTEXPR

- What would go into computing a version hash for this structure?
  - name of structure
  - data member type
  - data member name
  - data member order
  - data alignment

```
struct Vector3
{
    float x = 0.0f;
    float y = 0.0f;
    float z = 0.0f;
};

struct SpawnPoint
{
    Vector3 m_Position;
    Vector3 m_Direction;
};
```

COMPILE TIME RESOURCE VERSIONING USING CONSTEXPR (CONT)

```
using HashValue = unsigned int;

class constexprString {
    const char* p;
    std::size_t sz;
public:
    template<std::size_t N>
    constexpr constexprString(const char(&a)[N]) : p(a), sz(N - 1) {}
    constexpr char operator[](std::size_t n) const { return (n >= sz) ? '\0' : p[n]; }
    constexpr std::size_t size() const { return sz; }
};

//Constant Expression using recursion to create a hash value
constexpr HashValue GetHash_constexprString(const constexprString& _toHash, const unsigned index = 0)
{
    return ( index >= _toHash.size() ) ? 0 : (static_cast<unsigned>(_toHash[index]) * (index+1) \
        * s_PrimeTable[(static_cast<unsigned>(_toHash[index]) % s_PrimeNumCount)]) \
        + GetHash_constexprString(_toHash, index + 1);
}

template<std::size_t N>
constexpr HashValue GetHash_constexprString_Array(const constexprString(&_toHash)[N], const unsigned index = 0)
{
    return (index >= N) ? 0 : GetHash_constexprString(_toHash[index], 0) + \
        (GetHash_constexprString_Array(_toHash, index + 1) << index);
}
```

COMPILE TIME RESOURCE VERSIONING USING CONSTEXPR (CONT)

```
struct SpawnPoint
{
    Vector3 m_Position;
    Vector3 m_Direction;
};
```

```
struct SpawnPoint
{
    Vector3 m_Position;
    Vector3 m_direction;
};
```

```
struct spawnPoint
{
    Vector3 m_Position;
    Vector3 m_Direction;
};
```

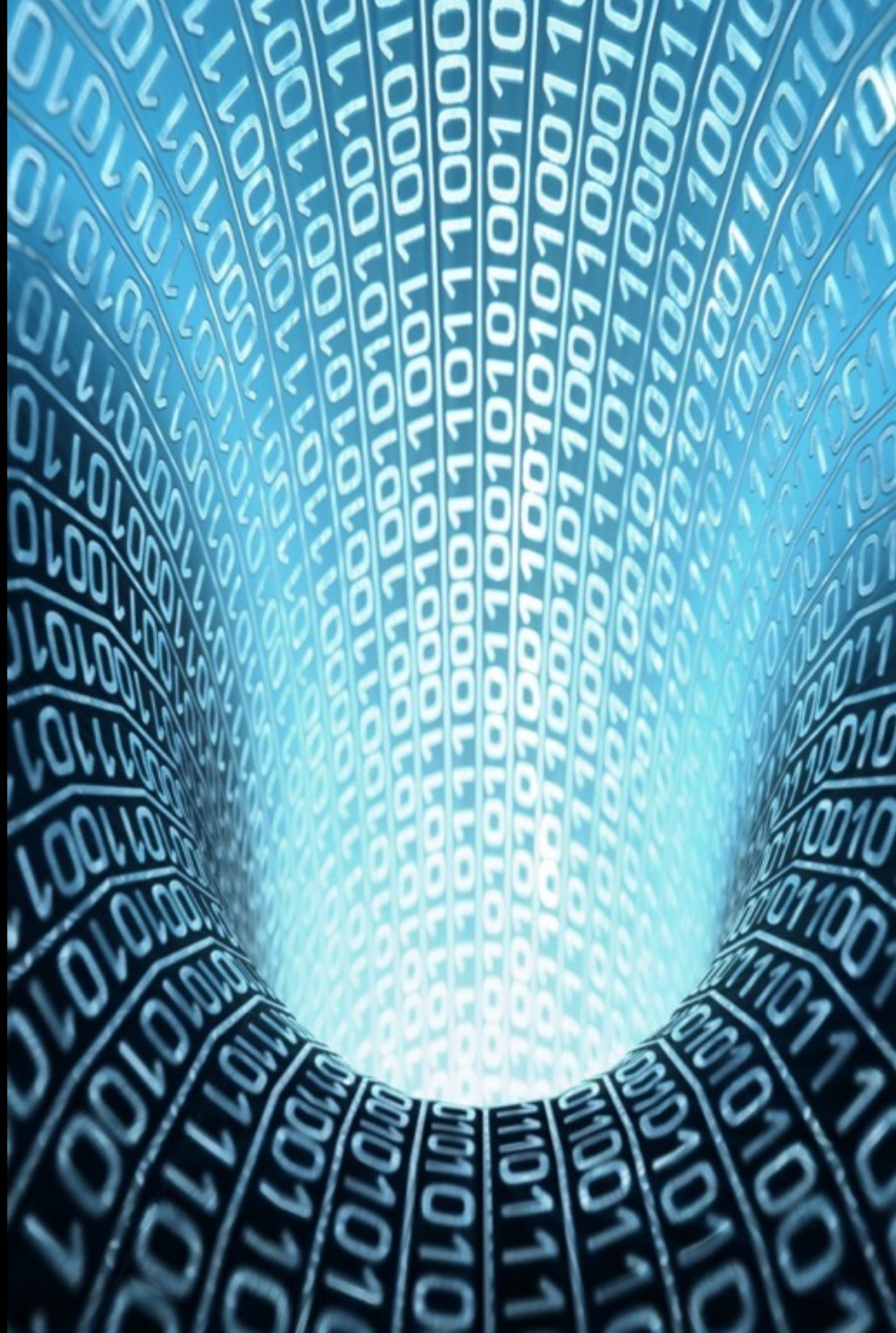
```
struct SpawnPoint
{
    Vector3 m_Direction;
    Vector3 m_Position;
};
```

COMPILE TIME RESOURCE VERSIONING USING CONSTEXPR (CONT)

```
RSC_DATA_BEGIN(Vector3)
→   RSC_DATA(Vector3, float, m_X)
→   RSC_DATA(Vector3, float, m_Y)
→   RSC_DATA(Vector3, float, m_Z)
RSC_DATA_END(Vector3)

RSC_DATA_BEGIN(SpawnPoint)
→   RSC_DATA(SpawnPoint, rscVector3, m_Position)
→   RSC_DATA(SpawnPoint, rscVector3, m_Direction)
RSC_DATA_END(SpawnPoint)
```

MASS CONCURRENCY

- Most (if not almost all) devices now have multiple cores and thread support
    - Why not use that power?
- Benefits
    - Exploit multi processor machines.
    - Exploit cloud processing (dynamic scaling)
    - Tasks can be small and simple (focused optimizations)
    - serialized sequences can be distributed
    - highly modular for version-ing of tasks/ systems
    - efficiency of doing multiple things at once out weighs serialized process cost.



MASS CONCURRENCY (CONT)

- Concerns:
  - Tasks need to be as independent as possible
  - Data manipulation needs to be thread safe (dead-locks or live-locks)
  - Need to keep in mind when data synchronization happens
  - Need to account for things executing in non determined order (ish)
  - Task queue speed (lock vs lock-free)
  - Thread context switching

TASKSYSTEM

```
namespace TaskMaster
{
    bool Initialize();
    void Shutdown();
    void Process();

    void AddTask(std::unique_ptr<ITask> task);
    std::future<bool> AddTrackedTask(std::unique_ptr<ITask> task, ITask* parent);
}
```

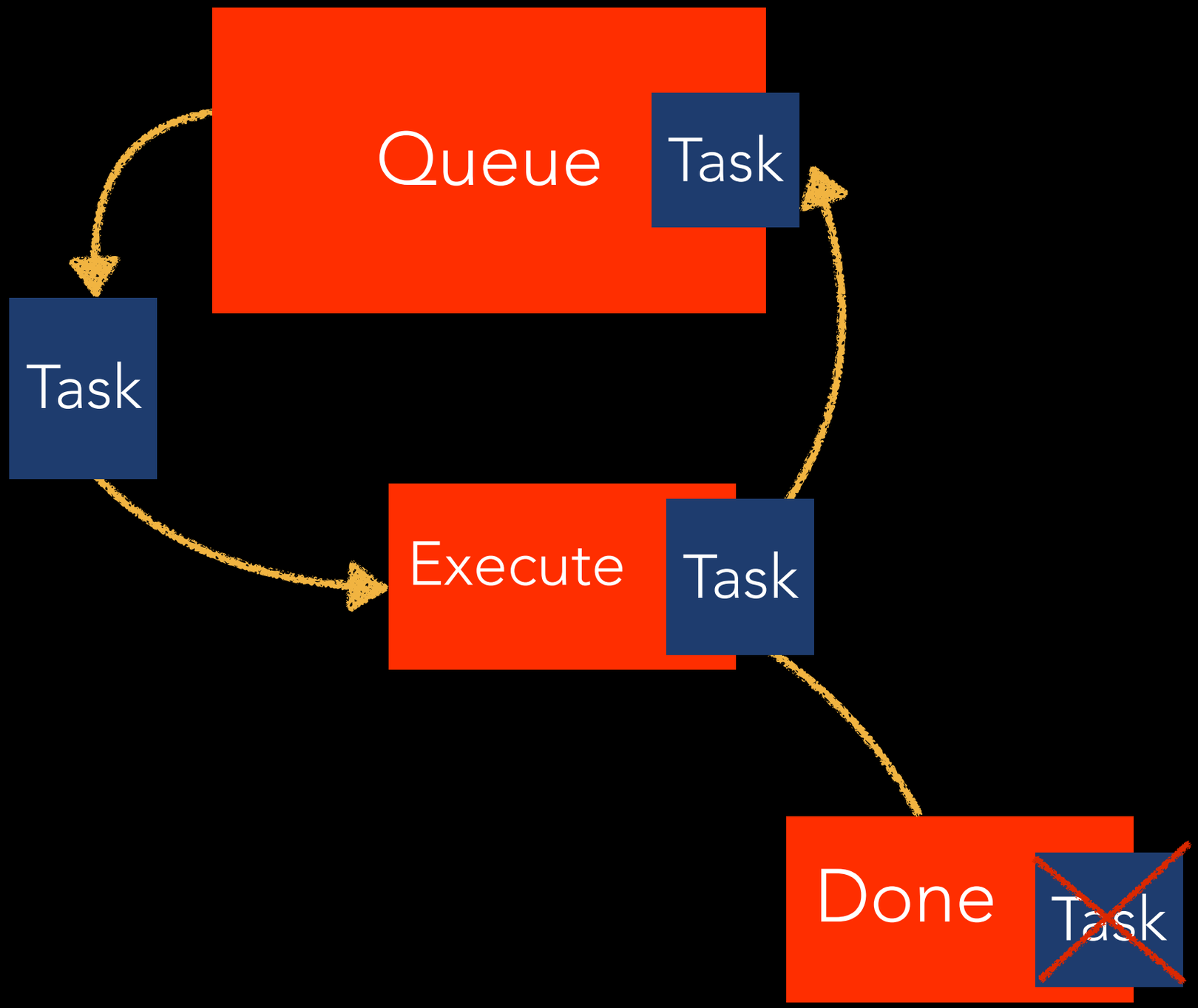
TASKSYSTEM (CONT)

```
void TaskMaster::AddTask(std::unique_ptr<ITask> task)
{
    assert(task);
    s_Tasks.push(std::move(task));
}

std::future<bool> TaskMaster::AddTrackedTask(std::unique_ptr<ITask> task, ITask* /*parent*/)
{
    assert(task);
    std::unique_ptr<TrackedTask> wrapper = std::make_unique<TrackedTask>(task);
    std::future<bool> status = wrapper->GetFuture();
    AddTask(std::move(wrapper));
    return status;
}
```

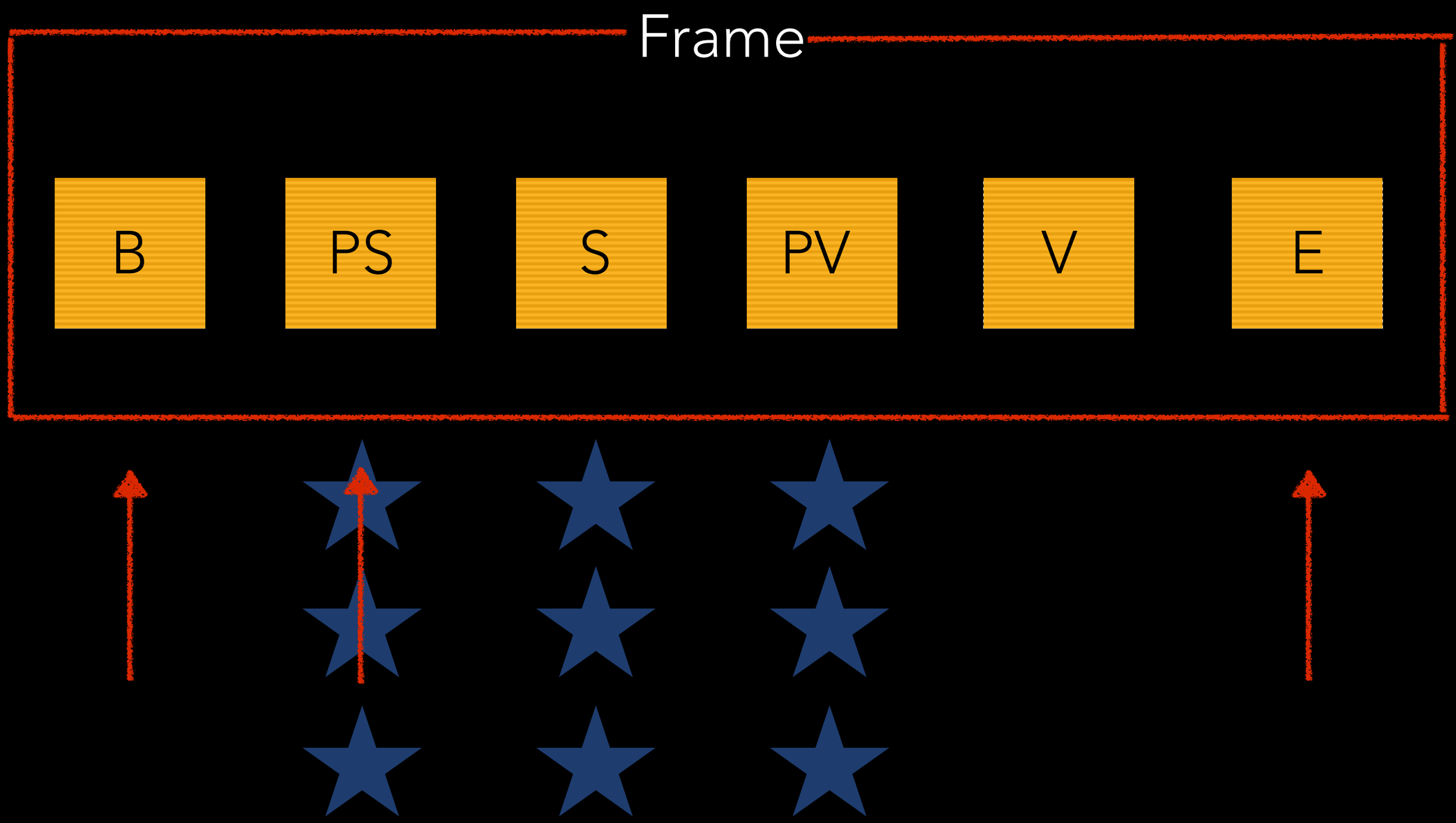
TASKSYSTEM (CONT) - PROCESSING A TASK

```
//Use std::thread::hardware_concurrency() to populate thread "pool" during initialize
std::atomic_bool s_RunThreads;
std::vector< std::unique_ptr< std::thread > > s_ThreadPool;

void runTask()
{
    std::thread::id id = std::this_thread::get_id();
    while (s_RunThreads)
    {
        std::unique_ptr<ITask> task = s_Tasks.pop();
        if (task != nullptr)
        {
            if (!s_RunThreads)
            {
                //no longer need to keep the task around ... we are shutting down
                break;
            }
            task->Run(); //Run it
            if (!task->IsComplete())
            {
                if (!s_RunThreads)
                {
                    //no longer need to keep the task around ... we are shutting down
                    break;
                }
                s_Tasks.push(std::move(task));
            }
        }
        else
        {
            std::this_thread::sleep_for(std::chrono::milliseconds(3));
        }
    }
}
```

TASKSYSTEM (CONT)-TASK FLOW



WHAT'S IN A FRAME?



WHAT'S IN A FRAME (CONT)

- Why is visualize not using parallelism?
    - prep for visualization does all prep-work of sorting and filtering visualize requests.
    - Visualize uses the gpu which typically is not multi threaded
    - SIMD can be used to help with processing data, but most prep-work should be done in prep stage.
    - Visual fancy features can be expensive so filtering and sorting should be done beforehand.

C++ STANDARD HIGHLIGHTS

- Standardization helps simplify code bases.
  - Move semantics speeding up data copying for all std data containers
  - Threading library
  - Algorithms Library
    - Algorithms work on raw arrays as well.
    - std::distance (getting index info from contiguous array members)

C++ STANDARD HIGHLIGHTS CONTINUED

- Std::function
- static_assert
- Constexpr
- Std::tuple (dynamic data grouping)
- Type traits and compile time checks
- Std::atomics
- Std::array, std::unordered_map, std::unordered_set

RIDDLE:

What can hold onto something bigger than itself and ensure proper clean-up?

C++ STANDARD HIGHLIGHTS SMART_PTRS

- Between class/struct data management
- clean up allocated memory (on proper shutdown and improper shutdown)
- Show how an object is supposed to be used
  - unique_ptr I am the owner (i.e. tasks,managers)
  - shared_ptr Ref counted item (i.e. asset resources)

RIDDLE

I am in front of you, you cannot see me, but you may see me later?

C++ STANDARD HIGHLIGHTS
STD::FUTURE

- A tool for getting results from tasks
- Provide dependency link between tasks
- The size of a std::future is equal to size of 2 pointers

X86	X64
8	16

NOTE: sizeof (std::future<bool>) == sizeof(std::future<int*>)

C++ STANDARD HIGHLIGHTS
STD::FUTURE (CONT)

```cpp
template <typename t>
bool futureReady(std::future<t>& toCheck)
{
    return toCheck.valid() && toCheck._Is_ready(); //NOTE: MS specific ...
}

std::future< std::unique_ptr< SystemActionGather::Result > > m_SystemActionGatherResult;
std::future<bool> m_SystemActionApply;

void PrepForSimulate::Execute()
{
    //collate all of the previous frames actions.
    if (!m_TriggeredGather)
    {
        std::unique_ptr<SystemActionGather> temp = std::make_unique<SystemActionGather>(m_FrameContext);
        m_SystemActionGatherResult = temp->GetResultFuture();
        TASK_MASTER::AddTask(std::move(temp));
        m_TriggeredGather = true;
    }
    else if (!m_TriggeredApply && futureReady(m_SystemActionGatherResult))
    {
        m_SystemActionApply = TASK_MASTER::AddTrackedTask(std::move(std::make_unique< SystemActionApply >(m_FrameContext, m_SystemActionGatherResult.get())), this);
        m_TriggeredApply = true;
    }
    else if (m_TriggeredGather && m_TriggeredApply && futureReady(m_SystemActionApply)
        )
    {
        //FrameContext state data should now be const ... from here on out.
        TASK_MASTER::AddTask(std::move(std::make_unique< Simulate >(m_FrameContext)));
        return; // no longer loop
    }
    WantToExecuteAgain();
}
```

RIDDLE

What is infinite in size, but you never seem to have enough of?

C++ STANDARD HIGHLIGHTS
STD::CHRONO

- Very important for animations and logic timing
- Typically you have to be very careful of units in use
    - unsigned int m_TimeStamp: // is this ms? s? ns?
    - unsigned int m_TimeSeconds; // better
    - m_TimeSeconds = 1000 //whoops did you mean 1 second or 1000ms or 1000s
- Chrono’s strongly typed classes are very helpful in knowing context of time
    - Chrono makes the conversions between units obvious and will fail to compile if conversion is invalid

C++ STANDARD HIGHLIGHTS
STD::CHRONO

```cpp
std::chrono::high_resolution_clock::time_point now = std::chrono::high_resolution_clock::now();
std::chrono::milliseconds diff = std::chrono::duration_cast<std::chrono::milliseconds>(now - internal::s_LastFrame);
found->m_TimeStep = std::min(std::chrono::milliseconds(15), diff);
internal::s_LastFrame = now;
```

```cpp
std::chrono::seconds test(1);
std::chrono::milliseconds testms(500);

std::chrono::milliseconds result = test + testms;

std::cout << "" << std::chrono::duration_cast<std::chrono::seconds>(result).count() << "s\n  "
    << std::chrono::duration_cast<std::chrono::milliseconds>(result).count() << "ms\n  "
    << std::chrono::duration_cast<std::chrono::nanoseconds>(result).count() << "ns\n";
```

```
1s
1500ms
1500000000ns
```

C++ STANDARD HIGHLIGHTS
STD::CHRONO (CONT)

```
std::chrono::seconds test(1);
std::chrono::milliseconds testms(500);

std::chrono::seconds result = test + testms;
```

std::chrono::seconds test

no suitable user-defined conversion from "std::chrono::duration<long long, std::milli>" to "std::chrono::seconds" exists

```
std::chrono::seconds test(1);

std::cout << "  " << test.count() << "s\n  "
    << std::chrono::duration_cast<std::chrono::milliseconds>(test).count() << "ms\n  "
    << std::chrono::duration_cast<std::chrono::nanoseconds>(test).count() << "ns\n";
```

```
1s
1000ms
1000000000ns
```

WHAT ABOUT STD::ASYNC?

- Could spin up a new thread for each call.
- Could use same thread to execute.
- A bit more wild west than is desired.

LANGUAGE FEATURES ON THE WAY

- Filesystem Library (C++17) — Available as experimental now

- std::variant (C++17)

    - resource sterilization/de-serialization

- As a statemachine

LANGUAGE FEATURES ON THE WAY (CONT)

- Std::optional (C++17)
  - Removed usage of sentinel objects
    - bool m_TriggeredSomething
    - Std::unique_ptr<Result> m_ResultFromSomething
    - Becomes
    - std:optional< std::unique_ptr<Result> > m_ResultFromSomething
- Various Refinements

LANGUAGE NICE TO HAVE THINGS

- Pool alloc/dealloc of objects (lock free pool)
- “Pure” keyword to make pure virtual functions in class/struct
  - Virtual void Function () =0; ==> pure void Function (); or void Function () =pure;
    - P0078R0: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0078r0.pdf
- Standardized Network Lib
  - N4332: Networking Library Proposal (https://isocpp.org/blog/2014/11/n4332)

LANGUAGE NICE TO HAVE THINGS CONT

- Ability to set a name/affinity to a thread through std … (if not using async/coroutines)
- std::atomic< std::unique_ptr<Object> >
  - N4058: Atomic Smart Pointers (https://isocpp.org/blog/2014/06/n4058)
- std::future<>.is_ready() … (MS has one as part of their implementation _Is_ready)?
- std::future.valid() is not as clear as expected.
  - Valid does not mean connected to promise of otherwise initialized (So use of sentinel is required. Also after get … state seems undefined)

LANGUAGE NICE TO HAVE THINGS CONT

- constexpr does not always mean compile time executed constexpr functions
    - const int test = constexpr_hash("TEST"); // This is executed at runtime
    - constexpr const int test = constexpr_hash("TEST"); // This is executed at compile time
- would like
    - const int test = constexpr_hash("TEST"); // This is executed at compile time
    - const int test = 12345678; //after compile time executes
    - constexpr const int test = constexpr_hash("TEST"); // This is executed the same

LANGUAGE NICE TO HAVE THINGS CONT

- smart_ptrs that can handle forward declarations with default destruction.

    - class ClassName; (implementation in ClassName.h/.cpp) using default destruction

- std::unique_ptr<ClassName> m_Ptr; //fails to compile

REFERENCES

- www.cppreference.com
- www.isocpp.org
- GDC 2015: Christian Gyrling Parallelizing the Naughty Dog Engine Using Fibers
    - http://www.gdcvault.com/play/1022186/Parallelizing-the-Naughty-Dog-Engine
    - http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Gyrling_Christian_Parallelizing_The_Naughty.pdf
- C++ Seasoning Sean Parent https://www.youtube.com/watch?v=qH6sSOr-yk8
- CppCon 2015: Sean Parent "Better Code: Data Structures" https://www.youtube.com/watch?v=sWgDk-o-6ZE
- CppCon 2014: Mike Acton "Data-Oriented Design and C++" https://www.youtube.com/watch?v=rX0ItVEVjHc
- CppCon 2014: Nicolas Fleury "C++ in Huge AAA Games" https://www.youtube.com/watch?v=qYN6eduU06s
- Modern C++: What You Need to Know https://www.youtube.com/watch?v=TJHgp1ugKGM
- CppCon 2015: Nicolas Guillemot & Sean Middleditch “Birth of Study Group 14..." https://www.youtube.com/watch?v=0tFxdreVx6s
- CppCon 2015: Michael Wong “C++11/14/17 atomics and memory model..." https://www.youtube.com/watch?v=DS2m7T6NKZQ
- CppCon 2015: Fedor Pikus PART 2 “Live Lock-Free or Deadlock (Practical Lock-free Programming) https://www.youtube.com/watch?v=1obZeHnAwz4
- CppCon 2014: Herb Sutter "Back to the Basics! Essentials of Modern C++ Style" https://www.youtube.com/watch?v=xnqTKD8uD64

Summary

- Standardized code is easier to maintain
- Standardized code is broadly tested
- C++11 offers many useful features for games
    - std::thread
    - smart pointers
    - std::chrono (timing lib)
    - std::future,std::atomic
    - constexpr

jasonjurecka@gmail.com / jjurecka@blizzard.com