GDC 12

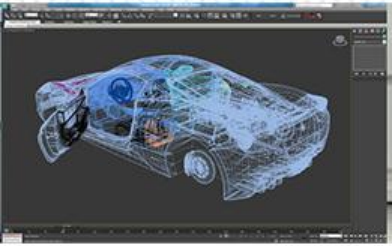
GAME DEVELOPERS CONFERENCE

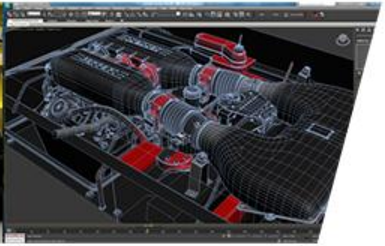
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9

2012

FORZA 4
PIPELINE ARCHITECTURE

Daniel Caruso
Lead Software Development Engineer
Turn 10 Studios





Pipeline and Tech Art

Combined “scrum” team of developers and technical artists responsible for designing, building, and maintaining Turn 10’s production pipeline and tools.

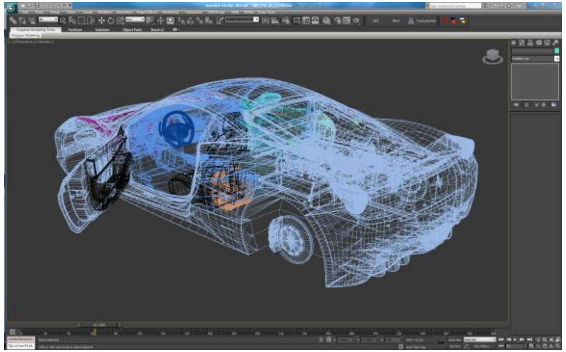
- Daniel Caruso (Lead SDE)
- Arthur Shek (Technical Art Director)
- Tatyana Dyshlova (PM)
- Peter Beck (SDE)
- Rob Fulwell (SDE)
- Nathan Holt (SDE)
- Chad Olsen (SDE)
- Ryan Petrie (SDE)
- Dan Tunnell (SDE)
- James O’Donnell (Technical Artist, AMAXRA)

Road Map

- The problem space
- Requirements
- Software architecture
- Hardware architecture and services
- Better performance through distributed computing

Mission of Pipeline

- Transform all the assets produced by the studio from source material to a working, executable game.



Pipeline

- This is not as simple as it used to be

Explosion of Complexity

- Volume and diversity of content
- Advanced asset optimization and offline pre-compute
- Security and asset protection
- Schedules and team size
- Parallel work efforts
- Licensing and legal

Forza 4 Content Complexity

Cars

- 500 cars, a 30% increase over Forza 3 with the same production timeline
- Accurate to < 2 centimeters
- AutoVista an interactive car experience <link autovista video>
    - 1,000,000 poly models, including engines and interiors
    - Accurate to sub centimeter
    - Complex animations, voice over, 3D UI and accurate car start sequences
    - Touched by all disciplines in the studio

Environments

- 26 core environments with over 100 course variations
- 20-40k files per environment
- Art assets too big to fit into 32 bit address spaces
- Entire environment is to big to work with even on high end 64 bit workstations
- Accurate reproduction of very large outdoor spaces

Forza 4 Content Complexity Cont.

- Over 100 different content and code processing tasks required to transform assets into a working game.
- Over half a million individual code and content source files totaling over 500GB of data
- Nearly 6TB of reference data, images and video (and growing)
- 500GB source ->Pipeline ->11.8 GB final game size on disk

Forza 4 Workflows and Staff Size

- All 100+ pipeline stages must be executed correctly
- Work must be coordinated with 350+ world wide artists, 30+ developers, 13 designers, 24 in house testers and 9 Producers
- No asset/game feature is completed by a single individual
- Individual assets must be worked on by multiple individuals concurrently
- The pipeline system as a whole, must support a large number of heterogeneous compute tasks executed in a specific order depending on the source asset type and the game consumable item it is part of.

Pipeline Mission Evolved

- Transform all the assets produced by the studio from source material to a working, executable game by building an infrastructure that can define and execute a diverse set of asset pipelines and combine their results
- Allow users to collaborate while protecting them from each other and themselves
- Manage and secure a large data set and ensure seamless access to data and compute resources both on site and on the other side of the globe
- Minimize down time and prevent studio wide production blocks

Pipeline Design Starts With Studio Process

All assets included in the official game build will be built by the build system servers directly from source files. No locally built files will be checked-in.

Solves

- Assets getting built incorrectly
- Assets getting built by combining stale previously built components
- Variance caused by heterogeneous user machine hardware
- Assets built with old or “custom” versions of the build tools
- Ensures that a assets can always be re-built from source alone

Problems

- Gates production on the overall build server bandwidth
- Requires fully automated builds off all asset

Mitigations

- Invest in sufficient server bandwidth
- Invest in creation of automation

Artists will only work on content using the last known good (LKG) tools and pipeline paired with a specific stable BVT passing build

Solves

- Content creators don’t get blocked by instabilities caused by code churn

Problems

- Latency in delivery of new features to content developers
- Latency for emergency fixes for content developers

Mitigations

- Keep LKG process light weight and mostly automated
- Run LKG at regular intervals at least 2x per week
- Provide emergency overrides for using the latest code if there is no other choice

The official game build will only contain "safe" content which has passed a set of automated BVT's and been promoted from "escrow" to "release".

Solves

- Test, design and developers getting blocked by bad content

Problems

- Latency of getting new content into game
- Content testers need to test latest content

Mitigations

- Create a automated content BVT which ensures content doesn't crash the game
- Automate promotion of BVT passing content
- Provide easy to use tooling for working with content directly from escrow

All content check-in's will pass a local data scrubber without errors prior to check-in

Solves

- Content not build to naming conventions and other required specifications, such as poly count limits
- Common modeling errors
- Units, bad dimensions, overlapping verts, etc.

Problems

- Overhead of running the scrubber
- Over detection of problems that are not blocking

Mitigations

- Keep scrubbing times low
- Only report errors for known breaking issues, otherwise log warnings

All code changes will pass though a rigorous code-review, check-in build and BVT process prior to check-in

Solves

- Common mistakes
- Ninja features
- Major performance regressions
- Major memory bloat and leaks
- Major regressions of core functionality
- Build breaks caused by not using the official build processes

Problems

- Adds latency to check-ins

Mitigations

- Keep build times, and BVT times as low as possible
- Totally worth the cost due to gains in overall production stability

Developers don’t go home until their code changes have been successfully integrated and built on the servers by a continuous integration system

Solves

- Check-in and go home only to leave behind a failed build in the morning
- Breaks caused by improperly integrated files
- Breaks caused by partial or otherwise erroneous check-ins

Problems

- If there is a queue of check-ins it can take several hours

Mitigations

- Check-in tomorrow if you don’t have the time today
- Group multiple check-in’s into a single server build if they are very close together (i.e. submitted within 10 minutes)

The Requirement Wall

- Continuous integration
- Concurrent asset editing
- Offsite workflows
- Automated source control synching
- Automated content acceptance tests
- Automated pre-check-in tests
- Automated performance tests
- Multiple concurrent builds
- Protect build stability from churn
- Protect artists from broken tools
- Data driven build process definition
- Fully automated production workflow
- Asset encryption, signing and hashing
- Logging and error reporting
- Disk layout creation and optimization
- Platform specific packaging
- High throughput
- Fast build times
- High availability
- User facing UI
- User facing scriptability
- Predictable/repeatable results
- Automated build deployment
- Robust back ups
- Scalable to production demands
- Support hundreds of assets types
- Notifications/subscriptions
- And more….

Walls Usually Win

Crawl, Walk, Run

Crawl

- Focus on the basic asset transformation problem
- Targeted only on the local machine
- Run each asset's build step's directly

Walk

- End to end automated systems on the local workstation
- Execution of local build processes from a service

Run

- Run build processes remotely on servers
- Connect execution of processes together to automate complex end to end workflows

Software Architecture

Micro and Macro Pipelines

Micro pipelines are the specific set of processing stages executed in a specific order required to transform a set of source assets into a single game consumable asset

- We sometime use the term “Build” or “Build Process” and erroneously “Export”
- May be executed on a local machine, on a server (often as part of a Macro Pipeline), or on a computational grid
- The pipeline is identical regardless of execution context,

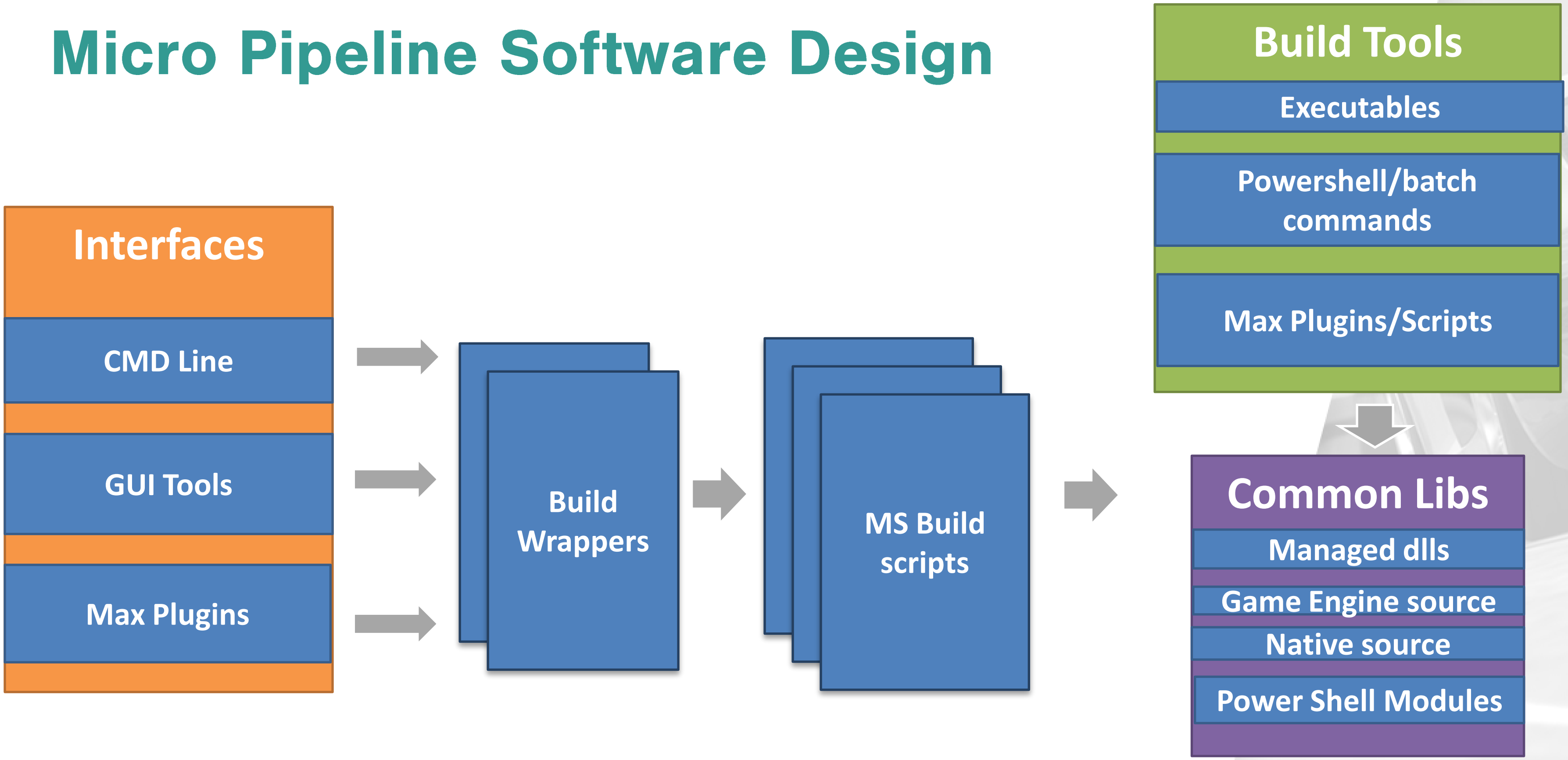
Macro Pipelines represent the automated process by which source assets move through studio workflows finally ending up included in the daily official game build.

- Executed in a variety of contexts depending on the various systems and individuals involved
- Implement studio “production process”
- Consists of at least one, but often many diverse micro pipelines,
- Provides the **ONLY** pathway for an asset to get into the official daily game build

Micro Pipeline Design Overview

- Use a collection of “build tools” to process asset source files
- Use MSBuild to define and execute these processing stages in the correct order
- Build user friendly wrappers for running specific MSBuild targets with the correct parameters

Micro Pipeline Software Design



MSBuild

- Comes with the .Net framework
- Allows arbitrary execution of executable, scripts and managed code
- Allows for dependency definition
- Supports parallel execution
- Provides an execution engine that is usable from both the command line, and managed code API's
- Provides logging and error reporting mechanisms
- Easy access to parameters through environment variables and command line arguments
- XML based and API support for programmatic construction of build scripts

Targets and Tasks

Targets

- Used to define entry points
- essentially an in-order list of things that need to be executed
- Supports dependency expression

Tasks

- Specific instructions to get executed
- See the **MSBuild Task Reference on MSDN** http://msdn.microsoft.com/en-us/library/7z253716.aspx
- Custom tasks can be created by combining existing tasks, using managed code or by simply running custom executables
- Task can be executed conditionally and specify on error behavior

Macro Pipeline Design

Basic Approach

- Use a collection of servers to execute arbitrary micro pipelines.
- Build a set of “Jobs” which execute collections of micro pipelines as required by a specific macro pipeline
- String these jobs together to form the complete macro pipeline
- Provide a simple way for end users to execute the Job Sequence

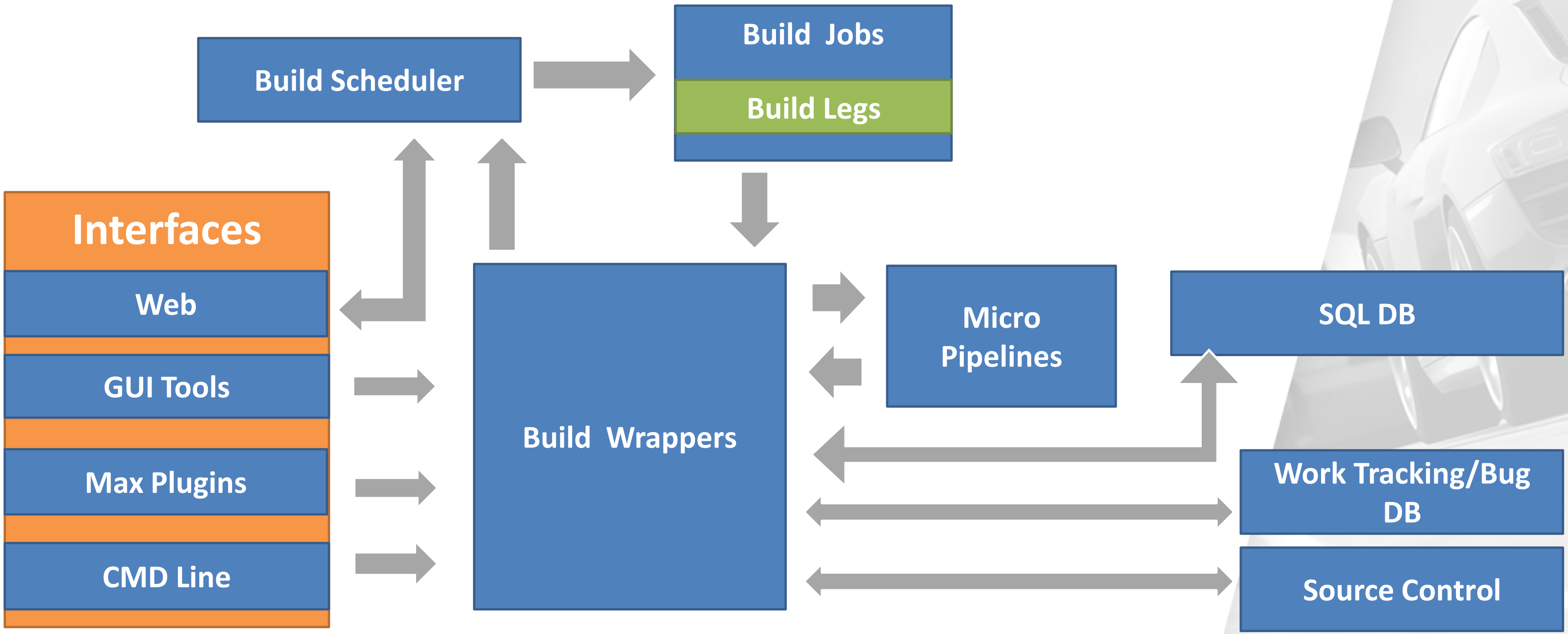
Difficulties

- Macro pipeline stages may begin with locally executed tasks, such as data scrubbers, local tests, check-in wizards, etc.
- Macro pipeline stages may include some manual steps, and require state to be saved while waiting for user input before proceeding to the next step
- Macro pipelines often require the output of other macro pipelines as input

Solutions

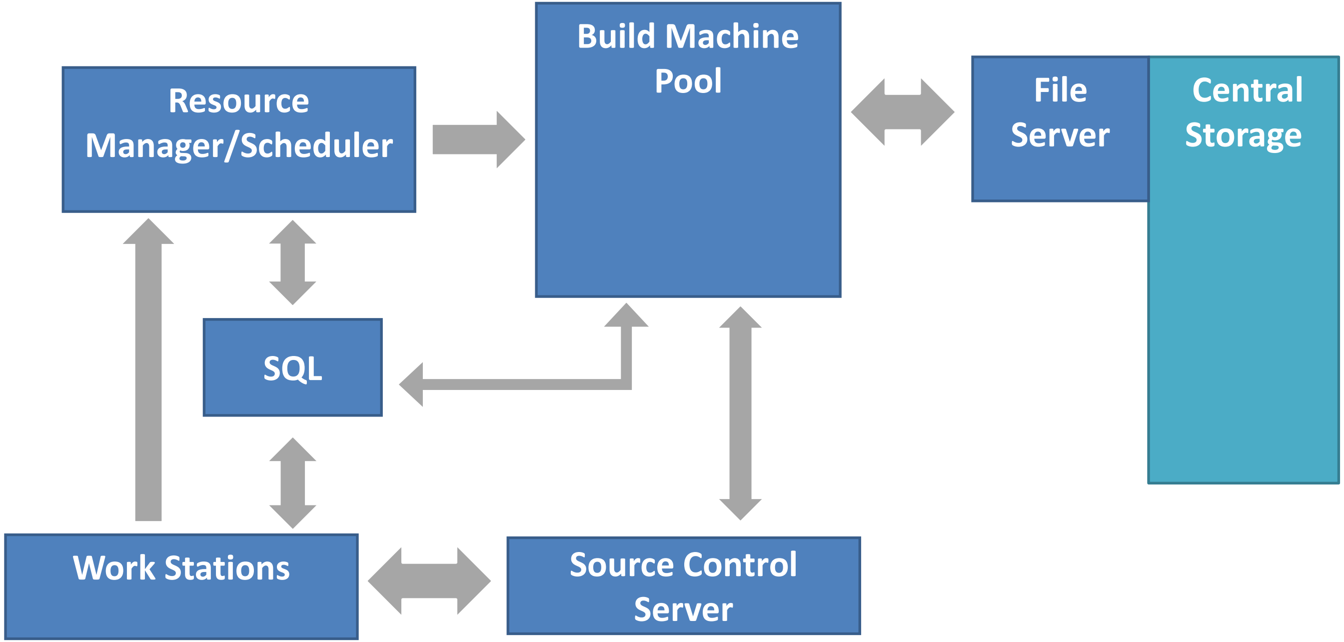
- Use Micro pipelines which schedule jobs on a server farm to execute other micro pipelines
- Build user facing tools to call these micro pipelines as appropriate
- Use source control and SQL to persists state
- Send e-mail and or automatically create and assign bugs out for manual process stages

Macro Pipeline Software Design



Hardware Architecture

Basic Infrastructure Components



Resource Manager and Job Scheduler

Acts as the operating system for the machine pool

- Manages Machine pools
- Manages Machine pool job queues
- Balances resources across jobs
- Provides a user facing service for requesting and monitoring jobs

Key Capabilities

- Job definition language
  - Express a dependency graph of related tasks
  - Allows a job or job task to schedule additional jobs programmatically
- Distributes sub tasks across machine pool resources based on dependency graphs
- Persistent storage of current working state, long term job execution history, and logs
- Fail over and recovery solutions

Forza 4 Solution

- Microsoft internally developed distributed job scheduling service called BuildTracker, built on top of SQL, XML, and Web Services

Computer Machine Pools

- Sets of servers dedicated to executing specific sets of micro pipeline tasks
- Each Machine Pool effectively has it's own "job" queue
- Each server has a "enlistment" into source control, a sizeable raid array, and high bandwidth connection to centralized storage
- Each server runs a service which connects to the job scheduler and is responsible for executing jobs assigned by the scheduler

Source Control Server

- Manage change history of source files
- Provides a basic concurrency management system for collaboration
- State management system for “built” assets that are not ready to be included in the official Build
- Stores final approved “built” assets
- Microsoft internal source control system
- Team Foundation Server (TFS) (More on this later)

SQL Server

- Hosts the working database for the scheduler
- Hosts a mirror of the latest in-game database
- Hosts several custom databases for storing state information
- Hosts test result and game performance databases

Workstations

- User machines
- Each machine has a source control enlistment
- Connection to Shared Storage
- Access to the SQL server
- Hosts running various user facing tools which can initiate execution of a macro pipeline and report overall execution progress
- Can serve as an additional compute resource in some cases

But what about the hardware?

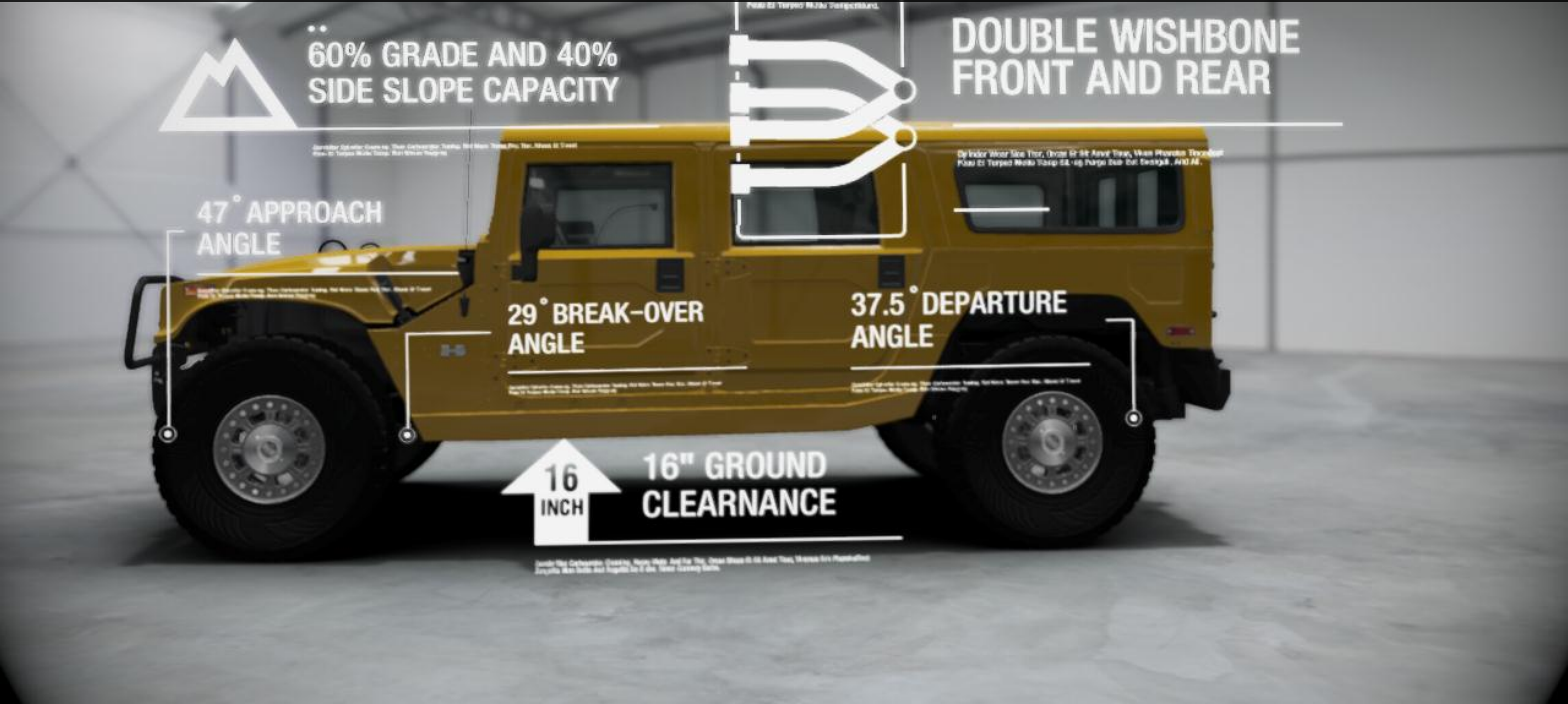
A simple approach would be to back each component with one or more dedicated and specialized machines

This strategy is very limiting:

- Does not scale well
- Is not able to adapt to changes in the component architecture quickly
- Is hard to maintain
- Is bad for availability
- It is not cost effective

But, In a small production environment this can be viable

Scaling and Reliability



Services

Pipeline services

- Software services which provide key pipeline features, such as the Job Scheduler, license servers, continuous integration, work item tracking, test automation, etc…
- Persistent processes which expose an API to TCP

Simple services

- Low compute requirements
- Low disk requirements
- Data base and web server dependencies

Advanced or specialized services

- Heavy compute, disk, memory or I/O requirements
- Physical access to connected hardware, such as usb devices, dongles, Xbox 360 dev kits, etc.
- Specialized hardware such as a high power GPU, or specific CPU architecture

Hosting Simple Services

Virtual machines (VM’s) can be used to great effect

- Service failures are isolated and won’t affect other services being co-hosted on the same physical machine
- Host machines can be configured in failover clusters
- Machine images can easily be moved to different hardware in case of a failure
- New services can quickly be added by creating a new machine image from a existing base machine configuration

To get the most out of virtual machines you will need a few things

- A handful of servers heavy on CPU and memory to run hypervisors for VM hosting
- A SAN system which hosts the VM images and provides pass through disks for local VM storage

Storage

Efficient VM hosting relies on a flexible, reliable, shared storage system

SAN’s (Storage Area Networks) provide the best technology available to meet these needs

- Entry level, turnkey, full featured SAN’s are available for <$100k
- Costs can be controlled by trading performance, capacity, and bandwidth as your needs demand, with some entry level options starting way bellow the 100k price point
- Traditional NAS (network attached storage) solutions can work in a budget pinch, but you will find you quickly get to 50% of the cost of a SAN without nearly 50% of the benefits
- SAN’s are good for more than just VM’s

Network Shares and SAN’s

- Traditional network shares can be robustly supported on top of a SAN
- A fail over cluster can mount a SAN volume and expose it to the network
- Cluster files systems can further extend this by enabling multiple servers to expose the same SAN volume to the network
- Combined with a load balancer this can deliver massive bandwidth

Hosting Advanced/Specialized Services

Requires dedicated physical machines

- Expect long hardware lead times
- Expect delays due to physical installation

Can usually leverage SAN attached storage

- A single host bus adapter card (HBA) is often cheaper than a robust hardware raid controller, and a bigger physical box
- As long as you aren't stressing the SANs aggregate bandwidth, performance will usually be better than local disc

Critical systems will need to have custom clustering or fail over solutions

- These are the weakest link in stability
- When possible store all state data in a high availability data base solution on a separate set of machines
- Keep frequent back ups
- Keep machines within warranty and replacement components on hand

As much as possible standardize these configurations ahead of time

High Availability Database

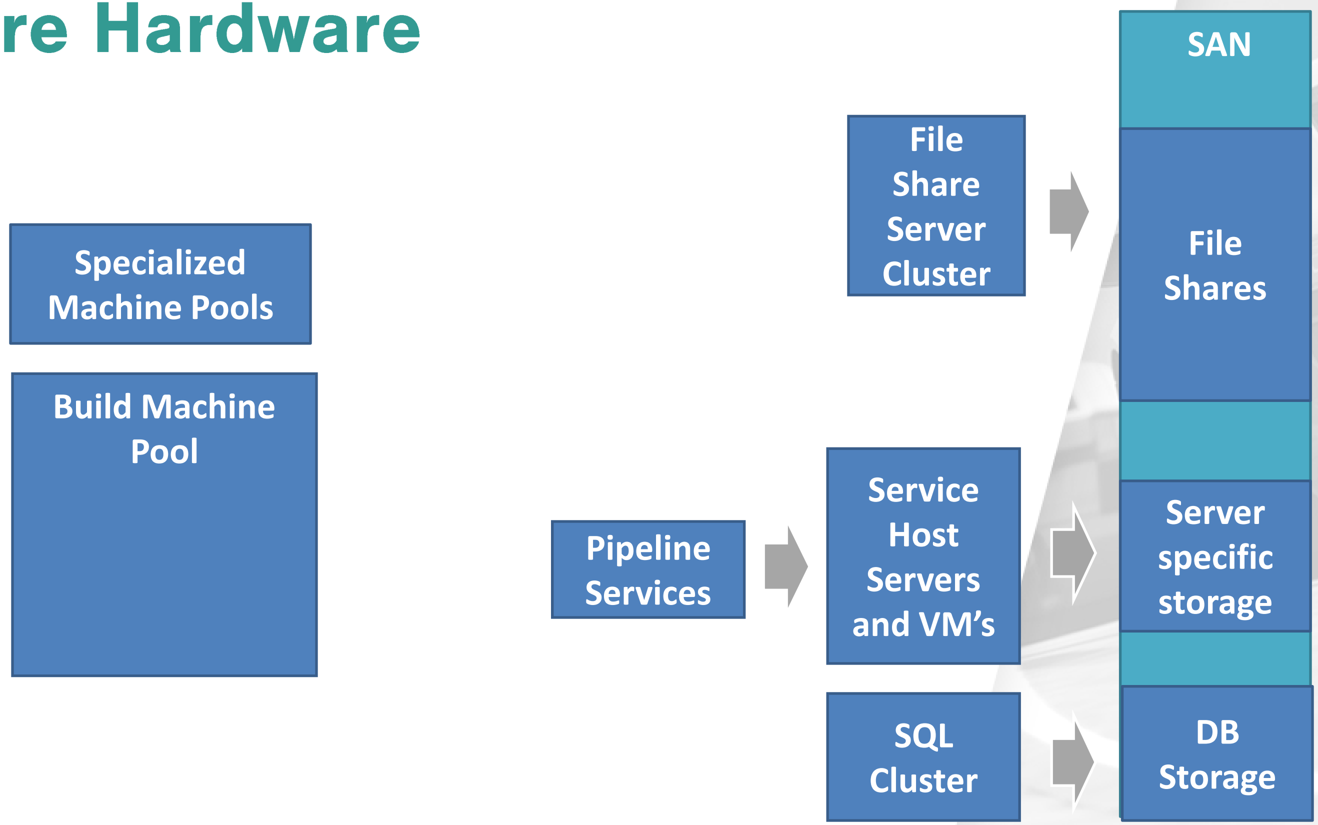
SQL fail over clusters are awesome!

- Multiple SQL instances can be setup to accommodate your overall performance requirements
- Leveraging a central storage solution you can then set these clusters to fail over to each other incase of any problems
- This will reduce performance but keep the system up and running

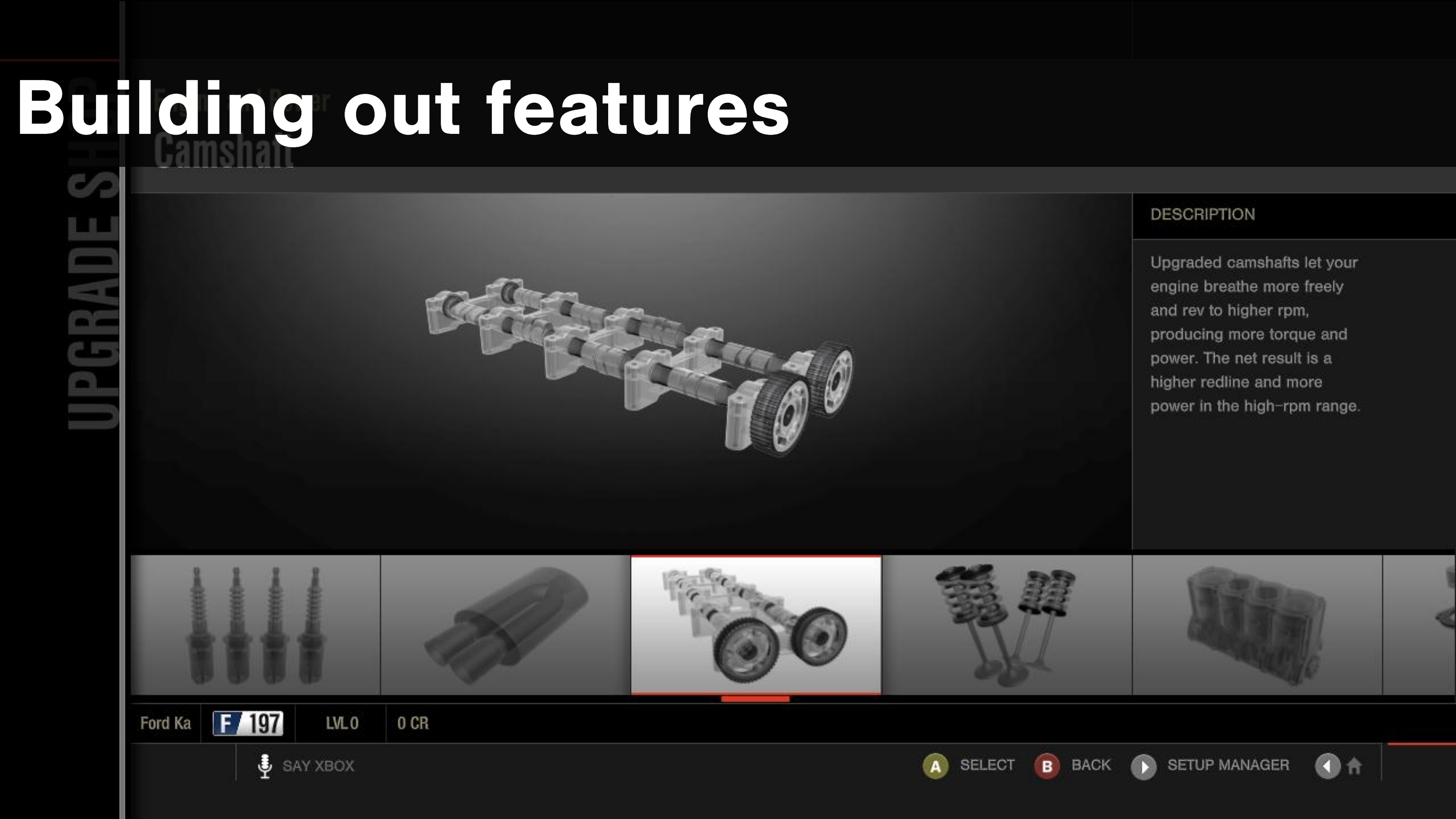
Go big on SQL hardware

- These are shared resources bang for the buck is high
- Lots of CPU (8+ cores) and big memory pools (> 32 GB)

Infrastructure Hardware



Building out features



Additional Services Needed

- Access for offsite content creation
- Continuous integration system
- Test automation system
- Mechanisms for providing a Last Known Good set of tools
- Mechanisms for providing an Escrow to Release promotion process
- Specialized computational grids for accelerating particular micro pipelines

Escrow, Release and Bin

Problem

- Provide a stateful storage mechanism which supports the escrow to release process as well as stores the most recent Last Known Good (LKG) tools and pipeline binaries

Solution

- Create special locations in source control for managing these “built” assets
- Escrow folder
  - All freshly built assets are checked in where they can be pulled from source control later for testing
- Release folder
  - After an asset has passed BVT’s and been “promoted” it is checked into the release folder where the official build will look for content assets.
- Bin folder
  - Build wrappers, MS Build files, Build Tools, scripts, and common code libs are check-ed in when built
  - A label is used to manage the LKG set of assets in the bin folder

Access for offsite content creation

Problem

- External vendors doing content production can not access internal source control servers due to security and other reasons beyond our control

Solution

- Move all content production to a Team Foundation Server (TFS) hosted in a extranet domain with a 1 way trust relationship to our main domain
- Run a replication service on our domain which Polls the TFS server for incoming change sets, copies them over to the replication server and then integrates them into its source depot enlistment

Hardware

- Beefy servers for running the TFS instance and replication service

Continuous Integration System

Problem

- Schedule a “buddy build” for each code check-in made

Solution

- Build a service which monitors the source control system changes to a given set of folders and file types
- When ever a change is detected que a “buddy build” Job on the job scheduler

Hardware

- Simple service host machine

Test Automation System

Problem

- Integrate automated test into the pipeline

Solution

- Build a test execution engine based on MsTest (the command line version of visual studio's test suite)
- Define specific test jobs which can be executed by a specialized machine pool
- Use a TFS server for centralized test result logging
- Add a specialized data base for detailed performance metrics
- Add a performance data processing service which adds collected data to the performance DB
- Add a build distribution service used by the test system to manage which game build and content is on test xbox 360's

Hardware

- Machine pool consisting of light weight physical machines, paired one to one with Xbox 360 developer kits
- Simple Service host machine for the data base processing system

Specialized Computational Grids

Problem

- Accelerate parallel computation tasks by providing access to large computational grids

Solution

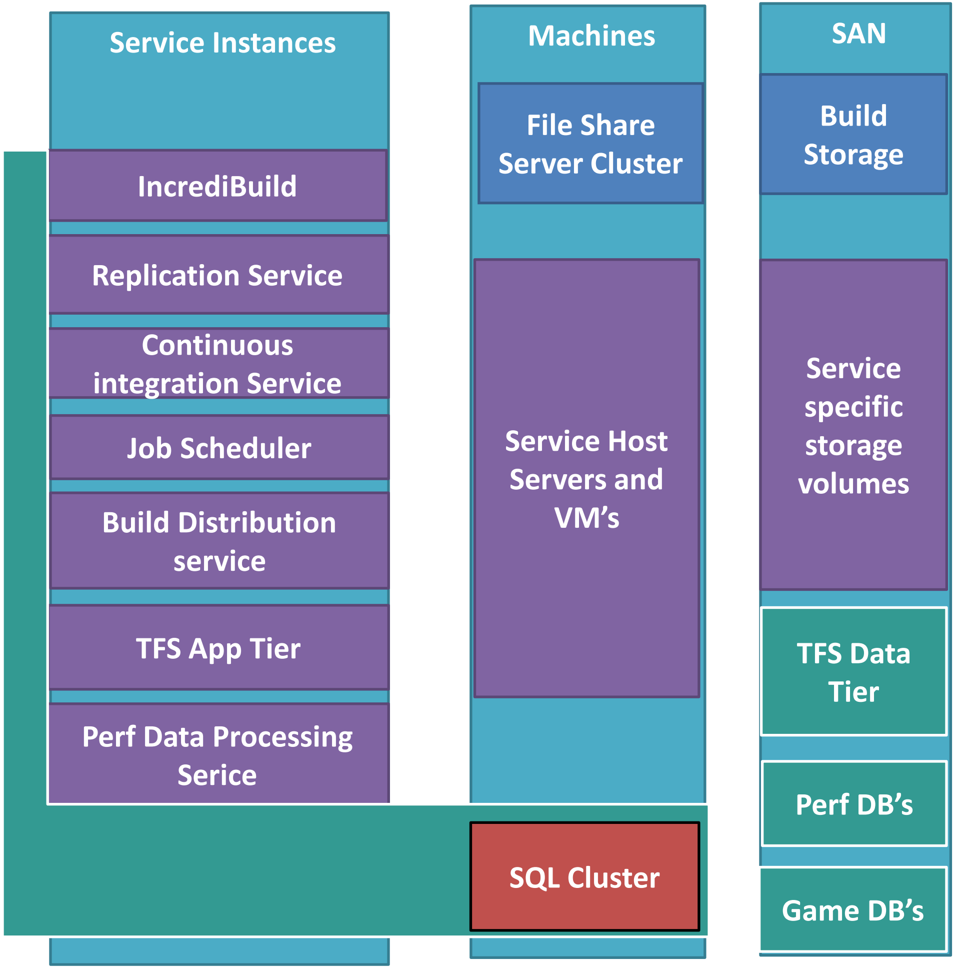
- Add Wrappers for interacting with Autodesk Backburner Add Wrappers for interacting with Incredibuild for native code builds

Hardware

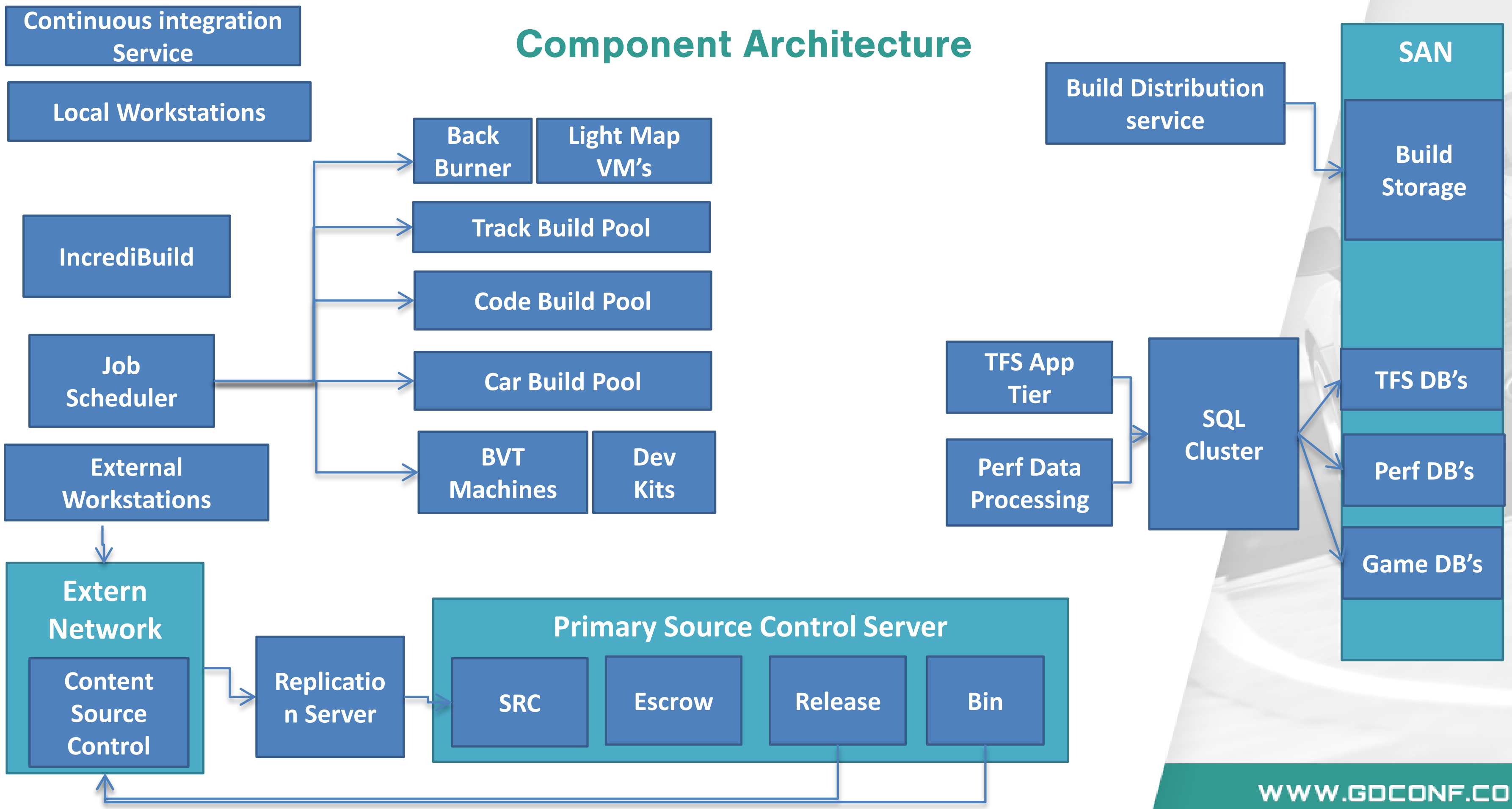
- Coordinator machines which host the grid controller
- A slew of physical machines, VM's, and user workstations connected to the grid

Putting it all Together

The Service Stack

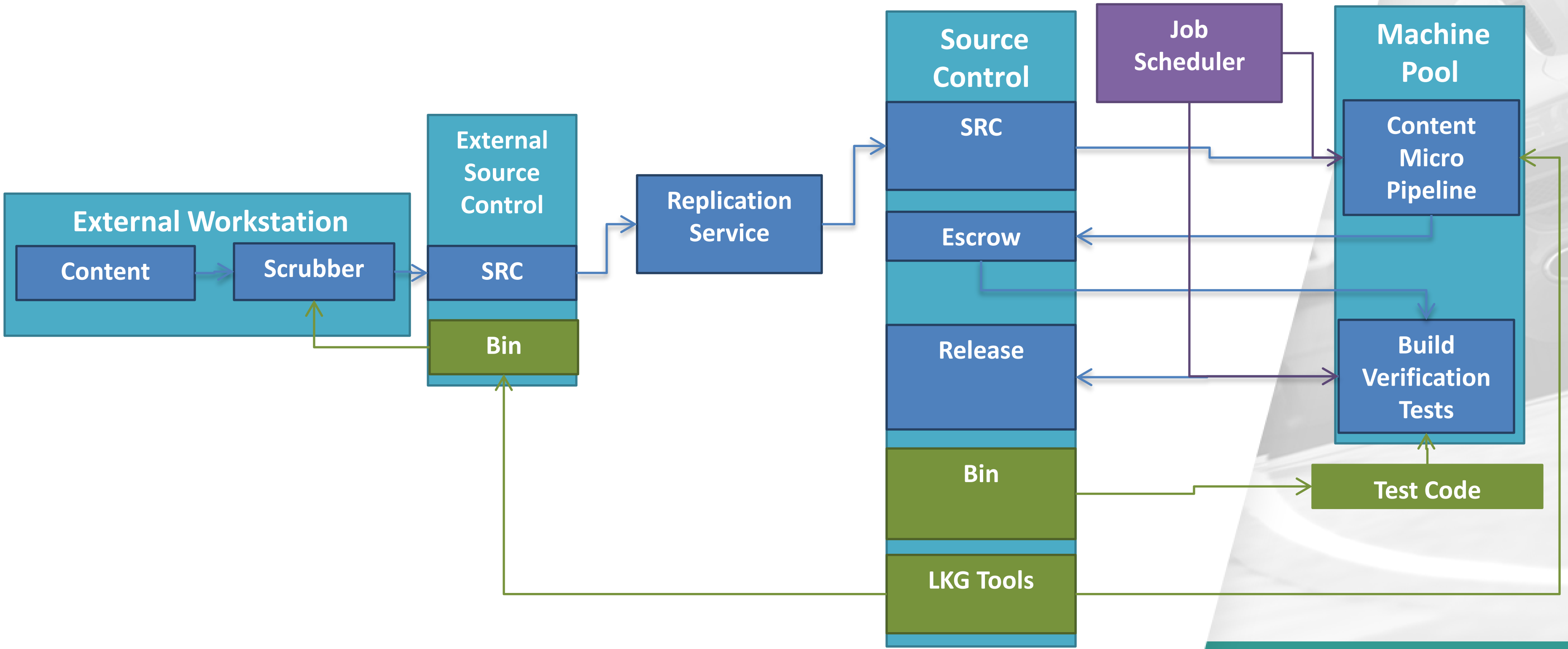


Component Architecture

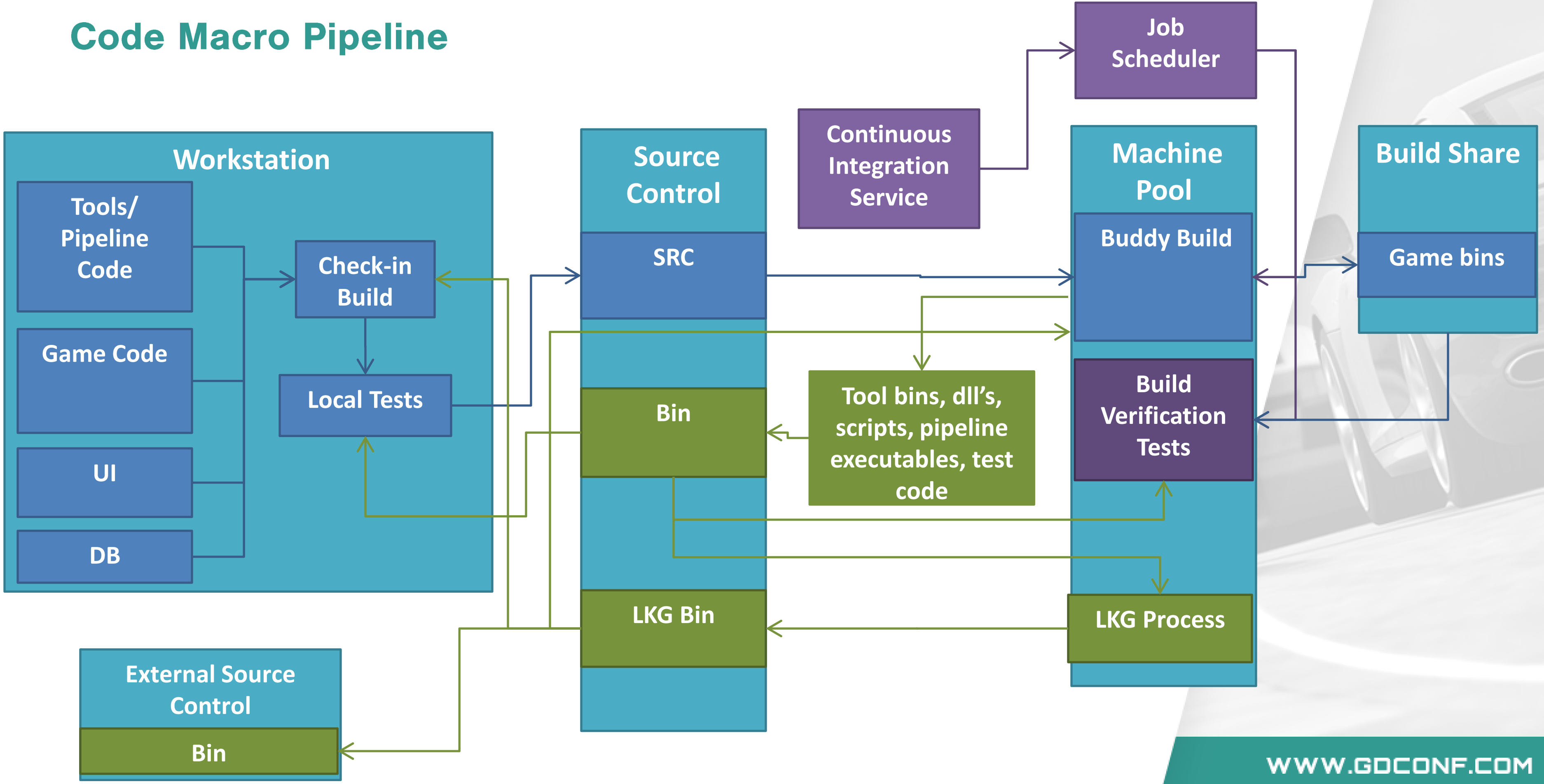


Macro Pipeline Examples

Content Macro Pipeline



Code Macro Pipeline



Go Faster With Distributed

Forza 4 Daily Build

- Takes ~4 hours to complete
- It uses 10x 8 core machines
- Estimated 20+ hours linear execution time
- The long running non distributes tasks account for most of this time
- During these tasks we only use 1-3 machines due to dependencies
- Our estimates predict a build time of less than 1 hour if we can break up each of these tasks into at least 3 equal sub tasks

- But that's all theoretical…

Distributed Car Builds

Initial Forza 4 car builds where over 24 hours

Optimization reduced this to

- Full AutoVista Car export 8-12 hours
- Full non AutoVista export 4-6 hours

Non mesh edit Incremental Builds

– Less than 10 minutes, for ~70% of builds
– Still average ~25 full builds per day during peak production

Can we leverage Xorax's XGE system to accelerate car builds?

Yes!

Distributed Car build cont.

90% of car build time is spent in single threaded computation of directional occlusion and illumination

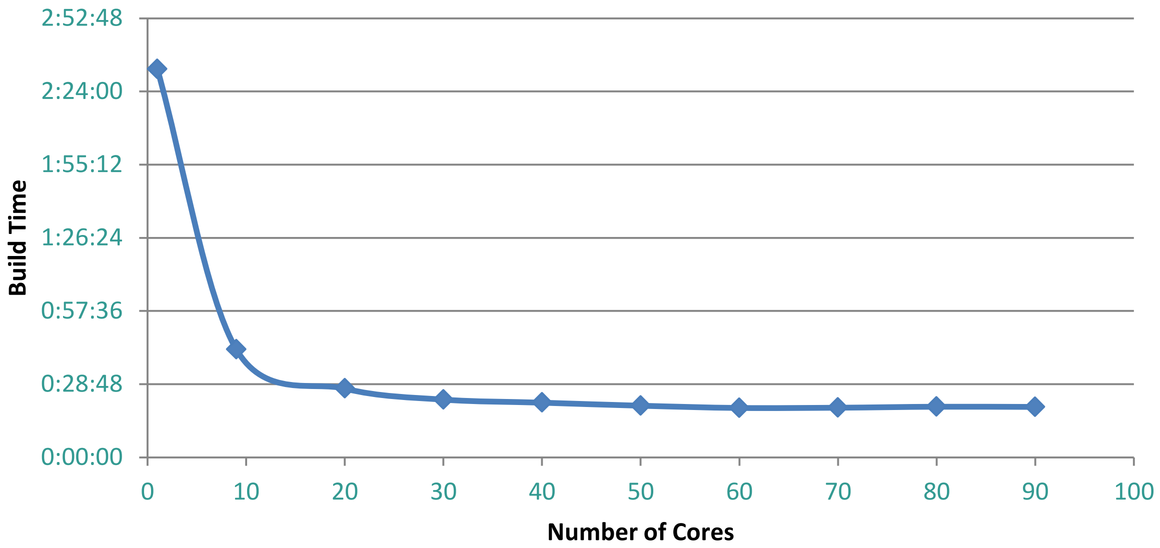
Computation of per vertex/per ray data is independent within a illumination bounce

A multi pass parallel algorithm can be used

- For each bounce read all previous bounce data
- Run direction occlusion and illumination in parallel for all vertices
- Write bounce data
- Repeat


Setup
Bounce 1
Bounce 2
Bounce 3
Consolidate
Build Monitor - 0 error(s), 0 warning(s)
File View Help
Agents
Progress
Win-deboagj2ch9 (CPU 1)
Win-deboagj2ch9 (CPU 2)
Win-deboagj2ch9 (CPU 3)
Win-deboagj2ch9 (CPU 4)
Turn10-artshek (CPU 1)
Turn10-artshek (CPU 2)
T10bttest02 (CPU 2)
Turn10-artshek (CPU 3)
Turn10-artshek (CPU 4)
T10bttest02 (CPU 1)
T10bttest02 (CPU 3)
T10bttest02 (CPU 7)
T10-nholt01 (CPU 1)
T10bttest01 (CPU 4)
T10bttest01 (CPU 5)
T10bttest02 (CPU 5)
T10bttest02 (CPU 6)
T10bttest02 (CPU 4)
T10bttest01 (CPU 6)
T10bttest02 (CPU 8)
T10bttest01 (CPU 1)
T10bttest01 (CPU 2)
T10bttest01 (CPU 3)
T10bttest01 (CPU 7)
T10bttest01 (CPU 8)
T10-nholt01 (CPU 8)
Vash (CPU 3)
Vash (CPU 4)
T10-nholt01 (CPU 6)
T10-nholt01 (CPU 3)
T10-nholt01 (CPU 2)
T10-nholt01 (CPU 5)
T10-nholt01 (CPU 7)
T10-nholt01 (CPU 4)
Vash (CPU 1)
Vash (CPU 2)
Rdmgslturnten04 (CPU 1)
Rdmgslturnten04 (CPU 2)
Rdmgslturnten04 (CPU 3)
Rdmgslturnten04 (CPU 4)
ModelCompiler
0 CPUs, 0.0 GHz
2:00
4:00
6:00
8:00
10:00
12:00
14:00
16:00
18:00
Done

Actual Performance



- AutoVista (1M poly) Cars 1.2-1.5 hours from 8-12 hours
- Non AutoVista(400k poly) Cars <25 minutes from 2-4 hours

Closing Arguments

- Growth in content complexity, studio size, geographic diversity and data set size is rapidly increasing the difficulty and scope of pipeline work
- Solving problems of this scale requires good engineering and cross team collaboration
- The Pipeline is the literal implementation of the overall studio software process
- A services approach to pipeline design allows us to leverage leading edge IT technology and solutions for a scalable, flexible, and resilient solution
- Distributed computation is an fundamental technique for reducing iteration times

Acknowledgments / Questions

Special Thanks:

Erin Devoy, Chris Tector (Turn 10), Daniel Adent (Turn 10), Colin Reed (Turn10), Cory Ross (Turn 10), Chris Butcher (Bungie/GDC), GTO Development Team (Microsoft Studios), Games IT (Microsoft Studios), Build Tracker Team (Microsoft), BG IT (Microsoft)

Software:

- Team Foundation Server : http://msdn.microsoft.com/en-us/vstudio/ff637362
- Incredibuild and Incredibuild XGE : http://xoreax.com/
- Power Shell : http://technet.microsoft.com/en-us/scriptcenter/dd742419
- MS Build : http://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx

Spherical Harmonic Lighting:

- P. Sloan, J. Kautz, and J. Snyder, "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments," *ACM Trans. Graphics,* pp. 527-536, 2002.
- Spherical Harmonic Lighting: The Gritty Details, Robin Green, http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.pdf

Appendix

Detailed Car Build Time Slides

Car Build Times

Non linear increase in content build times

- Forza 4 Million Poly models initial build times over 24 hours
- V3 build times where 3-5 hours
- Kills Iteration!

After Optimization

- Full AutoVista Car export 8-12 hours
- Full non AutoVista export 4-6 hours

Still not fast enough

Incremental Builds

Non-predictable dependency chains for Direction Occlusion and Illumination

Vast majority of changes are not to mesh data (70%)

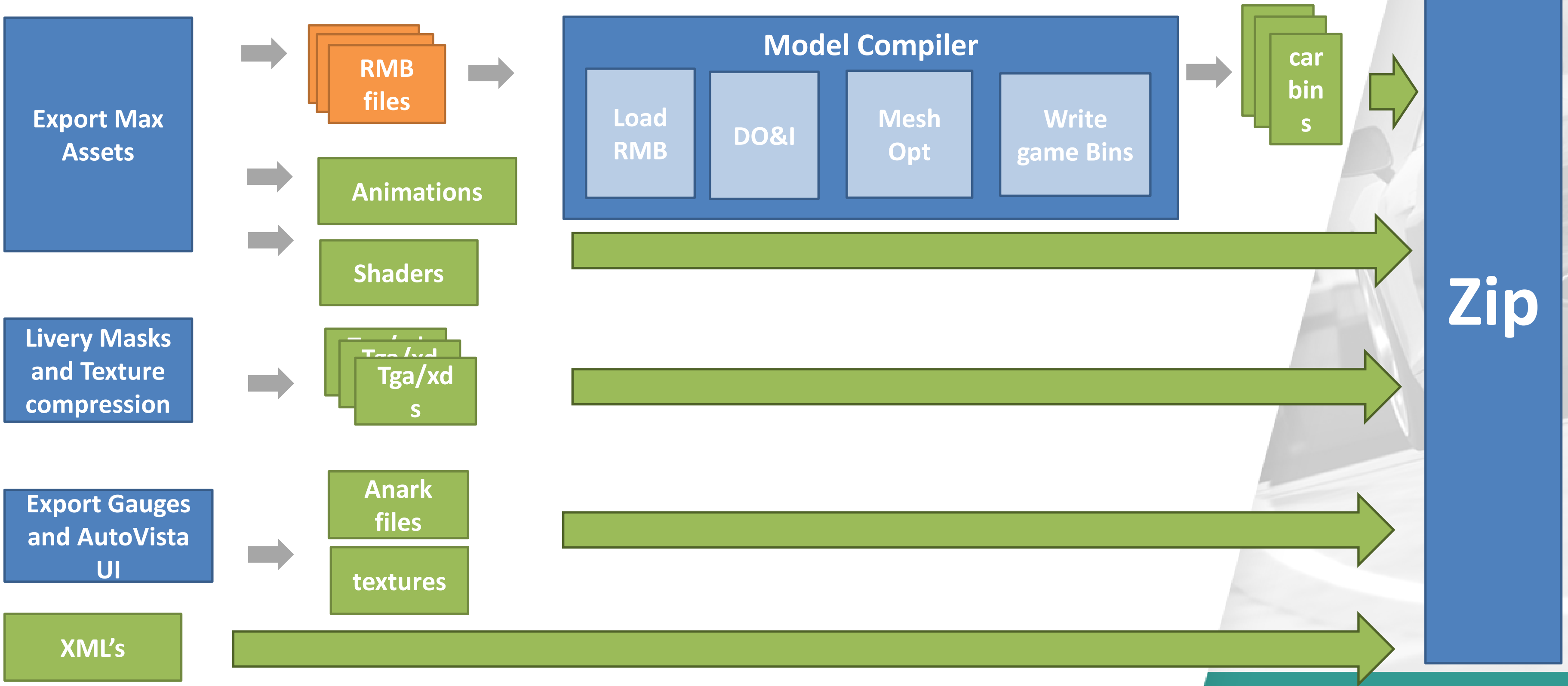
- No need to re-run max export or Directional Occlusion and Illumination for non mesh data changes

Non mesh edit builds < 10 minutes

Massive help but we still have to do a lot of full builds

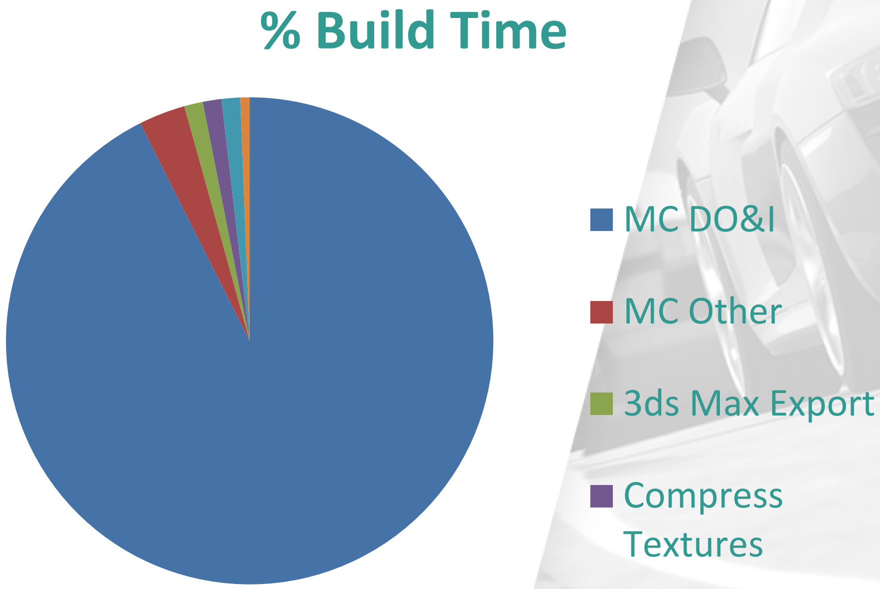
- ~25 full car builds per day during peak production

Car Micro Pipeline



Build Time perf analysis

90% of build time in Directional Occlusion and Illumination calculation from within the “Model Compiler” Tool



Breaking open the "Data Compiler"

Analysis of model compiler Directional Occlusion and Illumination code

- Per Mesh
  - Model load and special decomp
  - Per-vertex
    - Ray casting
  - Per Bounce (x3)
    - Per-vertex
    - Illumination calc per ray
  - Per-Vertex
    - Final Summation
  - Dump Data to disk

CPU usage is all on one core

Most time spent in Per-Vert Compute Operations

Distributed Directional Occlusion and Illumination

Approach

- Calculate directional occlusion and illumination for each vertex and each bounce independently then re-combine final data

Problem

- Calculation of N+1 bounce data requires access to All vertex data from Bounce N

Solution

- Create a multi pass approach, for bounce N each vert is independent of all other verts in bounce N
- Write all data from bounce N before starting bounce N+1,
- Run bounce N+1 passing all data from Pass N
- Read all final data and write final game asset

Trading Problems

- Distributed compute trades a CPU bound problem for a I/O bound problem
- Bounce data ~750 mb per bounce of data for autovista
- This is ~1.5 GB of write and read per bounce
- At a conservative 10 MB\s of throughput ~= 2.5 min per bounce
- Approx. 7.5 minutes of I/O overhead for a 3 bounce SH burn
- Bounce data non autovista ~80MB per bounce
- ~160 MB total I/O per bounce ~= 16 sec per bounce
- Less than 1 minute I/O over head for a 3 bounce SH burn

Working with Incredibuild's XGE

Virtualized file system

- XGE's virtualized file system makes data management trivial
- All file i/o is seamlessly routed to the requesting nodes file system via TCP
- All worker nodes see files on the requesting node as if they are local file with the same name and path as the requesting node
- i.e. any reference to c:\temp\mydata.xml is the file located on the requesting node's c:\temp no matter what node is accessing it.
- Simple file based data communication

Simple execution and synchronization via xgesubmit and xgewait

Limitations

- Allowing more than 20 nodes per job is not advised by Xorax
- We are currently using 40 nodes with good results
- Disk and network bandwidth of requesting machines
  - 1gbe caps at ~125 MB/s, but saturating your network will not make many friends
  - Local disc will usually be under 90MB/s unless using 2nd gen SSD's or large raid arrays
- No advanced message passing system

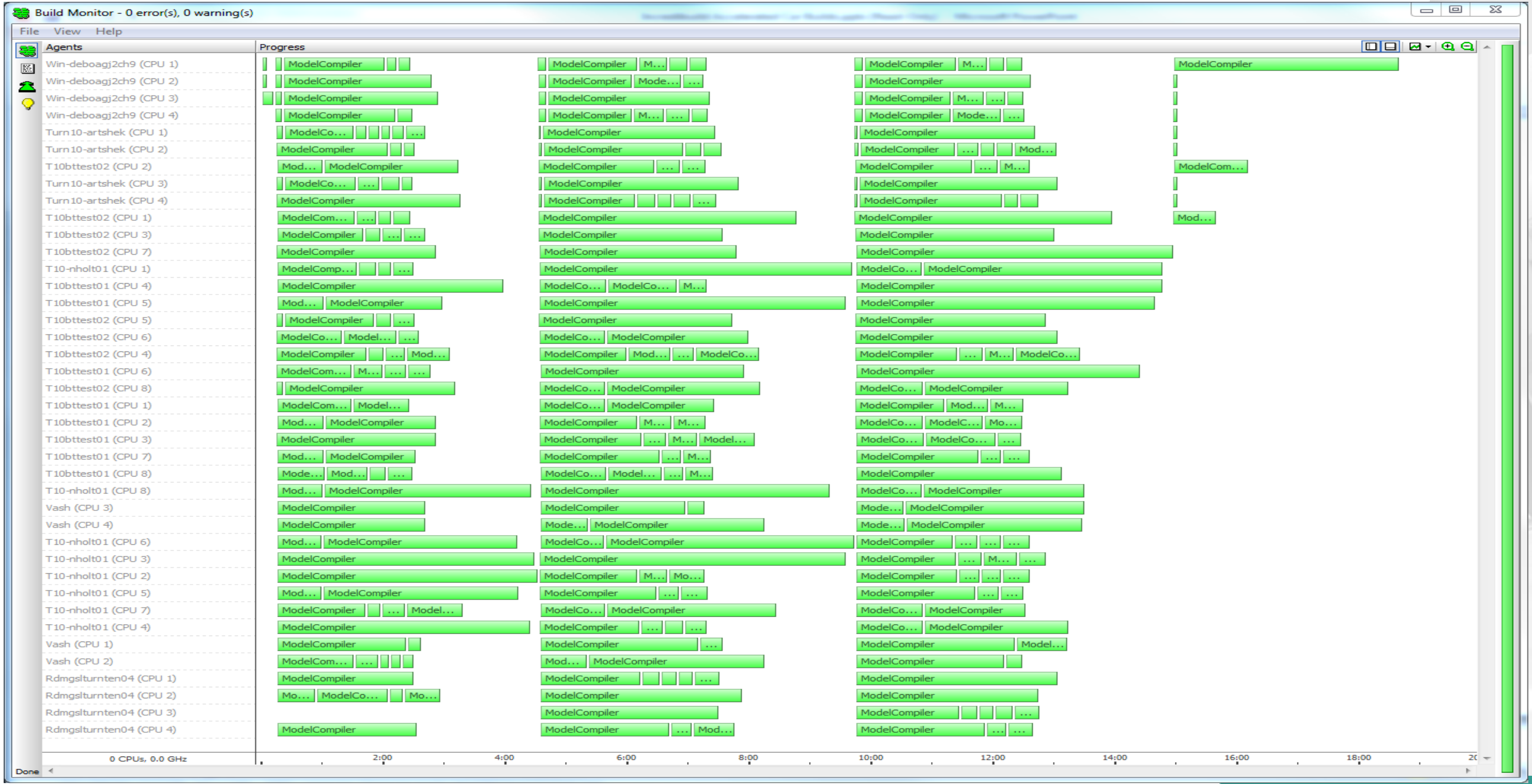
Gotcha's

– Writing "error" to the command's output stream will trigger Incredibuild to report errors even if these are just trace data

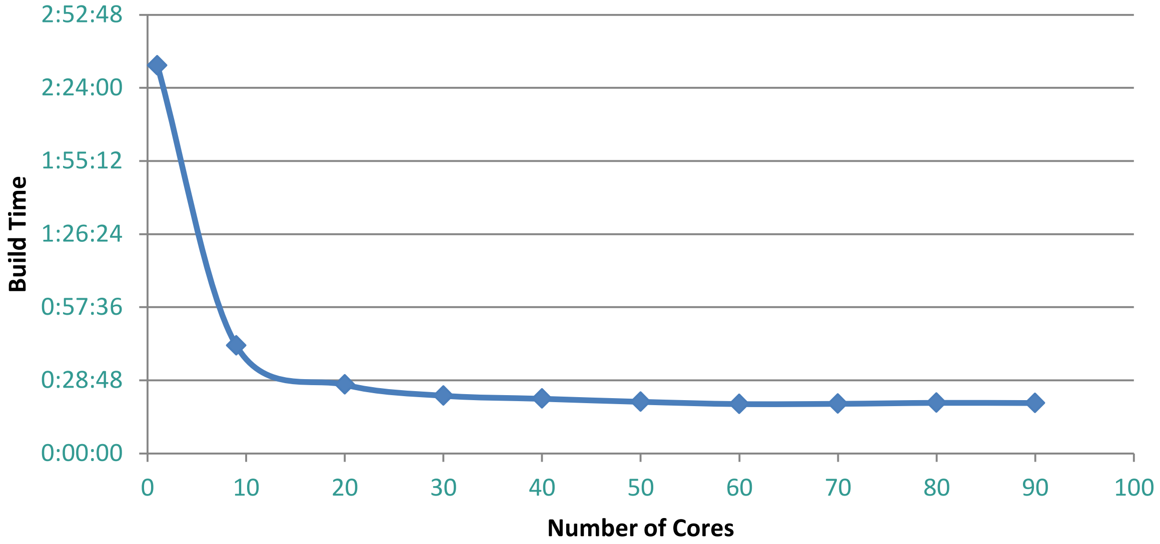
Change tools to be XGE friendly

- Design your systems to run calculations for a subset of data
- For algorithms with data dependencies use a multi pass approach where all independent calculations are run in a single pass and state is serialized to disc at the end of the pass
- Serialize data to uniquely named files
- Parameterize the pass stage, file names, and data subsets

Setup Bounce 1 Bounce 2 Bounce 3 Consolidate



Actual Performance



- AutoVista (1M poly) Cars 1.2-1.5 hours from 8-12 hours
- Non AutoVista(400k poly) Cars <25 minutes from 2-4 hours