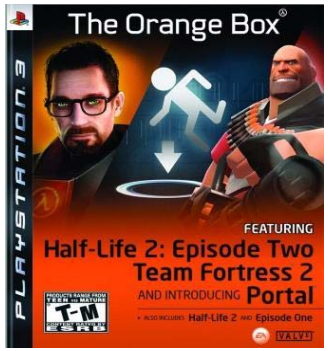
How To Go From PC to Cross Platform Development Without Killing Your Studio

Elan Ruskin
Valve

CMP
United Business Media

We Are a PC Shop That Recently Added Console.



- Some of this talk may seem *elementary* to console-exclusive developers …
- … but each one of these issues has burned an actual project.
- Experienced console devs will still find useful info here.

This Is a High-Level Talk

This Talk is Based On:

- Our work at Valve
- My work elsewhere
- Interviews with others throughout industry

What Landmines Await A PC Developer Going To Console?

Consoles are like PCs...

A PC

\+

Closed Platform

\+

Manufacturer QA

\+

Limited Memory

=

Common Problems of Crossplatform Development

- Developer Efficiency
- Certification Failure
- User Experience
- Programming Issues

Targeting Console is Similar to Targeting a Minspec PC

- Valve always tiers our PC experience

High-end (Shader Model 3)

Midrange (Shader Model 2)

Low-end (DirectX 8)

Now We Know Where The Mines Are…

Common Problems of Crossplatform Development

- Developer Efficiency
  - Staff allocation
  - Trouble Iterating
- Certification Failure
- User Experience
- Programming Issues

The Core Team

- **The Console Person**
- Your most experienced programmer.
- Understands the entire codebase.
- Senior enough to affect schedule.
- Gets the game running for the first time.
- Becomes an oracle by project end.

The Core Team

- **The TCR** (Technical **C**ertification **R**equirements) **Expert**
- Producer, Programmer, or QA.
- Learns every item on Microsoft/Sony's certification checklist.
- Builds test cases.
- TCR is not a job for one programmer.
  - Does need one person in charge.

The Core Team

- **The Devkit Guy**
- Gets people up and running.
- Sets up artists to look at their levels,
- Gets programmers set up with their debugger.
- Isn't a full time job, but can be a major distraction.
- Doesn't need to be a lead.

Common problems of cross platform development

- Developer Efficiency
  - Staff allocation
  - Trouble Iterating
- Certification Failure
- User Experience
- Programming Issues

Problem: Iteration is slow.

- Iterating on PC:

- Iterating on Devkit:

Keep your PC version working.

- Debug and load times always faster on PC than console.
- Runtime iteration much easier on PC
  - Edit & continue
  - Reloading assets
- Compiling slower for console target

Simulate console content features on PC

- PC workflow is more comfortable for artists
- PCs are cheaper than devkits
- Encourages experimentation

Cross platform assets

- Consoles have their own formats.
  - Do you byte swap on load?
- Consoles prefer assets compiled into big files.
  - PCs have disk caches.

The Catch-22:

Load asset files individually:

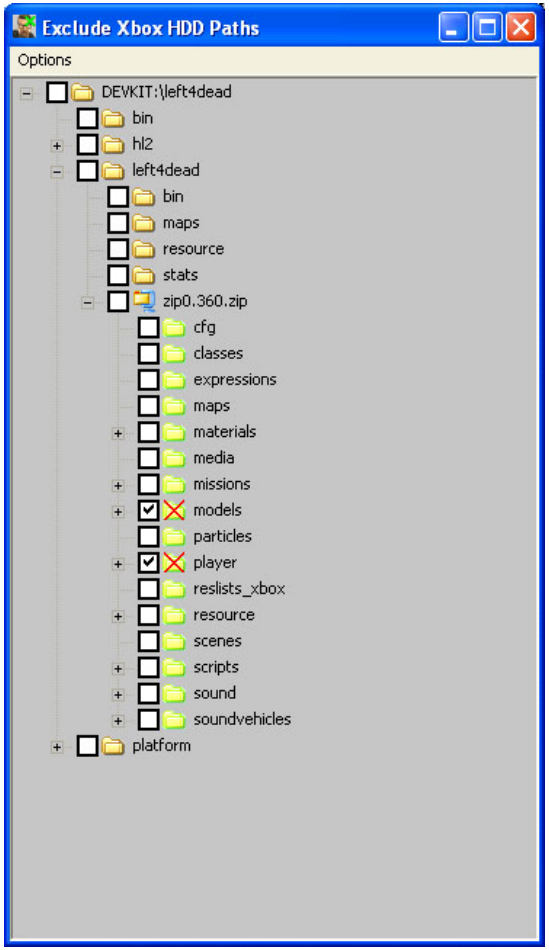
- launch times longer
- Changing data faster

Compile paks:

- code changes faster
- data changes slower

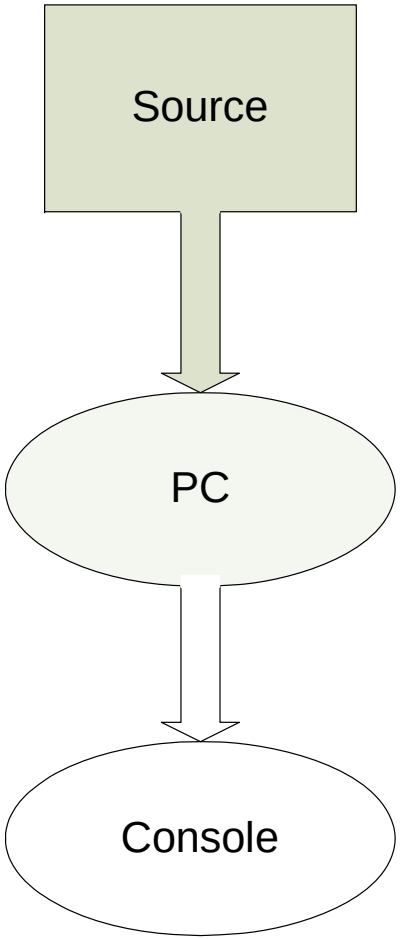
Our hybrid solution:

- Compile asset tree into paks nightly.
- Artists specify individual assets to override locally
- Best of both worlds



Branch In Pipeline, Not In Source

- You will need to recompile every asset
- Try not to diverge assets
- Make tools deal with platform differences, instead of artists.
- Keep the source art for everything.



Common problems of cross platform development

- Developer Efficiency
- Certification Failure
  - Out Of Memory
  - Starting Too Late
  - Multiplayer
- User Experience
- Programming Issues

Technical Certification Requirements / Technical Requirement Checklist / CERT

- The process by which console manufacturer ensures quality.
- A specific list of requirements that your game must meet.
- Pass, or don't ship.

Most Common Problems:

- Stability
- UI – very specific requirements
- Savegames
  - Need to be completable with no save media
- Online / LIVE

Problem:
Game Runs Out Of Memory.

PC:

Console:

Memory

- Memory is critically strict.
- The #1 reason levels get changed.
  - The later you wait, the more drastic the cuts.
- You will always wish you had worried about memory sooner.
- Account for *everything.*

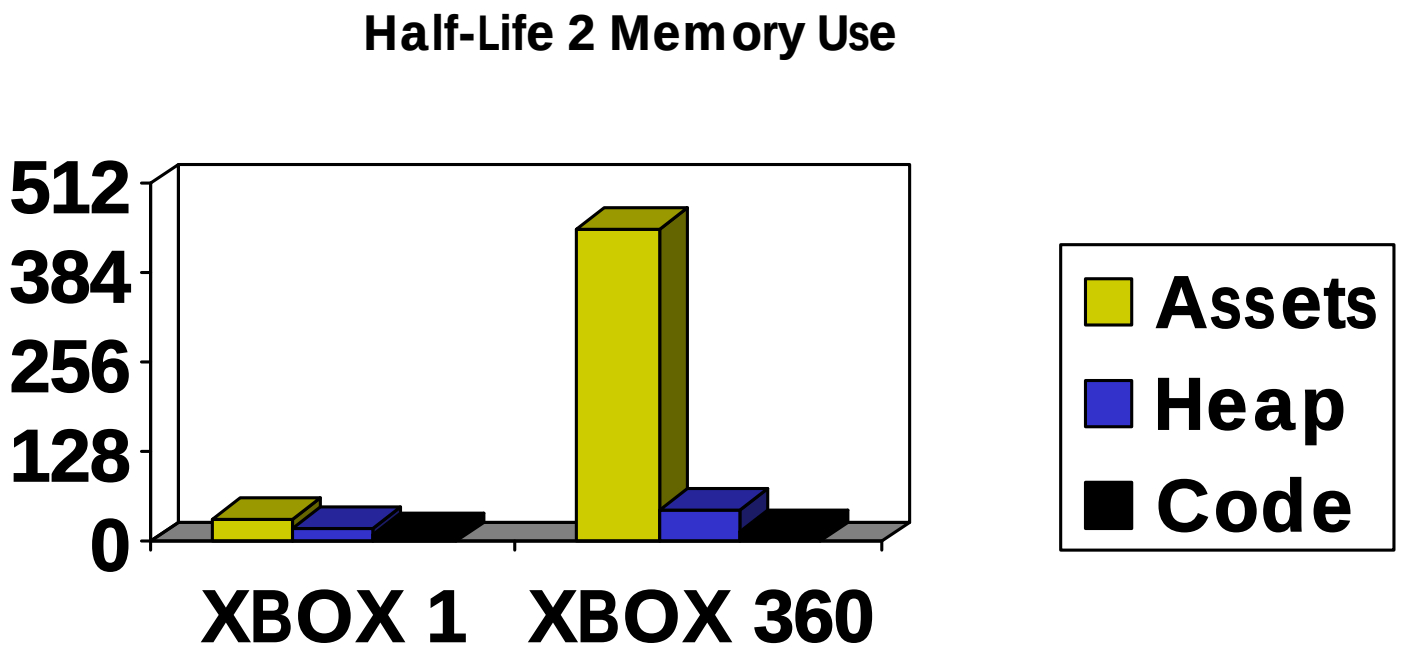
Dynamic Allocation Is Bad.

- If you don't know how much memory you're going to need, you don't know if you're going to run out.
- PC games tend to allocate memory ad-hoc.
- Keep track of where it goes.

Where Does Memory Go?

- Executable code
  - Does not change at runtime.
- Assets
  - Textures, level geometry, models, animations, sound, sprites, ...
  - Loaded into memory from disk.
- Heap
  - Data generated at runtime by code.
  - Anything that is not assets.

Assets Have Grown Faster Than Heap.



Squeezing assets, Step 1: Account.

- Track every asset allocated.
- Emit spreadsheets for each level.
- Automate this process.
- Do it every night.
- Will highlight all serious problems...
- ... and make new ones obvious.

Squeezing assets, step 2: Compress.

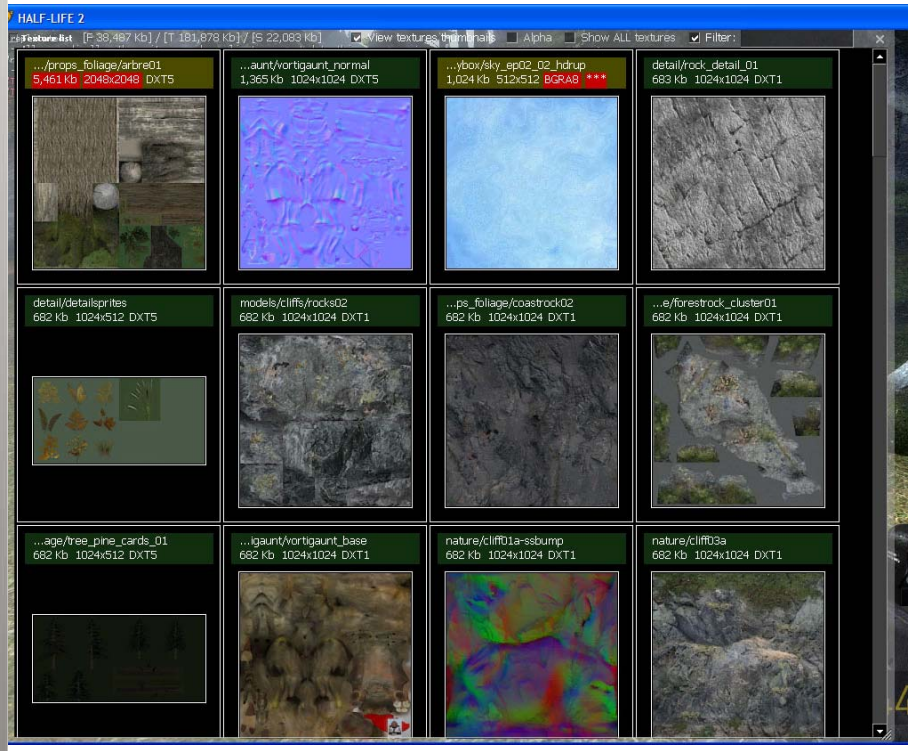
- Use platform-specific formats.
  - XMA, AAC have good ratios
- Leverage your shaders' and SPU's power
  - Compress normal maps, grayscale textures, animations...
- May need to split up textures

Squeezing assets, step 3: Reduce.

- Budget your textures / models / meshes carefully.
- It's easy to just downsample all your textures...
- But you can get much better results with careful targeting.

Squeezing textures

- 20% of the textures are 80% of the problem.
- Source has tools to show us *which* 20%:



Squeezing textures

- Halving one 1024x1024 texture saves as much memory as eliminating 32 128x128's.
- Focus on what's actually visible, so you can reduce where no one will notice.

Squeezing assets, step 4: Maintain.

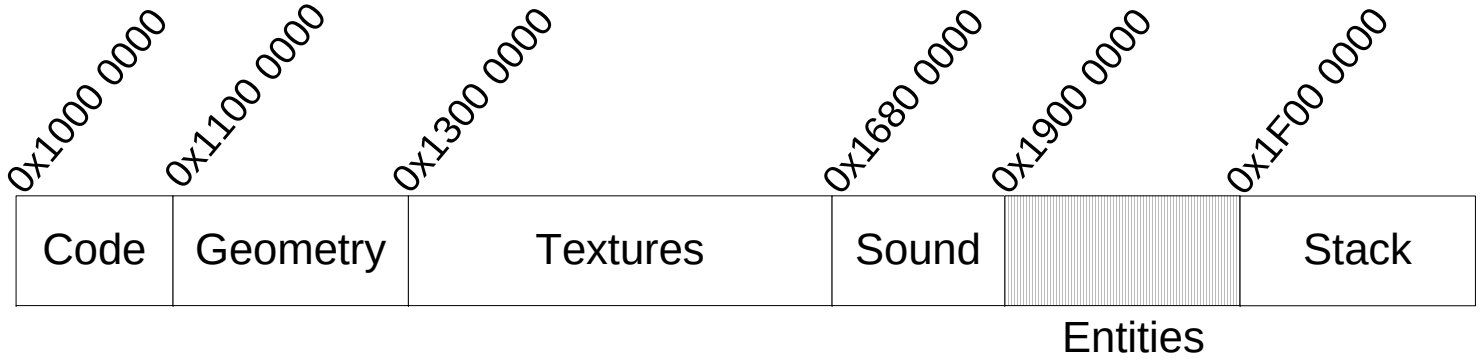
- Staying in memory is everyone's job.
- Know exactly when and where regressions occur.
- Find exactly which change to blame.

Squeezing assets, step 5: Panic.

- If all else fails... split levels.
- Remove characters.
- Decimate textures.
- Downsample animation.
- Dealing with memory *sooner* will spare you all these painful measures.

In An Ideal World

- Memory would be allocated at load time…



Managing Heap Growth

- Many games load assets ad hoc
  - Textures, models, animations, sound
- Code generates data too
  - Spawning entities, particle systems, AI state…
- Crashes most likely in level loads

malloc() Considered Harmful

(**new**/**delete** too)

- Gameplay systems most likely source of leaks.
- Good container classes provide easier management, less leakage.
- Only if you write your own allocator!

WHYTO: Make A Custom Memory Allocator

- Replace malloc(), calloc(), **new**, etc. with your own code.
- Better than STUDIONEW, STUDIODELETE macros:
- No big search and replace.
- You can't fix all the new/deletes in 3d party libraries...
- ... so link them to your own allocator.

HOWTO: Make A Custom Memory Allocator

- Override *every* function in the CRT .obj that contains malloc:
  - malloc, free, calloc, realloc…
- Put your implementation in its own .cpp
- Link this .cpp to every project in your game.
- Only works if you override the *whole* module...
- … so you need to re-do this if you change compilers or CRTs.

pwn your memory

- If you own every allocation, you can track every allocation.
  - Even those coming from the STL.
- Write global fixed-size pool allocators.
- Limit fragmentation.
- Look up the **Translation Lookaside Buffer**.

Track Memory Based On Exactly Who Allocates It

- Budget asset allocation by type
  - Texture, geometry, sound…
- Budget code allocation by purpose
  - AI::Navmesh, Particles, Rendertarget…
  - *Not* std::vector<int>

Track Memory Based On Exactly Who Allocates It

```
Thingy *WasteMemory( Thingy* input,
  std::vector<Thingy> &list )
{
  MEM_ALLOC_CREDIT(IMPORTANT_SYSTEM);
  globalSystemList.AddToTail(input);
  list.append(*input);
  Thingy *output = new Thingy;
  output = DeepCopy(input);
  return output;
}
```

Be Careful With Containers

- Container classes mean more:
  - Dynamic allocation
  - Range checking
  - Copying things around
- Use std::vector::reserve

We Do This Work For PC Too

- Disk swapping bad!
- Budget tracking means reliable information everywhere.
- Retrofitting later means touching a thousand different places.

Common problems of cross platform development

- Developer Efficiency
- Certification Failure
  - Out Of Memory
  - Starting Too Late
  - Multiplayer
- User Experience
- Programming Issues

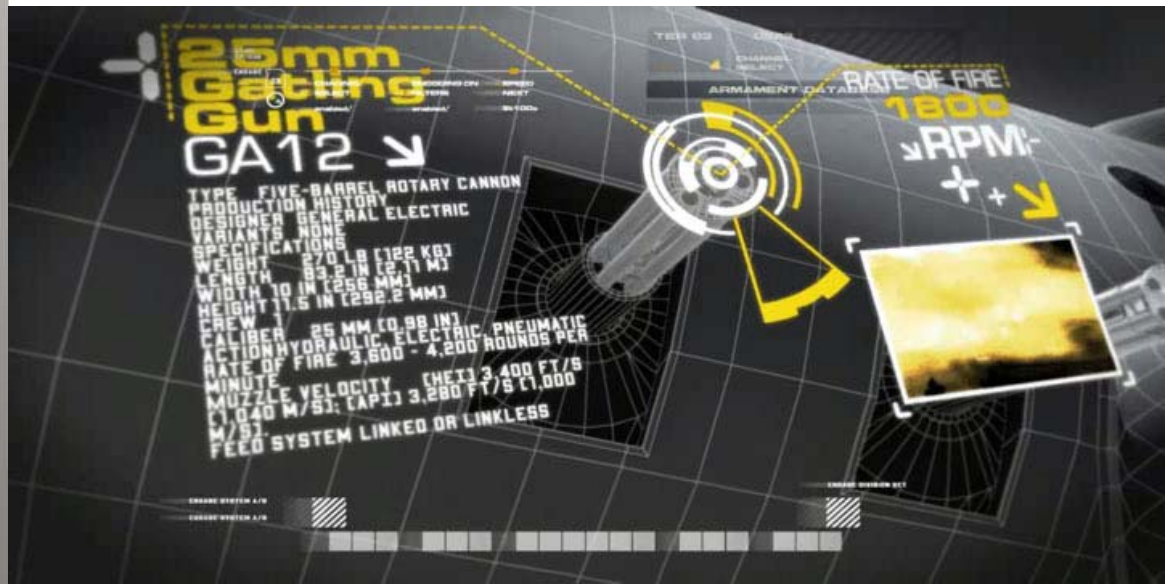
Other Interesting TCRs

- Load times no more than *x* seconds.
- Letting people play their MP3 collection in your game.
- Minimum refresh interval... even while loading.
- Compiled with recent SDK.

Solve It In Design

- Make cert requirements part of your architecture.
- Think about Achievements / Skill Points in your design.
  - (you can get them on PC with SteamWorks)

engaging player during load in *Call of Duty 4*



Savegames

- Use small individual files, not one large package
  - Fits on memory cards.
- Deal with losing memory card during save.
- Do you really need save-anywhere?
  - If you rely on quicksave,
  - preallocate a RAM disk big enough.

UI

- Consider title-safe and widescreen
  - Be readable in 4:3 SD and 16:9 720p.
  - Be readable in 4:3 SD… *in German.*
- TCR has specific requirements for UI layout & flow:
  - Need a way to pop dialogs on top
  - Manufacturer-approved graphics, names
- This usability work makes your PC game better.

Common problems of crossplatform development

- Developer Efficiency
- Certification Failure
  - Out Of Memory
  - Starting Too Late
  - Multiplayer
- User Experience
- Programming Issues

Multiplayer / LIVE

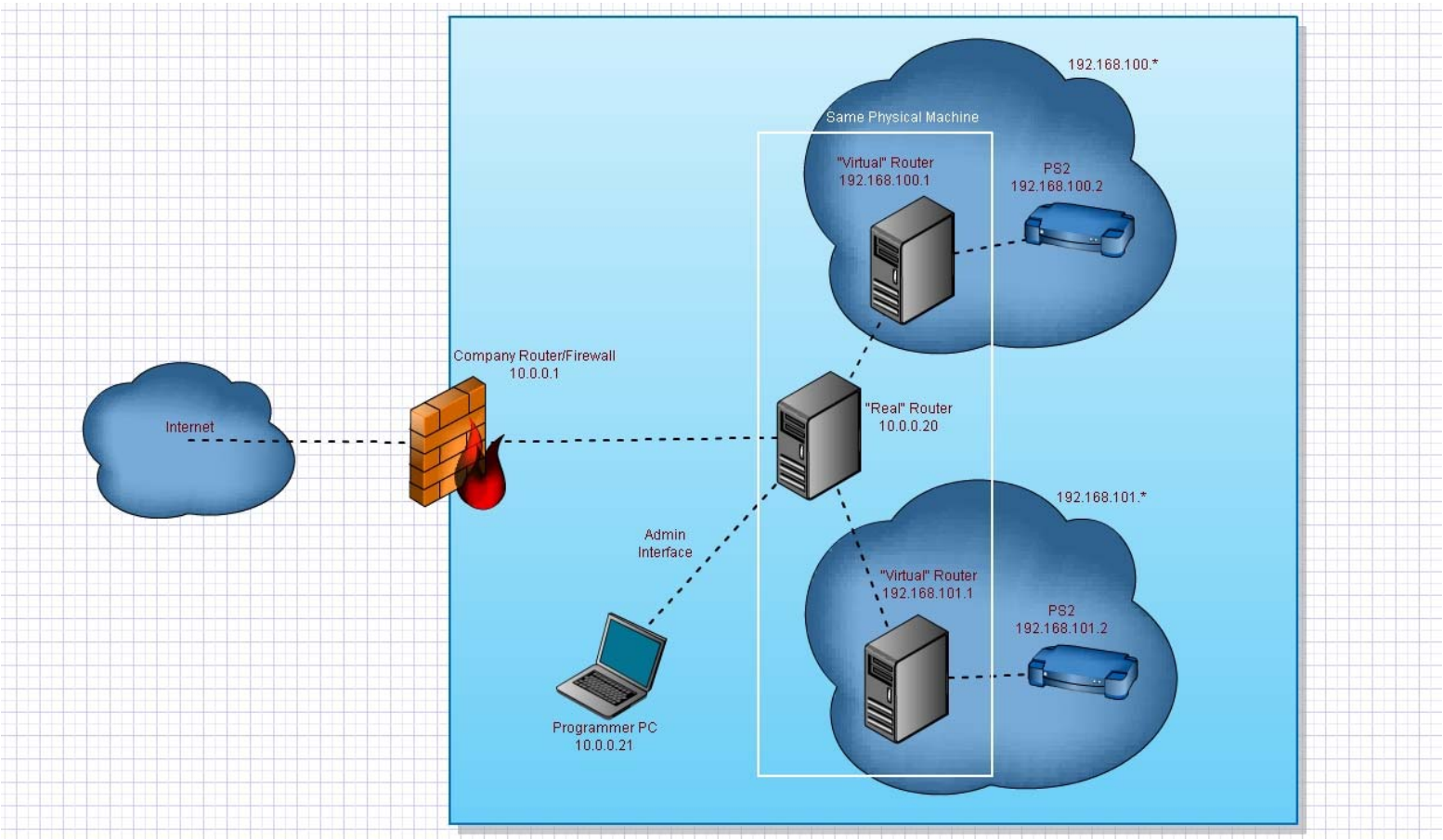
- Was the majority of our cert issues.
- Start working on this from day #1.
  - Do your preliminary work in sample apps.
  - Do not let it be blocked by engine development.
- Rich presence may require architecture changes.
- Enlist Microsoft/Sony's help.
  - They have lots of good tools for you.

Cort is playing TF: Badlands!
Ahead 3-2 in CTF

Multiplayer Testing

- Your office LAN is not the Internet.
- Debugging without encryption, voice hides problems.
- Some problems arise only with high load.
  - Do a beta if you can.
- Latency, bandwidth, ping, problems as always...

Multiplayer Testing: Simulate Your Own Network Backbone.



Courtesy Ben Stragnell

Common problems of crossplatform development

- Developer Efficiency
- Certification Failure
- User Experience
  - Load Times
  - Use of multicore
  - Controls
- Programming Issues

Problem: Game Takes Too Long To Load.

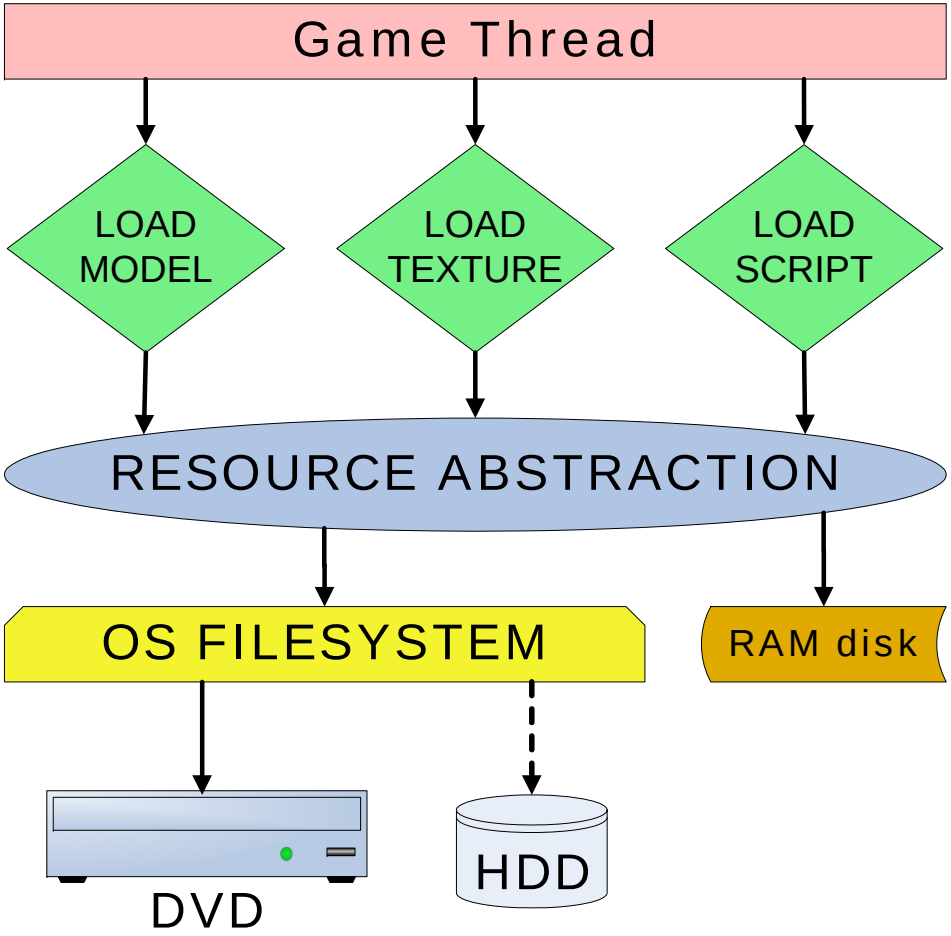
Optical Load Times

PROBLEM	SOLUTION
Seeks	Contiguous files, careful layout
Misaligned reads	Sector alignment
Buffered access	Unbuffered DMA I/O
Synchronous stalls	Asynchronous loading
Small files loaded on demand	Large, single files

Things That Make A DVD Load Faster

- .ZIP files
- Compression
  - (trade CPU for I/O bandwidth)
- Asynchronous loads in a separate thread

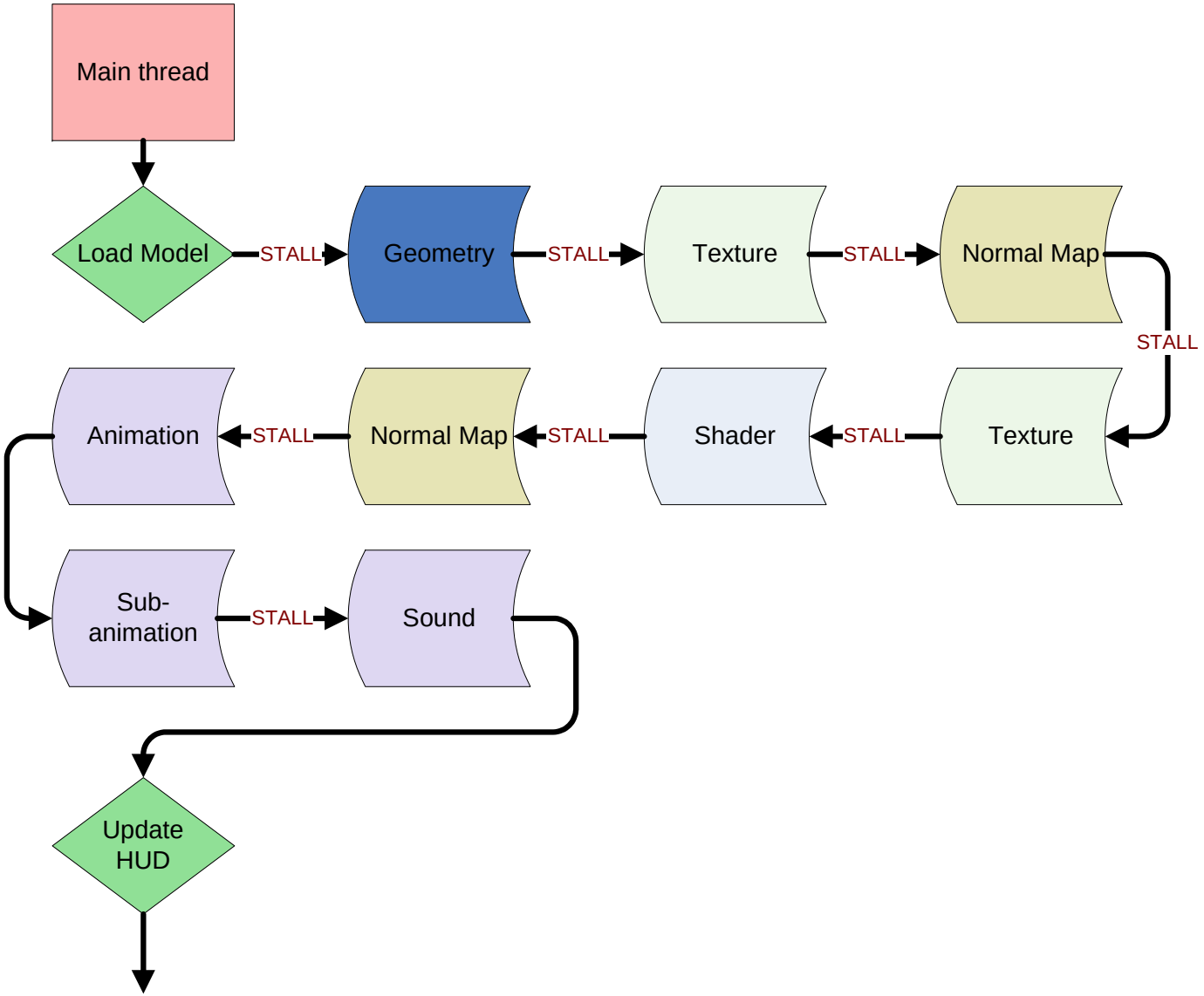
How We Refit Our Game Without Rewriting Everything



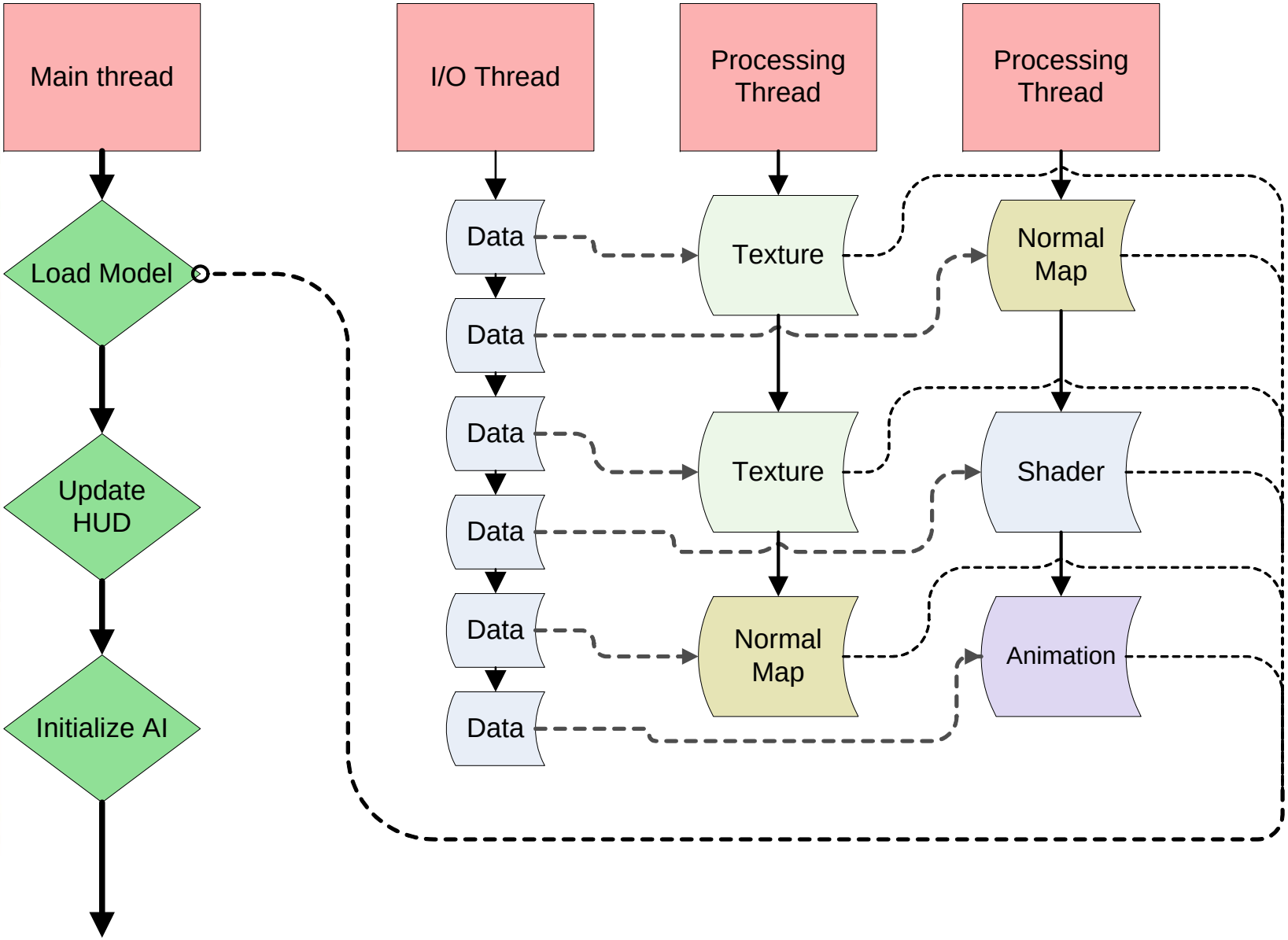
Three Categories Of Data

- Big
  - Textures, models, BSP, sprites, SFX
- Small
  - Config files, scripts, <4k odds and ends
- Really Big
  - Dialog, long animations
  - Stuff you don't need right away

Synchronous (naïve) loading



Asynchronous loading



Key features

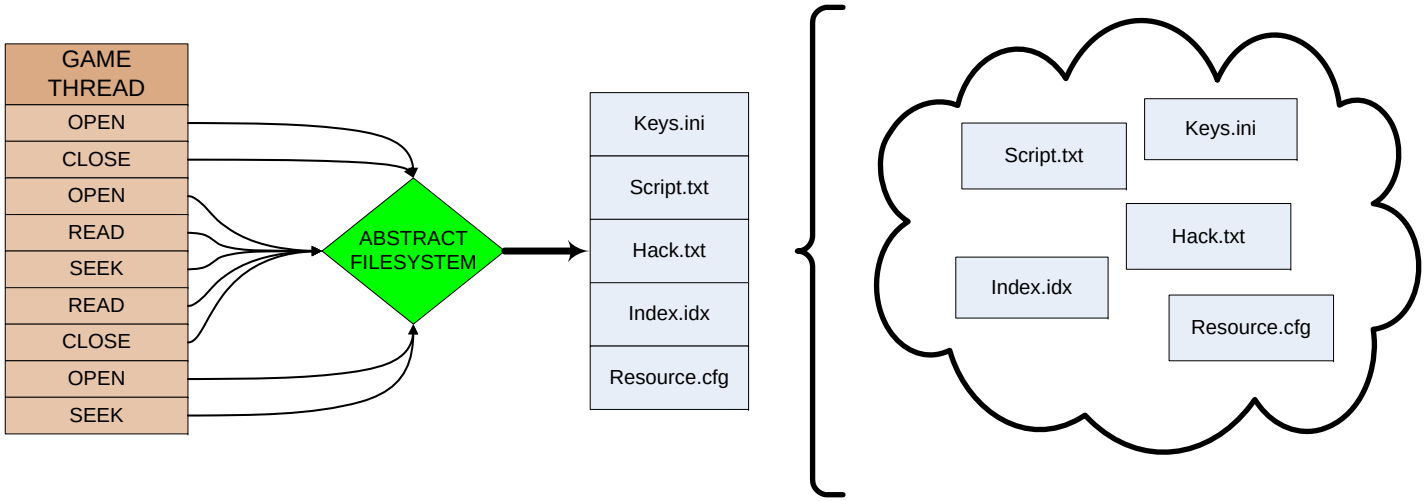
- I/O thread does unbuffered DMA transfers.
  - Keeps the disk *spinning continuously.*
- Lockless implementation
- Trade CPU/SPU for I/O bandwidth
- Return dummy values to sync loads.

Really big files: streaming

- Store the first ½ sec of each animation and audio always
- Asynchronously load the rest in the background
- Need a resource abstraction layer that can say:
  - We have the data
  - We're getting the data
  - We will never get the data

Small files

- Precompile all small ad-hoc files into one large blob
- Read it in one operation with level
- Create a fake file system
- Don't have to change game code!



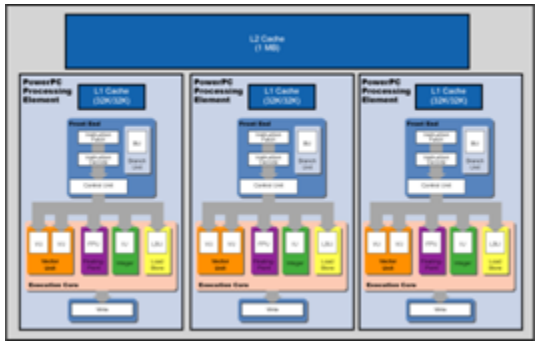
Know In Advance Each Level's Resource Needs

- If you're going to build a pak, you need to know what goes in it.
- Every single asset.
- Analyze loading dependencies.
- Crash when loading out-of-pak.

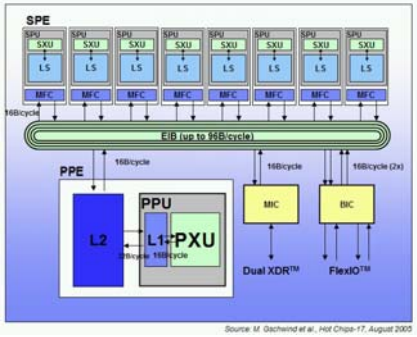
Common problems of cross platform development

- Developer Efficiency
- Certification Failure
- User Experience
  - Load Times
  - Use of multicore
  - Controls
- Programming Issues

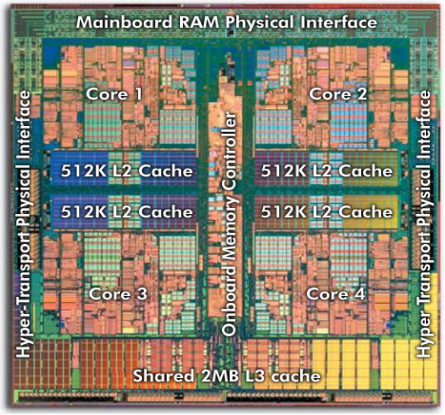
Going Multithreaded: It's not *next*-gen any more.



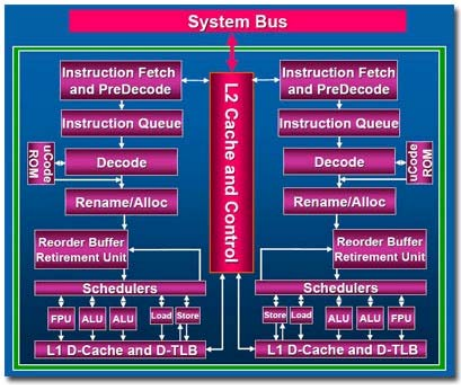
Xenon



Cell



AMD Barcelona



Intel Core 2

All Major Platforms Are Multicore.

- **360**: 3 symmetric PowerPC cores, 6 threads
- **PS3**: 1 PowerPC, 2 threads; 7 vector processors
- **Intel/AMD**: Quadcore now, 8-core tomorrow
- This is not “next gen”, it is “today.”

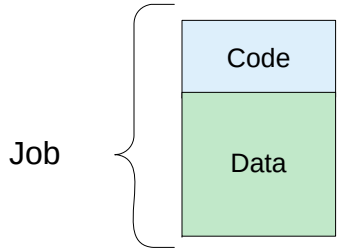
Our technique: Discussed here before

"*Dragged Kicking and Screaming: Source Multicore*"

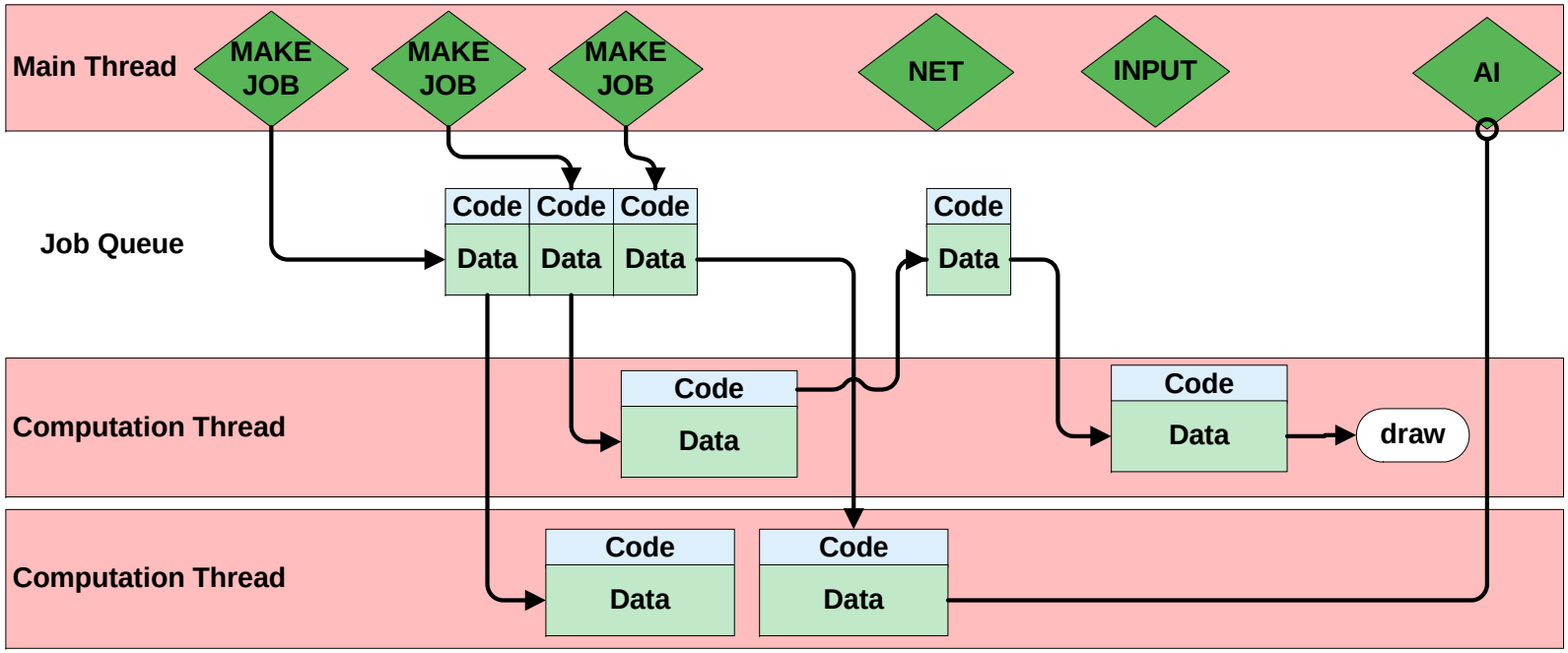
Tom Leonard (Valve), GDC 2007

http://www.valvesoftware.com/publications.html

Job queues: a summary



- A job is code and local data
- Put into a queue; other threads consume from queue



Worked Better Than We Expected!

- On the 360:
  - 50% performance improvement just from queuing graphics functions.
  - 4x increase in framerate with full implementation.
- On the PS3:
  - Game wouldn't run otherwise!
- Our game already had a client/server split.

De-Globalize Your Data

- Pack jobs' data up so they work locally.
  - Put global data into a closure.
- Avoid chasing pointers all over memory.
- Especially critical on PS3.
  - SPUs have only 256kb of memory.
  - Random memory access is huge stall.

PS3 Requires More Aggressive Threading

- All the power of the PS3 is in its Cells.
- The PPU will be always saturated.
- General C++ code does not run well on SPUs.
- Code memory is tight.

Some Things To Worry About

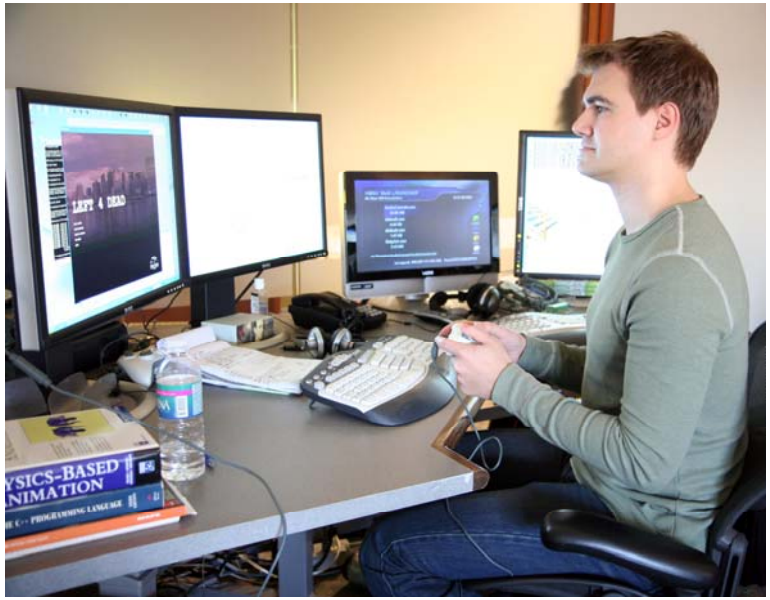
- Callbacks
- Synchronizing simulation clocks
- Mutexes (can make you slower than singlethreaded)
- Hardware threads useful only in certain cases – measure it.

Common problems of cross platform development

- Developer Efficiency
- Certification Failure
- User Experience
  - Load Times
  - Use of multicore
  - Controls
- Programming Issues

Problem: controls don't feel right.

- Have PC devs test with 360/PS3 controllers.
  - Yes, you can connect them to a PC.
- Makes everyone a usability tester all the time.
- PS3, 360 have different thumbstick calibrations.



Common problems of cross platform development

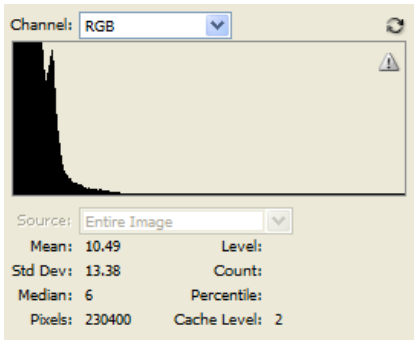
- Developer Efficiency
- Certification Failure
- User Experience
- Programming Issues
  - Graphics
  - Framerate / CPU

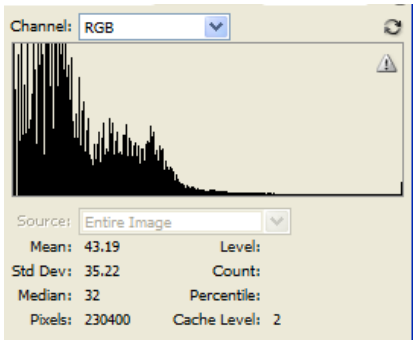
Time For *Good* Graphics!

Homestar Runner

TV pixel and color spaces differ from monitors

TVs rebalance histograms



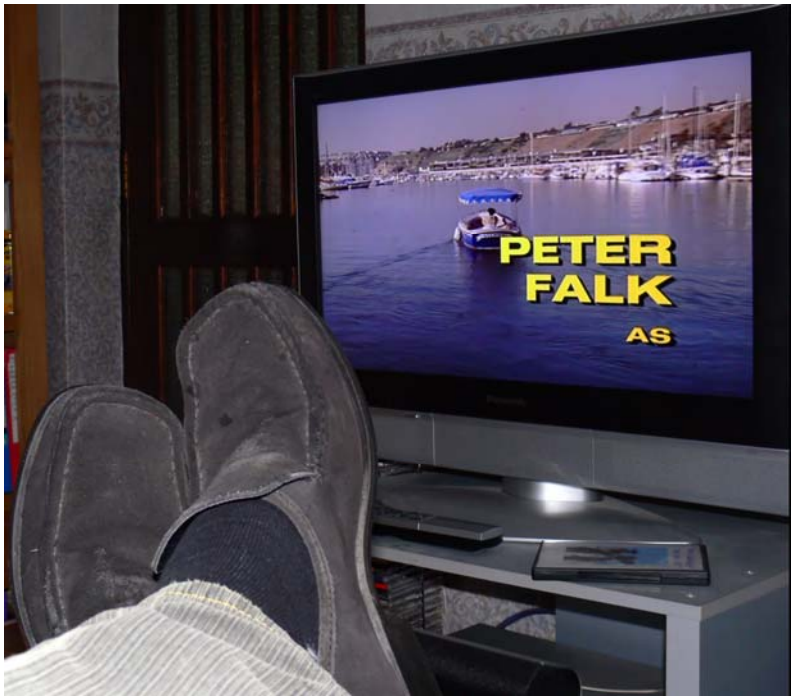


TVs vary in quality

- A common office fight:
  - Look good on a default-settings TV?
  - Or one that's been calibrated?
- TV default settings vary very widely.
- The solution:

Watch TV At The Office.

- Watch television on the displays you're developing with.
- Calibrate your TV so TV looks good.
- **Don't buy the same TV for everyone!**

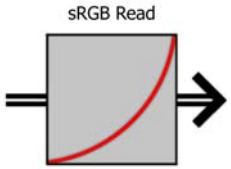


Shaders

- PC uses HLSL. Consoles use HLSL. Done.
- Shader compilers may be a bit different.
  - The few problems will be with the most complicated shader.
- GPU/CPU power balance a little different.
  - Shader conditionals perform well!
- We distribute our shader compiles.
  - Compile each shader for both platforms before checkin.
  - Compile everything offline nightly for regression testing.

sRGB

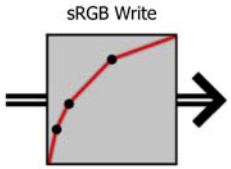
- sRGB read/write curve different on 360.
- Keep your source art
  - Compiling from another space loses precision.
- See Alex Vlachos' talk: "Post Processing In *The Orange Box*", Feb 18, 2008. http://www.valvesoftware.com/publications.html



PC Gamma to Linear

Linear to PC Gamma

360 Gamma to Linear



Linear to 360 Gamma

Other notes:

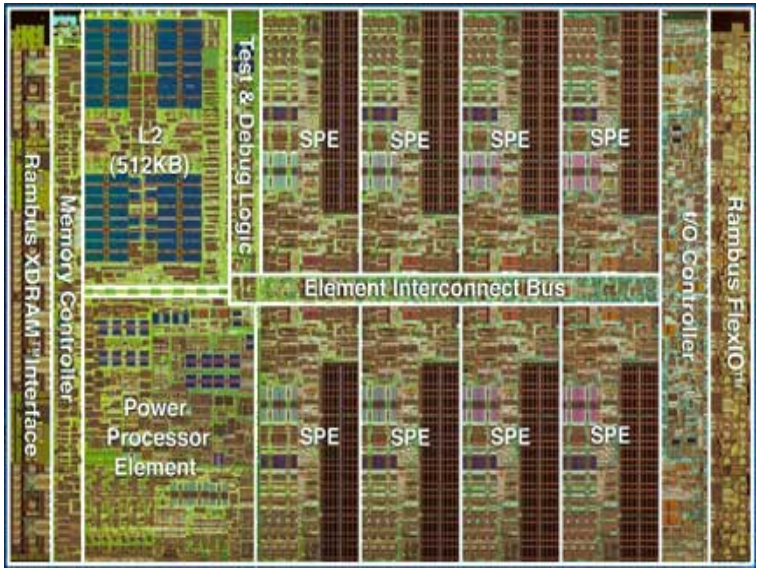
- PIX / GCM Hud excellent for very specific, actionable info.
- Look into tiled rendering on 360
  - Makes antialiasing easier, but isn't critical.
- If you're hung up on getting PC and console to match perfectly… let go.
  - No one is playing your game twice simultaneously side-by-side.
  - It just has to look good.

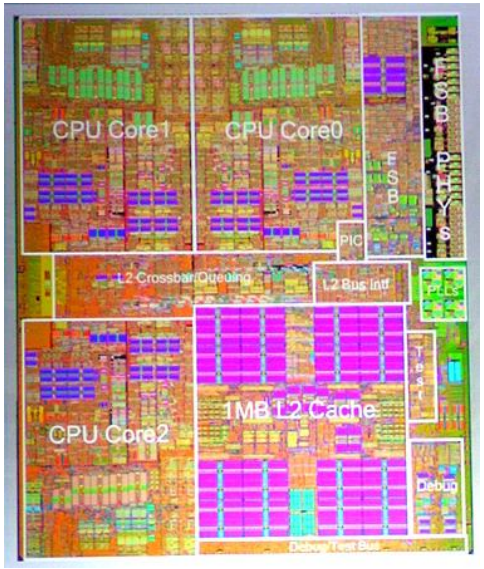
Common problems of crossplatform development

- Developer Efficiency
- Certification Failure
- User Experience
- Programming Issues
  - Graphics
  - Framerate / CPU

360, PS3 have in-order PowerPC CPUs.

- They do not rearrange instructions to eliminate dependencies.
- Sloppy code runs more slowly.
- Why? Reorder circuitry is costly, takes up space…
- …space now used for *additional entire cores!*





In-order PPCs run sloppy code more slowly than x86

- 25%-50% speed for straight cross-compiled code.
- Careful optimization gets close to parity.
- SIMD a bigger win on PPC than x86.
- Remember: on 360 you have *three of them*.

LEARN THE ASSEMBLY

- Sometimes you still have to do this.
- Use intrinsics, understand what they are doing.
- Helps debug release-build crashes.
  - Learn the calling convention, how to augur crash dumps.
- Double-check what compiler emits.

LEARN THE PIPELINE

- PPCs are high-latency, high-throughput
- Learn about all the hazards
  - Register dependency, load-hit-store, cache miss, microcode, ERAT, TLB…
  - Understand what the profiler is telling you.
  - 80% of perf from touching 20% of code.

Actually Use SIMD

- Abstract interface for all platforms.
- Push native vector class everywhere.
- Replace doubles with floats.

```
FORCEINLINE Vector Add ( const Vector & a,
                         const Vector & b )
{
#ifdef _X360
   return __vaddfp( a, b );
#elif defined(_SSE)
   return _mm_add_ps( a, b );
#else
   return Vector( a.x + b.x, a.y + b.y,
                  a.z + b.z, a.w + b.w );
#endif
}
```

#ifdef Is Not The Way To Go

- Compilers will elide code in an if() block that is always false.

```
#define IsX360() true
#define IsPC()   false

void DoStuff()
{
  if ( IsX360() )
  {
        PlatformSpecificFunction();
  }
  else if ( IsPC() )
  {
        WindowsSpecificFunction();
  }
  else
  {     // you might be on the Wii one day!
        GeneralCaseFunction(); // or throw an assert
  }
}
```

Use if() Instead of #ifdef.

- Stops “the PC guys broke the PS3 build again!”
- You may need stub functions
- Don’t assume “if” PC “else” 360. You might be on PS3 or Wii one day.

Not All Optimization Is Premature

- Don't "do a big perf pass at the end".
- Getting from 5fps to 15fps isn't optimization, it's a key feature.
- Have budgets from the start,
  - Have tools to stay inside them.

Things you need to buy: Devkits

- Development kits
  - Live debugging
  - Engine, system programmers – anyone whose bugs block someone else
- Test kits
  - Printf debugging.
  - Artists, QA, maybe gameplay programmers.
- Prepare for failure rate.

Cost:

Cost:

Other Suggestions

- For your first title: keep it simple!
- Keep people on kits.
- Work to the most constrained platform.

Measure Everything

- Measure everything yourself, as often as you can.
- Take nothing for granted.
- Verify your compiler output.

Recap

- Make cert part of your design.
- Memory will always be a struggle.
- Automate offline testing.
  - Regression is a bigger problem in cross-platform development.
- Keep the PC version working!

- Most importantly…

DO IT NOW

- The sooner you start, the better off you will be.
- Manufacturing lead times are longer on console, and you have TRC.

The Terrible Secret Of Cross-Platform Development:

All This Will Make Your PC Title Better!

- TRC is just a group of good usability rules.
- Memory efficiency helped us on every platform.
- PC games deserve shorter load times.
- Making money on the PC means hitting the low end.
- If it runs well on console, it's easy to make it run well on PC.
  - Steamworks even lets you have achievements and updates!

Special Thanks

- Iestyn Bleasdale-Shepherd
- Steve Bond
- Kerry Davis
- Vitaliy Genkin
- Brian Jacobsen
- Tom Leonard
- Jason Mitchell
- Aaron Seeler
- Jay Stelley
- Alex Vlachos
- Josh Weier
- *(and everyone at Valve)*
- Ted Jump
- Jon Parise
- Robert Pitt
- Kain Shin
- Ben Stragnell
- Cort Stratton

For questions and answers, please go to:

http://assemblyrequired.wordpress.com

See other Valve presentations at:

http://www.valvesoftware.com/publications.html