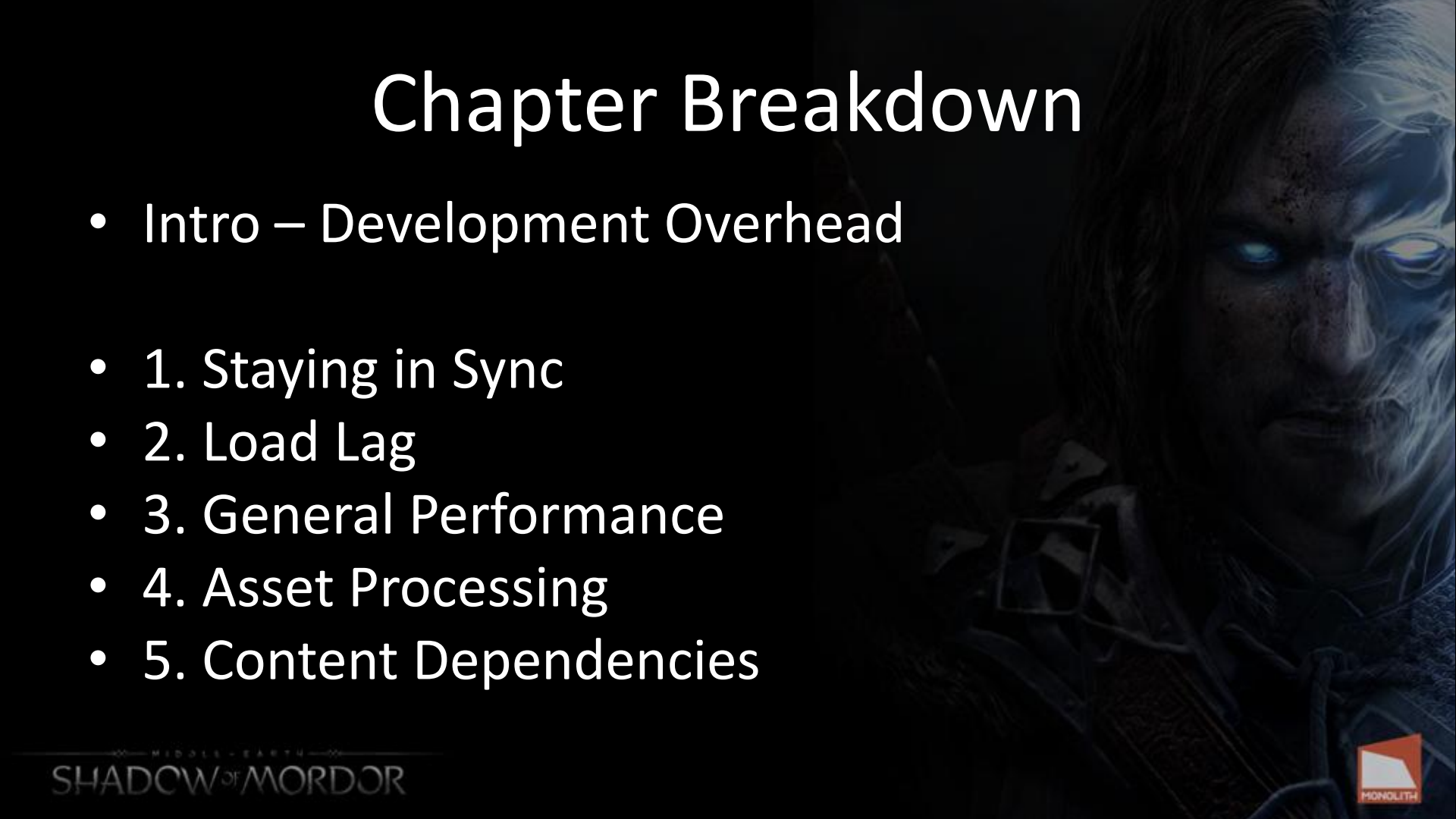
Strategies for Efficient Authoring of Content on **Middle-earth: Shadow of Mordor**

Doug Heimer
Lead Software Engineer, Monolith Productions

Chapter Breakdown

- Intro – Development Overhead

- 1. Staying in Sync
- 2. Load Lag
- 3. General Performance
- 4. Asset Processing
- 5. Content Dependencies



Development Overhead

- Overhead = damage over time to your game
- Time spent ‘not making the game’ is wasted time

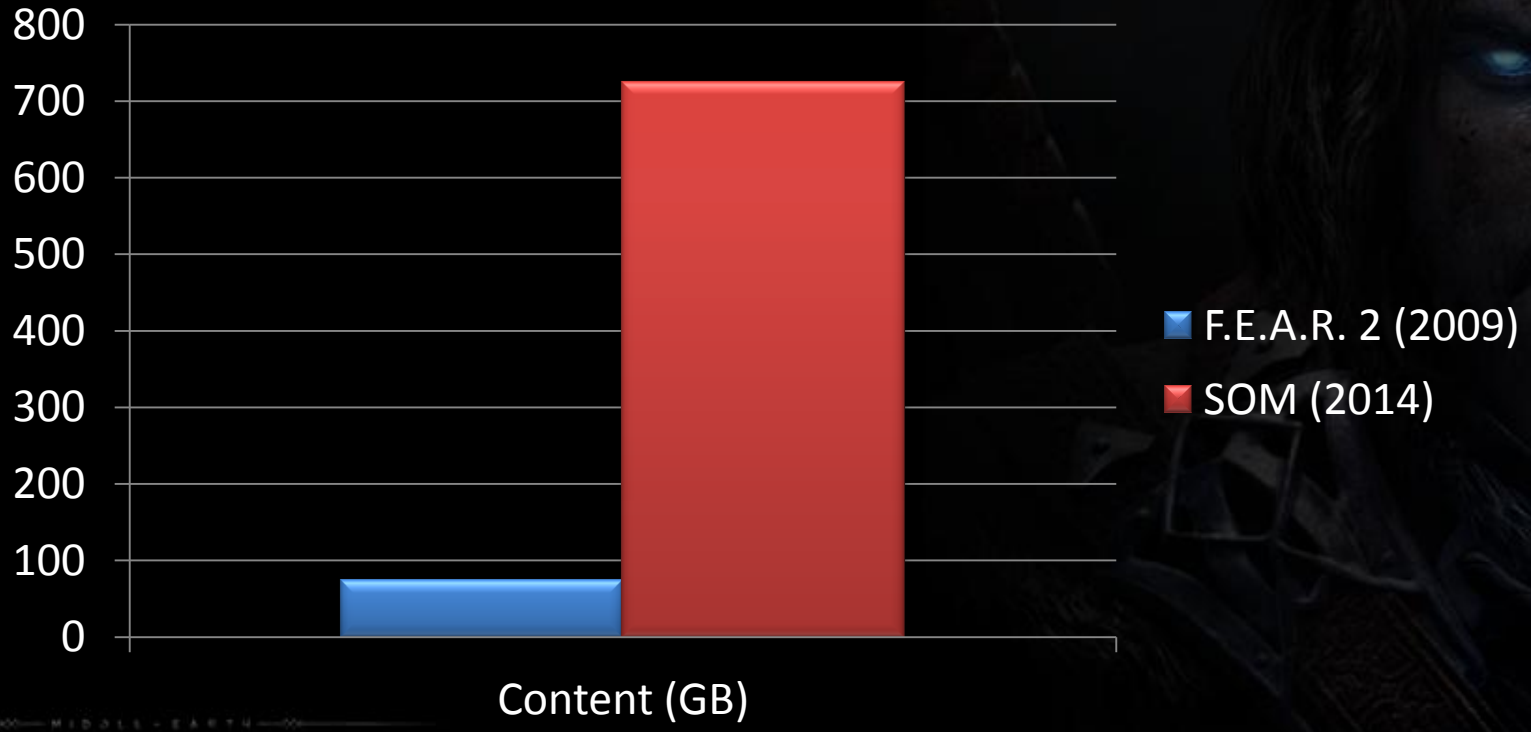
- Reduce that time!
  - Audit the development pipeline
  - Identify high ROI candidates to target

AAA Games have become huge

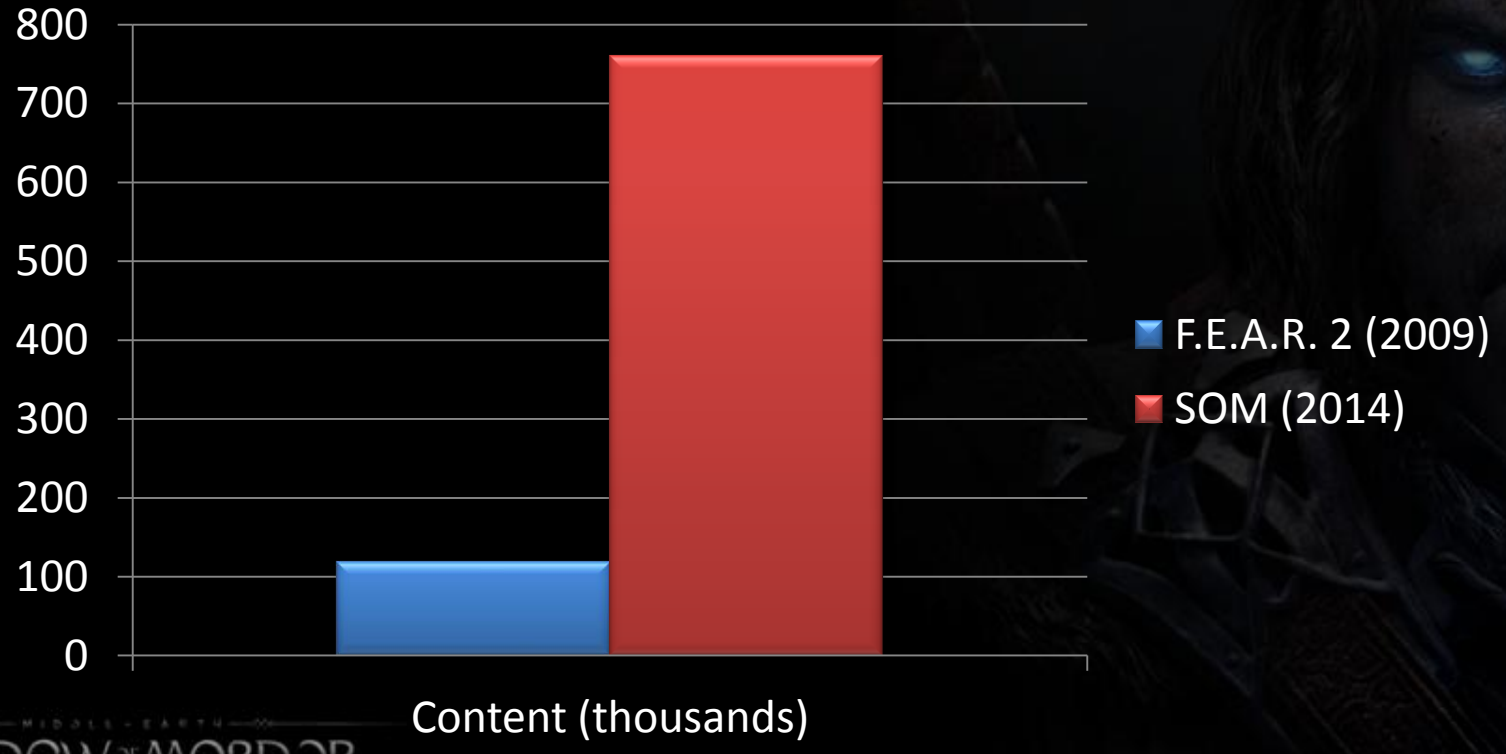
- More, bigger assets
- More asset dependencies
- More asset churn
- Longer offline asset processing times

- Greater authoring overhead for individual assets

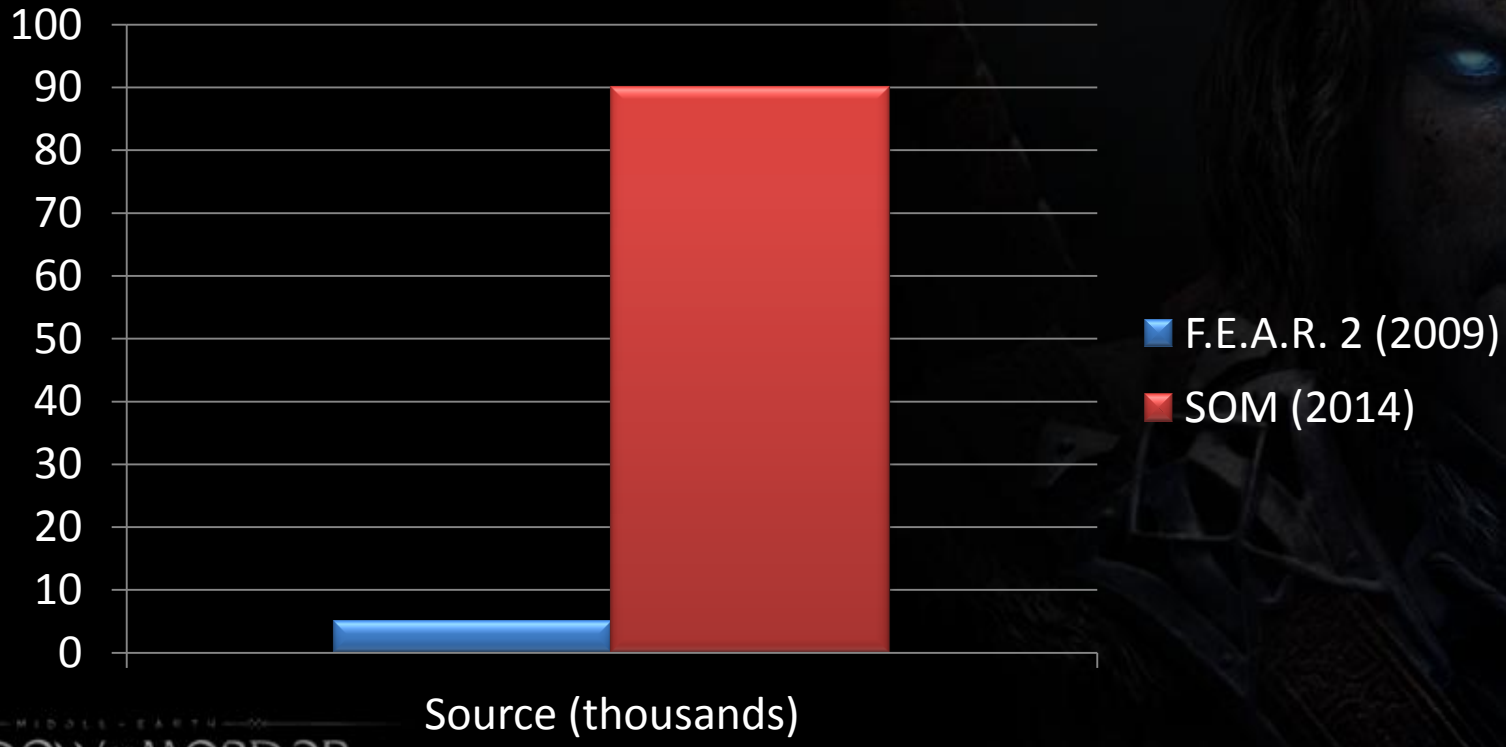
Asset Size (10x increase)



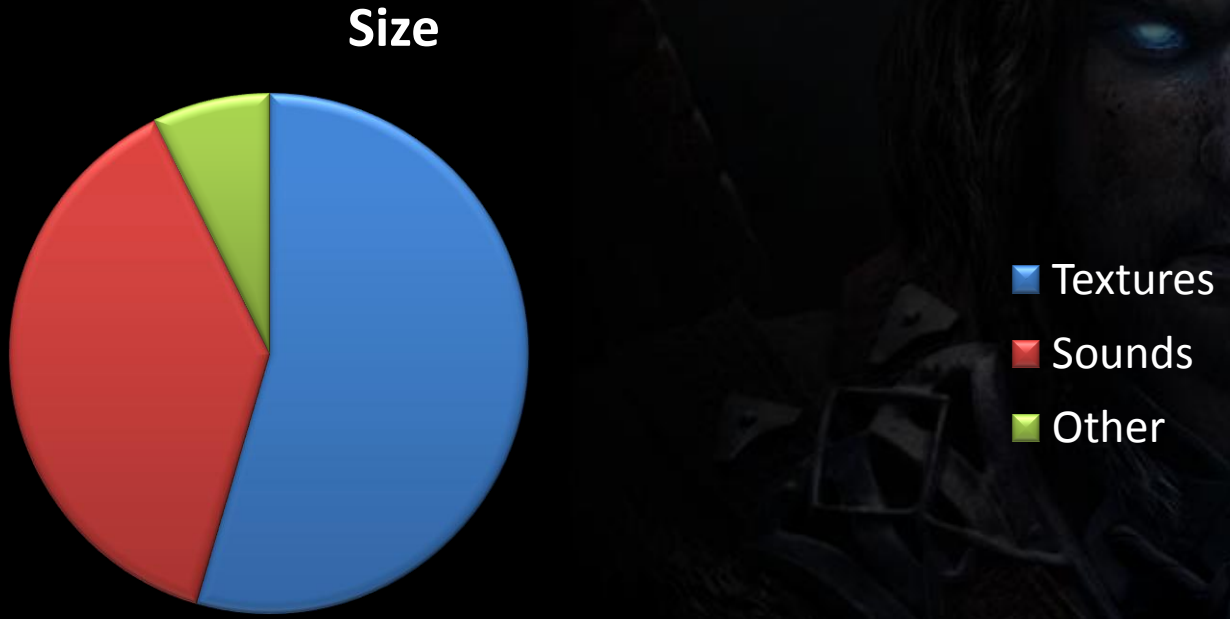
Asset Count (7x increase)



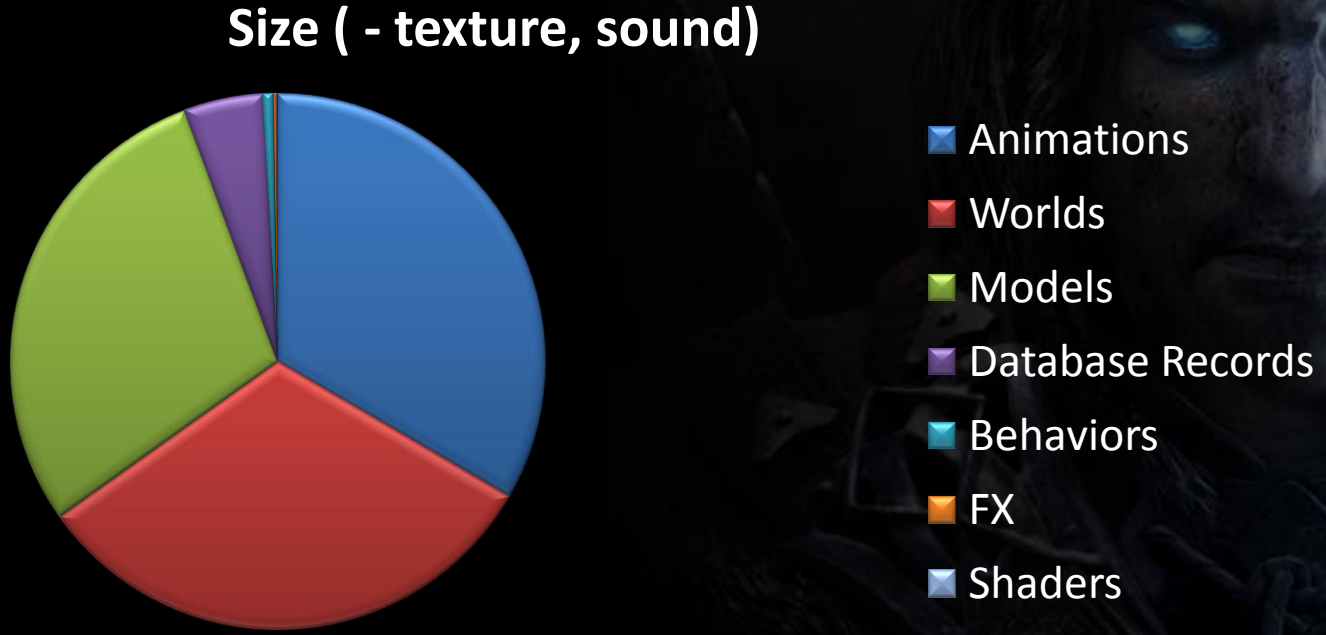
Database Records (20x increase)



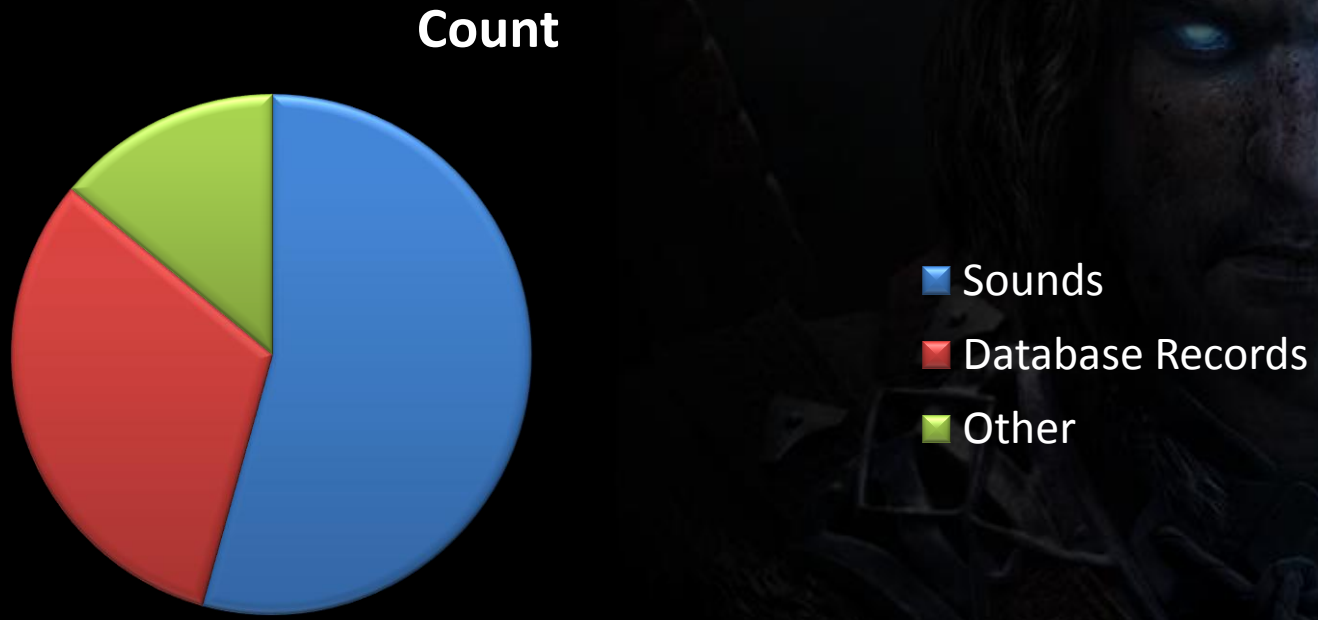
Asset Type Breakdown



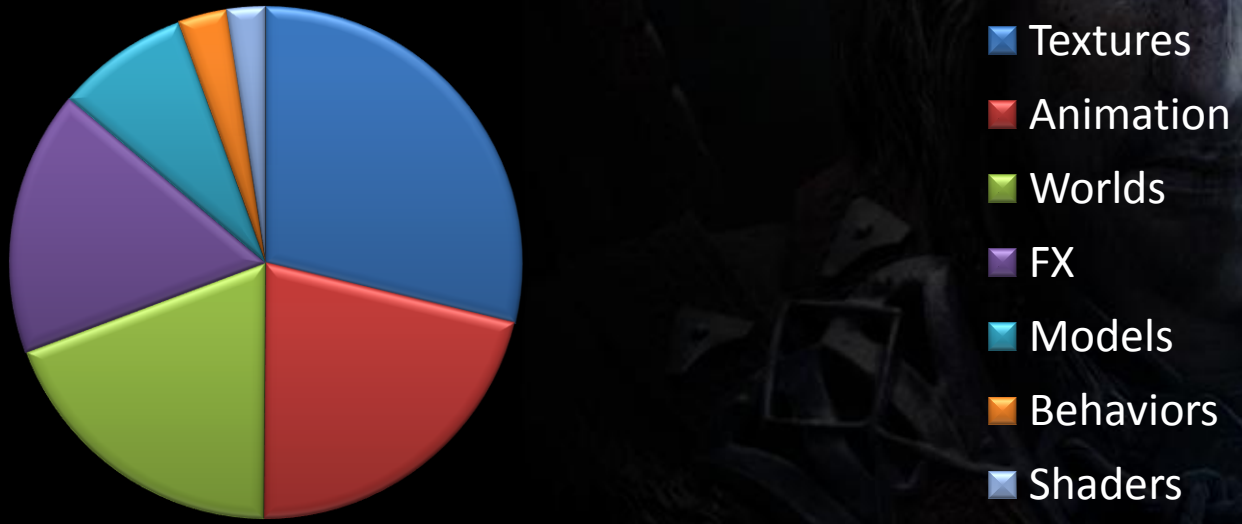
Asset Type Breakdown



Asset Type Breakdown



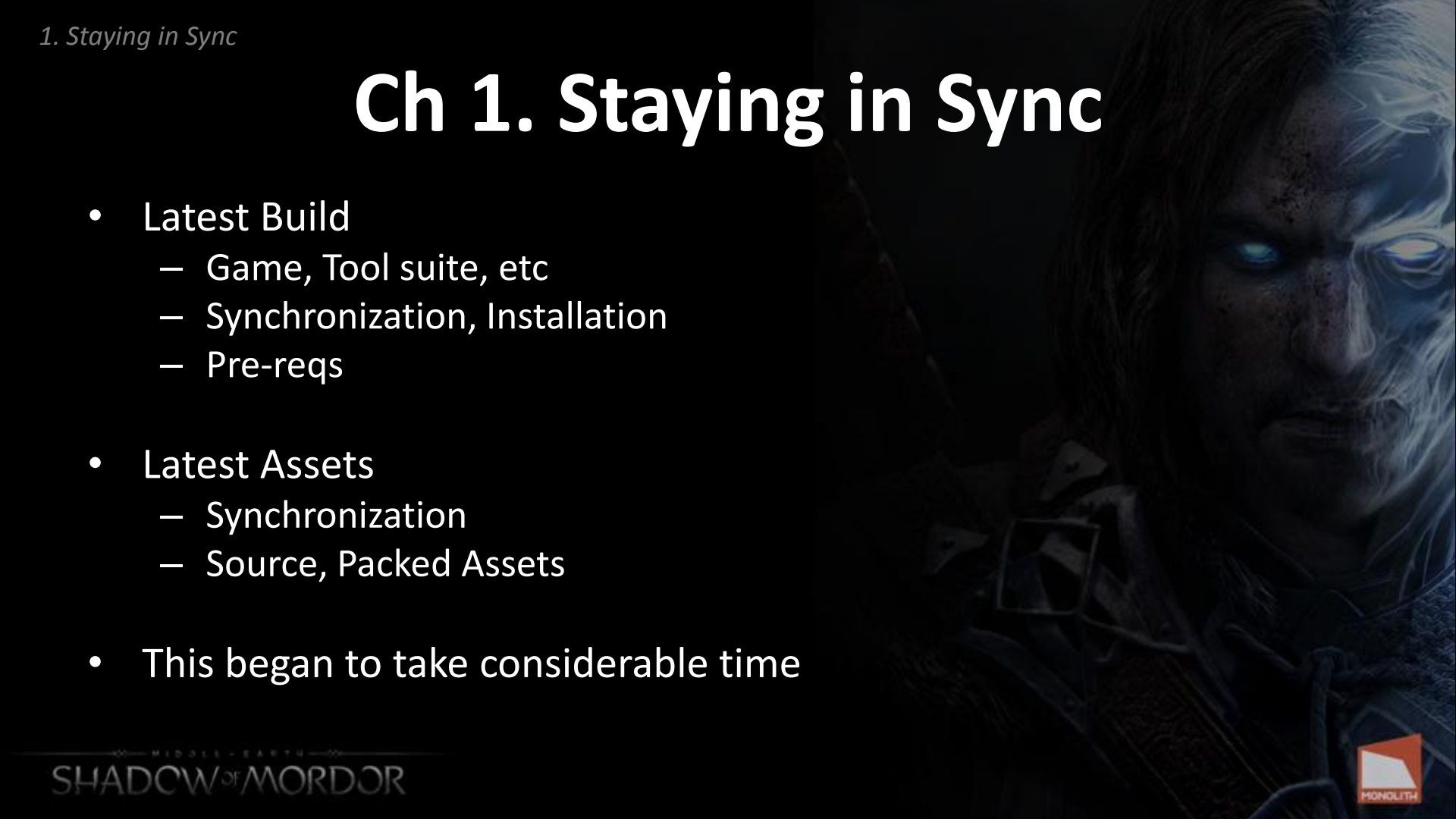
Asset Type Breakdown



Many assets compose this shot

Ch 1. Staying in Sync

- Latest Build
  - Game, Tool suite, etc
  - Synchronization, Installation
  - Pre-reqs
- Latest Assets
  - Synchronization
  - Source, Packed Assets
- This began to take considerable time

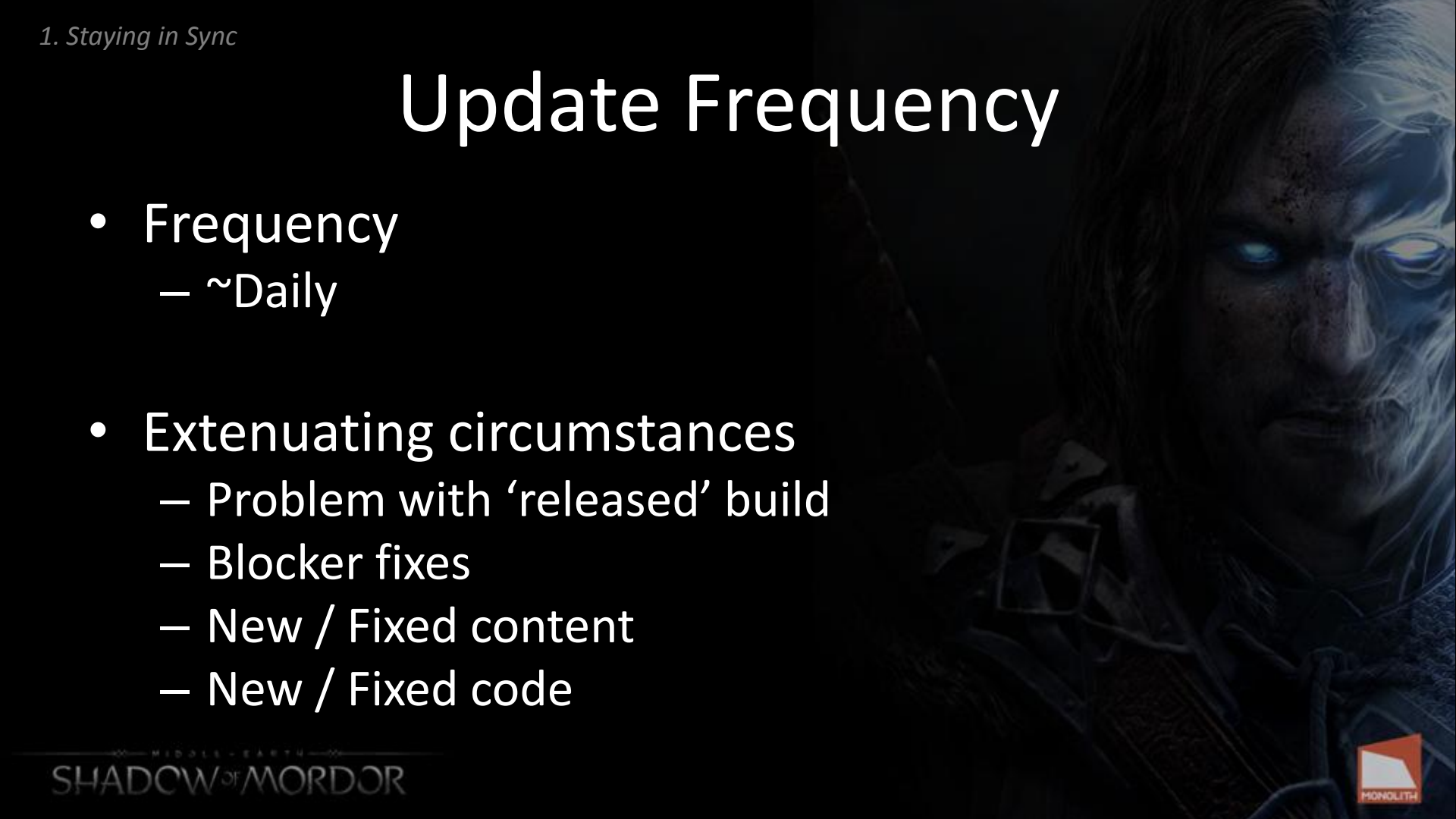


Definitions - Assets

- Assets
  - Source
    - Written and read by tools
    - Contains authored content
  - Packed
    - Written by packing process, consumed by runtime
    - Contains content loaded by game

Update Frequency

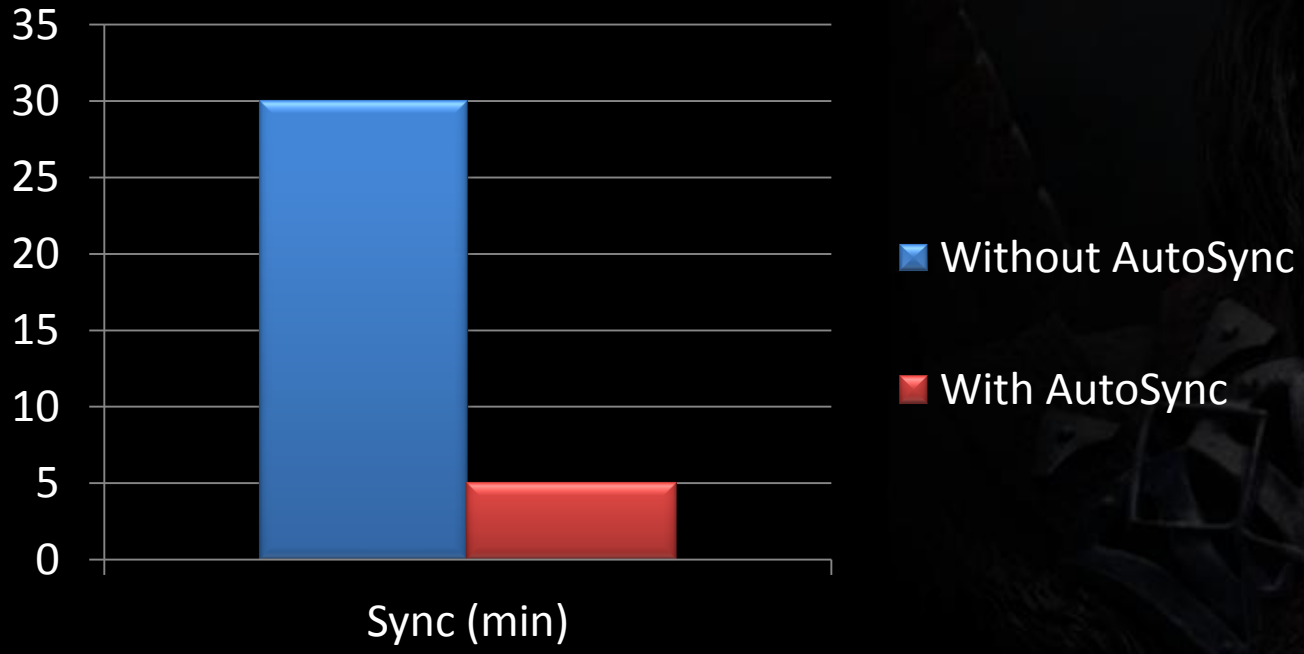
- Frequency
  - ~Daily
- Extenuating circumstances
  - Problem with ‘released’ build
  - Blocker fixes
  - New / Fixed content
  - New / Fixed code



Automated Syncing

- Pros
  - Offline time is free
- Cons
  - Nightly builds may not be finished
  - Requires infrastructure
    - Stagger to avoid server hit
    - ‘Offline’ is relative

Average sync times (6x faster)



Sync Faster

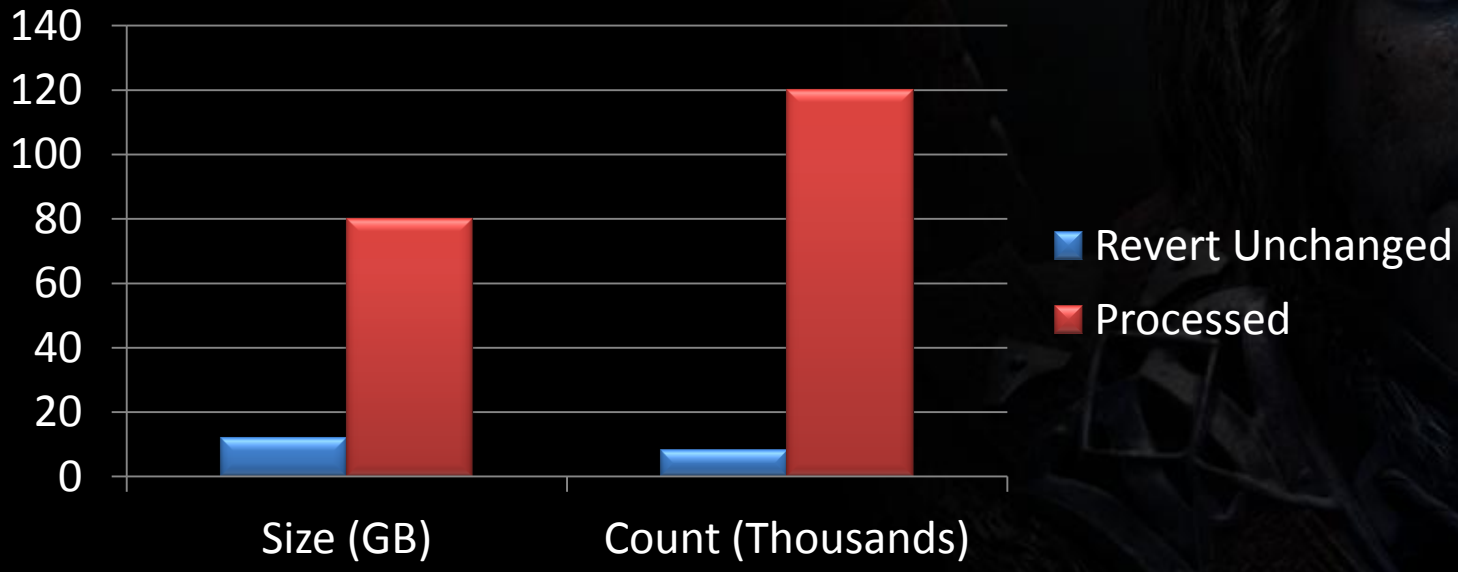
- Ensure sufficient network speed
  - Network pipe truncated
  - Old, bad switch
  - 360 network cable (2 twisted pair)
- Can be detected on PC

Sync Less

- Deterministic file format
- Processed asset files are 1:1 – produced the same way every time
- Don’t check in no-op revisions

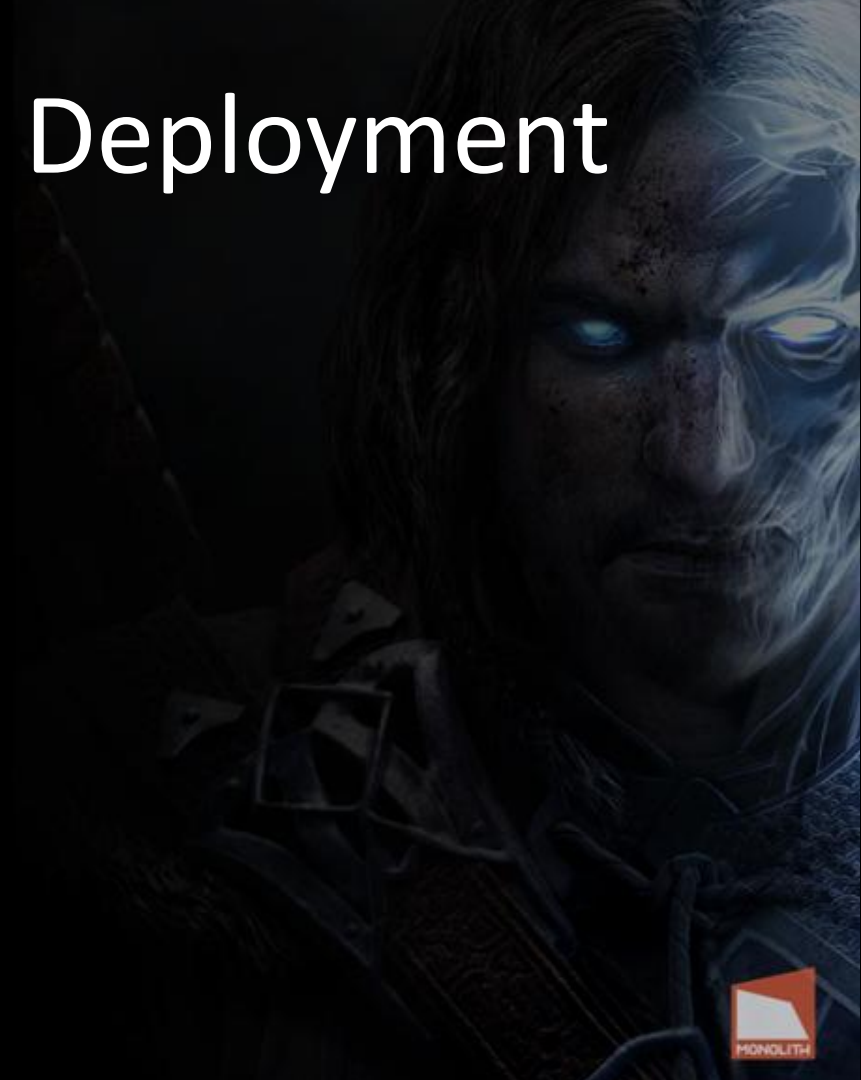
- Do not embed
  - Timestamps
  - Unseeded (new) GUIDS
  - Absolute Paths

Churn reduction (10x smaller)



Automated Build Deployment

- Pros
  - More Time Saved
- Cons
  - May need to reboot
    - Pre-reqs
  - What if binaries are still open
    - Potentially unsaved work
  - Significant state change, risk



Takeaway – Staying in Sync

- Candidate for automation
- Sync Faster
  - Ensure optimal connection
- Sync Less
  - Deterministic format

Ch 2. Load Lag

Loading Files to Author

- Began to take a long time
  - More data to load
  - More complex assets, split over multiple files
  - More complex operations

- Objective here is to get to edit as quickly as possible

Load Smarter

- Examine the source file format

- Emphasis on minimum footprint, maximum load speed
  - How fast can you get the data into memory (Disk)
  - How fast can the data be interpreted (CPU)

LTA Source file format

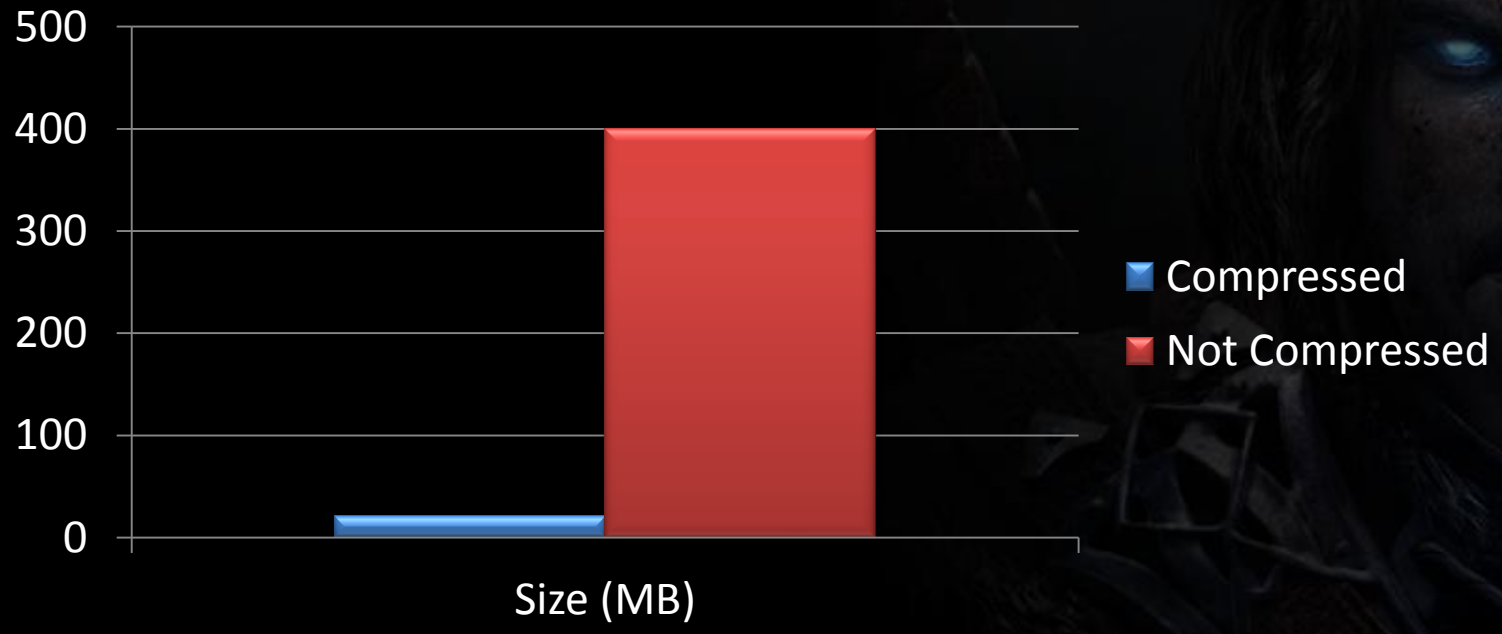
- LTA – Lith Tech ASCII (xml like text)
  - Human Read-able
  - Large file size
  - Slow to Interpret

- Encoded, Compressed ASCII
  - Smaller on disk
  - Faster to get in memory
  - Slower to Interpret

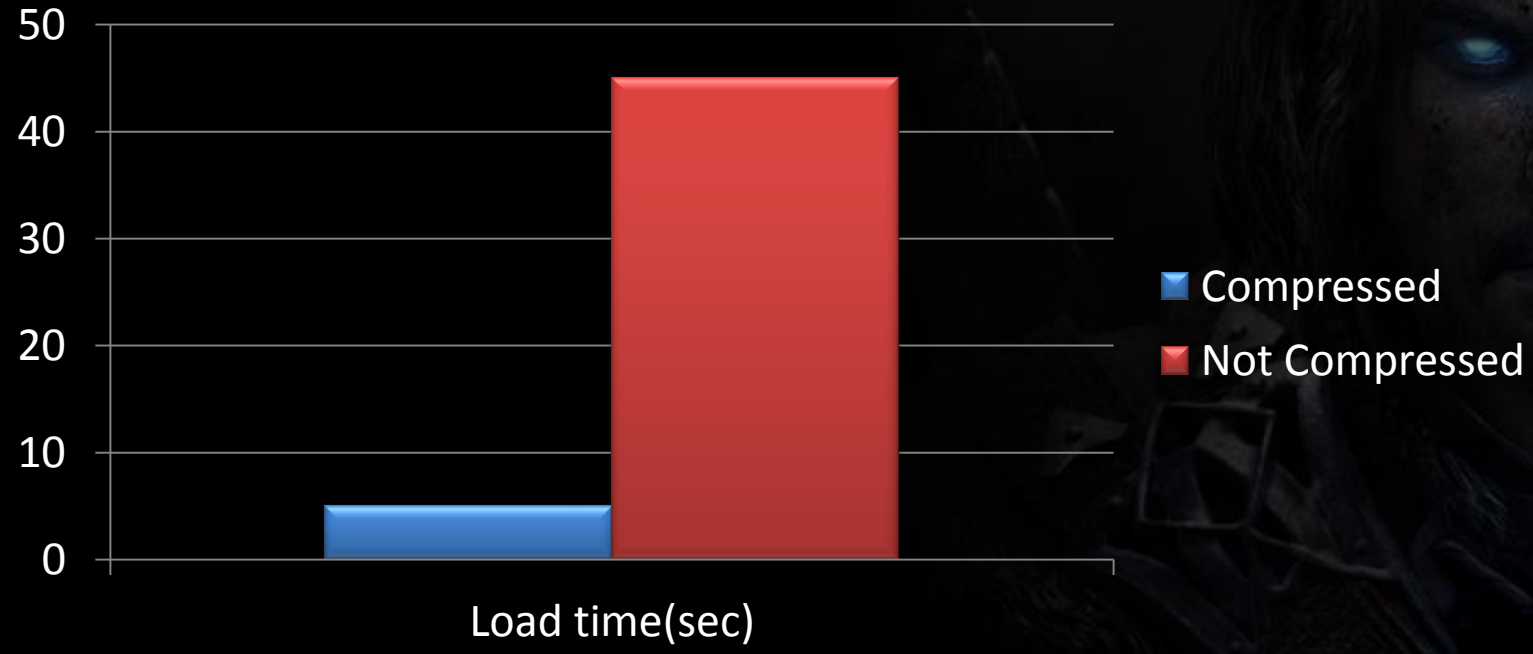
LTA Source file format

- Compressed Binary representation
  - Smaller on disk
  - Far faster, no need to parse text
  - File tree roots compressed independently (Zlib)
  - Load / decompress in parallel, or partially
  - CRC checking exposes file corruption
  - Provide utilities to convert to human read-able format

Footprint on disk (10x smaller)



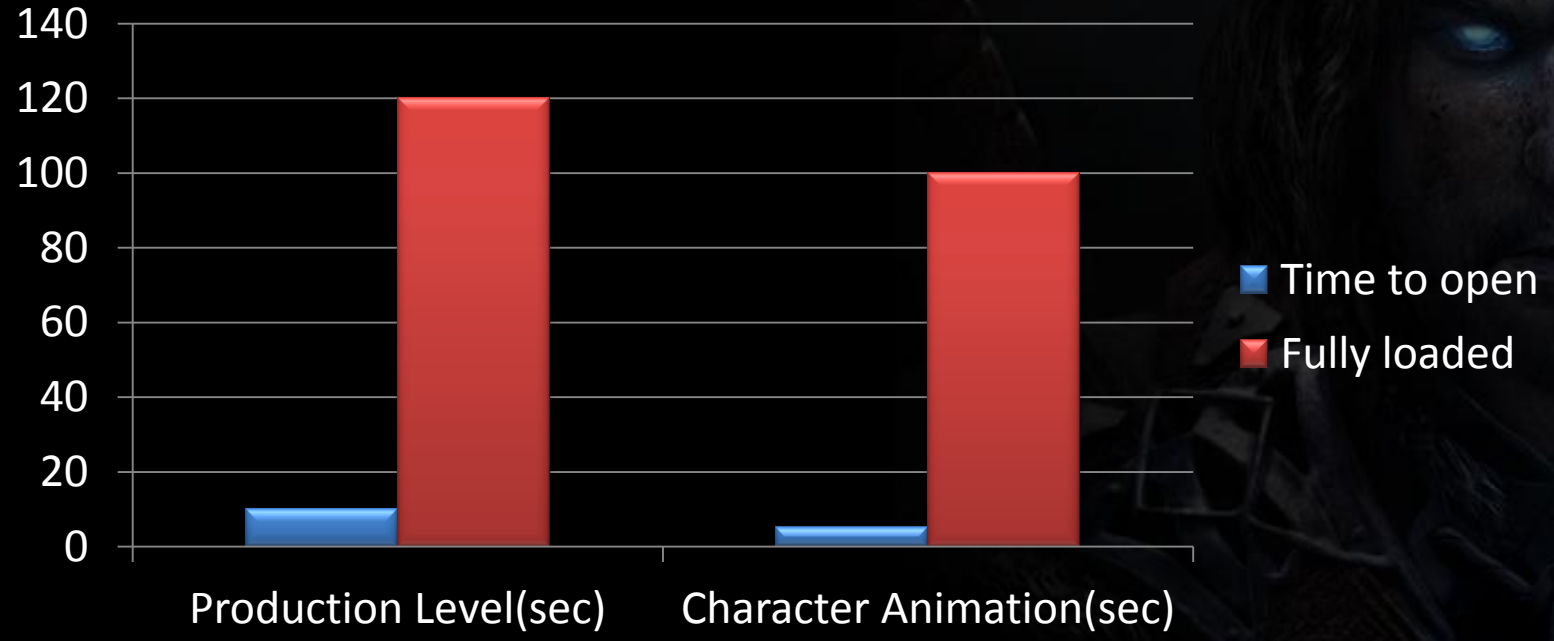
Load Time (10x faster)



Load on demand

- Don’t load most data until it is needed
  - Requires user actions to prompt additional loading
  - Ideal for self contained workflow
  - Tree controls work well for this
- Requires appropriate granularity of source files
  - Tough to do with a single file

Delay Load Times (15x faster)

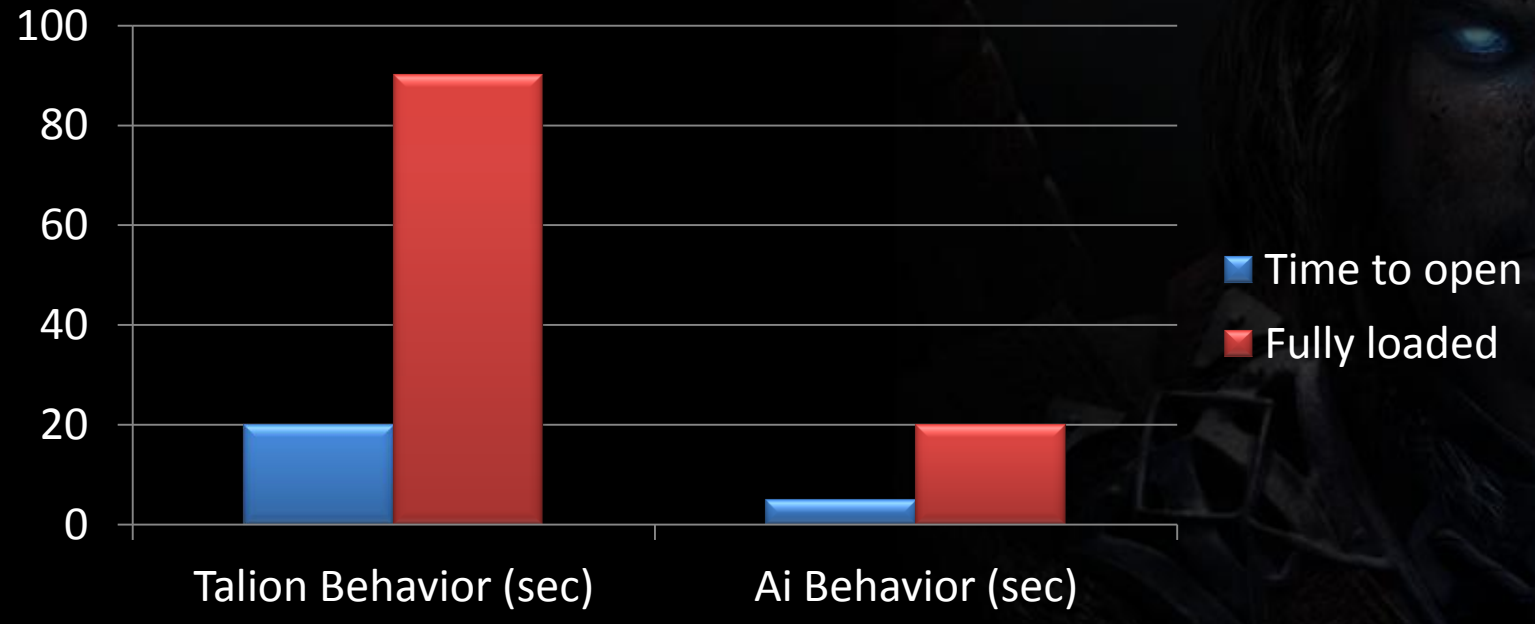


Load in the background

- Only load some of the data up front
  - Let the user begin editing when the initial chunk has loaded
  - Data needed, but not immediately
  - User blocks if necessary

- Ex. Visual Assist, Visual Studio Intellisense

Background Load Times (5x)



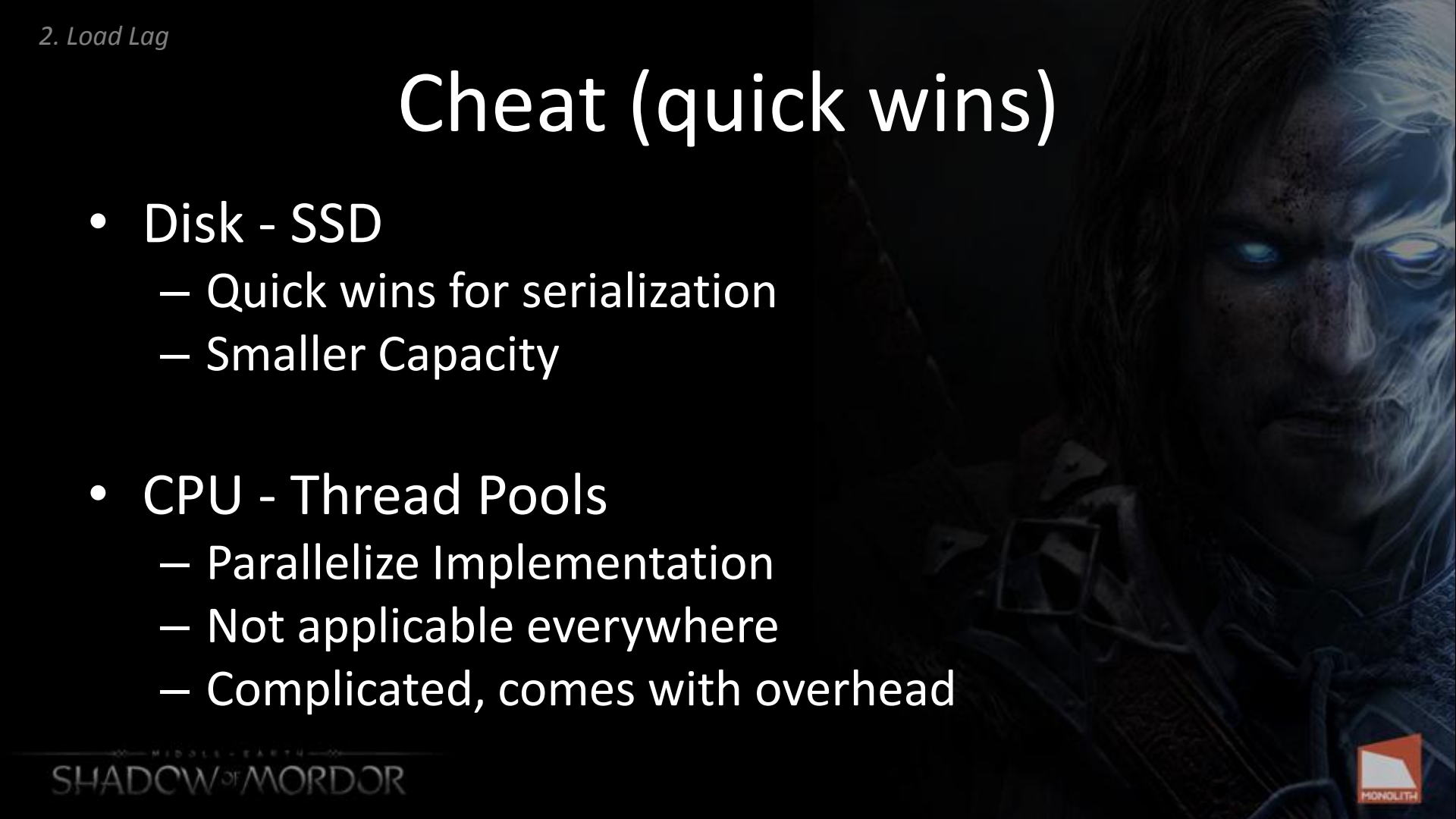
Death of a thousand cuts

- Need to load a ton of little files
  - Hard drive is bad at this
- Asynchronous IO will help
- Build a ‘checkpoint’
  - Single file, compressed for minimal footprint
  - Patch in cases where its out of date
  - 90k files / 800mb to checkpoint at 30 mb

Load Time (20x faster)

Cheat (quick wins)

- Disk - SSD
  - Quick wins for serialization
  - Smaller Capacity
- CPU - Thread Pools
  - Parallelize Implementation
  - Not applicable everywhere
  - Complicated, comes with overhead



Takeaway – Load Lag

- Load Smarter
  - Reduce Disk, CPU costs
  - Be creative
- Defer Loading
  - Temporarily, Indefinitely
- Cheat with hardware

Ch 3. General Performance

Nothing is future-proof

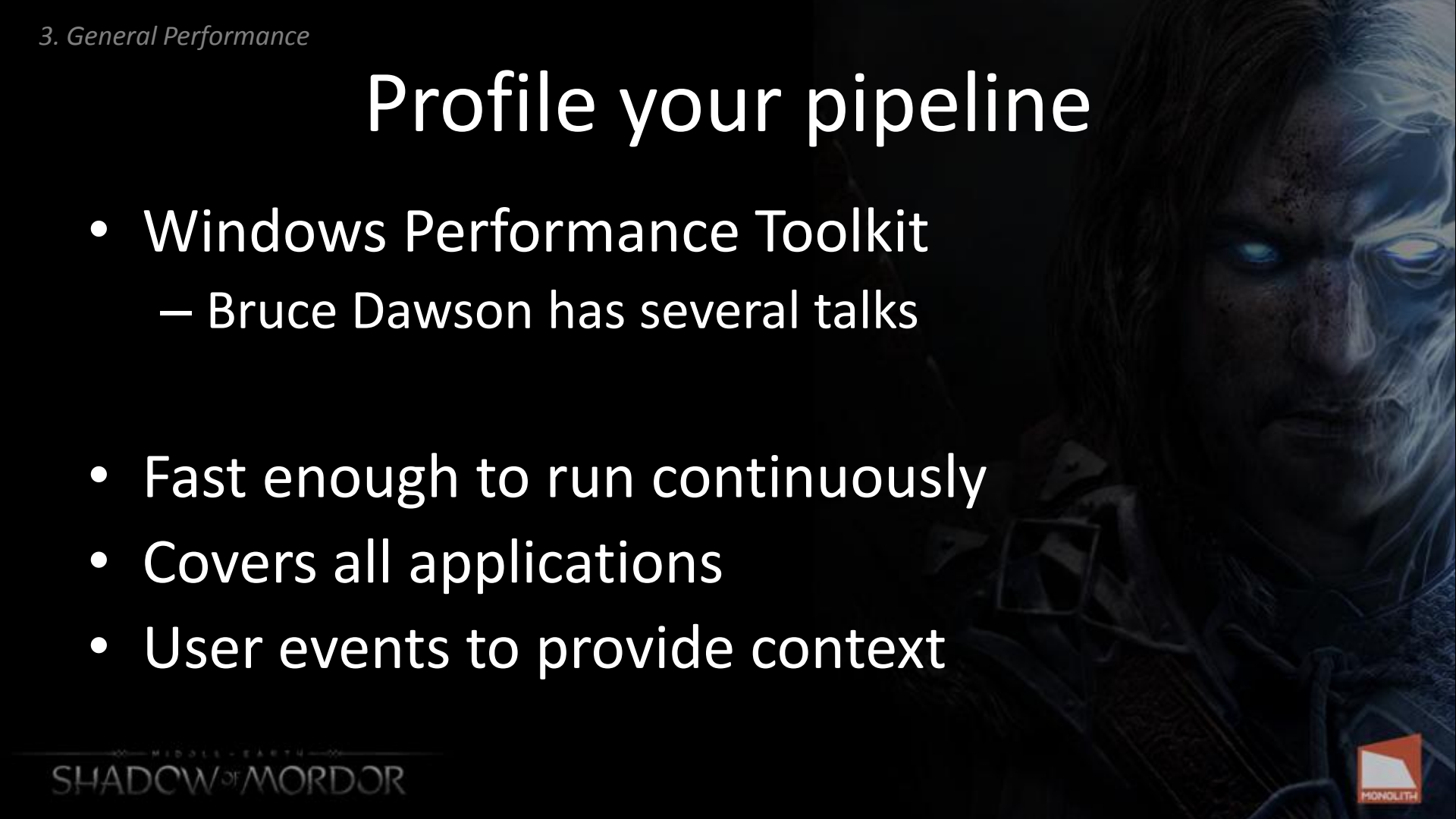
- There are always areas that scale poorly
  - Not realistic to support every future path

- Plan to spend time
  - Identifying points of failure
  - Invest resources addressing these

Profile your pipeline

- Windows Performance Toolkit
  - Bruce Dawson has several talks

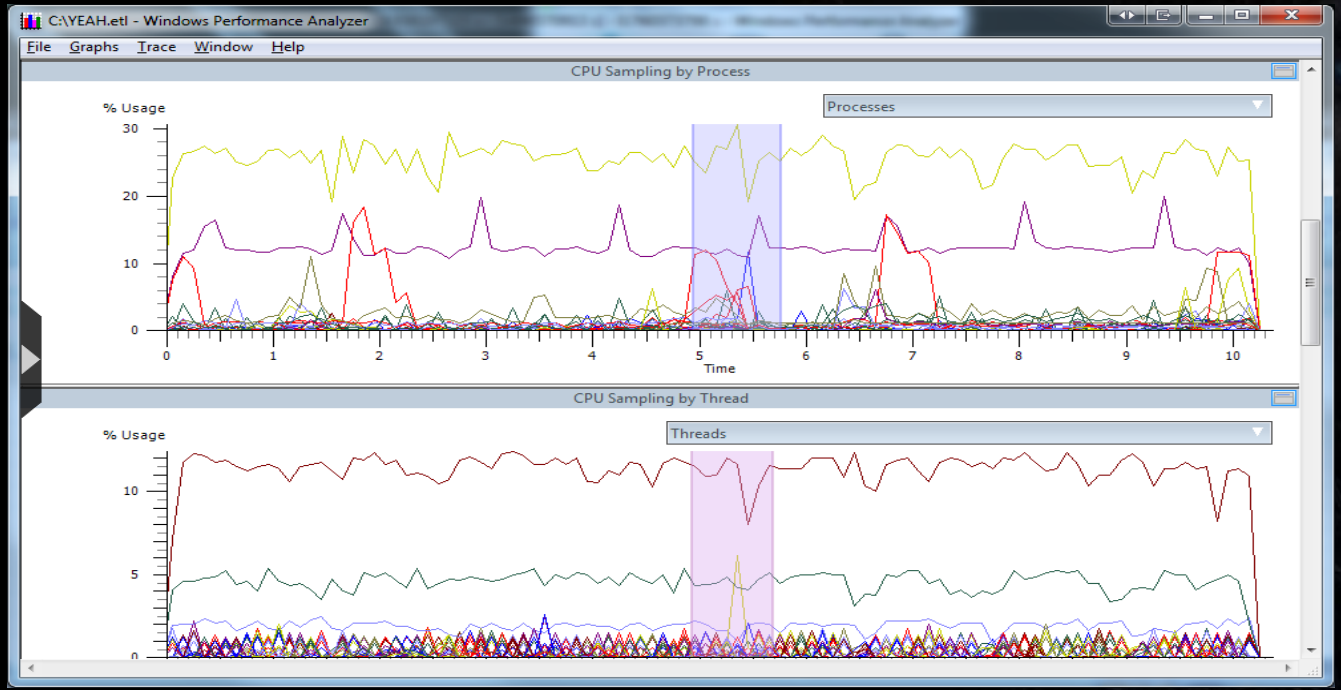
- Fast enough to run continuously
- Covers all applications
- User events to provide context



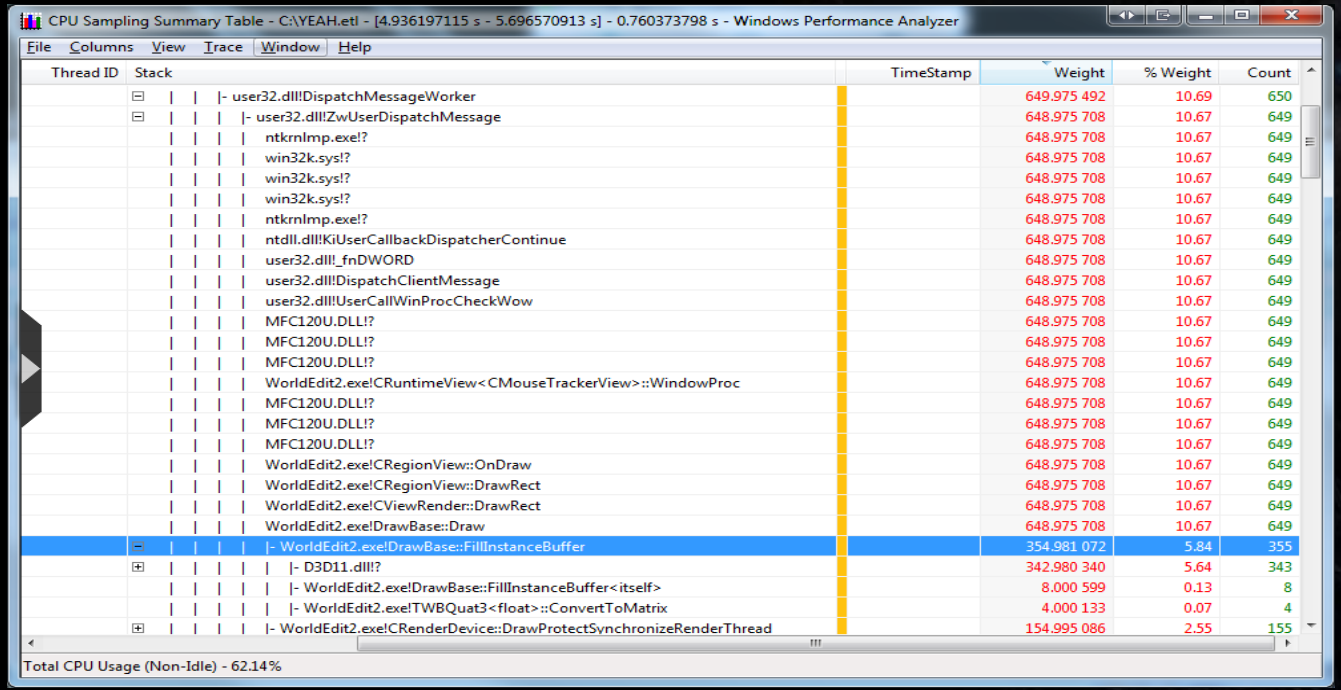
Performance Reports

- Help users provide feedback
- Gather high level metrics
- Identify / analyze pain points
- Detect unexpected use

Xperf Graph Image



Xperf Images #2

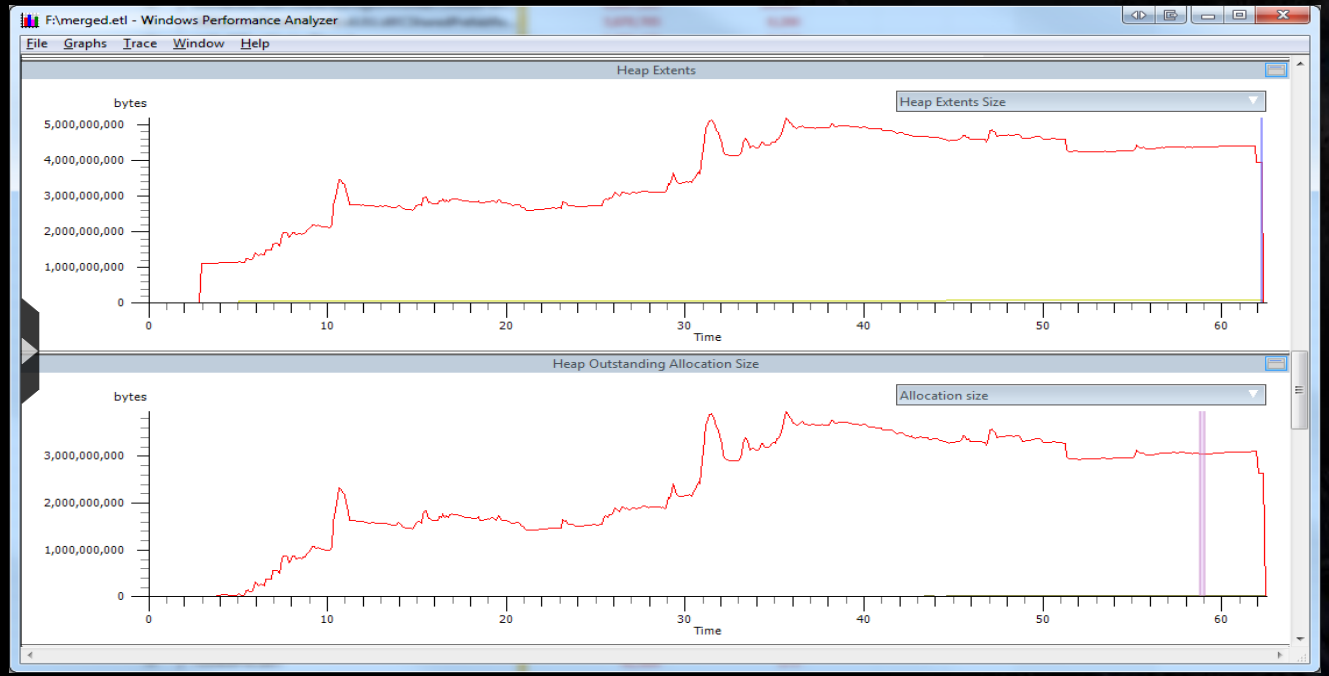


Heap tracking

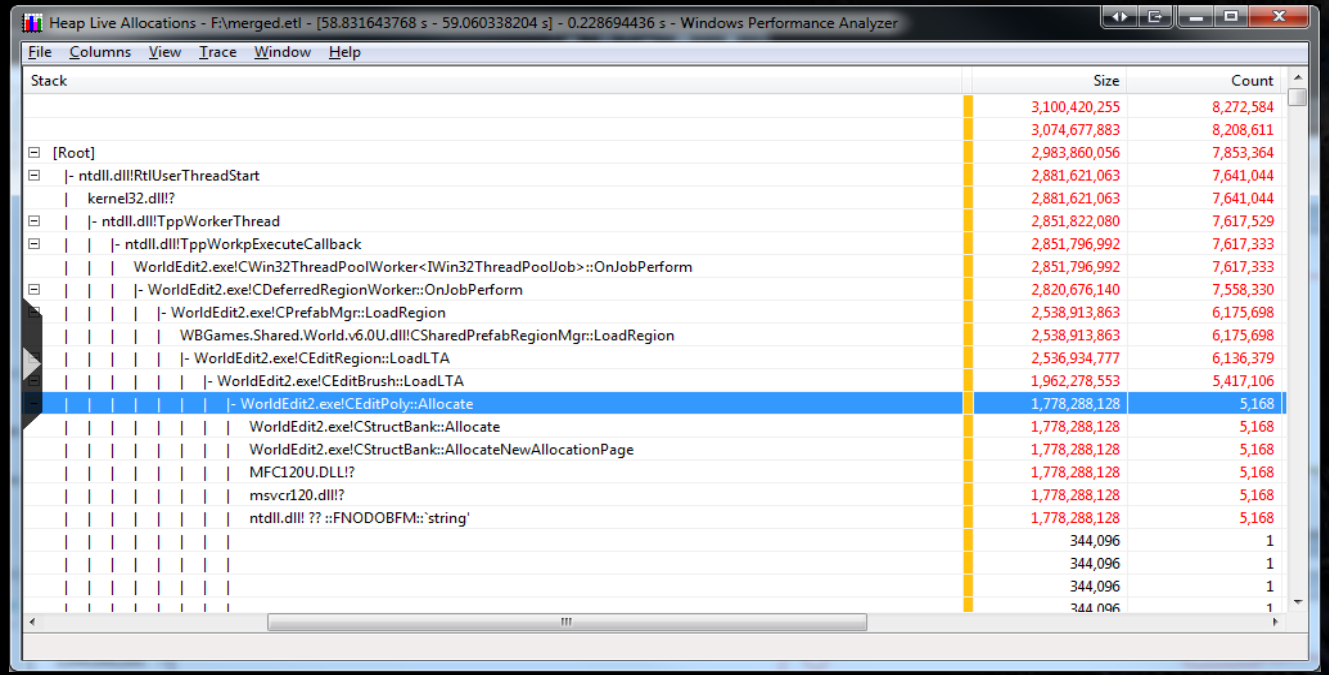
- Xperf does this as well
  - Much slower, application specific
  - Generates far more data
  - Captures every allocation

- When Ram is paged, the disk suffers
  - Use less memory

Xperf Heap Image 1



Xperf Heap Image 2



WMI provides useful data

- Windows Management Instrumentation
- Exposes
  - Hardware specifications
  - Hardware utilization details

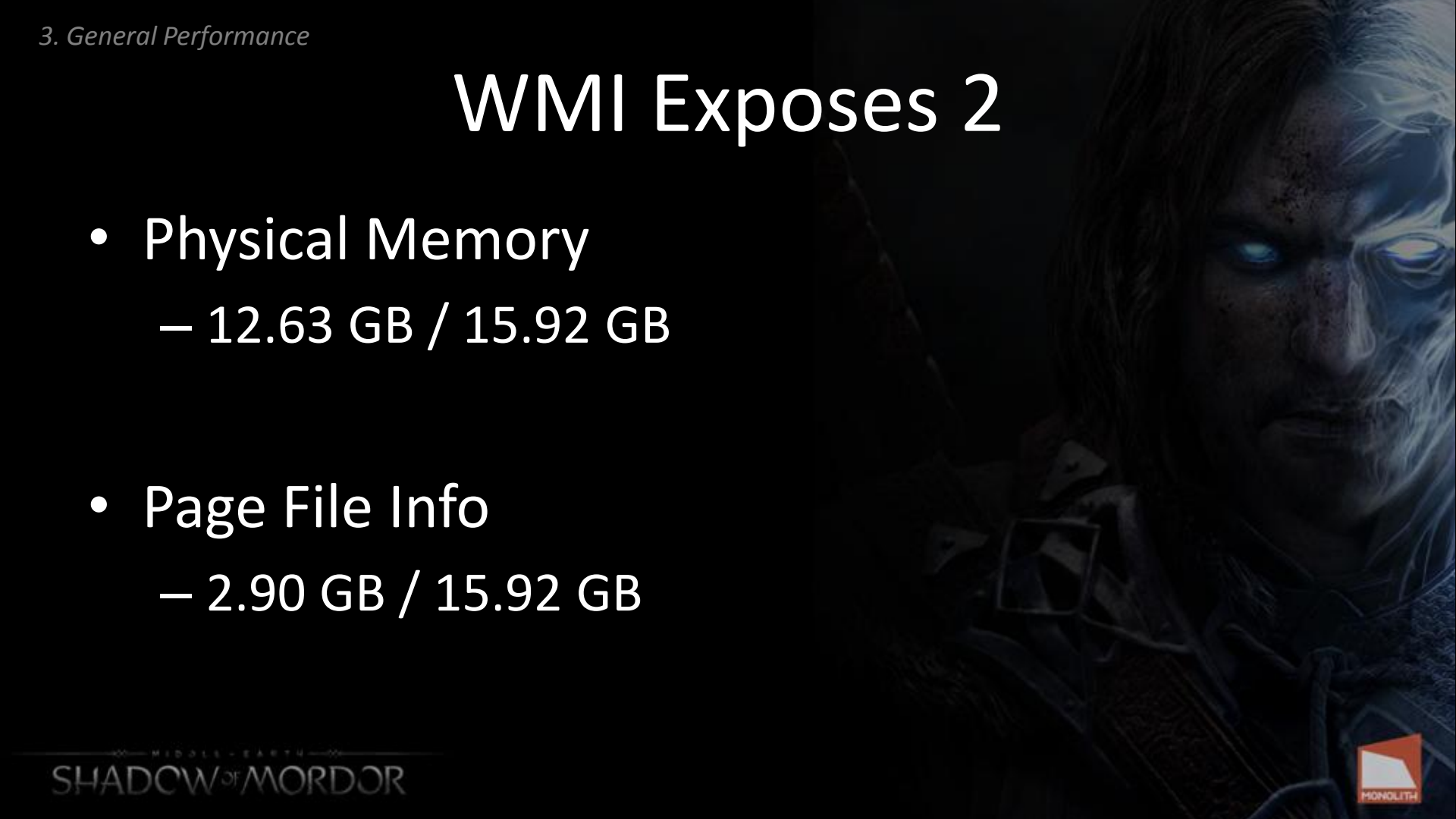
- WMI Implementation Details

WMI Exposes

- Processor Information
  - Intel(R) Core(TM) i7-3930K CPU @ 3.20GHz (6p, 12l cores)
- GPU Information
  - NVIDIA Quadro K5000
  - Dedicated 66.99 MB / 3.94 GB (in use)
  - Shared 3.84 GB / 7.42 GB (in use)

WMI Exposes 2

- Physical Memory
  - 12.63 GB / 15.92 GB
- Page File Info
  - 2.90 GB / 15.92 GB

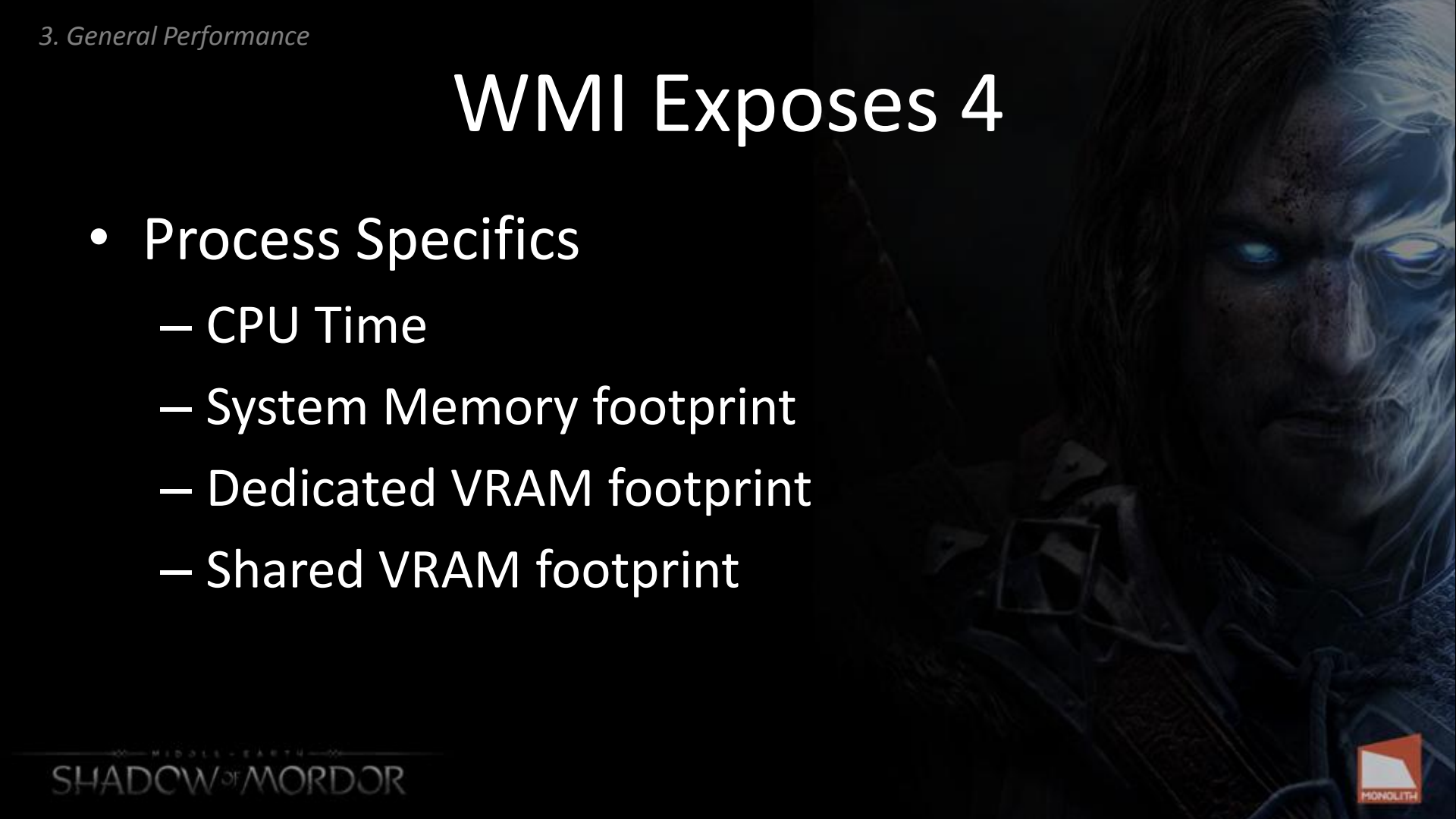


WMI Exposes 3

- Network Adapter Information
  - Intel[R] 82579LM: 1000 mbps
- Volume Information
  - C:\ - Available (51.60%) 240.33 GB / 465.75 GB
  - D:\ - Available (43.90%) 408.89 GB / 931.51 GB

WMI Exposes 4

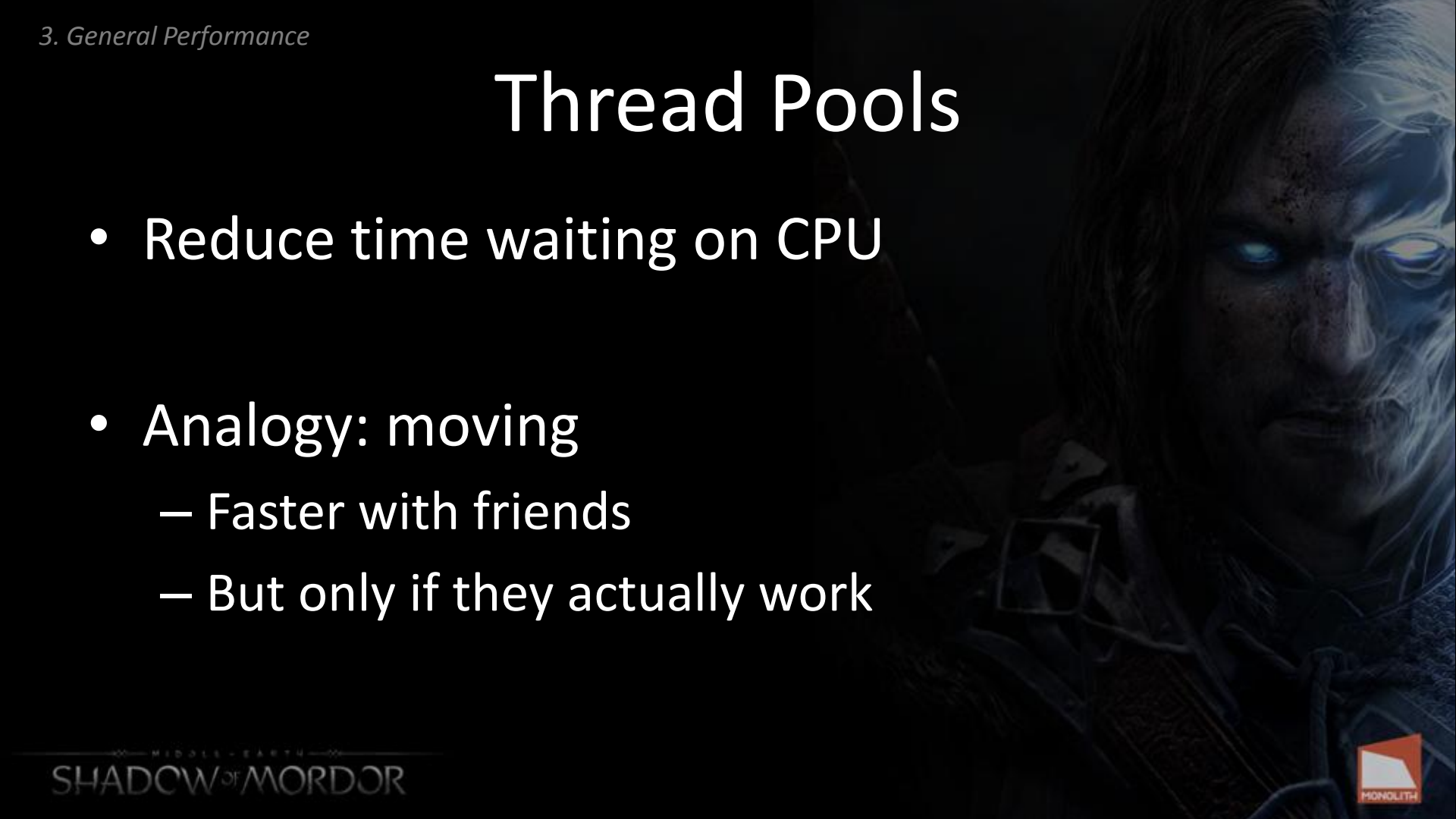
- Process Specifics
  - CPU Time
  - System Memory footprint
  - Dedicated VRAM footprint
  - Shared VRAM footprint



Thread Pools

- Reduce time waiting on CPU

- Analogy: moving
  - Faster with friends
  - But only if they actually work

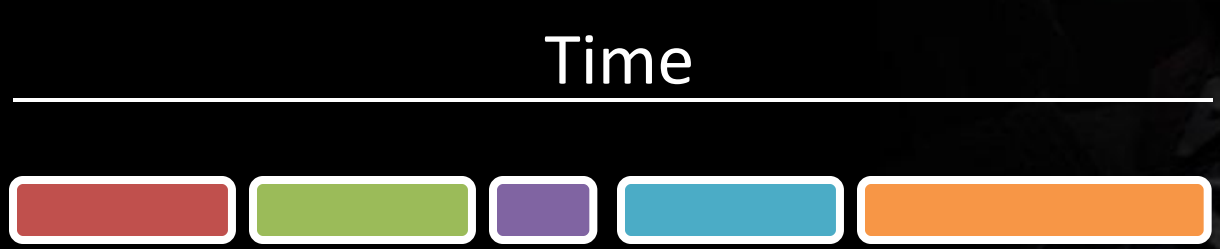


Thread Pools

- Pros
  - Can yield linear speed improvements
  - Fairly easy to drop in if appropriate
- Cons
  - Will not offset poor algorithm choices
  - Core contention
  - More complicated
  - Only as fast as the slowest job

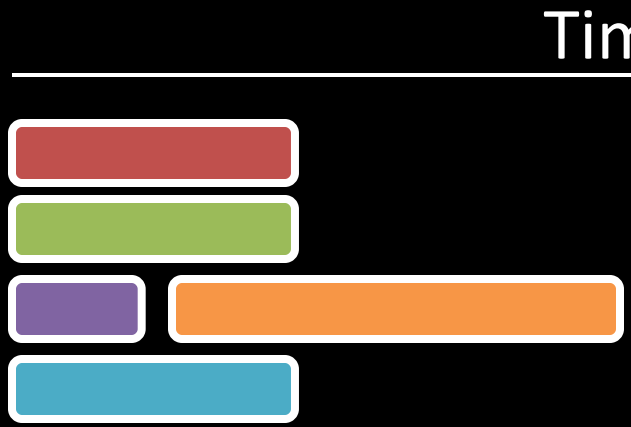
Thread Pools

- 5 jobs done in sequence on a single core

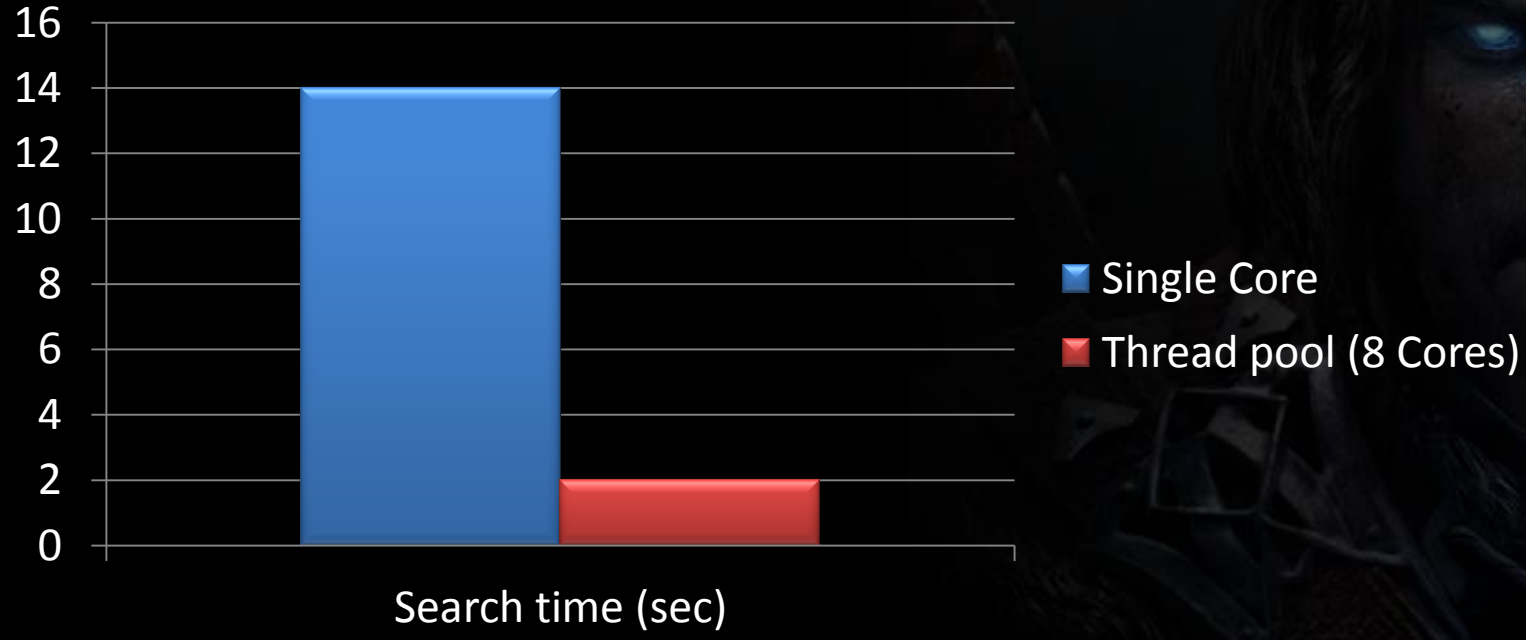


Thread Pools

- Same 5 jobs performed on 4 cores



Database Find Time (7x faster)



Takeaway – General Performance

- Profile your pipeline
  - What is the issue?
  - Don’t guess and hope
- High level information also useful

Ch 4. Asset Processing

Asset Processing

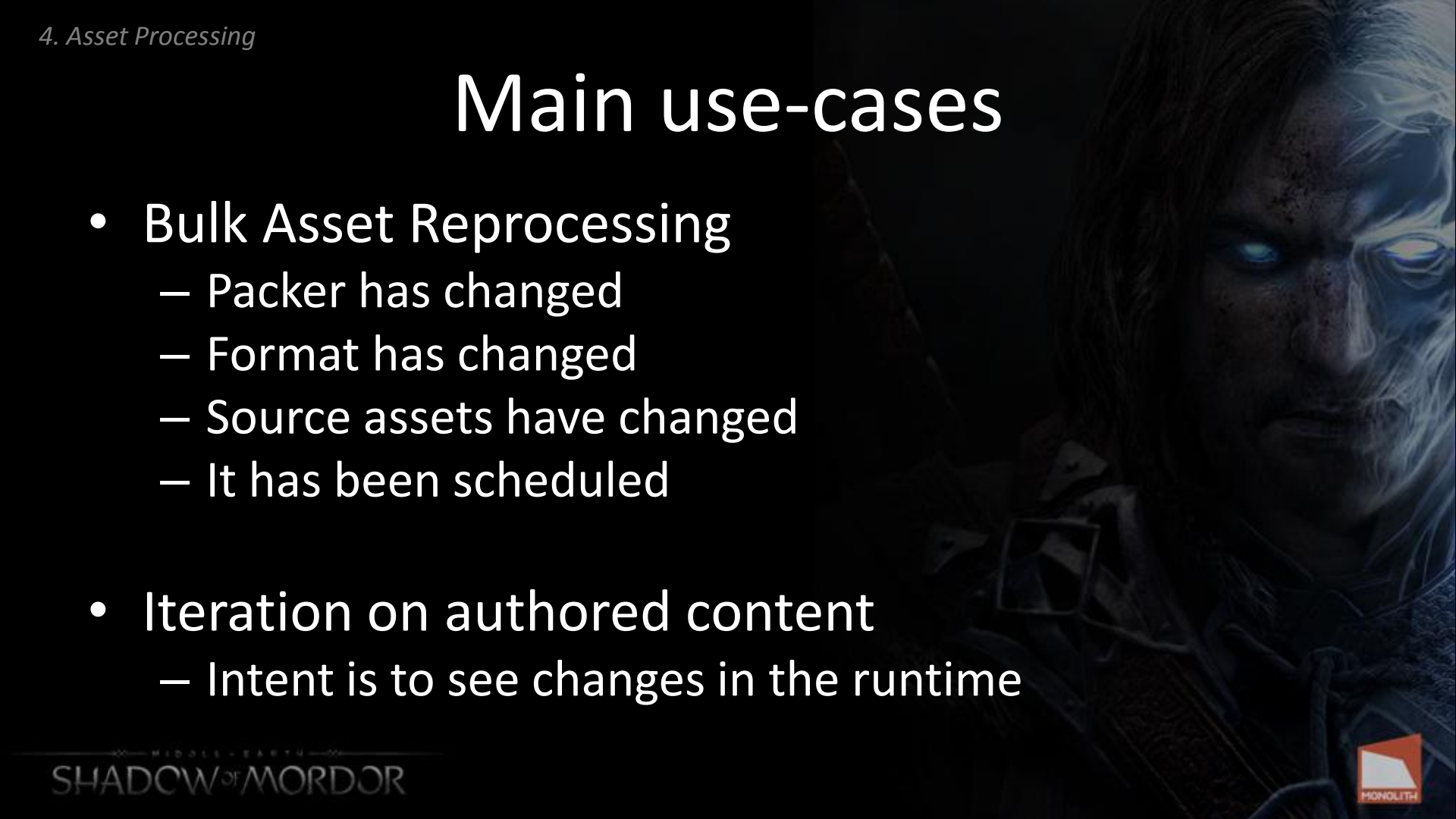
- The transformation of an asset so it can be consumed by the target
- We call this ‘Packing’
  - Known as baking, cooking, etc
- Time spent here on SOM went through the roof

Where is the time going?

- More assets to pack
- More targets to pack for (5 platforms)
- More offline processing (arms race)
- Many of the assets are larger, more complex
- Serialization
- Synchronize, Commit

Main use-cases

- Bulk Asset Reprocessing
  - Packer has changed
  - Format has changed
  - Source assets have changed
  - It has been scheduled
- Iteration on authored content
  - Intent is to see changes in the runtime

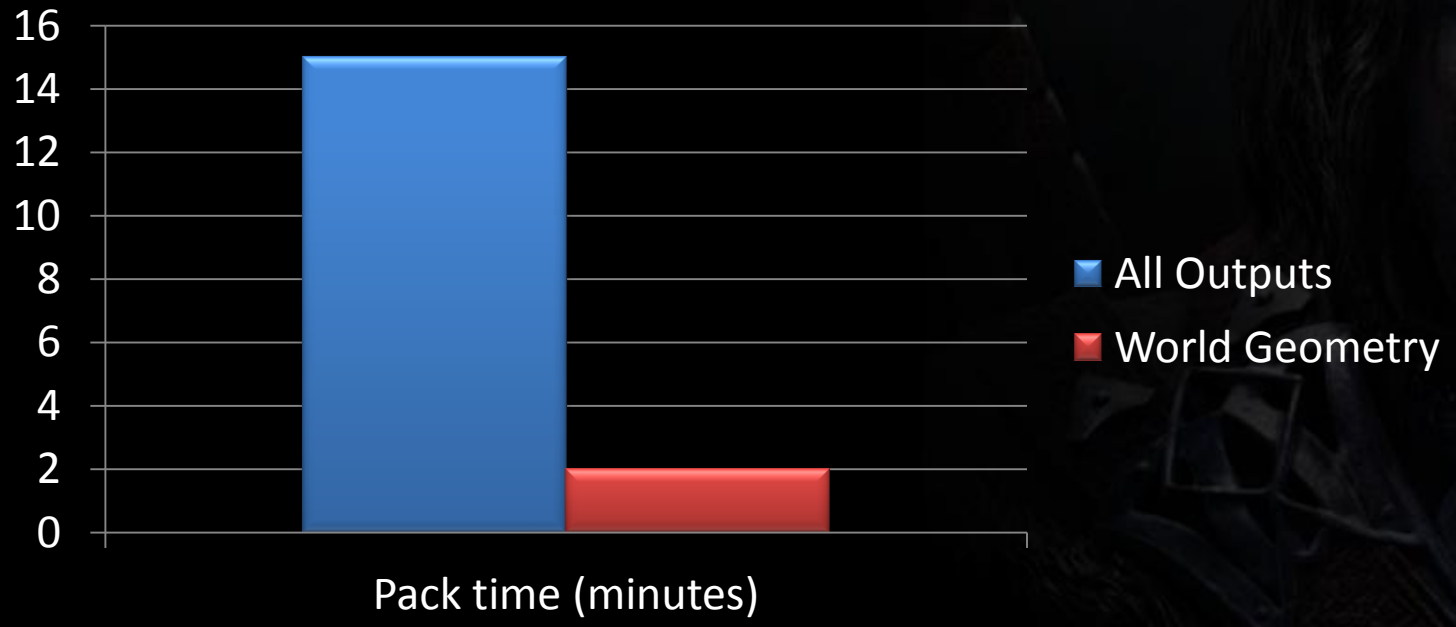


Content Iteration

- This needs to go quickly at all costs
  - No different than compile / link times

- Allow a user to only pack the things affected by their change
- Often means supporting more discrete outputs

World Pack Time (7x faster)



Flexibility > Speed?

- Runtime structures typically optimized for speed, not flexibility
- Consider allowing non-optimal formats to facilitate iteration

- Ex. GDB Patch Database
  - Consumed more memory
  - Improved iteration 1000x

Packing Database (1000x faster)

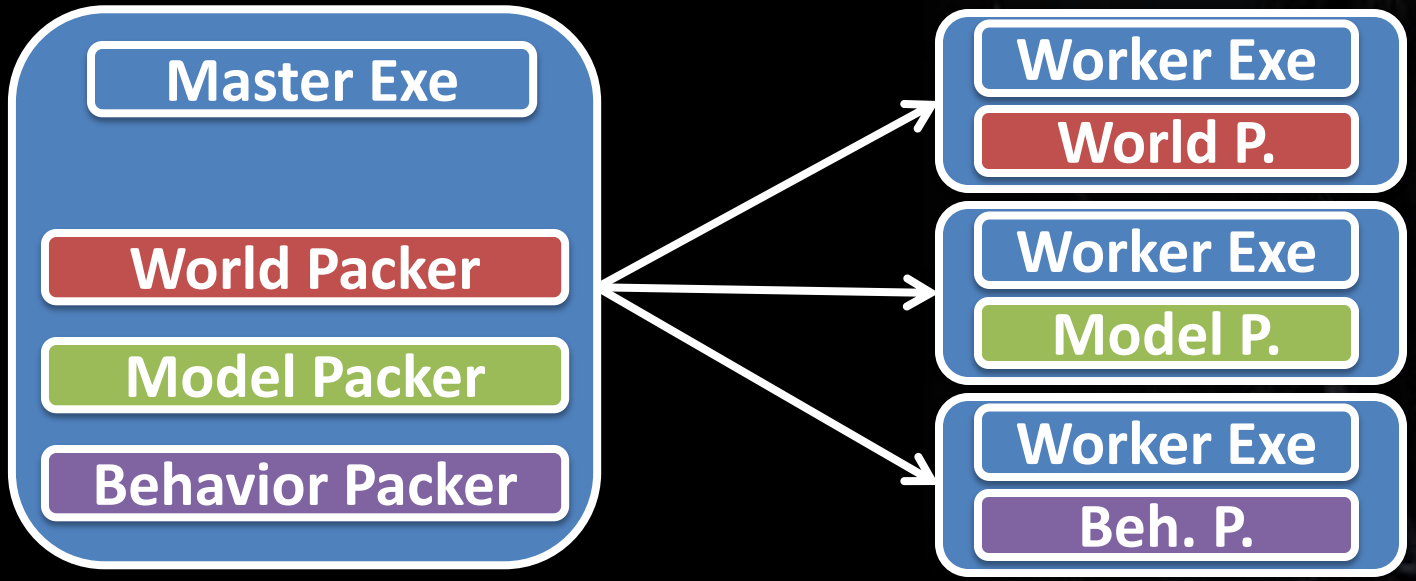
Bulk case - Process pools

- Pros
  - Win of thread pools without complexity or headache of multi-threaded code
    - New environment
  - More resilient
    - Crashing process self-contained
  - New address range

Bulk case - Process pools

- Cons
  - More Overhead
    - Slow to spin up
  - No shared memory
    - More serialization
  - Requires more infrastructure
  - Requires inter-process communication

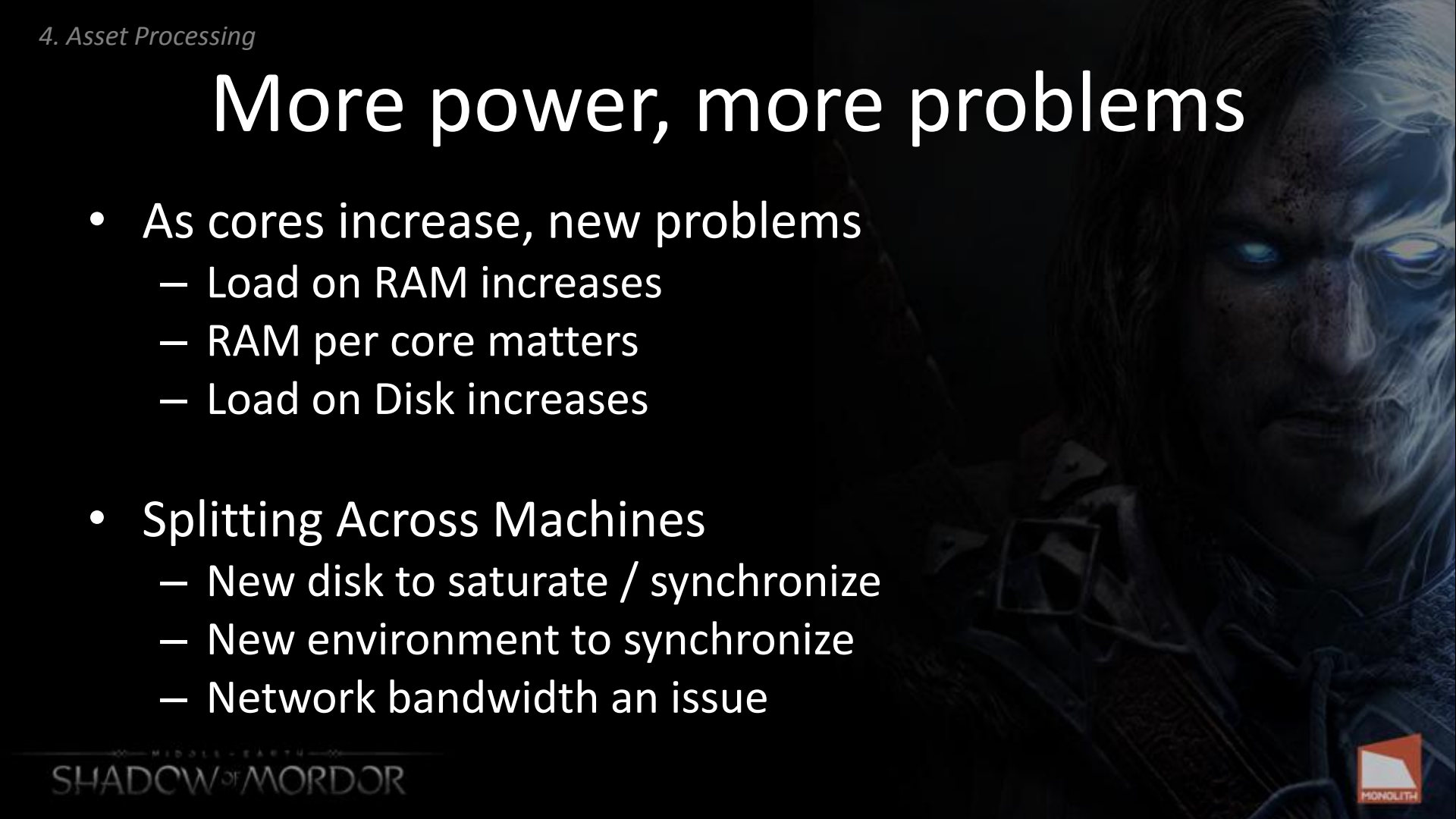
Asset Builder



More power, more problems

- As cores increase, new problems
  - Load on RAM increases
  - RAM per core matters
  - Load on Disk increases

- Splitting Across Machines
  - New disk to saturate / synchronize
  - New environment to synchronize
  - Network bandwidth an issue



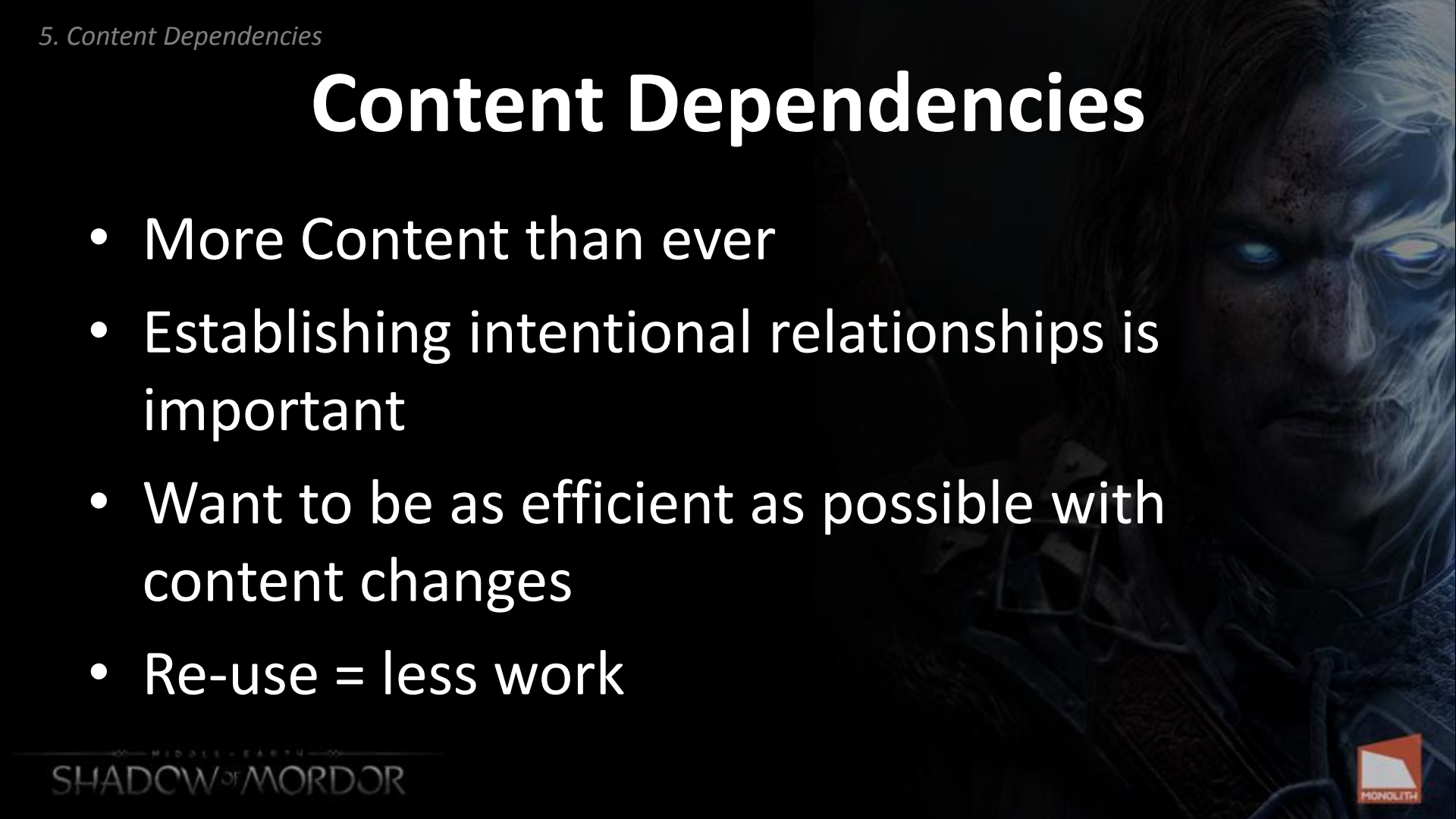
Takeaway– Asset Processing

- Content Iteration
  - Take as little time as possible
  - Process only what has changed
- Bulk Reprocessing
  - Farming more hardware requires infrastructure

Ch 5. Content Dependencies

Content Dependencies

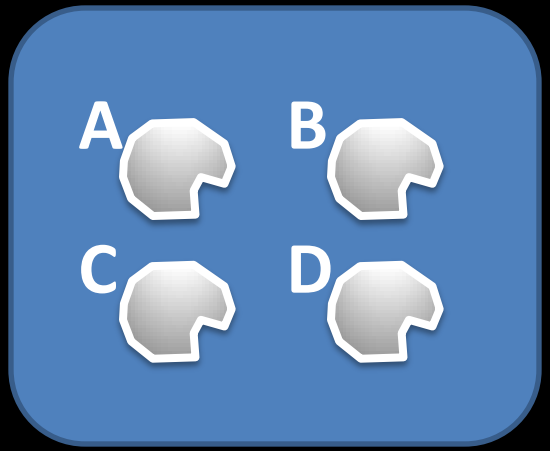
- More Content than ever
- Establishing intentional relationships is important
- Want to be as efficient as possible with content changes
- Re-use = less work

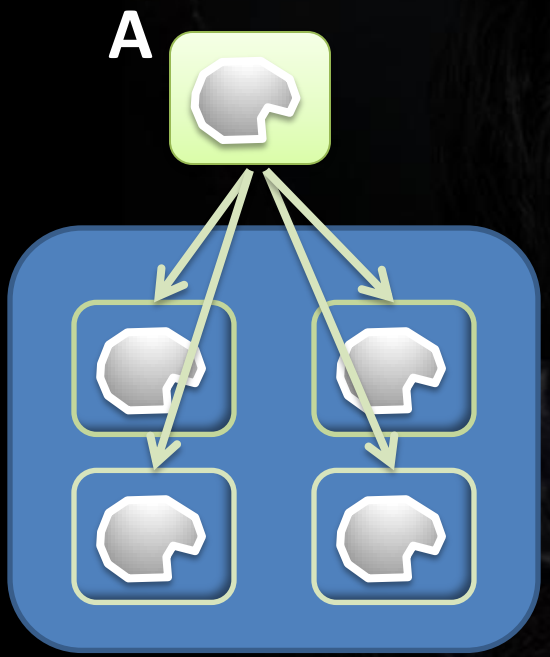


Asset Encapsulation

- ‘Has a’ relationship
- Formally wrap common data as reference-able prefab
- Change def. once, it propagates
- Can’t forget to change it in multiple places
- Complications when ‘slightly’ different

Diagram - Encapsulation

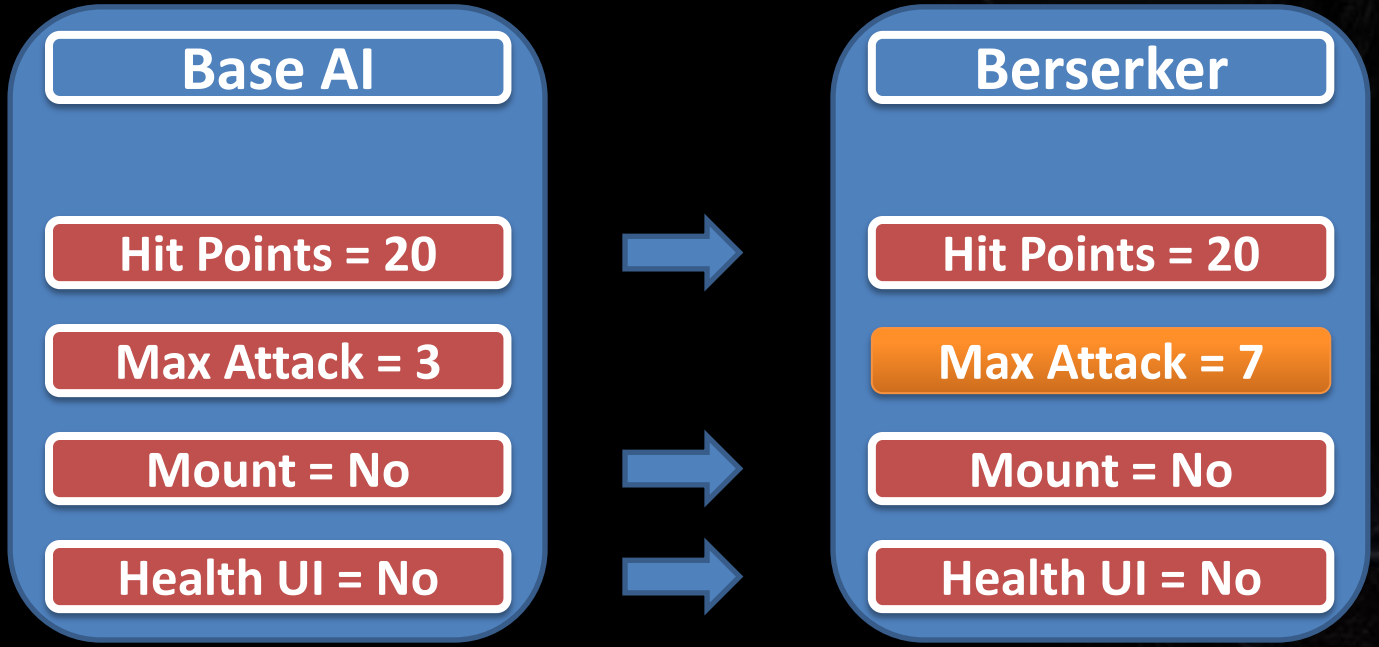




Data Inheritance

- ‘Is A’ Relationship for data
- Parent is an instance rather than a schema
- Allows the adoption of a subset of values
- Property sheets work well here

Diagram – Data Inheritance



Combining The Two

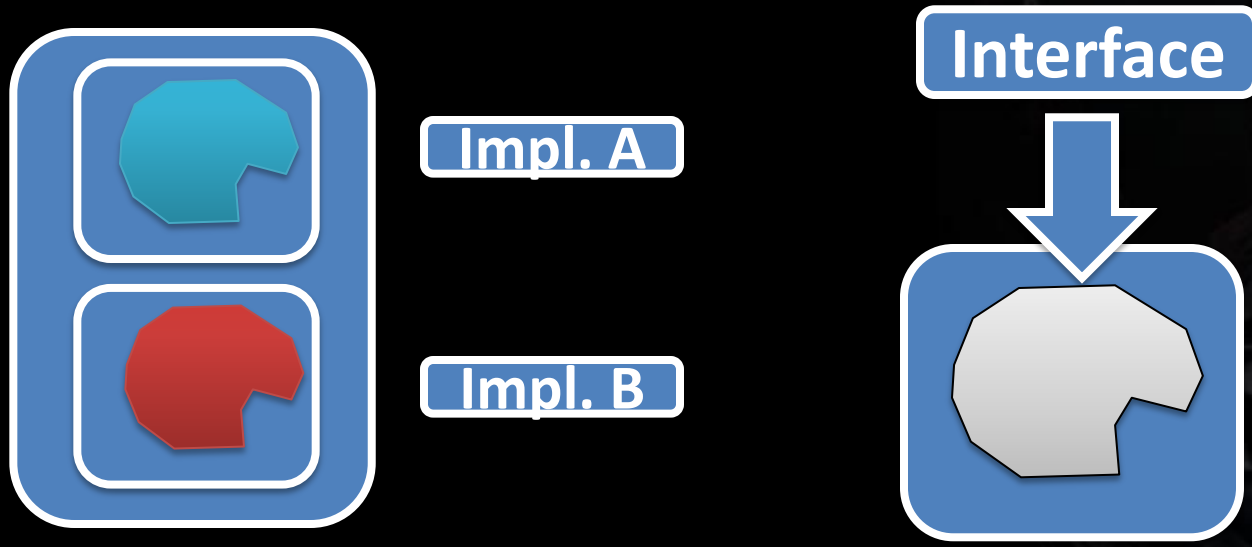
- Ex. Rock Prefab in a world
- Stamp it down in the world a few times
- Instances have a unique transform
- What if prefab has another property we want to adjust per instance
  - Specific properties
  - Supports unique game-play

Combining The Two

- The Instance needs a means of influencing the implementation

- 1. Asset internally provides multiple implementations that can be selected by the instance
- 2. Asset exposes an interface which is implemented by the instance

Instance Influence



Ex. Nested Composition

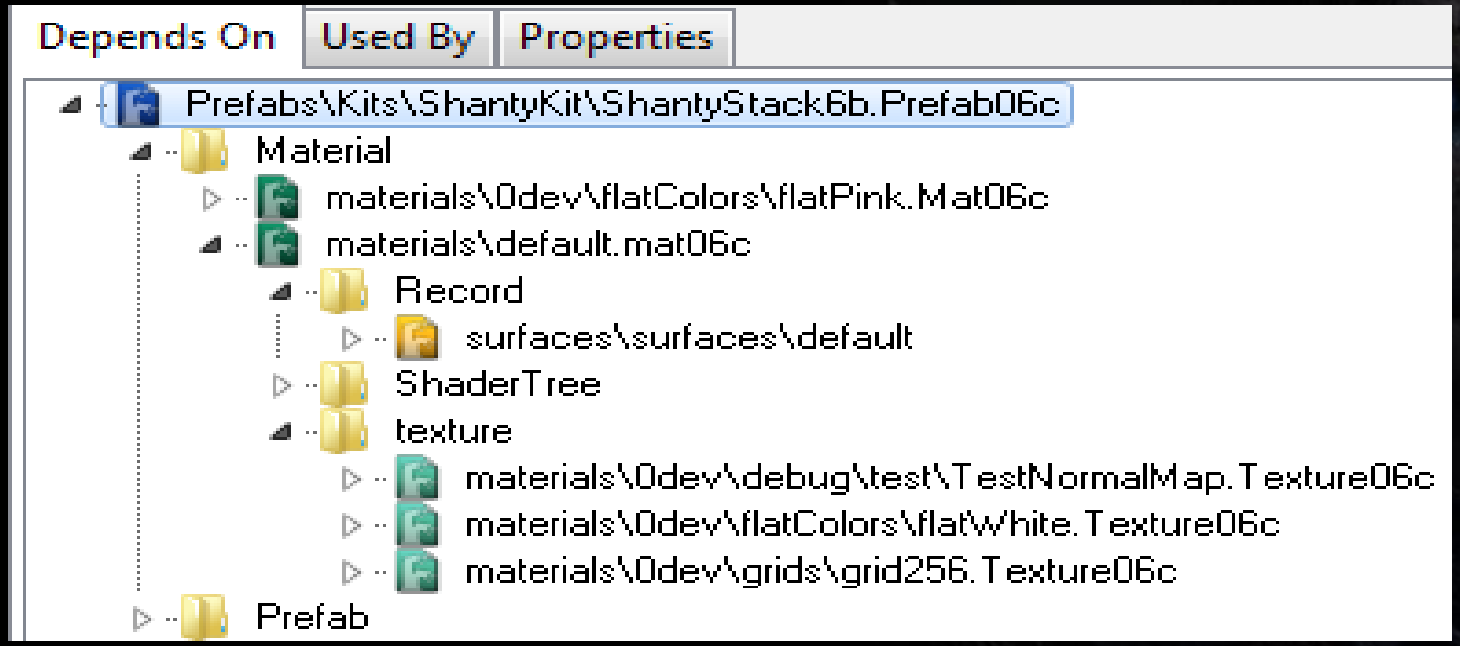
Ex. Instance Overrides

Enough Rope

- A deep prefab tree is intimidating, but is ideal for maximum re-use
- Tons can be updated with a single change

- Viewing asset relationships is essential
- Invest in visibility
  - Make it less complicated

Asset Dependencies



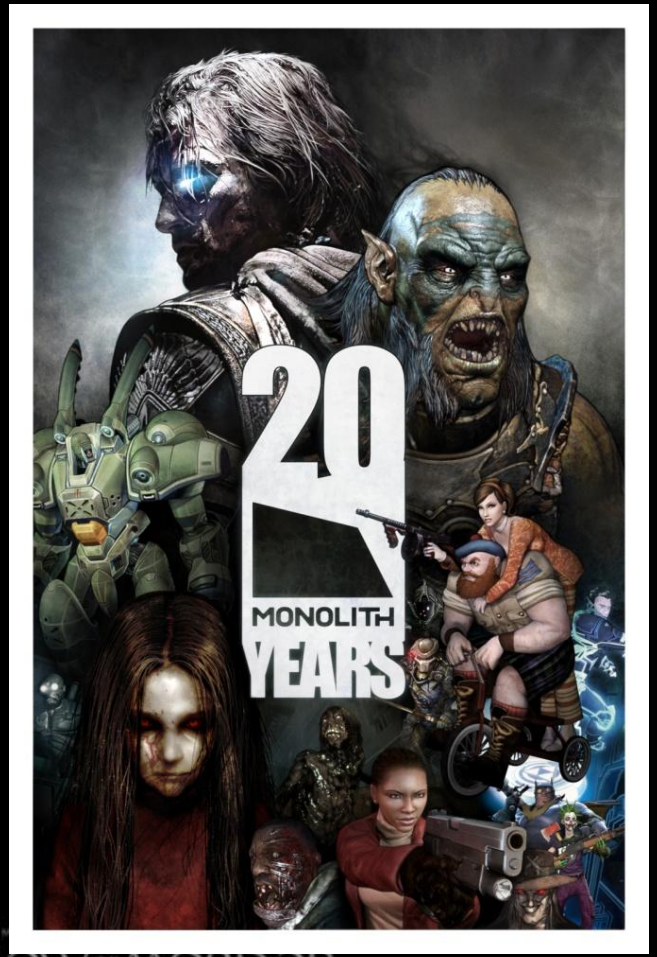
Hedge your bets, plan for issues

- Mistakes will be made
- Asset Validation
  - Pack time validation
  - Client asset validation pre-submission
  - Server asset validation post-submission

- Try to have things that detect problems for you

Recap

- Overhead = damage over time to your game
- Invest in productivity
  - Profile development pipeline
  - You may be surprised
  - Seek out low hanging fruit
  - Establish / expose asset relationships



MONOLITH IS HIRING!

www.lith.com/Jobs

SHADOW OF MORDOR

Q & A

- Doug Heimer – Lead Software Engineer
- doug.heimer@lith.com

- MONOLITH IS HIRING
  - www.lith.com/Jobs