DESTINY®

2

BUNGIE’S ASSET PIPELINE

‘DESTINY 2’ AND BEYOND

BRANDON MORO

Engineering Lead

OVERVIEW

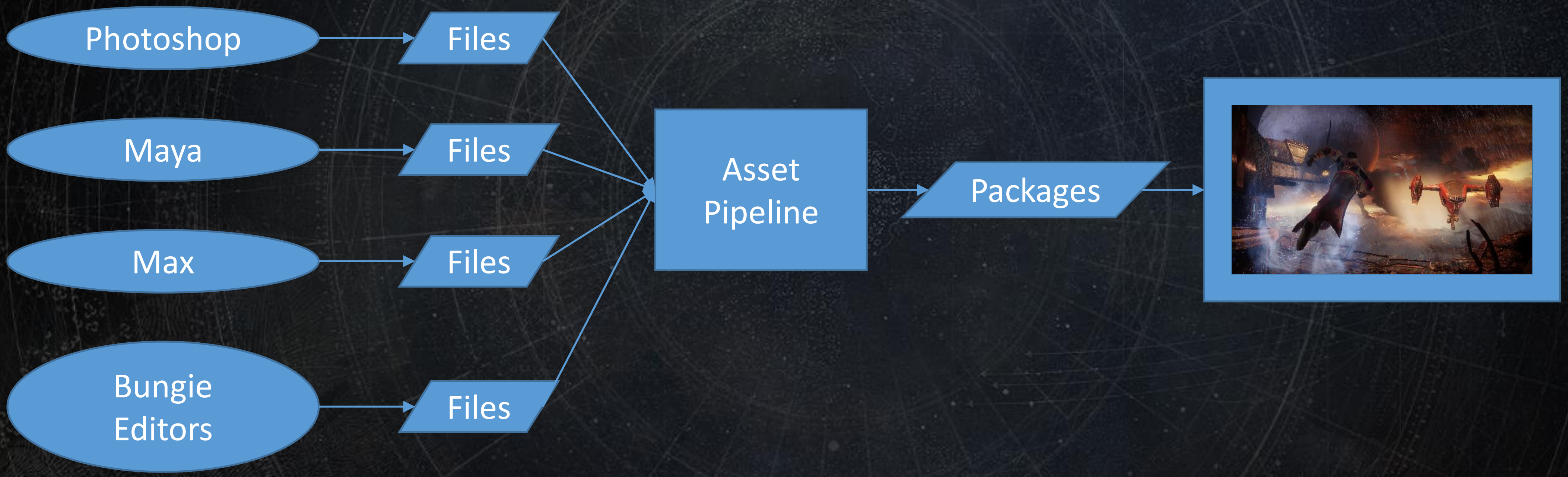
- Destiny Asset Pipeline
- Big Changes for Destiny 2
- Additional Iteration Improvements
- Conclusion

DESTINY

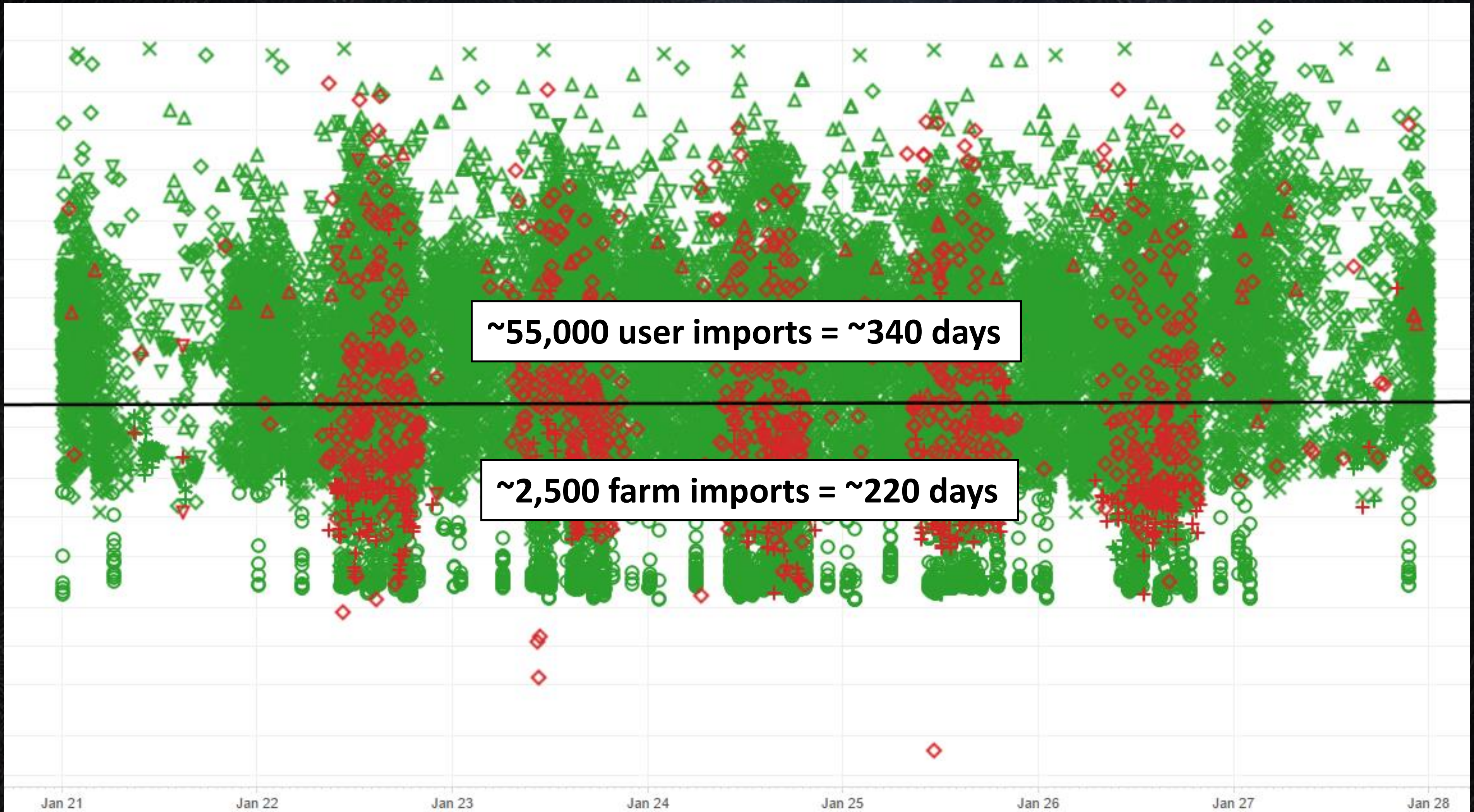
DESTINY

DESTINY ASSET PIPELINE

DESTINY ASSET PIPELINE



DESTINY ASSET PIPELINE

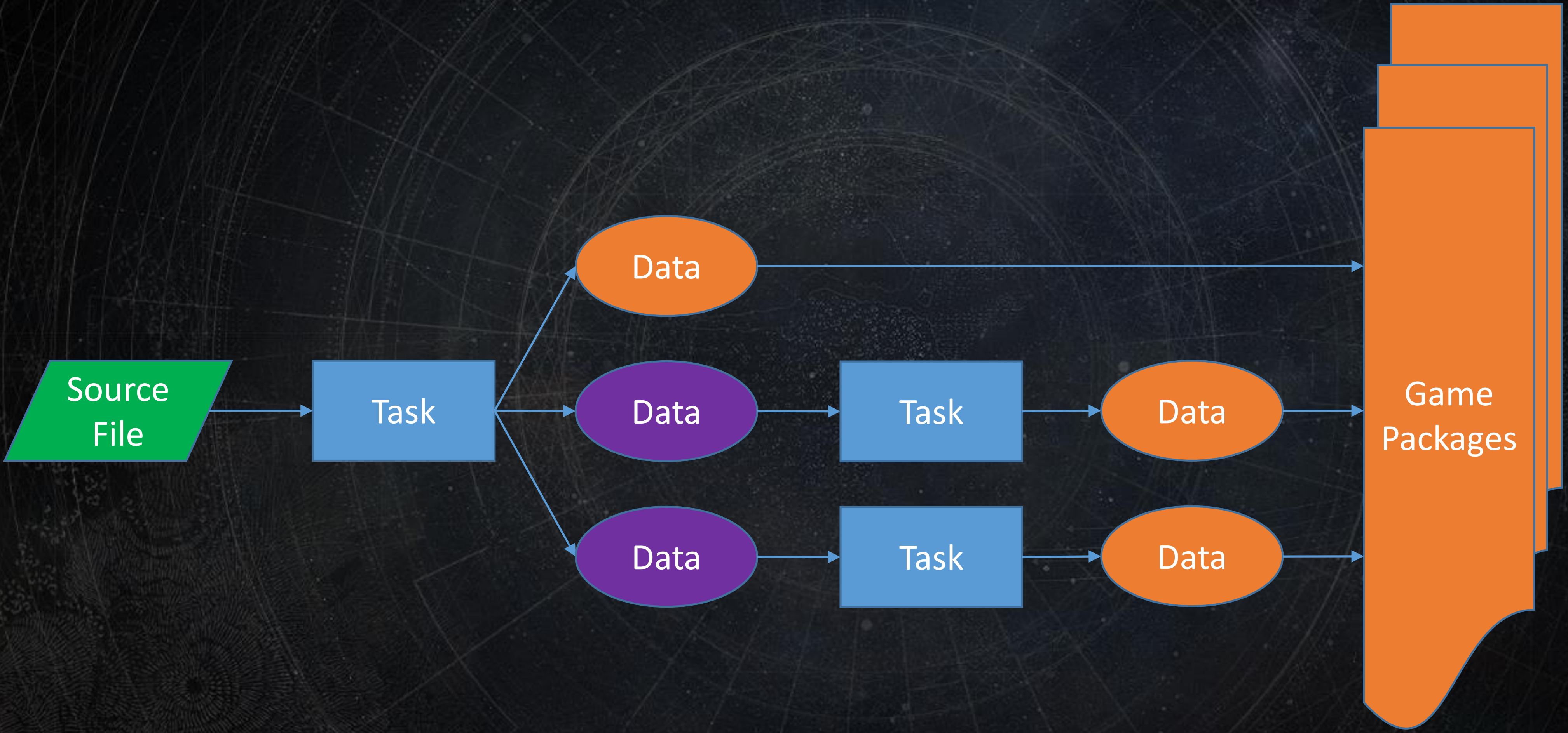


~55,000 user imports = ~340 days

~2,500 farm imports = ~220 days

THE DETAILS...

DEPENDENCY GRAPH



DEPENDENCY GRAPH

- Tasks (data transforms) are C++ functions
- Tasks read inputs, bind data to outputs
- Tasks can schedule additional tasks
  - Defines data dependencies between tasks

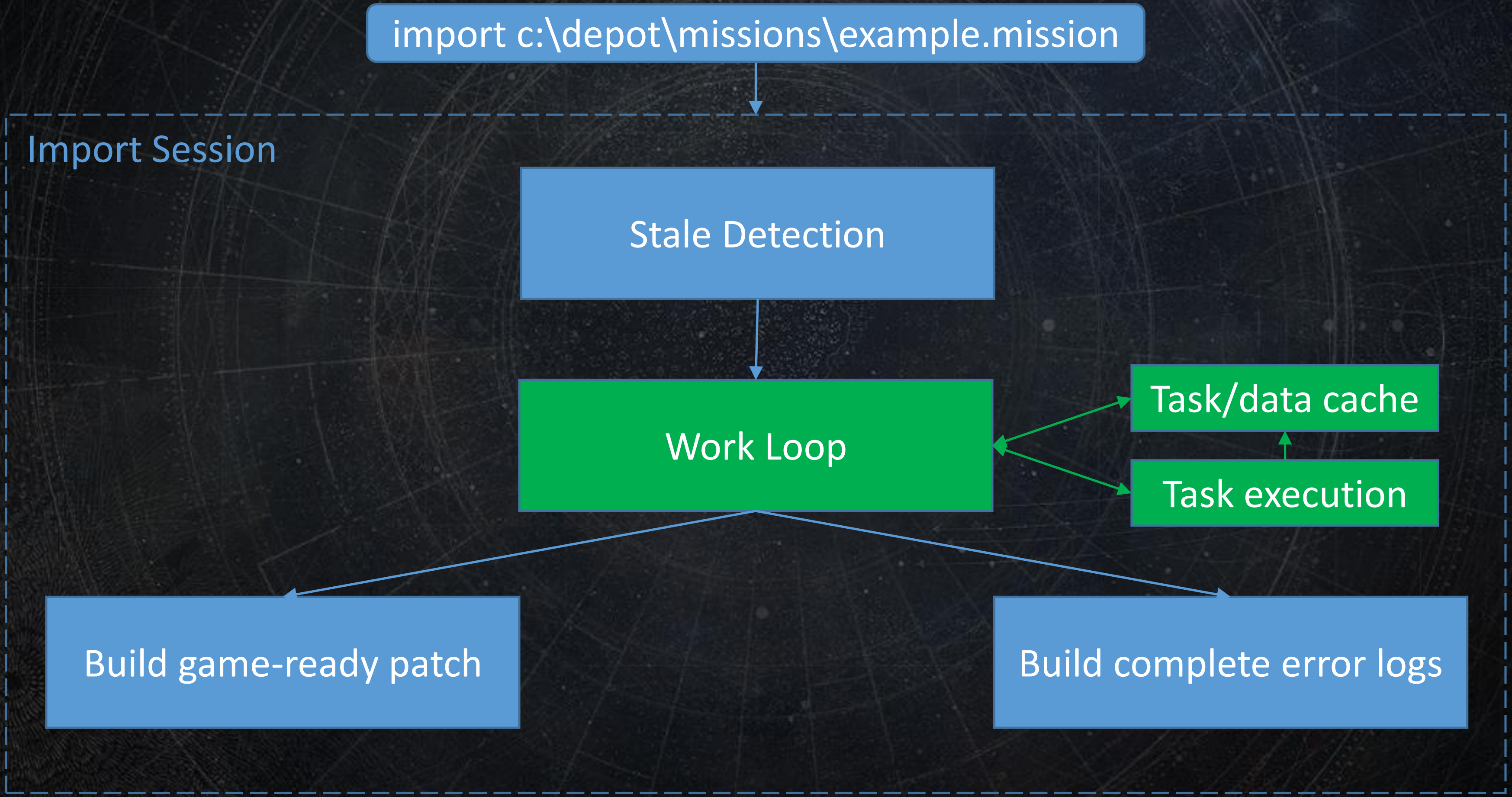
- Tasks are executed in parallel
- Task results are cached by hash of inputs
- System manages IO asynchronously

OUTPUTS

- Game data package files
  - Can be patches of previous packages
  - Game data is directly addressable by 32 bit id
    - Pro: No runtime fixup required
    - Con: All data that can reference each other must be processed together

- Content error inspection information

ORIGINAL DESTINY ASSET PIPELINE FLOW



CORE ISSUES

CORE ISSUES

Massive task and data counts

- Destiny 1 Earth – 9.1M tasks, 19.3M data elements
- Tracking structures are massive
- Difficult to inspect/understand
- Granularity stresses caching mechanisms

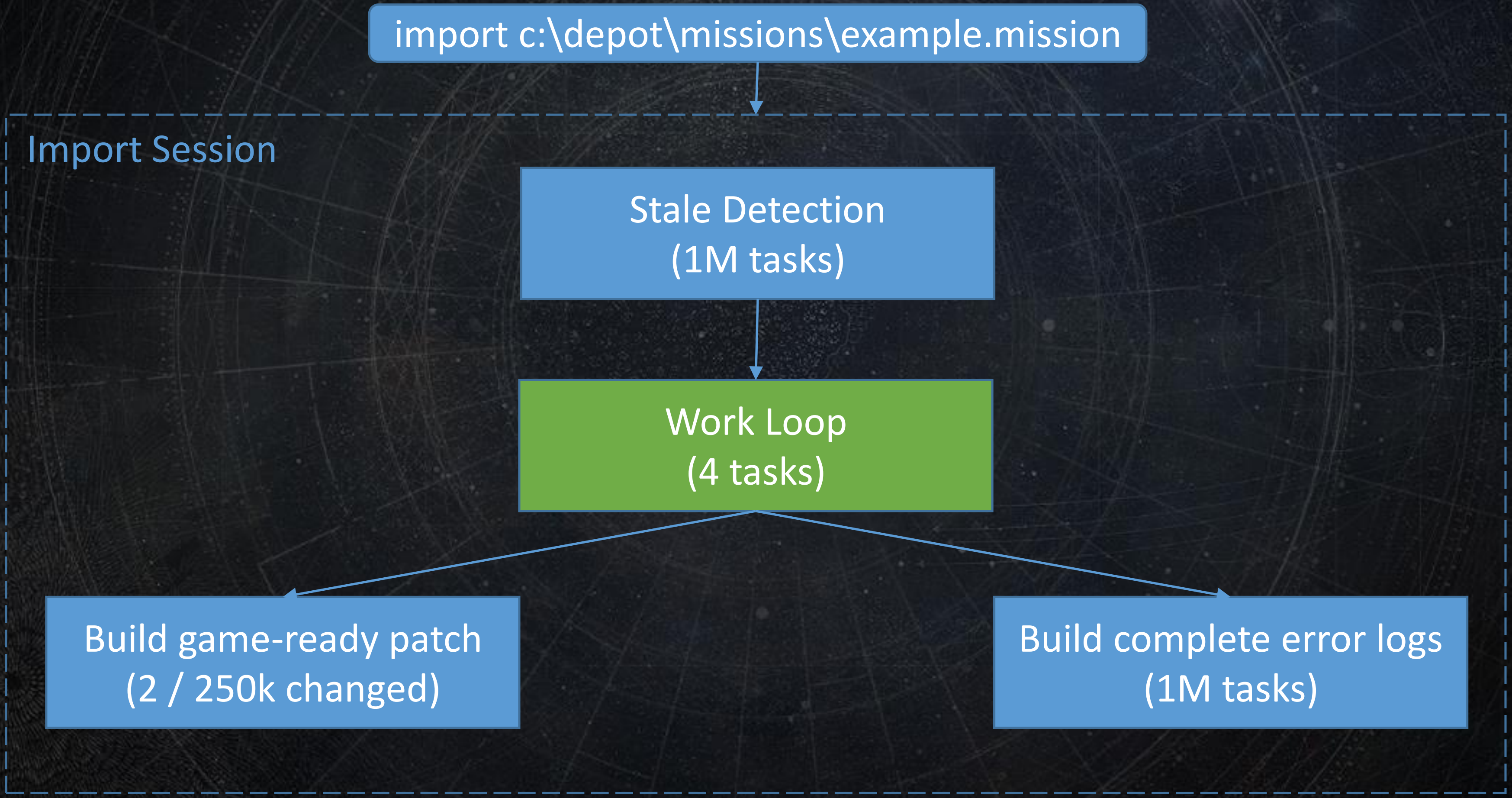
CORE ISSUES

Extremely over connected graphs

- Destiny 1 Earth, average piece of data:
  - Referenced by ~20 tasks (up to ~150k)
  - References ~10 other data (up to ~50k)

- High graph operation overhead
- Many tasks must run even for small source changes

CORE ISSUES - EXAMPLE

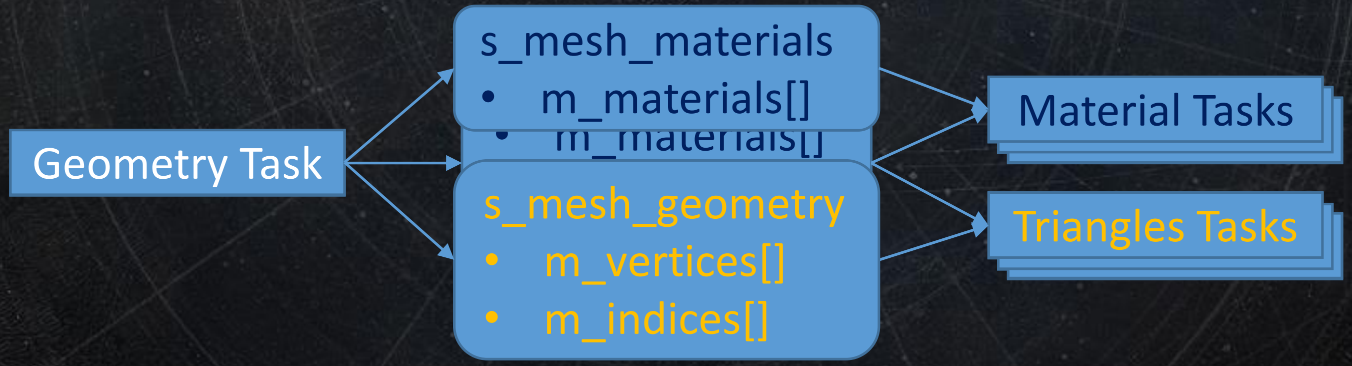

import c:\depot\missions\example.mission
Import Session
Stale Detection (1M tasks)
Work Loop (4 tasks)
Build game-ready patch (2 / 250k changed)
Build complete error logs (1M tasks)

HOW DID WE END UP HERE?

- System built in parallel with task code
- Scale ended up much larger than initially expected
  - In production, graph size and complexity approx. doubled every 2-3 weeks!

HOW DID WE END UP HERE?

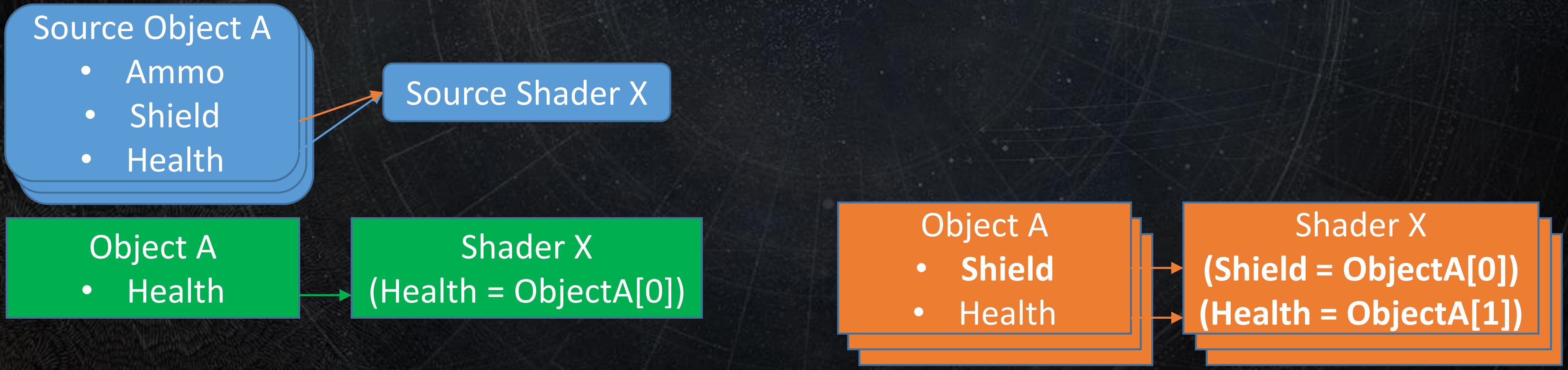
- Too easy to add data dependencies without understanding impact
- Data dependency granularity



HOW DID WE END UP HERE?

Strong focus on producing optimal game data

- Wide range of target hardware (PS3, PS4, Xbox 360, Xbox One)
- Focused on writing optimal engine code



Source Object A
- Ammo
- Shield
- Health

Source Shader X

Object A
- Health

Shader X
(Health = ObjectA[0])

Object A
- **Shield**
- Health

Shader X
(Shield = ObjectA[0])
(Health = ObjectA[1])

HOW DID WE END UP HERE?

For more details, check out:

"Asset Systems and Scalability" from HandmadeCon 2016 (on YouTube)

DESTINY 2

DESTINY 2

#1 Workflow Goal - Produce more content, much faster

- Now was our chance for a more drastic change
- But, we still would need
  - Backwards compatibility for source content
  - Minimal production wake

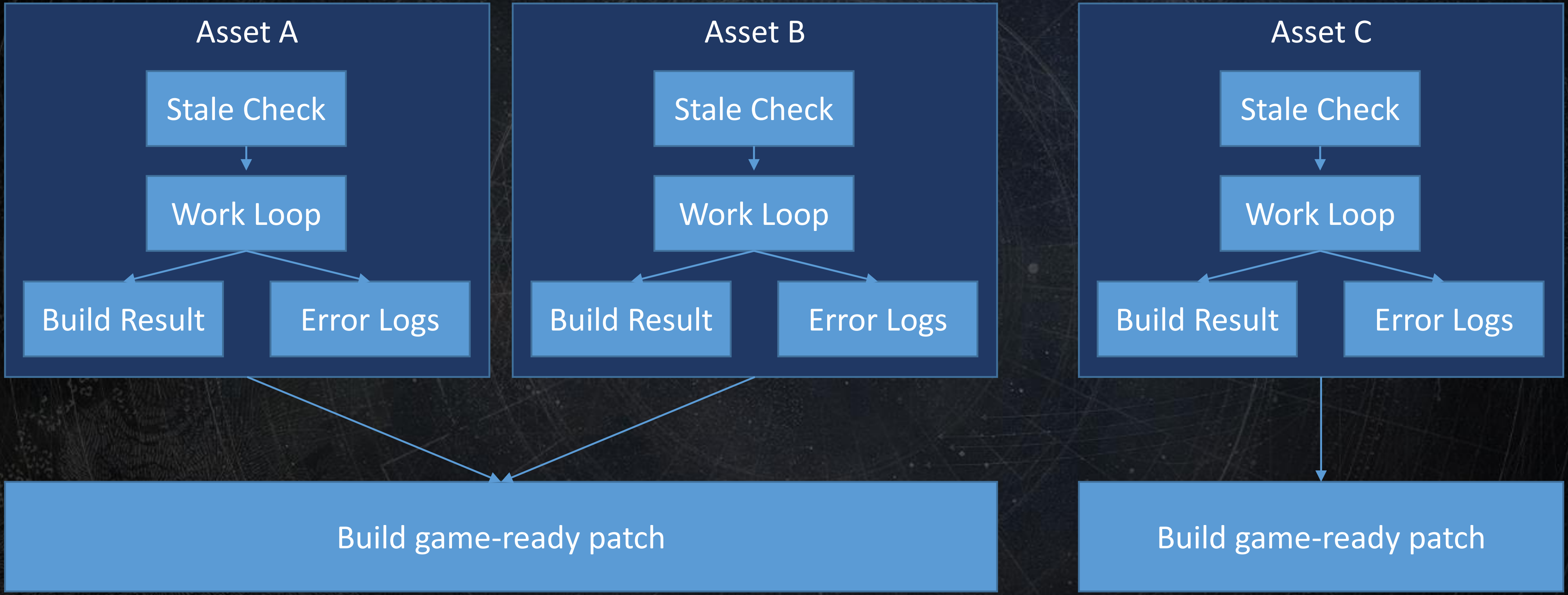
DESTINY 2 ASSET PIPELINE – BIG CHANGES

- Break up the graph into many smaller per-Asset graphs
    - Scope-of-context work is faster as task count drops
    - Run Asset graph operations in parallel
    - Easier to inspect and understand a smaller, single-Asset graph
    - More reasonable caching granularity

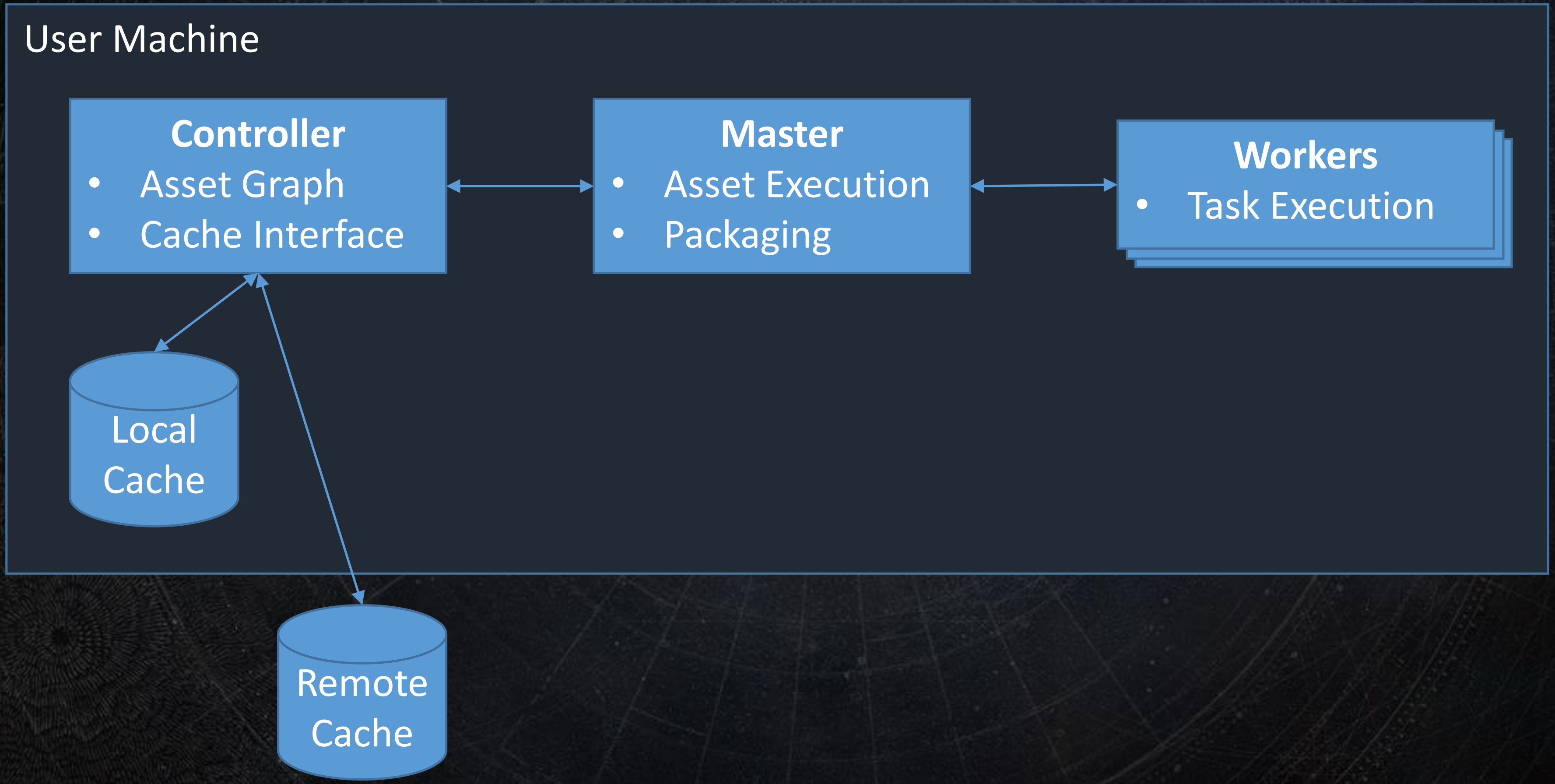
- Make it more difficult to introduce data dependencies between Assets and easier to understand workflow impact

- Allow Asset references to be fixed up at load time

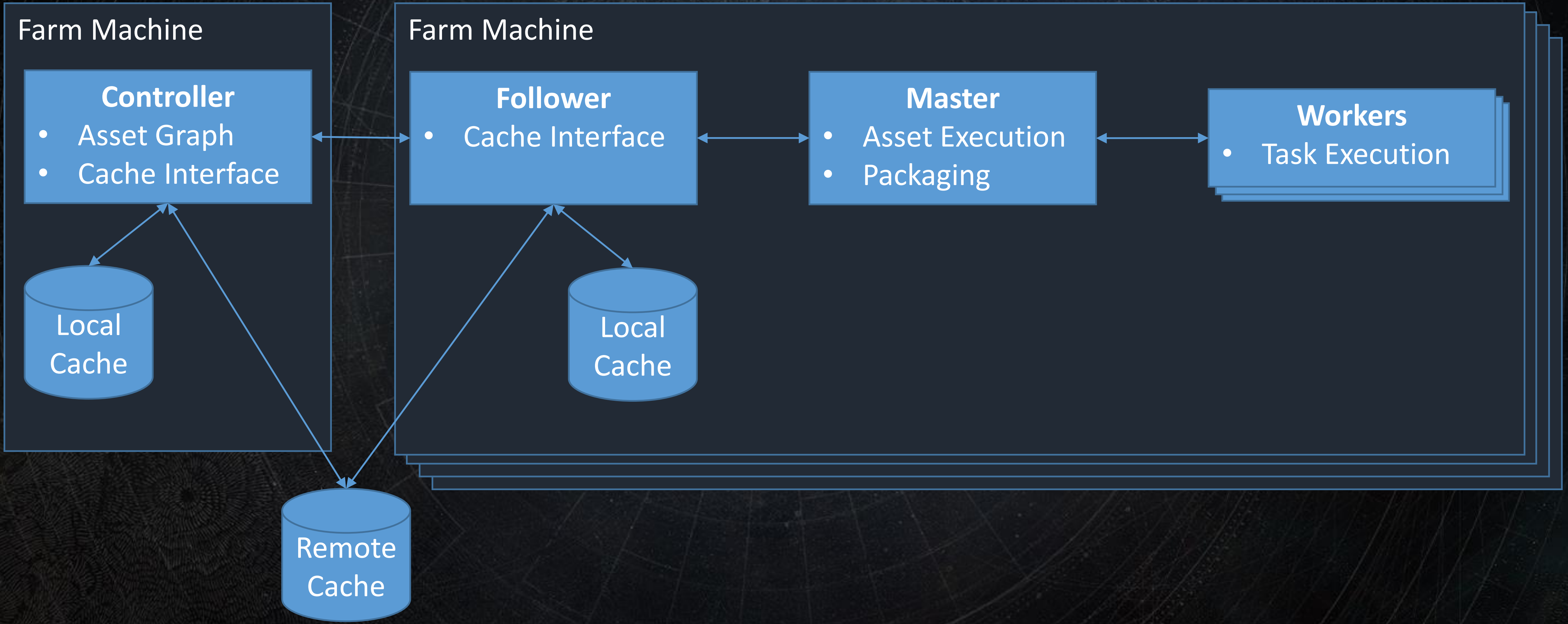
DESTINY 2 ASSET PIPELINE FLOW



DESTINY 2 ASSET PIPELINE - PARALLELISM



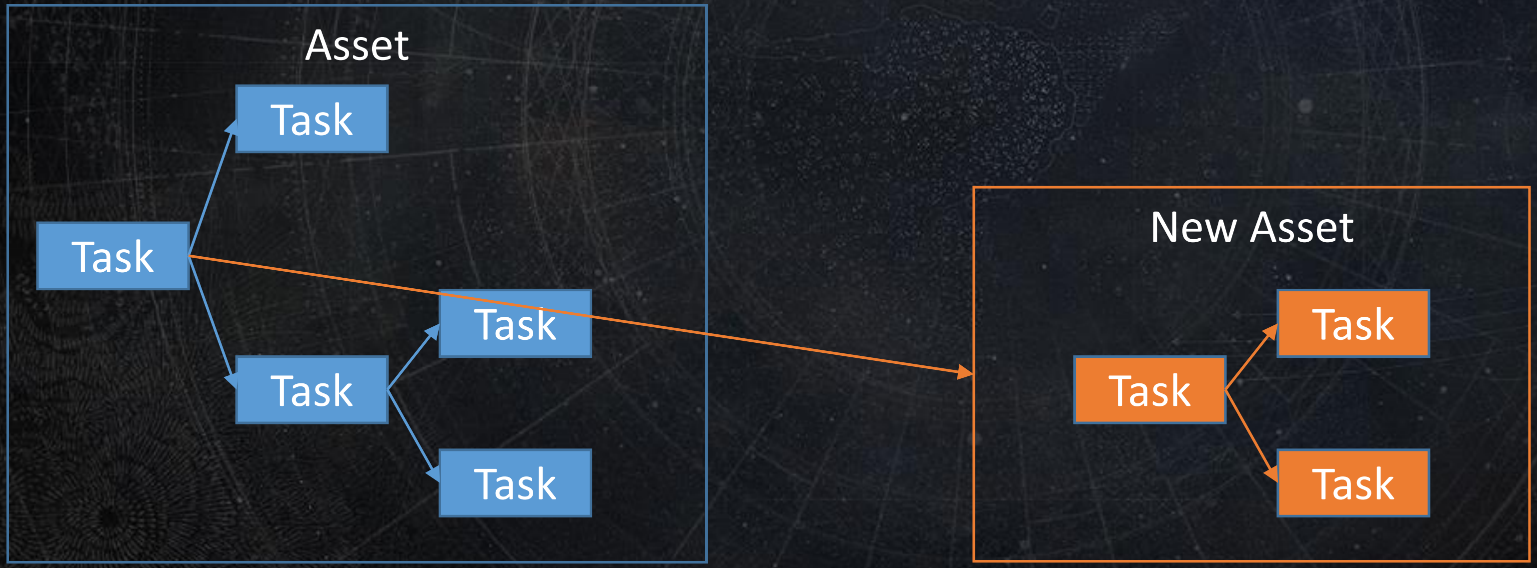
DESTINY 2 ASSET PIPELINE - PARALLELISM



WHAT WAS THE IMPACT?

WHAT WAS THE IMPACT?

Constraint: Backwards Compatibility / Minimal Production Wake



WHAT WAS THE IMPACT?

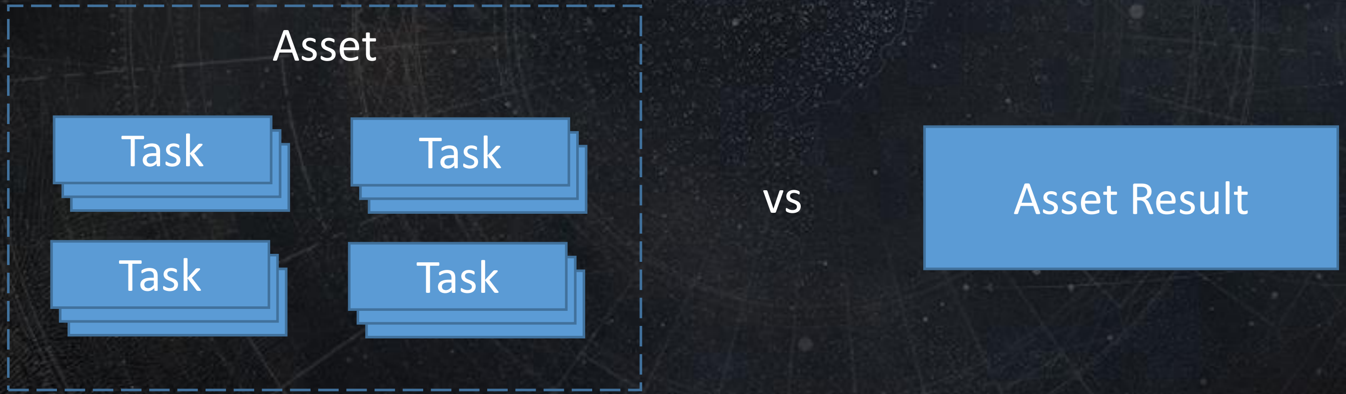
Issue: Difficult to understand / inspect

- Asset-level graph is much smaller, focused
- Build a report with details of all referenced Assets
  - Did it succeed?
  - Does it contain errors/warnings?
  - Was it re-used from a cache?
    - If not, why? “File rocket.definition changed”
- Create a understandable story of what occurred and why

WHAT WAS THE IMPACT?

Issue: Too granular to cache efficiently at massive scale

- Assets are more reasonable caching granularity than tasks



- Enabled always-on networked caches for everyone

WHAT WAS THE IMPACT?

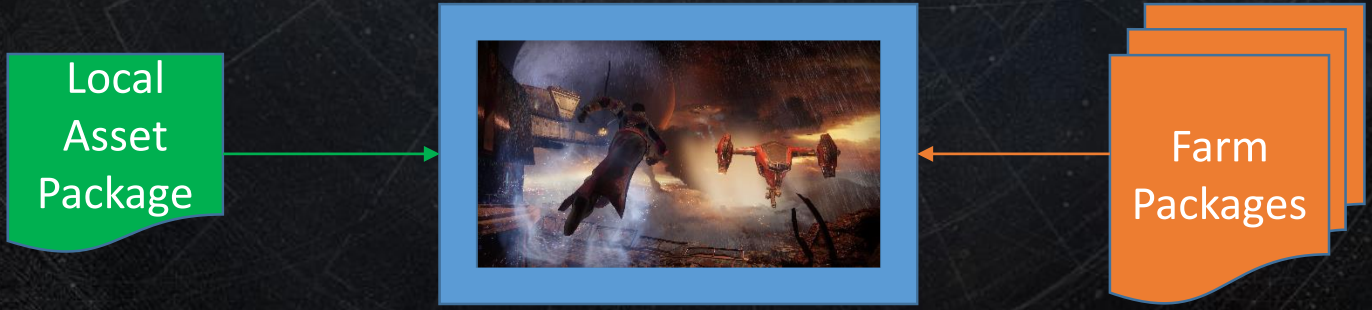
Issue: Over-connected graphs

- Asset-level dependencies are explicitly declared
- Easier to understand impact of Asset-level dependencies
- Asset caching reduces impact of overly connected task graphs

WHAT WAS THE IMPACT?

Issue: Must Import Full Context

- Load-time resolved Asset references
  - Import and package only the Asset you are editing
- Download packages from content build farm
  - See your changes in real shipping maps for ~free



BENDING RULES FOR ITERATION SPEED

BENDING RULES FOR ITERATION SPEED

- Typical dependency graph processing guarantees all inputs reflected in output
- Can speed up local iteration by bending this rule
- Asset boundaries enable interesting options for local iteration speed

BENDING RULES FOR ITERATION SPEED

“Free” Audio

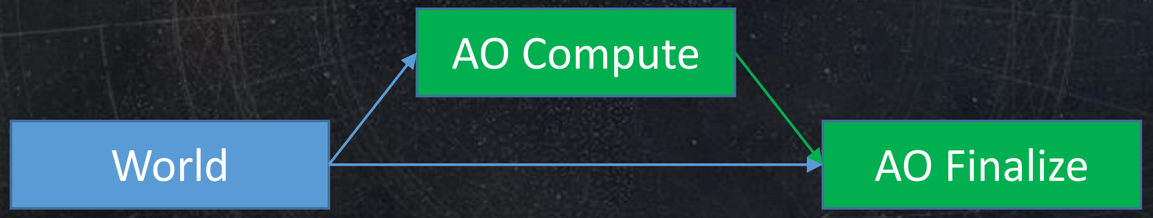
User option enables:

- Audio Asset cache compatibility checks ignore file references
- Still check major versions, etc
- No match = Fake “Missing” Asset result
  - Game already gracefully handles failed/missing audio

BENDING RULES FOR ITERATION SPEED

"Stale" Ambient Occlusion

- Split Ambient Occlusion into 2 Assets:



- 'AO Compute' can use modified Asset cache compatibility check

SUMMARY

SUMMARY

- Scaling issues with single, massive dependency graph
- Introduced Asset-level granularity
- Introduced Asset References with load-time fixup

SUMMARY

- These new tools enable building efficient workflows
- Continuing to work through and upgrade/optimize existing workflows
  - Remove/reduce costly data dependencies
  - Define new Assets
  - Adjust Asset boundaries

CONCLUSION

- Can combine major benefits of per-Asset and dependency graph based data pipelines
- Critical to understand impact before adding data dependencies

THANK YOU

bmoro@bungie.com

WE'RE HIRING

WWW.BUNGIE.NET/CAREERS

CAREERS@BUNGIE.COM