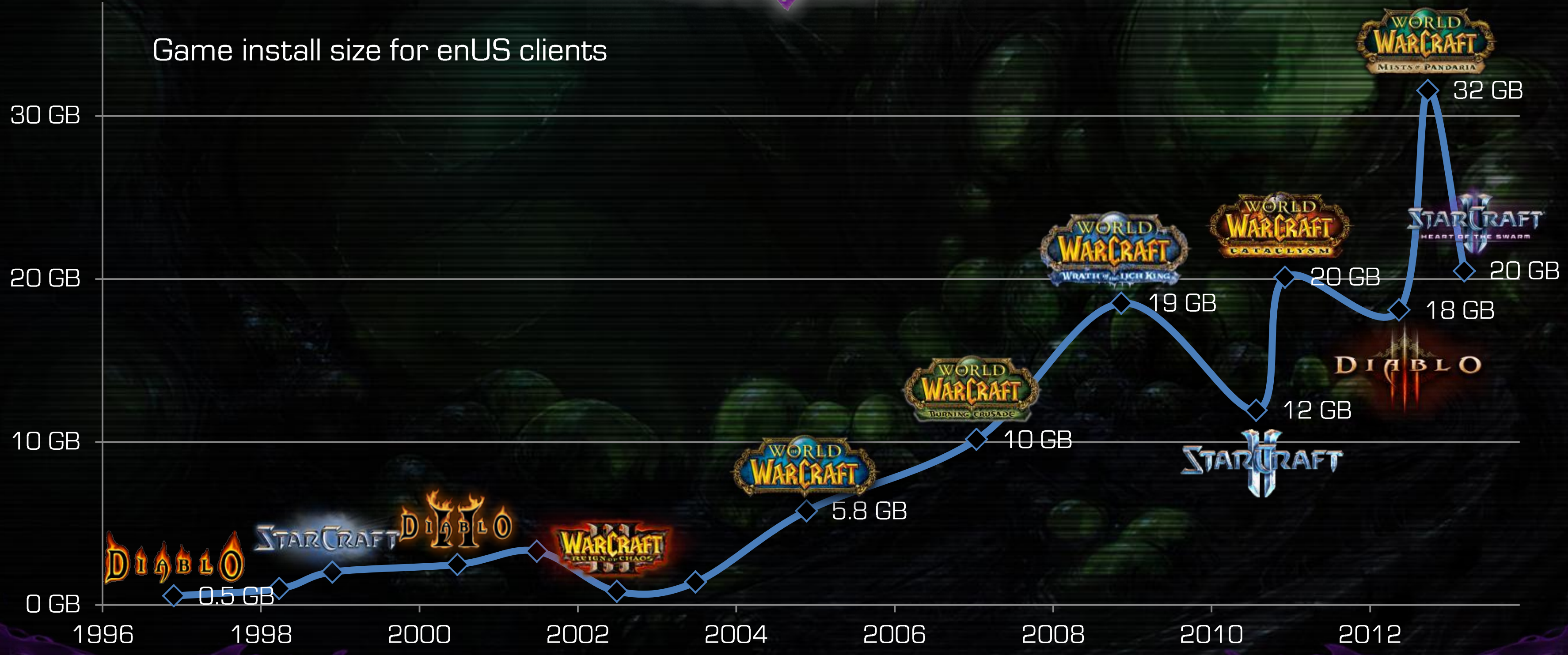
UNDER THE HOOD OF BLIZZARD'S INTERNAL BUILD SYSTEM

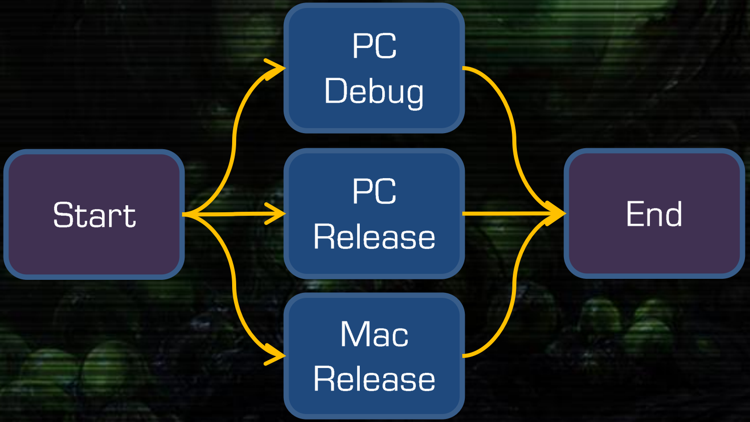
Blaine Whittle
bwhittle@blizzard.com

GAMES ARE GETTING BIGGER!

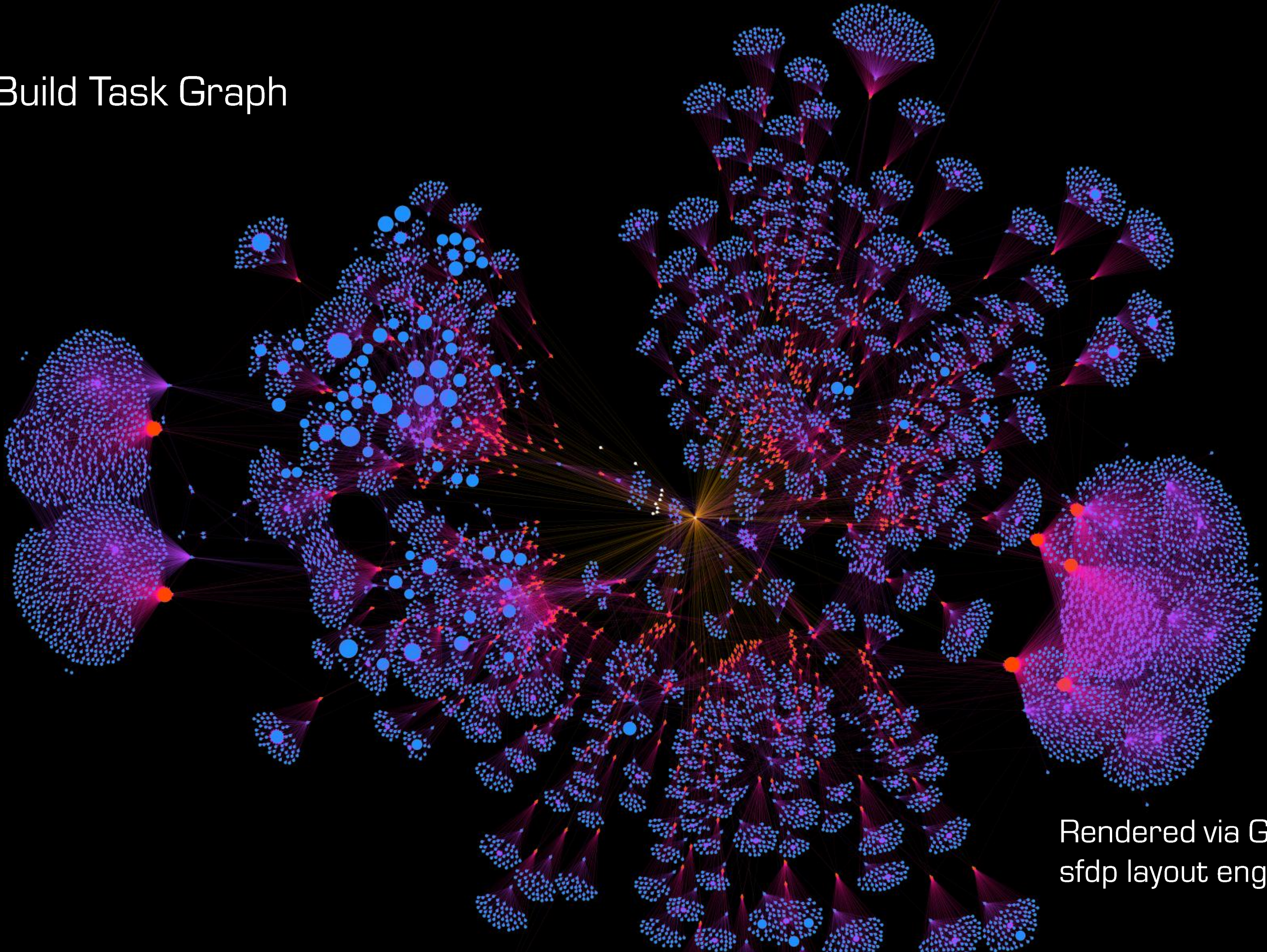


SMALL-SCALE DISTRIBUTED SYSTEMS

- 1-10 Jobs
- Top down scaling



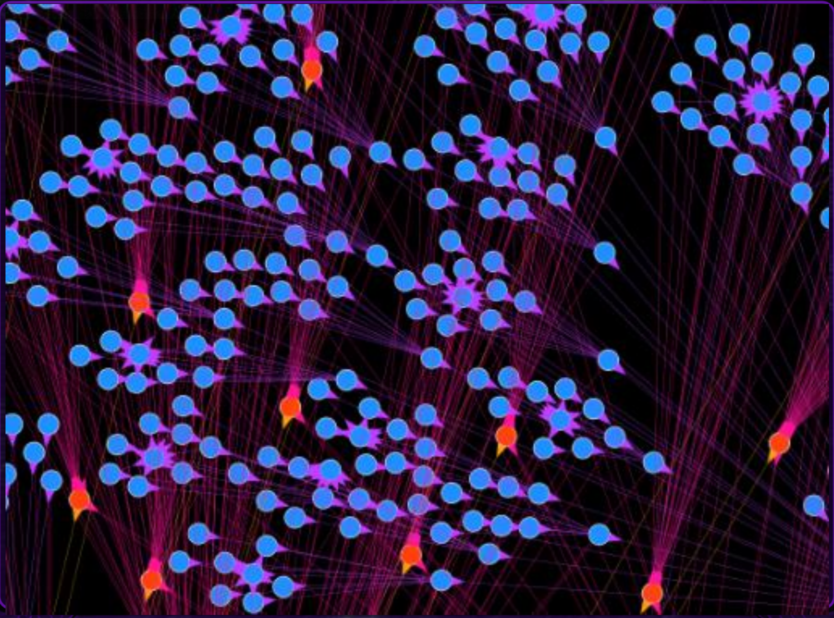
Code Build Task Graph



Rendered via Graphviz sfdp layout engine

LARGE-SCALE DISTRIBUTED SYSTEMS

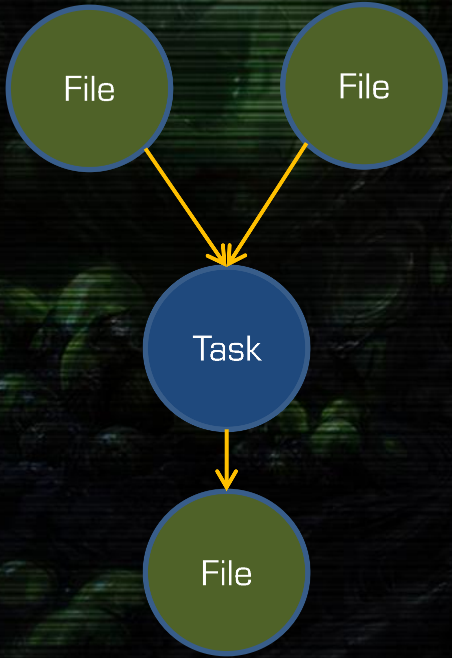
- 10,000 – 500,000 Tasks
- Bottom up scaling



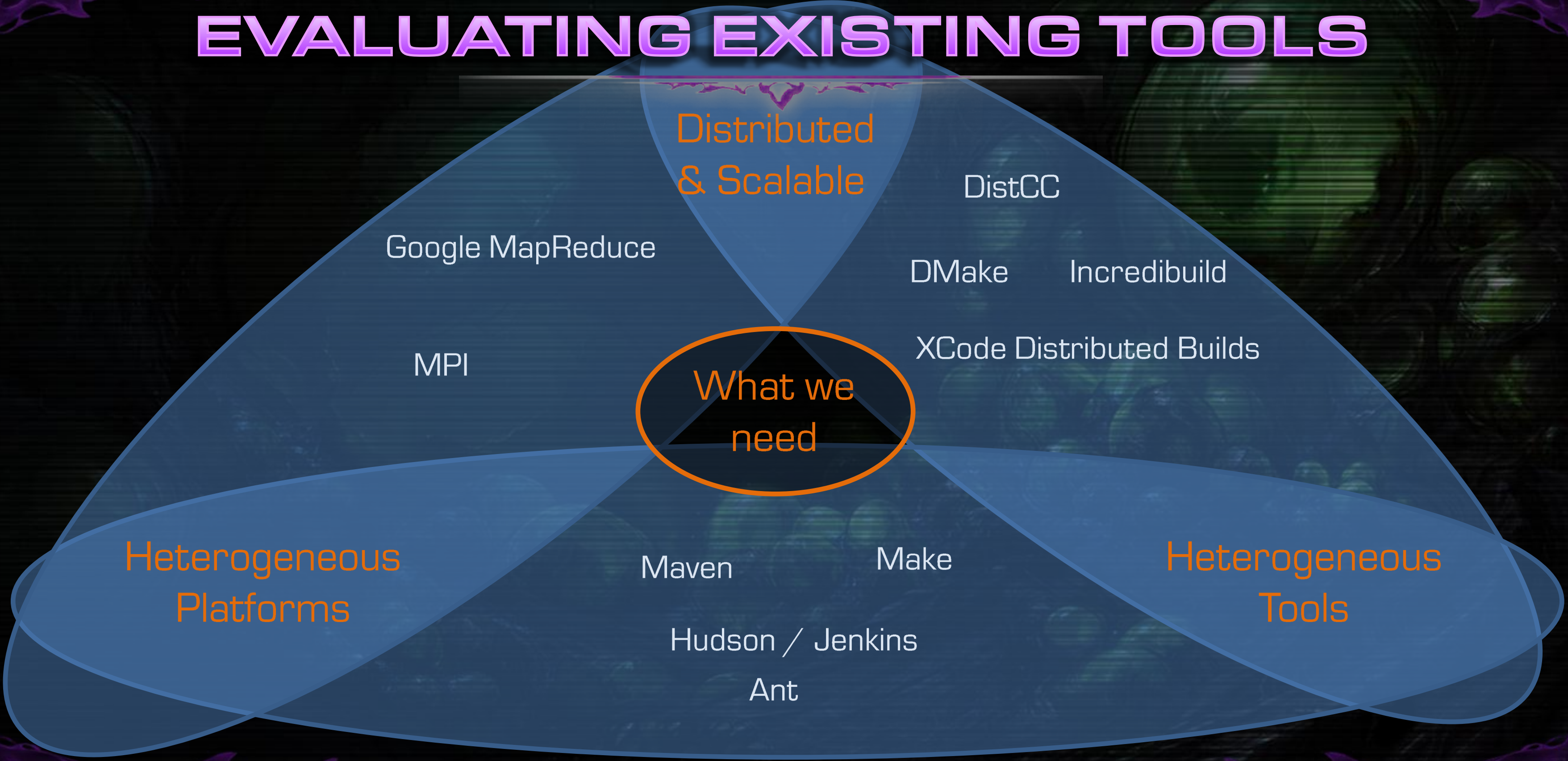
LARGE-SCALE DISTRIBUTED SYSTEMS

- Fine grained dependencies
  - Minimize IO

- Very lightweight tasks
  - High concurrency
  - Reattempt on error



EVALUATING EXISTING TOOLS



Distributed & Scalable

Heterogeneous Platforms

Heterogeneous Tools

What we need

Google MapReduce

MPI

DistCC

DMake

Incredibuild

XCode Distributed Builds

Maven

Make

Hudson / Jenkins

Ant

PROBLEM REQUIREMENTS

- Use graph analysis and dataflow paradigm for maximum parallelism
- Multiplatform (hosts and build targets)
- Incorporate existing processing tools into new framework
- Design for maximum execution performance

PROJECT “SANITY”

DATAFLOW

- A dataflow architecture is functional programming using nodes that are stream processors

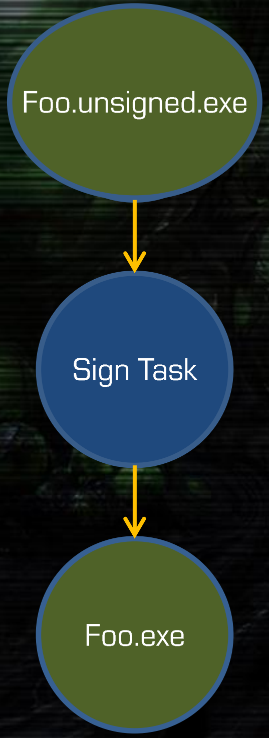
- In our dataflow pattern, the nodes are tasks that read immutable files and produce new files and new tasks

IMMUTABLE FILES

- During a Sanity build, file paths can only be written to once

- Tasks that normally modify a file in place become a copy on write

- Unlimited directories and filenames



FRAMEWORK

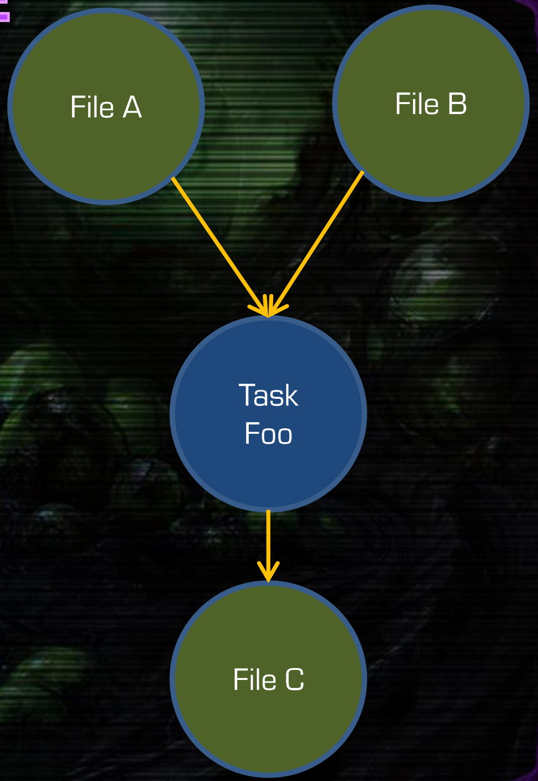
- Task = smallest unit of schedulable work
- Metatask = a task that may define additional tasks
  - Allows us to reason about the remaining work
  - If only tasks remain, we know our dependency graph is complete
- Rule = procedurally generates tasks based on file name patterns
  - Map Rule = 1 file → 1 task
  - Reduce Rule = N files → 1 task

TASK API

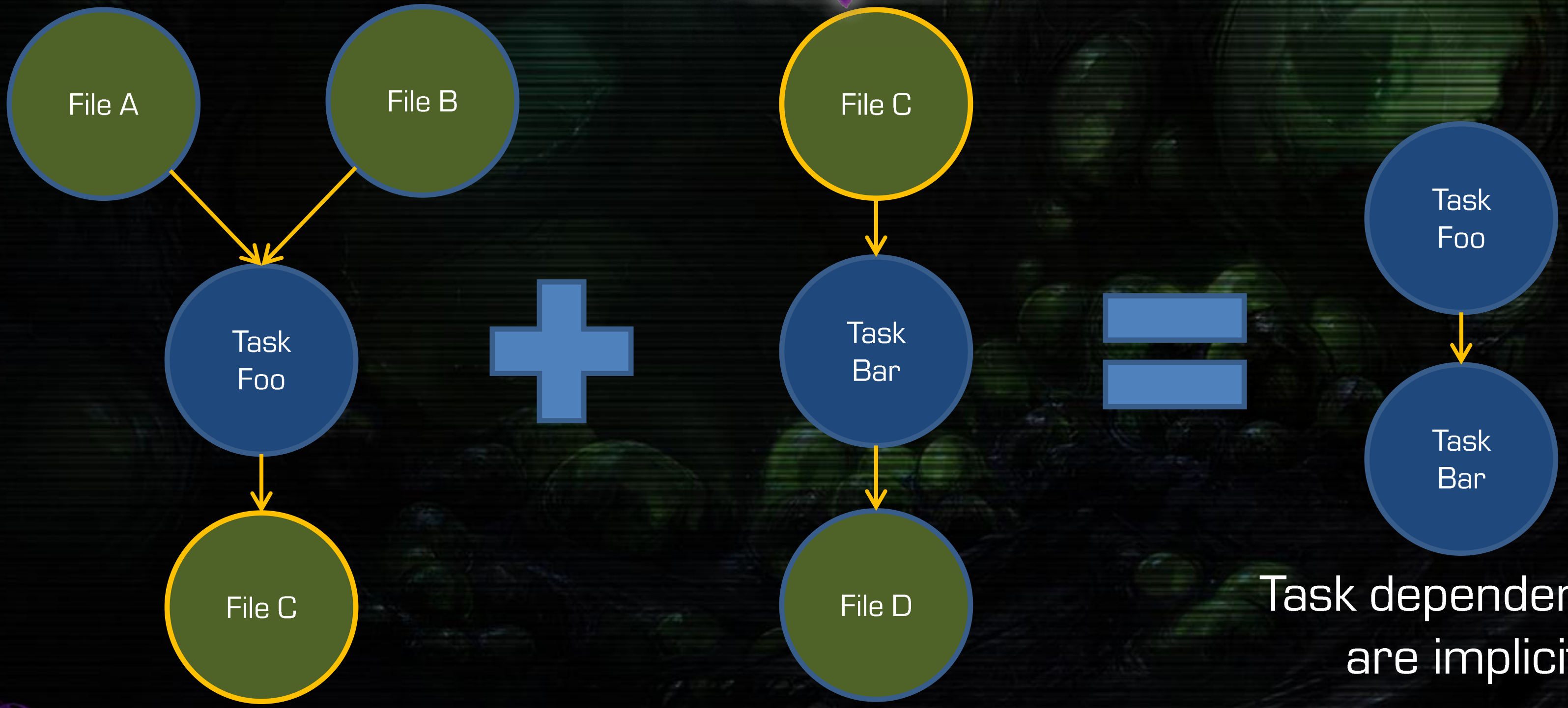
- Each task must implement two functions
  - Parse
  - Execute

TASK API - PARSE

- Parse
  - Takes an opaque data structure called a task line
  - Returns
    List of input files
    List of output files
  - Pure function / no side effects



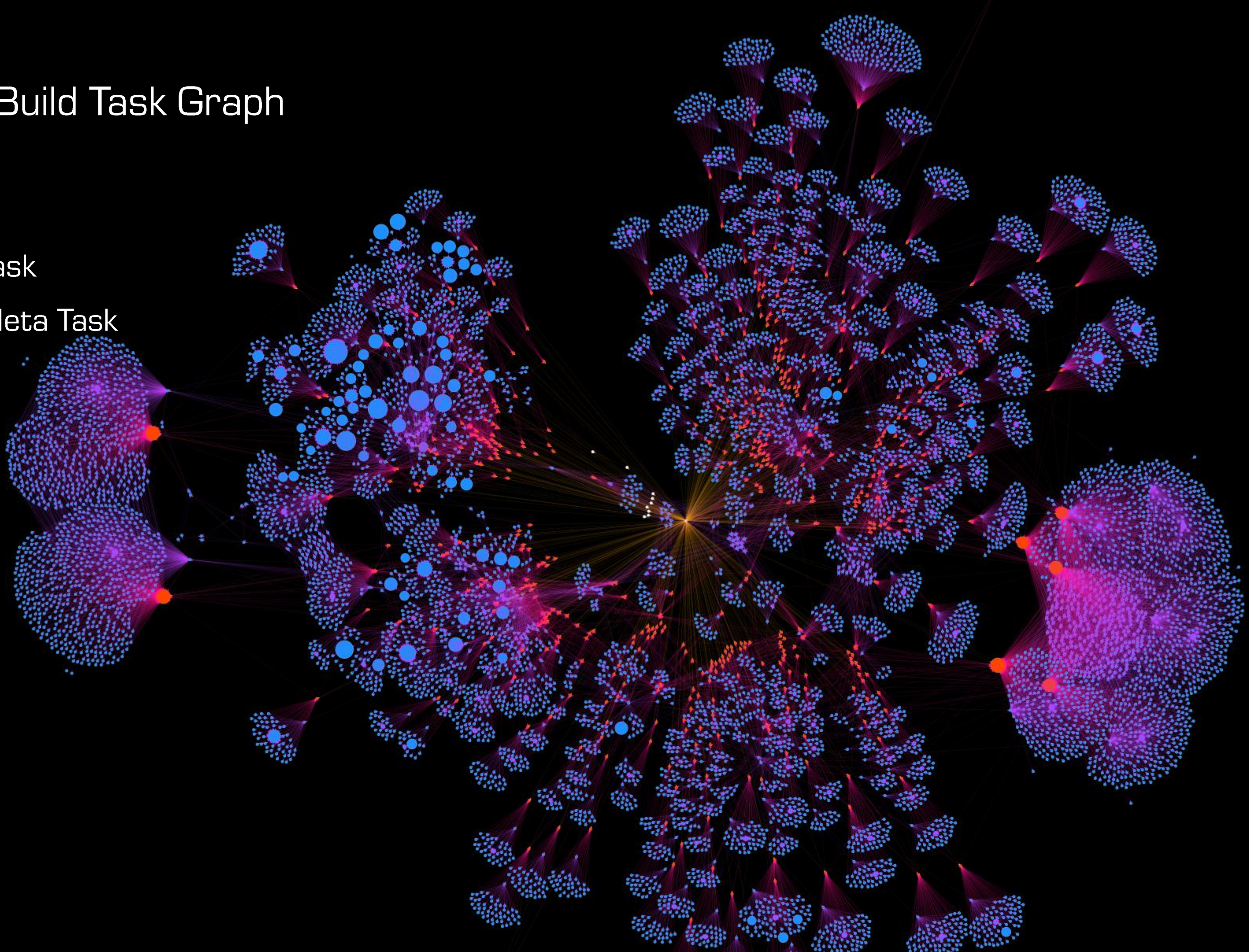
TASK API - PARSE



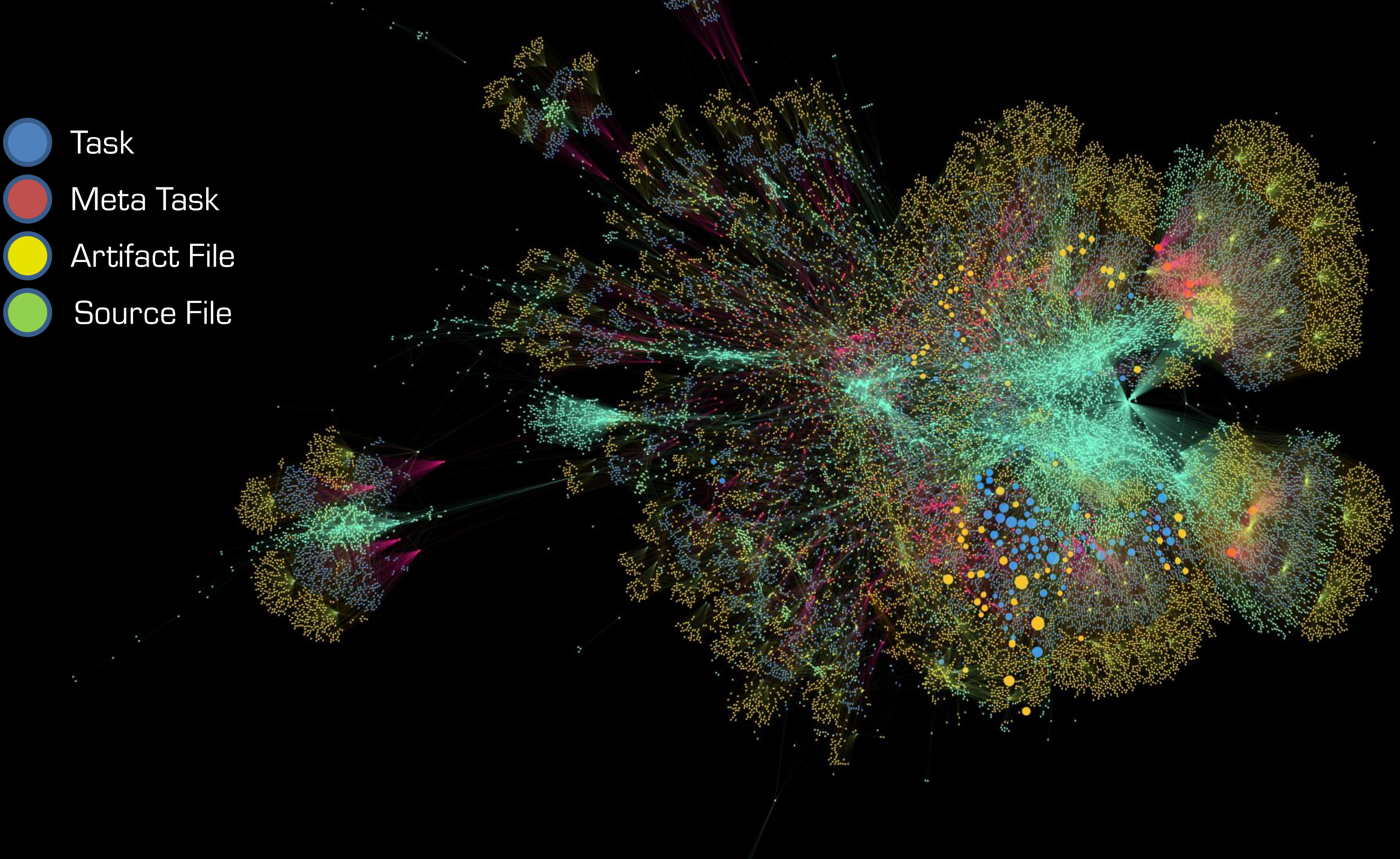
Task dependencies are implicit

Code Build Task Graph

- 🔵 Task
- 🔴 Meta Task



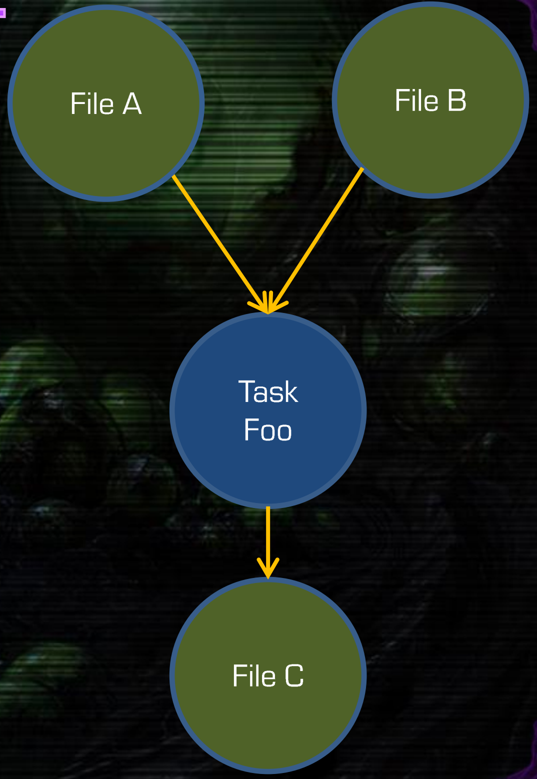
Code Build File & Task Graph



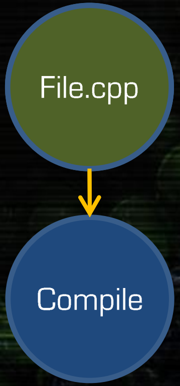
TASK API - PARSE

- Parse functions may not directly open or read files

- However Parse may return a set of closures for transforming file contents into a list of additional input file paths

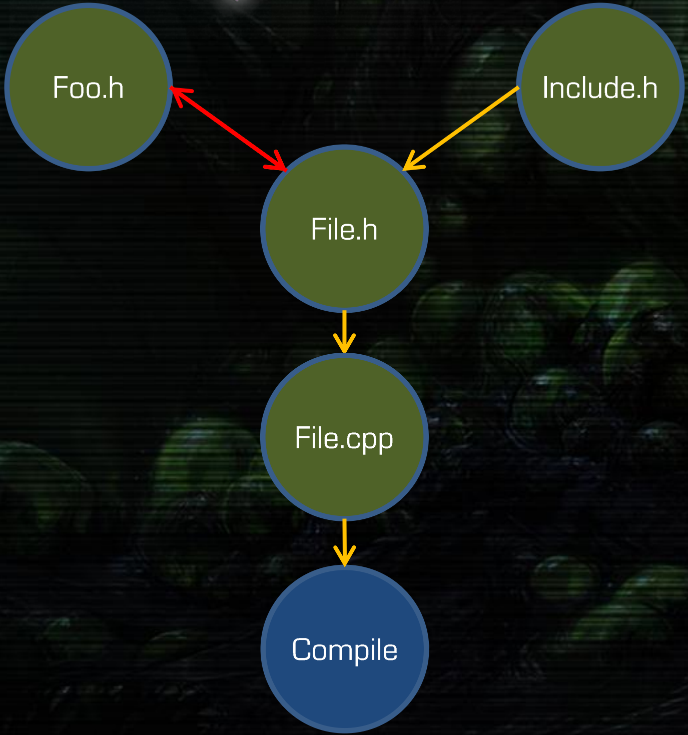


FILE FILE DEPENDENCIES



FILE FILE DEPENDENCIES

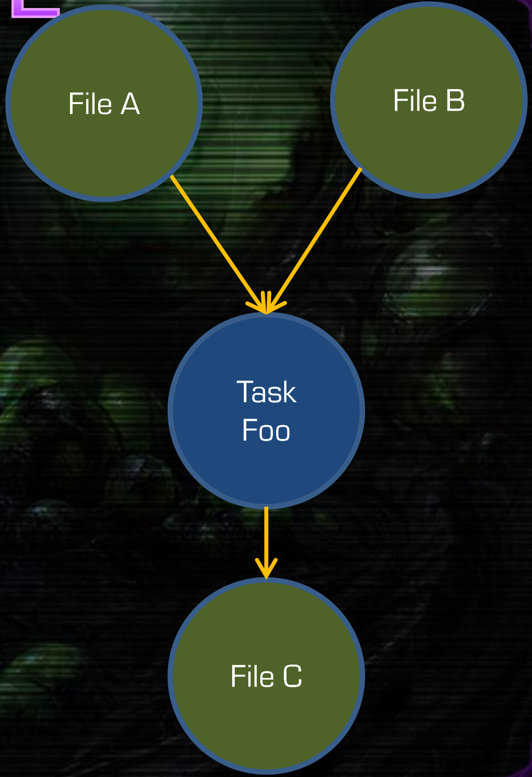
File dependencies can be cyclic



TASK API - EXECUTE

- Execute
  - Does the actual work, i.e. reads input files and creates output files
  - Returns either
    - Result (Success / Failure)
    - A new set of task lines (Metatasks)

All state is passed via task lines or files



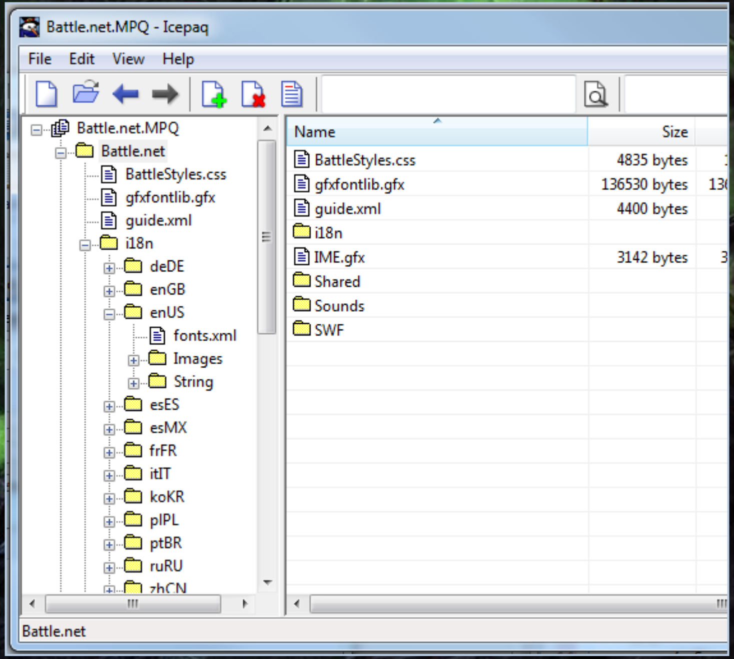
EXTENDED EXAMPLE

3 Metatasks

- 1 solution file
- 2 project files

26 Tasks

- 23 compile tasks
- 2 link tasks (1 lib)
- 1resource compile task



- Battle.net.MPQ
  - Battle.net
    - BattleStyles.css
    - gfxfontlib.gfx
    - guide.xml
    - i18n
      - deDE
      - enGB
      - enUS
        - fonts.xml
        - Images
        - String
      - esES
      - esMX
      - frFR
      - itIT
      - koKR
      - plPL
      - ptBR
      - ruRU
      - zhCN

Name	Size
BattleStyles.css	4835 bytes
gfxfontlib.gfx	136530 bytes
guide.xml	4400 bytes
i18n	
IME.gfx	3142 bytes
Shared	
Sounds	
SWF	

Battle.net

.SANITY FILE

```
[
    {tasks, [
        {vs_solution, "IcePaq/Icepaq.sln", "Release|Win32"}
    ]},
    {deliverables,[
        "IcePaq/win32_release/Icepaq.exe"
    ]},
    {deploy, [
        {location, {smb, "//someserver/someshare“}}
    ]},
    {vfs, [
        {"/",{rep, svn, "http://svn-repository/trunk/", head}}
    ]}
].
```

SOLUTION TASK PARSE

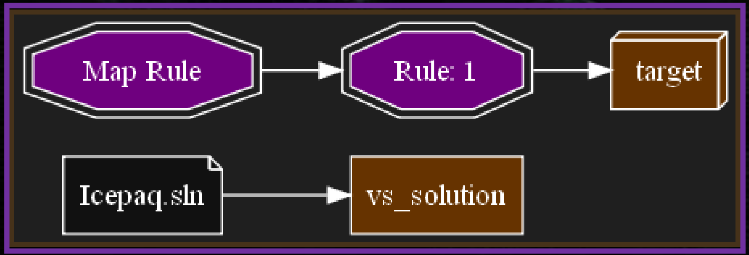
Task Line

```
{vs_solution,"IcePaq/Icepaq.sln","Release|Win32"}
```

Parse Result

Input Files	Icepaq.sln
Output Files	

Dependency Graph



🟤 Blocked Task or File

🟣 Evaluated Task or File

⚫ Unevaluated File

SOLUTION TASK EXECUTE

Task Line

```
{vs_solution,"IcePaq/Icepaq.sln","Release|Win32"}
```

Execute Result

```
{vc_project,["IcePaq/ConsoleMopaq.vcproj"],
            "IcePaq/Release/Mopaq.lib",
            "Release|Win32",
            [{solution_dir,"IcePaq/"}]}


{vc_project,[“IcePaq/Icepaq.vcproj"],
            "IcePaq/Release/Icepaq.exe",
            "Release|Win32",
            [{solution_dir,"IcePaq/"}]}
```

PROJECT TASK PARSE

Task Line

```
{vc_project,[“IcePaq/Icepaq.vcproj"],
            "IcePaq/Release/Icepaq.exe",
            "Release|Win32",
            [{solution_dir,"IcePaq/"}]}
```

Parse Result

Input Files	Icepaq.vcproj
Output Files	Icepaq.exe

PROJECT TASK PARSE

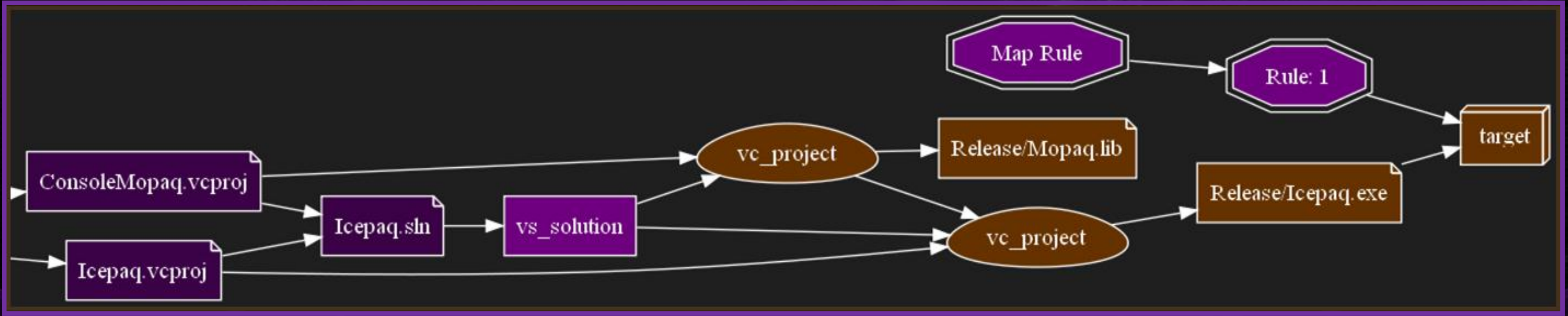
Task Line

```
{vc_project,["IcePaq/ConsoleMopaq.vcproj"],
            "IcePaq/Release/Mopaq.lib",
            "Release|Win32",
            [{solution_dir,"IcePaq/"}]}
```

Parse Result

Input Files	ConsoleMopaq.vcproj
Output Files	Mopaq.lib

Dependency Graph



PROJECT TASK EXECUTE

Task Line

```
{vc_project,["IcePaq/ConsoleMopaq.vcproj"],
            "IcePaq/Release/Mopaq.lib",
            "Release|Win32",
            [{solution_dir,"IcePaq/"}]}
```

Execute Result

```
{vc8_compile,"Contrib/Zlib/Contrib_zlib.c",
             ["IcePaq/Release/Contrib_zlib.obj"],
             [{vcproj,"IcePaq/ConsoleMopaq.vcproj"},
              {search_paths, "Contrib/Zlib",
                             "Tools/Mopaq/IcePaq”,
                             "BlizzardCore/Include",
                             "BlizzardCore/Source/Packages”,
                             "BlizzardCore/Source/Packages/Mopaq”,
                             "Shared","Contrib"]},
              {platform,"Win32"},
              {workdir, "IcePaq"}]}, …
```

COMPILE TASK PARSE

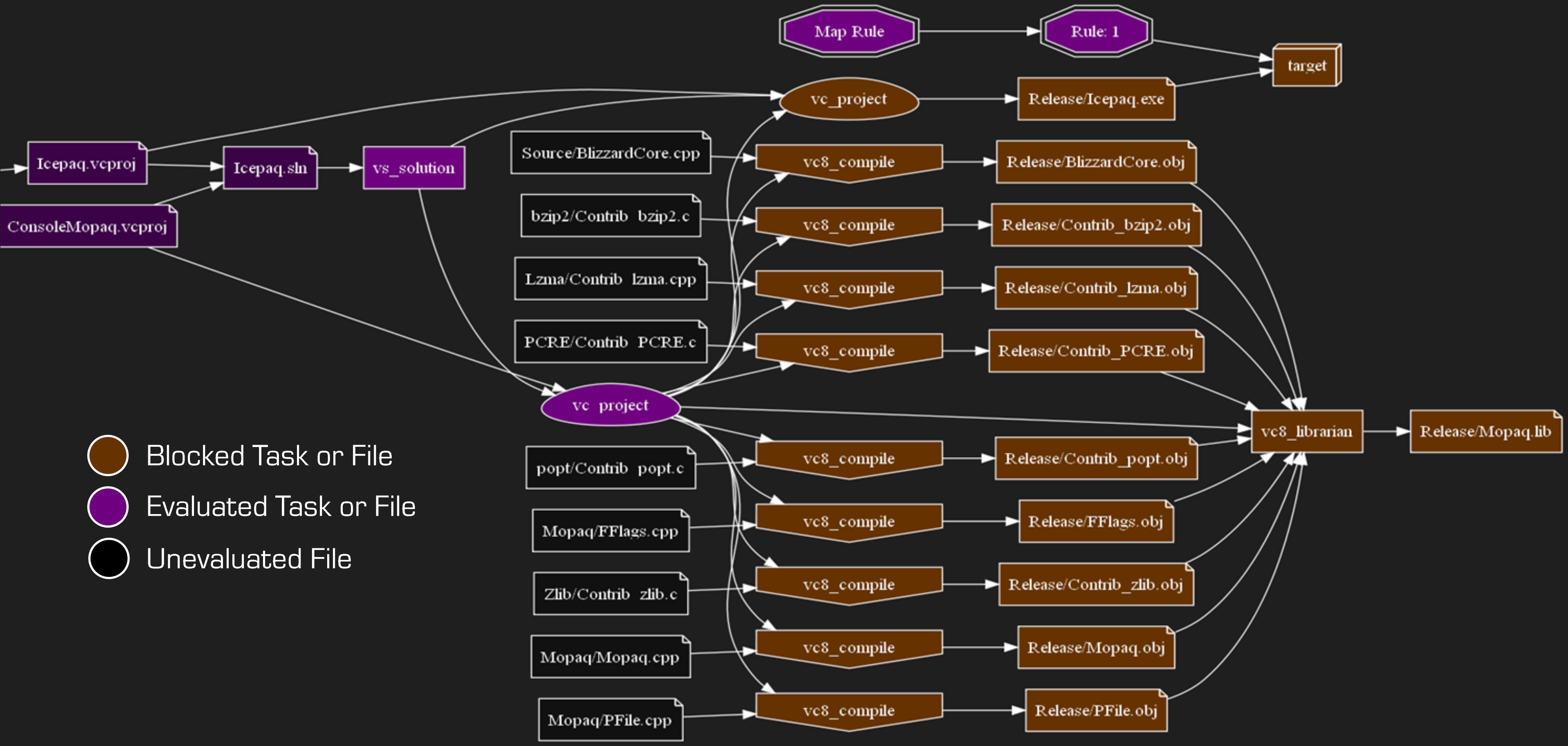
Task Line

```
{vc8_compile,"Contrib/Zlib/Contrib_zlib.c",
             ["IcePaq/Release/Contrib_zlib.obj"],
             [{vcproj,"IcePaq/ConsoleMopaq.vcproj"},
              {search_paths, "Contrib/Zlib",
                             "Tools/Mopaq/IcePaq”,
                             "BlizzardCore/Include",
                             "BlizzardCore/Source/Packages”,
                             "BlizzardCore/Source/Packages/Mopaq”,
                             "Shared","Contrib"]},
              {platform,"Win32"},
              {workdir, "IcePaq"}]},
```

Parse Result

Input Files	Contrib_zlib.c
Output Files	Contrib_zlib.obj

Repeated for other tasks in this project



● Blocked Task or File

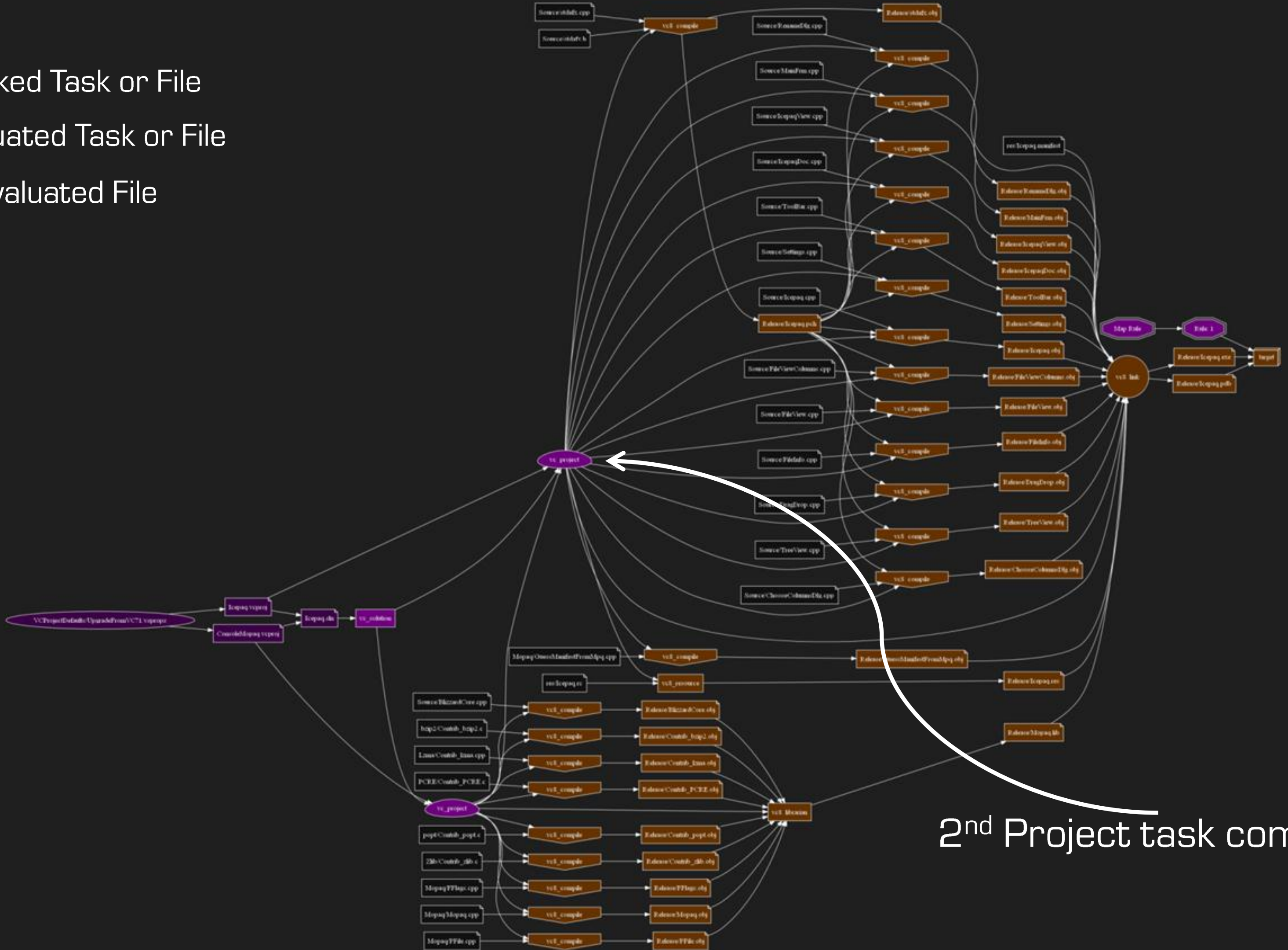
● Evaluated Task or File

● Unevaluated File

● Blocked Task or File

● Evaluated Task or File

● Unevaluated File

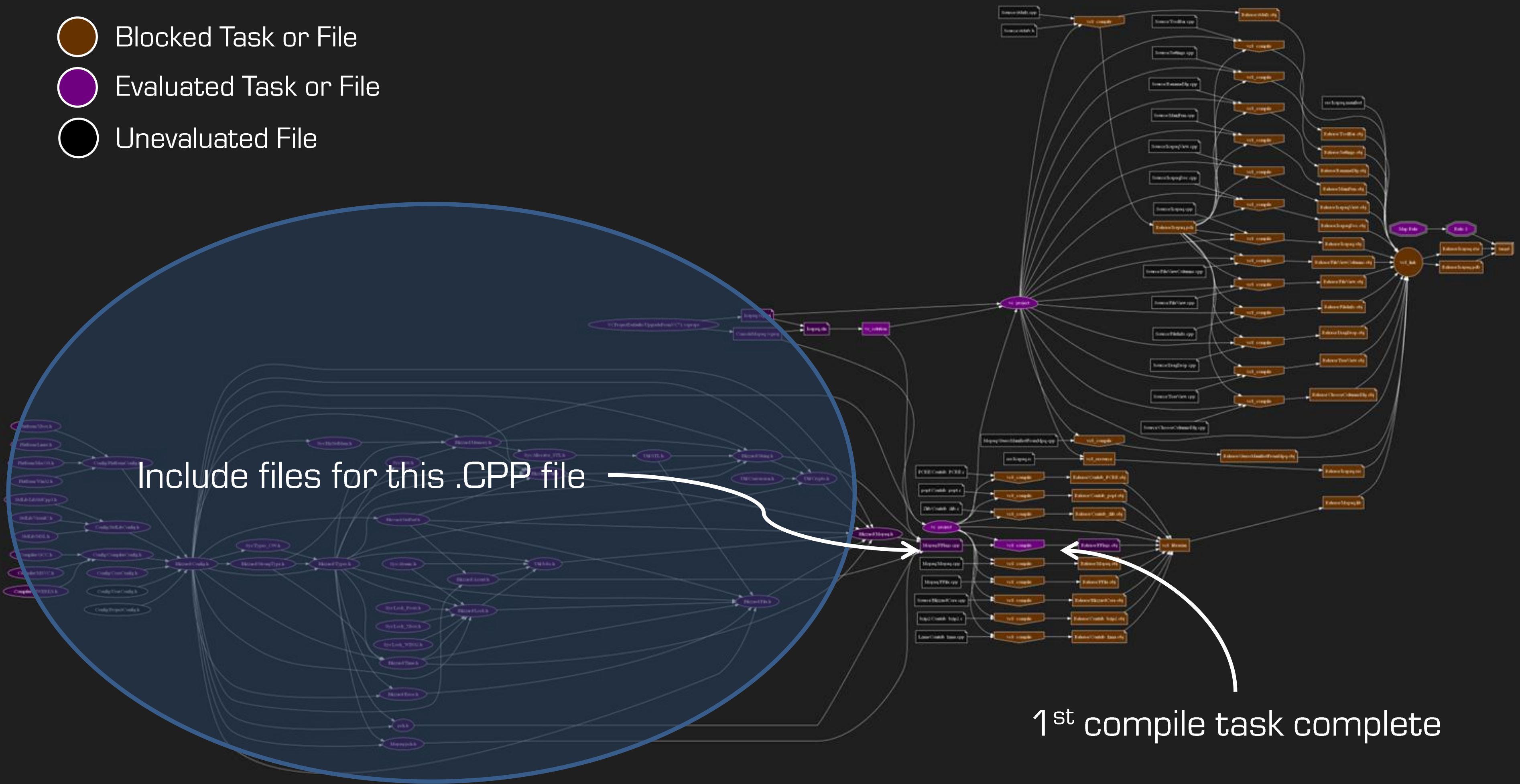


2nd Project task complete

● Blocked Task or File

● Evaluated Task or File

○ Unevaluated File



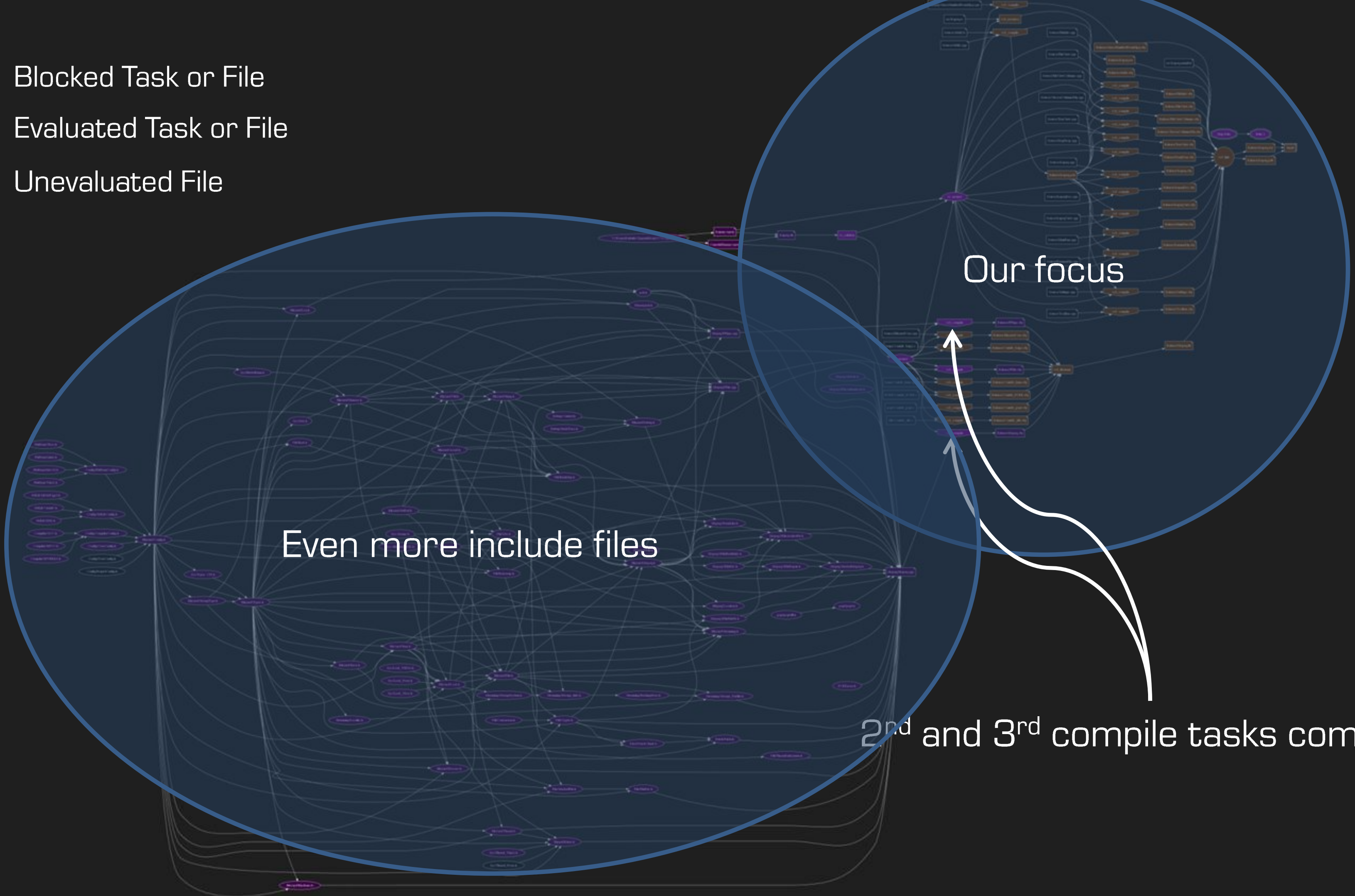
Include files for this .CPP file

1st compile task complete

🟤 Blocked Task or File

🟣 Evaluated Task or File

⚫ Unevaluated File



Our focus

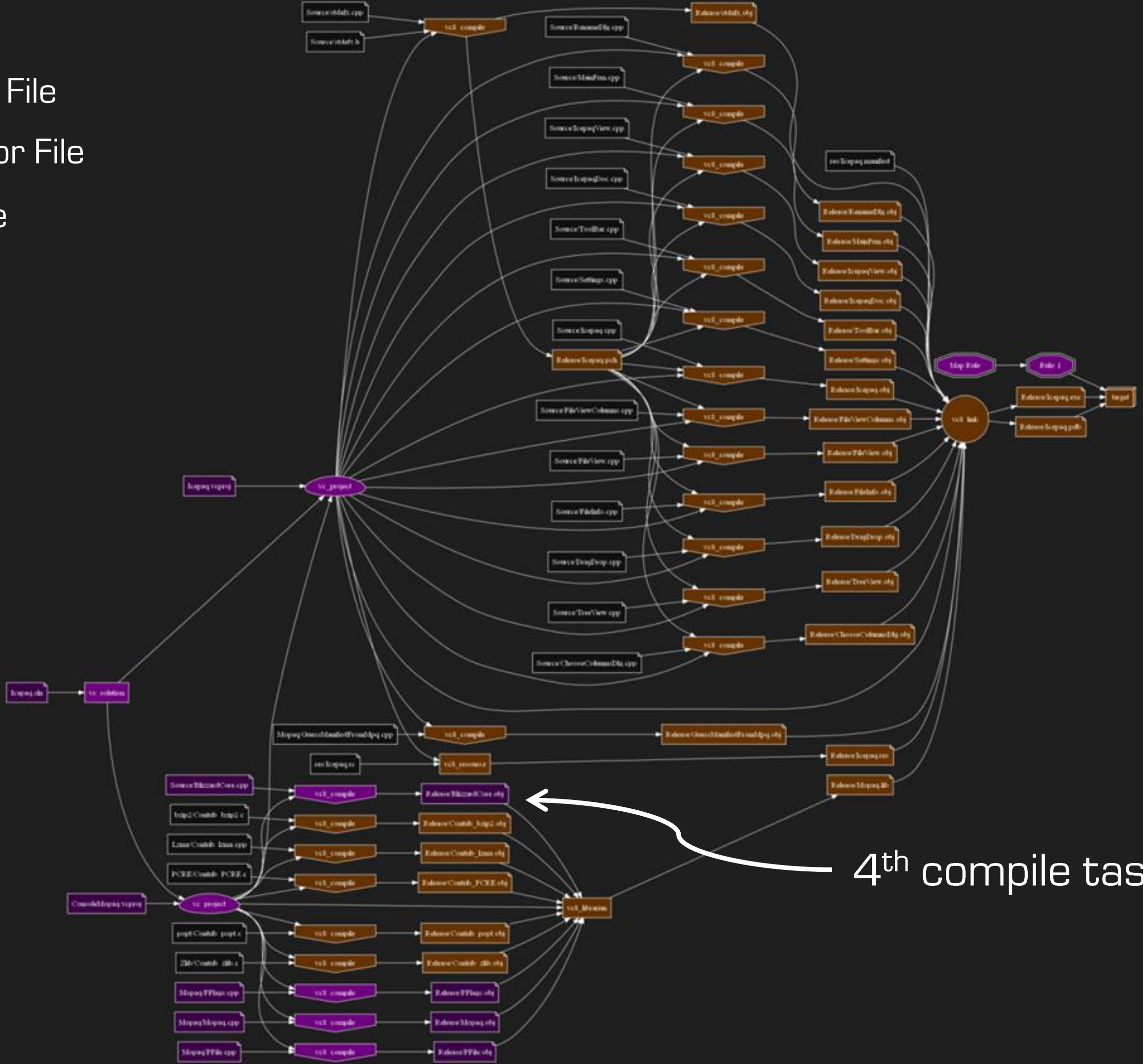
Even more include files

2nd and 3rd compile tasks complete

● Blocked Task or File

● Evaluated Task or File

● Unevaluated File

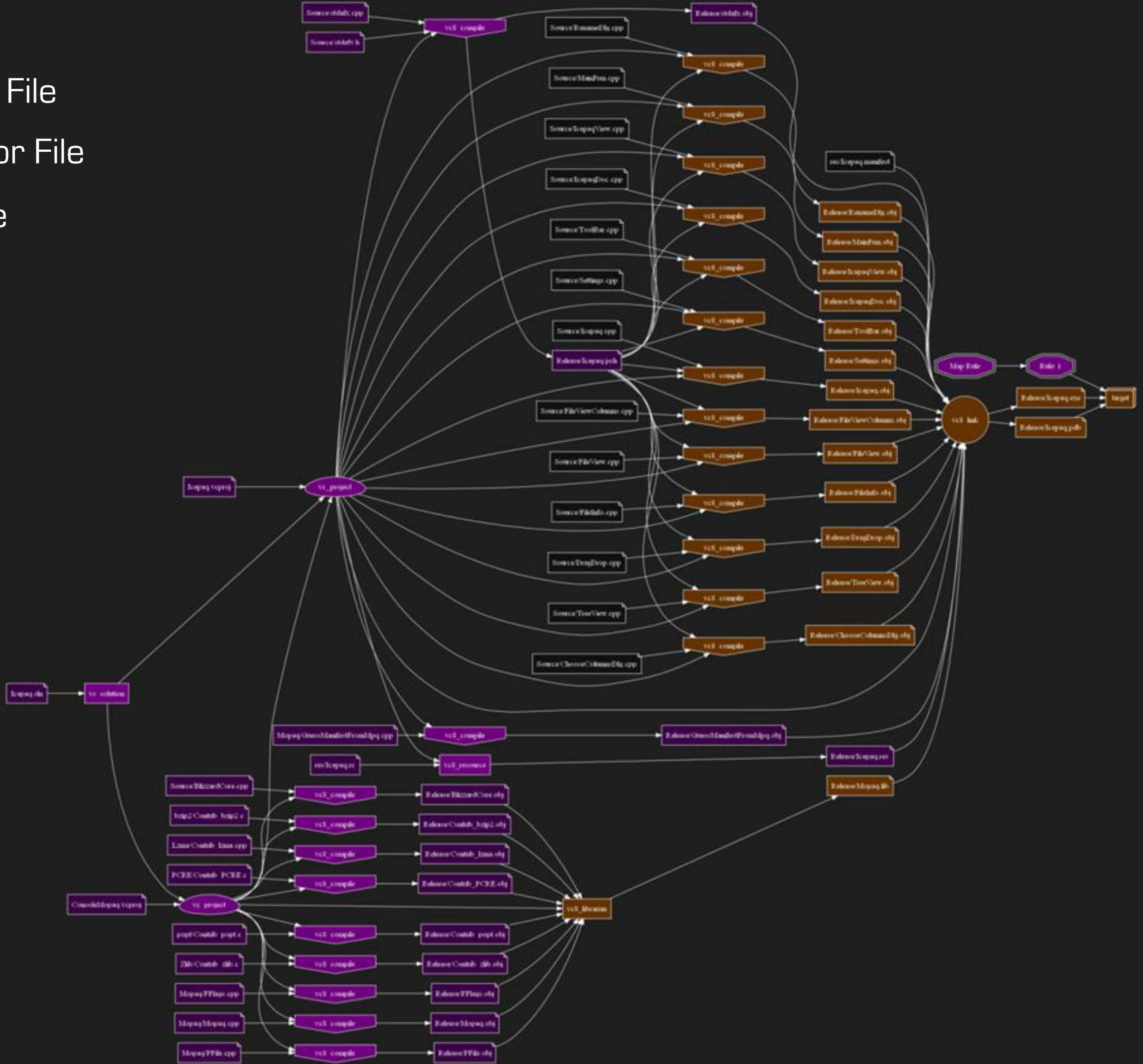


4th compile task complete

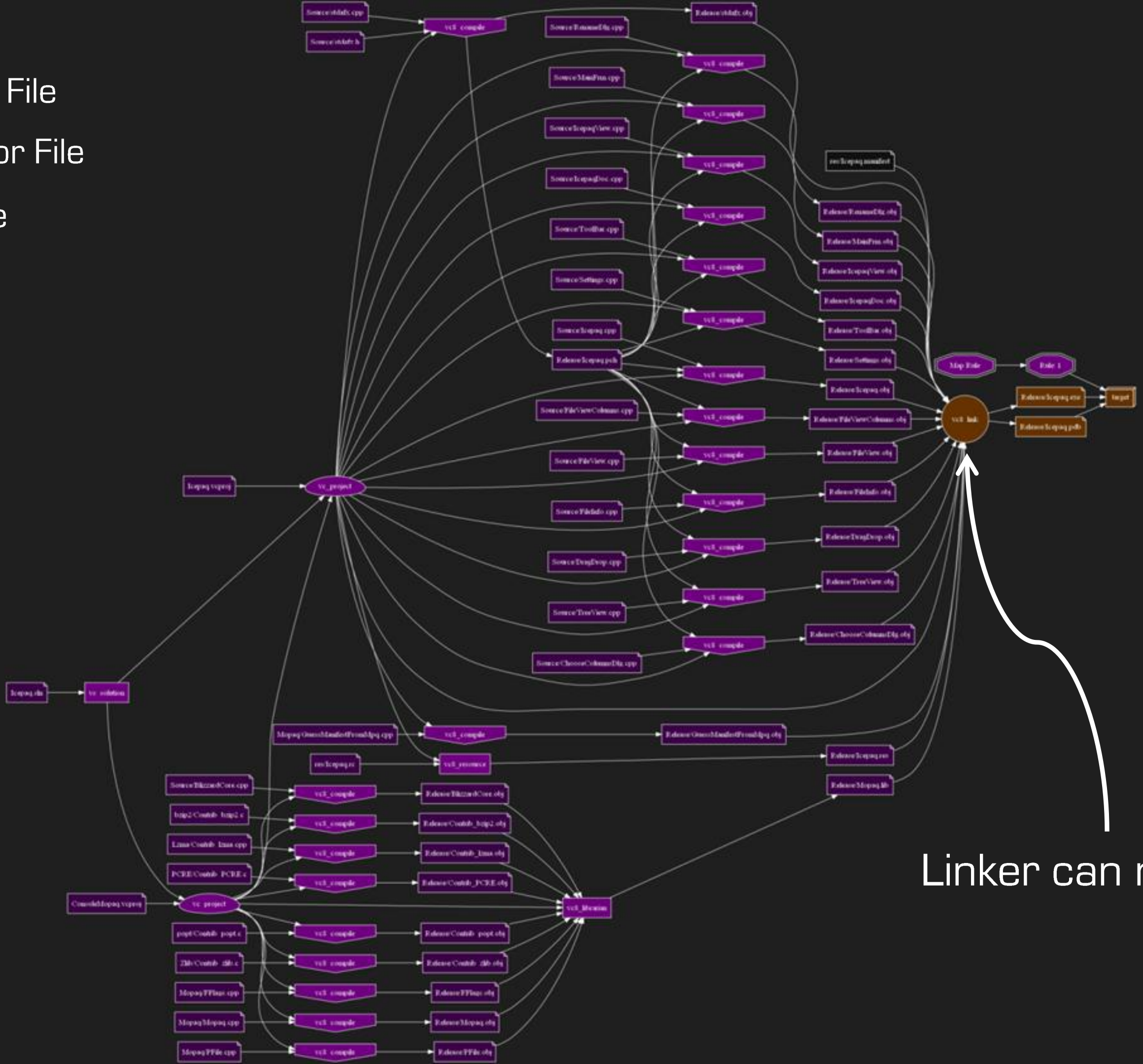
● Blocked Task or File

● Evaluated Task or File

● Unevaluated File



- 🟫 Blocked Task or File
- 🟪 Evaluated Task or File
- ⚫ Unevaluated File

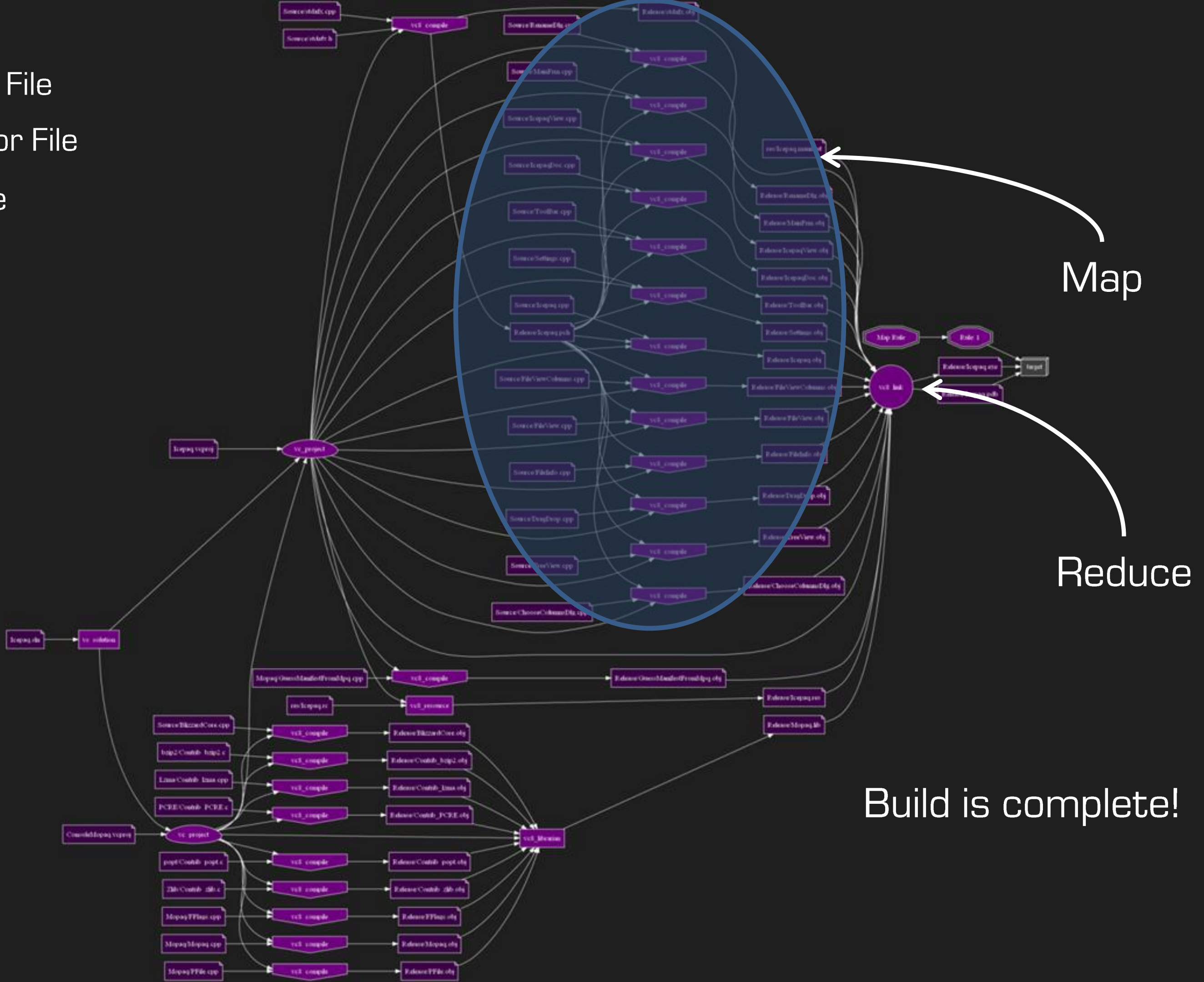


Linker can now run

● Blocked Task or File

● Evaluated Task or File

● Unevaluated File



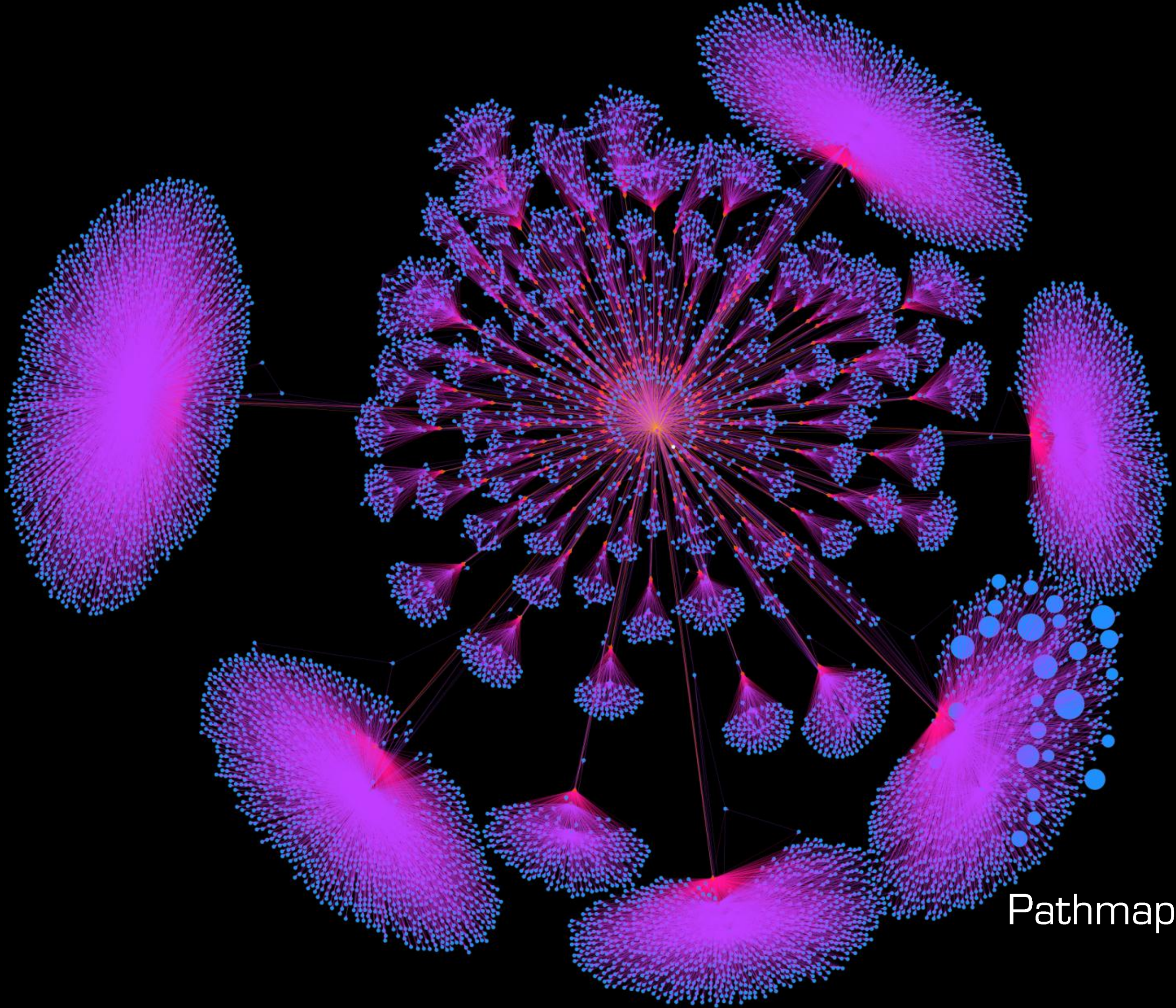
Map

Reduce

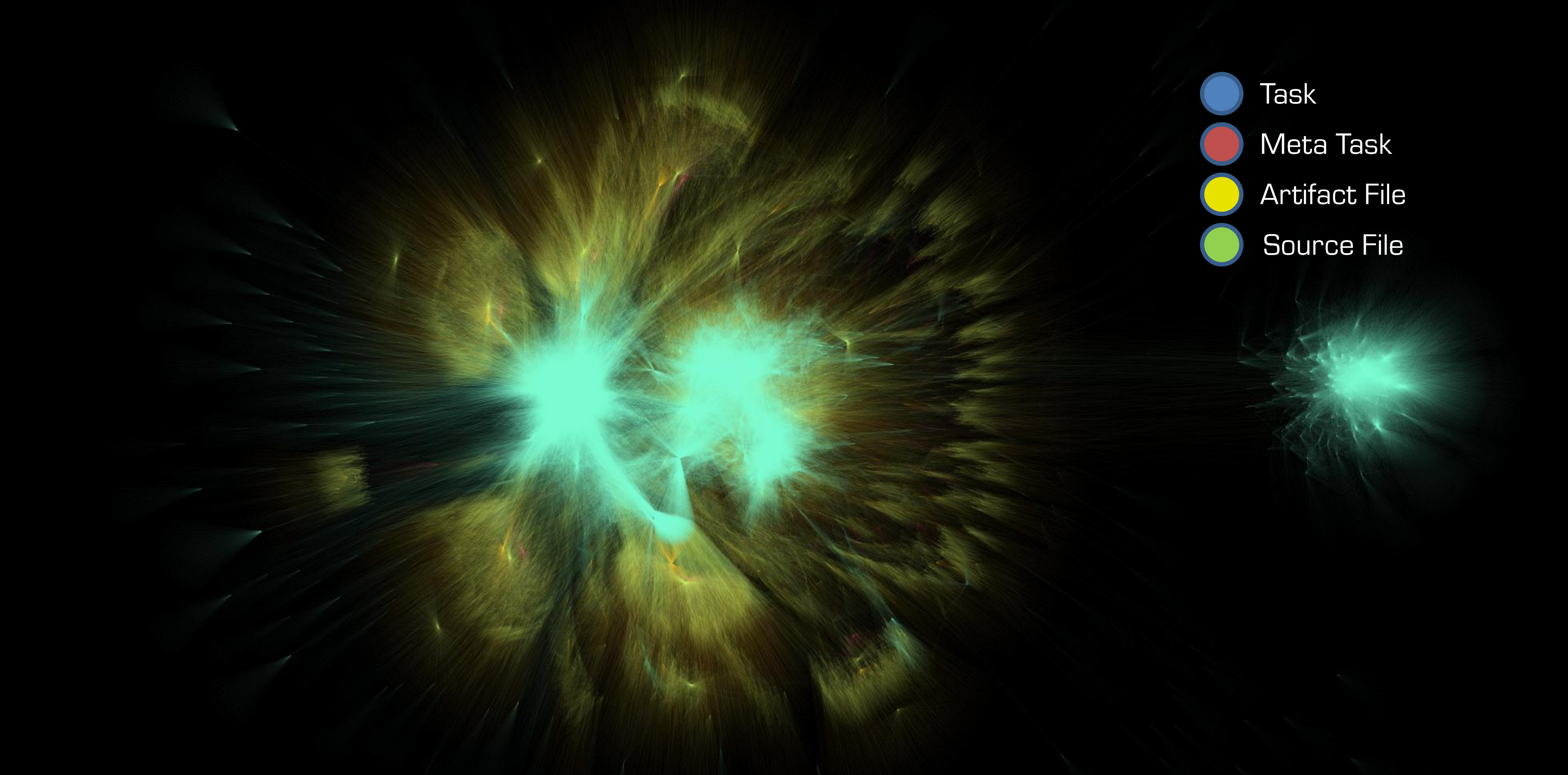
Build is complete!

USE CASES

- Code compilation
- Asset transformation
- Texture mipmap reduction
- Path map generation
- Shader compilation
- Archive creation
- Patch generation

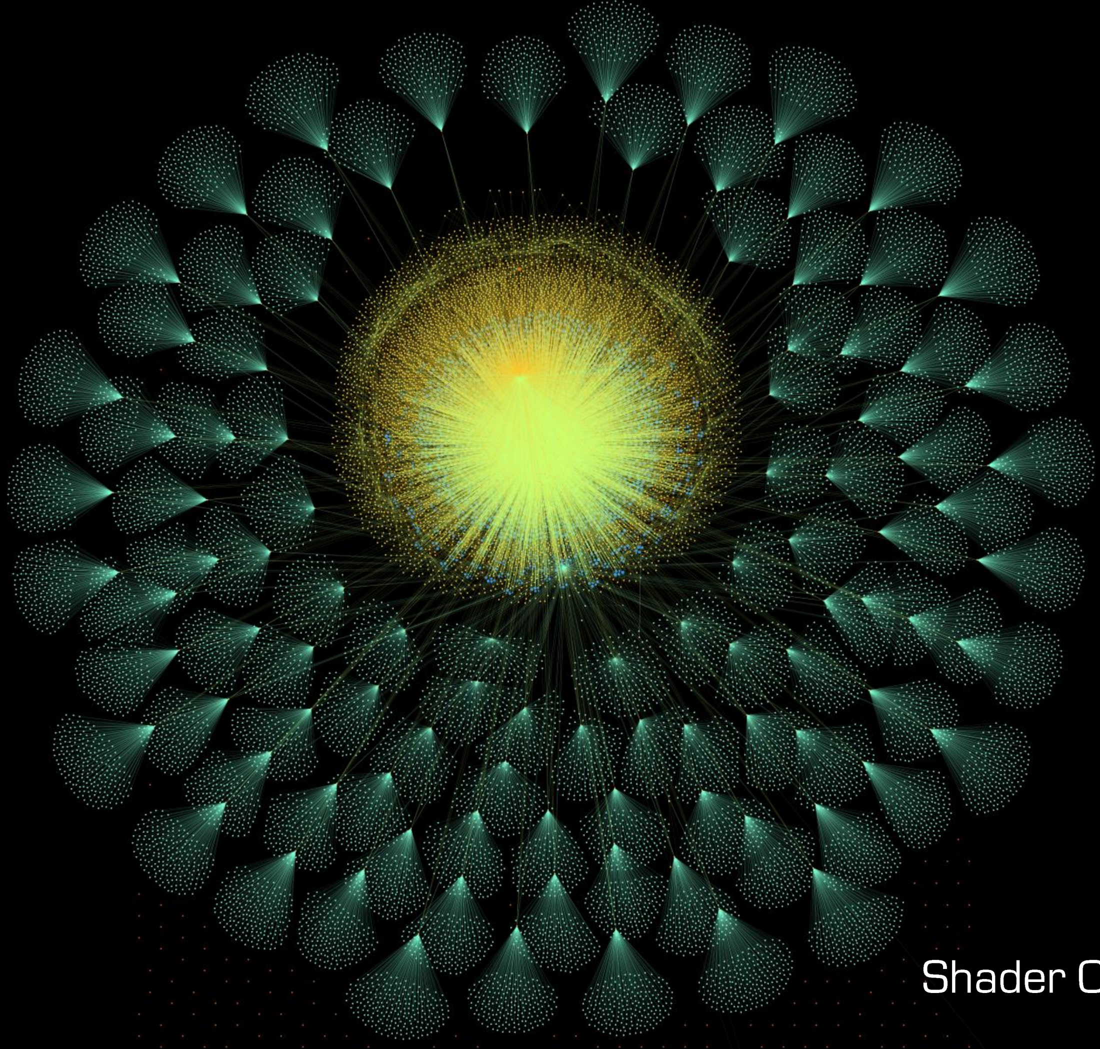


Pathmap Generation Task Graph



Pathmap Generation File & Task Graph

Shader Compilation File & Task Graph



LESSONS LEARNED

- Memory footprint issues related to large graphs
- Language difficulties with Erlang
  - Mnesia (built in distributed DB)
  - String handling performance
- Unrealized VS 2008 integration
- Replacing an 8 hour build process required many test runs of both the old and new systems

IMPLEMENTATION

FILE CACHE MANAGEMENT

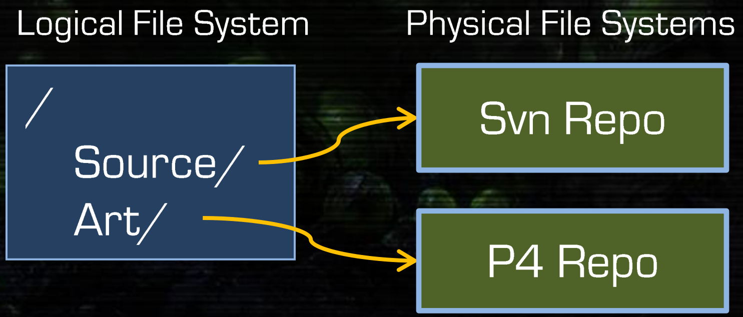
Traditional repo checkout system is problematic

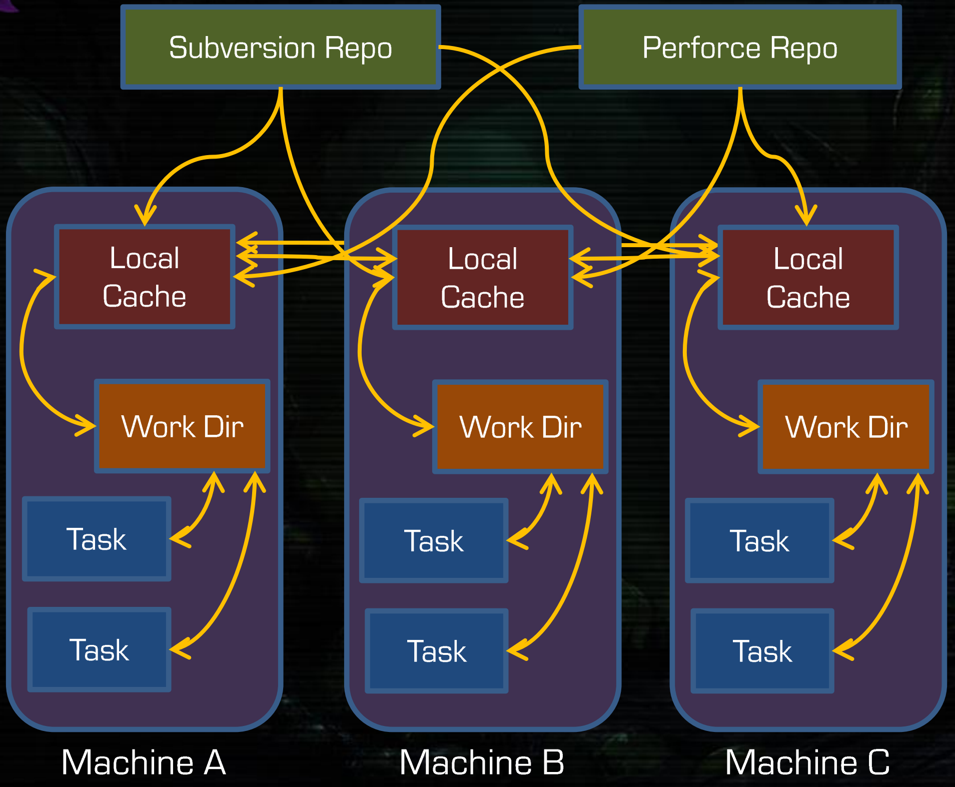
- Often fetches unused files
- Doesn't scale
  - Across branches
  - Across build machines
- File system tree includes build state that needs cleaning or repairing

FILE CACHE MANAGEMENT

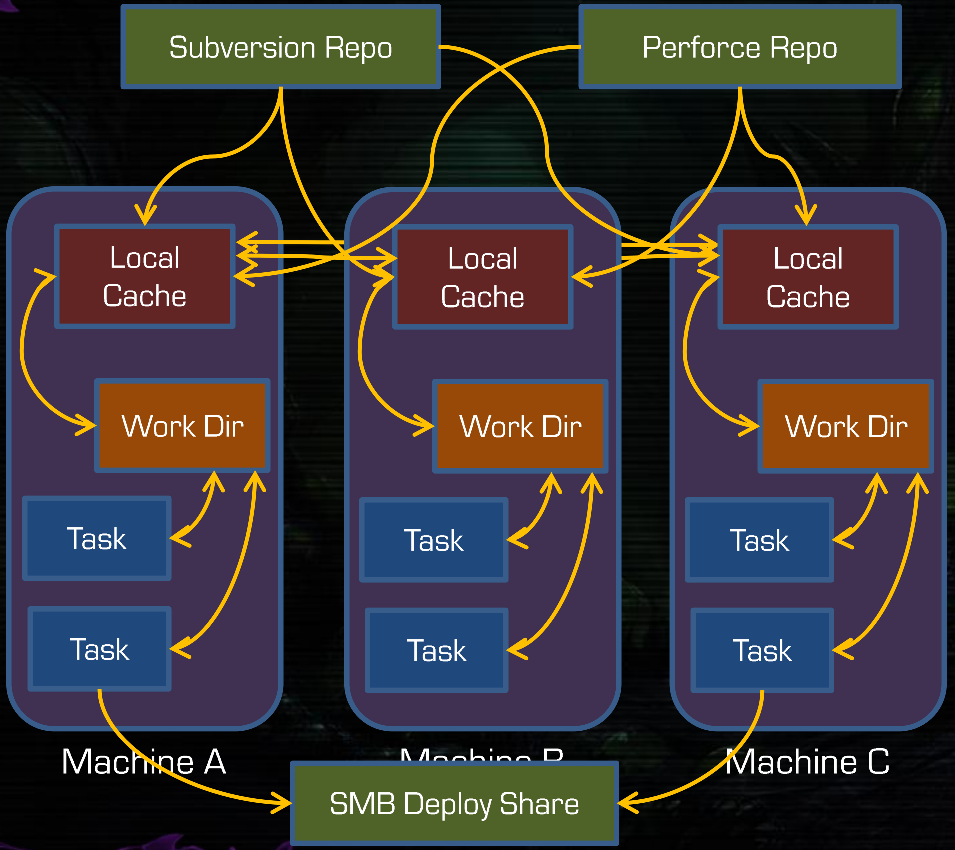
- Non-traditional repo interface
- Repo plugin driver architecture
  - Simple API
- Similar to GIT's internals
  - Content addressable cache
- Managed working directory via hard links to the local cache
- Ability to map repo paths to build paths

Virtual File System

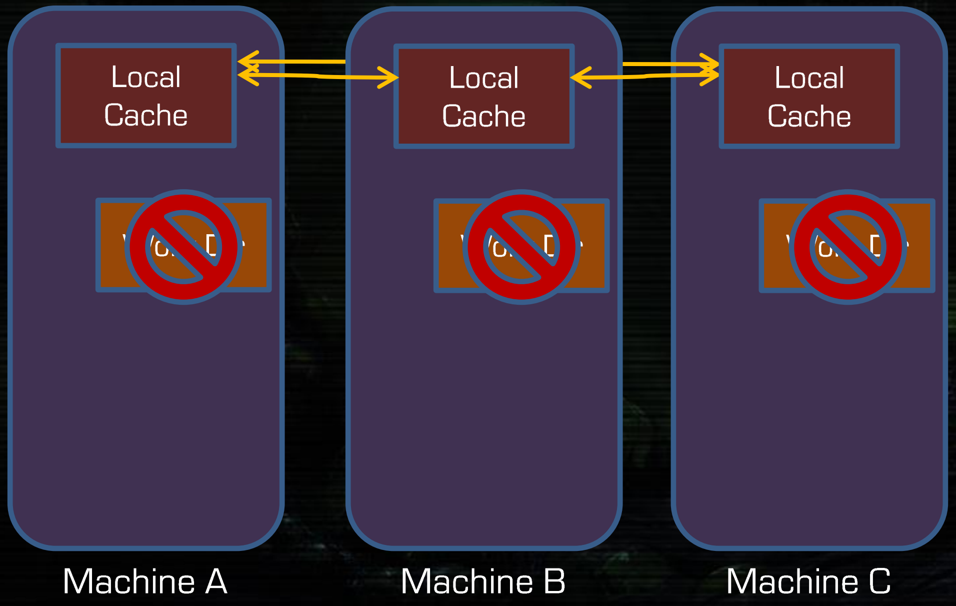




- Files not in cache are fetched (i.e. lazy)
- Files can be fetched from other machines
- Content addressable cache (md5)
- Hard linked to the working directory

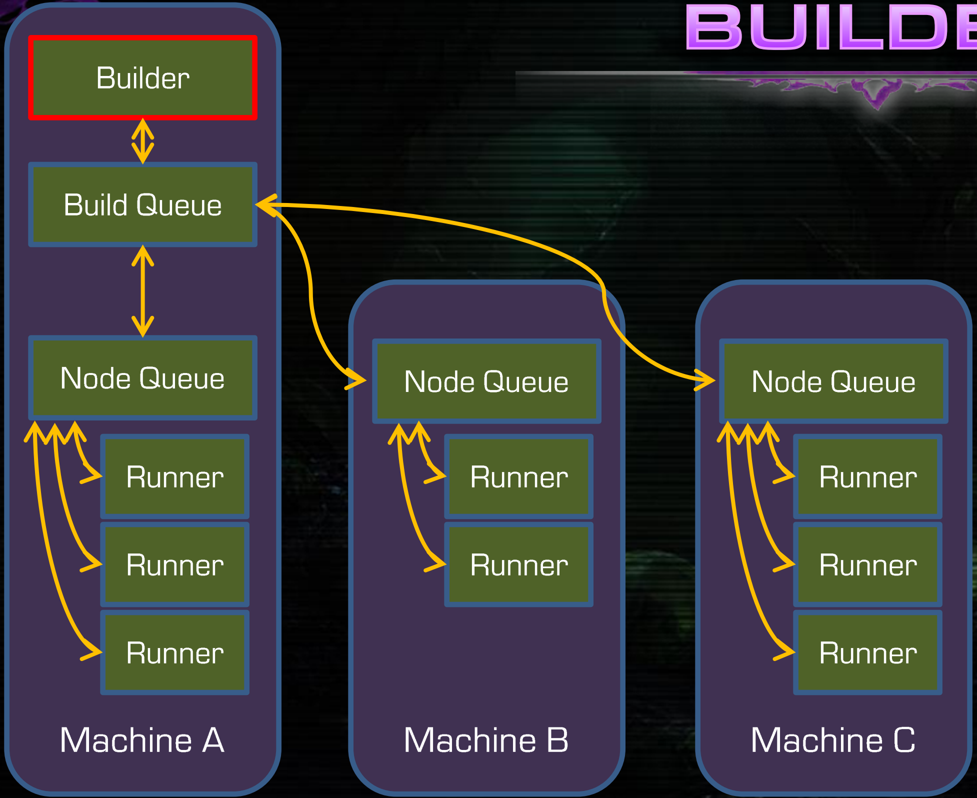


- Tasks write new files to the working dir
- Output files are hashed and hard linked to local cache
  - Immutability enforced via ACL
- Async deploy build results



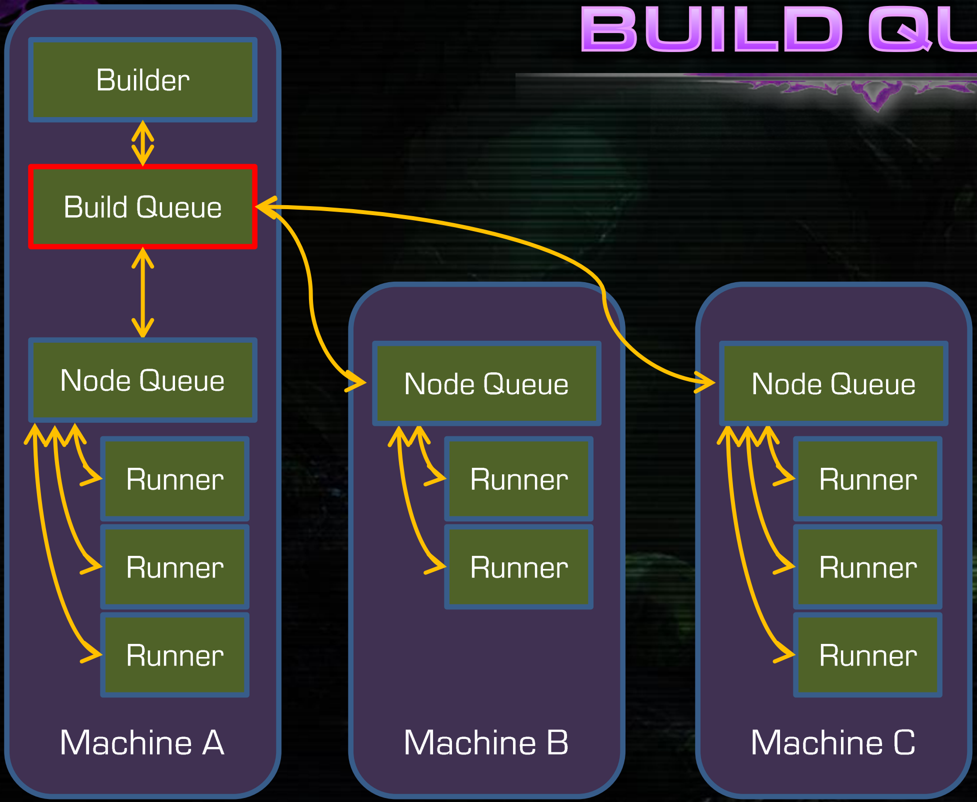
- Transient working directory
- Working directory can be recreated
  - Pause/Resume Build
  - Debugging

BUILDER



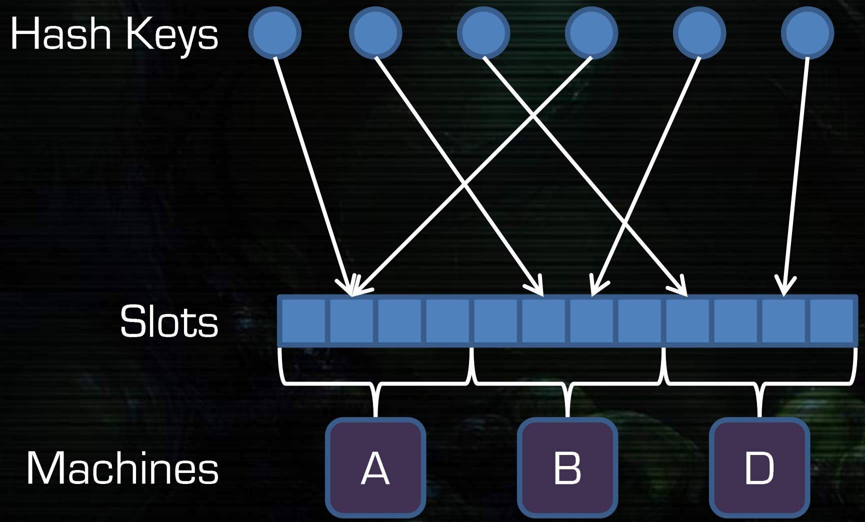
- Tracks task to task dependencies
- When a task is no longer blocked the task is dispatched to the build queue

BUILD QUEUE



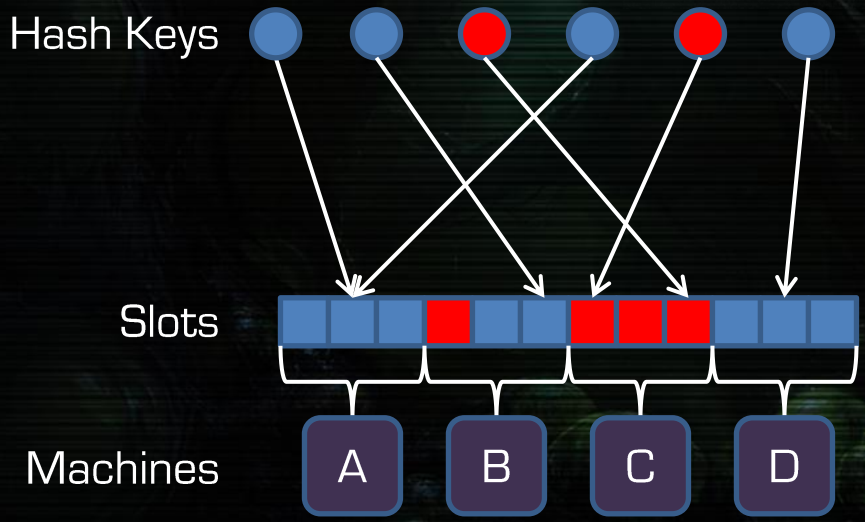
- Acts as a task router for the cluster
- Routes groups of tasks to specific machines based on a consistent hash of the parent task line

CONSISTENT HASHING



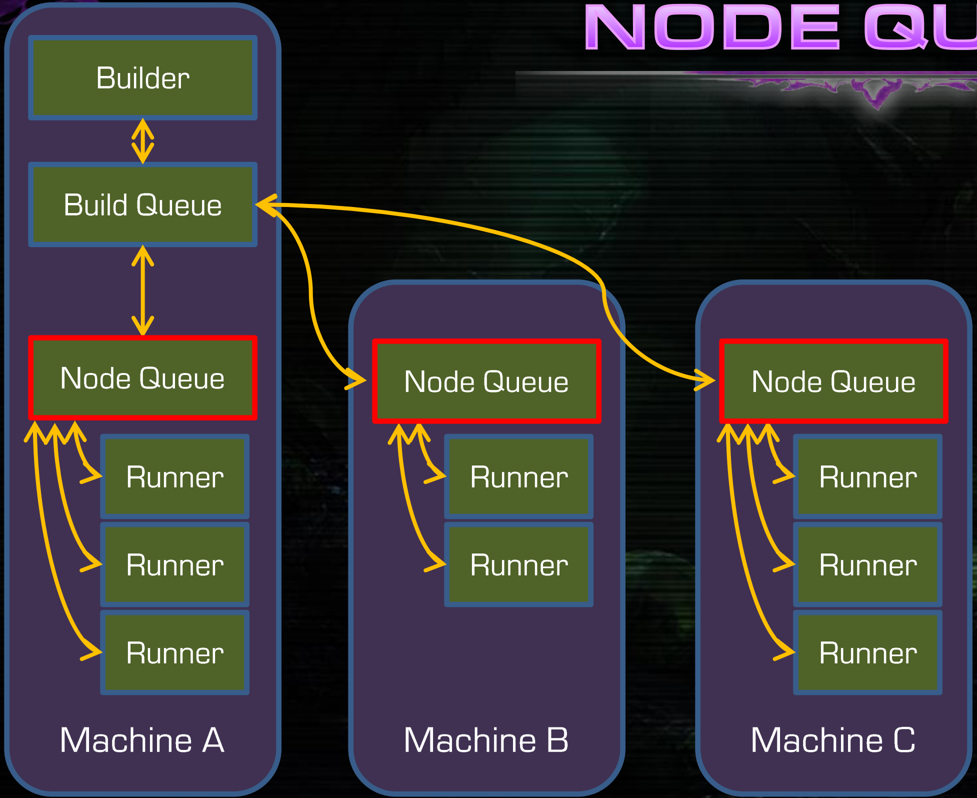
- Minimizes key map changes when nodes are added or removed
- Keys and machines are mapped to slots

CONSISTENT HASHING



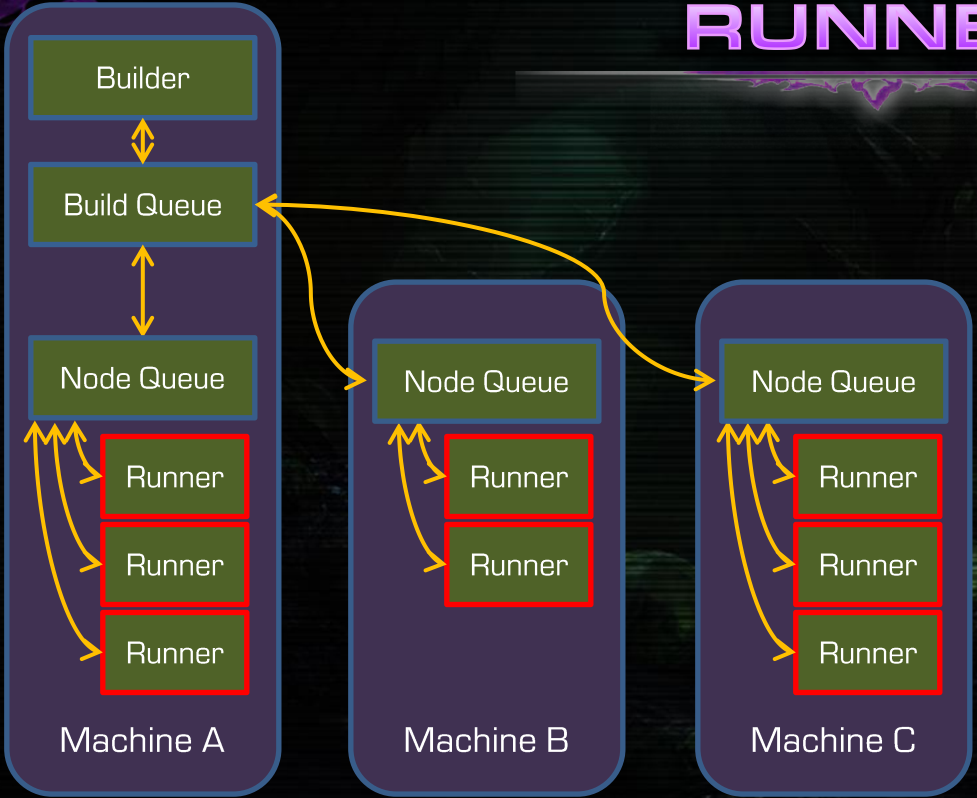
- When we add a node only keys / nodes (avg) are remapped

NODE QUEUE



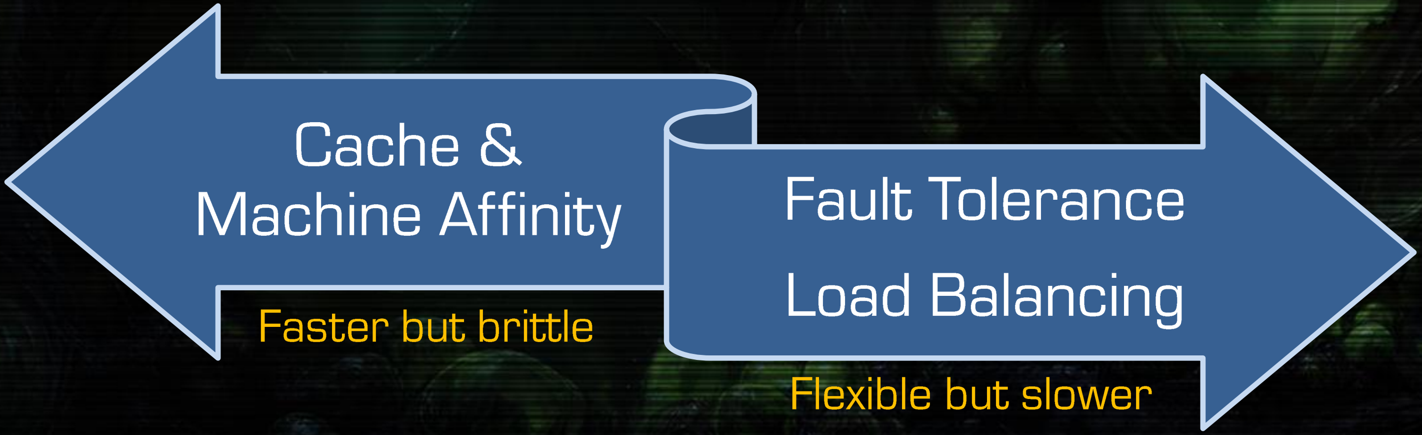
- Each machine manages its own task queue
- Allows low latency transitions to new tasks
- Max runners = CPU cores

RUNNER



- Exists for the lifetime of a single task
- Sets up the working directory and fetches any required files
- Verifies and hashes output files

LOAD BALANCING



Cache &
Machine Affinity

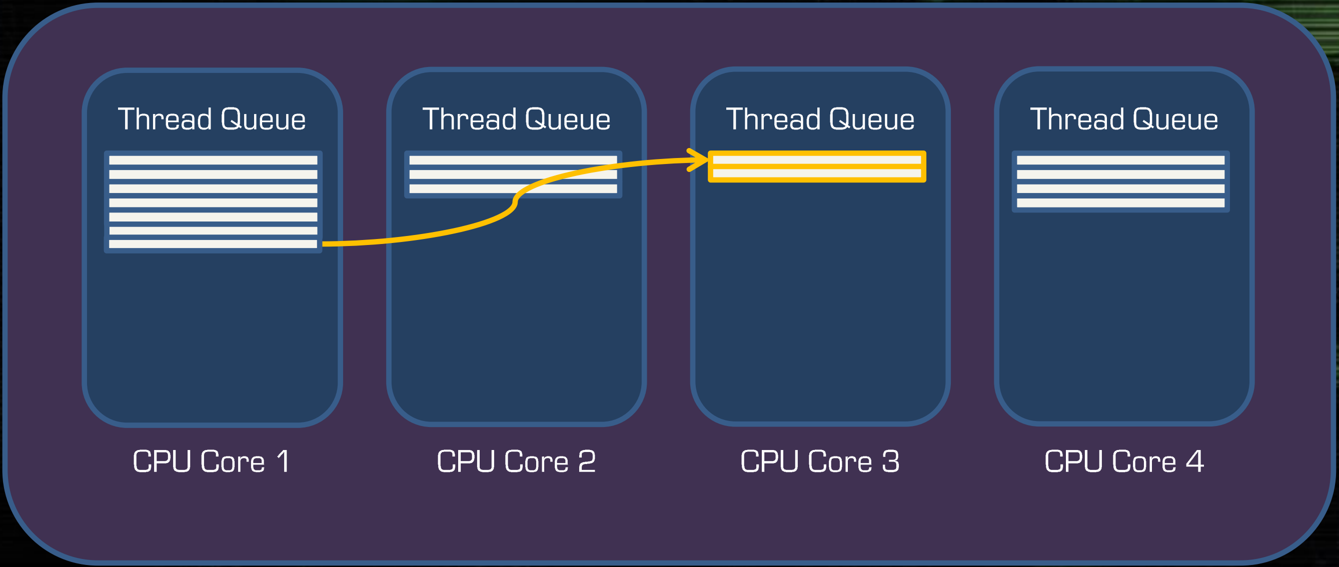
Faster but brittle

Fault Tolerance

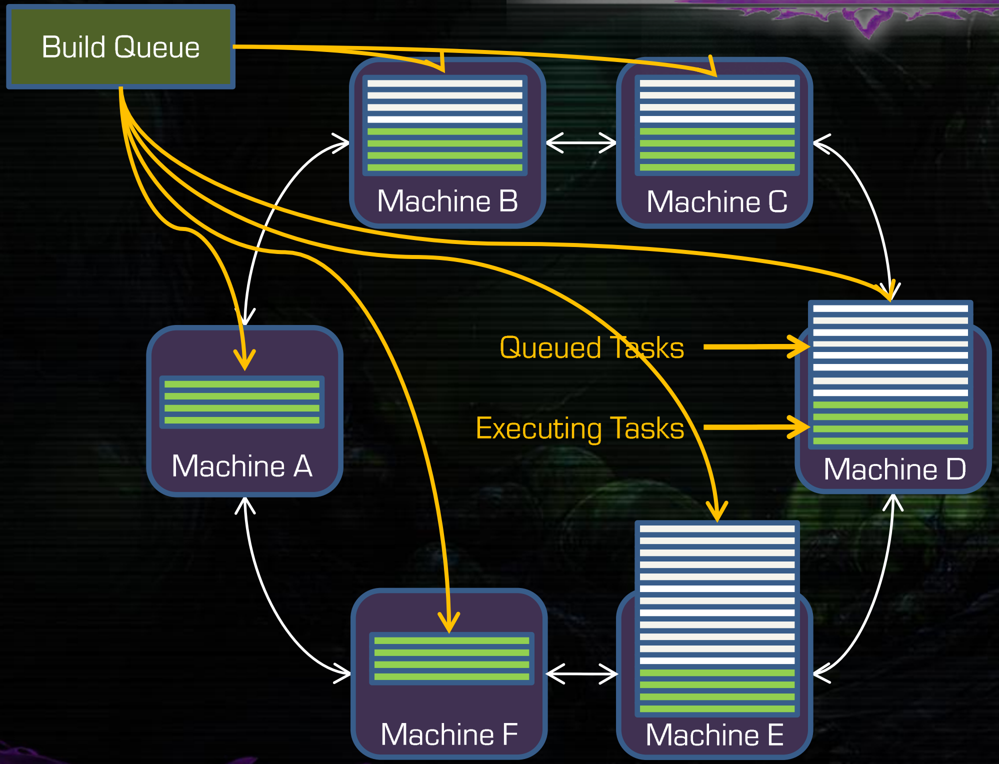
Load Balancing

Flexible but slower

THREAD WORK STEALING

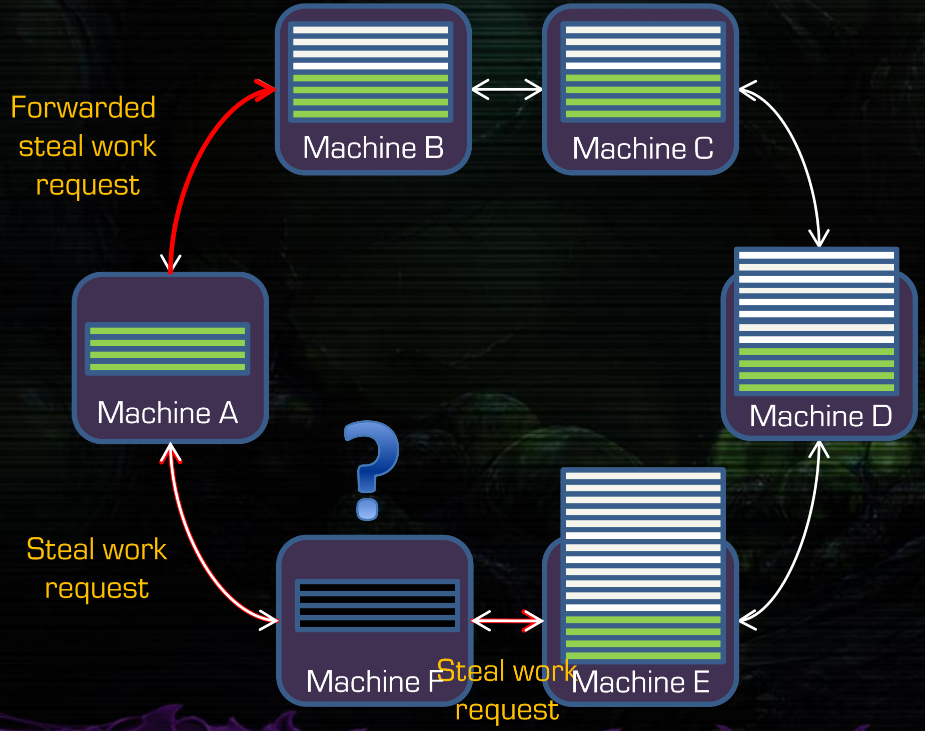


TASK STEALING



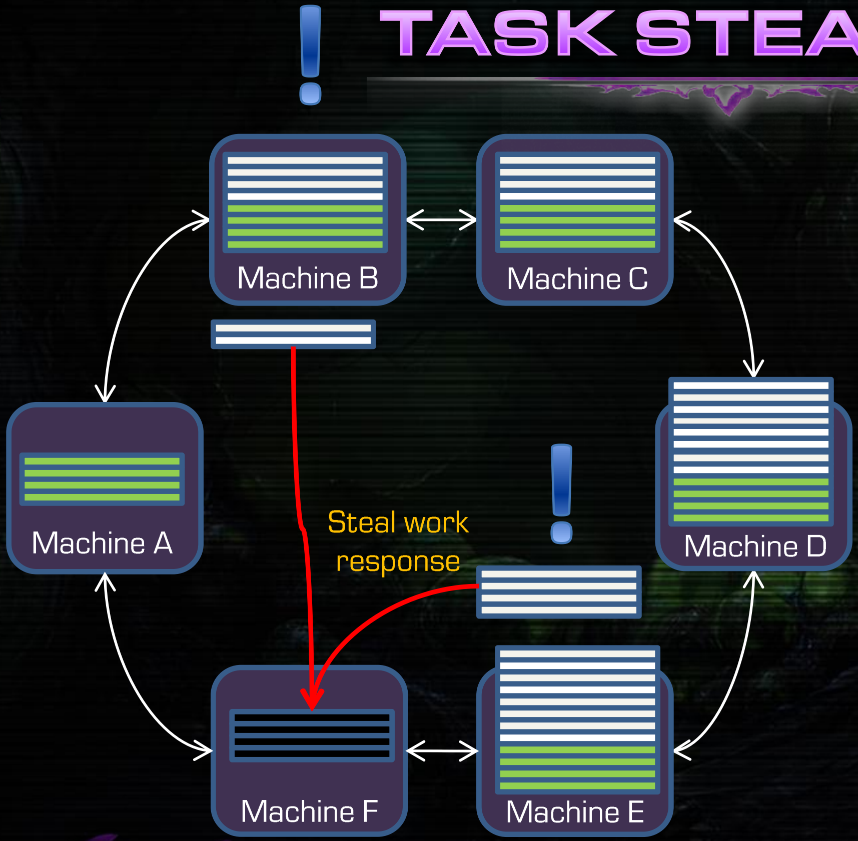
- Each machine executes tasks from its local queue
- Task stealing occurs between neighbors in a loop
- Build queue process manages the loop

TASK STEALING

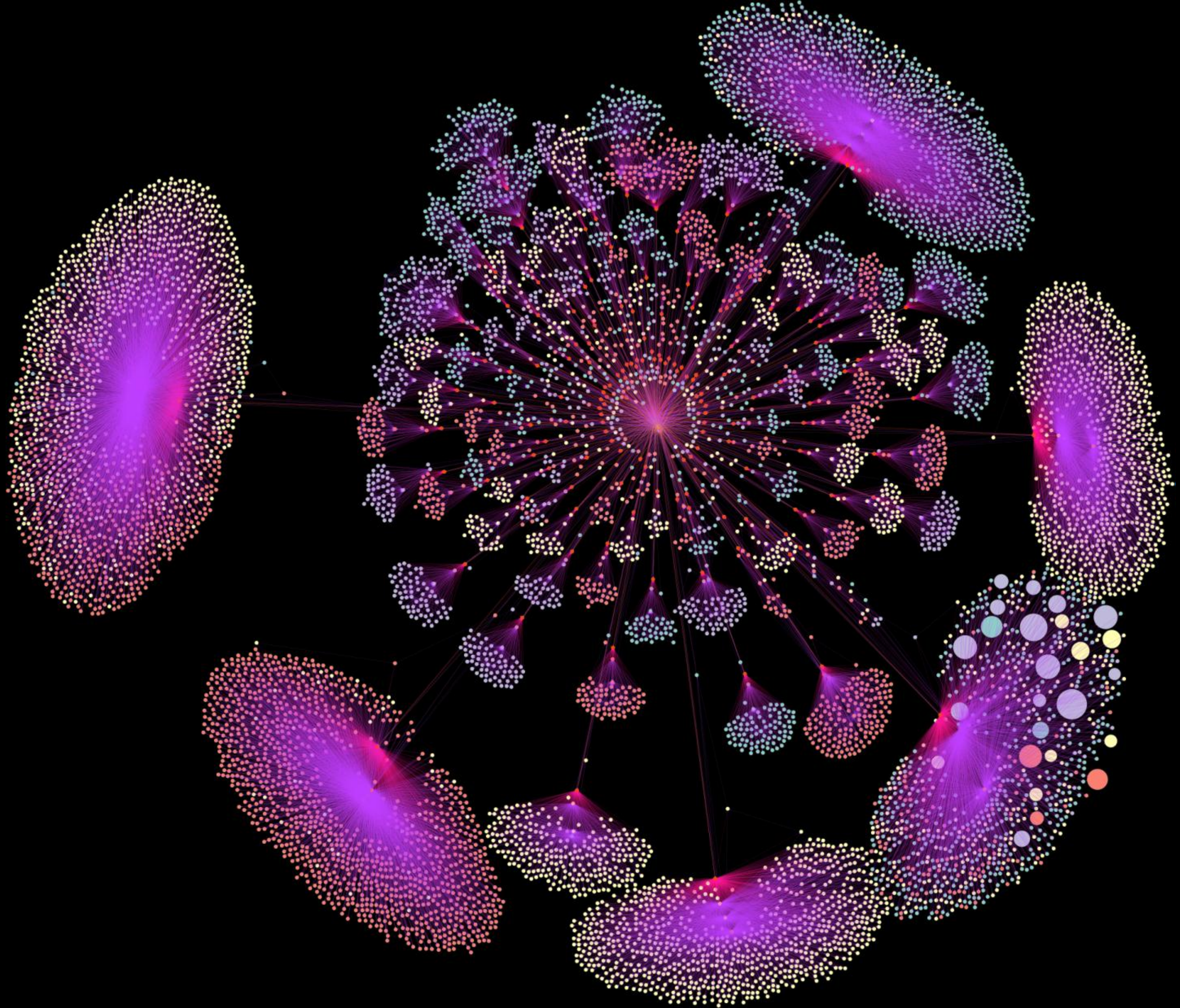


- Machine runs low on tasks event
- Asynchronous steal work requests are sent to both neighbors
- If a request can’t be fulfilled, its forwarded along the loop

TASK STEALING



- Response is sent directly to the original machine
- Build queue is notified of the ownership change

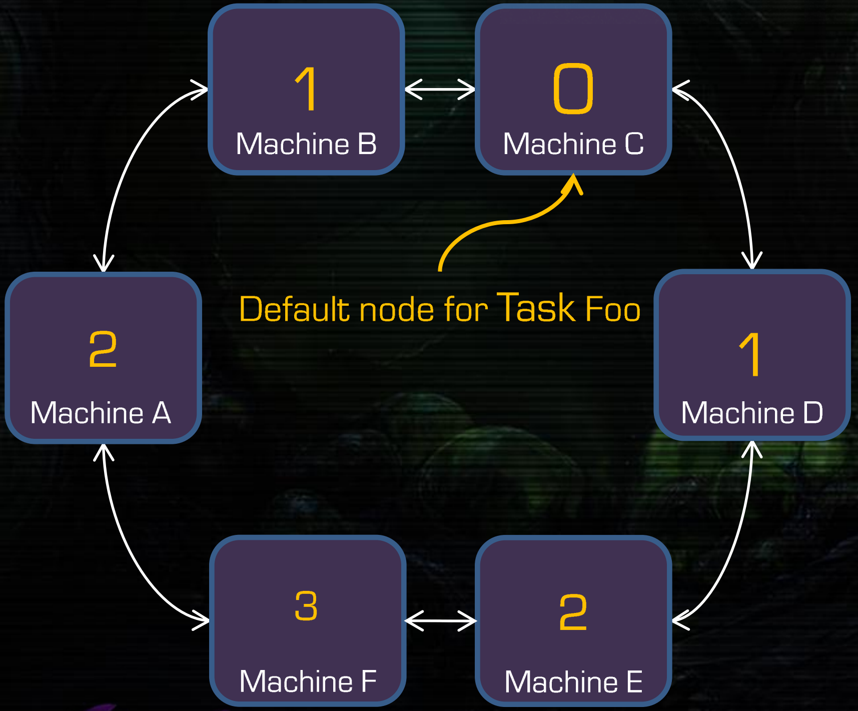


Node color represents the machine that executed the task

Smaller groups were all run on a single node

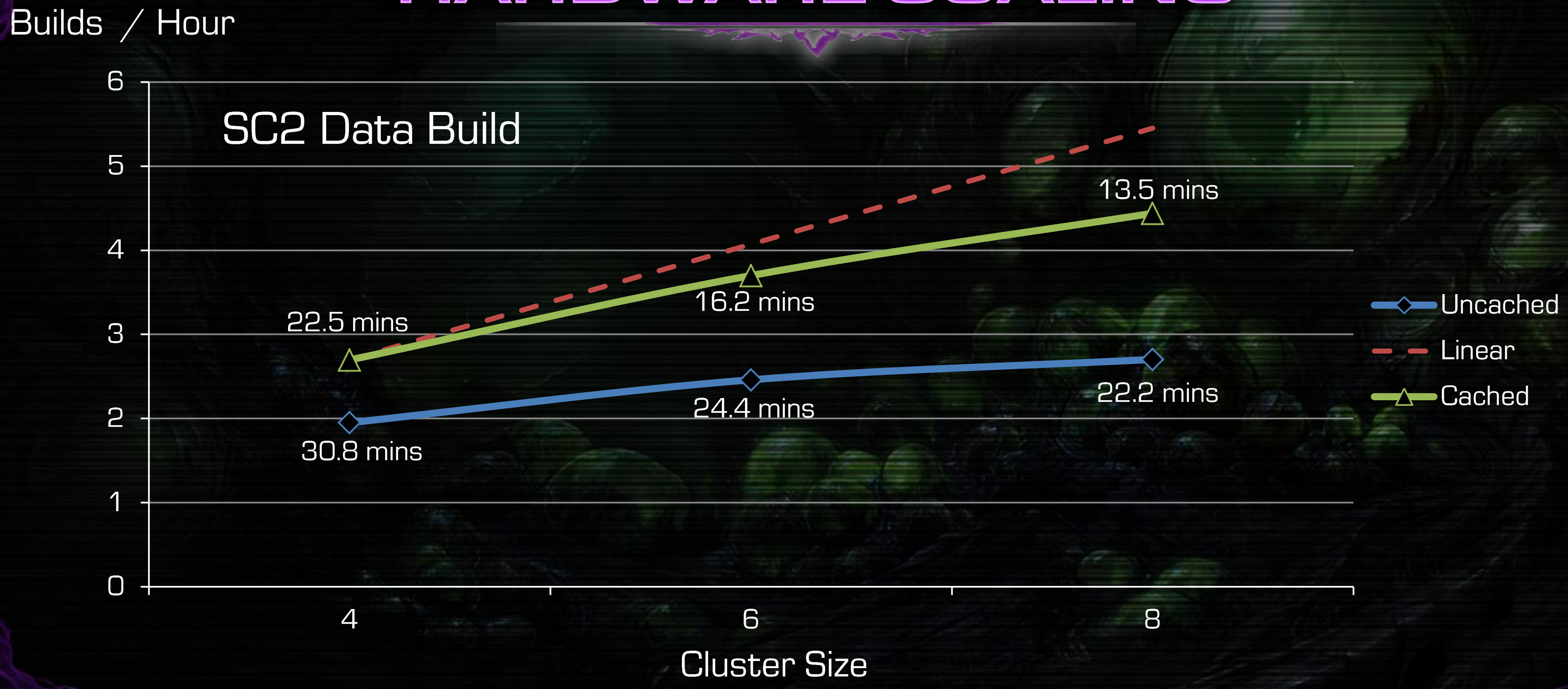
Largest groups ran on at least two nodes

SOFT CACHE AFFINITY

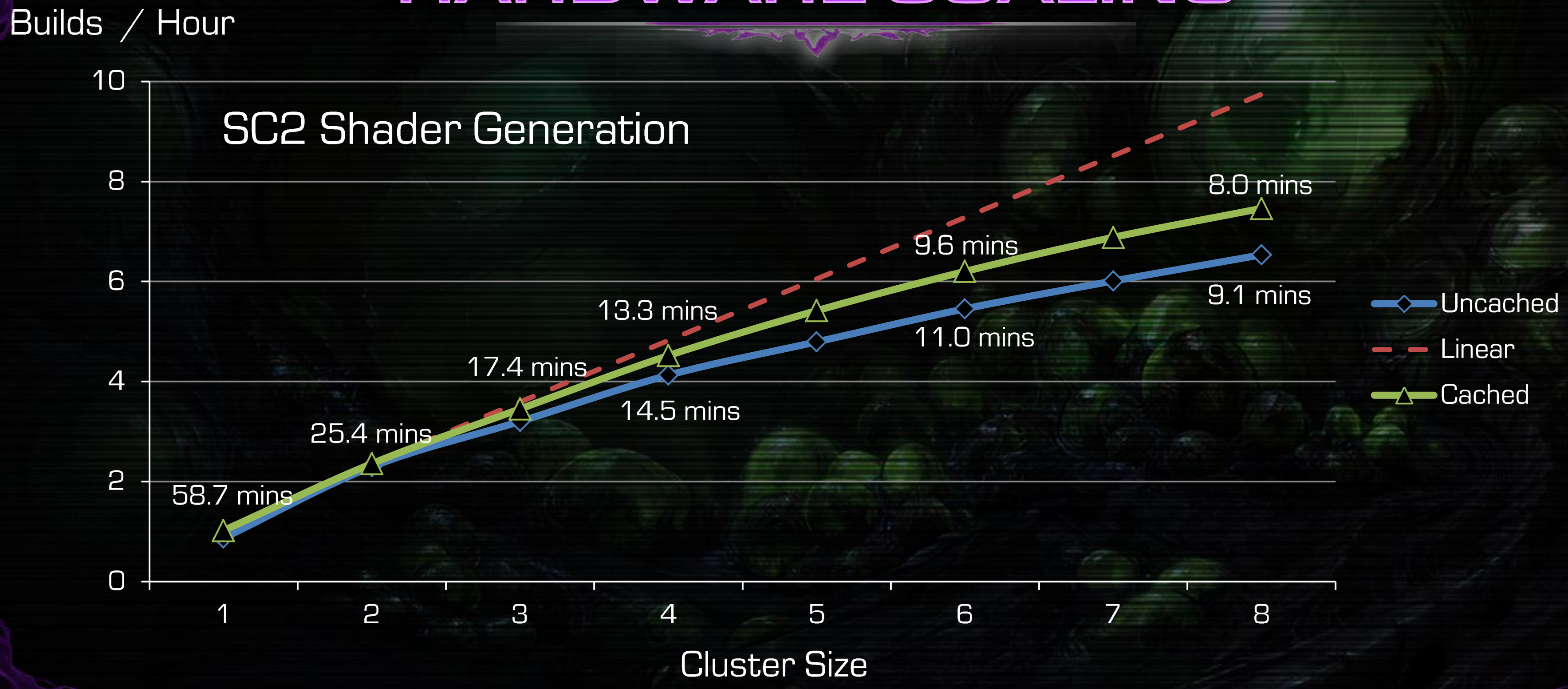


- Tasks are loosely bound to a machine based on the hash of its parent task
- Likelihood of executing a specific task is based on the distance along the loop

HARDWARE SCALING



HARDWARE SCALING



HETEROGENEOUS LOAD BALANCING

I’m a PC

I’m a Mac

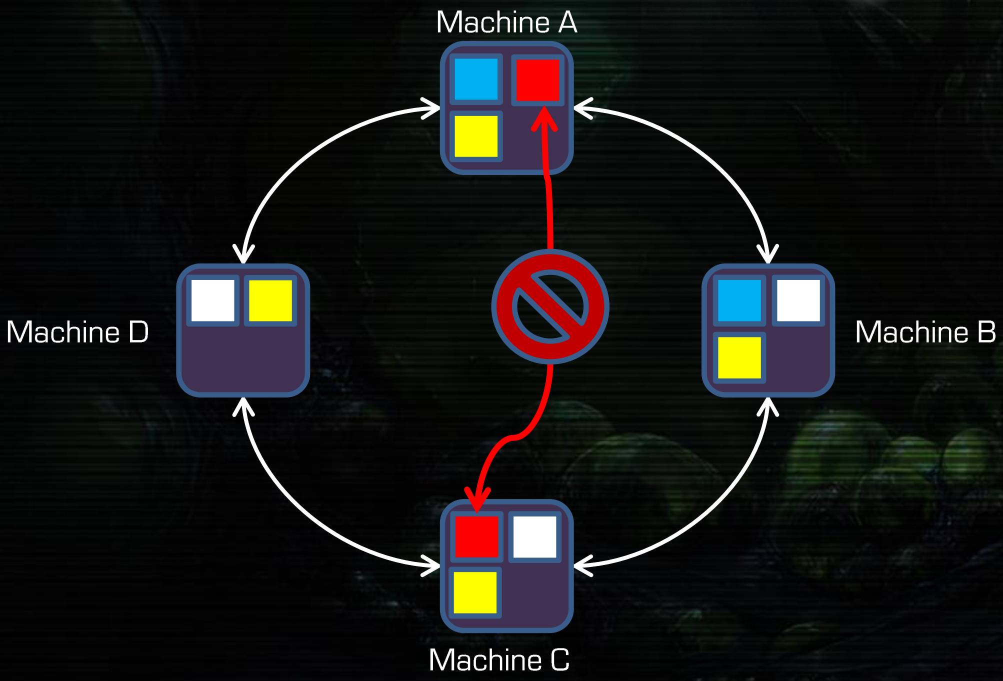
TASK MACHINE DEPENDENCIES

- Tasks have specific machine requirements
  - Visual Studio 2008
  - Perl
  - XCode 3.2
  - CUDA
  - Win64

TASK MACHINE MATRIX

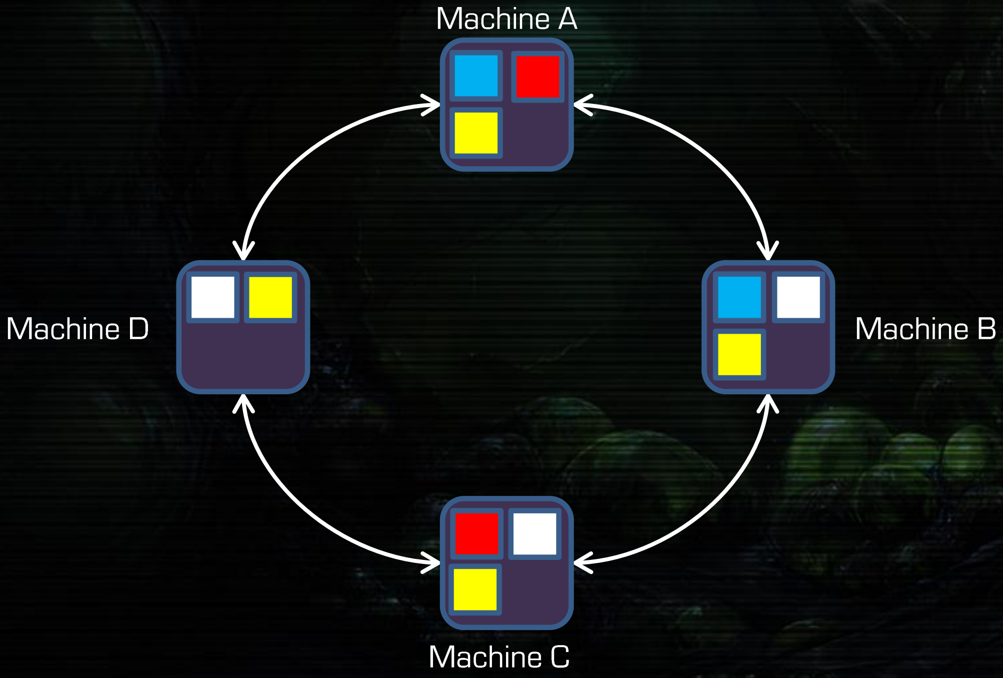
	Machine A	Machine B	Machine C	Machine D	
Task 1	X	X			🟦
Task 2	X		X		🟥
Task 3	X	X			🟦
Task 4		X	X	X	⬜
Task 5	X	X	X	X	🟨
Task 6	X	X	X	X	🟨
	🟦 🟥 🟨	🟦 ⬜ 🟨	🟥 ⬜ 🟨	⬜ 🟨	

LOOP ORDERING



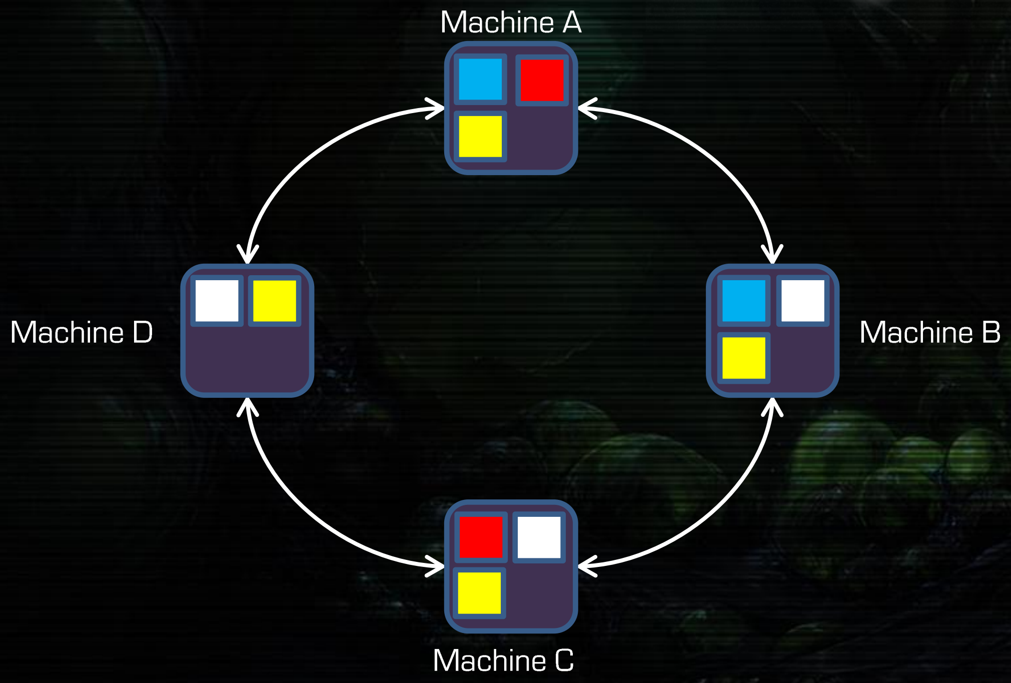
- Consistent ordering
- Avoid gaps between “colors”
- Gaps impede work stealing

LOOP ORDERING



- Similar to traveling salesman problem (NP hard)
- Order the machines from the most constrained colors to the least

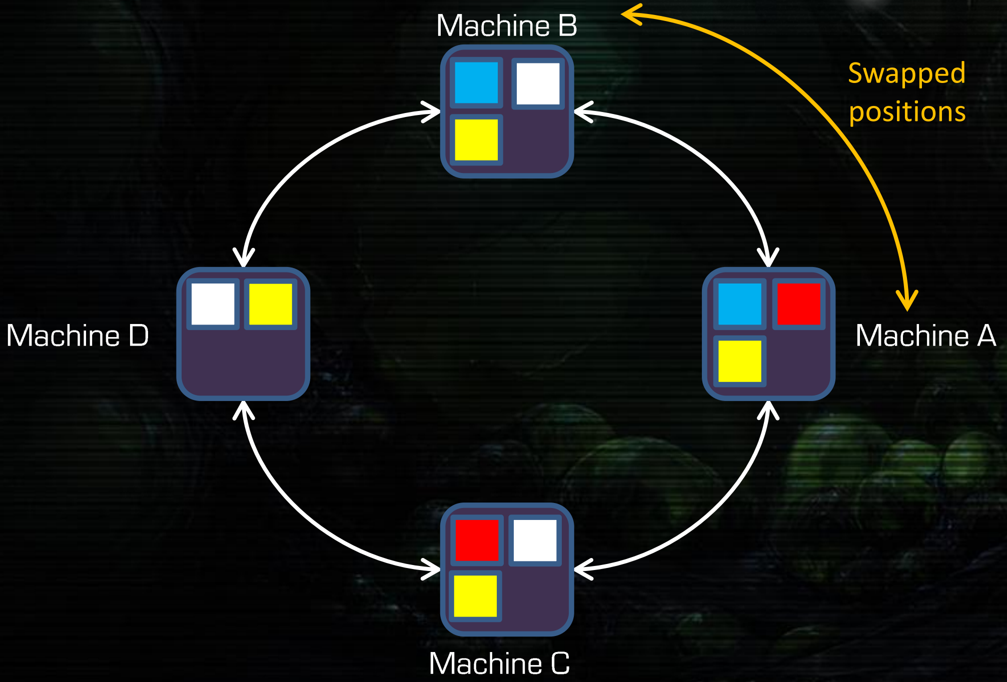
OPTIMAL ORDERING



Group by

- Blue (2 Machines)
- Red (2 Machines)
- White (3 Machines)
- Yellow (All machines)

OPTIMAL ORDERING



Group by

- Blue (2 Machines)
- Red (2 Machines)
- White (3 Machines)
- Yellow (All machines)

WE'RE HIRING

Q&A

bwhittle@blizzard.com

UNDER THE HOOD OF BLIZZARD'S INTERNAL BUILD SYSTEM

TOOL TIPS AND TRICKS

- No GUI dialogs or windowed error messages
- If an error occurs return a non zero result code
- If an error occurs because of a malformed input file, print out the filename and the error

TOOL TIPS AND TRICKS

- Avoid using exclusive read when opening input files

- Avoid using write access when opening input files that you don't actually write to

- Avoid editing files in place, or at the very least provide command line switches to explicitly name both the input and output files

TOOL TIPS AND TRICKS

- Don't have fixed / hardcoded file paths (especially absolute file paths). All filenames should be assignable via command line switches
- If the tool does any type of batch processing of files, ensure that individual steps can be executed independently via additional command line switches

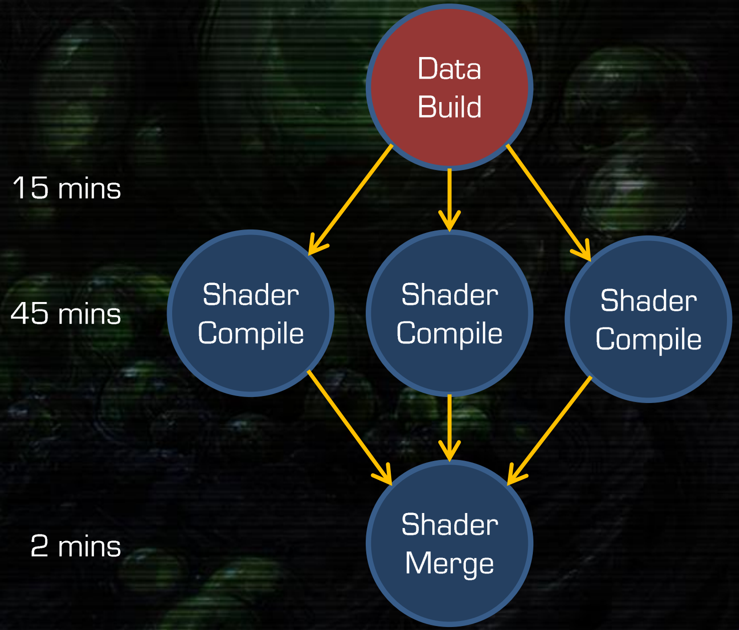
TOOL TIPS AND TRICKS

- Embed paths in data, not code (write out a text file describing the paths)

- Minimize dependencies

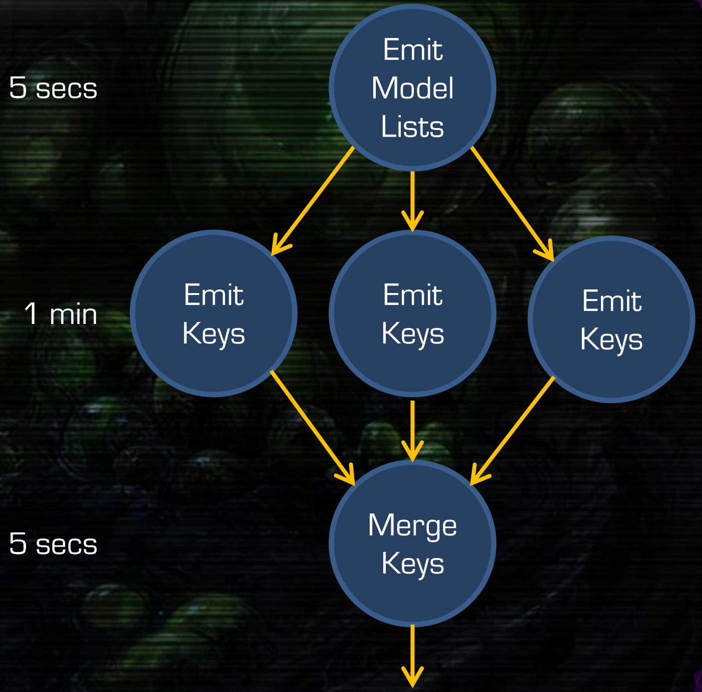
SHADER COMPILE (PREVIOUS)

- Wait for data build to complete
- Split shader compilation into 5 batches based on graphics quality
  - Copy entire game (all locales)
  - For each model in game generate shader pair for each unique shader key
- Merge shader caches



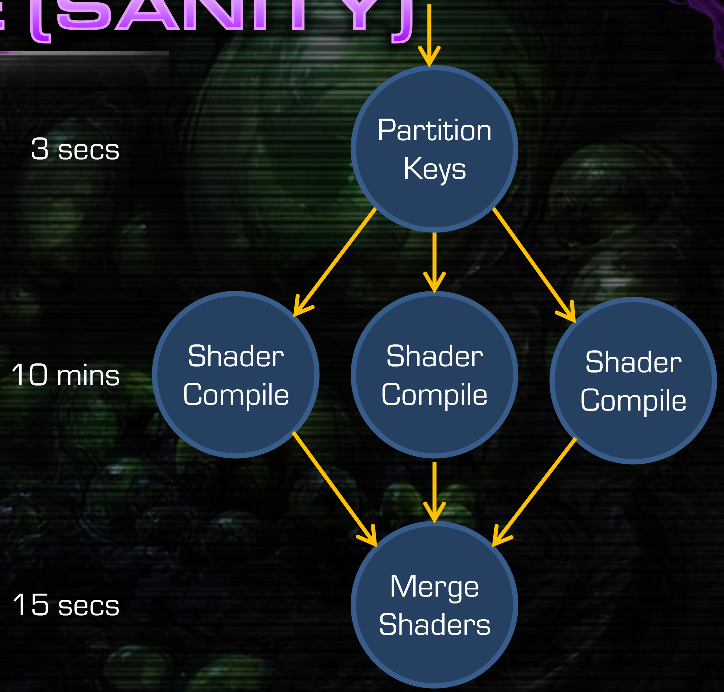
SHADER COMPILE (SANITY)

- Generate model lists
    - Partition into 100 bins
- For each bin and graphics setting, loop over models and emit shader keys
    - Assets automatically fetched if needed
    - New file format for keys
- Merge and dedup shader keys



SHADER COMPILE (SANITY)

- Partition keys into N bins (2,000)
- For each key bin generate shader pair from keys
- Merge shader caches via two layers
  - sqrt(N) = 45 caches per merge = 46 merge tasks


3 secs
Partition Keys
10 mins
Shader Compile
Shader Compile
Shader Compile
15 secs
Merge Shaders