

#WWDC18

# Building Faster in Xcode

Session 408

David Owens, Xcode Engineer  
Jordan Rose, Swift Engineer

# Building Faster in Xcode

# Building Faster in Xcode

**Increasing Build Efficiency**

**Reducing the Work on Rebuilds**

---

# Building Faster in Xcode

## **Increasing Build Efficiency**

Parallelizing your build process

Measuring your build time

## **Reducing the Work on Rebuilds**

Declaring script inputs and outputs

# Building Faster in Xcode

## Increasing Build Efficiency

---

Parallelizing your build process

Measuring your build time

Dealing with complex expressions

## Reducing the Work on Rebuilds

Declaring script inputs and outputs

Understanding dependencies in Swift

Limiting your Objective-C/Swift interface

# Parallelizing Your Build

Increasing build efficiency

# Xcode's Targets and Dependencies

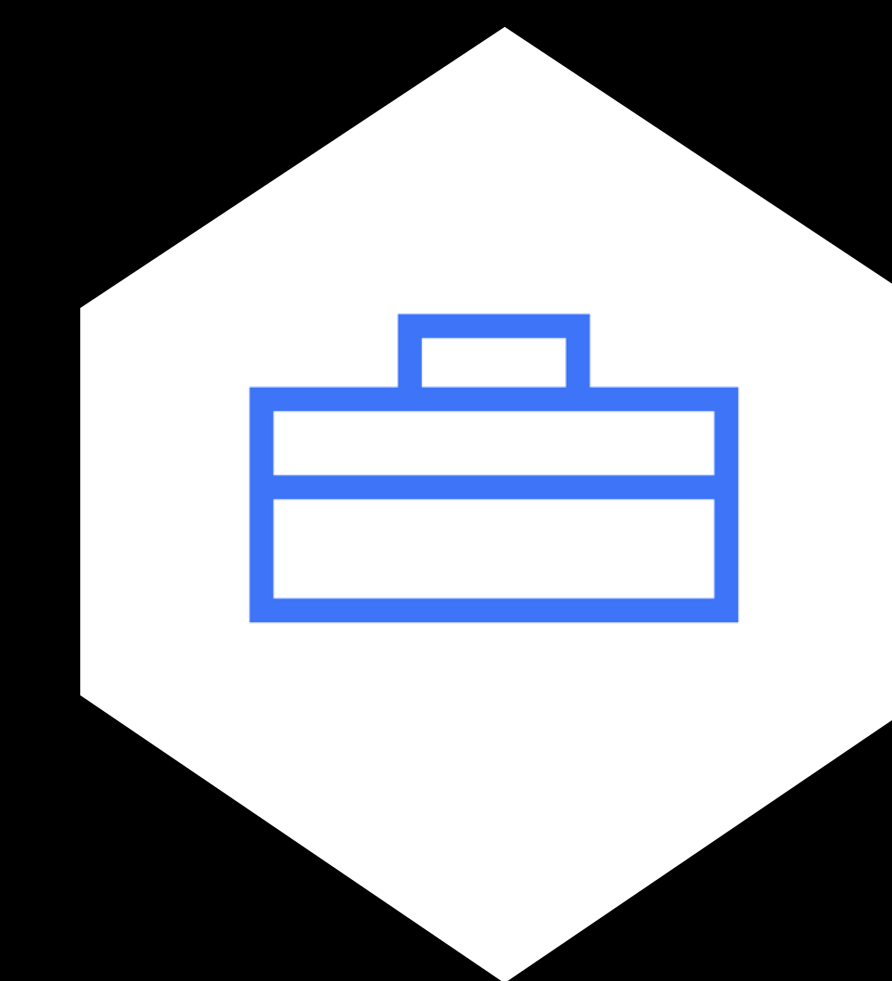
# Xcode's Targets and Dependencies

Target specifies a product to build

- iOS App
- Framework
- Unit Tests



Game



Framework



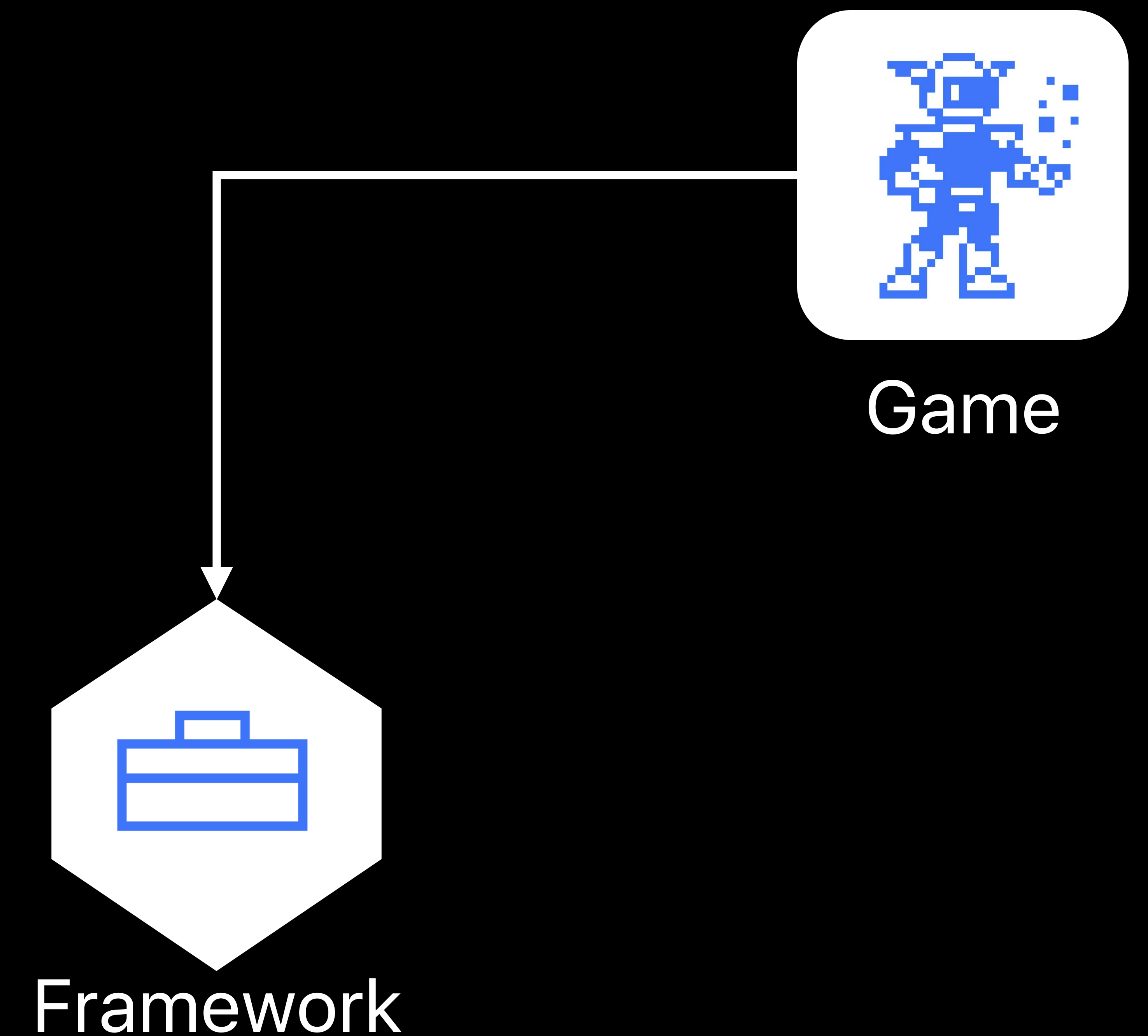
# Xcode's Targets and Dependencies

Target specifies a product to build

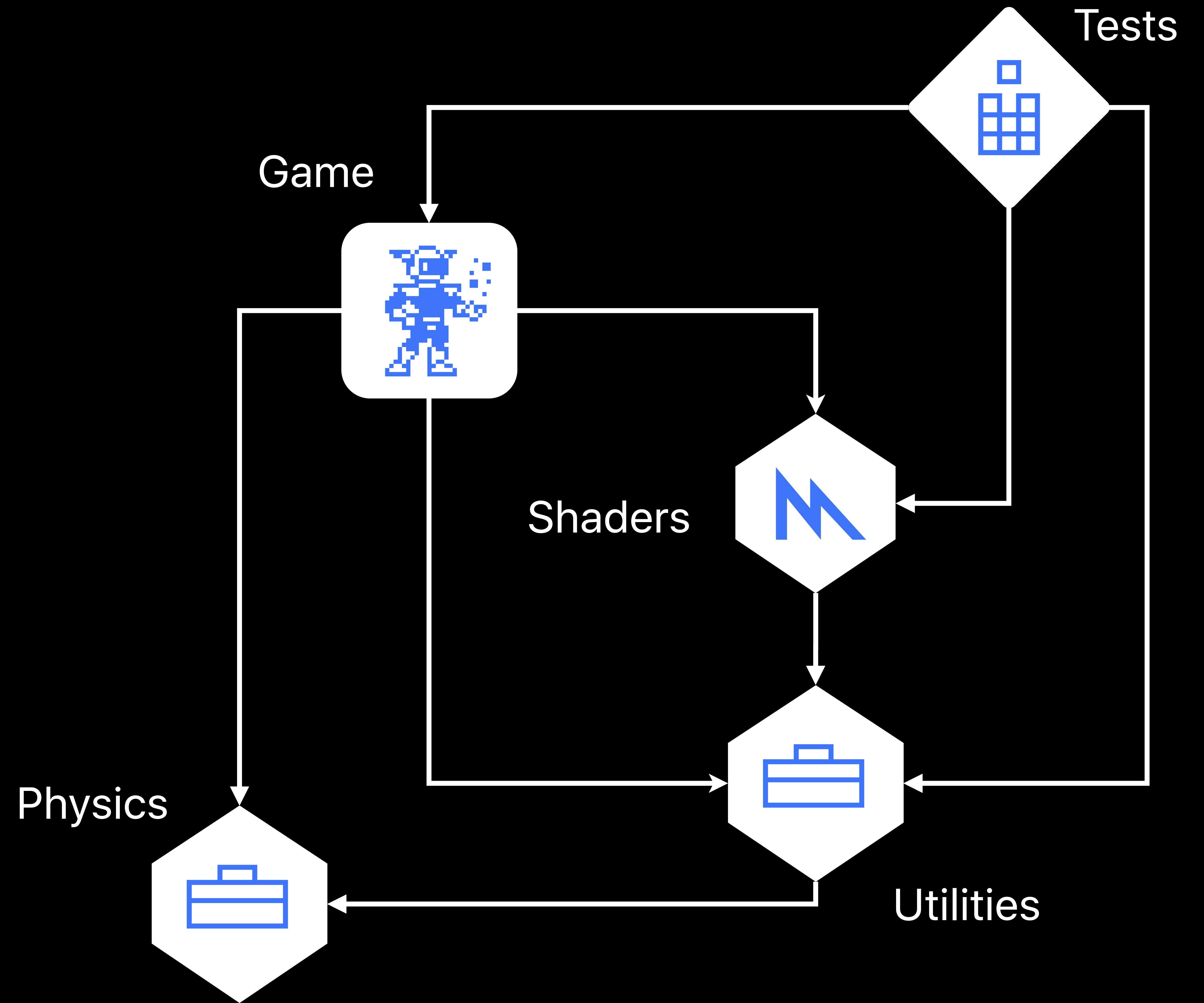
- iOS App
- Framework
- Unit Tests

Target dependency requires another target

- Explicit via Target Dependencies
- Implicit via Link Binary with Libraries

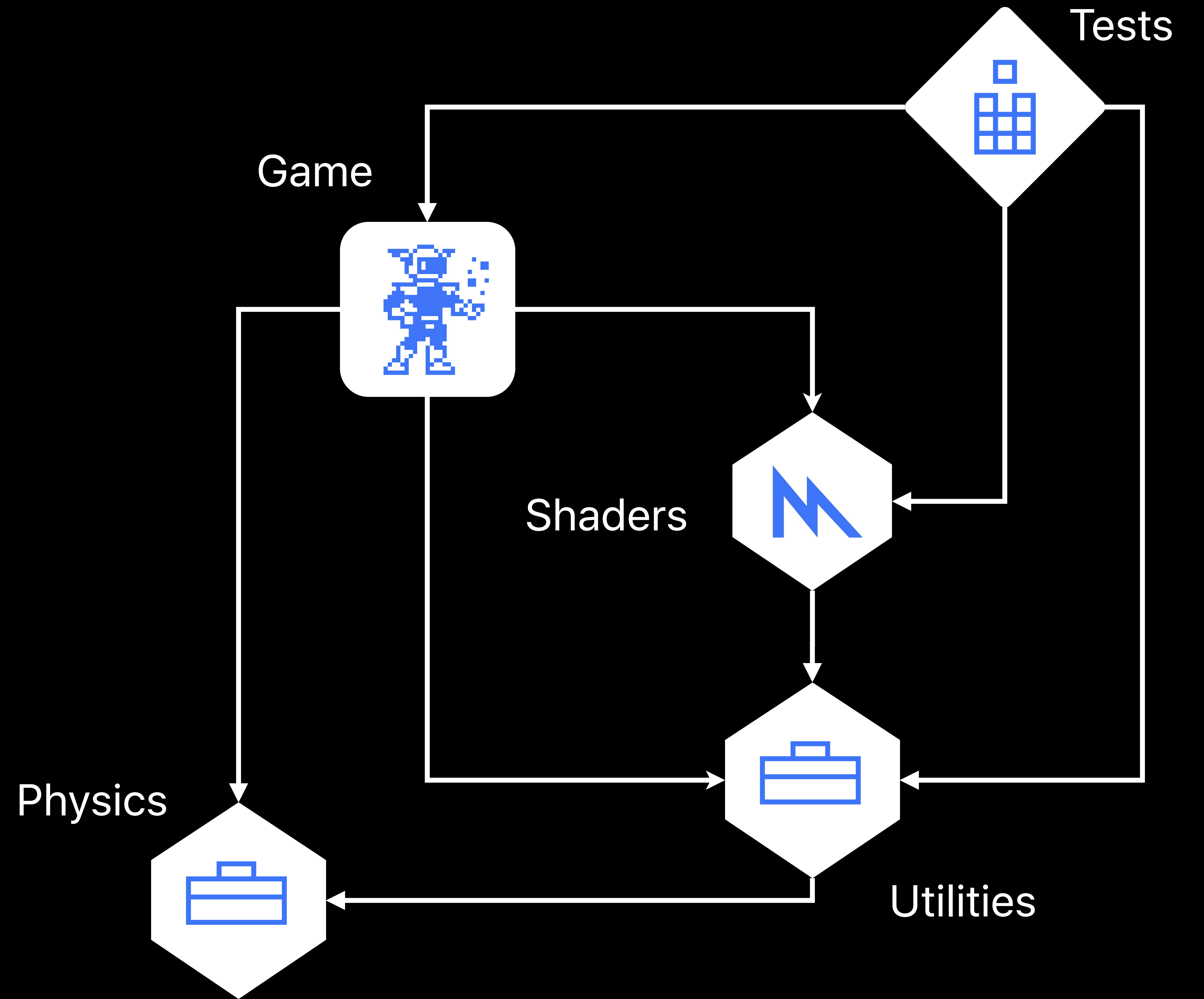


# Game Dependency Graph



# Game Dependency Graph

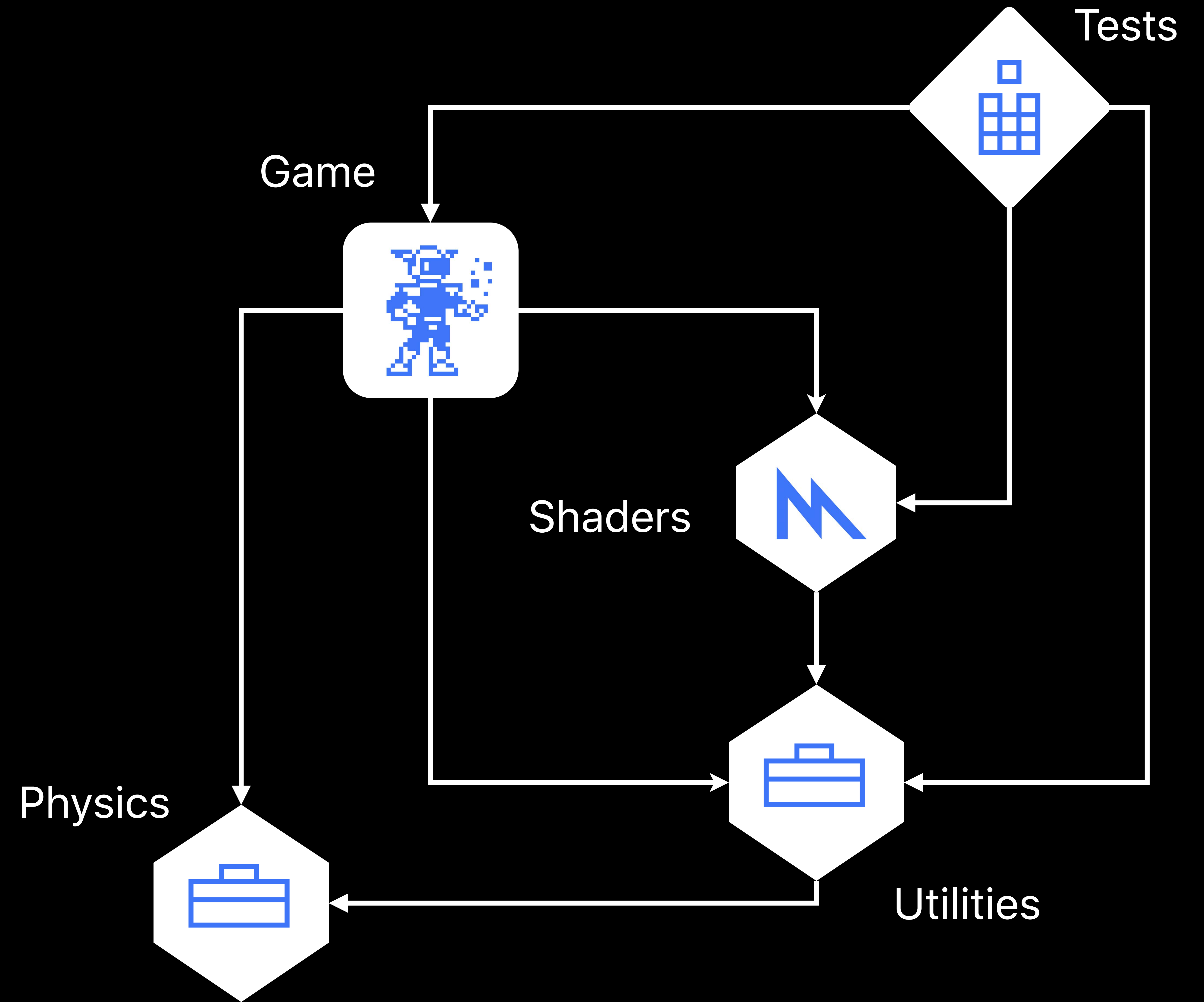
List of all targets to build



# Game Dependency Graph

List of all targets to build

Dependency between targets

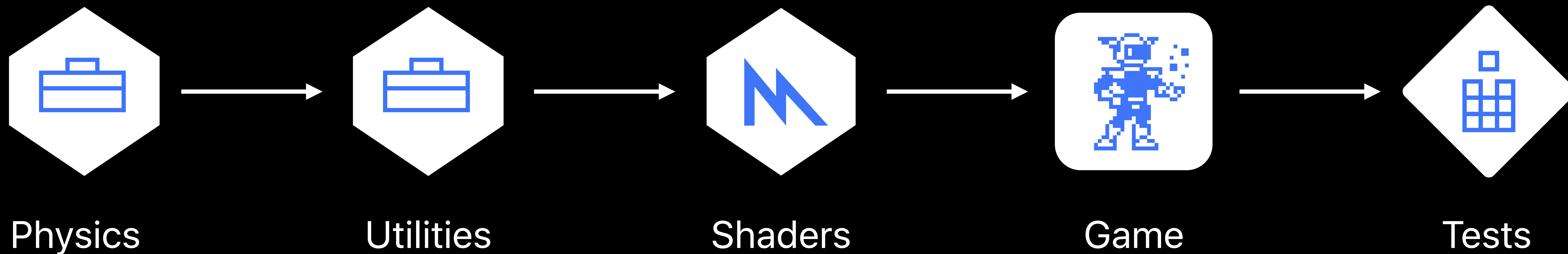


# Game Dependency Graph

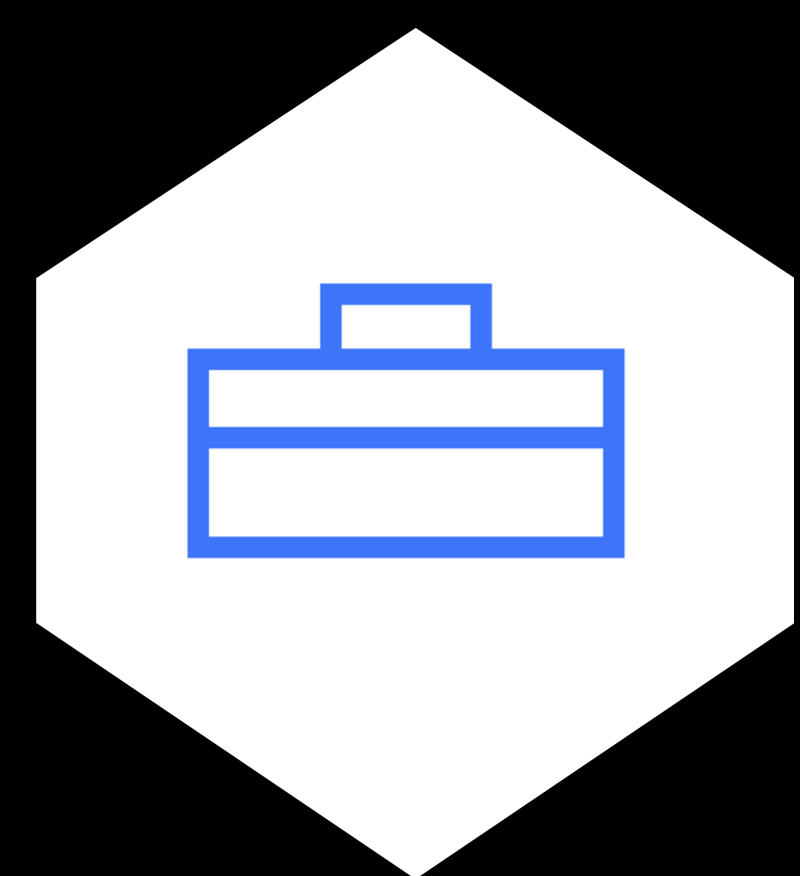
List of all targets to build

Dependency between targets

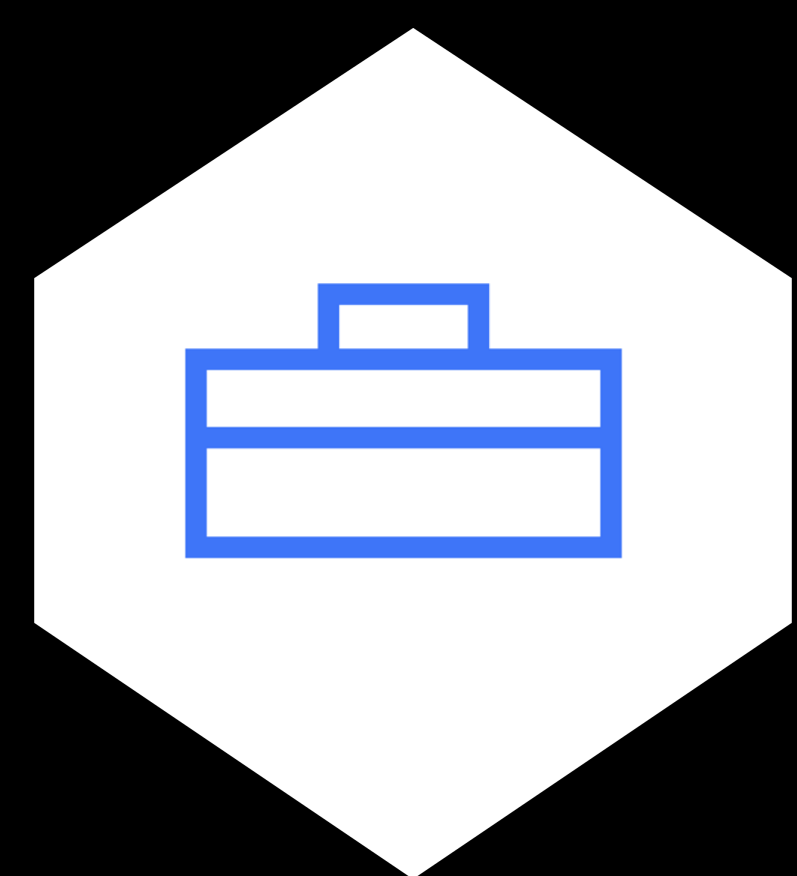
Build order can be derived



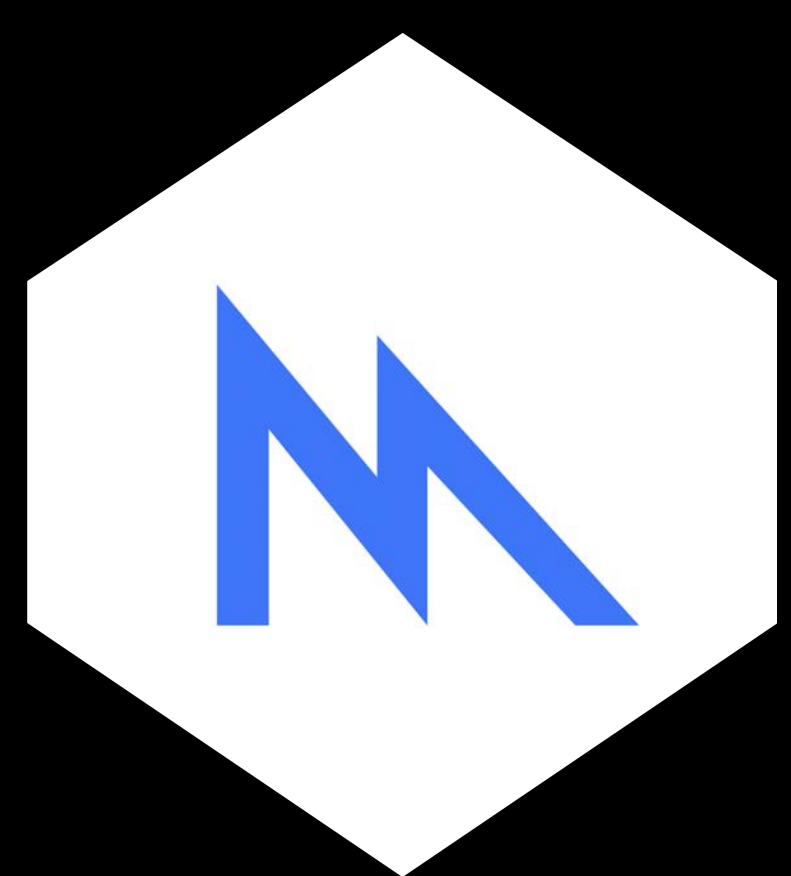
# Serialized Build Timeline



Physics



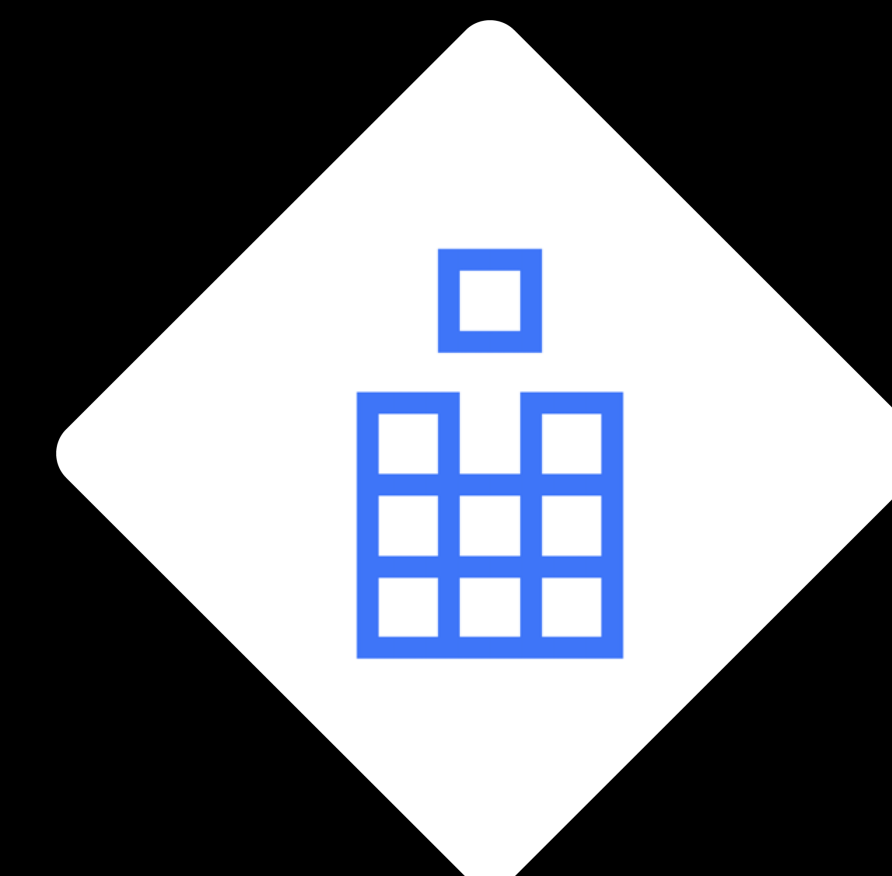
Utilities



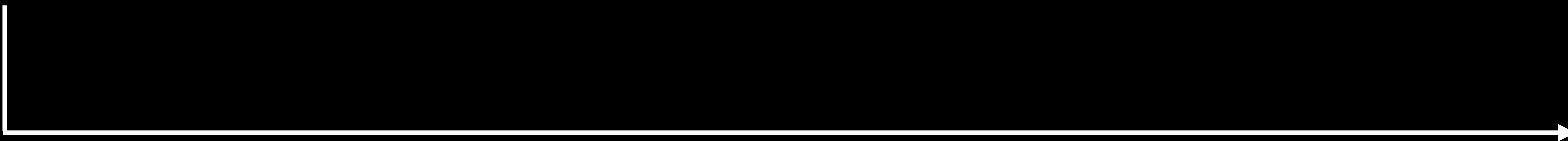
Shaders



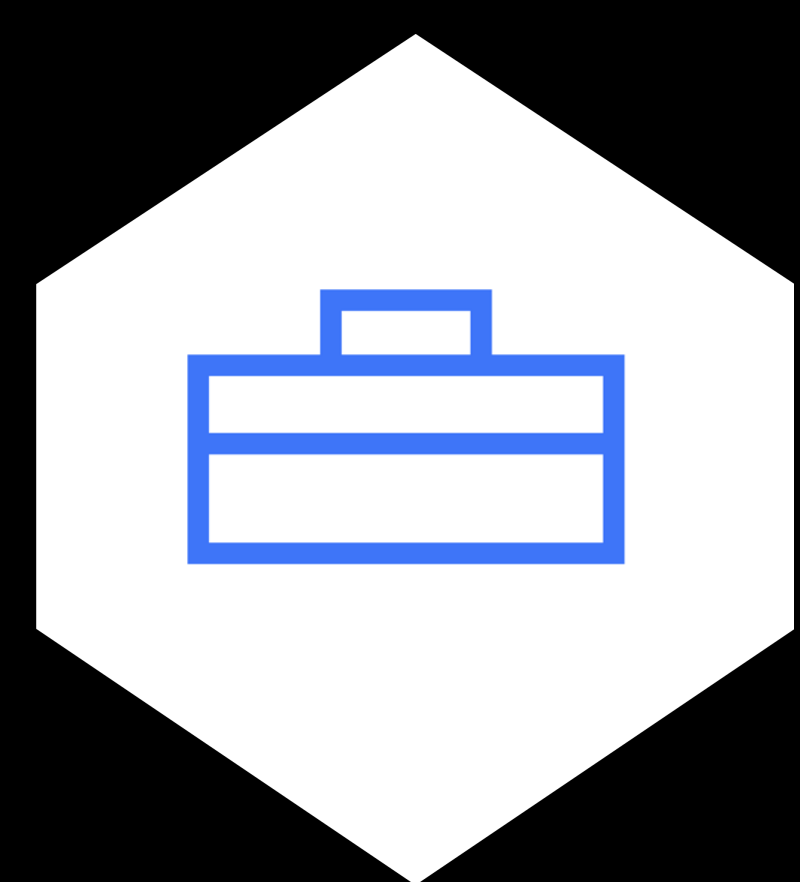
Game



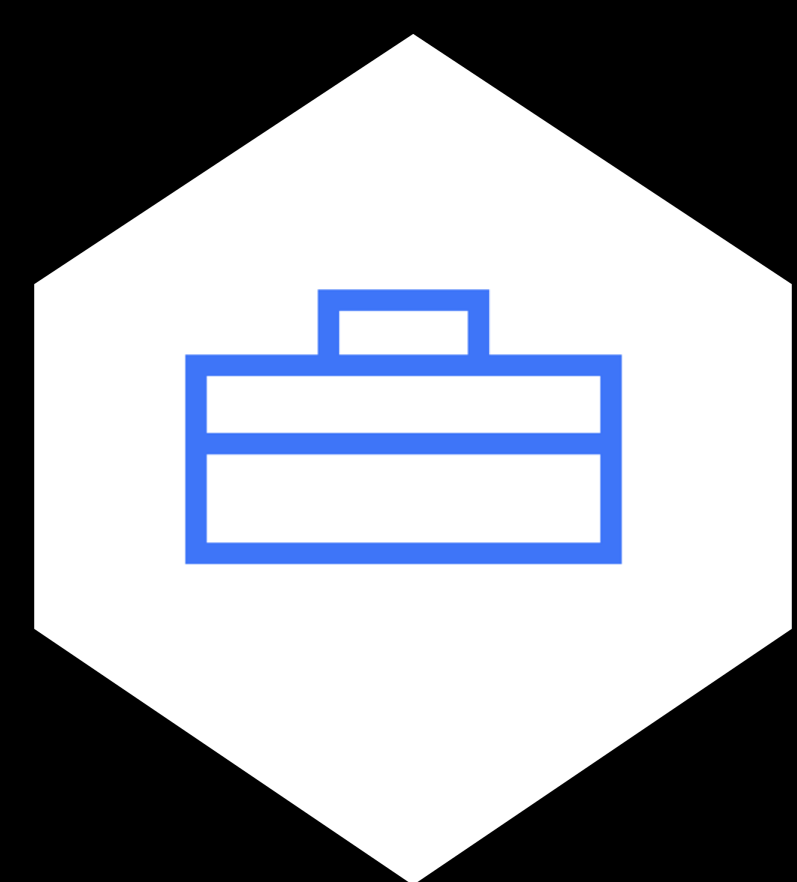
Tests



# Serialized Build Timeline



Physics



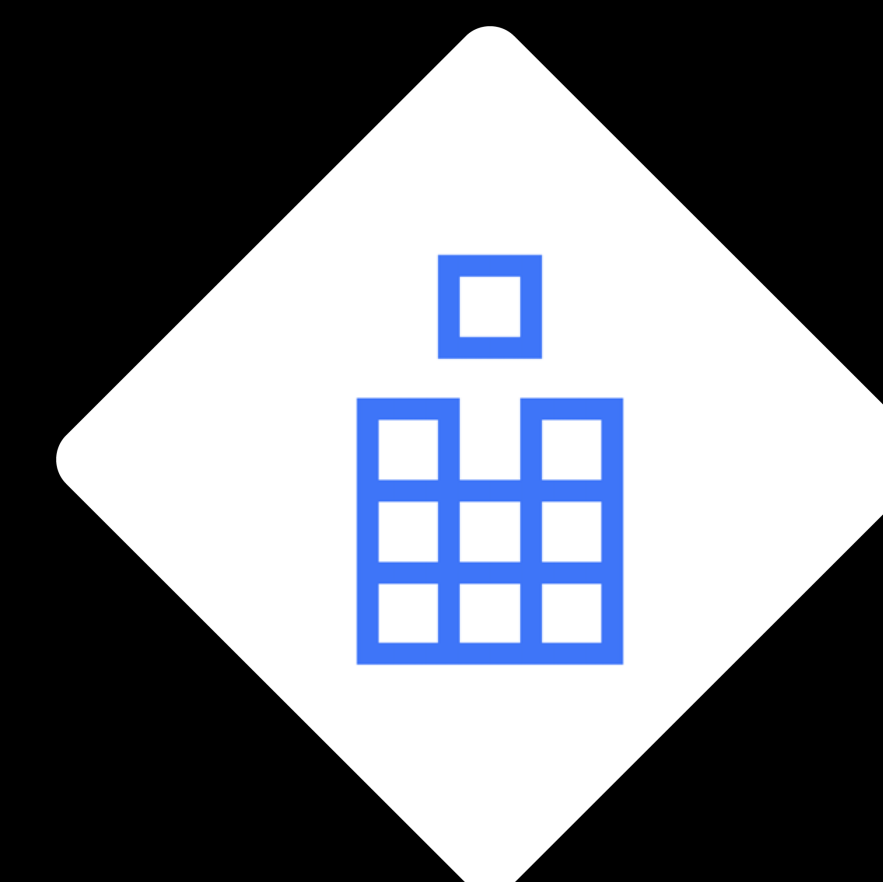
Utilities



Shaders



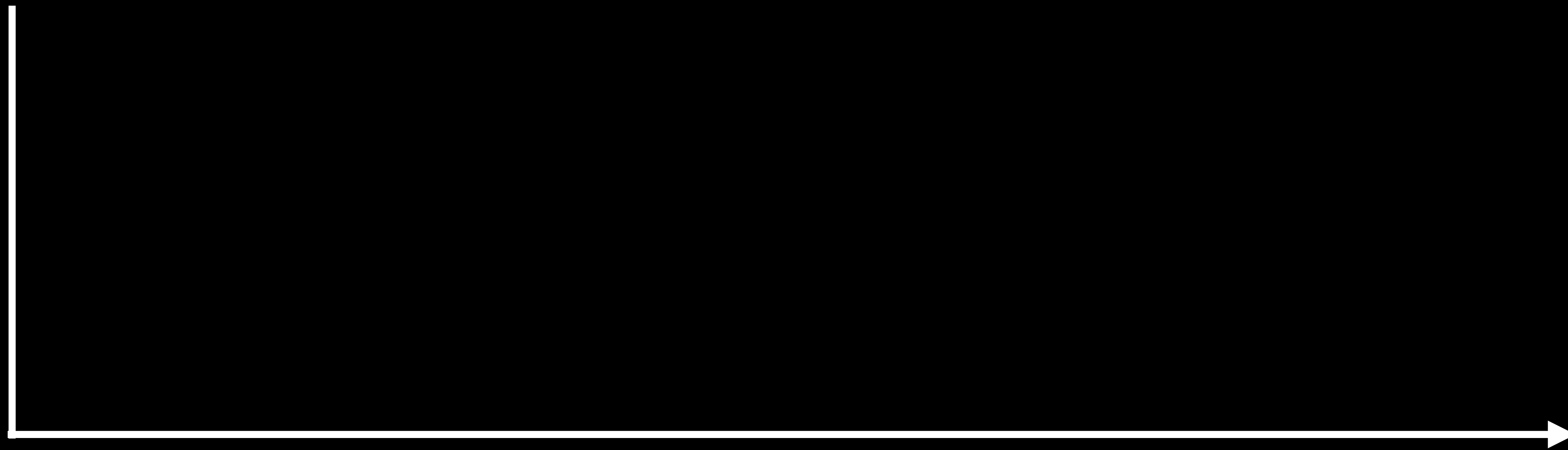
Game



Tests

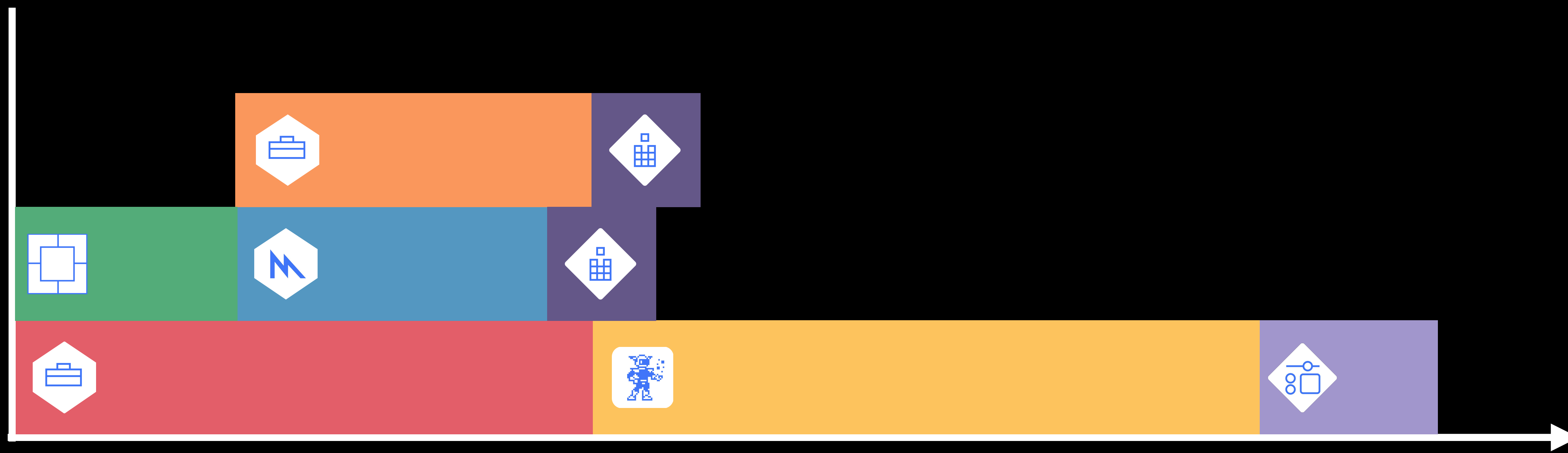


# Parallelized Build Timeline

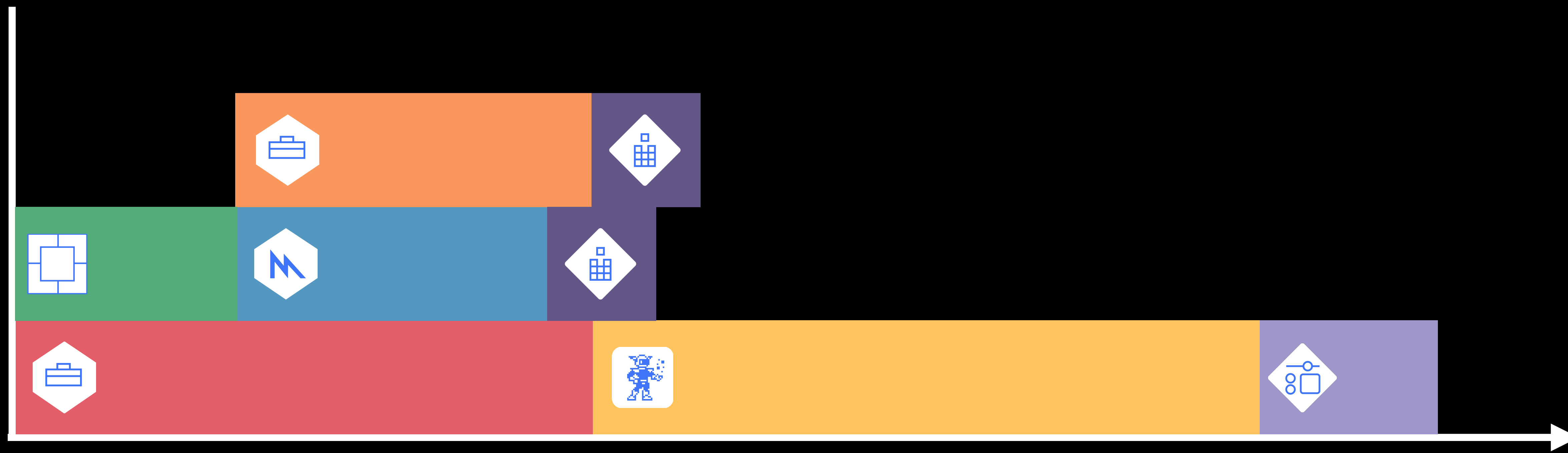




# Parallelized Build Timeline

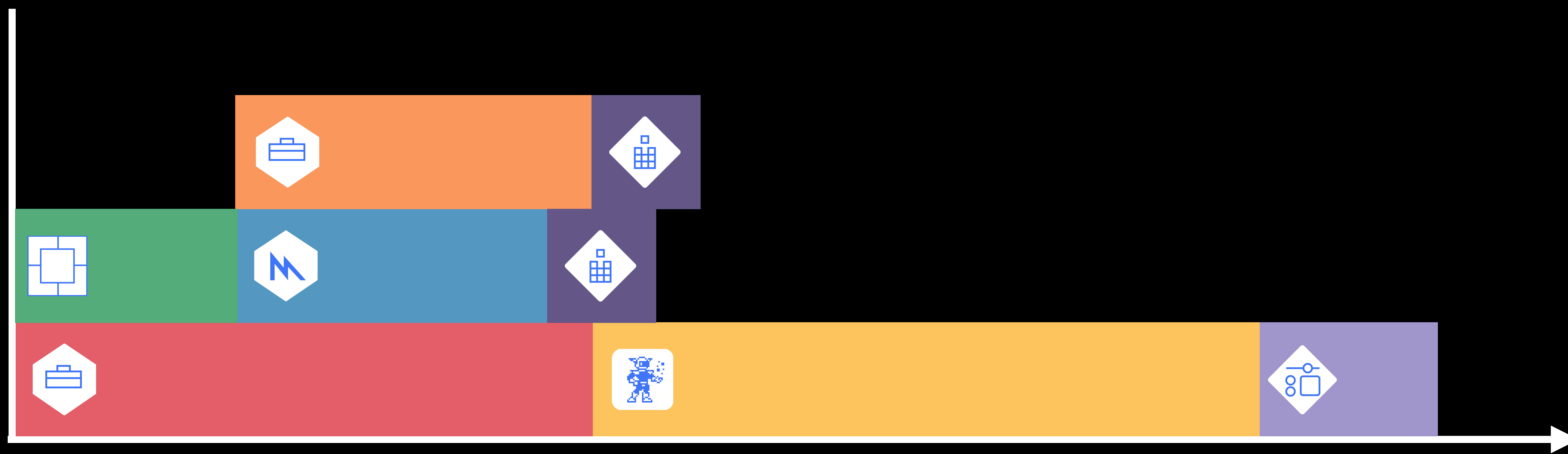


# Parallelized Build Timeline



Amount of work did not change

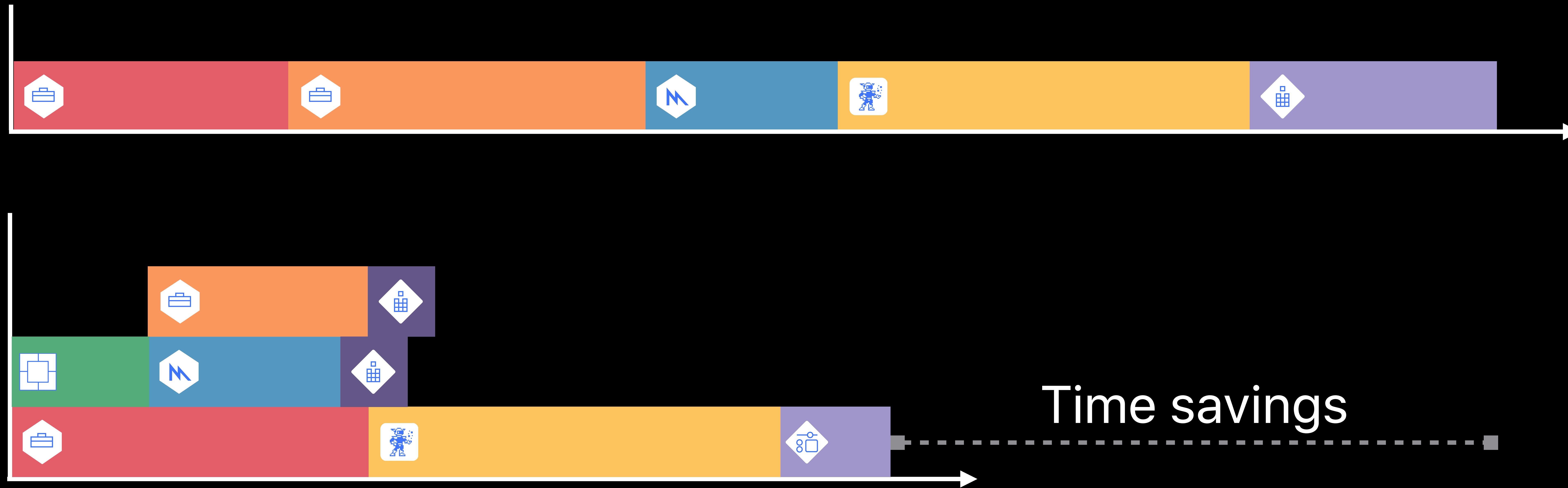
# Parallelized Build Timeline



Amount of work did not change

Time to build decreased

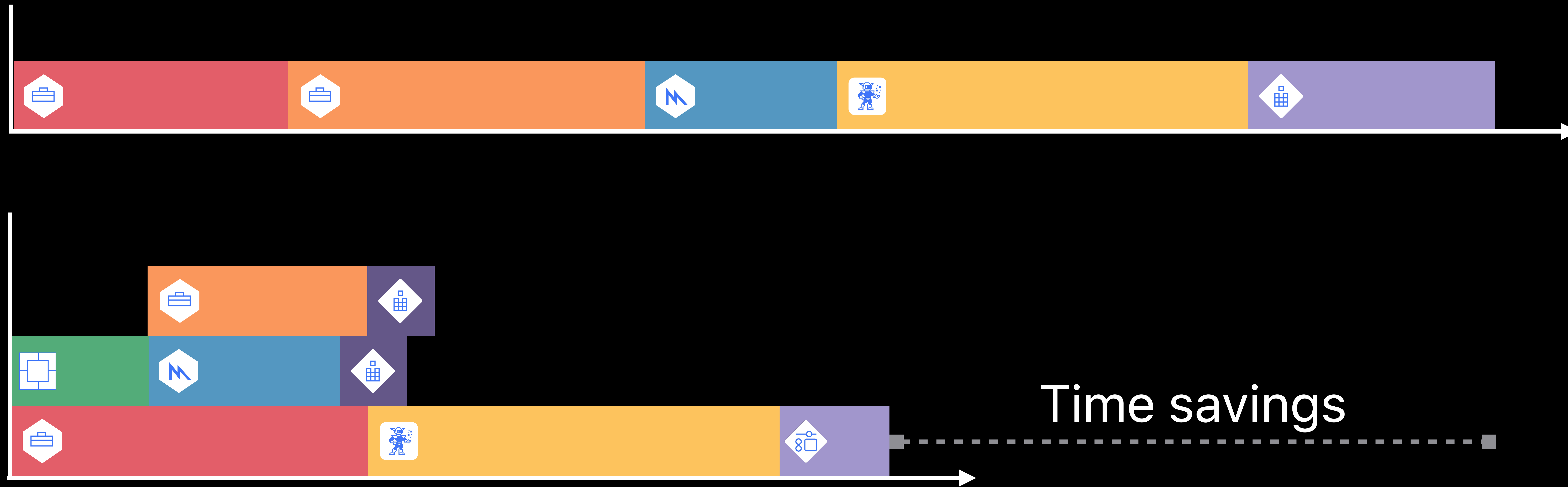
# Parallelized Build Timeline



Amount of work did not change

Time to build decreased

# Parallelized Build Timeline

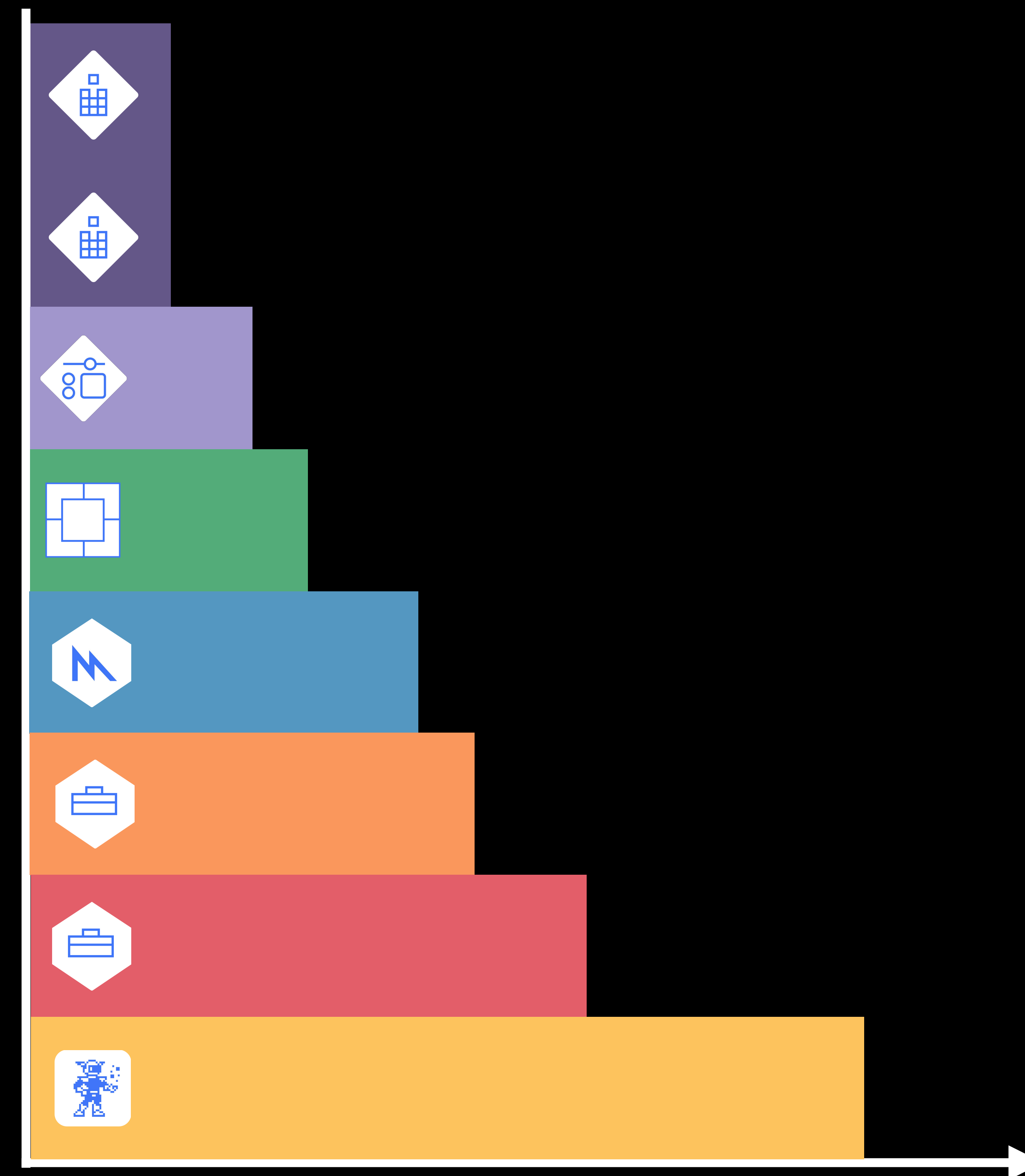


Amount of work did not change

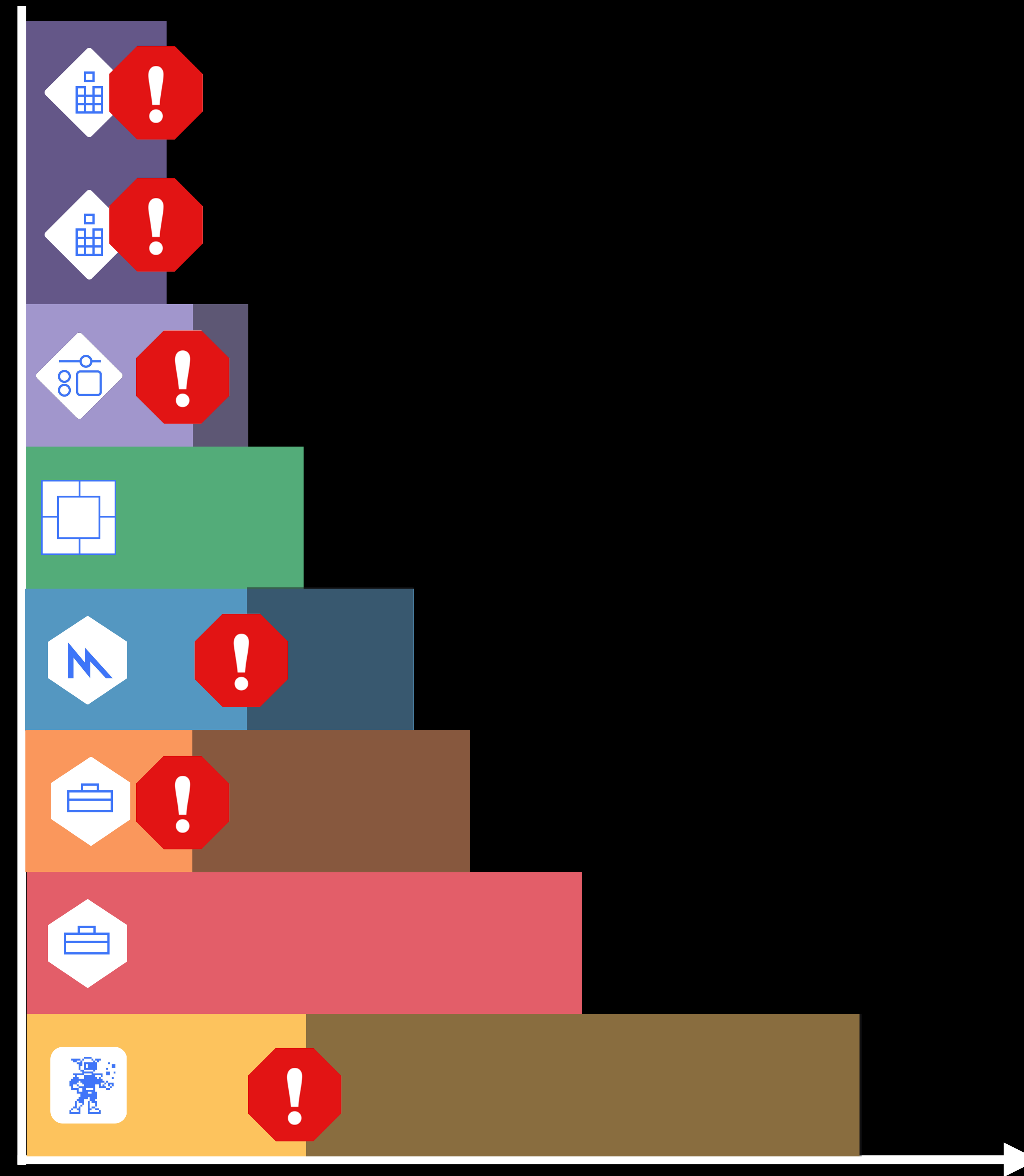
Time to build decreased

Increased hardware utilization

# Extreme Parallelization



# Extreme Parallelization



How do we get there?



Game Workspace | Build Game: Succeeded | Today at 9:33 PM

Game

- Shaders.xcod
- Assets
- Game
- Scripts
- Tests
- Utilities
- Products
- Frameworks
- Physics

- Game
- Utilities
- Shaders
- Algebra
- Physics
- Calculus

Edit Scheme...  
New Scheme...  
Manage Schemes...

Game Workspace | Build Game: Succeeded | Today at 9:33 PM

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

Filter

Target Dependencies (1 item)

- Shaders (Shaders)

Compile Sources (10 items)

Link Binary With Libraries (2 items)

Name	Status
Physics.framework	Required
Utilities.framework	Required

Copy Bundle Resources (2 items)

Embed Frameworks (2 items)

Generate Game Score Set

Filter

Game > My Mac

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
  - Physics

PROJECT

- Game

TARGETS

- Game
- Utilities
- Tests

Game > My Mac

**Build**  
2 targets

Build Options  Parallelize Build  
 Find Implicit Dependencies

Targets	Analyze	Test	Run	Profile	Archive
▶ Game	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
▶ Tests	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

+ - Filter

Duplicate Scheme Manage Schemes...  Shared Close

les

Filter

Status

Required

Required

Filter

Game

- Shaders.xcodeproj
- Assets
- Game
- Scripts
- Tests
- Utilities
- Products
- Frameworks
- Physics

PROJECT

- Game

TARGETS

- Game
- Utilities
- Tests

Game

- Build (2 targets)
- Run (Debug)
- Test (Debug)
- Profile (Release)
- Analyze (Debug)
- Archive (Release)

## Build Options

- Parallelize Build
- Find Implicit Dependencies

Target	Parallelize Build	Find Implicit Dependencies
Game	<input type="checkbox"/>	<input type="checkbox"/>
Tests	<input checked="" type="checkbox"/>	<input type="checkbox"/>

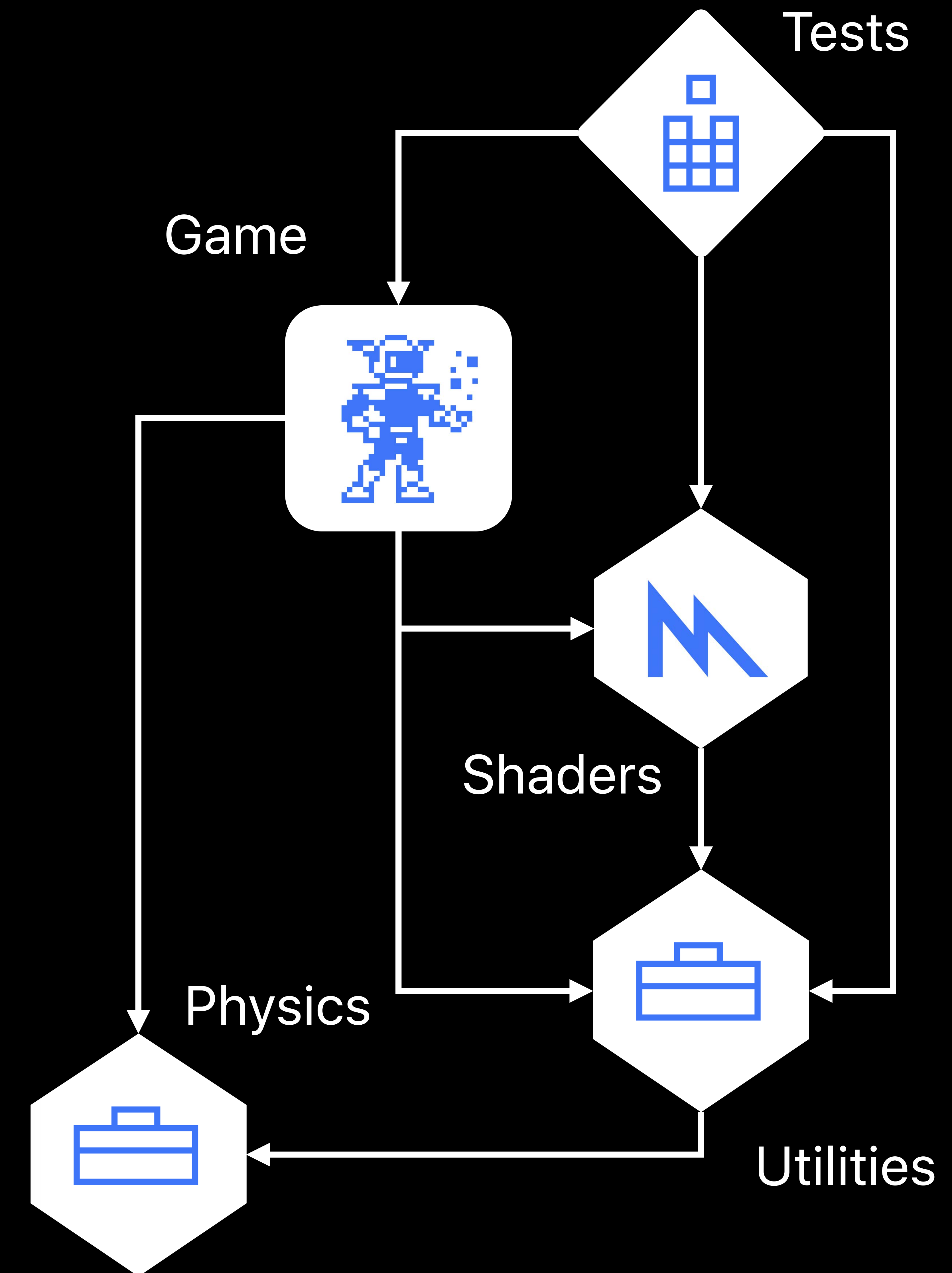
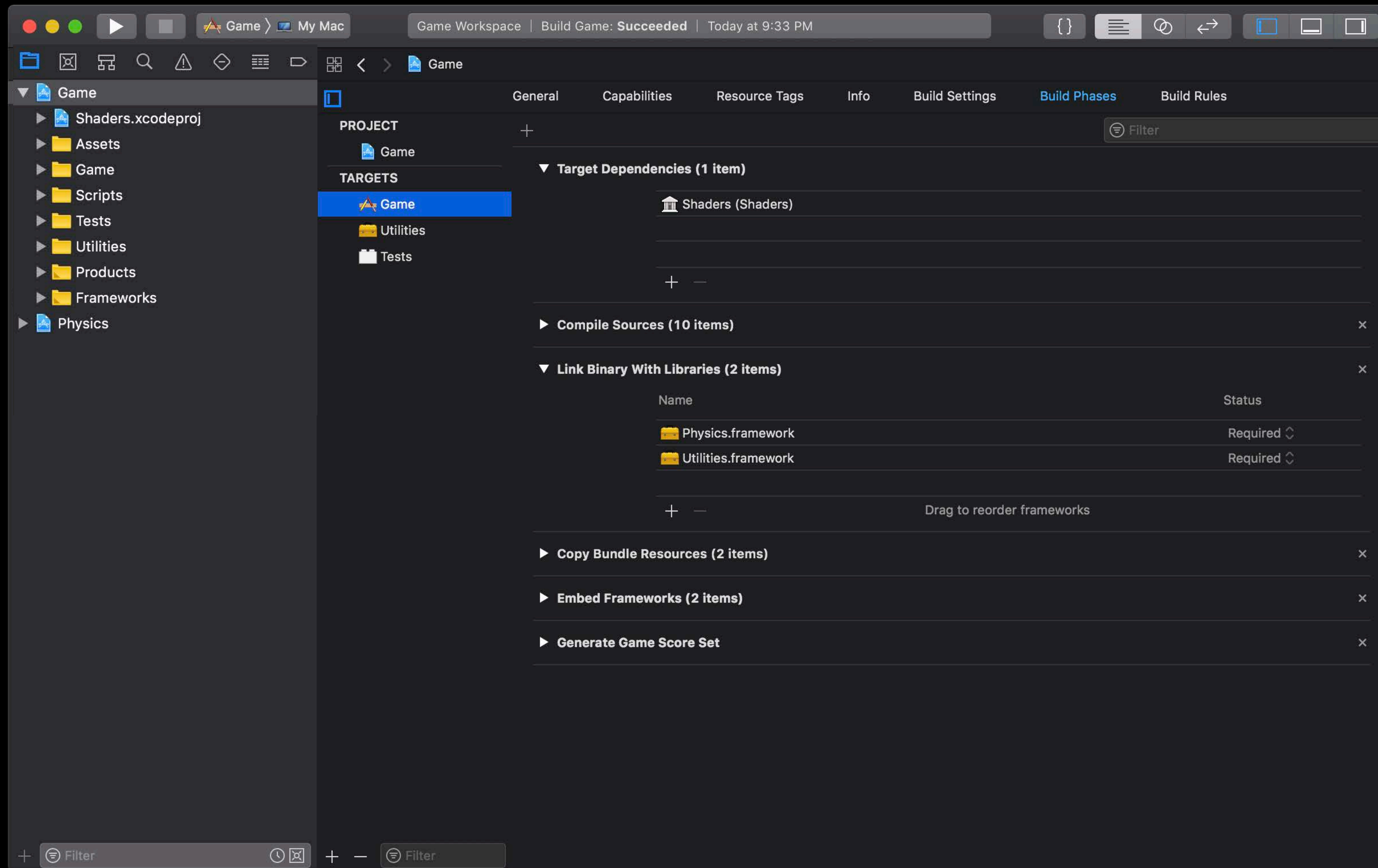
Shared

les

Filter

Status
Required
Required

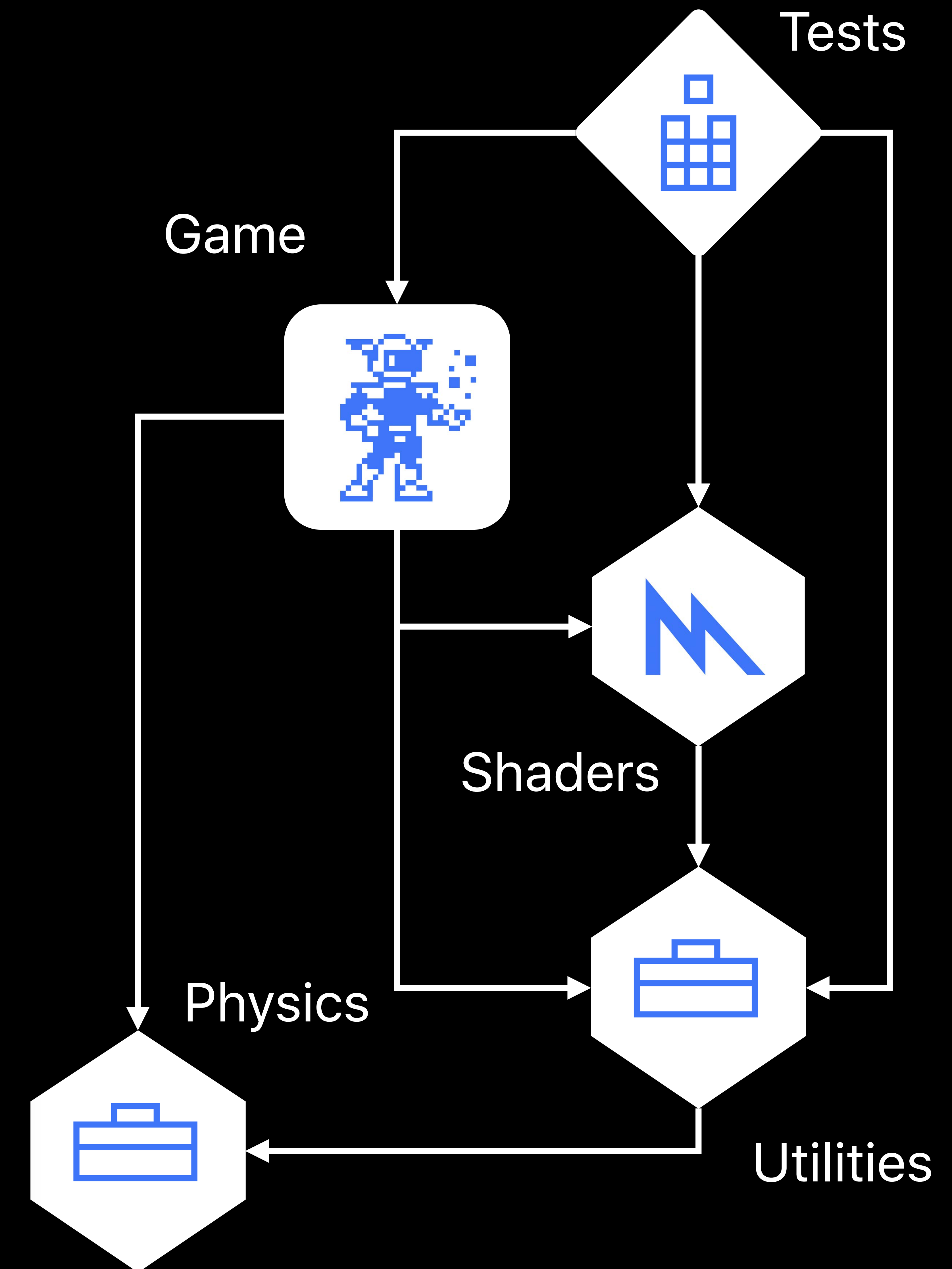
# Project Configuration



# Project Configuration

The screenshot shows the Xcode interface for configuring a game target. The left sidebar displays a project tree with folders: Game, Shaders.xcodeproj, Assets, Game, Scripts, Tests, Utilities, Products, Frameworks, and Physics. The main pane shows the 'Build Phases' tab for the 'Game' target. The build phases are as follows:

- Target Dependencies (1 item): Shaders (Shaders)
- Compile Sources (10 items)
- Link Binary With Libraries (2 items): Physics.framework (Required), Utilities.framework (Required)
- Copy Bundle Resources (2 items)
- Embed Frameworks (2 items)
- Generate Game Score Set



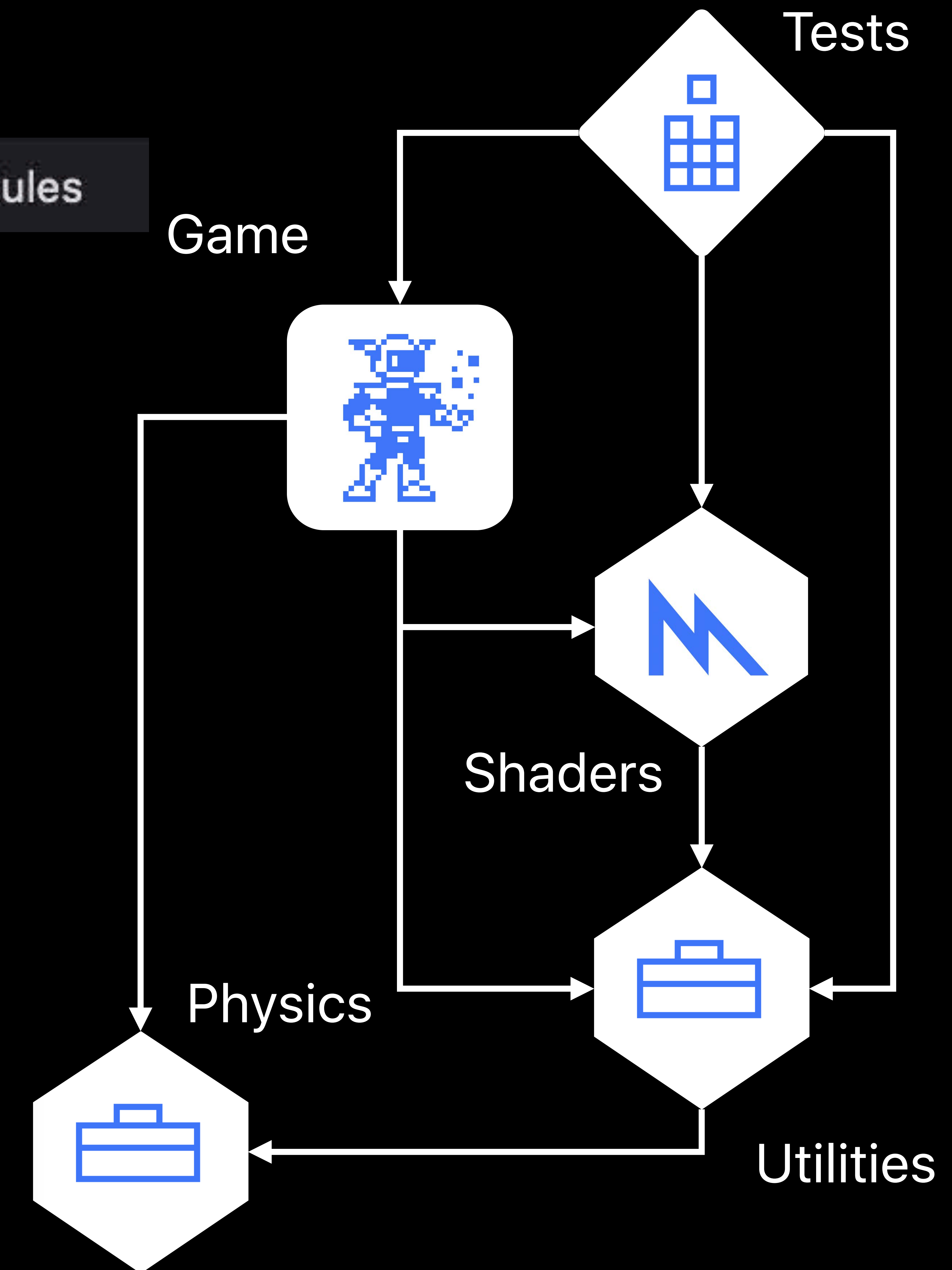
# Project Configuration

The screenshot shows the Xcode interface for configuring a project. On the left is a file browser with the following structure:

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
- Physics

The main area displays the 'Build Phases' tab for a target named 'Game'. The build phases are:

- Target Dependencies (1 item): Shaders (Shaders)
- Compile Sources (10 items)
- Link Binary With Libraries (2 items): Physics.framework (Required), Utilities.framework (Required)
- Copy Bundle Resources (2 items)
- Embed Frameworks (2 items)
- Generate Game Score Set



# Game Target Dependencies

The screenshot shows the Xcode interface for a project named 'Game'. The 'Build Phases' tab is selected, displaying the 'Target Dependencies' section for the 'Game' target. The dependencies are as follows:

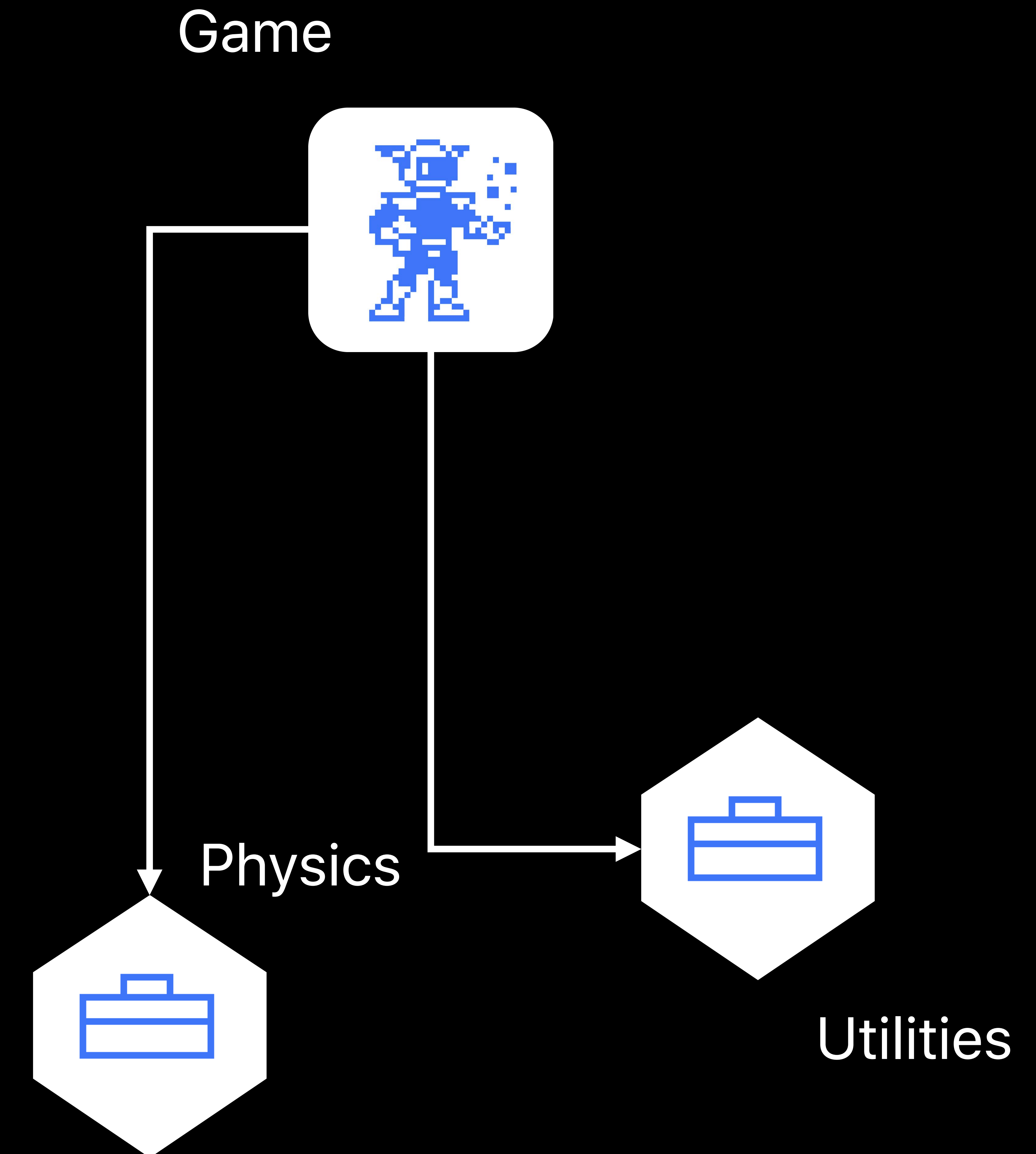
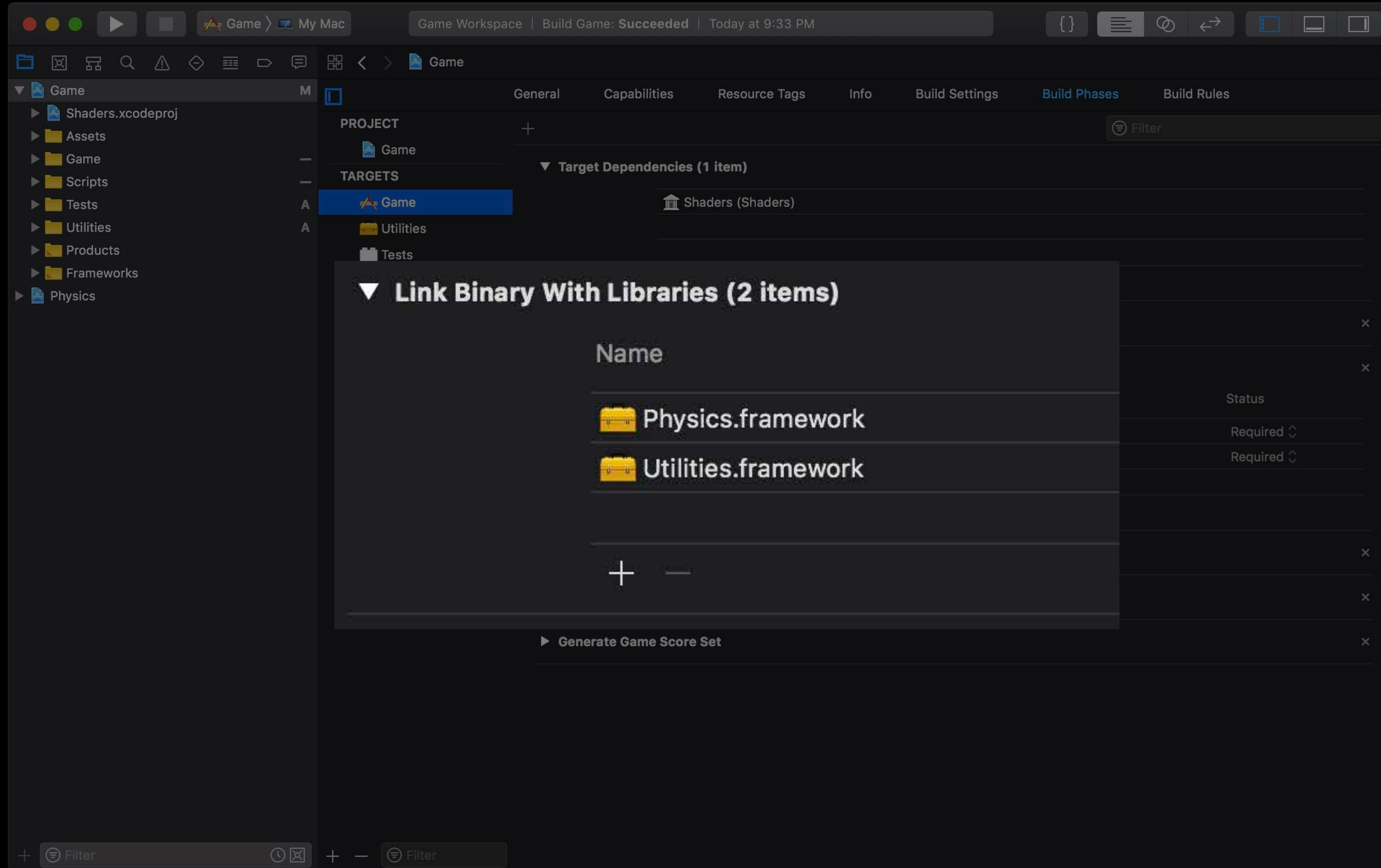
- Target Dependencies (1 item):**
  - Shaders (Shaders)
- Compile Sources (10 items)**
- Link Binary With Libraries (2 items):**

Name	Status
Physics.framework	Required
Utilities.framework	Required
- Copy Bundle Resources (2 items)**
- Embed Frameworks (2 items)**
- Generate Game Score Set**

Game

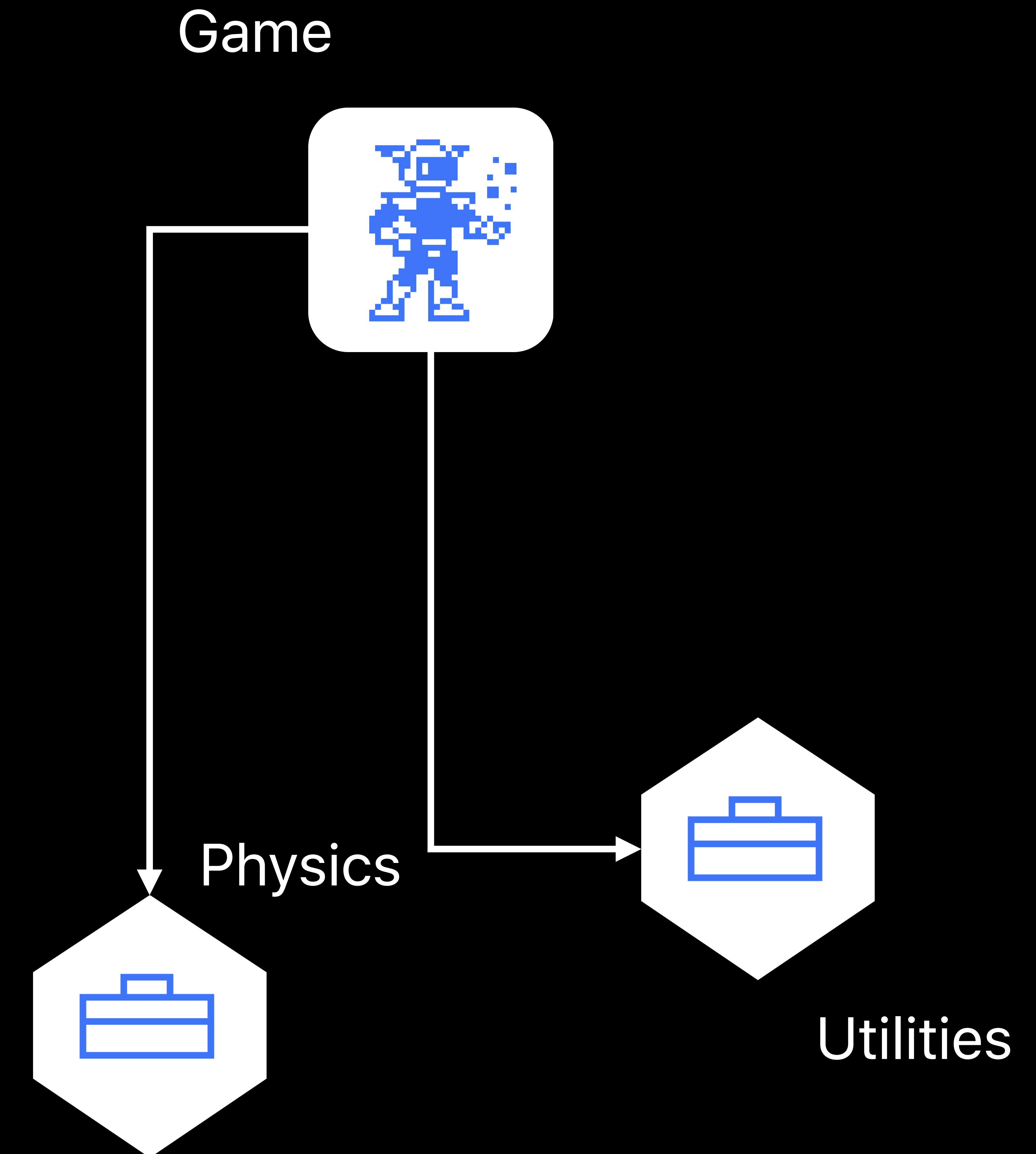
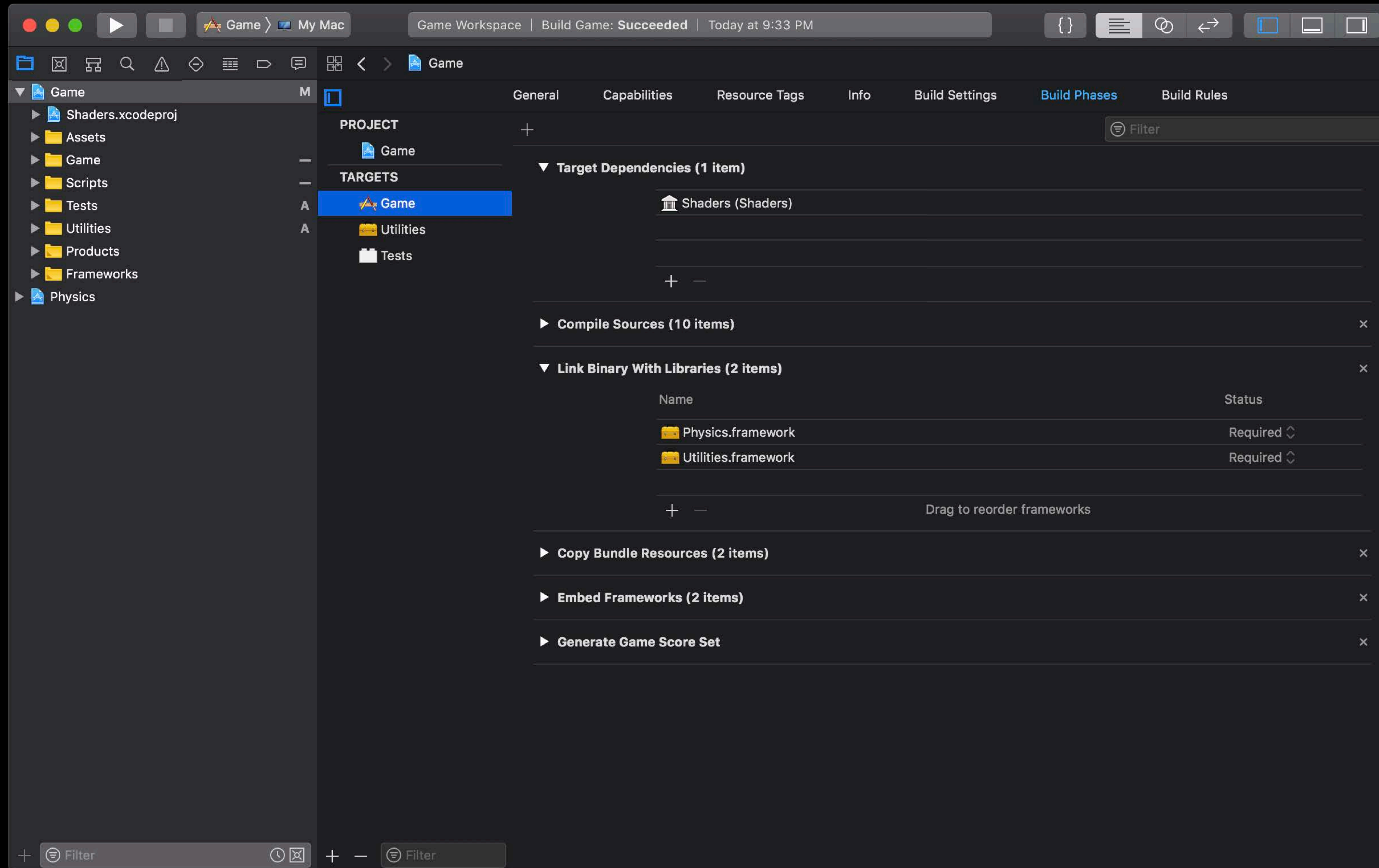


# Game Target Dependencies

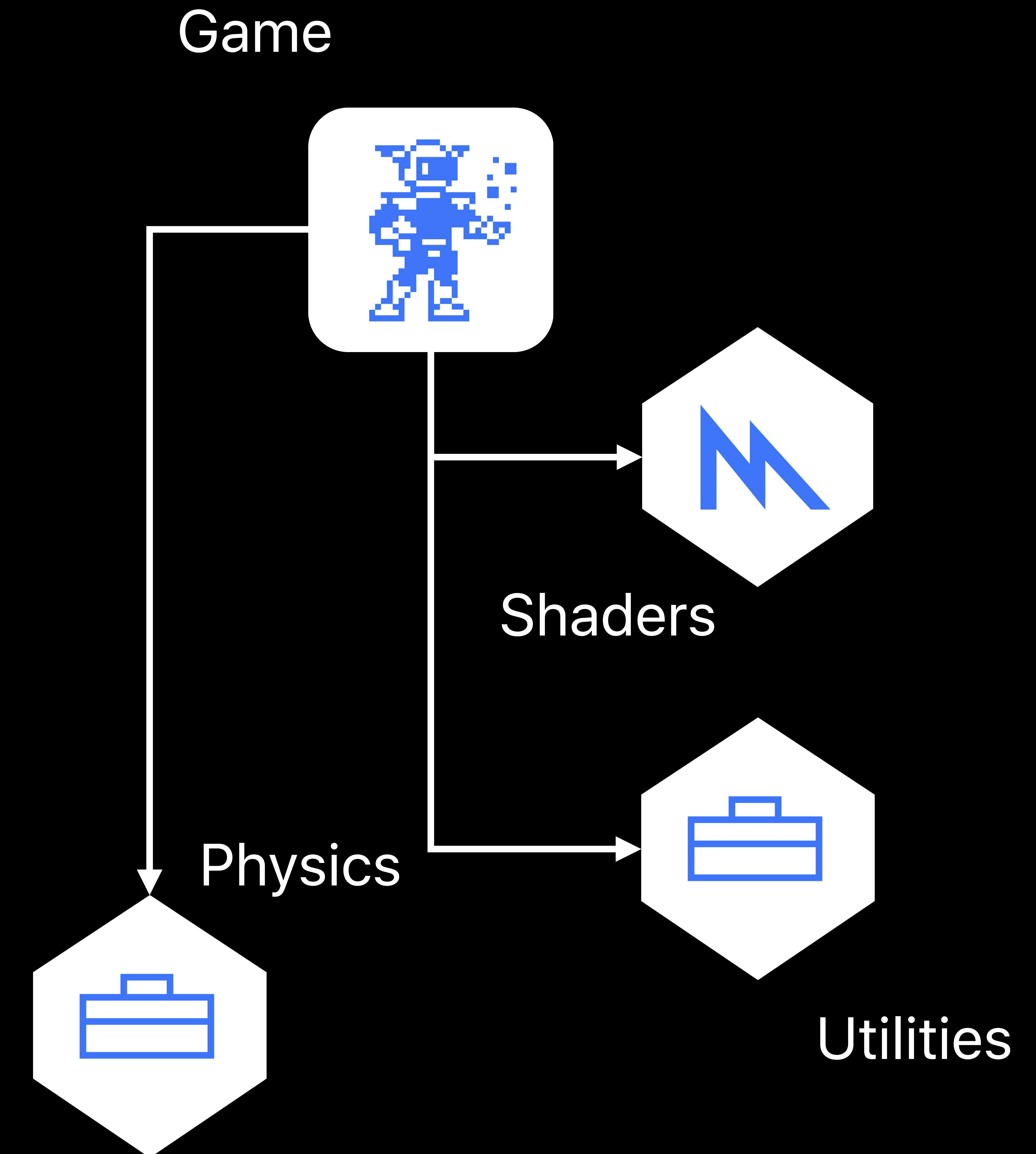
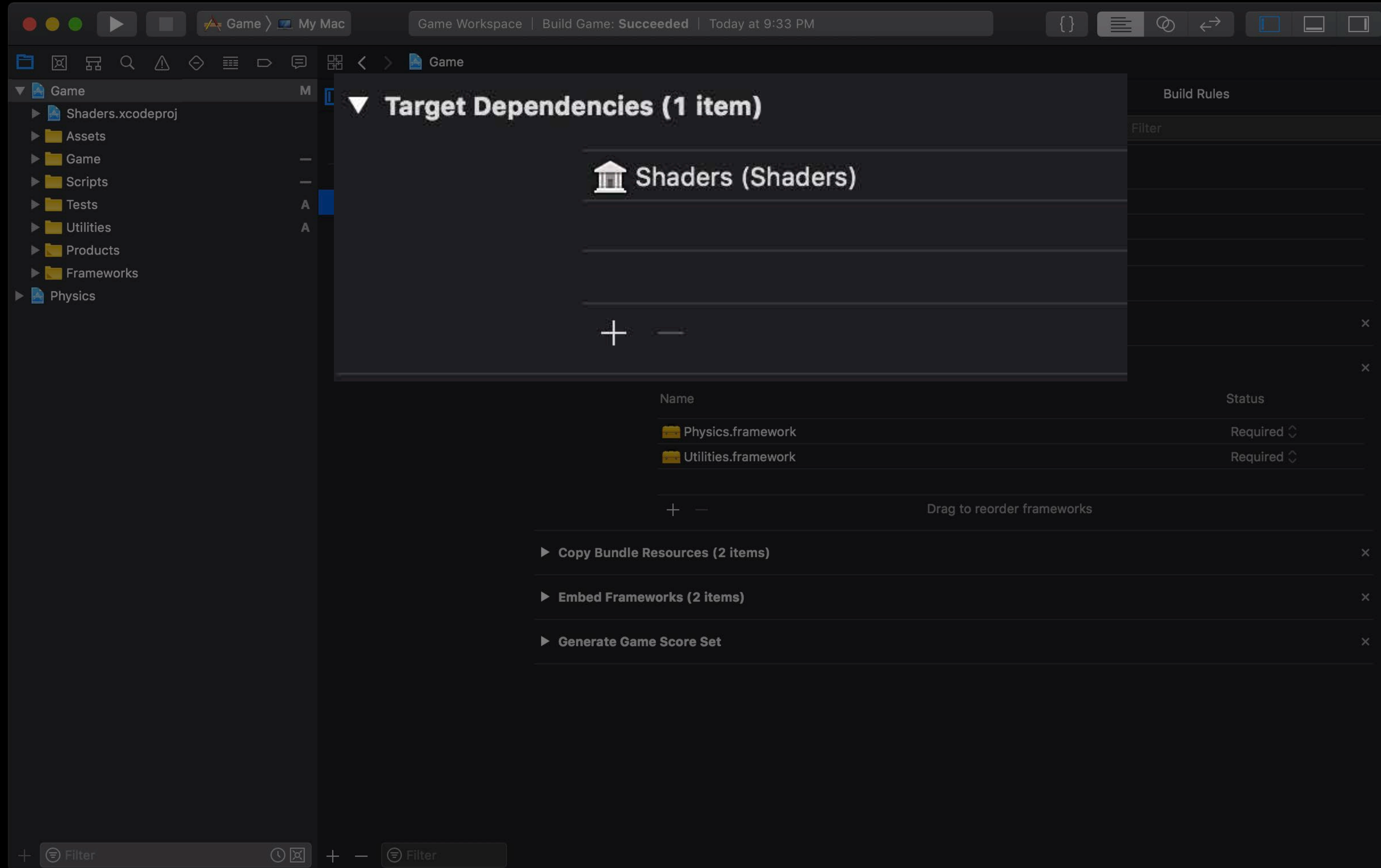




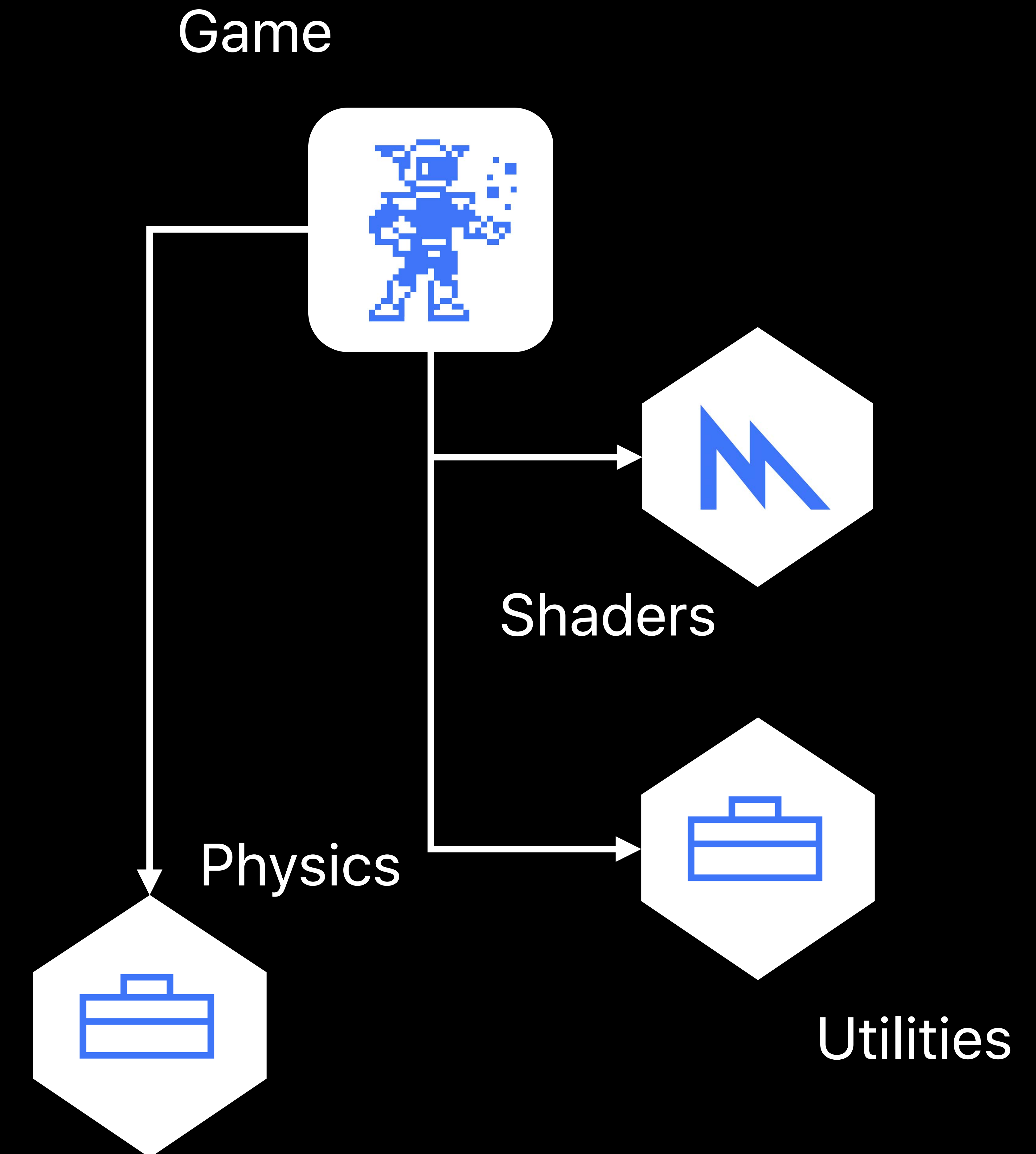
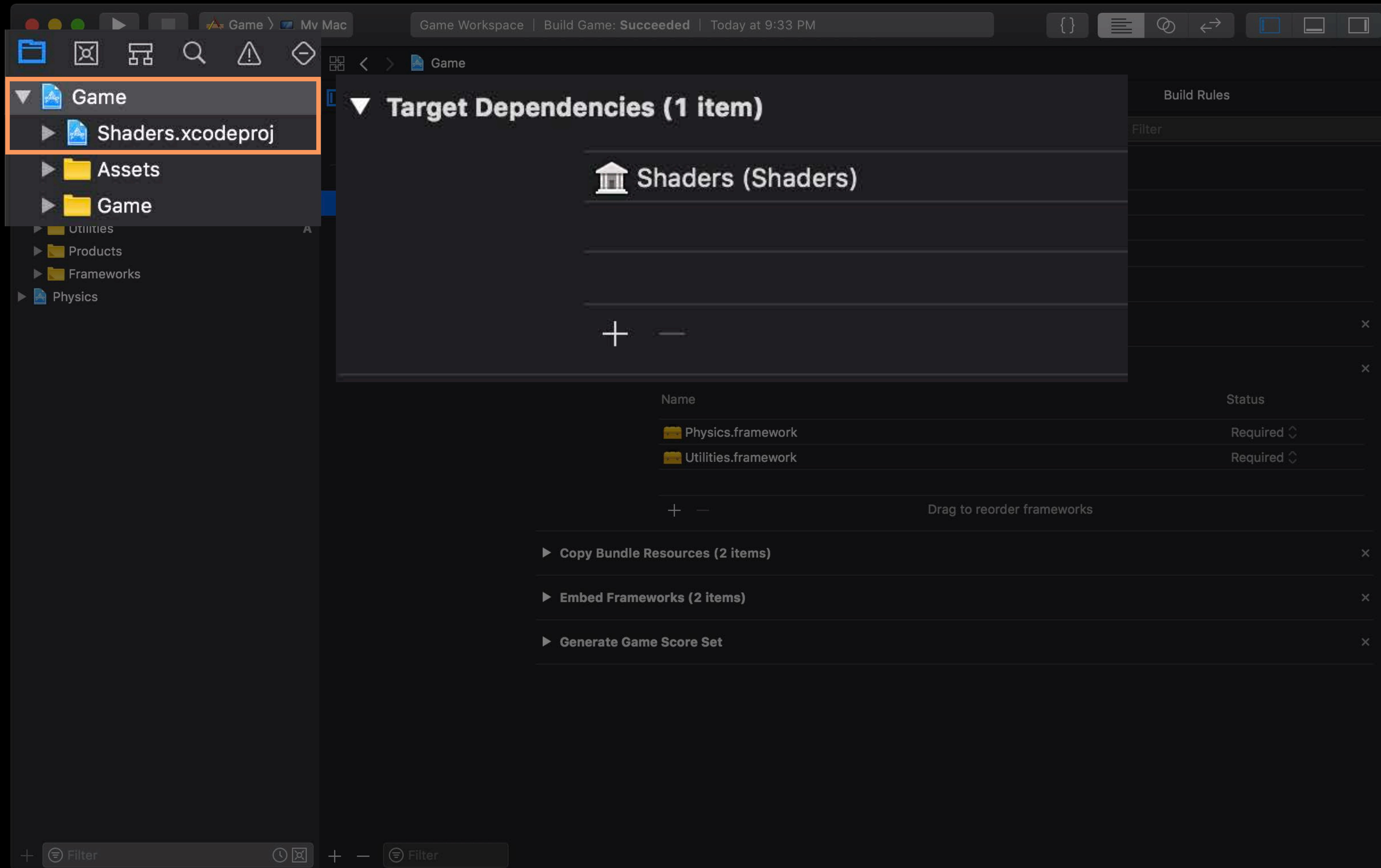
# Game Target Dependencies



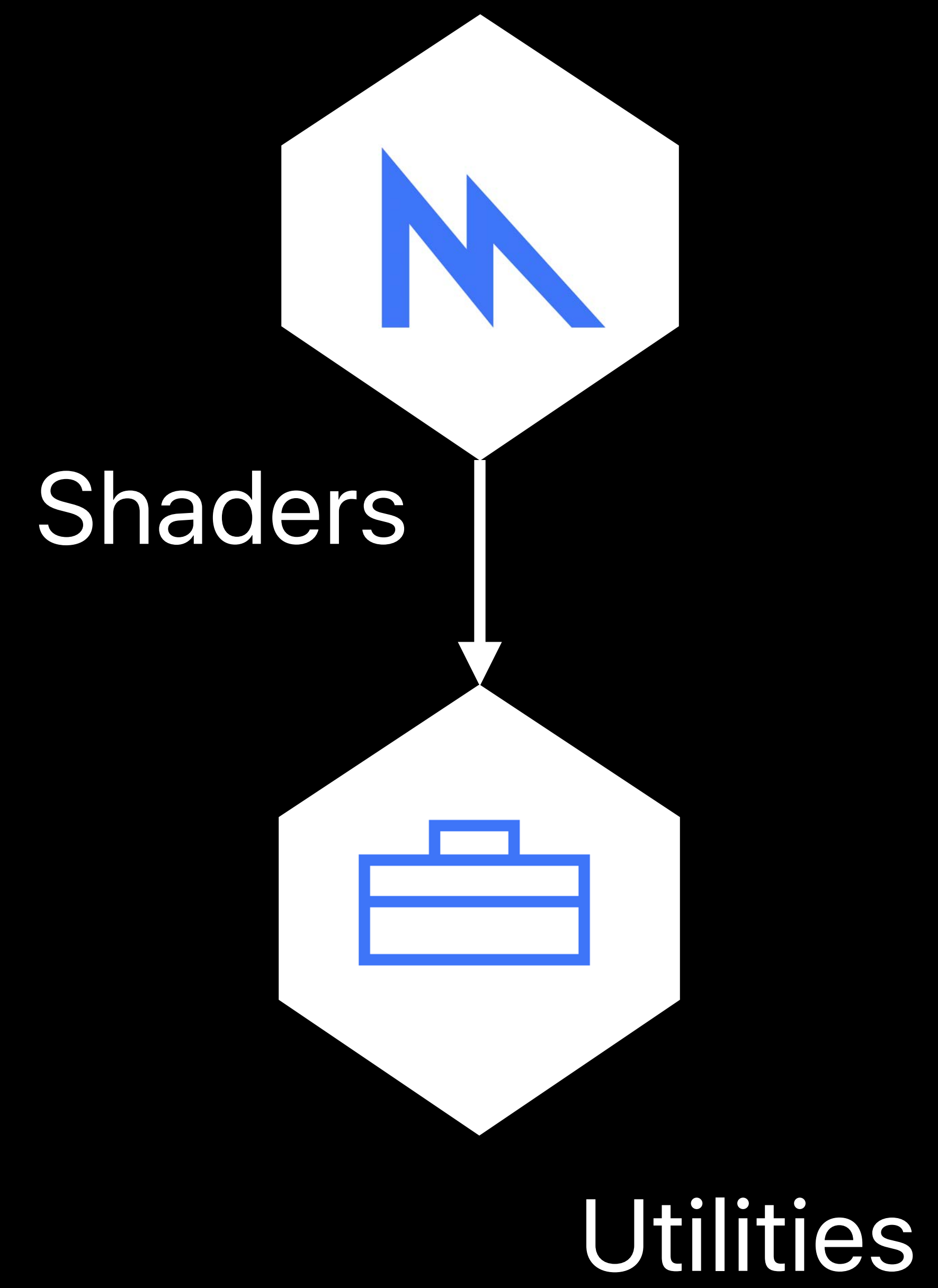
# Game Target Dependencies



# Game Target Dependencies



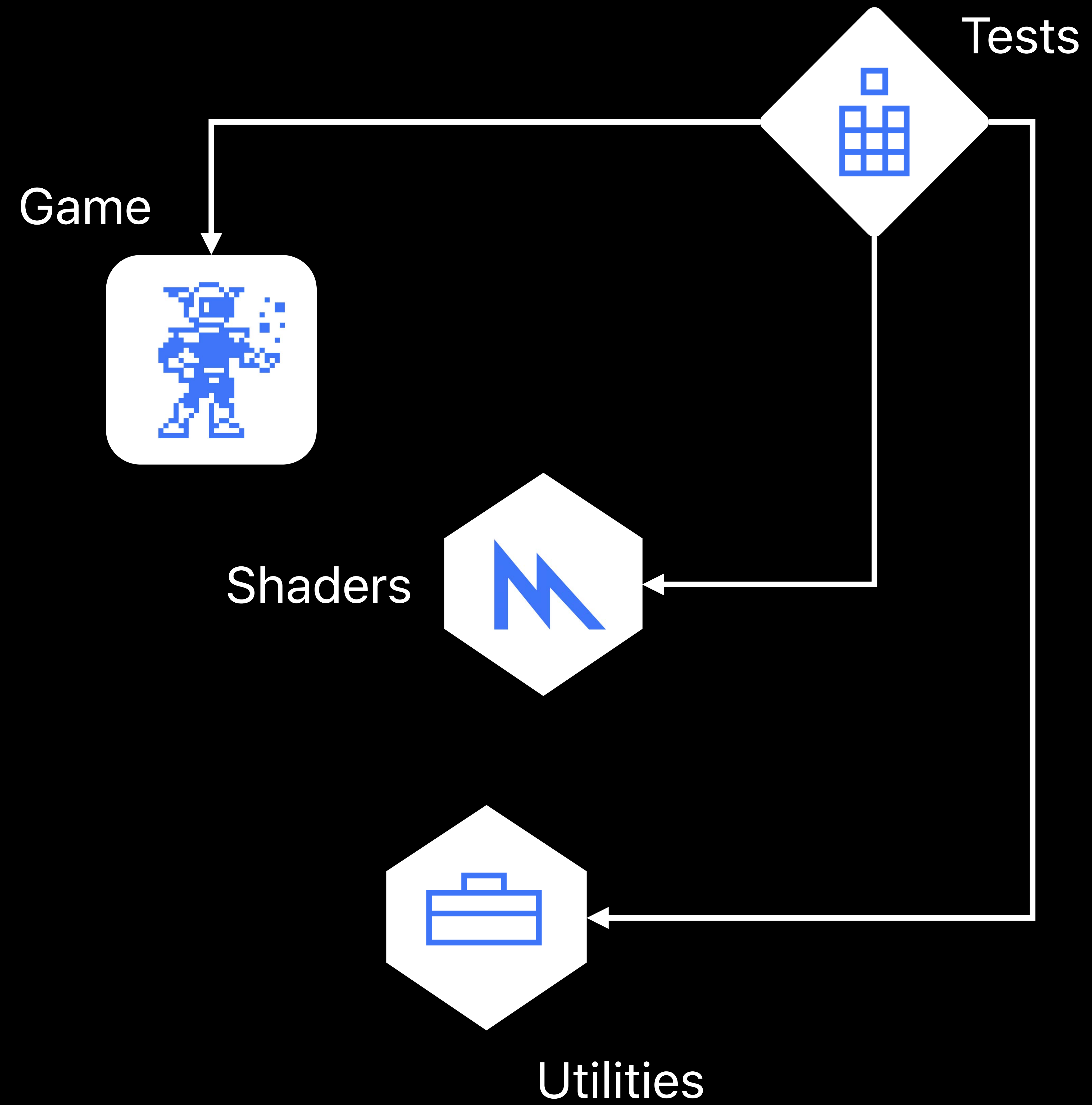
# Shaders Target Dependencies



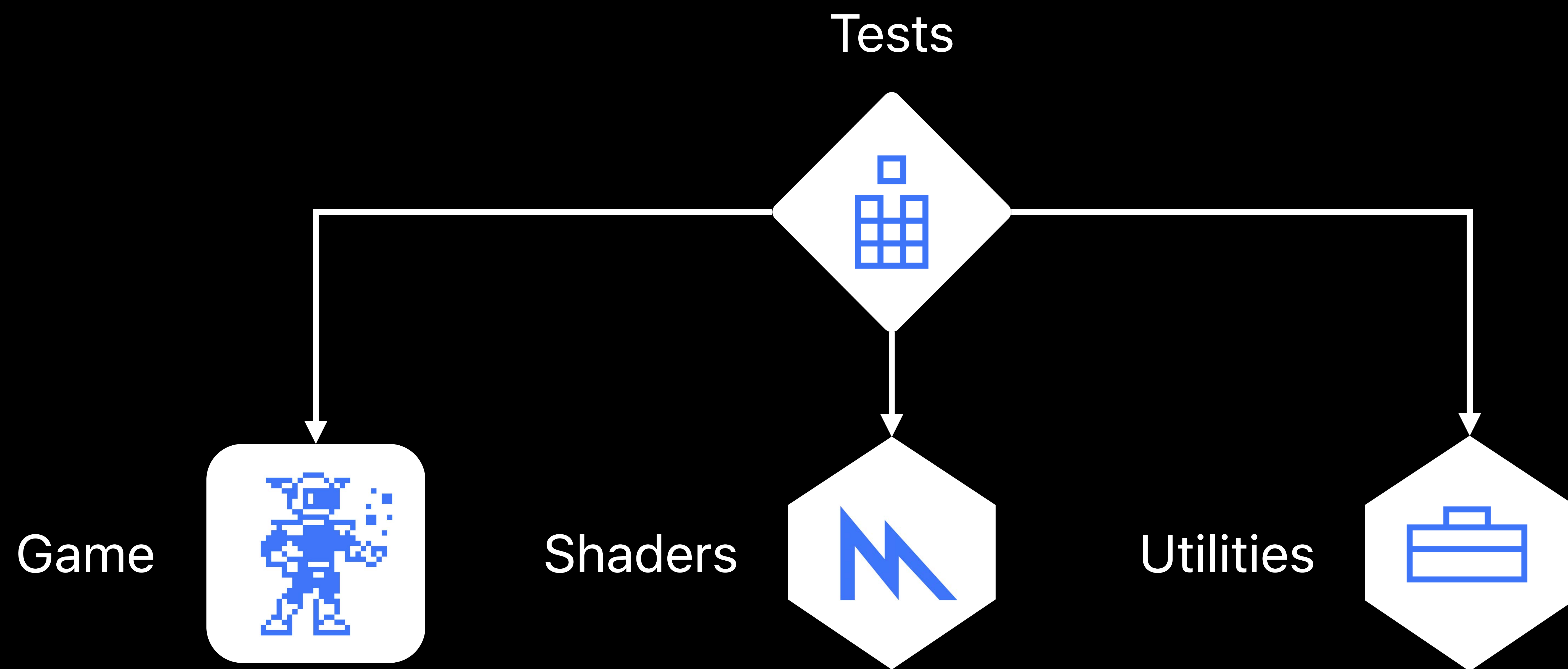
# Utilities Target Dependencies



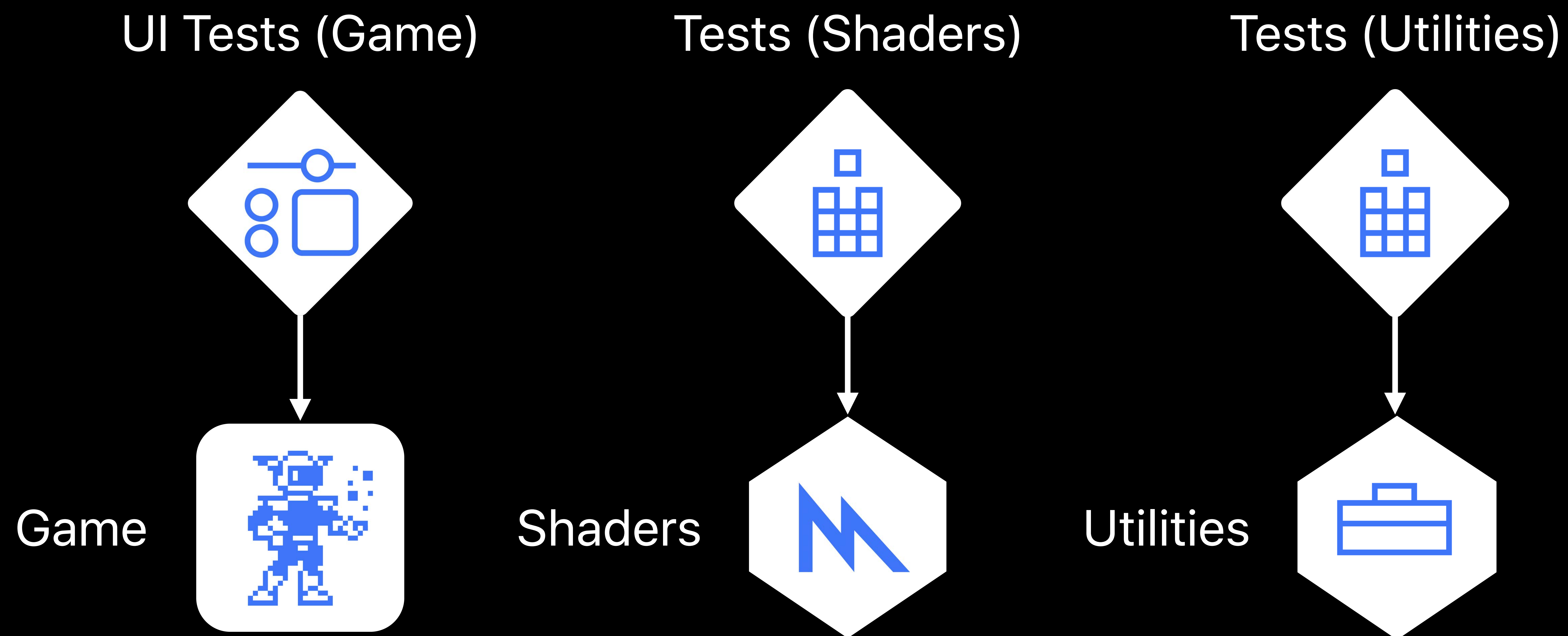
# Tests Target Dependencies



# Examine Tests Dependencies

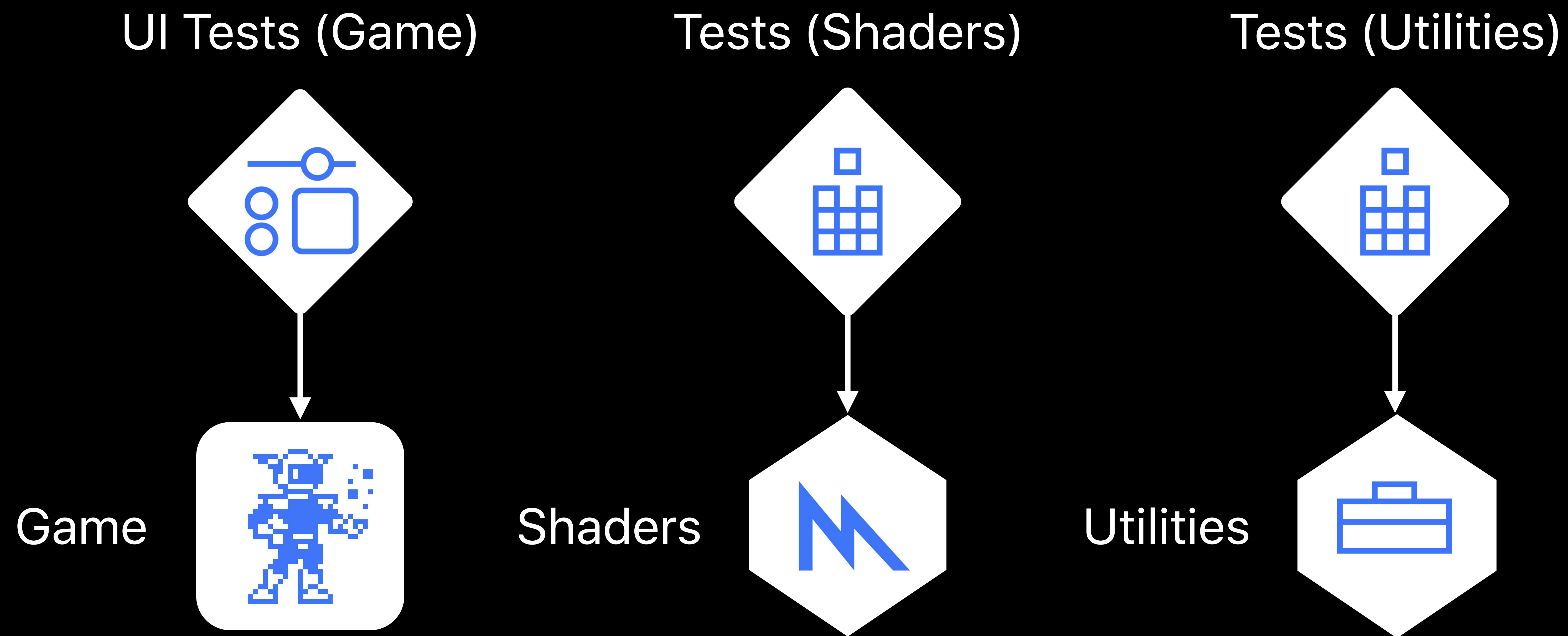


# Examine Tests Dependencies

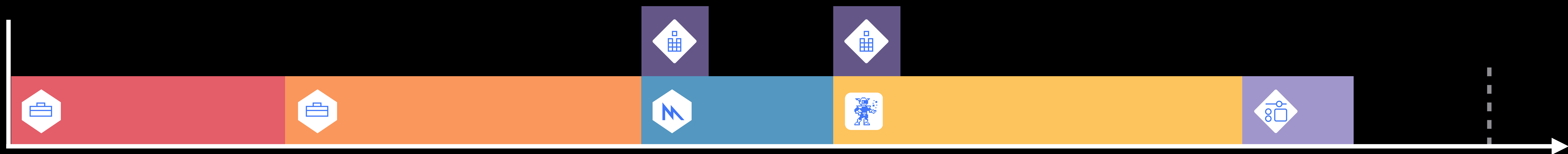
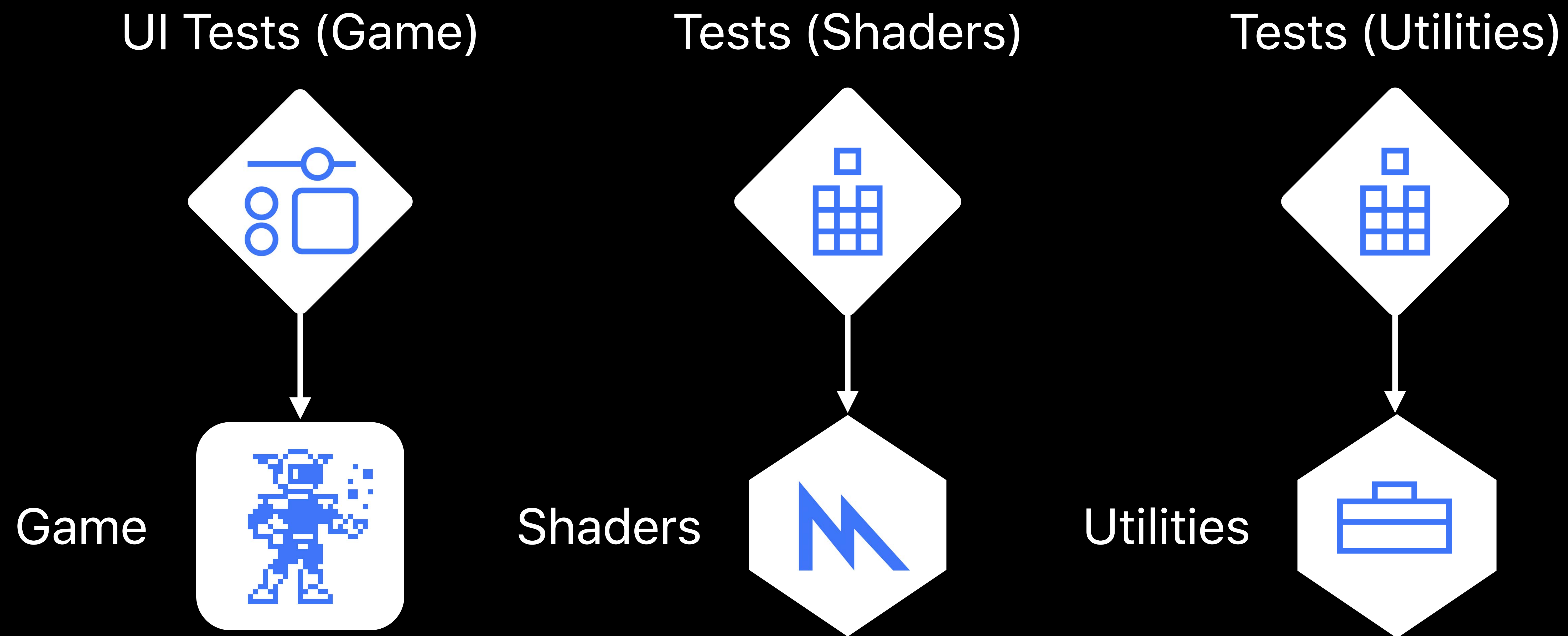




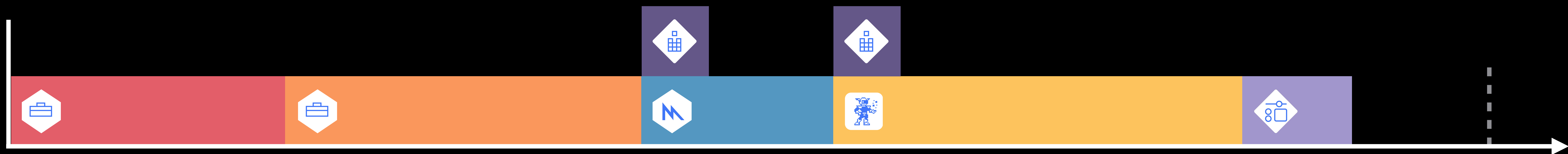
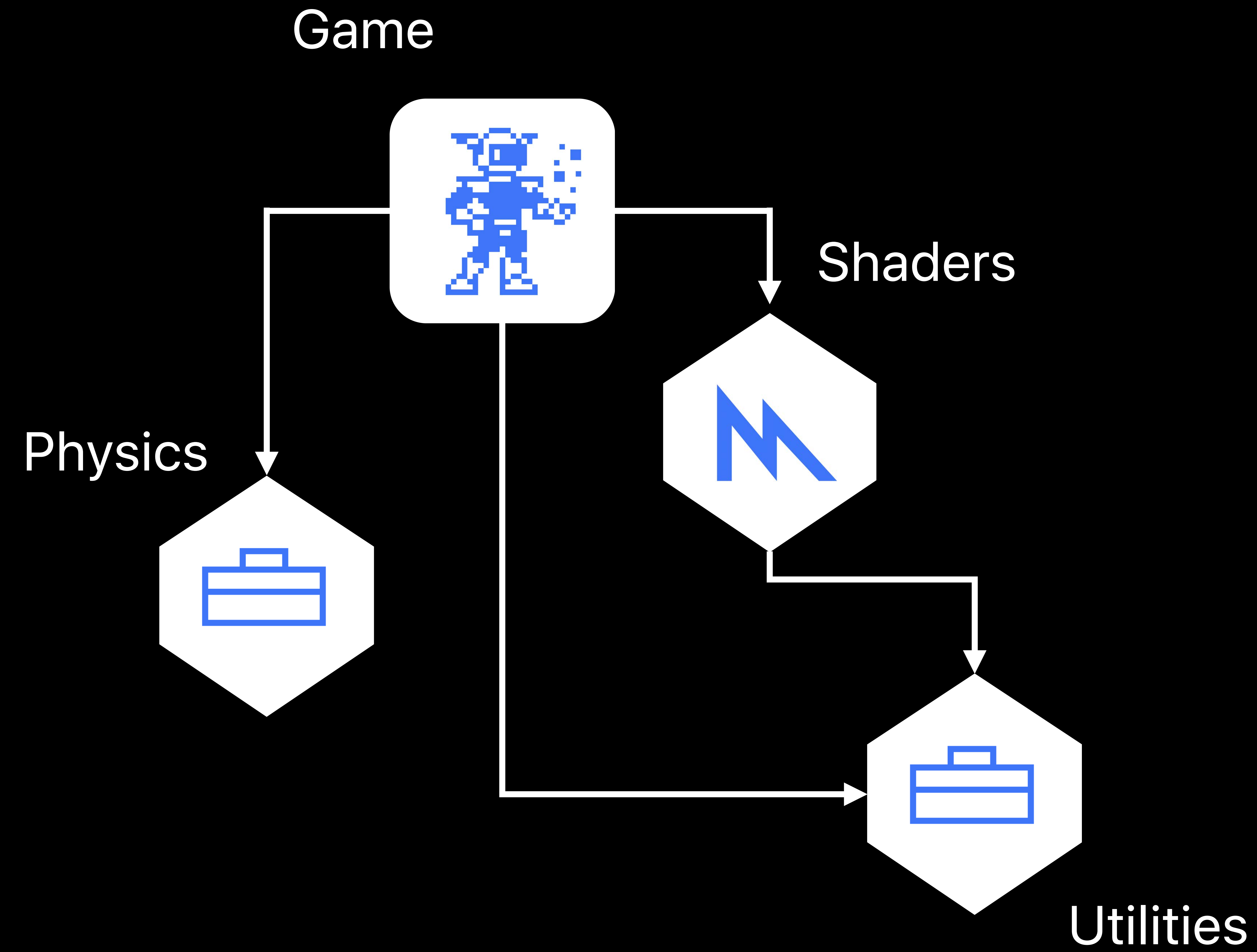
# Examine Tests Dependencies



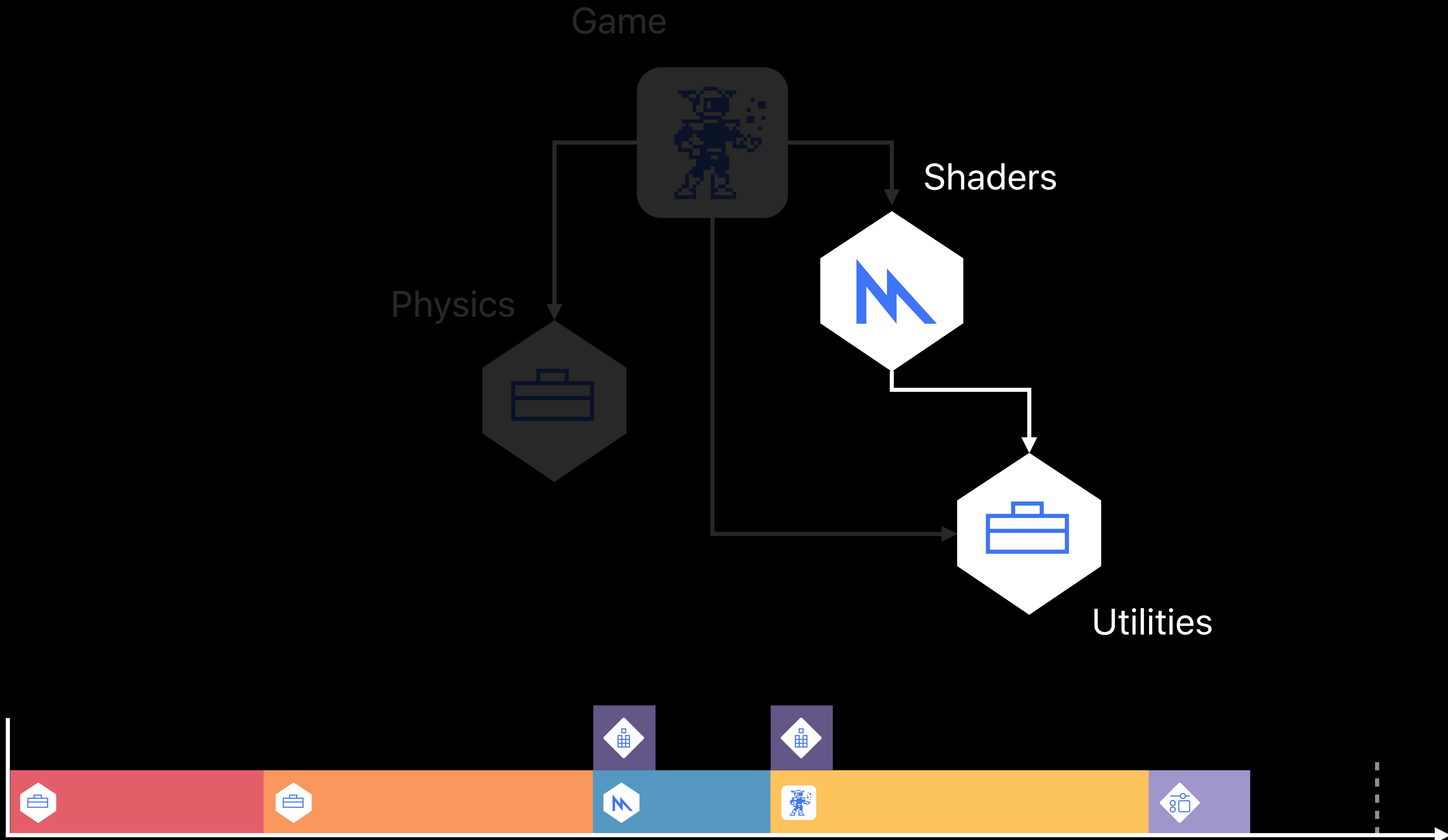
# Examine Tests Dependencies



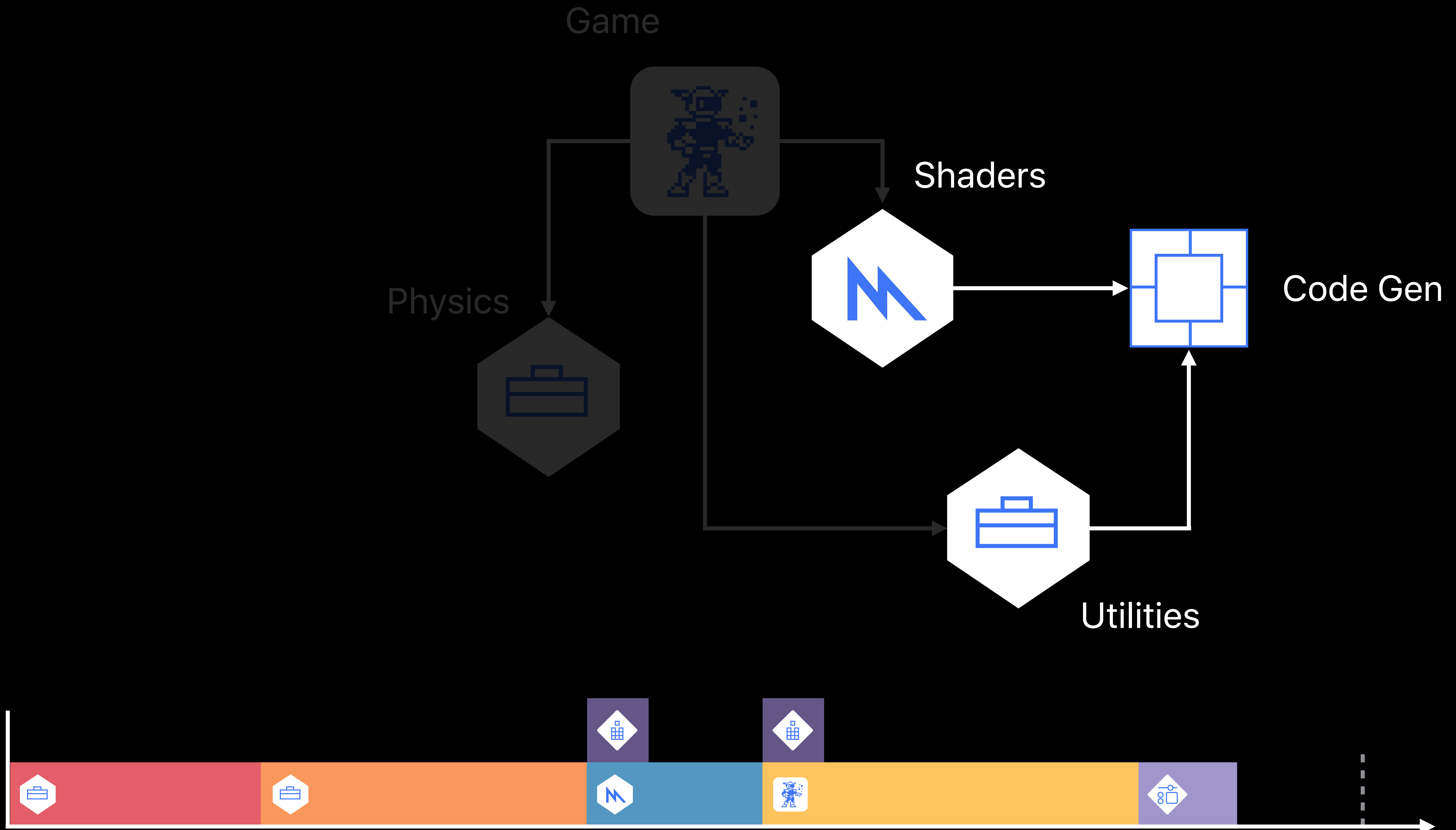
# Reduce Dependency Exposure



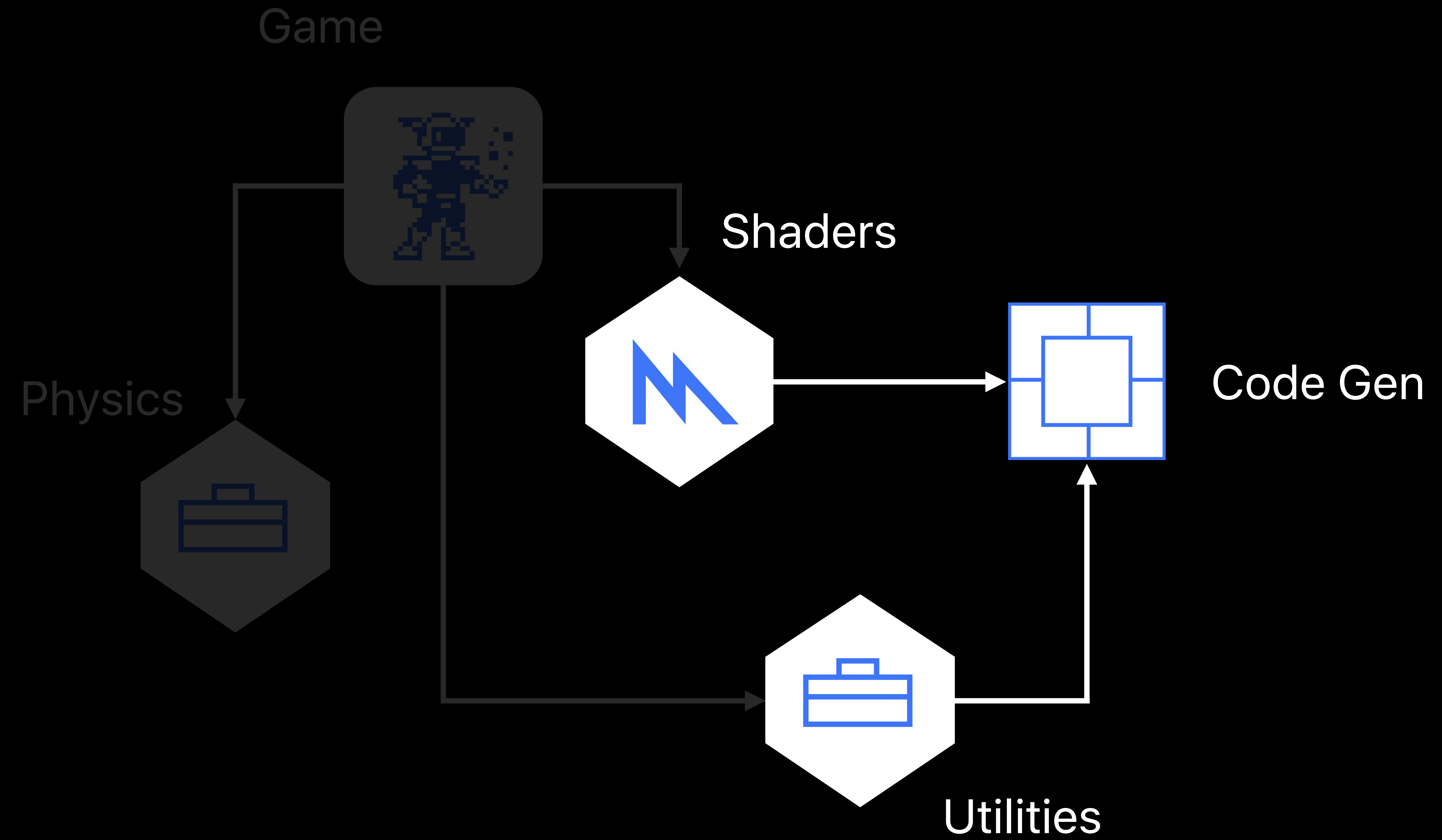
# Reduce Dependency Exposure



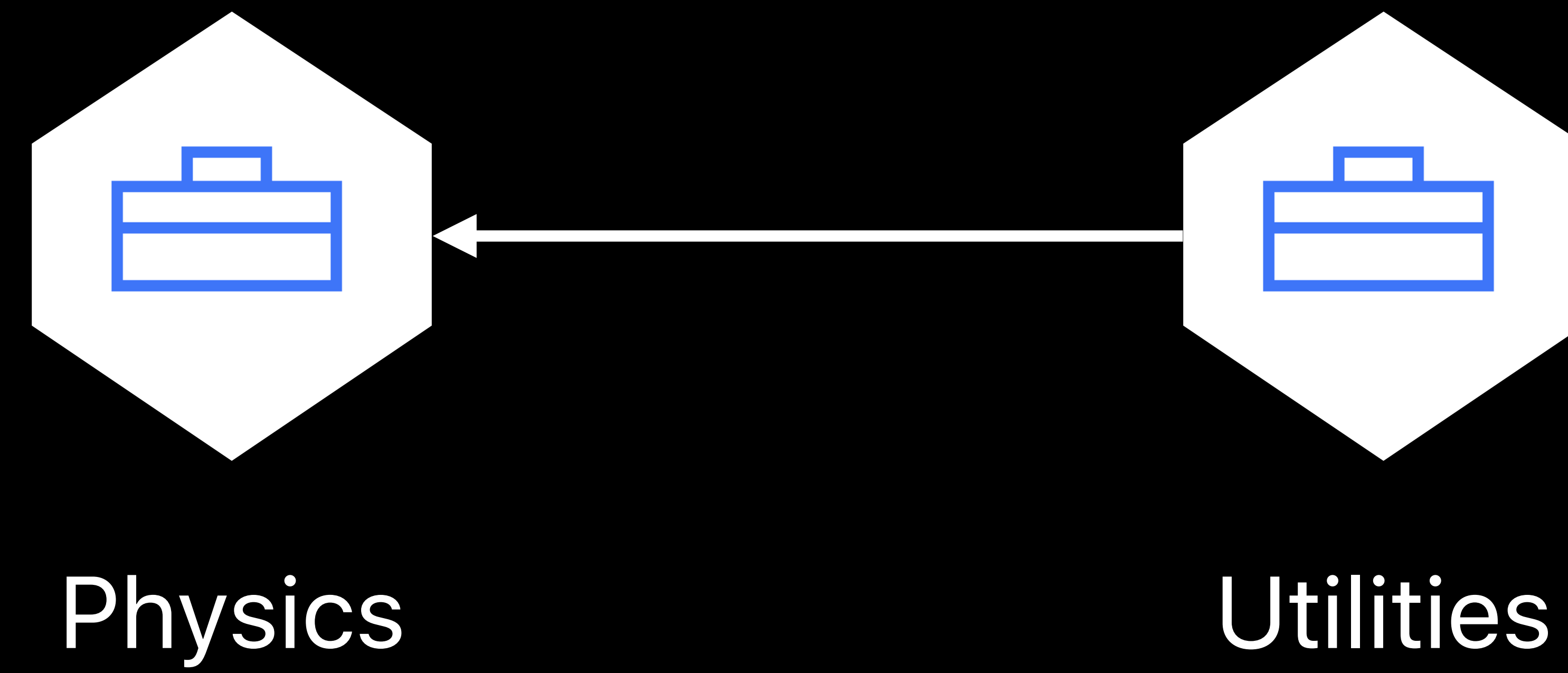
# Reduce Dependency Exposure



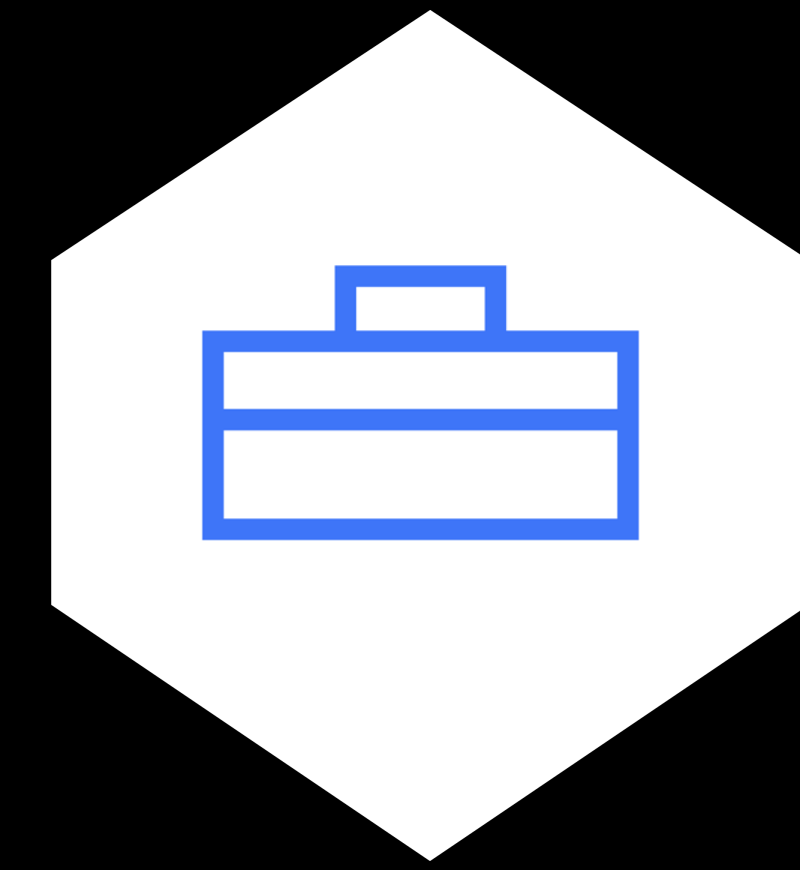
# Reduce Dependency Exposure



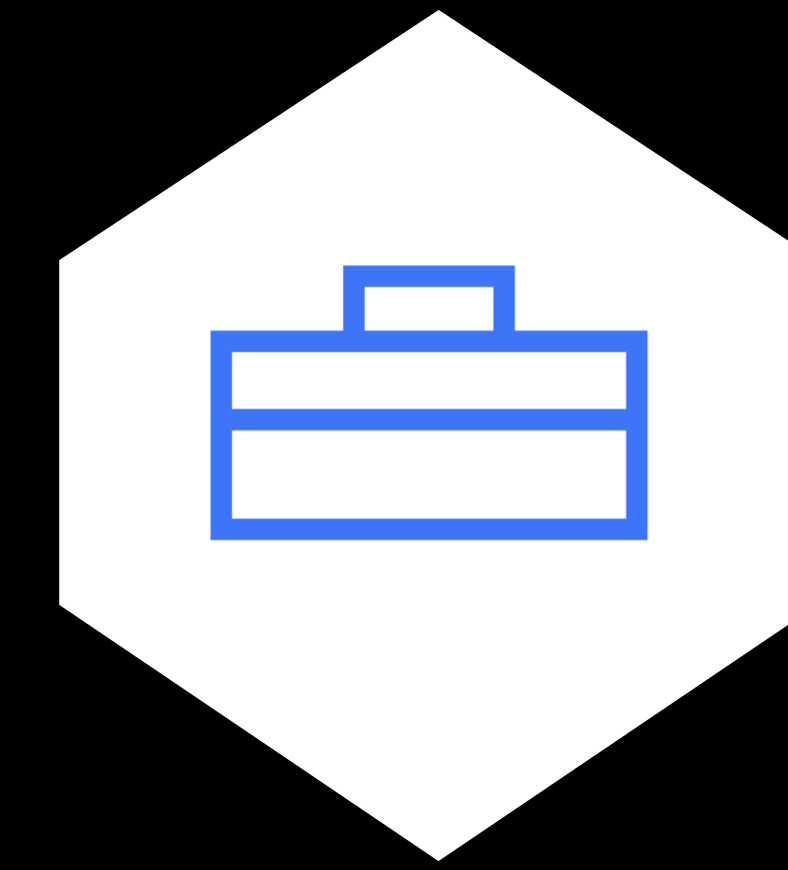
# Examine Unused Dependencies



# Examine Unused Dependencies



Physics

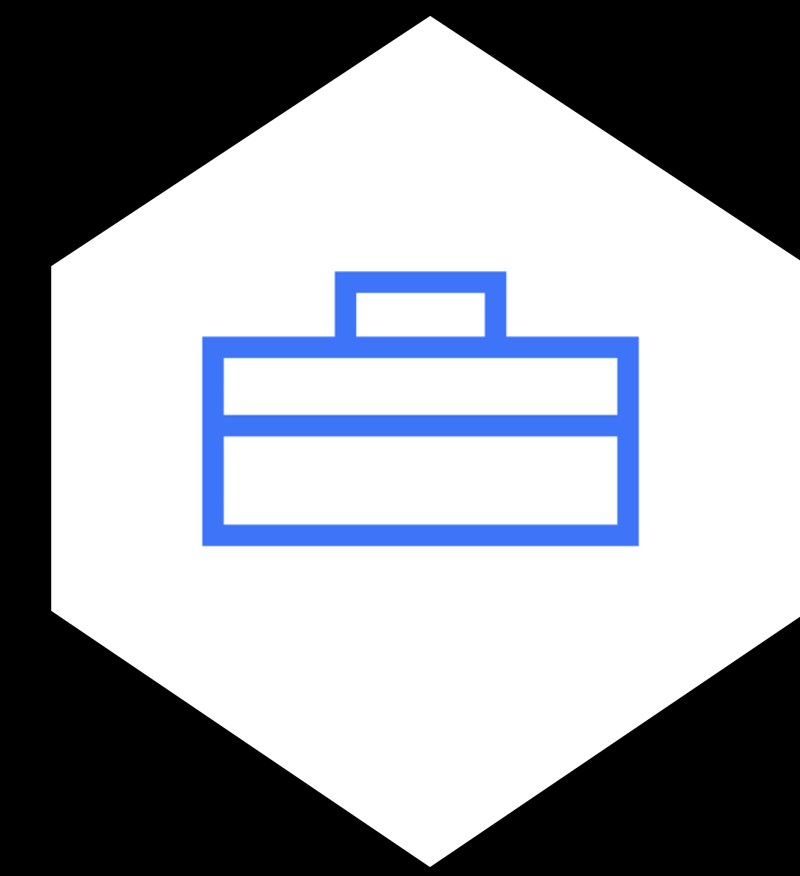


Utilities

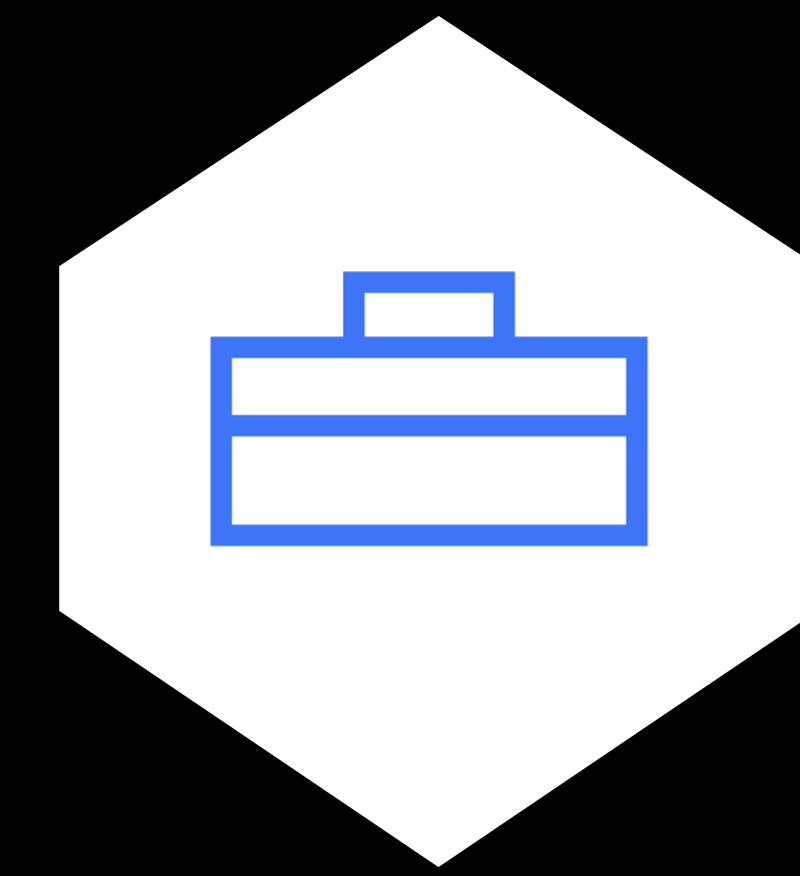




# Examine Unused Dependencies



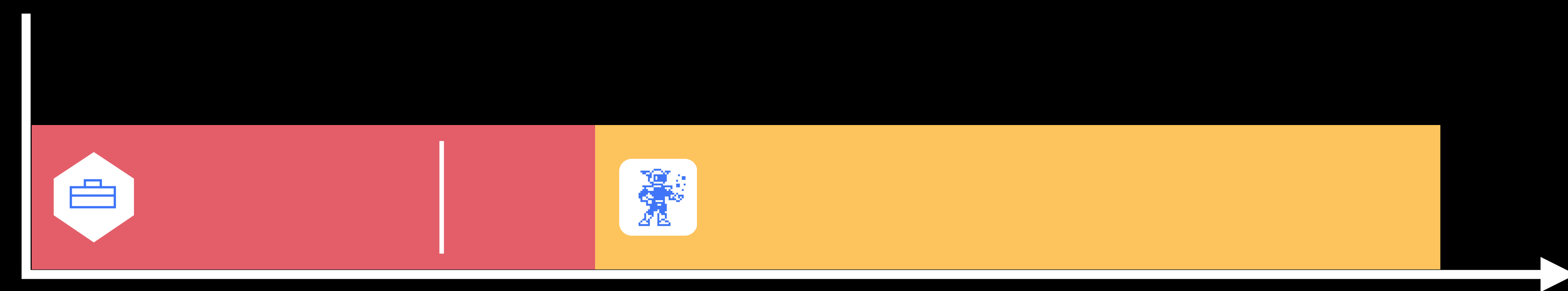
Physics



Utilities



# Target Build Process



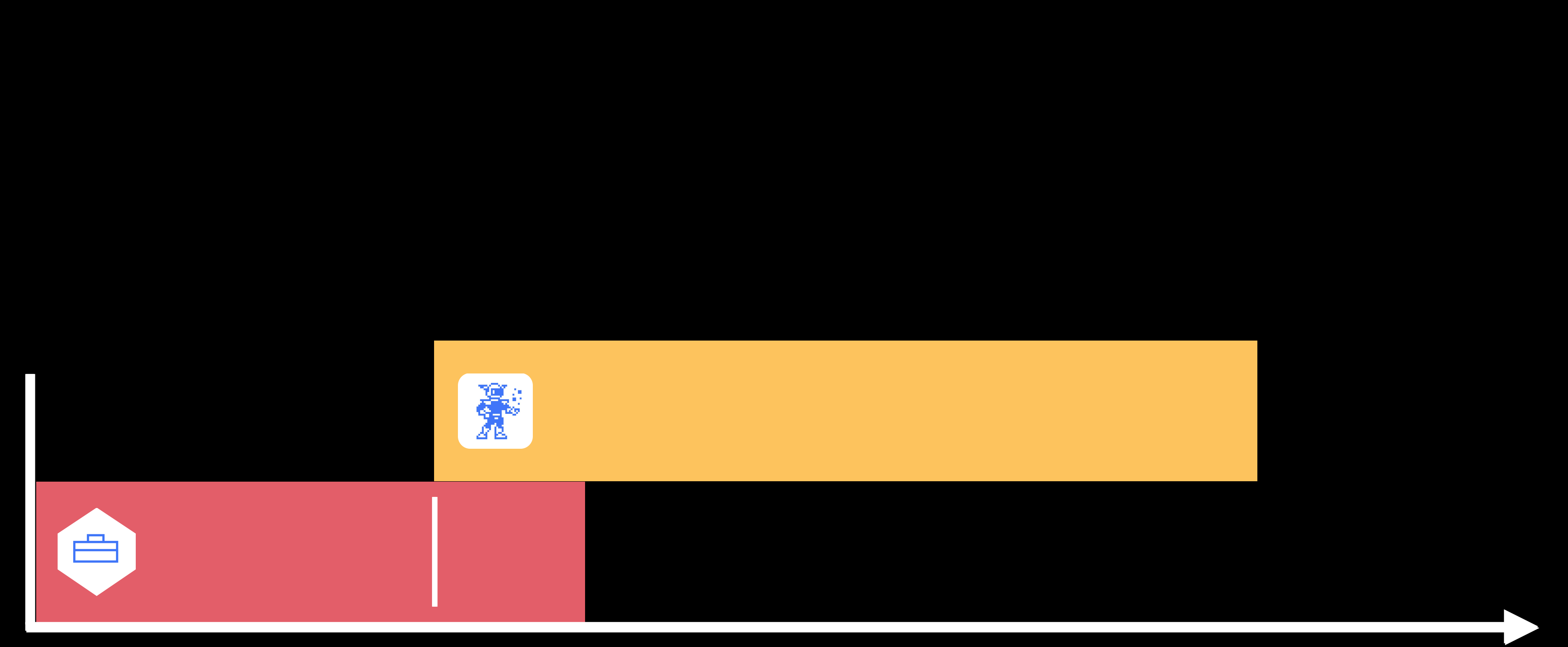
# Parallelized Target Build Process

NEW



# Parallelized Target Build Process

NEW



# Parallelized Target Build Process

NEW

Compile sources can start earlier



# Parallelized Target Build Process

NEW

Compile sources can start earlier

Waits only for what it needs



# Parallelized Target Build Process

NEW

Compile sources can start earlier

Waits only for what it needs

Must wait for Run Script phases



# Run Script Phases

Reducing the work on rebuilds



Allows you to customize your build process for your exact needs.

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
  - Physics

- PROJECT
- Game
- TARGETS
- Game
  - Utilities
  - Tests

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

- Filter
- Target Dependencies (1 item)
  - Compile Sources (10 items) x
  - Link Binary With Libraries (2 items) x
  - Copy Bundle Resources (2 items) x
  - Embed Frameworks (2 items) x
  - Generate Game Assets x

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

- Show environment variables in build log
- Run script only when installing

Input Files

\$(SRCROOT)/Assets/logo.psd  
\$(SRCROOT)/Assets/banner.psd

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

Output Files

\$(DERIVED\_FILE\_DIR)/Assets/logo.png  
\$(DERIVED\_FILE\_DIR)/Assets/banner.png

Output File Lists

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

Game

- Shaders.xcodeproj
- Assets
- Game
- Scripts
- Tests
- Utilities
- Products
- Frameworks
- Physics

Game

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT

- Game

TARGETS

- Game
- Utilities
- Tests

Target Dependencies (1 item)

Compile Sources (10 items)

Link Binary With Libraries (2 items)

Copy Bundle Resources (2 items)

Embed Frameworks (2 items)

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

Show environment variables in build log

Run script only when installing

\$(SRCROOT)/Assets/logo.psd  
 \$(SRCROOT)/Assets/banner.psd

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

Output Files

\$(DERIVED\_FILE\_DIR)/Assets/logo.png  
 \$(DERIVED\_FILE\_DIR)/Assets/banner.png

Output File Lists

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
  - Physics

- PROJECT
- Game
- TARGETS
- Game
  - Utilities
  - Tests

Filter

- Target Dependencies (1 item)
- Compile Sources (10 items)
- Link Binary With Libraries (2 items)
- Copy Bundle Resources (2 items)
- Embed Frameworks (2 items)
- Generate Game Assets

```
Shell /bin/sh
1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3
```

- Show environment variables in build log
- Run script only when installing

Input Files

\$(SRCROOT)/Assets/logo.psd  
\$(SRCROOT)/Assets/banner.psd

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

Output Files

\$(DERIVED\_FILE\_DIR)/Assets/logo.png  
\$(DERIVED\_FILE\_DIR)/Assets/banner.png

Output File Lists

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

Game

- Shaders.xcodeproj
- Assets
- Game
- Scripts
- Tests
- Utilities
- Products
- Frameworks
- Physics

Game

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT

- Game

TARGETS

- Game
- Utilities
- Tests

- Target Dependencies (1 item)
- Compile Sources (10 items)
- Link Binary With Libraries (2 items)
- Copy Bundle Resources (2 items)
- Embed Frameworks (2 items)
- Generate Game Assets

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

Show environment variables in build log

Input Files

- \$(SRCROOT)/Assets/logo.psd
- \$(SRCROOT)/Assets/banner.psd

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

Output Files

- \$(DERIVED\_FILE\_DIR)/Assets/logo.png
- \$(DERIVED\_FILE\_DIR)/Assets/banner.png

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

Output File Lists

- \$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
  - Physics

- PROJECT
- Game
- TARGETS
- Game
  - Utilities
  - Tests

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

- Filter
- Target Dependencies (1 item)
  - Compile Sources (10 items)
  - Link Binary With Libraries (2 items)
  - Copy Bundle Resources (2 items)
  - Embed Frameworks (2 items)
  - Generate Game Assets

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

- Show environment variables in build log
- Run script only when installing

Input Files

- \$(SRCROOT)/Assets/logo.psd
- \$(SRCROOT)/Assets/banner.psd

Input File Lists

- \$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

Output Files

- \$(DERIVED\_FILE\_DIR)/Assets/logo.png
- \$(DERIVED\_FILE\_DIR)/Assets/banner.png

Output File Lists

- \$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
  - Physics

PROJECT +

Game

TARGETS

Game

Utilities

Tests

- Target Dependencies (1 item)
- Compile Sources (10 items) x
- Link Binary With Libraries (2 items) x
- Copy Bundle Resources (2 items) x
- Embed Frameworks (2 items) x
- Generate Game Assets x

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

- Show environment variables in build log
- Run script only when installing

Input Files

\$(SRCROOT)/Assets/logo.psd

\$(SRCROOT)/Assets/banner.psd

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

+ -

\$(DERIVED\_FILE\_DIR)/Assets/logo.png

\$(DERIVED\_FILE\_DIR)/Assets/banner.png

+ -

Output File Lists

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

+ -

# File Lists

```
# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
```



# File Lists

Newline separated

```
# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
```

# File Lists

Newline separated

Support for build setting variables

```
# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
```

# File Lists

Newline separated

Support for build setting variables

Cannot be generated during the build

```
# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
```

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
  - Physics

- PROJECT
- Game
- TARGETS
- Game
  - Utilities
  - Tests

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

- Filter
- ▶ Target Dependencies (1 item)
  - ▶ Compile Sources (10 items) x
  - ▶ Link Binary With Libraries (2 items) x
  - ▶ Copy Bundle Resources (2 items) x
  - ▶ Embed Frameworks (2 items) x
  - ▼ Generate Game Assets x

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

- Show environment variables in build log
- Run script only when installing

Input Files

\$(SRCROOT)/Assets/logo.psd  
\$(SRCROOT)/Assets/banner.psd

+ -

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

+ -

Output Files

\$(DERIVED\_FILE\_DIR)/Assets/logo.png  
\$(DERIVED\_FILE\_DIR)/Assets/banner.png

+ -

Output File Lists

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

Game

- Shaders.xcodeproj
- Assets
- Game
- Scripts
- Tests
- Utilities
- Products
- Frameworks
- Physics

Game

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT +

Game

TARGETS

Game

Utilities

Tests

- Target Dependencies (1 item)
- Compile Sources (10 items)
- Link Binary With Libraries (2 items)
- Copy Bundle Resources (2 items)
- Embed Frameworks (2 items)

Generate Game Assets

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

- Show environment variables in build log
- Run script only when installing

Input Files

\$(SRCROOT)/Assets/logo.psd

\$(SRCROOT)/Assets/banner.psd

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

Output Files

\$(DERIVED\_FILE\_DIR)/Assets/logo.png

\$(DERIVED\_FILE\_DIR)/Assets/banner.png

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

- Game
  - Shaders.xcodeproj
  - Assets
  - Game
  - Scripts
  - Tests
  - Utilities
  - Products
  - Frameworks
  - Physics

- PROJECT
- Game
- TARGETS
- Game
  - Utilities
  - Tests

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

- Filter
- Target Dependencies (1 item)
  - Compile Sources (10 items)
  - Link Binary With Libraries (2 items)
  - Copy Bundle Resources (2 items)
  - Embed Frameworks (2 items)
  - Generate Game Assets

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

- Show environment variables in build log
- Run script only when installing

Input Files

\$(SRCROOT)/Assets/logo.psd  
\$(SRCROOT)/Assets/banner.psd

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

Output Files

\$(DERIVED\_FILE\_DIR)/Assets/logo.png  
\$(DERIVED\_FILE\_DIR)/Assets/banner.png

Output File Lists

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

Game

- Shaders.xcodeproj
- Assets
- Game
- Scripts
- Tests
- Utilities
- Products
- Frameworks
- Physics

Game

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT

- Game

TARGETS

- Game
- Utilities
- Tests

Filter

- Target Dependencies (1 item)
- Compile Sources (10 items)
- Link Binary With Libraries (2 items)
- Copy Bundle Resources (2 items)
- Embed Frameworks (2 items)
- Generate Game Assets

Shell /bin/sh

```

1 # Generate all of the game assets required in our game.
2 "$SRCROOT/Scripts/generate.sh"
3

```

- Show environment variables in build log
- Run script only when installing

Input Files

\$(SRCROOT)/Assets/logo.psd  
\$(SRCROOT)/Assets/banner.psd

+ -

Input File Lists

\$(SRCROOT)/Scripts/Inputs/assets.xcfilelist

+ -

Output Files

\$(DERIVED\_FILE\_DIR)/Assets/logo.png  
\$(DERIVED\_FILE\_DIR)/Assets/banner.png

Output File Lists

\$(SRCROOT)/Scripts/Outputs/assets.xcfilelist

+ -

# When the Run Script Phase is Run

No input files declared

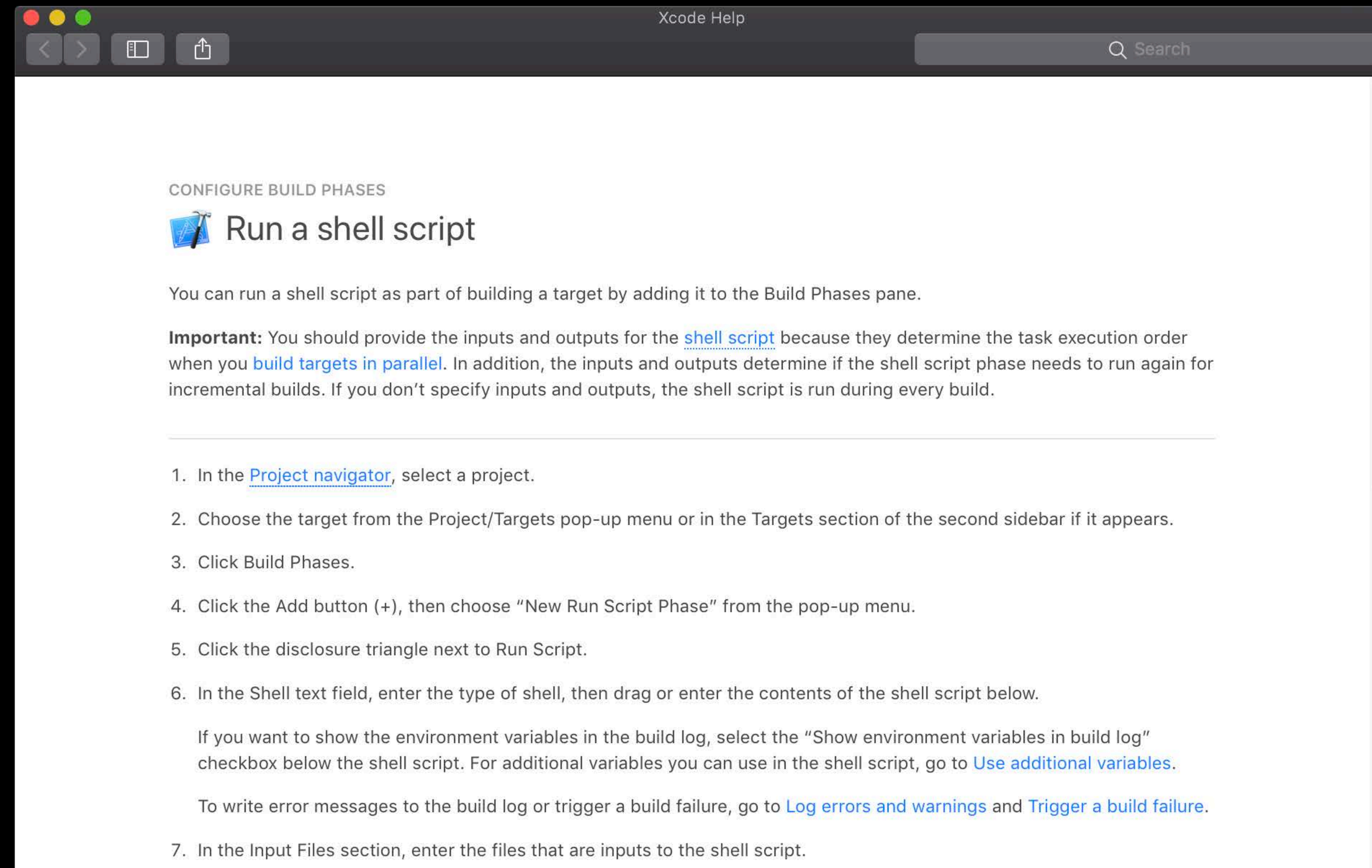
Input files changed

Output files missing



# Documentation

NEW



The screenshot shows a window titled "Xcode Help" with a search bar and navigation icons. The main content is a page titled "CONFIGURE BUILD PHASES" with a sub-section "Run a shell script" accompanied by a wrench icon. The text explains that shell scripts can be run as part of building a target. An important note states that inputs and outputs should be provided to determine execution order and incremental build behavior. A numbered list of seven steps guides the user through the process: selecting a project in the Project navigator, choosing a target, clicking Build Phases, adding a new Run Script Phase, clicking the Run Script disclosure triangle, entering shell type and script content, and finally, specifying input files. Additional links are provided for showing environment variables in the build log and for logging errors or triggering build failures.

CONFIGURE BUILD PHASES

## Run a shell script

You can run a shell script as part of building a target by adding it to the Build Phases pane.

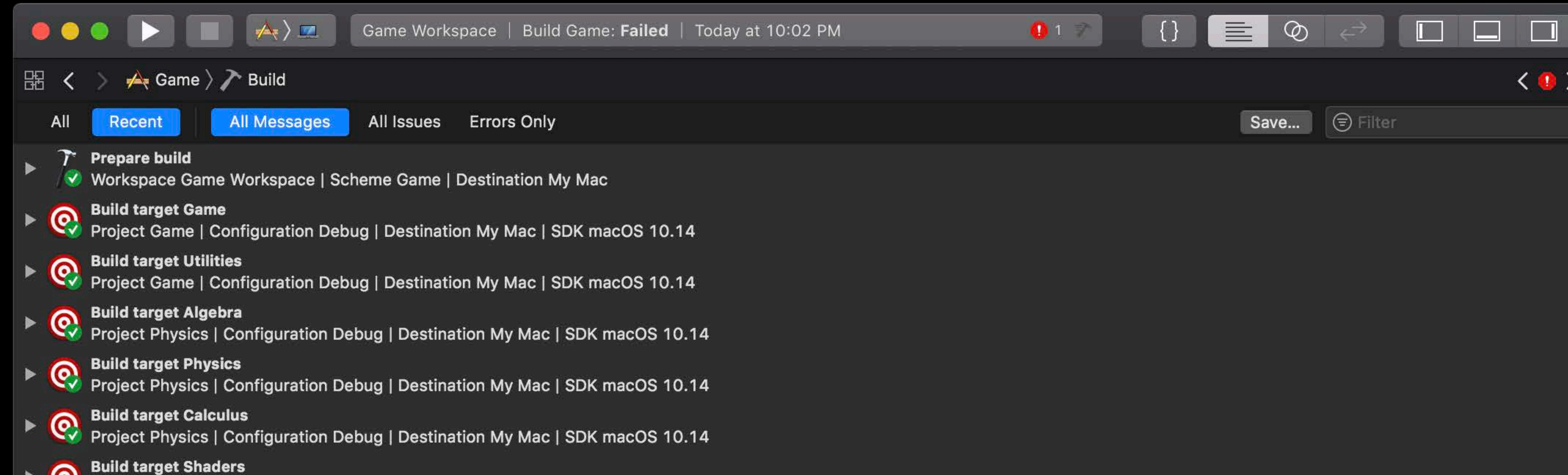
**Important:** You should provide the inputs and outputs for the [shell script](#) because they determine the task execution order when you [build targets in parallel](#). In addition, the inputs and outputs determine if the shell script phase needs to run again for incremental builds. If you don't specify inputs and outputs, the shell script is run during every build.

---

1. In the [Project navigator](#), select a project.
2. Choose the target from the Project/Targets pop-up menu or in the Targets section of the second sidebar if it appears.
3. Click Build Phases.
4. Click the Add button (+), then choose "New Run Script Phase" from the pop-up menu.
5. Click the disclosure triangle next to Run Script.
6. In the Shell text field, enter the type of shell, then drag or enter the contents of the shell script below.  
  
If you want to show the environment variables in the build log, select the "Show environment variables in build log" checkbox below the shell script. For additional variables you can use in the shell script, go to [Use additional variables](#).
7. In the Input Files section, enter the files that are inputs to the shell script.  
  
To write error messages to the build log or trigger a build failure, go to [Log errors and warnings](#) and [Trigger a build failure](#).

# Dependency Cycle Detection

NEW



Build system information 0.1 seconds

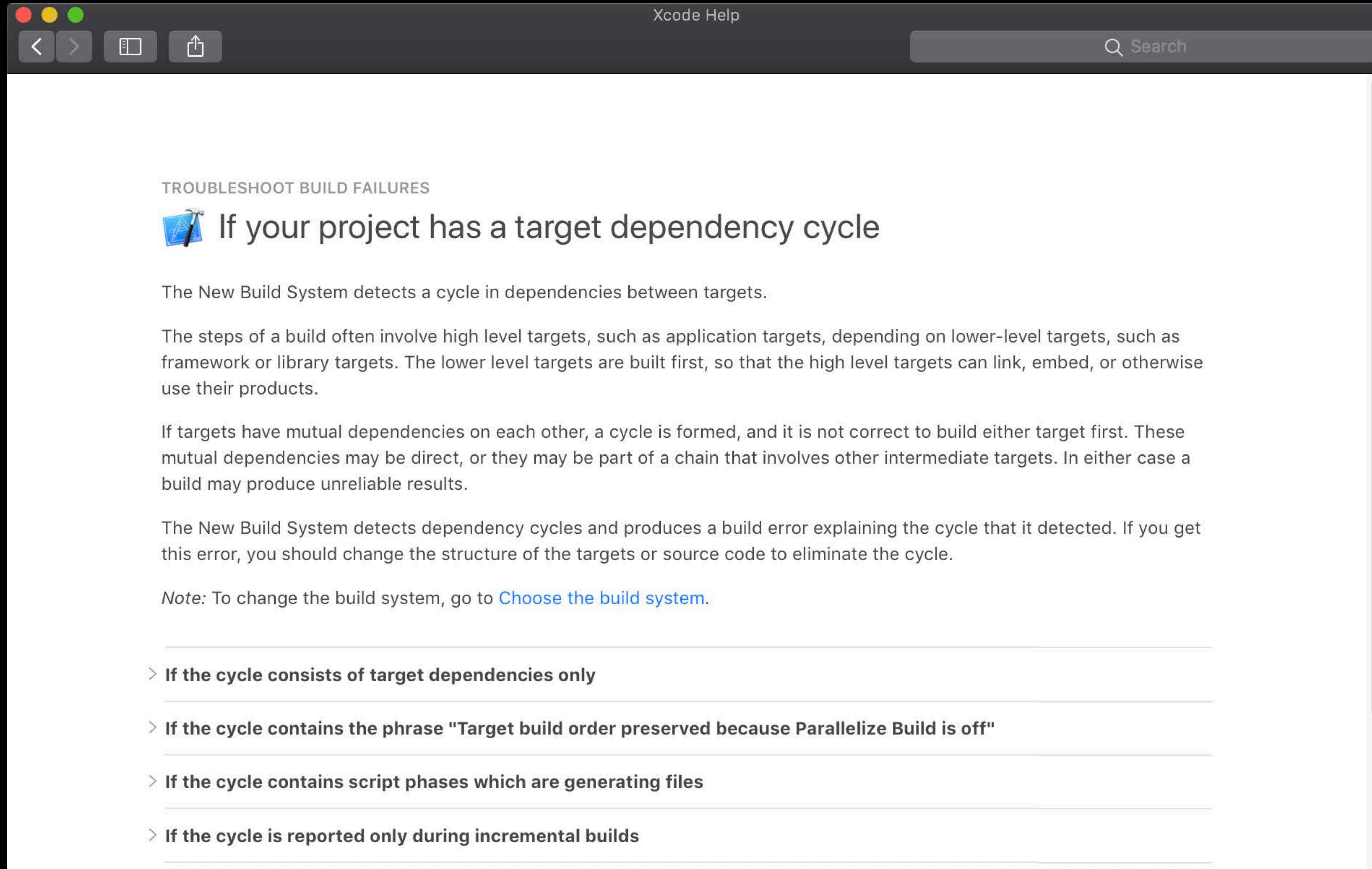
**!** Cycle in dependencies between targets 'Game' and 'Algebra'; building could produce unreliable results. **!** 1

**!** Build failed 5/29/18, 10:02 PM 1.1 seconds

1 error

# More on Dependency Cycles

NEW




The screenshot shows a window titled "Xcode Help" with a search bar and navigation icons. The main content is an article titled "TROUBLESHOOT BUILD FAILURES" with a sub-heading "If your project has a target dependency cycle" accompanied by a wrench icon. The article text explains that the New Build System detects dependency cycles between targets, which can occur due to mutual dependencies or chains of dependencies. It notes that such cycles can lead to unreliable build results. A note suggests changing the build system via a link "Choose the build system". At the bottom, there are four expandable sections, each starting with a right-pointing chevron and a bold heading.

Xcode Help

Search

TROUBLESHOOT BUILD FAILURES

 **If your project has a target dependency cycle**

The New Build System detects a cycle in dependencies between targets.

The steps of a build often involve high level targets, such as application targets, depending on lower-level targets, such as framework or library targets. The lower level targets are built first, so that the high level targets can link, embed, or otherwise use their products.

If targets have mutual dependencies on each other, a cycle is formed, and it is not correct to build either target first. These mutual dependencies may be direct, or they may be part of a chain that involves other intermediate targets. In either case a build may produce unreliable results.

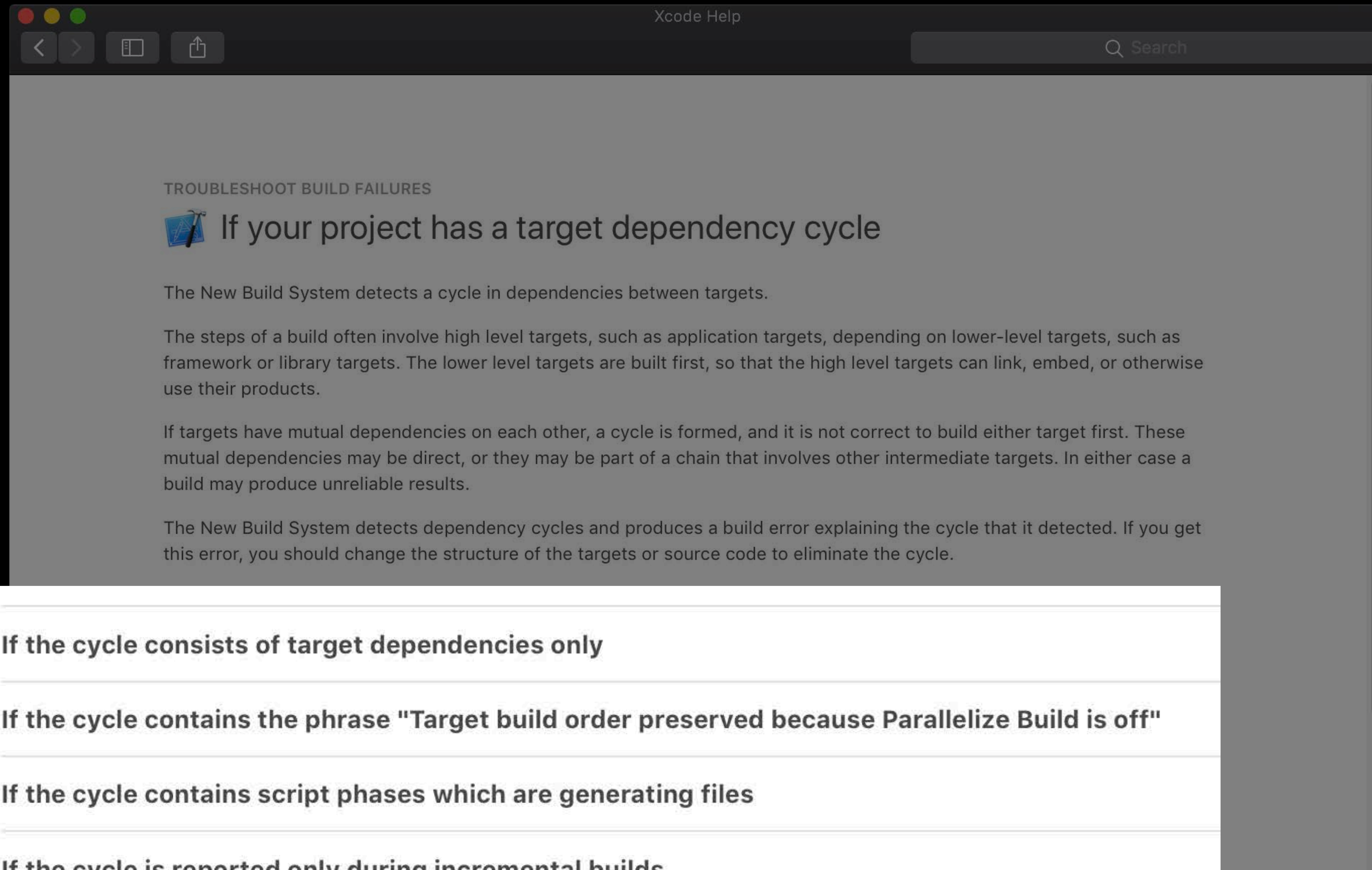
The New Build System detects dependency cycles and produces a build error explaining the cycle that it detected. If you get this error, you should change the structure of the targets or source code to eliminate the cycle.

*Note:* To change the build system, go to [Choose the build system](#).

- > **If the cycle consists of target dependencies only**
- > **If the cycle contains the phrase "Target build order preserved because Parallelize Build is off"**
- > **If the cycle contains script phases which are generating files**
- > **If the cycle is reported only during incremental builds**

# More on Dependency Cycles

NEW



The screenshot shows a window titled "Xcode Help" with a search bar and navigation icons. The main content is an article titled "TROUBLESHOOT BUILD FAILURES" with a sub-heading "If your project has a target dependency cycle" accompanied by a wrench icon. The article text explains that the New Build System detects cycles in dependencies between targets, where high-level targets depend on lower-level ones. It notes that mutual dependencies can form a cycle, leading to unreliable build results. The article concludes that the build system produces an error for such cycles, and users should modify target structures or source code to resolve them.

- > **If the cycle consists of target dependencies only**
- > **If the cycle contains the phrase "Target build order preserved because Parallelize Build is off"**
- > **If the cycle contains script phases which are generating files**
- > **If the cycle is reported only during incremental builds**

# Measuring Build Time

Increasing build efficiency

- Workspace Game Workspace | Scheme Game | Destination My Mac
      - Using new build system
      - Planning build
      - Using build description from memory
    - Build target Algebra
      - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Build target Utilities
      - Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Build target Shaders
      - Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14
      - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Particles.metal 0.3 seconds
      - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Environment.metal 0.3 seconds
      - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Effects.metal 0.3 seconds
      - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Shaders.metal 0.3 seconds
      - Linking Metal AIR /Users/owensd/Library/Developer/Xcode/DerivedData/Game\_Workspace-dorqcsttjkyvpordcebbiezmwaiico/Build/Products/Debug/Shaders.metallib 0.1 seconds
    - Build target Calculus
      - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Build target Physics
      - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Build target Game
      - Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Build Timing Summary
      - Workspace Game Workspace | Scheme Game | Destination My Mac
      - PhaseScriptExecution (1 task) 5.036 seconds
      - CompileAssetCatalog (1 task) 2.552 seconds
      - CompileMetalFile (5 tasks) 1.625 seconds
      - CompileStoryboard (1 task) 1.374 seconds
      - Ld (5 tasks) 0.340 seconds
      - CodeSign (3 tasks) 0.186 seconds
      - CompileC (4 tasks) 0.141 seconds
      - MetalLink (2 tasks) 0.047 seconds
      - Touch (5 tasks) 0.040 seconds
      - LinkStoryboards (1 task) 0.023 seconds
    - Build succeeded 5/25/18, 12:08 AM 9.4 seconds
      - No issues

All **Recent** All Messages All Issues Errors Only Save... Filter

- Prepare build
  - Workspace Game Workspace | Scheme Game | Destination My Mac
    - Using new build system
    - Planning build
    - Using build description from memory
- Build target Algebra
  - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
- Build target Utilities
  - Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14
- Build target Shaders
  - Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Assets.xcassets/Textures/Particles.metal 0.3 seconds
    - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Assets.xcassets/Textures/Environment.metal 0.3 seconds
    - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Assets.xcassets/Textures/Effects.metal 0.3 seconds
    - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Assets.xcassets/Textures/Shaders.metal 0.3 seconds
    - Linking Metal AIR /Users/owensd/Library/Developer/Xcode/DerivedData/.../MetalLib.s.metallib 0.1 seconds
- Build target Calculus
  - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
- Build target Physics
  - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
- Build target Game
  - Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14
- Build Timing Summary
  - Workspace Game Workspace | Scheme Game | Destination My Mac
    - PhaseScriptExecution (1 task) 5.036 seconds
    - CompileAssetCatalog (1 task) 2.552 seconds
    - CompileMetalFile (5 tasks) 1.625 seconds
    - CompileStoryboard (1 task) 1.374 seconds
    - Ld (5 tasks) 0.340 seconds
    - CodeSign (3 tasks) 0.186 seconds
    - CompileC (4 tasks) 0.141 seconds
    - MetalLink (2 tasks) 0.047 seconds
    - Touch (5 tasks) 0.040 seconds
    - LinkStoryboards (1 task) 0.023 seconds
  - Build succeeded 5/25/18, 12:08 AM 9.4 seconds**
  - No issues

# Build Log Filters

The screenshot shows the Xcode interface with a build log for a project named 'BuildBot'. The window title is 'BuildBot | Build BuildBot: Succeeded | Today at 10:48 PM'. The build log is filtered to show 'Recent' messages. The log contains the following tasks:

- ✓ Compile Storyboard file /Users/owensd/Library/Mobile Documents/com~apple~icloud~applecorporate/Documents/WWDC 2018/Building Faster in Xcode/BuildBot/BuildBot/Base.lproj/Main.storyboard
- ✓ Process /Users/owensd/Library/Mobile Documents/com~apple~icloud~applecorporate/Documents/WWDC 2018/Building Faster in Xcode/BuildBot/BuildBot/Info.plist 0.1 seconds
- ✓ Link Storyboards 0.1 seconds
- ✓ Copy GraphicsBot.metallib 0.1 seconds
- ✓ Copy BotLib.framework 0.1 seconds
- ✓ Sign /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app/Contents/Frameworks/GraphicsBot.metallib 0.1 seconds
- ✓ Sign /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app/Contents/Frameworks/BotLib.framework/Versions/A 0.1 seconds
- ✓ Copy Swift standard libraries into /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app 0.7 seconds
- ✓ Sign /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app 0.1 seconds
- ✓ Register /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app 0.1 seconds
- ✓ Touch 0.1 seconds

**Build Timing Summary**  
Workspace BuildBot | Scheme BuildBot | Destination My Mac

- CompileStoryboard (1 task) 0.583 seconds
- CompileMetalFile (1 task) 0.263 seconds
- Ld (2 tasks) 0.174 seconds
- CodeSign (3 tasks) 0.154 seconds
- CompileC (1 task) 0.035 seconds
- CompileAssetCatalog (1 task) 0.029 seconds
- LinkStoryboards (1 task) 0.022 seconds
- Touch (2 tasks) 0.019 seconds
- MetalLink (1 task) 0.015 seconds

**Build succeeded 5/16/18, 10:48 PM 2.2 seconds**  
No issues



# Build Log Filters

The screenshot shows the Xcode interface with the build log open. The top toolbar includes window management icons and a status bar showing 'BuildBot | Build BuildBot: Succeeded | Today at 10:48 PM'. The breadcrumb navigation shows 'BuildBot > Build'. The filter menu is open, highlighting 'All Messages'. The log output shows a successful build with various tasks and their durations.

BuildBot | Build BuildBot: Succeeded | Today at 10:48 PM

BuildBot > Build

All Recent All Messages All Issues Errors Only Save... Filter

- ✓ Process /Users/owensd/Library/Mobile Documents/com~apple~icloud~applecorporate/Documents/WWDC 2018/Building Faster in Xcode/BuildBot/BuildBot/Info.plist 0.1 seconds
- ✓ Link Storyboards 0.1 seconds
- ✓ Copy GraphicsBot.metallib 0.1 seconds
- ✓ Copy BotLib.framework 0.1 seconds
- ✓ Sign /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app/Contents/Frameworks/GraphicsBot.metallib 0.1 seconds
- ✓ Sign /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app/Contents/Frameworks/BotLib.framework/Versions/A 0.1 seconds
- ✓ Copy Swift standard libraries into /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app 0.7 seconds
- ✓ Sign /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app 0.1 seconds
- ✓ Register /Users/owensd/Library/Developer/Xcode/DerivedData/BuildBot-cnvdjrvgpabjvceivhspqetljftl/Build/Products/Debug/BuildBot.app 0.1 seconds
- ✓ Touch 0.1 seconds

Build Timing Summary  
Workspace BuildBot | Scheme BuildBot | Destination My Mac

- CompileStoryboard (1 task) 0.583 seconds
- CompileMetalFile (1 task) 0.263 seconds
- Ld (2 tasks) 0.174 seconds
- CodeSign (3 tasks) 0.154 seconds
- CompileC (1 task) 0.035 seconds
- CompileAssetCatalog (1 task) 0.029 seconds
- LinkStoryboards (1 task) 0.022 seconds
- Touch (2 tasks) 0.019 seconds
- MetalLink (1 task) 0.015 seconds

✓ Build succeeded 5/16/18, 10:48 PM 2.2 seconds  
No issues

# Build With Timing Summary


NEW


The image shows a screenshot of an IDE's menu system. The 'Product' menu is open, and the 'Perform Action' sub-menu is selected. Within 'Perform Action', the 'Build With Timing Summary' option is highlighted. Other options in the 'Perform Action' sub-menu include 'Run Without Building', 'Test Without Building', 'Profile Without Building', 'Test', 'Test Again', 'Profile', 'Profile Again', 'Compile File', 'Analyze File', 'Preprocess File', 'Assemble File', and 'Generate Optimization Profile...'. The 'Product' menu also includes 'Run', 'Test', 'Profile', 'Analyze', 'Archive', 'Build For', 'Build', 'Clean Build Folder', 'Stop', 'Scheme', 'Destination', and 'Create Bot...'. Keyboard shortcuts are provided for many of these actions.

Menu Item	Shortcut
Product	
Run	⌘R
Test	⌘U
Profile	⌘I
Analyze	⇧⌘B
Archive	
Build For	
Perform Action	
Run Without Building	⇧⌘R
Test Without Building	⇧⌘U
Profile Without Building	⇧⌘I
Build With Timing Summary	
Test	⇧⌘U
Test Again	⇧⌘G
Profile	
Profile Again	
Compile File	⇧⌘B
Analyze File	⇧⇧⌘B
Preprocess File	
Assemble File	
Generate Optimization Profile...	
Build	⌘B
Clean Build Folder	⇧⌘K
Stop	⌘.
Scheme	
Destination	
Create Bot...	

- Workspace Game Workspace | Scheme Game | Destination My Mac
      - Using new build system
      - Planning build
      - Using build description from memory
    - Build target Algebra
      - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Build target Utilities
      - Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14
    - Build target Shaders
      - Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14
        - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Particles.metal 0.3 seconds
        - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Environment.metal 0.3 seconds
        - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Effects.metal 0.3 seconds
        - Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Shaders.metal 0.3 seconds
        - Linking Metal AIR /Users/owensd/Library/Developer/Xcode/DerivedData/Game\_Workspace-dorqsttjyvpordcebbiezmaico/Build/Products/Debug/Shaders.metallib 0.1 seconds
      - Build target Calculus
        - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
      - Build target Physics
        - Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
      - Build target Game
        - Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14
      - Build Timing Summary
        - Workspace Game Workspace | Scheme Game | Destination My Mac
          - PhaseScriptExecution (1 task) 5.036 seconds
          - CompileAssetCatalog (1 task) 2.552 seconds
          - CompileMetalFile (5 tasks) 1.625 seconds
          - CompileStoryboard (1 task) 1.374 seconds
          - Ld (5 tasks) 0.340 seconds
          - CodeSign (3 tasks) 0.186 seconds
          - CompileC (4 tasks) 0.141 seconds
          - MetalLink (2 tasks) 0.047 seconds
          - Touch (5 tasks) 0.040 seconds
          - LinkStoryboards (1 task) 0.023 seconds
  - Build succeeded 5/25/18, 12:08 AM 9.4 seconds
    - No issues

▶  **Build target Algebra**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

▶  **Build target Utilities**  
Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14

▼  **Build target Shaders**  
Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Particles.metal 0.3 seconds


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Environment.metal 0.3 seconds


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Effects.metal 0.3 seconds


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Shaders.metal 0.3 seconds

✓ Linking Metal AIR /Users/owensd/Library/Developer/Xcode/DerivedData/Game\_Workspace-dorqcstjvkvpordcebbiezmaico/Build/Products/Debug/Shaders.metallib 0.1 seconds

▶  **Build target Calculus**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

▶  **Build target Physics**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

▶  **Build target Game**  
Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14

▼  **Build Timing Summary**  
Workspace Game Workspace | Scheme Game | Destination My Mac

↗ PhaseScriptExecution (1 task) 5.036 seconds

↗ CompileAssetCatalog (1 task) 2.552 seconds

↗ CompileMetalFile (5 tasks) 1.625 seconds

↗ CompileStoryboard (1 task) 1.374 seconds

↗ Ld (5 tasks) 0.340 seconds

↗ CodeSign (3 tasks) 0.186 seconds

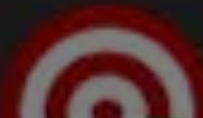
↗ CompileC (4 tasks) 0.141 seconds


↗ MetalLink (2 tasks) 0.047 seconds


↗ Touch (5 tasks) 0.040 seconds

↗ LinkStoryboards (1 task) 0.023 seconds

✓ **Build succeeded 5/25/18, 12:08 AM 9.4 seconds**  
No issues

▶  **Build target Algebra**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

▶  **Build target Utilities**  
Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14

▼  **Build target Shaders**  
Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Particles.metal 0.3 seconds


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Environment.metal 0.3 seconds

✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Effects.metal 0.3 seconds

✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Shaders.metal 0.3 seconds

✓ Linking Metal AIR /Users/owensd/Library/Developer/Xcode/DerivedData/Game\_Workspace-dorqcstjtkvpordcebbiezmaico/Build/Products/Debug/Shaders.metallib 0.1 seconds

▶  **Build target Calculus**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

▶  **Build target Physics**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

### **Build Timing Summary** Workspace Game Workspace | Scheme Game | Destination My Mac

➤ PhaseScriptExecution (1 task) 5.036 seconds

➤ CompileAssetCatalog (1 task) 2.552 seconds

➤ CompileMetalFile (5 tasks) 1.625 seconds

➤ CompileStoryboard (1 task) 1.374 seconds

➤ Ld (5 tasks) 0.340 seconds


➤ CodeSign (3 tasks) 0.186 seconds


➤ CompileC (4 tasks) 0.141 seconds


➤ MetalLink (2 tasks) 0.047 seconds

➤ Touch (5 tasks) 0.040 seconds

➤ LinkStoryboards (1 task) 0.023 seconds

▶  **Build target Algebra**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

▶  **Build target Utilities**  
Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14

▼  **Build target Shaders**  
Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Particles.metal 0.3 seconds


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Environment.metal 0.3 seconds


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Effects.metal 0.3 seconds


✓ Compile /Users/owensd/WWDC 2018/Building Faster in Xcode/Demo App/Before/Shaders/Shaders/Shaders.metal 0.3 seconds


✓ Linking Metal AIR /Users/owensd/Library/Developer/Xcode/DerivedData/Game\_Workspace-dorqcstjtkvpordcebbiezmaico/Build/Products/Debug/Shaders.metallib 0.1 seconds


▶  **Build target Calculus**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14


▶  **Build target Physics**  
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14


 **Build Timing Summary**  
Workspace Game Workspace | Scheme Game | Destination My Mac


 **PhaseScriptExecution (1 task) 5.036 seconds** ←


 CompileAssetCatalog (1 task) 2.552 seconds


 CompileMetalFile (5 tasks) 1.625 seconds


 CompileStoryboard (1 task) 1.374 seconds


 Ld (5 tasks) 0.340 seconds

 CodeSign (3 tasks) 0.186 seconds

 CompileC (4 tasks) 0.141 seconds

 MetalLink (2 tasks) 0.047 seconds

 Touch (5 tasks) 0.040 seconds

 LinkStoryboards (1 task) 0.023 seconds

```
$ xcodebuild -showBuildTimingSummary
```

## Build Timing Summary

```
CompileSwiftSources (1 task) | 5.434 seconds  
PhaseScriptExecution (1 task) | 5.046 seconds  
CompileAssetCatalog (1 task) | 2.788 seconds  
CompileStoryboard (1 task) | 1.880 seconds  
CompileMetalFile (5 tasks) | 1.735 seconds  
CopySwiftLibs (1 task) | 0.740 seconds  
Ld (2 tasks) | 0.306 seconds  
CodeSign (3 tasks) | 0.177 seconds  
CompileC (1 task) | 0.170 seconds  
MetalLink (2 tasks) | 0.046 seconds  
Ditto (4 tasks) | 0.032 seconds  
LinkStoryboards (1 task) | 0.023 seconds
```

```
$ xcodebuild -showBuildTimingSummary
```

## Build Timing Summary

```
CompileSwiftSources (1 task) | 5.434 seconds  
PhaseScriptExecution (1 task) | 5.046 seconds  
CompileAssetCatalog (1 task) | 2.788 seconds  
CompileStoryboard (1 task) | 1.880 seconds  
CompileMetalFile (5 tasks) | 1.735 seconds  
CopySwiftLibs (1 task) | 0.740 seconds  
Ld (2 tasks) | 0.306 seconds  
CodeSign (3 tasks) | 0.177 seconds  
CompileC (1 task) | 0.170 seconds  
MetalLink (2 tasks) | 0.046 seconds  
Ditto (4 tasks) | 0.032 seconds  
LinkStoryboards (1 task) | 0.023 seconds
```



# Source-Level Improvements

Jordan Rose, Swift Engineer


**Remove this build time workaround...**

Reducing the work on rebuilds

# Turn off Whole Module Mode For Debug Builds

## ▼ Swift Compiler - Code Generation

Name

 Game

## ▼ Compilation Mode

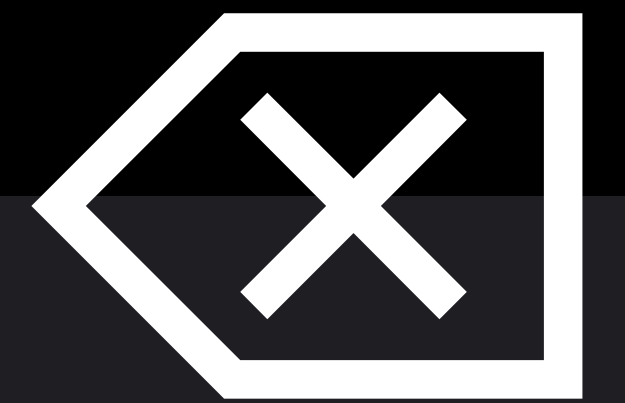
**Debug**

**+** Whole Module ⌵

**Release**

Whole Module ⌵

# Turn off Whole Module Mode For Debug Builds



## ▼ Swift Compiler - Code Generation


Name

 Game

## ▼ Compilation Mode

<Multiple values> ⌵

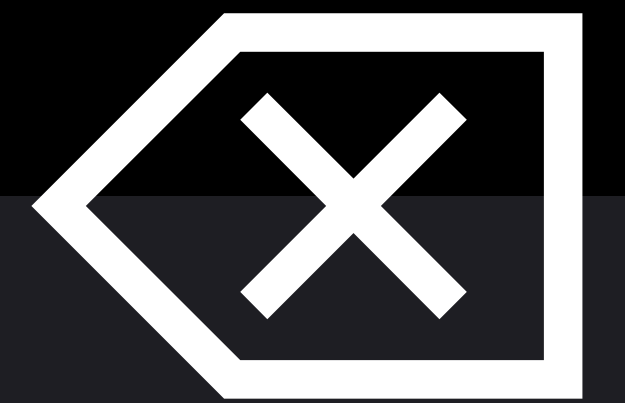
Debug

 Incremental ⌵

Release

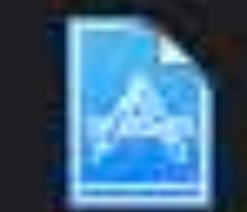
Whole Module ⌵

# Turn off Whole Module Mode For Debug Builds



## ▼ Swift Compiler - Code Generation


Name

 Game

## ▼ Compilation Mode

<Multiple values> ⌵

Debug

 Incremental ⌵

Release

Whole Module ⌵

---

What's New in Swift

WWDC 2018

---

Behind the Scenes of the Xcode Build Process

Hall 2

Friday 2:00PM

---

## **Increasing Build Efficiency**

---

Parallelizing your build process

Measuring your build time

Dealing with complex expressions

## **Reducing the Work on Rebuilds**

Declaring script inputs and outputs

Understanding dependencies in Swift

Limiting your Objective-C/Swift interface

Dealing with complex expressions

Understanding dependencies in Swift

Limiting your Objective-C/Swift interface

When a build takes a long time,  
there is often a key piece of information  
you can add to make it better.



# Dealing with Complex Expressions

Increasing build efficiency

# Use Explicit Types for Complex Properties

```
struct ContrivedExample {  
    var bigNumber = [4, 3, 2].reduce(1) {  
        soFar, next in  
        pow(next, soFar)  
    }  
}
```

# Use Explicit Types for Complex Properties

```
struct ContrivedExample {  
    var bigNumber: Double = [4, 3, 2].reduce(1) {  
        soFar, next in  
        pow(next, soFar)  
    }  
}
```

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
  
    }  
}
```

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
        soFar != nil && next != nil ? soFar! + next! :  
            (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
        soFar != nil && next != nil ? soFar! + next! :  
            (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```

error: the compiler is unable to type-check  
this expression in reasonable time

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
        soFar != nil && next != nil ? soFar! + next! :  
            (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        (soFar: Int?, next: Int?) -> Int? in  
        soFar != nil && next != nil ? soFar! + next! :  
            (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```



# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
        soFar != nil && next != nil ? soFar! + next! :  
            (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
        soFar != nil && next != nil ? soFar! + next! :  
        (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
        soFar != nil && next != nil ? soFar! + next! :  
            (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```

# Provide Types in Complex Closures

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
        soFar != nil && next != nil ? soFar! + next! :  
            (soFar != nil ? soFar! : (next != nil ? next! : nil))  
    }  
}
```

# Break Apart Complex Expressions

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
  
    }  
}
```

# Break Apart Complex Expressions

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
            if soFar != nil && next != nil { return soFar! + next! }  
            if soFar != nil { return soFar! }  
            if next != nil { return next! }  
            return nil  
        }  
    }  
}
```

# Break Apart Complex Expressions

```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {  
    return [i, j, k].reduce(0) {  
        soFar, next in  
            if let soFar = soFar {  
                if let next = next { return soFar + next }  
                return soFar  
            } else {  
                return next  
            }  
        }  
    }  
}
```

# Use AnyObject Methods and Properties Sparingly

```
weak var delegate: AnyObject?  
func reportSuccess() {  
    delegate?.myOperationDidSucceed(self)  
}
```



# Use AnyObject Methods and Properties Sparingly

```
weak var delegate: AnyObject?  
func reportSuccess() {  
    delegate?.myOperationDidSucceed(self)  
}
```

# Use AnyObject Methods and Properties Sparingly

```
weak var delegate: AnyObject?  
func reportSuccess() {  
    delegate?.myOperationDidSucceed(self)  
}
```

# Use AnyObject Methods and Properties Sparingly

```
weak var delegate: AnyObject?  
func reportSuccess() {  
    delegate?.myOperationDidSucceed(self)  
}
```



# Use AnyObject Methods and Properties Sparingly

Define a protocol instead

```
weak var delegate: AnyObject?  
func reportSuccess() {  
    delegate?.myOperationDidSucceed(self)  
}
```

```
protocol MyOperationDelegate: class {  
    func myOperationDidSucceed(_ operation: MyOperation)  
}
```

# Use AnyObject Methods and Properties Sparingly

Define a protocol instead

```
weak var delegate: MyOperationDelegate?
func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}


protocol MyOperationDelegate: class {
    func myOperationDidSucceed(_ operation: MyOperation)
}
```

# Use AnyObject Methods and Properties Sparingly

Define a protocol instead

```
weak var delegate: MyOperationDelegate?
func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}

protocol MyOperationDelegate: class {
    func myOperationDidSucceed(_ operation: MyOperation)
}
```

A white line starts from the `self` parameter in the `myOperationDidSucceed` call within the `reportSuccess` function. It extends horizontally to the right, then turns vertically downwards, and finally turns horizontally to the left, ending with an arrowhead pointing to the `myOperationDidSucceed` method signature in the `MyOperationDelegate` protocol definition.

# Understanding Dependencies in Swift

Reducing the work on rebuilds

# Incremental Builds Are File-Based

```
struct Point {  
    var x, y: Double  
}
```





# Incremental Builds Are File-Based

```
struct Point {  
    var x, y: Double  
}
```



```
let point = Point(x: 3.0, y: 4.0)
```

```
let distance =  
    sqrt(point.x * point.x +  
         point.y * point.y)
```



# Incremental Builds Are File-Based

```
struct Point {  
    var x, y: Double  
}
```



```
let point = Point(x: 3.0, y: 4.0)
```

```
let distance =  
    sqrt(point.x * point.x +  
         point.y * point.y)
```



# Incremental Builds Are File-Based

```
struct Point {  
    var x, y: Double  
    init(x: Double, y: Double) {  
        assert(x.isNormal)  
        assert(y.isNormal)  
        self.x = x  
        self.y = y  
    }  
}
```



```
let point = Point(x: 3.0, y: 4.0)  
  
let distance =  
    sqrt(point.x * point.x +  
        point.y * point.y)
```



# Incremental Builds Are File-Based

```
struct Point {  
    var x, y: Double  
    init(x: Double, y: Double) {  
        assert(x.isNormal)  
        assert(y.isNormal)  
        self.x = x  
        self.y = y  
    }  
}
```



```
let point = Point(x: 3.0, y: 4.0)  
  
let distance =  
    sqrt(point.x * point.x +  
        point.y * point.y)
```



# Incremental Builds Are File-Based

Changes in function bodies do not affect other files

```
struct Point {  
    var x, y: Double  
    init(x: Double, y: Double) {  
        assert(x.isFinite)  
        assert(y.isFinite)  
        self.x = x  
        self.y = y  
    }  
}
```



```
let point = Point(x: 3.0, y: 4.0)
```

```
let distance =  
    sqrt(point.x * point.x +  
        point.y * point.y)
```



# Incremental Builds Are File-Based

Changes in function bodies do not affect other files

```
struct Point {  
    var x, y: Double  
    init(x: Double, y: Double) {  
        assert(x.isFinite)  
        assert(y.isFinite)  
        self.x = x  
        self.y = y  
    }  
}
```



```
let point = Point(x: 3.0, y: 4.0)  
  
let distance =  
    sqrt(point.x * point.x +  
        point.y * point.y)
```



# Incremental Builds Are File-Based

Unrelated changes outside function bodies can still result in rebuilding

```
struct Point {  
    var x, y: Double  
    init(x: Double, y: Double) {  
        assert(x.isFinite)  
        assert(y.isFinite)  
        self.x = x  
        self.y = y  
    }  
}
```



```
struct PathSegment {  
    var start, end: Point  
}
```

```
let point = Point(x: 3.0, y: 4.0)
```

```
let distance =  
    sqrt(point.x * point.x +  
        point.y * point.y)
```



# Incremental Builds Are File-Based

Unrelated changes outside function bodies can still result in rebuilding

```
struct Point {  
    var x, y: Double  
    init(x: Double, y: Double) {  
        assert(x.isFinite)  
        assert(y.isFinite)  
        self.x = x  
        self.y = y  
    }  
}
```

```
struct PathSegment {  
    var start, end: Point  
}
```

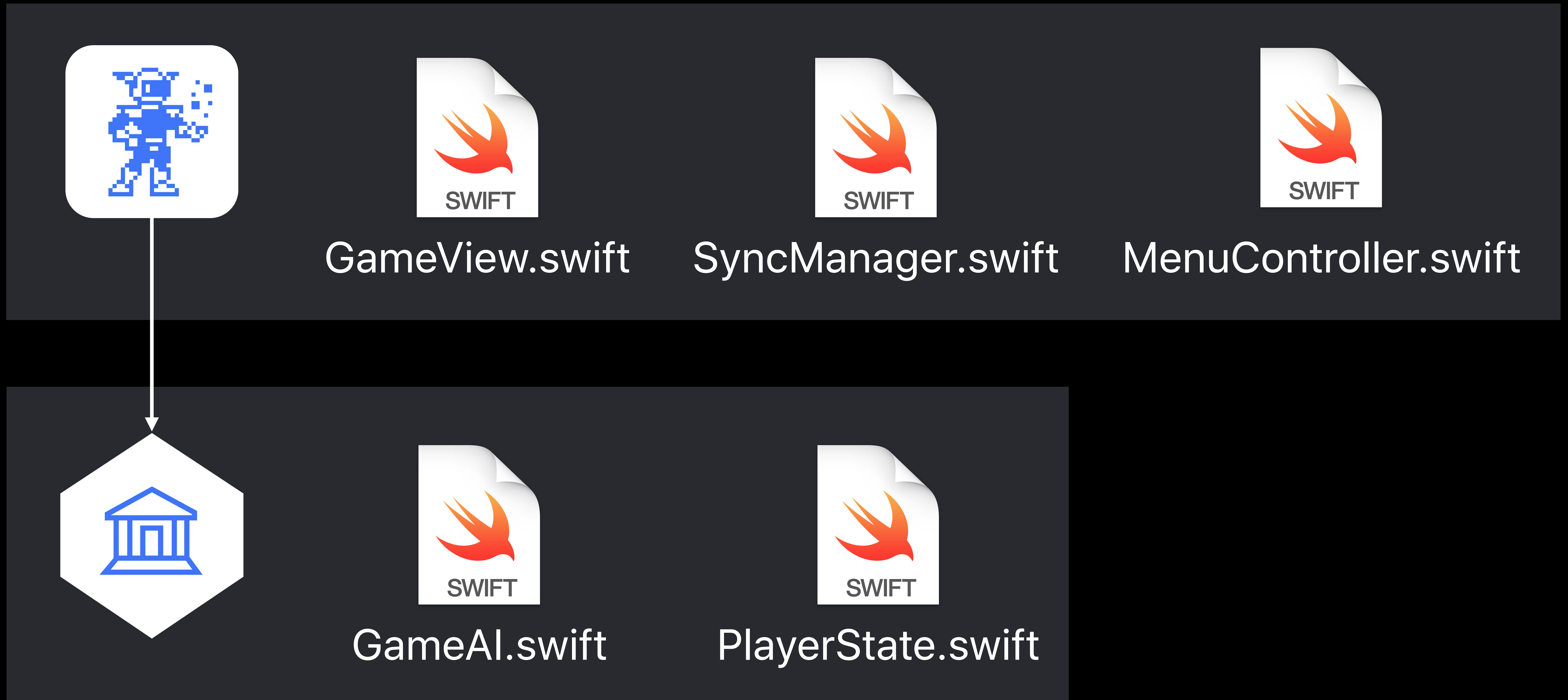


```
let point = Point(x: 3.0, y: 4.0)  
  
let distance =  
    sqrt(point.x * point.x +  
        point.y * point.y)
```

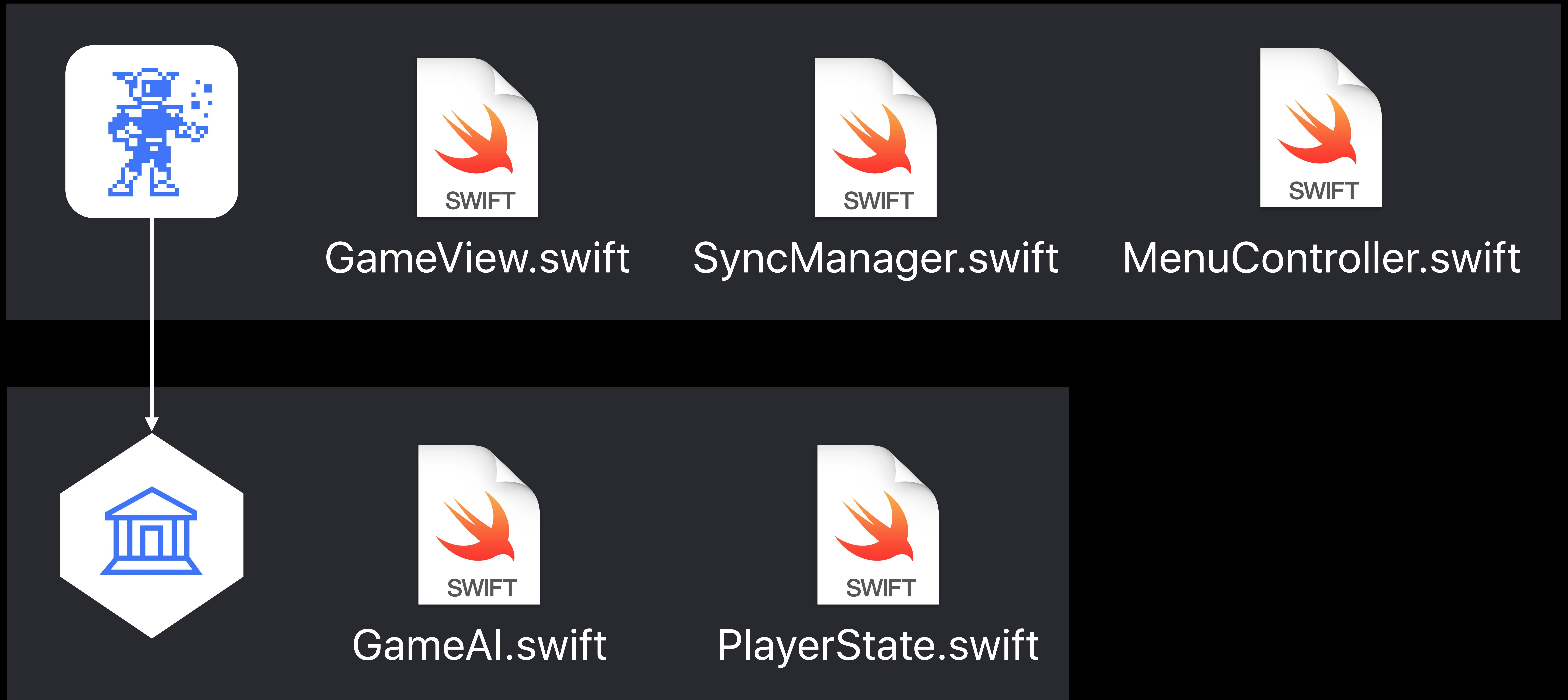




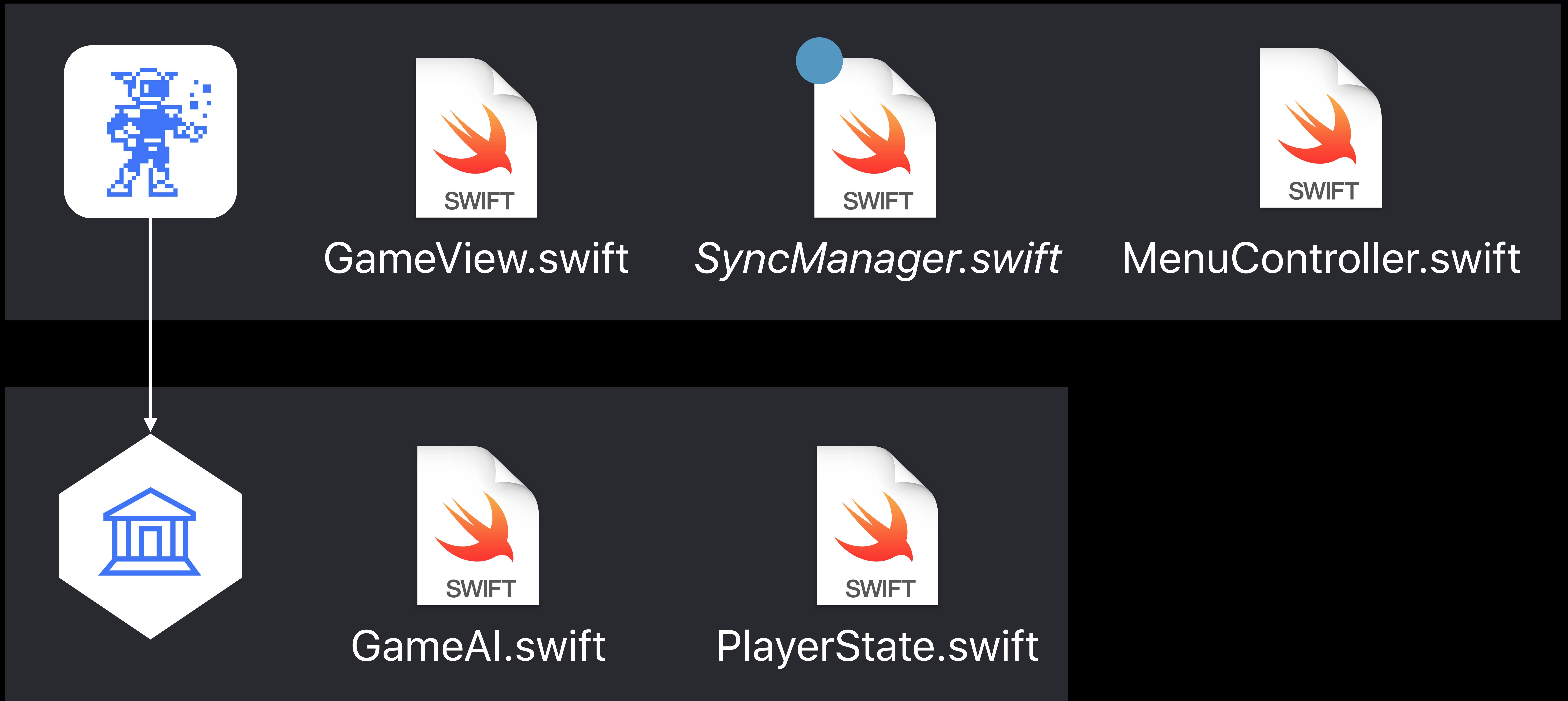
# Dependencies Within a Target Are Per-File



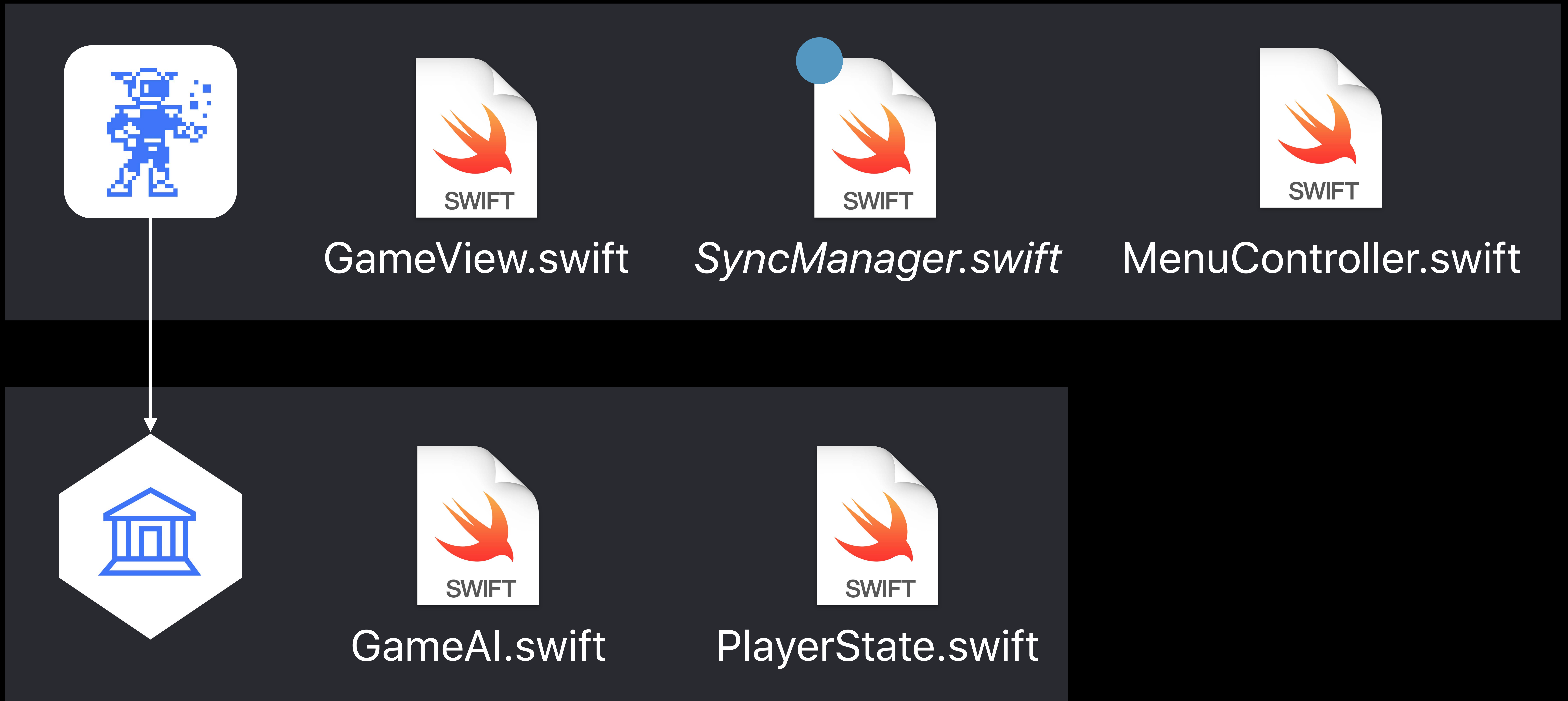
# Dependencies Within a Target Are Per-File



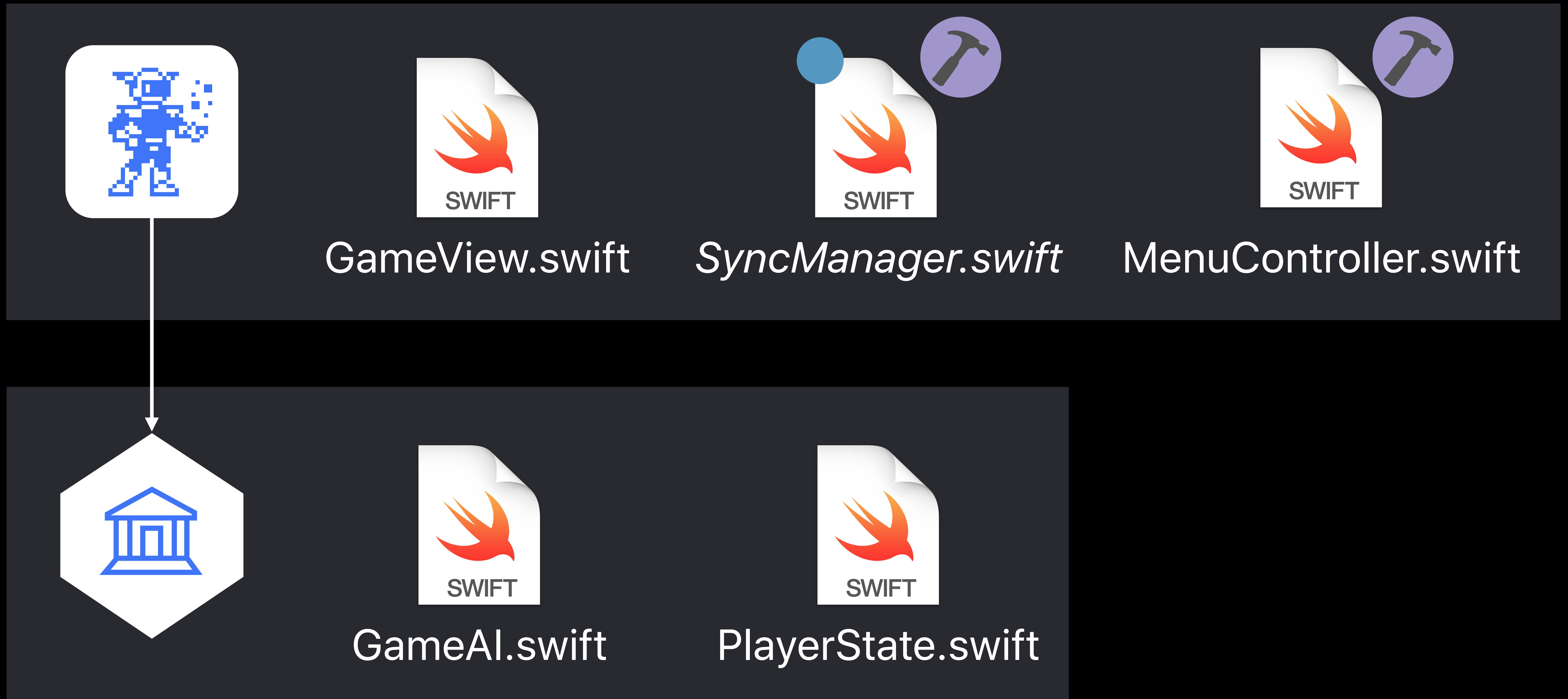
# Dependencies Within a Target Are Per-File



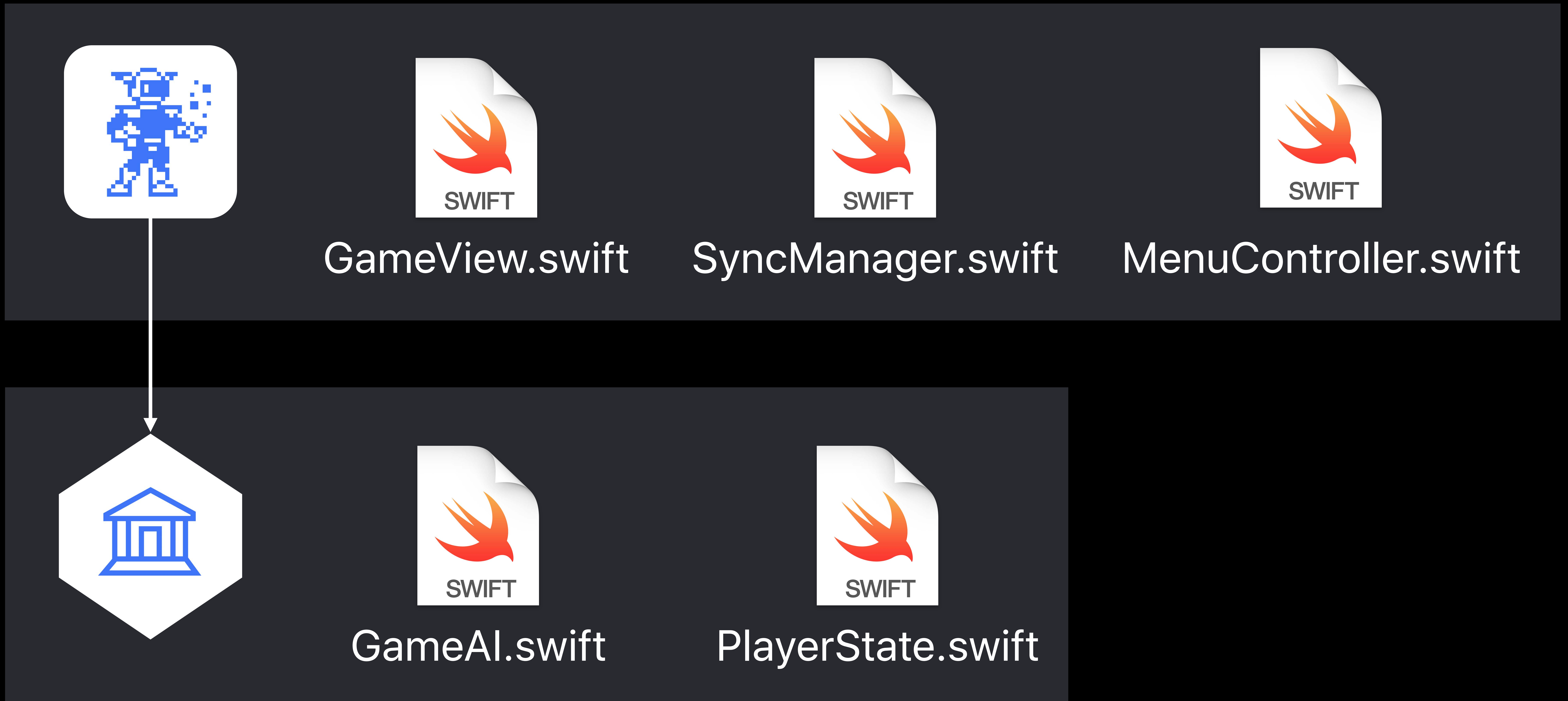
# Dependencies Within a Target Are Per-File



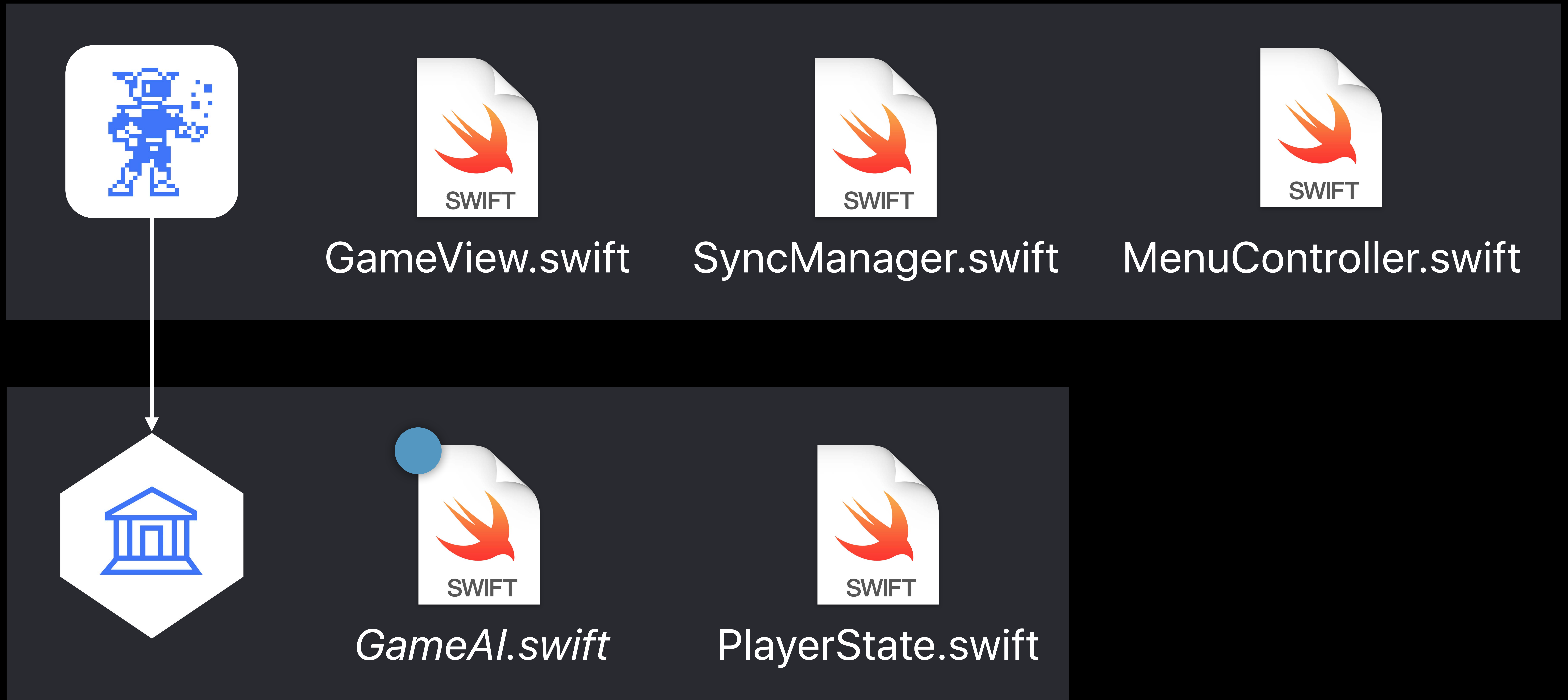
# Dependencies Within a Target Are Per-File



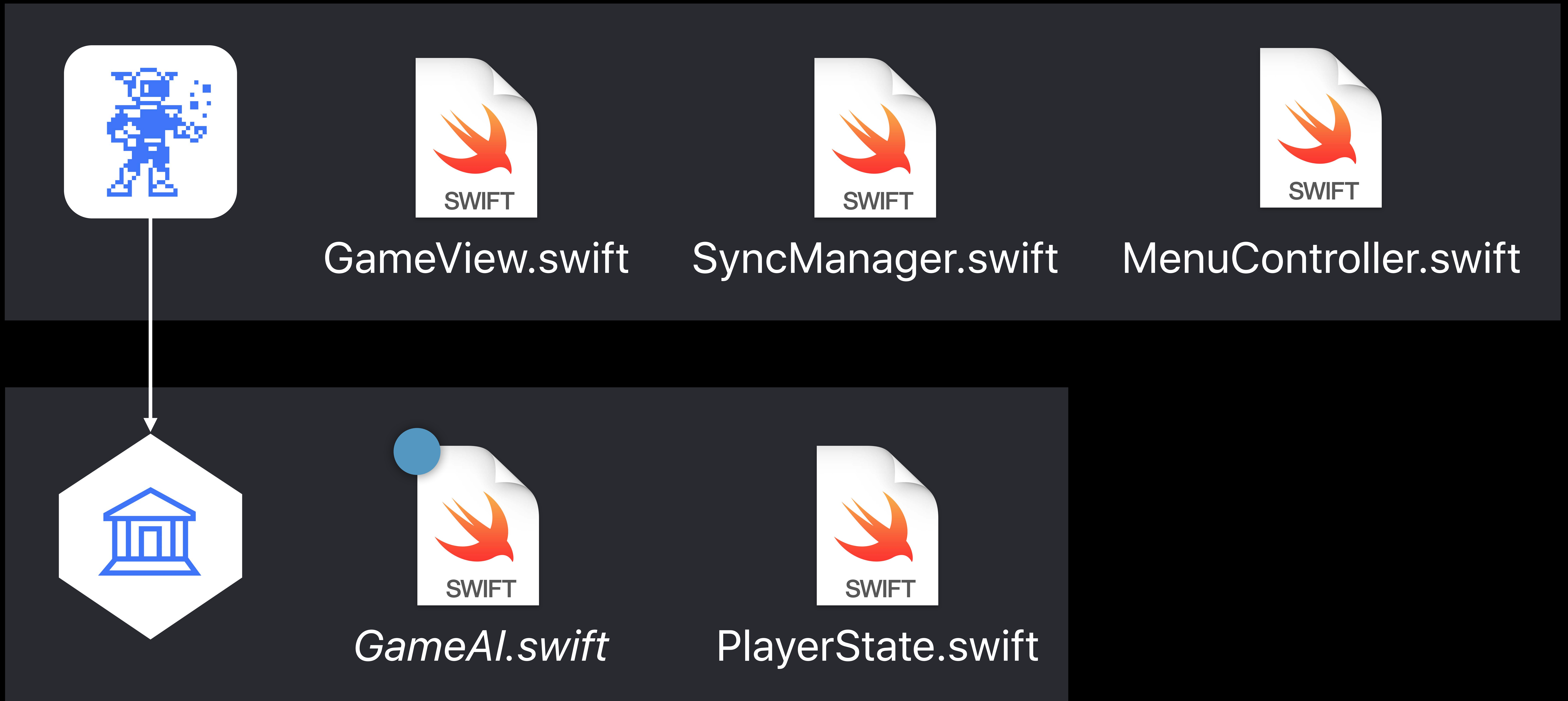
# Cross-Target Dependencies Are Coarse-Grained



# Cross-Target Dependencies Are Coarse-Grained

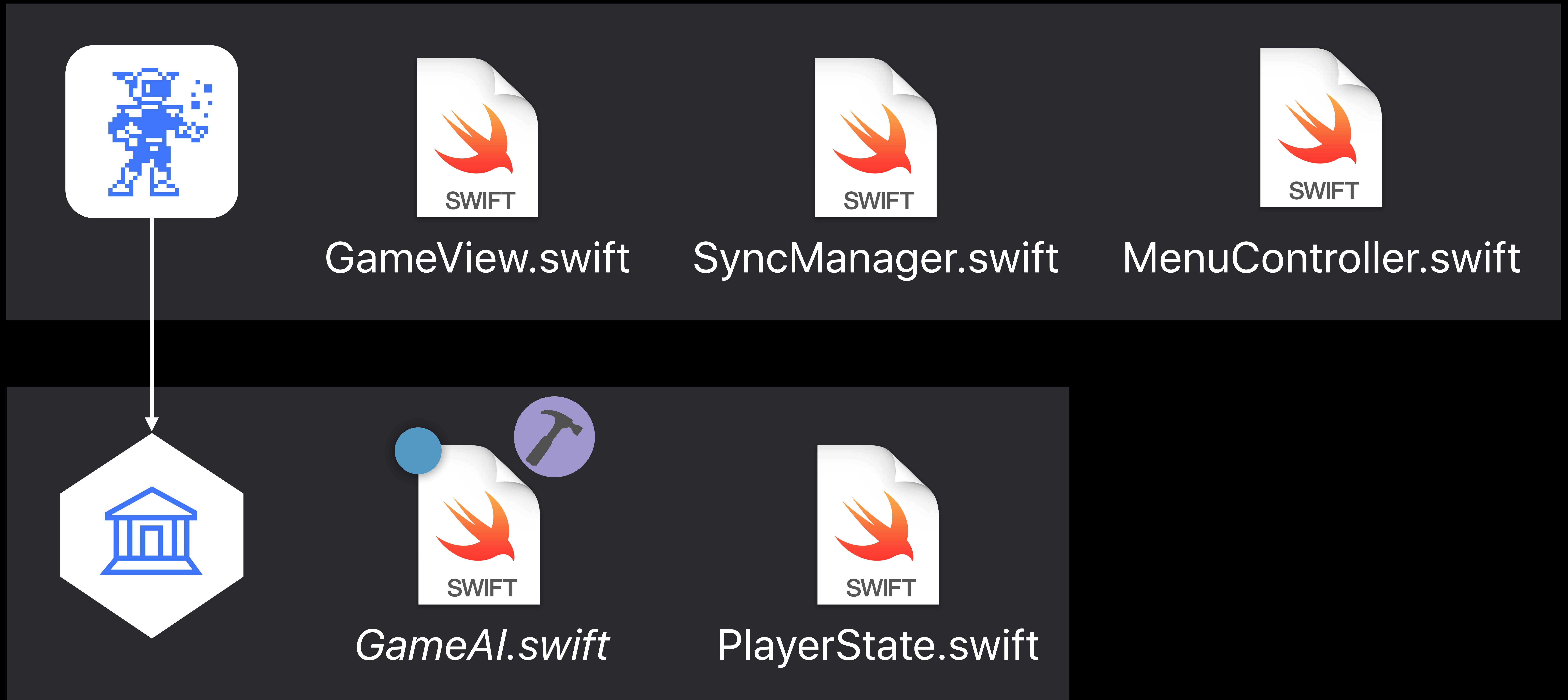


# Cross-Target Dependencies Are Coarse-Grained

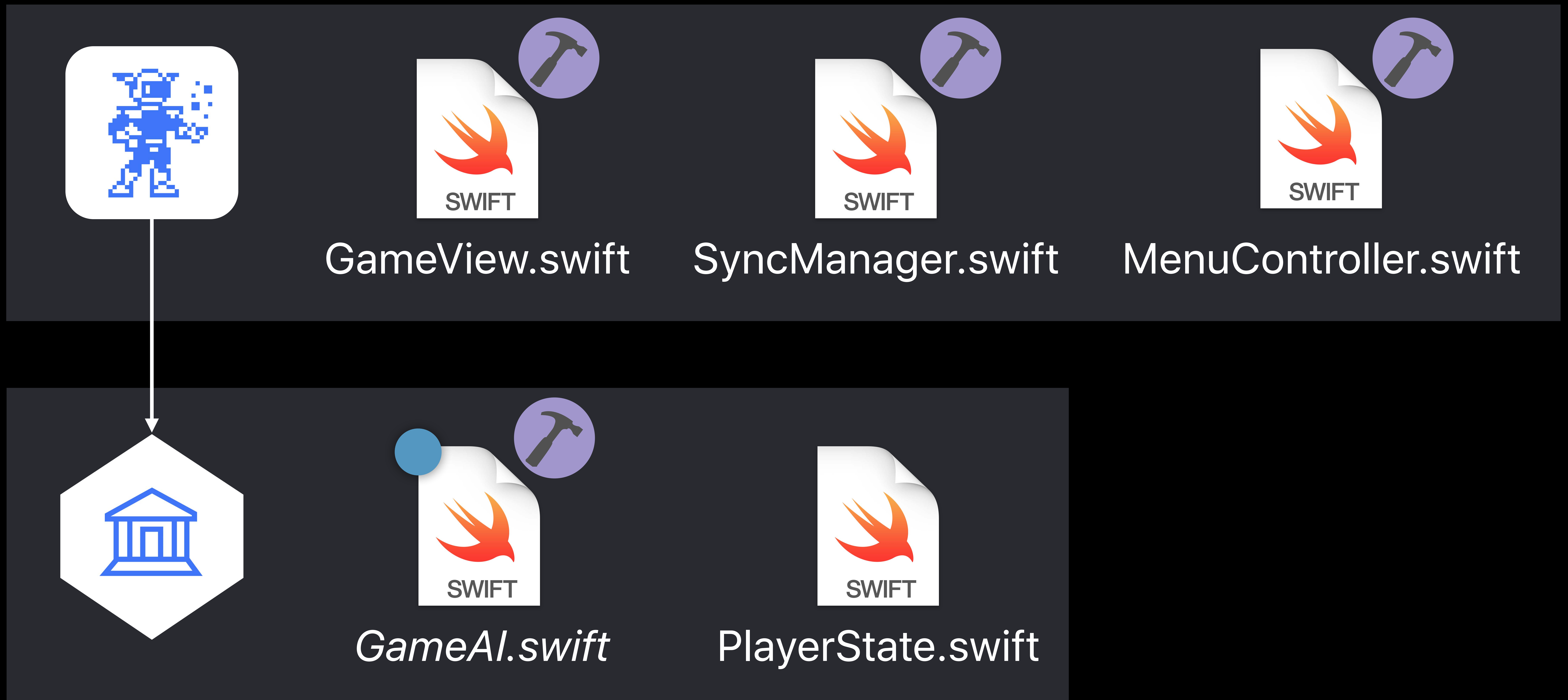




# Cross-Target Dependencies Are Coarse-Grained



# Cross-Target Dependencies Are Coarse-Grained



# Swift Dependency Rules

# Swift Dependency Rules

Compiler must be conservative

# Swift Dependency Rules

Compiler must be conservative

Changes in function bodies do not affect the file's interface

# Swift Dependency Rules

Compiler must be conservative

Changes in function bodies do not affect the file's interface

Dependencies within a module are per-file

# Swift Dependency Rules

Compiler must be conservative

Changes in function bodies do not affect the file's interface

Dependencies within a module are per-file

Dependencies across targets are for the whole target

# Limiting Your Objective-C/Swift Interface

Reducing the work on rebuilds

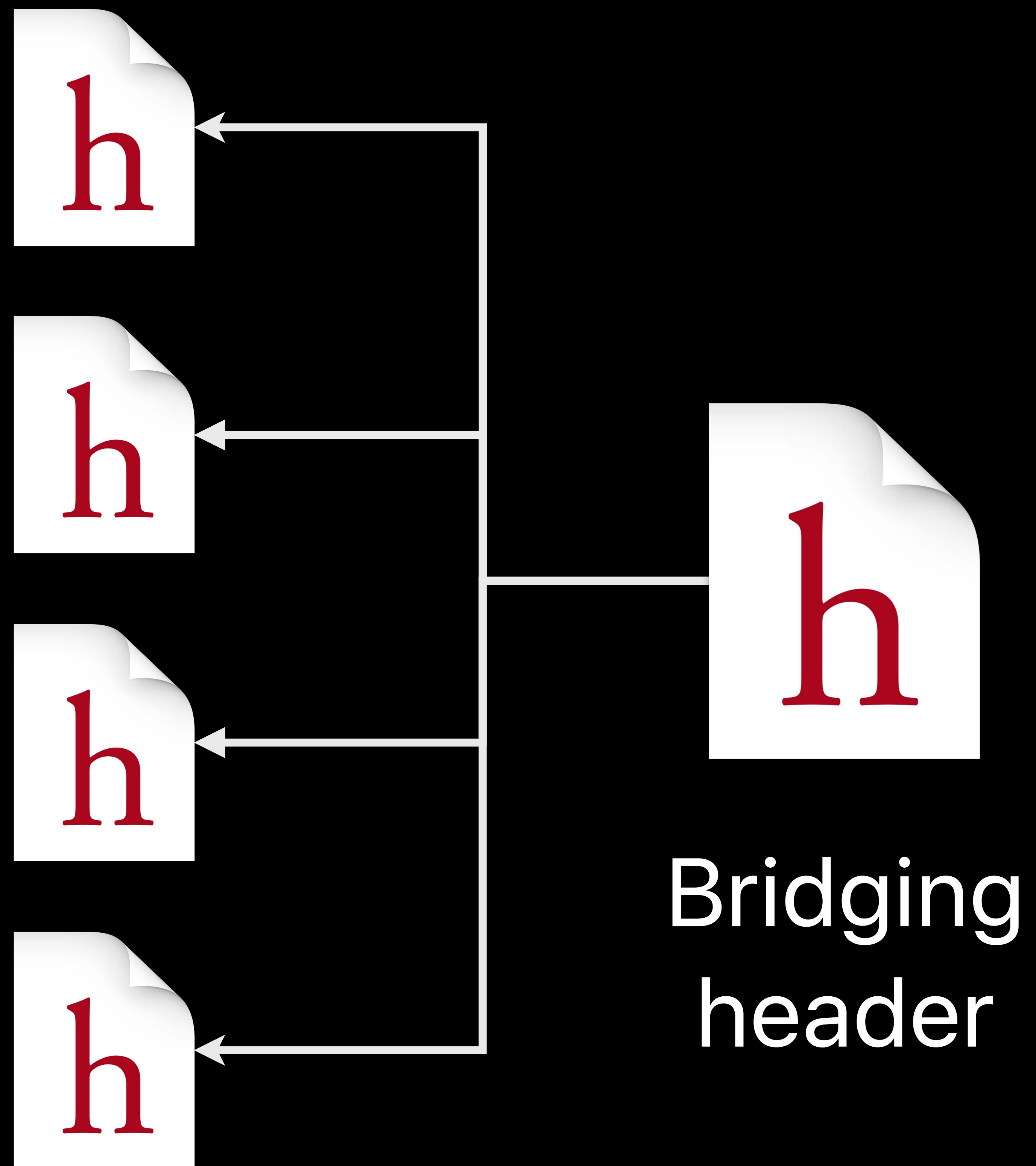


# Mixed-Source App Targets

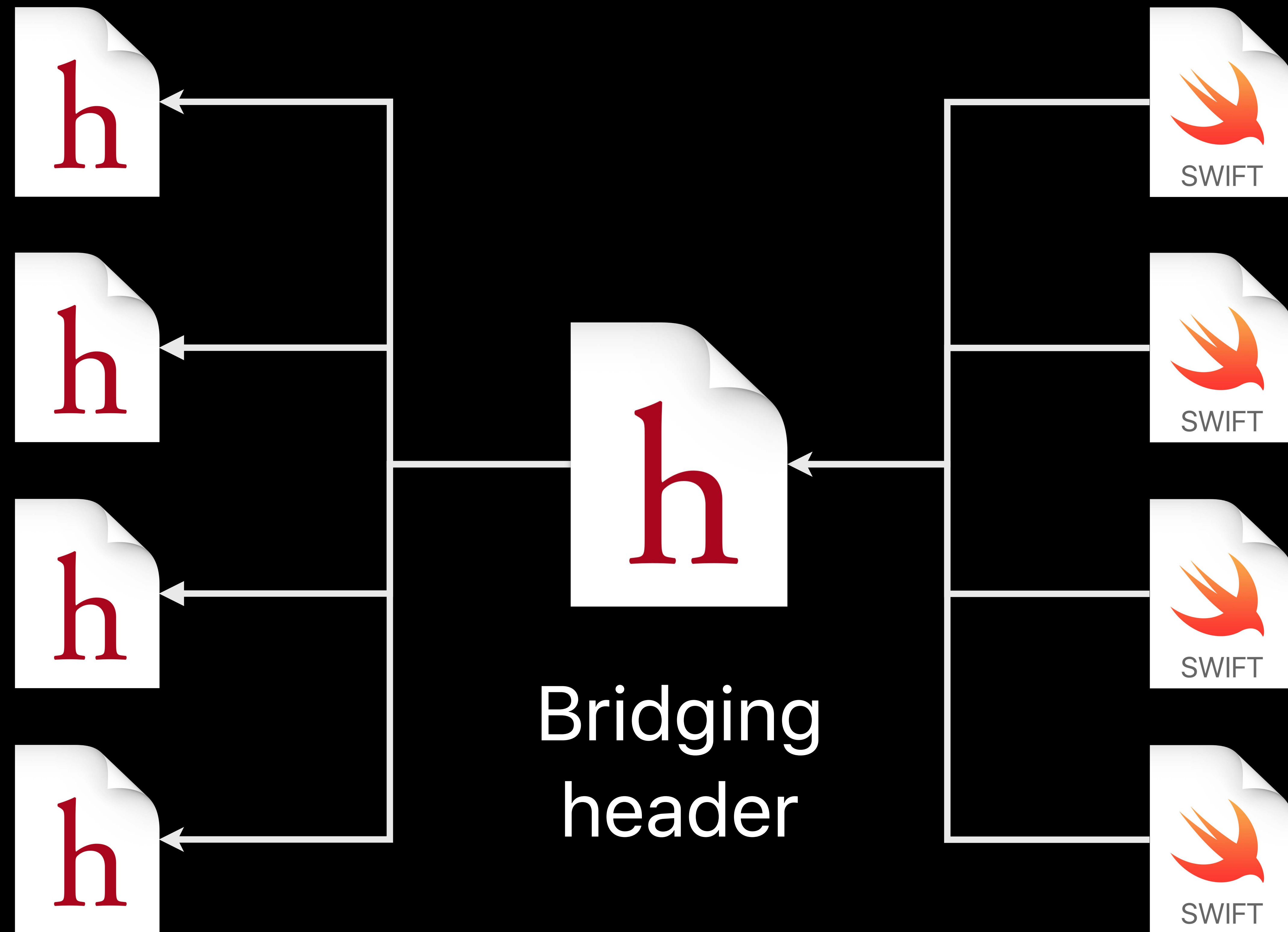
# Mixed-Source App Targets



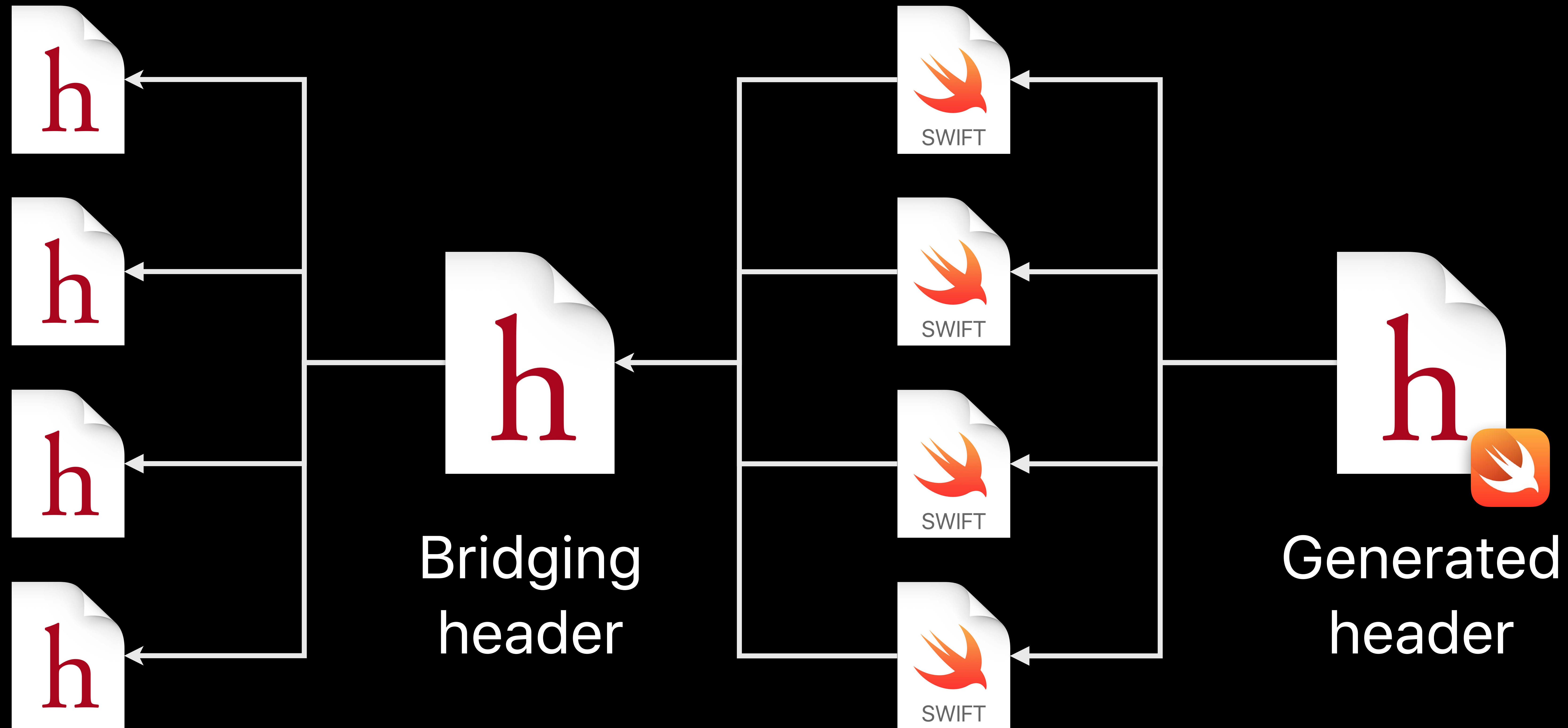
# Mixed-Source App Targets



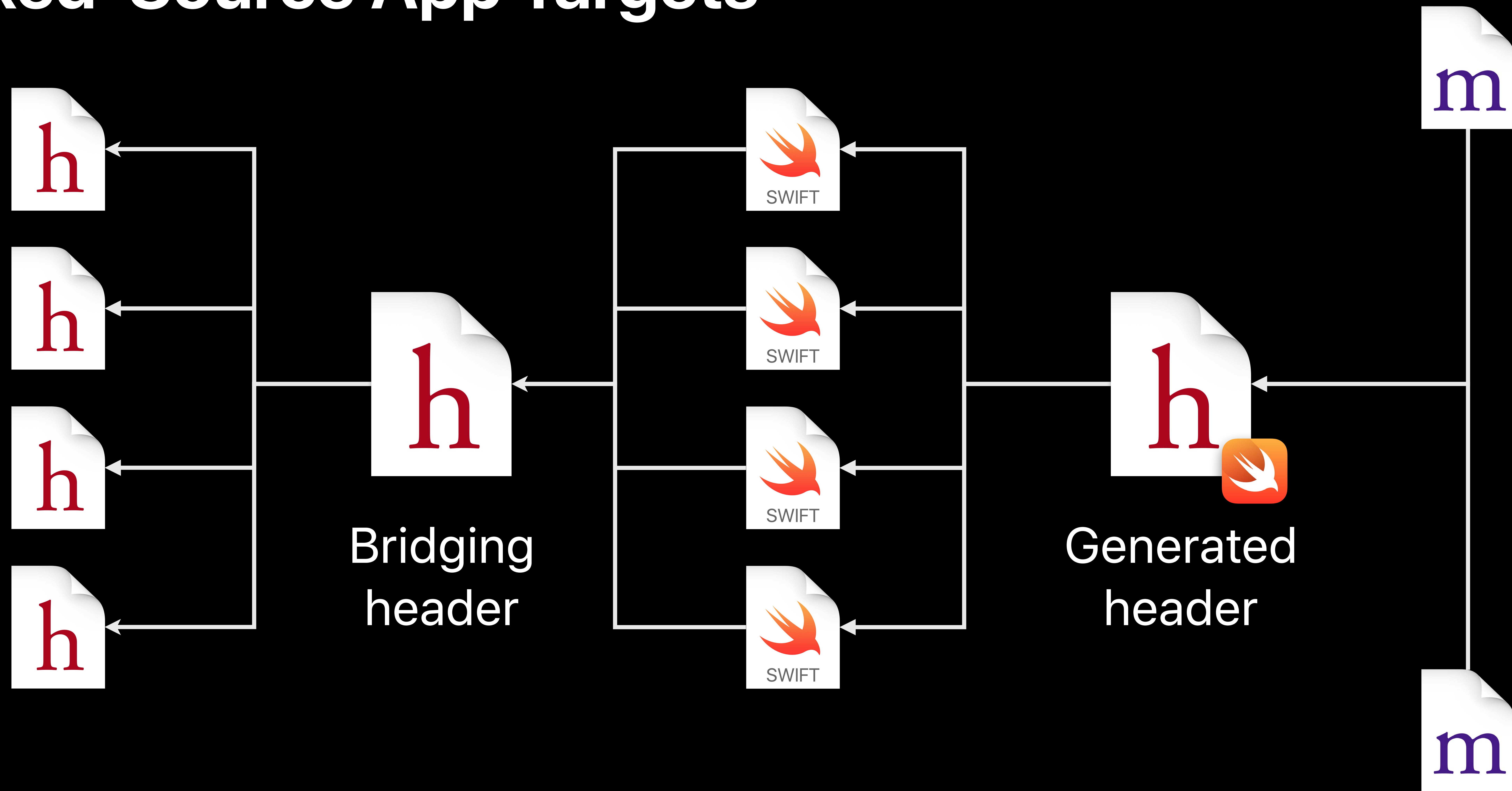
# Mixed-Source App Targets



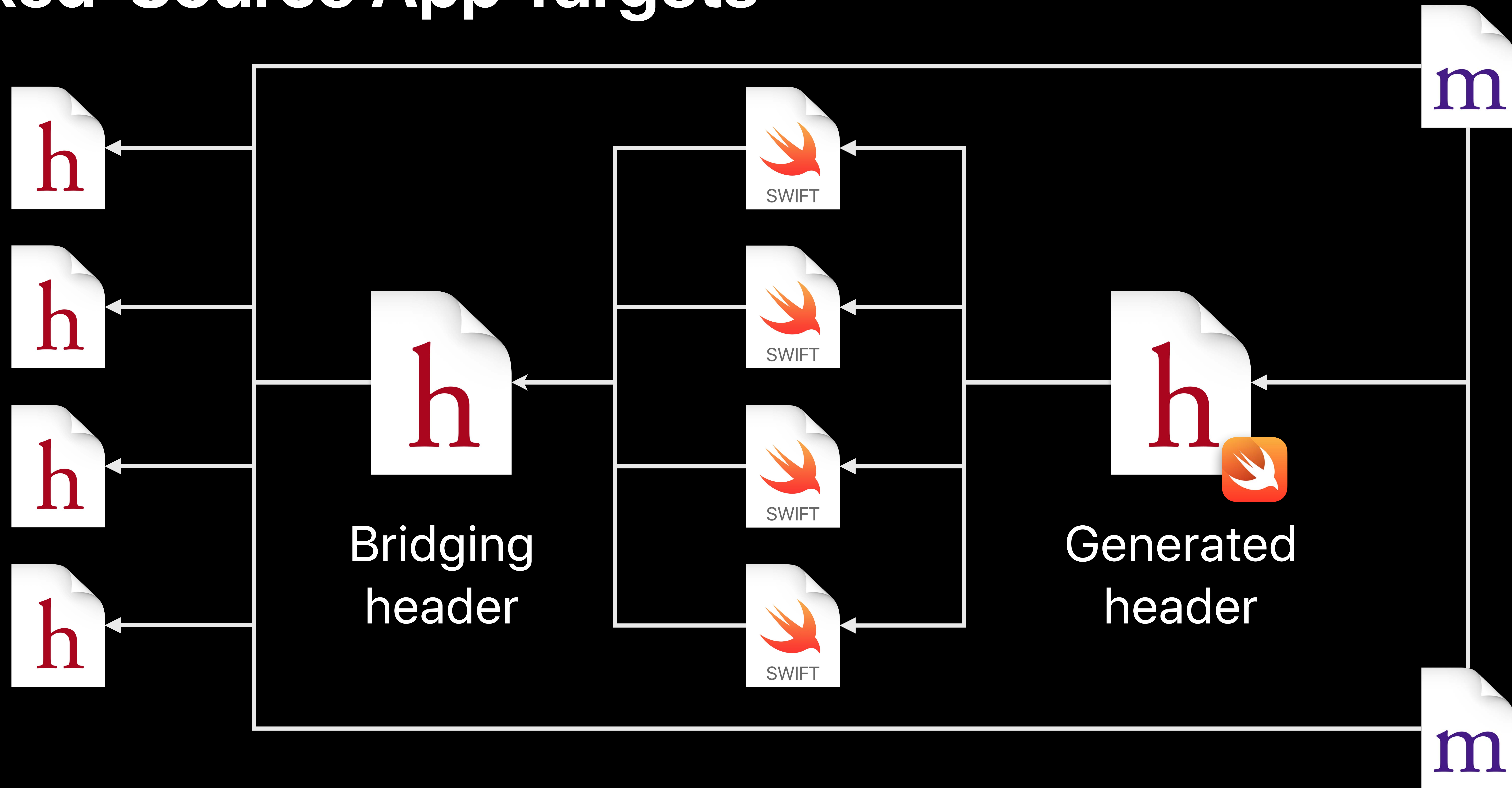
# Mixed-Source App Targets



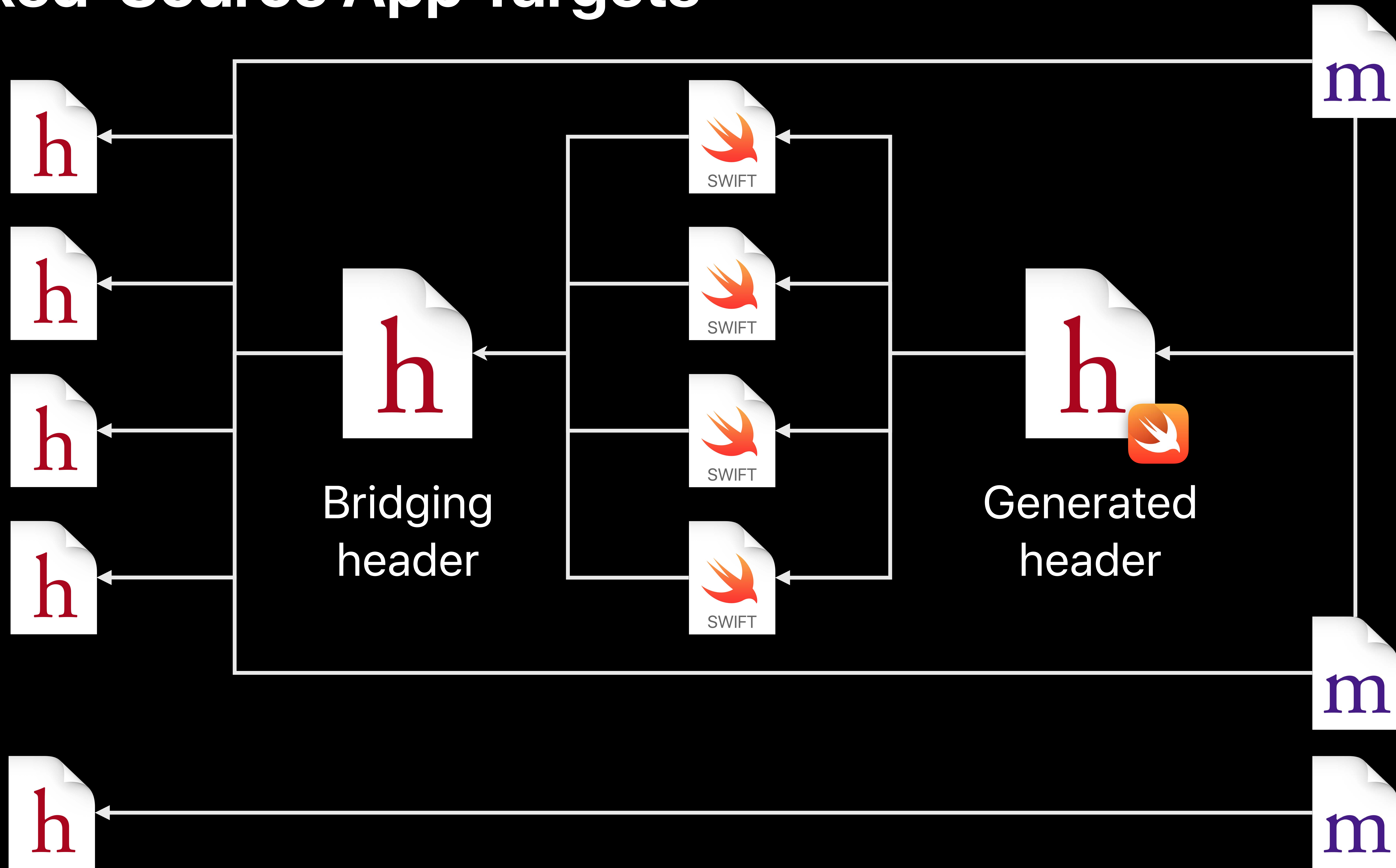
# Mixed-Source App Targets



# Mixed-Source App Targets

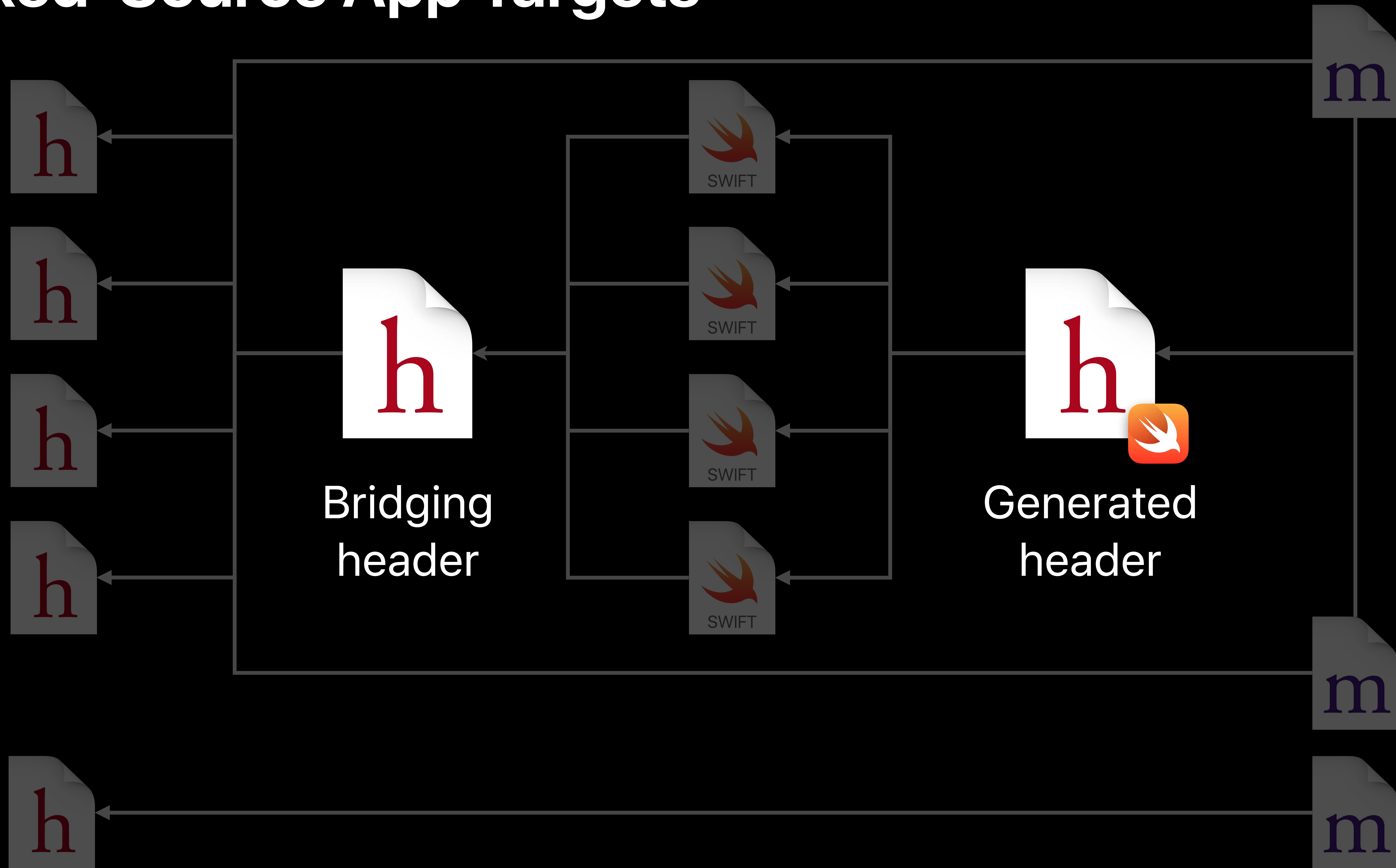


# Mixed-Source App Targets





# Mixed-Source App Targets



# Keep Your Generated Header Minimal

Use `private` when possible

```
class MainViewController: UIViewController {  
    @IBOutlet var statusField: UITextField!  
    @IBAction func close(_ sender: Any?) { ... }  
}
```



# Keep Your Generated Header Minimal

Use `private` when possible

```
class MainViewController: UIViewController {  
    @IBOutlet var statusField: UITextField!  
    @IBAction func close(_ sender: Any?) { ... }  
}
```



```
@property IBOutlet UITextField *statusField;  
- (IBAction)close:(id)sender;
```



# Keep Your Generated Header Minimal

Use `private` when possible

```
class MainViewController: UIViewController {  
    @IBOutlet private var statusField: UITextField!  
    @IBAction private func close(_ sender: Any?) { ... }  
}
```



```
@property IBOutlet UITextField *statusField;  
- (IBAction)close:(id)sender;
```



# Keep Your Generated Header Minimal

Use `private` when possible

```
class MainViewController: UIViewController {  
    @IBOutlet private var statusField: UITextField!  
    @IBAction private func close(_ sender: Any?) { ... }  
}
```



# Keep Your Generated Header Minimal

Use `private` when possible

```
class MainViewController: UIViewController {  
    @IBOutlet private var statusField: UITextField!  
    @IBAction private func close(_ sender: Any?) { ... }  
}
```



# Keep Your Generated Header Minimal

Use `private` when possible

```
@objc func keyboardWillShow(_: Notification) {  
    // Important keyboard setup code here.  
}  
  
// ...  
NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow(_:)), ...)
```

# Keep Your Generated Header Minimal

Use `private` when possible

```
@objc private func keyboardWillShow(_: Notification) {  
    // Important keyboard setup code here.  
}  
  
// ...  
NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow(_:)), ...)
```



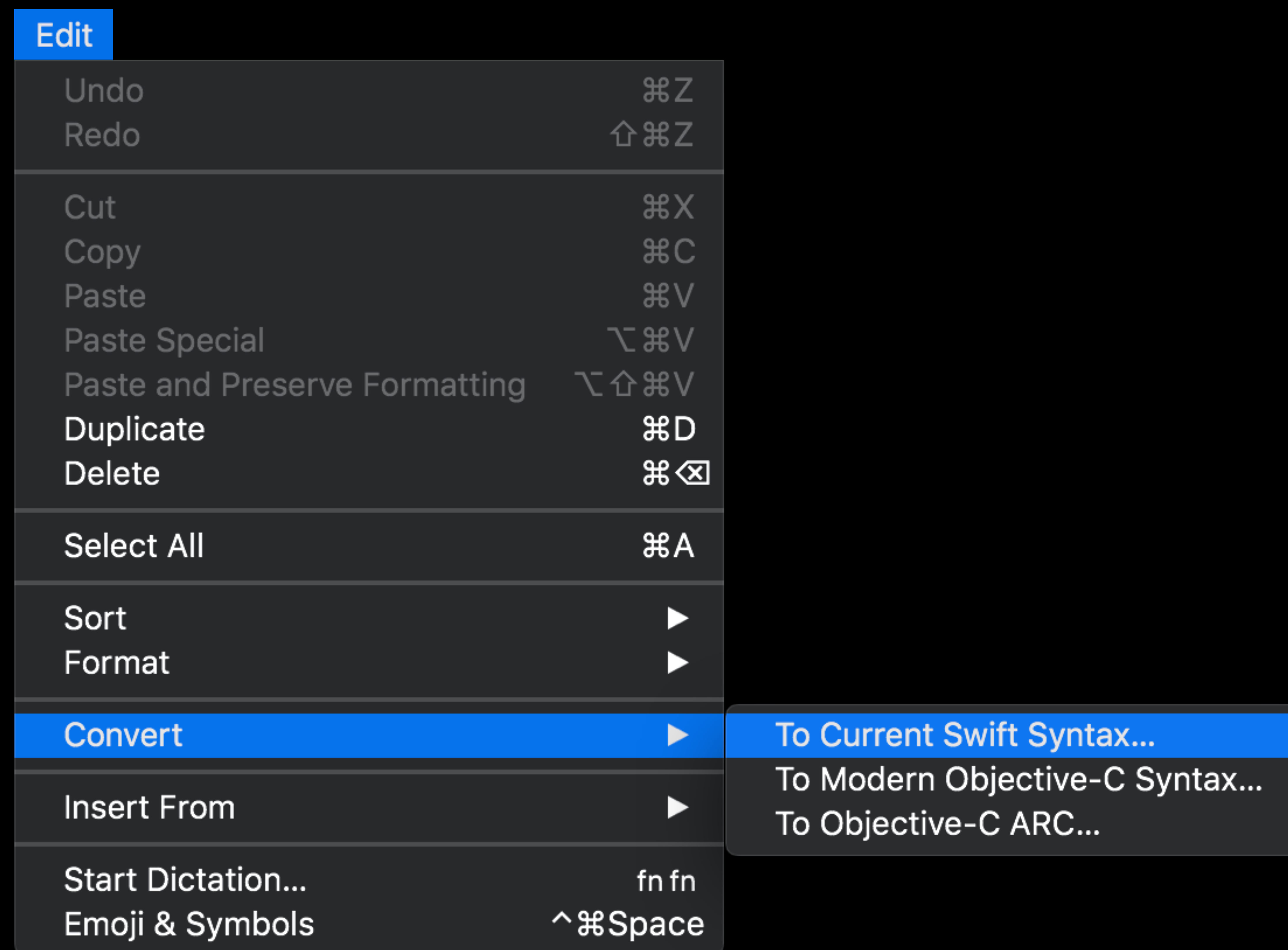
# Keep Your Generated Header Minimal

Use block-based APIs

```
self.observer = NotificationCenter.default.addObserver(  
    forName: UIKeyboardWillShow, object: nil, queue: nil) {  
    // Important keyboard setup code here.  
}
```

# Keep Your Generated Header Minimal

## Migrate to Swift 4



# Keep Your Generated Header Minimal

Turn off "Swift 3 @objc Inference"

## ▼ Swift Compiler - Code Generation

Setting

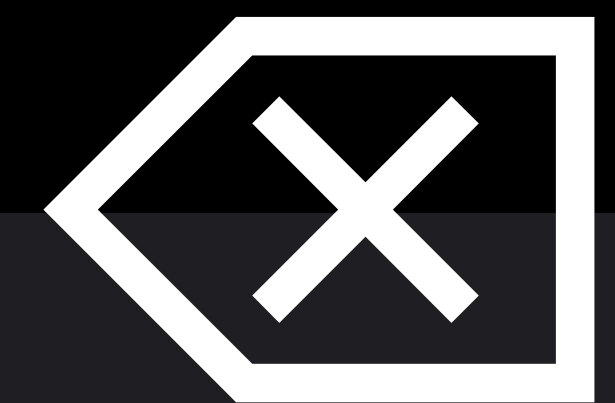
 Game

▶ Swift 3 @objc Inference


On 

# Keep Your Generated Header Minimal

Turn off "Swift 3 @objc Inference"



▼ **Swift Compiler - Code Generation**

Setting |  Game

▶ **Swift 3 @objc Inference** | Default ▾

# Keep Your Bridging Header Minimal

```
// Use this file to import your target's public headers that  
// you would like to expose to Swift.
```

```
#import "MyBackendAPI.h"  
#import "MyViewController.h"  
#import "MyUserDefaultsHelpers.h"
```



MyApp-Bridging-Header.h

# Keep Your Bridging Header Minimal

```
// Use this file to import your target's public headers that  
// you would like to expose to Swift.
```

```
#import "MyBackendAPI.h"
```

```
#import "MyViewController.h"
```

```
#import "MyUserDefaultsHelpers.h"
```



MyApp-Bridging-Header.h

# Keep Your Bridging Header Minimal

```
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"

@interface MyViewController: UIViewController
@property (nonatomic) MyNetworkManager *networkManager;
// ...
@end
```



MyViewController.h

# Keep Your Bridging Header Minimal

```
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"

@interface MyViewController: UIViewController
@property (nonatomic) MyNetworkManager *networkManager;
// ...
@end
```



MyViewController.h



# Keep Your Bridging Header Minimal

```
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"

@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
```



MyViewController.h

# Keep Your Bridging Header Minimal

Use categories to break up your interface

```
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"
@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
```



MyViewController.h

# Keep Your Bridging Header Minimal

Use categories to break up your interface

```
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"
@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
```



MyViewController.h

```
#import "MyViewController.h"
@interface MyViewController ()
@end
```



MyViewController+Internal.h

# Keep Your Bridging Header Minimal

Use categories to break up your interface

```
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"
@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
```



MyViewController.h

```
#import "MyViewController.h"
@interface MyViewController ()
@end
```



MyViewController+Internal.h

# Keep Your Bridging Header Minimal

Use categories to break up your interface

```
#import <UIKit/UIKit.h>
@interface MyViewController: UIViewController
// ...
@end
```



MyViewController.h

```
#import "MyNetworkManager.h"
#import "MyViewController.h"
@interface MyViewController ()
@property (nonnull) MyNetworkManager *networkManager;
@end
```



MyViewController+Internal.h

# Keep Your Bridging Header Minimal

Use categories to break up your interface

```
#import <UIKit/UIKit.h>
@interface MyViewController: UIViewController
// ...
@end
```



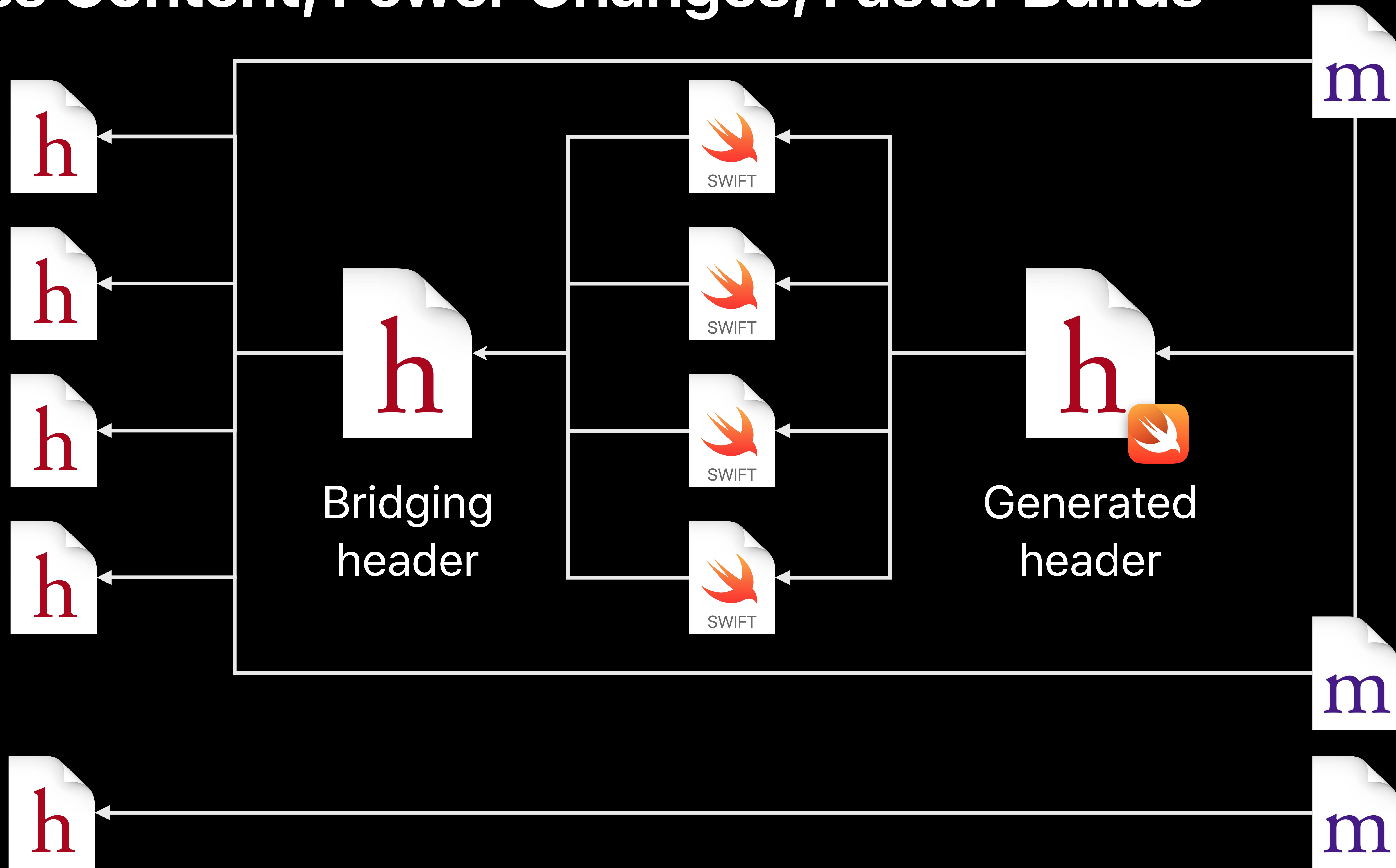
MyViewController.h

```
#import "MyNetworkManager.h"
#import "MyViewController.h"
@interface MyViewController ()
@property (nonnull) MyNetworkManager *networkManager;
@end
```

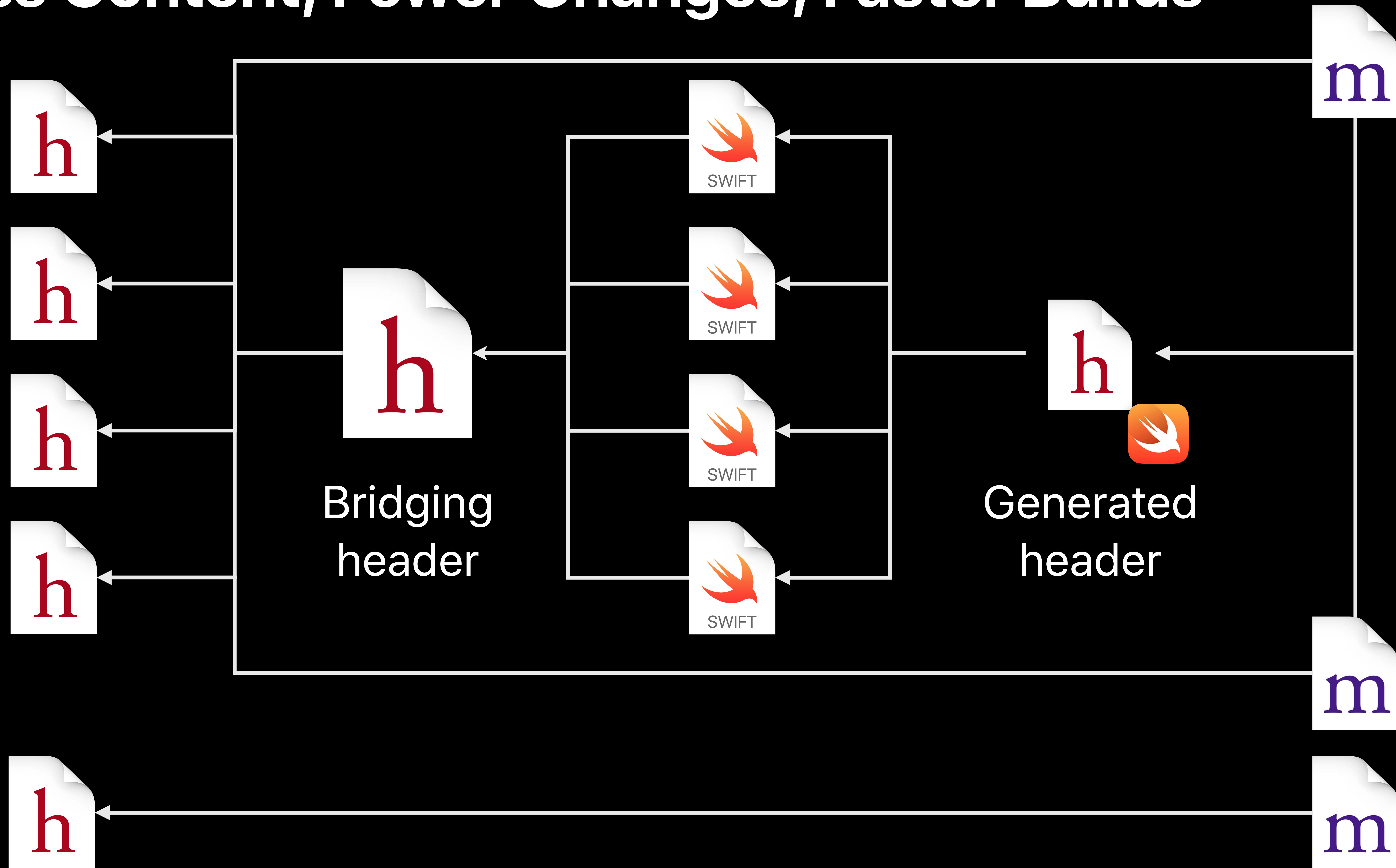


MyViewController.m

# Less Content, Fewer Changes, Faster Builds

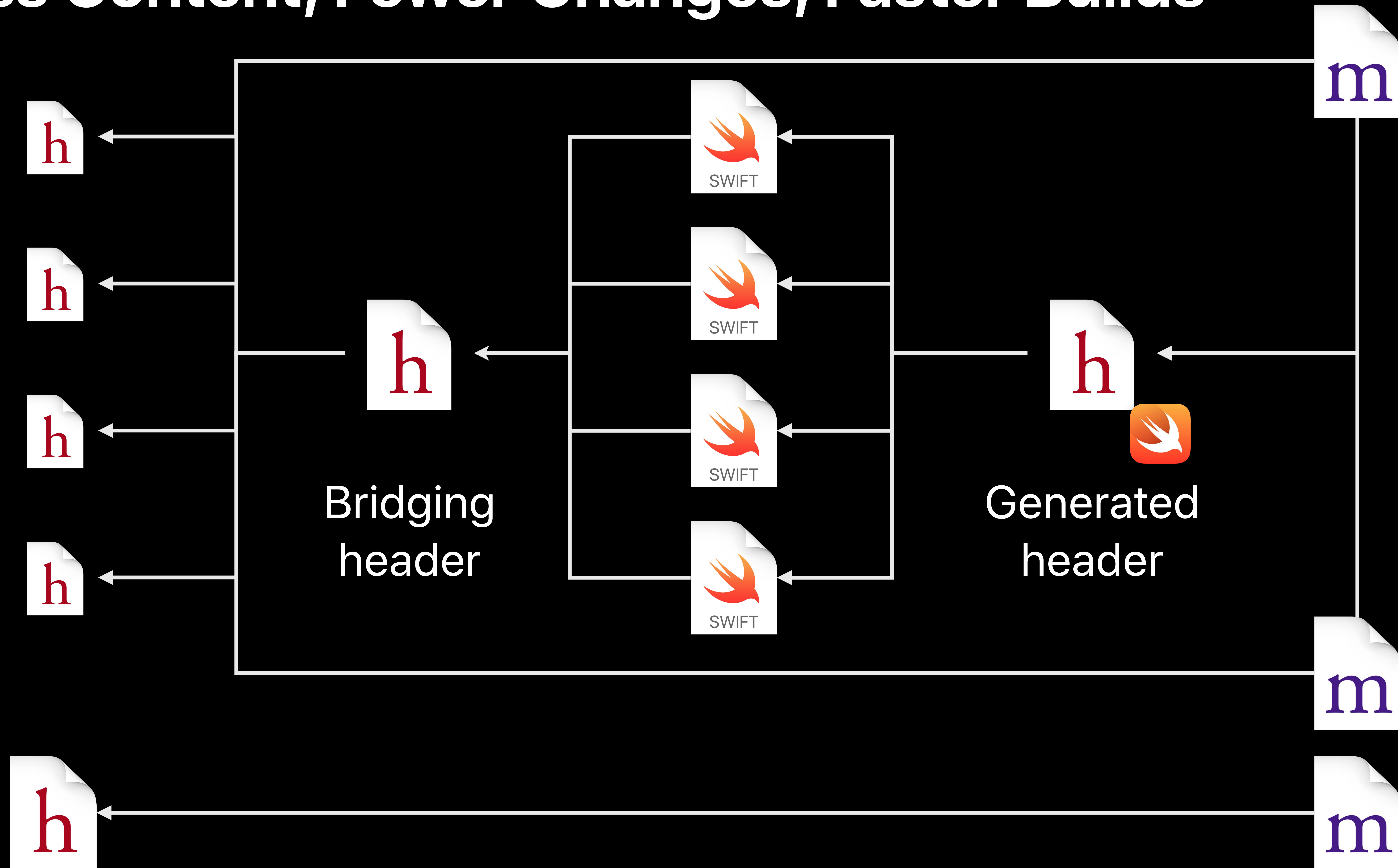


# Less Content, Fewer Changes, Faster Builds





# Less Content, Fewer Changes, Faster Builds



# Summary

# Building Faster in Xcode

## Increasing Build Efficiency

---

Parallelizing your build process

Measuring your build time

Dealing with complex expressions

## Reducing the Work on Rebuilds

Declaring script inputs and outputs

Understanding dependencies in Swift

Limiting your Objective-C/Swift interface

# More Information

<https://developer.apple.com/wwdc18/408>

---

Building Your App with Xcode

Technology Lab 8

Thursday 12:00PM

---

Building Your App with Xcode

Technology Lab 8

Friday 3:00PM

---

 **WWDC18**