GDC 12

CUTTING THE PIPE

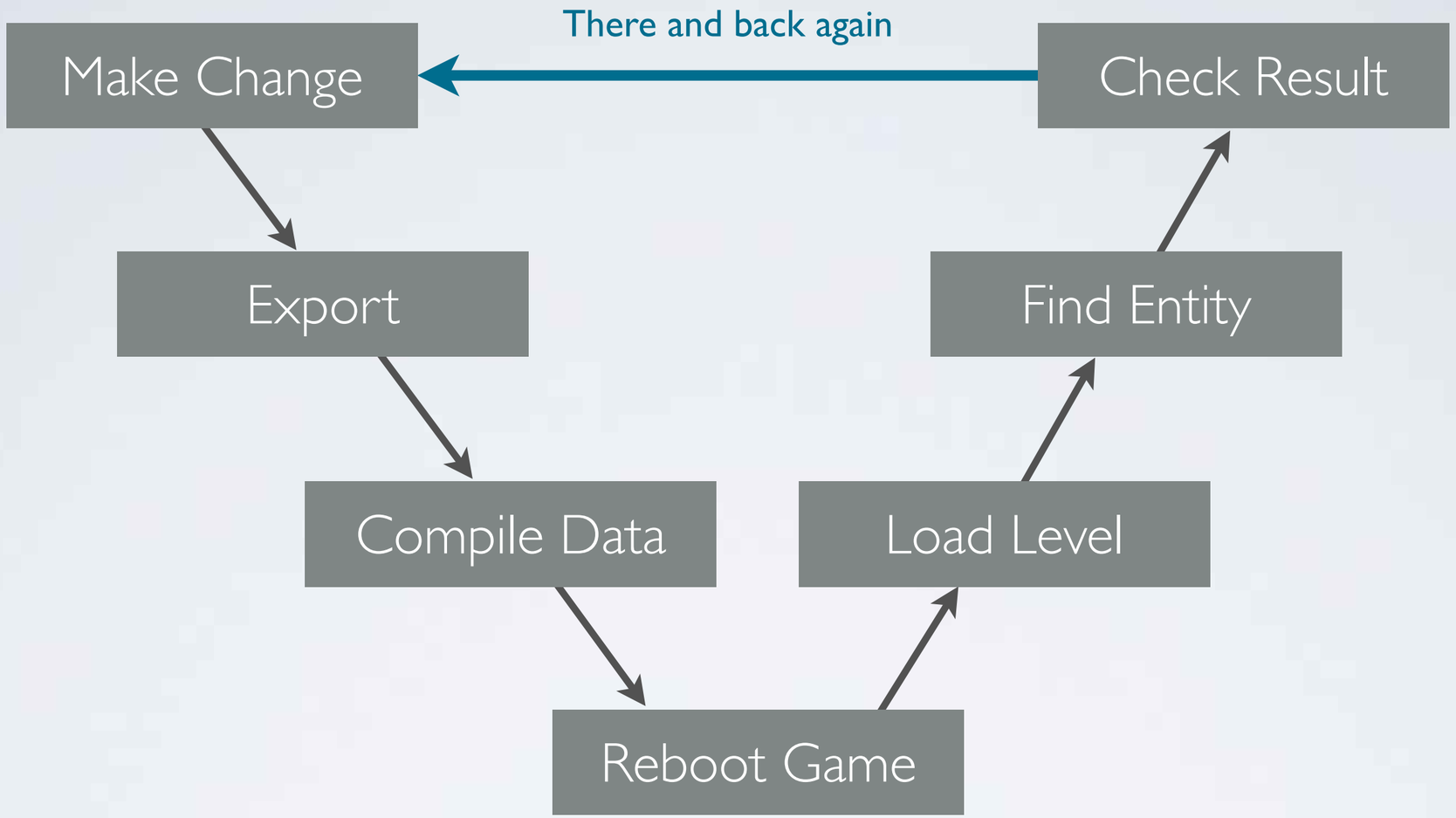
Achieving Sub-Second Iteration Times

Niklas Frykholm

Bitsquid

bitsquid

GAME DEVELOPERS CONFERENCE 2012
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9



The Iteration Treadmill

Minutes (hours?) until a change can be seen in-game

Why Faster Iteration Times?

- Productivity
  - Time lost waiting for builds
- Quality
  - More tweaking
  - Assets tested in-game on console
- Note: This talk is about optimizing pipeline *latency* not *throughput*
  - Time required to update a single dirty resource

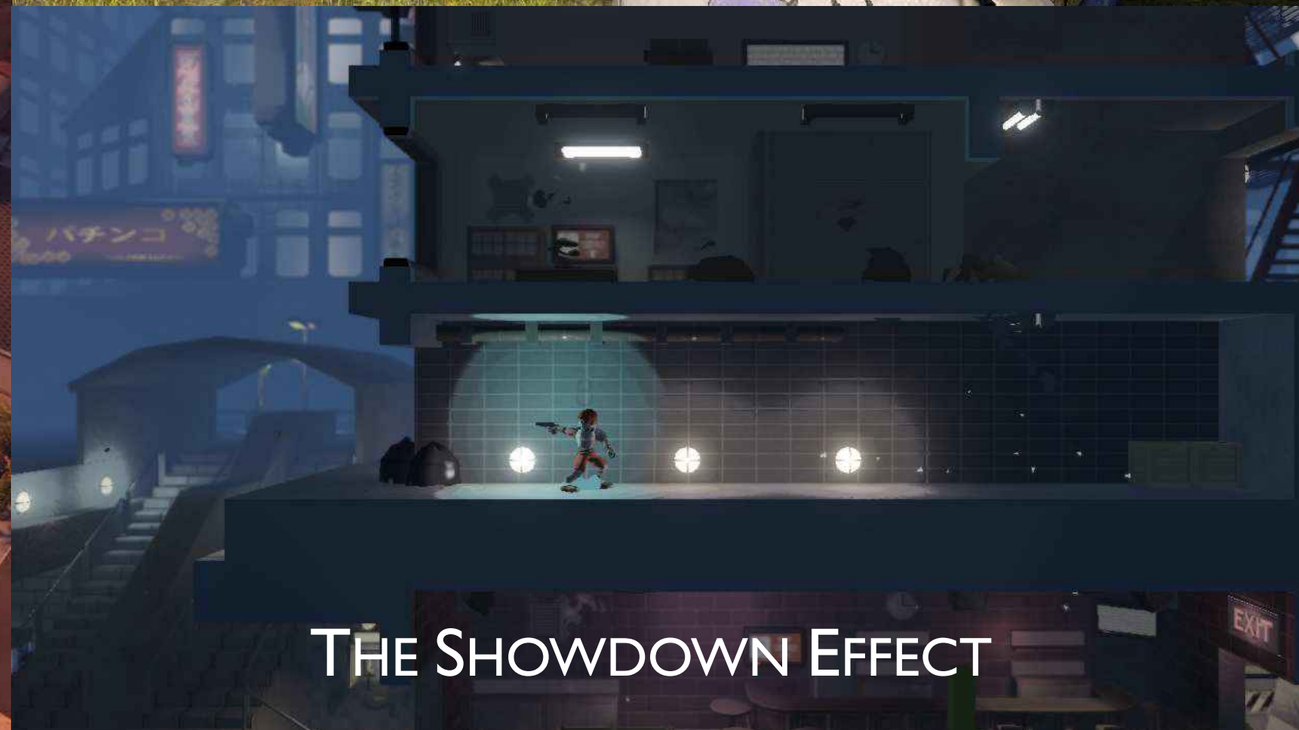
Hamilton's Great Adventure

50+ levels

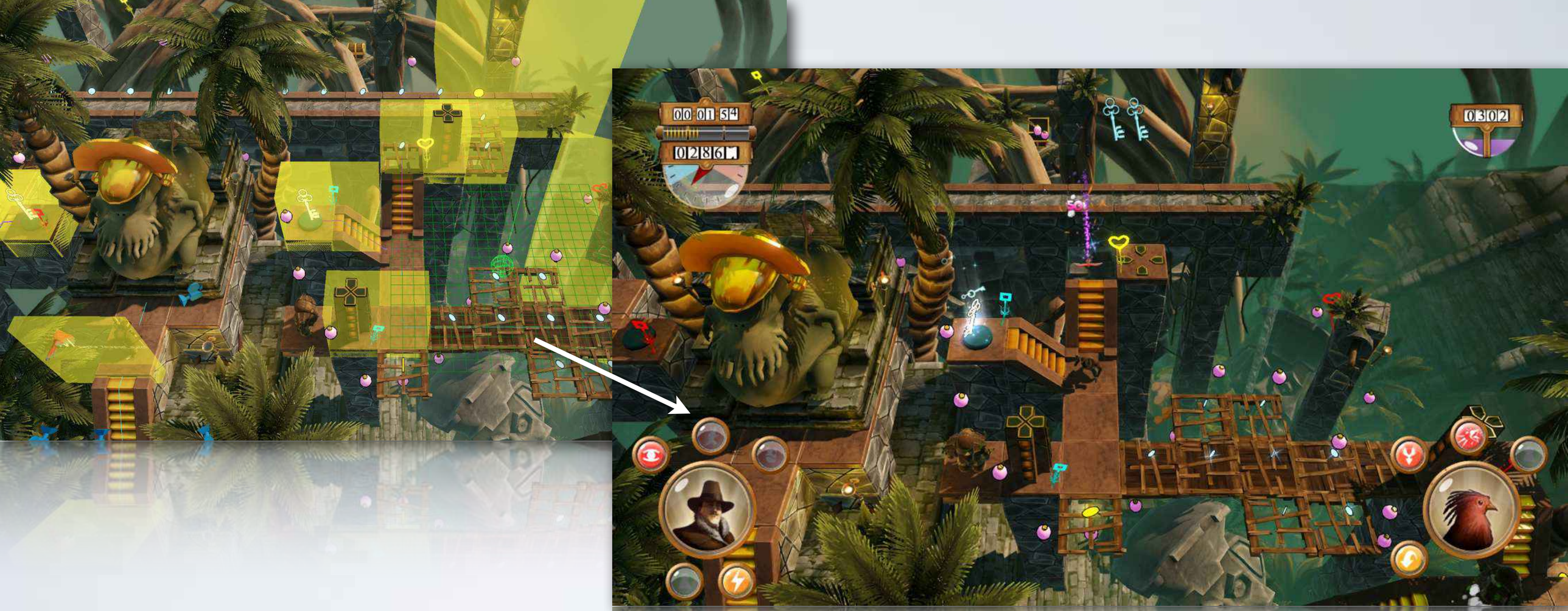
KRATER

WAR OF THE ROSES

RED FRONTIER

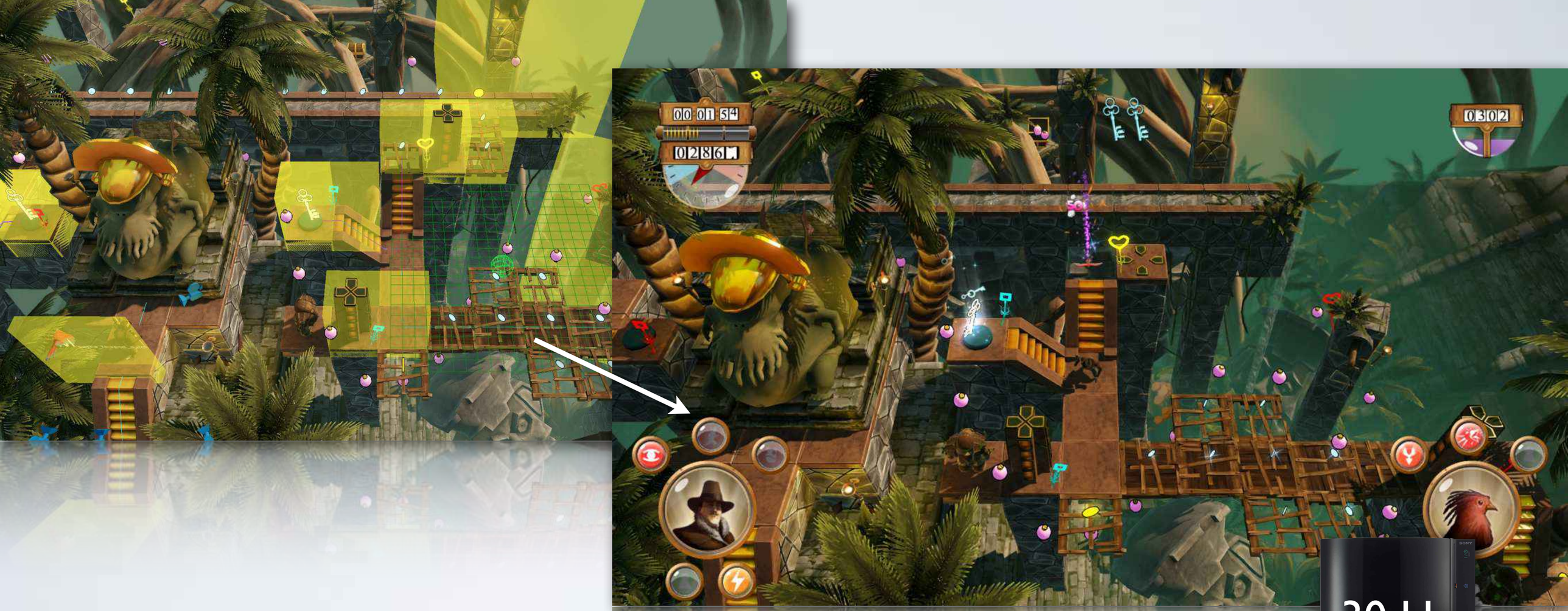


THE SHOWDOWN EFFECT



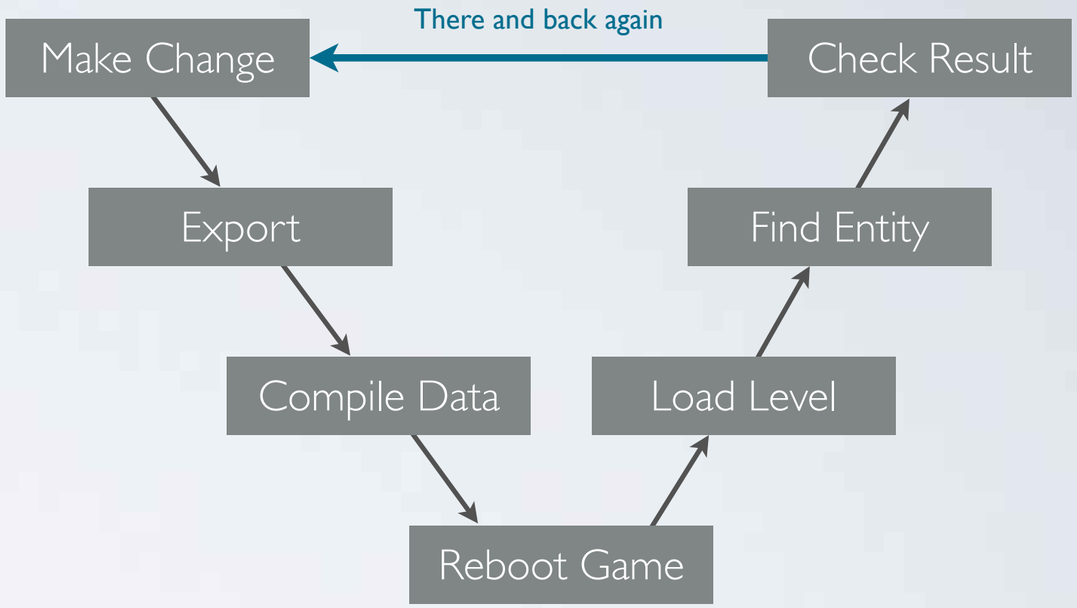
Ambitious Goal

See change ”immediately” (<1 second)



No Cheating!

Target hardware + Target frame rate



What about Live Edit?

Do we even need a pipeline?

Problems with Live Editing

- The game is not always the best editor
- Versioning is tricky if game data is a living binary image
  - Collaborative work and merging changes is also tricky
- Cross-platform? Editing on PS3? X360? iOS?
- Data formats suitable for editing do not have optimal runtime performance

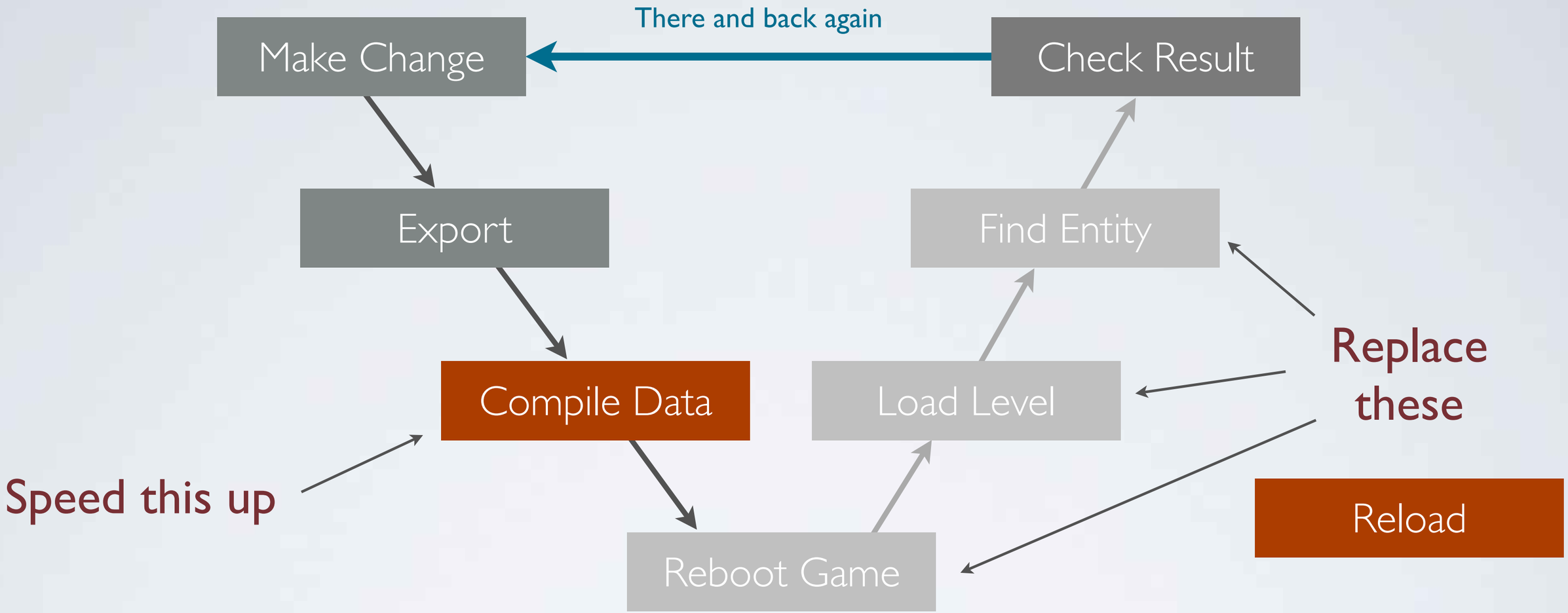
Fast Iterations: The Best of Both Worlds

Fast Games

- Binary resources
- Load-in-place
- No seek times

Fast Workflows

- Short compile time
- Hot reload
- Immediate feedback



Attack Strategy

Compile as fast as possible and replace *reboot* with *reload*

Divide and Conquer

- Recompiling and reloading all data (>1 GB) can never be fast enough
- We must work in smaller chunks

  Regard the game data as a collection of individual resources

  where each resource can be compiled separately

  and then reloaded while the game is running

Individual Resources

- Identified by type + name
- Both are unique string identifiers (gets hashed)

  The name *comes from* a path, but we treat it as an ID (only compare by equality)

type: texture
name: textures/vegetation/grass

source file: textures/vegetation/grass.texture

Compiling Resources

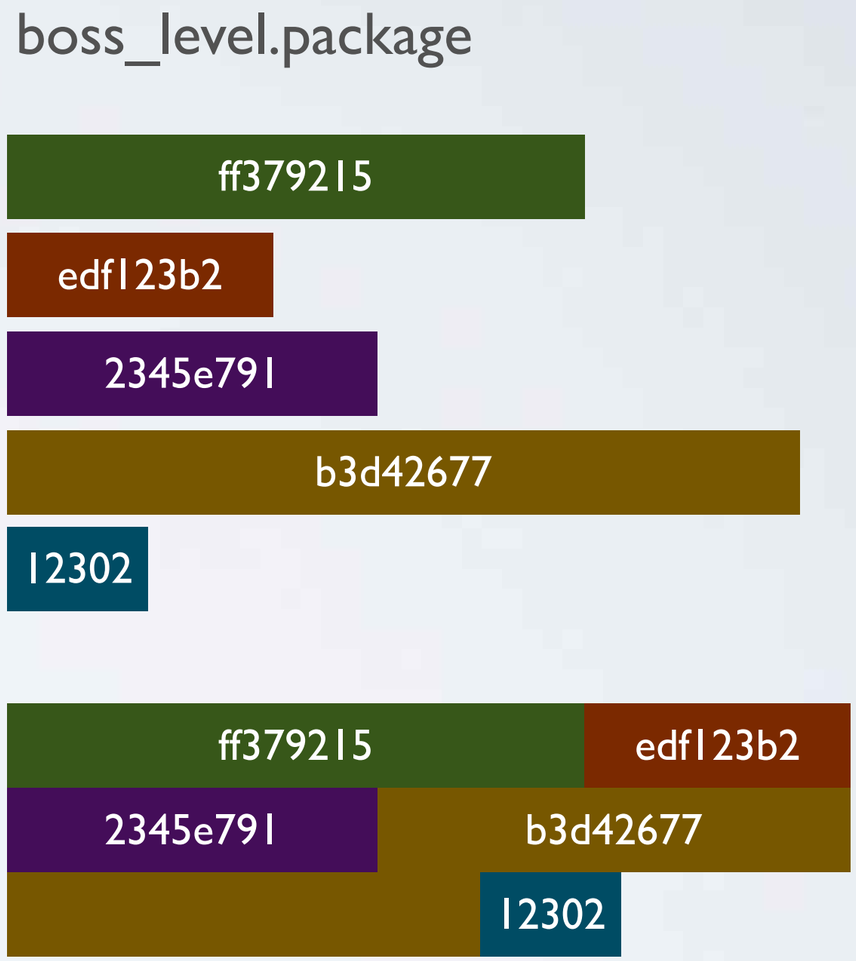
- Each resource compiles to a platform specific runtime optimized binary blob
- Identified by name hash

(data compile)

(in-game resource manager)

Loading Resources

- Resources are grouped into *packages* for loading
- Packages are streamed in by a background thread
- During development, resources are stored in individual files named by the hash
- For final release, the files in a package are bundled together for linear loading



Reloading Resources

- Running game listens on TCP/IP port

  Messages are JSON structs

- Typical commands from our tools

  Enable performance HUD

  Show debug lines

  Lua REPL (read-eval-print-loop)

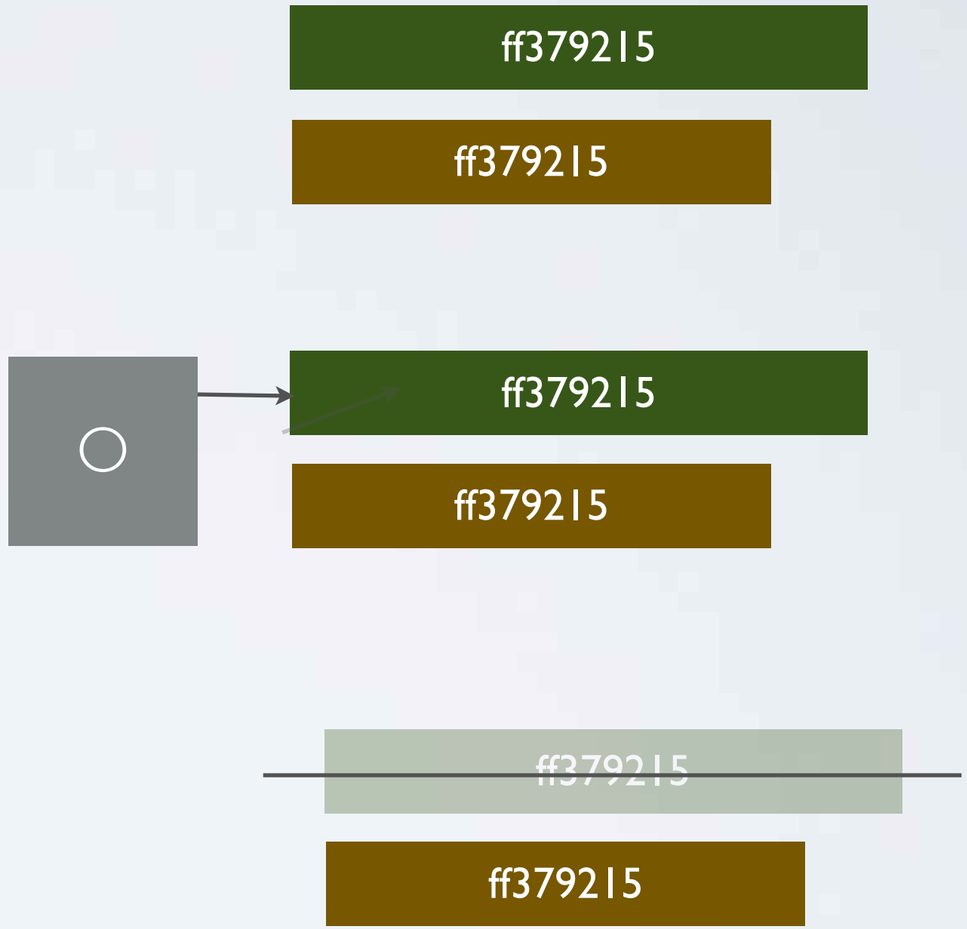
  Reload resource

- Also used for all our tool visualization

> reload texture vegetation/grass

Reloading Resources (Details)

- Load the new resource
- Notify game systems based on type
  - Pointer to old and new resource
- Game system decides what to do
  - Delete instances (sounds)
  - Stop and start instances (particles)
  - Keep instance, update it (textures)
- Destroy/unload the old resource



EXAMPLE: RESOURCE RELOADING

```
if (type == unit_type) {
    for (unsigned j=0; j<app().worlds().size(); ++j)
        app().worlds()[j].reload_units(old_resource, new_resource);
}
```

```
void World::reload_units(UnitResource *old_ur, UnitResource *new_ur)
{
    for (unsigned i=0; i<_units.size(); ++i) {
        if (_units[i]->resource() == old_ur)
            _units[i]->reload(new_ur);
    }
}
```

```
void Unit::reload(const UnitResource *ur)
{
    Matrix4x4 m = _scene_graph.world(0);
    destroy_objects();
    _resource = ur;
    create_objects(m);
}
```

Problematic Issues

- Deploying data to console
- Handling big resources
- Resources that are slow to compile
- Reloading code

Issue: Deploy to Console

- Deploying data to consoles can be slow

  File transfer programs not adapted for sub-second iterations

- Solution: Run a file server on the PC – consoles loads all files from there

  Transparent file system backend

Issue: Big Resources

- Very big resources (>100 MB) can never be compiled & loaded quickly
- Find a suitable resource granularity

  Don't put all level geometry in a single file

  Have geometry for entities in separate files

  Let the level object reference the entities that it uses

Issue: Slow Resources

- Lengthy compiles make fast iterations impossible

  Lightmaps, navmeshes, etc.

- Separate *baking* from *compiling*

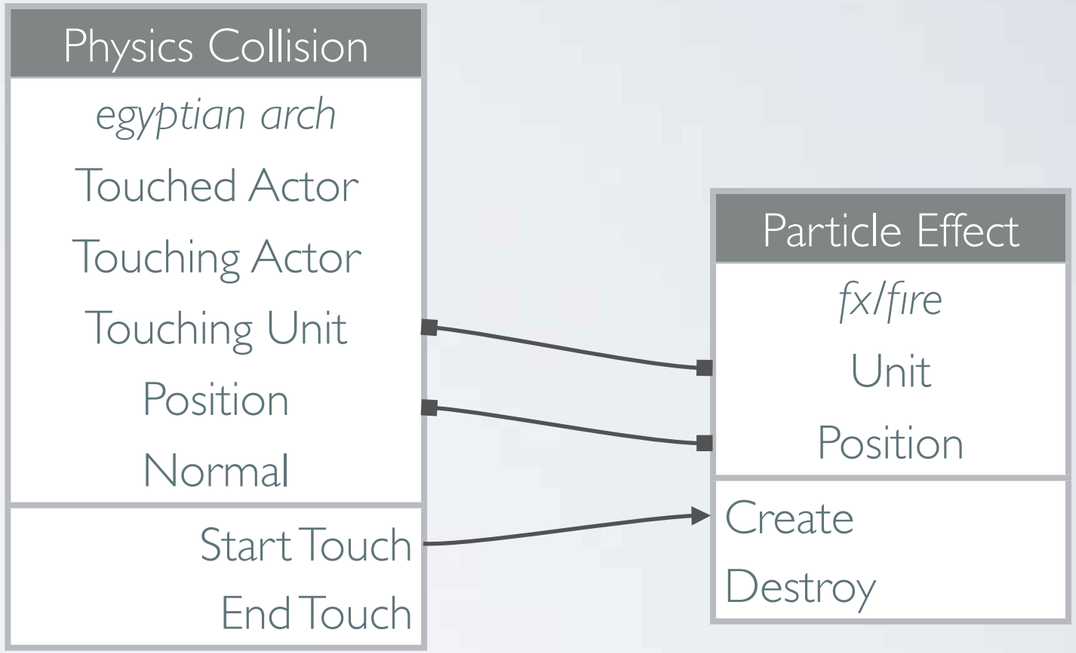
  Baking is always an explicit step: ”make lightmaps now” (editor button)

  The baked data is saved in the *source* data and checked into repository

  Then compiled as usual (from raw texture to platform compressed)

Issue: Reloading Code

- The trickiest resource to reload
- Four kinds of code
  - Shaders (Cg, HLSL)
  - Flow (visual scripting)
  - Lua
  - C++
- Flow & shaders treated as normal resources
  - Just binary data



Flow script

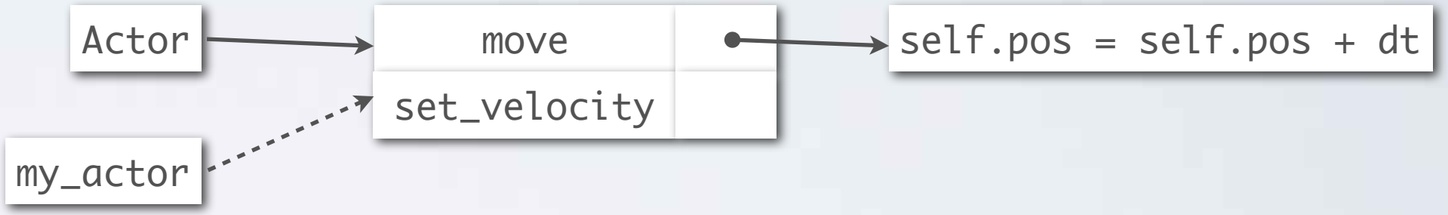
Live Reloading Lua

Makes sure that when reloading, changes are applied to existing Actor class.

Without this, reloading would create a new Actor class and existing Actor objects would not see the code changes.

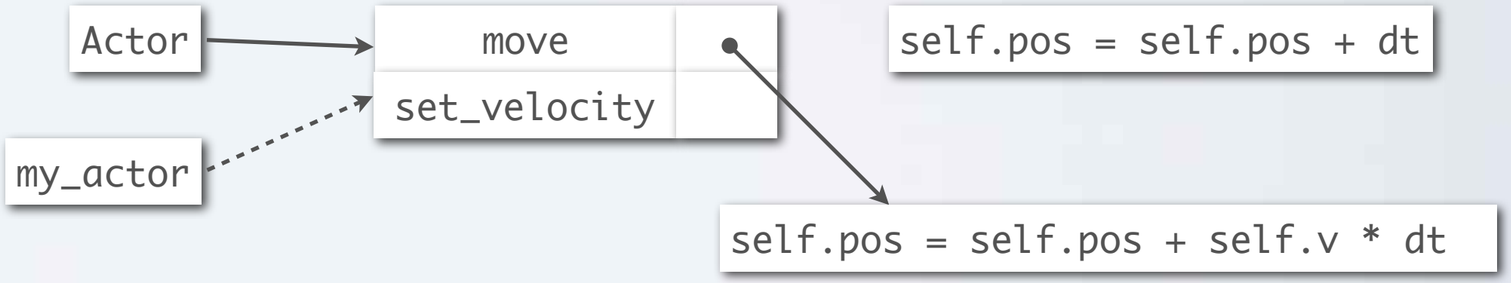
original version

```
Actor = Actor or class()

function Actor:move(dt)
    self.pos = self.pos + dt
end
```



update

```
Actor = Actor or class()

function Actor:move(dt)
    self.pos = self.pos + self.v * dt
end
```



Reloading Code: C++

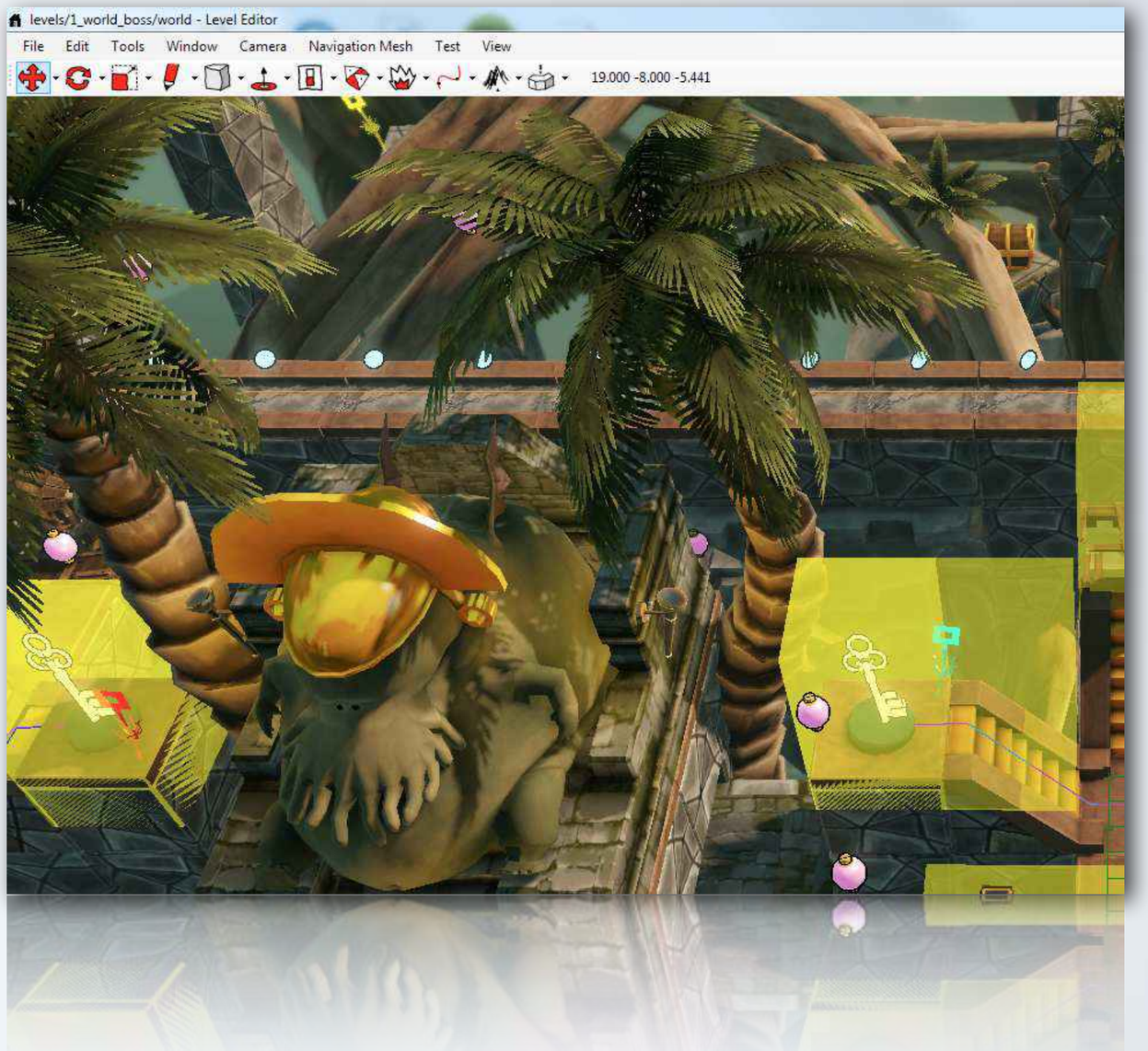
- Tools support ”Restart Exe”

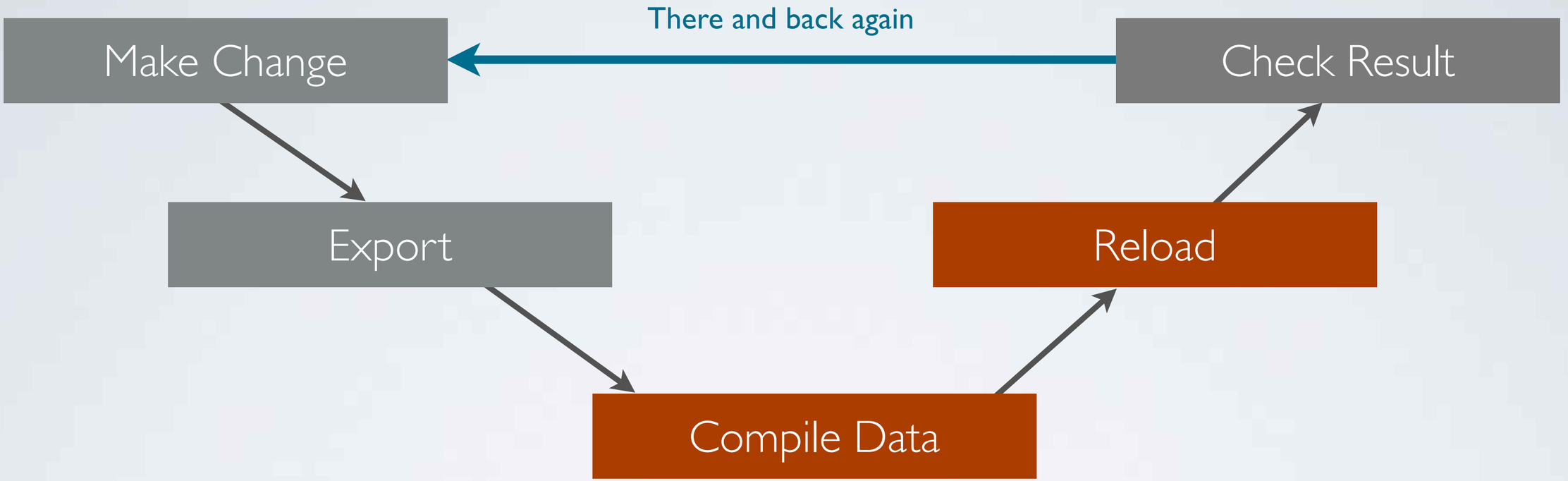
  The exe is reloaded, but you are still at the same location seeing the same objects, just with new engine code

  State is held by tool

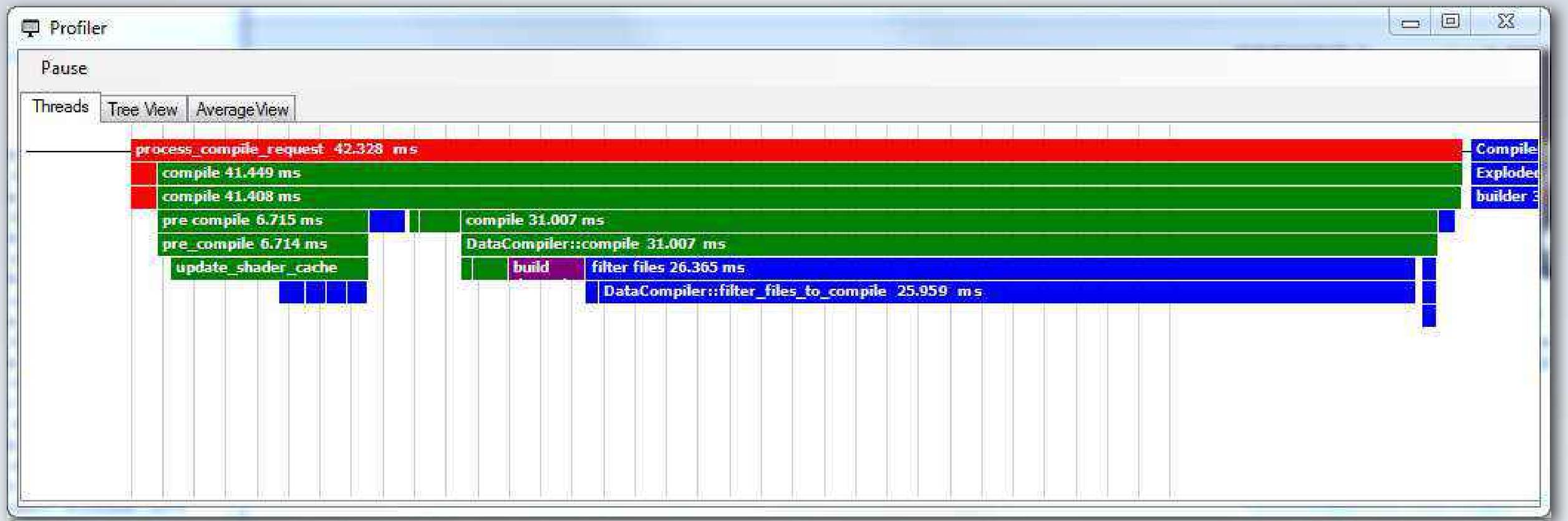
- Does not meet <1s goal, but still very useful

  Small exe size helps





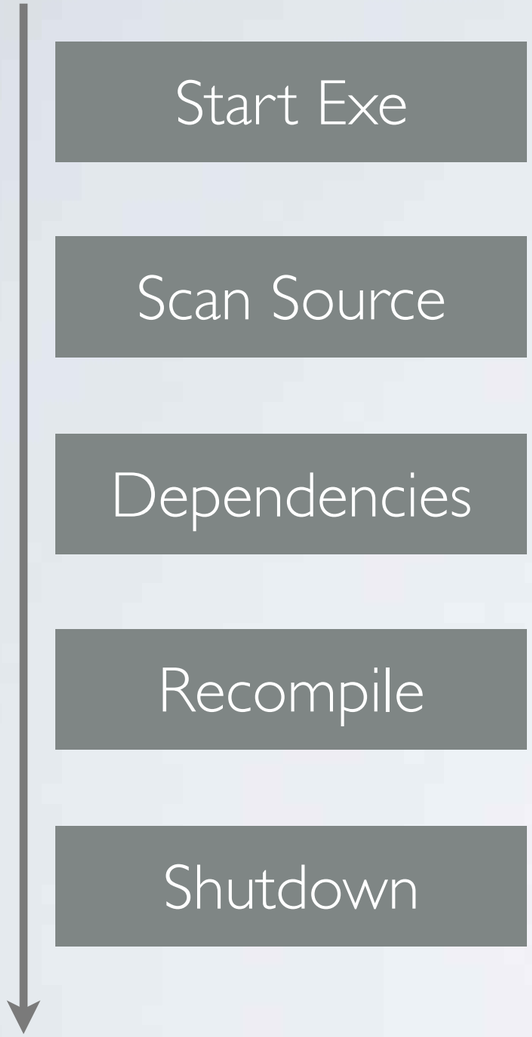
Fast Compiles



Tip: Use the Profiler, Luke

Your tools want some of that performance junkie love too

Incremental Compile



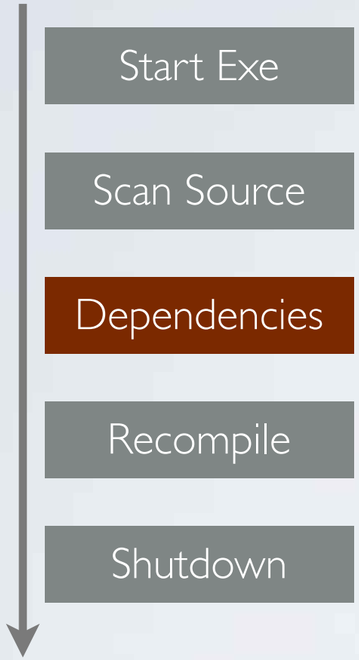
- Find all source data modified since last compile
- Determine the runtime data that depends on those files
- Recompile the necessary parts

- Important that the process is rock solid

  Trust is hard to gain and easy to lose

  ”It is safest to do a full recompile”

Challenge: Dependencies



- *base.shader_source* includes *common.shader_source*

  Needs recompile if *common.shader_source* changes

- How can we know that without reading every file?

- Solution: A compile database

  Stores information from previous runs

  Open at start, save updates at shutdown

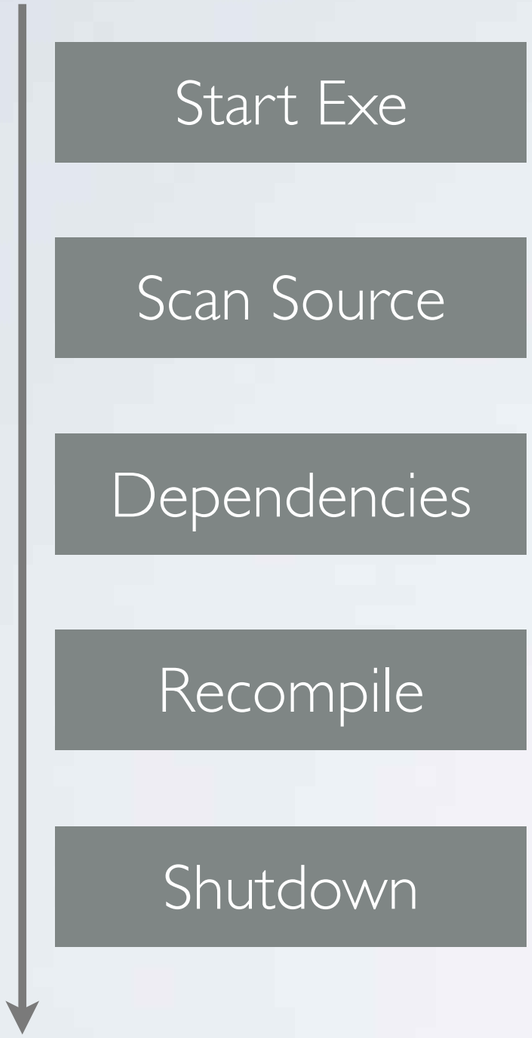
- When a file is compiled, store its dependencies in the database

  Determine them automatically by tracking open_file()

Challenge: Binary Versions

- If the binary format for texture resources changes, every texture needs to be recompiled
- Solution: Reuse the database:
  - Store the binary version of each compiled resource in the database
  - Check against current version in data compiler
  - Recompile if there is a mismatch
- We use the same code base (even the same exe) for the data compiler and the runtime, so binary versions are always in sync

Still Lots of Overhead for Compiling a Single File



Touches disk, ~2 s

Walks entire source tree to check modification times
Touches disk, proportional to project size 5-20 s

Reading and saving database, ~1 s

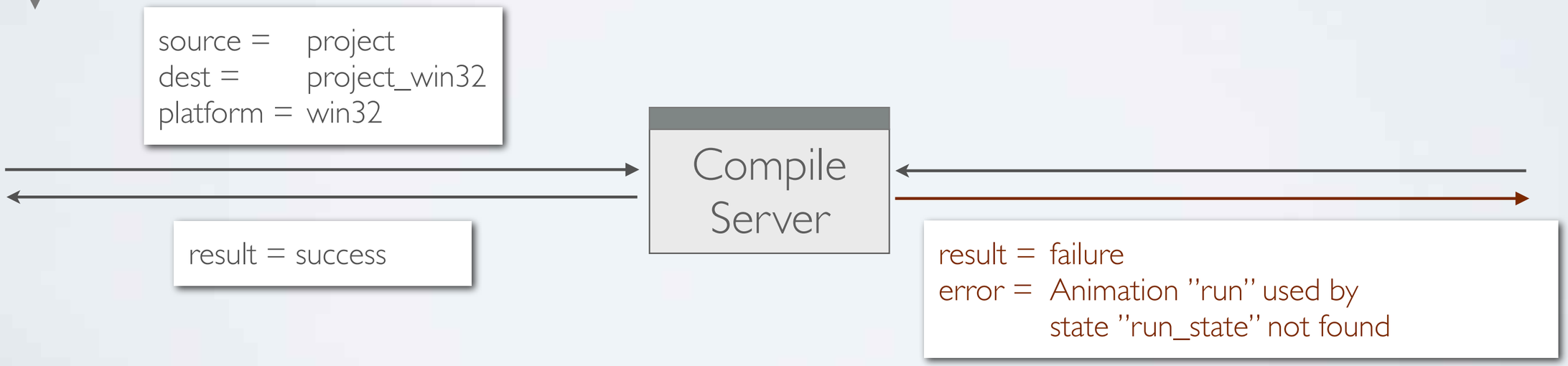
Proportional to the number of modified files
Ok, this is necessary work that needs to be done

STARTUP & SHUTDOWN

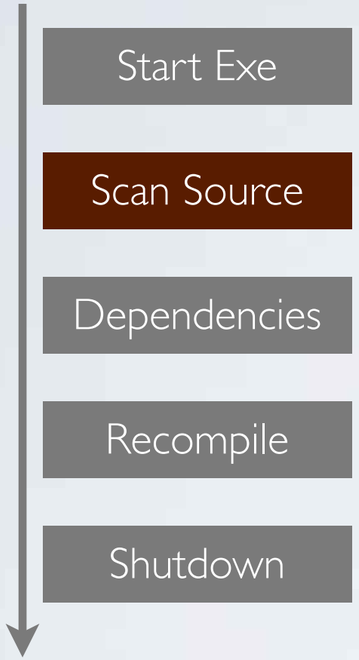
- Several seconds are spent just booting and shutting down the compiler process
- Solution: Reuse the process!

  Run as server

  Receive compile requests over TCP/IP



Scan Source




```
foreach (file in source)
   dest = destination_file(file)
   if mtime(file) > mtime(dest)
      compile(file)
```

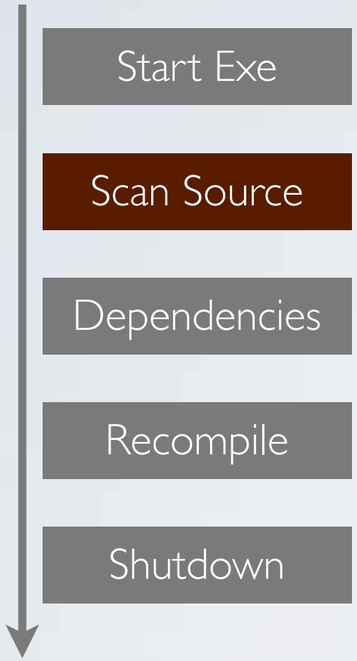
- Slow: Checks *mtime* of every project file
- Fragile: Depends on dates

  If a backup copy is retored we could have *mtime(file)* < *mtime(dest)*

  Crash while writing *dest* is bad

  Trust is important: We *never* want to force a full recompile

Idea: Explicit Compile Lists



- Tool sends a list of the files that it wants recompiled
- Tool keeps track of the files that have changed

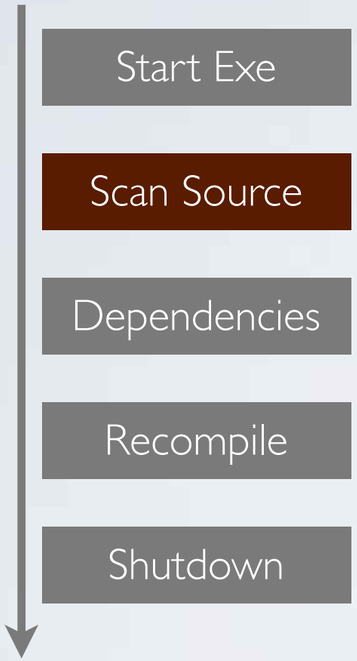
  Texture editor knows all textures the user has changed
- Fast
- Fragile: doesn't work outside tools

  svn/git/hg update

  texture edited in Photoshop

  Lua files edited in text editor

Solution: Directory Watcher



- Do a complete scan when server starts
- After initial scan, use directory watching to detect changes

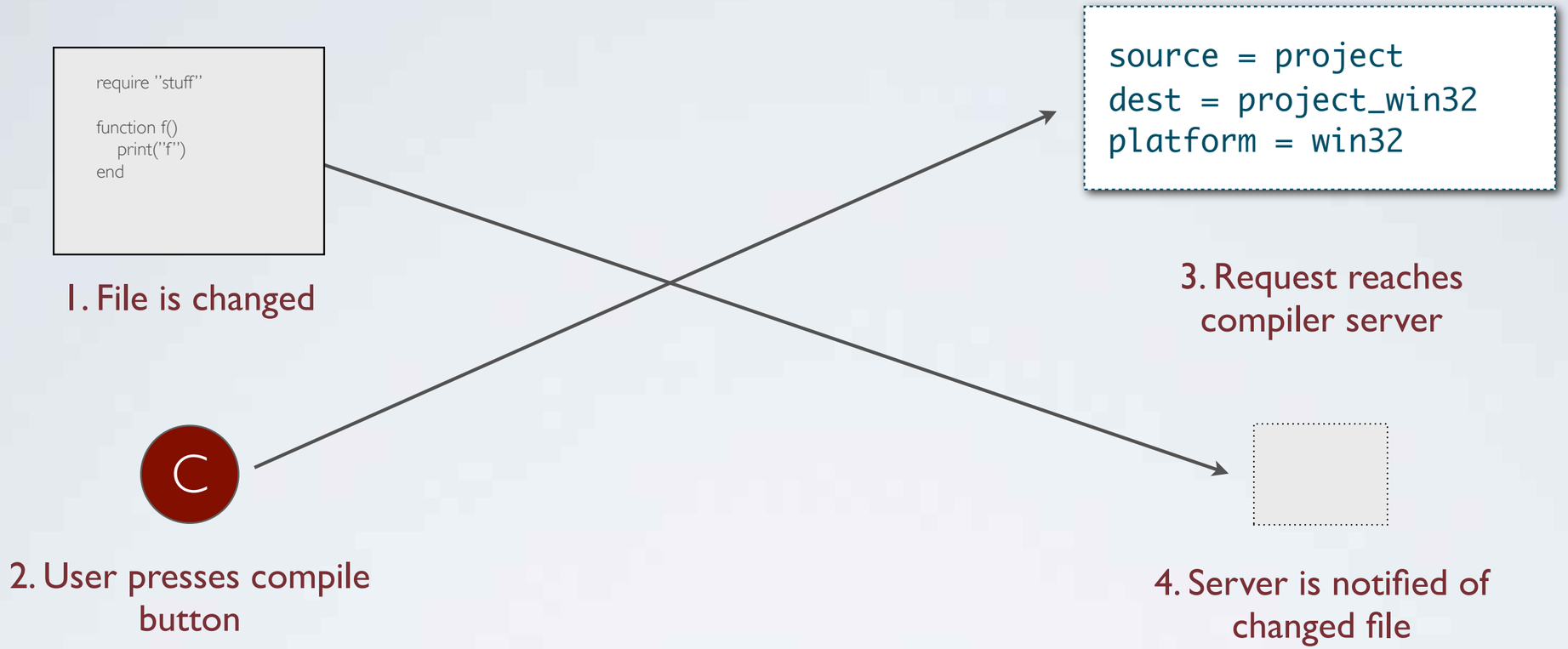
  ReadDirectoryChangesW(...)

- No further scans needed
- Use database to avoid fragility

  Store *mtime* from last successful compile in database

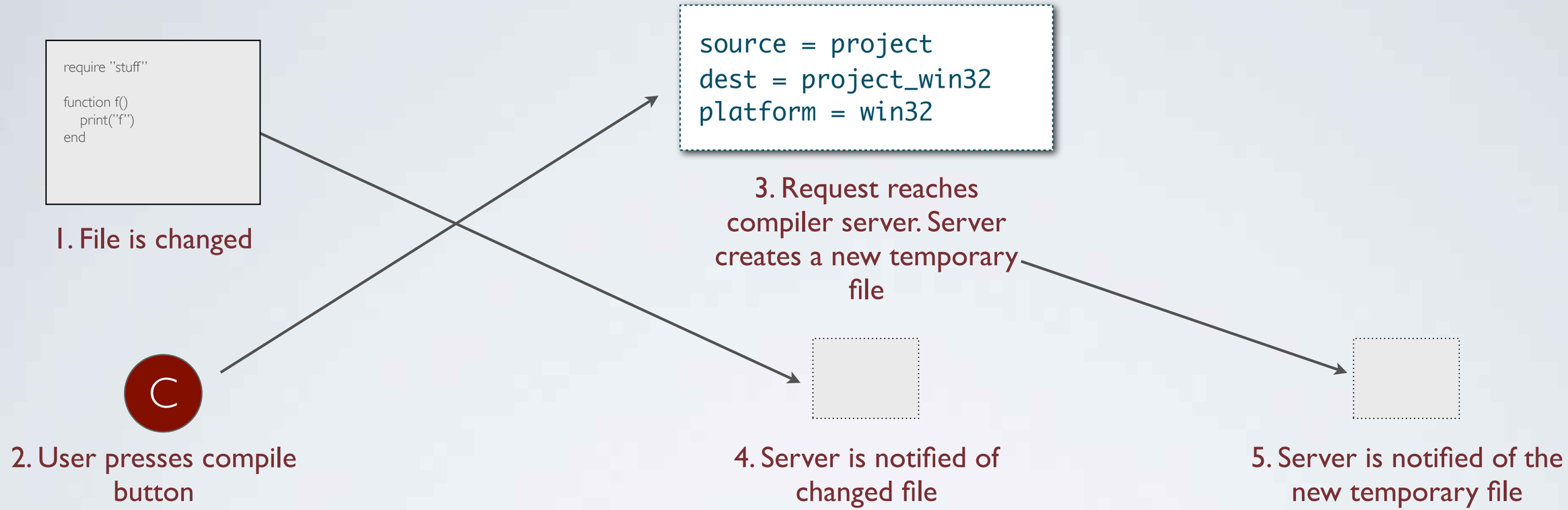
  If *mtime* or file size differs during scan – recompile

  If directory watcher notifies us of a change – recompile



DIRECTORY WATCHER RACE CONDITION

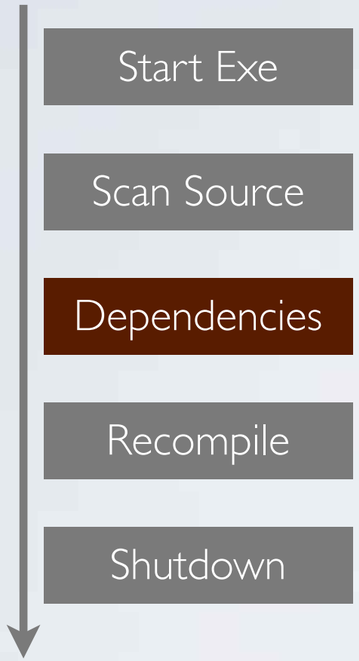
We don't know how long it takes to be notified



Race Condition Trick

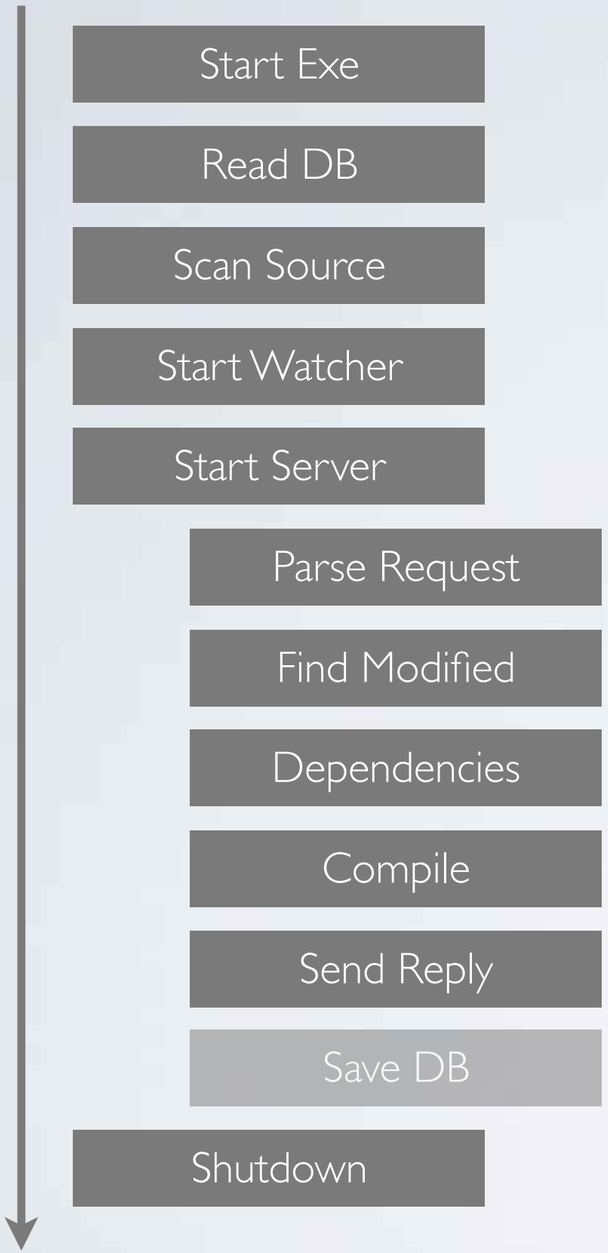
Use temporary file as a ”fence”

Dependencies



- Since we don't destroy the process, we can keep the dependency database in-memory
  - Only needs to be read from disk when server starts
- We can save the database to disk as a background process
  - When we ask for a recompile, we don't have to wait for the database to be saved
  - It is saved later when the compiler is idle

Final Process

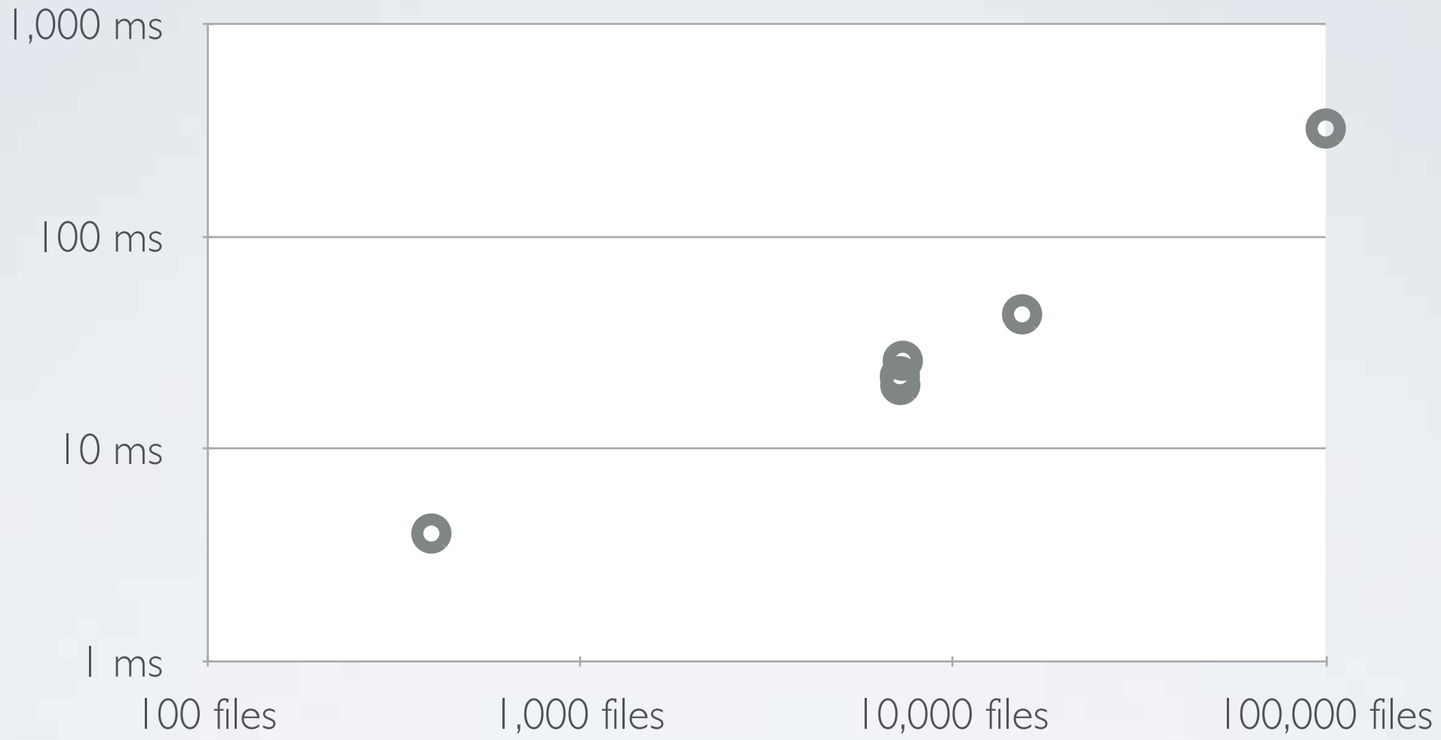


- The only disk access when processing requests is:
  - Compiling the modified files
  - Creating the directory watcher ”fence” file
- Otherwise everything happens in memory

Results

Project	Size	Zero Compile	Min Change
Hamilton	7 200 files	20 ms	25 ms
Undisclosed	15 300 files	43 ms	49 ms
Test	100 000 files	322 ms	366 ms

→ Happy Content Creators


1,000 ms
100 ms
10 ms
1 ms
100 files
1,000 files
10,000 files
100,000 files

RESULTS

General Rules

- Think about resource granularity
  - Reasonably sized for individual compile/reload
- TCP/IP is your friend
  - Prefer to do things over the network to accessing disk
  - Run processes as servers to avoid boot times
- Use database + directory watcher to track file system state
  - Database can also cache other information between compiler runs
  - Keep in-memory, reflect to disk in background

www.bitsquid.se

niklas.frykholm@bitsquid.se

niklasfrykholm

QUESTIONS

bitsquid

GAME DEVELOPERS CONFERENCE 2012
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9