Game Architecture

Continuous Integration

Today’s Agenda

- What is, and Why do, continuous integration?
- Version control
- Build
- Test

“If it hurts, do it more often.”

-Anonymous

The What and the Why

- A team is working on a game.

The What and the Why

- A team is working on a game.
- Everyday each team member changes the game.
  - Alice adds a new AI behavior
  - Bob creates some new sound effects
  - Chris fixes a few bugs

The What and the Why

- A team is working on a game.
- Everyday each team member changes the game.
    - Alice adds a new AI behavior
    - Bob creates some new sound effects
    - Chris fixes a few bugs
- **Integration** is combining those changes into a functional (shippable) game.

The What and the Why

- A team is working on a game.
- Everyday each team member changes the game.
  - Alice adds a new AI behavior
  - Bob creates some new sound effects
  - Chris fixes a few bugs
- **Integration** is combining those changes into a functional (shippable) game.
- **Continuous integration** is doing this constantly.
  - Implies automation
  - Automated integration
  - Automated build
  - Automated test
  - Automated packaging for ship

Why?!?

Integration gets harder, the lower the frequency.

Why?!?

Integration gets harder, the lower the frequency.

- More changes accumulate, so...

Why?!?

Integration gets harder, the lower the frequency.

- More changes accumulate, so...
- More likely something is broken, so...

Why?!?

Integration gets harder, the lower the frequency.

- More changes accumulate, so...
- More likely something is broken, so...
- The reasoning behind changes is less known, so...

Why?!?

Integration gets harder, the lower the frequency.

- More changes accumulate, so...
- More likely something is broken, so...
- The reasoning behind changes is less known, so...
- Fixing breakage takes longer, so…

Why?!?

Integration gets harder, the lower the frequency.

- More changes accumulate, so...
- More likely something is broken, so...
- The reasoning behind changes is less known, so...
- Fixing breakage takes longer, so…
- … more things break

Shorter cycles make everything easier and faster.

Why!?! Part 2

At ‘AAA’ scale:

- 100s of developers, likely all over the world
- 1000s of changes per day
- Several targets
- Large game

Implications:

- Understanding ramifications of any change in reasonable time not tractable
- Breaking the integrated game is expensive ($10k+ per hour*)

As scale increases, the stronger the case for continuous integration becomes.

Version Control

Because we need a shared place to put our work.

Not Version Control

```
C:\GA-FinalProject
C:\GA-FinalProject-BACKUP
C:\GA-FinalProject-JUNK

entity.cpp.old
entity.cpp.bak
entity-latest-works.cpp
```

Version Control

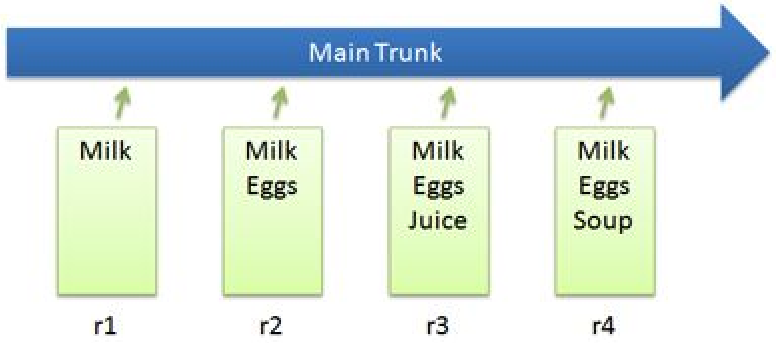
Database of all versions of all* files used to develop your game.

- **Synchronization.** Get the latest, shared, integrated version.
- **Undo.** Throw away some local bad changes and go back pristine version.
- **Backup and restore.** Go back to the way things were in the past.
- **Track changes.** Tie messages explaining changes with changes.
- **Track ownership.** See who and when changes were made.
- **Branch and merge.** Make a big change off to the side. Merge when ready.

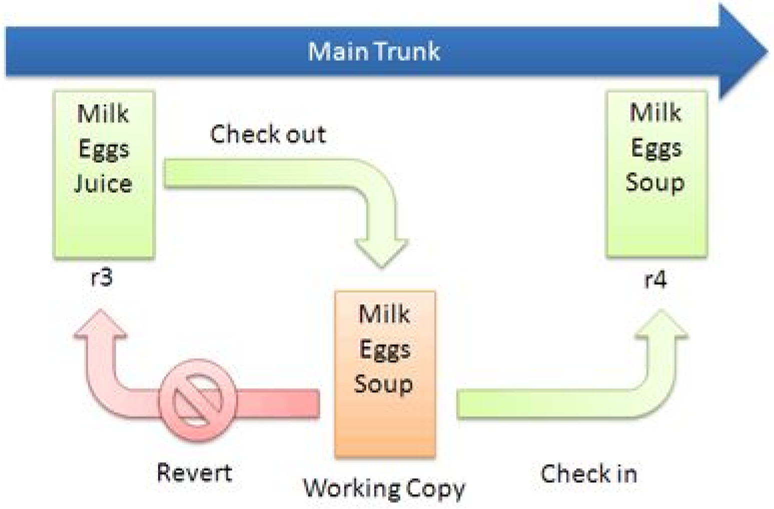
Version Control Terms

- **Depot, Repository** - The database.
- **Check-in, Commit, Submit** - Upload local changes to database.
- **Check-out, Sync, Update** - Download changes from database.
- **Revision** - Version of the a file.
- **Head** - Latest version of a file.
- **Revert** - Undo local changes and re-download from database.
- **Diff** - Differences between two files.
- **Conflict** - When pending changes break each other.
- **Resolve** - Process of eliminating conflicts.
- **Main, Trunk** - The primary location of files in a project.
- **Branch, Stream** - Separate copy of files in a project for specific use.

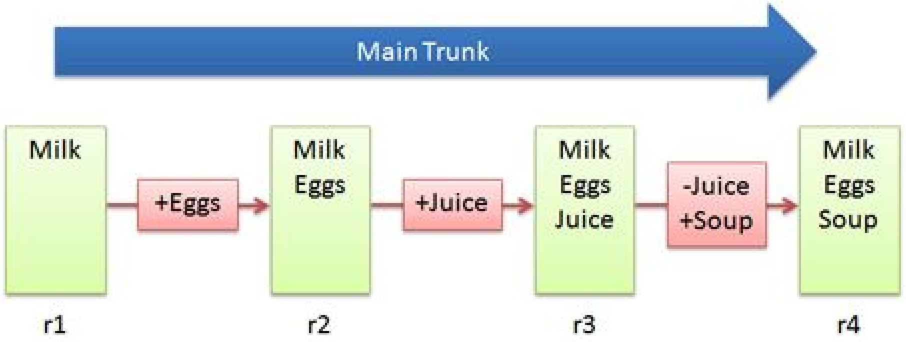
Basic Checkins



Checkout and Edit

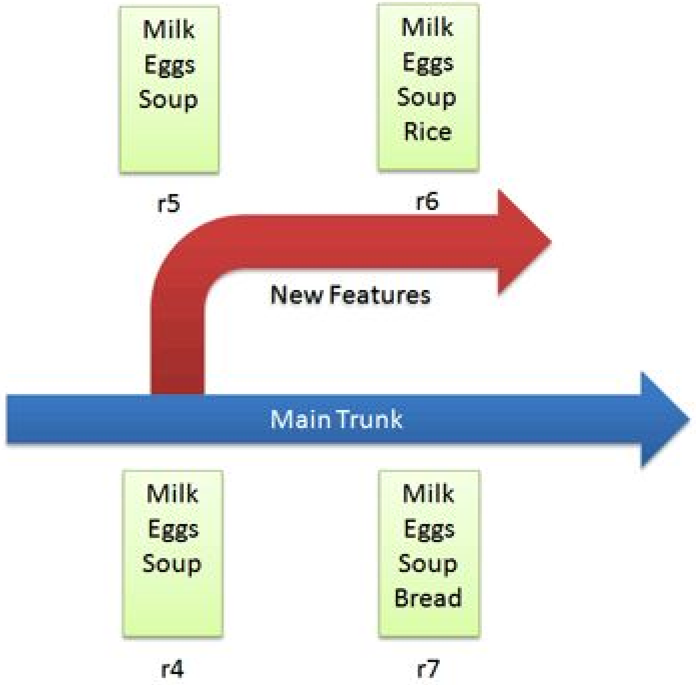


Basic Diffs

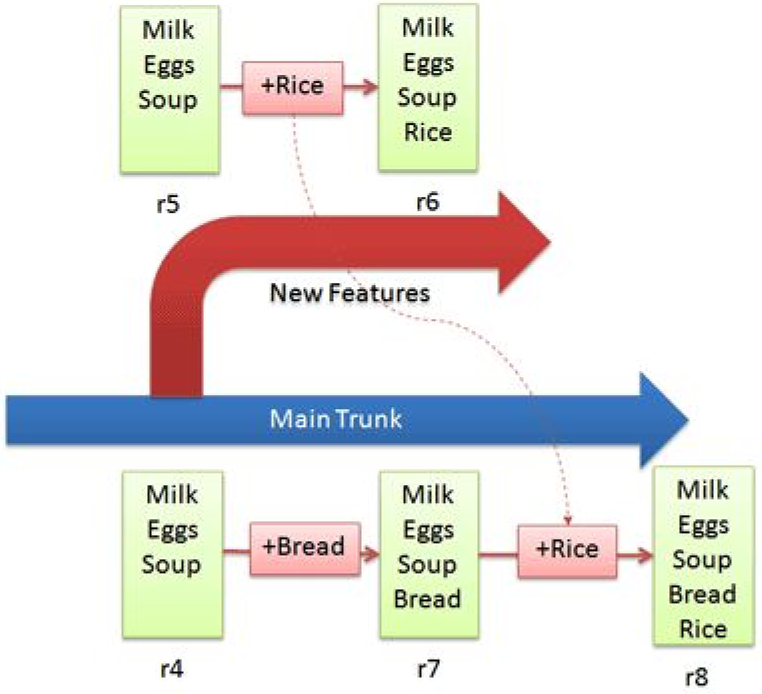


Main Trunk
Milk
+Eggs
Milk
Eggs
+Juice
Milk
Eggs
Juice
-Juice
+Soup
Milk
Eggs
Soup
r1
r2
r3
r4

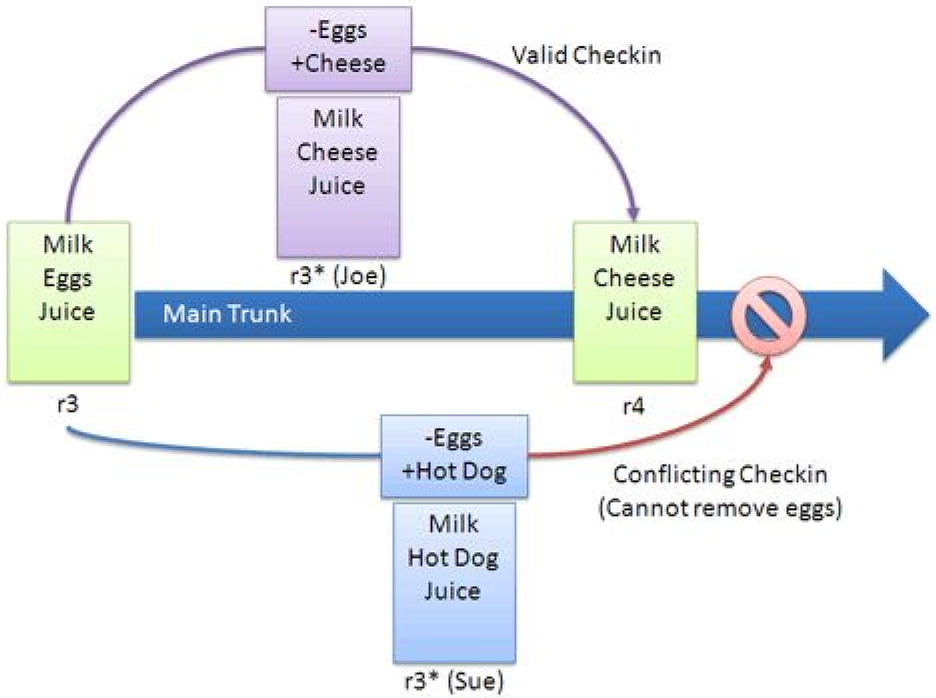
Branching


Milk
Eggs
Soup
r5
Milk
Eggs
Soup
Rice
r6
New Features
Main Trunk
Milk
Eggs
Soup
r4
Milk
Eggs
Soup
Bread
r7

Merging


Milk
Eggs
Soup
+Rice
Milk
Eggs
Soup
Rice
r5
r6
New Features
Main Trunk
Milk
Eggs
Soup
+Bread
Milk
Eggs
Soup
Bread
+Rice
Milk
Eggs
Soup
Bread
Rice
r4
r7
r8

Conflicts


-Eggs
+Cheese
Valid Checkin
Milk
Cheese
Juice
r3* (Joe)
Milk
Eggs
Juice
Main Trunk
Milk
Cheese
Juice
r3
r4
-Eggs
+Hot Dog
Conflicting Checkin
(Cannot remove eggs)
Milk
Hot Dog
Juice
r3* (Sue)

Centralized vs Distributed Version Control

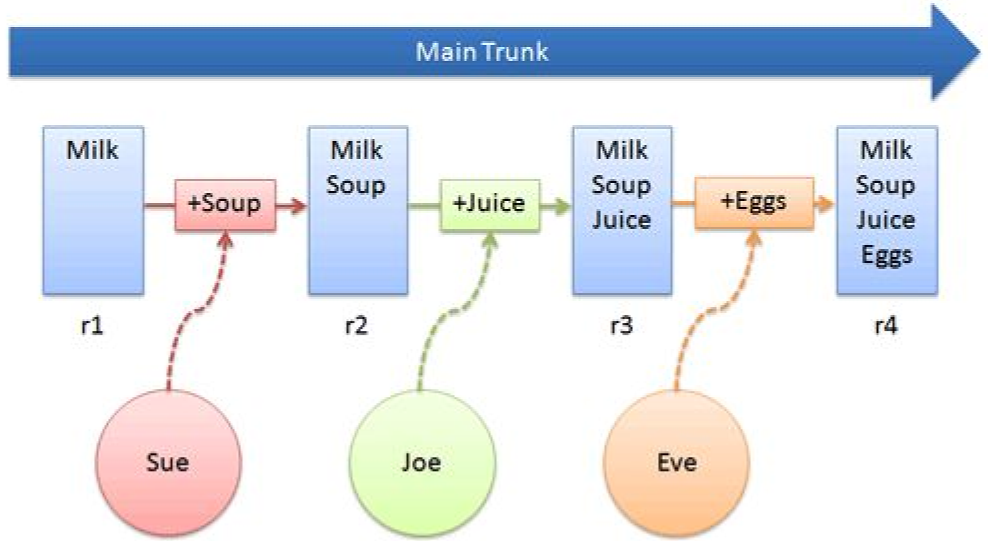
Centralized:

- Single authoritative database everyone uses
- CVS, Perforce, Subversion

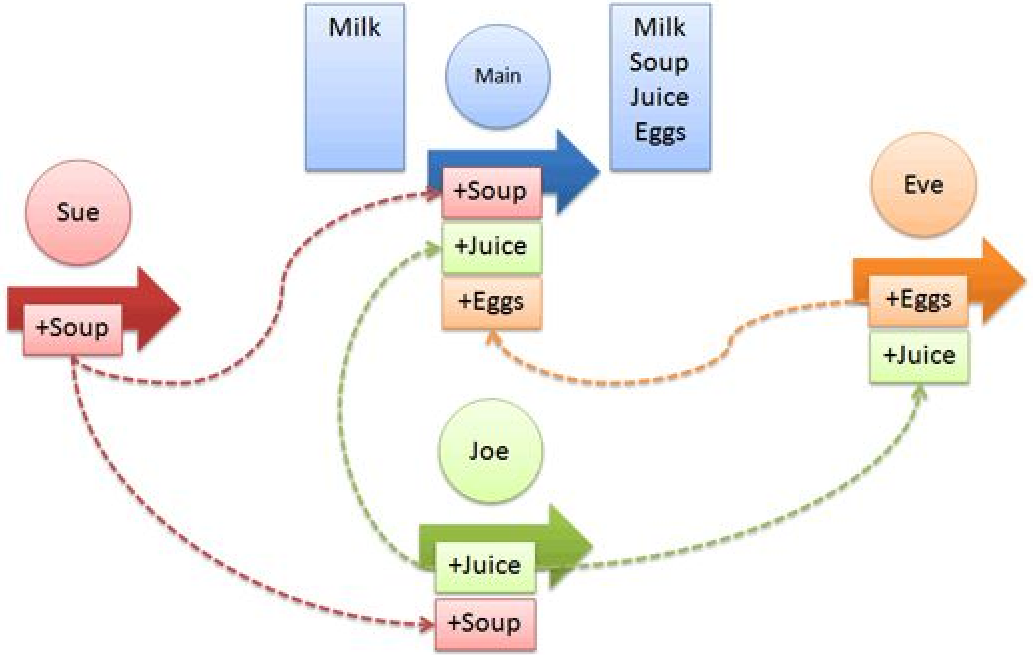
Distributed:

- Every user has their own database
- Users can share changes between databases (push and pull)
- Git, Mercurial

Centralized VCS



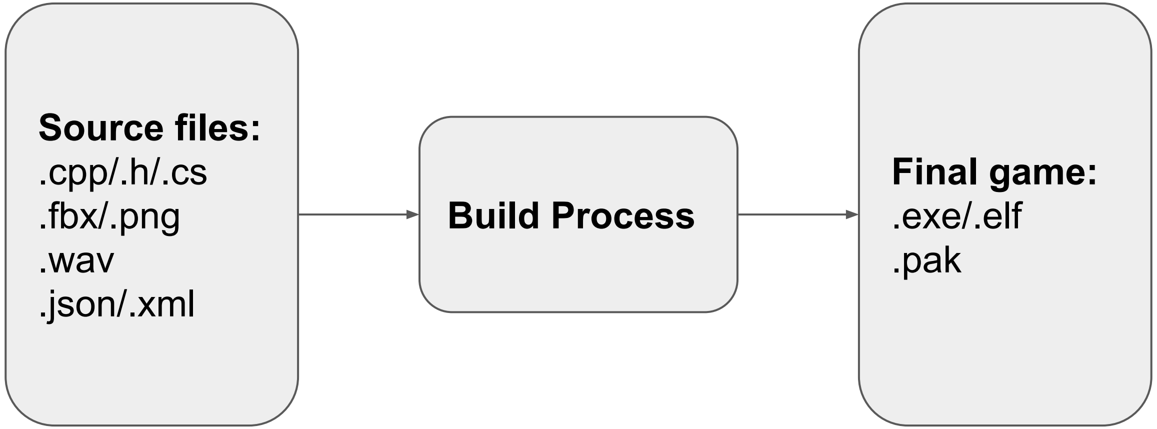
Distributed VCS



Build

Harder than you might think.

Build: The Basic Idea



Properties of a good build system

- Deterministic and correct
- Only builds what is needed
- Uses all available CPU power
- Scales well
- Can be automated

Properties of a good build system

- **Deterministic and correct**
- Only builds what is needed
- Uses all available CPU power
- Scales well
- Can be automated

Not this:

- “If you ever change file X, then you have to manually clean build.”
- “If you get that crazy error, try building again. If that doesn’t work, try a clean build.”

This:

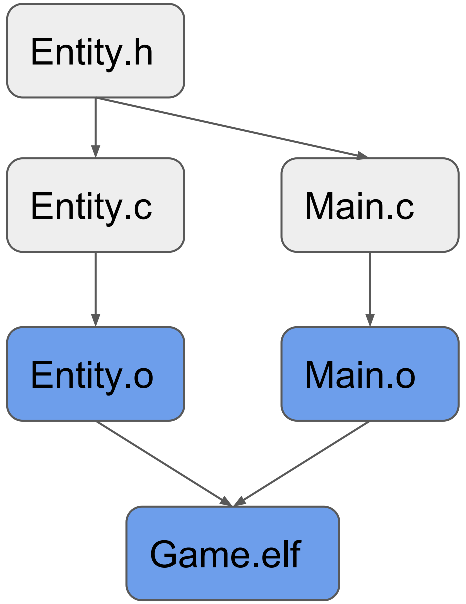
- Build understands full dependency tree.
- Building from same source results in same output, on any machine.

Properties of a good build system

- Deterministic and correct
- **Only builds what is needed**
- Uses all available CPU power
- Scales well
- Can be automated

Build knows full dependency tree and uses it:

- Change Entity.h - build all
- Change Main.c - build main.o & game.elf

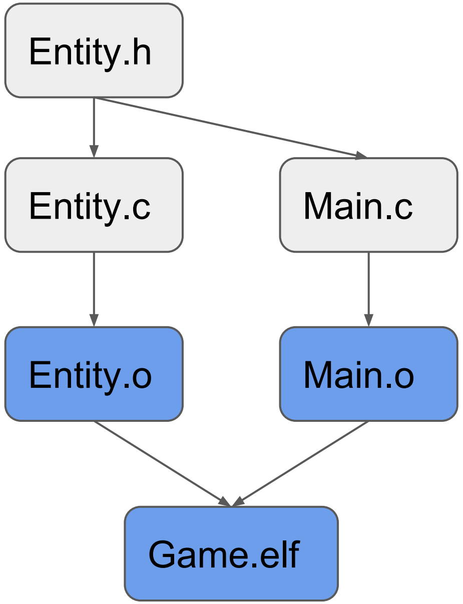


Properties of a good build system

- Deterministic and correct
- Only builds what is needed
- **Uses all available CPU power**
- Scales well
- Can be automated

Build knows full dependency tree and uses it:

- Build Entity.o and Main.o in parallel
- Build Game.elf once .o files are done



Properties of a good build system

- Deterministic and correct
- Only builds what is needed
- Uses all available CPU power
- **Scales well**
- Can be automated

Engine code:

- 1,000s of C++ files
- Stable output formats
- High quality compilers

Content (models, textures, sounds, etc):

- 100,000s of files in assorted formats
- Unstable output formats
    - Code changes can necessitate large rebuilds of content
- Content bakers/cookers/optimizers of varying quality levels

Build system must scale to content.

Properties of a good build system

- Deterministic and correct
- Only builds what is needed
- Uses all available CPU power
- Scales well
- **Can be automated**

- Command line invoke
- Console/tty output

We'll use this shortly.

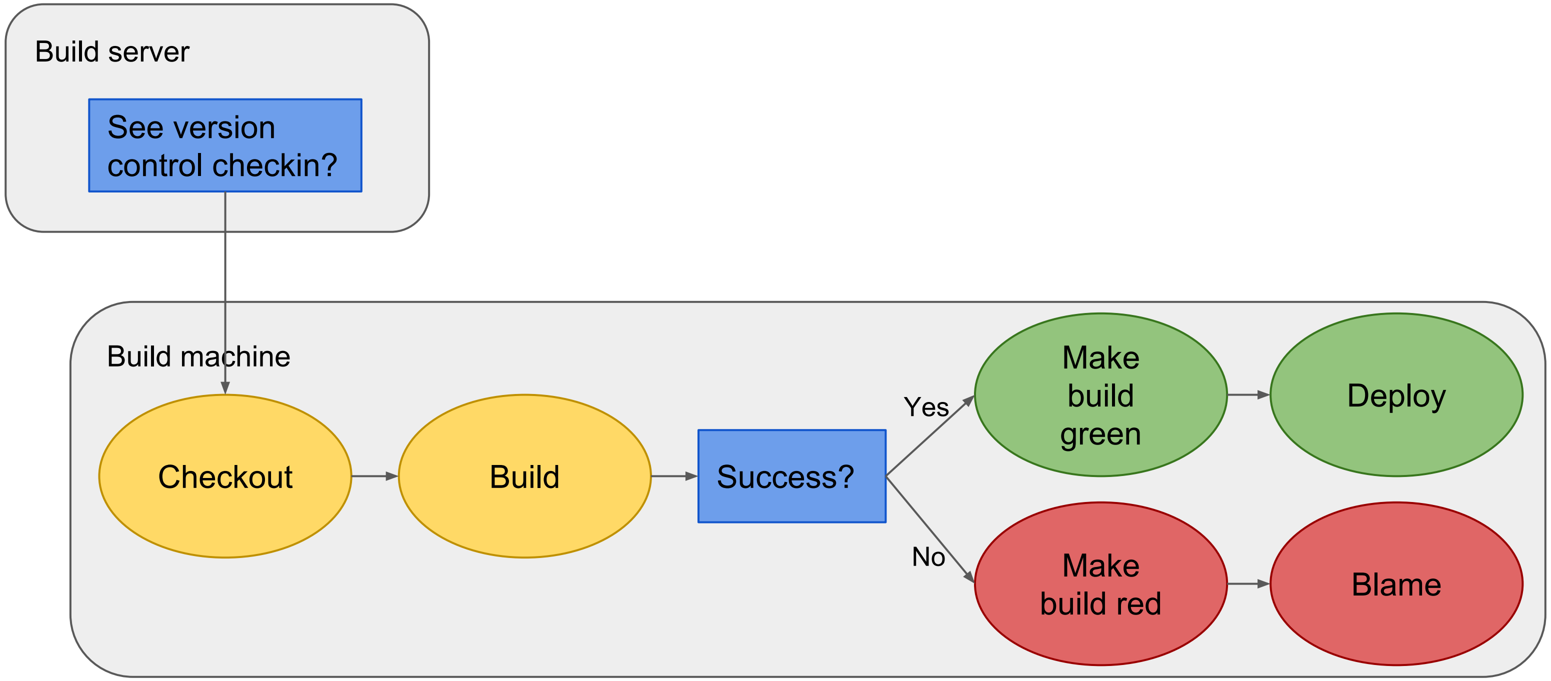
Structure of a typical build system

- Use **checksums** or **timestamps** to determine if a file is dirty (needs building)
- **Scanners** determine dependency tree
  - E.g. *Entity.c* depends on *Entity.h* because scanner reads #include directive
- Dirty-ness flows down the dependency tree
  - E.g. *Entity.o* is dirty because it depends on *Entity.c* and *Entity.c* is dirty
- Files **pattern match** to **rules** and trigger **actions** to generate **targets**
  - CompileRule(*.c) -> CompileAction -> CompileTarget(*.o)
  - LinkRule(*.o) -> LinkAction -> LinkTarget(game.elf)

Build Options

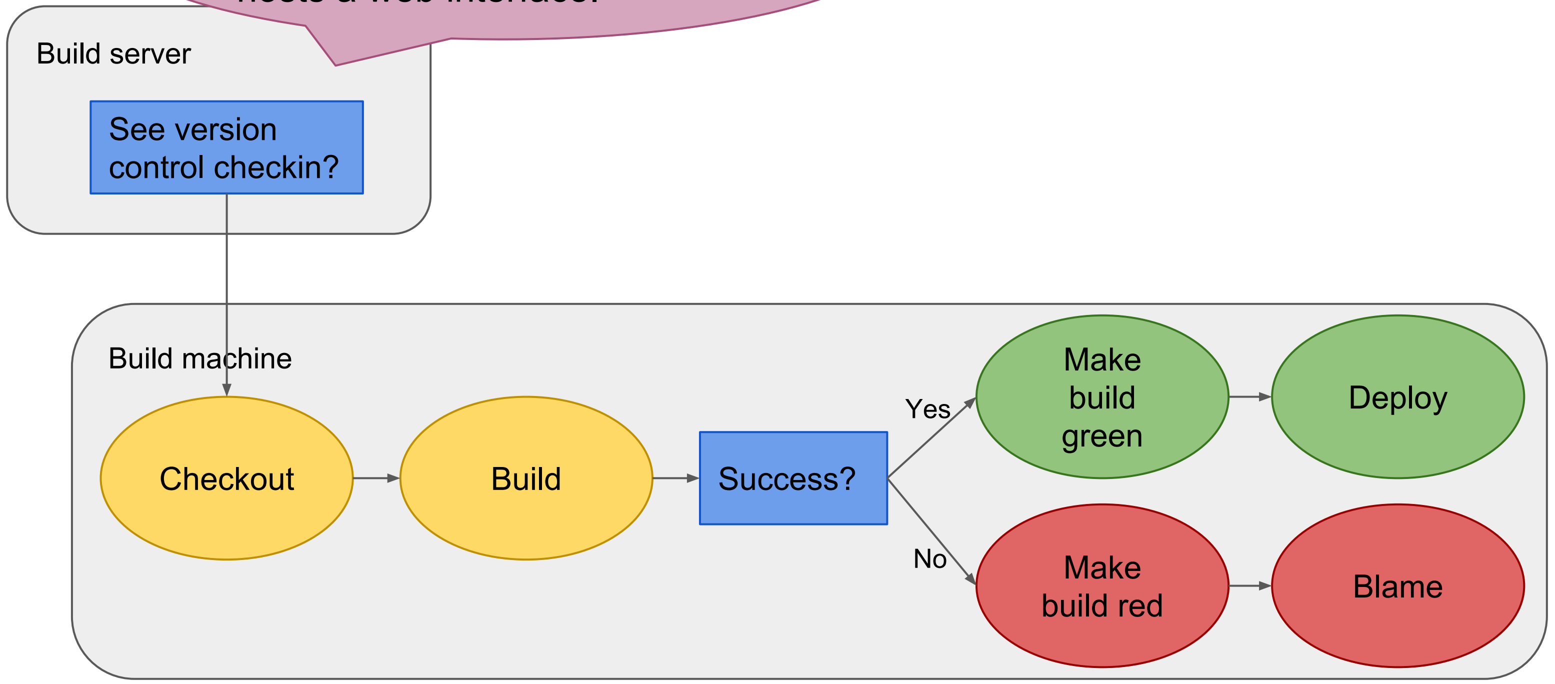
- Use something off-the-shelf
  - Make
  - Nmake (Visual Studio)
  - Jam
  - Etc.
- Roll your own
  - Beyond the scope of this lecture

Automating the build

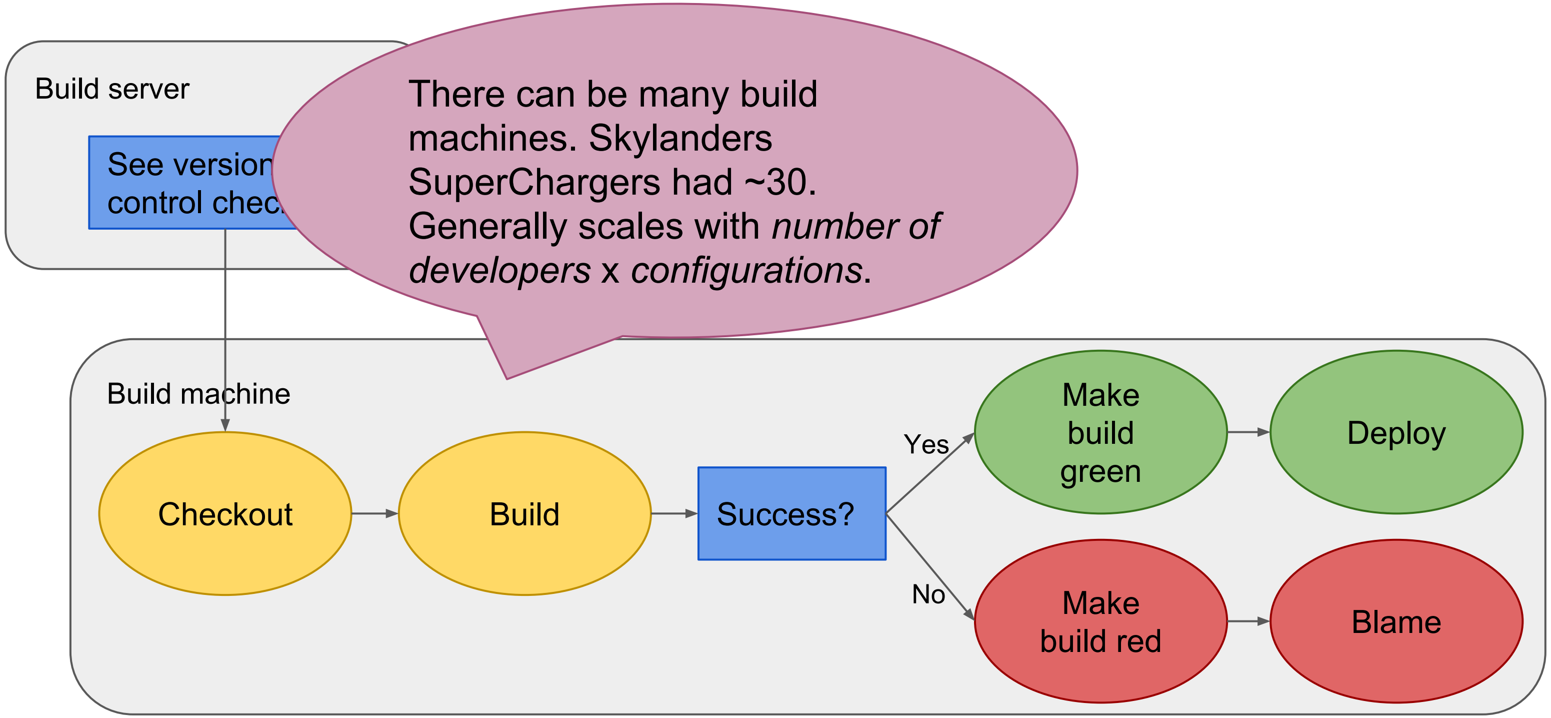


Autom

Build server manages an array of build machines and usually hosts a web interface.

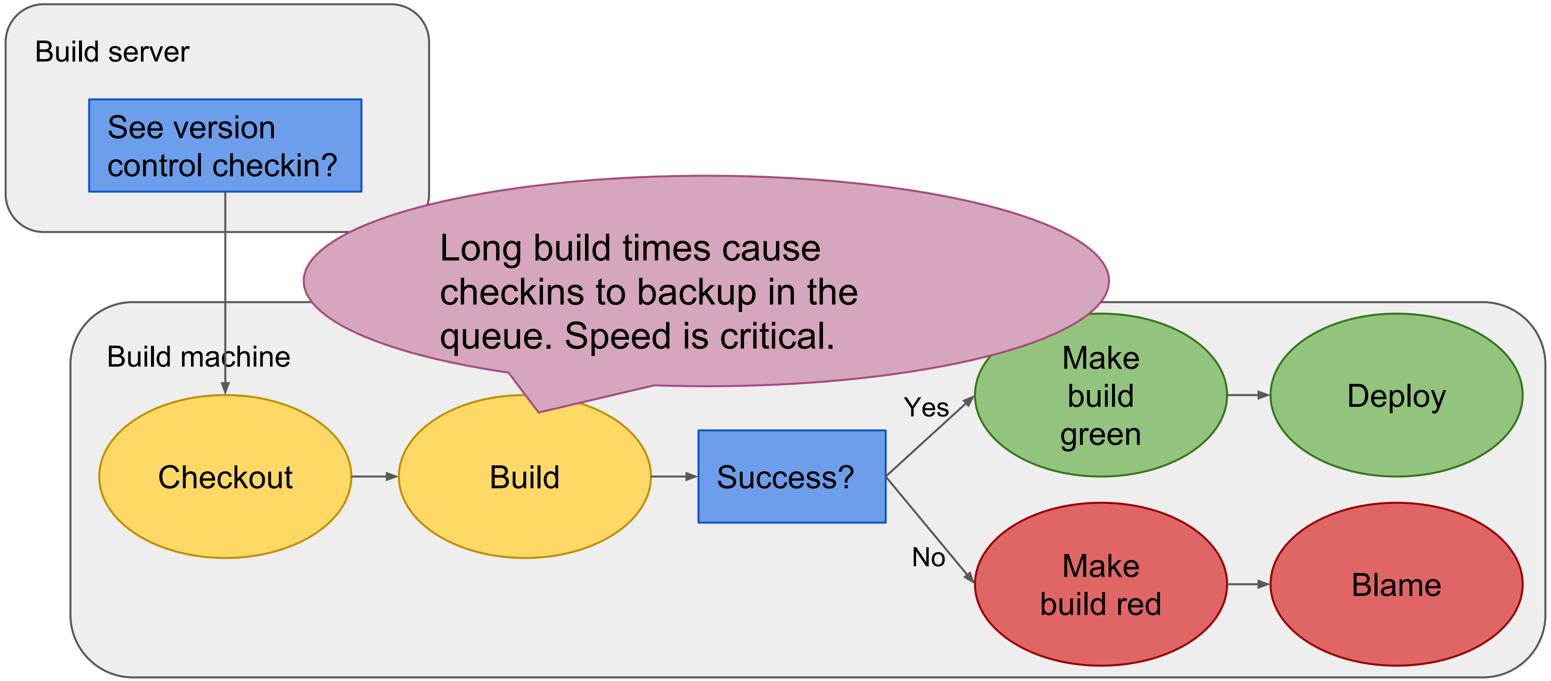


Automating the build

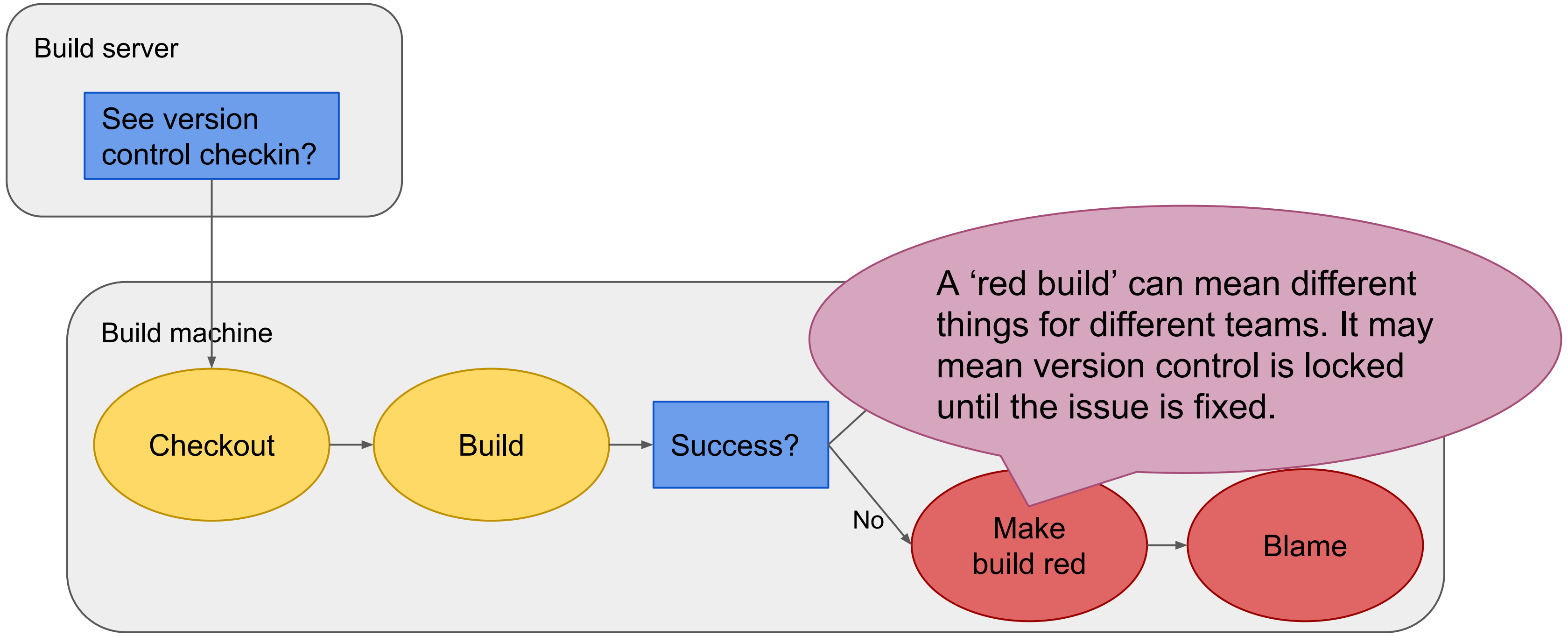


There can be many build machines. Skylanders SuperChargers had ~30. Generally scales with *number of developers* x *configurations*.

Automating the build

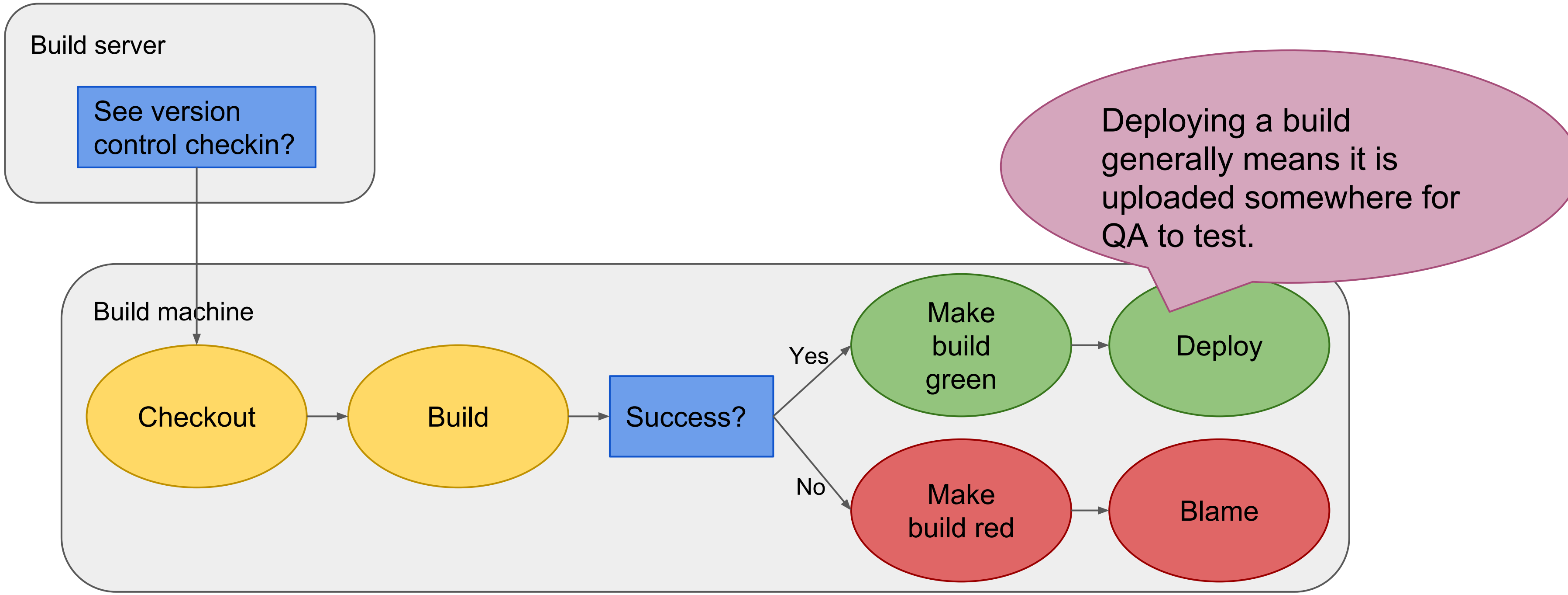

Build server
See version control checkin?
Build machine
Checkout
Build
Success?
Yes
No
Make build green
Deploy
Make build red
Blame
Long build times cause checkins to backup in the queue. Speed is critical.

Automating the build



A ‘red build’ can mean different things for different teams. It may mean version control is locked until the issue is fixed.

Automating the build


Build server
See version control checkin?
Build machine
Checkout
Build
Success?
Yes
No
Make build green
Deploy
Make build red
Blame
Deploying a build generally means it is uploaded somewhere for QA to test.

Build automation options

Off-the-shelf:

- Jenkins
- CruiseControl
- Etc.

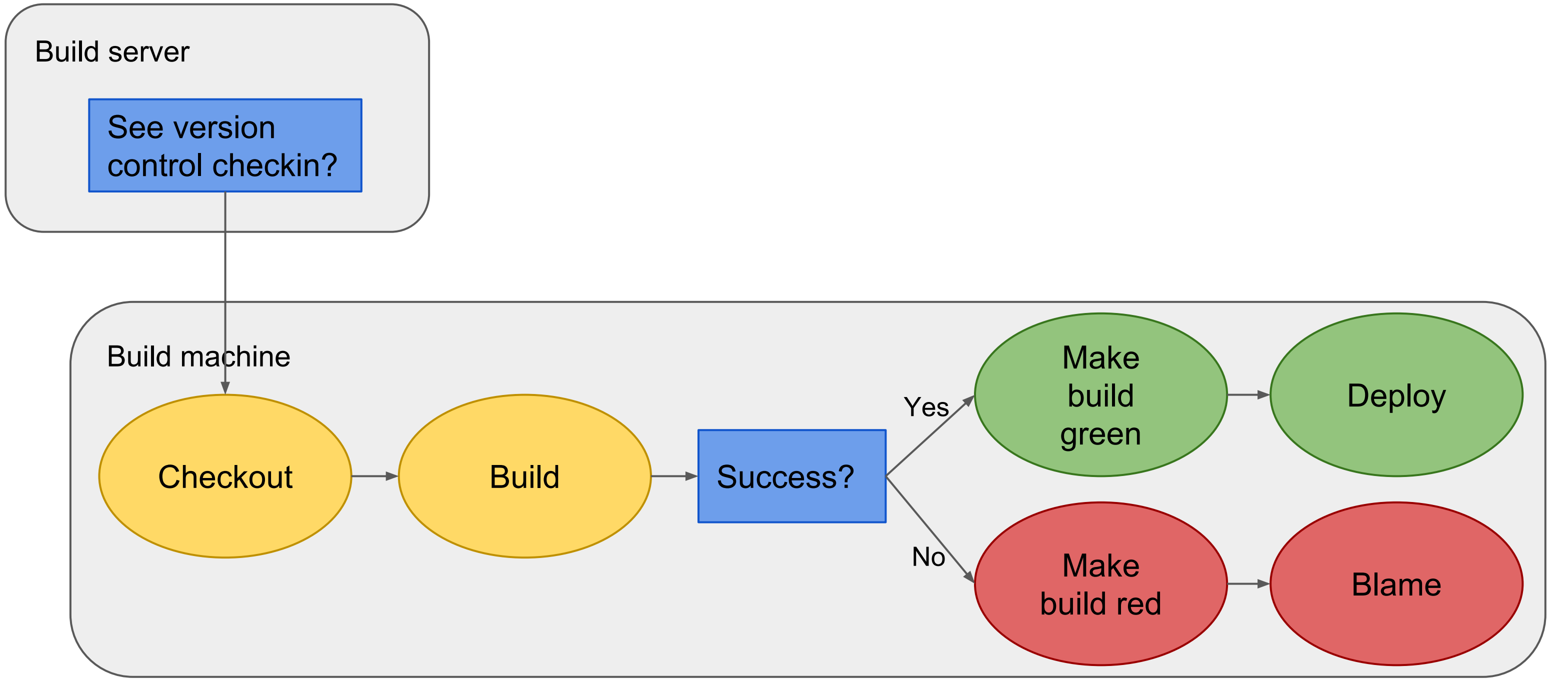
Roll your own:

- Outside the scope of this lecture.

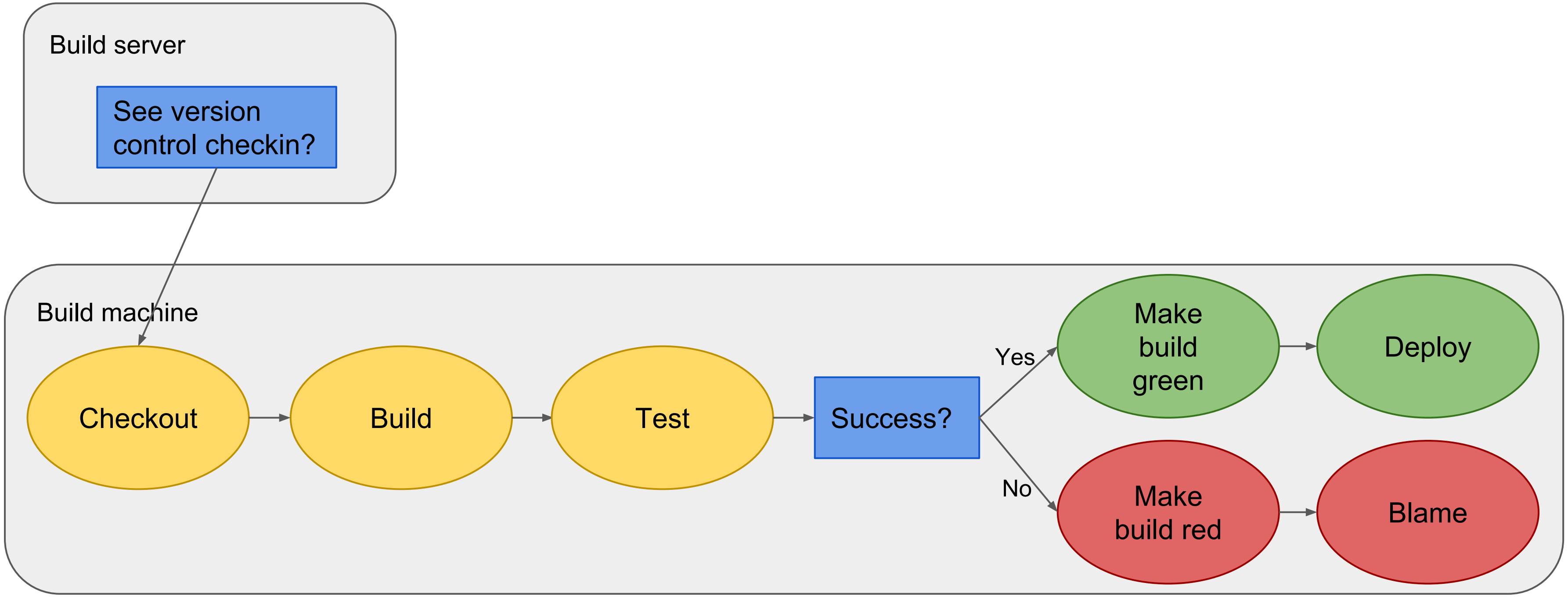
Testing

Just do it.

We are forgetting something...



That’s better.



Testing Terms

- Types of tests
- Coverage
- Test Driven Development (TDD)
- Mock object
- Blackbox testing
- Testing framework

Testing Terms

- **Types of tests**
- Coverage
- Test Driven Development (TDD)
- Mock object
- Blackbox testing
- Testing framework

- **Unit tests** - Test for a single unit of code (e.g. a function).
- **Integration tests** - Tests of interacting systems of code.
- **Regression tests** - Tests that demonstrate a fixed bug to help prevent the bug in the future.
- **Smoke tests** - Simple high level test. For example, boot the game, load all the levels, make sure nothing crashes.

Testing Terms

- Types of tests
- **Coverage**
- Test Driven Development (TDD)
- Mock object
- Blackbox testing
- Testing framework

Degree to which tests exercise all the code/content in a game.

80% coverage might mean:

- 80% of the functions are called by tests.
- 80% of the lines of code are run by tests.
- 80% of levels are loaded during tests.

Testing Terms

- Types of tests
- Coverage
- **Test Driven Development (TDD)**
- Mock object
- Blackbox testing
- Testing framework

Methodology where tests are written first.

Initially all the tests fail. Implementation is done when they all pass.

Testing Terms

- Types of tests
- Coverage
- Test Driven Development (TDD)
- **Mock object**
- Blackbox testing
- Testing framework

A fake object/system that feeds a test specific data. Isolates the code under test from issues elsewhere in the code.

For example, a fake network connection can be used in network unit tests.

Testing Terms

- Types of tests
- Coverage
- Test Driven Development (TDD)
- Mock object
- **Blackbox testing**
- Testing framework

Testing the interface to a system without consideration for the internals.

For example, a test that generates controller input to drive a player character and observes the results.

Testing Terms

- Types of tests
- Coverage
- Test Driven Development (TDD)
- Mock object
- Blackbox testing
- **Testing framework**

System in charge of enumerating tests, running tests, and reporting the results.

- Lots of off-the-shelf options
- Starting with simple assert() works fine

HOWTO Test

- There is a fair amount of testing dogma out there.
- The appropriate amount and type of testing:
  - Balances costs of testing versus costs of not testing.
  - Is highly specific to engine, type of game, team composition, etc.
  - Requires good engineering judgement.
- Costs of testing
  - Time to write the tests (often not significant)
  - Time to run the tests (possibly for every automated build)
  - Time to maintain the tests
- Costs of not testing
  - Delays in uncovering bugs make them much harder to fix
  - Harder to change code or content with confidence

Testing Rules of Thumb

- Core engine code should have unit and integration tests.
- Code related to high level ‘game feel’ should have less/no tests.
- Smoke tests should load all levels / visit all major areas.
- Tests should be concentrated in areas most likely to break.
- Tests should increase team efficiency.
  - Tests that rarely break and take a long time to run should be removed.
- Tests should be run regularly by automated systems.
- Consider supporting *golden image* tests.

Just do it

- It’s easy to procrastinate on testing.
- Don’t overthink it.
- Keep it simple.
- Something is usually better than nothing.

Summary

Foundation of any large but agile game development effort:

- Version control
- Build
- Test

End Lecture

Version control images copied from:

https://betterexplained.com/articles/a-visual-guide-to-version-control/