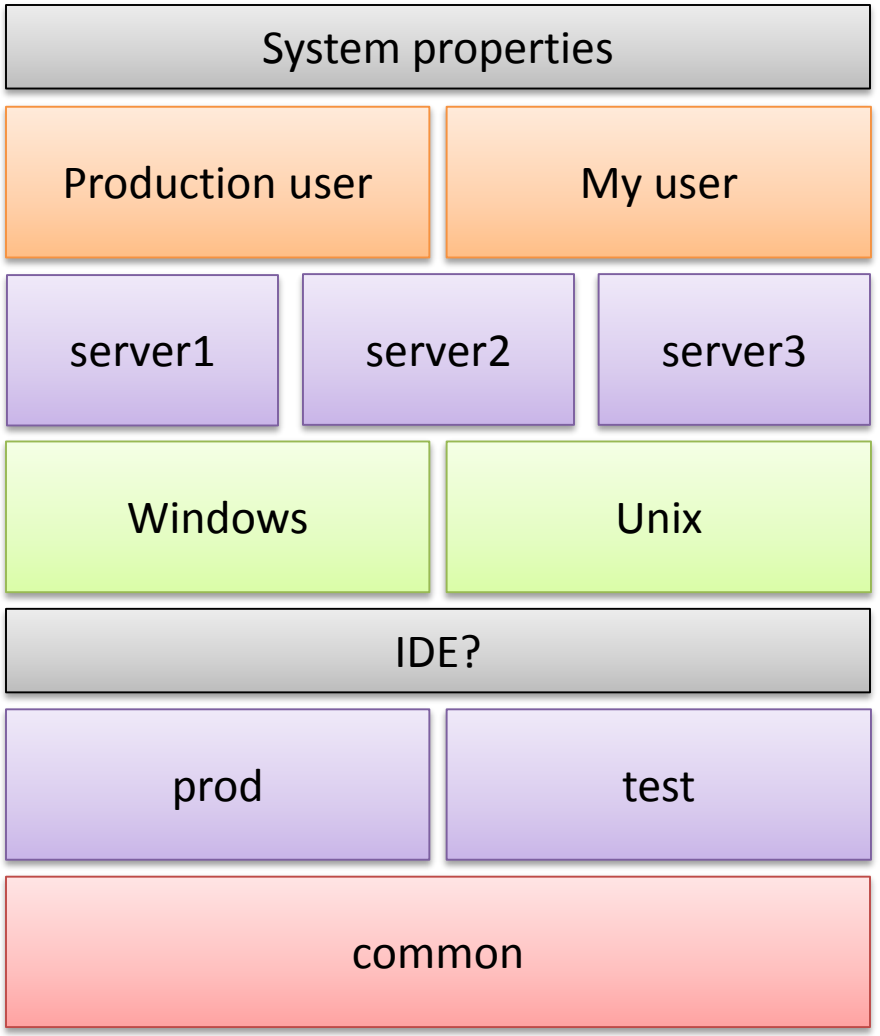
Bootstrapper

The Bootstrapper loads system properties based on

- Environments (prod, qa, dev)
- OS (Windows, Unix)
- Machines
- Users
- Whether you're in an IDE

A hierarchy of property sources allows fine-grained specification of defaults and overrides.



Property group ordering

Properties have a priority, based on the group in which they are loaded.

Each group is typically defined in its own file, e.g.

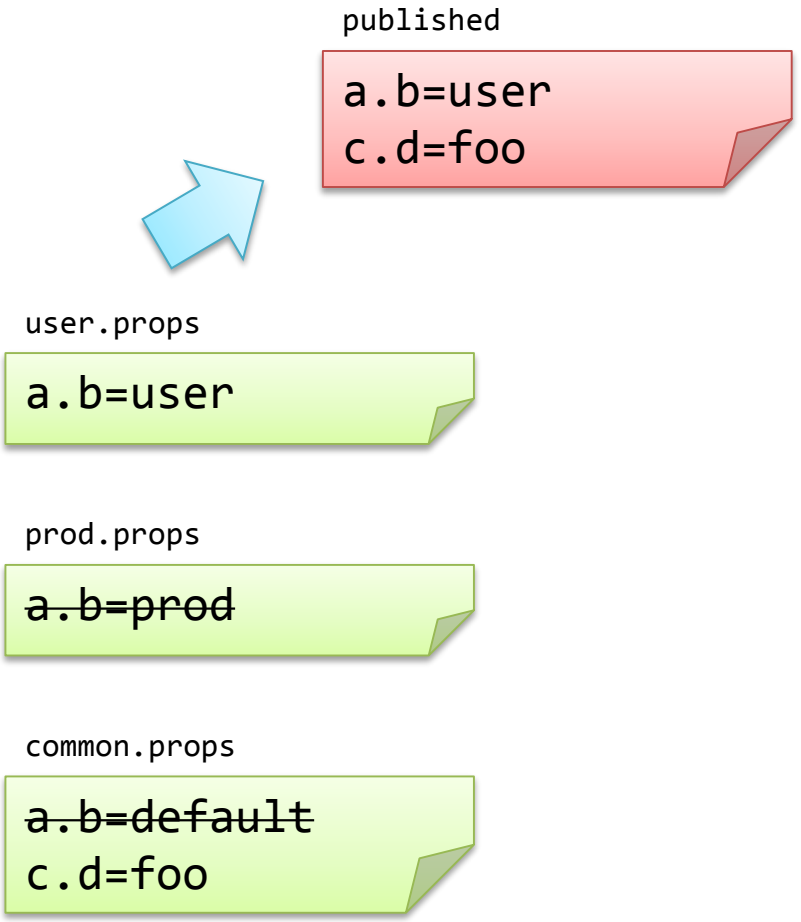
- common.properties
- prod.properties

1. Properties are loaded in **priority order, highest first** (with the exception of additional groups)
2. Properties from higher priority groups may override the location of other property files, using the property `bootstrap.properties.<group>.file`

1. System properties
2. User
3. Machine
4. Operating system
5. IDE
6. Additional groups
7. Environment
8. Common

Property overriding

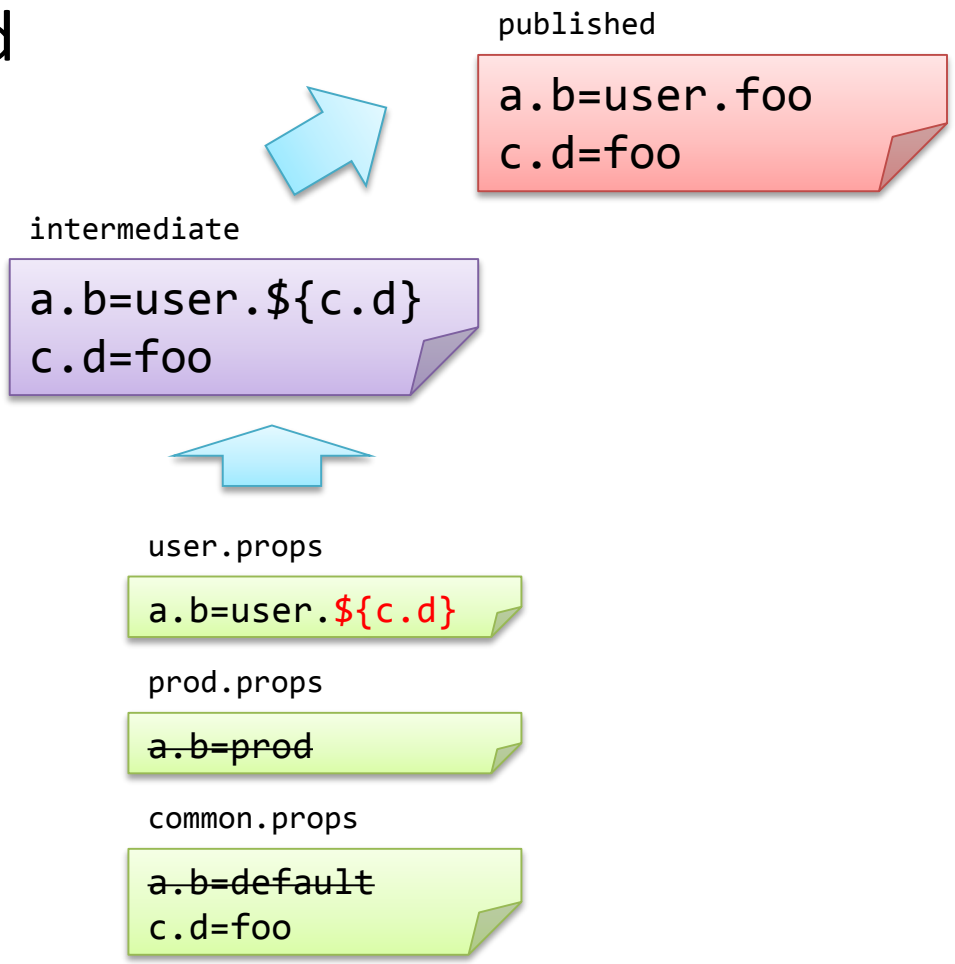
Properties defined in higher-priority files take precedence.



1. Load properties in descending priority order
2. Property `a.b` is defined in multiple places; the **highest priority source wins**, in this case the user properties

Property resolution

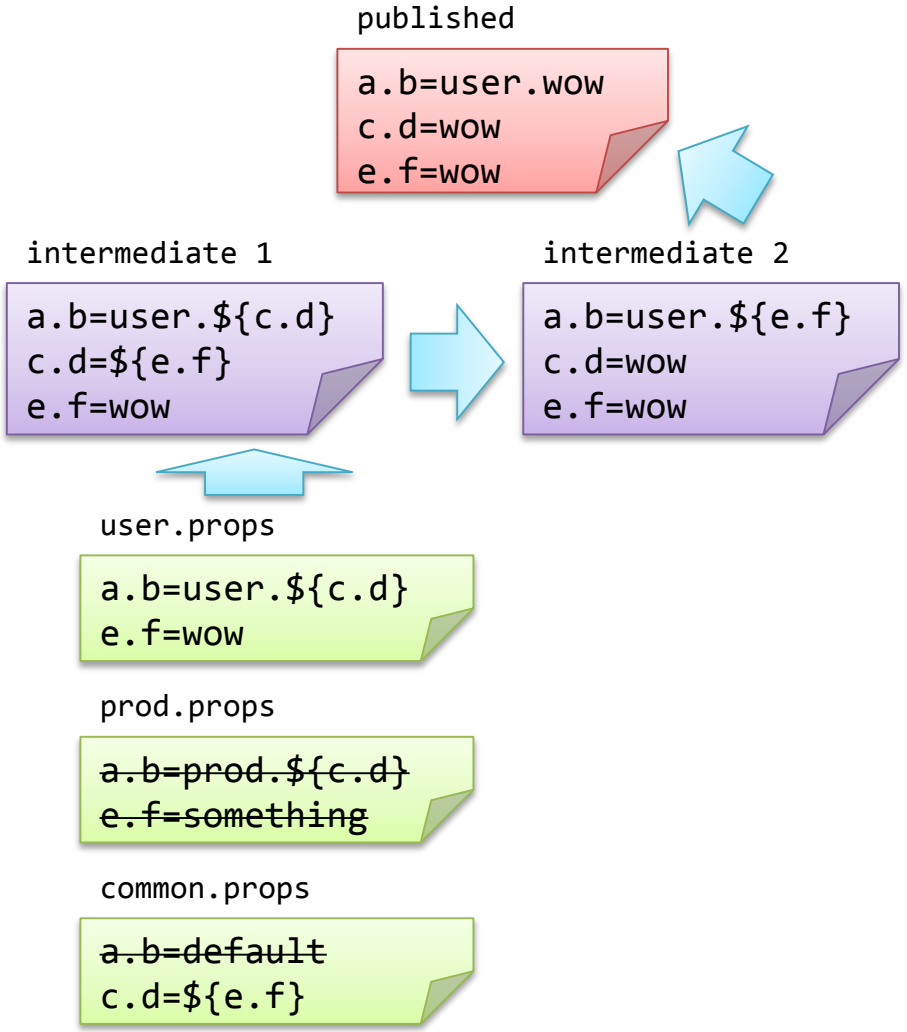
Properties can be defined in terms of others, using placeholders for lazy binding.



1. Load properties in descending priority order
2. Property `a.b` is taken from highest priority source, as before
3. Finally, outstanding placeholders are **resolved among other properties**
4. Note that placeholders can refer to properties defined in both higher and lower priority sources

Indirect property resolution

Properties can depend on other properties, and those dependencies can be overridden.



1. Reference to `c.d` resolved as a reference to `e.f`
2. Property `a.b` references resolved step by step
3. Definition of `e.f` is overridden, the higher priority value is used for its final value and for resolved values

Use in your main method

Normal usage is at application startup

```
public static void main(String ... args) {
    BootstrapMain.withApplicationName("my-awesome-app").publishTo(systemProperties());
    // rest of application
}
```

- This will publish all generated properties into system properties
- Properties can be used from the returned `PropertyProvider`

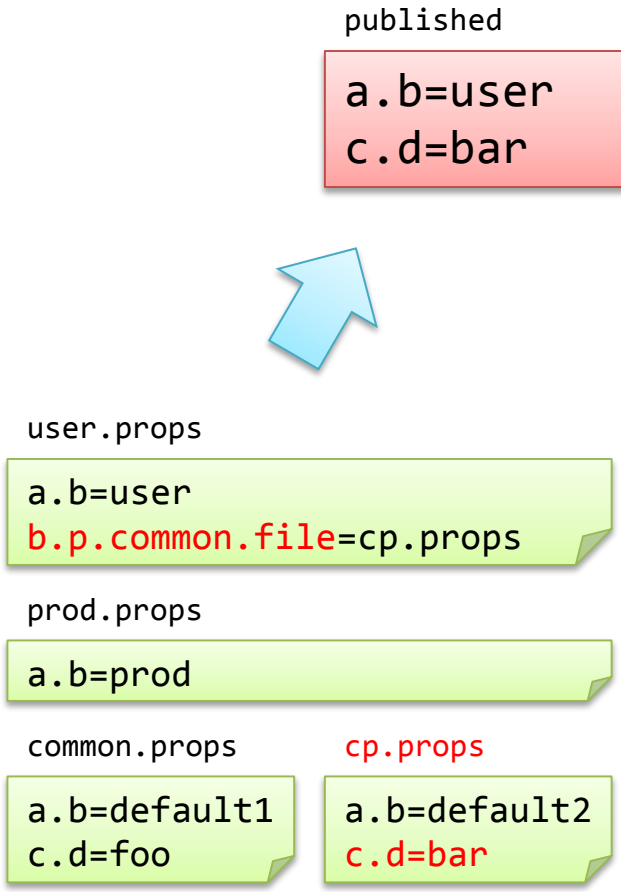
Overriding property source locations

Properties have default file locations, but these can be changed by the

`bootstrap.properties.<group>.file`

property

1. Reference to `c.d` resolved as a reference to `e.f`
2. Property `a.b` references resolved step by step
3. Definition of `e.f` is overridden, the higher priority value is used for its final value and for resolved values



Additional property groups

- Use additional property groups for GUIs, or other specific application types.
- Higher priority than environment properties.

```
BootstrapMain.withAdditionPropertyGroups("GUI")
```

1. Additional property group names are defined by `bootstrap.properties.additional.group` as a comma-separated list.
2. Files for this group are defined in `bootstrap.properties.<group-name>.file`
3. Properties from this group are loaded last, so you can define these locations in `common.properties`

1. System properties
2. User
3. Machine
4. Operating system
5. IDE
6. **Additional groups**
7. Environment
8. Common

Other PropertyProvider options

Rather than publishing to system properties, you can use other providers.

- Publish generated properties into a **map-backed provider**. This can be injected directly into components.

```
MapBackedPropertyProvider pp =
    BootstrapMain.withApplicationName("my-awesome-app").publishTo(newMap());
```

- [useful?] Publish generated properties into a **ThreadGroup-local provider**, visible to all threads within that group. Useful for configuring different multi-threaded services in one JVM.

```
ThreadGroup g = Thread.current().getThreadGroup(); // or another group
ThreadGroupPropertyProvider pp = BootstrapMain
    .withApplicationName("my-awesome-app").publishTo(threadGroup(g));
```