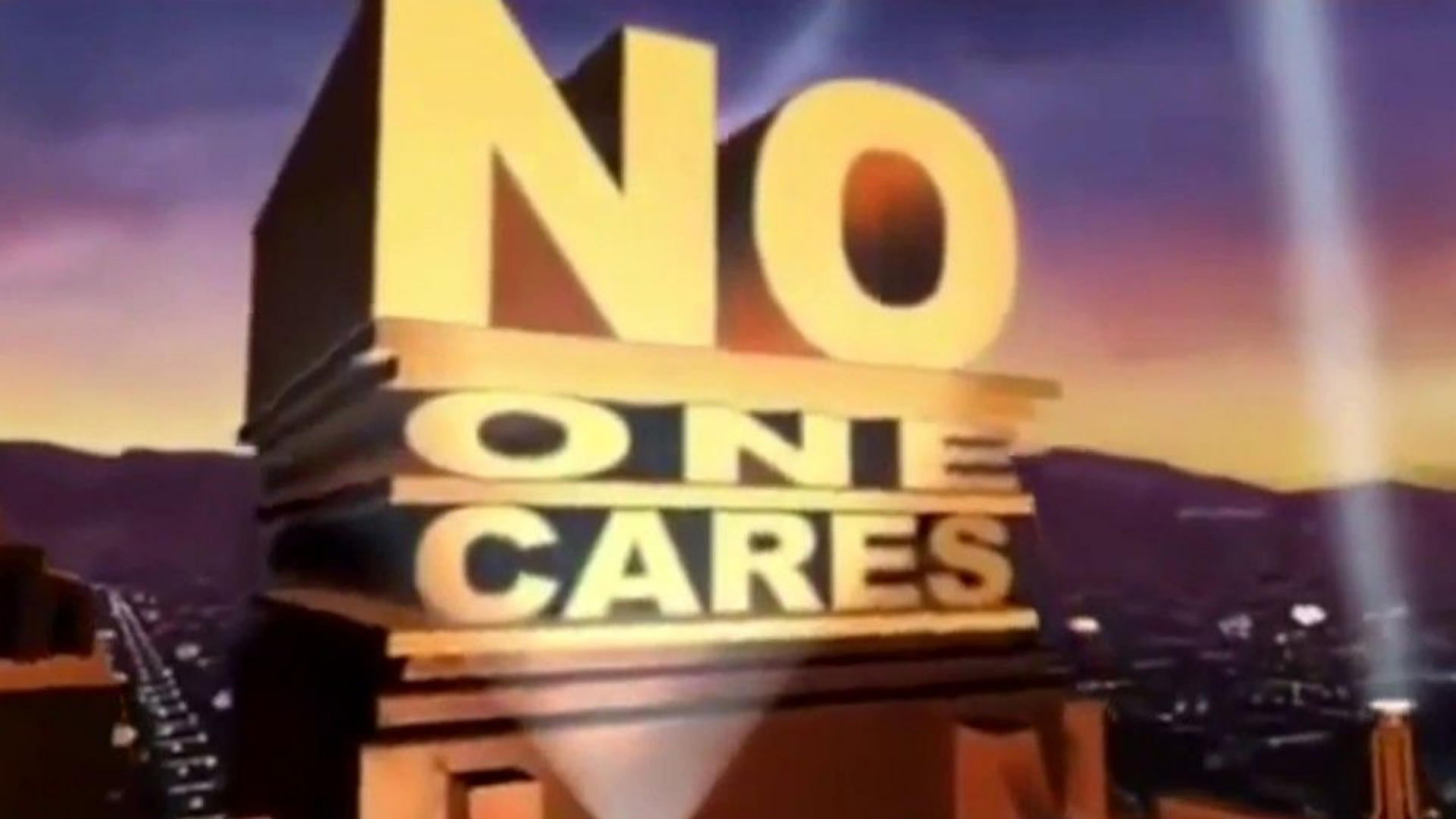

Modern CFML
No More Copy and Paste
Ortus Webinar - September 2020
Gavin Pickin


NO
ONE
CARES

Who am I?

- Software Consultant for Ortus Solutions
- Work with ColdBox, CommandBox, ContentBox every day
- Working with Coldfusion for 22 years
- Working with Javascript just as long
- Love learning and sharing the lessons learned
- From New Zealand, live in Bakersfield, Ca
- Loving wife, lots of kids, and countless critters

@gpickin on twitter

http://www.ortussolutions.com

Ortus Support can help

- We try to provide a lot of free content and advice, but sometimes you need some paid support.
- We help a lot of customers with legacy migration and modernization
- Lots of paid content, trainings, conference and support options
- For a full list of services: https://www.ortussolutions.com/services
- For support packages: https://www.ortussolutions.com/services/support

Including a new support option - Space Ninja Subscription

SpaceNinja™ Subscription

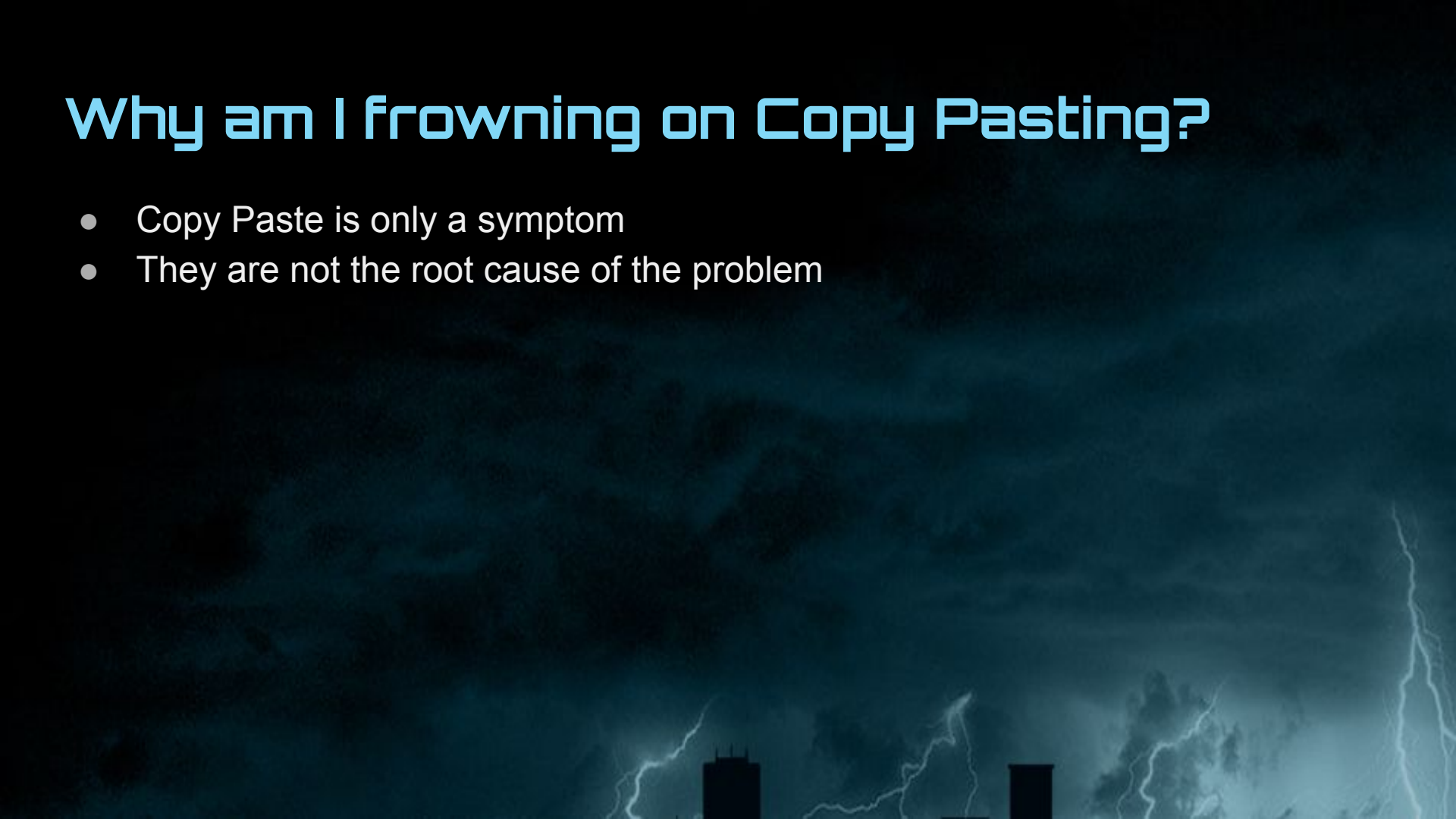
Get unrestricted product support, and then some...

$2,000/Year

NO Retainer
Schedule Video Calls
Less than 24 hours Reponse Times
Covers ALL BOX Products
Covers ALL Ortus Libraries
Dedicated Slack Channel
+ 1 Year Access to CFCasts
+ 1 Year License for FORGEBOX PRO
+ 20% OFF on all Workshops
+ You will be supporting all our Open Source Initiatives and we'll be adding you to our Open Source Sponsors page

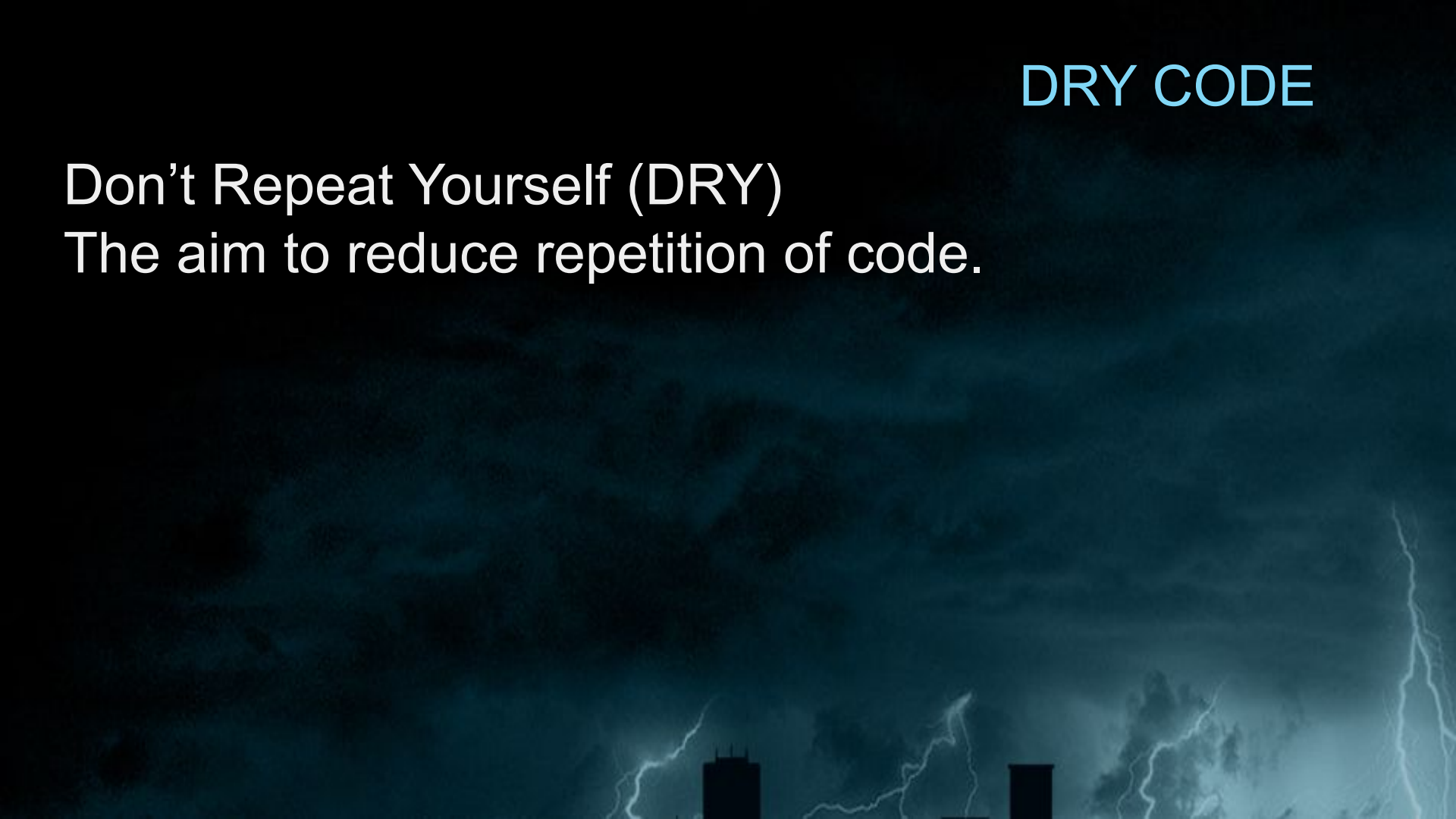
Why am I frowning on Copy Pasting?

- Copy Paste is only a symptom
- They are not the root cause of the problem



DRY CODE

Don’t Repeat Yourself (DRY)
The aim to reduce repetition of code.



WET CODE

Write Everything Twice (WET)
Code that doesn’t adhere to DRY principle.

Maintainability

- The biggest benefit of using DRY is maintainability.
- Fixing bugs in one location is easier than remembering and finding all the occurrences of that logic.
- Updating logic or adding functionality is easier in 1 location instead of many

Another Acronym - FIRST

Components Should Be (FIRST):

- Focused
- Independent
- Reusable
- Small &
- Testable

https://addyosmani.com/first/

Readability

- If a developer understands the principles of dry code, they are more likely to understand clean code principles, and that should lead to more readable code.
- Code reuse usually encourages smaller more manageable functions, which promotes lower cognitive load, and therefore more readable.

Programs must be written for people to read, and only incidentally for machines to execute.

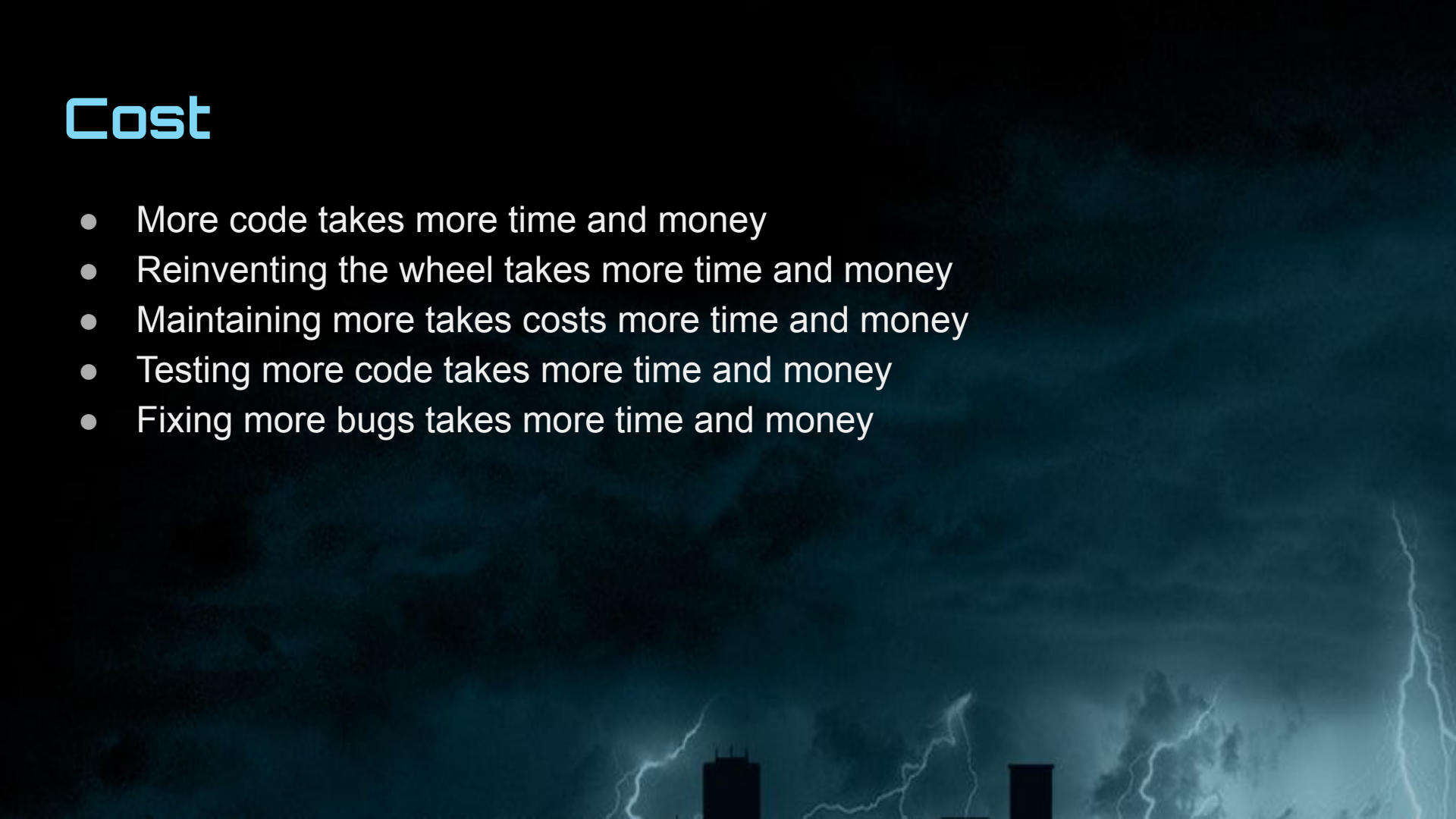
Harold Abelson.

Testing

- Usually DRY code is broken down into smaller pieces, usually it is easier to test
- The more code you have, the more paths and functions to cover, the harder to test
- Using DRY Code from libraries and frameworks means more eyes on it, and sometimes those libraries and frameworks have tests for their code already.

Cost

- More code takes more time and money
- Reinventing the wheel takes more time and money
- Maintaining more takes costs more time and money
- Testing more code takes more time and money
- Fixing more bugs takes more time and money



How do we prevent Copy-Pasting?

We need fast and simple ways to share code in a project, and between projects

How to write DRYer code in CFML

Some tools CFML gives you

- CFINCLUDE
- CFMODULE
- Custom Tags
- User Defined Functions (UDFs)
- CFCs with Functions
- Externally Sourced Frameworks / Libraries

CFInclude

- Well known
- Popular
- Easy to use
- Can reference local or absolute locations for easy sharing

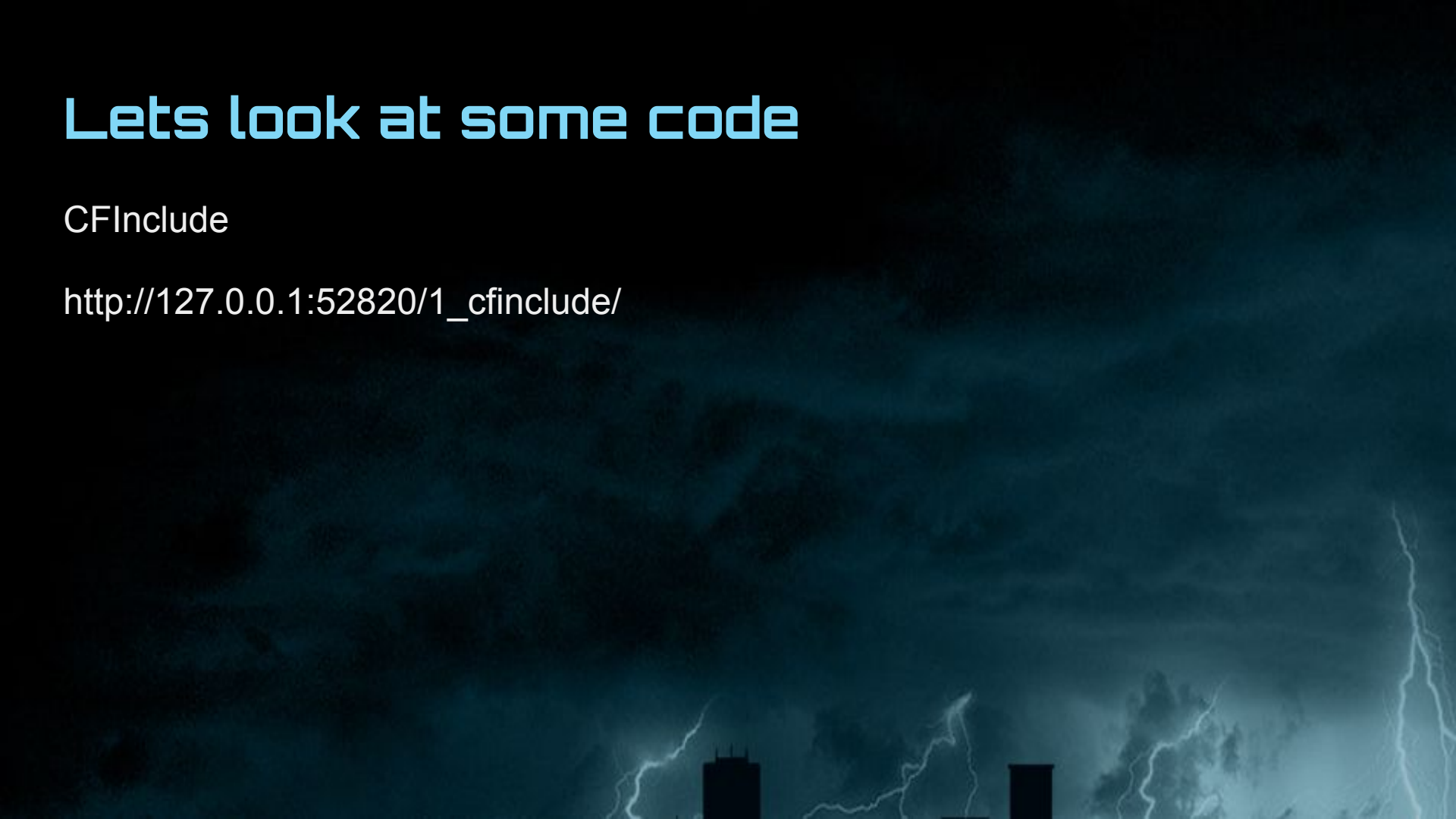
Cons:

- It absorbs everything around it
- Hard to control inputs

Lets look at some code

CFInclude

http://127.0.0.1:52820/1_cfinclude/



CFModule

- Black boxes your code, so it only can access what you give it
- Can make your code more flexible, since you can control the inputs
- Can access the caller for access to the parent - can break encapsulation
- Can reference local or absolute locations for easy sharing

Cons

- Not well known
- People don’t seem to like it - especially since it’s equivalent to a custom tag
- Can access the caller for access to the parent - breaks encapsulation and create unintended or unexpected side effects

Lets look at some code

CFModule

http://127.0.0.1:52820/2_cfmodule/

Custom Tags

- Gives you tag syntax to allow you to wrap content nicely (Great for UI)
- Can make your code more flexible, since you can control the inputs
- You can access that tag content inside of your custom tag as well
- Can access the caller for access to the parent - can break encapsulation
- Engines has central Custom Tag locations for sharing between projects

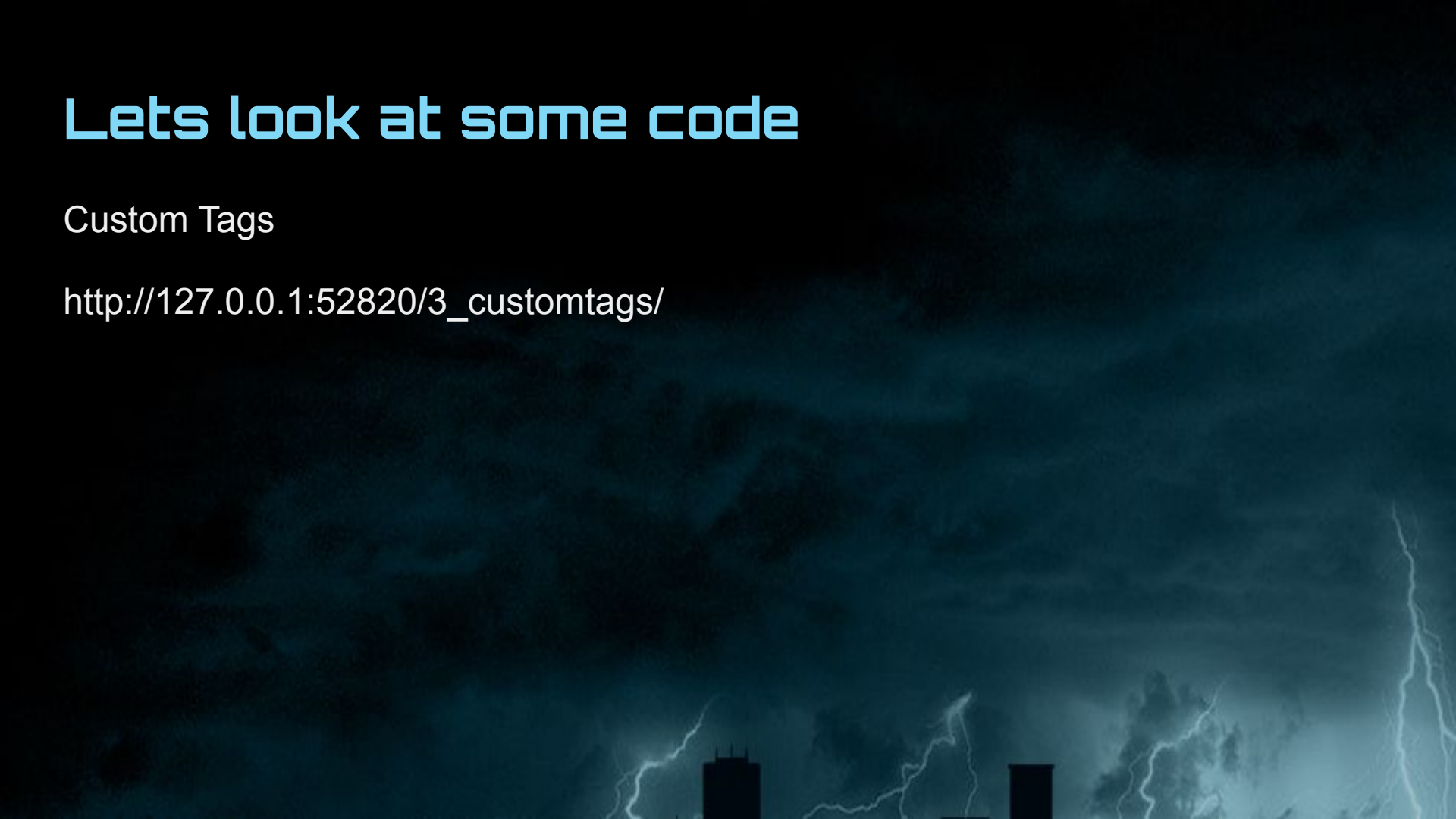
Cons

- Since it wraps code, it executes two times, and confuses people sometimes
- Executes 2 times, even with a self closing tag
- People don’t seem to like it - although it was powerful
- Can access the caller for access to the parent - breaks encapsulation and create unintended or unexpected side effects

Lets look at some code

Custom Tags

http://127.0.0.1:52820/3_customtags/



User Defined Functions (UDFs)

- Can make your code more flexible, since you can control the inputs
- Simple to create, include multiple for simplicity
- Only runs when you call the function
- Can access the calling environment if you need to

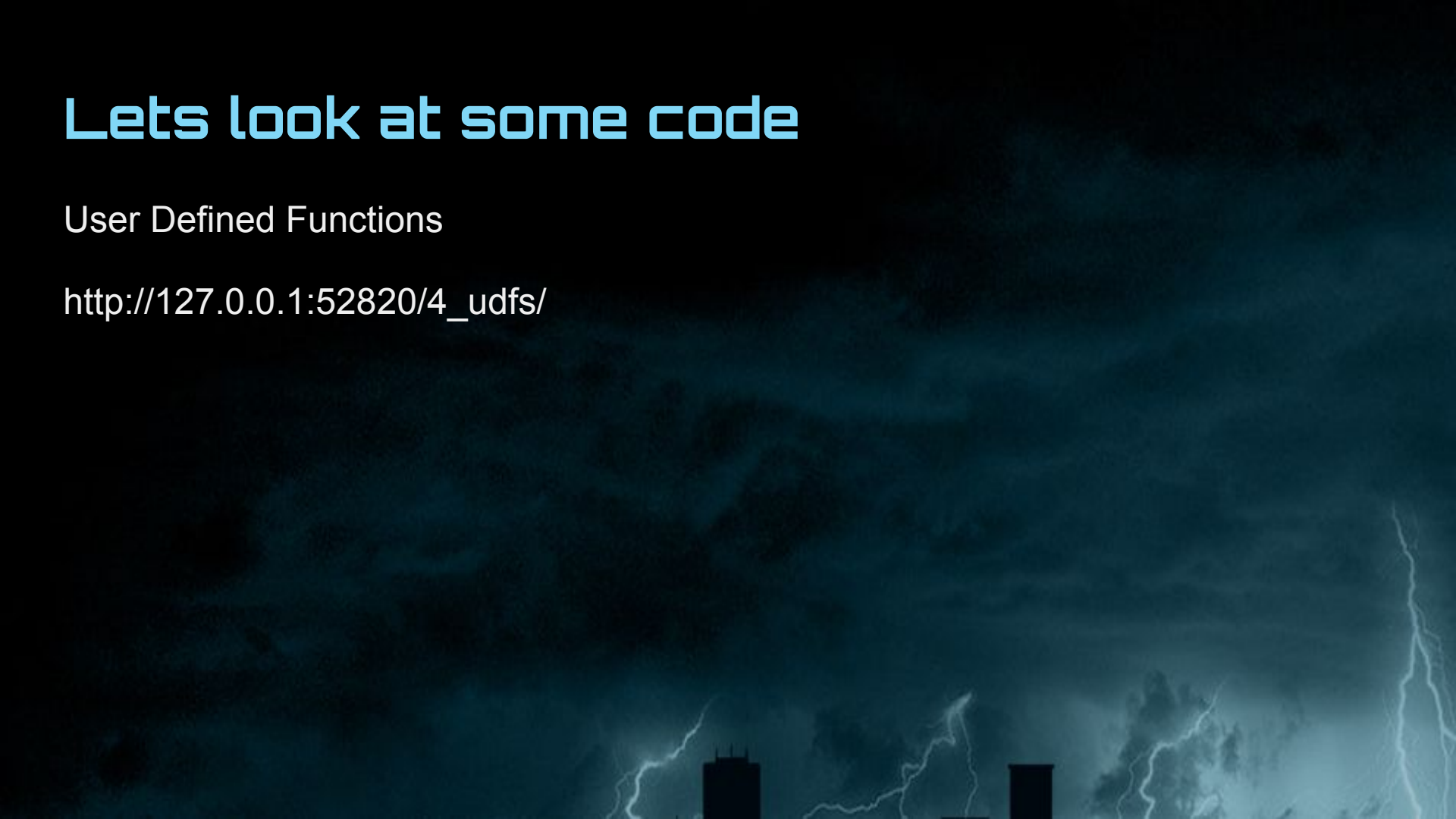
Cons

- Can access the calling environment - breaks encapsulation and create unintended or unexpected side effects
- Including lots of functions everywhere can bloat your bytecode?

Lets look at some code

User Defined Functions

http://127.0.0.1:52820/4_udfs/



CFCs with Functions

- Can make your code more flexible, since you can control the inputs
- Simple to create, include multiple in utility cfcs
- Only runs when you call the function
- You can mixin cfm files into CFCs
- You can extend CFCs to other CFCs
- Data doesn’t leak from the page into your function

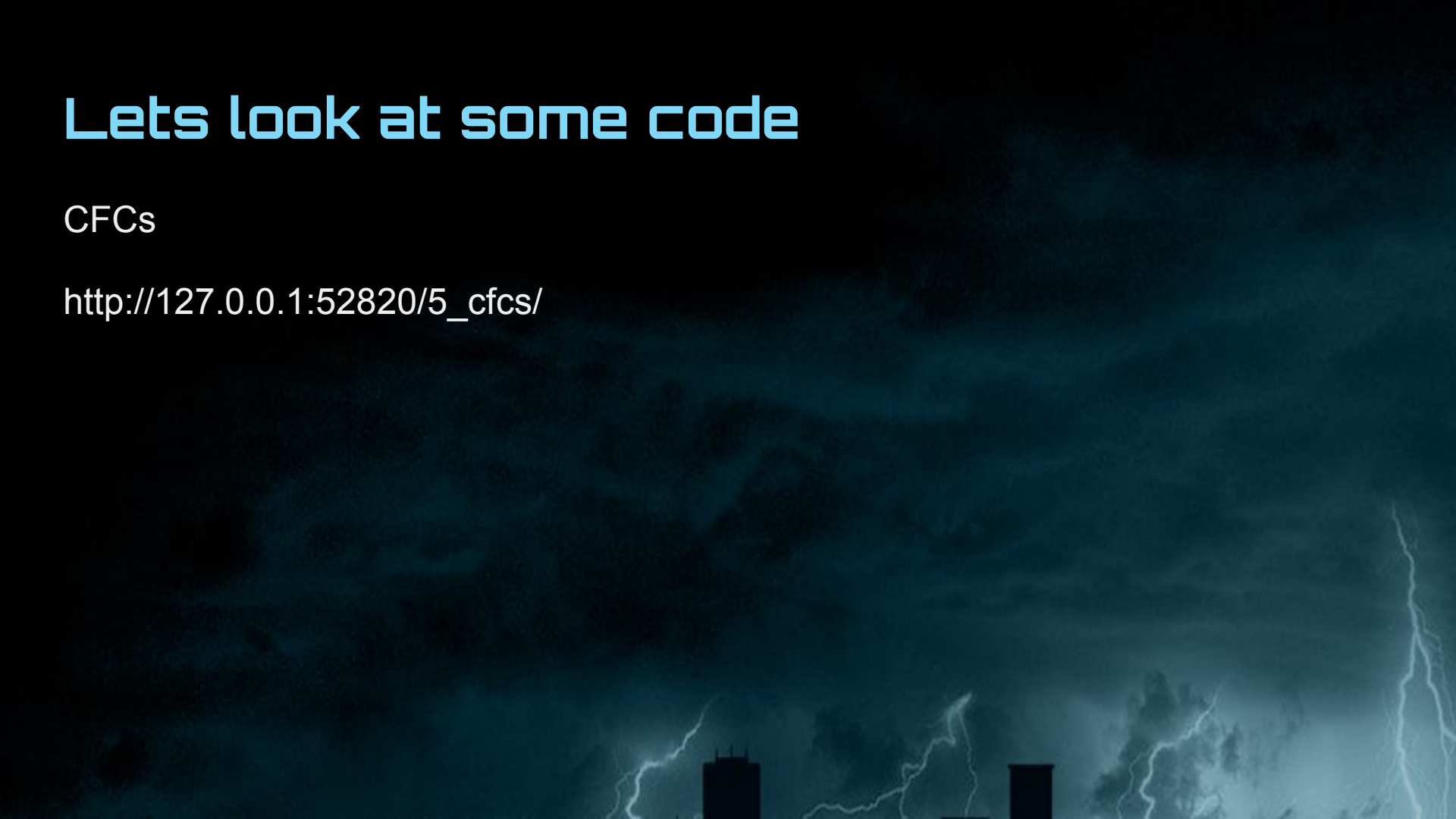
Cons

- Creating CFCs inside of other CFCs and in every page can bloat your bytecode?
- If you mixin the same functions to every CFC, isn’t that going to bloat your bytecode?

Lets look at some code

CFCs

http://127.0.0.1:52820/5_cfcs/



WireBox and how to manage CFCs

WireBox alleviates the need for custom object factories or manual object creation in your ColdFusion (CFML) applications.

It provides a standardized approach to object construction and assembling that will make your code easier to adapt to changes, easier to test, mock and extend.

Advantages of a DI Framework

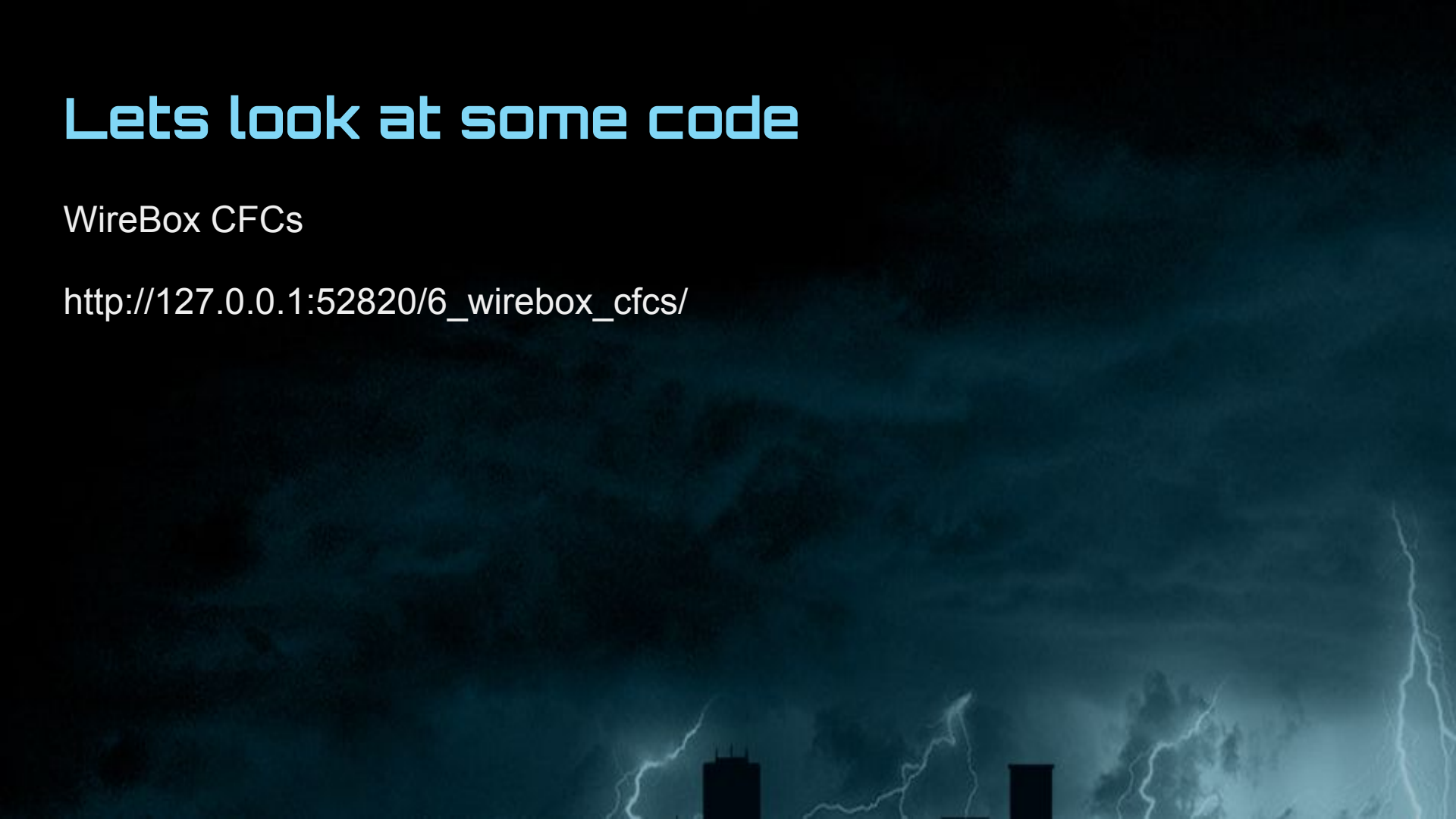
Compared to manual Dependency Injection (DI), using WireBox can lead to the following advantages:

- You will write less boilerplate code.
- By giving WireBox DI responsibilities, you will stop creating objects manually or using custom object factories.
- You can leverage object persistence scopes for performance and scalability. Even create time persisted objects.
- You will not have any object creation or wiring code in your application, but have it abstracted via WireBox. Which will lead to more cohesive code that is not plagued with boilerplate code or factory code.
- Objects will become more testable and easier to mock, which in turn can accelerate your development by using a TDD (Test Driven Development), BDD (Behavior Driven Development) approach.
- Once WireBox leverages your objects you can take advantage of AOP or other event life cycle processes to really get funky with OO.

Lets look at some code

WireBox CFCs

http://127.0.0.1:52820/6_wirebox_cfcs/



Externally Sourced Frameworks / Libraries

- Stops you from reinventing the wheel
- Learn from others mistakes and gain from their security and coding practices
- Contributions from many people can give you options you haven’t thought about yet.

Managing your Libraries and Frameworks

If you are managing your libraries:

- You just have a bunch of (possibly modified) files
- You'll have to guess where they came from and figure out how to update and if you should update
- You'll have to guess what version they are (or trust some README).
- You have to commit all of these BIG chunky libraries and frameworks into source control
- Have different conventions for different libraries
- Encourages machine wide installations, which can restrict other projects

Common Myths to Package Management

- Another tool to manage and deal with - Modern developers have lots of tools, this one extra one will save you time and headaches
- Old school is simpler - not when you have to manage your nightmare lib folder manually.
- What about building without Internet - Most package managers have offline modes
- Builds are faster without packaging - offline and artifact based builds are very fast

Package Management - ForgeBox

Knowing which libraries you use, and which version, is very important if you:

- need to update a library due to a critical bug / security hole;
- or just need to check whether an announced security hole affects you.

Installation Method Comparisons

Install https://github.com/ColdBox/coldbox-platform/archive/master.zip
2.607mb zipped
3.43mb installed

Install https://downloads.ortussolutions.com/ortussolutions/coldbox/6.0.0/coldbox-6.0.0.zip
494kb zipped
1.67mb installed

Install coldbox@6
494kb zipped
1.67mb installed

Installation Method Comparisons

ForgeBox packages saves you because:

- You can tell if the version you have is the most recent by package information
- You don’t have to download the repo/zip every time because you don’t know what version the contents of the repo/zip is
- Only downloads new versions if you haven’t got that download in artifacts

ForgeBox supports semver ranges, and only installs the updates you specify with the version in the package, and the semver range in your box.json

Download penalties?

- ColdBox is small
- No jars
- No binaries
- I have fast internet (usually)

What if your packages are bigger, your internet is slower?

Those penalties add up

FAST!

Can code be too dry?

- Moderation is the key to most things
- You don’t want overdry cake, same with your code

- Don’t prematurely optimize your Code
- Don’t assume 2 pieces of code should be combined, sometimes the differences warrant code separation.

Questions

Thanks

Repo: https://github.com/gpickin/202009_webinar_moderncfml