S.O.L.I.D Principles

Chris Klug | Dev/Architect @ Active Solution | @zerokoll

Who am I?

The S.O.L.I.D Principles

How many of you...

...have heard about S.O.L.I.D?

How many of you…

…can name all 5 principles?

How many of you...

...can explain Liskov's Substitution Principle?

S	O	L	I	D
SRP	OCP	LSP	ISP	DIP
Single Responsibility Principle	Open/Closed Principle	Liskov Substitution Principle	Interface Segregation Principle	Dependency Inversion Principle

Single Responsibility Principle

"A class should have only one reason to change"

"There is always only one reason to change…changing requirements"

-Brilliant developer no. 1

There can be only one requirement that, when changed, will cause a class to change…

What about things like repositories for example?

And the benefits?

Open/Closed Principle

"Software entities should be open for extension, but closed for modification"

"So I should write code that can be made better without changing it? Are you on drugs?"

-Brilliant developer no. 2

Once it's done, it's done!

Meyer vs. Polymorphic

And the benefits?

Liskov Substitution Principle

*"Let **q(x)** be a property provable about objects **x** of type **T**. Then **q(y)** should be provable for objects **y** of type **S** where **S** is a subtype of **T** "*

"Ehh…what did she say?!?"

– People attending Barbara Liskov's keynote entitled "Data abstraction and hierarchy"

A subclass should behave in such a way that it will not cause problems when used instead of the superclass

"Rules"

Contravariance of method arguments in sub class

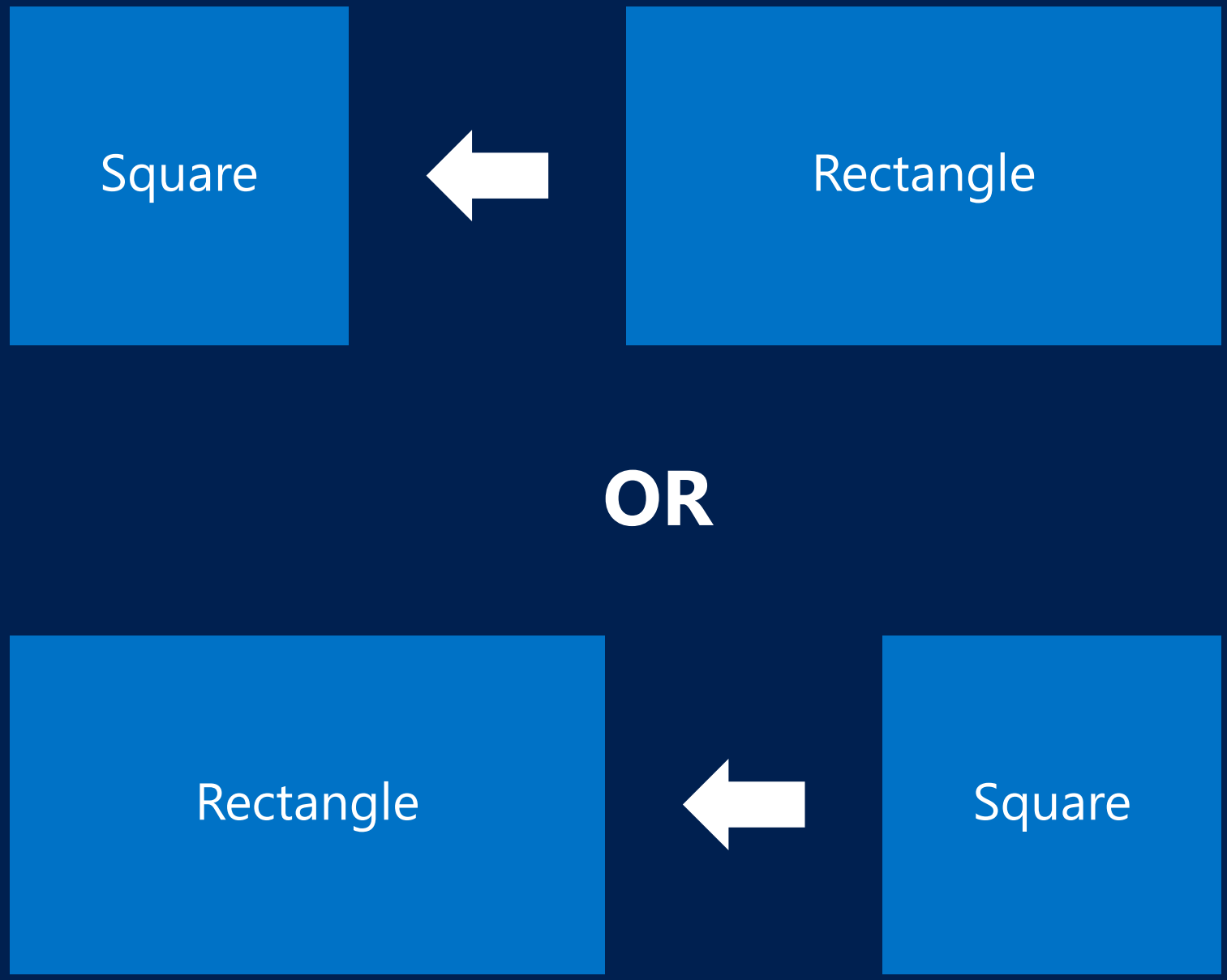
Covariance of return types in the sub class

No new exception types are allowed to be thrown, unless they are sub classes of previously used ones

Preconditions cannot be strengthened in a subtype

Postconditions cannot be weakened in a subtype

The history constraint


Square
Rectangle
OR
Rectangle
Square

What about abstract base classes and interfaces?

And the benefits?

Interface Segregation Principle

"Clients should not be forced to depend upon interfaces that they don't use."

"Who cares if a client gets a bit more than he needs? It can just ignore it..."

- Made up person no 1

Breaking down interfaces in smaller pieces make them easier to implement, and offers more control over who sees what

And the benefits?

Dependency Inversion Principle

"A. High-level modules should not depend on low-level modules. Both should depend on abstractions.

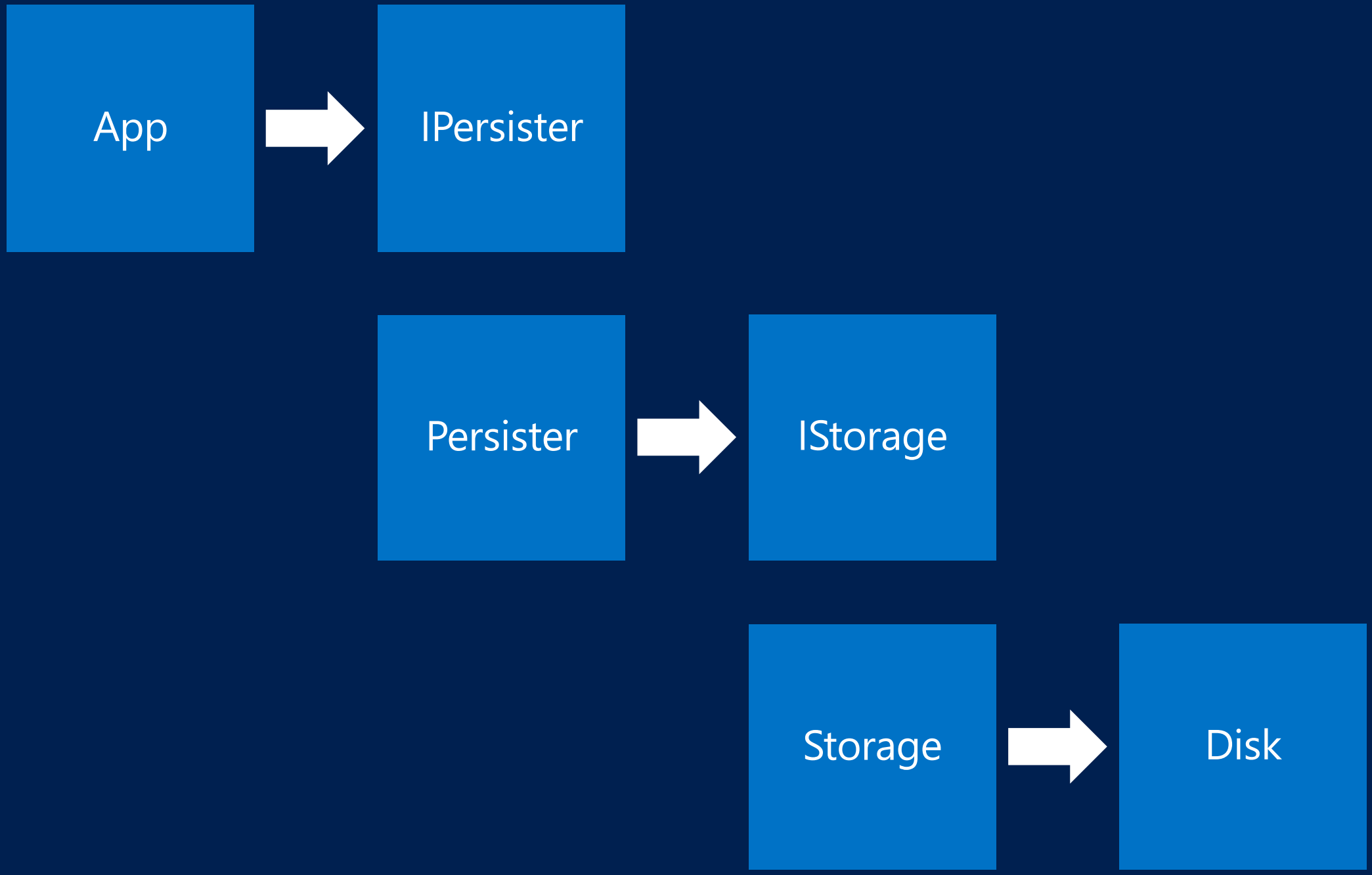
B. Abstractions should not depend upon details. Details should depend upon abstractions."

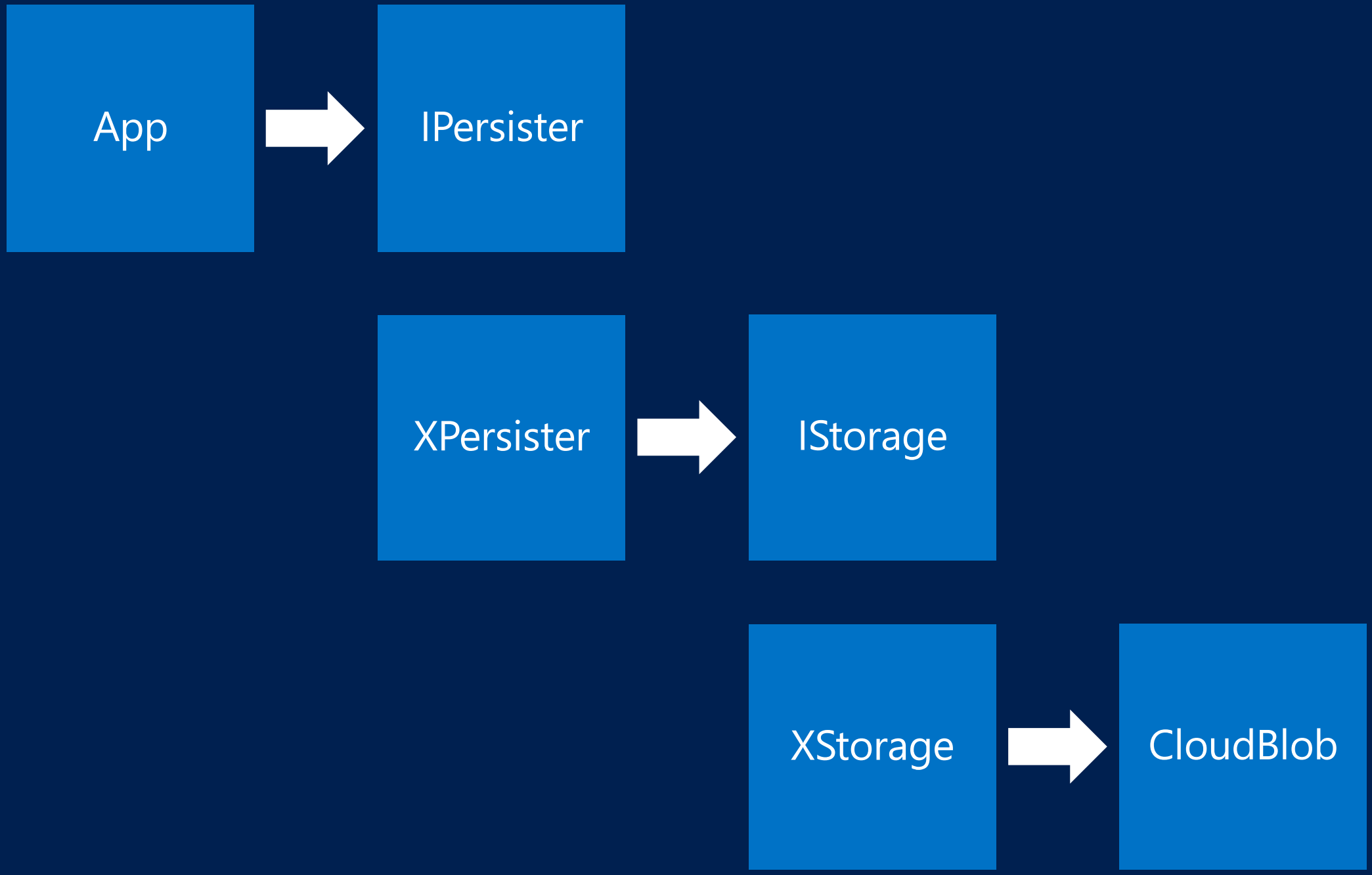
"Ok, so everyone should depend on abstractions. Is anyone actually going to implement anything, or is this whole thing just going to be an abstraction?"

- Made up person no 2

By making sure classes don't depend on specific implementations, it becomes easy to change things around...


App
Persister
System.IO.
File


App
IPersister
Persister
IStorage
Storage
Disk


App
IPersister
XPersister
IStorage
XStorage
CloudBlob

And the benefits?

Some considerations

- Over engineering and premature optimization
- Readability
- Debatability
- 50 vs. 230

But…

Single Responsibility Principle

“Pointlessly Vague Principle”

- Dan north

"A class should have only one reason to change"

"This principle is about people"

- Uncle Bob himself

"Aim for high cohesion and low coupling, and refactor if necessary"

- me

Open/Closed Principle

Coined in the 80s in a book about EIFFEL, justifying the design choices made in EIFFEL by Meyer who designed EIFFEL

Changing a library by adding fields or methods forced all existing clients to update their code…

Also…if your code isn't published somewhere like NuGet or NPM, you can do whatever you want…

Liskov Substitution Principle

…is all about behavioral subtyping, not syntactical compatibility…

*"Let **q(x)** be a property provable about objects **x** of type **T**. Then **q(y)** should be provable for objects **y** of type **S** where **S** is a subtype of **T** "*

“Just, don't break the contract”

- me

Interface Segregation Principle

“If you have done the S properly, this is probably not required”

- me

Dependency Inversion Principle

Is **AWESOME**!

Is **AWESOME**! But…

Thank you!

Chris Klug | @zerokoll