Smart ways to break backward compatibility

Jakub Winkler · qoliber

30-minute talk · 15-minute argument afterwards

GET YOUR SEMANTIC VERSIONING SHIT IN ORDER.

MAJOR

Incompatible
public change

MINOR

Compatible functionality
or deprecation

PATCH

Compatible
bug fix

Deliberate change ≠ surprise breakage.

Declared public API · stable versions, 1.0.0+

Semantic Versioning 2.0.0

Illustrative · documented styling extension point

1.4.0 → 1.4.1. STILL A PATCH?

PACKAGE 1.4.0

```
@badge-color: #ffcc00;
```

PACKAGE 1.4.1

```
@badge-accent: #ffcc00;
// old name removed; no alias
```

UNCHANGED CHILD THEME

```
.product-badge { color: @badge-color; }
```

```
variable @badge-color is undefined
```

Less · variables

1.3.0 → 1.4.0. WHO HAS TO CHANGE?

BEFORE

```
public function __construct(
    Context $context
) {}
```

AFTER

```
public function __construct(
    Context $context,
    Config $config
) {}
```

UNCHANGED CHILD-CLASS CALL

```
parent::__construct($context);
```

ArgumentCountError

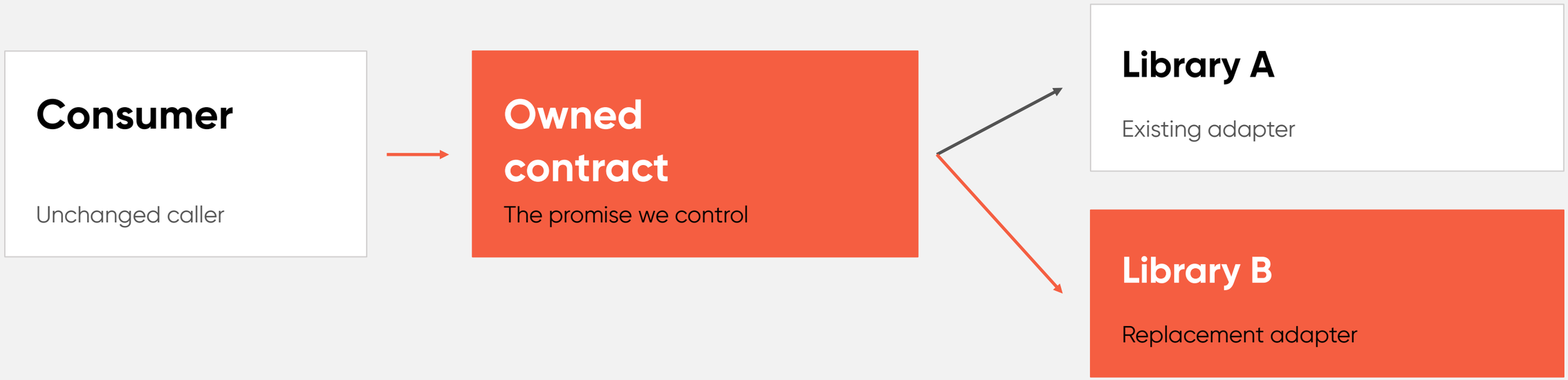
Too few arguments.

Autowiring cannot rewrite this explicit call.

PHP · argument rules

Adobe · constructor guidance

NEW LIBRARY. SAME CONTRACT?



Same supported behaviour: can be compatible.

Changed public types, requirements or behaviour? Evaluate the break.

SemVer · dependency changes

Why is compatibility sacred—but versioning negotiable?

Keep the promise. Or announce the change.

IT PROTECTS REAL COMMITMENTS.

01

Working integrations

Keep useful workflows working.

02

Merchant investment

Respect what people already built.

03

Predictable updates

A fix should not hide a migration.

04

Time to migrate

Give consumers a deliberate route.

WHEN YESTERDAY BECOMES MANDATORY.

01

Parallel code paths

Another path. Another test obligation.

02

Preserved quirks

An accident becomes a dependency.

03

Blocked replacement

The workaround gets a veto.

04

Less room to build

More effort goes into standing still.

Evaluate the cost—not every stable interface is harmful.

The BC Church.

Every old behaviour is sacred.
Every removal is heresy.

Doing the same thing over and over again and expecting different results.

Keep it. Wrap it. Work around it. Repeat.

Attribution caution · not a verified Einstein quote

THE PROMISE HAS AN IMPLEMENTATION COST.

PRESERVE THE EXISTING CALLER

```
?NewDependency $new = null

$this->new = $new
    ?? ObjectManager::getInstance()
        ->get(NewDependency::class);
```

Existing callers survive.

Another path.
Another obligation.

Adobe's policy also includes deprecation and removal rules.

Adobe · backward compatibility policy

OTHER PROJECTS CHOSE A DESTINATION.

Microsoft	**.NET Framework → .NET Core** New platform. Explicit compatibility limits.
Laravel 9	**SwiftMailer → Symfony Mailer** Replace an unmaintained dependency.
Symfony	**Guard → authenticators** Deprecated in 5.3. Removed in 6.0.
Apple	**Rosetta** Translation during a platform transition.

Sources in project labels · full links in notes

"freeze the API at the cost of also freezing innovation and technical progress."

Lübke et al. · Interface Evolution Patterns

Original paper · PDF page 18

THE COST OF FOREVER.

> “an anchor that keeps a project stuck in the stone age”
>
> **Anthony Ferrara**

> “very nasty, very expensive technical debt”
>
> **chromatic**

> “Opportunities for innovation are missed.”
>
> **Lübke et al.**

> “this sort of technical debt sticks around forever”
>
> **Dan Luu**

Different arguments and contexts—not a consensus against compatibility.

Starting point—or ceiling?

Compatibility can enable adoption.

It must not end the design discussion.

Fine.
Replatform the admin.

NEBULA

A new admin model.
Not a replacement for the whole store.

TRANSLATE THE CONTRACT. DON'T CLONE THE OLD ENGINE.

Native definition first. Supported conversion when needed.

Magento UI Component config
Supported grids / forms

→

PHP resolver + converter registry
The translation boundary

→

Nebula definition
Data · operations · ACL

`UI Bridge · nebula/module-ui-bridge`

Not a runtime for arbitrary Knockout or RequireJS customizations.

UI Bridge · scope and resolution

Native definition engine

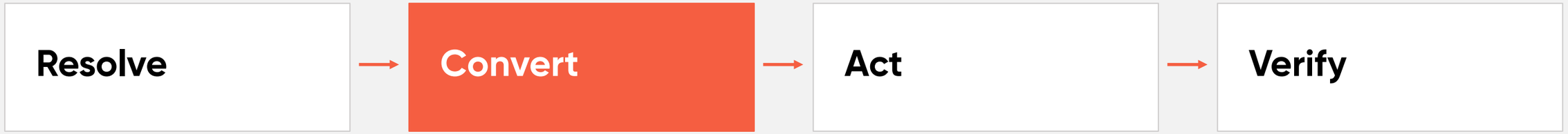
Isn’t that backward compatibility?

Yes.

At a chosen boundary.

Preserve the useful contract. Free the implementation.

EXISTING COMPONENT.
NEW DEFINITION.
REAL OPERATION.



A rendered screen is not proof. Verify the operation.

REHEARSAL PREP: select the real component / operation; recordings not embedded.

UI Bridge · supported conversion and precedence

Demo 02 · live + recorded fallback · 3:30

NOW THE NEW MODEL OWNS IT.

Native definition → **Useful change** → **Verified result**

No runtime conversion for this component.

REHEARSAL PREP: select the real component / operation; recordings not embedded.

UI Bridge · supported conversion and precedence

DEBT DOESN'T DISAPPEAR. RETIRE A PATH.

01

Migrate a consumer

→

02

Verify the new contract

→

03

Remove the obsolete path

When support commitments permit removal.

A useful bridge can remain. Accidental forever is the problem.

TEN TOOLS.
TWO WE USE AND RECOMMEND.

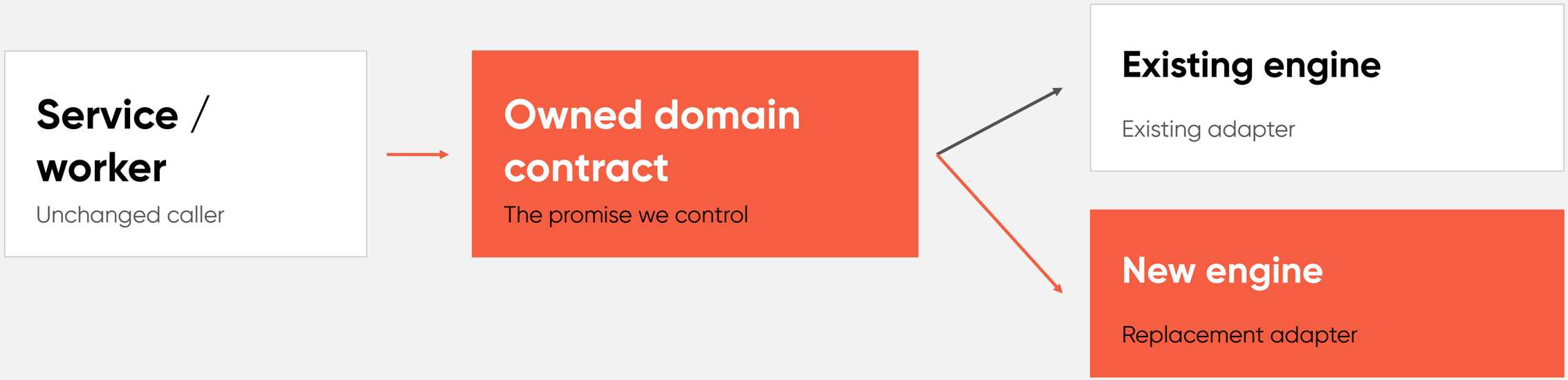
01	**Technology-neutral contracts** Ports and Adapters	**02** **New engine + bounded bridge** Anti-Corruption Layer
03	Strangler Fig	04 Branch by Abstraction
05	Expand → Migrate → Contract	06 Versioned contracts
07	Automated migration / codemods	08 Shadow comparison
09	Deprecation usage telemetry	10 Consumer-driven contract tests

Ports and Adapters

Anti-Corruption Layer

All ten · full sources in notes

STABLE CONTRACT. REPLACEABLE BACKEND.



Remember Library A → Library B?

Ports and Adapters

COURAGE IS NOT A RELEASE PROCESS.

01

Name the contract

Consumers · supported behaviour

02

Prove the benefit

A reason worth the migration

03

Stage the migration

Expand · migrate · verify · remove

04

Test real consumers

Results · data · invalid input · ACL

05

Own the bridge

Scope · owner · migration criteria

06

Publish and recover

Guide · known-good build · data plan

A major number is not a migration guide. A code rollback is not a data rollback.

Parallel Change

Consumer contract verification

Nebula · recovery limits

Protect the commitment.
Replace the machinery.
Build the bridge.

Jakub Winkler · qoliber

Which contract is sacred?

15 minutes for questions.

Jakub Winkler · qoliber

WHAT THE NUMBERS PROMISE.

1

MAJOR

Incompatible
public API change

2

MINOR

Compatible functionality
or deprecation

3

PATCH

Compatible
bug fix

Declared public API · stable versions, 1.0.0+ · version zero differs.

Semantic Versioning 2.0.0

A SECURITY LABEL DOES NOT ERASE THE MIGRATION COST.

Magento product numbering ≠ individual module SemVer

An exception can be justified.
Explain what changes and what consumers should test.

Adobe · product release policy

Adobe · module versions and exceptions

THE COST OF FOREVER.

"an anchor that keeps a project stuck in the stone age"

Anthony Ferrara

"Opportunities for innovation are missed."

Lübke et al.

"without having to maintain backwards compatibility with the old, 'wrong' design"

Joel Spolsky

"Only deprecation — and removal — can remove backwards compatibility debt."

chromatic

"very nasty, very expensive technical debt"

chromatic

"this sort of technical debt sticks around forever"

Dan Luu

"almost half of the methods were deprecated"

Maintainer interviewed by Bogart et al.

"Let's not try to live with them forever"

Anthony Ferrara

Different arguments and contexts—not a consensus against compatibility.