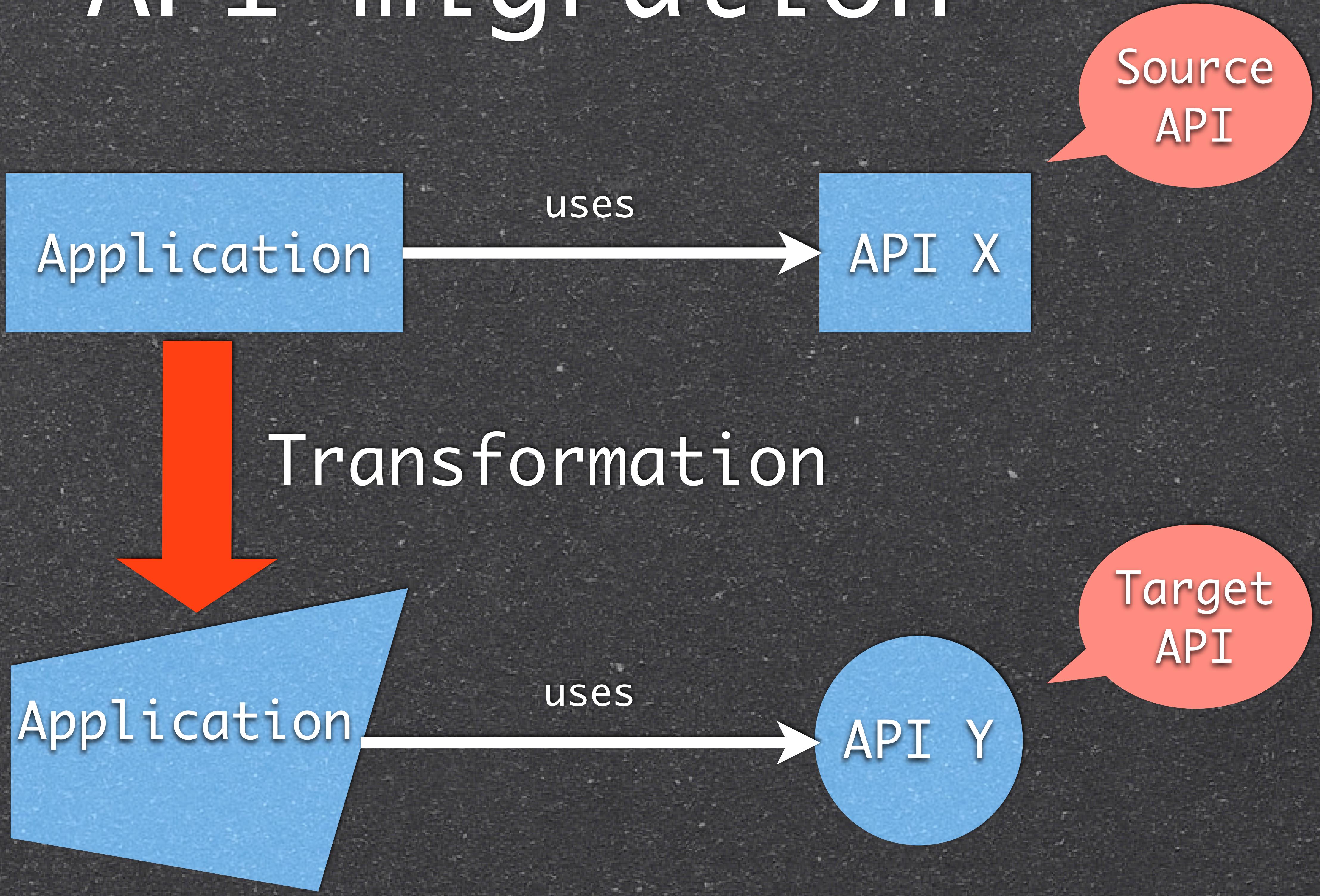
Study of an API Migration for two XML APIs

Thiago Bartholomei
Krzysztof Czarnecki
Ralf Lämmel
Tijs van der Storm

API migration

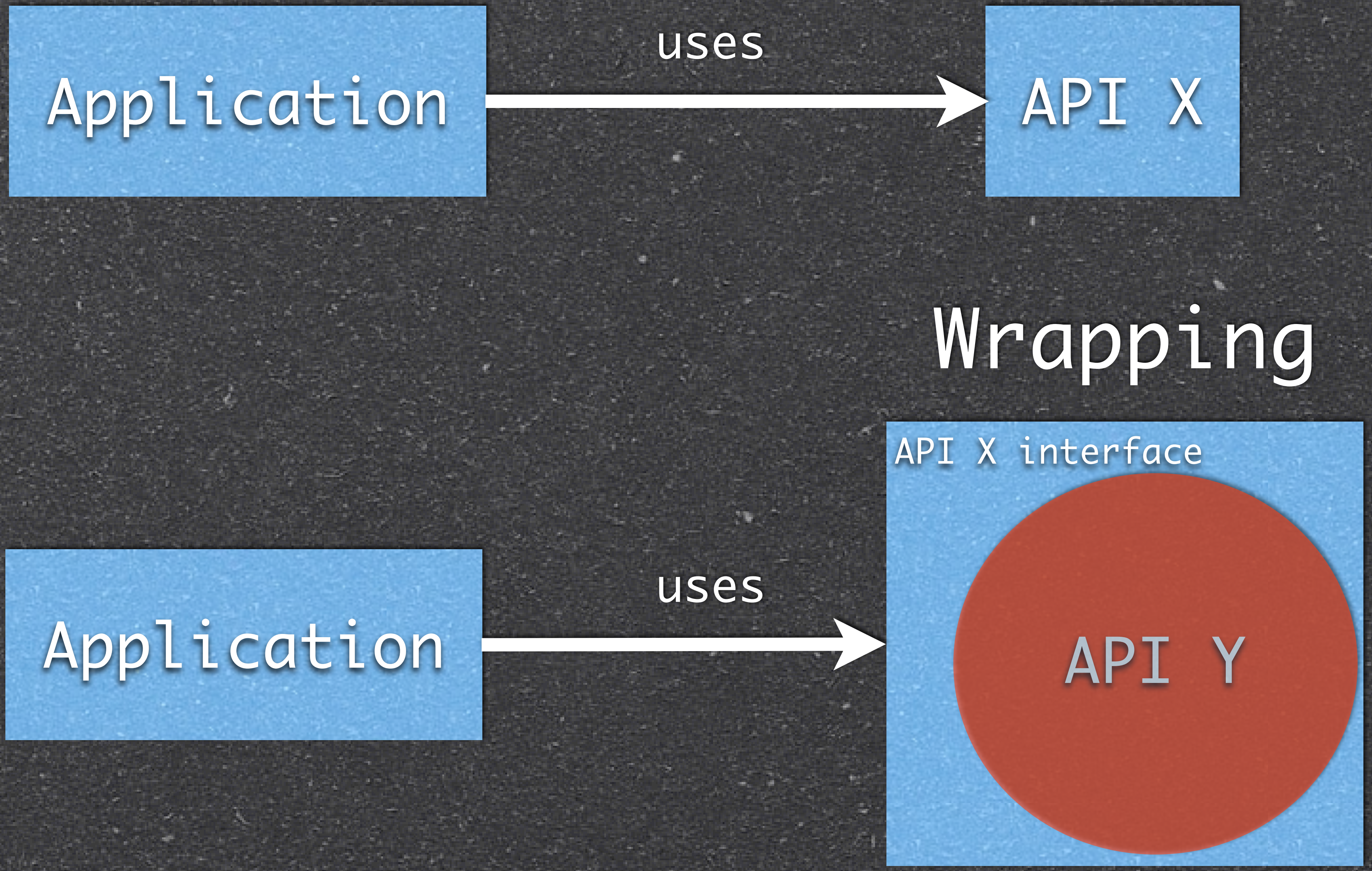


API migration -- why?

- ... because the target API is
  - more modern, reliable, efficient, etc., or
- ... because the source API is
  - no longer supported, or
  - causing worrisome license costs, or
- ... because the app
  - should use less APIs per domain.

API migration

by wrapper-based reimplementation



Plan

- Illustrate difficulty of API migration in the XML domain.
- Describe a wrapper-based migration study between two Java XML APIs: JDOM vs. XOM.
- Observations

API Differences

Construct a phone order <u>with JDOM</u>

```
// JDOM
Element order =
  new Element("order").
  addContent(new Element("product").
     addContent("iPhone")).
  addContent(new Element("customer").
     addContent("1234"));
```

This-returning and method chaining

Construct a phone order with XOM

Void result type

```
// XOM
Element order = new Element("order");
Element product = new Element("product");
product.appendChild("iPhone");
order.appendChild(product);
Element customer = new Element("customer");
customer.appendChild("1234");
order.appendChild(customer);
```

Position- vs. identity- based replacement

```
// JDOM
int index = order.indexOf(oldProduct);
order.setContent(index, newProduct);
```

```
// XOM
order.replaceChild(oldProduct, newProduct);
```

Less vs. more strict pre-conditions

```
// JDOM
order.removeContent(product);
order.removeContent(product);
  // quietly completes.
```

```
// XOM
order.removeChild(product);
order.removeChild(product); // throws!
```

Eager vs. lazy queries

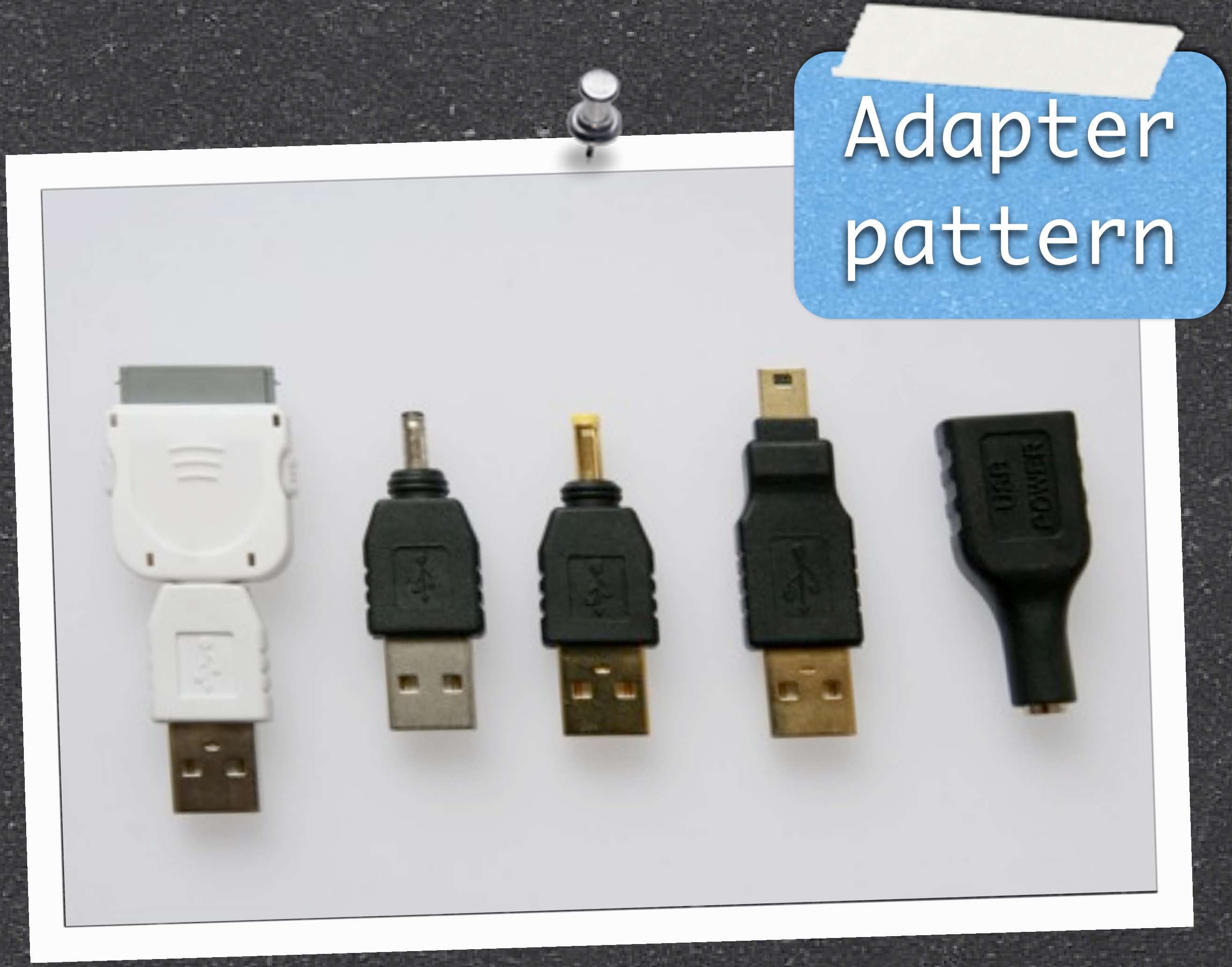
```
// XOM
Elements es = order.getChildElements();
for (int i = 0; i < es.size(); i++)
    es.get(i).detach();
```

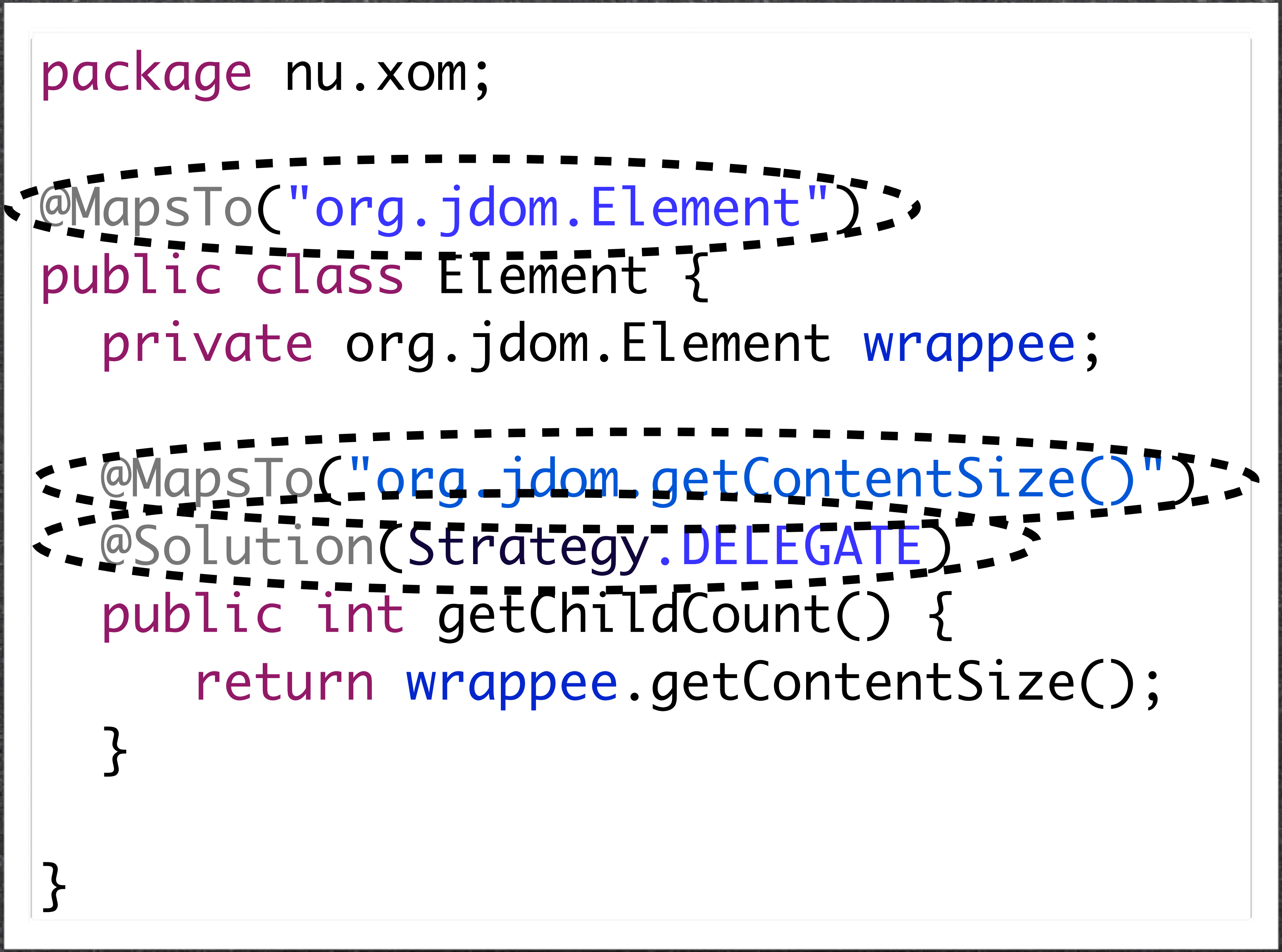
```
// JDOM (illegal, throws exception)
for (Object k : order.getChildren())
     ((Element)k).detach();
```

Wrapper-based migration

Wrapping process

- Start with empty wrapper
- Iteratively implement methods
- Test and diagnose
- Judgment call
- Fix and @annotate





```
package nu.xom;

@MapsTo("org.jdom.Element")
public class Element {
  private org.jdom.Element wrappee;

  @MapsTo("org.jdom.getContentSize()")
  @Solution(Strategy.DELEGATE)
  public int getChildCount() {
     return wrappee.getContentSize();
  }

}
```

“JDOM as XOM”

Annotations

- Progress of implementation
  - e.g., done, todo, wontdo
- Adaptation levels
  - delegate, reimplement
- Generic issues
  - pre, post, invariant
- Domain-specific issues
  - Serialization, Base URI

Observations

Test suite-based compliance

- Estimate compliance using tests
  - API's test suite
    - Comprehensive XOM test suite
  - An application's test suite:
    - Chemistry Development Kit (CDK)

Compliance results

Test suite	Compliant test cases	Non-Comp. test cases	API Method coverage	#Number of test cases
API	417	280	156	697
App	752	0	35	752

Notice

Full compliance for application was reached without manual modifications.

except for 3 test-cases that depended on the order of attributes

Method-compliance levels (relative to a test suite)

- Always

  exercised in successful test cases only
- Sometimes

  also involved in failing test cases
- Never

  only involved in failing testcases
- Unused

  not exercised at all

Method-compliance levels

Test suite	always	some times	never	unused
API	75	77	4	79
App	35	0	0	200

Notice

There are methods with issues w.r.t. the API's test suite that have no issues w.r.t. the application's test suite.

See the paper for detailed numbers on these matters.

Method-adaptation levels

- 1 Pure delegation
- 2 + Pre/postprocessing
- 3 Composite
- 4 Reimplementation

# methods per level				
1	2	3	4	rest
107	66	31	27	4

Judgment call

Aiming for more compliance means higher adaptation levels.

Summary

- Wrapper is good enough for application.
- Full compliance for API’s test suite
  - Not reached, and
  - Not necessary, and
  - Not practical.
- Wrapper is instructive for transformation.
- Study feeds into method for API migration.