"Using Grok to Walk Like a Duck"

The Zope 3 Component Architecture

Brandon Craig Rhodes

Plone Conference 2008, Washington, DC

How many methods does a Plone ATFolder have?

Contributors CreationDate Creator Creators DELETE Date Description EffectiveDate ExpirationDate Format HEAD Identifer Identifier LOCK Language MKCOL MKCOL_handler MOVE
ModificationDate OPTIONS PROPFIND PROPPATCH PUT PUT_factory ZopeFind Publisher Rights SQLConnectionIDs Schema Schemata SearchableText Subject TRACE Title Type UID
UNLOCK Vocabulary ZQueryIds ZopeFind ZopeFindAndApply __url __before_publishing_travers __bobo_traverse__ __browser_default__ __call__ __class_init__ __contains__ __delitem__
__getitem__ __init__ __iter__ __len__ __of__ __repr__ __setitem__ _facade _canCopy _catalogRefs _catalogUID _checkId _ct_defaultAddableTypeIds _ct_defaultConstrainTypesMo
_ct_vocabularyPossibleTypes _datify _delOb _delObject _delPropValue _delProperty _delReferenceAnnotations _facade _deleteOwnershipAfterAdd _editMetadata _effective_date
_expiration_date _filteredItems _findUniqueId _getCatalogTool _getCopy _getOb getPortalTypeName _getReferenceAnnotations _getURL _getWorkflowTool _get_id _has_user_defined_role
_importObjectFromFile _isBeingAccessedAsZClassDef _isBeingUsedAsAMethod _isIDAutoGenerated _isSchemaCurrent _migrateGetValue _migrateSetValue _notifyOfCopyTo _postCopy
_processForm _propertyMap _referenceApply _register _renameAfterCreation _setId _setOb _setObject _setPortalTypeName _setPropValue _setProperty _setRoles _setUID
_subobject_permissions _uncatalogRefs _uncatalogUID _unregister _updateCatalog _updateProperty _updateSchema _verifyObjectPaste _wrapperCheck absolute_url absolute_url_path
ac_inherited_permissions access_debug_info acquiredRolesAreUsedBy addCreator addDTMLDocument addDTMLMethod addReference addSubObjects all_meta_types allowDiscussion
allowedContentTypes at_post_create_script at_post_edit_script autoOrderItems bobobase_modification_time canSetConstrainTypes canSetDefaultPage canSetLayout cb_dataItems
cb_dataValid cb_isCopyable cb_isMoveable cb_userHasCopyOrMovePermiss changeOwnership checkCreationFlag checkIdAvailable class_manage_path cleanupLayers cmf_edit
contentEffective contentExpired contentIds contentItems contentValues copyLayoutFromParent created dav__init dav__simpleifhandler dav__validate decodeFolderFilter defaultIsDiscussable
defaultLanguage defaultRights defaultView deleteReference deleteReferences edit editIsDiscussable editMetadata effective encodeFolderFilter exclude_from_nav expires EditLink
filtered_manage_options filtered_meta_types folderlistingFolderContents generateNewId get getActionInfo getAttribute getAttributeNode getAttributes getAvailableLayouts getBRefs
getBRelationships getBackReferenceImpl getBRefs getCMFObjectsSubsetIds getCatalogs getCharset getChildNodes getConstrainTypesMode getContentType getDefault
getDefaultAddableTypes getDefaultLayout getDefaultPage getDefaultSorting getEffectiveDate getElementsByTagName getExcludeFromNav getExpirationDate getField getFilename
getFirstChild getFolderWhenPortalFactory getIcon getId getImmediatelyAddableTypes getLastChild getLayout getLocallyAllowedTypes getLocation getMetadataHeaders
getNextPreviousEnabled getNextPreviousParentValue getNextSibling getNodeName getNodeType getNodeValue getObjectPosition getOwner getOwnerDocument getOwnerTuple
getParentNode getPhysicalPath on.getPhysicalRoot getPortalTypeName getPreviousSibling getPrimaryField getProperty getPropertyType getRawConstrainTypesMode getRawContributors
getRawCreation_date getRawCreators getRawDescription getRawEffectiveDate getRawExcludeFromNav getRawExpirationDate getRawId getRawImmediatelyAddableTyp getRawLanguage
getRawLocallyAllowedTypes getRawLocation getRawModification_date getRawNextPreviousEnabled getRawRelatedItems getRawRights getRawSubject getRawTitle getReferenceImpl
getReferenceMap getReferencePng getRefs getRefs getRelatedItems getRelationships getSiteManager getSortAuto getSortFolderishFirst getSortReverse getSubObject getTagName getTypeInfo
getWrappedField getWrappedOwner getContentType get_local_roles get_local_roles_for_userid get_portal_metadata get_request_var_or_attr get_size get_valid_userids hasChildNodes
hasObject hasProperty hasRelationshipTo has_local_roles http__etag http__parseMatchList http__processMatchHeaders http__refreshEtag getIcon indexObject initializeArchetype
initializeLayers invokeFactory isBinary isDiscussable isTemporary isTransformable items keys languages listContributors listCreators listDAVObjects listFolderContents list_imports
locked_in_version manage_CopyContainerAllItem manage_CopyContainerFirstIt manage_DAVget manage_FTPget manage_FTPlist manage_FTPstat manage_access _facade
addDTMLDocument addDTMLMethod addDTMLMethod manage_addFile manage_addFolder manage_addImage _facade manage_addOrderedFolder manage_addProperty
manage_addSiteRoot manage_addUserFolder manage_addZGadflyConnection manage_addZGadflyConnection manage_afterAdd manage_afterClone manage_afterMKCOL manage_afterPUT
manage_beforeDelete _facade _facade manage_changeProperties manage_changePropertyTypes manage_clone manage_copyObjects manage_cutObjects manage_defined_roles _facade
manage_delObjects manage_delProperties manage_editMetadata manage_editProperties manage_editRoles manage_editedDialog manage_exportObject manage_fixupOwnershipAfterA
manage_getPermissionMapping manage_hasId manage_importObject manage_pasteObjects _facade manage_renameObject manage_renameObjects _facade _facade _facade _facade
manage_undo_transactions manage_workflowsTab manage_workspace manage_zmi_logout markCreationFlag modified modified_in_version moveObject moveObjectToPosition
moveObjectsByDelta moveObjectsDown moveObjectsToBottom moveObjectsToTop moveObjectsUp notifyModified notifyWorkflowCreated objectIds objectIds_d objectItems objectItems_d
objectMap objectMap_d objectValues objectValues_d opaqueIds opaqueItems opaqueValues orderObjects owner_info permission_settings permissionsOfRole possible_permissions
post_validate pre_validate processForm propdict propertyDescription propertyIds propertyItems propertyLabel propertyMap propertyValues raise_standardErrorMessage rawIsDiscussable
reference_url reindexObject reindexObjectSecurity restrictedTraverse rolesOfPermission setConstrainTypesMode setContentType setContributors setCreationDate setCreators setDefaultPage
setDefaultSorting setDefaults setDescription setEffectiveDate setExcludeFromNav setExpirationDate setFilename setFormat setId setImmediatelyAddableTypes setLanguage setLayout
setLocallyAllowedTypes setLocation setModificationDate setNextPreviousEnabled setRelatedItems setRights setSiteManager setSortAuto setSortFolderishFirst setSortReverse setSubject
setTitle superValues tabs_path_default tabs_path_info this title_and_id title_or_id tpURL tpValues undoable_transactions unindexObject unmarkCreationFlag unrestrictedTraverse update
userCanTakeOwnership userdefined_roles users_with_local_role cb_dataValid valid_roles valid_property_id valid_roles validate validate_field validate_preferredTypes validate_roles values
virtual_url_path widget wl_clearLocks wl_delLock wl_getLock wl_hasLock wl_isLocked wl_lockItems wl_lockTokens wl_lockValues wl_lockmapping wl_setLock

Contributors CreationDate Creator Creators DELETE Date Description EffectiveDate ExpirationDate Format HEAD Identifer Identifier LOCK Language MKCOL MKCOL_handler MOVE ModificationDate OPTIONS PROPFIND PROPPATCH PUT PUT_factory ZopeFind Publisher Rights SQLConnectionIDs Schema Schemata SearchableText Subject TRACE Title Type UID UNLOCK Vocabulary ZQueryIds ZopeFind ZopeFindAndApply __url __before_publishing_travers __bobo_traverse__ __browser_default__ __call__ __class_init__ __contains__ __delitem__ __getitem__ __init__ __iter__ __len__ __of__ __repr__ __setitem__ _facade _canCopy _catalogRefs _catalogUID _checkId _ct_defaultAddableTypeIds _ct_defaultConstrainTypesMo _ct_vocabularyPossibleTypes _datify _delOb _delObject _delPropValue _delProperty _delReferenceAnnotations _facade _deleteOwnershipAfterAdd _editMetadata _effective_date _expiration_date _filteredItems _findUniqueId _getCatalogTool _getCopy _getOb getPortalTypeName _getReferenceAnnotations _getURL _getWorkflowTool _get_id _has_user_defined_role _importObjectFromFile _isBeingAccessedAsZClassDef _isBeingUsedAsAMethod _isIDAutoGenerated _isSchemaCurrent _migrateGetValue _migrateSetValue _notifyOfCopyTo _postCopy _processForm _propertyMap _referenceApply _register _renameAfterCreation _setId _setOb _setObject _setPortalTypeName _setPropValue _setProperty _setRoles _setUID _subobject_permissions _uncatalogRefs _uncatalogUID _unregister _updateCatalog _updateProperty _updateSchema _verifyObjectPaste _wrapperCheck absolute_url absolute_url_path ac_inherited_permissions access_debug_info acquiredRolesAreUsedBy addCreator addDTMLDocument addDTMLMethod addReference addSubObjects all_meta_types allowDiscussion allowedContentTypes at_post_create_script at_post_edit_script autoOrderItems bobobase_modification_time canSetConstrainTypes canSetDefaultPage canSetLayout cb_dataItems cb_dataValid cb_isCopyable cb_isMoveable cb_userHasCopyOrMovePermiss changeOwnership checkCreationFlag checkIdAvailable class_manage_path cleanupLayers cmf_edit contentEffective contentExpired contentIds contentItems contentValues copyLayoutFromParent created dav__init dav__simpleifhandler dav__validate decodeFolderFilter defaultIsDiscussable defaultLanguage defaultRights defaultView deleteReferenc deleteReferences edit editIsDiscussable editMetadata effective encodeFolderFilter exclude_from_nav expires EditLink filtered_manage_options filtered_meta_types folderlisting rContents ge getAttribute eNode getAttributes getAvailableLayouts getBRefs getBRelationships getBackReferenceImpl getBR MFObjectsSub getCatalogs getCh getChild getC rainTypesMode getContentType getDefault getDefaultAddableTypes getDefaultLayout getDefau DefaultSorting EffectiveDate getEl ntsByTagName getEx FromNav getExpirationDate getField getFilename getFirstChild getFolderWhenPortalFactory g getImmediately AddableTypes getLa ild getLayout getLocall wedTypes getLocation getMetadataHeaders getNextPreviousEnabled getNextPreviousPare ue get Sibling getNodeName getNodeTy etNodeValue getObjectPo n getOwner getOwnerDocument getOwnerTuple getParentNode getPhysicalPath on.getPhysicalRo etPortalT Name getPreviousSibling getPrim ield getProperty getPrope pe getRawConstrainTypesMode getRawContributors getRawCreation_date getRawCreators getRawD ption get ffectiveDate getRawExcludeFr av getRawExpirationDate g wId getRawImmediatelyAddableTyp getRawLanguage getRawLocallyAllowedTypes getRawLocati etRawModif n_date getRawNextPreviousE ed getRawRelatedItems getR ghts getRawSubject getRawTitle getReferenceImpl getReferenceMap getReferencePng getRefs fs getRelatedI getRelationships getSiteMa getSortAuto getSortFolderish getSortReverse getSubObject getTagName getTypeInfo getWrappedField getWrappedOwner g get_local_roles_for_u get_portal_metadata get_reque r_or_attr get_size get_valid_userids hasChildNodes hasObject hasProperty hasRelations http__parseMatch http__processMatchHeaders http eshEtag getIcon indexObject initializeArchetype initializeLayers invokeFactory isBinary isDiscussable isTe ary isTransformable ite ys languages listContributors listC rs listDAVObjects listFolderContents list_imports locked_in_version manage_CopyContainerAllItem ma CopyContainerFirstIt ge_DAVget manage_FTPget manag Plist manage_FTPstat manage_access _facade addDTMLDocument addDTMLMethod addDTMLMeth manage_addFile mana ddFolder manage_addImage _facade ge_addOrderedFolder manage_addProperty manage_addSiteRoot manage_addUserFolder manage_addZGa Connection manage ZGadflyConnection manage_afterAdd afterClone manage_afterMKCOL manage_afterPUT manage_beforeDelete _facade _facade manage_changePrope manage_changeP rtyTypes manage_clone manage manage_cutObjects manage_defined_roles _facade manage_delObjects manage_delProperties manage_editMeta manage_editPro s manage_editRoles manage_editedDialog manage_exportObject manage_fixupOwnershipAfterA manage_getPermissionMapping manage_hasId manage_importObject manage_pasteObjects _facade manage_renameObject manage_renameObjects _facade _facade _facade _facade manage_undo_transactions manage_workflowsTab manage_workspace manage_zmi_logout markCreationFlag modified modified_in_version moveObject moveObjectToPosition moveObjectsByDelta moveObjectsDown moveObjectsToBottom moveObjectsToTop moveObjectsUp notifyModified notifyWorkflowCreated objectIds objectIds_d objectItems objectItems_d objectMap objectMap_d objectValues objectValues_d opaqueIds opaqueItems opaqueValues orderObjects owner_info permission_settings permissionsOfRole possible_permissions post_validate pre_validate processForm propdict propertyDescription propertyIds propertyItems propertyLabel propertyMap propertyValues raise_standardErrorMessage rawIsDiscussable reference_url reindexObject reindexObjectSecurity restrictedTraverse rolesOfPermission setConstrainTypesMode setContentType setContributors setCreationDate setCreators setDefaultPage setDefaultSorting setDefaults setDescription setEffectiveDate setExcludeFromNav setExpirationDate setFilename setFormat setId setImmediatelyAddableTypes setLanguage setLayout setLocallyAllowedTypes setLocation setModificationDate setNextPreviousEnabled setRelatedItems setRights setSiteManager setSortAuto setSortFolderishFirst setSortReverse setSubject setTitle superValues tabs_path_default tabs_path_info this title_and_id title_or_id tpURL tpValues undoable_transactions unindexObject unmarkCreationFlag unrestrictedTraverse update userCanTakeOwnership userdefined_roles users_with_local_role cb_dataValid valid_roles valid_property_id valid_roles validate validate_field validate_preferredTypes validate_roles values virtual_url_path widget wl_clearLocks wl_delLock wl_getLock wl_hasLock wl_isLocked wl_lockItems wl_lockTokens wl_lockValues wl_lockmapping wl_setLock

471

Both Zope 2 and Plone solve problems by piling more and more methods on an object

Zope 3 does something different

Zope 3 uses Adapters

What are adapters?

Let's talk programming.

Many programming languages use *static* typing

```
float half(int n)
{
    return n / 2.0;
}
```

```
float half(int n)
{
    return n / 2.0;
}
```

Python typing is *dynamic*

```
def half(n):
    return n / 2.0
```

You don't worry about whether an object is of the right type

You simply try using it

“Duck Typing”

(Alex Martelli)

“Duck Typing”

Walks like a duck?
Quacks like a duck?
It's a duck!

```
def half(n):
    return n / 2.0
```

```
def half(n):
    return n / 2.0
```

(Is *n* willing to be divided by two?
Then it's number-ish enough for us!)

Now, imagine…

Imagine a wonderful duck-processing library to which you want to pass an object

But…

The object you want to pass *isn't* a duck?

What if it *doesn't* already quack?

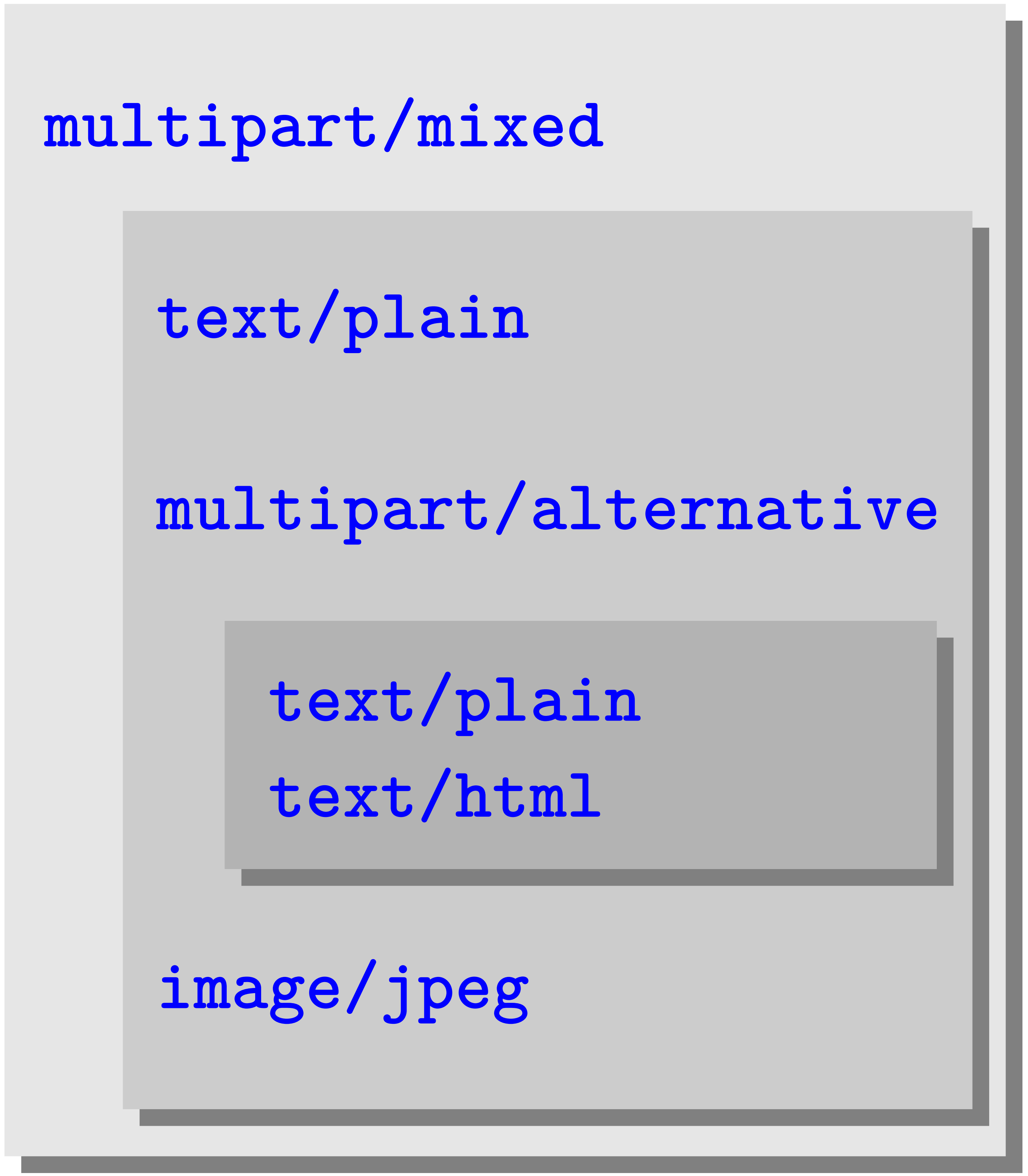
What if it bears
not the least resemblance
to a duck!?

Example!

You have a “Message” object from the Python “email” module

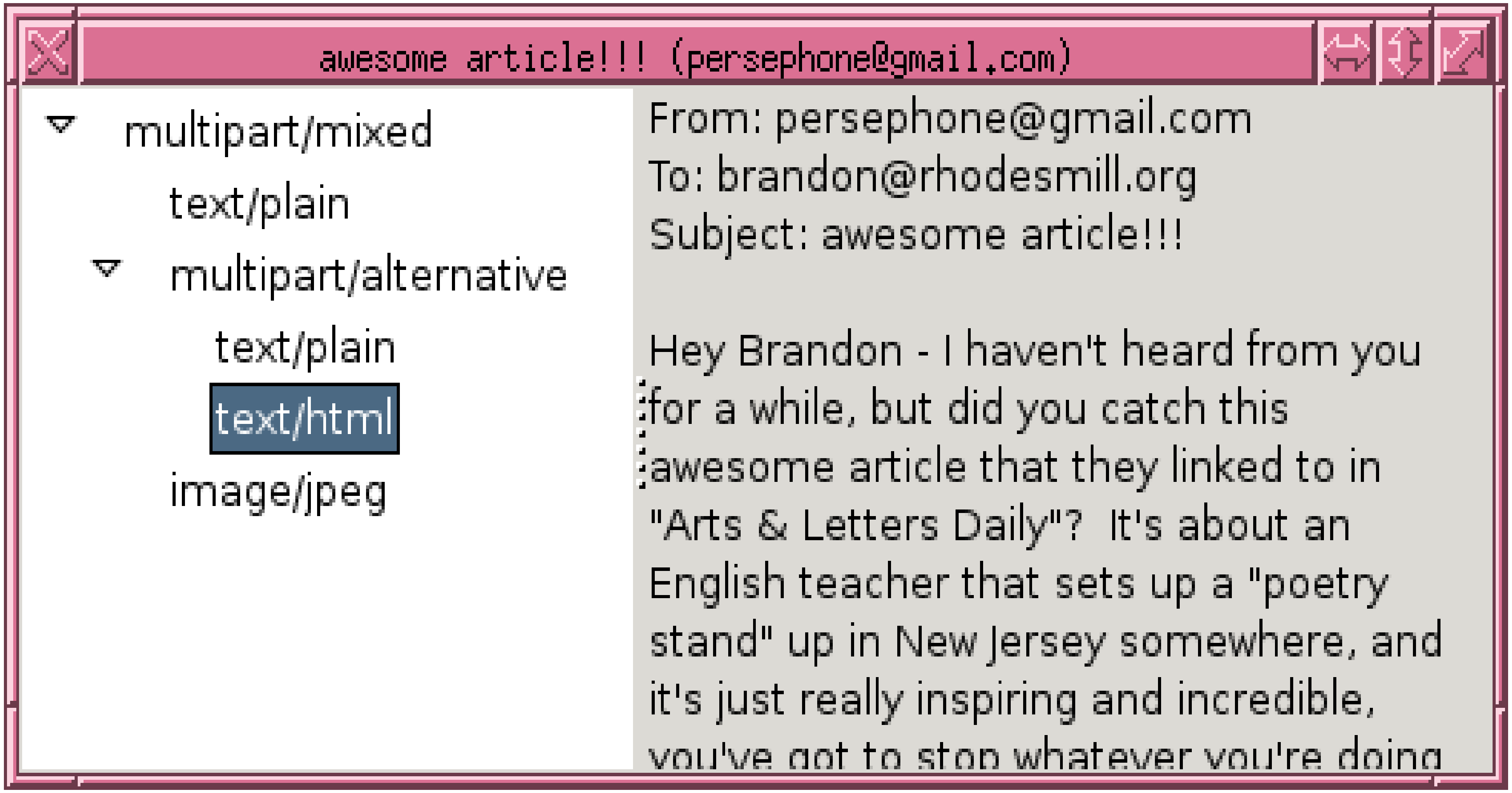
```
>>> from email import message_from_file
>>> e = message_from_file(open('msg.txt'))
>>> print e
<email.message.Message instance at ...>
>>> e.is_multipart()
True
>>> for part in e.get_payload():
...     print part.get_content_type()
text/plain
text/html
```

Messages can be recursive



Imagine that we are writing a Plone email browsing system

And we want to show the parts in a TreeWidget



```
▽ multipart/mixed
    text/plain
  ▽ multipart/alternative
      text/plain
      text/html
    image/jpeg
```

From: persephone@gmail.com
To: brandon@rhodesmill.org
Subject: awesome article!!!

Hey Brandon - I haven't heard from you for a while, but did you catch this awesome article that they linked to in "Arts & Letters Daily"? It's about an English teacher that sets up a "poetry stand" up in New Jersey somewhere, and it's just really inspiring and incredible, you've got to stop whatever you're doing

- multipart/mixed

- ▼ multipart/mixed
  - text/plain
  - ▶ multipart/alternative
  - image/jpeg

- ▼ multipart/mixed
  - text/plain
  - ▼ multipart/alternative
    - text/plain
    - text/html
  - image/jpeg

The Tree widget operates on any object with:

```
method name() - returns name under which
    this tree node should be displayed

method children() - returns list of child
    nodes in the tree

method __len__() - returns number of child
    nodes beneath this one
```

How can we add these behaviors to our Message?

(How can we make an object which is *not* a duck behave like a duck?)

1. Subclassing

Create a “TreeMessage” class that inherits from the “Message” class…

```
class TreeMessage(Message):

  def name(self):
    return self.get_content_type()

  def children(self):
    if not self.is_multipart(): return []
    return [ TreeMessage(part) for part
             in self.get_payload() ]

  def __len__(self):
    return len(self.children())
```

What will the test suite look like?

Remember:

“Untested code
is broken code”

— Philipp von Weitershausen,
Martin Aspeli

Your test suite
must instantiate a
“TreeMessage” and verify
its tree-like behavior…

```
txt = """From: persephone@gmail.com
To: brandon@rhodesmill.org
Subject: what an article!

Did you read Arts & Letters Daily today?
"""
m = message_from_string(txt, TreeMessage)
assert m.name() == 'text/plain'
assert m.children() == []
assert m.__len__() == 0
```

We were lucky!

Our test can cheaply instantiate Messages.

```
txt = """From: persephone@gmail.com
To: brandon@rhodesmill.org
Subject: what an article!

Did you read Arts & Letters Daily today?
"""
m = message_from_string(txt, TreeMessage)
assert m.name() == 'text/plain'
assert m.children() == []
assert m.__len__() == 0
```

What if we were subclassing an LDAP connector?!

We'd need an LDAP server just to run unit tests!

We were lucky (#2)!

The “message_from_string()” method let us specify an alternate factory!

```
txt = """From: persephone@gmail.com
To: brandon@rhodesmill.org
Subject: what an article!

Did you read Arts & Letters Daily today?
"""
m = message_from_string(txt, TreeMessage)
assert m.name() == 'text/plain'
assert m.children() == []
assert m.__len__() == 0
```

Final note: *we have just broken the “Message” class's behavior!*

Python library manual 7.10.1 defines “Message”:

`__len__():`

Return the total number of headers, including duplicates.

```
>>> t = """From: persephone@gmail.com
To: brandon@rhodesmill.org
Subject: what an article!

Did you read Arts & Letters Daily today?
"""
>>> m = message_from_file(t, Message)
>>> print len(m)
3
>>> m = message_from_file(t, TreeMessage)
>>> print len(m)
0
```

So how does subclassing score?

- ✓ No harm to base class

- ✓ No harm to base class
- ● Cannot test in isolation

- ✓ No harm to base class
- ● Cannot test in isolation
- ● Need control of factory

- ✓ No harm to base class
- ● Cannot test in isolation
- ● Need control of factory
- ● Breaks if names collide

- ✓ No harm to base class
- ● Cannot test in isolation
- ● Need control of factory
- ● Breaks if names collide

Subclassing: D

2. Using a mixin

Create a “TreeMessage” class that inherits from both “Message” and a “Mixin”…

```
class Mixin(object):
  def name(self):
    return self.get_content_type()
  def children(self):
    if not self.is_multipart(): return []
    return [ self.__class__(part) for part
                  in self.get_payload() ]
  def __len__(self):
    return len(self.children())

class TreeMessage(Message, Mixin): pass
```

Your test suite can then inherit from a fake, mocked-up “message”…

```
class FakeMessage(Mixin):
  def get_content_type(self):
    return 'text/plain'
  def is_multipart(self): return False
  def get_payload(self): return ''

m = FakeMessage()

assert m.name() == 'text/plain'
assert m.children() == []
assert m.__len__() == 0
```

How does
a mixin rate?

✓ No harm to base class

- ✓ No harm to base class
- ✓ Can test mixin by itself

- ✓ No harm to base class
- ✓ Can test mixin by itself
- ● Need control of factory

- ✓ No harm to base class
- ✓ Can test mixin by itself
- ● Need control of factory
- ● Breaks if names collide

- ✓ No harm to base class
- ✓ Can test mixin by itself
- ● Need control of factory
- ● Breaks if names collide

Mixin: C

3. Monkey patching

To “monkey patch” a class, you add or change its methods dynamically…

```
def name(self):
  return self.get_content_type()
def children(self):
  if not self.is_multipart(): return []
  return [ Message(part) for part
              in self.get_payload() ]
def __len__(self):
  return len(self.children())

Message.name = name
Message.children = children
Message.__len__ = __len__
```

Is this desirable?

✓ Don't care about factory

- ✓ Don't care about factory
- ● Changes class itself

- ✓ Don't care about factory
- ● Changes class itself
- ● Broken by collisions

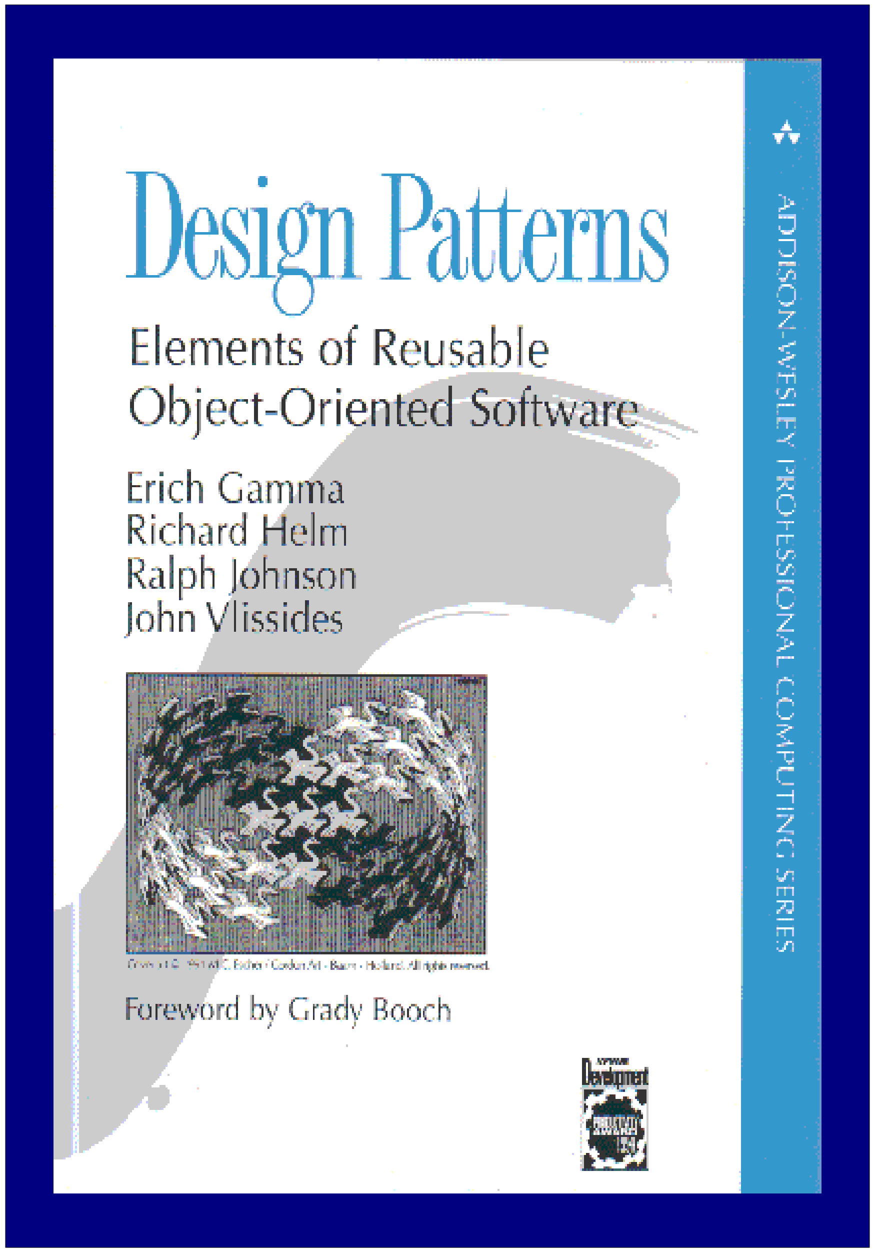
- ✓ Don't care about factory
- ● Changes class itself
- ● Broken by collisions
- ● Patches fight each other

- ✓ Don't care about factory
- ● Changes class itself
- ● Broken by collisions
- ● Patches fight each other
- ● Ruby people do this

- ✓ Don't care about factory
- ● Changes class itself
- ● Broken by collisions
- ● Patches fight each other
- ● Ruby people do this

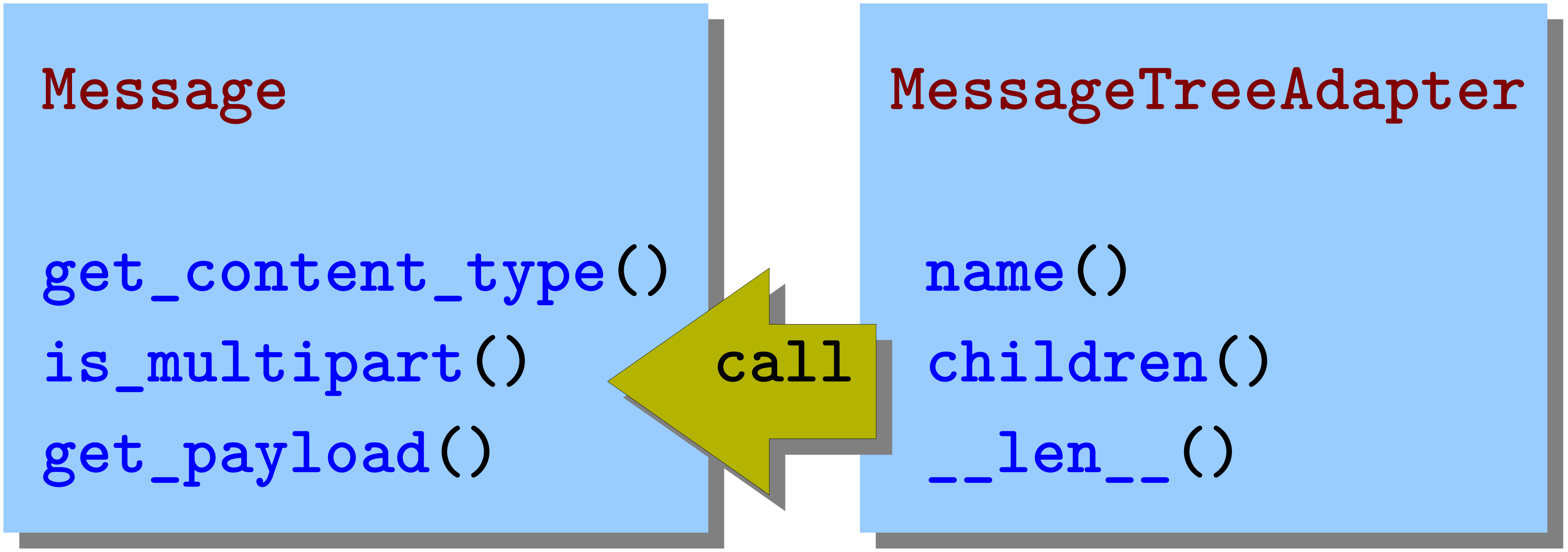
Monkey patching: F

4. Adapter


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Touted in the Gang of Four book (1994)

Idea: provide “Tree” functions through an entirely separate object



```python
class MessageTreeAdapter(object):
  def __init__(self, message):
    self.m = message
  def name(self):
    return self.m.get_content_type()
  def children(self):
    if not self.m.is_multipart(): return []
    return [ MessageTreeAdapter(part)
      for part in self.m.get_payload() ]
  def __len__(self):
    return len(self.children())
```

How does wrapping look in your code?

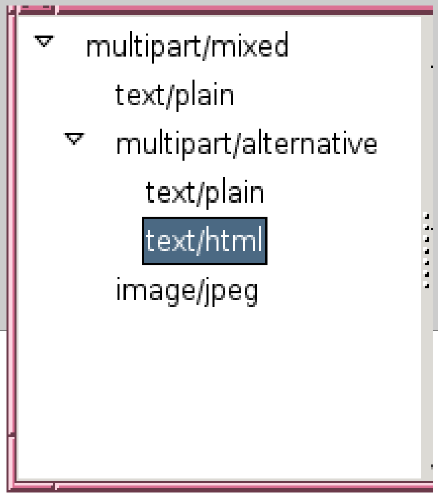
```
IMAP library (or whatever)
returns a Message "msg"
```

Message object

```
tw = TreeWidget(MessageTreeAdapter(msg))
```

Adapted object

```
TreeWidget
```



```
▽ multipart/mixed
    text/plain
  ▽ multipart/alternative
      text/plain
      text/html
    image/jpeg
```

Test suite can try adapting a mock-up object

```
class FakeMessage(object):
  def get_content_type(self):
    return 'text/plain'
  def is_multipart(self): return True
  def get_payload(self): return []

m = MessageTreeAdapter(FakeMessage())
assert m.name() == 'text/plain'
assert m.children() == []
assert m.__len__() == 0
```

How does the Adapter design pattern stack up?

✓ No harm to base class

- ✓ No harm to base class
- ✓ Can test with mock-up

- ✓ No harm to base class
- ✓ Can test with mock-up
- ✓ Don't need factories

- ✓ No harm to base class
- ✓ Can test with mock-up
- ✓ Don't need factories
- ✓ No collision worries

- ✓ No harm to base class
- ✓ Can test with mock-up
- ✓ Don't need factories
- ✓ No collision worries
- ● Wrapping is annoying

✓ No harm to base class

✓ Can test with mock-up

✓ Don't need factories

✓ No collision worries

● Wrapping is annoying

Adapter: B

Q: Why call wrapping "annoying"?

The example makes
it look so easy!

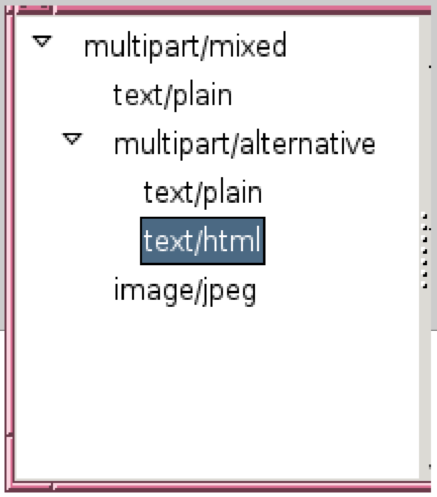
```
IMAP library (or whatever)
returns a Message "msg"
```

Message object

```
tw = TreeWidget(TreeMessageAdapter(msg))
```

Adapted object

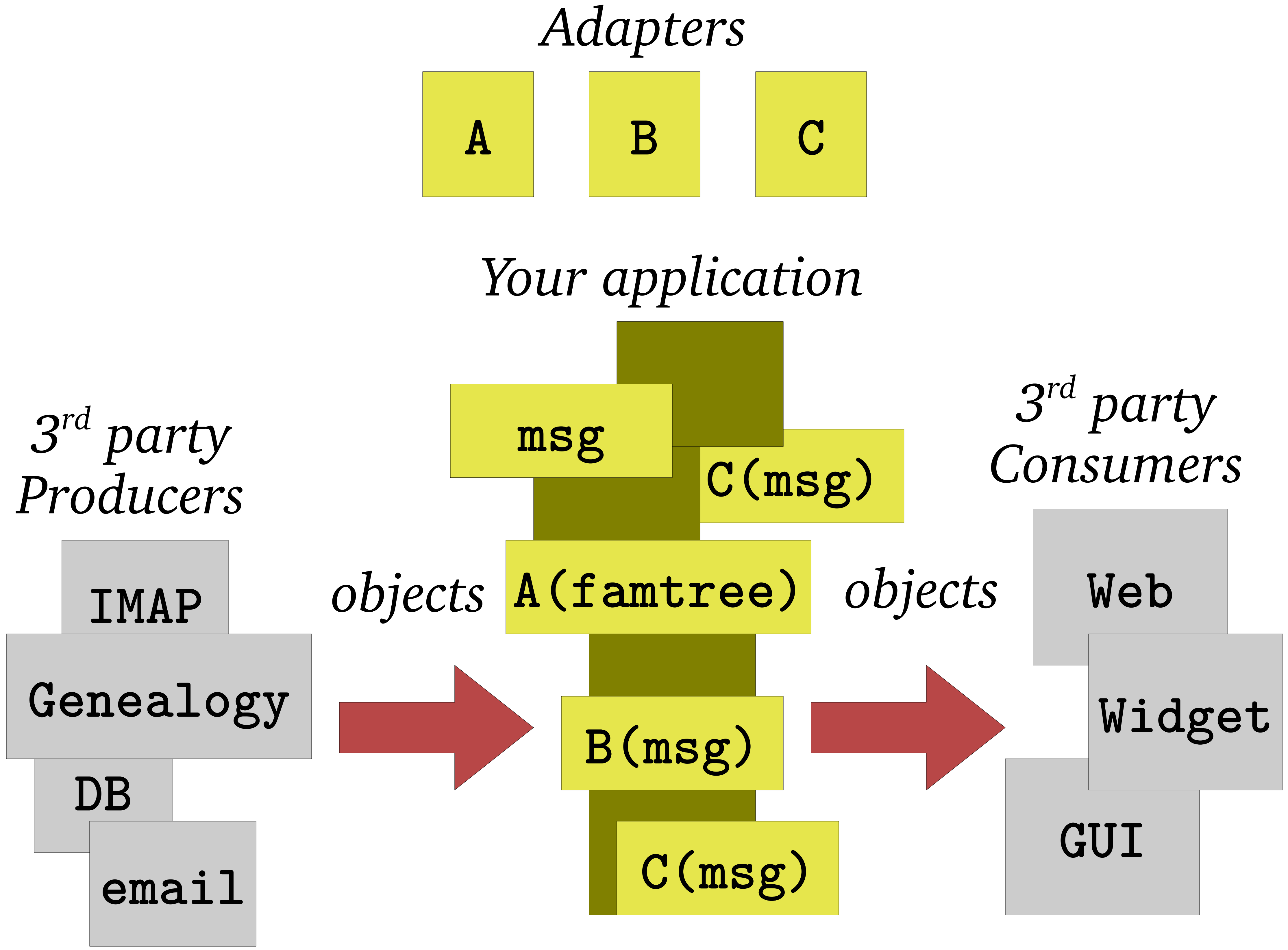
```
TreeWidget
```



```
▽ multipart/mixed
    text/plain
  ▽ multipart/alternative
      text/plain
      text/html
    image/jpeg
```

A: The example looks easy because it only does adaptation *once*!

But in a real application,
it happens all through
your code…


Adapters
A
B
C
Your application
msg
C(msg)
A(famtree)
B(msg)
C(msg)
3rd party Producers
IMAP
Genealogy
DB
email
objects
objects
3rd party Consumers
Web
Widget
GUI

This makes you repeat yourself.

This *also* locks you in to using that particular adapter, since you use it by name in your code.

How can you avoid repeating yourself, and scattering information about adapters and consumers everywhere?

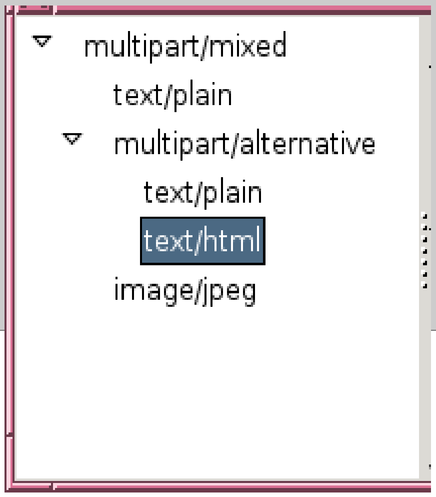
IMAP library (or whatever)
returns a Message "msg"

Message object

```
tw = TreeWidget(TreeMessageAdapter(msg))
```

Adapted object

TreeWidget



```
▽ multipart/mixed
    text/plain
  ▽ multipart/alternative
      text/plain
      text/html
    image/jpeg
```

```
tw = TreeWidget(TreeMessageAdapter(msg))
tw = TreeWidget(TreeMessageAdapter(msg))
```

The key is seeing that this code conflates *two* issues!

```
tw = TreeWidget(TreeMessageAdapter(msg))
```

Why does this line work?

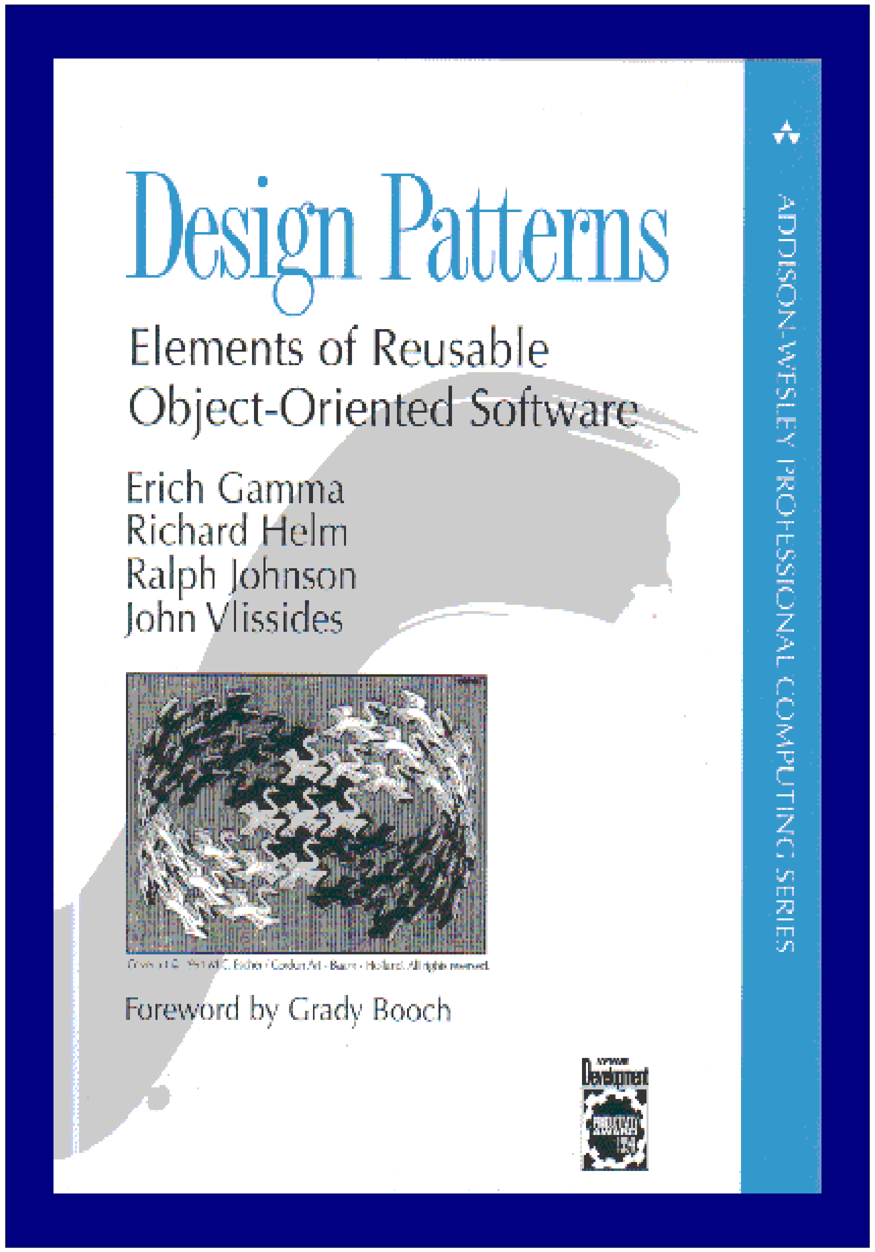
```
tw = TreeWidget(TreeMessageAdapter(msg))
```

It works because a TreeWidget *needs* what our adapter *provides*.

```
tw = TreeWidget(TreeMessageAdapter(msg))
```

But if *we* call the adapter then the **need** = **want** is *hidden inside of our head!*

We need to define what the TreeWidget needs that our adapter provides!


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

An *interface* is how we specify a set of behaviors

Modula 3 (1988)

Java (1995)

An *interface* is how we specify a set of behaviors

ZOPE®

For the moment, forget Zope-the-web-framework

Instead, look at Zope the Component Framework:

zope.interface
zope.component

With three simple steps, Zope will put adapters around classes *for you* —

and rid your code of manual adaptation!

1. *Define* an interface
2. *Register* our adapter
3. *Request* adaptation

Define

```
from zope.interface import Interface

class ITree(Interface):
  def name():
    """Return this tree node's name."""
  def children():
    """Return this node's children."""
  def __len__():
    """Return how many children."""
```

Register

```
from zope.component import provideAdapter

provideAdapter(MessageTreeAdapter,
               adapts=Message,
               provides=ITree)
```

Request

```
class TreeWidget(...):
  def __init__(self, arg):
    tree = ITree(arg)
    ...
```

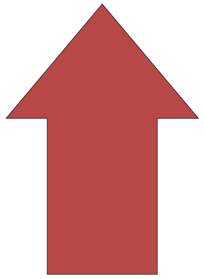
Request

```
class TreeWidget(...):
  def __init__(self, arg):
    tree = ITree(arg)
    ...
```

```
Zope will: 1. Recognize need
           2. Find the registered adapter
           3. Wrap and return the argument
```

Request

```
class TreeWidget(...):
  def __init__(self, arg):
    tree = ITree(arg)
    ...
```



(Look! Zope is Pythonic!)

```
i = int(32.1)
l = list('abc')
f = float(1024)
```

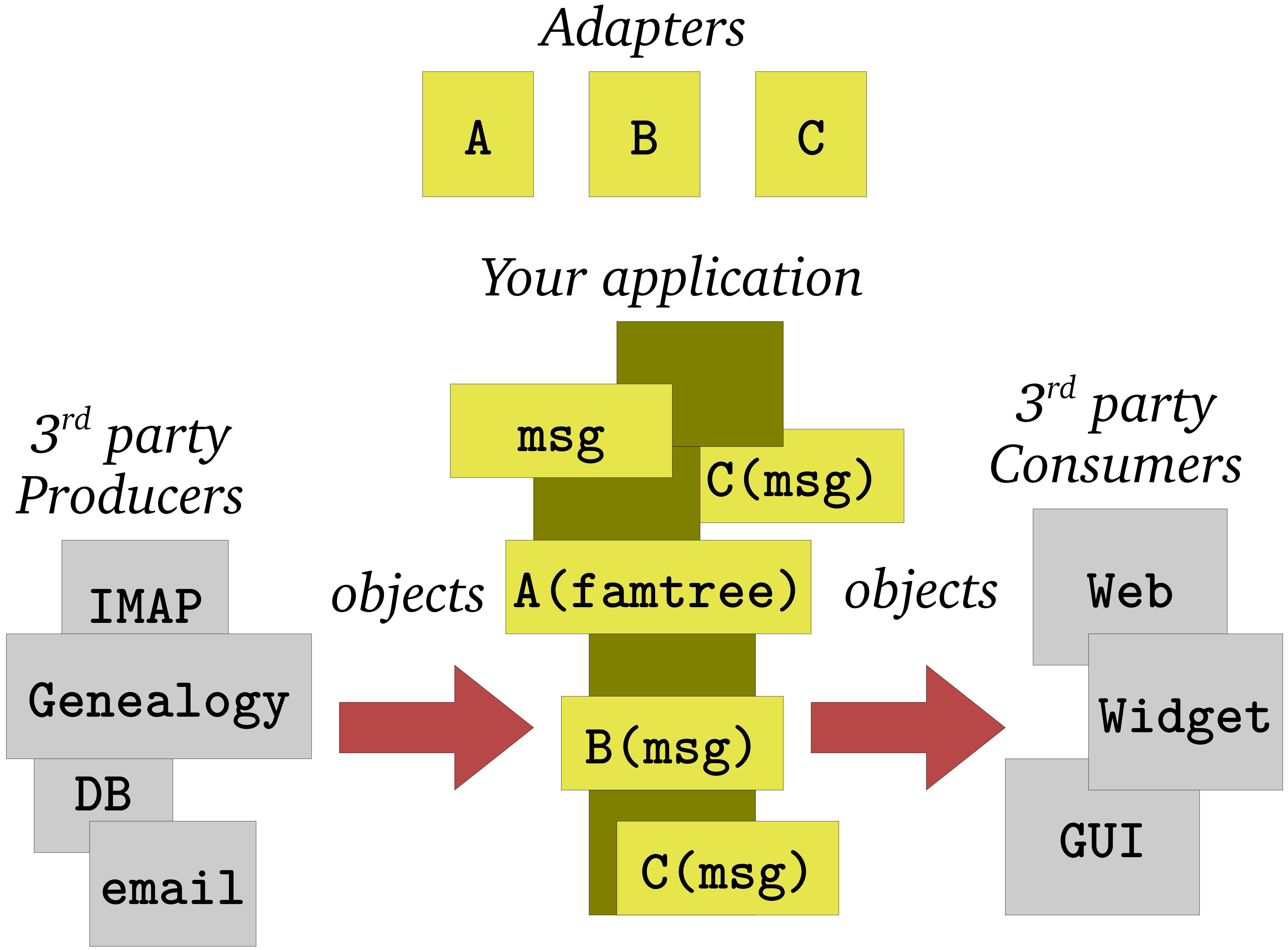
And that's it!

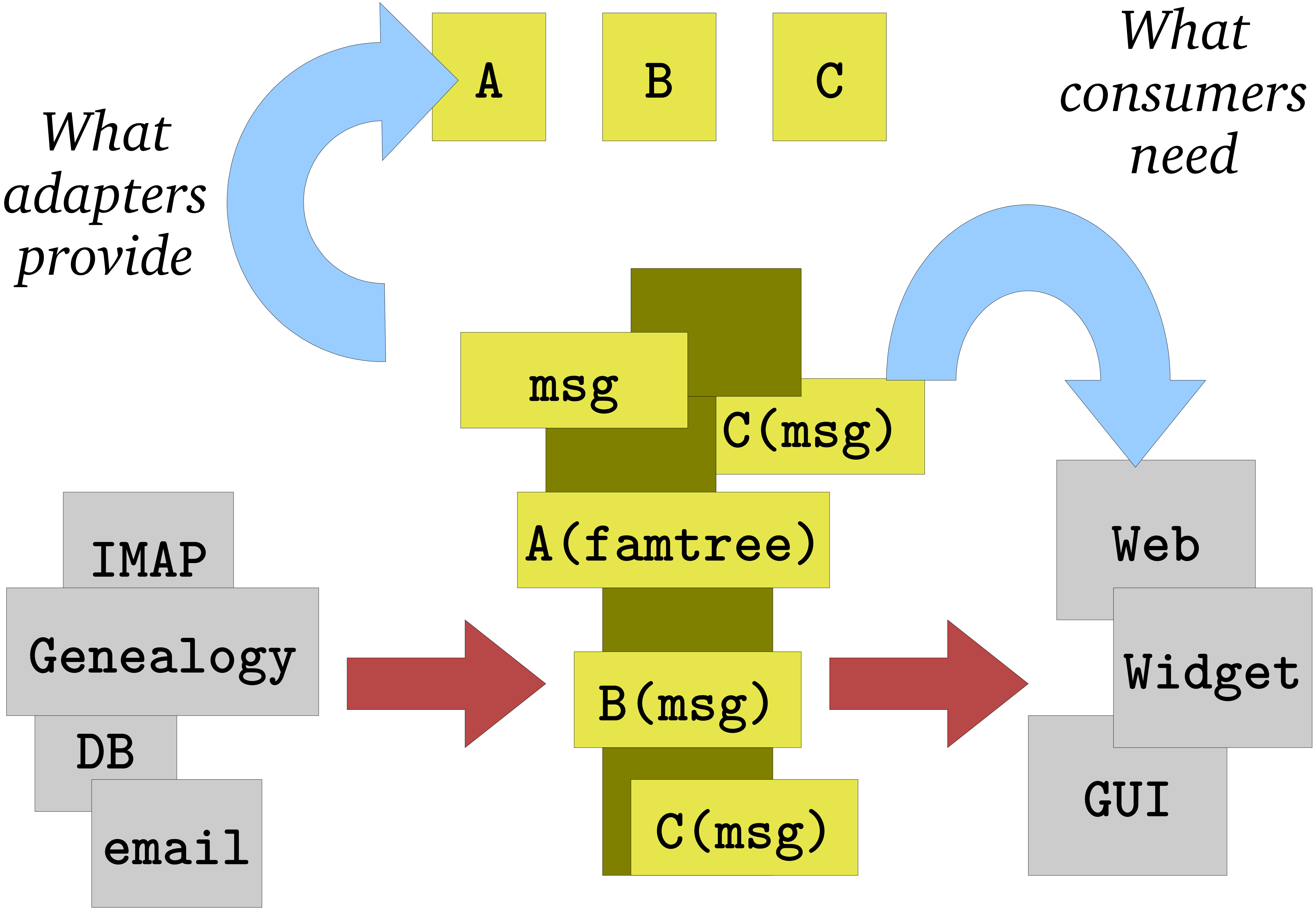
And that's it!

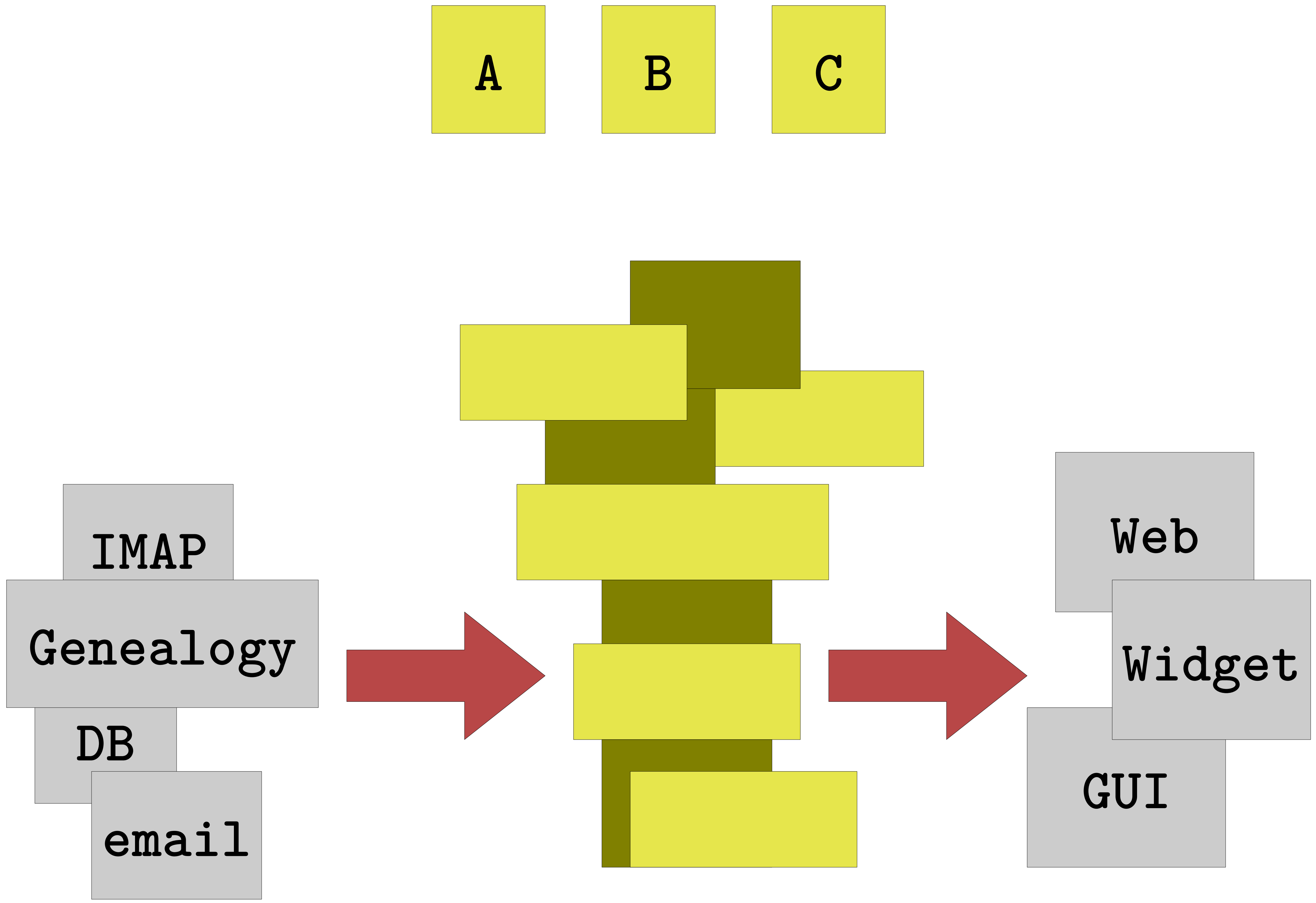
Define an interface
Register our adapter
Request adaptation

- ✓ No harm to base class
- ✓ Can test with mock-up
- ✓ Don't need factories
- ✓ No collision worries
- ✓ Adapters now dynamic!

Registered adapter: A


Adapters
A
B
C
Your application
msg
C(msg)
A(famtree)
B(msg)
C(msg)
3rd party Producers
IMAP
Genealogy
DB
email
objects
objects
3rd party Consumers
Web
Widget
GUI


What adapters provide
A
B
C
What consumers need
msg
C(msg)
A(famtree)
B(msg)
C(msg)
IMAP
Genealogy
DB
email
Web
Widget
GUI


A
B
C
IMAP
Genealogy
DB
email
Web
Widget
GUI

The finale

Adapting for the Web

dum … dum … dum …

DAH DUM!

Grok

Web framework
built atop Zope 3
component architecture

Grok makes
Zope 3 simple to use
(and to present!)

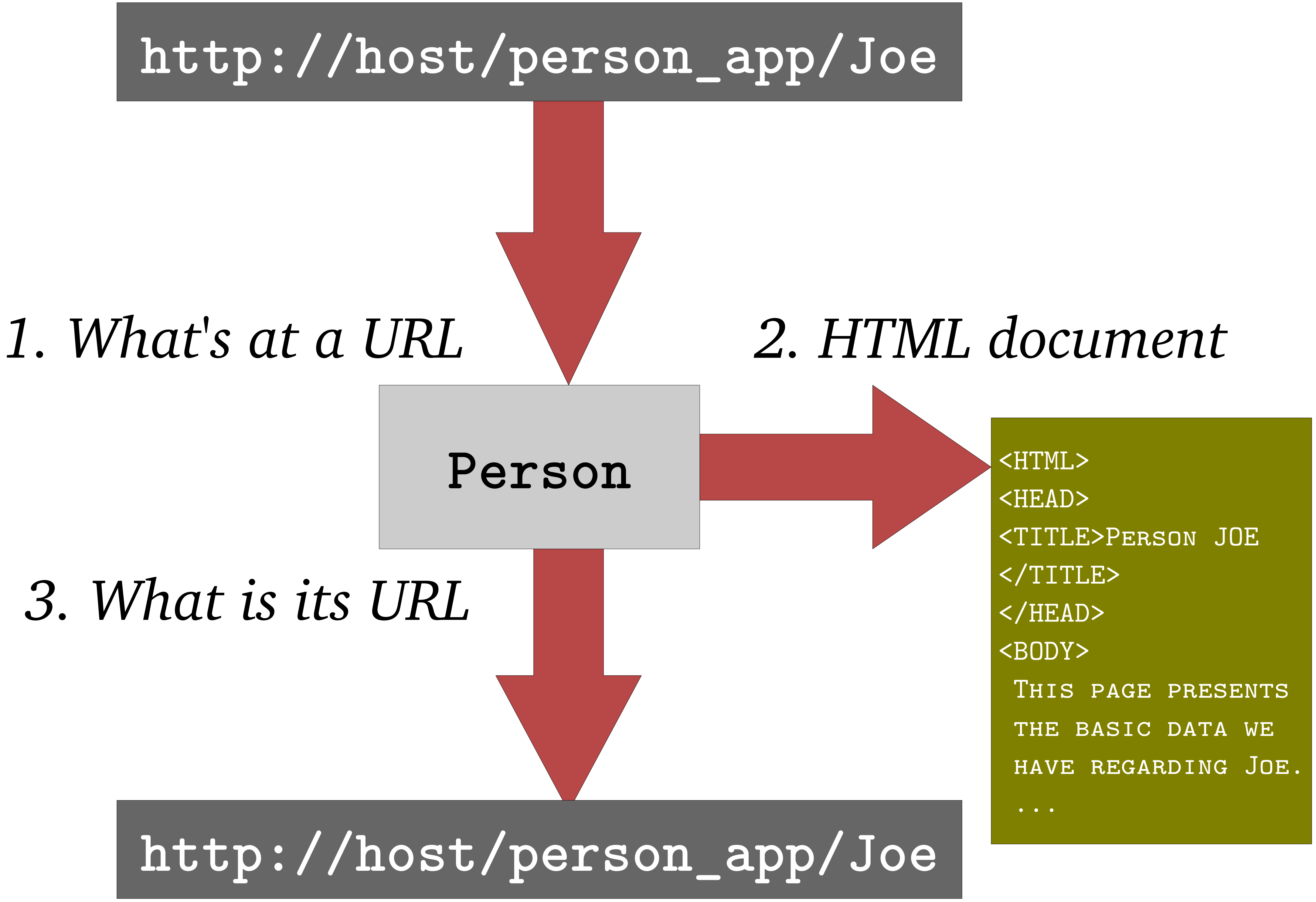
Imagine a Person class

The Person class was written by someone else

The Person class is full of business logic, and stores instances in a database

We want to browse
Person objects on the Web

What might the Web need the object to do?



```
<HTML>
<HEAD>
<TITLE>Person JOE
</TITLE>
</HEAD>
<BODY>
 This page presents
 the basic data we
 have regarding Joe.
 ...
```

1.

What's at this URL?

What's at this URL?

```
http://host/person_app/Joe
```

```
# how Zope processes this URL:
r = root
j = ITraverser(r).traverse('person_app')
k = ITraverser(j).traverse('Joe')
return k
```

What's at this URL?

```
http://host/person_app/Joe
```

```
# what we write:
class PersonTraverser(grok.Traverser):
  grok.context(PersonApp)
  def traverse(self, name):
    if person_exists(name):
      return get_person(name)
    return None
```

2.

How does a Person render?

How does a Person render?

app.py

```
class PersonIndex(grok.View):
   grok.context(Person)
   grok.name('index')
```

app_templates/personindex.pt

```
<html><head><title>All about
 <tal tal:replace="context/name"/>
</title></head>...
```

3.

What is a person's URL?

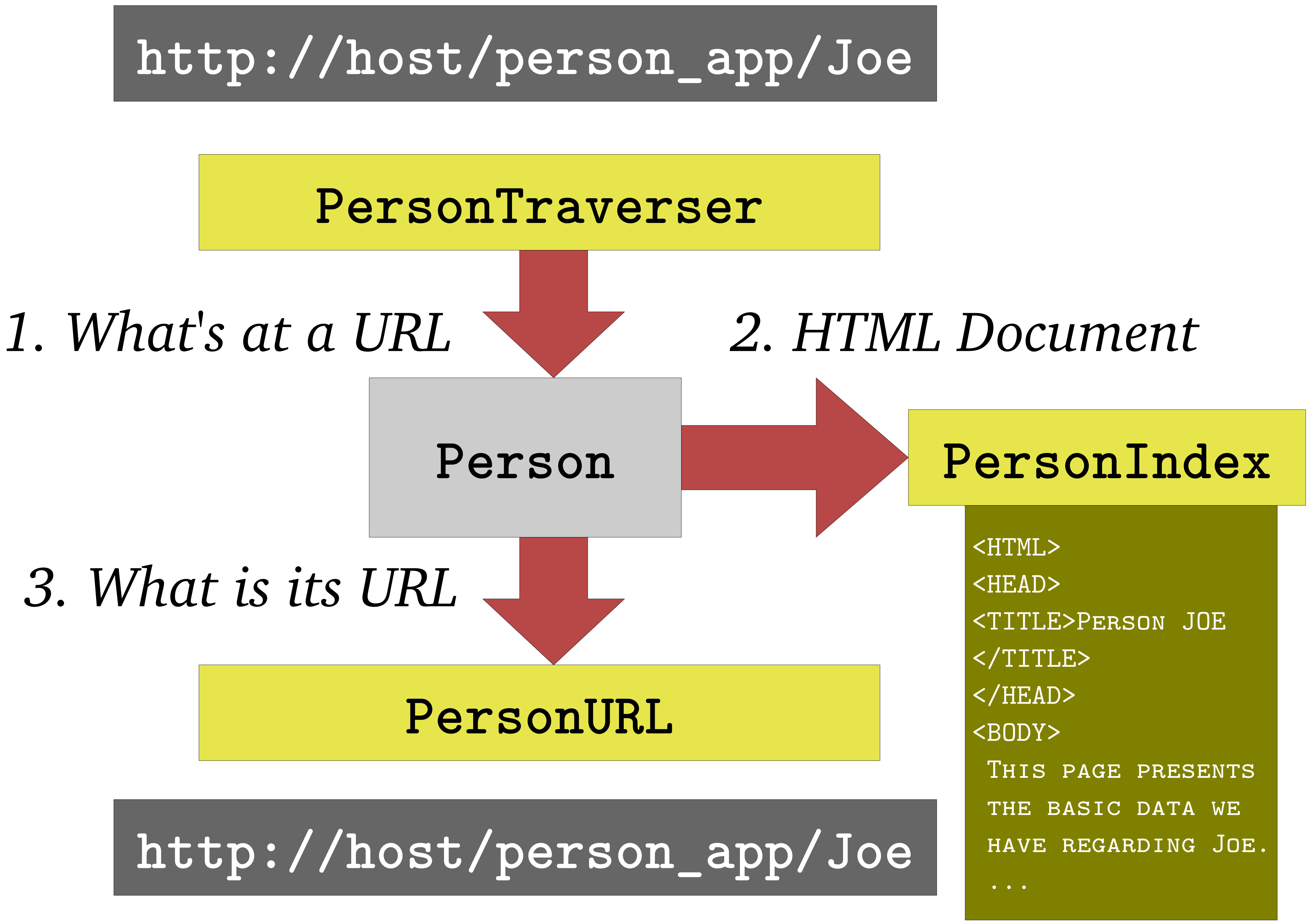
What is a person's URL?

```
class PersonURL(grok.MultiAdapter):
  grok.adapts(Person, IHTTPRequest)
  grok.implements(IAbsoluteURL)
  def __init__(self, person, req):
    self.person, self.req = person, req
  def __call__(self):
    base = grok.url(grok.getSite())
    return base + '/' + self.person.name
```

$6 + 3 + 8 = 17$ lines

6 + 3 + 8 = 17 lines

And the object has not been harmed!



```
<HTML>
<HEAD>
<TITLE>Person JOE
</TITLE>
</HEAD>
<BODY>
 This page presents
 the basic data we
 have regarding Joe.
 ...
```

Other Zope adapter uses

Other Zope adapter uses

Indexing — Index, Query, Search, ...
Data schemas — Schema, Vocabulary, DublinCore ...
Form generation — AddForm, EditForm, ...
Security — SecurityPolicy, Proxy, Checker, ...
Authentication — Login, Logout, Allow, Require, ...
Copy and paste — ObjectMover, ObjectCopier, ...
I18n — TranslationDomain, Translator, ...
Appearance — Skins, macros, viewlets, ...

Much, much more!

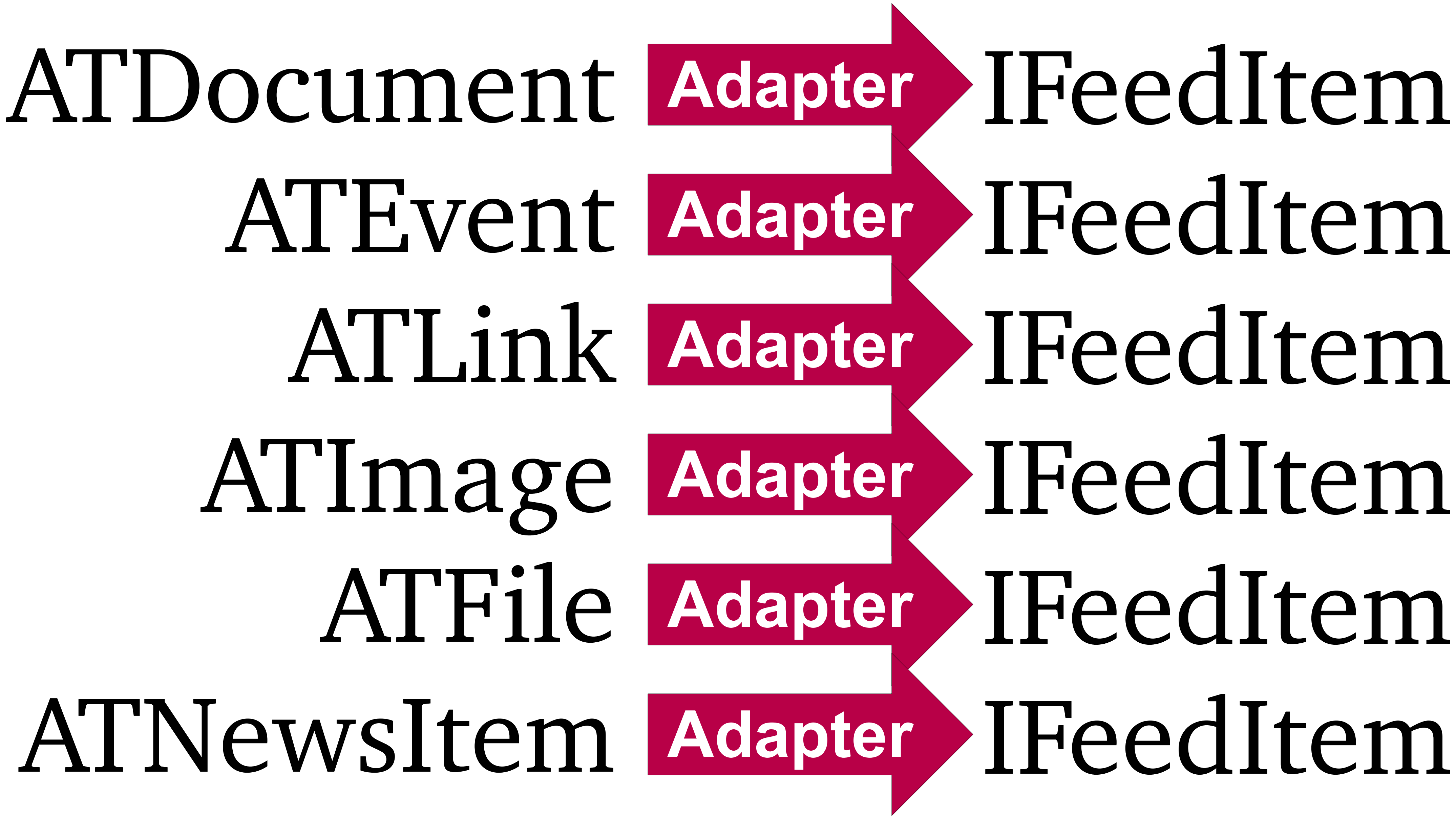
Other Zope adapter uses

And… “Vice”,
the Plone RSS/Atom feed engine that Paul Bugni presented on yesterday!

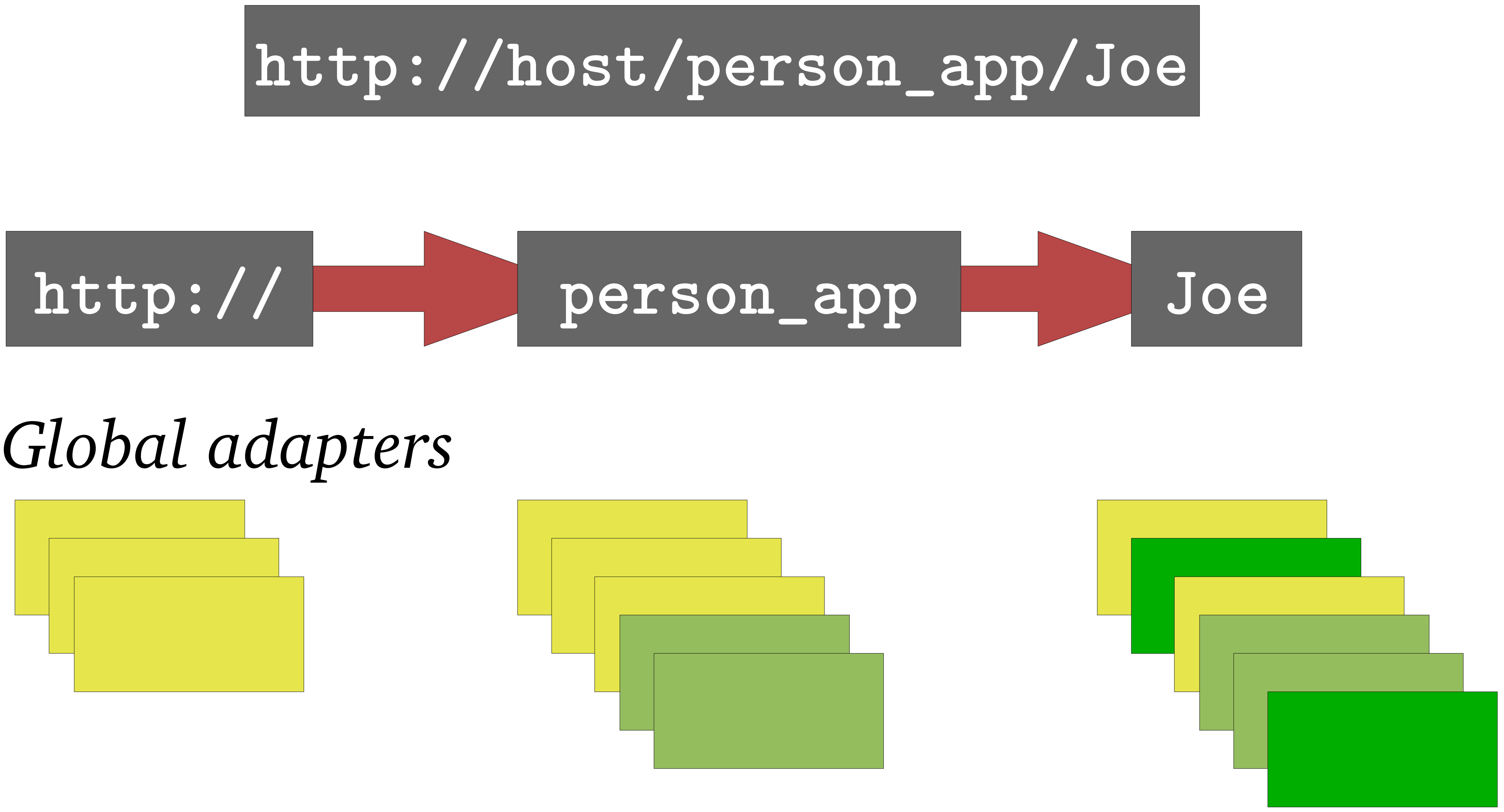
How does Vice give the AT content types RSS superpowers?

The same 3 steps!

Define an interface
Register adapters
Request adaptation


ATDocument
Adapter
IFeedItem
ATEvent
Adapter
IFeedItem
ATLink
Adapter
IFeedItem
ATImage
Adapter
IFeedItem
ATFile
Adapter
IFeedItem
ATNewsItem
Adapter
IFeedItem

Adapters can be local!



Local adapters add, override

Coming Attraction

five.grok

five.grok

Lennart Regebro
Martin Aspeli

Thank you!

http://zope.org/Products/Zope3
http://grok.zope.org/
http://rhodesmill.org/brandon/adapters
http://regebro.wordpress.com/
zope-dev@zope.org mailing list
grok-dev@zope.org mailing list
Web Component Development with Zope 3 by PvW