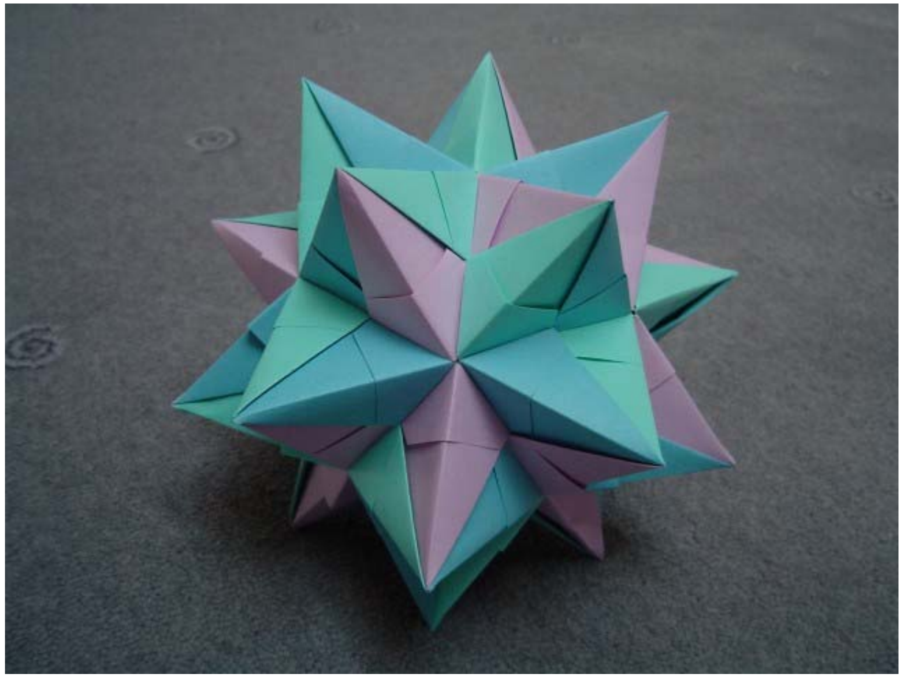
Eversion 101
An Introduction to Inside-Out Objects

David Golden
dagolden@cpan.org



June 26, 2006
YAPC::NA

Image source: Michal Kosmulski

An introduction to the inside-out technique

- Inside-out objects first presented by Dutch Perl hacker Abigail in 2002
  - Spring 2002 – First mention at Amsterdam.pm,
  - June 28, 2002 – YAPC NA "Two alternative ways of doing OO"
  - July 1, 2002 – First mention on Perlmonks

- Gained recent attention (notoriety?) as a recommended ***best practice*** with the publication of Damian Conway's *Perl Best Practices*

- Offer some interesting advantages... but at the cost of substantial complexity
  - Big question: Do the benefits outweight the complexity?

- Agenda for this tutorial:
  - Teach the basics
  - Describe the complexity
  - Let you decide

Eversion 101 Lesson Plan: Five C's

001 Concepts

010 Choices

011 Code

100 Complexity

101 CPAN

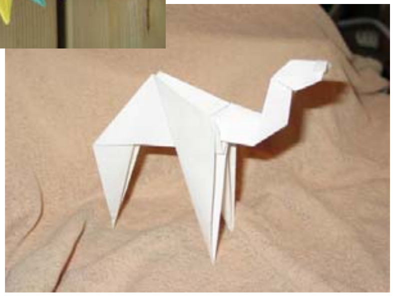


Image sources: Michal Kosmulski,Bill Odom

001 Concepts

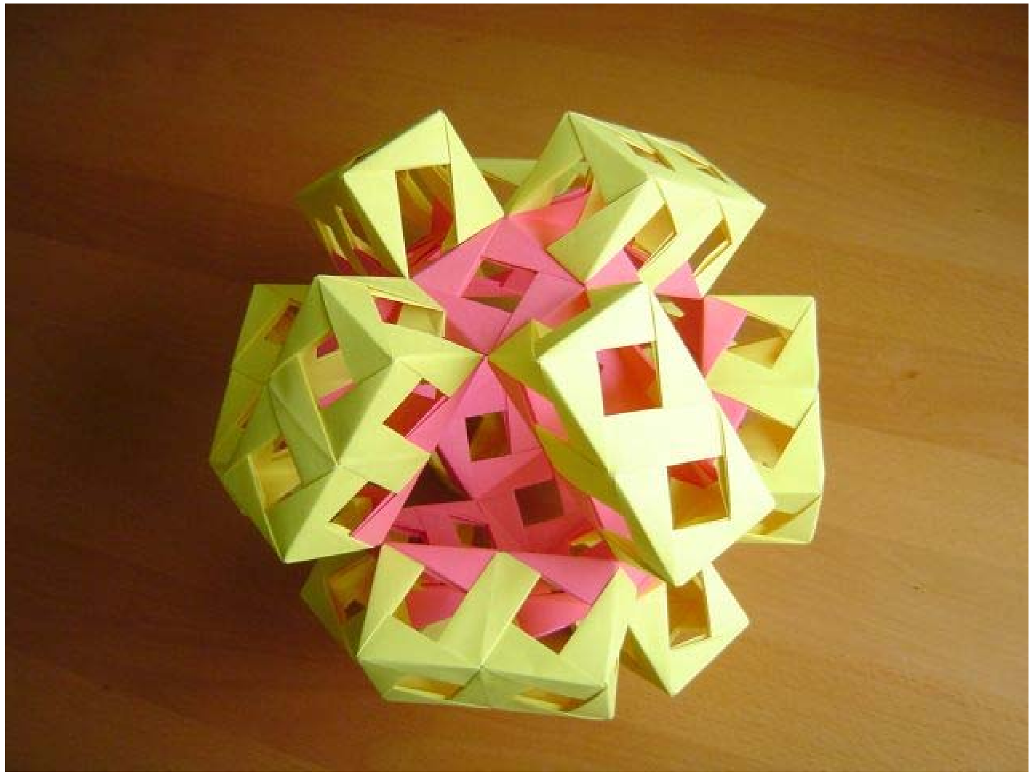


Image source: Michal Kosmulski

Three ideas at the core of this tutorial

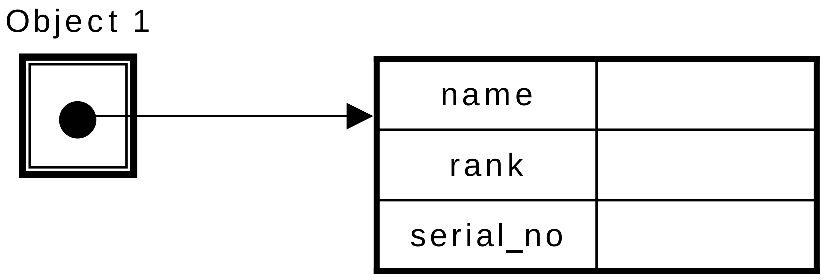
1. Encapsulation using lexical closure
2. Objects as indices versus objects as containers
3. Memory addresses as unique identifiers

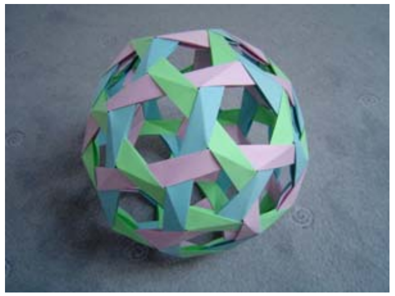
TIMTOWTDI: Everything else is combinations and variations

'Classic' Perl objects reference a data structure of properties

Hash-based object

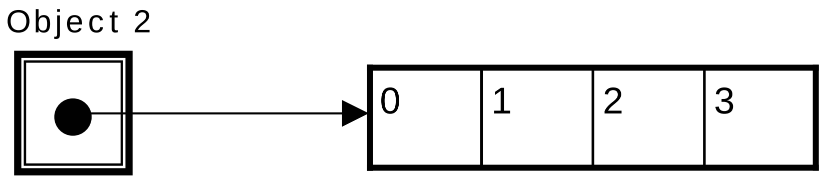
```
$obj = bless {}, "Some::Class";
```





Array-based object

```
$obj = bless [], "Some::Class";
```



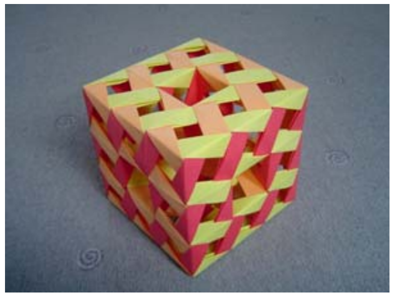


Image source: Michal Kosmulski

Complaint #1 for classic objects: No enforced encapsulation

- Frequent confusion describing the encapsulation problem
  - *Not* about **hiding** data, algorithms or implementation choices
  - *It is* about minimizing coupling with the code that uses the object

- The real question: *Culture versus control?*
  - Usually a matter of strong personal opinions
  - Advisory encapsulation: 'double yellow lines'
  - Enforced encapsulation: 'Jersey barriers'

- The underlying challenge: Tight coupling of superclasses and subclasses
  - Type of reference for data storage, e.g. hashes, array, scalars, etc.
  - Names of keys for hashes
  - 'Strong' encapsulation isn't even an option

Complaint #2: Hash key typos (and proliferating accessors)

- A typo in the name of a property creates a bug, not an error[1]
  - Code runs fine but results aren't as expected

```
$self->{naem} = 'James';
print $self->{name};  # What happened?
```

- Accessors to the rescue (?!)
  - Runtime error where the typo occurs
  - Every property access gains function call overhead

```
$self->naem('James'); # Runtime error here
print $self->name();
```

- My view: accessor proliferation for typo safety is probably not best practice
  - Private need for typo safety shouldn't drive public interface design
  - Couples implementation and interface

[1] Locked hashes are another solution as of Perl 5.8

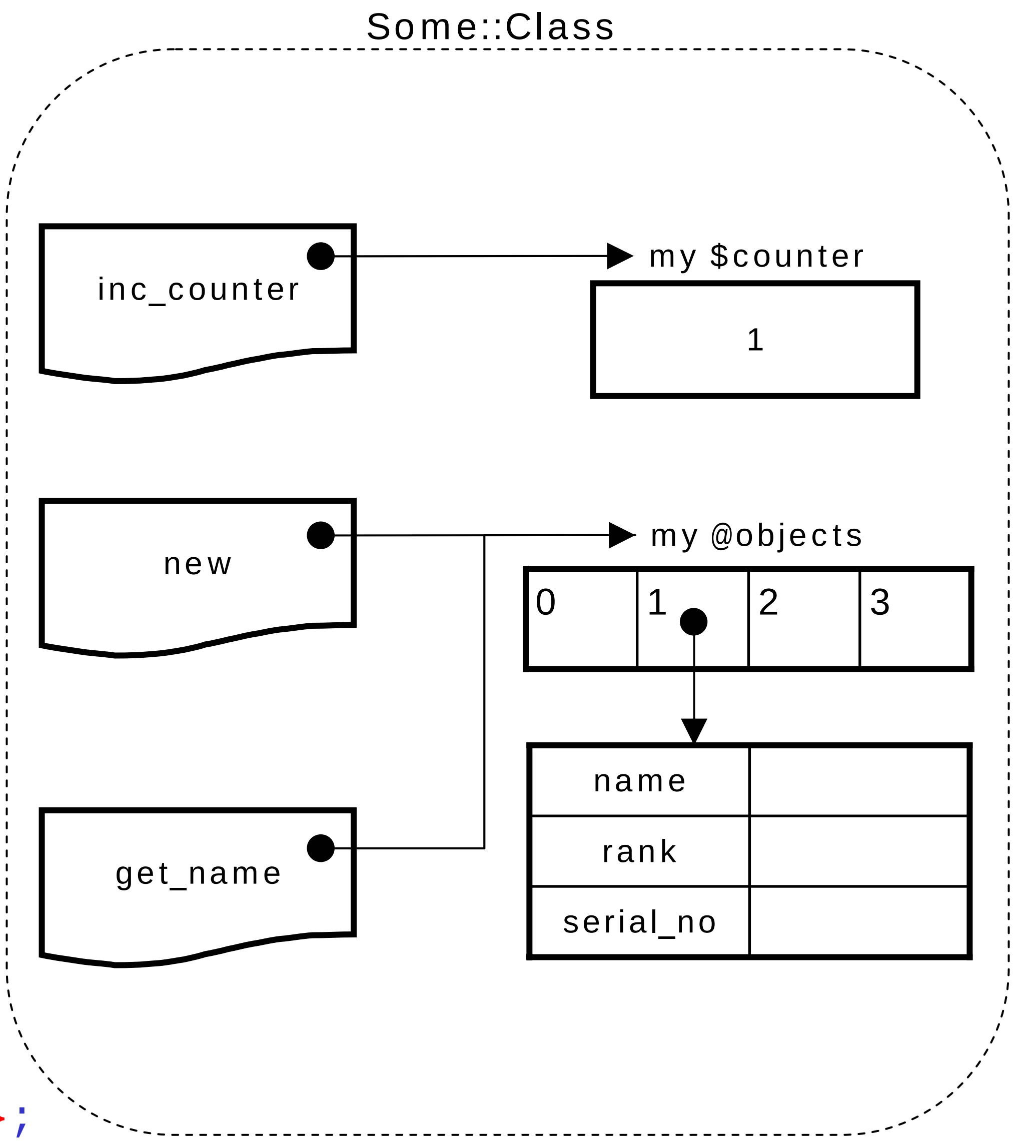
Eureka! We can enforce encapsulation with lexical closure

- Class properties always did this

```
package Some::Class;

my $counter;
sub inc_counter {
  my $self = shift;
  $counter++;
}
```

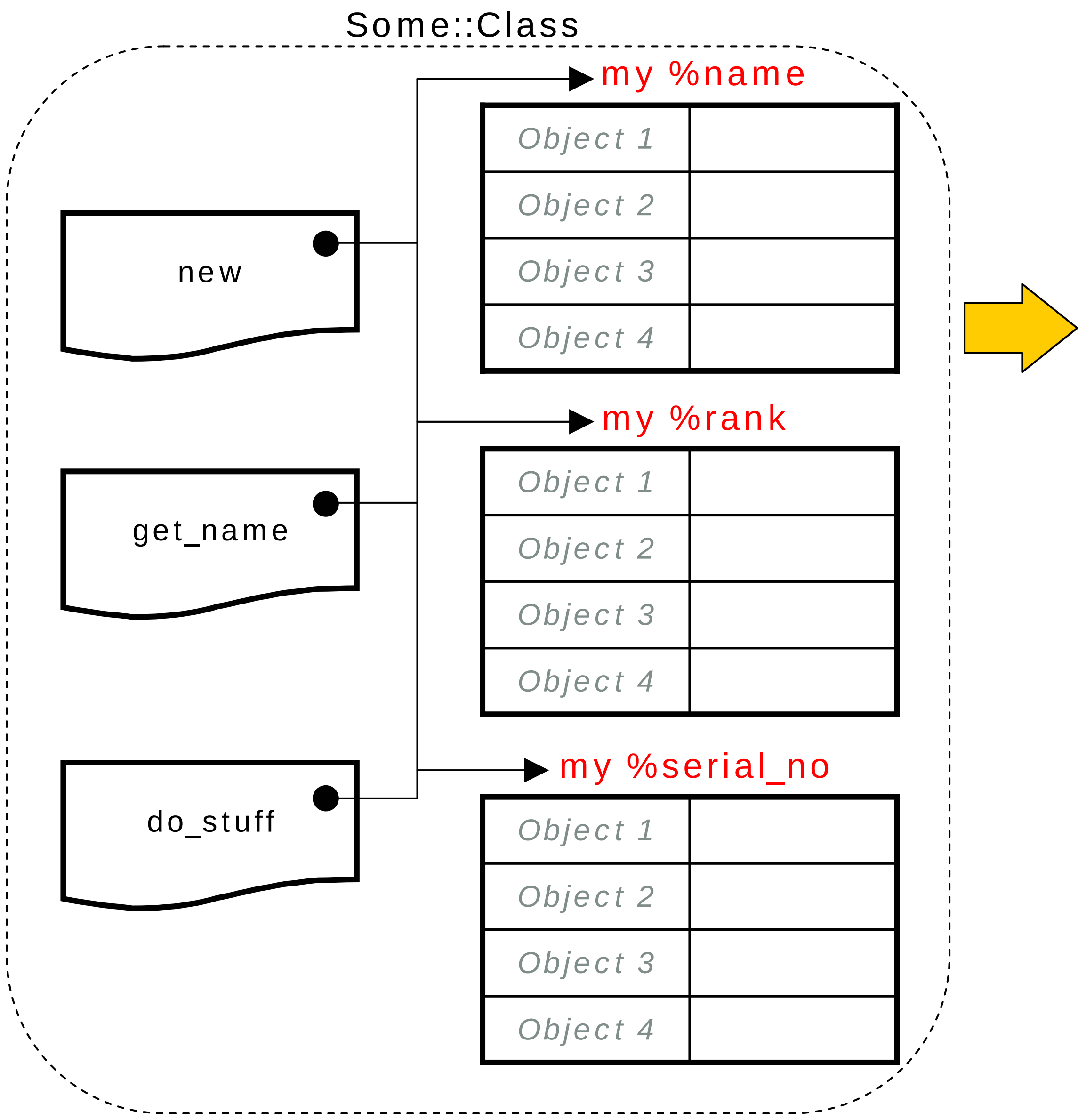
- Damian Conway's *flyweight pattern*[2]

```
my @objects;

sub new {
  my $class = shift;
  my $id = scalar @objects;
  $objects[$id] = {};
  return bless \$id, $class;
}

sub get_name {
  my $self = shift;
  return $objects[$$self]{name};
}
```



[2] A brief version of this was introduced in *Advanced Perl Programming, 1st edition as ObjectTemplate*

'Inside-Out' objects use an index into lexicals for *each* property



Lexical properties give compile-time typo checking under `strict`!

```
# Correct:
$name{ $$self };

# Compiler error:
$naem{ $$self };
```

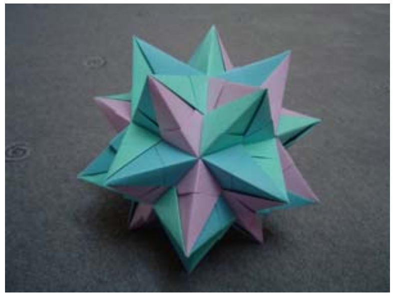
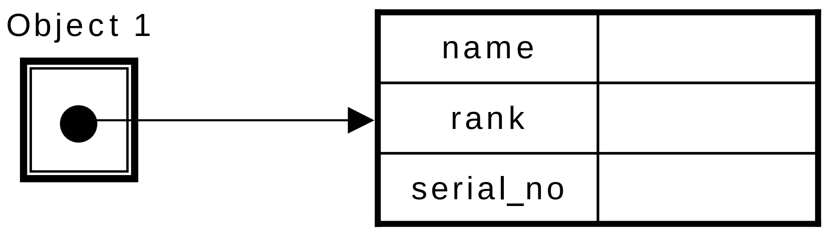
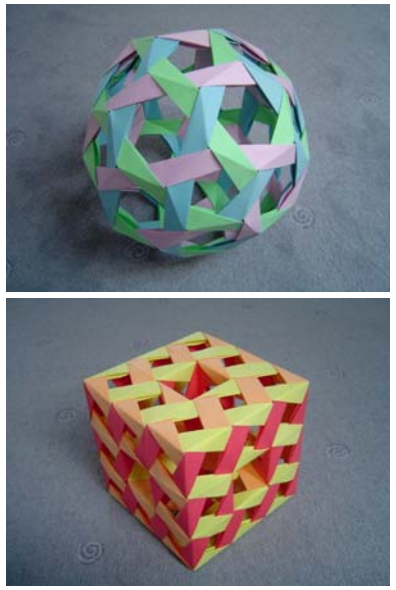


Image source: Michal Kosmulski

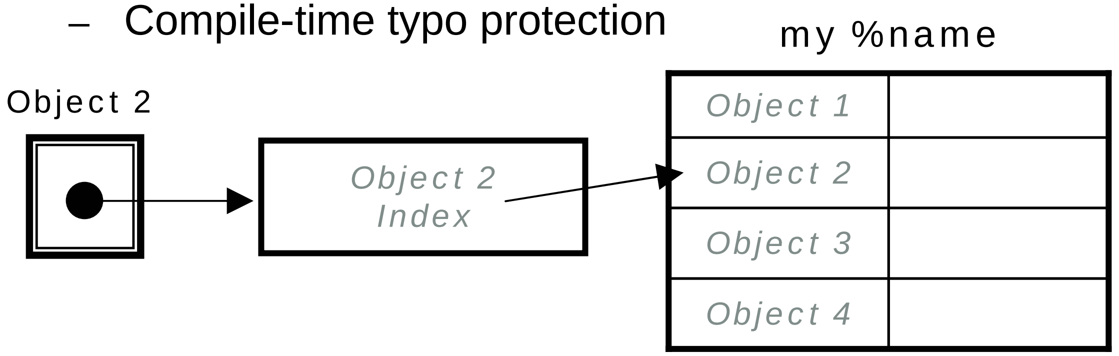
Review: 'Classic' versus 'Inside-Out'

- Classic: **Objects as containers**
  - Object is a reference to a data structure of properties
  - No enforced encapsulation
  - Hash-key typo problem





- Inside-Out: **Objects as indices**
  - Object is an index into a lexical data structure for each property
  - Enforced **encapsulation using lexical closure**
  - Compile-time typo protection



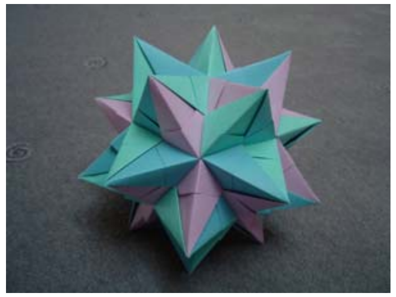


Image source: Michal Kosmulski

010 Choices

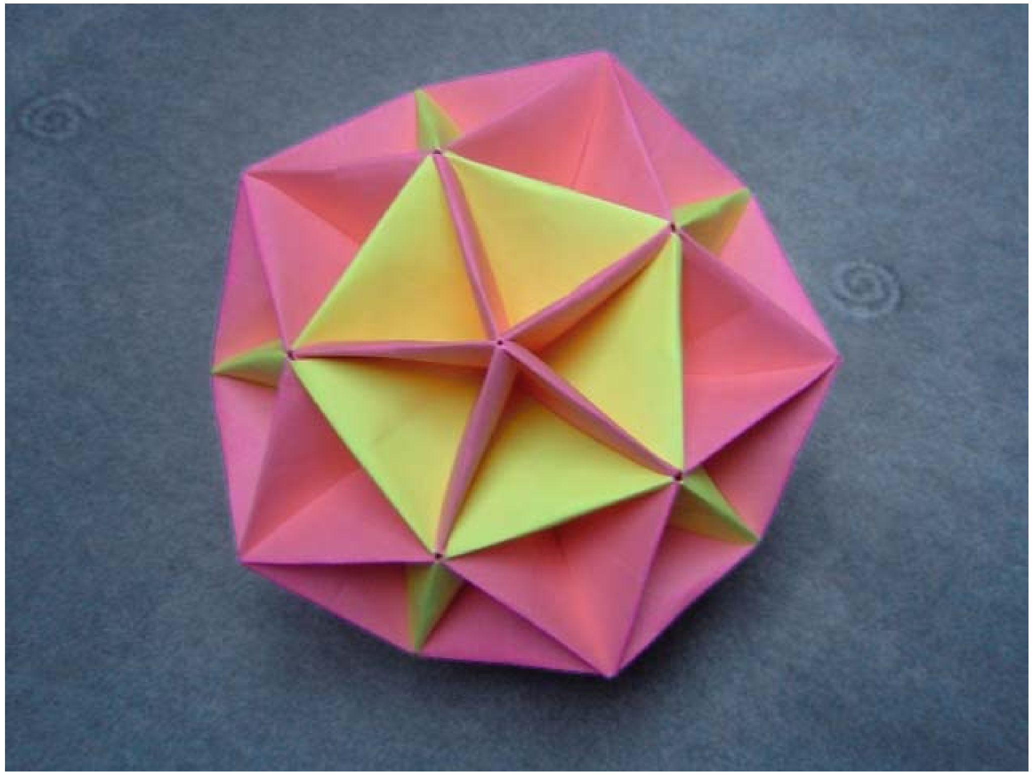
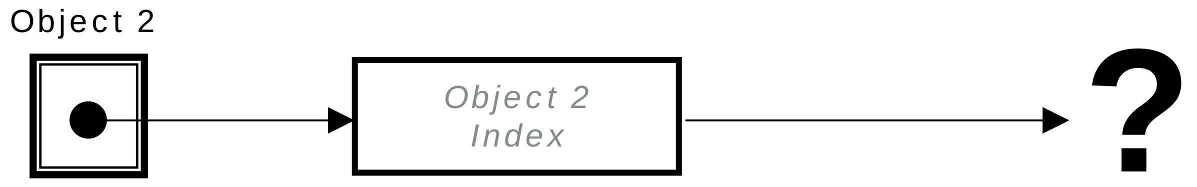
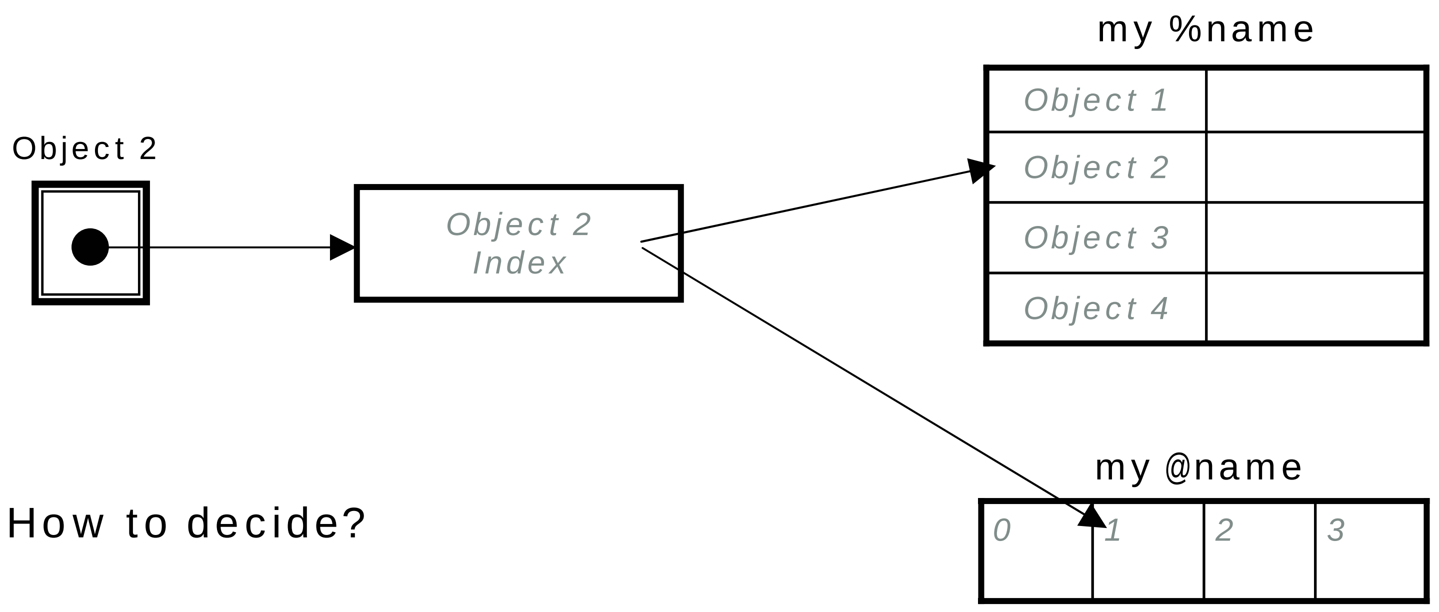


Image source: Michal Kosmulski

What data structure to use for inside-out properties?



What data structure to use for inside-out properties?



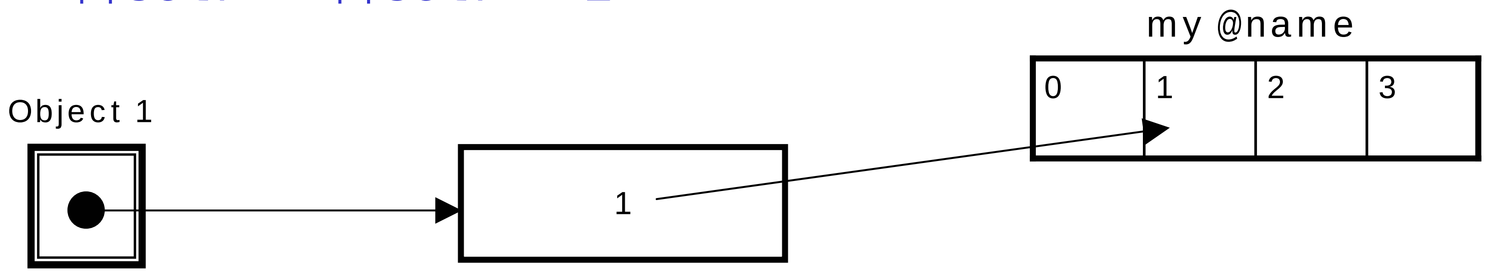
How to decide?

- Array
  - Fast access
  - Index limited to sequential integers
  - **Needs DESTROY to recycle indices** to prevent runaway growth of property arrays
- Hash
  - Slow(er) access
  - Any string as index
  - Uses much more memory (particularly if keys are long)
  - **Needs DESTROY to free property memory** to avoid leakage

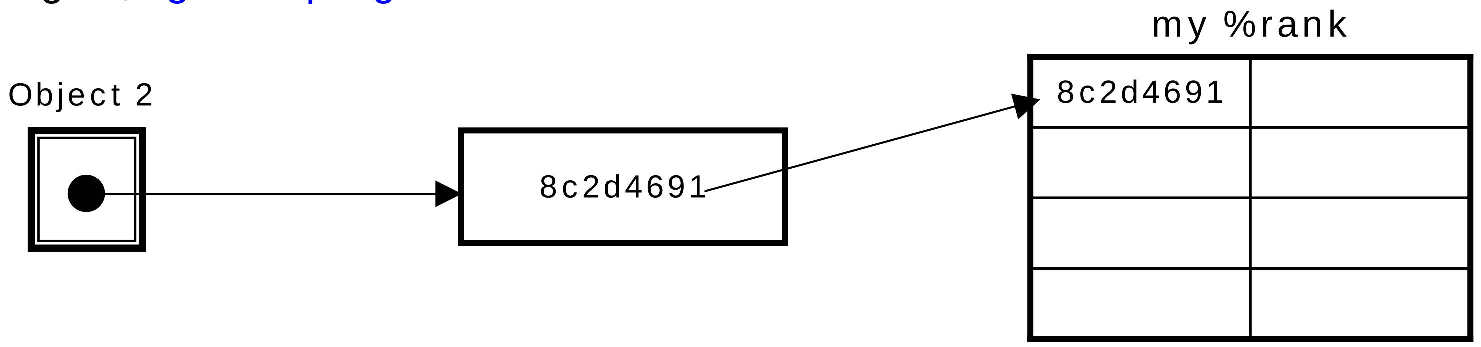
What index? (And stored how?)

- Sequential number, stored in a blessed scalar
  - Tight coupling – subclasses must also use a blessed scalar
  - Subclass must use an index provided by the superclass
  - Unless made read-only, objects can masquerade as other objects, whether references to them exist or not!

```
$$self = $$self + 1
```

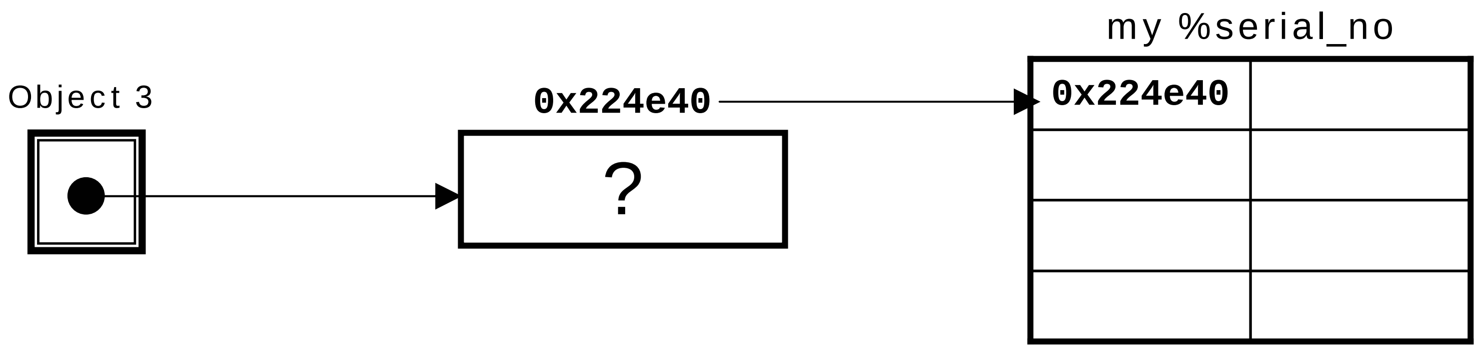


- A unique, hard-to-guess number, stored in a blessed scalar (e.g. with `Data::UUID`)
  - Again, tight coupling – subclasses must also use a blessed scalar



An alternative: use the memory address as a unique identifier

- Unique and consistent for the life of the object
  - Except under threads (needs a `CLONE` method)



- Several ways to get the memory address; **only `refaddr()` is safe**[3]

```
$property{ refaddr $self }
```

- Otherwise, overloading of stringification or numification can give unexpected results

```
$property{ "$self" }
$property{ $self   } # like "$self"
$property{ 0+$self }
```

[3] Available in `Scalar::Util`

Using the memory address directly allows 'black-box' inheritance

- When used directly as `refaddr $self`, *the type of blessed reference no longer matters*
  - Subclasses don't need to know or care what the superclass is using as a data type
  - Downside is slight overhead of `refaddr $self` for each access

- *Black-box inheritance*[4] – using a superclass object as the reference to bless
  - a.k.a. 'foreign inheritance' or 'opaque inheritance'
  - An alternative to facade/delegator/adaptor patterns and some uses of tied variables
  - Superclass doesn't even have to be an inside-out object

```
use base 'Super::Class';

sub new {
    my $class = shift;
    my $self = Super::Class->new( @_ );
    bless $self, $class;
    return $self;
}
```

- There is still a problem for multiple inheritance of different base object types

[4] Thanks to Boston.pm for name brainstorming

These choices give four types of inside-out objects

✓ 1. **Array-based** properties, with **sequential ID's** stored in a blessed scalar
   - Fast and uses less memory
   - Insecure unless index is made read-only
   - Requires index recycling
   - Subclasses must also use a blessed scalar – no black-box inheritance

? 2. **Hash-based** properties, with a **unique, hard-to-guess number** stored in a blessed scalar
   - Slow and uses more memory
   - Robust, even under threads
   - Subclasses must also use a blessed scalar – no black-box inheritance

✗ 3. **Hash-based** properties, with the **memory address** *stored in a blessed scalar*
   - Subclasses must also use a blessed scalar – no black-box inheritance
   - Combines the worst of (2) and (4) for a slight speed increase

✓ 4. **Hash-based** properties, with the **memory address used directly**
   - Slow and uses more memory
   - Black-box inheritance possible
   - **Not thread-safe unless using a `CLONE` method**

011 Code

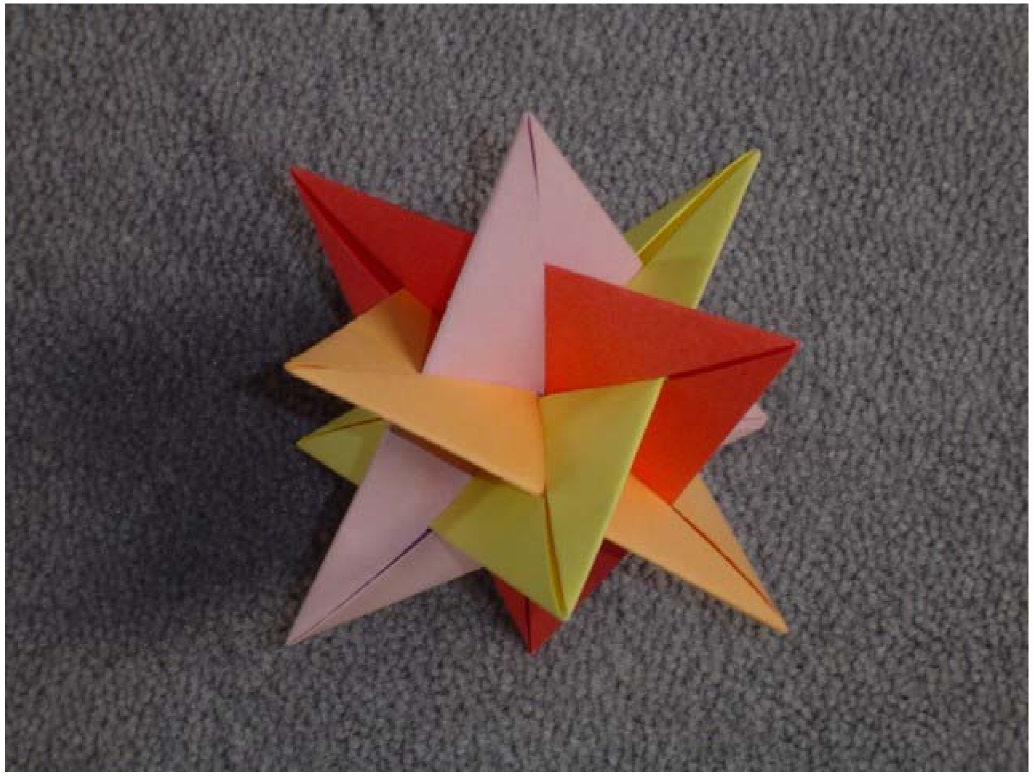


Image source: Michal Kosmulski

File::Marker: a simple inside-out objects with black-box inheritance

Key Features

- Useable ***directly*** as a filehandle (`IO::File`) without tying

  ```
  $fm = File::Marker->new( $filename );
  $line = <$fm>;
  ```

- Set named markers for the current location in an opened file

  ```
  $fm->set_marker( $mark_name );
  ```

- Jump to the location indicated by a marker

  ```
  $fm->goto_marker( $mark_name );
  ```

- Let users jump back to the last jump point with a special key-word

  ```
  $fm->goto_marker( "LAST" );
  ```

- Clear markers when opening a file

  ```
  $fm->open( $another_file ); # clear all markers
  ```

File::Marker constructor

```
use base 'IO::File';
use Scalar::Util qw( refaddr );

my %MARKS = ();

sub new {
    my $class = shift;
    my $self = IO::File->new();
    bless $self, $class;
    $self->open( @_ ) if @_;
    return $self;
}

sub open {
    my $self = shift;
    $MARKS{ refaddr $self } = {};
    $self->SUPER::open( @_ );
    $MARKS{ refaddr $self }{ 'LAST' } = $self->getpos;
    return 1;
}
```

Full version of File::Marker available on CPAN

- Uses `strict` and `warnings`
- Argument validation
- Error handling
- Extensive test coverage
- Thread safety

File::Marker destructor and methods

```
sub DESTROY {
    my $self = shift;
    delete $MARKS{ refaddr $self };
}

sub set_marker {
    my ($self, $mark) = @_;
    $MARKS{ refaddr $self }{ $mark } = $self->getpos;
    return 1;
}

sub goto_marker {
    my ($self, $mark) = @_;
    my $old_position = $self->getpos; # save for LAST
    $self->setpos( $MARKS{ refaddr $self }{ $mark } );
    $MARKS{ refaddr $self }{ 'LAST' } = $old_position;
    return 1;
}
```

Seeing it in action

file_marker_example.pl

```
use strict;
use warnings;
use File::Marker;

my $fm = File::Marker->new(
    "textfile.txt"
);

print scalar <$fm>, "--\n";

$fm->set_marker("line2");

print <$fm>, "--\n";

$fm->goto_marker("line2");

print scalar <$fm>;
```

textfile.txt

```
this is line one
this is line two
this is line three
this is line four
```

Output

```
this is line one
--
this is line two
this is line three
this is line four
--
this is line two
```

Complexity

Image source: Michal Kosmulski

Five pitfalls

1. Not using `DESTROY` to free memory or reclaim indices
2. **Serialization – without special precautions**

Inherent to all inside-out objects

3. Not using `refaddr()` to get a memory address
4. **Not providing `CLONE` for thread-safety**

Only if using memory addresses

5. **Using a CPAN implementation that gets these wrong**

Serialization requires extra work

- Programmers often assume an object reference is a data structure

```
Dump( $object );  # implicitly breaks encapsulation
```

- OO purists might say that objects should provide a dump method

```
$object->dump();  # OO-style
```

- But, what if objects are part of a larger non-OO data structure?

```
@list = ( $obj1, $obj2, $obj3 );
freeze( \@list ); # What now?
```

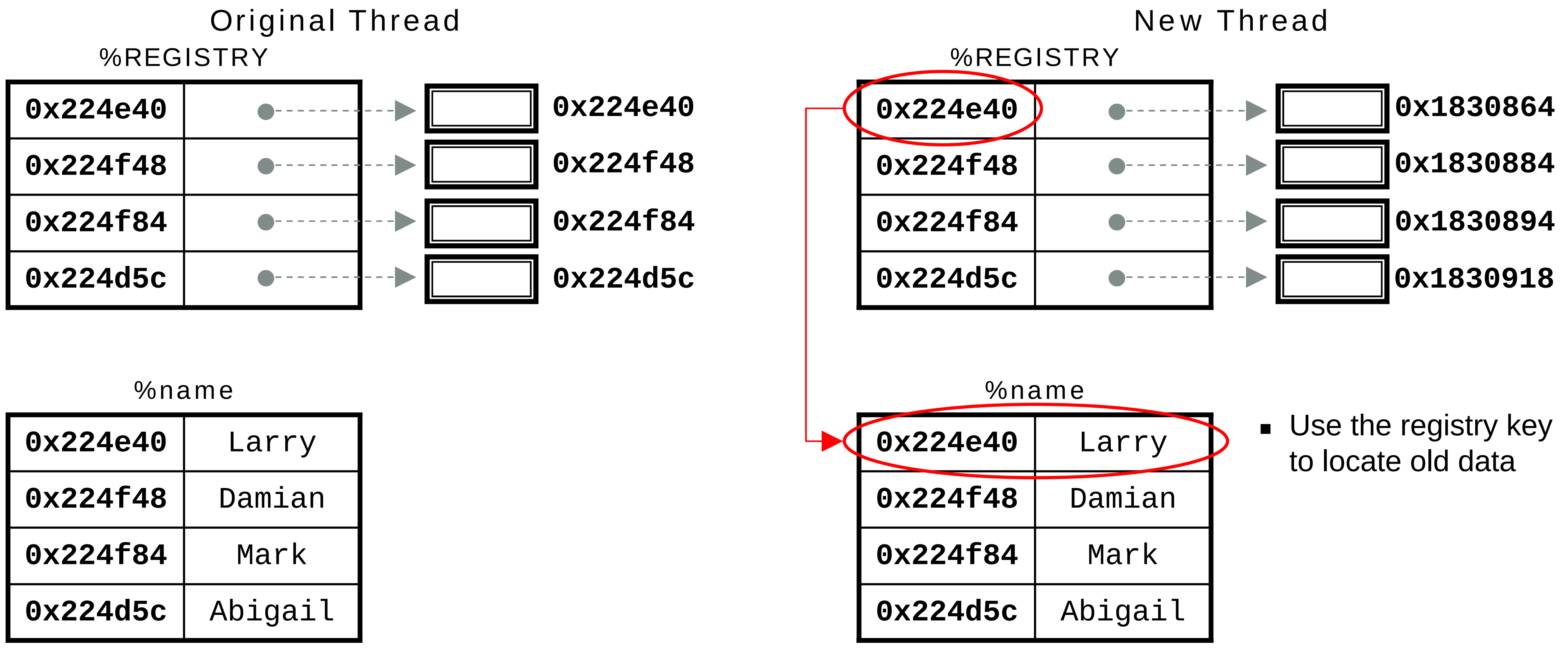
- Fortunately, `Storable` provides hooks for objects to control their serialization

```
STORABLE_freeze();
STORABLE_thaw();
STORABLE_attach(); # for singletons
```

- Of `Data::Dumper` and clones, only `Data::Dump::Streamer` provides the right kind of hooks (but doesn't easily support singleton objects... yet)

Use **CLONE** for thread-safe **refaddr** indices

- Starting with Perl 5.8, thread creation calls `CLONE` once per package, if it exists
  - Called from the context of the *new* thread
  - Works for Win32 pseudo-forks (but not for Perl 5.6)
- Use a registry with weak references to track and remap old indices
  - weaken provided by the XS version of Scalar::Util



Original Thread %REGISTRY:

0x224e40	● → 0x224e40
0x224f48	● → 0x224f48
0x224f84	● → 0x224f84
0x224d5c	● → 0x224d5c

New Thread %REGISTRY:

0x224e40	● → 0x1830864
0x224f48	● → 0x1830884
0x224f84	● → 0x1830894
0x224d5c	● → 0x1830918

Original Thread %name:

0x224e40	Larry
0x224f48	Damian
0x224f84	Mark
0x224d5c	Abigail

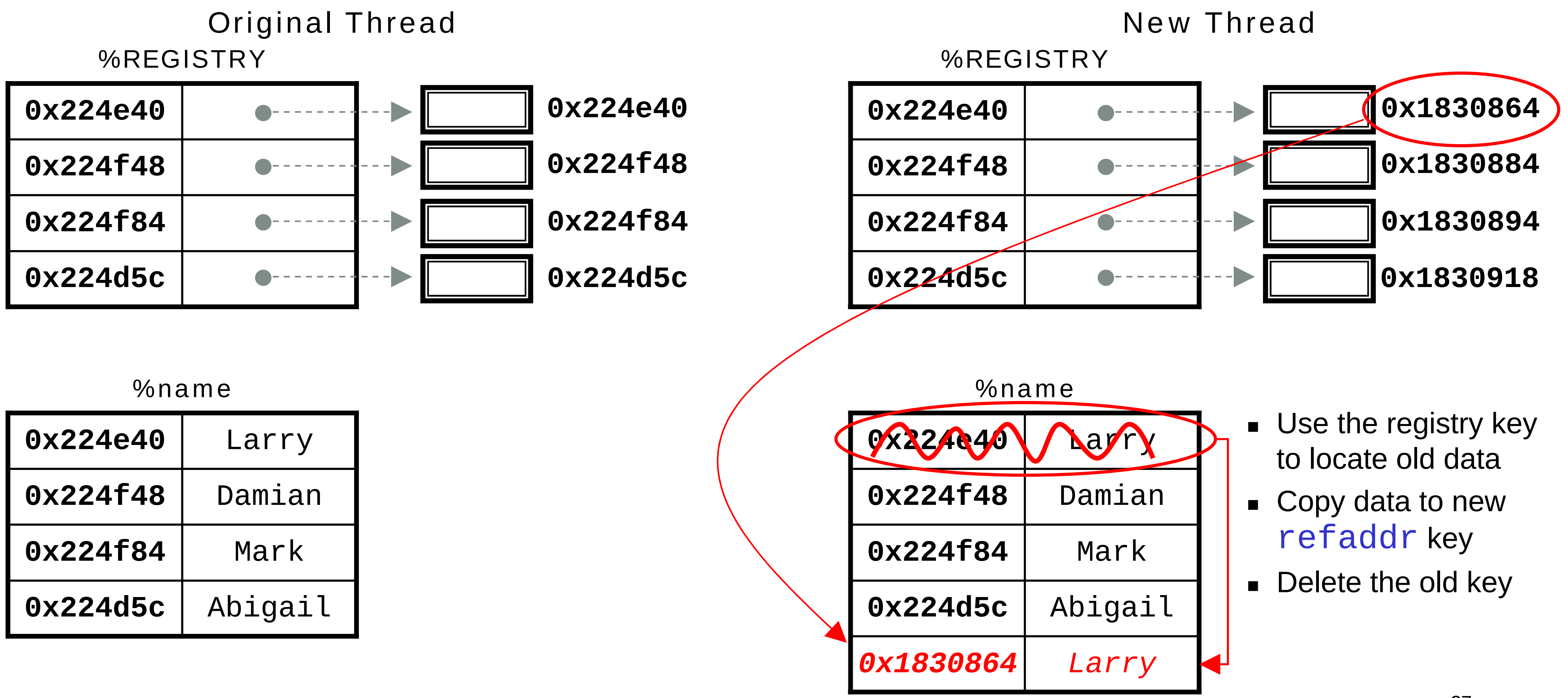
New Thread %name:

0x224e40	Larry
0x224f48	Damian
0x224f84	Mark
0x224d5c	Abigail

- Use the registry key to locate old data

Use `CLONE` for thread-safe `refaddr` indices

- Starting with Perl 5.8, thread creation calls `CLONE` once per package, if it exists
  - Called from the context of the *new* thread
  - Works for Win32 pseudo-forks (but not for Perl 5.6)
- Use a registry with weak references to track and remap old indices
  - weaken provided by the XS version of Scalar::Util



Original Thread %name

0x224e40	Larry
0x224f48	Damian
0x224f84	Mark
0x224d5c	Abigail

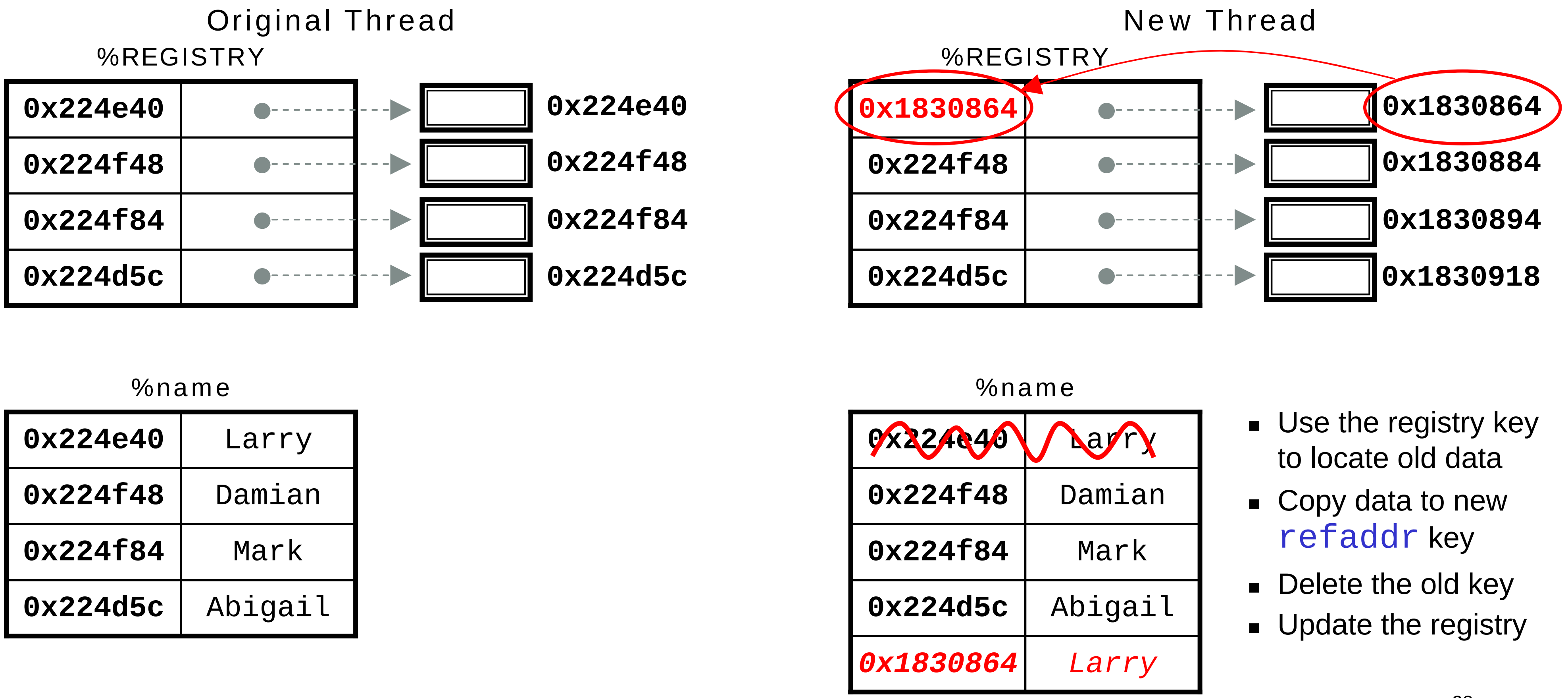
New Thread %name

~~0x224e40~~	~~Larry~~
0x224f48	Damian
0x224f84	Mark
0x224d5c	Abigail
0x1830864	*Larry*

- Use the registry key to locate old data
- Copy data to new `refaddr` key
- Delete the old key

Use CLONE for thread-safe refaddr indices

- Starting with Perl 5.8, thread creation calls CLONE once per package, if it exists
  - Called from the context of the *new* thread
  - Works for Win32 pseudo-forks (but not for Perl 5.6)
- Use a registry with weak references to track and remap old indices
  - weaken provided by the XS version of Scalar::Util



- Use the registry key to locate old data
- Copy data to new refaddr key
- Delete the old key
- Update the registry

CPAN

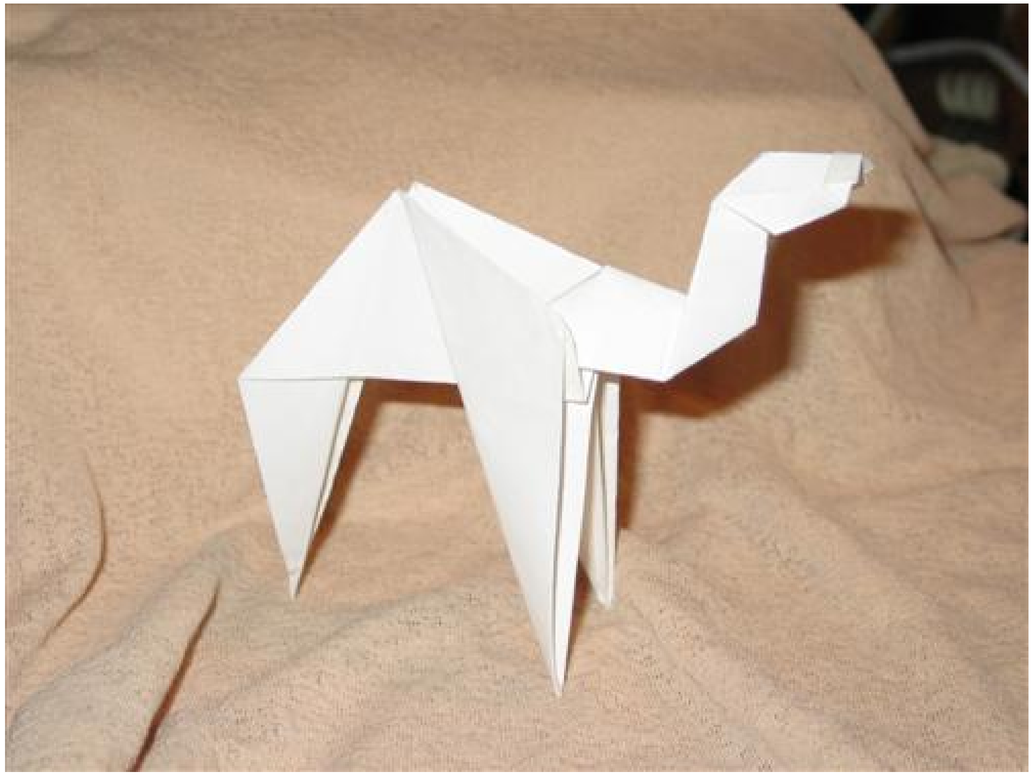


Image source: Bill Odom

Two CPAN modules to consider and several to (probably) avoid

✓ ▪ `Object::InsideOut`
  - Currently the most flexible, robust implementation of inside-out objects
  - But, black-box inheritance handled via delegation (including multiple inheritance)

✓ ▪ `Class::InsideOut` *(disclaimer: I wrote this one)*
  - A safe, simple, minimalist approach
  - Manages inside-out complexity but leaves all other details to the user
  - Supports black-box inheritance directly

? ▪ `Class::Std`
  - Rich support for class hierarchies and overloading
  - But, not yet thread-safe
  - Hash-based with memory-address, but not in a way that allows black-box inheritance

✗ ▪ All of these have flaws or limitations:

```
Base::Class                                 Lexical::Attributes
Class::BuildMethods                         Object::LocalVars
Class::MakeMethods::Templates::InsideOut
```

!? ▪ ... but coming "soon" in Perl 5.10: `Hash::Util::FieldHash`

Questions?

Bonus Slides

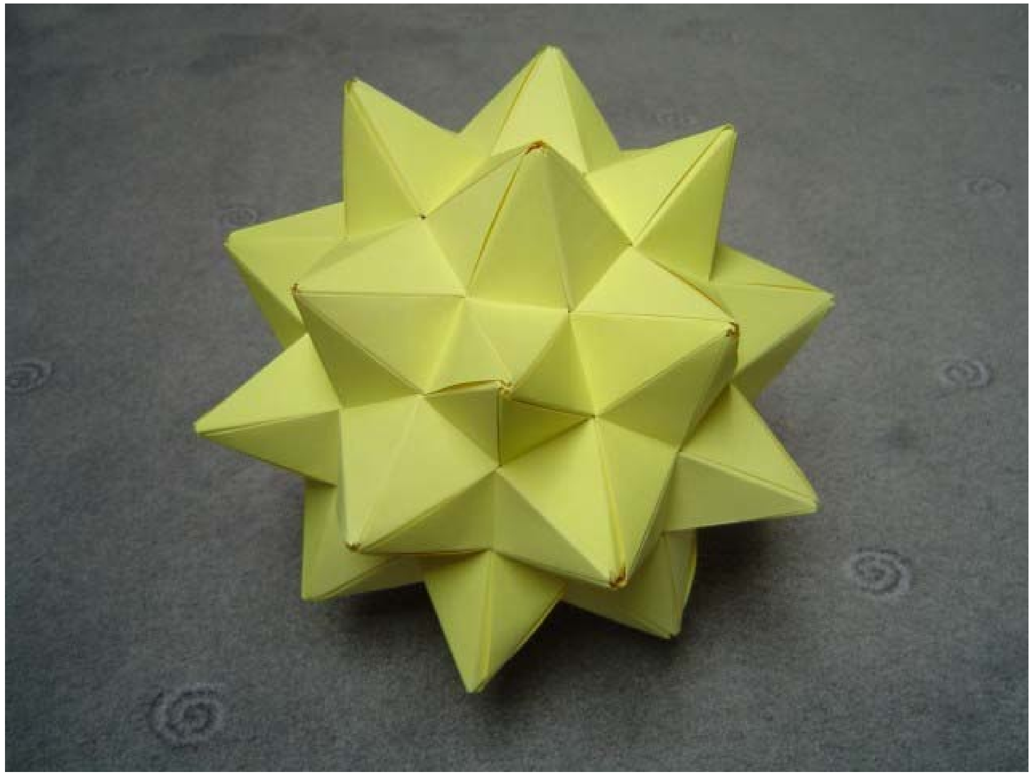


Image source: Michal Kosmulski

File::Marker with thread safety, part one

```
use base 'IO::File';
use Scalar::Util qw( refaddr weaken );

my %MARKS = ();
my %REGISTRY = ();

sub new {
    my $class = shift;
    my $self = IO::File->new();
    bless $self, $class;
    weaken( $REGISTRY{ refaddr $self } = $self );
    $self->open( @_ ) if @_;
    return $self;
}

sub DESTROY {
    my $self = shift;
    delete $MARKS{ refaddr $self };
    delete $REGISTRY{ refaddr $self };
}
```

File::Marker with thread safety, part two

```
sub CLONE {
    for my $old_id ( keys %REGISTRY ) {

        # look under old_id to find the new, cloned reference
        my $object = $REGISTRY{ $old_id };
        my $new_id = refaddr $object;

        # relocate data
        $MARKS{ $new_id } = $MARKS{ $old_id };
        delete $MARKS{ $old_id };

        # update the weak reference to the new, cloned object
        weaken ( $REGISTRY{ $new_id } = $object );
        delete $REGISTRY{ $old_id };
    }
    return;
}
```

Inside-out CPAN module comparison table

Module	Storage	Index	CLONE?	Serializes?	Other Notes
★ Object::InsideOut (1.27)	Array or Hash	Array: Integers Hash: Cached refaddr $self	Yes	Custom dump() Storable hooks	▪ black-box inheritance using delegation pattern ▪ Custom :attribute handling ▪ mod_perl safe ▪ Good thread support
★ Class::InsideOut (1.00)	Hash	refaddr $self	Yes	Storable hooks	▪ Simple, minimalist approach ▪ Supports direct black-box inheritance ▪ mod_perl safe
Class::Std (0.0.8)	Hash	refaddr $self	No	Storable hooks with Class::Std:: Storable	▪ Custom :attribute handling; ▪ mod_perl safe ▪ No black-box inheritance support ▪ Rich class hierarchy support

Inside-out CPAN module comparison table (continued)

Module	Storage	Index	CLONE?	Serializes?	Other Notes
Base::Class (0.11)	Hash of Hashes ('Flyweight')	"$self"	No	Dumper to STDERR only No Storable support	▪ Lexical storage in Base::Class ▪ Autogenerates all properties/accessors via AUTOLOAD
Class::BuildMethods (0.11)	Hash of Hashes ('Flyweight')	refaddr $self	No	Custom dump() No Storable support	▪ Lexical storage in Class::BuildMethods, not the class that uses it; provides accessors for use in code
Class::MakeMethods ::Template::InsideOut (1.01)	Hash	"$self"	No	No	▪ Part of a complex class generator system; steep learning curve

Inside-out CPAN module comparison table (continued)

Module	Storage	Index	CLONE?	Serializes?	Other Notes
Lexical::Attributes (1.4)	Hash	refaddr $self	No	No	▪ Source filters for Perl-6-like syntax
Object::LocalVars (0.16)	Package global hash	refaddr $self	Yes	No	▪ Custom :attribute handling ▪ mod_perl safe ▪ Wraps methods to locally alias $self and properties ▪ Highly experimental

Some CPAN Modules which use the inside-out technique

- `Data::Postponed`
  - Delay the evaluation of expressions to allow post facto changes to input variables
- `File::Marker` (from this tutorial)
  - Set and jump between named position markers on a filehandle
- `List::Cycle`
  - Objects for cycling through a list of values
- `Symbol::Glob`
  - Remove items from the symbol table, painlessly

References for further study

- Books by Damian Conway
  - *Object Oriented Perl.* Manning Publications. 2000
  - *Perl Best Practices*. O'Reilly Media. 2005

- Perlmonks – see my scratchpad for a full list: <http://perlmonks.org/index.pl?node_id=360998>
  - Abigail-II. "Re: Where/When is OO useful?". July 1, 2002 <http://perlmonks.org/index.pl?node_id=178518>
  - Abigail-II. "Re: Tutorial: Introduction to Object-Oriented Programming". December 11, 2002 <http://perlmonks.org/index.pl?node_id=219131>
  - demerphq. "Yet Another Perl Object Model (Inside Out Objects)". December 14, 2002 <http://perlmonks.org/index.pl?node_id=219924>
  - xdg. "Threads and fork and CLONE, oh my!". August 11, 2005 <http://perlmonks.org/index.pl?node_id=483162>
  - jdhedden. "Anti-inside-out-object-ism". December 9, 2005 <http://perlmonks.org/index.pl?node_id=515650>

- Perl documentation (aka "perldoc") – also at <http://perldoc.perl.org>
  - perlmod
  - perlfork