What's All the Fuss?

An Inside-Out Object Tutorial

David A. Golden

January 17, 2006

Perl Seminar NY

Copyright © 2006 David A. Golden

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*

- Despite their benefits, they bring significant complexity and are not universally welcomed

- Goals of this tutorial:
  - Review the pros and cons
  - Teach the basics of the inside-out technique
  - Provide a quick overview of inside-out tools on CPAN

Copyright © 2006 David A. Golden

Q. Why do people like them?
A. Safety and flexibility

- Enforced encapsulation of properties
- Property-name typos are compile-time errors, not run-time bugs
- 'Foreign inheritance' allows subclassing any blessed reference

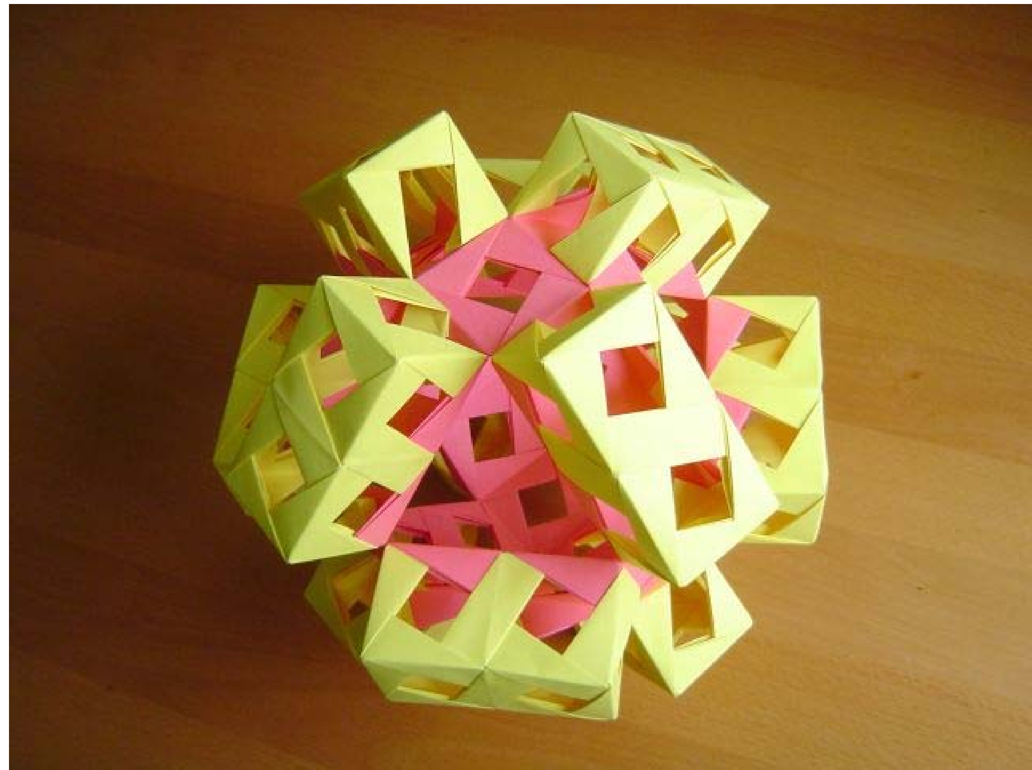
Copyright © 2006 David A. Golden

Q. Why do people hate them?
A. Complexity! (And often a lack of need.)

- Other programmers not familiar with the technique
- Enforcing encapsulation may not be a priority for everyone
- Foreign inheritance is kludgy
- Risk of memory leaks
  - `DESTROY` is mandatory
- Not automatically thread-safe
  - Must implement `CLONE` (which requires Perl 5.8)
  - Requires XS version of `Scalar::Util` for `weaken`
- Inside-out objects are not easily serialized or dumped
  - `STORABLE_freeze` and `STORABLE_thaw` can be tricky to get right
- Inheritance trees complicate all of the above issues
- Many inside-out module generators on CPAN are flawed or try to do too much
  - Perl attributes that aren't mod_perl compatible
  - Source filters or other syntactic sugar

Copyright © 2006 David A. Golden

Concepts



Copyright © 2006 David A. Golden

Image source: Michal Kosmulski

Three ideas at the core of this tutorial

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

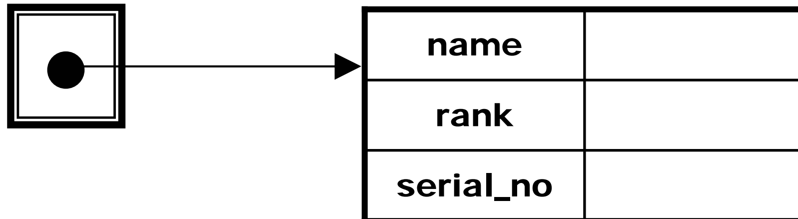
Everything else is the result of TIMTOWTDI

Copyright © 2006 David A. Golden

'Classic' Perl objects reference a data structure of properties

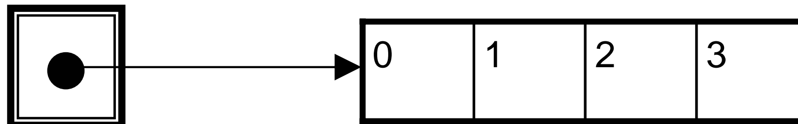
Hash-based object

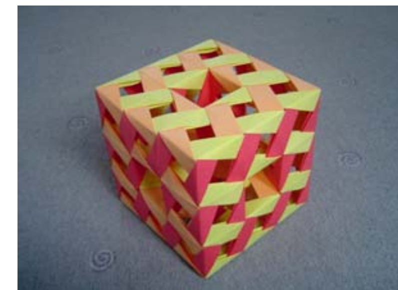
```
$obj = bless {}, "Some::Class";
```



Array-based object

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





Copyright © 2006 David A. Golden

Image source: Michal Kosmulski

Complaint #1 for classic objects: No enforced encapsulation

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

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

- Tight coupling of superclasses and subclasses
  - Type of reference for data storage
  - Names of keys for hashes
  - 'Strong' encapsulation isn't even an option

Copyright © 2006 David A. Golden

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

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

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

- Accesors 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();
```

- Accessor proliferation is not best practice
  - Private need for accessors shouldn't drive public interface design
  - Couples implementation and interface
  - *'Objects as structs'* – thinking about objects as just data structures with accessors

Copyright © 2006 David A. Golden

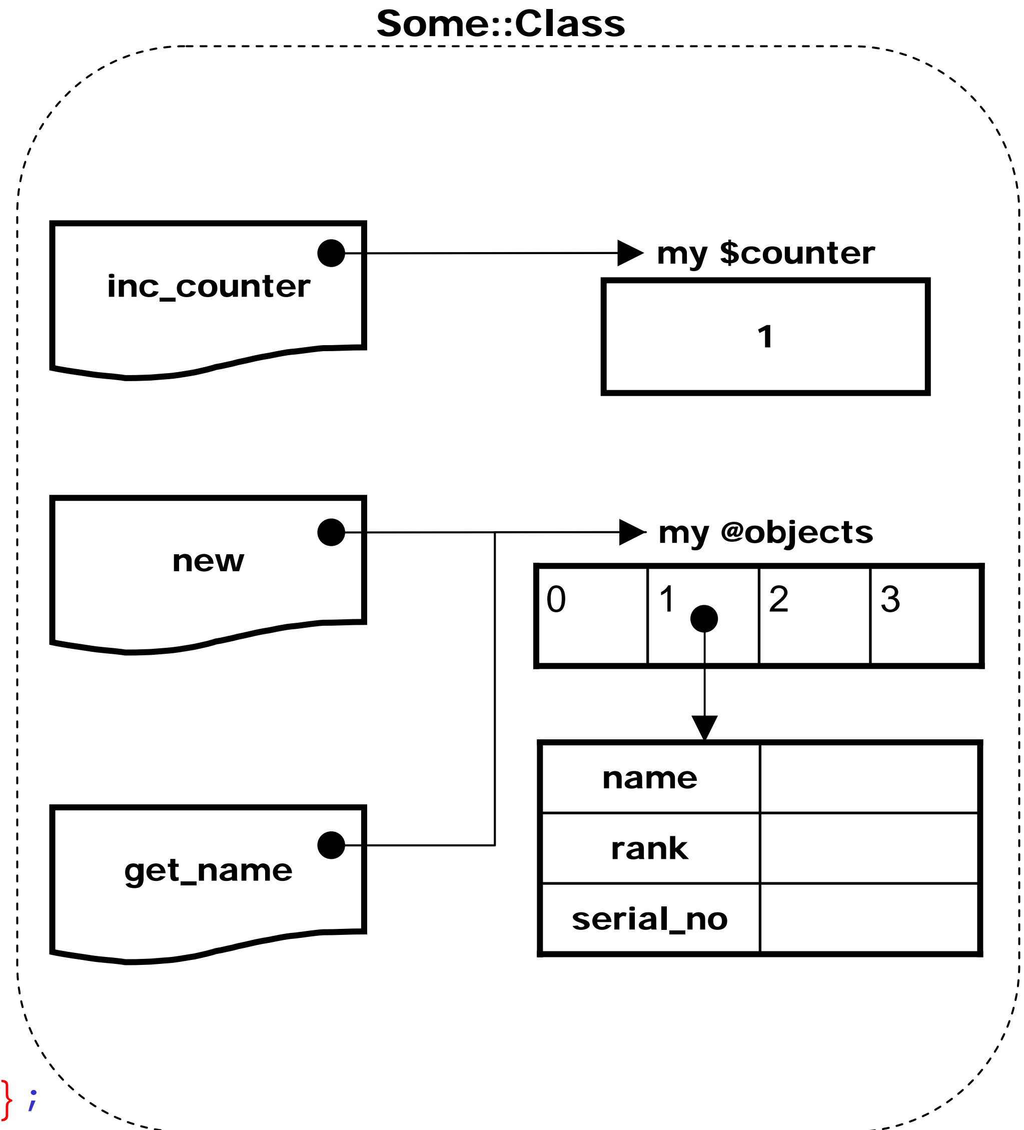
Eureka! We can enforce encapsulation with lexical closure!

- Class properties always did this

```
package Some::Class;

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

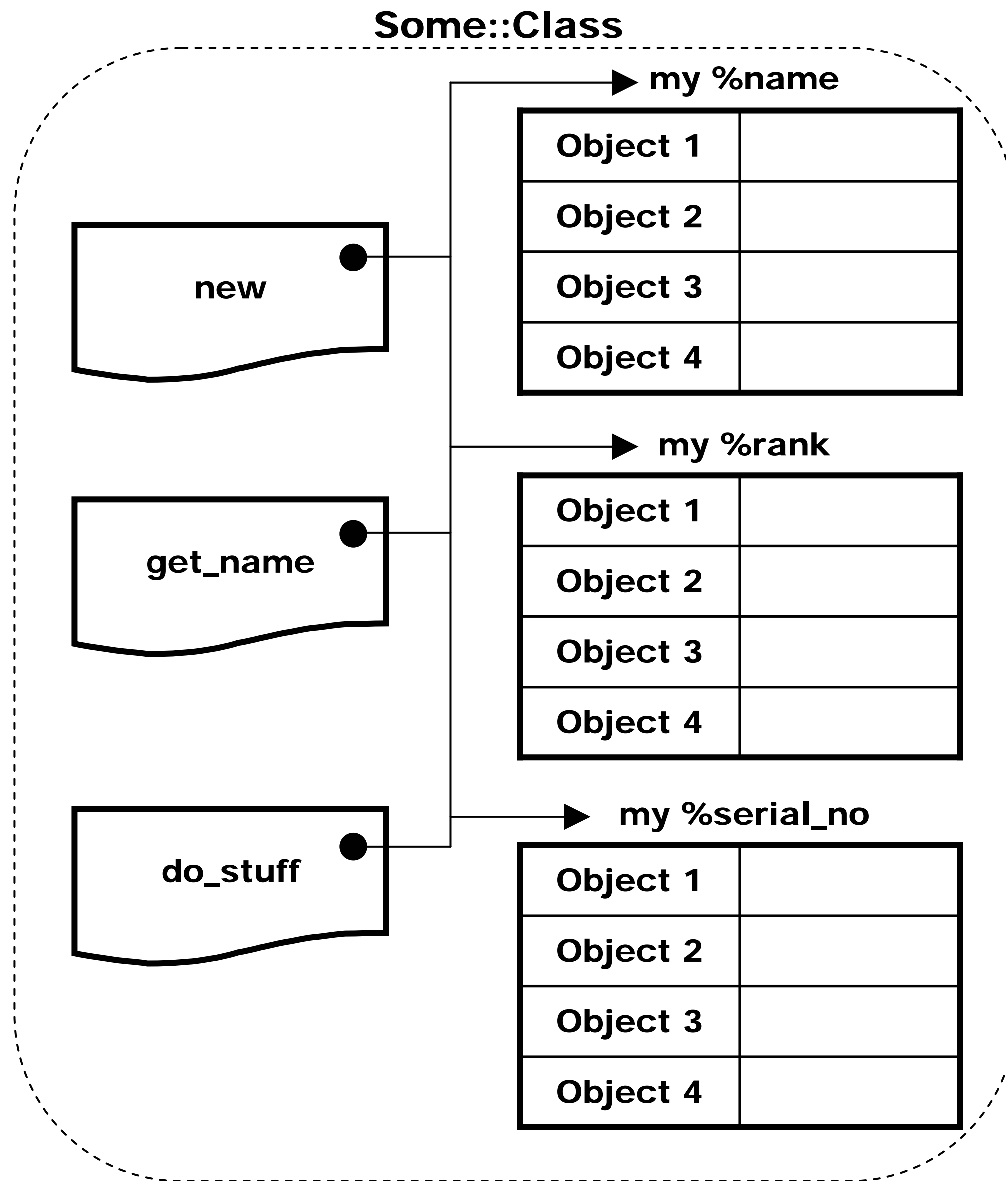
- Damian Conway's "flyweight pattern"

```
my @objects;

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

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



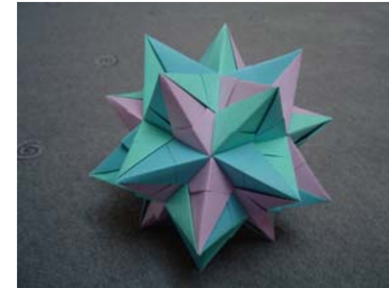
Copyright © 2006 David A. Golden

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



Lexical properties give compile-time typo checking!

```
$name{ $$self };
$naem{ $$self }; # Error!
```

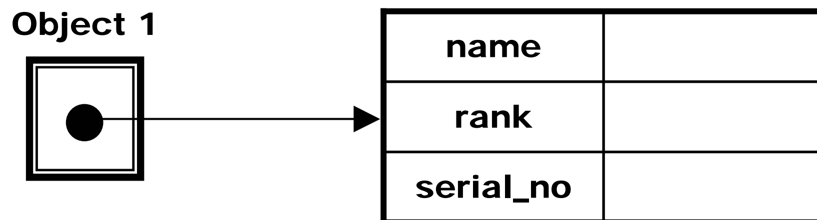


Copyright © 2006 David A. Golden

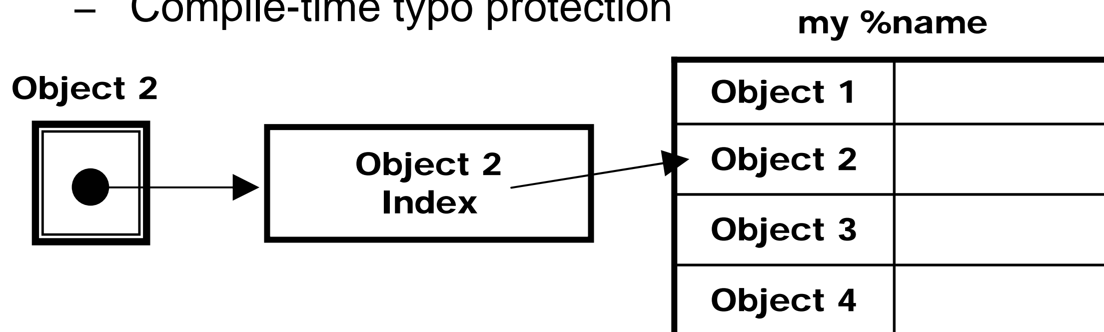
Image source: Michal Kosmulski

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



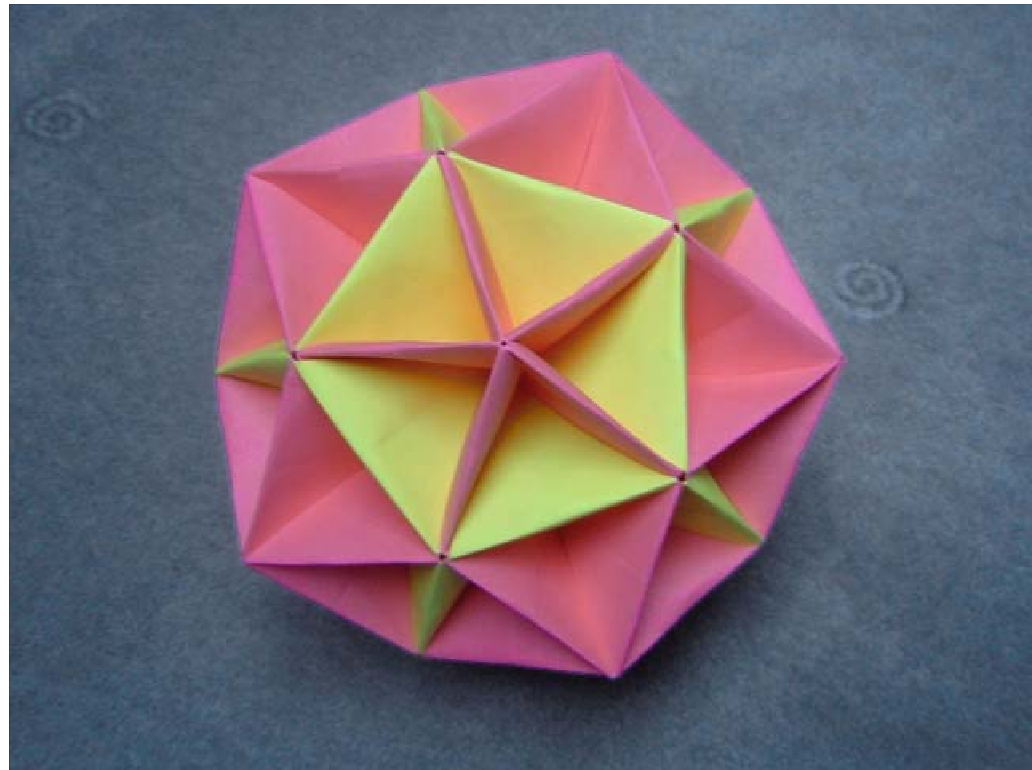
- Inside-Out: Objects as indices
  - Object is an index into a lexical data structure for each property
  - Enforced encapsulation via lexical closure
  - Compile-time typo protection



Copyright © 2006 David A. Golden

Image source: Michal Kosmulski

Choices



Copyright © 2006 David A. Golden

Image source: Michal Kosmulski

What data structure to use for inside-out properties?

- Array
  - Fast access
  - Index limited to sequential integers
  - Requires 'recycling' of indices to prevent undue memory growth of property arrays

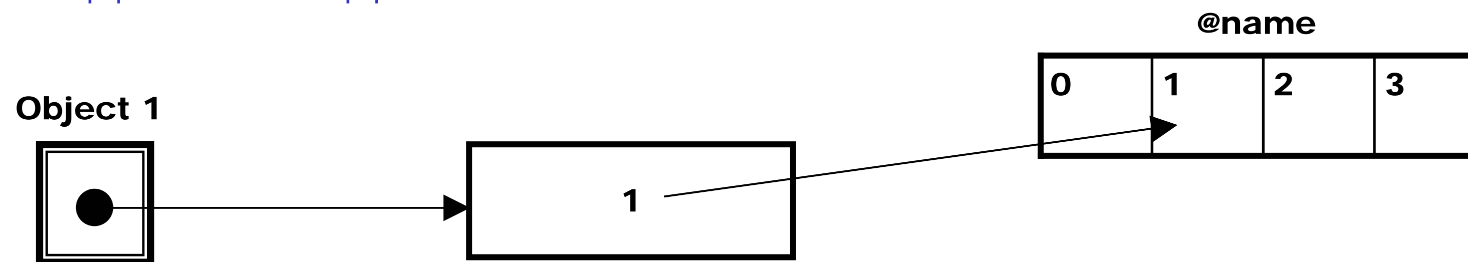
- Hash
  - Slow(er) access
  - Any string as index
  - Uses much more memory (particularly if keys are long)
  - Keys for destroyed objects must be deleted to avoid memory leakage

Copyright © 2006 David A. Golden

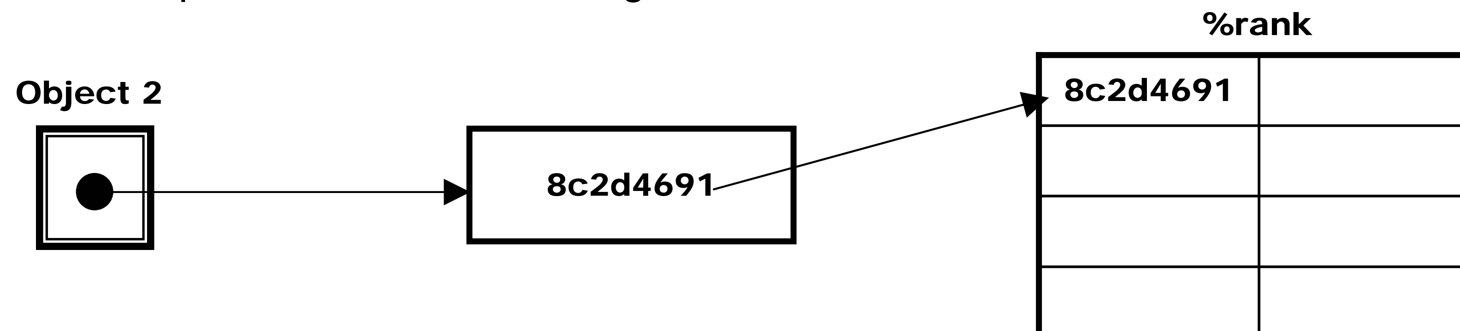
What index? (And stored how?)

- Sequential number, stored in a blessed scalar
  - No encapsulation for subclassing – 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
```



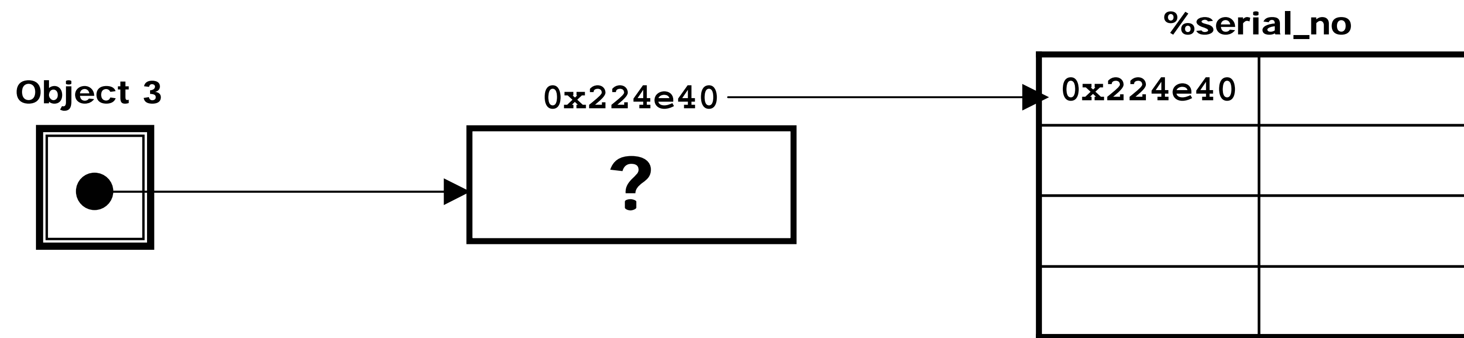
- A unique and hard-to-guess number, stored in a blessed scalar (e.g. with `Data::UUID`)
  - No encapsulation for subclassing – subclasses must also use a blessed scalar



Copyright © 2006 David A. Golden

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 `Scalar::Util::refaddr()` is safe

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

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

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

Copyright © 2006 David A. Golden

Using the memory address directly allows 'foreign inheritance'

- When used directly as `refaddr $self`, *the type of blessed reference no longer matters*
  - The reference has no bearing on inside-out properties
  - Subclasses don't care what the superclass is using as a data type
  - Downside is cost of calculating `refaddr $self` for each access

- *Foreign inheritance* – blessing a superclass object
  - 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

Copyright © 2006 David A. Golden

Summary of the combinations

✓ 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 foreign 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 foreign inheritance

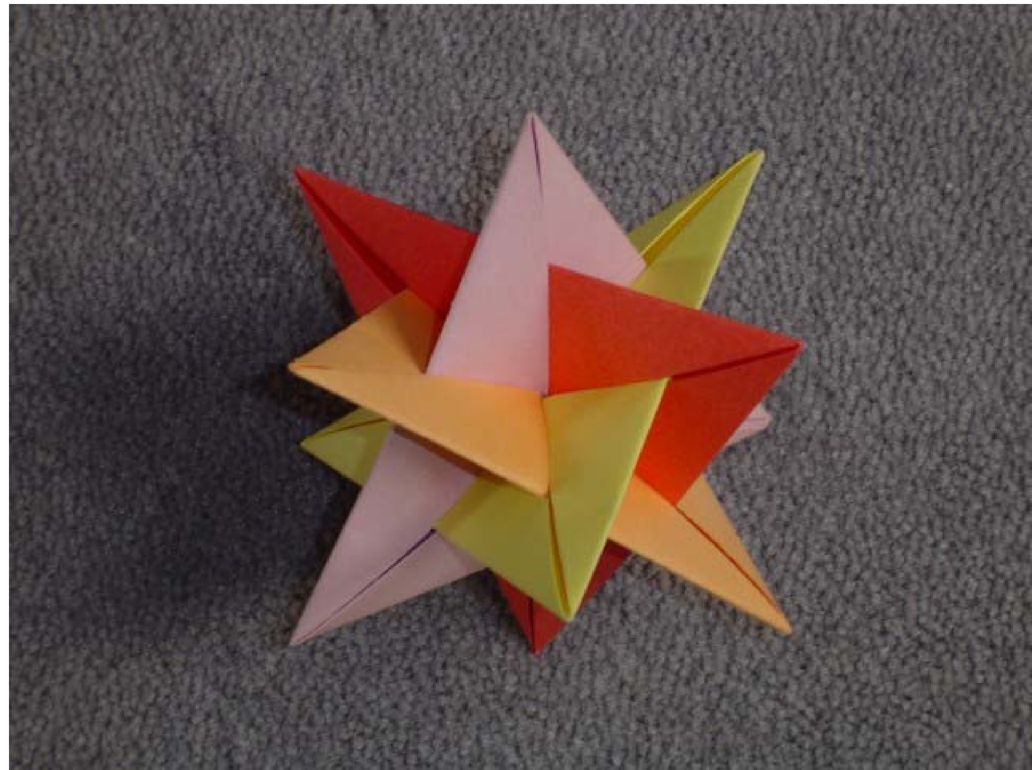
✘ 3. **Hash-based** properties, with the **memory address** stored in a blessed scalar
   - Subclasses must also use a blessed scalar – no foreign inheritance
   - Combines the worst of (2) and (4) for a slight speed increase

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

Copyright © 2006 David A. Golden

Code Example

File::Marker



Copyright © 2006 David A. Golden

Image source: Michal Kosmulski

File::Marker - Set and jump between named position markers

- Useable directly as a filehandle (subclass `IO::File`)

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

- Clear markers when opening a file

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

- 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

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

Copyright © 2006 David A. Golden

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

Copyright © 2006 David A. Golden

File::Marker destructor and methods

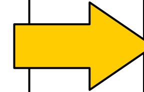
```
sub DESTROY {
    my $self = shift;
    delete $MARKS{ refaddr $self };
}

sub set_marker {
    my ($self, $mark) = @_;
    my $position = $self->getpos;
    $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;
}
```

Copyright © 2006 David A. Golden

Seeing it in action

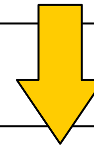
file_marker_example.pl

```
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
```

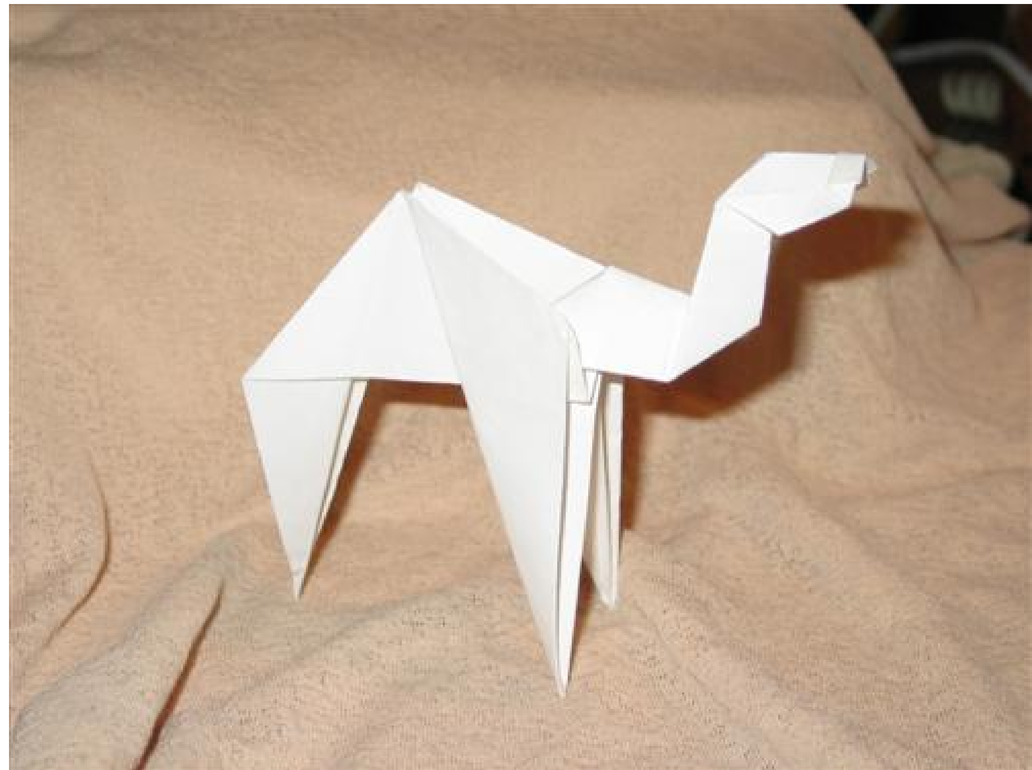


Output

```
$ perl file_marker_example.pl
this is line one

this is line two
this is line three
this is line four

this is line two
```

Copyright © 2006 David A. Golden

CPAN



Copyright © 2006 David A. Golden

Image source: Bill Odom

Two CPAN modules to consider and several to avoid

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

✓ ▪ `Class::InsideOut` (disclaimer: mine and still somewhat experimental)
  - A safe, simple, minimalist approach
  - Manages inside-out complexity but leaves all other details to the user
  - Supports foreign inheritance directly

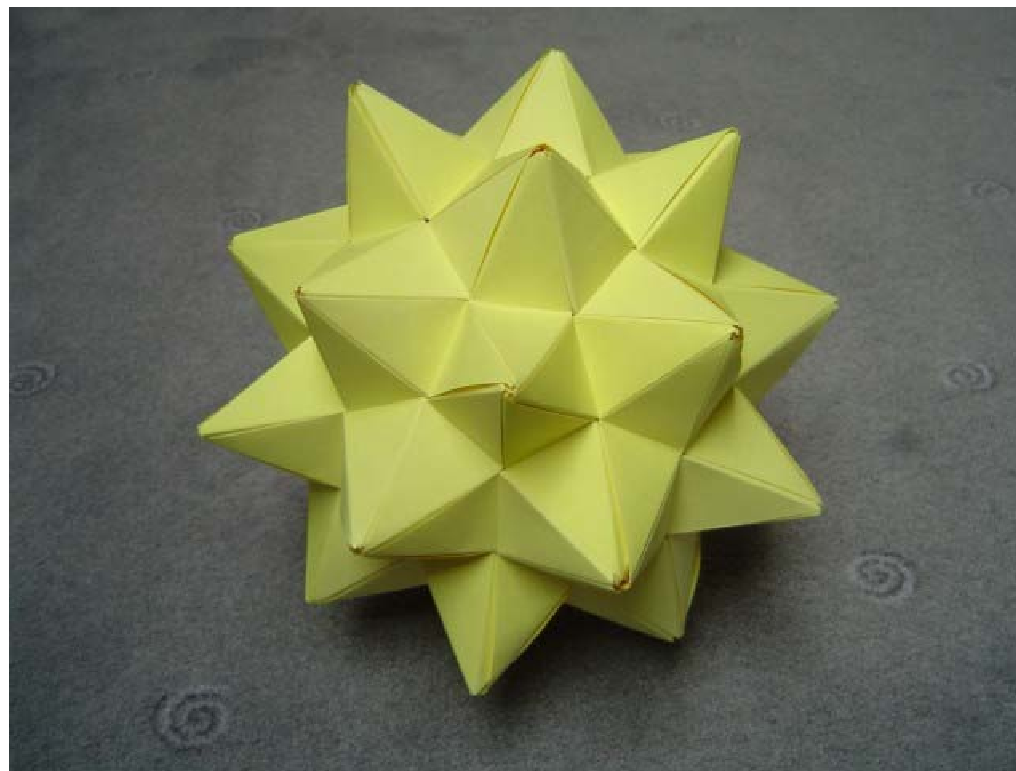
✘ ▪ All of these have flaws or major limitations:
  - `Class::BuildMethods`
  - `Class::Std`
  - `Class::MakeMethods::Templates::InsideOut`
  - `Lexical::Attributes`
  - `Object::LocalVars`

Copyright © 2006 David A. Golden

Questions?

Copyright © 2006 David A. Golden

Bonus Slides

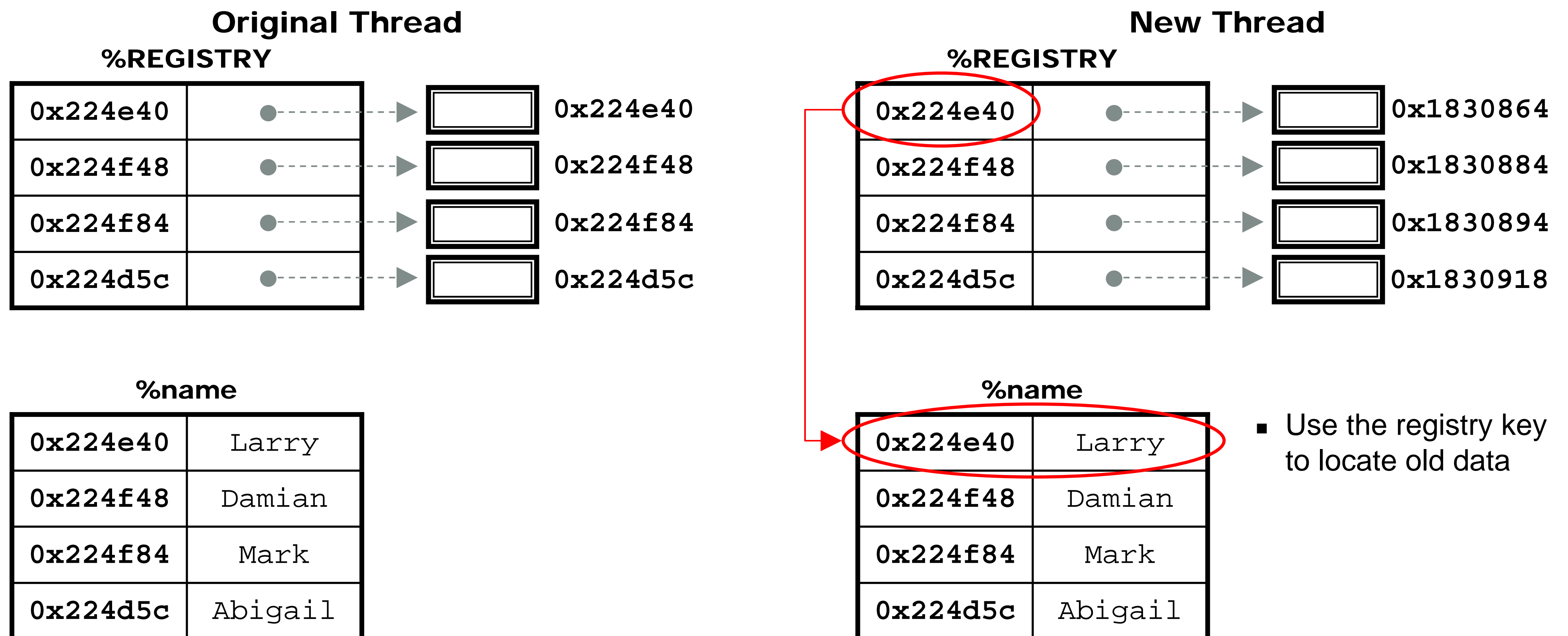


Copyright © 2006 David A. Golden

Image source: Michal Kosmulski

Use CLONE for thread-safe refaddr indices

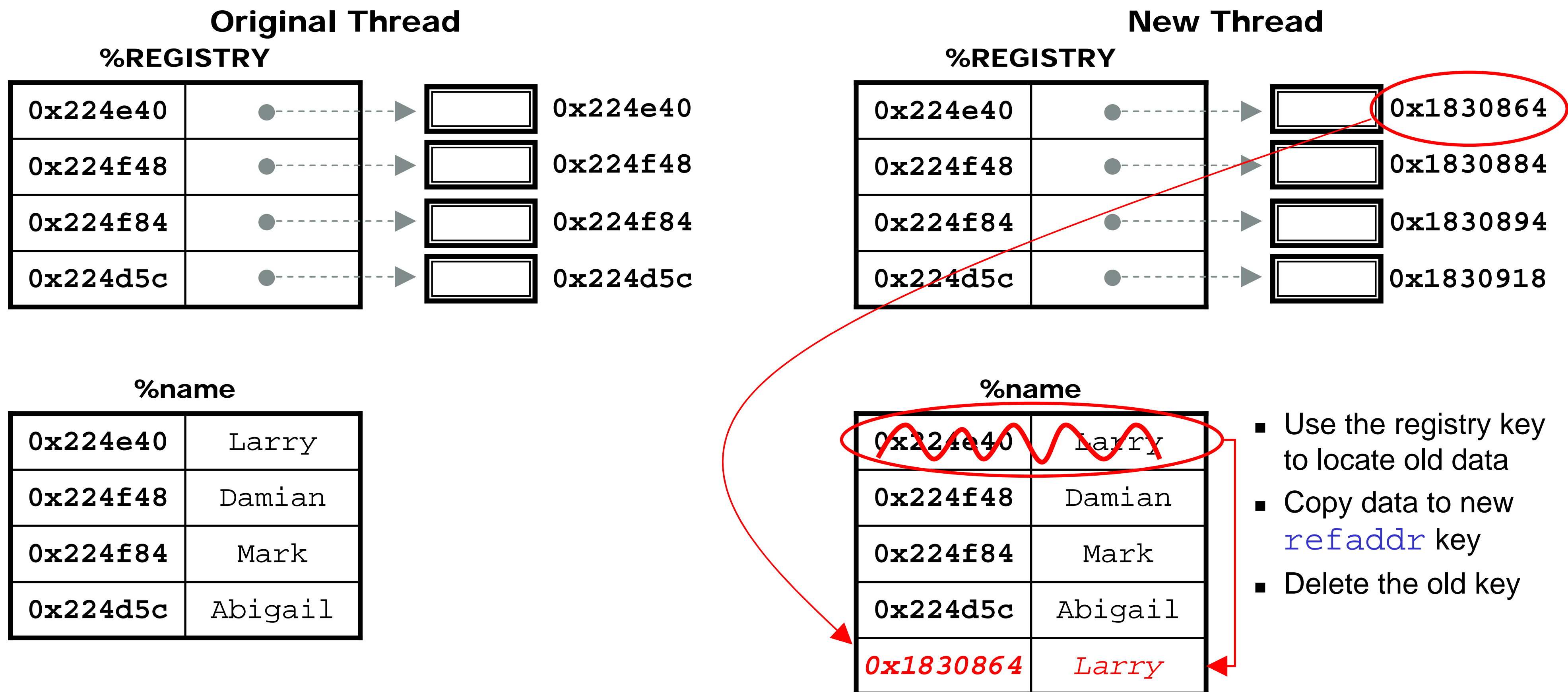
- Starting with Perl 5.8, thread creation calls CLONE once per package, if it exists
  - Called from the context of the *new* thread
  - Works for Win32 pseudo-forks (but not for Perl 5.6)
- Use a registry with weak references to track and remap old indices
  - weaken provided by the XS version of Scalar::Util



Copyright © 2006 David A. Golden

Use CLONE for thread-safe refaddr indices

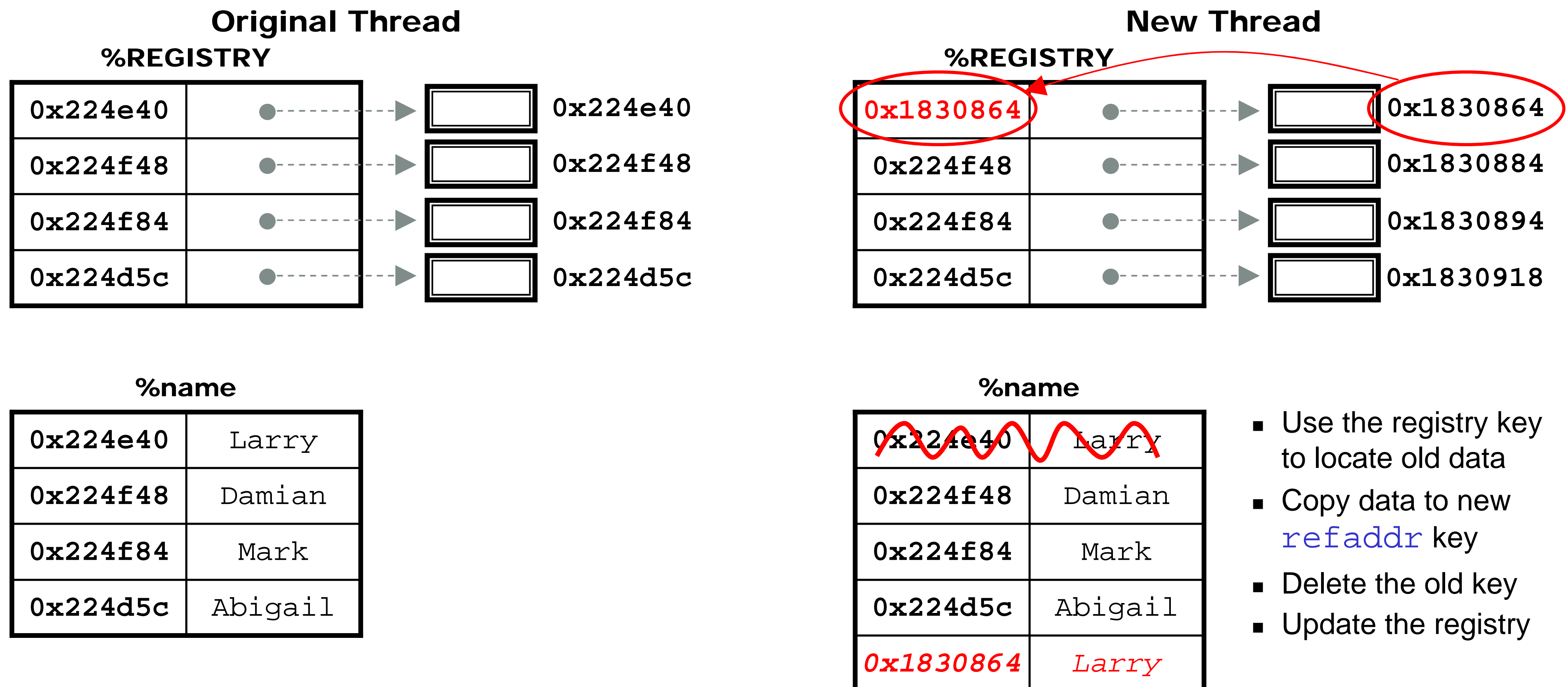
- Starting with Perl 5.8, thread creation calls CLONE once per package, if it exists
  - Called from the context of the *new* thread
  - Works for Win32 pseudo-forks (but not for Perl 5.6)
- Use a registry with weak references to track and remap old indices
  - weaken provided by the XS version of Scalar::Util



Copyright © 2006 David A. Golden

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



Copyright © 2006 David A. Golden

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 };
}
```

Copyright © 2006 David A. Golden

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;
}
```

Copyright © 2006 David A. Golden

Reference

Copyright © 2006 David A. Golden

Image source: Michal Kosmulski

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	▪ Foreign inheritance using delegation pattern ▪ Custom :attribute handling ▪ mod_perl safe ▪ Good thread support
Class::InsideOut (0.07)	Hash	refaddr $self	Yes	Storable hooks	▪ Simple, minimalist approach ▪ Supports direct foreign inheritance ▪ mod_perl safe ▪ Still somewhat experimental
Class::Std (0.0.4)	Hash	refaddr $self	No	Storable hooks with Class::Std:: Storable	▪ Custom :attribute handling; breaks under mod_perl ▪ No foreign inheritance support ▪ Rich class hierarchy support

Copyright © 2006 David A. Golden

Inside-out CPAN module comparison table (continued)

Module	Storage	Index	CLONE?	Serializes?	Other Notes
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
Lexical::Attributes (1.4)	Hash	refaddr $self	No	No	▪ Source filters for Perl-6-like syntax
Class::MakeMethods ::Template::InsideOut (1.01)	Hash	"$self"	No	No	▪ Part of a complex class generator system; steep learning curve
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

Copyright © 2006 David A. Golden

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

Copyright © 2006 David A. Golden

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

Copyright © 2006 David A. Golden