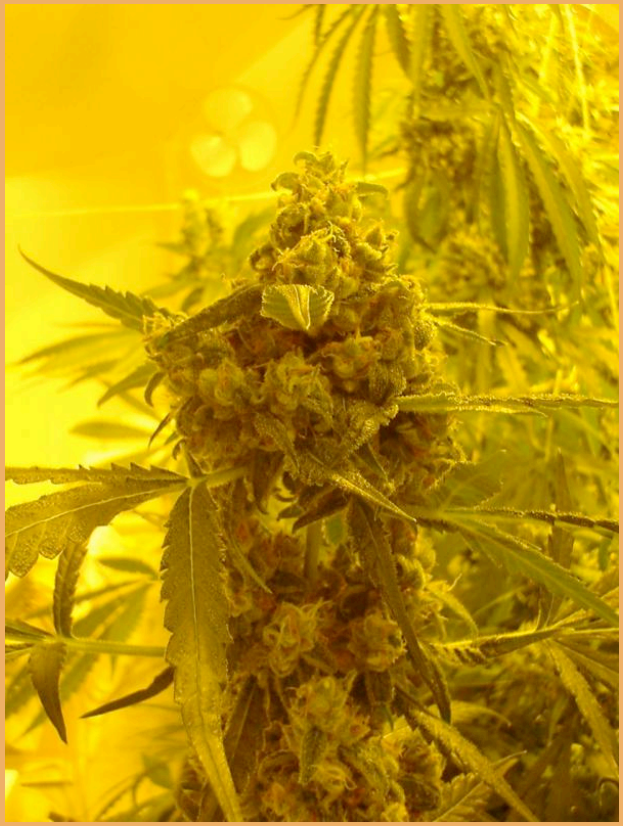
Hash Crash Course

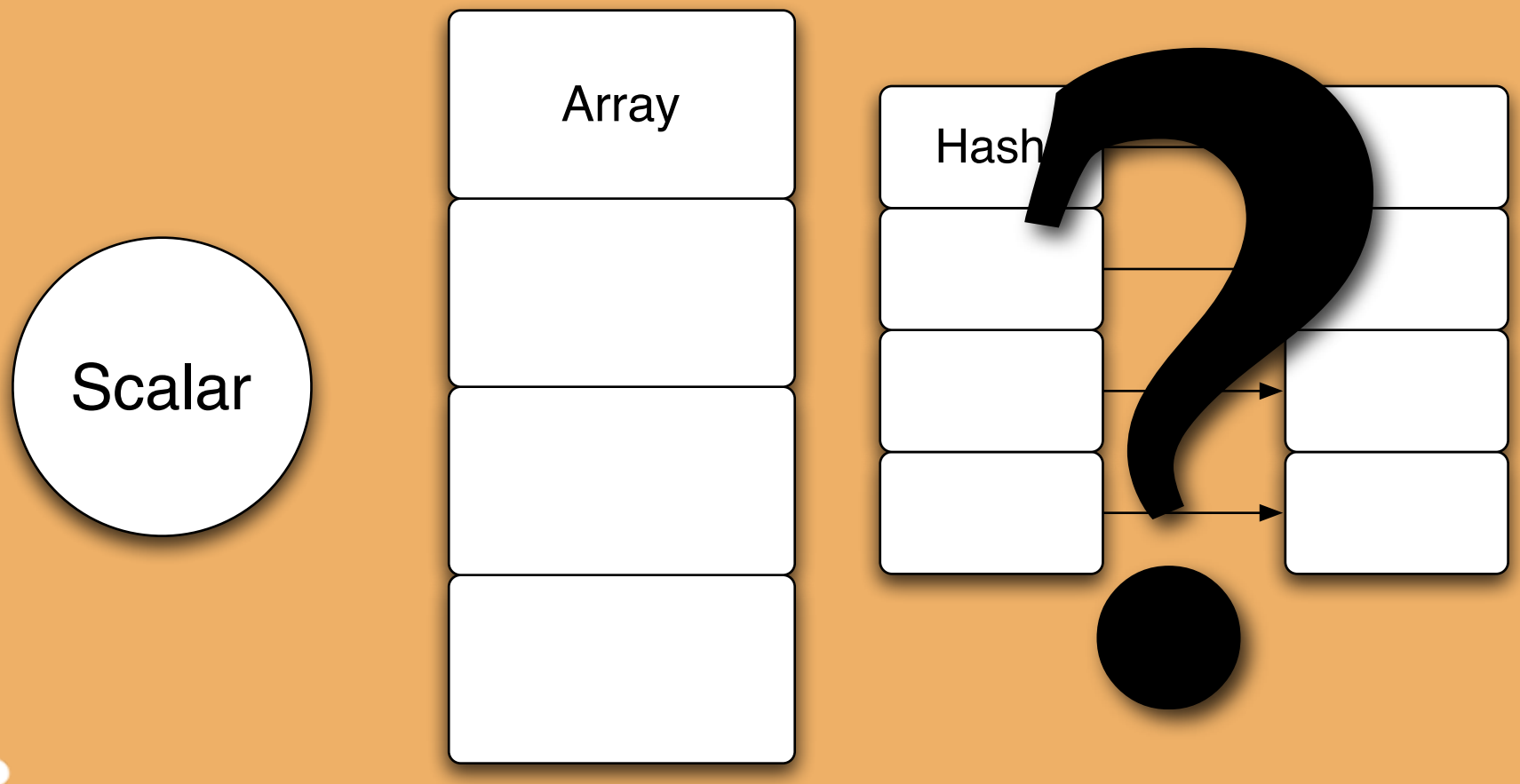
Simon Cozens

Let's just get this out of the way



And now, on with the talk

Perl's Three Data Structures



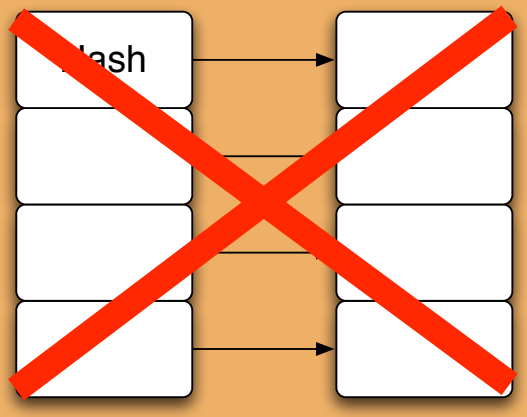
"A Dictionary"

'the hash is a "dictionary", a mapping between one thing and another'

```
%french = (
    apple  => "pomme",
    pear   => "poire",
    orange => "Léon Brocard"
);

$french{"apple"} # "pomme"
```

But...

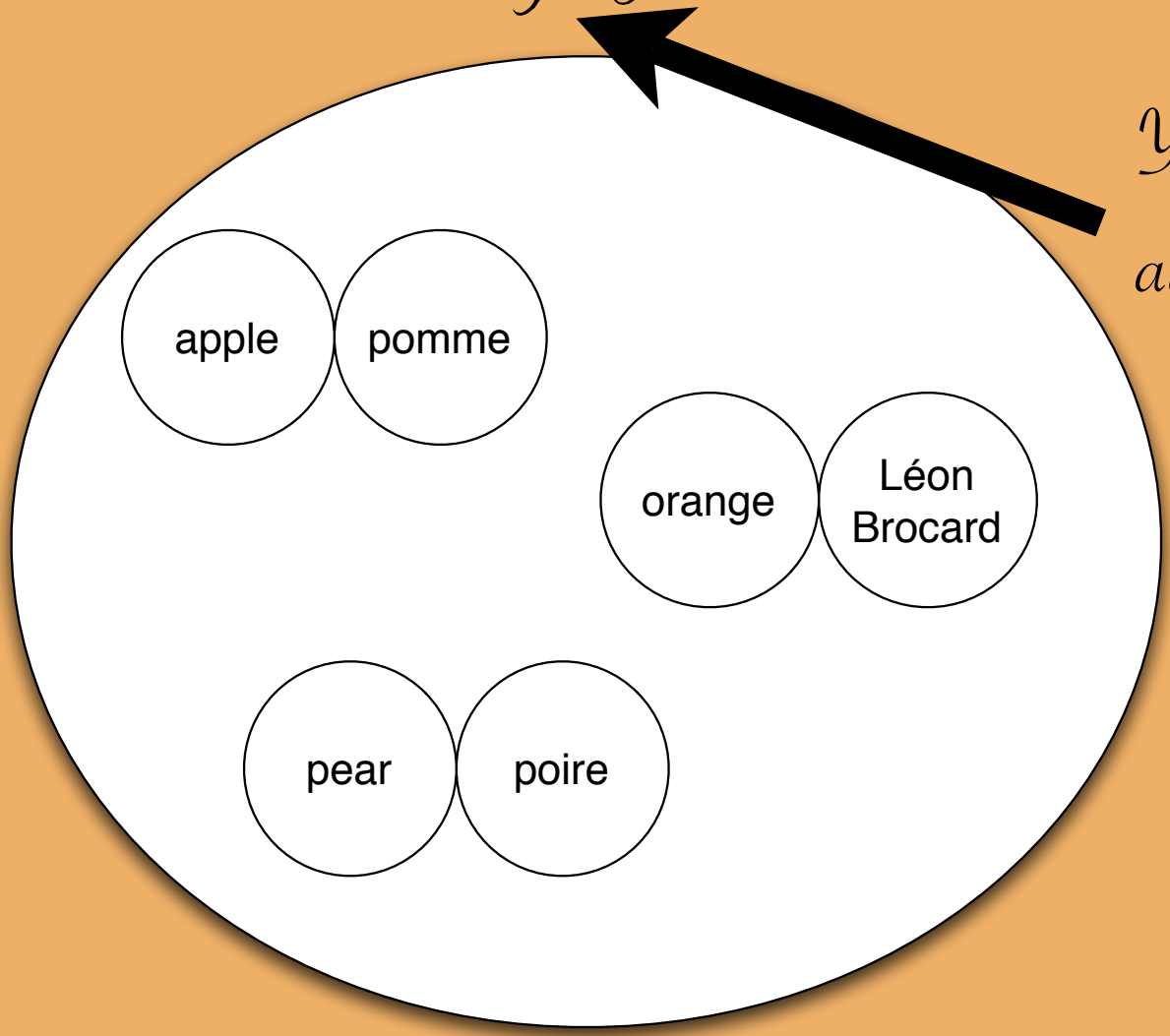


- It doesn't preserve order
- That doesn't tell us much
- We hardly ever use hashes like that anyway

Exceptions...

- *There will be many*
- *Command line arguments*
- `$args{“help”}`

"A Bag of Pairs"



You could also call it a

set

Hashes in Real Life

Hashes are mainly used for

"answering questions about lists"

That means...

- Counting
- Uniqueness
- Caching
- Searching
- Dispatch tables

Counting

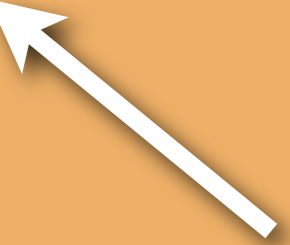
```
my $apples = 0;
for (@list) {
    $apples++ if $_ eq "apple";
}
```

Counting

```
my $apples = 0;
my $pears = 0;
for (@list) {
    $apples++ if $_ eq "apple";
    $pears++  if $_ eq "pear";
}
```

Hmm....

```
my $apples = 0;
my $pears = 0;
for (@list) {
    ${$_}++;
}
```



The list contains `*`

This sets `$* = 1`

None of your regexes match

It takes days to debug

Oh, the embarrassment

A Histogram

```
my %histogram;
for (@list) {
    $histogram{$_}++;
}
```

Who cares if `$histogram{"*"} = 1`

General Principle

Replace a

set of related variables

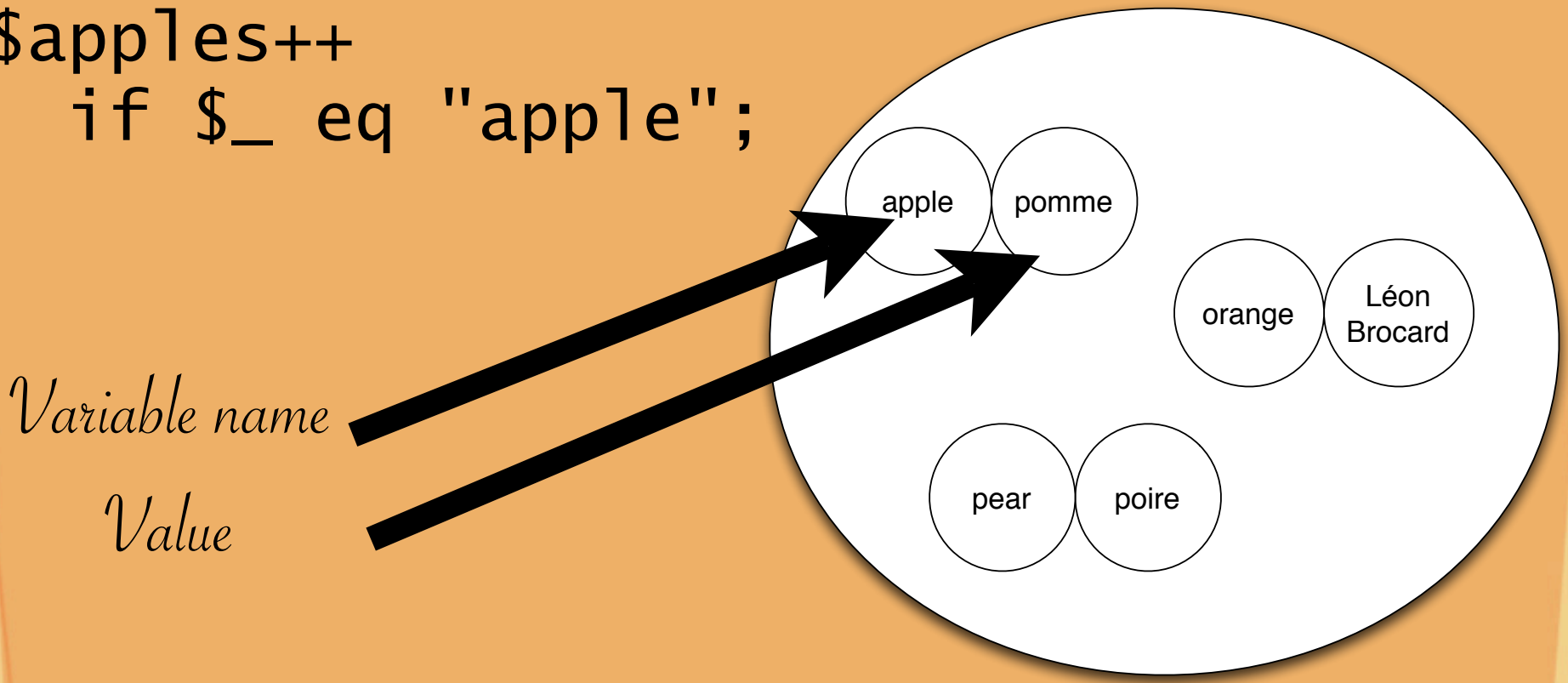
with a hash

A hash is a "safe private symbol-table"

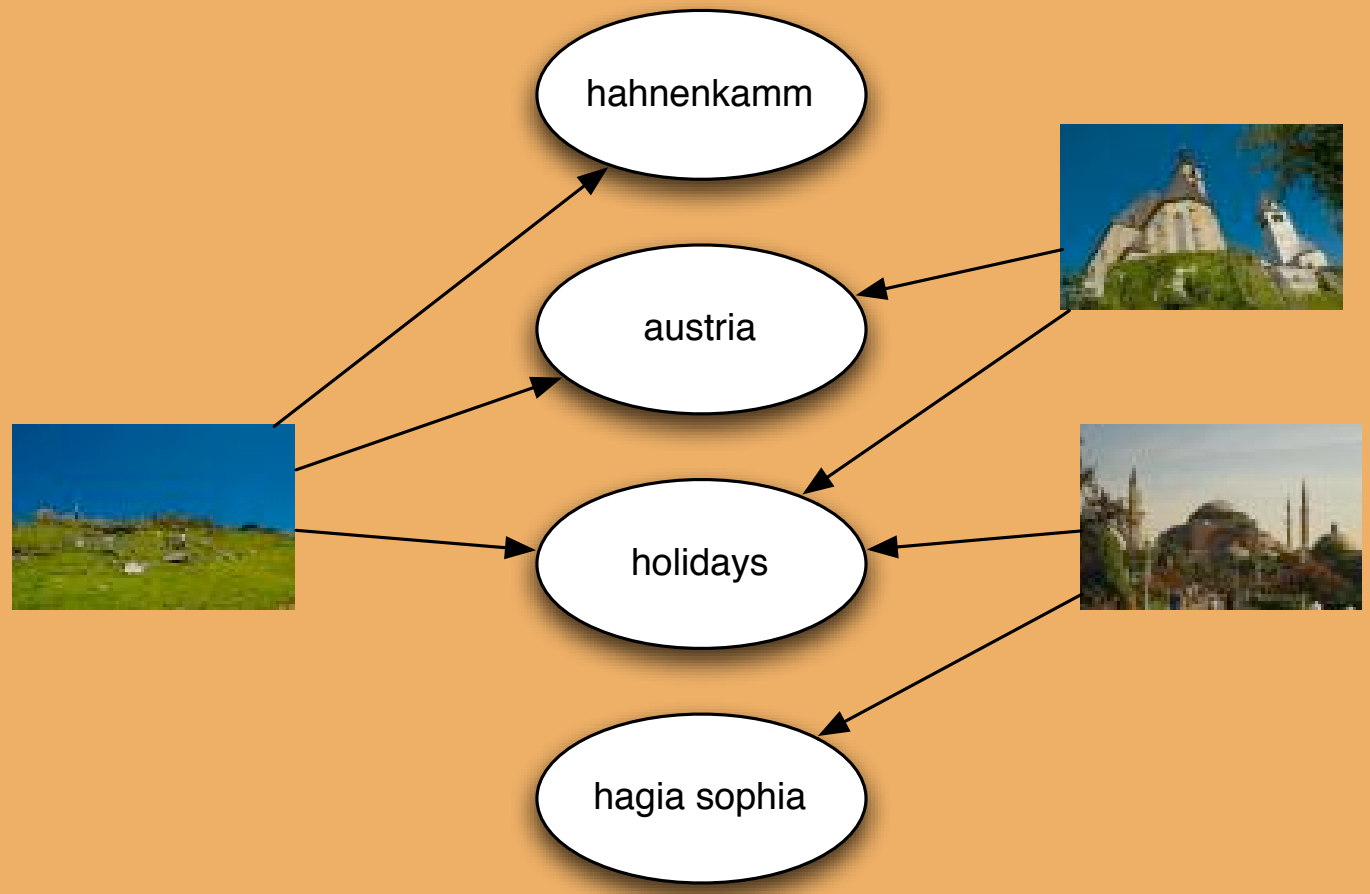
(Actually, a symbol table is an unsafe hash...)

Private variables?

```
$apples++
  if $_ eq "apple";
```

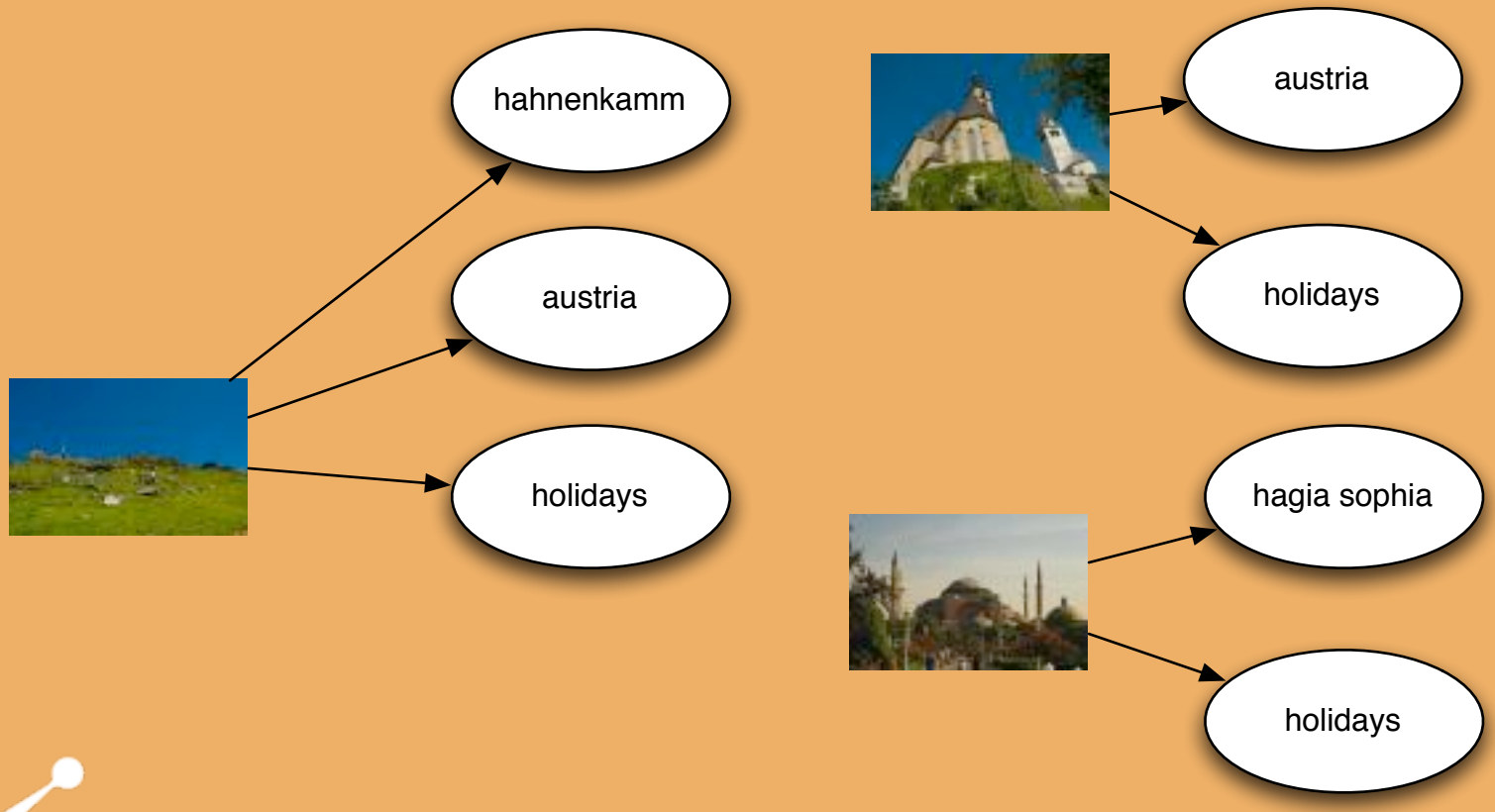


Counting numbers of different things



We ask each photo for its tags...

```
push @tags, $_->tags for @photos
my $count = @tags; # 7
```



We find the set of tags used

```
for my $photo (@photos) {
  $tags{$_}++ for $photo->tags
  # $tags{“austria”}
  # $tags{“holidays”}, etc.
}

my $count = keys %tags; # 4
```

A "question about a list"

```
# How many times was Austria tagged?

my $count = $tags{“Austria”}; # 2
```

Uniqueness

Another "question about a list" - what were the unique tags?

```
for my $photo (@photos) {
  $tags{$_}++ for $photo->tags
}

my @unique = keys %tags;
```

Combining the two - Popularity

Another "question about a list" - what were the most popular tags?

```
for my $photo (@photos) {
   $tags{$_}++ for $photo->tags
}
```

How many times

```
my @top_five = (
    sort { $tags{$b} <=> $tags{$a} }
      keys %tags
    )[0..4];
```

Unique tags

Uniqueness revisited

```
for (@list) { $unique{$_}++ }
@unique = keys %unique;
```

```
for (@list) { $unique{$_} = 1 }
@unique = keys %unique;
```

```
%unique = map { $_ => 1 } @list;
@unique = keys %unique;
```

Actually I lied...

Tags are objects

```
my @tags;
push @tags, Memories::Tag->retrieve_random
  for 1..10;
%unique = map { $_ => 1 } @tags;
@unique = keys %unique;

print $unique[0]->name;
# Can't locate object method "test" via
# package "Memories::Tag(0x1801380)"
```

The solution

Map the name to the tag

```
%unique = map { $_->name => $_ } @list;
@unique = values %unique;
```

Then retrieve by value

Have I seen this before?

```
my @tags;
my %seen;
while (@tags < 10) {
    my $candidate =
            Memories::Tag->retrieve_random;
    next if $seen{$candidate->name}++;
    push @tags, $candidate;
}
```

Caching

Have I seen this before?

Caching

Have I done this before?

What was the answer last time?

Caching - Retrieving values

```
my %cache;

sub retrieve {
    my ($self, $id) = @_;
    return $cache{$id} if exists $cache{$id};
    return $cache{$id} = $self->_hard_retrieve($id);
}
```

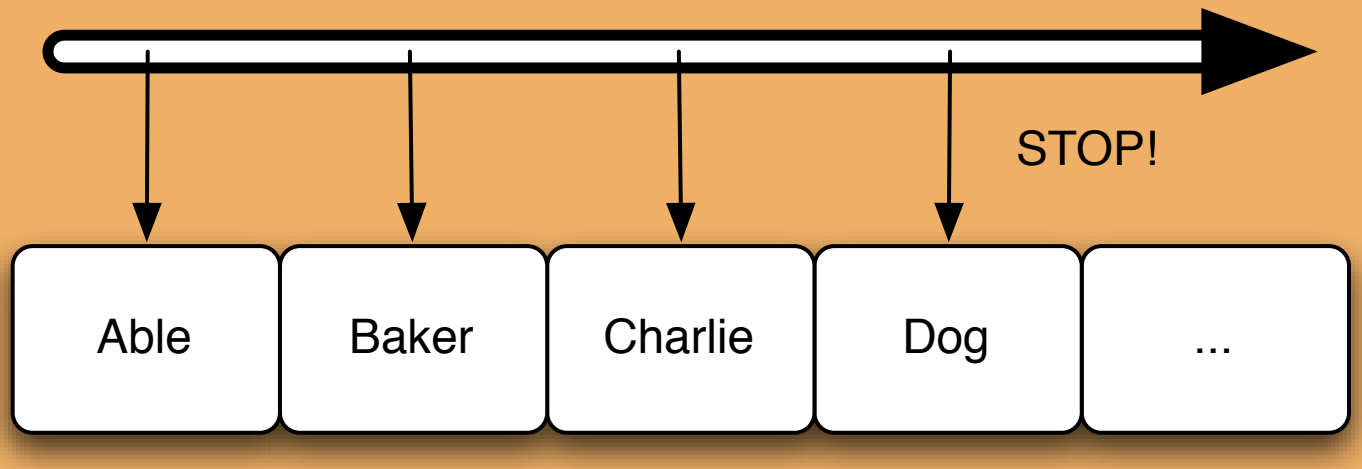
Mind the cache doesn't get full!

Cache::Cache handles all this

See also Memoize.pm

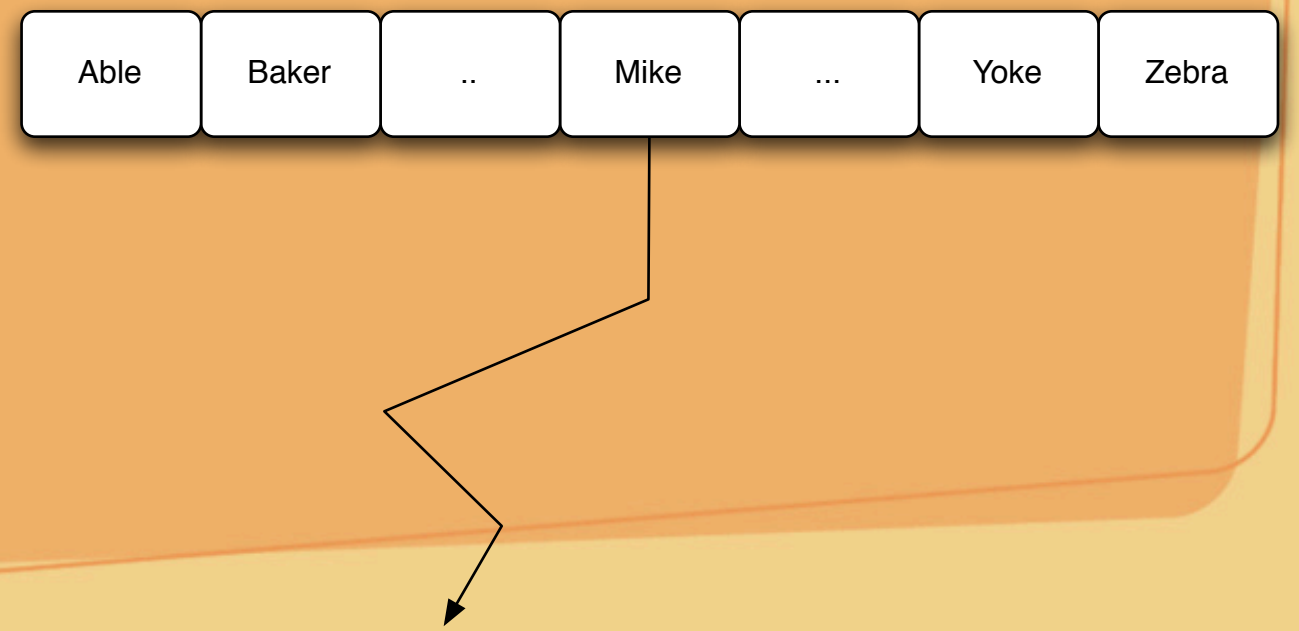
Searching - linear

```
my $index;
for $index (0..@chambers) {
    last if $chambers[$index] eq $bullet;
}
print "Found at index $index"
    if $index < @chambers;
```



Searching - binary

```
my ($lower, $upper) = (0, $#names);
while ($lower <= $upper) {
      my $index = ($lower + $upper) / 2;
      if ($names[$index] lt $target) {
          $lower = $index + 1;
      } elsif ($names[$index] gt $target) {
          $upper = $index - 1;
      } else { return $index }
}
# Not found!
```



I'll name that tune in none, Lionel...

```
my %search = map {
                    $names[$_] => $_
                  } 0..$#names;

print $search{"George"};
```

More To Discover...

- Config files
- Dispatch tables
- More besides
- http://simon-cozens.org/programmer/articles/hashes.pod

Thank you!