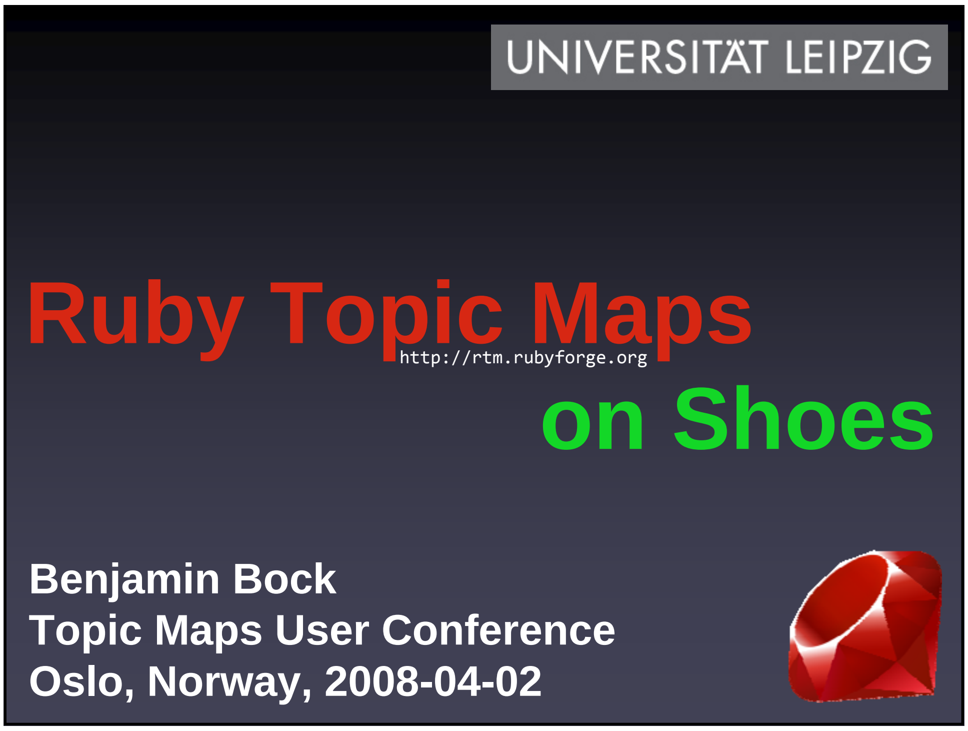


UNIVERSITÄT LEIPZIG

Ruby Topic Maps on Shoes

http://rtm.rubyforge.org

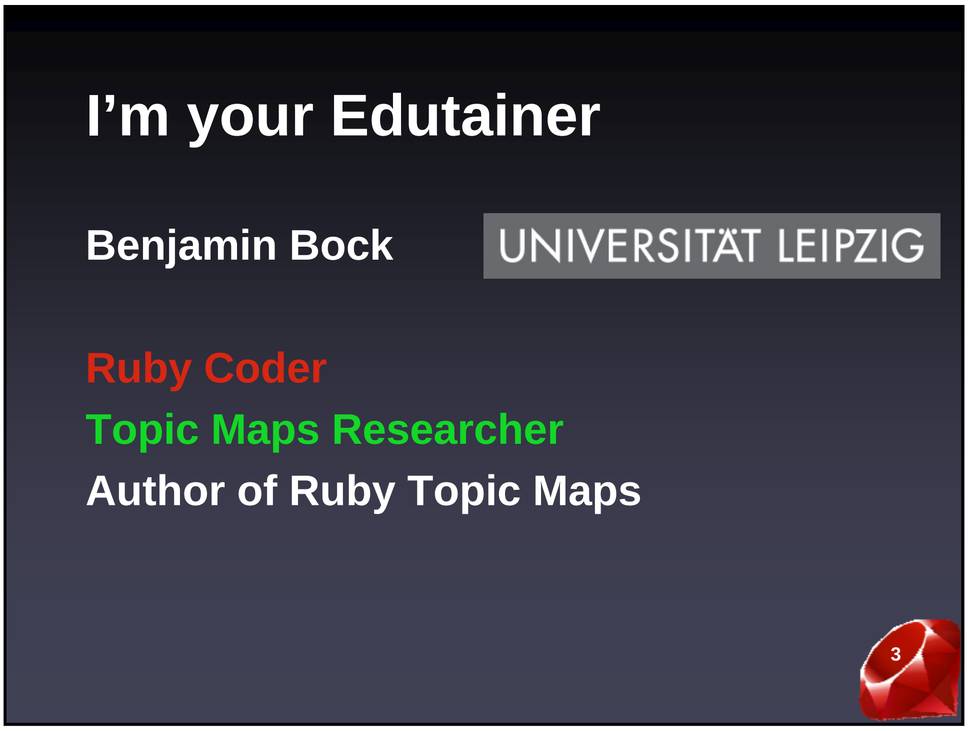
Benjamin Bock
Topic Maps User Conference
Oslo, Norway, 2008-04-02

Whassup today?

Ruby Topic Maps

Shoes, a Tiny Toolkit

2



I’m your Edutainer

Benjamin Bock

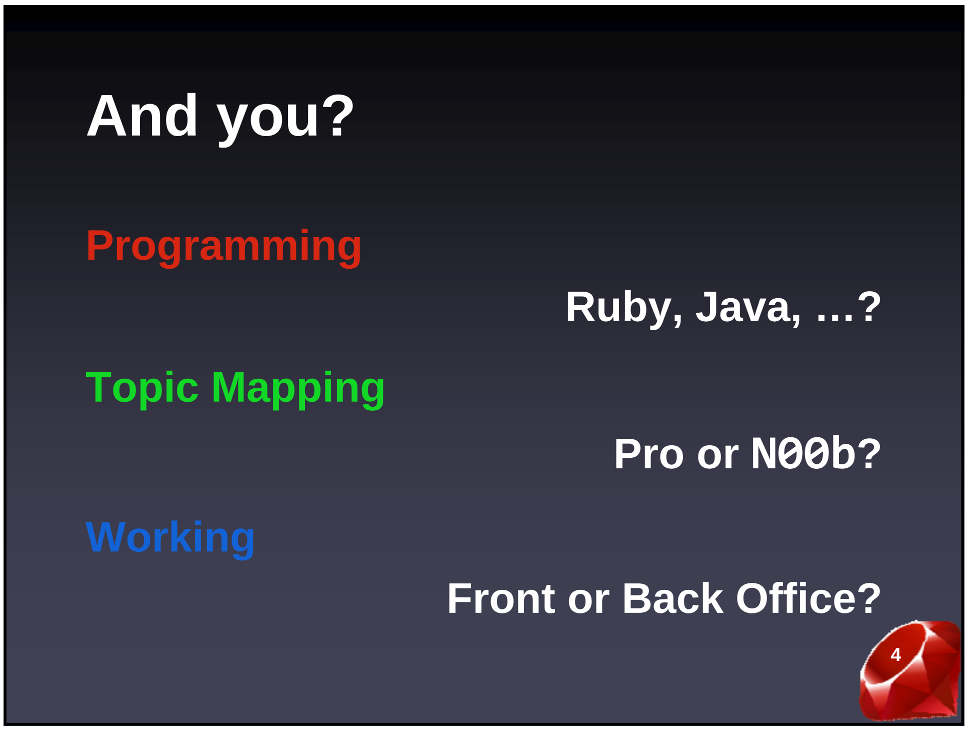
UNIVERSITÄT LEIPZIG

Ruby Coder

Topic Maps Researcher

Author of Ruby Topic Maps

3



And you?

Programming

Ruby, Java, …?

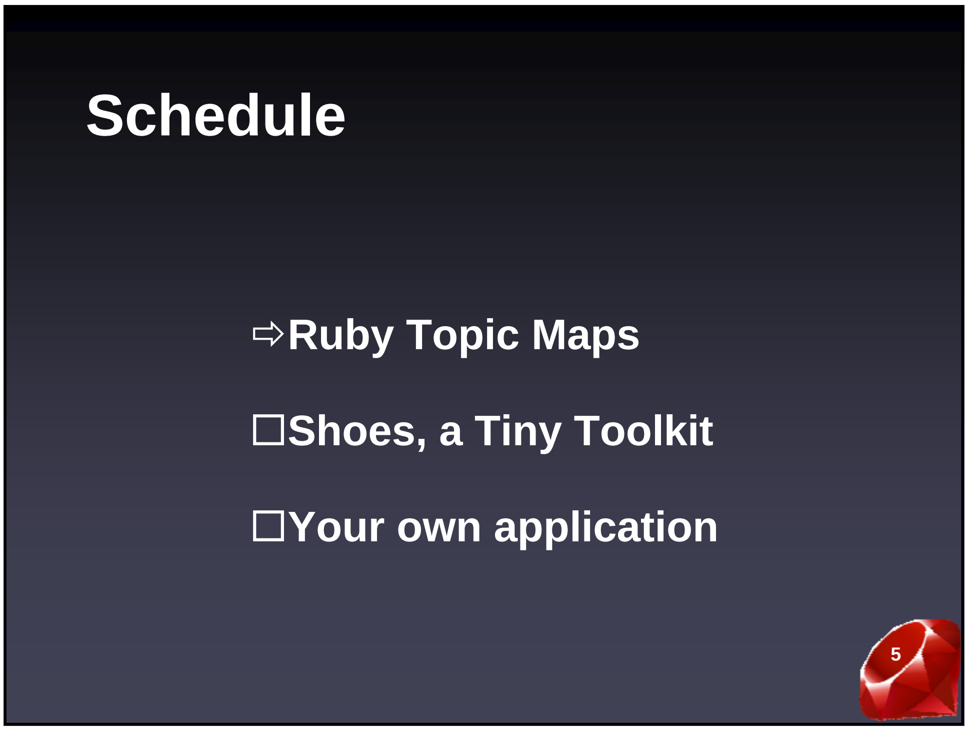
Topic Mapping

Pro or N00b?

Working

Front or Back Office?

4



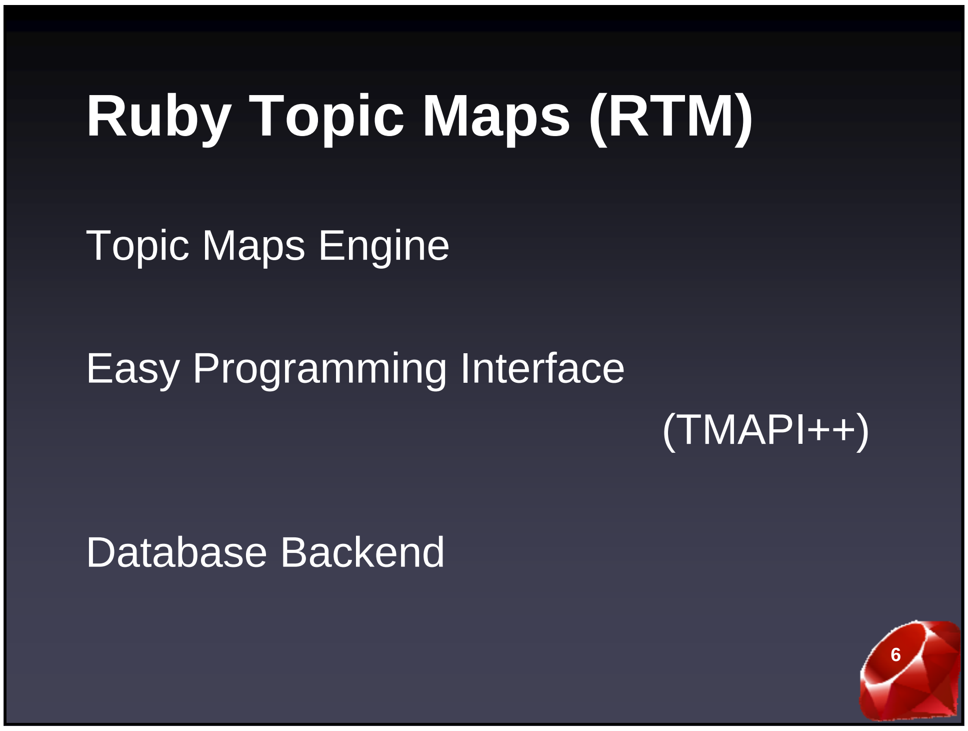
Schedule

⇨Ruby Topic Maps

☐Shoes, a Tiny Toolkit

☐Your own application

5



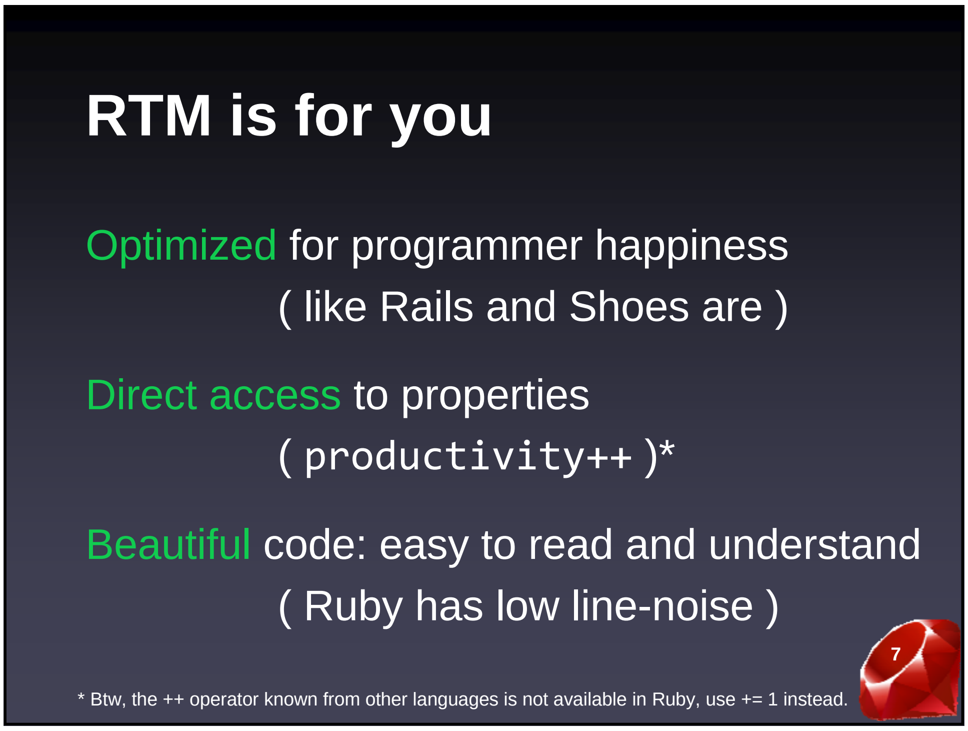
Ruby Topic Maps (RTM)

Topic Maps Engine

Easy Programming Interface
(TMAPI++)

Database Backend

6



RTM is for you

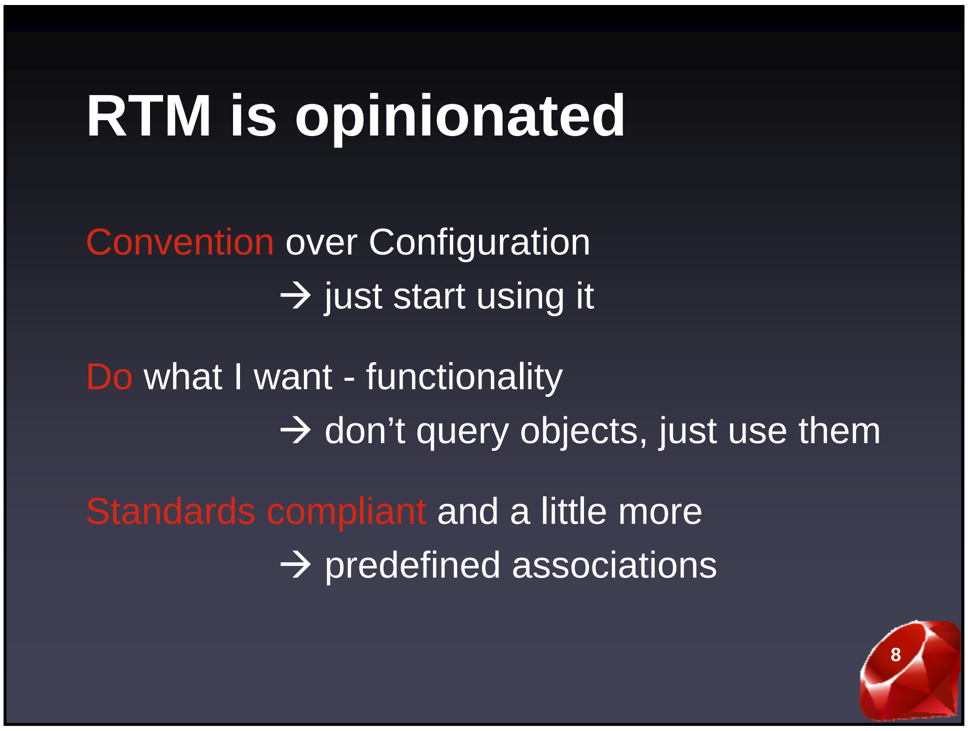
Optimized for programmer happiness
(like Rails and Shoes are)

Direct access to properties
(`productivity++`)*

Beautiful code: easy to read and understand
(Ruby has low line-noise)

* Btw, the ++ operator known from other languages is not available in Ruby, use += 1 instead.

7



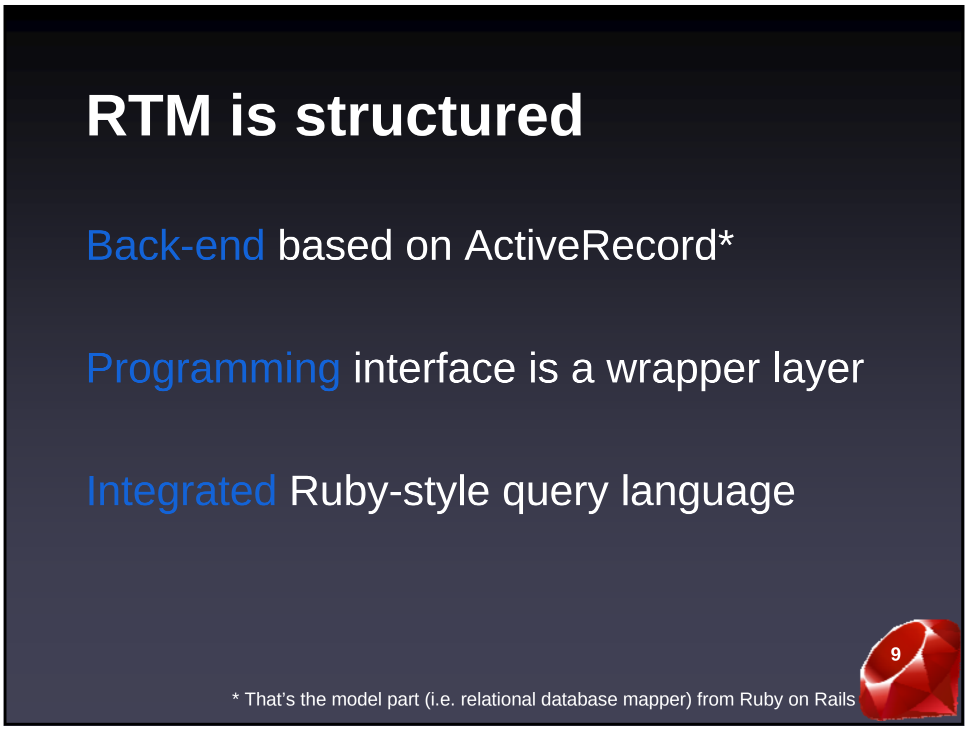
RTM is opinionated

Convention over Configuration
→ just start using it

Do what I want - functionality
→ don't query objects, just use them

Standards compliant and a little more
→ predefined associations

8



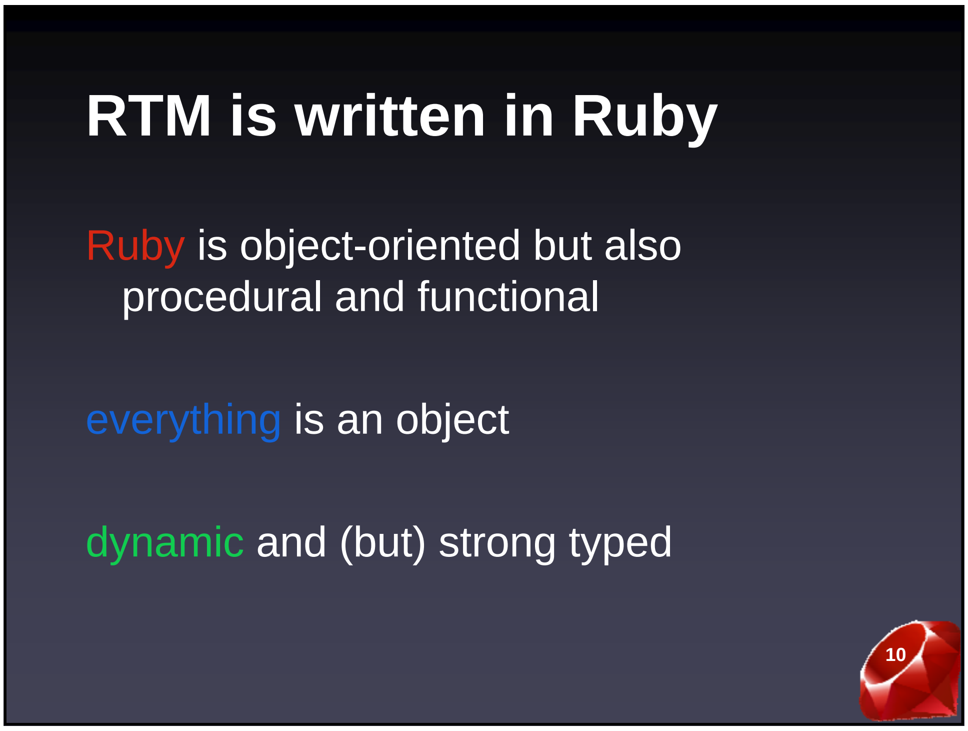
RTM is structured

Back-end based on ActiveRecord*

Programming interface is a wrapper layer

Integrated Ruby-style query language

* That’s the model part (i.e. relational database mapper) from Ruby on Rails

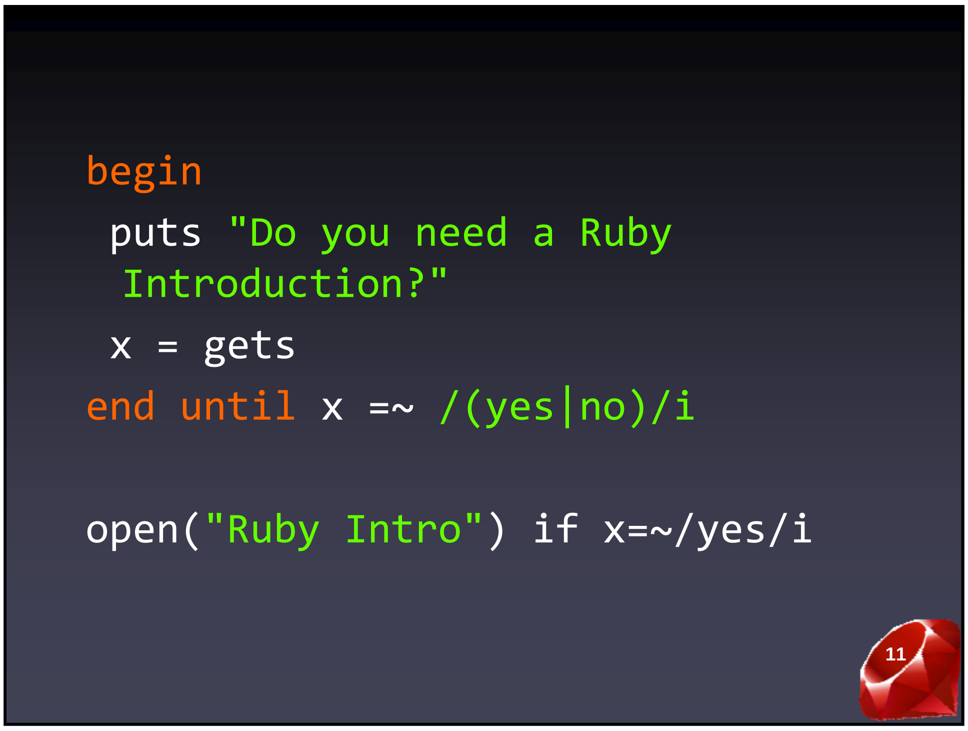


RTM is written in Ruby

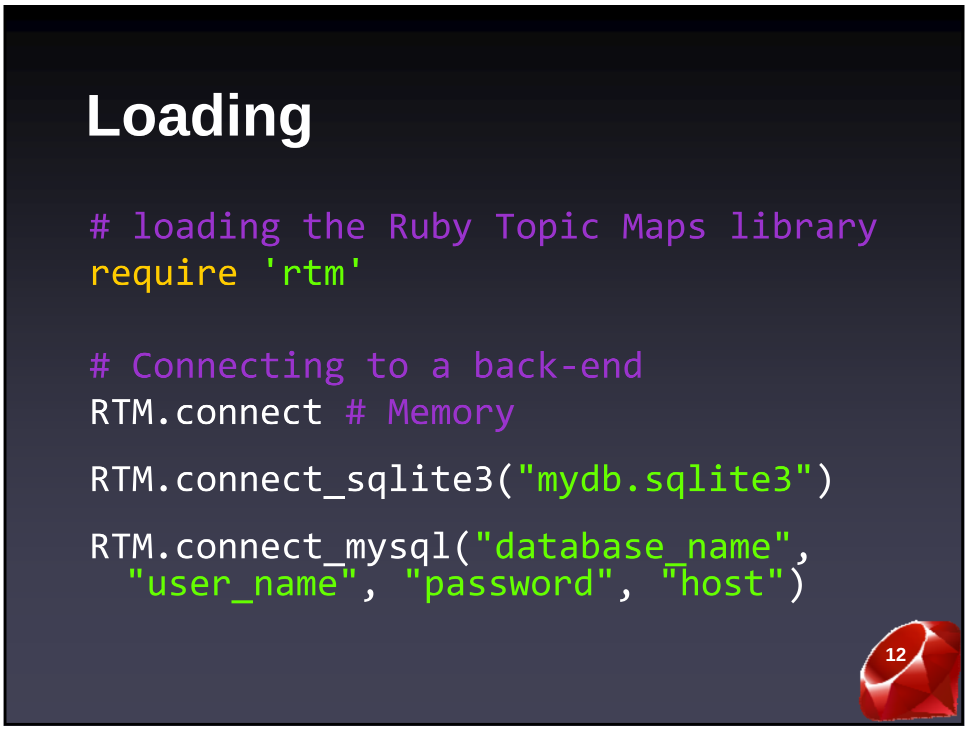
Ruby is object-oriented but also procedural and functional

everything is an object

dynamic and (but) strong typed

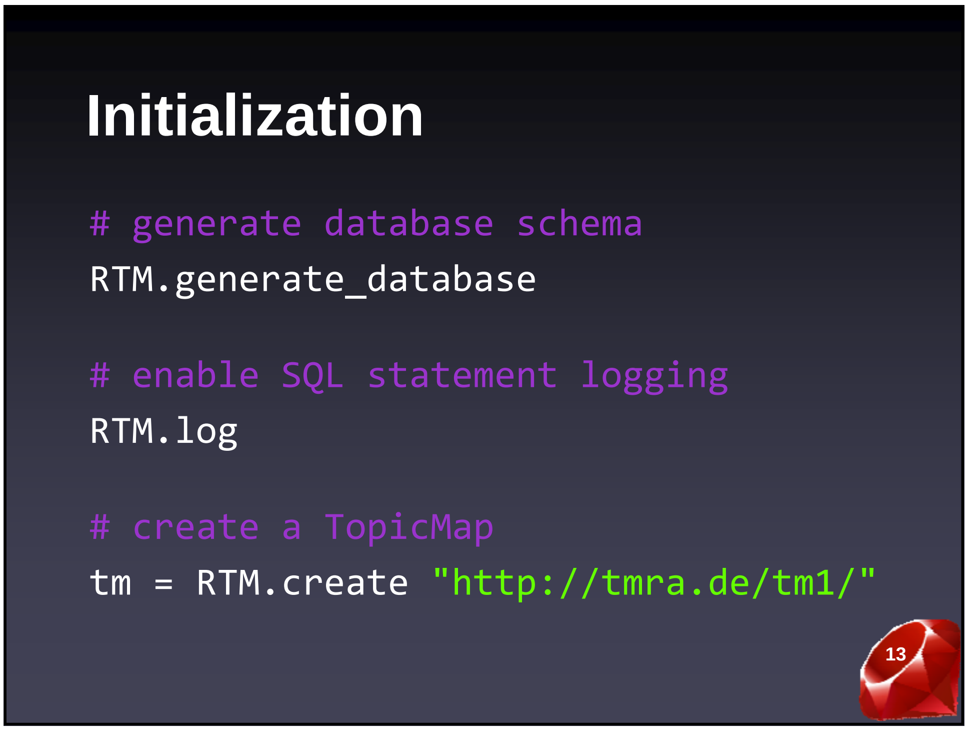


```ruby
begin
 puts "Do you need a Ruby
  Introduction?"
 x = gets
end until x =~ /(yes|no)/i

open("Ruby Intro") if x=~/yes/i
```



Loading

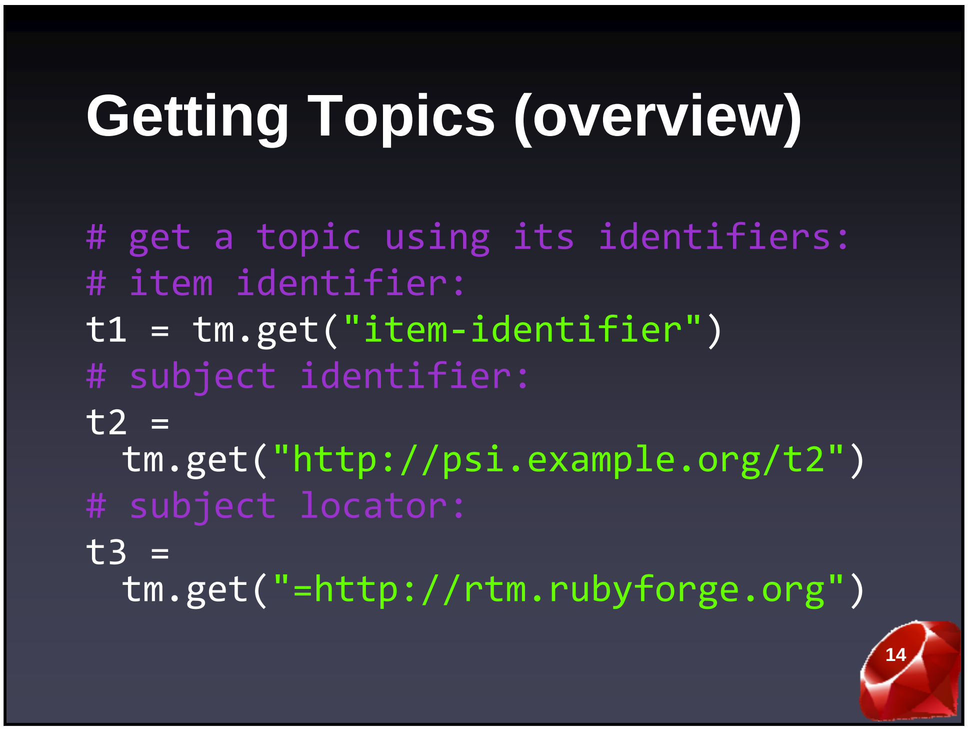
```ruby
# loading the Ruby Topic Maps library
require 'rtm'

# Connecting to a back-end
RTM.connect # Memory

RTM.connect_sqlite3("mydb.sqlite3")

RTM.connect_mysql("database_name",
  "user_name", "password", "host")
```



Initialization

```
# generate database schema
RTM.generate_database

# enable SQL statement logging
RTM.log

# create a TopicMap
tm = RTM.create "http://tmra.de/tm1/"
```

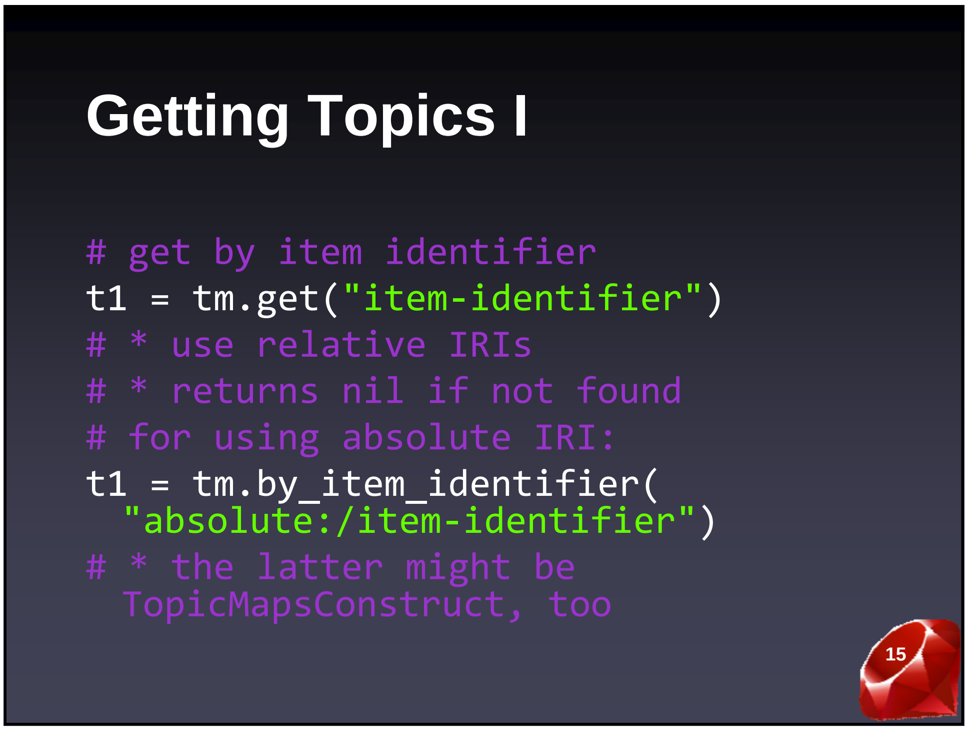
13



Getting Topics (overview)

```
# get a topic using its identifiers:
# item identifier:
t1 = tm.get("item-identifier")
# subject identifier:
t2 =
  tm.get("http://psi.example.org/t2")
# subject locator:
t3 =
  tm.get("=http://rtm.rubyforge.org")
```

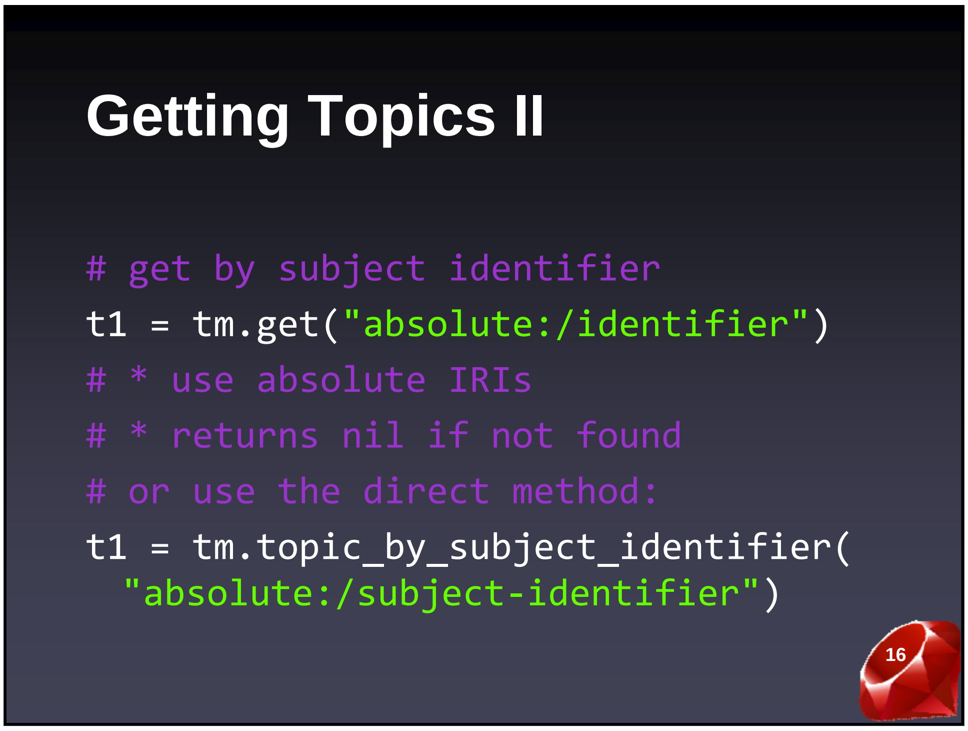
14



Getting Topics I

```
# get by item identifier
t1 = tm.get("item-identifier")
# * use relative IRIs
# * returns nil if not found
# for using absolute IRI:
t1 = tm.by_item_identifier(
  "absolute:/item-identifier")
# * the latter might be
  TopicMapsConstruct, too
```

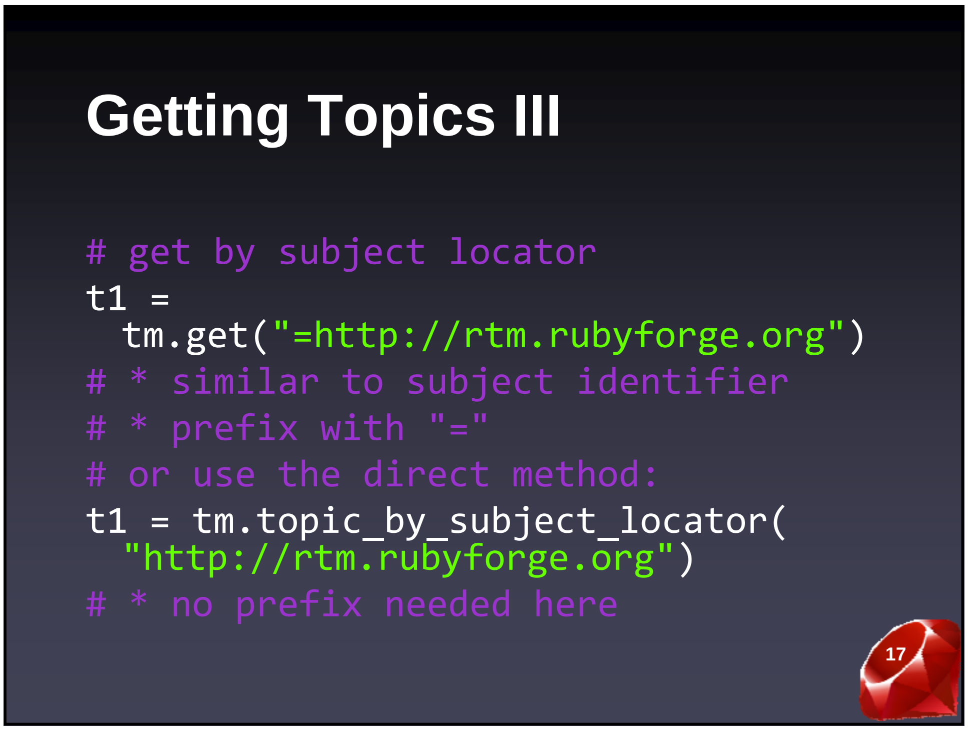
15



Getting Topics II

```
# get by subject identifier
t1 = tm.get("absolute:/identifier")
# * use absolute IRIs
# * returns nil if not found
# or use the direct method:
t1 = tm.topic_by_subject_identifier(
  "absolute:/subject-identifier")
```

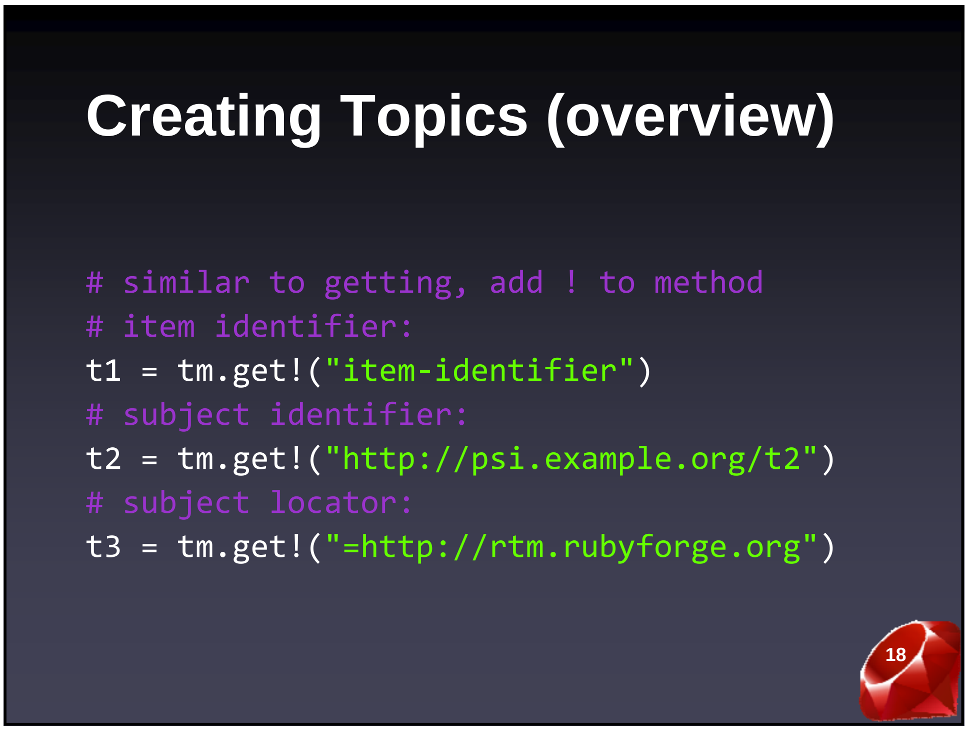
16



Getting Topics III

```
# get by subject locator
t1 =
  tm.get("=http://rtm.rubyforge.org")
# * similar to subject identifier
# * prefix with "="
# or use the direct method:
t1 = tm.topic_by_subject_locator(
  "http://rtm.rubyforge.org")
# * no prefix needed here
```

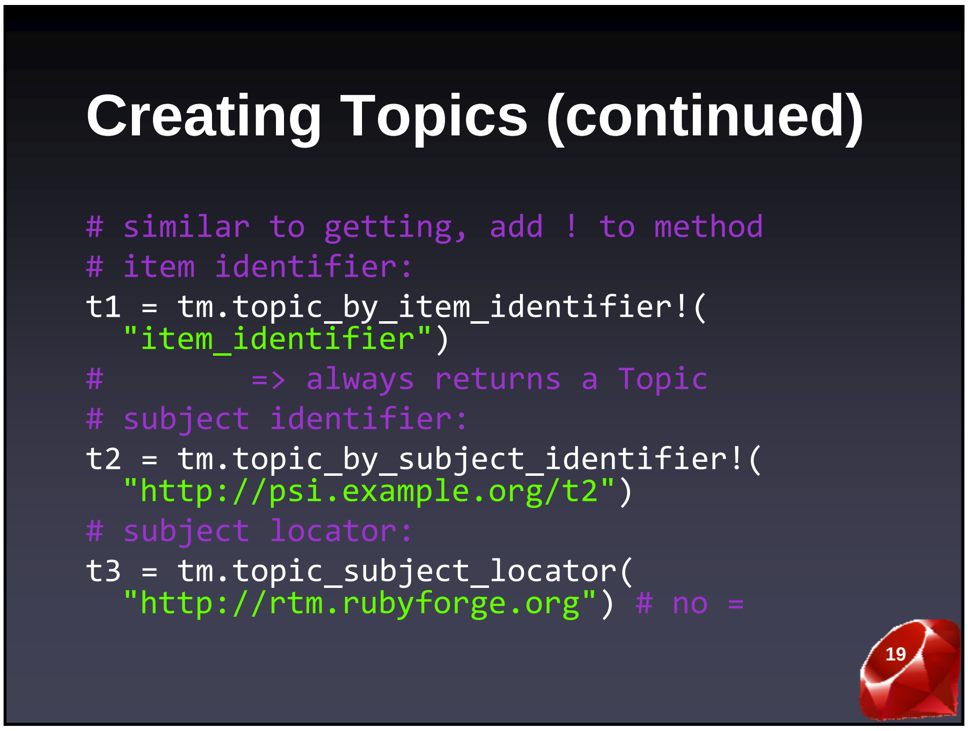
17



Creating Topics (overview)

```
# similar to getting, add ! to method
# item identifier:
t1 = tm.get!("item-identifier")
# subject identifier:
t2 = tm.get!("http://psi.example.org/t2")
# subject locator:
t3 = tm.get!("=http://rtm.rubyforge.org")
```

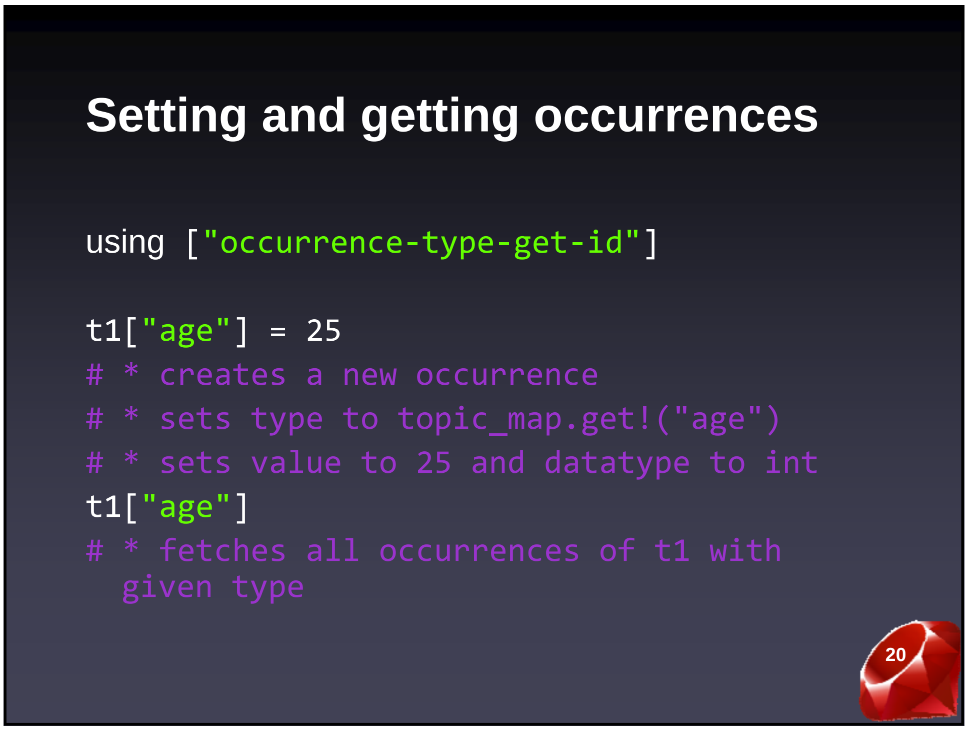
18



Creating Topics (continued)

```
# similar to getting, add ! to method
# item identifier:
t1 = tm.topic_by_item_identifier!(
  "item_identifier")
#        => always returns a Topic
# subject identifier:
t2 = tm.topic_by_subject_identifier!(
  "http://psi.example.org/t2")
# subject locator:
t3 = tm.topic_subject_locator(
  "http://rtm.rubyforge.org") # no =
```

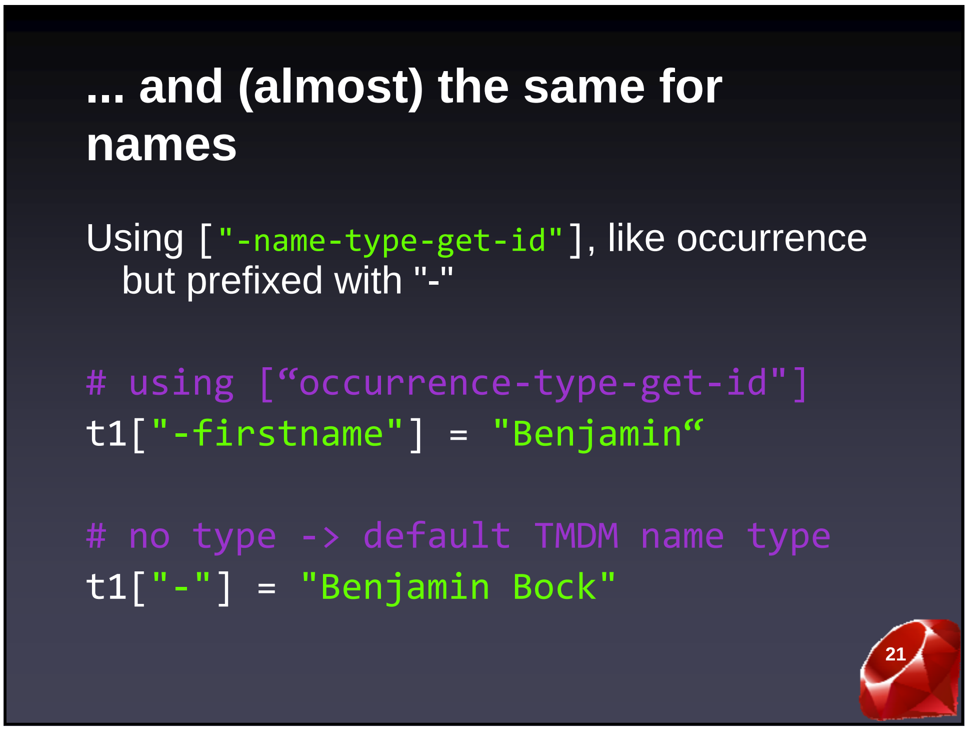
19



Setting and getting occurrences

```
using ["occurrence-type-get-id"]

t1["age"] = 25
# * creates a new occurrence
# * sets type to topic_map.get!("age")
# * sets value to 25 and datatype to int
t1["age"]
# * fetches all occurrences of t1 with
  given type
```

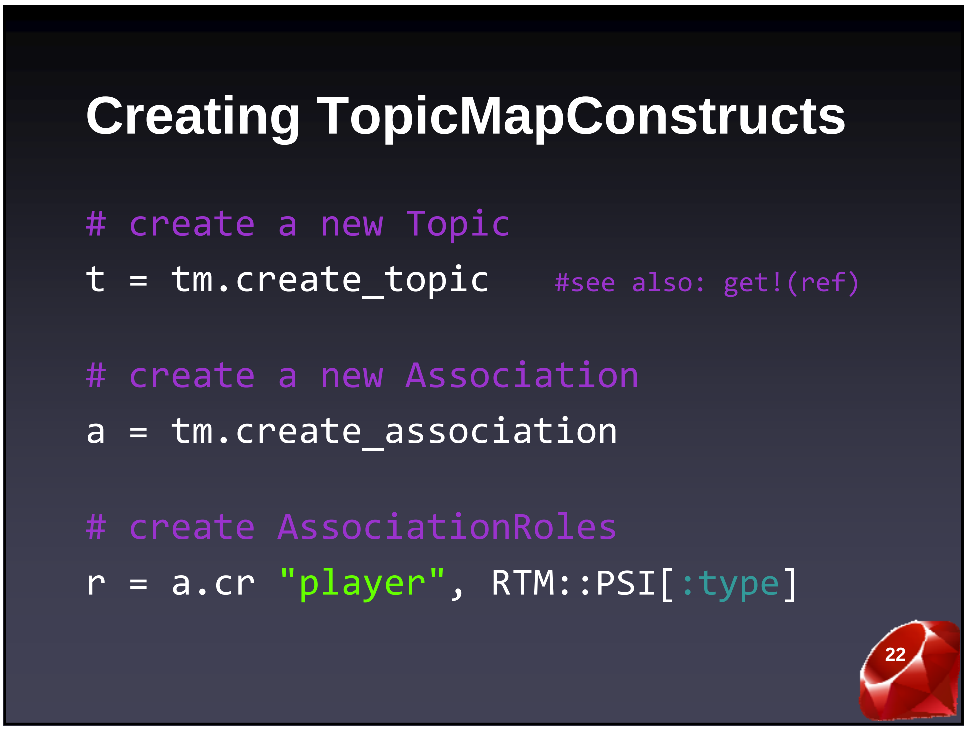
20



... and (almost) the same for names

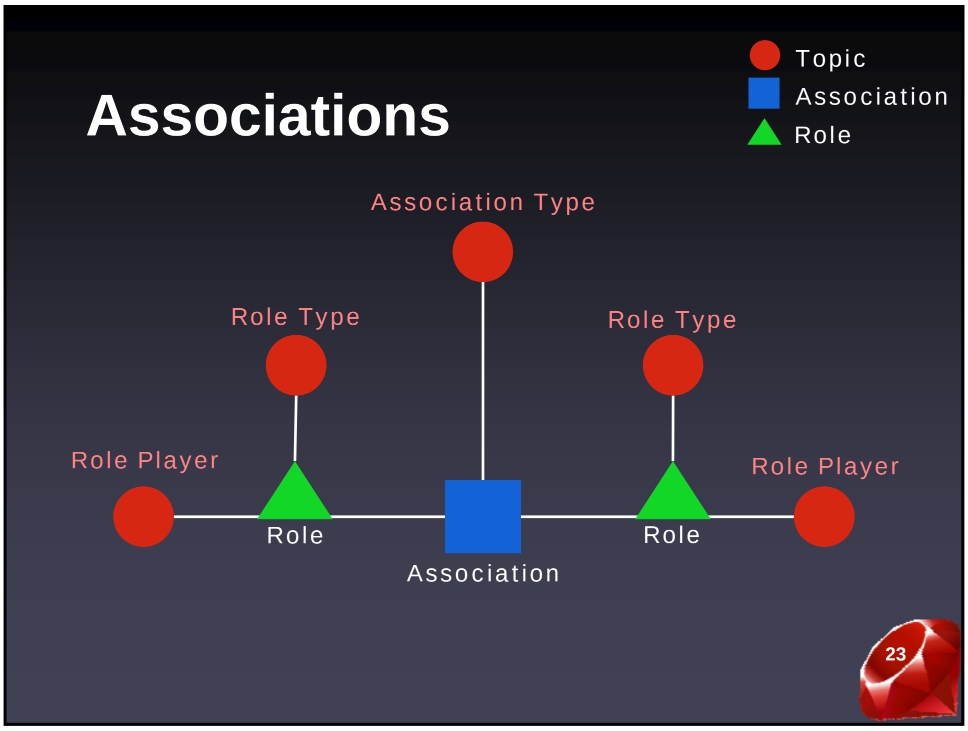
Using [`"-name-type-get-id"`], like occurrence but prefixed with "-"

```
# using [“occurrence-type-get-id"]
t1["-firstname"] = "Benjamin“

# no type -> default TMDM name type
t1["-"] = "Benjamin Bock"
```



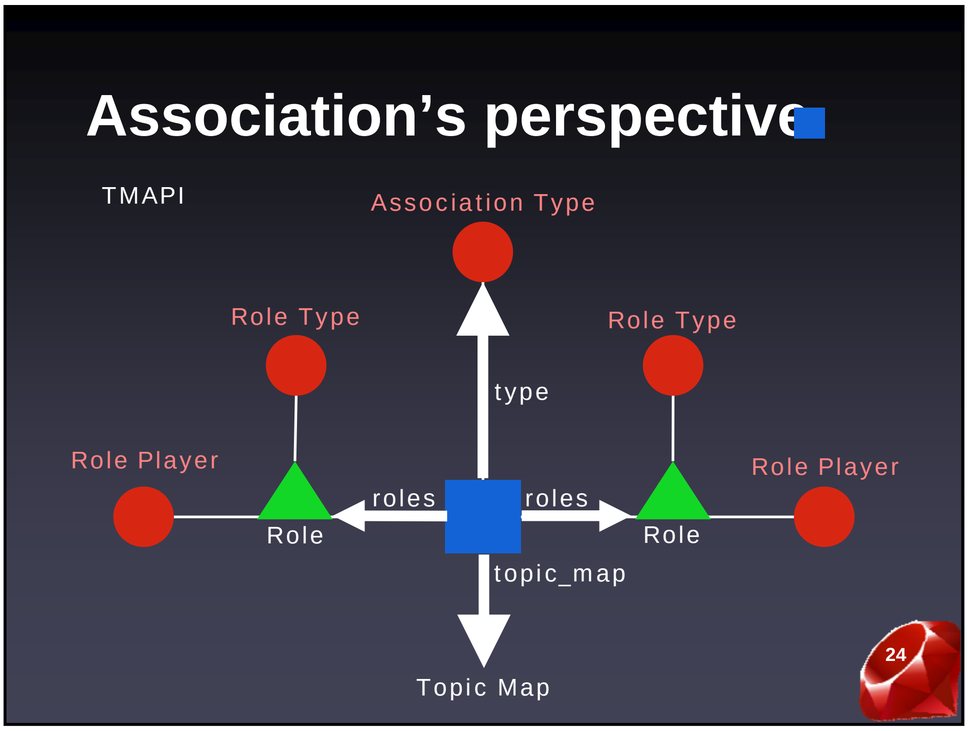
Creating TopicMapConstructs

```
# create a new Topic
t = tm.create_topic    #see also: get!(ref)

# create a new Association
a = tm.create_association

# create AssociationRoles
r = a.cr "player", RTM::PSI[:type]
```



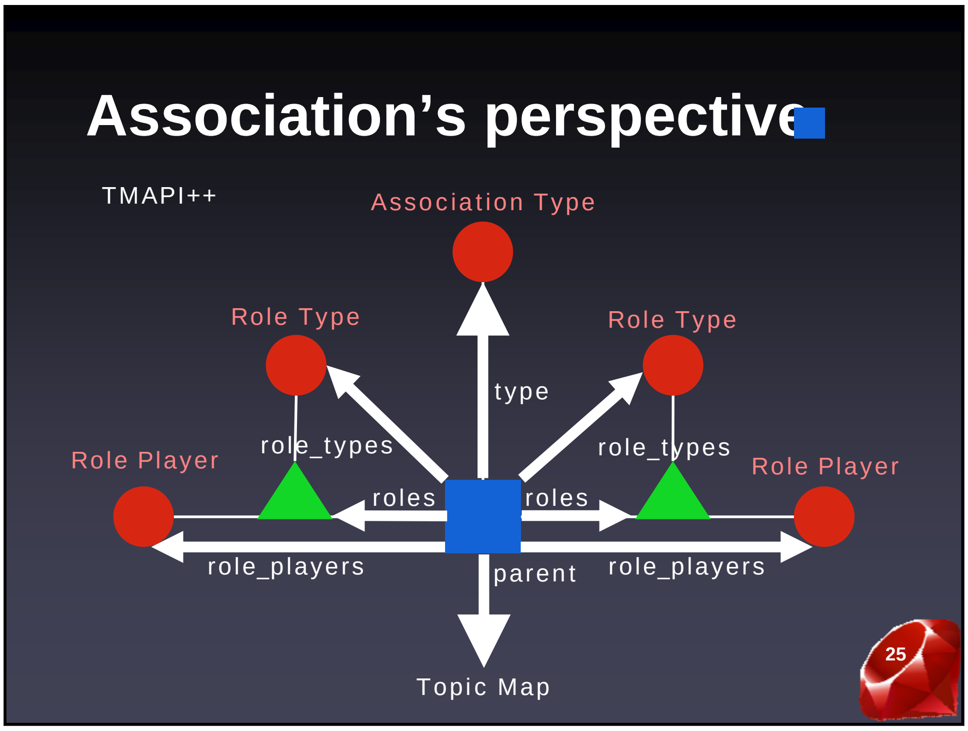
Associations

● Topic
■ Association
▲ Role

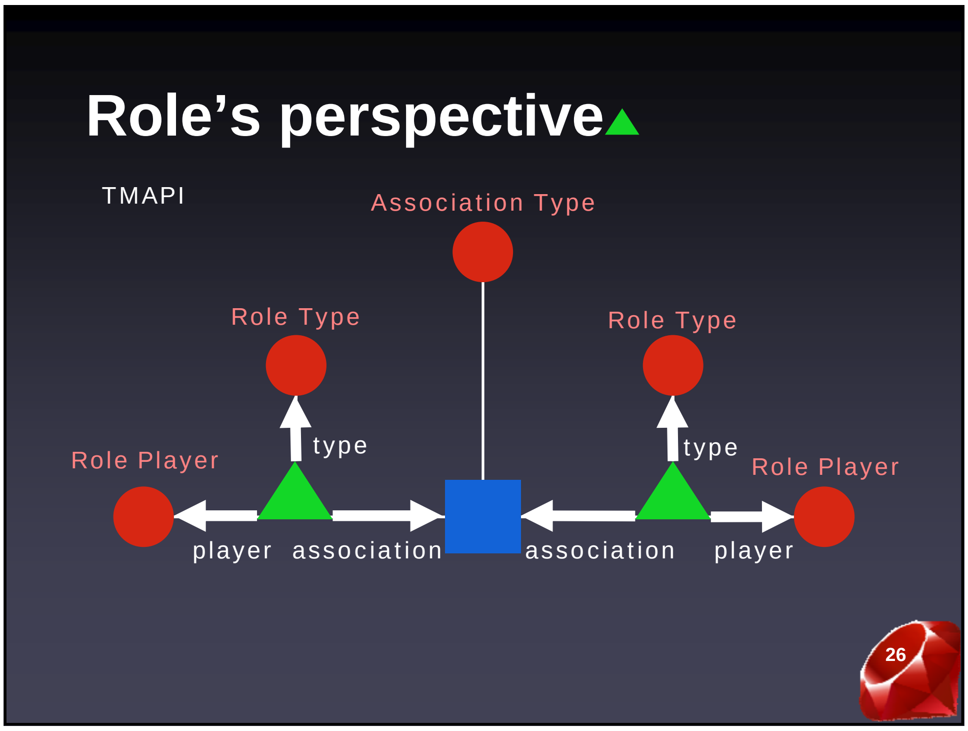


Association's perspective

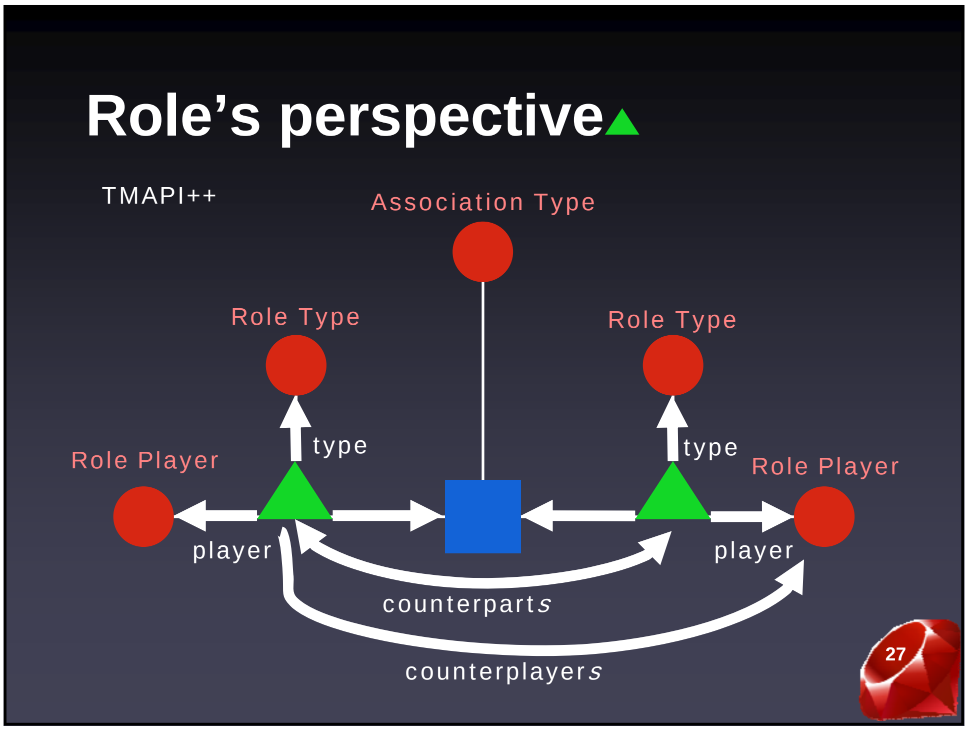
TMAPI



Association's perspective

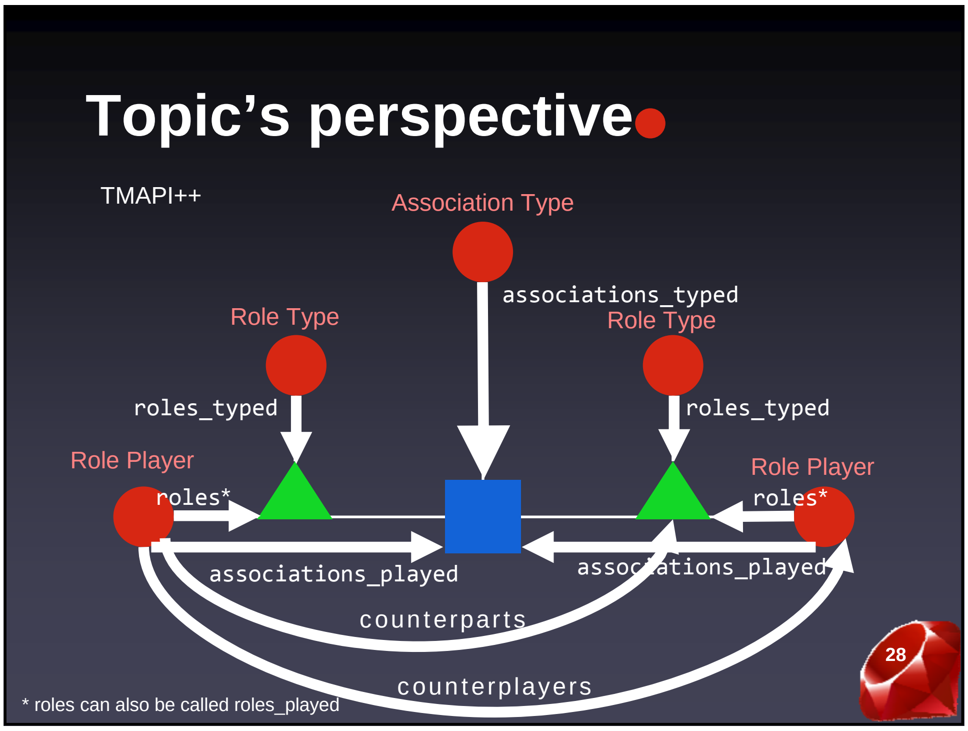


Role's perspective



Role's perspective▲

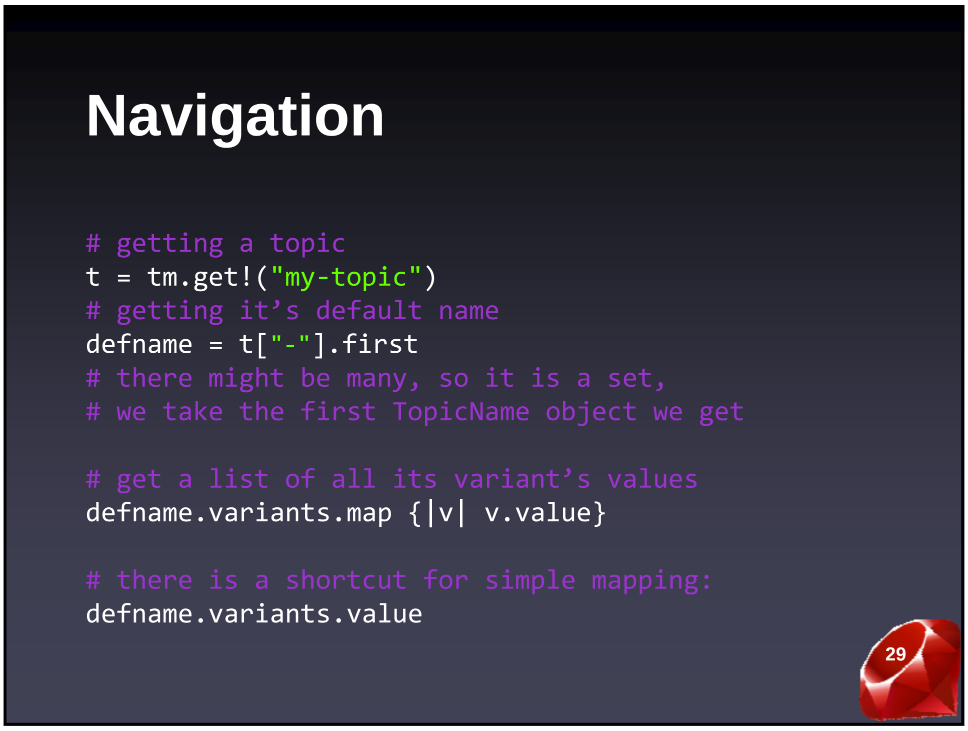
TMAPI++



Topic's perspective●

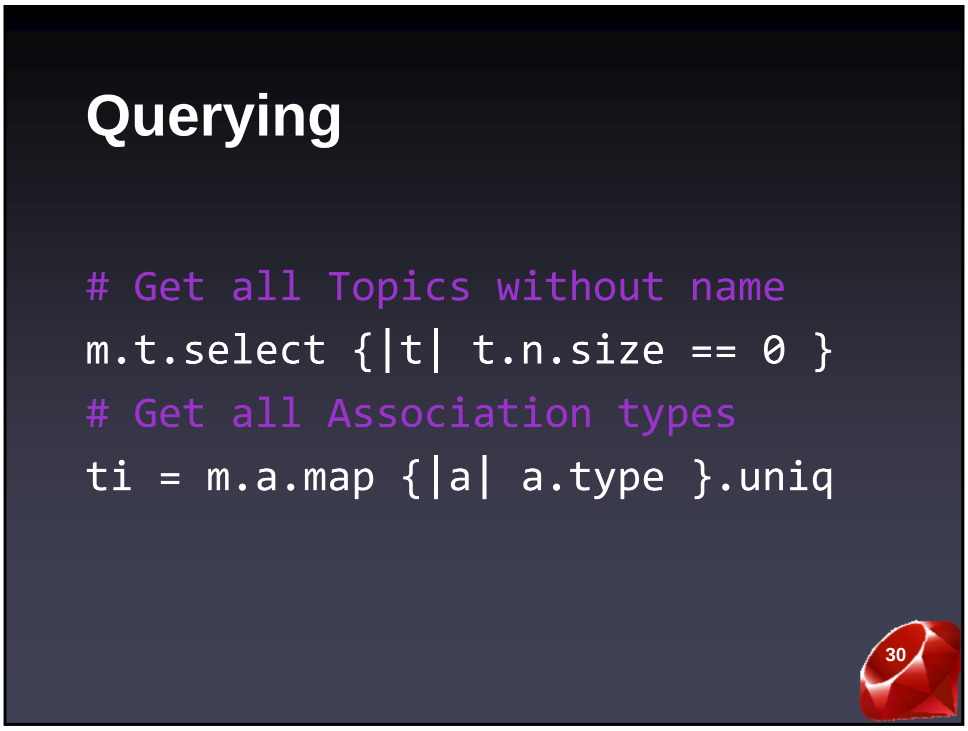
TMAPI++

* roles can also be called roles_played



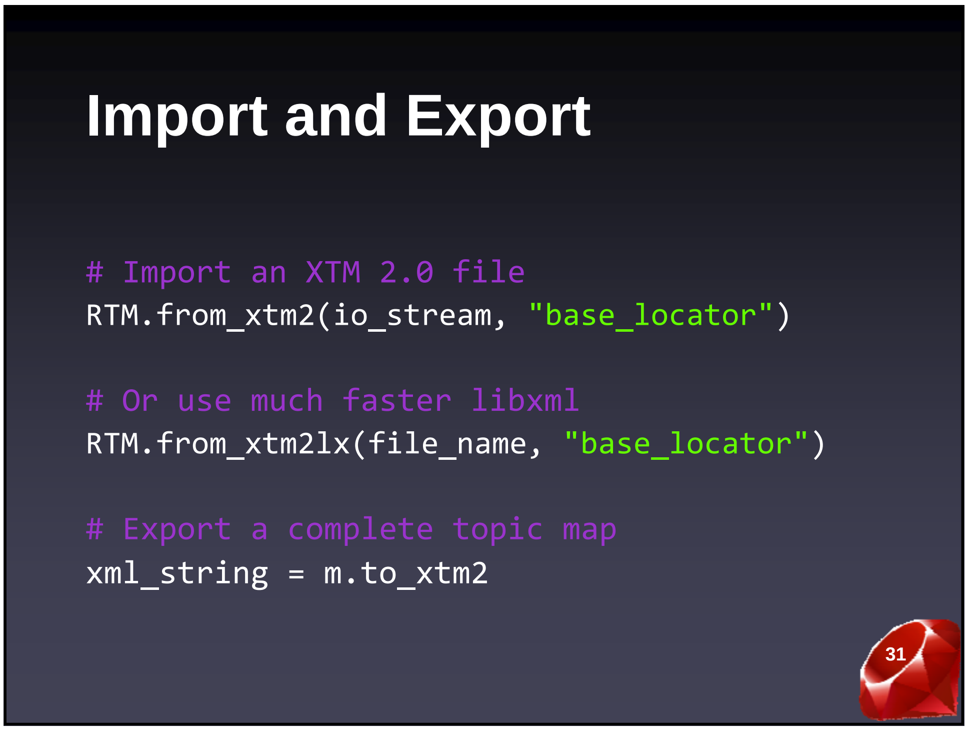
Navigation

```
# getting a topic
t = tm.get!("my-topic")
# getting it's default name
defname = t["-"].first
# there might be many, so it is a set,
# we take the first TopicName object we get

# get a list of all its variant's values
defname.variants.map {|v| v.value}

# there is a shortcut for simple mapping:
defname.variants.value
```



Querying

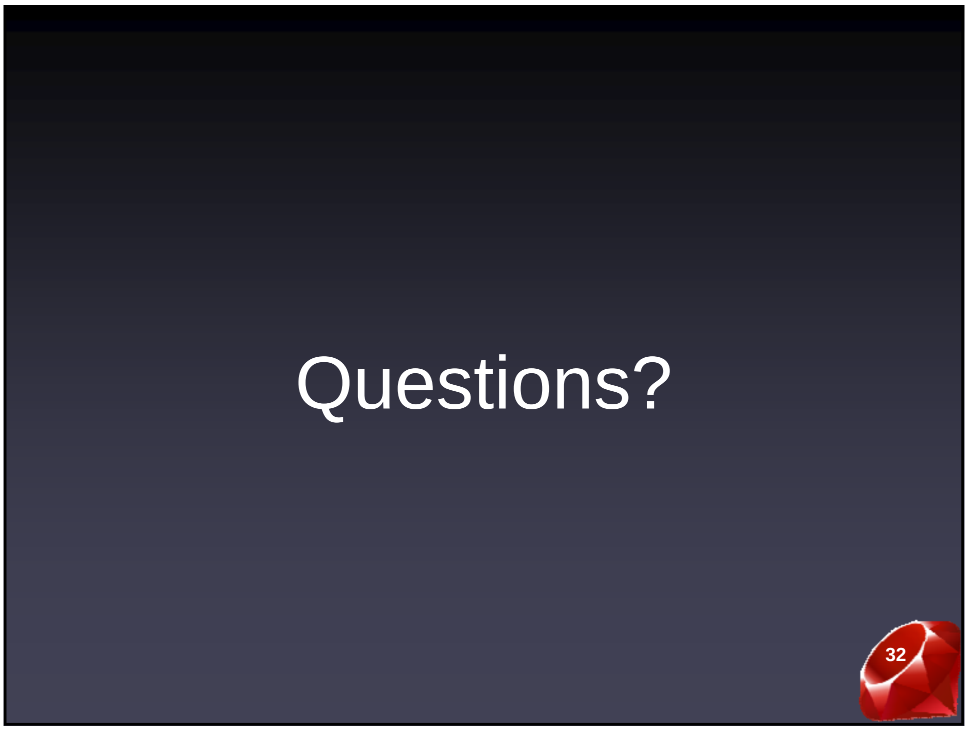
```
# Get all Topics without name
m.t.select {|t| t.n.size == 0 }
# Get all Association types
ti = m.a.map {|a| a.type }.uniq
```



Import and Export

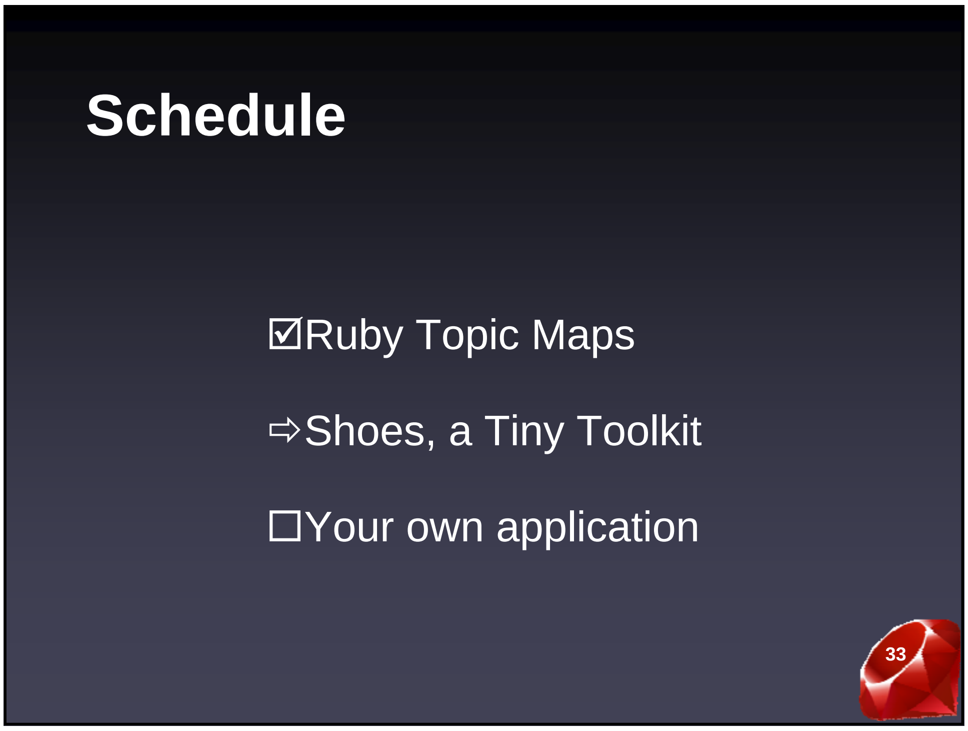
```
# Import an XTM 2.0 file
RTM.from_xtm2(io_stream, "base_locator")

# Or use much faster libxml
RTM.from_xtm2lx(file_name, "base_locator")

# Export a complete topic map
xml_string = m.to_xtm2
```

31



Questions?

32

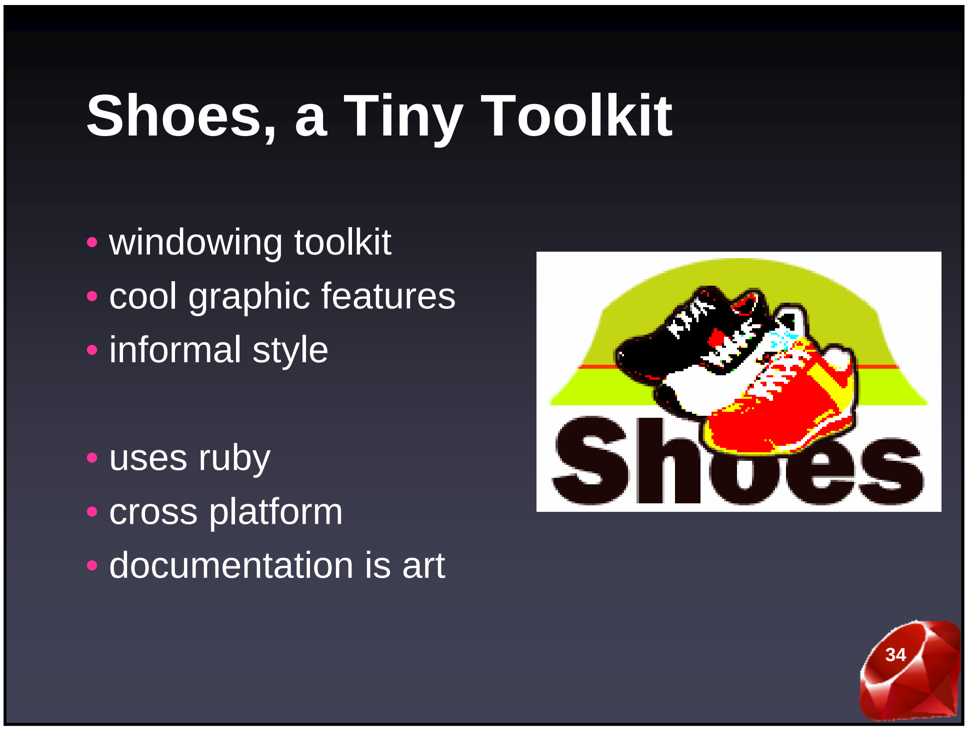


Schedule

☑Ruby Topic Maps

⇨Shoes, a Tiny Toolkit

☐Your own application



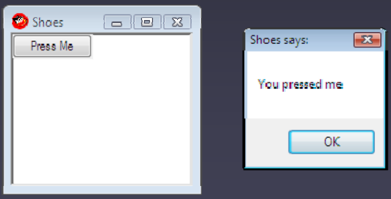
Shoes, a Tiny Toolkit

- windowing toolkit
- cool graphic features
- informal style

- uses ruby
- cross platform
- documentation is art

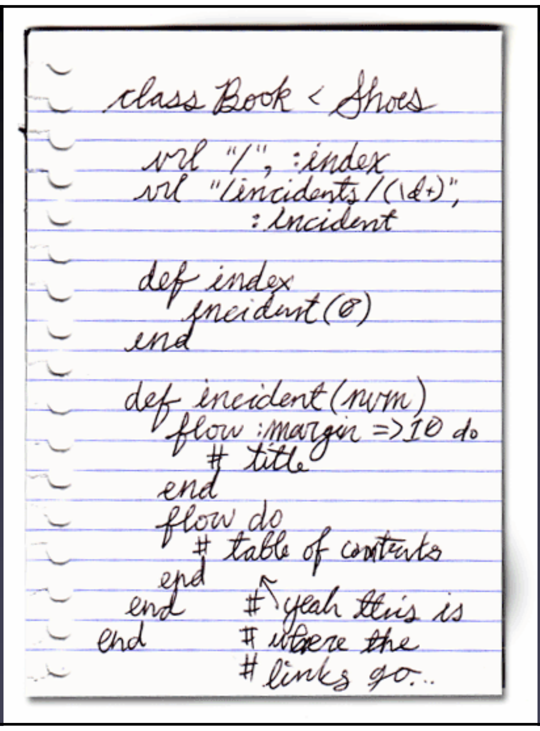
Hello World

```
Shoes.app {
  button("Press Me") {
     alert("You pressed me")
  }
}
```



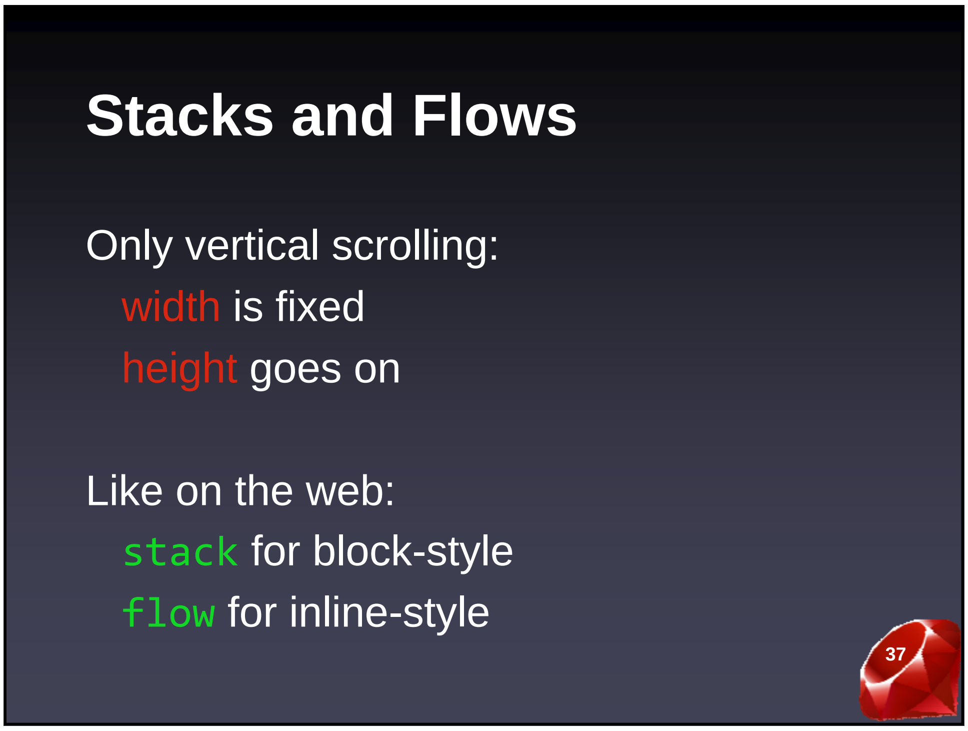
You pressed me

35



```
class Book < Shoes
  url "/", :index
  url "/incidents/(\d+)",
         :incident

  def index
    incident(0)
  end

  def incident(num)
    flow :margin => 10 do
      # title
    end
    flow do
      # table of contents
    end
  end       # ↑yeah this is
end         # where the
            # links go..
```

36

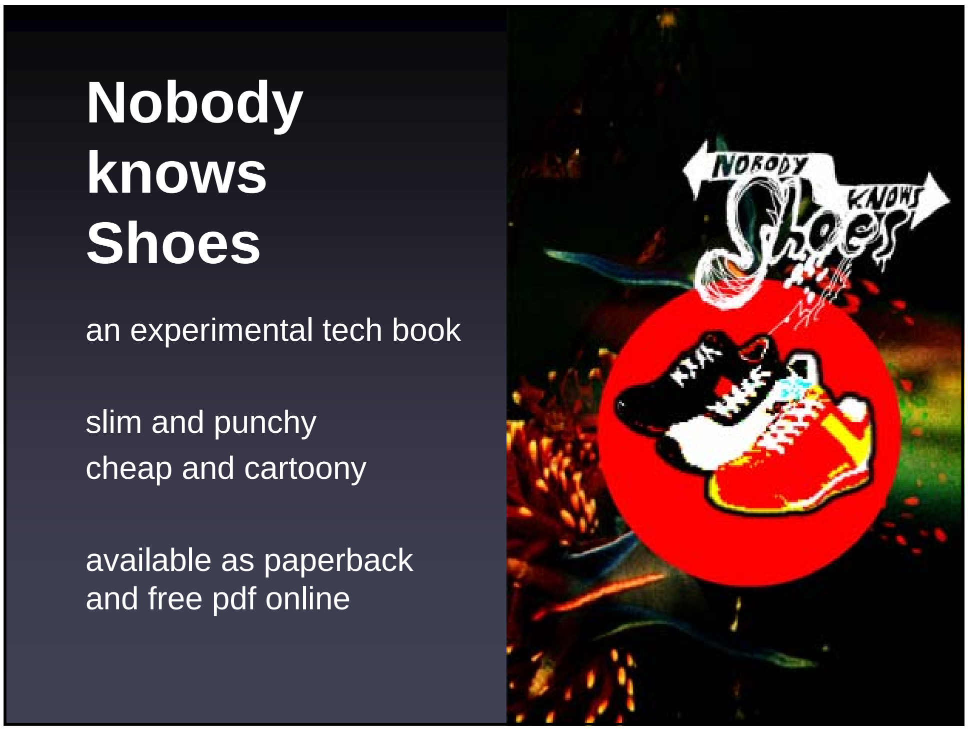


Stacks and Flows

Only vertical scrolling:
- width is fixed
- height goes on

Like on the web:
- `stack` for block-style
- `flow` for inline-style

37

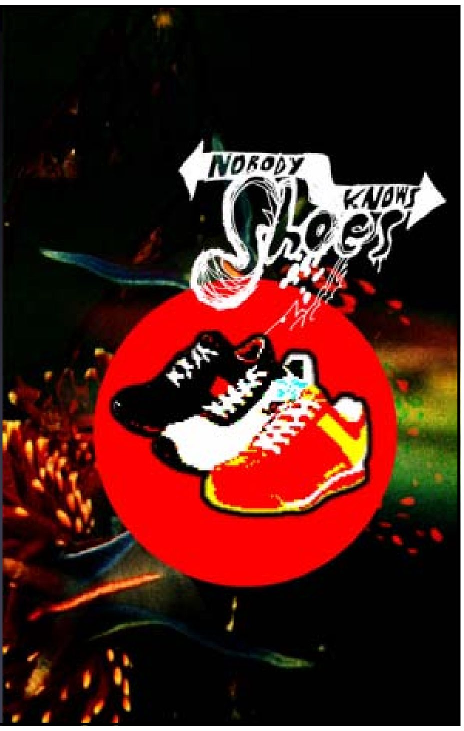


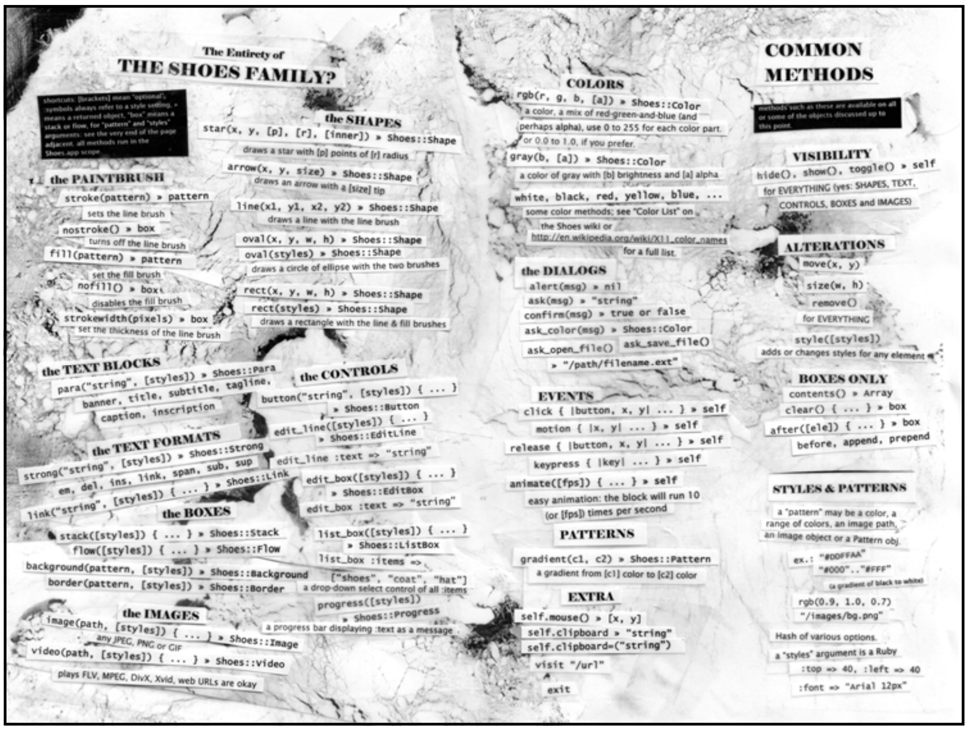
Nobody knows Shoes

an experimental tech book

slim and punchy
cheap and cartoony

available as paperback
and free pdf online





The Entirety of

THE SHOES FAMILY?

shortcuts: [brackets] mean "optional", :symbols always refer to a style setting, » means a returned object, "box" means a stack or flow, for "pattern" and "styles" arguments: see the very end of the page adjacent. all methods run in the Shoes.app scope.

the PAINTBRUSH

```
stroke(pattern) » pattern
```
sets the line brush
```
nostroke() » box
```
turns off the line brush
```
fill(pattern) » pattern
```
set the fill brush
```
nofill() » box
```
disables the fill brush
```
strokewidth(pixels) » box
```
set the thickness of the line brush

the SHAPES

```
star(x, y, [p], [r], [inner]) » Shoes::Shape
```
draws a star with [p] points of [r] radius
```
arrow(x, y, size) » Shoes::Shape
```
draws an arrow with a [size] tip
```
line(x1, y1, x2, y2) » Shoes::Shape
```
draws a line with the line brush
```
oval(x, y, w, h) » Shoes::Shape
oval(styles) » Shoes::Shape
```
draws a circle of ellipse with the two brushes
```
rect(x, y, w, h) » Shoes::Shape
rect(styles) » Shoes::Shape
```
draws a rectangle with the line & fill brushes

the TEXT BLOCKS

```
para("string", [styles]) » Shoes::Para
banner, title, subtitle, tagline,
caption, inscription
```

the TEXT FORMATS

```
strong("string", [styles]) » Shoes::Strong
em, del, ins, link, span, sub, sup
link("string", [styles]) { ... } » Shoes::Link
```

the BOXES

```
stack([styles]) { ... } » Shoes::Stack
flow([styles]) { ... } » Shoes::Flow
background(pattern, [styles]) » Shoes::Background
border(pattern, [styles]) » Shoes::Border
```

the IMAGES

```
image(path, [styles]) { ... } » Shoes::Image
```
any JPEG, PNG or GIF
```
video(path, [styles]) { ... } » Shoes::Video
```
plays FLV, MPEG, DivX, Xvid, web URLs are okay

the CONTROLS

```
button("string", [styles]) { ... }
  » Shoes::Button
edit_line([styles]) { ... }
  » Shoes::EditLine
edit_line :text => "string"
edit_box([styles]) { ... }
  » Shoes::EditBox
edit_box :text => "string"
list_box([styles]) { ... }
  » Shoes::ListBox
list_box :items =>
  ["shoes", "coat", "hat"]
```
a drop-down select control of all :items
```
progress([styles])
  » Shoes::Progress
```
a progress bar displaying :text as a message

COLORS

```
rgb(r, g, b, [a]) » Shoes::Color
```
a color, a mix of red-green-and-blue (and perhaps alpha), use 0 to 255 for each color part or 0.0 to 1.0, if you prefer.
```
gray(b, [a]) » Shoes::Color
```
a color of gray with [b] brightness and [a] alpha
```
white, black, red, yellow, blue, ...
```
some color methods; see "Color List" on the Shoes wiki or http://en.wikipedia.org/wiki/X11_color_names for a full list.

the DIALOGS

```
alert(msg) » nil
ask(msg) » "string"
confirm(msg) » true or false
ask_color(msg) » Shoes::Color
ask_open_file()  ask_save_file()
  » "/path/filename.ext"
```

EVENTS

```
click { |button, x, y| ... } » self
motion { |x, y| ... } » self
release { |button, x, y| ... } » self
keypress { |key| ... } » self
animate([fps]) { ... } » self
```
easy animation: the block will run 10 (or [fps]) times per second

PATTERNS

```
gradient(c1, c2) » Shoes::Pattern
```
a gradient from [c1] color to [c2] color

EXTRA

```
self.mouse() » [x, y]
self.clipboard » "string"
self.clipboard=("string")
visit "/url"
exit
```

COMMON METHODS

methods such as these are available on all or some of the objects discussed up to this point.

VISIBILITY

```
hide(), show(), toggle() » self
```
for EVERYTHING (yes: SHAPES, TEXT, CONTROLS, BOXES and IMAGES)

ALTERATIONS

```
move(x, y)
size(w, h)
remove()
```
for EVERYTHING
```
style([styles])
```
adds or changes styles for any element

BOXES ONLY

```
contents() » Array
clear() { ... } » box
after([ele]) { ... } » box
before, append, prepend
```

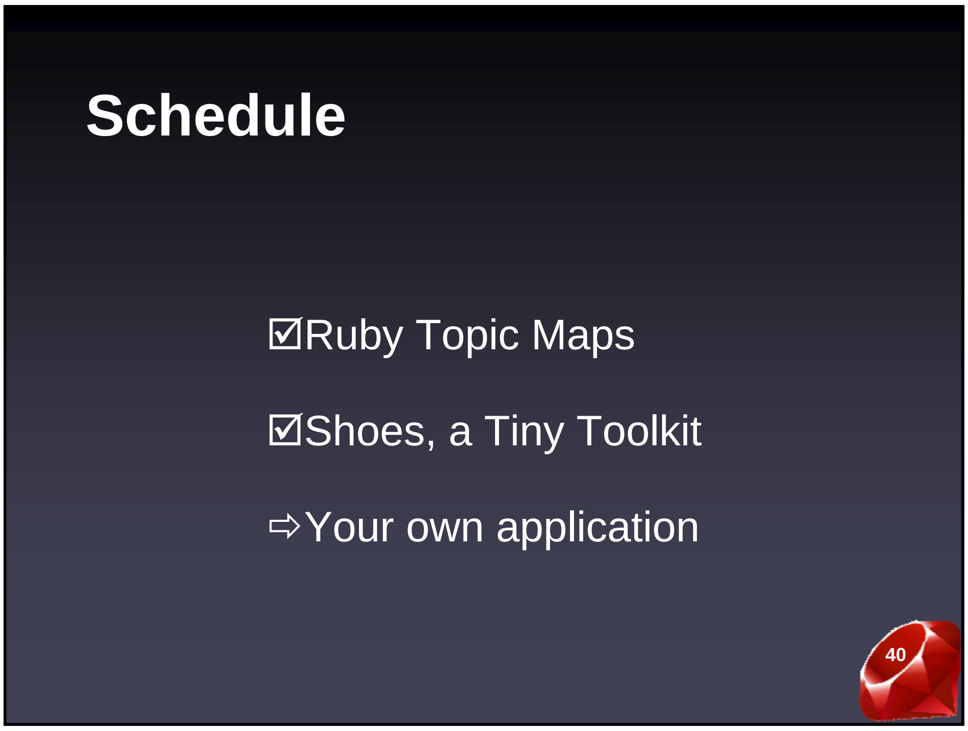
STYLES & PATTERNS

a "pattern" may be a color, a range of colors, an image path, an image object or a Pattern obj.

```
ex.: "#00FFAA"
     "#000".."#FFF"
```
(a gradient of black to white)
```
rgb(0.9, 1.0, 0.7)
"/images/bg.png"
```

a "styles" argument is a Ruby Hash of various options.

```
:top => 40, :left => 40
:font => "Arial 12px"
```



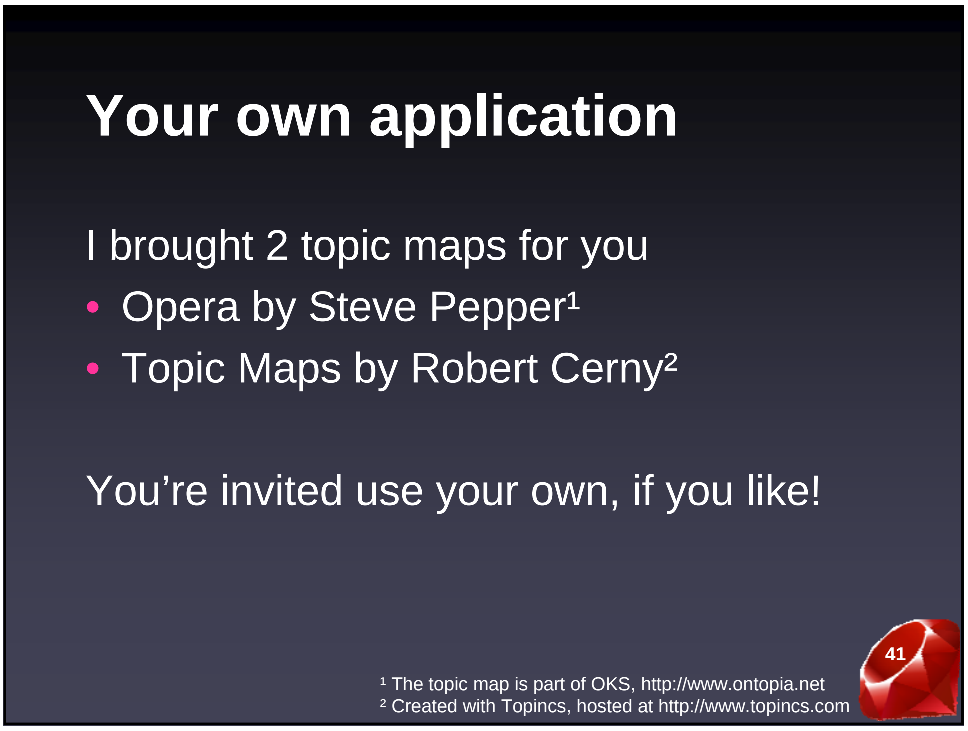
Schedule

☑Ruby Topic Maps

☑Shoes, a Tiny Toolkit

⇨Your own application

40



Your own application

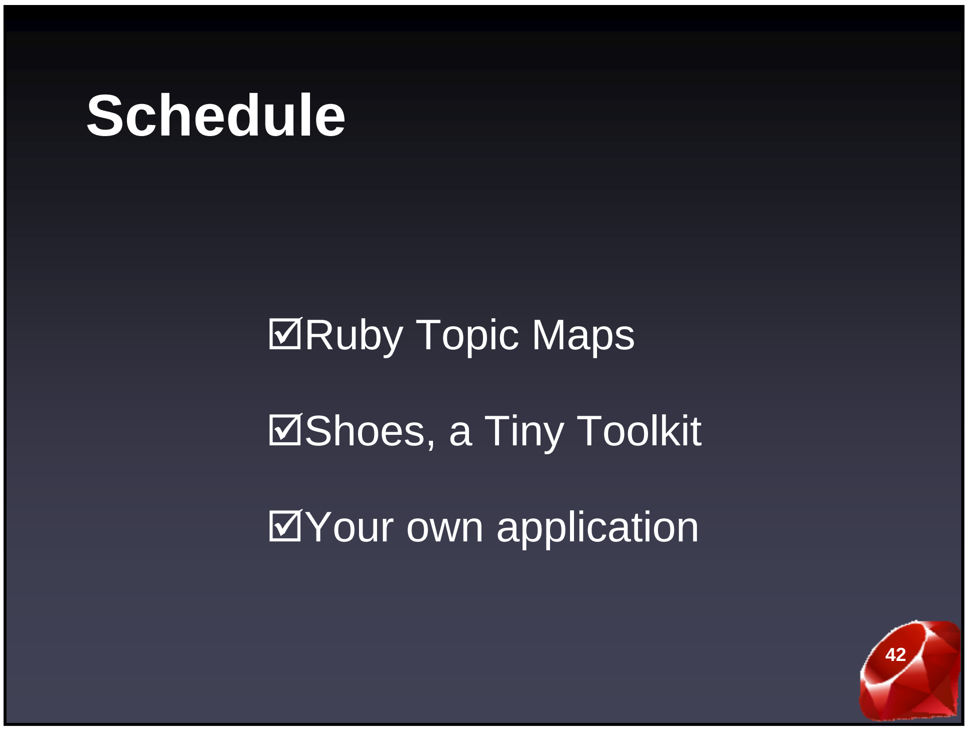
I brought 2 topic maps for you

- Opera by Steve Pepper[1]
- Topic Maps by Robert Cerny[2]

You're invited use your own, if you like!

[1] The topic map is part of OKS, http://www.ontopia.net
[2] Created with Topincs, hosted at http://www.topincs.com

41



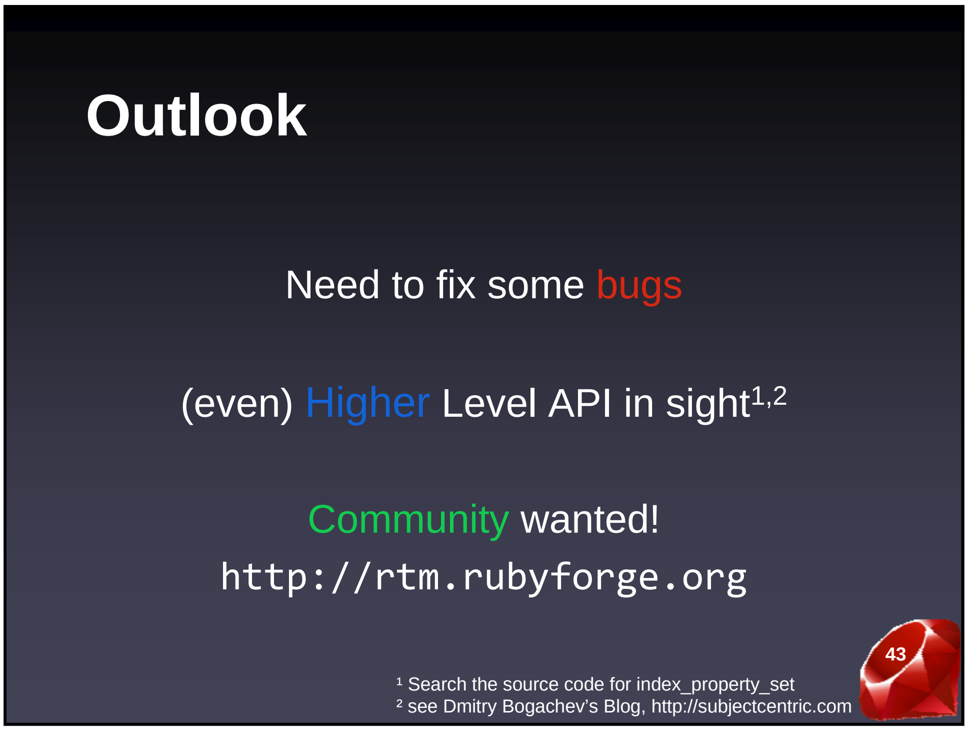
Schedule

☑Ruby Topic Maps

☑Shoes, a Tiny Toolkit

☑Your own application

42



Outlook

Need to fix some bugs

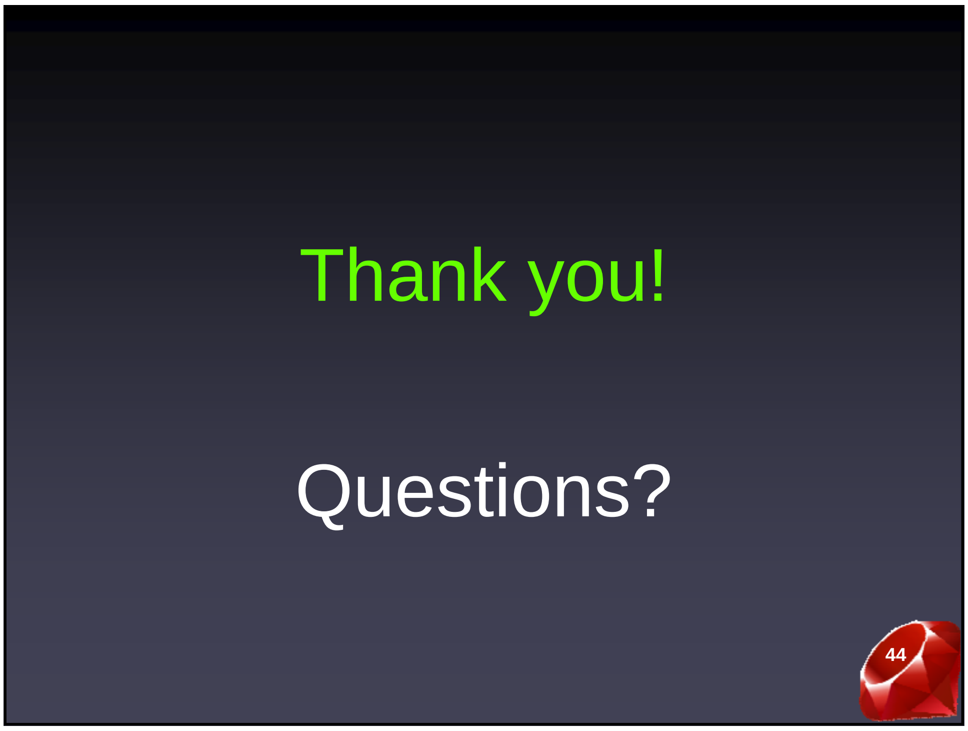
(even) Higher Level API in sight[1,2]

Community wanted!

`http://rtm.rubyforge.org`

[1] Search the source code for index_property_set
[2] see Dmitry Bogachev's Blog, http://subjectcentric.com

43



Thank you!

Questions?

44