Snap und Heist - HaL7

Halle (Saale), Germany

July 13 2012

Matthias Fischmann & Sönke Hahn

{sh,mf}@zerobuzz.net

# Installation

- `darcs get http://patch-tag.com/r/shahn/hal-snap-2012/` (or open it in browser)
- `cabal install snap -fhint`
- `snap --help`

# Snap Overview: Features

- Stand-alone, high-performance HTTP server.
- HTML template rendering with Heist.
- Snaplets for component-based development.
- Snap core provides
  - Snap.Snaplet.Heist
  - Snap.Snaplet.Session
  - Snap.Snaplet.Auth

# Overview: Snap Code Base

*(as of 2012-07-08)*

- 'cabal list snaplet' matches 12 packages:

  *snaplet-acid-state snaplet-environments snaplet-hdbc snaplet-i18n snaplet-recaptcha snaplet-redis snaplet-redson snaplet-sedna snaplet-tasks . . .*

- github search for 'snaplet' yields 26 results:

  *snap-guestbook gruzeSnaplet snaplet-persistence . . .*

# Alternatives to Snap (1)

- http://www.yesodweb.com/
- http://happstack.com/

All three frameworks are Haskell, so you always get great performance, high-quality byte code, robust threading etc.

Trade-offs in the details:

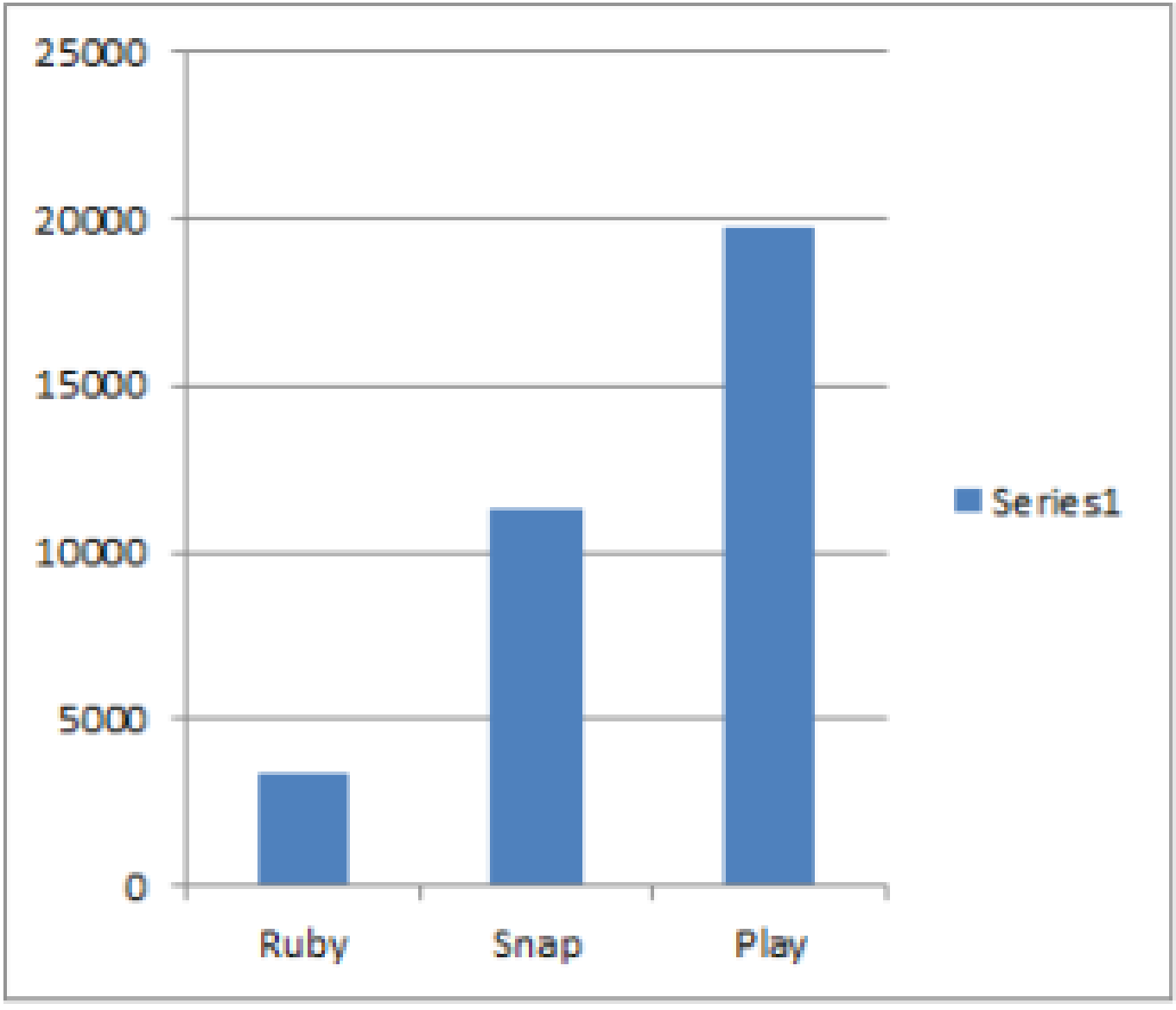
- http://softwaresimply.blogspot.de/2012/04/hopefully-fair-and-useful-comparison-of.html

# Alternatives to Snap (2)

- http://www.haskell.org/haskellwiki/Web/Frameworks contains pointers to 6 more projects.
- http://playframework.org/
  java / scala

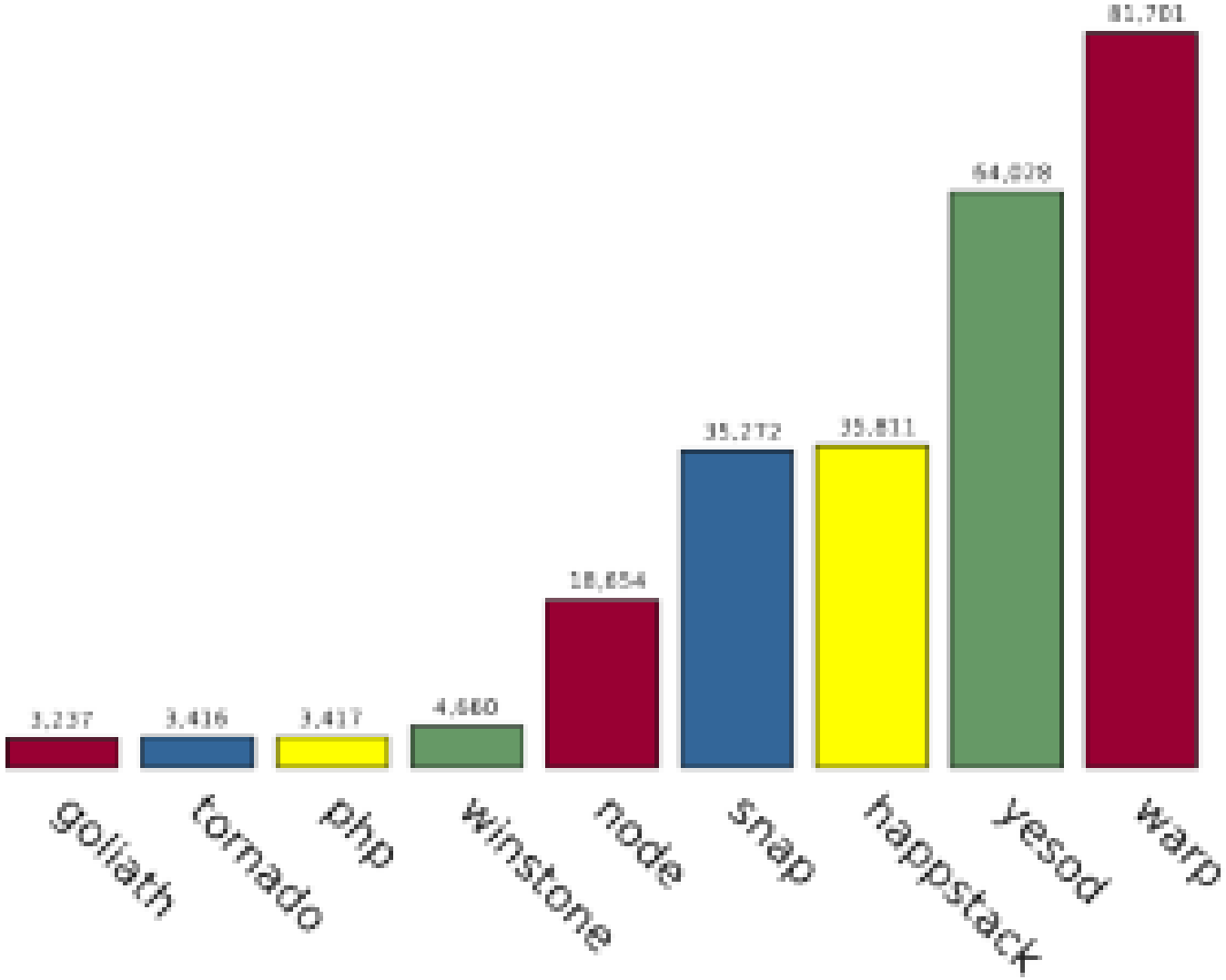
# Snap Overview: Performance (1)

http://news.ycombinator.com/item?id=1380405



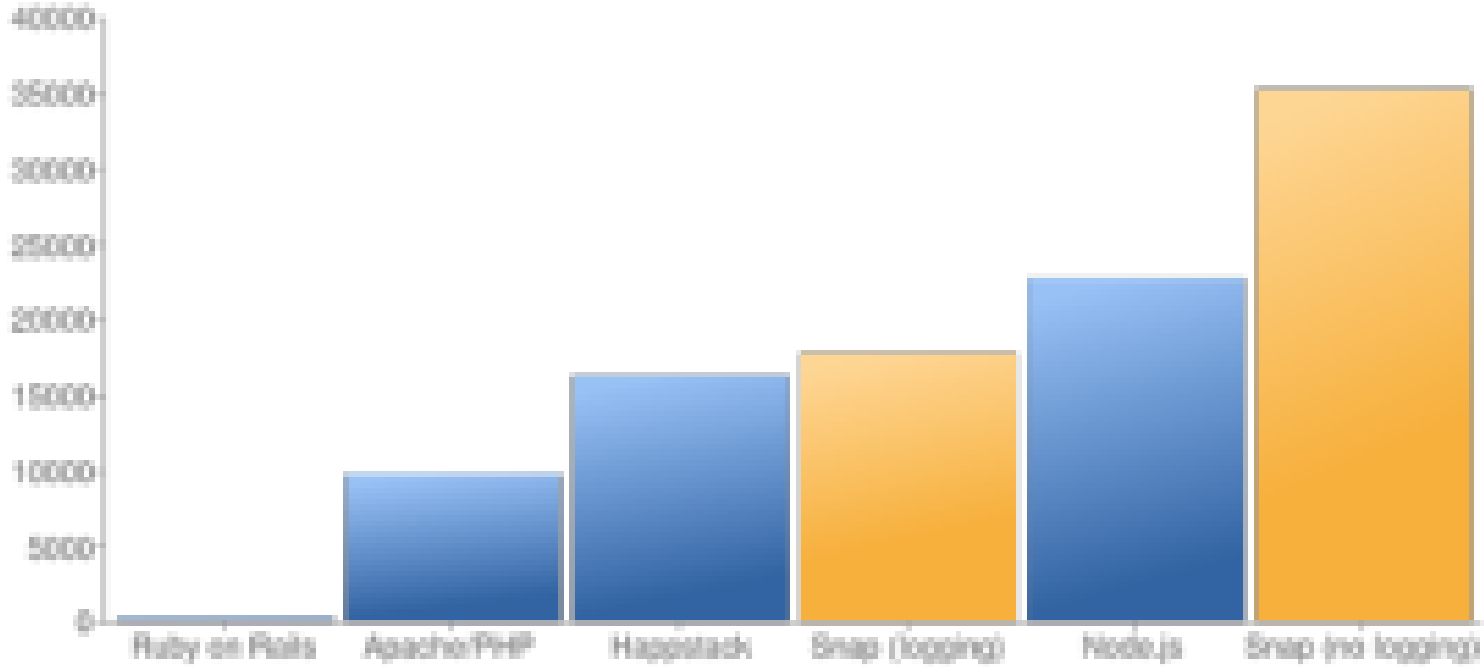
# Snap Overview: Performance (2)

http://www.yesodweb.com/blog/2011/03/preliminary-warp-cross-language-benchmarks



# Snap Overview: Performance (3)

http://snapframework.com/blog/2010/11/17/snap-0.3-benchmarks



# Snap's Software Stack

- cabal and ghc to build a snap application
- snap library
- snap executable (for 'init' only)
- applications built with snap are web servers
  - stand-alone
  - as a sub-site behind nginx/apache/...
- documentation
  - snap website
  - haddock
  - hoogle

# The first Snap session

- `snap init`
- `cabal install -fdevelopment`
- run the executable
- point your browser to http://localhost:8000

# -fhint

with running server:

- change source code
- click reload in browser
- be amazed!
  - changed source code modules are
    - located,
    - recompiled, and
    - linked on the fly!
  - compile time errors are displayed in browser
    - (this is not are a security issue, unless you install your production system with -fdevelopment :-)

# Routing

Routing is a mapping of URLs to Handlers

```
addRoutes [ ("login/", handleLogin :: Handler)
          , ("preferences/", editPrefs :: Handler)
          , ("static/", serveDirectory "static" :: Handler)
          , ...
          ]

type Handler = ?
-- Something like: Request -> Response
```

# Handlers

Snap has its own Handler Type:

```
data Handler g l a
```

`Handler g l` is a Monad. It can

- write to the body of the HTTP response;
- modify the HTTP response;
- read the HTTP request object;
- read GET/POST parameters;
- do IO-stuff (via `liftIO`);
- throw and catch exceptions (see also: `wrapHandlers`);
- re-route (refuse to respond to a request);
- access two Snap states (global and snaplet-local).

# Snaplets

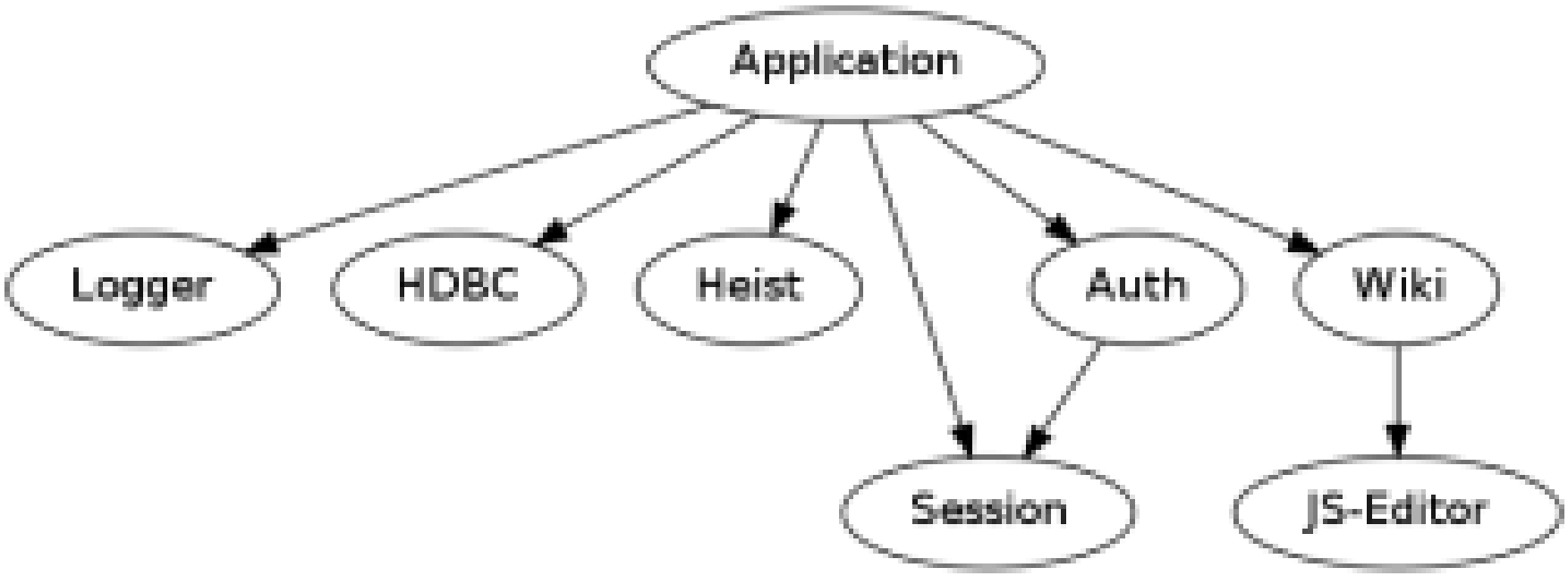
Motivation

- Parts of websites should be composable
- Parts should be self-contained with minimal interface
- Composition of these parts should be easy and safe

Proposed solution: Snaplets

# Creating Websites by Composing Snaplets

Example of a snaplet tree:



# What are Snaplets?

Snaplets have their own

- initialisation procedure,
- state (per request),
- URL-Path,
- URL-Routing,
- configuration,
- filesystem directory,
- set of handlers to access their functionality.

# The Two States

A Handler

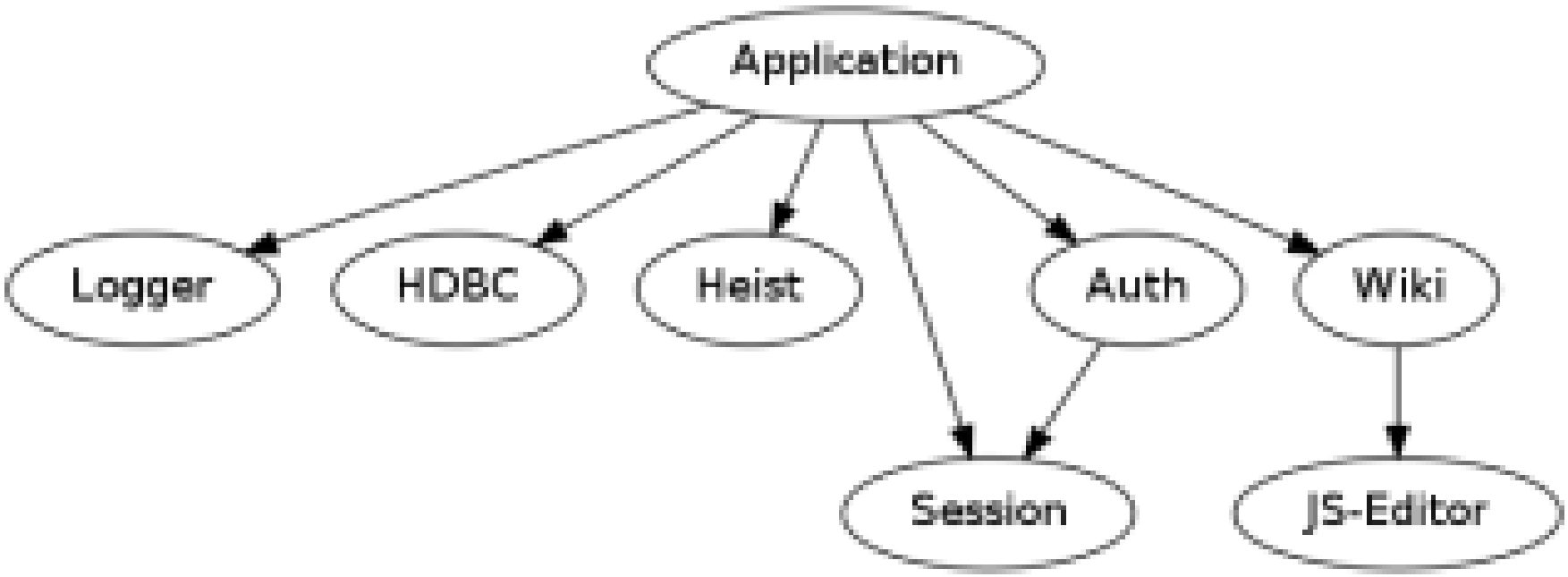
`Handler g l a`

has always two states:

- the global state `g`
  the root node of the snaplet tree
- the local state `l`
  the state of the local snaplet

The two states allow for explicit definition of what snaplet handlers can access.

# Accessing other Snaplets



# Lenses

Snap uses lenses as a way to access snaplets by moving around the snaplet tree stored in `App`. A lense is used in a handler of one local state to call a handler on another local state.

```
Lens Application (Snaplet Wiki)
```

provides

- getter: `Application -> Snaplet Wiki`
- setter: `Snaplet Wiki -> Application -> Application`
- ...

Lenses are generated by makeLens (Template Haskell)

# Different possibilities for Intra-Snaplet-Communication

Mechanisms:

- direct child node access (**with**)
- sibling access (**withTop**)
- lense as handler argument
- lense stored in snaplet state
- via HasSnaplet class instance

Note:

- You will encounter most of these mechanisms in the Snap code, even without using extra packages.
- All of them are sometimes the best choice.
- In the end, they are all just different ways of passing a lense.

# snaplet-hdbc

Excersise: Connect the `dbCounter` function to a route.

# Heist (1)

Heist is an independent library for dynamic HTML/XML template rendering:

- simple tag / attribute substitution with templates (`<bind>`)
- including of templates in templates with name substitution (`<apply>`)
- tag / attribute substitution with splices (**haskell code**)
- Templates are files (suffix: `.tpl`) that contain xml expressions and that are processed at run time.

# Heist (2)

Snap provides a snaplet for

- rendering directly into the HTTP response object maintained in the application state
- maintaining splices inside the application state (local to handlers and global to application)

more info:

- http://softwaresimply.blogspot.de/2011/04/heist-in-60-seconds.html
- https://snapframework.com/docs/tutorials/heist/

# Heist: bind (1)

```
<bind tag="longname">
  Einstein, Feynman, Heisenberg, and Newton Research
  Corporation Ltd.<sup>TM</sup>
</bind>
<p>
  We at <longname/> have research expertise in many areas
  of physics.  Employment at <longname/> carries
  significant prestige.  The rigorous hiring process
  developed by <longname/> is leading the industry.
</p>
```

# Heist: bind (2)

Attribute substitution:

```
<bind tag="paragraph_class">editor_comment</bind>
<p class="${paragraph_class}">...</p>
```

The rendered output looks something like this:

```
<p class="editor_comment">
  The foo identifier is substituted into the class
  attribute of the paragraph tag.
</p>
```

# Heist: apply (1)

Example: include navigation bar HTML sub-tree in different pages.

nav.tpl:

```
<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/faq">FAQ</a></li>
  <li><a href="/contact">Contact</a></li>
</ul>
```

# Heist: apply (2)

home.tpl:

```
<body>
  <h1>Home Page</h1>
  <apply template="nav"/>
  <p>Welcome to our home page</p>
</body>
```

# Heist: apply (3)

The navigation bar can make use of a sub-tree provided by apply tag.

nav.tpl:

```
<p> you are on page: <content/></p>
<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/faq">FAQ</a></li>
  <li><a href="/contact">Contact</a></li>
</ul>
```

# Heist: apply (4)

home.tpl:

```
<body>
  <h1>Home Page</h1>
  <apply template="nav">
    home
  </apply>
  <p>Welcome to our home page</p>
</body>
```

# Heist: apply (5)

Advanced topics:

- Emulating multiple <content/>s
- Splices (programming xml sub-trees with Haskell)

# Heist snaplets (1)

```
data App = App
    { _heist :: Snaplet (Heist App)
    , ...

instance HasHeist App where
    heistLens = subSnaplet heist

app :: SnapletInit App App
app = makeSnaplet name desc Nothing $ do
    h <- nestSnaplet "" heist $ heistInit "templates"
    ...
    return $ App h ...
  where
    name = "app"
    desc = "An snaplet example application."
```

# Heist snaplets (2)

```
handleLogin :: Handler App (AuthManager App) ()
handleLogin = heistLocal (bindSplices errs) $
                    render "login"
  where
    errs :: [(Text, Splice m)] = ...

routes = [ ("/login", with auth handleLogin)
      ...
```

# Heist snaplets (3)

You can also define a generic handler and a generic route for all paths with a certain prefix:

...

# Not Covered

- `Snap.Snaplet.Session`
- `Snap.Snaplet.Auth`
- Advanced topics in routing (there was a tutorial on that online somewhere?)
- digestive-functors (https://github.com/tjroth/snapbase-proj.git)

# Homework :)

Snap efficient and robust, but it lacks infrastructure. Porting simple little features from other frameworks such as rails into snaplets would be a good exercise. The top 6 of the authors of this slide set:

- tag cloud
- user preferences editor
- blog aka newsticker aka forum
- multi-user file system
- glue for JS text editor (possibly trivial)
- wiki