ROSS: Past, Present, and Future

Elsa Gonsiorowski
July 12, 2016

Summer of CODES 2016!!

Development Timeline

1999	GA Tech Time Warp implementation by Shawn Pearce and Dave Bauer
May 2009	ROSS v5.0 released on Source Forge
Aug 2013	ROSS released on GitHub
Jan 2015	Simplified ROSS
Apr 2015	Travis CI Testing
Feb 2016	ROSS Blog

Feature Additions

Last April	Optimistic Realtime Scheduler
Summer of CODES	Lamport Path Statistic Commit Function RIO v2
Developing	Visualization
Future	Large Message Support Preemptive Termination Event Retraction Ties

ROSS Schedulers

- Sequential `--sync=1`
- Conservative `--sync=2`
- Optimistic `--sync=3`
- Optimistic Debug `--sync=4`
    - Non-parallel
    - Serial execution until out of events, then rolls back to time zero
- Optimistic Realtime `--sync=5`
    - Typical Time Warp scheduler, interrupt for GVT based on wall-clock-time elapsed

Lamport Path

- Each LP and Event has a *Lamport Path* counter
- For each LP the count starts at 0
- For each new event, set path to LP’s path +1
- Just before each forward event handler,
  LP’s path is updated to MAX(LP’s Path, Event’s Path)+1

Lamport Path
Implementation

- Path should be deterministic!
- State-save LP’s path value before updating for an event
- Use event memory as buffer
- But only after event has been processed

```
// state-save and update the LP’s critical path
unsigned int prev_lpath = lp->lpath;
lp->lpath = MAX(lp->lpath, ev->lpath) + 1;
(*lp->type->event)(…);
ev->lpath = prev_lpath;
```

Lamport Path
Implementation

- Path should be deterministic!
- State-save LP’s path value before updating for an event
- Use event memory as buffer
- But only after event has been processed

```
// state-save and update the LP’s critical path
unsigned int prev_lpath = lp->lpath;
lp->lpath = MAX(lp->lpath, ev->lpath) + 1;
(*lp->type->event)(…);
ev->lpath = prev_lpath;
```

Lamport Path
Implementation

- Path should be deterministic!
- State-save LP’s path value before updating for an event
- Use event memory as buffer
- But only after event has been processed

```
// state-save and update the LP's critical path
unsigned int prev_lpath = lp->lpath;
lp->lpath = MAX(lp->lpath, ev->lpath) + 1;
(*lp->type->event)(…);
ev->lpath = prev_lpath;
```

Lamport Path
Implementation

- Path should be deterministic!
- State-save LP's path value before updating for an event
- Use event memory as buffer
- But only after event has been processed

```
// state-save and update the LP's critical path
unsigned int prev_lpath = lp->lpath;
lp->lpath = MAX(lp->lpath, ev->lpath) + 1;
(*lp->type->event)(…);
ev->lpath = prev_lpath;
```

Lamport Path
Implementation

- Path should be deterministic!
- State-save LP’s path value before updating for an event
- Use event memory as buffer
- But only after event has been processed

```
// state-save and update the LP’s critical path
unsigned int prev_lpath = lp->lpath;
lp->lpath = MAX(lp->lpath, ev->lpath) + 1;
(*lp->type->event)(…);
ev->lpath = prev_lpath;
```

Critical Path

- Goal: Determine the absolute minimum running time for a simulation
  - Mapping Independent
  - Processing time for an event
- Find maximum length critical path

Critical Path

Implementation

```
// state-save and update the LP’s critical path
unsigned int prev_cp = lp->cp;
lp->cp = MAX(lp->cp, ev->cp);
lp->cycle = get_cycle()
(*lp->type->event)(…);
 //tw_event_new:
 //   ev->cp = lp->cp + get_cycle() - lp->cycle
lp->cp += get_cycle() - lp->cycle;
ev->cp = prev_cp;
```

Critical Path

Implementation

```
// state-save and update the LP's critical path
unsigned int prev_cp = lp->cp;
lp->cp = MAX(lp->cp, ev->cp);
lp->cycle = get_cycle()
(*lp->type->event)(…);
 //tw_event_new:
 //   ev->cp = lp->cp + get_cycle() - lp->cycle
lp->cp += get_cycle() - lp->cycle;
ev->cp = prev_cp;
```

Critical Path

Implementation

```
// state-save and update the LP's critical path
unsigned int prev_cp = lp->cp;
lp->cp = MAX(lp->cp, ev->cp);
lp->cycle = get_cycle()
(*lp->type->event)(…);
 //tw_event_new:
 //   ev->cp = lp->cp + get_cycle() - lp->cycle
lp->cp += get_cycle() - lp->cycle;
ev->cp = prev_cp;
```

Commit Function

- Simple callback, similar to event handlers

```
void commit(state *s, msg *m,
            tw_bf *bf, tw_lp *lp);
```

```
tw_lptype  {(init_f)    init,
            (pre_run_f) prerun,
            (event_f)   forward,
            (revent_f)  reverse,
            (commit_f)  commit,
            (final_f)   finish,
            (map_f)     map,
            sizeof(state)}
```

Commit Function
Scheduling

- Sequential and Conservative:
  called after an event is processed
- Optimistic and Optimistic Realtime:
  called during event fossil collection after GVT
  in the order they were initially processed
- Optimistic Debug:
  not called

RIO Version 2

- ROSS IO
- Library for *Checkpoint Restart*
- Used in 5.9 million LP gates simulation
- Simplified API
- Documentation (on the blog)
- Optimistic Mode!

RIO API
Model Callbacks

- LP Size

```
size_t lps (void *state, tw_lp *lp);
```

- LP Serialize

```
void ser (void *state, void *buffer, tw_lp *lp);
```

- LP Deserialize

```
void ds (void *state, void *buffer, tw_lp *lp);
```

RIO API
Global Variable Settings

- `io_lptypes g_io_lp_types[ ]`
- `unsigned int g_io_number_of_files, --io-files=n`
- `unsigned int g_io_events_buffered_per_rank`

RIO API
Function Calls

- Init after setting vars, before tw_run

```
void io_init();
```

- Invoke load before calling tw_run

```
void io_load_checkpoint(char *cp_name,
                  [PRE_INIT, INIT, POST_INIT] );
```

- Invoke store after calling tw_run (before tw_end)

```
void io_load_checkpoint(char *cp_name, int dfn);
```

Ties in ROSS currently

- Users must add random jitter to each event
  - Non-Obvious to model developers
  - Accumulated impact on time
  - Still possible to have tie events

```
tw_event_new(destination,
             tw_rand_exponential(lp->rng, mean)
              + lookahead,
             lp)
```

- Statistic counter for number of ties detected during run

Ties
Desired Mechanism

- Symmetric across all ties
- Invariant on IDs, receiver, sender
- **Determined by the programmer**
    - Option for a callback implemented by model
    - Deterministic default policy included by ROSS
- Since it will not be rolled back,
  tie-break function cannot modify LP state

Ties

Events Trigger Actions

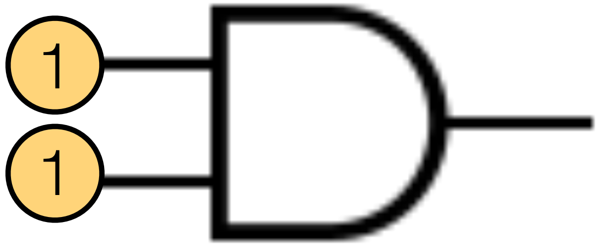
- A set of events trigger an ordered set of actions

1. Set of tied events is passed to a tie-breaker callback
    - Reorder the list
    - Remove an event/action
    - Add an event/action
2. Process each event in the returned list separately

Ties

Alternate Use Case 1: Logic Gates

- Ties are allowed: All-at-once strategy
- Give the event handlers all events that tie
    - Linked list, deterministically ordered by ROSS



- This set of events is treated as a single event for causality/ rollback purposes
- ==> must be handled by model.

Ties

Alternate Use Case 2: Real “Random” Ordering

- Some models may in fact want a random ordering
- Calling the RNG represents a change of LP state
- *Is there a way to support deterministic, yet random ordering without having to rollback the tie breaker function?*

Ties

Possible Future Implementation

- Tie-breaking callback function

```
void tie_break(state *s, tw_lp *lp, tw_event *evs);
//we need both |- - LP Info - - - ||- event set -|
```

- This function should order the *actions* taken from this set of events (ROSS terminology: *action* = *message*)
- Two new ROSS API functions: `tw_msg_free`, `tw_msg_new` which turn on/off a “do_action” flag
- Model developer must manipulate linked-list pointers

Ties

Possible Future Implementation

- Issue: where does the “process these actions” loop occur?

- Answer: in ROSS, not the event handler

- model can figure out something special to handle All-at-once case.

Ties
Caveats

- How should we deal with the addition/removal of events in tie breaker function?
- Potential for N rollbacks for N tie events at the same LP
- What about zero-delay messages?
  - *Should we even allow users to send zero-delay messages?*
  - Possible solution: implement a hidden “time” field
  - Implementation shouldn’t impact lazy cancellation

Ties

Rollback

1. ROSS orders the tied event set
2. ROSS calls model’s tie-breaker function
3. Model function edits the event set: sort/add/remove
   The remove method simply flips a bit to turn off processing
4. ROSS calls event handler for each event in the set
5. Anti-message for an event in the set arrives
6. Each forward event is rolled back
7. Anti-message destroys the event in the set
8. ROSS orders the tied event set:
   previous add/remove actions are undone

ROSS Improvements

Additional Features

GitHub Repository

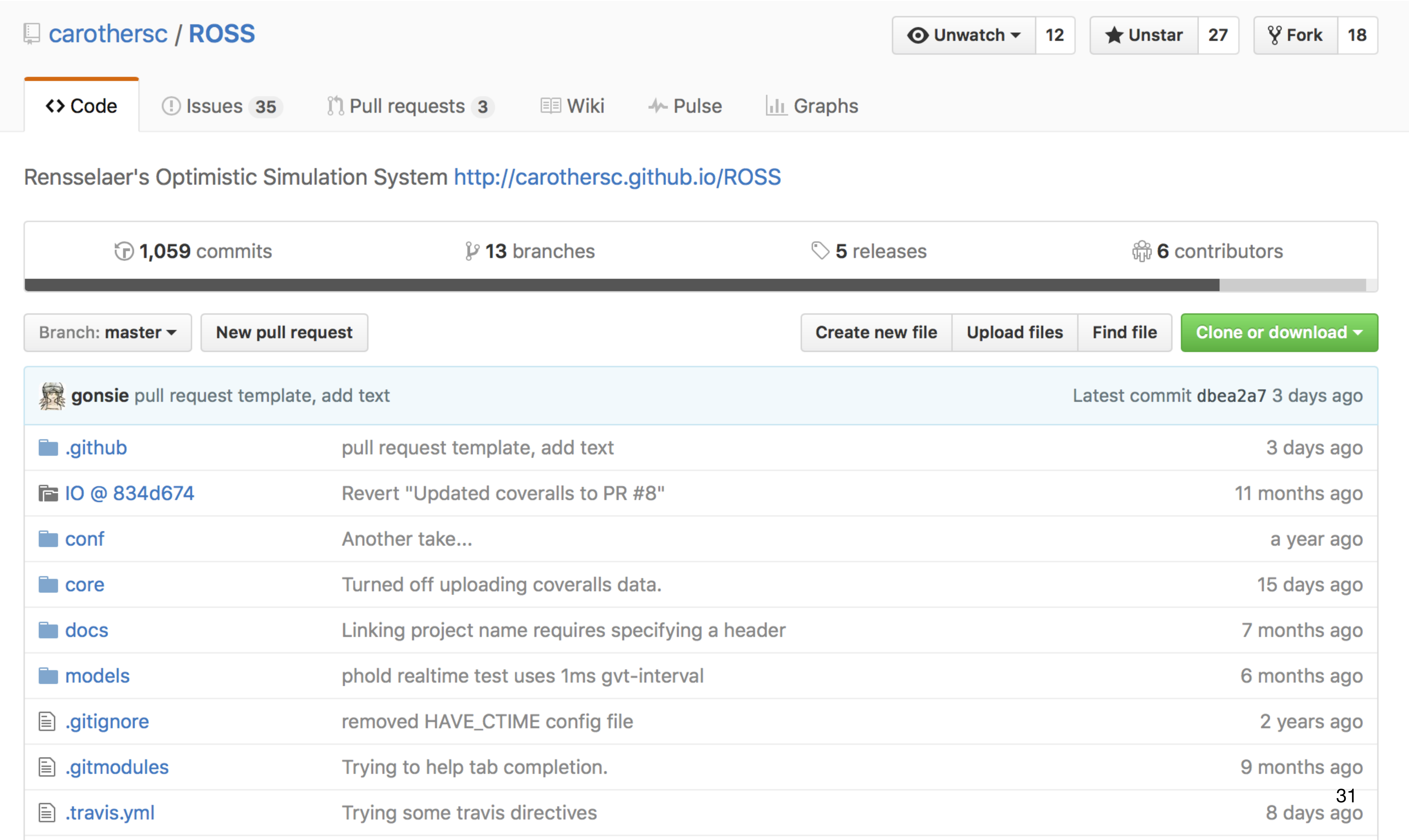
- GitHub Issue
- Development
- Test
- Pull Request

Publicity and Documentation

Website

- Documentation

GitHub



Rensselaer's Optimistic Simulation System http://carothersc.github.io/ROSS

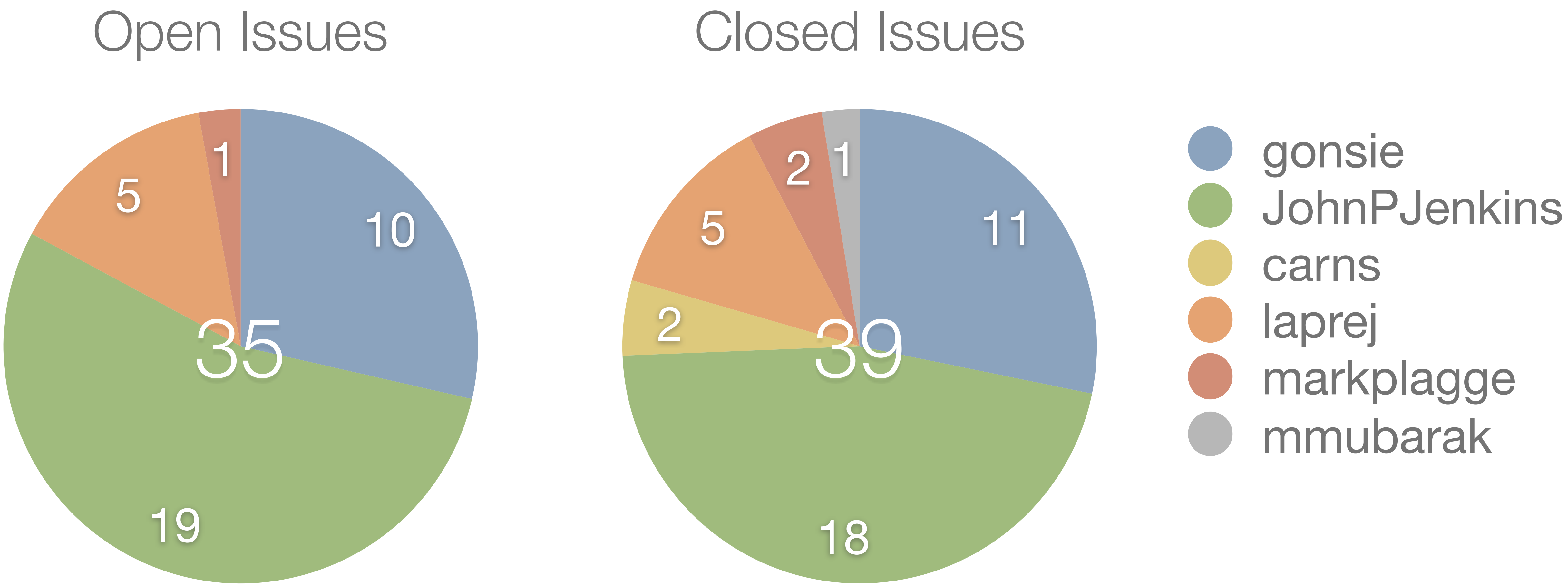
gonsie pull request template, add text — Latest commit dbea2a7 3 days ago

.github	pull request template, add text	3 days ago
IO @ 834d674	Revert "Updated coveralls to PR #8"	11 months ago
conf	Another take...	a year ago
core	Turned off uploading coveralls data.	15 days ago
docs	Linking project name requires specifying a header	7 months ago
models	phold realtime test uses 1ms gvt-interval	6 months ago
.gitignore	removed HAVE_CTIME config file	2 years ago
.gitmodules	Trying to help tab completion.	9 months ago
.travis.yml	Trying some travis directives	8 days ago

GitHub
Issues

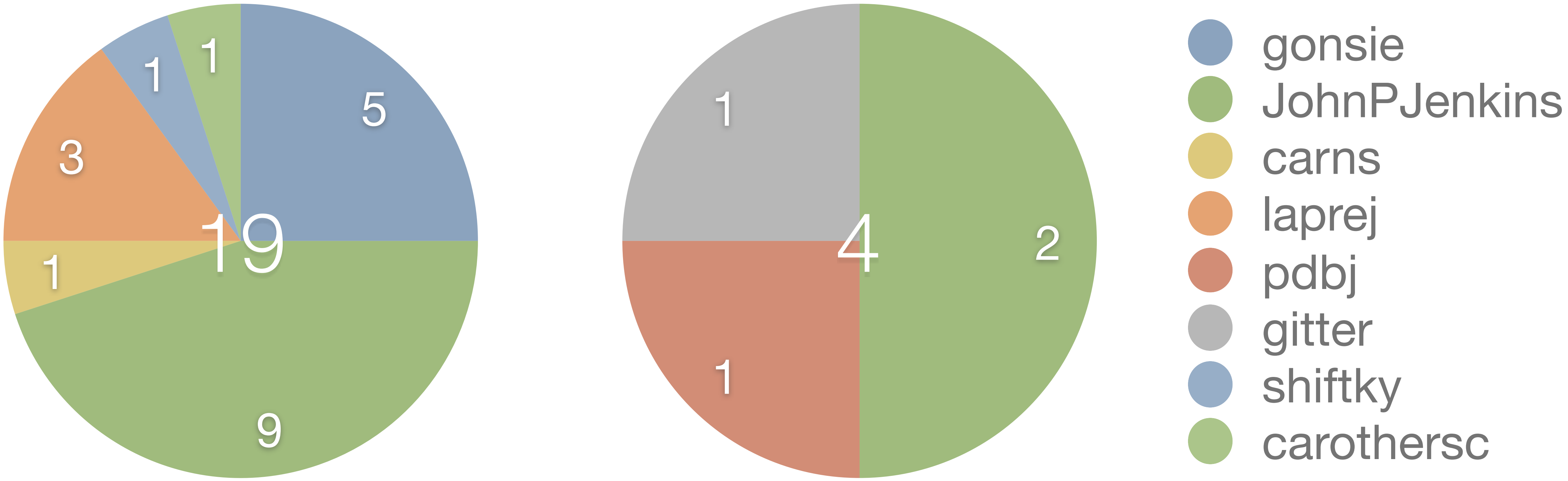
- Active Website & Blog: ross.cs.rpi.edu

- Hosted on GitHub: github.com/carothersc/ROSS



GitHub
Pull Requests

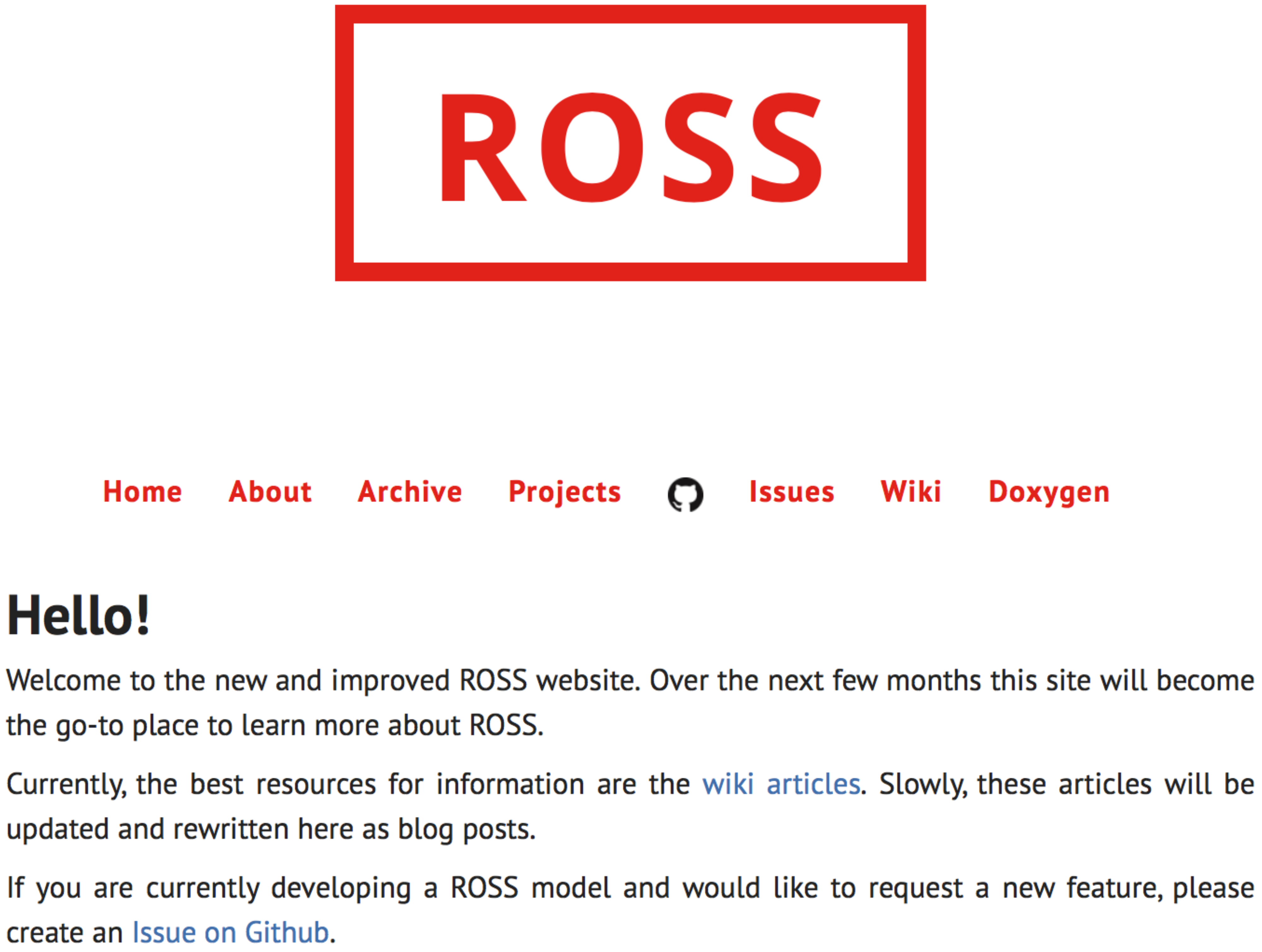
- Active Website & Blog: ross.cs.rpi.edu
- Hosted on GitHub: github.com/carothersc/ROSS



Accepted Pull Requests

Declined Pull Requests

GitHub
Website: gh-pages branch



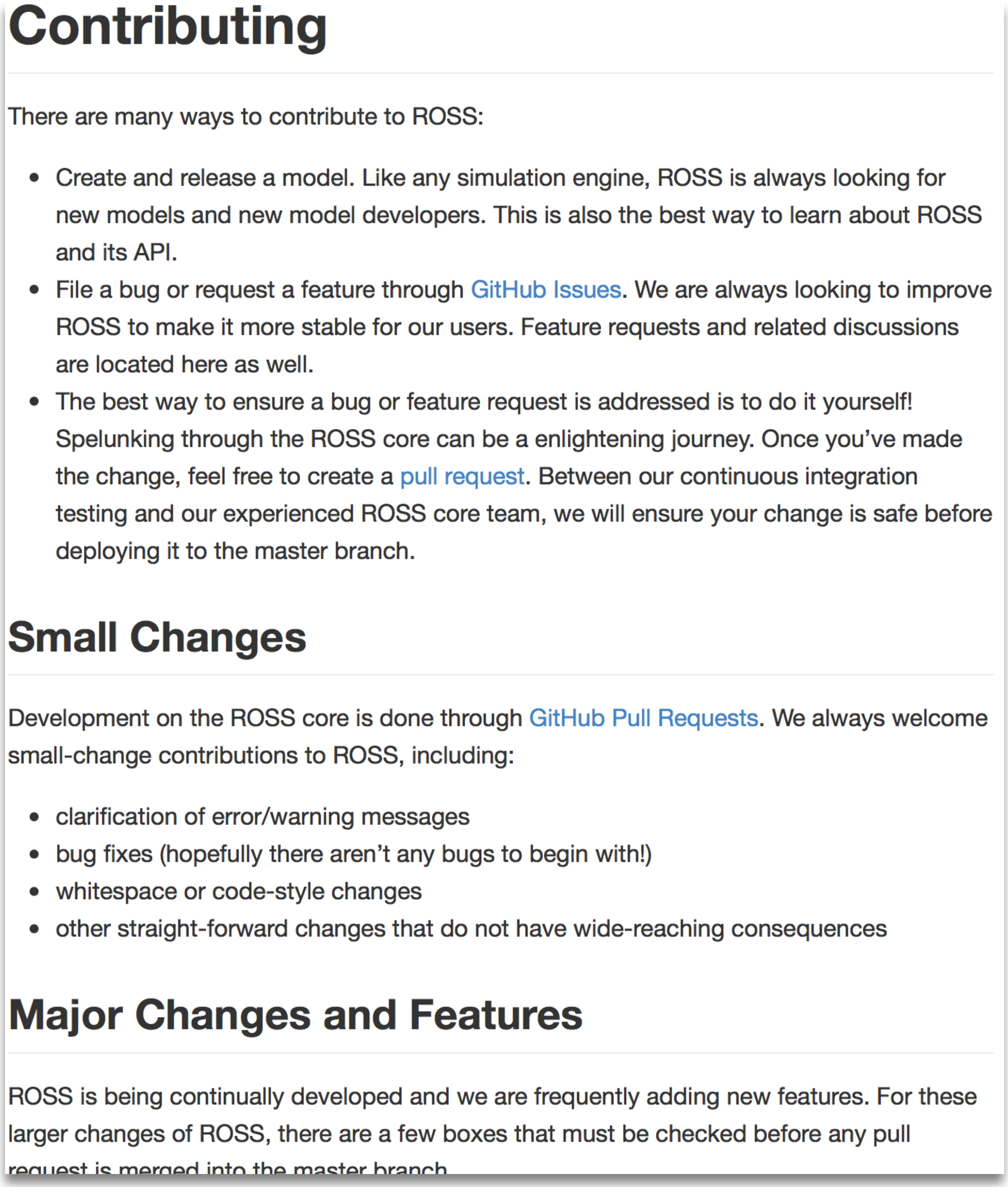
Hello!

Welcome to the new and improved ROSS website. Over the next few months this site will become the go-to place to learn more about ROSS.

Currently, the best resources for information are the wiki articles. Slowly, these articles will be updated and rewritten here as blog posts.

If you are currently developing a ROSS model and would like to request a new feature, please create an Issue on Github.

GitHub
Contributing Guides



Contributing

There are many ways to contribute to ROSS:

- Create and release a model. Like any simulation engine, ROSS is always looking for new models and new model developers. This is also the best way to learn about ROSS and its API.
- File a bug or request a feature through GitHub Issues. We are always looking to improve ROSS to make it more stable for our users. Feature requests and related discussions are located here as well.
- The best way to ensure a bug or feature request is addressed is to do it yourself! Spelunking through the ROSS core can be a enlightening journey. Once you've made the change, feel free to create a pull request. Between our continuous integration testing and our experienced ROSS core team, we will ensure your change is safe before deploying it to the master branch.

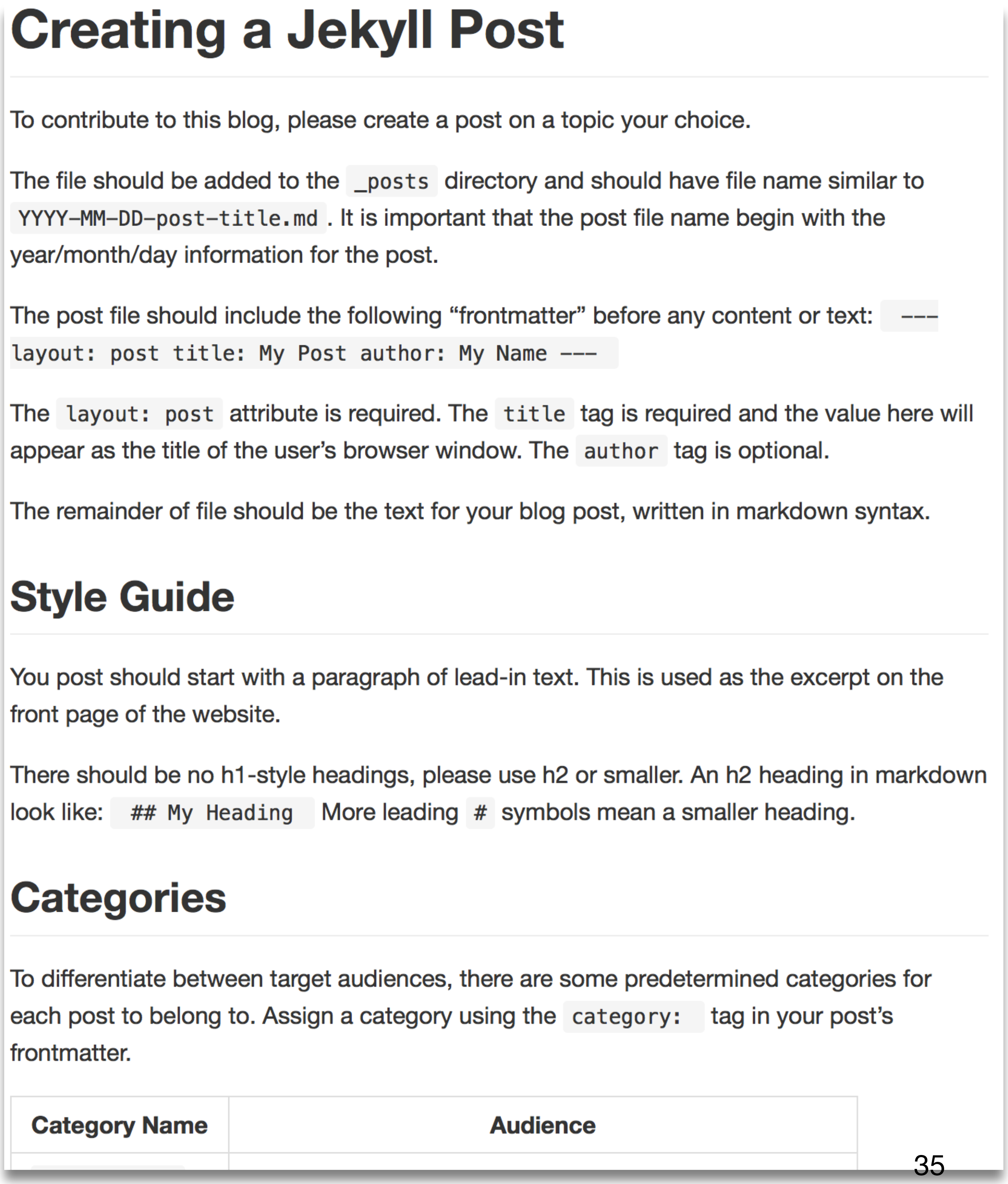
Small Changes

Development on the ROSS core is done through GitHub Pull Requests. We always welcome small-change contributions to ROSS, including:

- clarification of error/warning messages
- bug fixes (hopefully there aren't any bugs to begin with!)
- whitespace or code-style changes
- other straight-forward changes that do not have wide-reaching consequences

Major Changes and Features

ROSS is being continually developed and we are frequently adding new features. For these larger changes of ROSS, there are a few boxes that must be checked before any pull request is merged into the master branch.



Creating a Jekyll Post

To contribute to this blog, please create a post on a topic your choice.

The file should be added to the `_posts` directory and should have file name similar to `YYYY-MM-DD-post-title.md`. It is important that the post file name begin with the year/month/day information for the post.

The post file should include the following "frontmatter" before any content or text: `--- layout: post title: My Post author: My Name ---`

The `layout: post` attribute is required. The `title` tag is required and the value here will appear as the title of the user's browser window. The `author` tag is optional.

The remainder of file should be the text for your blog post, written in markdown syntax.

Style Guide

You post should start with a paragraph of lead-in text. This is used as the excerpt on the front page of the website.

There should be no h1-style headings, please use h2 or smaller. An h2 heading in markdown look like: `## My Heading` More leading `#` symbols mean a smaller heading.

Categories

To differentiate between target audiences, there are some predetermined categories for each post to belong to. Assign a category using the `category:` tag in your post's frontmatter.

Category Name	Audience

Get Involved

- Add completed models to
  github.com/carothersc/ROSS-Models

- Request a feature or report an issue:
  GitHub Issues

- Make a change
  .github/CONTRIBUTING.md guide, PR template

- Write a blog post!
  CONTRIBUTING.md guide in gh-pages branch

You are the future!

Thank you