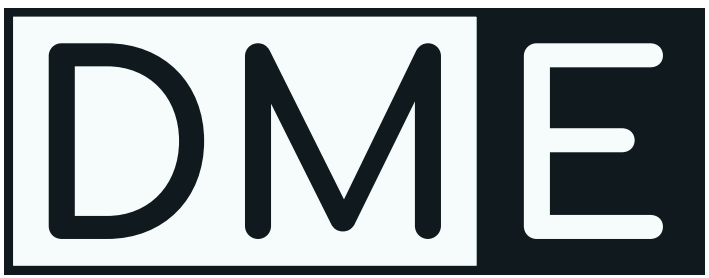
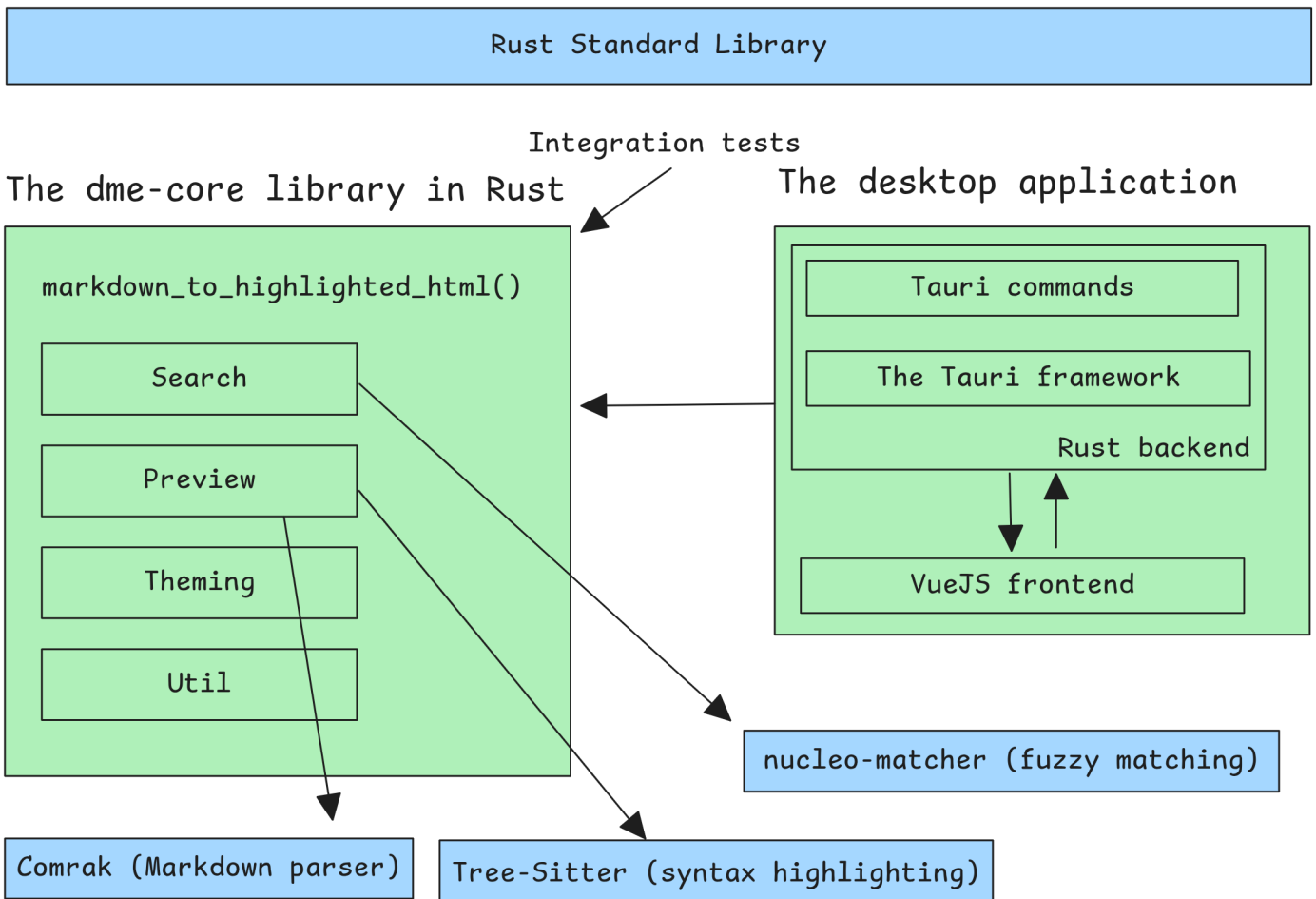



```
cargo test
cargo build --release
```

Ready to enter the Delightful Markdown Experience ?

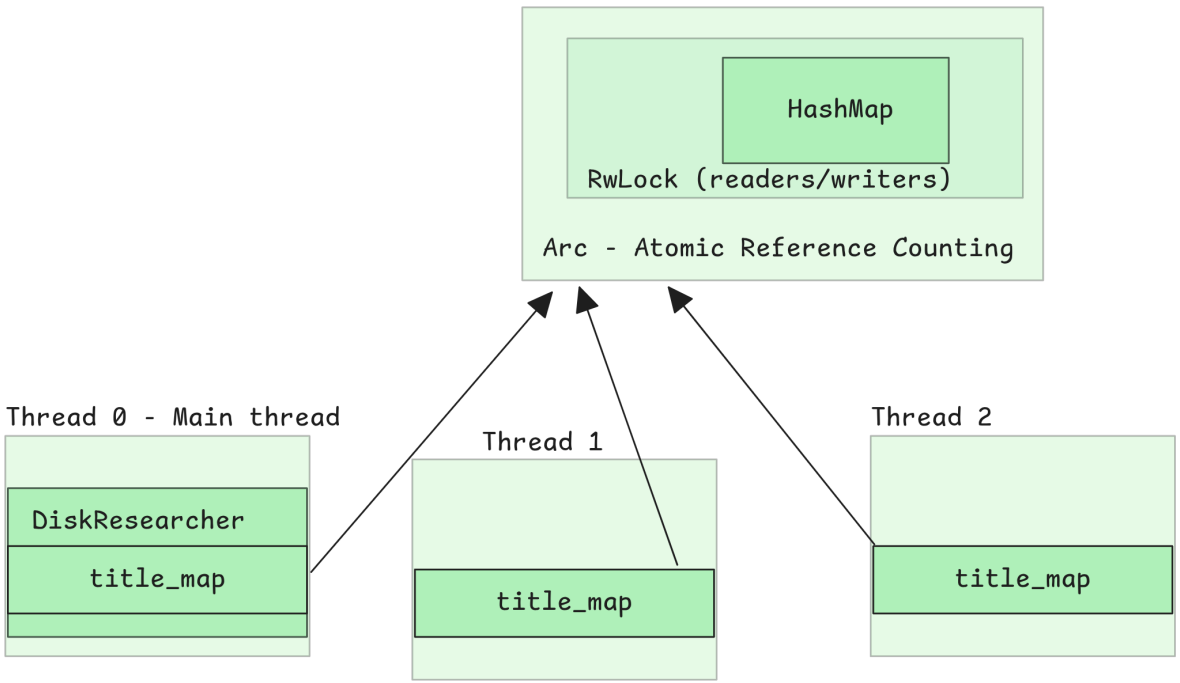
Demo !

Global architecture



Search strategy

- Split the data
- Prepare shared ressource
- Computation
- Update shared ressource




```
pub struct DiskResearcher {
    markdown_paths_set: Arc<RwLock<Vec<String>>>,
    title_map: Arc<RwLock<HashMap<String, Vec<String>>>>,
    // ...
}
```

Search strategy


```
for chunk in all_paths.chunks(chunk_size) {
    let chunk = chunk.to_vec(); // copy chunk
    let title_map = Arc::clone(&self.title_map);

    thread::spawn(move || {
        for path in chunk {
            let titles = DiskResearcher::extract_markdown_titles(&path);
            {
                let mut map = title_map.write().unwrap();
                for title in titles {
                    map.entry(title).or_default().push(path.clone())
                }
            }
```

Syntax highlighting with Tree-sitter

tree-sitter.json

```
{
  "grammars": [
    {
      "name": "css",
      "camelcase": "CSS",
      "scope": "source.css",
      "path": ".",
      "file-types": [ "css" ],
      "highlights": "queries/highlights.scm",
      "injection-regex": "^css$"
    }
  ],
  // ..
}
```

queries/highlights.scm

```
"~" @operator
">" @operator
"+" @operator
"-" @operator
...

(class_name) @property
(id_name) @property
(namespace_name) @property
```

```
<span class="operator">+</span>
```

grammar.js

```
...
 rules: {
    stylesheet: $ =>
repeat($._top_level_item),

    // Statements
    import_statement: $ => seq(
      '@import',
      $._value,
      sep(',', $._query),
      ';',
    ),
...
```

catpuccin-latte.toml

```
"constant" = "peach"
"constant.character" = "teal"
"constant.character.escape" = "pink"

"string" = "green"
...

[palette]
rosewater = "#dc8a78"
flamingo = "#dd7878"
pink = "#ea76cb"
```

C parser

```
src> tree
├── grammar.json
├── node-types.json
├── parser.c
├── scanner.c
└── tree_sitter
    ├── alloc.h
    ├── array.h
    └── parser.h
```

```
~/.local/share/tree-sitter-grammars>
tree-sitter-c
tree-sitter-cpp
tree-sitter-css
tree-sitter-csv
...
```

Highlighting steps

- Download, compile, load
- Language configuration
- Load a highlighter
- Highlight some code
- Render HTML
- Include in bigger doc

```
// git clone --depth 1 --single_branch
let only_latest_commits: Option<u32> = Some(1);
let mut args = vec!["clone", git_clone_url];
if let Some(count) = only_latest_commits {
    args.push("--depth");
    args.push(&count.to_string());
}
if single_branch {
    args.push("--single-branch");
}

args.push(&count.to_string());
          ^^^^^^^^^^^^^^^^^^ - temporary value is freed at the end of
this statement
          |
          creates a temporary value which is freed while still in use

args.push("--single-branch");
---- borrow later used here
```

Future of DME ?

- Making syntax highlighting parallel
- Making grammars installation parallel
- Making a full text search
- Themes management
- PDF export
- Visual theme configuration
- More benchmarking and optimisations

Our experience with the paradigm

- Avoided thousands of possible errors
- Hard to think about advanced memory references
- No memory crash at runtime

Our experience with Rust

- The standard library
- Tree-Sitter library
- Unit and integration testing
- Type expressiveness
- Be forced to manage errors
- Liked the functionnal part of Rust
- Compilers contextual errors
- Proposed fixes and refactoring