

Second Year Report

Patrick Ferris

*Department of Computer Science and Technology
Pembroke College*

Supervisor: *Prof. Anil Madhavapeddy*
Cosupervisor: *Prof. Srinivasan Keshav*

June 30, 2024

Contents

1. Introduction	3
1.1. Reading this Report	3
2. A Year in Review	4
2.1. Progress	4
2.1.1. Shelter	4
2.1.2. Void Processes	5
2.1.3. eBPF Opentrace	6
2.1.4. Experiments in computational, authoring systems	6
2.1.5. In relation to the first year report	7
2.2. Teaching	7
2.2.1. Part II Projects	8
3. Thesis Outline	9
3.1. Chapter 2: Background & Related Work	9
3.2. Chapter 3: Infrastructure for a Computational Wiki	9
3.3. Chapter 4: Front-end for a Computational Wiki	9
3.4. Chapter 5: Content for a Computational Wiki	10
3.5. Chapter 6: Conclusion	10
4. Timeline	11
5. Talks and Publications	12
5.1. Published	12
5.1.1. Preprints	13
5.2. In Review	13
5.3. Working Papers	13
Bibliography	14
Index of Figures	15
Index of Listings	15

1. Introduction

A brief reminder that in my first year report the central thesis was to provide

... a combination of specification authoring systems with tight integration into a system for reproducible scientific programming.

Phrased differently, I am looking at how we can improve the interface between computer programming and data science to help express scientific methodologies and improve readability and reproducibility.

1.1. Reading this Report

Similarly to last year in my first year report, there are a few reading aids scattered across this shorter report.

? Question

Questions are moments to engage the reader in directions my mind are going. They could make for discussion points with assessors.

⚡ Concern

Concerns are tasks, ideas and goals that I feel might be hard to meet. They *should* be accompanied with some resolution to help mitigate the concern.

2. A Year in Review

As big soft buffetings come at the car sideways
And catch the heart off guard and blow it open.
— from *Postscript* by Seamus Heaney

2.1. Progress

In my first year report the timeline consisted of three main segments: uncertainty in climate science, the design of a specification language called Joe and ideas about a computational wiki. Most of the work this year, contrary to the timeline, focused on the building blocks for the computation wiki. The primary work was building Shelter.

2.1.1. Shelter

Shelter is what I am calling a *metashell*, an environment around your usual shell session that helps manage processes and resources to aid in reproducible data science.

```
shelter> echo hello > hello.txt
shelter[main#bbd691a] : { mode: rw }> @ session exp1
shelter[exp1#bbd691a] : { mode: rw }> echo world >> hello.txt
shelter[exp1#dd5bab8] : { mode: rw }> @ session main
shelter[main#bbd691a] : { mode: rw }> echo "to the" >> hello.txt
shelter[main#b9abef4] : { mode: rw }> cat hello.txt
hello
to the
shelter[main#d1c1728] : { mode: rw }> @ session exp1
shelter[exp1#dd5bab8] : { mode: rw }> @ rebase main
shelter[exp1#df1b4e1] : { mode: rw }> cat hello.txt
hello
to the
world
```

Listing 1: An example `shelter` session

Listing 1 shows a minimal example of using Shelter starting with a fresh session. Internally, sessions are persisted in a Git-like store, with each session corresponding to a branch. Rebasing mechanisms allow users to layer their work on top of other sessions. This capability makes it straightforward to implement an `@ undo` meta-command¹.

In Figure 1, we give a diagrammatic introduction to the core meta-commands available to the user. I am currently drafting a paper on Shelter for EuroSys 2026 (see Section 5.3).

¹We distinguish between shell commands and meta-commands for controlling aspects of Shelter.

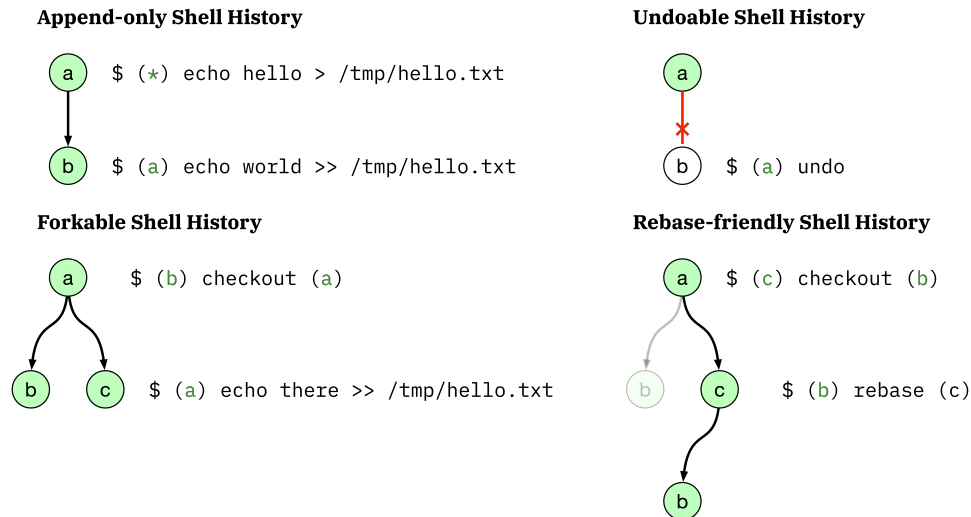


Figure 1: Core meta-commands in Shelter

The backend of Shelter uses a two-tier storage model. On-disk, data is stored using ZFS [7]. Layered on top is a metadata store that makes use of mergeable replicated data types [6]. This allows for a fairly decentralised storage model where metadata is easily synchronised across nodes for discoverability whilst large data can be lazily transported using built-in ZFS sending and receiving.

? Privacy models in a Git-like store?

If a suitably cryptographic hash function underlies a git-like store then certain security properties can be guaranteed. However, it would be good to add some encryption model into both layers reusing ZFS's encryption primitives.

2.1.2. Void Processes

As part of this work, I developed Jake Hillion's master thesis on *void processes* further by building an OCaml library for manipulating them [5]. The idea is to have tighter control over a process' capabilities, forcing a notion of least-privilege onto Linux's namespacing. Listing 2 shows the creation of a void process that only has access to the Alpine root-filesystem (readonly) and some mounted test data (also readonly).

```

1 Eio.Switch.run @@ fun sw ->
2 (* Docker base image fetching... *)
3 let alpine_img = get_alpine_image ~fs ~proc in
4 (* Test data *)
5 let mount_src = test_data fs "Hello, World!" in
6 let args = ["/bin/ash"; "-c"; "/bin/cat /data/data.txt"];
7 in
8 (* Creation of a void process specification *)
9 let void =
10   let open Void in
11     empty
12   |> rootfs ~mode:R alpine_img
13   |> mount ~mode:R ~src:mount_src ~tgt:"data"
14   |> exec args
15 in
16 let t = Void.spawn ~sw void in
17 (* ... *)
  
```

Listing 2: An example of using *void* an OCaml library for building void processes

? Is everything a workshop paper?

Void processes are a good idea, but they have not been fully researched and turned into a paper. To what extent should smaller, self-contained ideas be turned into papers even if only at the workshop-tier?

2.1.3. eBPF Opentrace

Another component of Shelter is a low-level tracing system to understand read-write dependencies of a process. I developed an eBPF-based program called `opentrace`² to help track reads and writes to files.

This has quite a few more features than other available tools including (limited) `io-uring` support and cgroup-based child process tracking. For example, in the environments that `opentrace` must run in, it is common for a shell to spawn a process that also spawns many more processes. These all need to be tracked to understand the reads and writes that are being performed. Consider the OCaml program in Listing 3.

```
1 let () =
2   Eio_posix.run @@ fun env ->
3   Eio.Path.(save ~create:(`Or_truncate 0o664) (env#fs / "hello.txt") "hello") ;
4   Eio.Process.run env#process_mgr
5   [ "/bin/bash"; "-c"; "echo 'heya' > heya.txt" ]
```

Listing 3: An OCaml program that writes data and spawns a process that also writes data

Ideally, we need to know that both `hello.txt` and `heya.txt` are created by executing the program.

```
1 $ sudo opentrace exec --cgroups --flags="O_CREAT|O_TRUNC" ./main.exe
2 pid,cgid,comm,kind,flags,mode,filename,return
3 153187,530807,"main.exe",openat,526914,436,"hello.txt",5
4 153192,530807,"bash",openat,577,438,"heya.txt",3
```

Listing 4: The output from running `opentrace` on the program from Listing 3

2.1.4. Experiments in computational, authoring systems

Thus far, I have discussed building blocks for the infrastructure of a computational wiki (see Section 3.2 from the thesis outline).

Another core aspect of the system I propose is an authoring system for scientific methodologies. This remains the least developed section of my research (see Section 3.3). However, I have been experimenting with novel ideas in this space including Forester, a self-proclaimed “tool for scientific thought”.

Forester presents a $\text{\LaTeX}$ -inspired frontend to a scientific authoring system centred on the notion of *forests* – collections of trees formed by grouping atomic, independent ideas. From this, I built Graft, a compiler from Markdown to Forester syntax.

Consider an abridged copy of my weekly report from June 2nd 2025: Forester as a Target Syntax.

²Available on tangled.sh

```

1  ---
2  title: Forester as a Target Syntax
3  author: Patrick Ferris
4  date: 2025-06-02
5  ---
6
7  ~~~forester
8  \put\transclude/numbered{false}
9  ~~~
10
11 This week included some time finishing [opentrace](open-trace) and subsequently
12 folding it into [shelter](). I have been writing up some more of the draft paper
13 for [shelter]() which I am excited to share in the near future.
14
15 ## Forester and Graft
16
17 I spent a bit of time finally pulling out my changes to [Forester]() to add markdown
18 and bibtex support into a standalone tool: [graft]().
19
20 ~~~forester
21 \transclude{graft}
22 ~~~
23
24 It is very satisfying to find the separation of concerns works quite well...
25
26 ### Maths Support
27
28 As part of that process, [graft]() now supports Markdown KaTeX. For example:
29
30 ~~~forester
31 \transclude{mrdts}
32 ~~~

```

Listing 5: A markdown document that is compiled to Forester

Via the markdown codeblocks we can escape back into Forester syntax for the more involved commands (like transclusion, a method of *including* one tree inside another). All of the links in this example are links to other trees.

This atomicity seems like a promising idea for the structure of a scientific authoring tool, but in discussions with its author we agreed there should be room for scientific narration too.

2.1.5. In relation to the first year report

In reference to my original timeline in the first year report, the work this year has been primarily focused on the *Computational Wiki* collection of tasks which include, (1) exploring VURLs³, (2) integrating version control in Joe, and (3) building a decentralised storage model. In short, this year I have built and researched core aspects of the underlying systems that can support a computational wiki with a Joe-like specification language.

The work on Forester (Section 2.1.4) has mainly contributed to a better understanding of what might make a good authoring system for the computational wiki. Ideas are forming about whether or not bigraphs would make an appealing data-structure for describing the interconnectedness of such a wiki more formally [12]. I wish to explore the suitability of such a design choice and the potential benefits of the accompanying algebra and reactive system over the next year.

2.2. Teaching

This year, as last year, I taught two undergraduate courses in the Department of Computer Science and Technology: *Foundations of Computer Science* and *Discrete Mathematics*.

³As a reminder, these are *versioned* URLs

⚡ Too much teaching?

This year, I spent a lot of time teaching. I believe it is fair to say I went above and beyond what is strictly required of me as a supervisor. I did so because I love teaching. That being said, teaching does not usually entail a completed thesis. Therefore, I have decided to take on no students next year at all and instead focus on the thesis and the research. I hope to come back to teaching after the PhD.

2.2.1. Part II Projects

I supervised four final-year, undergraduate projects this year with Prof. Anil Madhavapeddy as my university teaching officer. I thoroughly enjoyed teaching this year, but it definitely took a lot of my time.

Type Error Debugging in Hazel

This dissertation added *type slices* and *cast slices* to the Hazel programming language to help debug type errors. This work led to the submission of an extended abstract to the 10th International Workshop on Type-Driven Development at ICFP (see Section 5.2).

A Universal Data Access DSL

This work further some ideas I had relating to Joe from my first year report around data description in a specification language. Concretely, a student implemented an idea Anil and I had about accessing data universally via a POSIX pathname-like interface.

Implementing Parallel Algebraic Effect Handlers in OCaml

This dissertation explored the work of Xie et al. on *parallel algebraic effect handlers* in OCaml [14]. Concretely this added a new `traverse` handler for effects that allowed users to programmatically control how a data-structure (e.g. a tree) would be traversed, including in a parallel way.

Effective Geospatial Code in OCaml

Finally, a student looked to reimplement much of Yirgacheffe⁴ in OCaml, making use of the type system and compilation to hopefully improve performance for certain tasks.

⁴<https://github.com/quantifyearth/yirgacheffe/>

3. Thesis Outline

“What do you mean?” asked the Ghost.
“Why, if you are interested in the country only for the sake
of painting it, you’ll never learn to see the country.”
— from *The Great Divorce* by C.S. Lewis

I am not tied to these chapter titles. I believe they will change quite significantly, but for now they act as means to convey the *narrative* I wish to tell in my thesis. In what comes next, there is a color-coded system (with symbolic disambiguation) to highlight the progress across three axes for each chapter: is the chapter written, how is the research for the chapter going and if there is any associated paper with the research in the chapter.

Not Started ✗ Started ≈ Good Progress + Completed ✓ Not Applicable ∅

Before outlining the thesis, I will reiterate the core objective: *to build a scientific authoring system that supports reproducible, computational workflows.*

3.1. Chapter 2: Background & Related Work

Chapter ≈ Research ∅ Paper ∅

This chapter can draw on parts of the literature review in the first year report alongside my own works this year⁵. This chapter will always be in a state of flux as chapters 3, 4 and 5 change. I have also built up a healthy Zotero library of references.

Going forward, I would like to improve my wider reading of papers. My particular area of research unfortunately straddles systems, a bit of theory and elements of HCI which makes it hard to find a single good source of research. I have found my weekly (or almost weekly) logs a valuable addition to my tools to help consolidate and collect ideas.

3.2. Chapter 3: Infrastructure for a Computational Wiki

Chapter ✗ Research + Paper +

This chapter will consist primarily of a recontextualised version of the paper associated with Shelter. The core research is summarised in Section 2.1.1 with the draft paper discussed in Section 5.3.

3.3. Chapter 4: Front-end for a Computational Wiki

Chapter ✗ Research ≈ Paper ✗

This chapter will focus on aspects like *literate programming*, *typesetting* and Joe.

⁵The eager reader may find my weeknotes and posts useful.

⚡ Least developed section

This part of the research so far is the least developed. Whilst I have been exploring this space of ideas for a long-time, I am not as familiar with the latest research.

Alongside the paper writing for Shelter (Section 2.1.1), this is my next task in the my timeline (Section 4).

In theory, this chapter will discuss the implementation of Joe. I refer readers back to Section 4.2 of the first year report for the ideas underlying this chapter.

3.4. Chapter 5: Content for a Computational Wiki

Chapter ≈ Research ≈ Paper ∅

Acting as an evaluation of sort, this chapter looks to reimplement methodologies and specifications from various fields (ecology, computer science, physics etc.) within the framework we have created.

In particular I envisage rewriting different methodologies as *case studies*, evaluating the wiki as means to communicate, explore and compute.

- The PACT methodology [1].
- The LIFE methodology [3].
- A fairly self-contained RFC from the IETF, like RFC 5389 (STUN) [11].

3.5. Chapter 6: Conclusion

Chapter ✗ Research ∅ Paper ∅

It seems appropriate to leave this chapter until after all research is completed.

4. Timeline

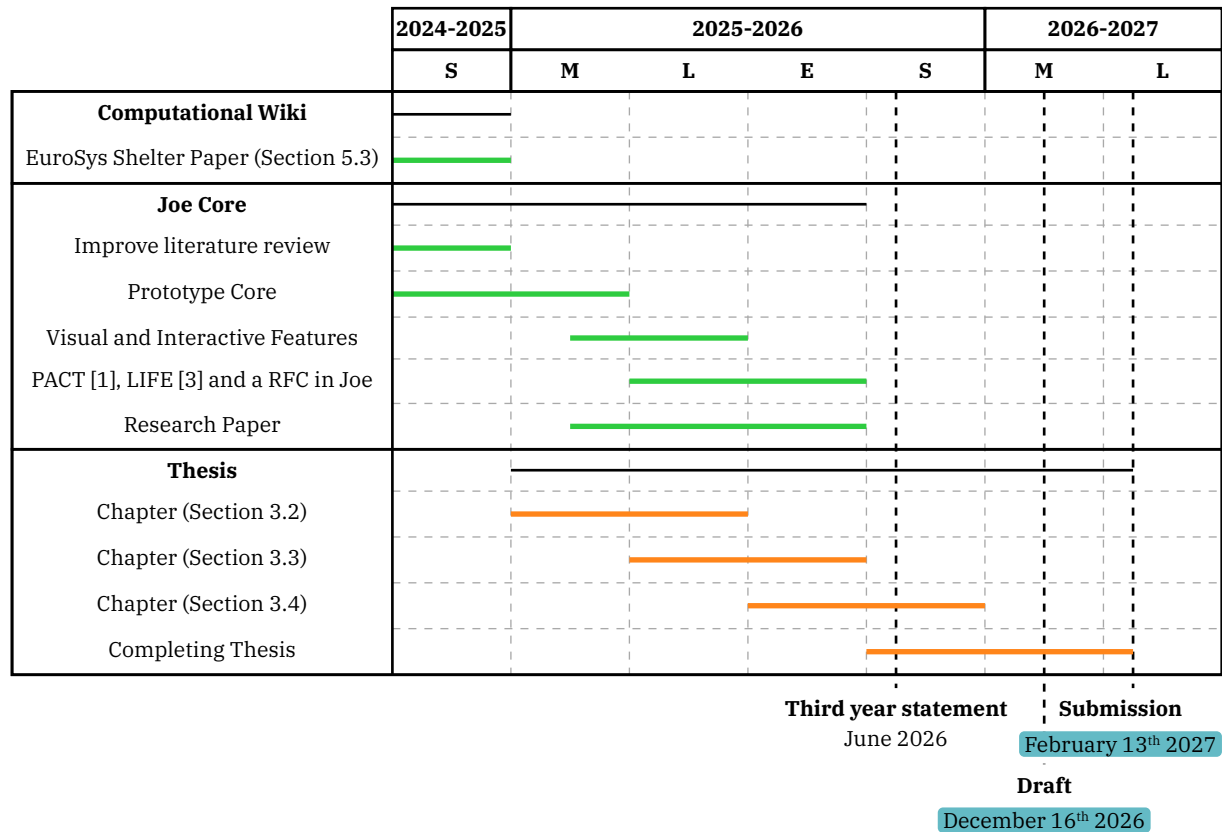


Figure 2: A proposed timeline for research over the next year and a half.

I believe there is a healthy amount of contingent time from Easter term next year until the end of my funding to help overcome any unforeseen difficulties with submitting on my proposed, self-imposed deadline.

5. Talks and Publications

5.1. Published

There were two main published pieces of work this year.

Functional Networking for Millions of Docker Desktops (Experience Report)⁶[9]

Docker is a developer tool used by millions of developers to build, share and run software stacks. The Docker macOS and Windows desktop clients have long used a novel combination of virtualisation and OCaml unikernels to seamlessly run Linux containers on these non-Linux hosts. We reflect on a decade of shipping this functional OCaml code into production across hundreds of millions of developer desktops, and discuss the lessons learnt from our experiences in integrating OCaml deeply into the container architecture that now drives much of the global cloud. We conclude by observing just how good a fit for systems programming that the unikernel approach has been, particularly when combined with the OCaml module and type system.

Anil Madhavapeddy, Dave Scott, Patrick Ferris, Ryan Gibb and Thomas Gazagnaire
30th ACM SIGPLAN International Conference on Functional Programming (ICFP)

Modularizing Reasoning about AI Capabilities via Abstract Dijkstra Monads [13]

A talk that outlined the vision for a recently originated research project aiming to build a formally verified prototype of a foundational “operating system” for safeguarded AI, called Bastion, that grounds these activities within programming language theory, namely by combining dependent type theory (as a practical general-purpose theory of computational structures and proofs), Dijkstra monads (as a flexible formalism for reasoning about an AI agent’s computational effects), and abstract types (for modularizing reasoning to individual components that can be separately developed by various stakeholders, including AI safety researchers, formal methods experts, organizations of various scales, and governments seeking to develop actionable, specific policy).

Code examples are available on GitHub. Slides and abstract.

Cyrus Omar, Patrick Ferris and Anil Madhavapeddy
12th ACM SIGPLAN Workshop on Higher-Order Programming with Effects (HOPE)

In addition to this I was a co-author on three successful submissions to the 1st International Workshop on Low Carbon Computing: Carbon-Aware Name Resolution, Lineage-first Computing and Emission Impossible [2, 4, 10]. I am also a co-author on *Steps towards an Ecology for the Internet* which has been accepted into Aarhus’ decennial conference [8].

⁶Note that at the time of writing this has been *definitively accepted* but not publicly announced.

5.1.1. Preprints

Solving Package Management via Hypergraph Dependency Resolution

We define HyperRes, a formal system for describing versioned dependency resolution using a hypergraph that is expressive enough to model many ecosystems and solve dependency constraints across them. We define translations from dozens of existing package managers to HyperRes and comprehensively demonstrate that dependency resolution can work across ecosystems that are currently distinct. This does not require users to shift their choice of package managers; instead, HyperRes allows for the translation of packaging metadata between ecosystems, and for solving to be precisely specialised to a particular deployment environment.

Ryan Gibb, Patrick Ferris, David Allsopp, Michael Dales, Mark Elvers, Thomas Gazagnaire, Sadiq Jaffer, Thomas Leonard, Jon Ludlam and Anil Madhavapeddy

[ArXiv](#)

5.2. In Review

Generating a corpus of Hazel programs from ill-typed OCaml programs (Extended Abstract)

When developing a new programming language, having a large corpus of both correct and incorrect programs allows language designers to test and explore the capabilities of their new language. However, bootstrapping such a corpus of incorrect programs is time consuming and arduous.

We therefore explore how to reuse code from more mature languages to generate a corpus of ill-typed code for newer ones. We have developed a compiler to Hazel, an emerging language with typed holes, from the more mature OCaml ecosystem. We find it practical to generate a comprehensive corpus of ill-typed programs for Hazel development, and discuss future larger scale efforts towards bridging ecosystems.

[hazel_of_ocaml](#) [source code](#).

Patrick Ferris and Anil Madhavapeddy

[10th ACM SIGPLAN International Workshop on Type-Driven Development](#)

5.3. Working Papers

Shelter: A metashell for reproducible shell sessions

We present Shelter, a new shell for reversible computing. Shelter sessions can be constructed in an interactive, low-latency, exploratory mode and subsequently replayed and combined to ease the transition from exploratory mode to production builds. Shelter enhances modern scientific computing workflows by tracking the lineage of datasets, versioning external network resources, and encouraging hypothetical exploration. Shelter combines modern Linux filesystems, eBPF, and namespaces into a new interface tracked by git-like version control. This allows Shelter to construct interactive POSIX environments and manage network and filesystem side-effects resulting from command execution. Shelter exposes a familiar shell environment with the twist that users can undo almost all operations, rebase previous sessions with fresh data and publish them for use by other OCI clients such as Docker.

Patrick Ferris, Ryan Gibb, Michael Dales and Anil Madhavapeddy

[21st Edition of EuroSys](#)

Bibliography

- [1] Andrew Balmford, David Coomes, Michael Dales, Patrick Ferris, James Hartup, Sadiq Jaffer, Srinivasan Keshav, Miranda Lam, Anil Madhavapeddy, Robin Message, and et al. 2023. PACT Tropical Moist Forest Accreditation Methodology v2.0. *Cambridge Open Engage* (2023). <https://doi.org/10.33774/coe-2023-g584d-v5>
- [2] Michael Dales, Patrick Ferris, and Anil Madhavapeddy. 2024. Lineage first computing: towards a frugal userspace for Linux. 2024. Retrieved from https://www.sicsa.ac.uk/wp-content/uploads/2024/11/LOCO2024_paper_30.pdf
- [3] Alison Eyres, Thomas Ball, Michael Dales, Tom Swinfield, Andy Arnell, Daniele Baisero, América Paz Durán, Jonathan Green, Rhys E. Green, Anil Madhavapeddy, and Andrew Balmford. 2024. LIFE: A Metric for Quantitatively Mapping the Impact of Land-Cover Change on Global Extinctions. <https://doi.org/10.33774/coe-2023-gpn4p-v4>
- [4] Ryan Gibb, Patrick Ferris, and Anil Madhavapeddy. 2024. Carbon-Aware Name Resolution. 2024. Retrieved from https://www.sicsa.ac.uk/wp-content/uploads/2024/11/LOCO2024_paper_28.pdf
- [5] Jake Hillion. 2022. Void Processes: Minimising privilege by default. Retrieved from <https://blog.hillion.co.uk/posts/void-processes/dissertation/jsh77-dissertation.pdf>
- [6] Gowtham Kaki, Swarn Priya, KC Sivaramakrishnan, and Suresh Jagannathan. 2019. Mergeable replicated data types. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–29.
- [7] Michael W Lucas and Allan Jude. 2015. *FreeBSD Mastery: ZFS*. Tilted Windmill Press.
- [8] Anil Madhavapeddy, Sam Reynolds, Alec P Christie, David A Coomes, Michael W Dales, Patrick Ferris, Ryan Gibb, Hamed Haddadi, Sadiq Jaffer, Josh Millar, and others. 2025. Steps towards an Ecology for the Internet. *arXiv preprint arXiv:2506.06469* (2025).
- [9] Anil Madhavapeddy, Dave Scott, Patrick Ferris, Ryan Gibb, and Thomas Gazagnaire. 2025. Functional Networking for Millions of Docker Desktops (Experience Report). In *The 30th ACM SIGPLAN International Conference on Functional Programming*, October 2025.
- [10] Jessica Man, Sadiq Jaffer, Patrick Ferris, Martin Kleppmann, and Anil Madhavapeddy. 2024. Emission Impossible: privacy-preserving carbon emissions claims. 2024. Retrieved from https://www.sicsa.ac.uk/wp-content/uploads/2024/11/LOCO2024_paper_35.pdf
- [11] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. 2008. Session Traversal Utilities for NAT (STUN). <https://doi.org/10.17487/RFC5389>
- [12] Robin Milner. 2001. Bigraphical reactive systems. In *CONCUR 2001—Concurrency Theory: 12th International Conference Aalborg, Denmark, August 20–25, 2001 Proceedings* 12, 2001. 16–35.
- [13] Cyrus Omar, Patrick Ferris, and Anil Madhavapeddy. 2024. Modularizing Reasoning about AI Capabilities via Abstract Dijkstra Monads. In *The 12th ACM SIGPLAN Workshop on Higher-Order Programming with Effects (HOPE)*, September 2024. Retrieved from <https://icfp24.sigplan.org/home/hope-2024>
- [14] Ningning Xie, Daniel D Johnson, Dougal Maclaurin, and Adam Paszke. 2024. Parallel Algebraic Effect Handlers. *Proceedings of the ACM on Programming Languages* 8, ICFP (2024), 756–788.

Index of Figures

Figure 1 Core meta-commands in Shelter	5
Figure 2 A proposed timeline for research over the next year and a half.	11

Index of Listings

Listing 1 An example <code>shelter</code> session	4
Listing 2 An example of using <code>void</code> an OCaml library for building void processes	5
Listing 3 An OCaml program that writes data and spawns a process that also writes data	6
Listing 4 The output from running <code>opentrace</code> on the program from Listing 3	6
Listing 5 A markdown document that is compiled to Forester	7