CausalRPC

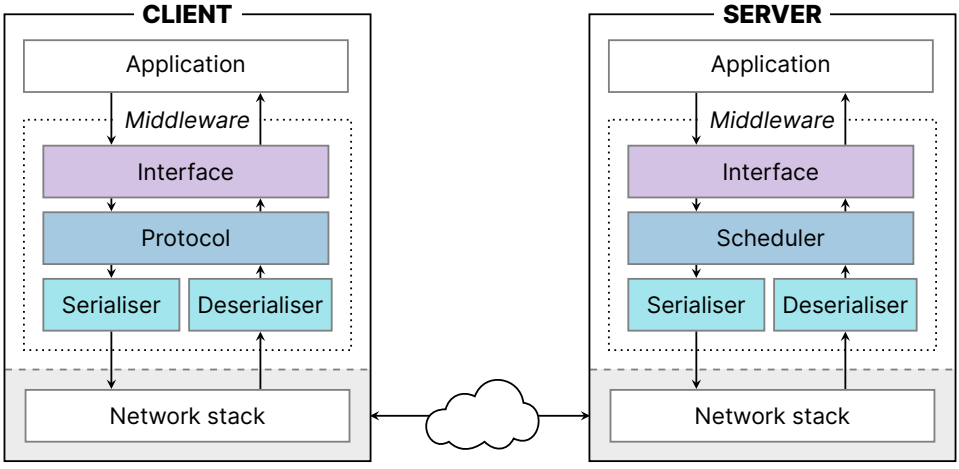
Distributed computation over Irmin

Craig Ferguson (@CraigFe)

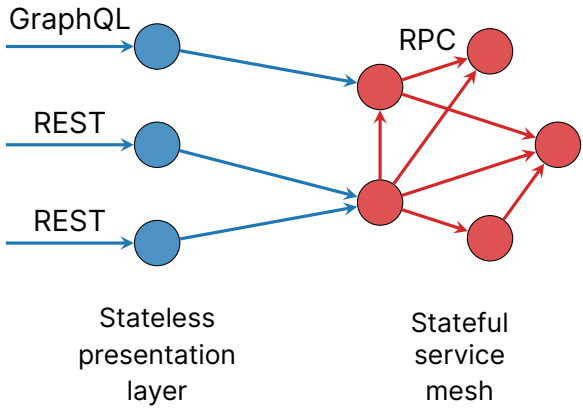
Friday 23rd August

Remote Procedure Call (RPC)

Remote computation at the level of individual function calls.



Trendy microservice architectures



Use RPC for remote access to a resource (storage, private network etc.).

Two birds and one cannonball

Two birds and one cannonball

Problem: consistency under race conditions

- breaks RPC abstraction
- results in additional round-trips to server

Two birds and one cannonball

Problem: consistency under race conditions

- breaks RPC abstraction
- results in additional round-trips to server

Problem: tracing distributed systems

- neither the middleware nor the application layer has full context
- linear log outputs on remote systems must be correlated

Two birds and one cannonball

Problem: consistency under race conditions

- breaks RPC abstraction
- results in additional round-trips to server

Problem: tracing distributed systems

- neither the middleware nor the application layer has full context
- linear log outputs on remote systems must be correlated

Idea: middleware that understands state

- application explains the underlying state to the middleware
- middleware resolves conflicts and constructs a trace as it goes

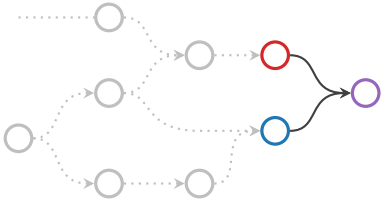
Introducing Irmin

- Distributed, immutable Merkle trees with push/pull sync – just like Git!
- Pure OCaml; unikernel-compatible.
- Parameterised on:
  - Storage backend (Memory, FS, Git repo, Redis, ...)
  - User-defined mergable contents (leaves of the tree)

Aside: why *three*-way merge?

Aside: why *three*-way merge?

Two-way CRDT[1] merge

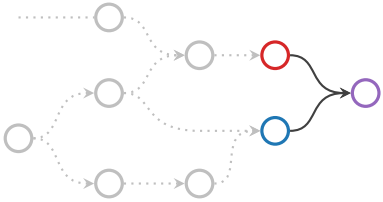


state → state → state

[1] M. Shapiro et al. *Conflict-Free Replicated Data Types*, 2011.

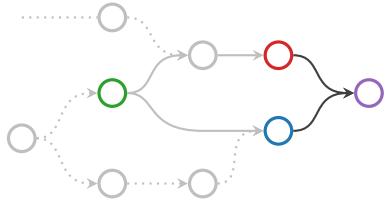
Aside: why *three*-way merge?

Two-way CRDT[1] merge



state → state → state

Three-way Irmin merge

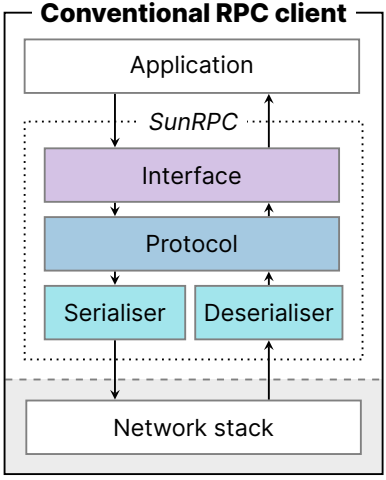


lca : state → state → state → state

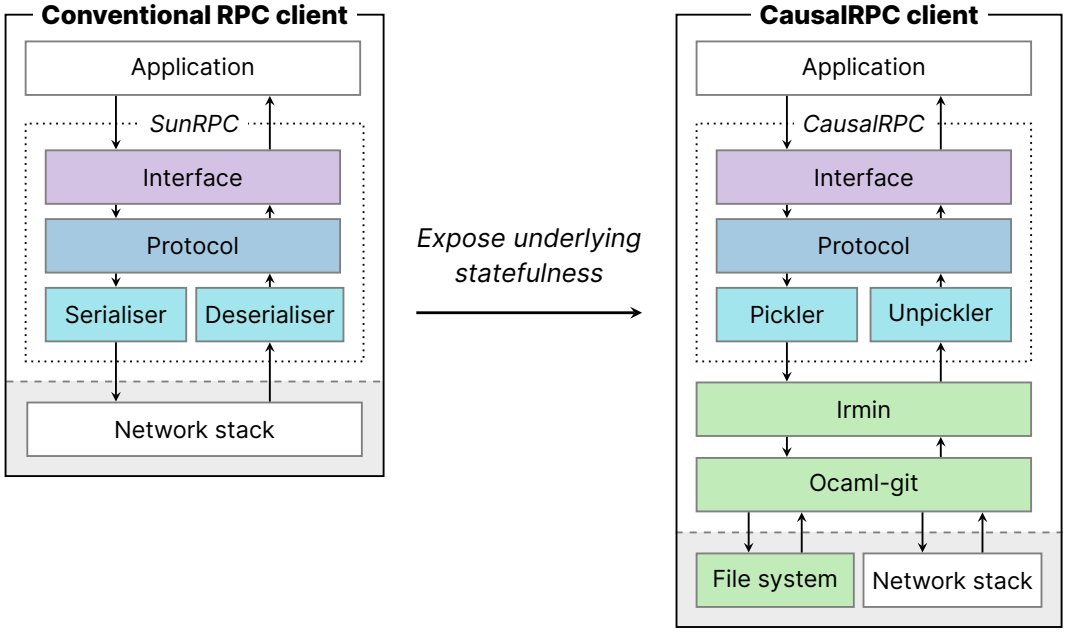
Lowest common ancestor allows the merge operation to reason about *intent*.

[1] M. Shapiro et al. *Conflict-Free Replicated Data Types*, 2011.

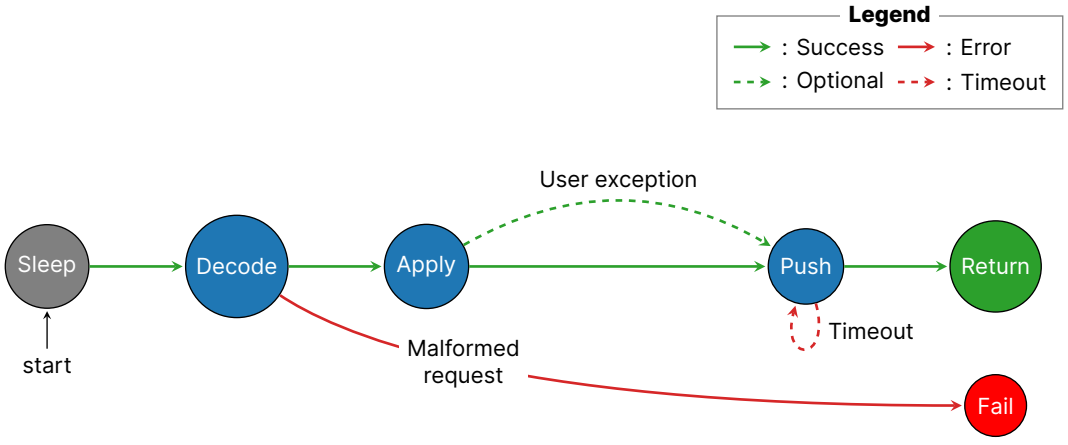
State-aware RPC



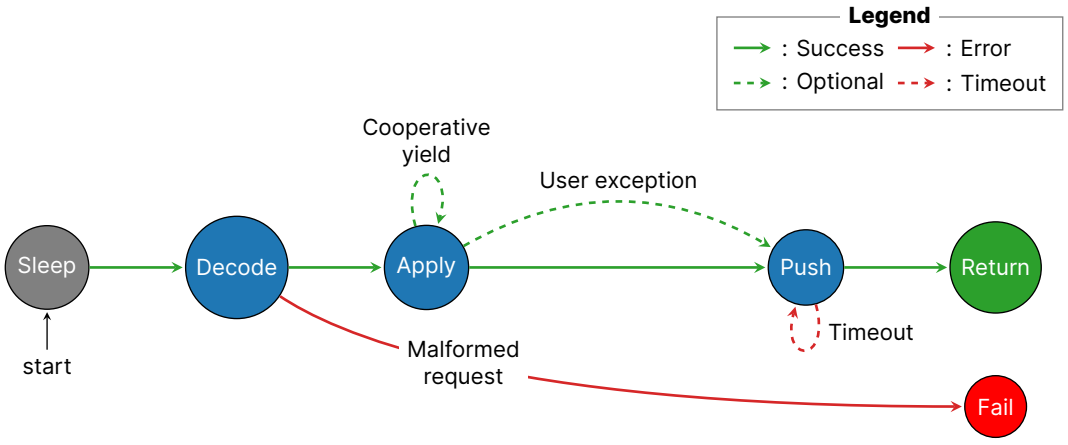
State-aware RPC



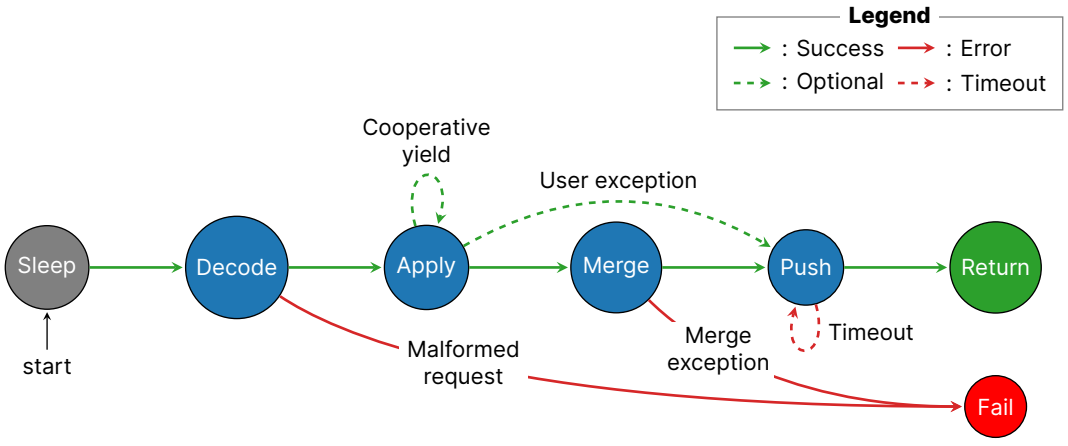
The life of an RPC thread



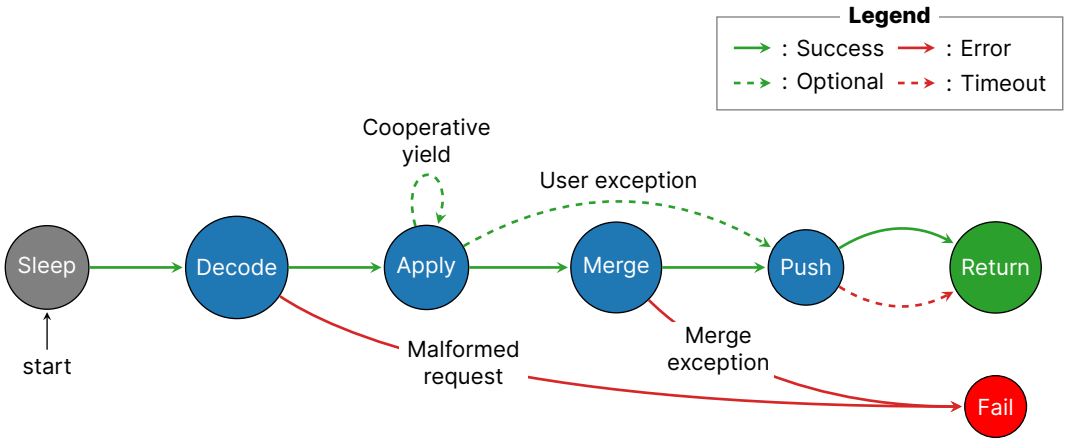
The life of an RPC thread



The life of an RPC thread

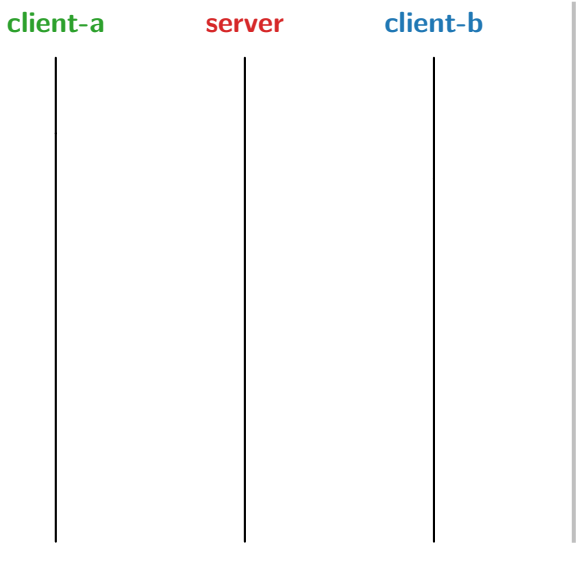


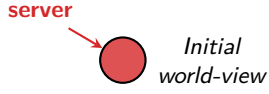
The life of an RPC thread



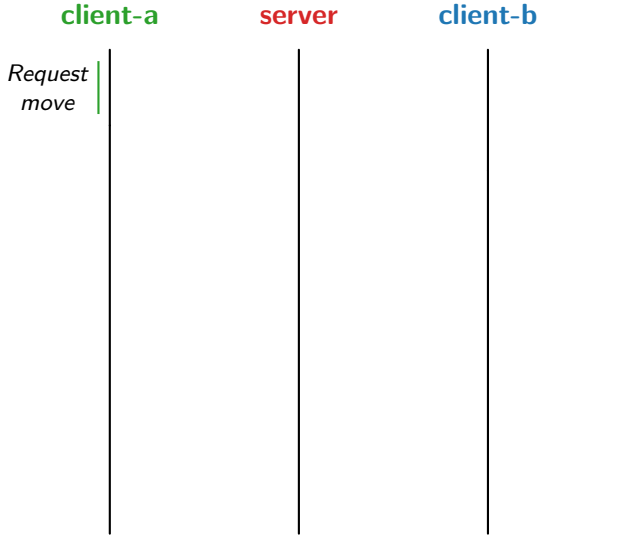
Mergeable filesystems

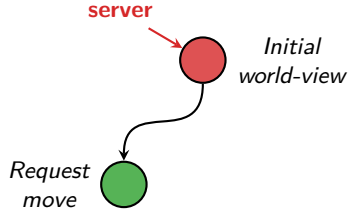
Mergeable filesystems



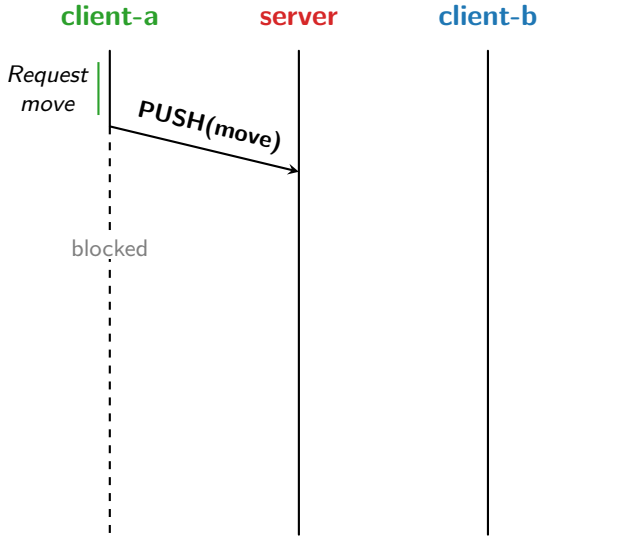


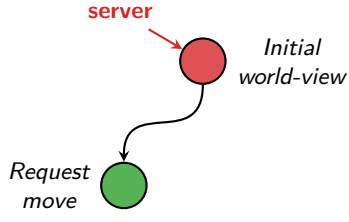
Mergeable filesystems



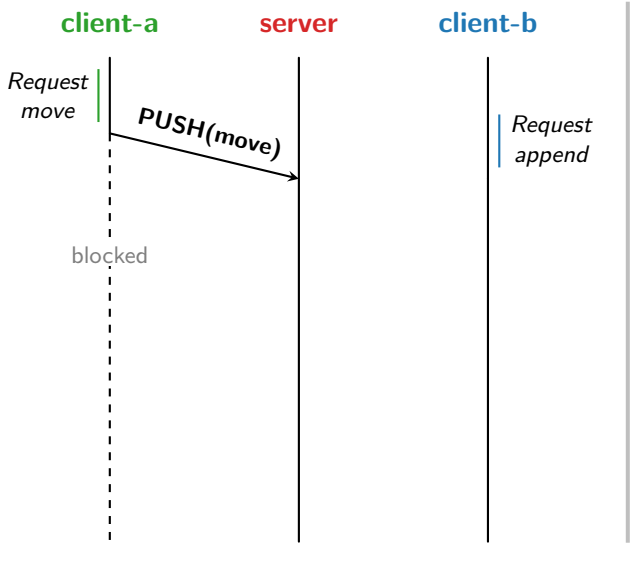


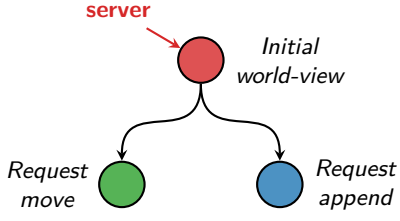
Mergeable filesystems



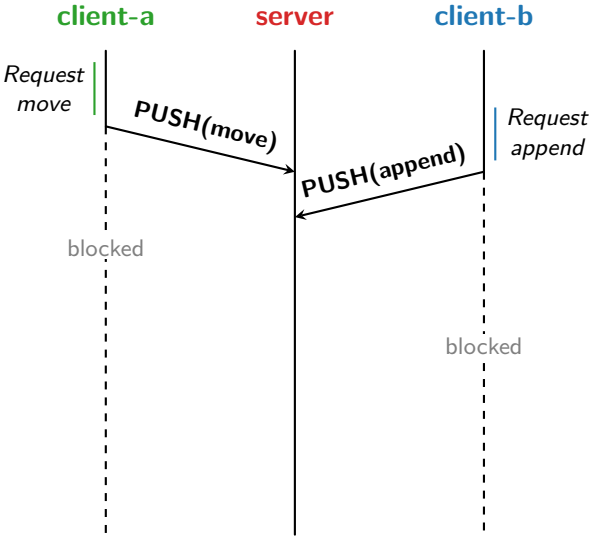


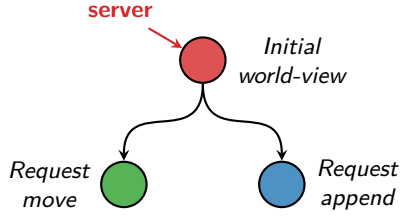
Mergeable filesystems



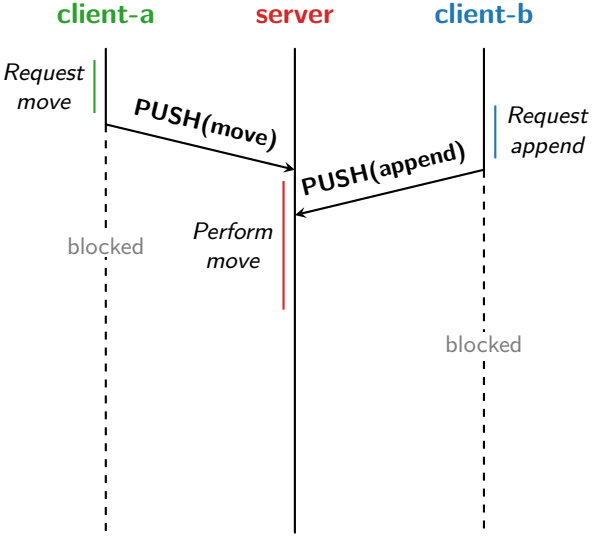


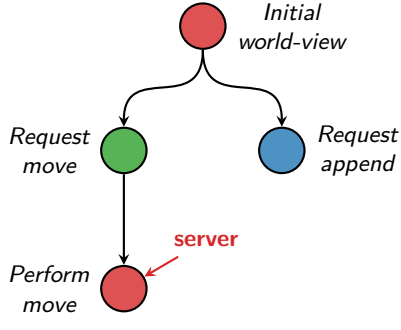
Mergeable filesystems



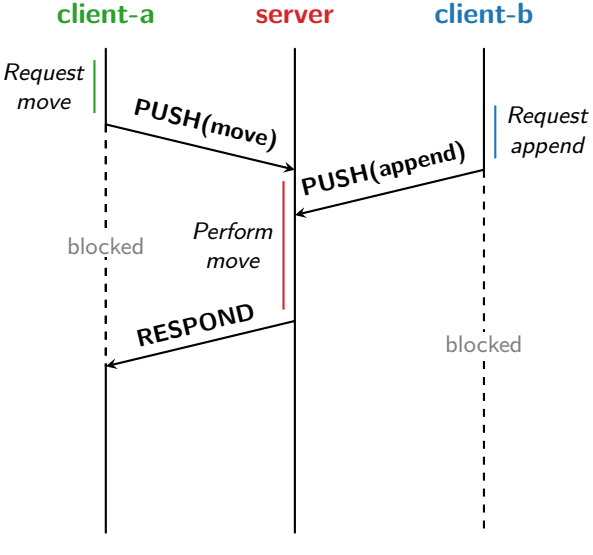


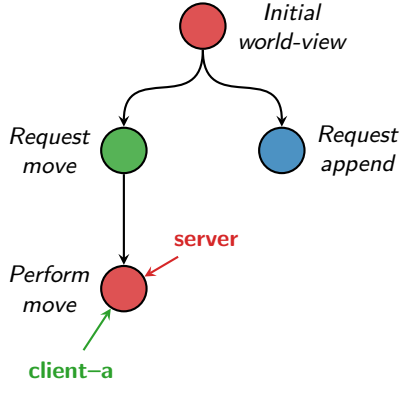
Mergeable filesystems



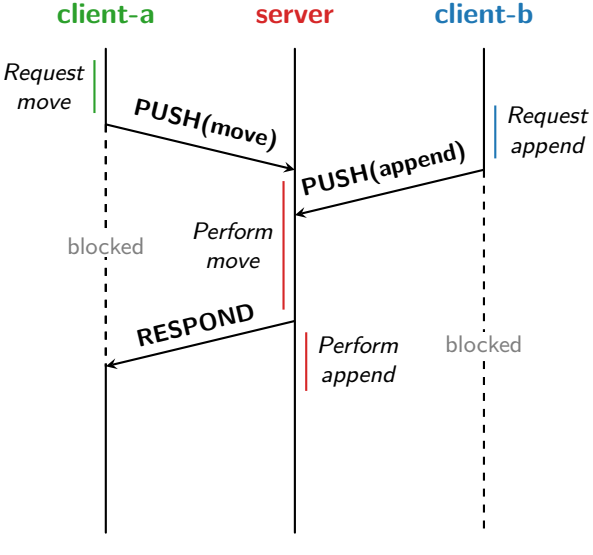


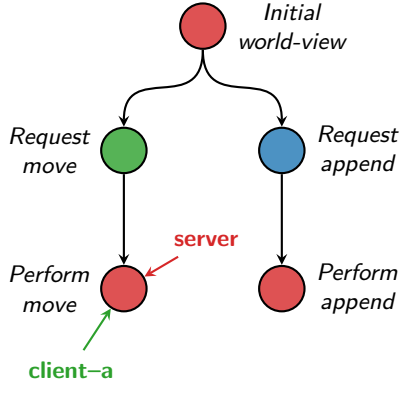
Mergeable filesystems



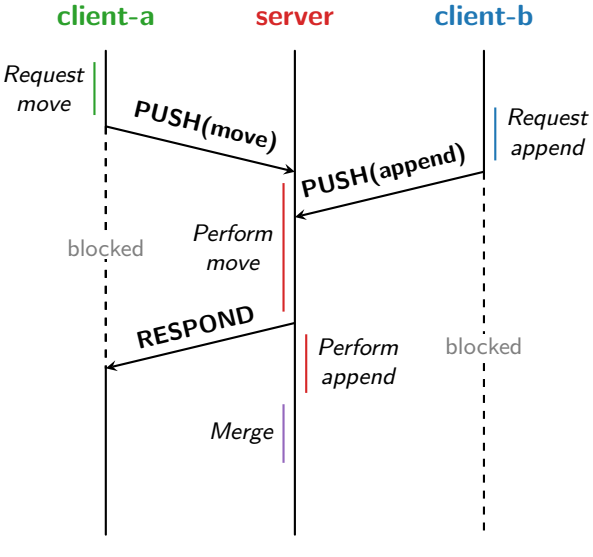


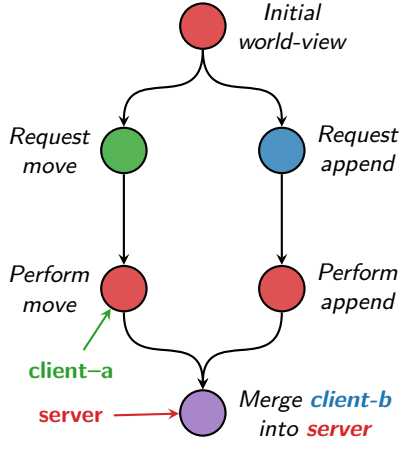
Mergeable filesystems



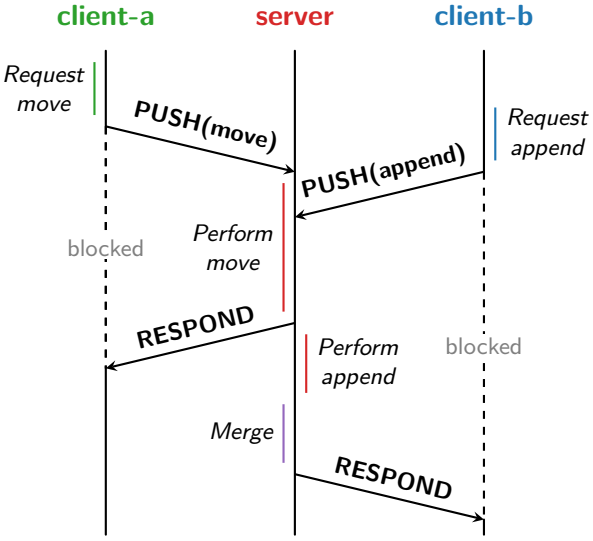


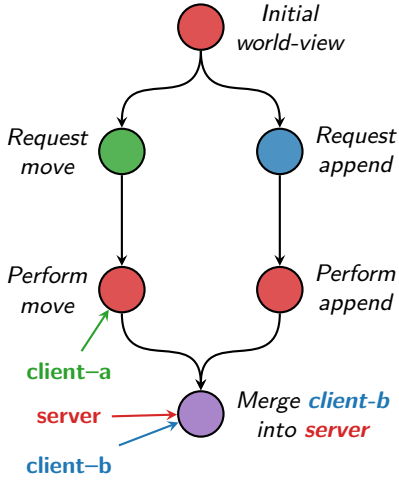
Mergeable filesystems





Mergeable filesystems





Mergeable filesystems: trace output

```
$ git log
```

Mergeable filesystems: trace output

```
$ git log

* server Initialised Filesystem store
|\
* | clientA Request <move>: /tmp/sharedFile -> /log/sharedFile
* | server (client--a) Perform <move> for clientA
| * clientB Request <append>: /tmp/sharedFile
| * server  Perform <append> operation for clientB
|/
*   server (server, client--b) Merge client--b into server
```

Why OCaml?

Portability – Inherits the Mirage mindset towards platform-independence.

Why OCaml?

Portability – Inherits the Mirage mindset towards platform-independence.

Type-safety – Strict interface definition using an embedded DSL.

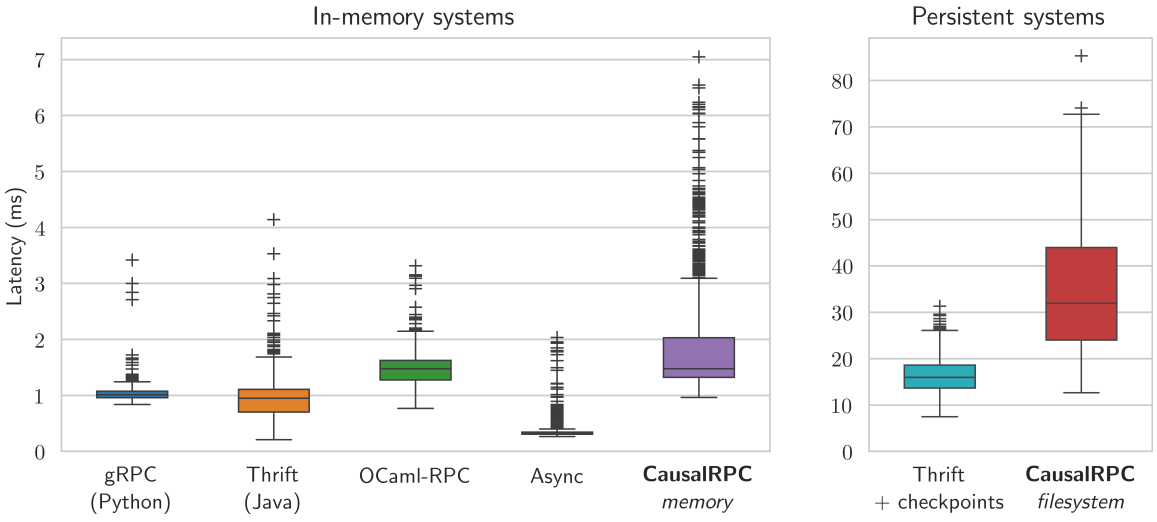
Why OCaml?

Portability – Inherits the Mirage mindset towards platform-independence.

Type-safety – Strict interface definition using an embedded DSL.

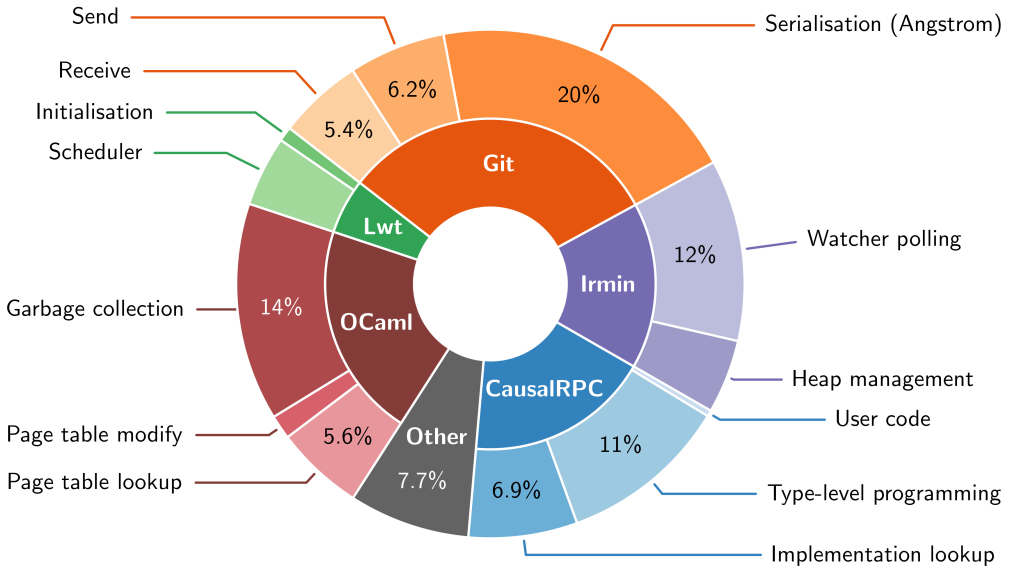
Coolness-factor (!!!) – Protocol design in OCaml is really fun.

Performance: latency



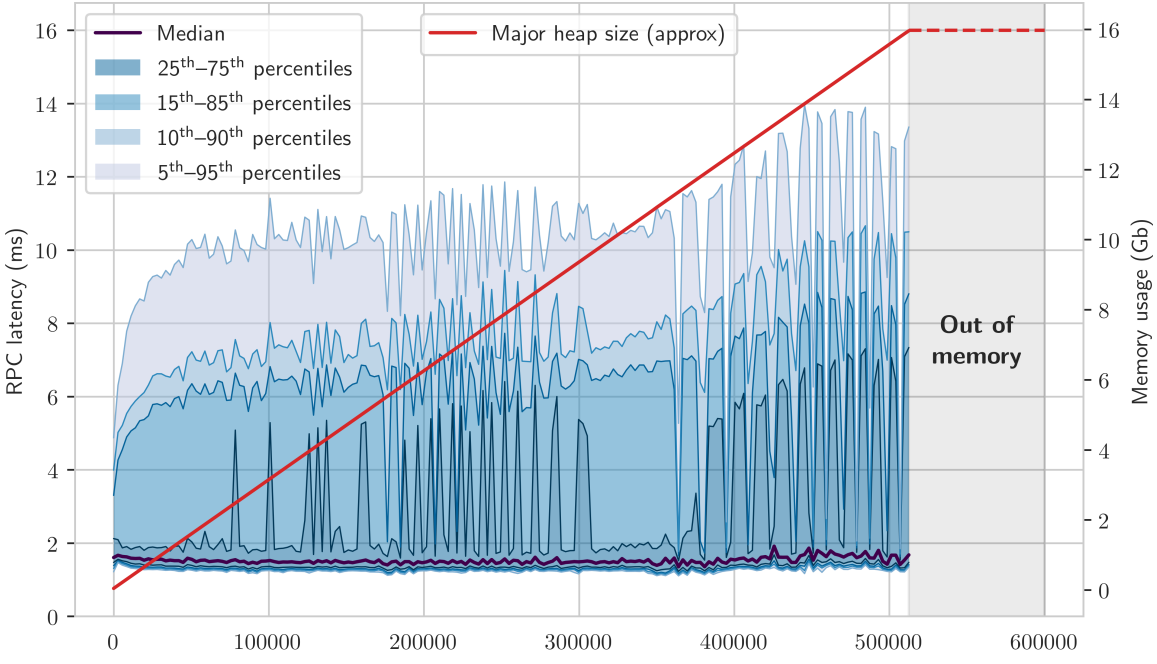
Measured over 1000 RPCs for a single client-server pair over a Docker bridge network.

Server run-time cost profile from first packet receipt to final packet send:



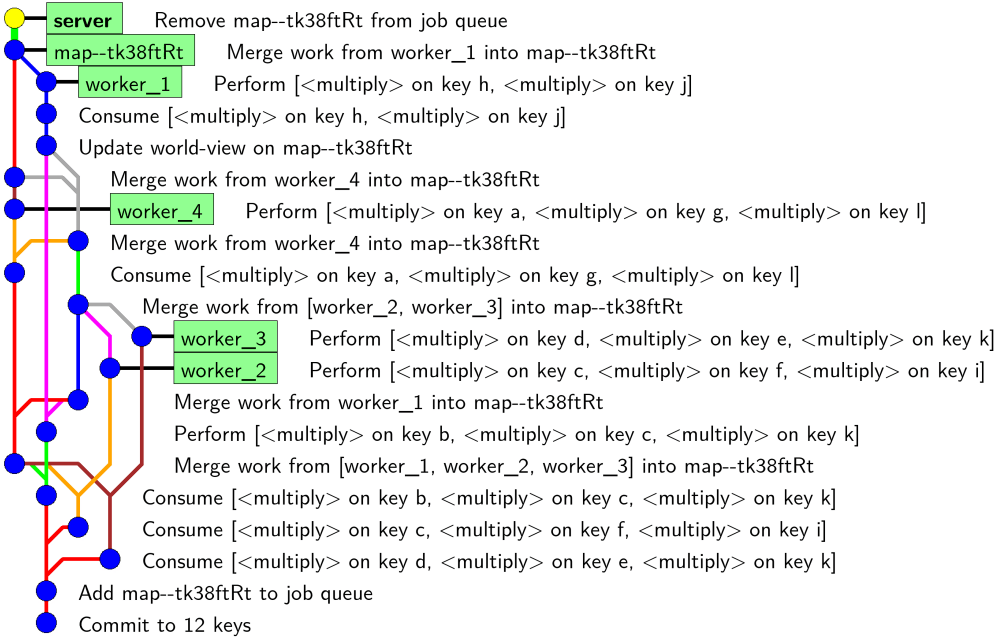
Source: *gprof and a huge amount of patience.*

Performance: memory-usage



Generalisation to work clusters

Generalisation to work clusters

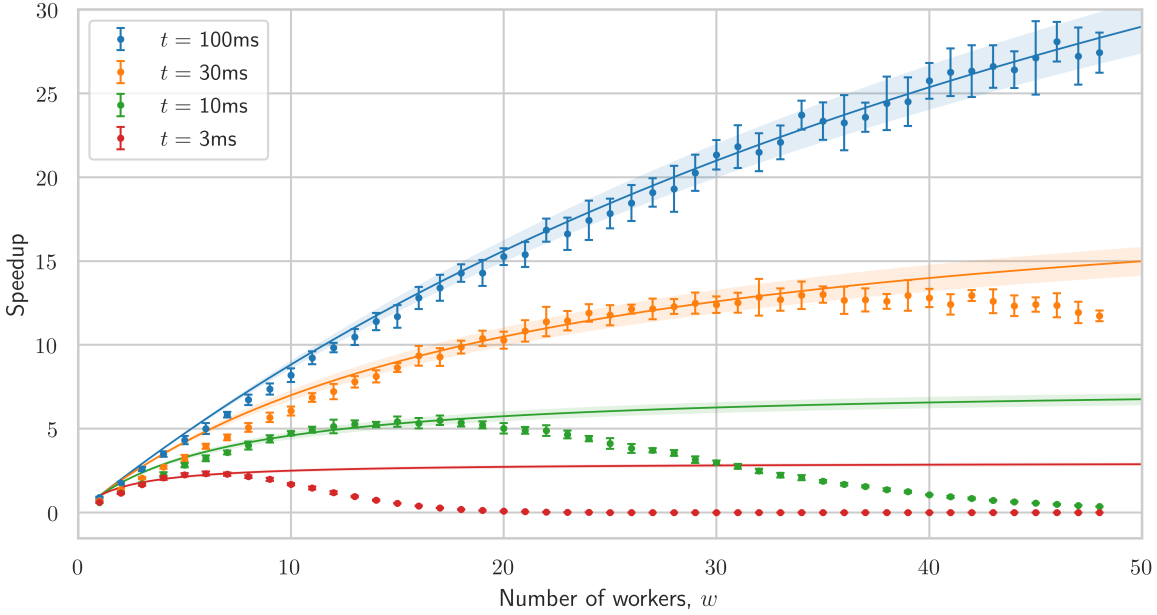


Work cluster scalability

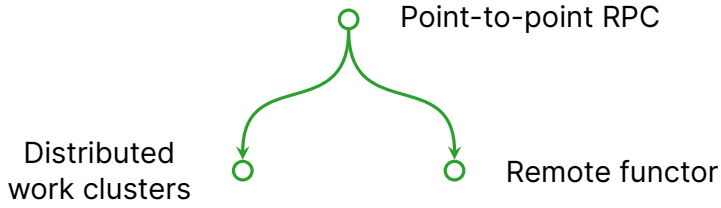
Compare cluster performance to Amdahl's law prediction:

Work cluster scalability

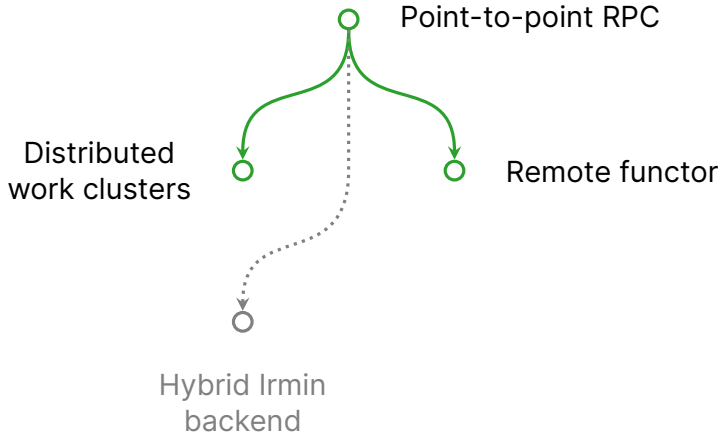
Compare cluster performance to Amdahl's law prediction:



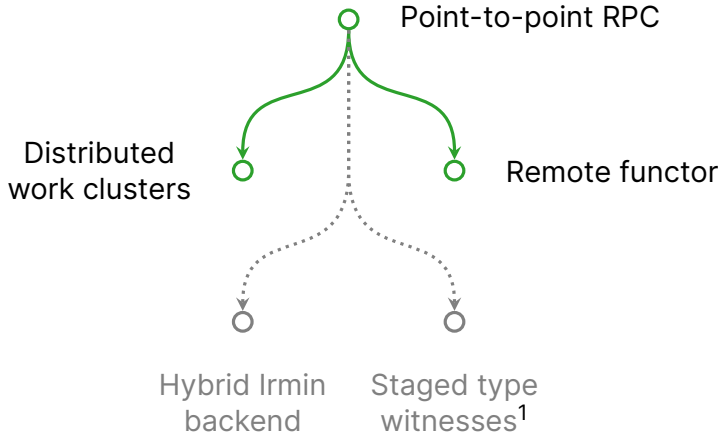
What next?


Point-to-point RPC
Distributed
work clusters
Remote functor

What next?

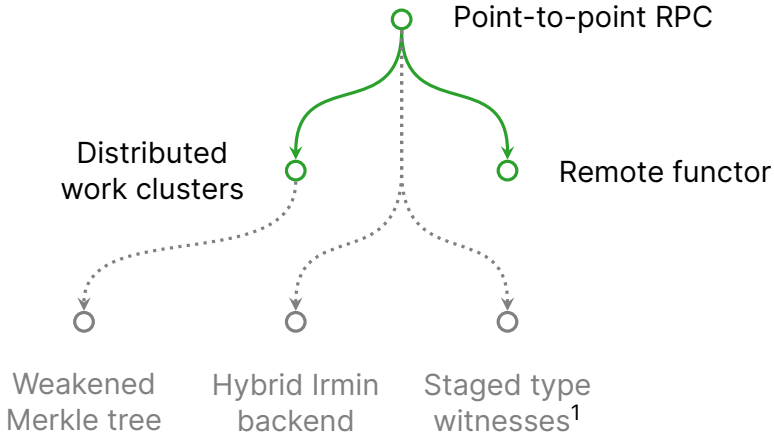

Point-to-point RPC
Distributed
work clusters
Remote functor
Hybrid Irmin
backend

What next?



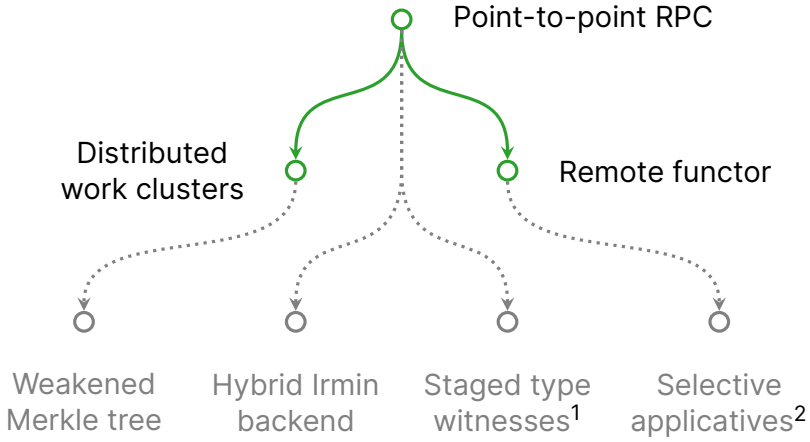
[1] N. Krishnaswami and J. Yallop. *A Typed, Algebraic Approach to Parsing*, 2019.

What next?



[1] N. Krishnaswami and J. Yallop. *A Typed, Algebraic Approach to Parsing*, 2019.

What next?



[1] N. Krishnaswami and J. Yallop. *A Typed, Algebraic Approach to Parsing*, 2019.

[2] A. Mokhov et al. *Selective Applicative Functors*, 2019.

Take-away lessons

1. Simple distributed system abstractions set challenging design constraints.

Take-away lessons

1. Simple distributed system abstractions set challenging design constraints.
2. Improve semantics by squashing the stack; use OCaml to maintain abstraction and flexibility.

Take-away lessons

1. Simple distributed system abstractions set challenging design constraints.
2. Improve semantics by squashing the stack; use OCaml to maintain abstraction and flexibility.
3. Consider non-linear log structures in your programs; *casual* logs make it easy to reason about intent.

Thanks For Listening

Related work

(Much) more detail in the full dissertation:

`https://craigfe.io/out/causalrpc.pdf`

mergeable types

- Conflict-free replicated datatypes. [M. Shapiro et al. 2011]
- Mergeable types. [B Farinier et al. 2015]

embedded DSLs for RPC

- Pickler combinators. [A. Kennedy. 2004]

patterns for remote execution

- Remote functors. [A. Gill et al. 2015]
- Selective applicative functors. [A. Mokhov et al. 2019]

Work cluster performance