Immutable Distributed Infrastructure for Unikernels

WG2.8, Greece

Anil Madhavapeddy (speaker)
with Benjamin Farinier and Thomas Gazagnaire
University of Cambridge Computer Laboratory

May 26, 2015

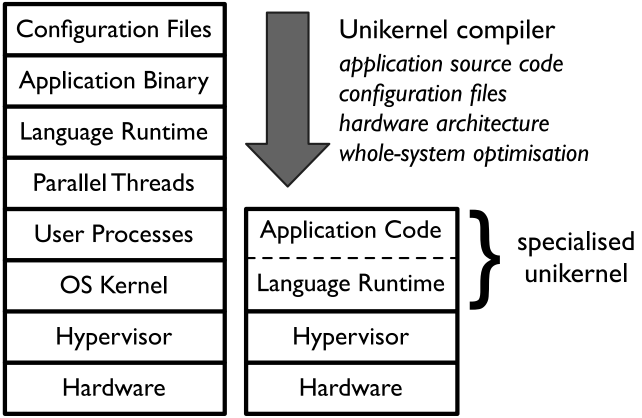
- Background
  - Unikernels
  - Irmin, a large-scale, immutable, branch-consistent storage
- Weakly consistent data structures
  - Mergeable queues
  - Mergeable ropes
- Benchmarking Irmin
- Use Cases

- Modern systems are built in **memory-safe programming languages**.
- We build elaborate libraries and applications to express complex logic.
- ...and watch it all come crashing down when it interfaces with the OS.

Mirage OS is a library operating system that constructs unikernels for secure, high-performance network applications across a variety of cloud computing and mobile platforms.

Mirage OS is a **library operating system** that constructs unikernels for secure, high-performance network applications across a variety of cloud computing and mobile platforms.

Mirage OS is a library operating system that constructs **unikernels** for secure, high-performance network applications across a variety of cloud computing and mobile platforms.

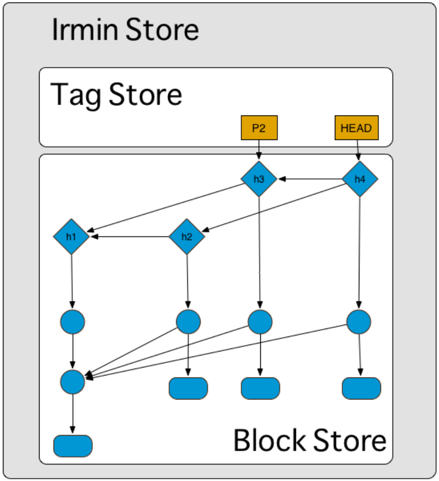

Configuration Files
Application Binary
Language Runtime
Parallel Threads
User Processes
OS Kernel
Hypervisor
Hardware
Unikernel compiler
application source code
configuration files
hardware architecture
whole-system optimisation
Application Code
Language Runtime
Hypervisor
Hardware
specialised unikernel

What should a storage interface to unikernels look like?

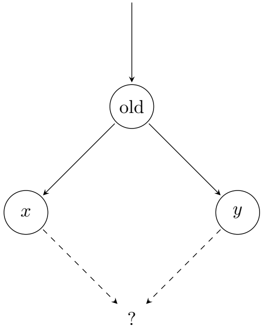
- Highly distributed systems
- Frequent failure is an option
- Debugging and tracing must be built into the fabric

Irmin, large-scale, immutable, branch-consistent storage

- Irmin is a library to **persist** and **synchronize distributed data structures** both on-disk and in-memory
- It enables a style of programming very similar to the **Git workflow**, where distributed **nodes fork, fetch, merge and push** data between each other
- The general idea is that you want every active node to get a **local** (partial) **copy of a global database** and always be very explicit about how and when data is shared and migrated


Irmin Store
Tag Store
P2
HEAD
h3
h4
h1
h2
Block Store

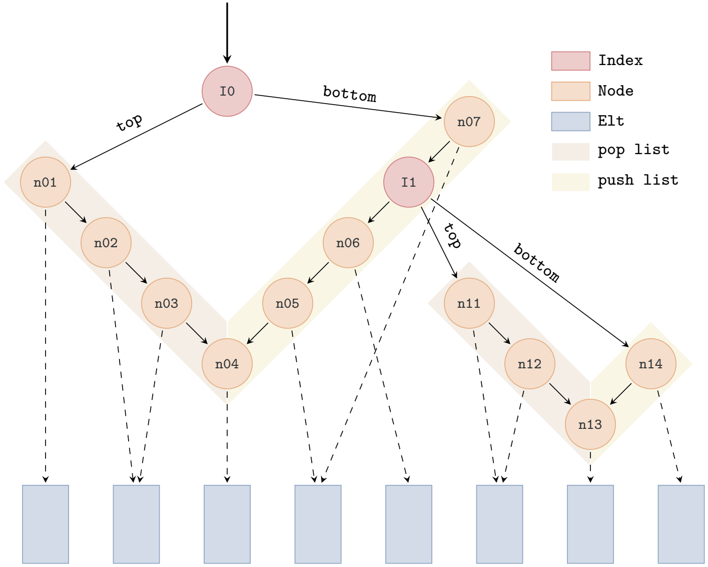
```
type t = ...
(** User-defined contents. *)
type result = [ ‘Ok of t |
   ‘Conflict of string ]

val merge: old:t → t → t →
   result
(** 3-way merge functions. *)
```

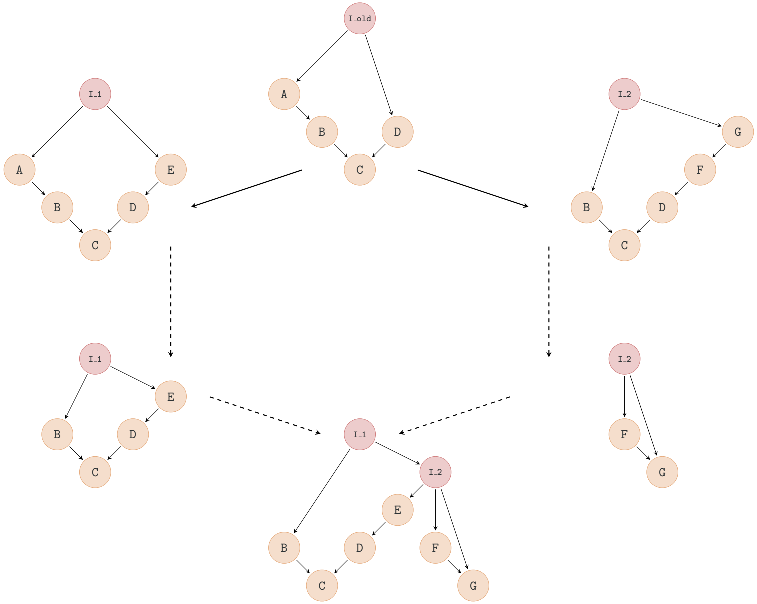


Weakly consistent data structures

Mergeable queues

```
module type IrminQueue.S = sig
  type t
  type elt

  val create : unit → t
  val length : t → int
  val is_empty : t → bool

  val push : t → elt → t
  val pop : t → (elt * t)
  val peek : t → (elt * t)

  val merge : IrminMerge.t
end
```


I0
top
bottom
n01
n02
n03
n04
n05
n06
n07
I1
top
bottom
n11
n12
n13
n14
Index
Node
Elt
pop list
push list


I_old
A
B
D
C
I_1
A
E
B
D
C
I_2
G
F
B
D
C
I_1
E
B
D
C
I_2
F
G
I_1
I_2
E
B
D
F
C
G

Current state

Operation	Read	Write	
Push	0	2	$O(1)$
Pop	2 on average	1 on average	$O(1)$
Merge	n	1	$O(n)$

Current state

Operation	Read	Write	
Push	0	2	$O(1)$
Pop	2 on average	1 on average	$O(1)$
Merge	n	1	$O(n)$

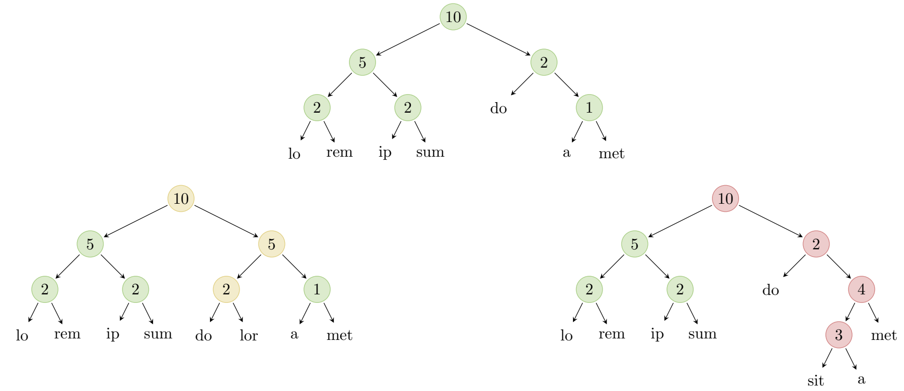
With a little more work

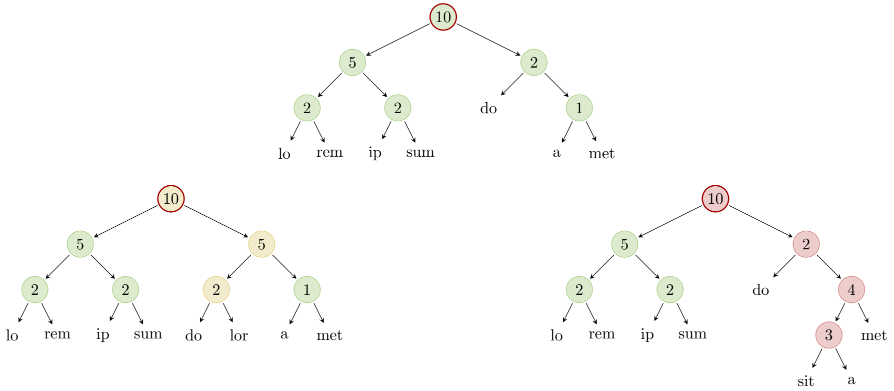
Operation	Read	Write	
Push	0	2	$O(1)$
Pop	2 on average	1 on average	$O(1)$
Merge	$\log n$	1	$O(\log n)$

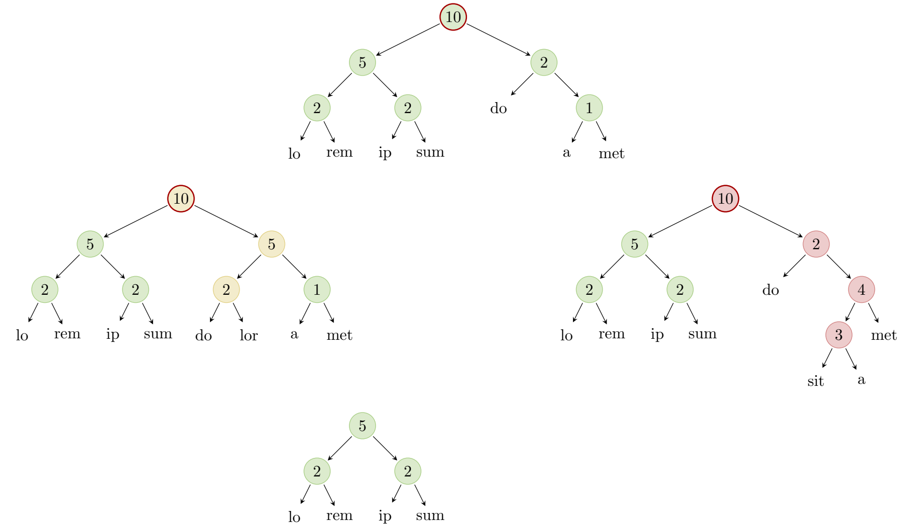
Mergeable ropes

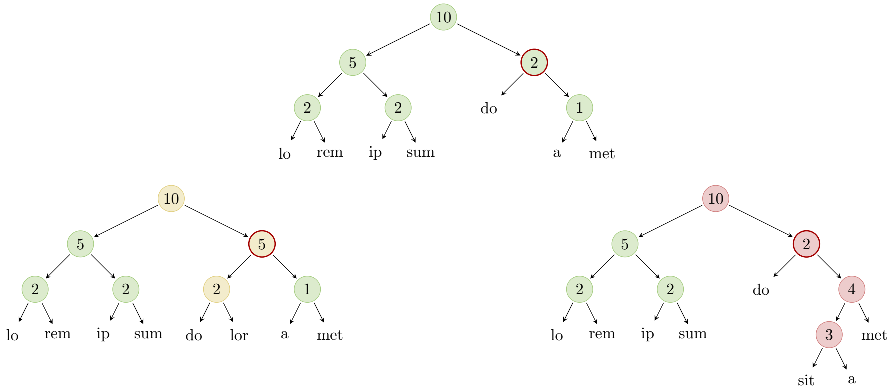
```
module type IrminRope.S = sig
  type t
  type value (* e.g char *)
  type cont  (* e.g string *)

  val create : unit → t
  val make : cont → t
  ...
  val set : t → int → value → t
  val get : t → int → value
  val insert : t → int → cont → t
  val delete : t → int → int → t
  val append : t → t → t
  val split : t → int → (t * t)

  val merge : IrminMerge.t
end
```

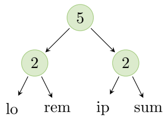
Operation	Rope	String
Set/Get	$O(\log n)$	$O(1)$
Split	$O(\log n)$	$O(1)$
Concatenate	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$
Merge	$\log(f(n))$	$f(n)$

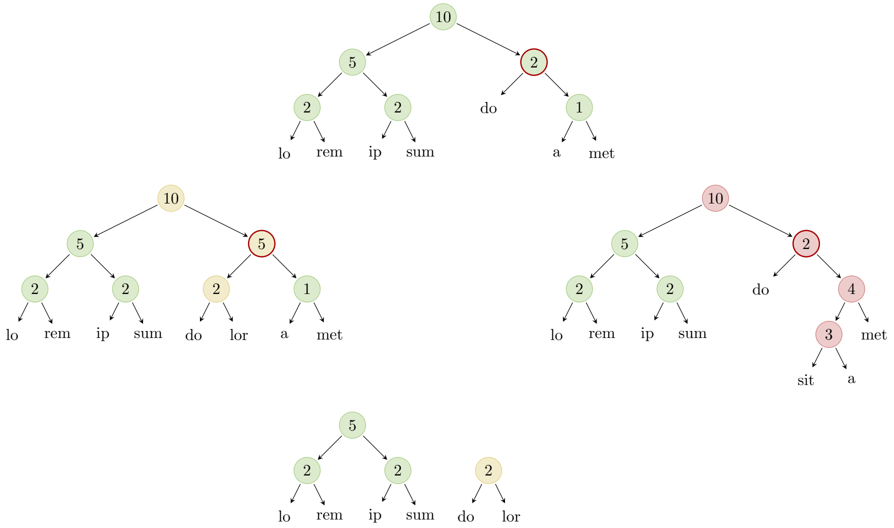

10
5
2
2
2
do
1
lo
rem
ip
sum
a
met
10
5
5
2
2
2
1
lo
rem
ip
sum
do
lor
a
met
10
5
2
2
2
do
4
lo
rem
ip
sum
3
met
sit
a

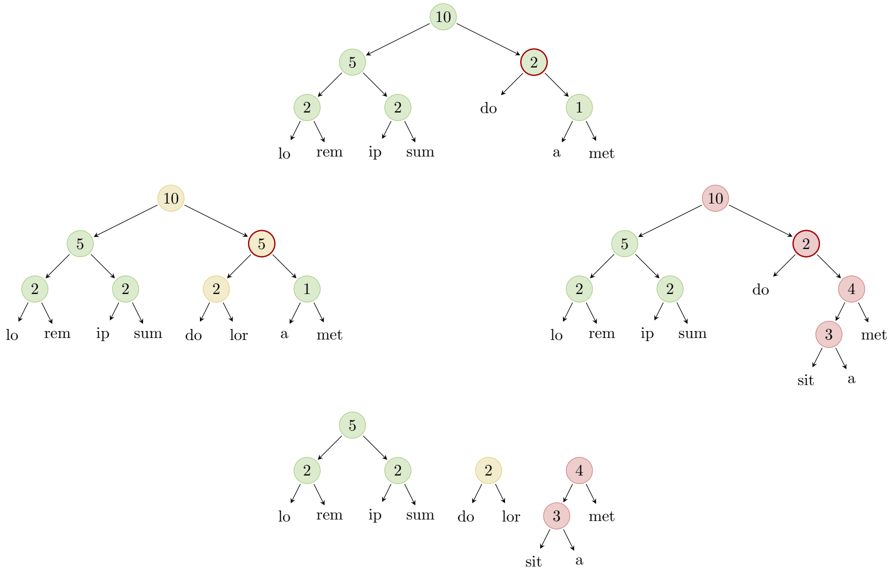

10
5
2
2
2
do
1
lo
rem
ip
sum
a
met
10
5
5
2
2
2
1
lo
rem
ip
sum
do
lor
a
met
10
5
2
2
2
do
4
lo
rem
ip
sum
3
met
sit
a

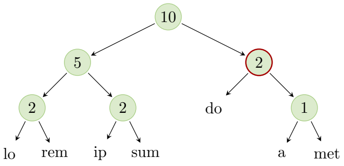

10
5
2
2
2
1
do
lo
rem
ip
sum
a
met
10
5
5
2
2
2
1
lo
rem
ip
sum
do
lor
a
met
10
5
2
2
2
4
3
do
lo
rem
ip
sum
met
sit
a
5
2
2
lo
rem
ip
sum

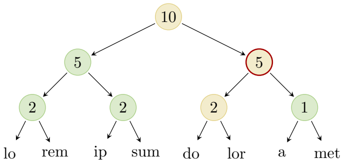

10
5
2
2
lo
rem
ip
sum
2
do
1
a
met
10
5
2
2
lo
rem
ip
sum
5
2
do
lor
1
a
met
10
5
2
2
lo
rem
ip
sum
2
do
4
3
sit
a
met

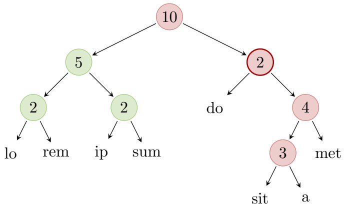

5
2
2
lo
rem
ip
sum

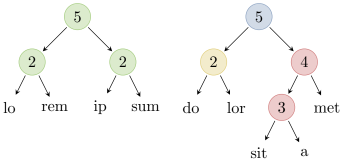

10
5
2
2
lo
rem
ip
sum
2
do
1
a
met
10
5
2
2
lo
rem
ip
sum
5
2
do
lor
1
a
met
10
5
2
2
lo
rem
ip
sum
2
do
4
3
sit
a
met
5
2
2
lo
rem
ip
sum
2
do
lor

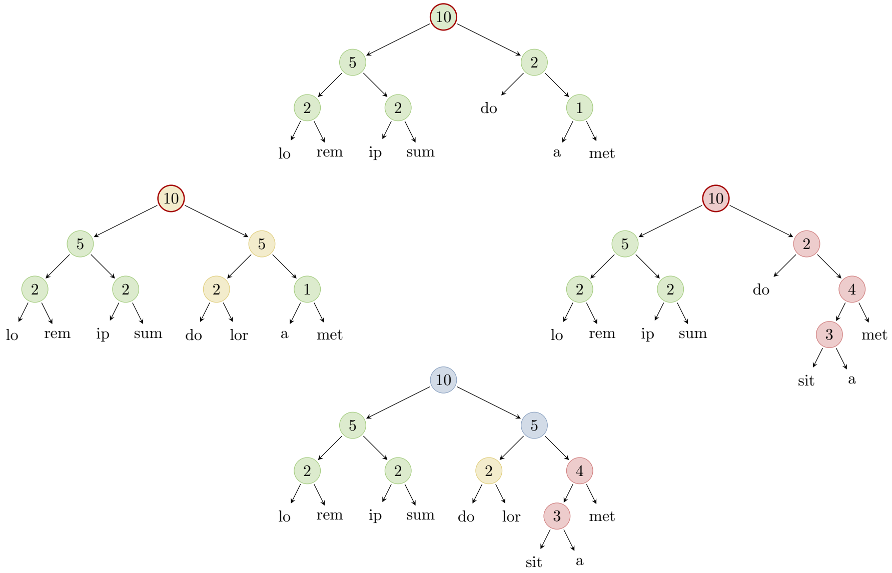

10
5
2
2
2
do
1
lo
rem
ip
sum
a
met
10
5
5
2
2
2
1
lo
rem
ip
sum
do
lor
a
met
10
5
2
2
2
do
4
lo
rem
ip
sum
3
met
sit
a
5
2
2
2
4
lo
rem
ip
sum
do
lor
3
met
sit
a

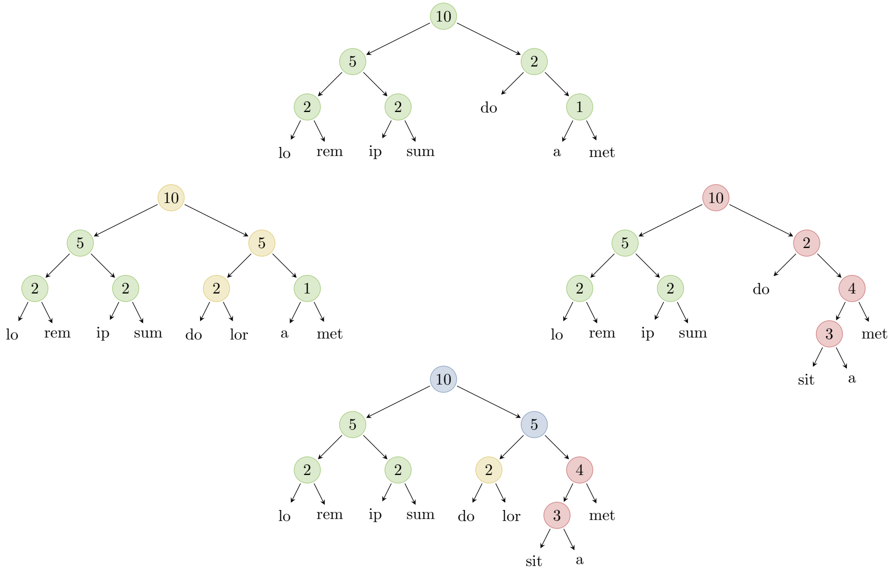

10
5
2
2
2
do
1
lo
rem
ip
sum
a
met


10
5
5
2
2
2
1
lo
rem
ip
sum
do
lor
a
met

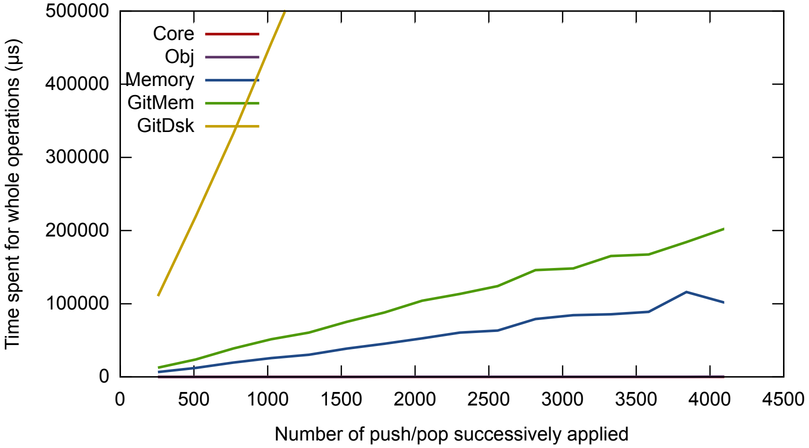

10
5
2
2
2
do
4
lo
rem
ip
sum
3
met
sit
a


5
5
2
2
2
4
lo
rem
ip
sum
do
lor
3
met
sit
a


10
5
2
2
2
do
1
lo
rem
ip
sum
a
met
10
5
5
2
2
2
1
lo
rem
ip
sum
do
lor
a
met
10
5
2
2
2
do
4
lo
rem
ip
sum
3
met
sit
a
10
5
5
2
2
2
4
lo
rem
ip
sum
do
lor
3
met
sit
a

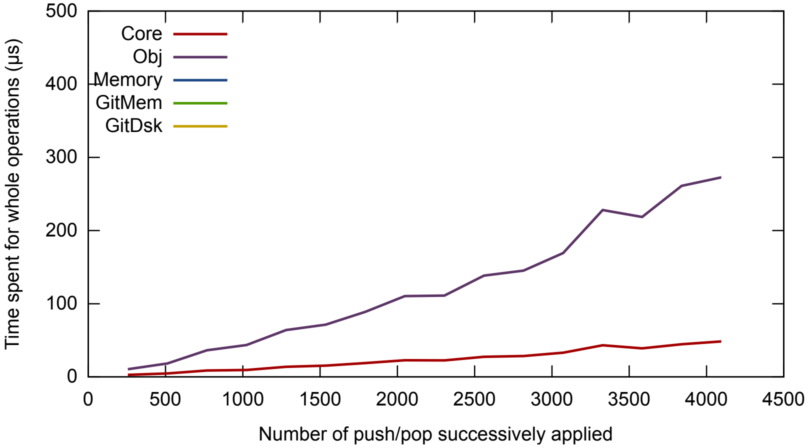

10
5
2
2
2
do
1
lo
rem
ip
sum
a
met
10
5
5
2
2
2
1
lo
rem
ip
sum
do
lor
a
met
10
5
2
2
2
do
4
lo
rem
ip
sum
3
met
sit
a
10
5
5
2
2
2
4
lo
rem
ip
sum
do
lor
3
met
sit
a

Benchmarking Irmin

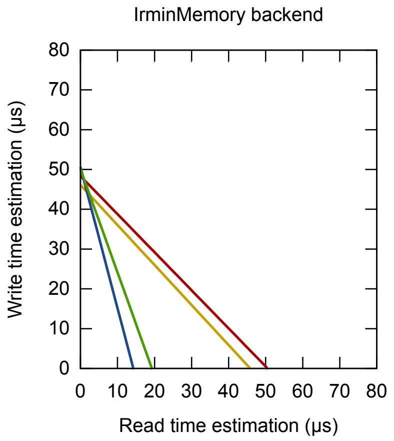


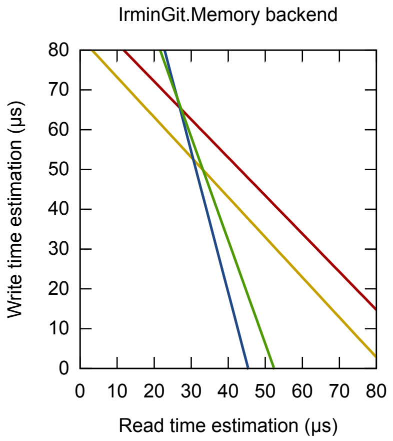
Time spent for whole operations (µs)
500000
400000
300000
200000
100000
0
Core
Obj
Memory
GitMem
GitDsk
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Number of push/pop successively applied

```
module ObjBackend ...  = struct
  type t = unit
  type key = K.t
  type value = V.t

  let create () = return ()
  let clear () = return ()

  let add t value =
    return (Obj.magic (Obj.repr value))

  let read t key =
    return (Obj.obj (Obj.magic key))

  let mem t key = return true
  ...
end
```



500
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Core
Obj
Memory
GitMem
GitDsk
Time spent for whole operations (µs)
Number of push/pop successively applied

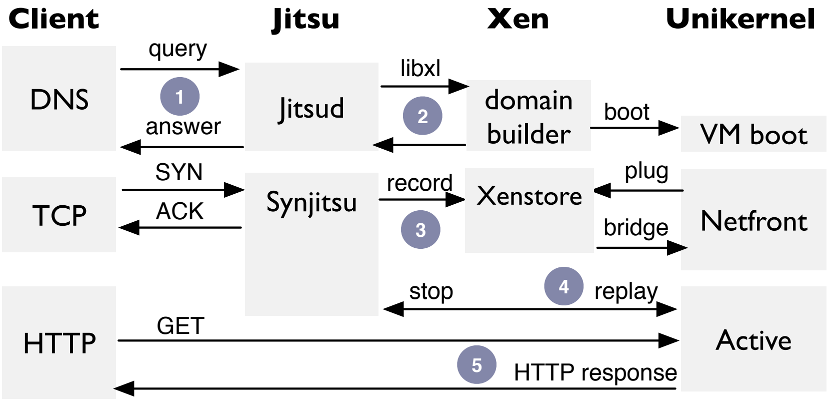

IrminMemory backend
Write time estimation (µs)
Read time estimation (µs)
0
10
20
30
40
50
60
70
80

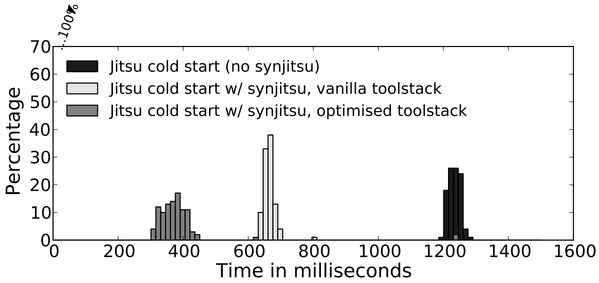

IrminGit.Memory backend
Write time estimation (µs)
Read time estimation (µs)
0
10
20
30
40
50
60
70
80

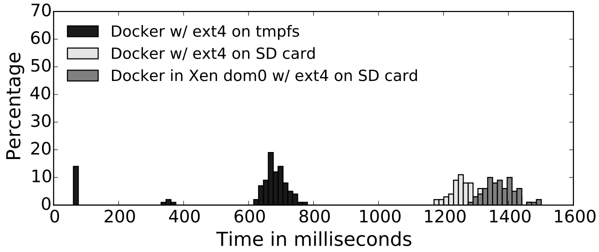
Status

- HTTP REST APIs for remote clients.
- JavaScript compilation for pure browser operation.
- Bidirectional lenses (Git commits map to Irmin commits from any direction)
- Open source at `https://github.com/mirage/irmin`
- JFLA 2015 paper got great feedback. Now building more data structures, evaluating block scheduling.

- **Jitsu: Just-In-Time Summoning of Unikernels**
  `https://www.youtube.com/watch?v=DSzvFwIVm5s`
- Ported complex Xen toolstack to use Irmin.
- Jitsu becomes the `inetd` of Xen:
  - Launch VMs in response to network requests in real-time.
  - Irmin coordinates toolstack RPCs with low latency.
  - Connection setup is proxied to eliminate packet loss.


Client
Jitsu
Xen
Unikernel
DNS
query
1
answer
Jitsud
libxl
2
domain builder
boot
VM boot
TCP
SYN
ACK
Synjitsu
record
3
Xenstore
plug
bridge
Netfront
stop
4
replay
HTTP
GET
Active
5
HTTP response


...100%
Jitsu cold start (no synjitsu)
Jitsu cold start w/ synjitsu, vanilla toolstack
Jitsu cold start w/ synjitsu, optimised toolstack
Percentage
0
10
20
30
40
50
60
70
0
200
400
600
800
1000
1200
1400
1600
Time in milliseconds


Docker w/ ext4 on tmpfs
Docker w/ ext4 on SD card
Docker in Xen dom0 w/ ext4 on SD card
Percentage
0
10
20
30
40
50
60
70
0
200
400
600
800
1000
1200
1400
1600
Time in milliseconds