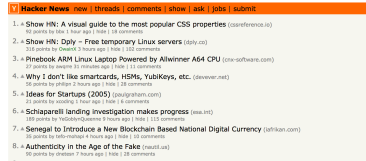
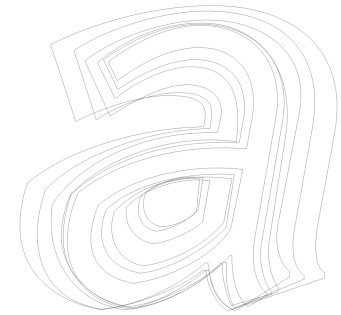


The Racket Virtual Machine as an application of CS 4400



Some Racket Applications



Hacker News

Arc

Racket



Game Content

DC

Racket

Practical Typography

Pollen

Racket



Telescope Controller

DrRacket

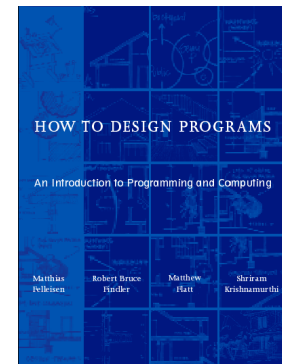
Racket



Synthesized Program

Rosette

Racket

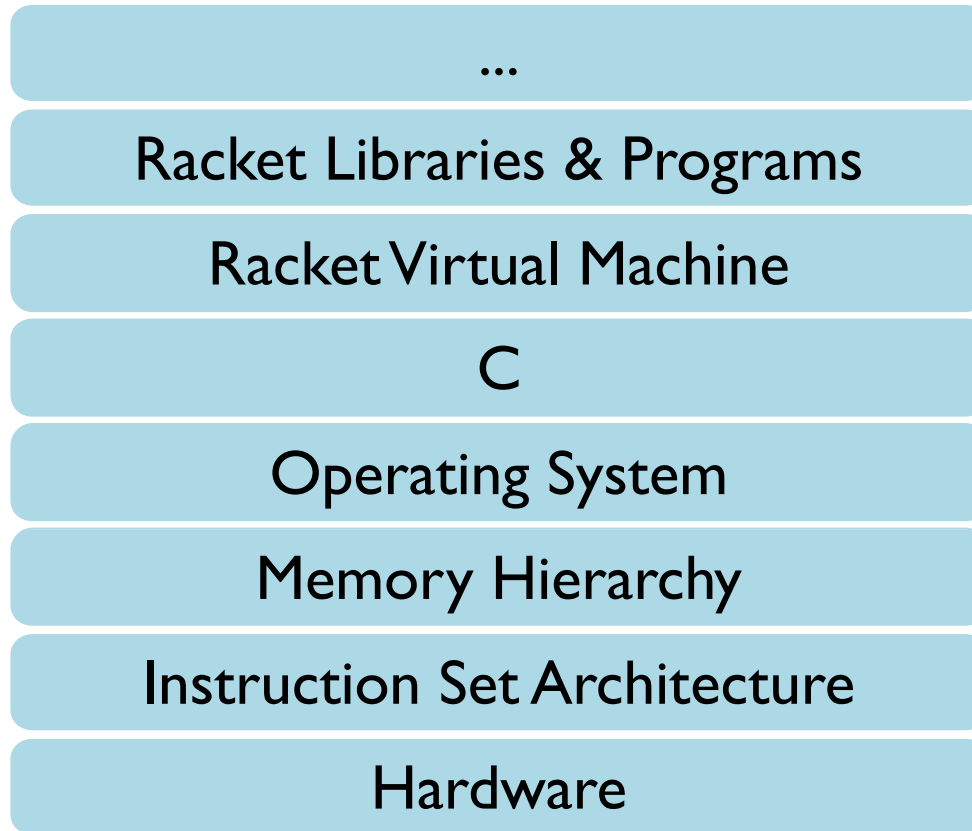


Homework

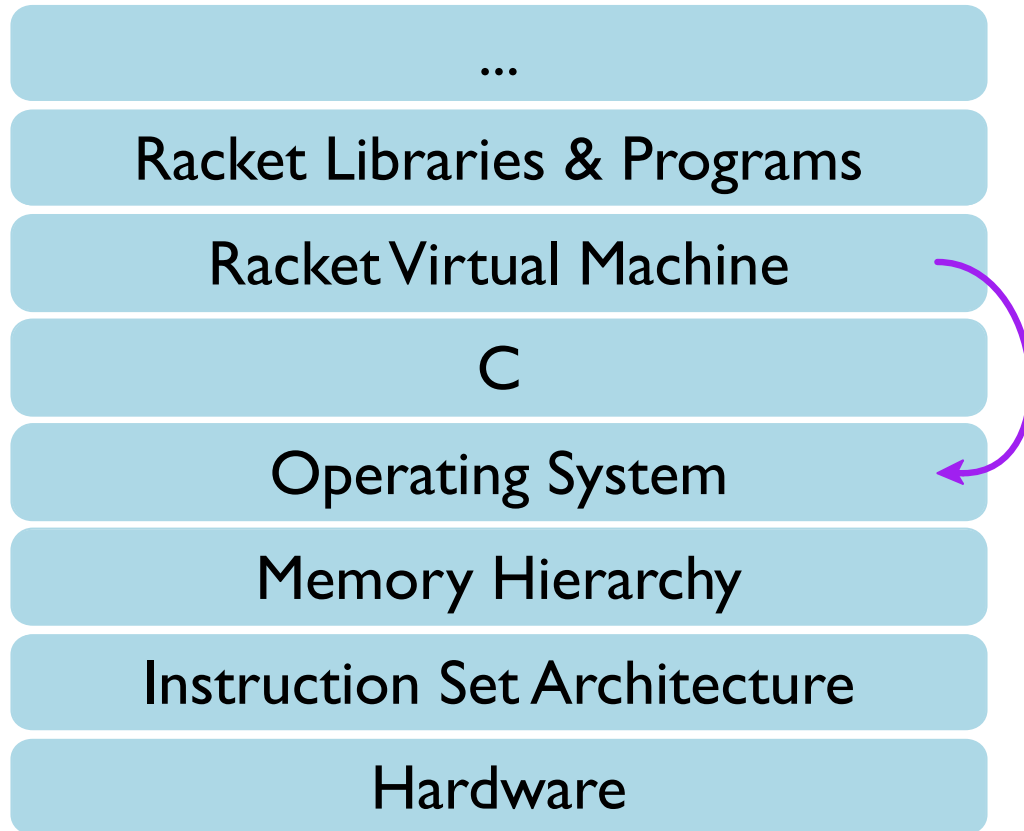
Beginner Student

Racket

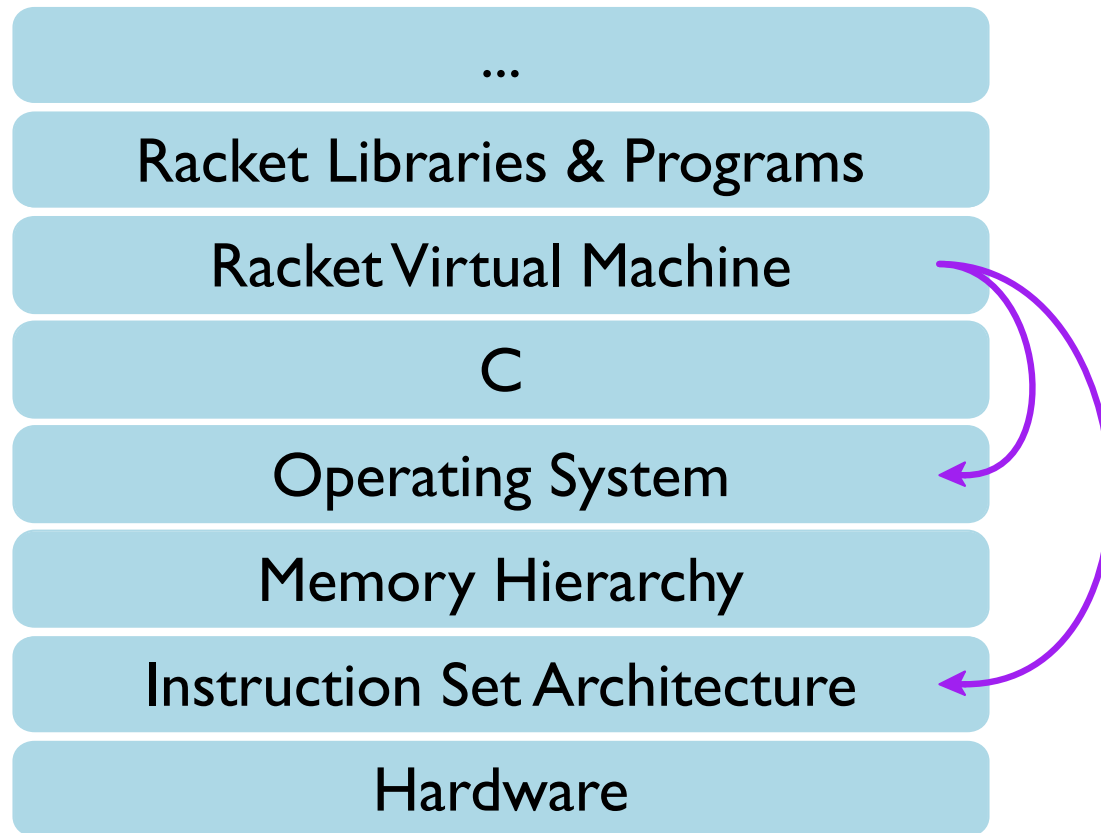
Virtual Machines



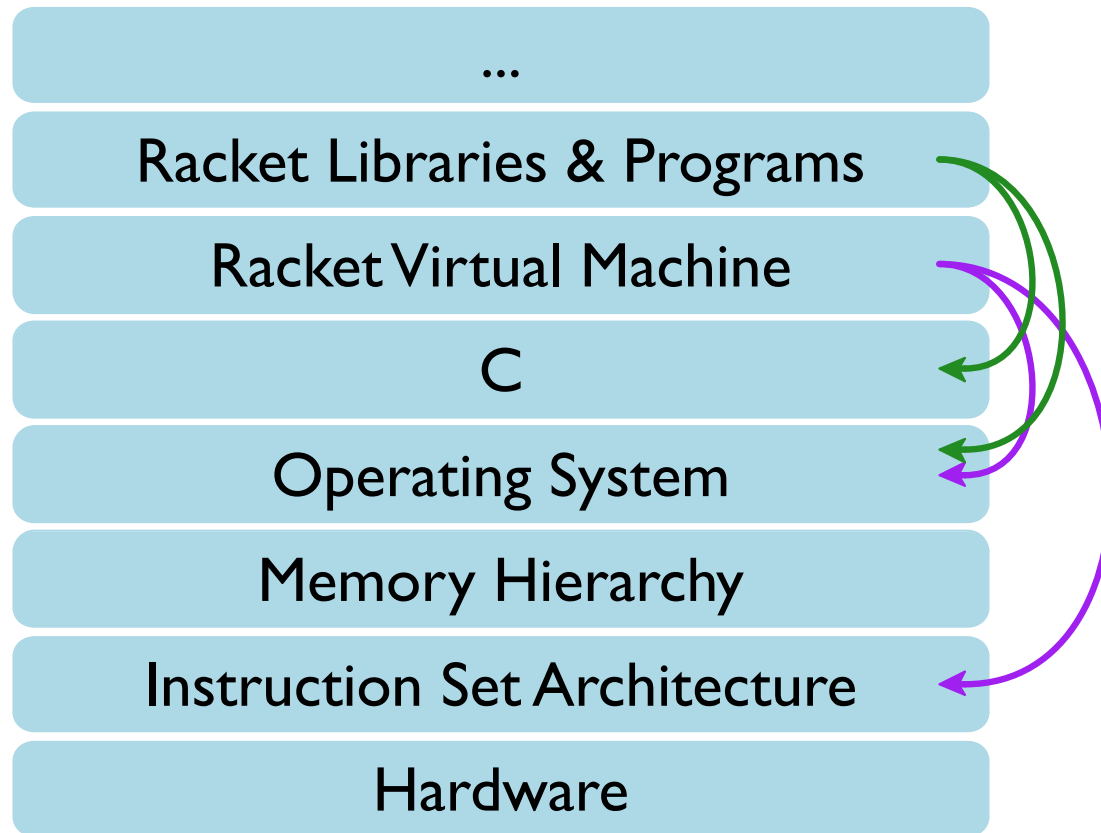
Virtual Machines



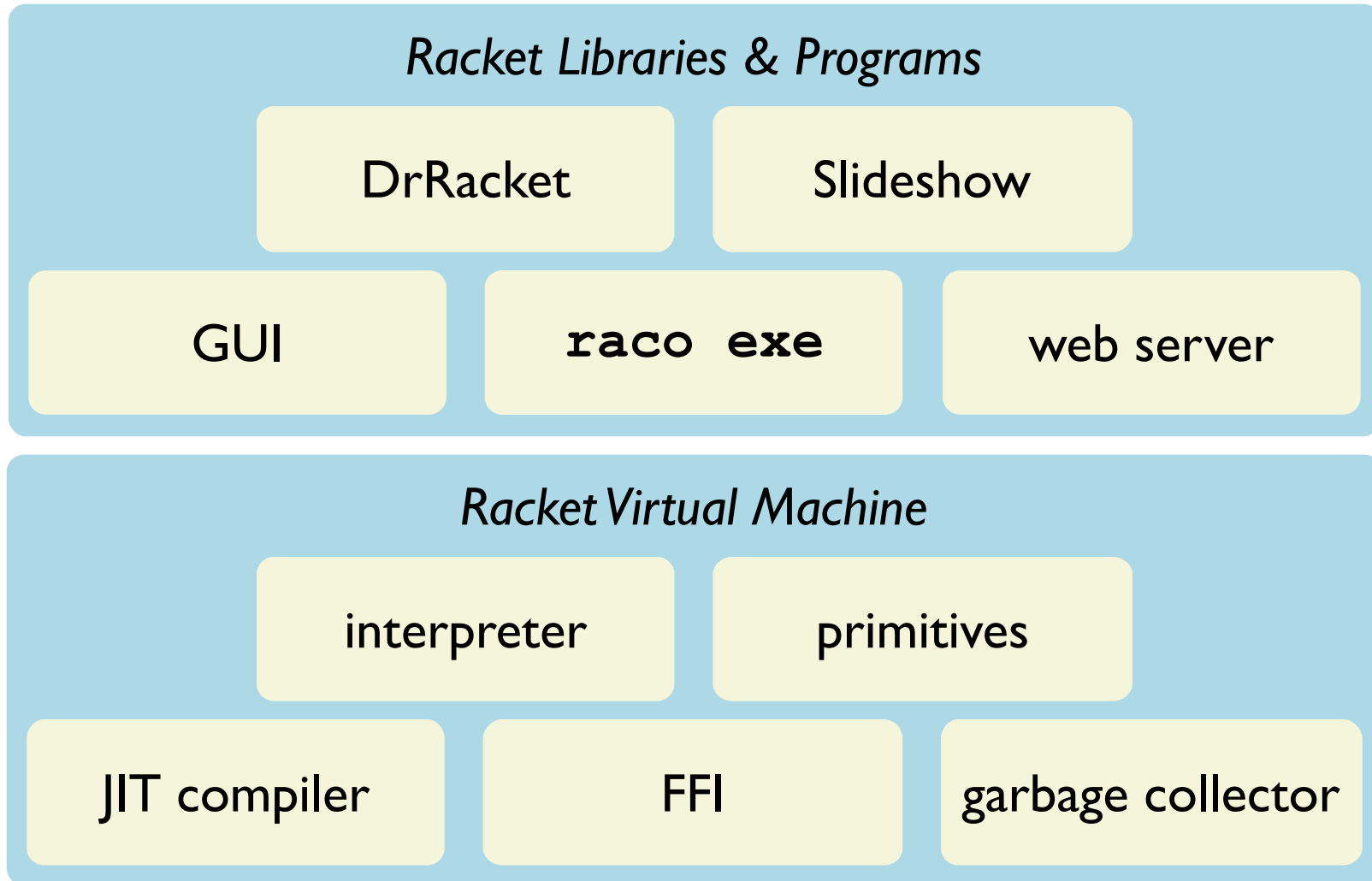
Virtual Machines



Virtual Machines



Racket Virtual Machine



C

The Racket runtime system is implemented in C

`src/racket`

Representing Numbers

- Representing fixnums: `SCHEME_INTP` and `SCHEME_TYPE` `scheme.h`
- `ADD` for fixnums `numarith.c`
- `SCHEME_RATIONAL_FROM_FLOAT` for `inexact->exact` `ratfloat.inc`

x86-64 Machine Model

Just-in-Time (JIT) compiler:

- `ARITH_ADD`

`jitarith.c`

Representing Control Flow

Just-in-Time (JIT) compiler:

- `"unbox"` implementation `jitinline.c`
- `list_ref_code` implementation `jitcommon.c`

See also github.com/mflatt/jit-demo

Representing Procedures

- `unsafe/ffi`

```
(define atoi
  (get-ffi-obj "atoi"
    #f
    (fun string -> int)))
```

- `backtrace`
- `continuations`

Arrays

- `array-ref`

`ffi/unsafe.rkt`

Structures

- `Scheme_Object` `scheme.h`
- `Scheme_Bignum` `schpriv.h`
- `Scheme_Small_Bignum` `schpriv.h`
- `Scheme_IR_Local` `schpriv.h`

Optimization

- `scheme_application_type` case in `scheme_do_eval` [eval.c](#)
- `XFORM_ASSERT_NO_CONVERSION` and `fd_write_string` vs. `fd_write_string_slow` [port.c](#)

More on Optimization

- Branch-prediction interaction in
`scheme_generate_non_tail_call` [jitcall.c](#)

Memory Hierarchy, Locality, Caches

- `repair_heap`'s fused loops for the `SIZE_CLASS_SMALL_PAGE` case

`newgc.c`

Linking

The `unsafe/ffi` functions work by dynamically loading shared libraries

- `ffi-lib` uses `dlopen`
- `get-ffi-obj` uses `dlsym` `rktio/rktio_dll.c`

ELF and Relocation

`raco exe` creates an executable by

- copying a stub binary that links to the Racket runtime system
- adding a new ELF section to hold bytecode for the Racket source

`collects/compiler/private/elf.rkt`

Processes

Racket runs `/bin/uname` to get the result of

```
(system-type 'machine)
```

```
string.c
```

More on Processes

The `subprocess` execs an arbitrary program

Implementation uses `fork` and `execve`, and `waitpid`

`rktio/rktio_process.c`

File Descriptors

Racket's I/O uses file descriptors directly

- `fd_get_string_slow`

`port.c`

`rktio/rktio_fd.c`

Signals

- **SIGINT** handler in `main` `main.c`
- **SIGCHLD** handler related to `subprocess`
`rktio/rktio_process.c`

Virtual Memory

Garbage collector allocates pages using `mmap`

Write permission is disabled to implement a ***write barrier*** for generational collection

Handler calls `designate_modified_gc`

`newgc.c`

Dynamic Memory Allocation

- `allocate`

`newgc.c`

More on Memory Allocation

- `do_malloc` uses a free list

`sgc.c`

- Segmented allocation

`sgc.c`

Garbage Collection

- Bootstrap with conservative collector [sgc.c](#)
- Convert C code to cooperate with precise GC
- Production GC is fairly complex [newgc.c](#)

See also github.com/mflatt/jit-demo

Network Programming

- DNS

[net/dns.rkt](#)

More Network Programming

- Web server
- `raco pkg`
- Git checkout

[net/git-checkout.rkt](#)

Concurrency

- File and network reads are multiplexed internally
- `getaddrinfo_in_thread` [rktio/rktio_network.c](#)

Synchronization

- Mutex at Racket-thread level protects hash tables (e.g., `hash_table_count`) [list.c](#)
- GC keeps a list of threads for cooperation on macOS [gc2/vm_osx.c](#)

...And More

What topics crucial to Racket *weren't* covered in CS 4400?

- Programming and data structures
- Interpreters and compilers
- Databases
- GUIs and graphics
- Rules and strategies for portability