Optimizing Synchronization Primitives

Zebediah Figura

Some Win32 sync primitives

- Kernel objects
  - Events (auto, manual)
  - Mutexes
  - Semaphores
  - Timers
  - Keyed events
  - Processes, threads, named pipes…
- In-process objects
  - Critical sections
  - SRW locks
  - Condition variables
  - `WaitOnAddress()`/ `WakeByAddress()`

Lots of little quirks…

- `PulseEvent()`
- Cross-process by name or `DuplicateHandle()`
- Wait-all
- Alertable waits
- `NtQuery*()`

Kernel objects are slow

- wineserver round trip, because cross-process
- wineserver is single-threaded
- Many modern games are very multi-threaded
- Some work was already done

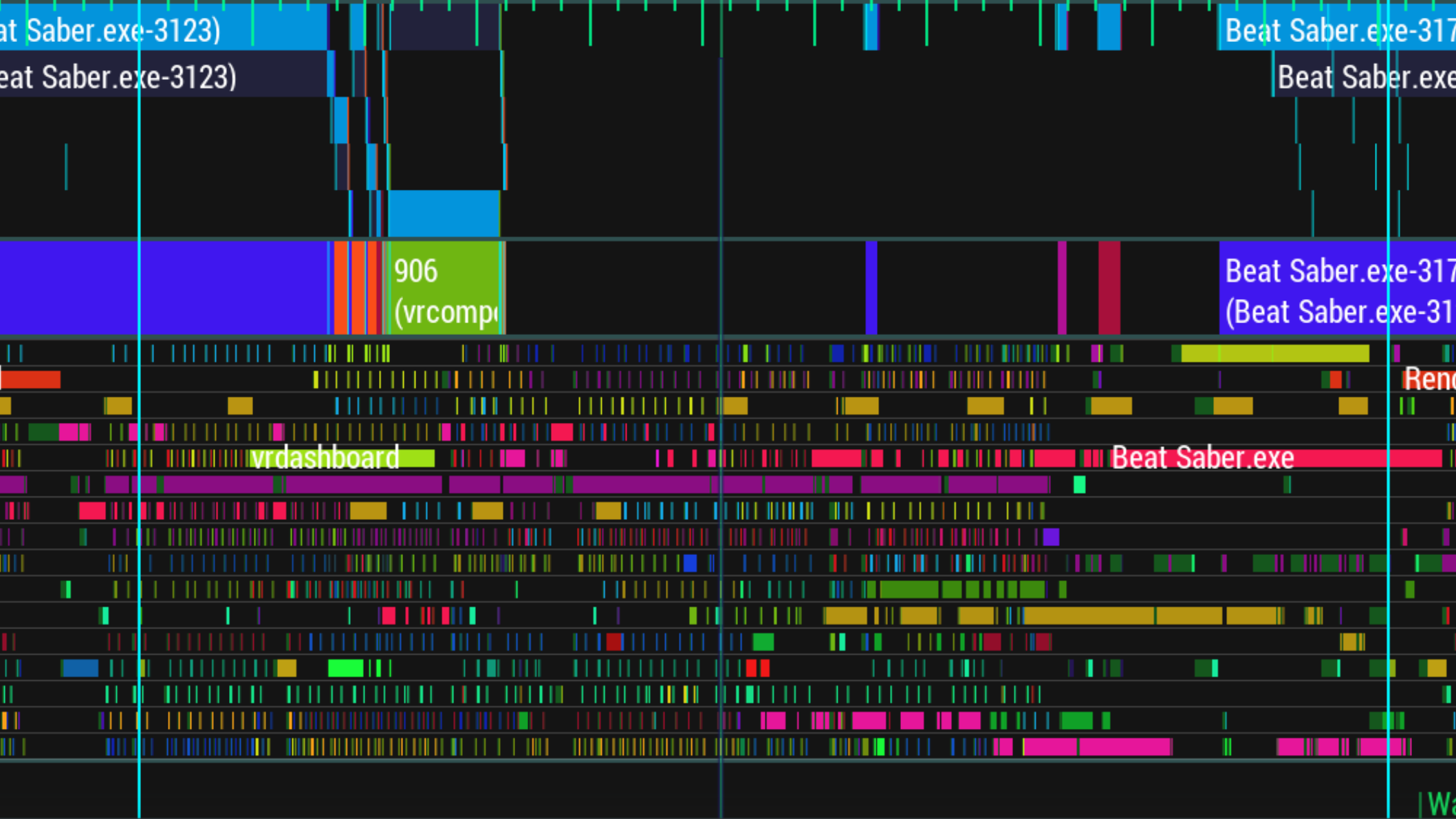

Beat Saber
Beat
wineserver
wineserver
wineserver

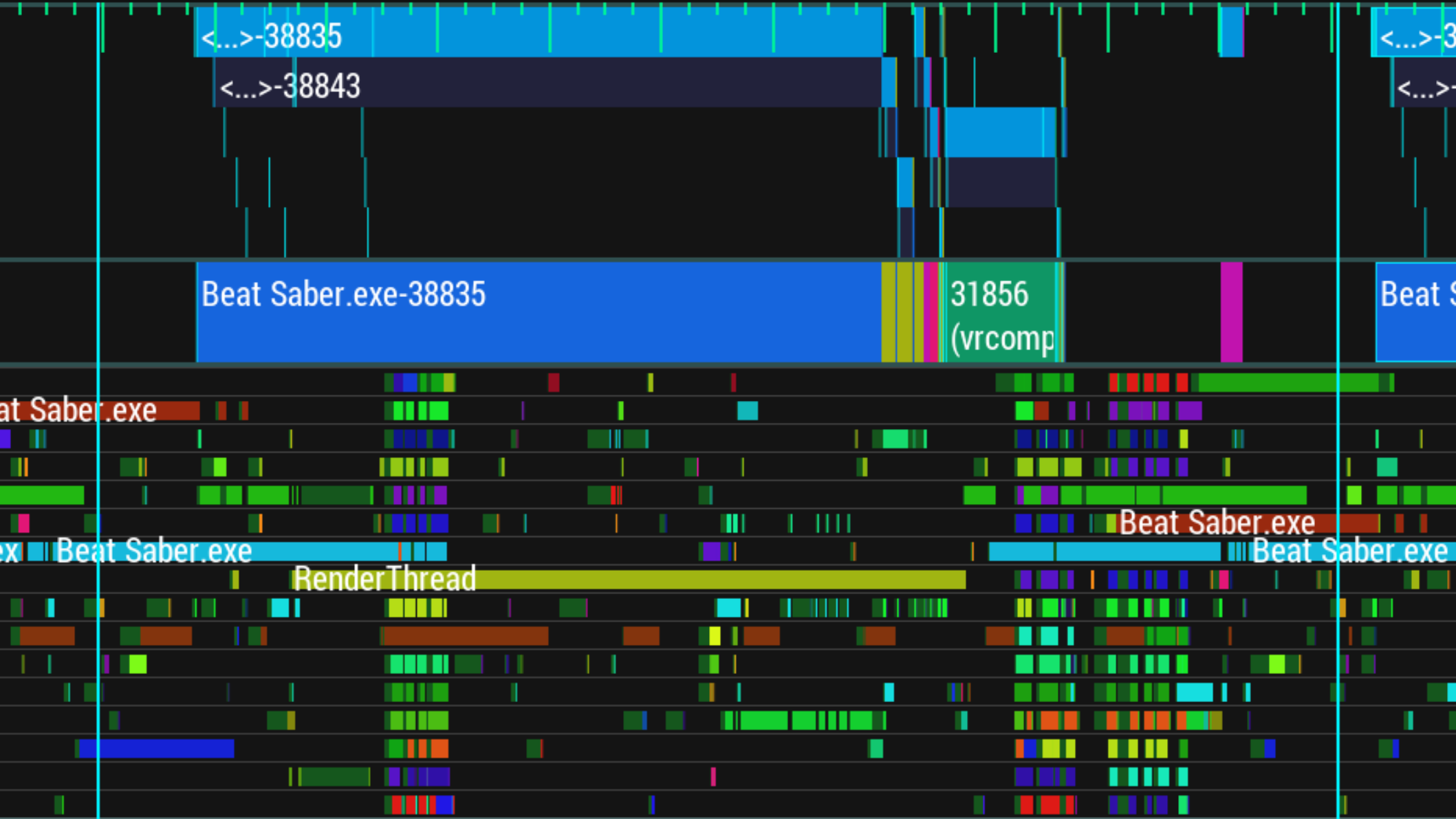
`eventfd(2)`

- Single 64-bit value, increment on `write(2)`, decrement on `read(2)`, blocks if zero
- We can sleep or check if signaled with `poll(2)`
- Events and mutexes are signaled if nonzero
- Semaphores use `EFD_SEMAPHORE`
- Extra state (mutex, semaphore) eventually needs shared memory
- APC waits just use another “event”

How did it turn out?

- Pretty good
- The kernel does all of the locking, so correctness isn't too hard
- Wait-all is broken, but not in a way that matters
- `PulseEvent()` is also broken, but it doesn't matter


at Saber.exe-3123)
eat Saber.exe-3123)
Beat Saber.exe-317
Beat Saber.exe
906
(vrcompo
Beat Saber.exe-317
(Beat Saber.exe-31
Rend
vrdashboard
Beat Saber.exe


<...>-38835
<...>-38843
Beat Saber.exe-38835
31856
(vrcomp
Beat Saber.exe
Beat Saber.exe
Beat Saber.exe
RenderThread

But can it get faster?

- Well… maybe
- Kernel is doing all of the work, so esync is fast
- But system calls are expensive
  - But Windows has to do them too…
- Store event state locally

But can it get *faster?*

- Futexes don’t need a syscall to get/set state
- But we can’t wait on more than one
- …unless we change the kernel

But can it go upstream?

- We can't use shared memory
  - We need it for mutexes, semaphores, `NtQuery*()`, event optimizations
- `PulseEvent()` is broken
- Wait-all is kind of broken

But can it go upstream?

- Add the missing pieces in the kernel
  - Pros: simple, matches Windows
  - Cons: performance could be better, is this right for Linux?
- Use migration
  - Pros: avoids shared memory
  - Cons: very hard to get right
- Do scheduling in user space
  - Pros: `PulseEvent()` and wait-all can be correct, best performance?
  - Cons: Needs locking, is a bit tricky, APCs?