A Software Microscope

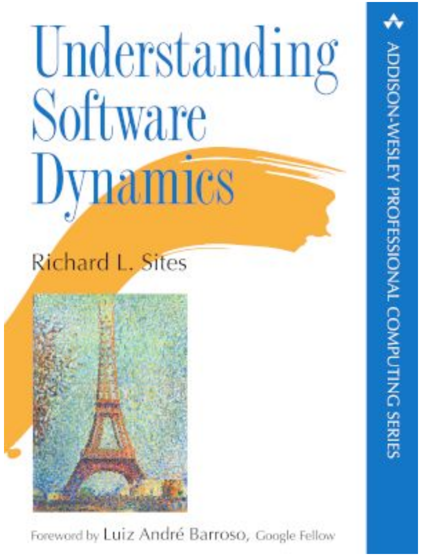
Dick Sites
Scalable Tools Workshop
June 2022

Talk outline

Goals
Kernel-User tracing
Complex software

Example: waiting for CPU
Example: Executing too slowly
Example: Waiting for locks

The Knuth challenge
Summary



Software is like pond water

It is the behavior over time that matters

Goals

See what every CPU core is executing every nanosecond

See for every process when it is executing and when it is blocked

See for a blocked process what it is waiting for

See interference between processes

See interference between the operating system and processes

With less than 1% overhead in a busy time-constrained system

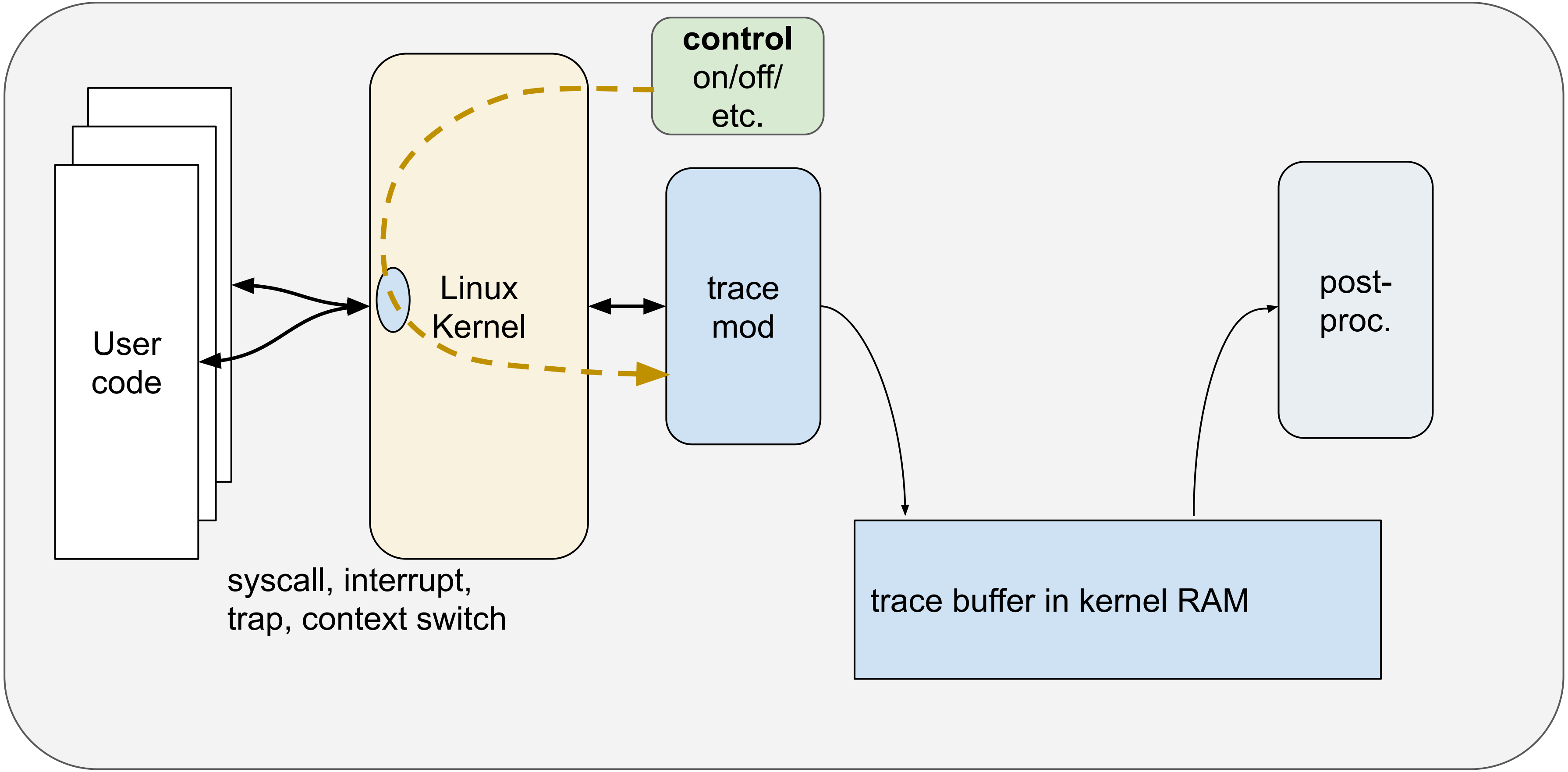
Kernel-User Tracing

Kernel-User tracing

KUtrace is a software microscope that records a *trace* of every transition between kernel code and user code on every CPU core, with less than 1% overhead.

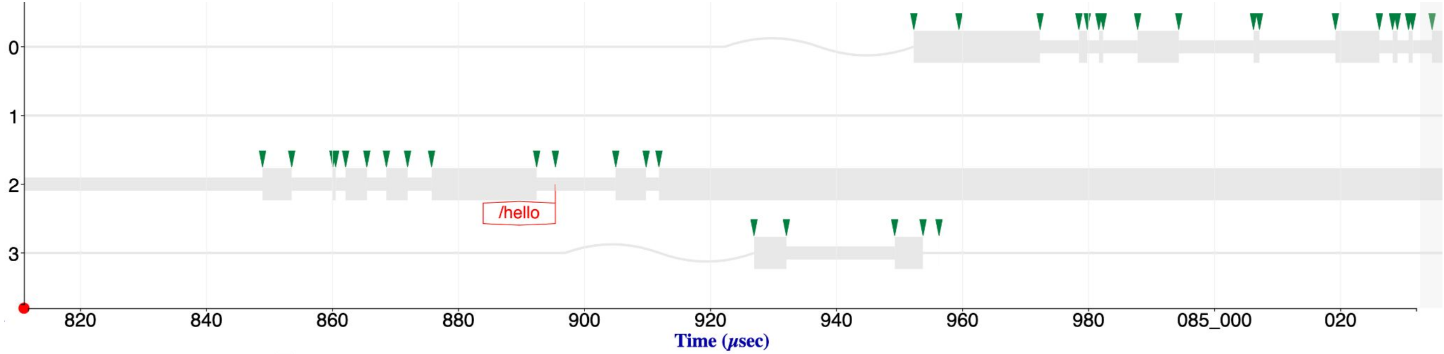
It is implemented via a small set of Linux kernel patches that record four-byte transition events into a reserved kernel RAM buffer.

Postprocessing turns raw traces into dynamic HTML timelines that you can pan and zoom.


control
on/off/
etc.
User
code
Linux
Kernel
trace
mod
post-
proc.
syscall, interrupt,
trap, context switch
trace buffer in kernel RAM

KUtrace events

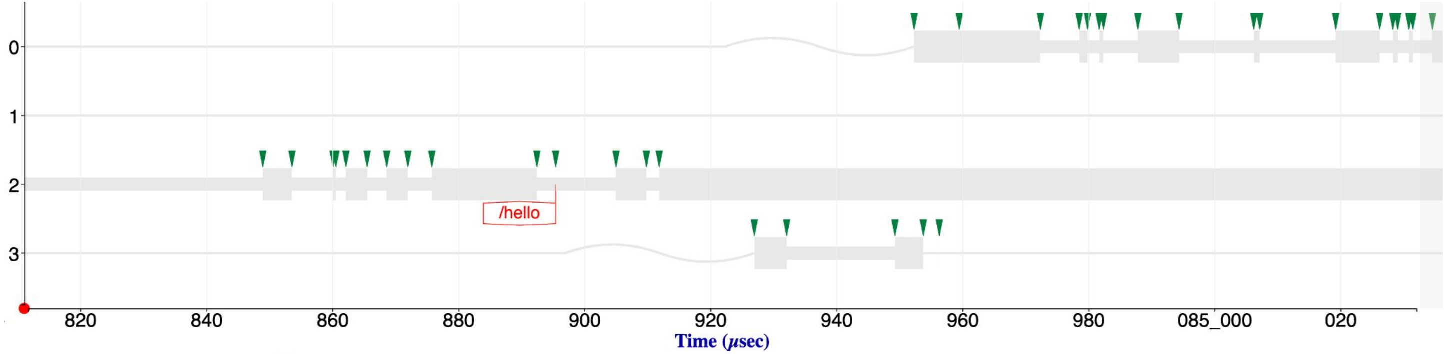
Events



Each green triangle is a kernel-user **transition**, recorded as a four-byte event: 20 bits of timestamp and 12 bits of which event -- which syscall/return, interrupt/return, fault/return, context switch

KUtrace events postprocessed into timespans

Events



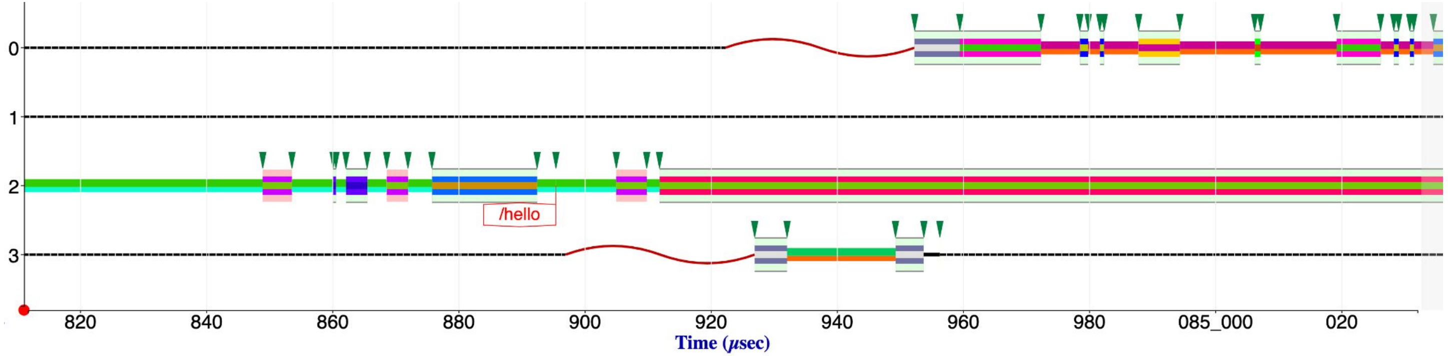
Timespans

thin black: idle

half height: user

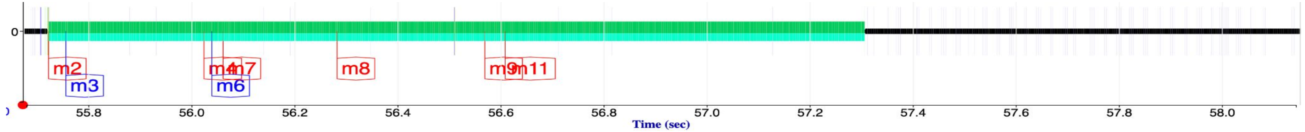
full height: kernel

sine: exiting low power mode

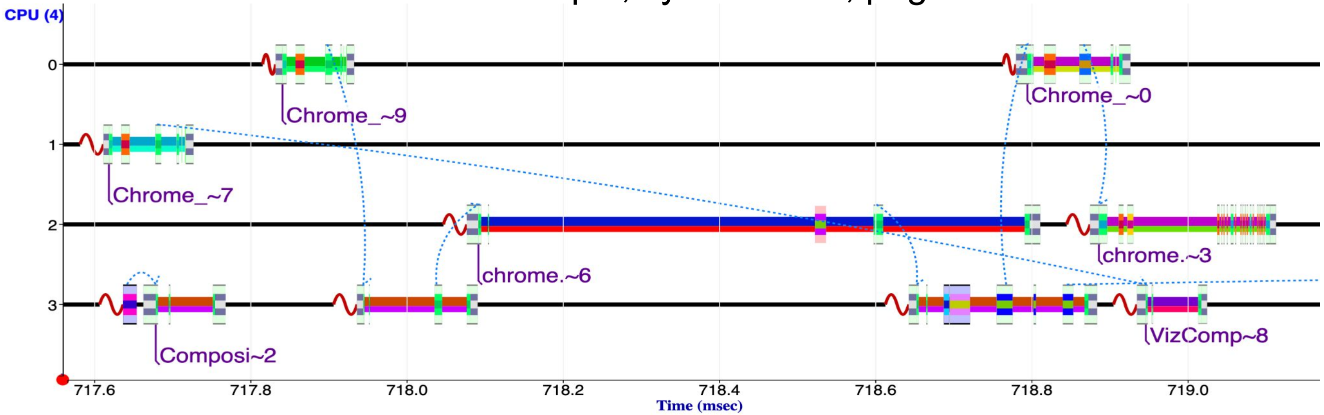


Complex Software Dynamics

Simple software: Single thread, CPU bound (e.g. benchmarks)



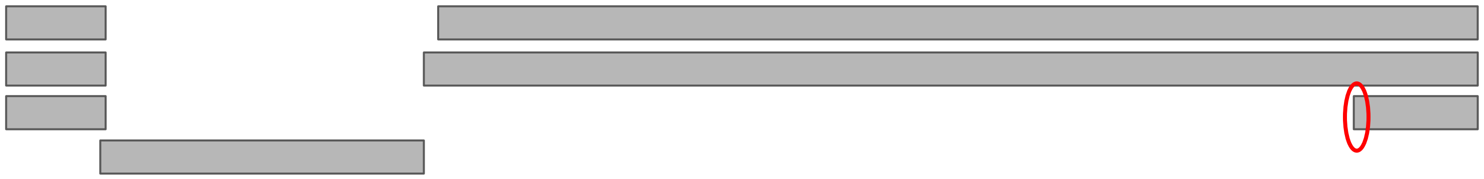
Complex software: Multiple threads blocking and waking each other up, interrupts, system calls, page faults



Waiting for CPU

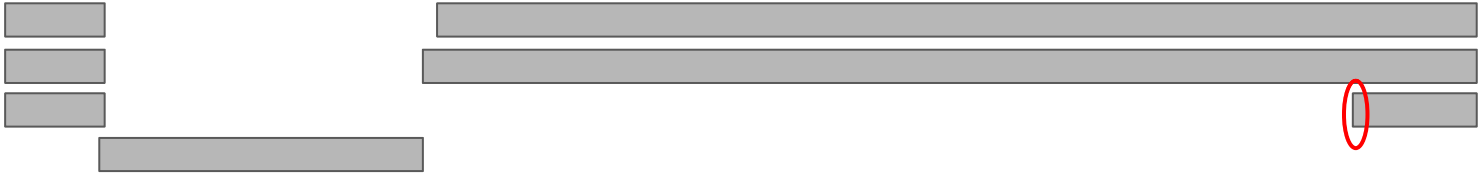
Waiting for CPU

Invisible: Three threads wait on a fourth, then resume. Why longer wait?



Waiting for CPU

Invisible: Three threads wait on a fourth, then resume. Why longer wait?

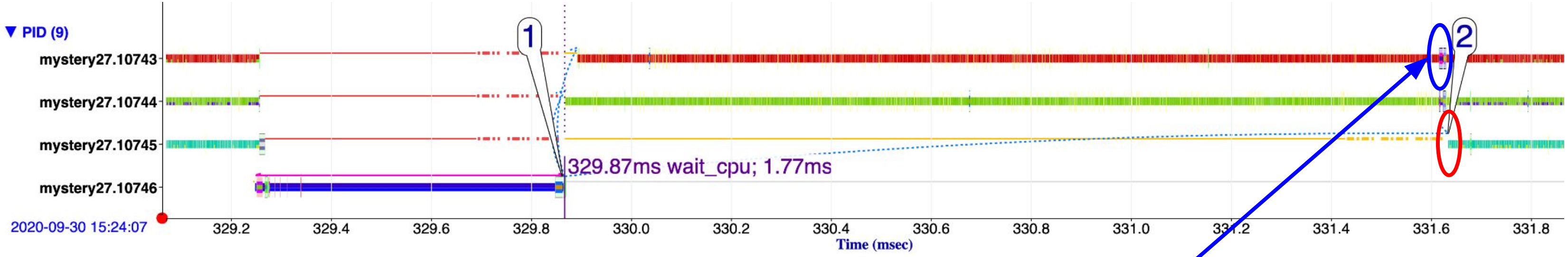


Visible: Long one is waiting almost 2 msec to get a CPU assigned

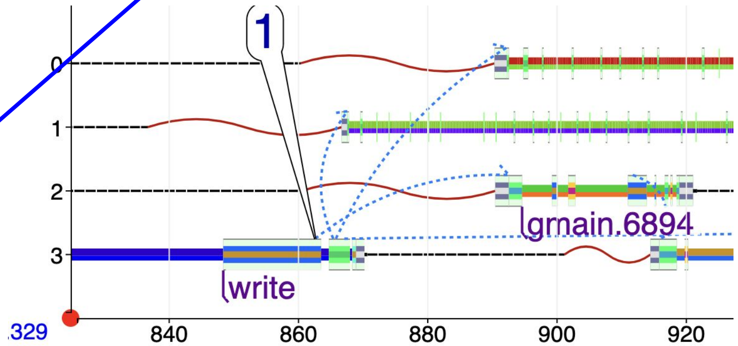
Waiting for CPU

Invisible: Why longer wait?

Visible: Long one is waiting almost 2 msec to get a CPU assigned



At (1), fourth thread does a write that wakes up gmain (Gnome display), and *then* restarts first three threads. Not enough CPUs to go around, so last wakeup waits. Linux scheduler fail: waits until a **timer interrupt** 1.77 msec later to restart.



Waiting for CPU, summary

Waiting for CPU comes from ...

- Busy CPUs
- Scheduler's too-strong affinity to task's last-used core
- Delays coming out of power-saving states
- Complex interactions between user code, kernel code, and the scheduler

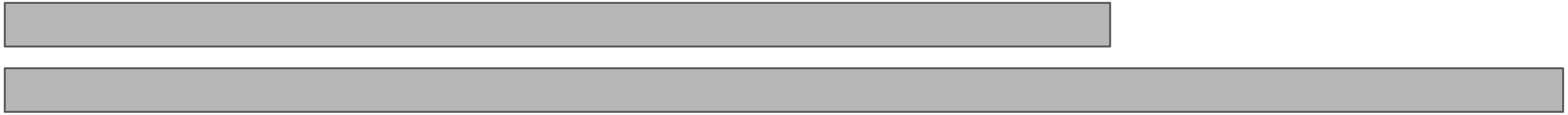
Wakeup events tell us what a thread was waiting for.

KUtrace has such low overhead that it does not disturb Heisenbugs.

Executing Too Slowly

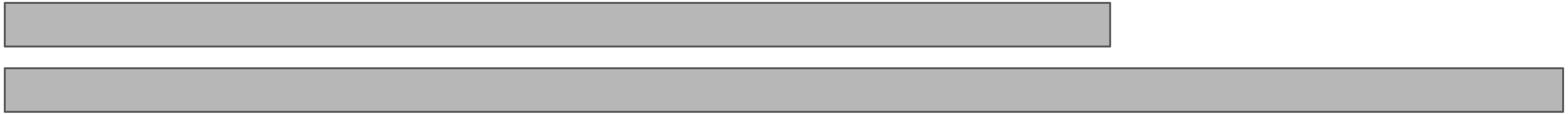
Executing Too Slowly

Invisible: Two runs of same identical benchmark. Why 40% slowdown?

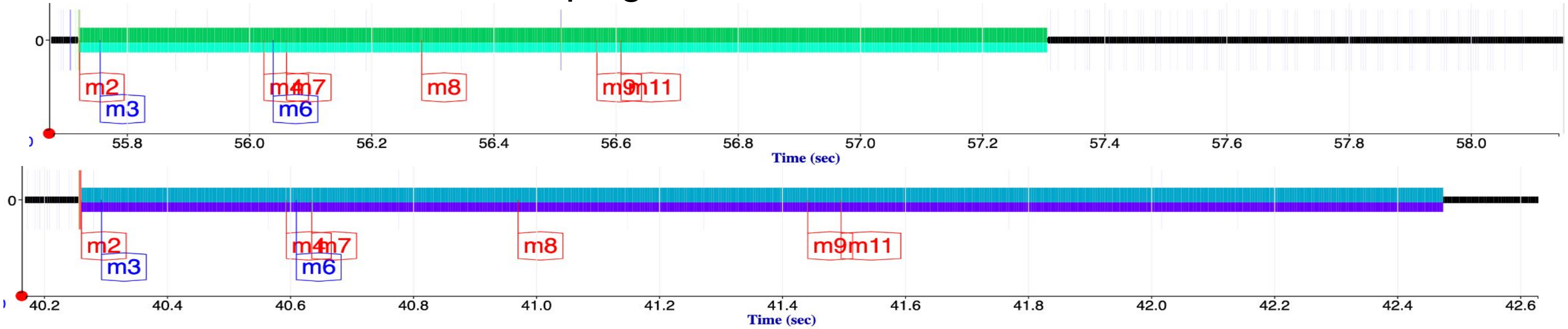


Executing Too Slowly

Invisible: Two runs of same identical benchmark. Why 40% slowdown?



Visible: Some but not all loops get 35-65% slower



Executing Too Slowly

The same code but sometimes executing slowly means that there is some form of **interference** --

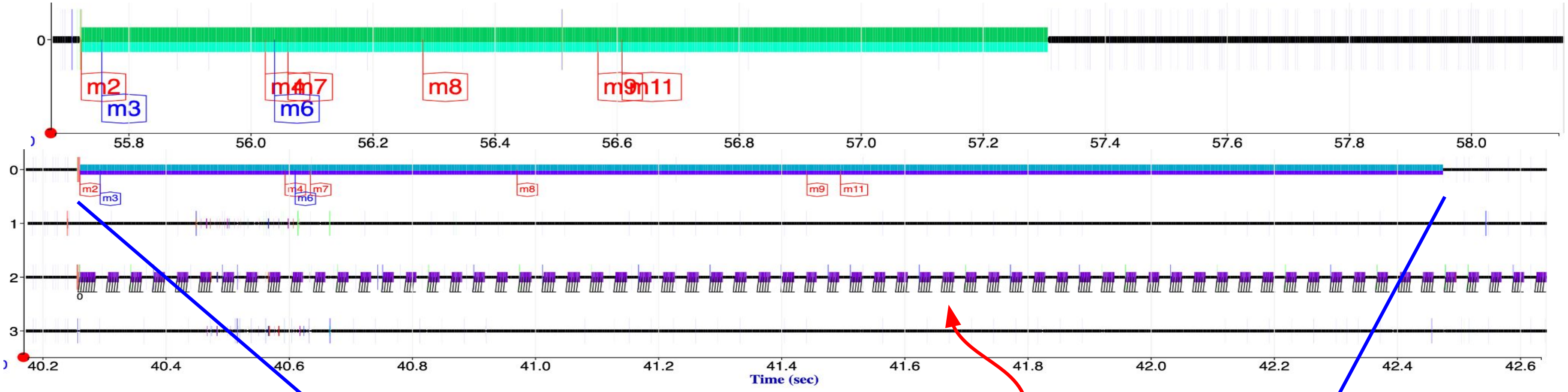
which can only come from use of shared hardware resources or shared software critical sections.

Interference comes from **what else** is running.

Executing Too Slowly

Invisible: Two runs. Why 40% slowdown?

Visible: What else is running?

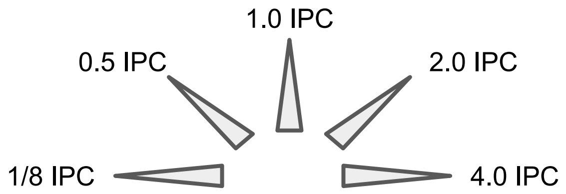


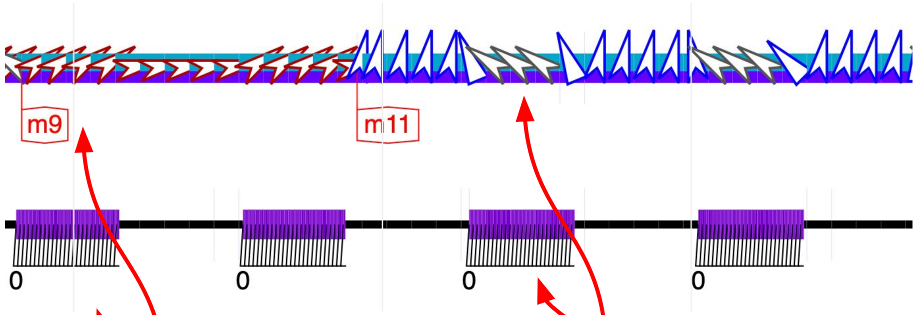
The long run executes slowly because of **another** program.
(Interference is at the floating divide execution unit.
Loops m2 to m6 do not use much floating-point.)

Executing Too Slowly

Invisible: Two runs. Why 40% slowdown?

Visible: What else is running?





The long run executes slowly because of **another** program.
When it runs, the benchmark IPC drops (speedometer triangles).
1.4x for m9 loop, **3x** for m11 loop.

Executing Too Slowly, summary

Executing too slowly comes from ...

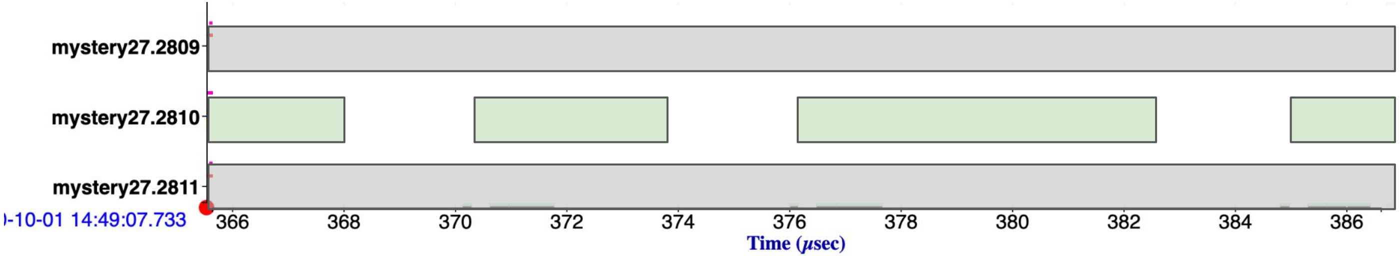
- Other-thread, other-program, or operating-system *interference* from use of some shared resource: CPU, memory, disk, network, locks
- Power-saving slow CPU clock frequency
- Slow exit from power saving

Microsecond-scale IPC reveals the interference between tasks.

Waiting for Locks

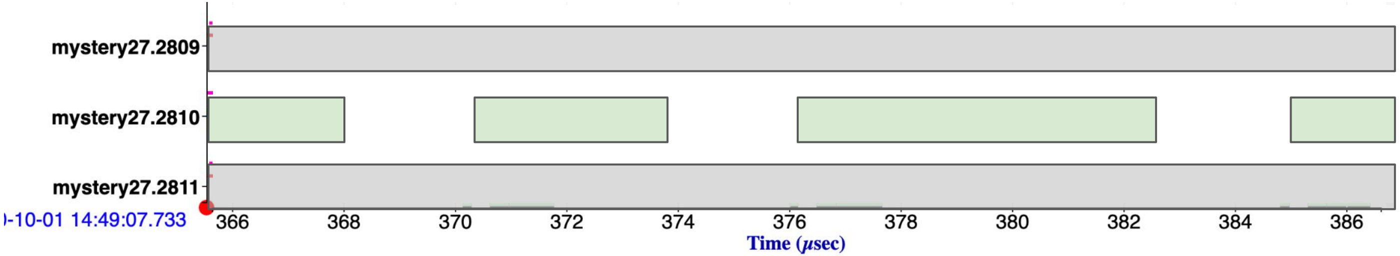
Waiting for Locks

Invisible: Two threads wait a long time for lock; middle thread has it

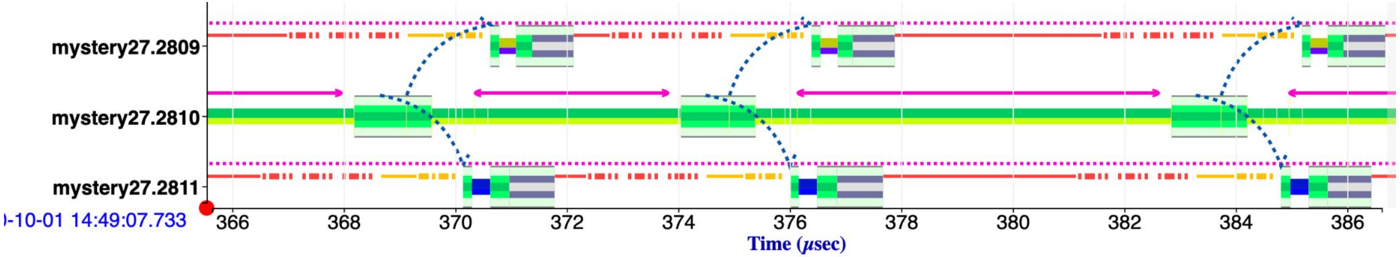


Waiting for Locks

Invisible: Two threads wait a long time for lock; middle thread has it



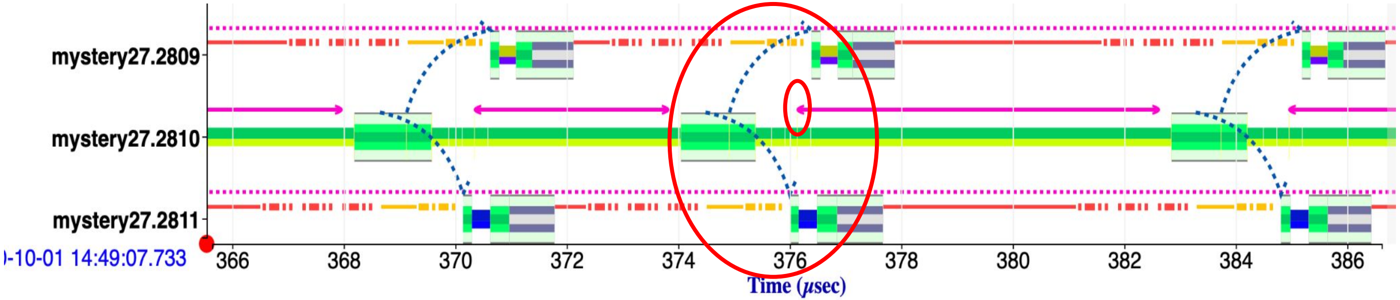
Visible: Middle thread **re-acquires** lock multiple times



Waiting for Locks

Invisible: Middle thread **starves out** the others

Visible: Middle thread re-acquires lock multiple times



Each time middle thread frees the lock, it wakes up the other two. But **before they can run**, it re-acquires the lock. Rinse and repeat ... goes on for 84 msec!

Waiting for Locks, summary

Waiting for locks comes from

- Other threads that are holding the lock
- (Hint: fix those, not the waiting thread)
- (But first you have to know which ones)

Seeing lock acquire, hold, release is important.

Recording *which* lock is important.

The Knuth Challenge

Make a thorough analysis of everything your computer does during one second of computation. -- Don Knuth 1989

The Knuth Challenge

Make a thorough analysis of everything your computer does during one second of computation. -- Don Knuth 1989

"Sites and KUtrace met my 33-year-old one-second Challenge"
-- Don Knuth, March 2022

Overall Summary

Summary

See what every CPU core is executing every nanosecond

See for every process when it is executing and when it is blocked

See for a blocked process what it is waiting for

See interference between processes

See interference between the operating system and processes

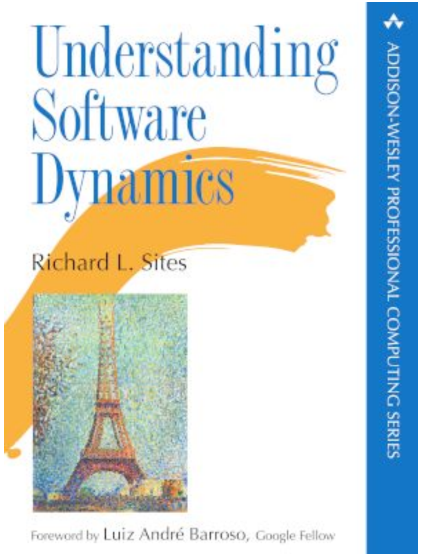
With less than 1% overhead in a busy time-constrained system

KUtrace does all this

References

Book:

Richard L. Sites, *Understanding Software Dynamics*,
Addison-Wesley 2022

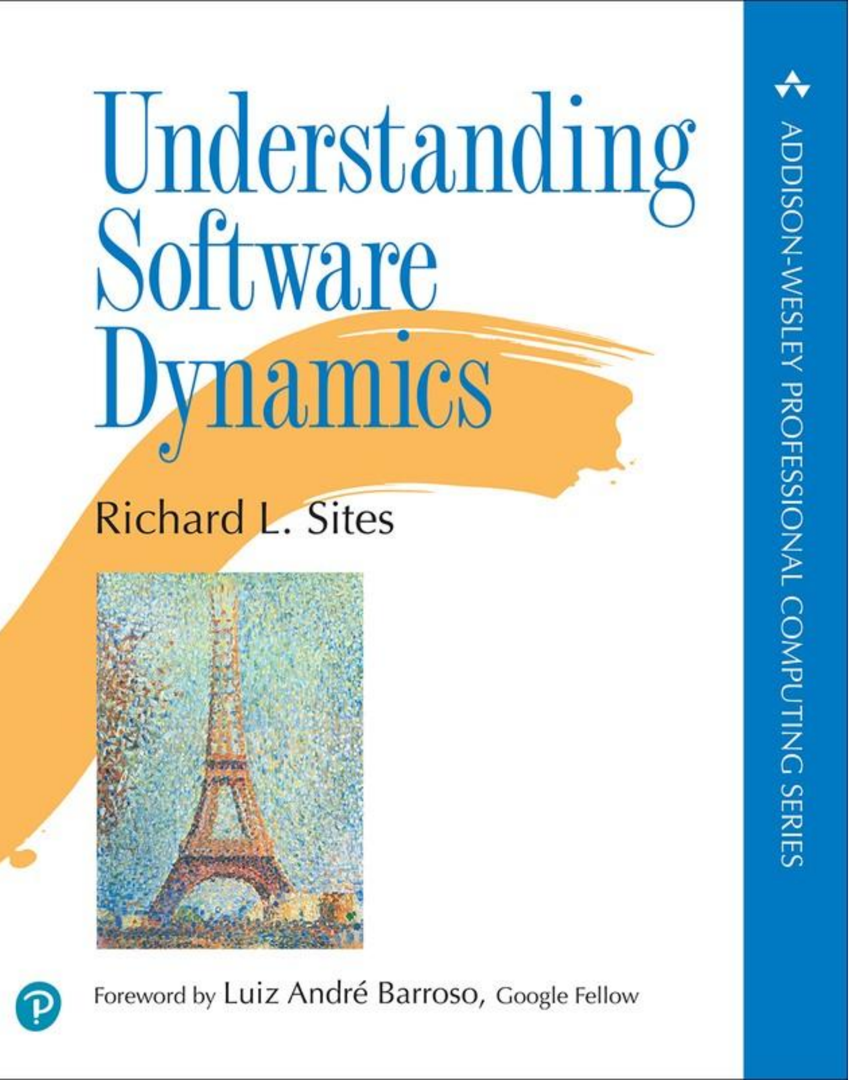


Patches for AMD x86, Intel x86, ARM 64-bit (RPi-4B),
RISC-V, plus postprocessing code, book code, book HTML:

github.com/dicksites/kutrace

Longer talk, Stanford EE380, March 2022 (Knuth comment at 1:10:00):

https://www.youtube.com/watch?v=D_qRuKO9qzM



Order & Save 35%* on Book or eBook at informit.com/dsites

- Use code **DSITES** during checkout
- Offer only good at informit.com
- Free U.S. shipping
- eBook – DRM-Free PDF, EPUB, & MOBI files

<u>Also Available</u>
Booksellers including Amazon and bn.com, and in O’Reilly’s Online Learning subscription service (aka Safari).

*Discount code DSITES is only good at informit.com and cannot be used on the already discounted book + eBook bundle or combined with any other offer.

**Outside the U.S. print books. Please check your local or online store where you purchase technical related books. If your order print books from InformIT, your order is subject to import duties and taxes, which are levied once the package reaches the destination country.