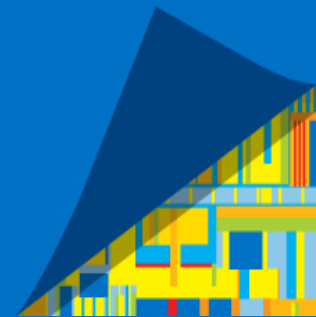
WSTS 2015

Semantics and Software APIs for the Efficient Use of Time in Time Sensitive Systems

Using General Purpose Compute Hardware

Kevin B. Stanton, Ph.D.

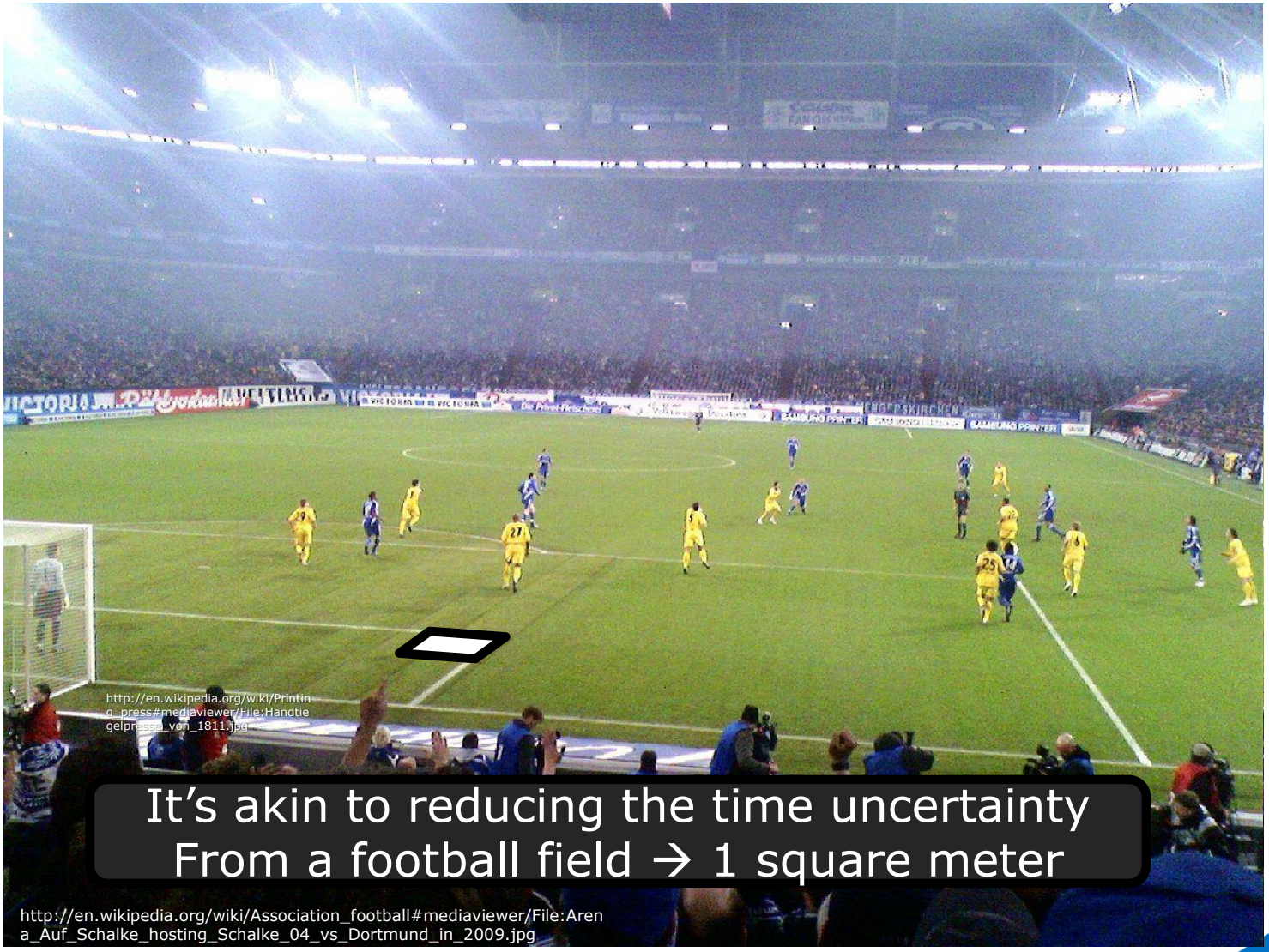
Sr. Principal Engineer, Intel



ABSTRACT

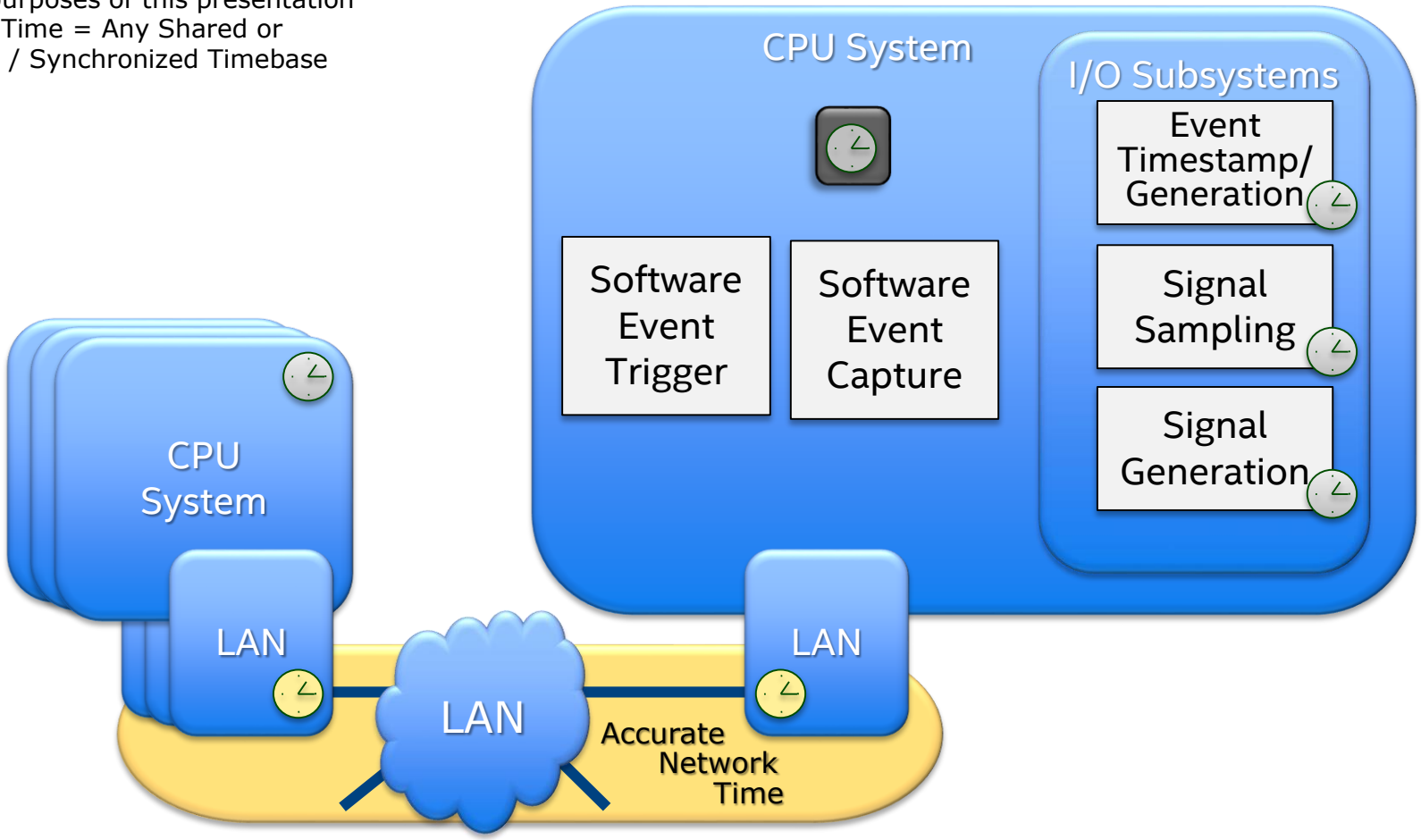
While enormous improvements have been made in the accuracy of the TIME delivered to networked systems, access to precise, distributed time by software applications and computer peripherals has not kept pace. Enhancements are needed to provide software semantics and APIs at all levels in the system in order to make network time fully usable within computer and CPS devices and their peripherals without causing unnecessary disruption to current software methods. This talk outlines both architectural and implementation challenges and proposes a framework for addressing them.

Now that we have Synchronized Network Time, e.g. IEEE 1588/PTP



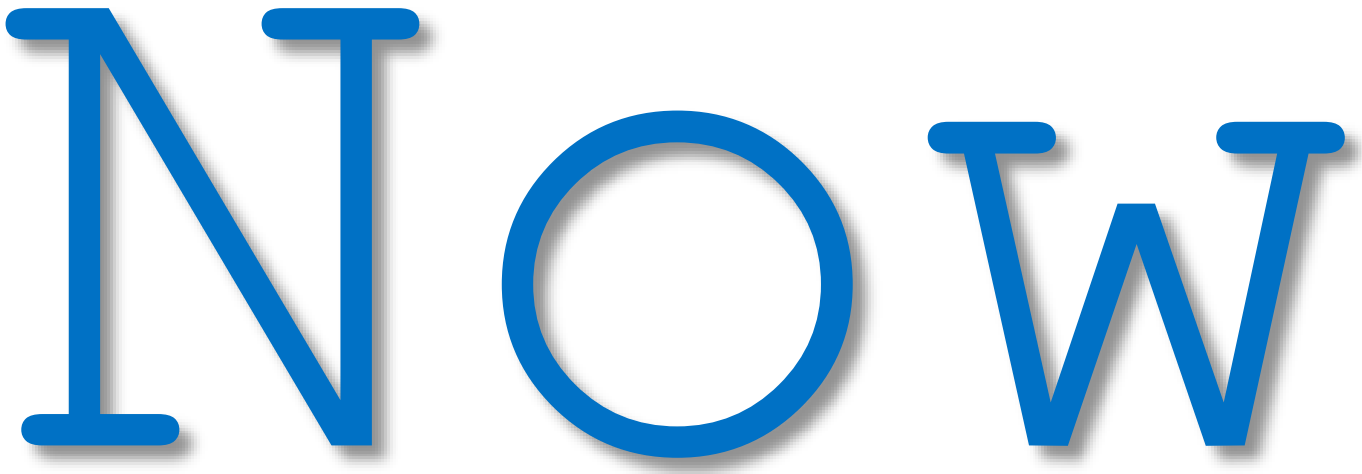
System Time Architecture

For the purposes of this presentation
Network Time = Any Shared or
Common / Synchronized Timebase



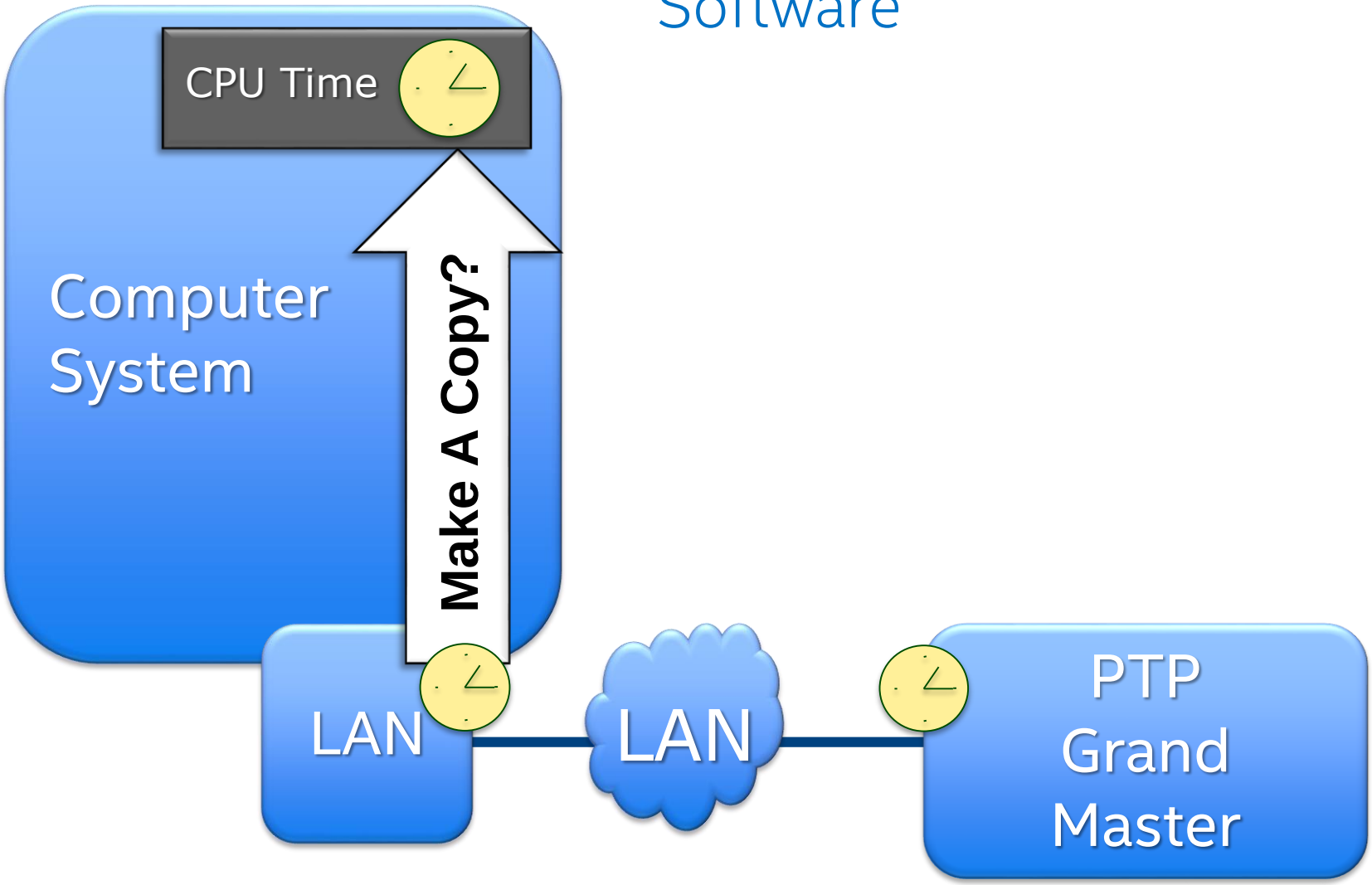
Modern Computer Systems have Multiple Internal Timebases—That Won't (and Need Not) Change

Application Software separated from Network Time by a large chasm



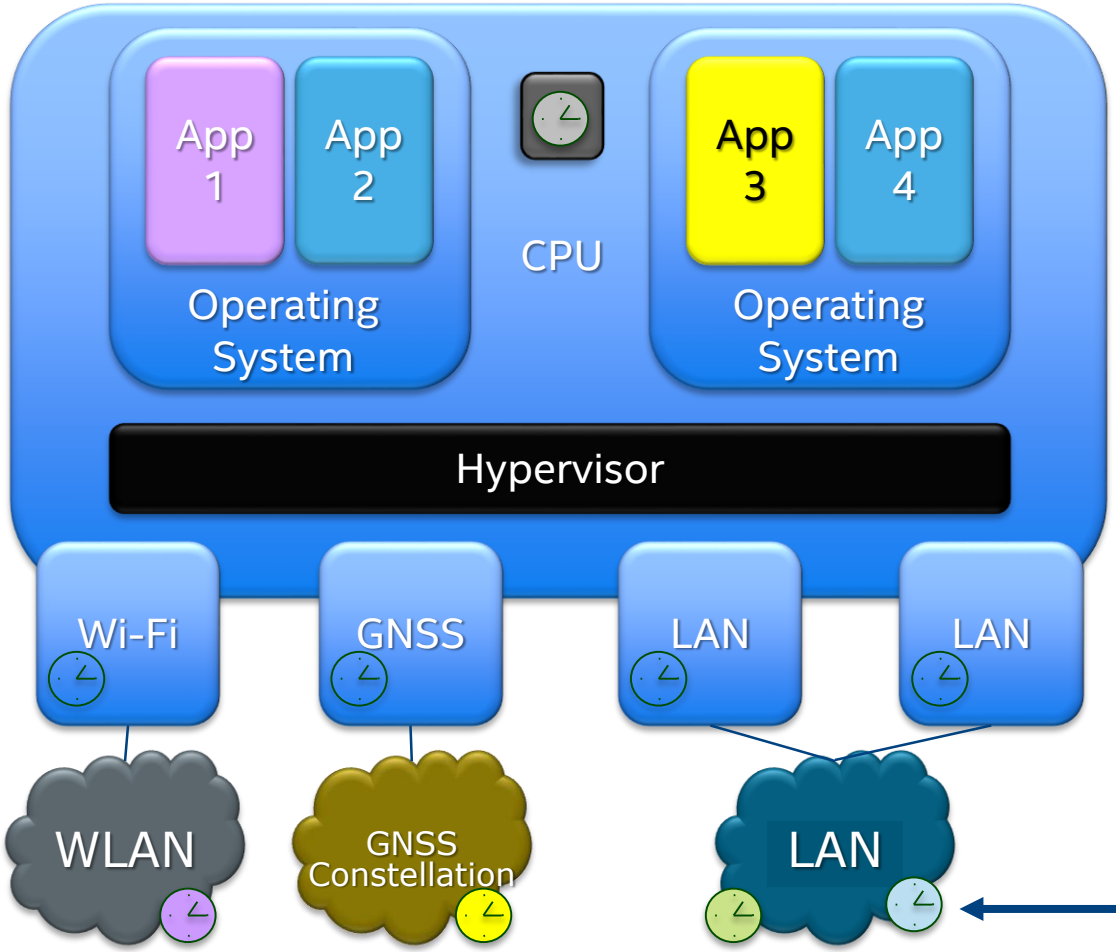
For software, “NOW” is never really Now
Reading Network Time can be VERY SLOW

One Approach for Bringing Network Time Near to the Software



Replicating Network Time in the CPU Hardware Doesn't Scale—See next slide

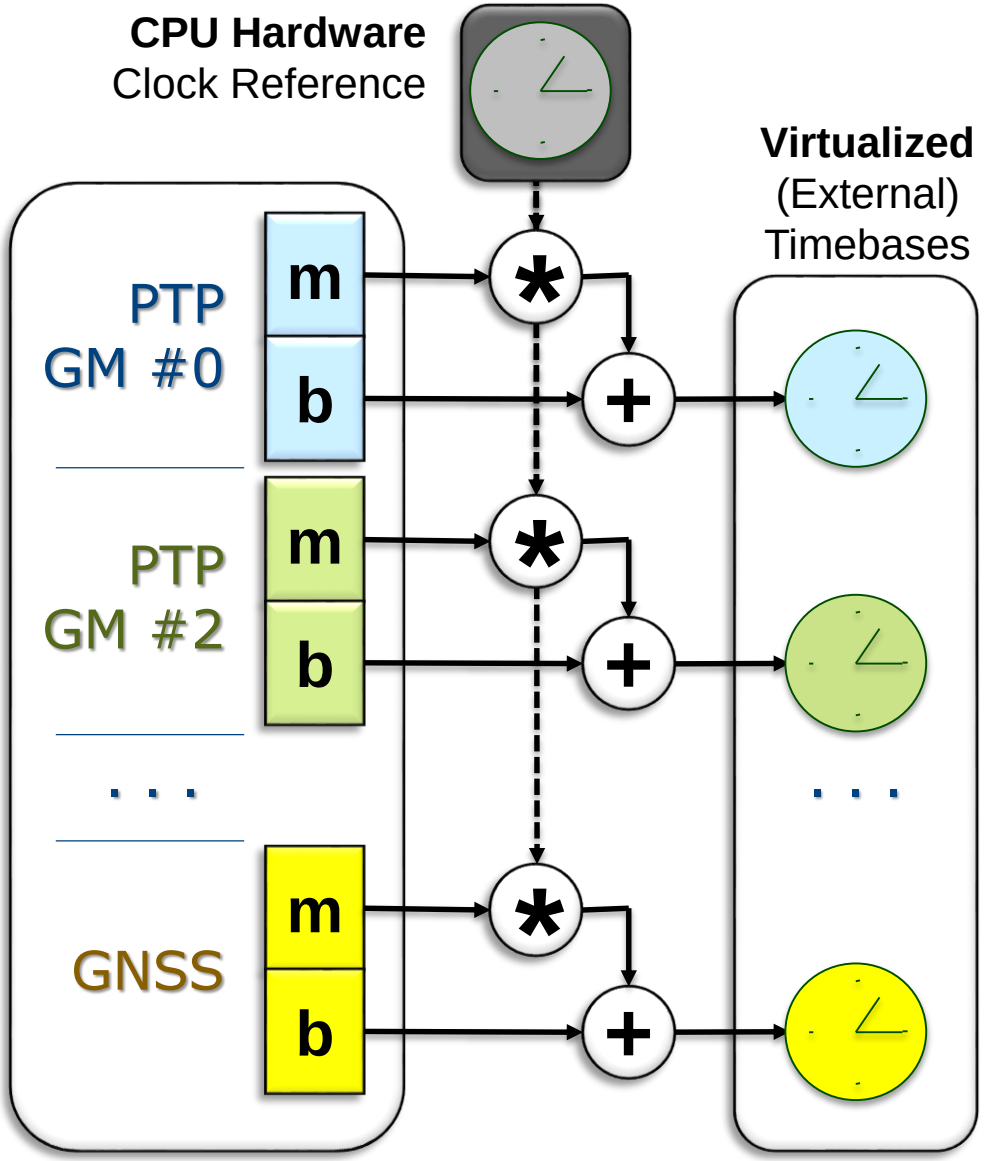
Modern Computer Systems Aren't So Simple



The Reality:

- There are Multiple Network Time Sources
- Some Applications Track Multiple Time Sources Simultaneously
- Multiple Virtual Machines / Operating Systems

Multiple Hardware Times in the CPU Doesn't Scale



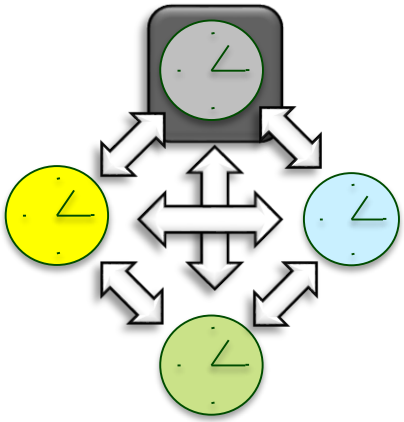
Scalable Timebase Representation

Here's what's needed:

1. A Stable HW Reference
2. Instantaneous * and + Ops
3. Exact measure of m and b

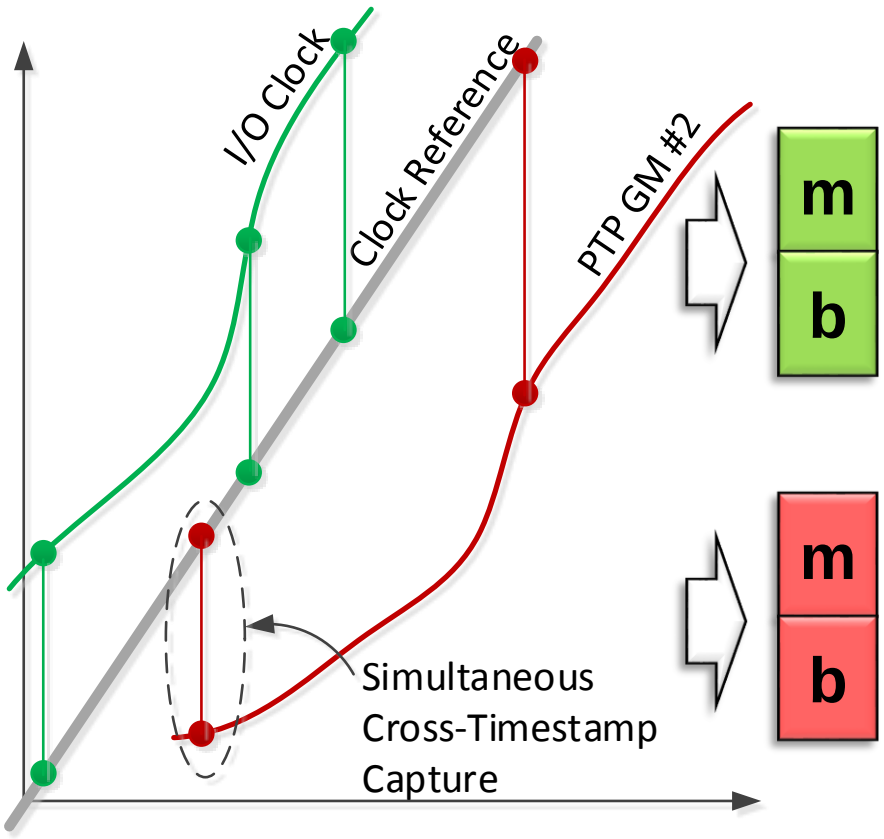
Any Timebase ←→ Any Timebase

[I wish I had time to talk about POSIX Clocks]



Given $[(m_0,b_0), (m_1,b_1), \ldots]$, Can Convert CPU time into any External Clock, and Vice Versa

Measuring m and b



Cross-Timestamp pairs of clocks

1. System Clock Reference, and
2. The Other Clock

→The goal is Simultaneity

Software Cross-Timestamping example: The Linux IOCTL: PTP_SYS_OFFSET

Should be done with tight sequence of instructions (with interrupts disabled)

However:

- PTP_SYS_OFFSET does not
- The System Clock that's referenced is CLOCK_REALTIME, which is a software-disciplined timebase

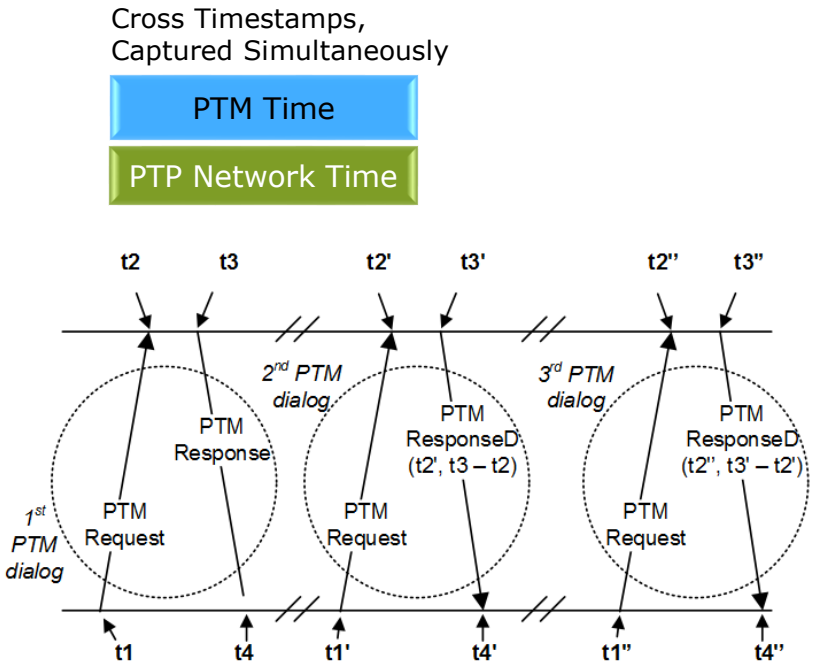
Proposed Enhancements:
- Capture the pair in the LAN driver
- Together with CLOCK_MONOTONIC_RAW

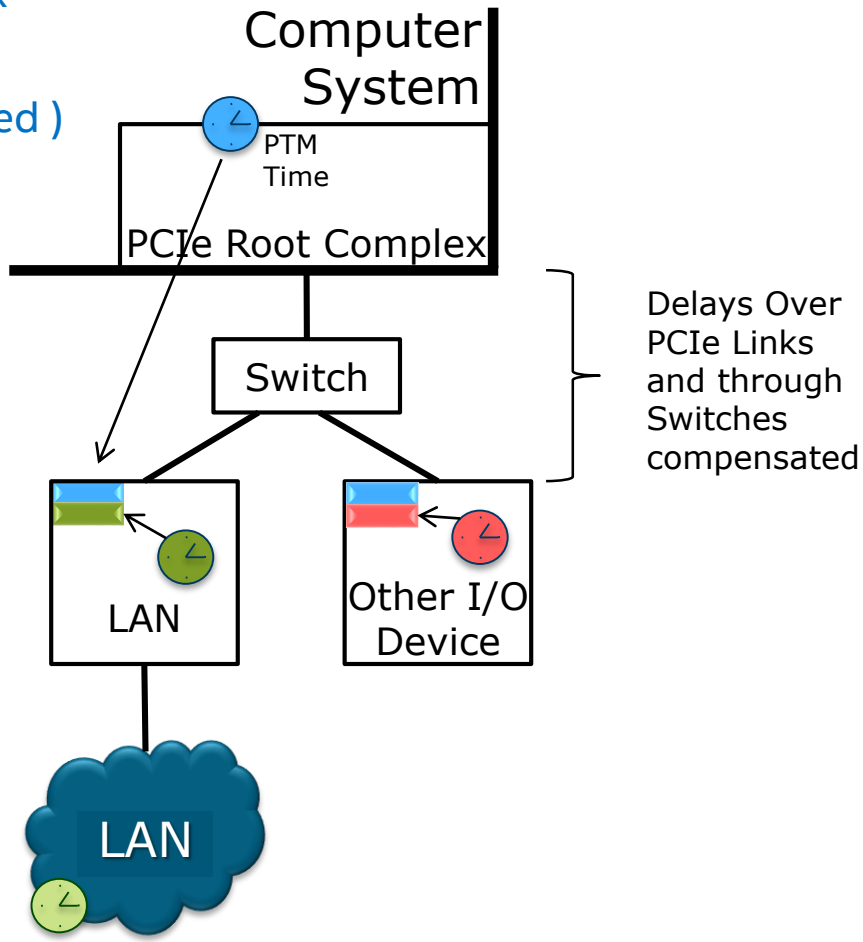
Hardware Representation of Multiple Clocks In the CPU Doesn't Scale
Cross-Timestamping DOES Scale.

Hardware Measurement with PCIe PTM
(Precision Time Measurement)

Scenario:

1. Device initiates *PTM Request* to Root Complex (trigger is device-specific)
2. PTM Time is Returned (delays are compensated)
3. (PTM Time, PTP Time) Written to Memory



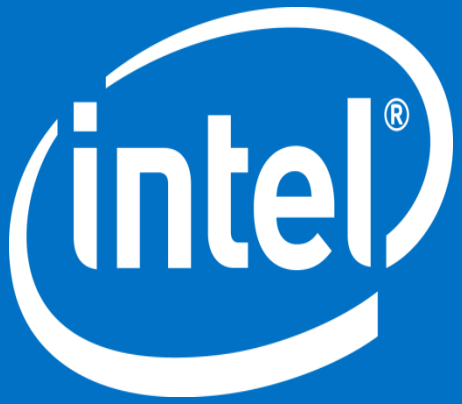


PCIe PTM Passes Time *In-Band* Over PCIe [Including Switches]

Summary

- Modern Computer Systems have Multiple *Internal* Timebases
- For Software, “Now” is never Now, Network Time is Far Away
- Replicating Network Time in CPU Hardware Doesn’t Scale
- Given [(m0,b0), (m1,b1), …], Software Can Convert CPU Time into the Relevant External Timebase, and Vice Versa
- Cross-Timestamping Scales—Closer to the Hardware is Better
- An Example: PCIe PTM Passes Time In-Band Over PCIe [Including Switches]

I welcome further discussion: Kevin dot B dot Stanton at Intel


intel®