November 18, 2020

Computer Timekeeping (and Synchronization)

KEVIN STANTON, PH.D.
SR. PRINCIPAL ENGINEER, INTEL CORPORATION

Time-Sync Singularity

- What if my local notion of "now" could be used to unambiguously order events

  ➔The proof of causality

- ***Temporal Inflection Point***

- Note: Much of this presentation was developed for a Keynote at ISPCS 2019, in Portland Oregon (International Symposium on Precision Clock Synchronization)
- This was in some ways a continuation of the theme of "The Last Inch Problem" (Eidson and Stanton at ISPCS, 2015)

A Simple Question: “Is it Now, Now?”

```
clock_gettime(CLOCK_REALTIME, &now);
```

```
// What does now mean?


// With respect to whom?

// With what accuracy?

// With what level of trust?

// When actually was now?
```

The above function returns a Number with units of ns.
Future Applications need to know more!

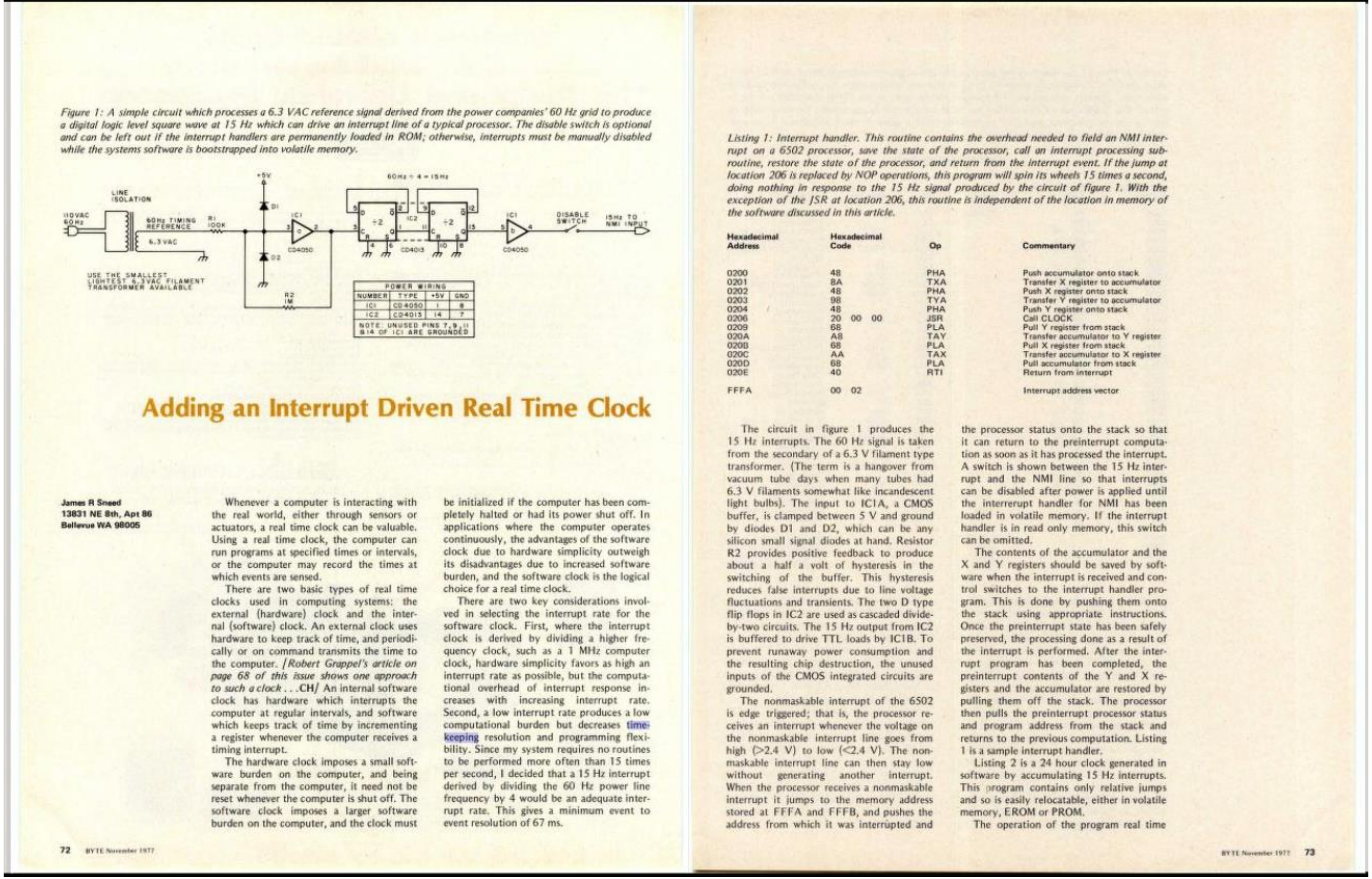
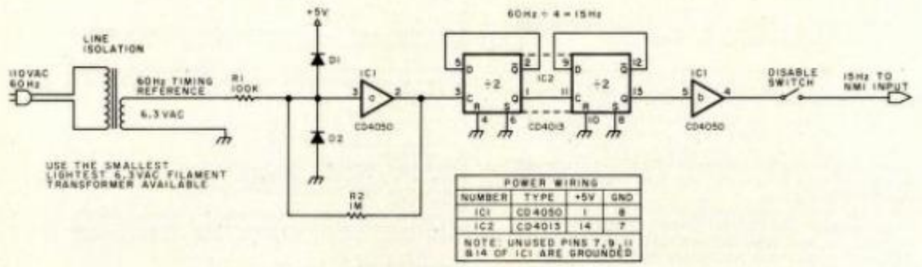


Figure 1: A simple circuit which processes a 6.3 VAC reference signal derived from the power companies' 60 Hz grid to produce a digital logic level square wave at 15 Hz which can drive an interrupt line of a typical processor. The disable switch is optional and can be left out if the interrupt handlers are permanently loaded in ROM; otherwise, interrupts must be manually disabled while the systems software is bootstrapped into volatile memory.



POWER WIRING			
NUMBER	TYPE	+5V	GND
IC1	CD4050	1	8
IC2	CD4013	14	7
NOTE: UNUSED PINS 7, 9, 11 & 14 OF IC1 ARE GROUNDED			

Adding an Interrupt Driven Real Time Clock

James R Sneed
13831 NE 8th, Apt 86
Bellevue WA 98005

Whenever a computer is interacting with the real world, either through sensors or actuators, a real time clock can be valuable. Using a real time clock, the computer can run programs at specified times or intervals, or the computer may record the times at which events are sensed.

There are two basic types of real time clocks used in computing systems: the external (hardware) clock and the internal (software) clock. An external clock uses hardware to keep track of time, and periodically or on command transmits the time to the computer. *[Robert Grappel's article on page 68 of this issue shows one approach to such a clock . . . CH]* An internal software clock has hardware which interrupts the computer at regular intervals, and software which keeps track of time by incrementing a register whenever the computer receives a timing interrupt.

The hardware clock imposes a small software burden on the computer, and being separate from the computer, it need not be reset whenever the computer is shut off. The software clock imposes a larger software burden on the computer, and the clock must be initialized if the computer has been completely halted or had its power shut off. In applications where the computer operates continuously, the advantages of the software clock due to hardware simplicity outweigh its disadvantages due to increased software burden, and the software clock is the logical choice for a real time clock.

There are two key considerations involved in selecting the interrupt rate for the software clock. First, where the interrupt clock is derived by dividing a higher frequency clock, such as a 1 MHz computer clock, hardware simplicity favors as high an interrupt rate as possible, but the computational overhead of interrupt response increases with increasing interrupt rate. Second, a low interrupt rate produces a low computational burden but decreases timekeeping resolution and programming flexibility. Since my system requires no routines to be performed more often than 15 times per second, I decided that a 15 Hz interrupt derived by dividing the 60 Hz power line frequency by 4 would be an adequate interrupt rate. This gives a minimum event to event resolution of 67 ms.

Listing 1: Interrupt handler. This routine contains the overhead needed to field an NMI interrupt on a 6502 processor, save the state of the processor, call an interrupt processing subroutine, restore the state of the processor, and return from the interrupt event. If the jump at location 206 is replaced by NOP operations, this program will spin its wheels 15 times a second, doing nothing in response to the 15 Hz signal produced by the circuit of figure 1. With the exception of the JSR at location 206, this routine is independent of the location in memory of the software discussed in this article.

```
Hexadecimal   Hexadecimal
Address       Code          Op     Commentary

0200          48            PHA    Push accumulator onto stack
0201          8A            TXA    Transfer X register to accumulator
0202          48            PHA    Push X register onto stack
0203          98            TYA    Transfer Y register to accumulator
0204          48            PHA    Push Y register onto stack
0206          20  00  00    JSR    Call CLOCK
0209          68            PLA    Pull Y register from stack
020A          A8            TAY    Transfer accumulator to Y register
020B          68            PLA    Pull X register from stack
020C          AA            TAX    Transfer accumulator to X register
020D          68            PLA    Pull accumulator from stack
020E          40            RTI    Return from interrupt

FFFA          00  02               Interrupt address vector
```

The circuit in figure 1 produces the 15 Hz interrupts. The 60 Hz signal is taken from the secondary of a 6.3 V filament type transformer. (The term is a hangover from vacuum tube days when many tubes had 6.3 V filaments somewhat like incandescent light bulbs). The input to IC1A, a CMOS buffer, is clamped between 5 V and ground by diodes D1 and D2, which can be any silicon small signal diodes at hand. Resistor R2 provides positive feedback to produce about a half a volt of hysteresis in the switching of the buffer. This hysteresis reduces false interrupts due to line voltage fluctuations and transients. The two D type flip flops in IC2 are used as cascaded divide-by-two circuits. The 15 Hz output from IC2 is buffered to drive TTL loads by IC1B. To prevent runaway power consumption and the resulting chip destruction, the unused inputs of the CMOS integrated circuits are grounded.

The nonmaskable interrupt of the 6502 is edge triggered; that is, the processor receives an interrupt whenever the voltage on the nonmaskable interrupt line goes from high (>2.4 V) to low (<2.4 V). The nonmaskable interrupt line can then stay low without generating another interrupt. When the processor receives a nonmaskable interrupt it jumps to the memory address stored at FFFA and FFFB, and pushes the address from which it was interrupted and the processor status onto the stack so that it can return to the preinterrupt computation as soon as it has processed the interrupt. A switch is shown between the 15 Hz interrupt and the NMI line so that interrupts can be disabled after power is applied until the interrerupt handler for NMI has been loaded in volatile memory. If the interrupt handler is in read only memory, this switch can be omitted.

The contents of the accumulator and the X and Y registers should be saved by software when the interrupt is received and control switches to the interrupt handler program. This is done by pushing them onto the stack using appropriate instructions. Once the preinterrupt state has been safely preserved, the processing done as a result of the interrupt is performed. After the interrupt program has been completed, the preinterrupt contents of the Y and X registers and the accumulator are restored by pulling them off the stack. The processor then pulls the preinterrupt processor status and program address from the stack and returns to the previous computation. Listing 1 is a sample interrupt handler.

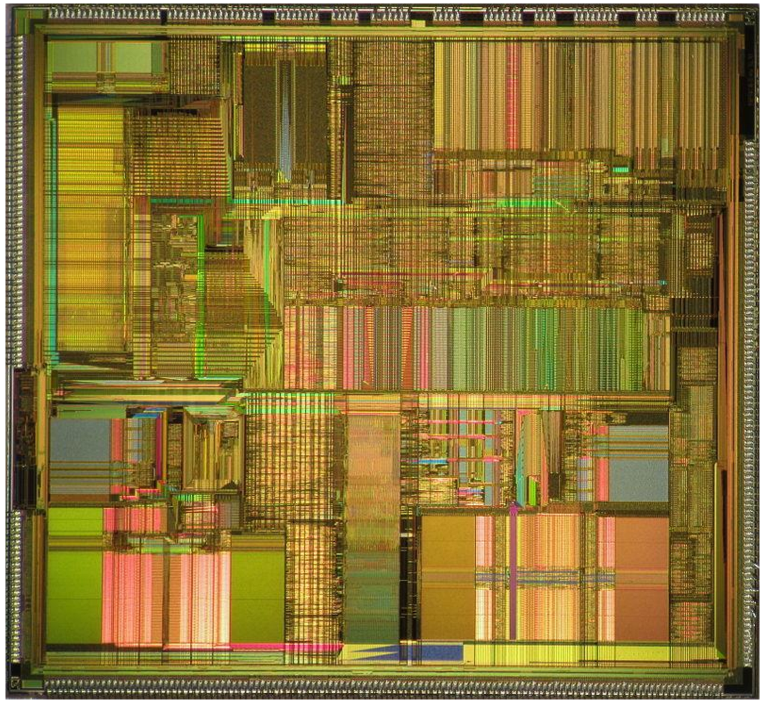
Listing 2 is a 24 hour clock generated in software by accumulating 15 Hz interrupts. This program contains only relative jumps and so is easily relocatable, either in volatile memory, EROM or PROM.

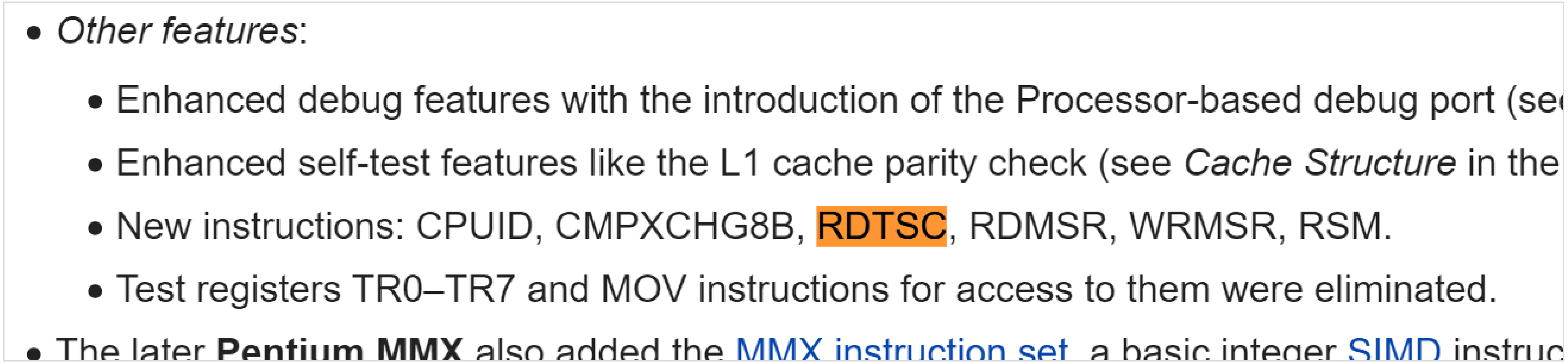
The operation of the program real time

Https://archive.org/details/byte-magazine-1977-11/page/n49?q=timekeeping

Time

Introducing, a Timekeeping INSTRUCTION!

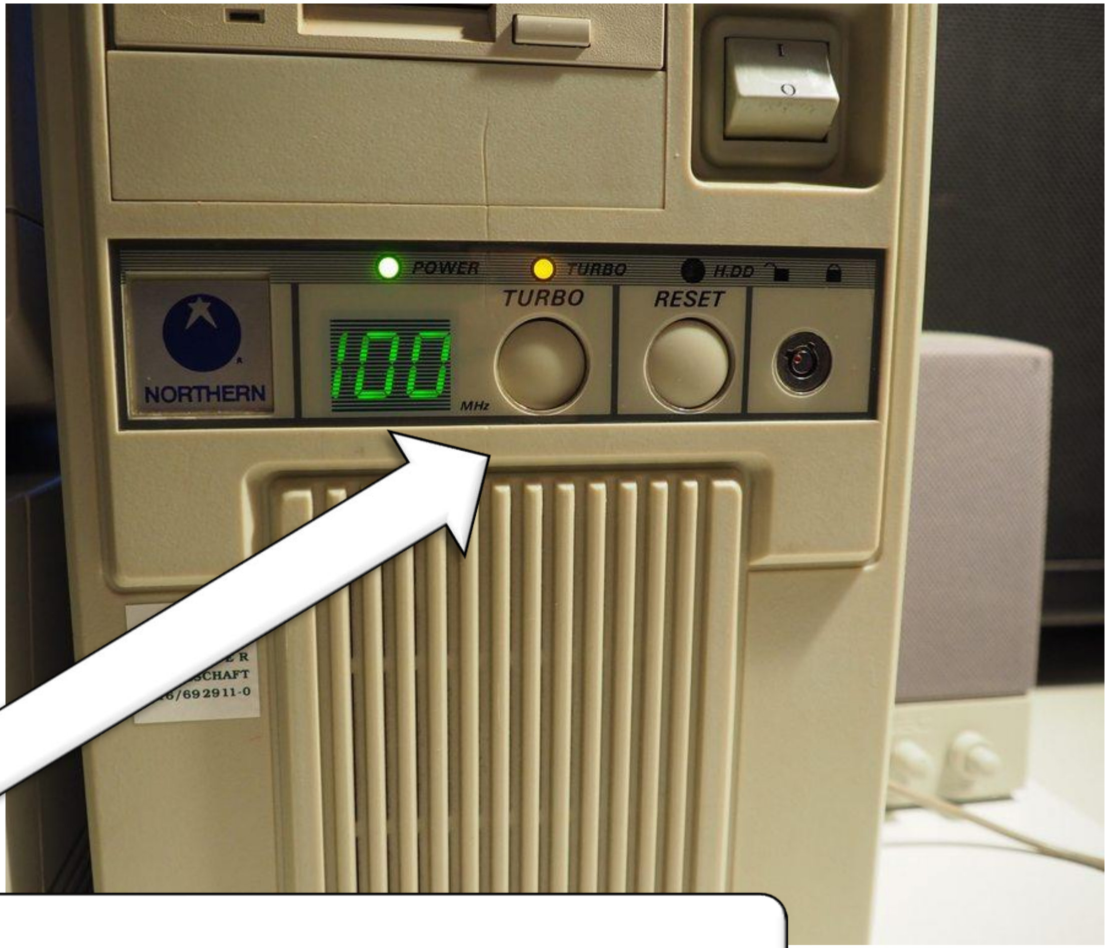




- *Other features*:
  - Enhanced debug features with the introduction of the Processor-based debug port (se
  - Enhanced self-test features like the L1 cache parity check (see *Cache Structure* in the
  - New instructions: CPUID, CMPXCHG8B, RDTSC, RDMSR, WRMSR, RSM.
  - Test registers TR0–TR7 and MOV instructions for access to them were eliminated.
- The later **Pentium MMX** also added the MMX instruction set, a basic integer SIMD instruc

https://en.wikipedia.org/wiki/P5_(microarchitecture)

Games Tuned Timing to the CPU Performance



The "Turbo Button"

Time Dilation to un-"Slow Down the World"

Battery Powered: Low-Power CPU Features +Turbo

CPU Frequency Modulated with Demand

Package Pwr: Current package power consumption

Package Pwr: Current package power consumption

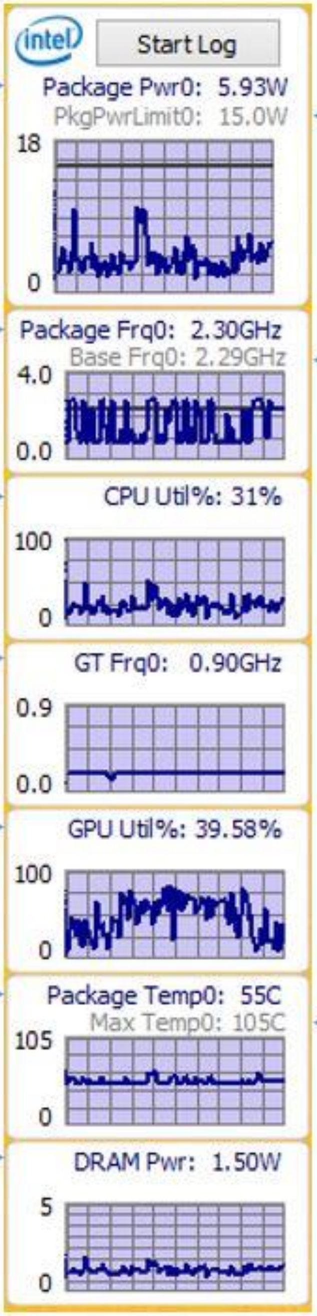
CPU Util: Current CPU utilization percent

GT Freq: Current GPU frequency

GPU Util: Current GPU utilization percent

Package Temp: Current package temperature

DRAM Pwr: Current DRAM power consumption



Start Log
Package Pwr0: 5.93W
PkgPwrLimit0: 15.0W
Package Frq0: 2.30GHz
Base Frq0: 2.29GHz
CPU Util%: 31%
GT Frq0: 0.90GHz
GPU Util%: 39.58%
Package Temp0: 55C
Max Temp0: 105C
DRAM Pwr: 1.50W

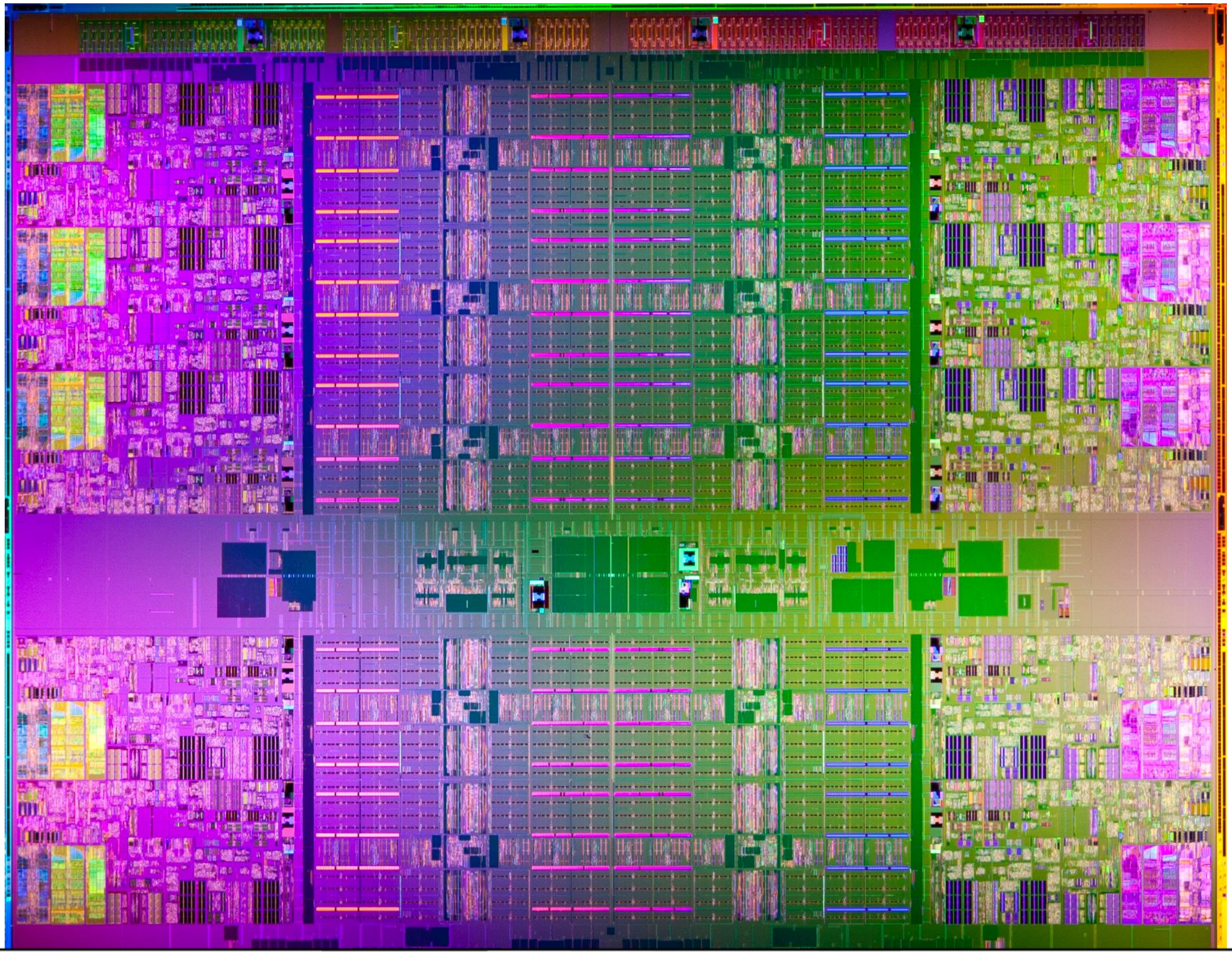
PkgPwrLimit: Current average package power limit

Base Freq: Graded CPU frequency

Max Temp: Maximum package temperature

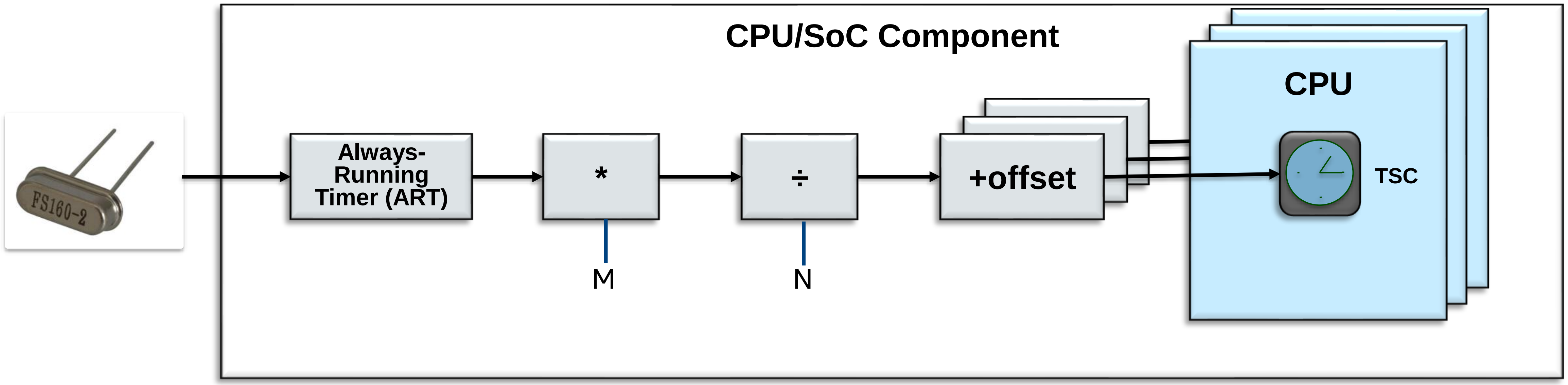
https://software.intel.com/en-us/articles/intel-power-gadget

Multi-Core CPUs



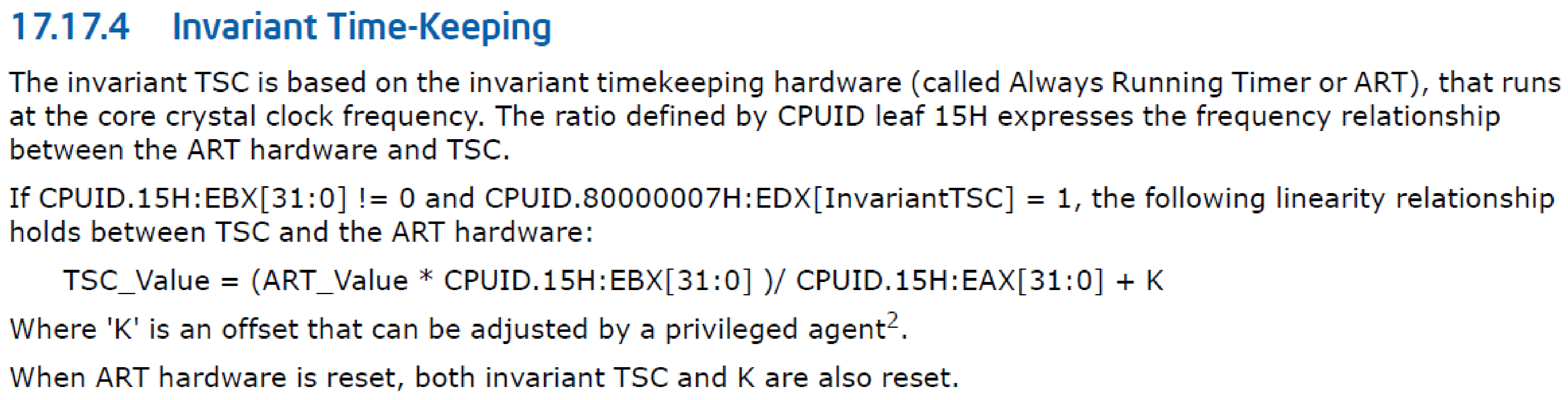
Time Synchronization Required WITHIN the CPU Complex

Oscillator → Local CPU Time (e.g. TSC)



TSC as a Function of Oscillator Frequency

Some Representative Definitions



17.17.4 Invariant Time-Keeping

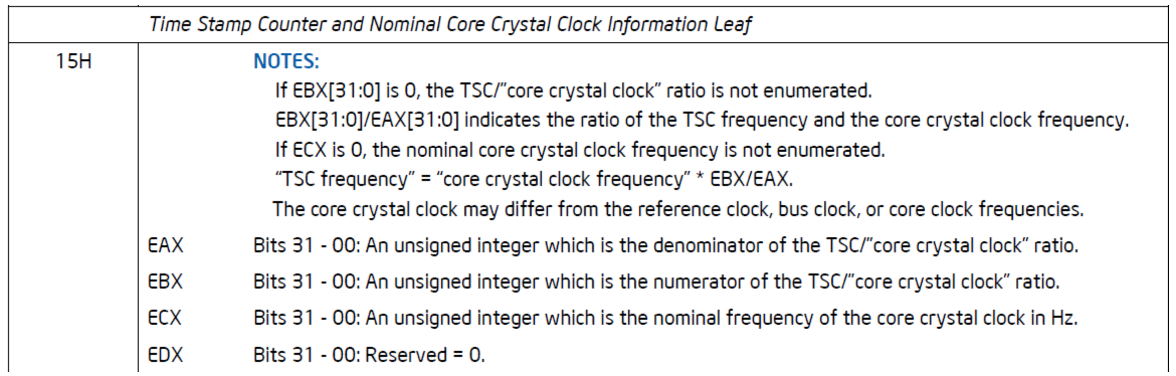
The invariant TSC is based on the invariant timekeeping hardware (called Always Running Timer or ART), that runs at the core crystal clock frequency. The ratio defined by CPUID leaf 15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX[InvariantTSC] = 1, the following linearity relationship holds between TSC and the ART hardware:

TSC_Value = (ART_Value * CPUID.15H:EBX[31:0])/ CPUID.15H:EAX[31:0] + K

Where 'K' is an offset that can be adjusted by a privileged agent[2].

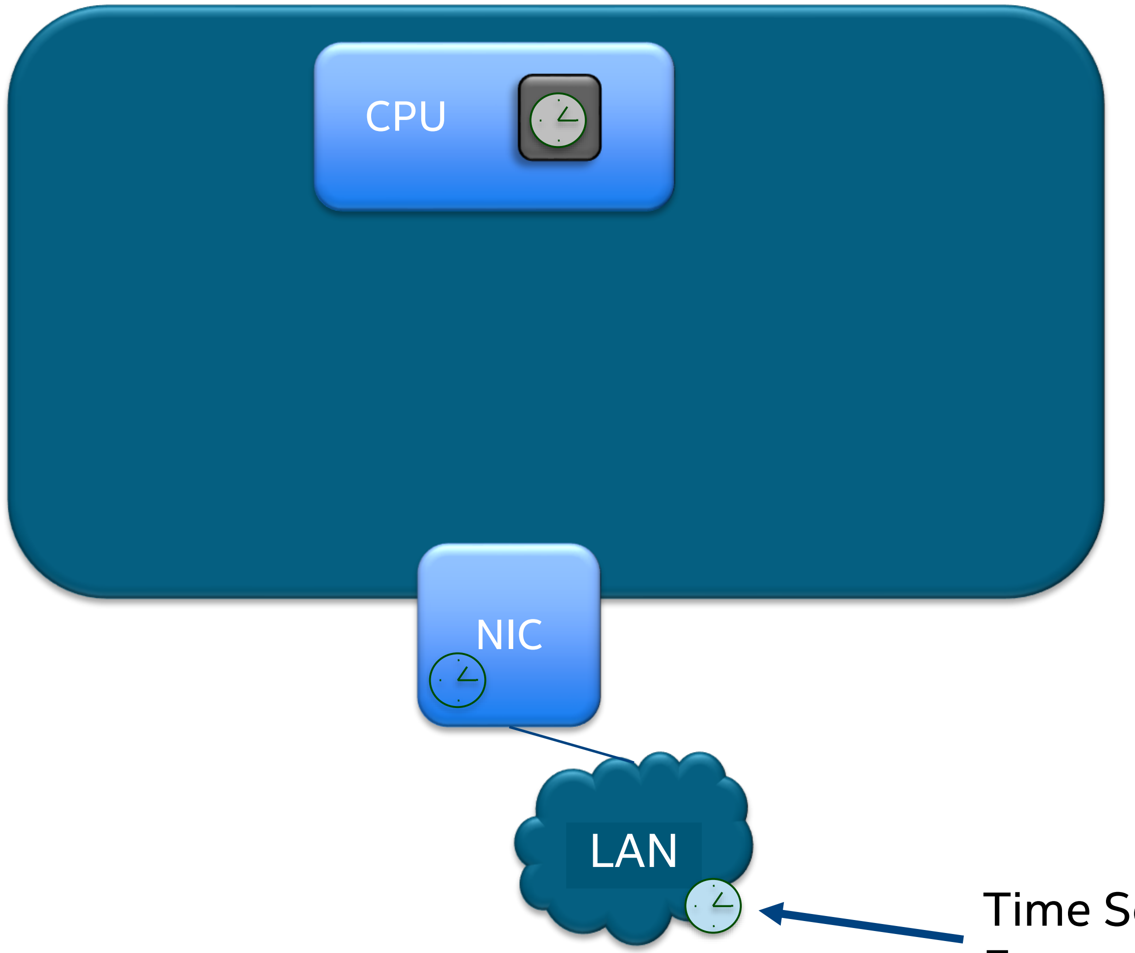
When ART hardware is reset, both invariant TSC and K are also reset.



	Time Stamp Counter and Nominal Core Crystal Clock Information Leaf	
15H		**NOTES:** If EBX[31:0] is 0, the TSC/"core crystal clock" ratio is not enumerated. EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency. If ECX is 0, the nominal core crystal clock frequency is not enumerated. "TSC frequency" = "core crystal clock frequency" * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	EAX	Bits 31 - 00: An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31 - 00: An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.
	ECX	Bits 31 - 00: An unsigned integer which is the nominal frequency of the core crystal clock in Hz.
	EDX	Bits 31 - 00: Reserved = 0.

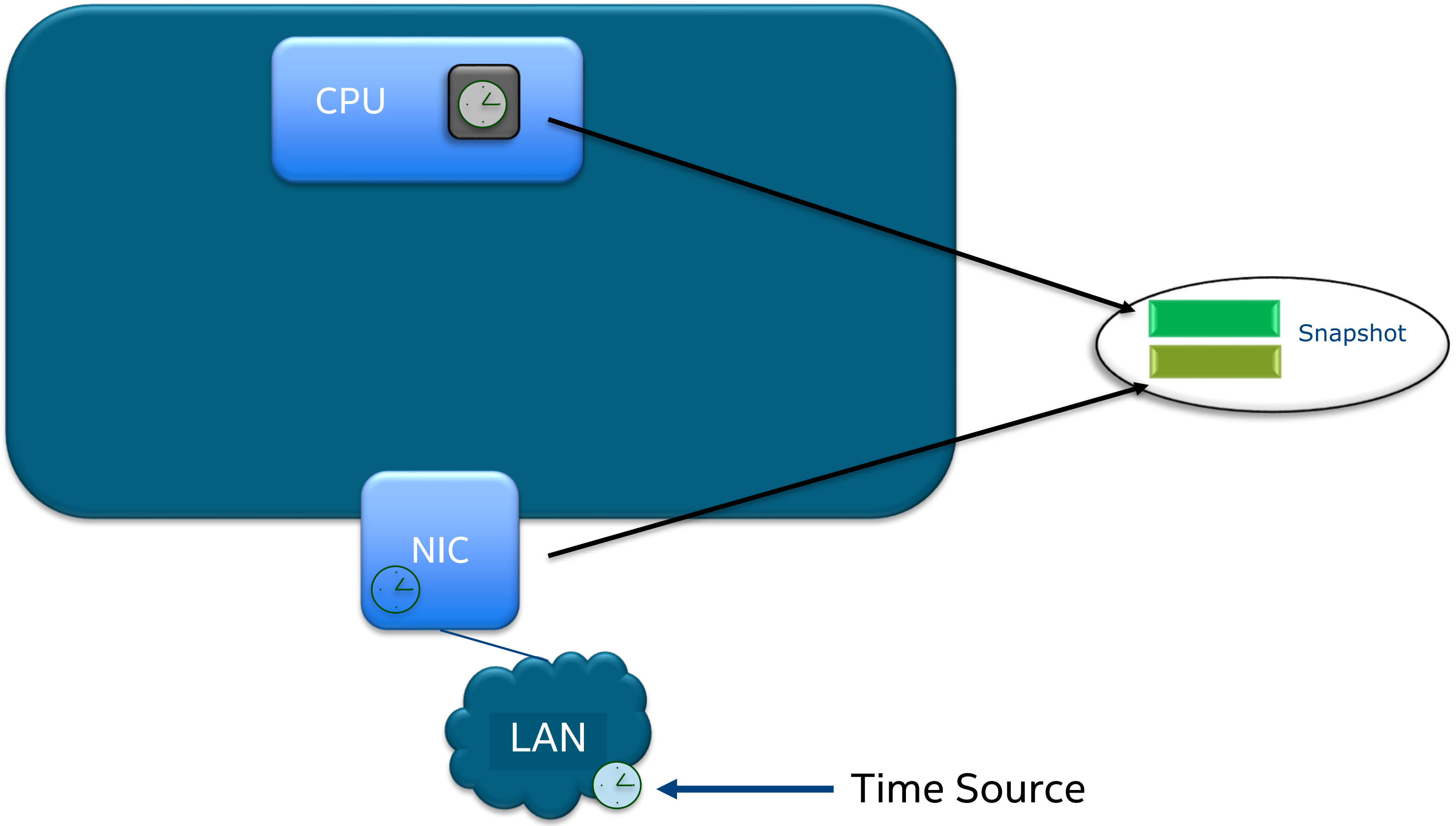
https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html

Synchronized Time comes From The Outside



Time Source
E.g. (one of many profiles of) PTP/1588

“NIC” Time is FAR from Software



Time Source

CPU Time → Synchronized Time

Time "now"

```
(1) clock_gettime(CLOCK_MONOTONIC_RAW, &now);
```

- Returns current TSC value scaled to nominal nanoseconds

```
(2) clock_gettime(CLOCK_MONOTONIC,     &now);
```

- Returns current TSC value scaled to track TAI, in nanoseconds

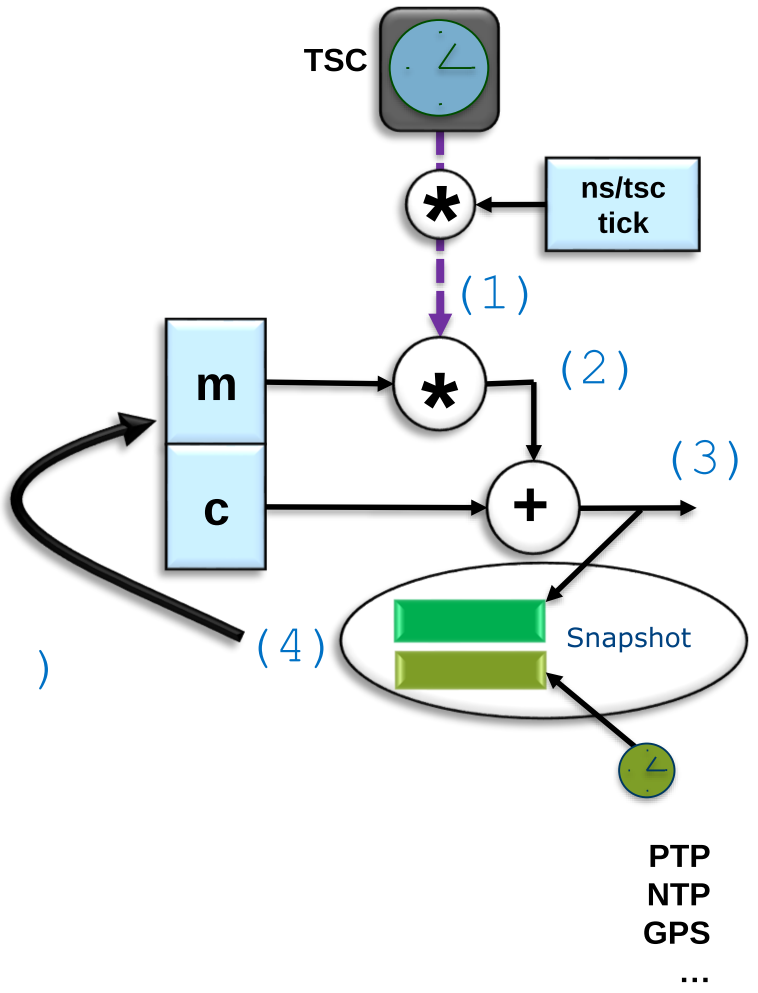
```
(3) clock_gettime(CLOCK_REALTIME,      &now);
```

- Returns CLOCK_MONOTONIC + (now-1/1/1970) [incl. leap seconds]

Cross-Timestamp

```
(4) ioctl(phc_fd,PTP_SYS_OFFSET[_PRECISE], &offset )
```

- returns the triple:
  - `eth_ptp_time; realtime; monotonic_raw`



POSIX: Piecewise-Linear Clock Model: y[n]=mx[n]+c

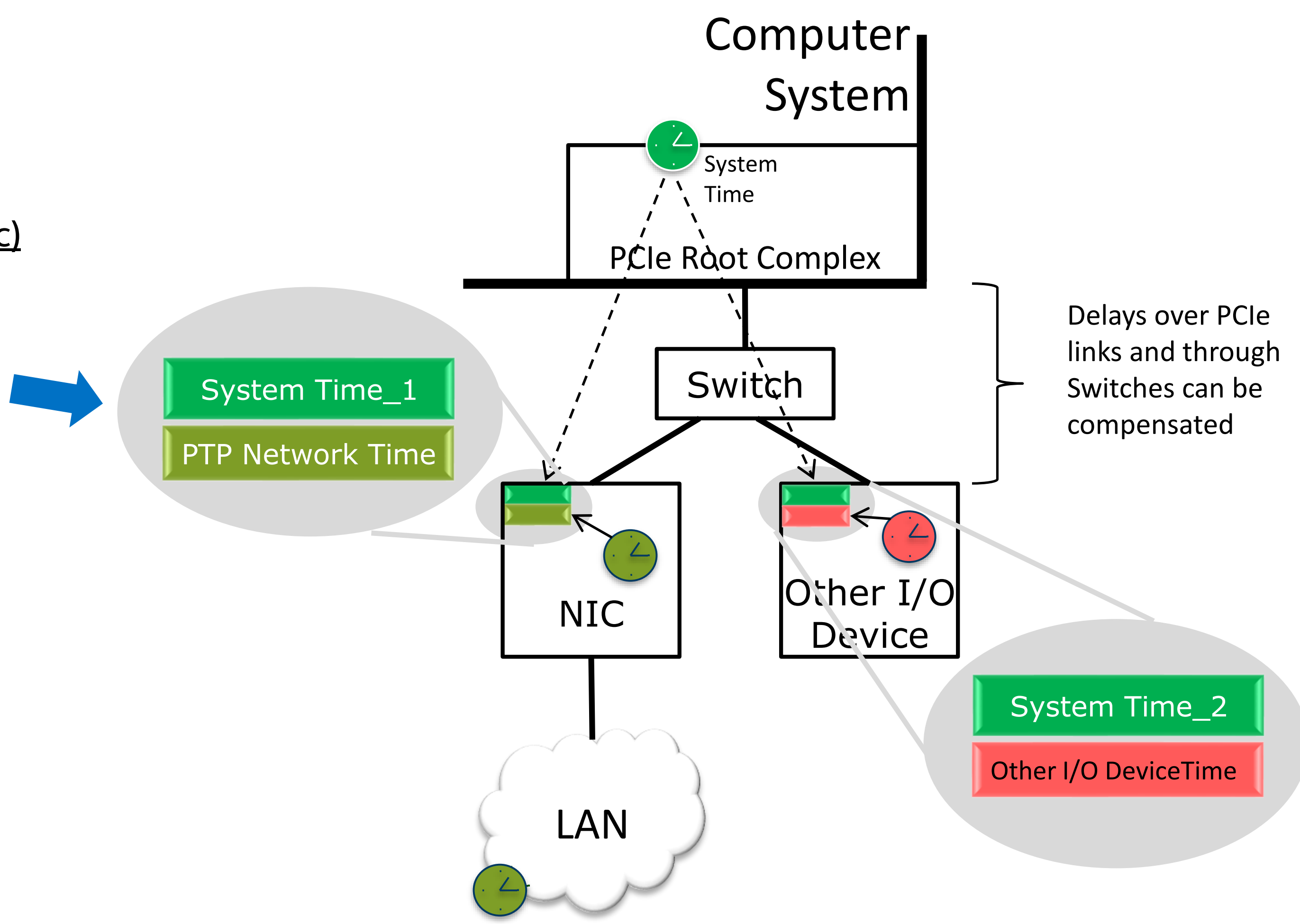
Using PCIe PTM to Cross-Timestamp

(PTM=Precision Time Measurement)

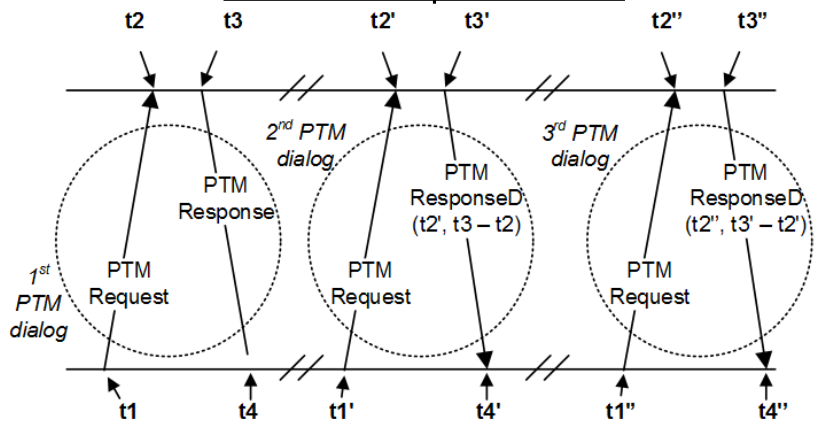
Sample Scenario:

1. Device Driver Triggers Cross-Timestamp
2. Device initiates *PTM Request* TLP to Root Complex
3. System Time is Returned (delays are compensated)
4. (PTM Time, PTP Time) returned to NIC Device Driver
5. Software "disciplines" Coefficients per clock: m (and c)

Cross Timestamps, Captured Simultaneously

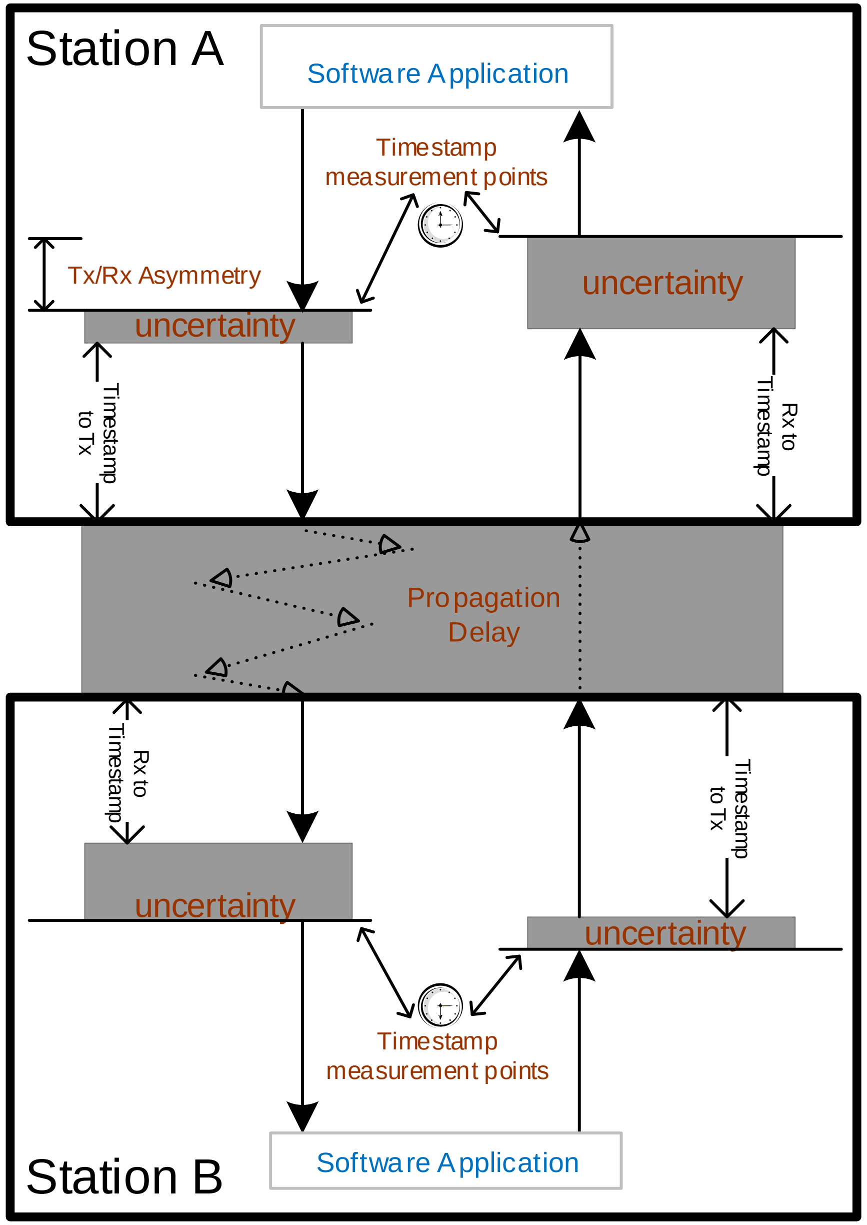


From PCIe Specification



Cross Timestamps ➔ 'm' and 'c' Coefficients

The effect of path asymmetry (using .11 TM as the example)



Hardware Timestamping require a *reference* for Rx and Tx timestamp measurements

Each system knows the delay between the PHY and where the actual Rx and Tx timestamps are captured

Each timestamp includes some uncertainty

The channel also introduces path asymmetry (and additional uncertainty)

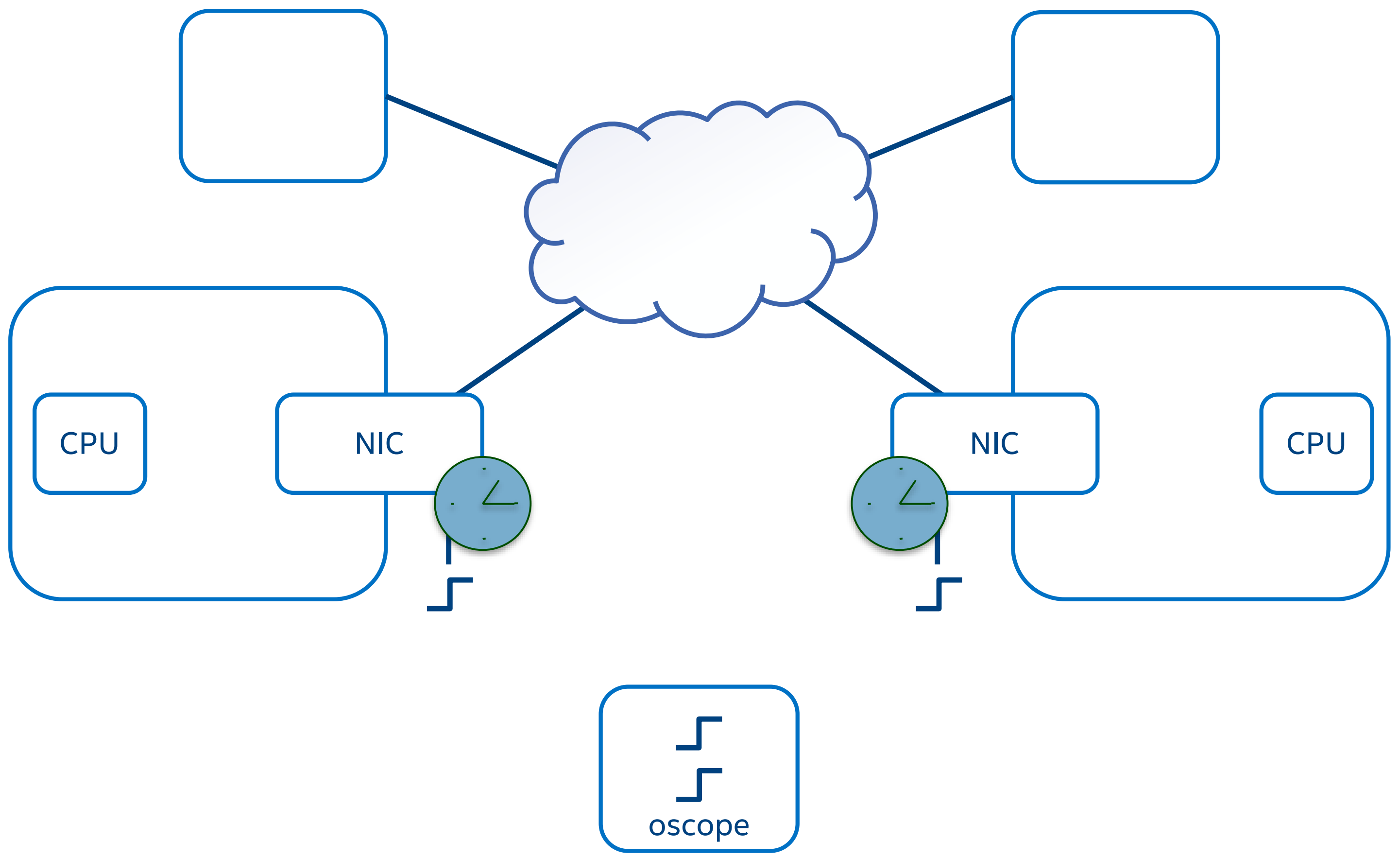
In practice, correction of the difference between the Rx offset and the Tx offset in each system is sufficient information

- The rest appears as fixed channel delay and channel uncertainty

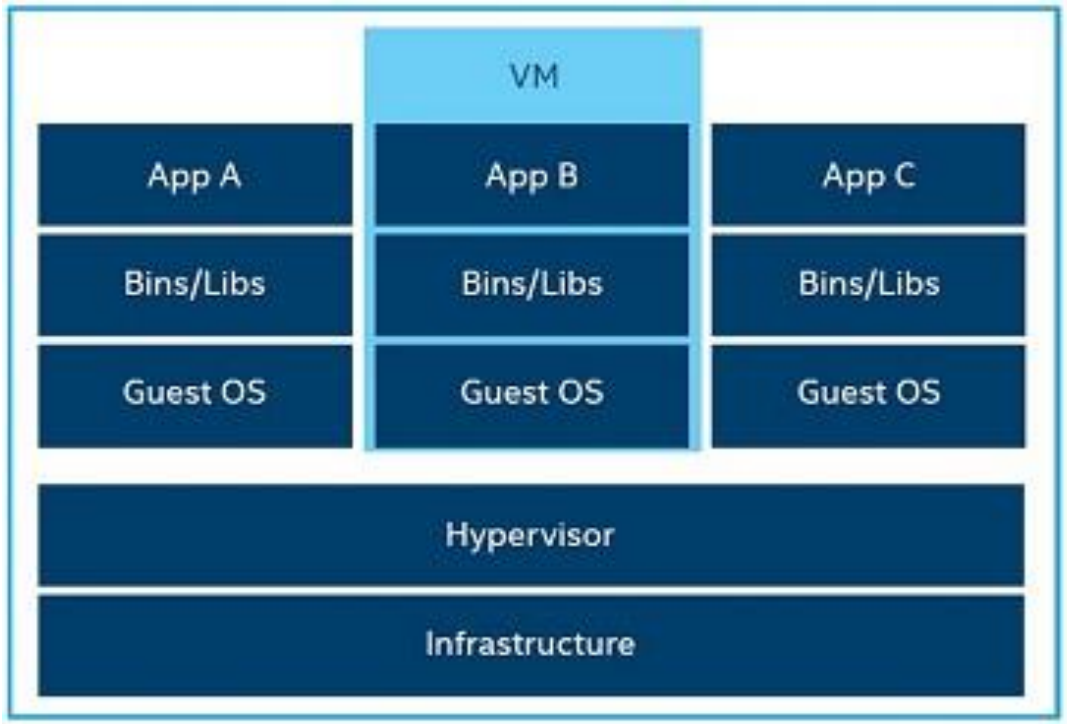
Instantaneous time error = Instantaneous delay asymmetry /2

Long-term average time error = average delay asymmetry

Measuring time-synchronization



Virtual Time?



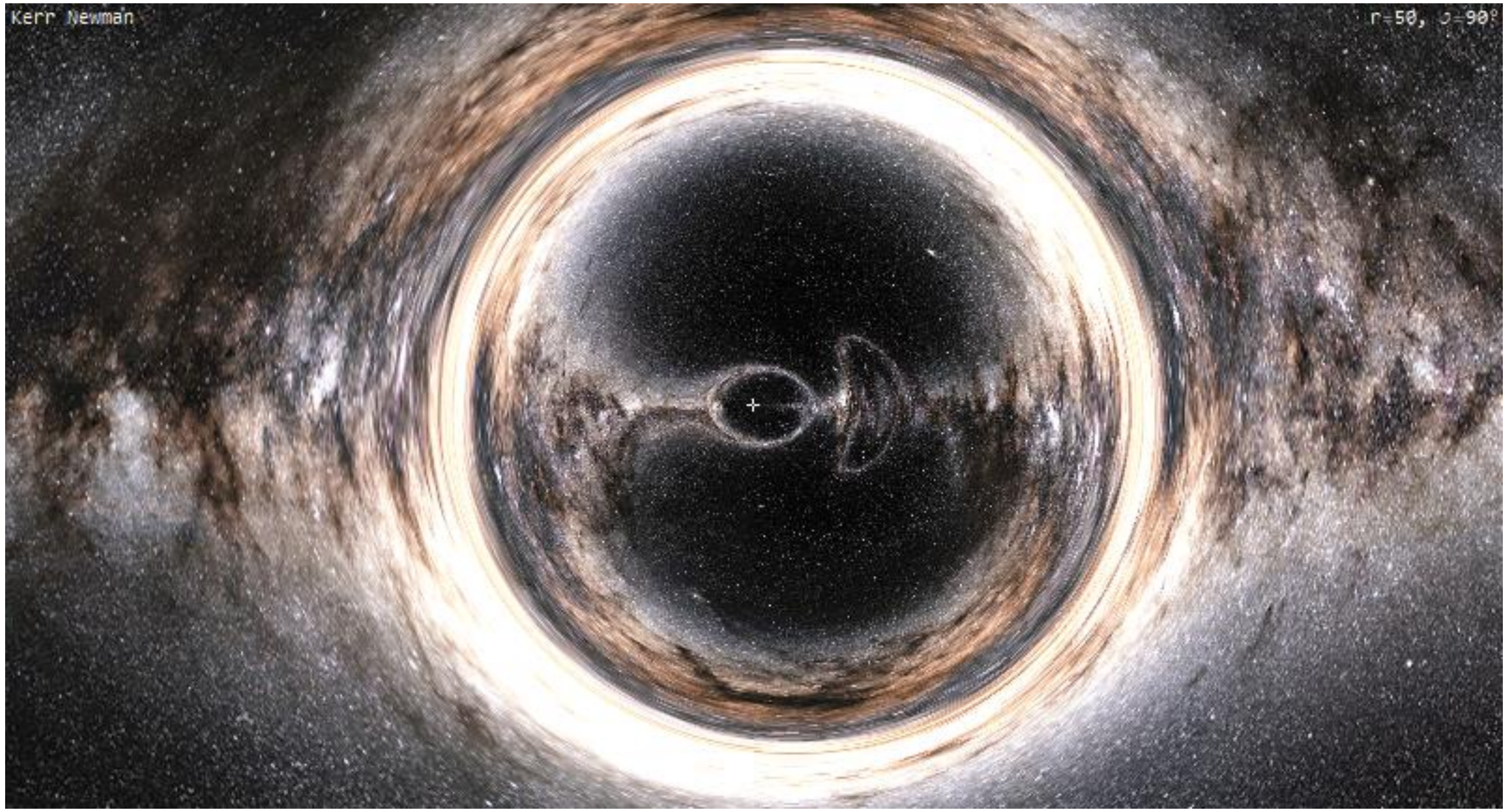
Should virtual machine provide virtual time or Physical Time?

Credit: Vikram D. https://software.intel.com/en-us/articles/workload-consolidation-in-industrial-iot

The Time-Accuracy Singularity

Trusted Time that's KNOWN to be BETTER than the BEST-CASE communication latency

- The limit of causality



Temporal Coordination Demands Dependable Accurate Time

https://www.youtube.com/watch?v=ufK2XRGUjuc

Summary / Call to Action

Software Timekeeping is Ready for the Next Big Challenge:

Help address the big challenges:

1. A plurality of "Nows" that I can TRUST
2. "Now" with QUANTIFIED WORST-CASE ACCURACY wrt Source
3. Reach THE SINGULARITY: Worst-case time accuracy that's better than the best-case network latency

So that Distributed Software knows what it can do with ***"Now"***