Time Uncertainty API

Maciek Machnikowski | netDev 0x18

Prior work

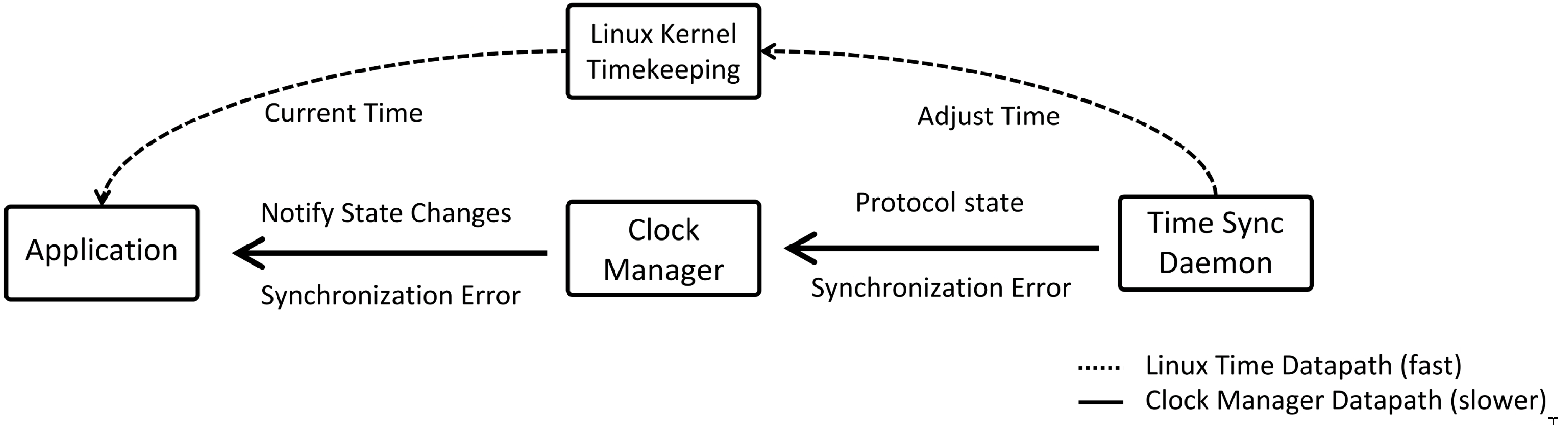
- SPEC, Precision Time API, Meta, Nvidia
- Clock Manager (Intel)
  - https://github.com/intel-staging/libptpmgmt_iaclocklib
- AWS Nitro Time Sync
  - https://github.com/aws/clock-bound
  - https://github.com/amzn/amzn-drivers/tree/master/kernel/linux/ena#PHC
- Google TrueTime
  - Used in Google Spanner
- https://chrony-project.org/

SPEC, Precision Time API, Meta, Nvidia

- Comprehensive API for Precision Time
- Replaces current Kernel APIs with functions that include precision
- Not implementable at Kernel level
  - Too comprehensive
  - Changes far too many APIs

Clock Manager (Intel)

https://github.com/intel-staging/libptpmgmt_iaclocklib
https://github.com/intel-staging/linux-ptp_iaclocklib



Clock bound (AWS)

https://github.com/aws/clock-bound

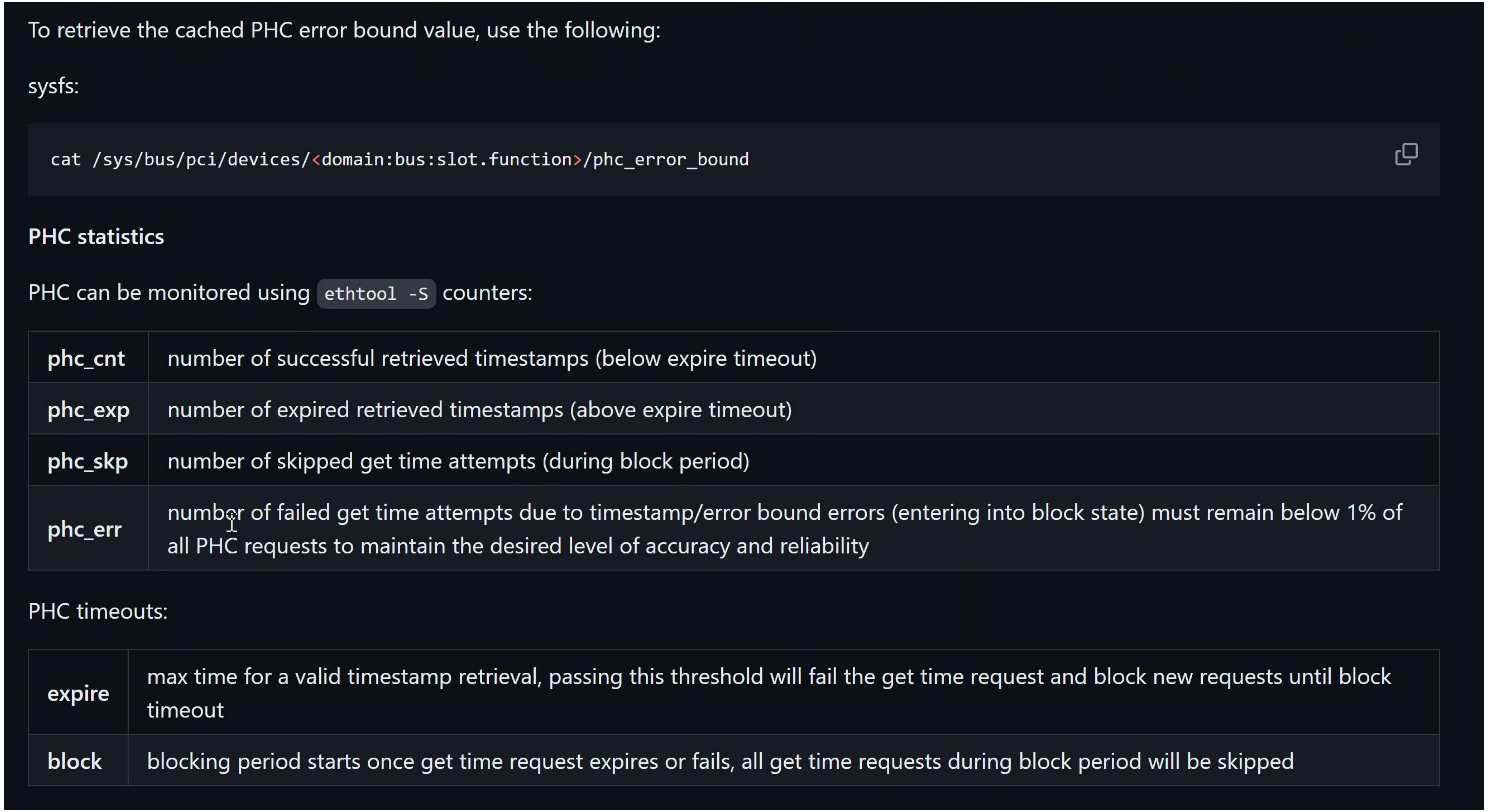
- Proxy to the clock
- Returns:
  - Earliest
  - Latest
  - Clock status

AWS Nitro Time Sync

https://github.com/amzn/amzn-drivers/tree/master/kernel/linux/ena#PHC

- Split in two parts
  - ClockBound Daemon
  - ENA Driver
- The driver exposes error bound via sysfs
  - cat /sys/bus/pci/devices/<domain:bus:slot.function>/phc_error_bound

ENA driver



To retrieve the cached PHC error bound value, use the following:

sysfs:

```
cat /sys/bus/pci/devices/<domain:bus:slot.function>/phc_error_bound
```

PHC statistics

PHC can be monitored using `ethtool -S` counters:

phc_cnt	number of successful retrieved timestamps (below expire timeout)
phc_exp	number of expired retrieved timestamps (above expire timeout)
phc_skp	number of skipped get time attempts (during block period)
phc_err	number of failed get time attempts due to timestamp/error bound errors (entering into block state) must remain below 1% of all PHC requests to maintain the desired level of accuracy and reliability

PHC timeouts:

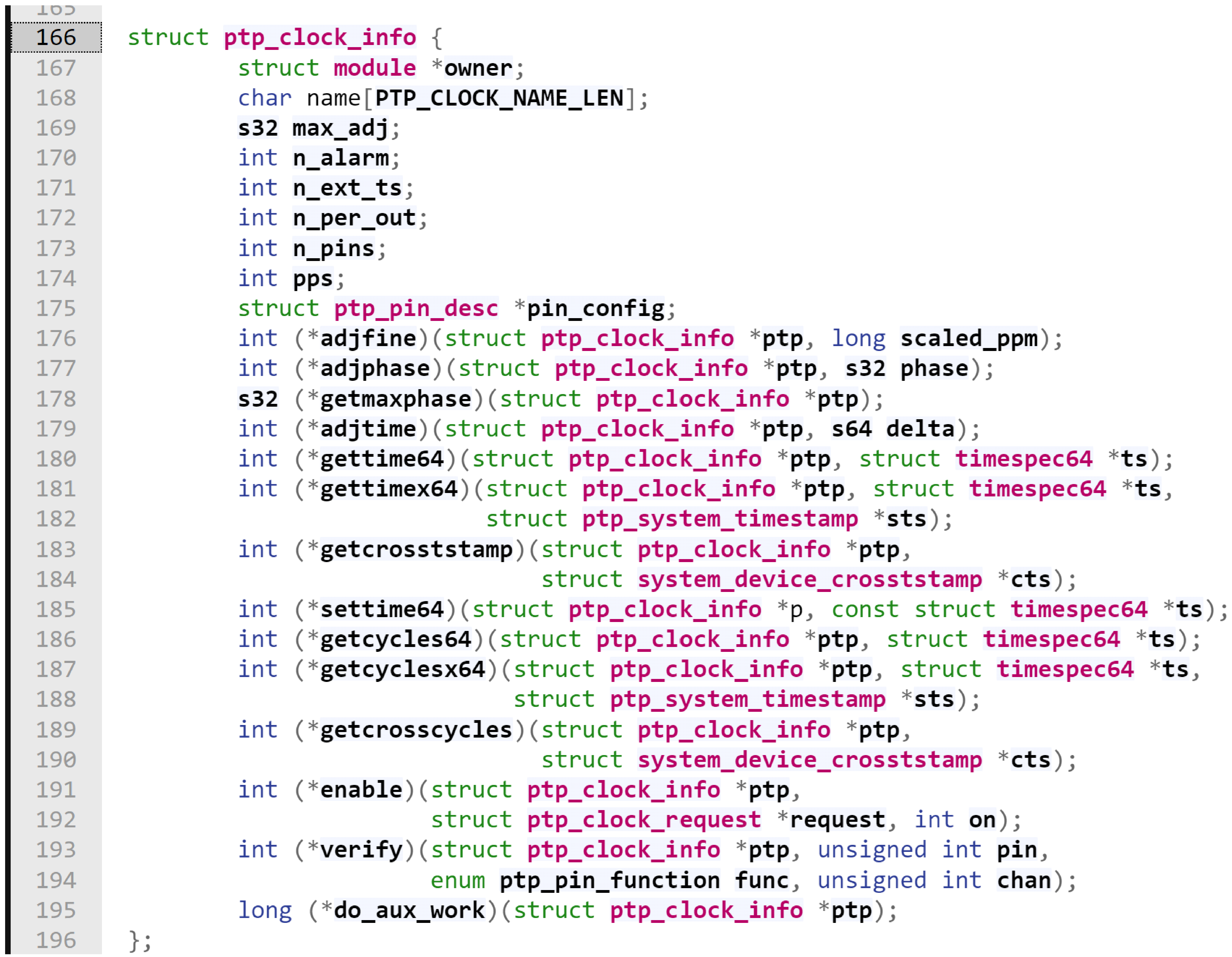
expire	max time for a valid timestamp retrieval, passing this threshold will fail the get time request and block new requests until block timeout
block	blocking period starts once get time request expires or fails, all get time requests during block period will be skipped

Non-goals

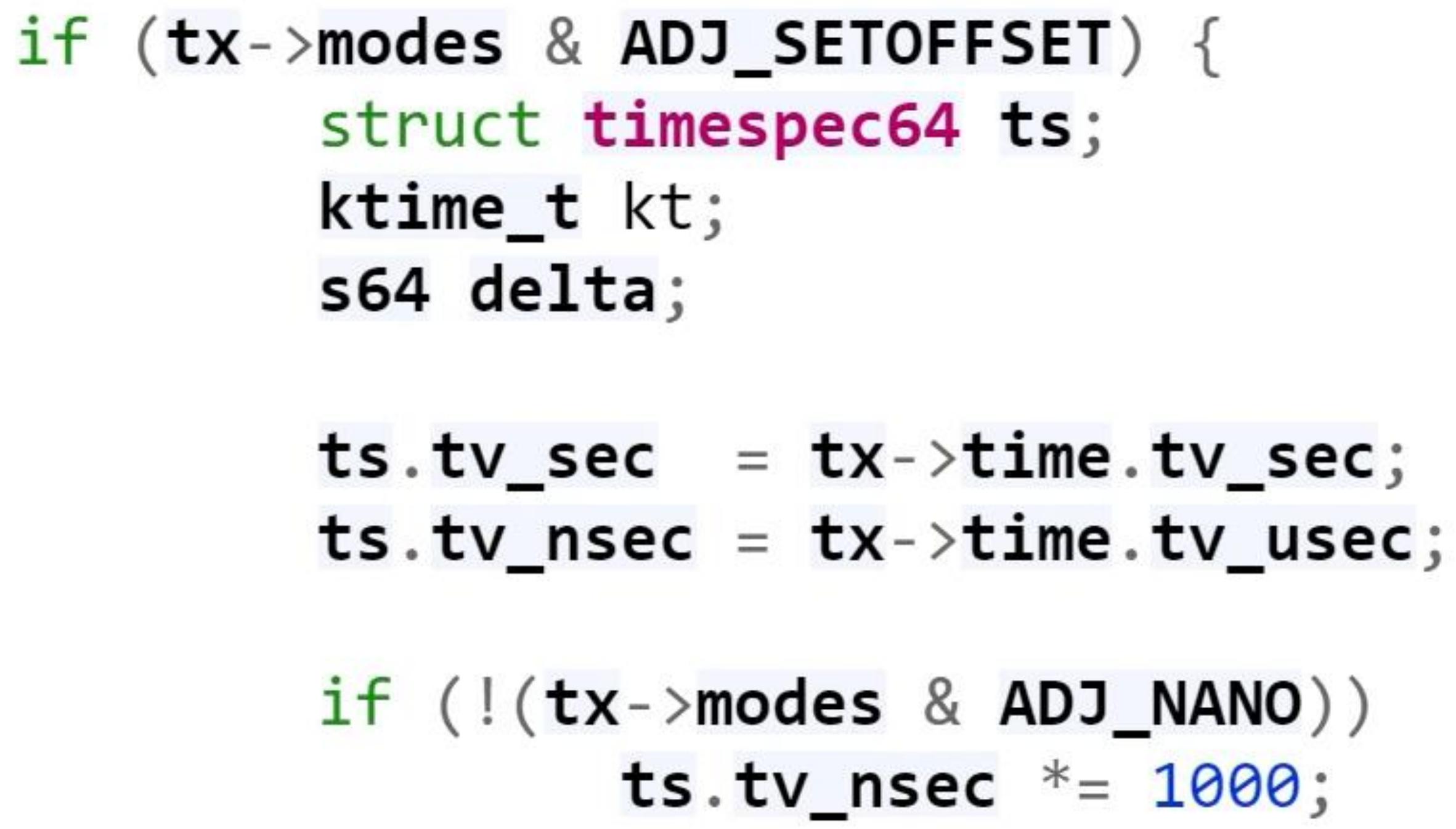
- Not trying to define daemon/lib API for reading clock
  - Earliest/latest calculation
  - Understanding which PHC is a source of time

Current API

https://elixir.bootlin.com/linux/latest/source/include/linux/ptp_clock_kernel.h#L57



```c
struct ptp_clock_info {
        struct module *owner;
        char name[PTP_CLOCK_NAME_LEN];
        s32 max_adj;
        int n_alarm;
        int n_ext_ts;
        int n_per_out;
        int n_pins;
        int pps;
        struct ptp_pin_desc *pin_config;
        int (*adjfine)(struct ptp_clock_info *ptp, long scaled_ppm);
        int (*adjphase)(struct ptp_clock_info *ptp, s32 phase);
        s32 (*getmaxphase)(struct ptp_clock_info *ptp);
        int (*adjtime)(struct ptp_clock_info *ptp, s64 delta);
        int (*gettime64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
        int (*gettimex64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                          struct ptp_system_timestamp *sts);
        int (*getcrosststamp)(struct ptp_clock_info *ptp,
                              struct system_device_crosststamp *cts);
        int (*settime64)(struct ptp_clock_info *p, const struct timespec64 *ts);
        int (*getcycles64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
        int (*getcyclesx64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                            struct ptp_system_timestamp *sts);
        int (*getcrosscycles)(struct ptp_clock_info *ptp,
                              struct system_device_crosststamp *cts);
        int (*enable)(struct ptp_clock_info *ptp,
                      struct ptp_clock_request *request, int on);
        int (*verify)(struct ptp_clock_info *ptp, unsigned int pin,
                      enum ptp_pin_function func, unsigned int chan);
        long (*do_aux_work)(struct ptp_clock_info *ptp);
};
```



```c
if (tx->modes & ADJ_SETOFFSET) {
        struct timespec64 ts;
        ktime_t kt;
        s64 delta;

        ts.tv_sec  = tx->time.tv_sec;
        ts.tv_nsec = tx->time.tv_usec;

        if (!(tx->modes & ADJ_NANO))
                ts.tv_nsec *= 1000;
```

Challenges

- Error is usually not pushed to the kernel
  - Except adjphase

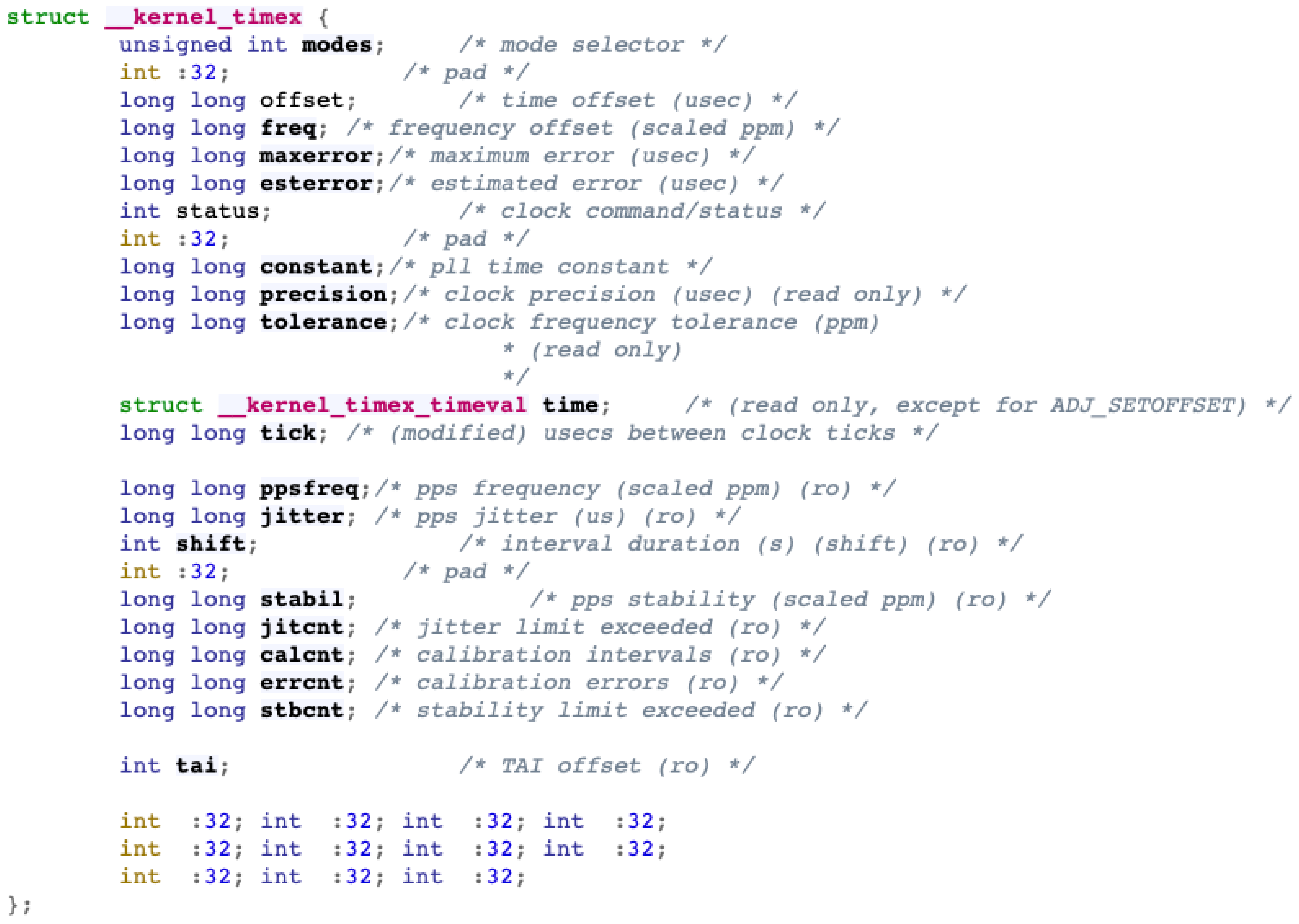
Missing APIs

- Last reported error
- Clock state
- Info about the oscillator stability (PPB)
    - Allows calculating error bounds as last error +/- time elapsed since it was measured * stability

- Stretch goal:
    - Programmable static error (e.g. quantization)
    - block clock read when the error is not in bounds
    - Block time read when error not in pre-set bounds
    - Dataset of the GM

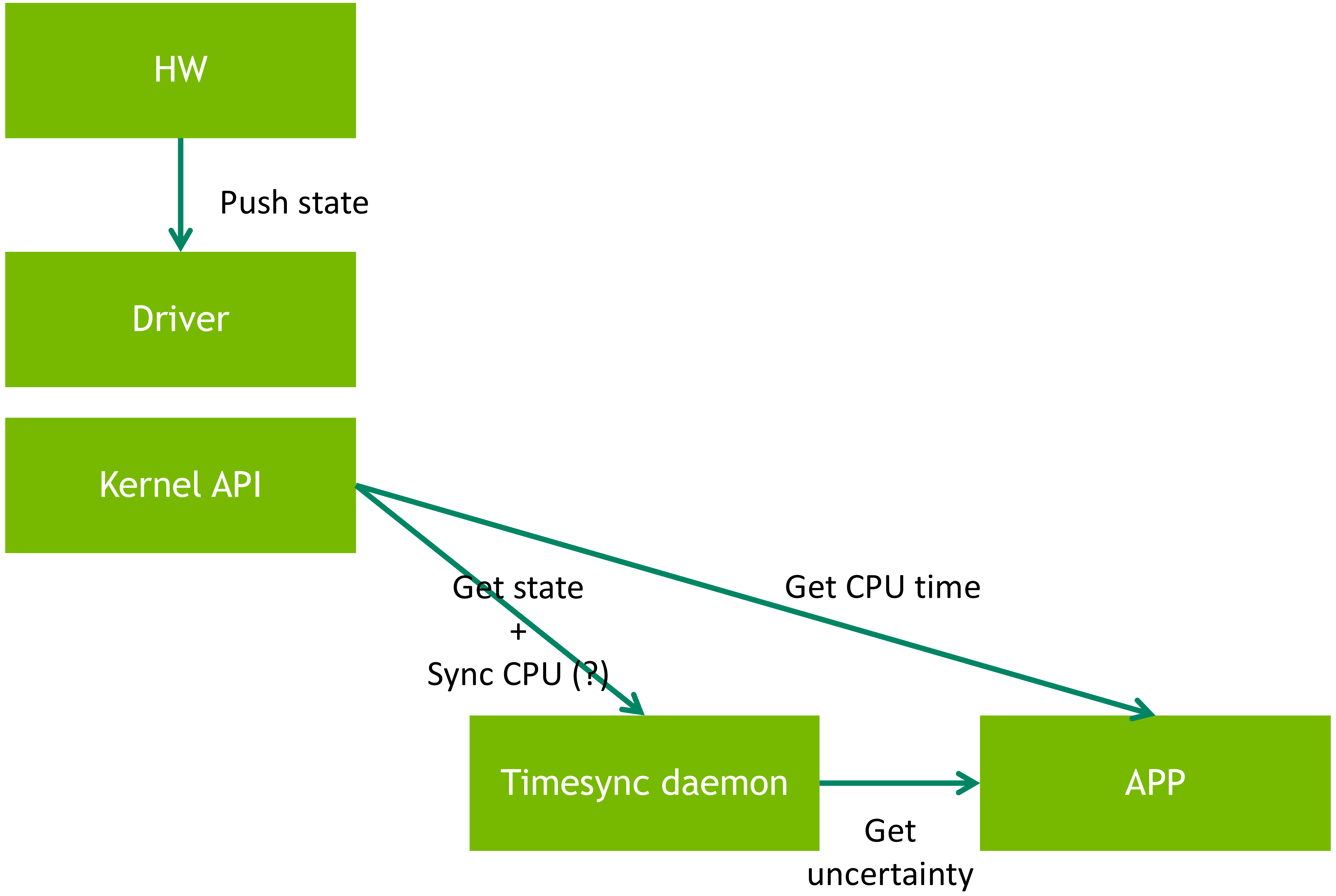
clock_adjtime

- Operates on the `timex` structure
- If called without any flags set – it returns info about the clock
  - For PTP clocks - information is limited to the current freq offset

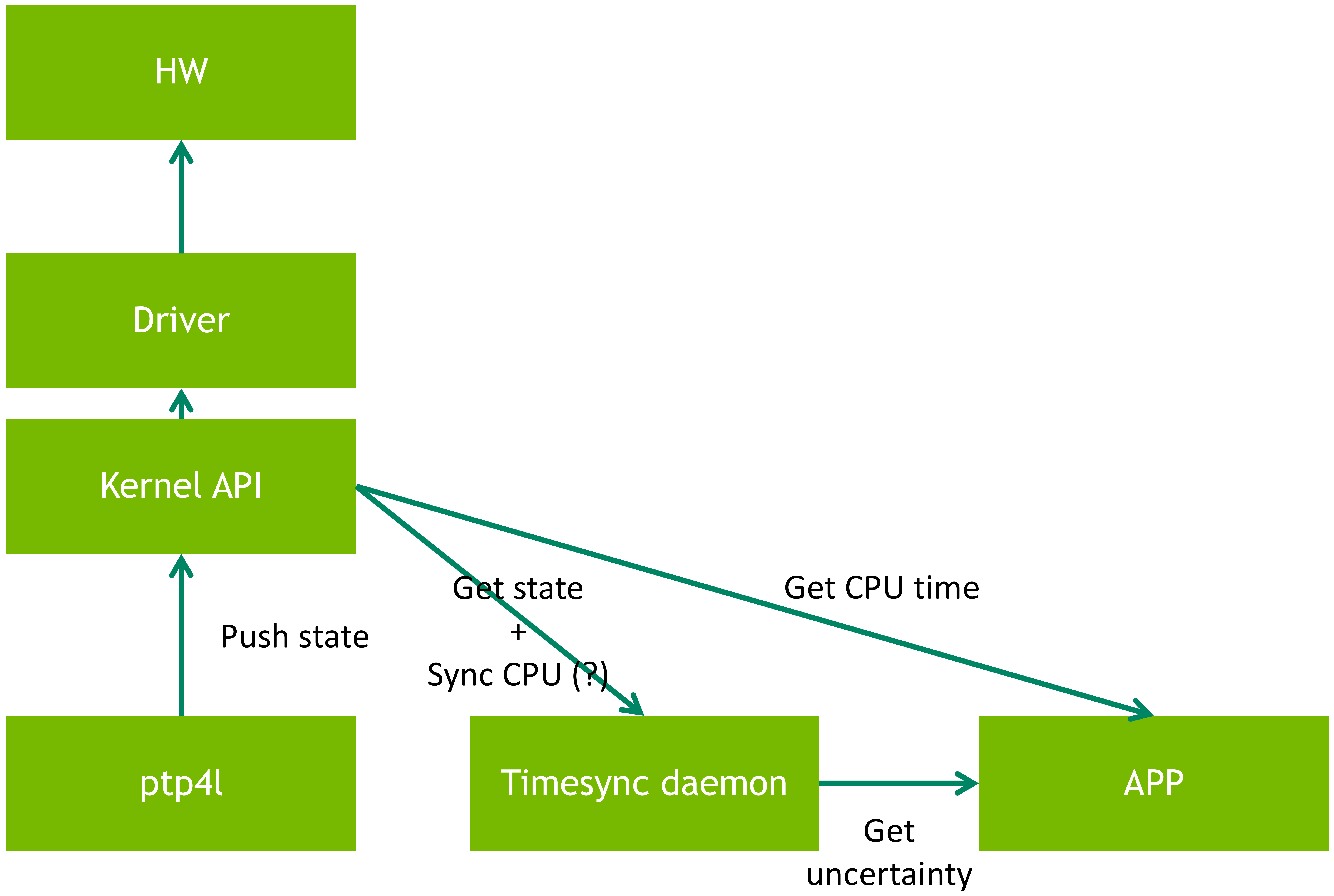


```c
struct __kernel_timex {
        unsigned int modes;     /* mode selector */
        int :32;            /* pad */
        long long offset;       /* time offset (usec) */
        long long freq; /* frequency offset (scaled ppm) */
        long long maxerror;/* maximum error (usec) */
        long long esterror;/* estimated error (usec) */
        int status;             /* clock command/status */
        int :32;            /* pad */
        long long constant;/* pll time constant */
        long long precision;/* clock precision (usec) (read only) */
        long long tolerance;/* clock frequency tolerance (ppm)
                                   * (read only)
                                   */
        struct __kernel_timex_timeval time;     /* (read only, except for ADJ_SETOFFSET) */
        long long tick; /* (modified) usecs between clock ticks */

        long long ppsfreq;/* pps frequency (scaled ppm) (ro) */
        long long jitter; /* pps jitter (us) (ro) */
        int shift;              /* interval duration (s) (shift) (ro) */
        int :32;            /* pad */
        long long stabil;            /* pps stability (scaled ppm) (ro) */
        long long jitcnt; /* jitter limit exceeded (ro) */
        long long calcnt; /* calibration intervals (ro) */
        long long errcnt; /* calibration errors (ro) */
        long long stbcnt; /* stability limit exceeded (ro) */

        int tai;                /* TAI offset (ro) */

        int  :32; int  :32; int  :32; int  :32;
        int  :32; int  :32; int  :32; int  :32;
        int  :32; int  :32; int  :32;
};
```

Device owns sync, state pushed to the OS



OS owns sync, state pushed to the device



Proposal

- Add `getlastrerror` function
  - Read the last error value from the device
  - Read the system clock and save as last update time
    - not handled by timex structure
    - should we push it to ethtool stats?
  - Kernel_timex has
    - `maxerror` and `esterror`;
    - `timex.mode` accepts `ADJ_ESTERROR` and `ADJ_MAXERROR`
- Add `getclockstate` function
    - Return the state of the clock:
      - Unknown (never locked)
      - locked
      - freerunning (after it was locked at least once)
    - Timex only supports setting/clearing STA_UNSYNC

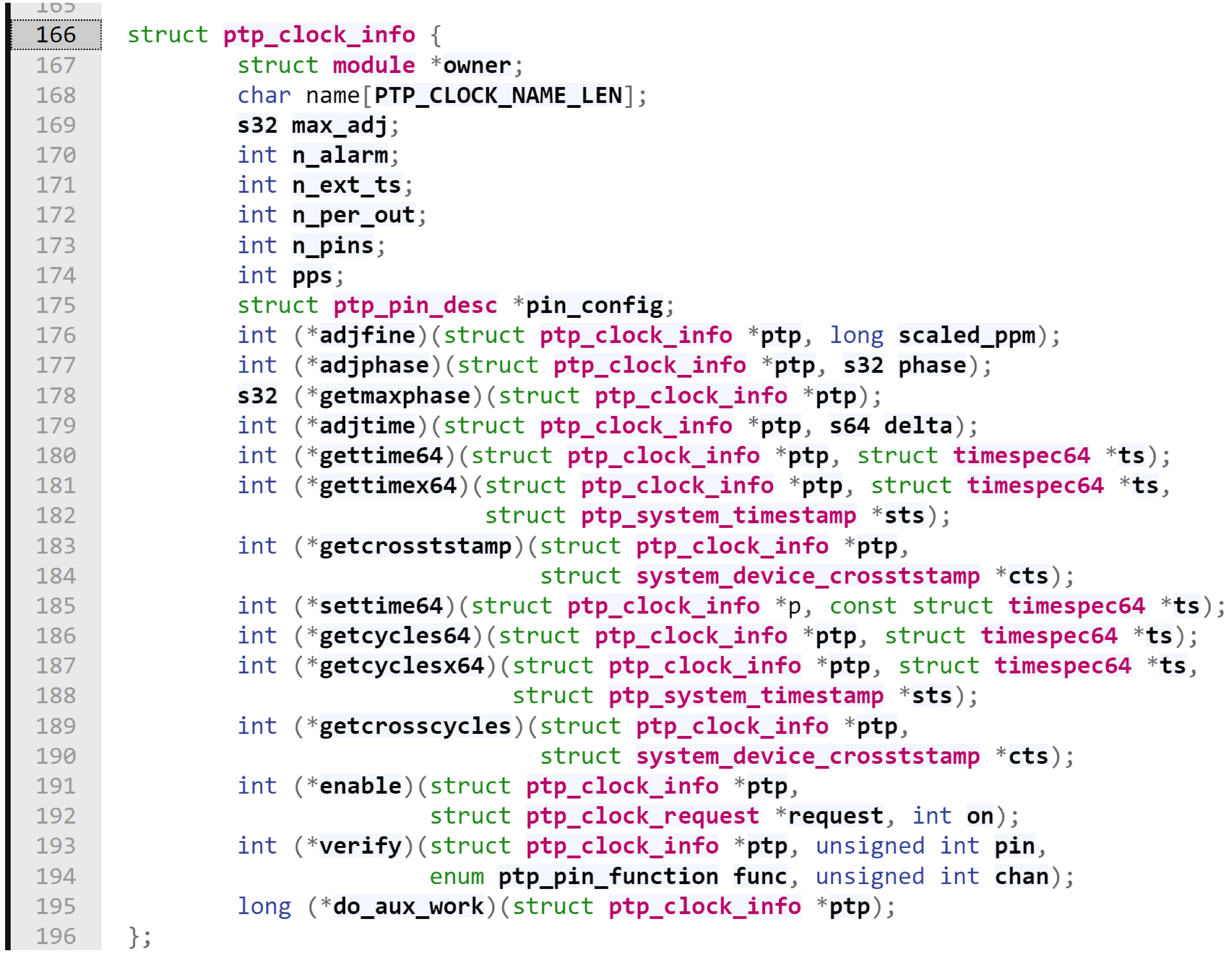
Proposal

- Add `getstabil` function
  - Return the ppb or ppt of the oscillator on the device
    - can dynamically change and will call into the driver – e.g. SyncE
  - Maximum expected frequency error from the point the last error was registered
  - Kernel_timex has the
    - `stabil`; */* pps stability (scaled ppm) (ro) */*
    - `tolerance`; */* clock frequency tolerance (ppm) */*
- Add `setclockstate` function (make it only handled by the driver)
  - ADJ_STATUS ADJ_ESTERROR and other relevant flags in `timex.mode`
  - accept the last error, device timestamp of that error measurement and clock state

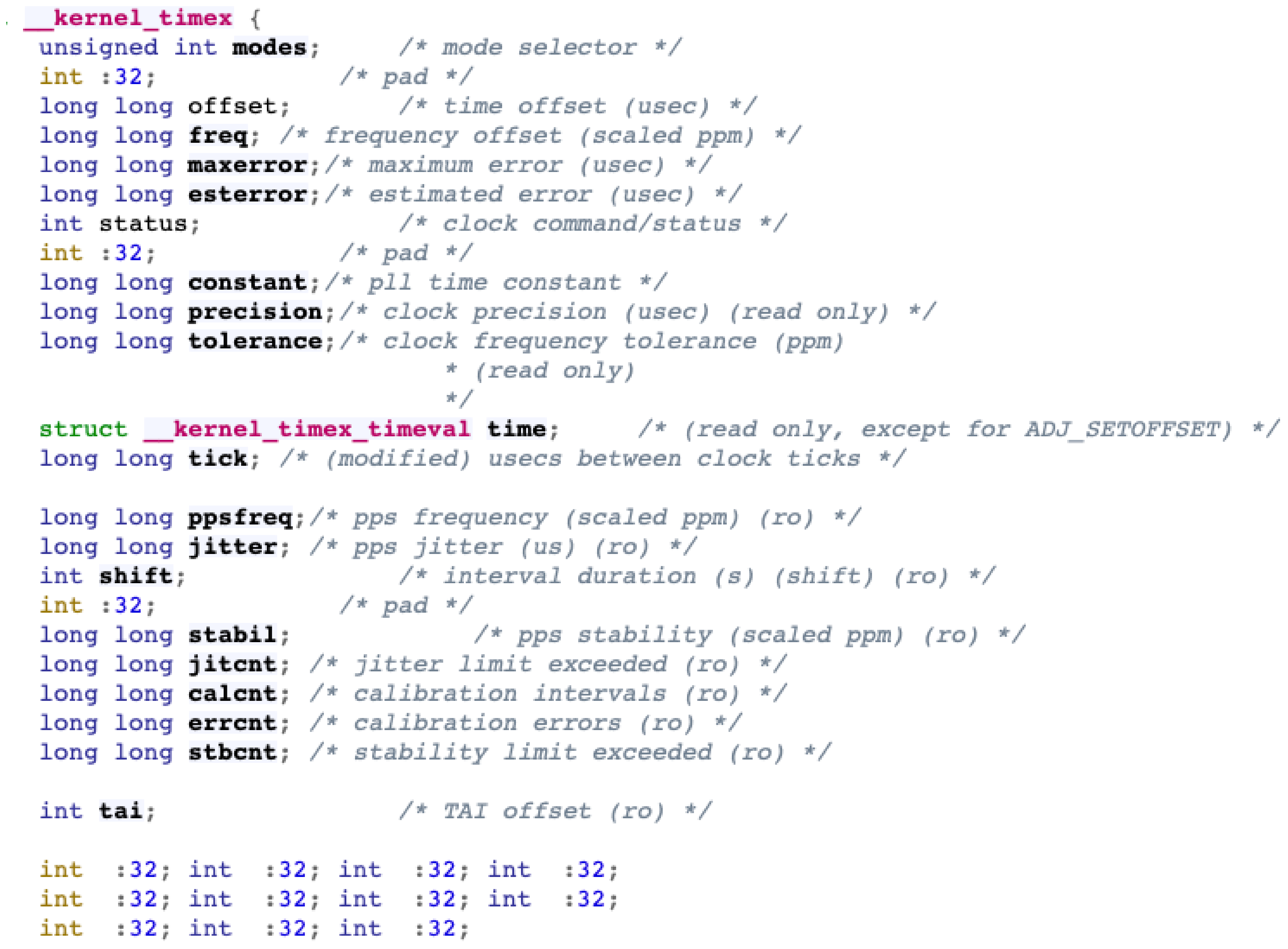
Proposal

- OPTIONAL:
- Add programmable baseline error – depending on the network – not handled by any APIs
  - maybe split in programmable and static from the driverInfo about the dataset?
- Add function returning max error = (last error + (current time – last error time) * precision)
- Add option to block clock read when the error is not in bounds
  - Need some timeout to recover from this state
  - Need some interface to program the threshold and timeout
- Some stats/counters?
- Info about the dataset?

clock_adjtime



```
struct ptp_clock_info {
        struct module *owner;
        char name[PTP_CLOCK_NAME_LEN];
        s32 max_adj;
        int n_alarm;
        int n_ext_ts;
        int n_per_out;
        int n_pins;
        int pps;
        struct ptp_pin_desc *pin_config;
        int (*adjfine)(struct ptp_clock_info *ptp, long scaled_ppm);
        int (*adjphase)(struct ptp_clock_info *ptp, s32 phase);
        s32 (*getmaxphase)(struct ptp_clock_info *ptp);
        int (*adjtime)(struct ptp_clock_info *ptp, s64 delta);
        int (*gettime64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
        int (*gettimex64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                          struct ptp_system_timestamp *sts);
        int (*getcrosststamp)(struct ptp_clock_info *ptp,
                              struct system_device_crosststamp *cts);
        int (*settime64)(struct ptp_clock_info *p, const struct timespec64 *ts);
        int (*getcycles64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
        int (*getcyclesx64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                            struct ptp_system_timestamp *sts);
        int (*getcrosscycles)(struct ptp_clock_info *ptp,
                              struct system_device_crosststamp *cts);
        int (*enable)(struct ptp_clock_info *ptp,
                      struct ptp_clock_request *request, int on);
        int (*verify)(struct ptp_clock_info *ptp, unsigned int pin,
                      enum ptp_pin_function func, unsigned int chan);
        long (*do_aux_work)(struct ptp_clock_info *ptp);
};
```



```
__kernel_timex {
 unsigned int modes;      /* mode selector */
 int :32;             /* pad */
 long long offset;        /* time offset (usec) */
 long long freq; /* frequency offset (scaled ppm) */
 long long maxerror;/* maximum error (usec) */
 long long esterror;/* estimated error (usec) */
 int status;              /* clock command/status */
 int :32;             /* pad */
 long long constant;/* pll time constant */
 long long precision;/* clock precision (usec) (read only) */
 long long tolerance;/* clock frequency tolerance (ppm)
                          * (read only)
                          */
 struct __kernel_timex_timeval time;     /* (read only, except for ADJ_SETOFFSET) */
 long long tick; /* (modified) usecs between clock ticks */

 long long ppsfreq;/* pps frequency (scaled ppm) (ro) */
 long long jitter; /* pps jitter (us) (ro) */
 int shift;               /* interval duration (s) (shift) (ro) */
 int :32;             /* pad */
 long long stabil;            /* pps stability (scaled ppm) (ro) */
 long long jitcnt; /* jitter limit exceeded (ro) */
 long long calcnt; /* calibration intervals (ro) */
 long long errcnt; /* calibration errors (ro) */
 long long stbcnt; /* stability limit exceeded (ro) */

 int tai;                 /* TAI offset (ro) */

 int  :32; int  :32; int  :32; int  :32;
 int  :32; int  :32; int  :32; int  :32;
 int  :32; int  :32; int  :32;
```