“Daylight saving time: Only the government would believe that you could cut a foot off the top of a blanket, sew it to the bottom, and have a longer blanket.”

– Origin Unclear
Date and Time

Anti-Patterns for Date and Time:
- Daylight savings time hard-coded
- Time kept without handling time problems
  - Daylight savings time, time zones, mobility
- Internationalization not considered

Date and Time issues
- Incrementing time is tricky
  - Leap years, leap seconds, DST changes
- Determining and displaying local time is tricky
  - Where are you? Time zone, local DST rules, local display rules
- Reconciling time is tricky
  - No two computers ever agree EXACTLY on time; updating time has tradeoffs
Keeping Time

- **Local time**: based on local microcontroller clock rate
  - System clock oscillator & timer interrupts ➔ software time

- **Drift rate**: local clock *rate* vs. perfect time
  - Example, 0.002% is 0.00002 seconds/sec drift rate

- **Accuracy**: local clock *value* vs. perfect time
  - Example: Node#2 is 1 second slower than true time
  - Every local clock has a different accuracy

- **Offset**: difference between two different clocks
  - Example: Node#1 is 3 seconds faster than Node#3
  - Every pair of local clocks has a different offset

- **Precision**: maximum offset between any two local clocks
  - Example: Node#3 is slowest in system; Node#1 is fastest in system. Precision is their offset = 3 seconds
  - There is only one precision value for a system

**Example Calculations**

- **Reference Time**: 8:01
- **Node 1**: 8:02 (OFFSET: +0:01)
- **Node 2**: 8:00 (OFFSET: -0:01)
- **Node 3**: 7:59 (OFFSET: -0:02)

**Precision Calculation**

\[ 8:02 - 7:59 = 0:03 \]
How do you fix an incorrect clock?

- Typically only periodic access to clock server
- **State correction**
  - Fast forward/reverse to correct time
  - Time jumps or even flows backwards
- **Rate correction**
  - Speed up/slow down local tick rate
  - Rate of time is slightly incorrect for a while

**Network Time Protocol (NTP)**

- Time maintenance service
  - Uses Internet access to estimate and track time
- Complex, and behavior depends upon options
Time Zones & Daylight Savings Time

- **Original time zones for UK rail schedules**
  - Not necessarily whole hours

- **DST changes arbitrary via governments**
  - WW II permanent DST (“War Time”)
  - Arizona does not observe DST
    - Navajo nation within Arizona does DST
    - Hopi Nation does not do DST

- **DST Dates differ by location**
  - US and Europe differ since 2007
  - Northern vs. Southern hemisphere
    - Fall and Spring are reversed!
Local Time

Types of time:
- Solar time: Based on mean sun position
- Local time: Time in your time zone
- GMT: Greenwich Mean Time / time zone zero
  - Official time is Universal Coordinated Time (UTC)

Sunrise/Sunset depends on where you are
- Sun rises earlier at eastern end of time zone
- Depends upon latitude & longitude
  - Sun angle, length of day (including “midnight sun”)
  - Length of day increases slightly with altitude
- “Mean Sun” differs from actual sun
  - Depends on date and year
  - “Equation of time” calculation

https://goo.gl/uP6Wki
Mobility and Time

- Time depends upon your location
  - What time is it? Which event happened earlier?

- Date line (+1 day in Asia)
  - It is very far from a straight line
  - Changes occasionally

- Potential issues:
  - System with multiple users in different time zones
  - System moves between time zones
    - While powered on; while powered off
  - What if system is turned off during DST change?

- Best practice: keep time with GMT/UTC
  - Keep time globally in GMT/UTC
  - Display time locally (add in effects of DST, time zones)
Time Leaps

- **Leap years:** about 365.25 days/year
  - February 29th is leap day every 4 years, ...
    - except every 100 years, ...
      » except every 400 years
    - 1900, 2100 are **NOT** leap years
      » 2000 was a leap year

- **Leap seconds added to UTC:**
  - Earth’s rotation is not constant
  - Every once in a while, an extra second is inserted:
    - 61 second minute: 23:59:00, 23:59:60, 00:00:00 (+1 day)
    - Theoretically could have negative leap second

- **Local rule changes can cause time leaps**
  - Changing time zone of a location

- **Time rollovers can appear as a huge backward leap**
  - Y2K/Millenium Bug: 99 → 00 rollover on 2-digit years
  - Unix cron rollover: 03:14:07 UTC on 19 January 2038

---

**The Guardian**

**Zune bricking code uncovered: it's a leap year mistake, and not Microsoft's**

The flaw that made Zunes freeze has been tracked down, it seems, to a piece of bad programming emanating from Freescale (the semiconductor company spun off from Motorola).

Simply put, there was a loop to allow for leap years (as 2008 was). However, it didn’t have any way to get past the beginning of the 366th day of the year.

```
year = ORIGINYEAR; /* = 1980 */
while (days > 365)
{ if (IsLeapYear(year))
  { if (days > 366)
    { days -= 366; year += 1;
    }  // MISSING ELSE!
  }
} else
{ days -= 365; year += 1;
}
```
What day is 02/03/16?
- US: Feb 3, 2016; Europe: 2 March 2016; or 1916?
  - What day does a week start on? (Sunday or Monday?)
- Mapping to traditional Chinese Lunisolar calendar?
  - Complete with Leap Month(s)

Example internationalization issues:
- AM/PM vs. 24 hour clock
- English vs. Metric (F/C, ft/meter, mph/kph, miles/km, ...)
- Currency signs, numeric notation (decimal vs. comma)
- Character sets (e.g. ASCII vs. unicode), word lengths
- Keyboard data entry (e.g., ASCII vs. Asian character entry vs. Arabic)
- Left to right, right to left, top to bottom text flow
- Gender in language, gender identification
Use validated time-keeping libraries
- It is complicated to get this right
- Need a way to change DST, time zones, etc.
- Consider what happens if your system moves

Think through internationalization
- What do you support/not support?

Pitfalls:
- In general, relying on accurate distributed time (“before”/“after”)
- Hard-coding changeable time rules (DST dates, time zones)
- Not considering internationalization needs