What you need to know about `datetimes`

Taavi Burns
taavi@freshbooks.com @jaaaarel
Defect Poacher & Futurist

Time is complicated

- I know more about it now than I did 3 weeks ago
- I'll probably know more by the end of PyCon!

Outline

- What's an "Instant"?
- Time standards
- Timezones
- Python modules: `time`, `calendar`, Naive vs Aware `datetimes`, `pytz`
- Interoperability

An Instant

- "An instant is a infinitesimal moment in time, a moment whose passage is instantaneous."
  - http://en.wikipedia.org/wiki/Instant

An Instant

- The instant represented by
  - 2012-03-10 10:30:00 PST
- Is the same instant represented by
  - 2012-03-10 13:30:00 EST
- Let's get into that...

Time standards

- "A time standard is a specification for measuring time: either the rate at which time passes; or points in time; or both."
- http://en.wikipedia.org/wiki/Time_standard
- UT1, TAI, UTC, POSIX, ...

UT_1

- Based entirely on Earth's rotation
- Length of a second depends on tides, weather, incoming comets...
- Earth Rotation Angle =

$2\pi(0.7790572732640 + 1.00273781191135448T_u)$ radians

  - http://en.wikipedia.org/wiki/Ut1

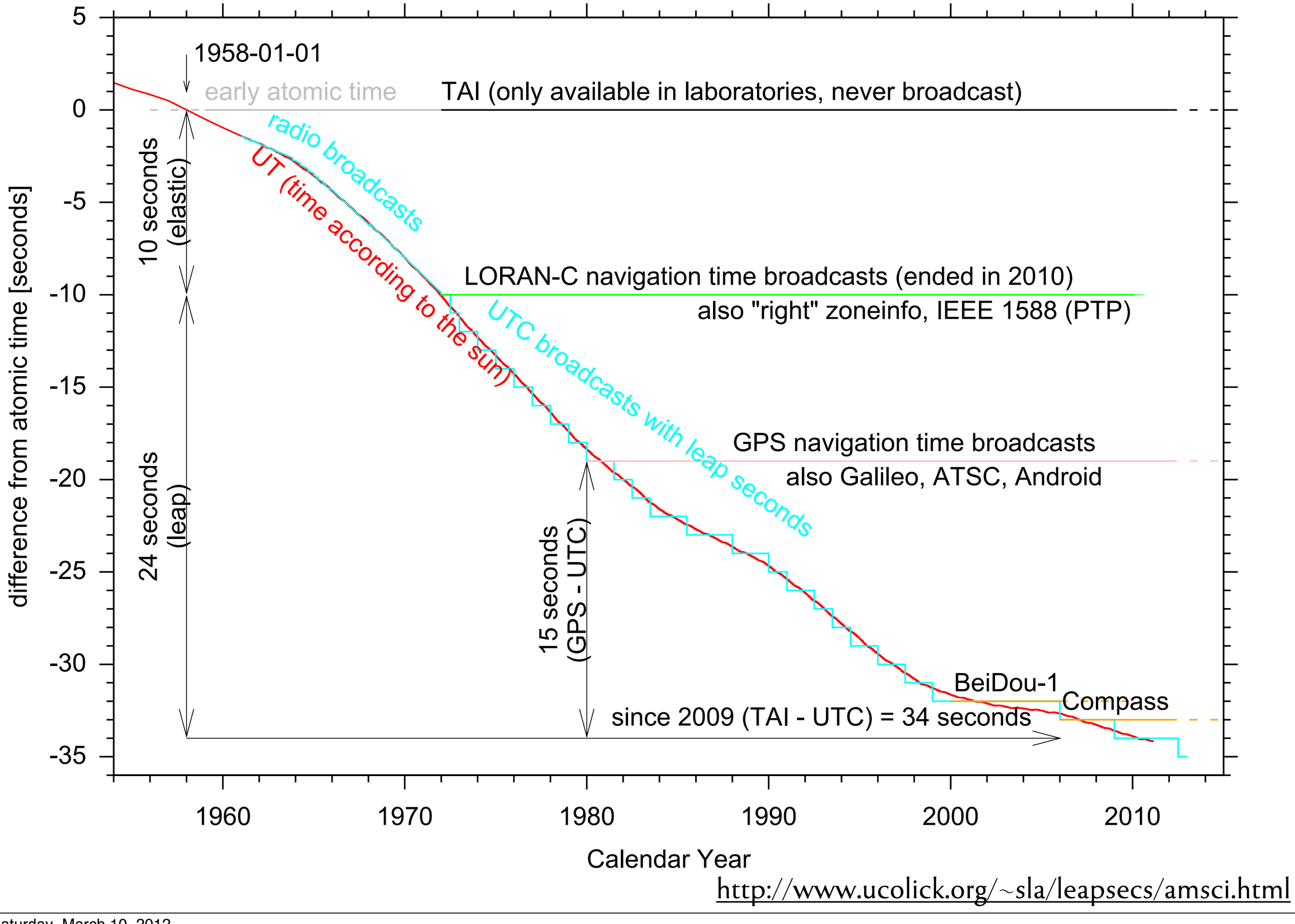
TAI

- International Atomic Time
- Average of 200 atomic clocks worldwide
- Counts SI seconds: “By definition, radiation produced by the transition between the two hyperfine ground states of caesium (in the absence of external influences such as the Earth’s magnetic field) has a frequency of exactly 9 192 631 770 Hz.”
  - http://en.wikipedia.org/wiki/Caesium_standard
- Is drifting ahead of “reality” by keeping a straight beat

UTC

- Coordinated Universal Time
- Ticks SI seconds, which are (usually) shorter than UT1 seconds
  - Hence leap-seconds are added to keep UTC within 0.9s of UT1
  - 2012-06-30 23:59:60 UTC is a real time!
- Since 1972, is an integral number of seconds offset from TAI

Time scales since the cesium atomic frequency standard



http://www.ucolick.org/~sla/leapsecs/amsci.html

POSIX (UNIX) timestamp

- Count of SI seconds since 1970-01-01 00:00:00 UTC
  - (except that UTC wasn't that "simple" until 1972…)
- Except the leap seconds, which are "ignored"
  - Exactly 86400 seconds in a day
  - Leap seconds are superimposed on the next second

POSIX (UNIX) timestamp

- Leap years ARE accounted for
  - ```
    def _days_before_year(year):
        y = year - 1
        return y*365 + y//4 - y//100 + y//400
    ```
- Will not agree with history books on dates before October 1582, and is muddy into the early 1900s
- http://en.wikipedia.org/wiki/Gregorian_calendar
    ```

Two kinds of leaps

- Leap seconds keep the sun above at noon (daytime is daytime)
- Leap days keep the solstice/equinox days in the right places (summer is summer)

Timezones

- It used to be that every town had its own clock, calibrated to local noon
- No wonder the trains never ran on time! (it was hard to tell what time it was!)
- "The first time zone in the world was established on December 1, 1847, on the island of Great Britain by railway companies using GMT kept by portable chronometers."
  - http://en.wikipedia.org/wiki/Time_zone

http://upload.wikimedia.org/wikipedia/commons/1/18/Worldwide_Time_Zones_%28including_DST%29.png

GMT != UTC

- GMT is the mean solar time at the Greenwich Observatory, closer to UT1
- If you don't care about sub-second precision, maybe you don't have to worry
- But it's confusing, because in England, GMT gives way to BST (British Summer Time) in the summer!
- UTC NEVER has daylight savings, noon is always 12:00
- Multiple conventions: Noon as 12:00 or 00:00!
- http://en.wikipedia.org/wiki/Gmt

Use UTC for everything

- In the words of Armin Ronacher:
  - “Always measure and store time in UTC or UNIX timestamps”
  - “Do not use offset aware datetimes”

Except user I/O

- In the words of Armin Ronacher:
  - "If you are taking in user input that is in local time, immediately convert it to UTC. If that conversion would be ambiguous let the user know."
  - "Rebase for Formatting (then throw away that filthy offset aware datetime object)"
- From http://lucumr.pocoo.org/2011/7/15/eppur-si-muove/

Time for some Python

- `time`
- `calendar`
- `datetime`
- `pytz` (from pypi)

`time`

- Interface to `libc`
  - Think `thread` and `os.fork`
- Deals with POSIX timestamps and `struct_time`
- Timezones supported only by setting `os.environ["TZ"]`
- `struct_time` is naive, but has an `is_dst` flag
  - given a DST-aware timezone, it indicates whether DST is in effect or not
  - Helps to disambiguate 01:30

`time`

- Use `time.time()` to get the current POSIX timestamp
- Use `time.gmtime(t)` to get a `struct_time` representing the UTC time of:
  - (if `t == None`) the current instant, or
  - the provided POSIX timestamp

`calendar`

- Unrelated to `datetimes`, except for...
- Use `calendar.timegm(tuple)` to turn a `struct_time` ("time tuple") in UTC into a POSIX timestamp
- http://bugs.python.org/issue6280 proposed to move it to the `time` module beside `gmtime()`, but rejected

`datetime`

- Python object interface to dates, times, intervals, and timezones
  - Think `threading` and `subprocess`
- Two varieties:
  - Naive, which have no timezone information
  - Aware, which have timezone information
  - They do not mix!
- Unfortunately, there are still a lot of sharp edges

`datetime` - Do

- Use `pytz`
  - Ships with a copy of the Olson Timezone Database
  - Comes with NECESSARY helper functions to create local, aware (and correct) `datetimes`!
- Update `pytz` regularly for changes in timezones (including updated DST changes)

datetime - Do

- Make UTC aware datetimes the easy way:
  - `>>> datetime(2011, 11, 6, 5, 30,`
    `...    tzinfo=pytz.UTC)`
    `datetime.datetime(2011, 11, 6, 5, 30,`
    `tzinfo=<UTC>)`

datetime - Do

- Use `pytz.timezone().localize()` to make an aware `datetime` in a given timezone:
  - `>>> helsinki = pytz.timezone('Europe/Helsinki')`

```
>>> helsinki.localize(
...    datetime(2011, 11, 6, 5, 30))
datetime.datetime(2011, 11, 6, 5, 30,
tzinfo=<DstTzInfo 'Europe/Helsinki' EET+2:00:00
STD>)
```

datetime - Do

- Tell `.localize()` to NOT guess if you care about which side of DST things lie:
  - `>>> toronto = pytz.timezone('America/Toronto')`

```
>>> toronto.localize(
...    # Is this EDT or EST?
...    datetime(2011, 11, 6, 1, 30),
...    is_dst=None)
pytz.tzinfo.AmbiguousTimeError: 2011-11-06
01:30:00
```

datetime - Do

- Get an aware datetime for this instant in a given timezone the easy way:
  - `>>> toronto = pytz.timezone('America/Toronto')`

```
>>> datetime.now(toronto)
datetime.datetime(2012, 3, 5, 16, 40, 12,
967922, tzinfo=<DstTzInfo 'America/Toronto'
EST-1 day, 19:00:00 STD>)
>>> _.date()
datetime.date(2012, 3, 5)
```

datetime - Don't

- Make non-UTC aware datetimes the ~~easy~~ wrong way:
  - ```
    >>> toronto = pytz.timezone('America/Toronto')
    >>> datetime(2011, 6, 1, 0, 0, # summer = DST!
    ...   tzinfo=toronto)
    datetime.datetime(2011, 6, 1, 0, 0,
    tzinfo=<DstTzInfo 'America/Toronto' EST-1 day,
    19:00:00 STD>)
    >>> _.isoformat()
    '2011-06-01T00:00:00-05:00'
    ```
    ```

datetime - Don't

- Make non-UTC aware datetimes the ~~easy~~ wrong way:
  - `>>> datetime(2011, 11, 6, 5, 30,`
    `...    tzinfo=helsinki)`

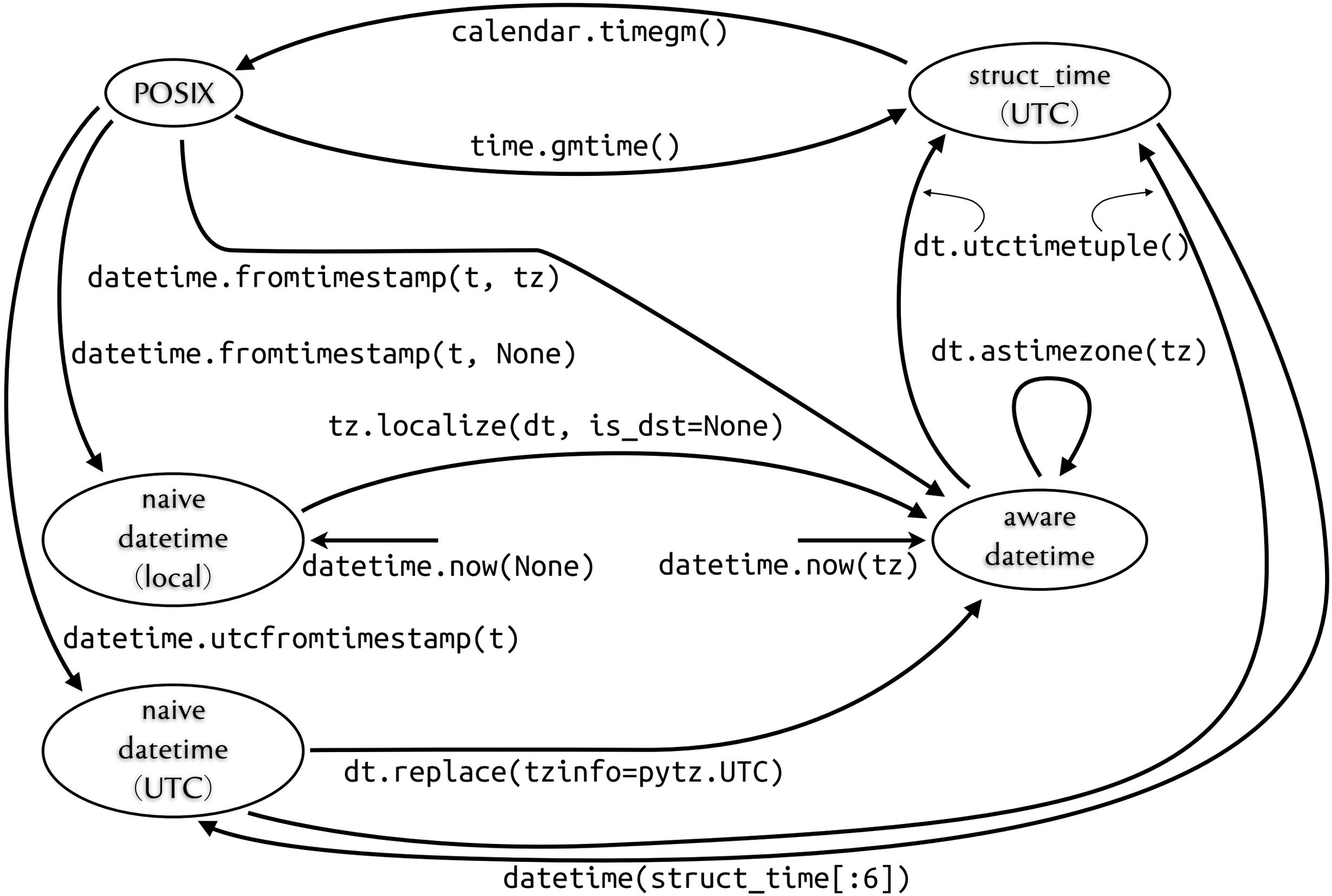
```
datetime.datetime(2011, 11, 6, 5, 30,
tzinfo=<DstTzInfo 'Europe/Helsinki' HMT+1:40:00
STD>)
```

lol wut

datetime - Don't

- Use `.replace()` to add a timezone to a naive `datetime`:
  - `>>> datetime(2011, 11, 6, 5, 30).replace(`
    `...  tzinfo=helsinki)`
    `datetime.datetime(2011, 11, 6, 5, 30,`
    `tzinfo=<DstTzInfo 'Europe/Helsinki' HMT+1:40:00`
    `STD>)`

lol wut



dt = datetime instance; tz = pytz.timezone(...) instance; t = POSIX

http://taaviburns.ca/what_you_need_to_know_about_datetimes/datetime_transforms.html

Interoperability

- The world isn't all Python
- MySQL
- PostgreSQL
- SQLite
- JavaScript

MySQL

- `date`
  - Pure dates, no timezones
- `datetime`
  - Like a Python naive datetime
- `timestamp`
  - Stored internally as a POSIX timestamp
  - Translated on I/O based on the connection timezone

MySQL

- `CONVERT_TZ(dt,from_tz,to_tz)` if you need to do translations
- Of course, DST WILL bite you (unless you're very careful, and store the concrete TZ for all local times)!

PostgreSQL

- `date`
  - Pure dates, no timezones
- `time`
  - “We recommend not using the type `time with time zone`”
- `timestamp`
  - Stored internally as seconds since 2000-01-01 00:00:00 UTC
  - Translated on I/O based on the connection timezone

PostgreSQL

- `AT TIME ZONE` operator if you need to translate
- http://www.postgresql.org/docs/8.1/static/functions-datetime.html#FUNCTIONS-DATETIME-ZONECONVERT

SQLite

- TEXT
  - “as ISO8601 strings (‘YYYY-MM-DD HH:MM:SS.SSS’)”
- REAL
  - “as Julian day numbers, the number of days since noon in Greenwich on November 24, 4714 B.C. according to the proleptic Gregorian calendar”
- INTEGER
  - “as Unix Time, the number of seconds since 1970-01-01 00:00:00 UTC”

SQLite

- The built-in date and time functions provide limited timezone support with magic modifier strings like `unixepoch`, `localtime`, and `utc`
- http://www.sqlite.org/lang_datefunc.html

JavaScript

- `Date` objects are in the local timezone and current DST setting
- `getUTC*` methods if you need them
- "The local time is the time known to the computer where JavaScript is executed"
- And you can interact with POSIX timestamps:
  - `new Date(posixTimestamp * 1000);`
  - `var posixTimestamp = Date.now()/1000;`
  - `(new Date(posixTimestamp * 1000)).getTime() / 1000 == posixTimestamp`

JavaScript

- But don't try to do anything in a different timezone, or across DST boundaries. :(

Things I read

- http://en.wikipedia.org/wiki/International_Atomic_Time
- https://www.eff.org/press/releases/eff-wins-protection-time-zone-database
- http://www.ucolick.org/~sla/leapsecs/amsci.html
- http://www.cl.cam.ac.uk/~mgk25/time/leap/
- http://www.bipm.org/en/si/si_brochure/chapter2/2-1/second.html
- http://www.iana.org/time-zones
- http://lucumr.pocoo.org/2011/7/15/eppur-si-muove/
- http://unix4lyfe.org/time/
- http://opensourcehacker.com/2008/06/30/relativity-of-time-shortcomings-in-python-datetime-and-workaround/

Things I read

- http://www.mail-archive.com/leapsecs@rom.usno.navy.mil/msg00109.html
- http://pypi.python.org/pypi/pytz/
- http://labix.org/python-dateutil
- And more...

Presented at PyCon US 2012 - Santa Clara, CA, USA

Bonus topics

- Leap seconds
- Implementing `timegm()`

Leap seconds

- `os.environ['TZ'] = 'right/UTC'`
  `time.tzset()`
- `mktime` becomes the inverse of `gmtime`!
- Beware: your timestamps are now NON-POSIX, and you'll probably be 24 seconds early for everything (and 25 after June)

Implementing timegm

- If you have Python 2.7 or above, you could just:
  - ```
    _EPOCH_DATETIME = datetime(1970, 1, 1)
    _SECOND = timedelta(seconds=1)
    def timegm(tuple):
        return (datetime(*tuple[:6]) -
            _EPOCH_DATETIME) // _SECOND
    ```
    ```

Presented at PyCon US 2012 - Santa Clara, CA, USA