Timezones in Python

Benjamin “Zags” Zagorsky

Who am I?

BENJAMIN ZAGORSKY

-------- ZAG--S--

ZAGS

zags@zagaran.com

Chief Technology Officer at Zagaran, Inc.

zagaran

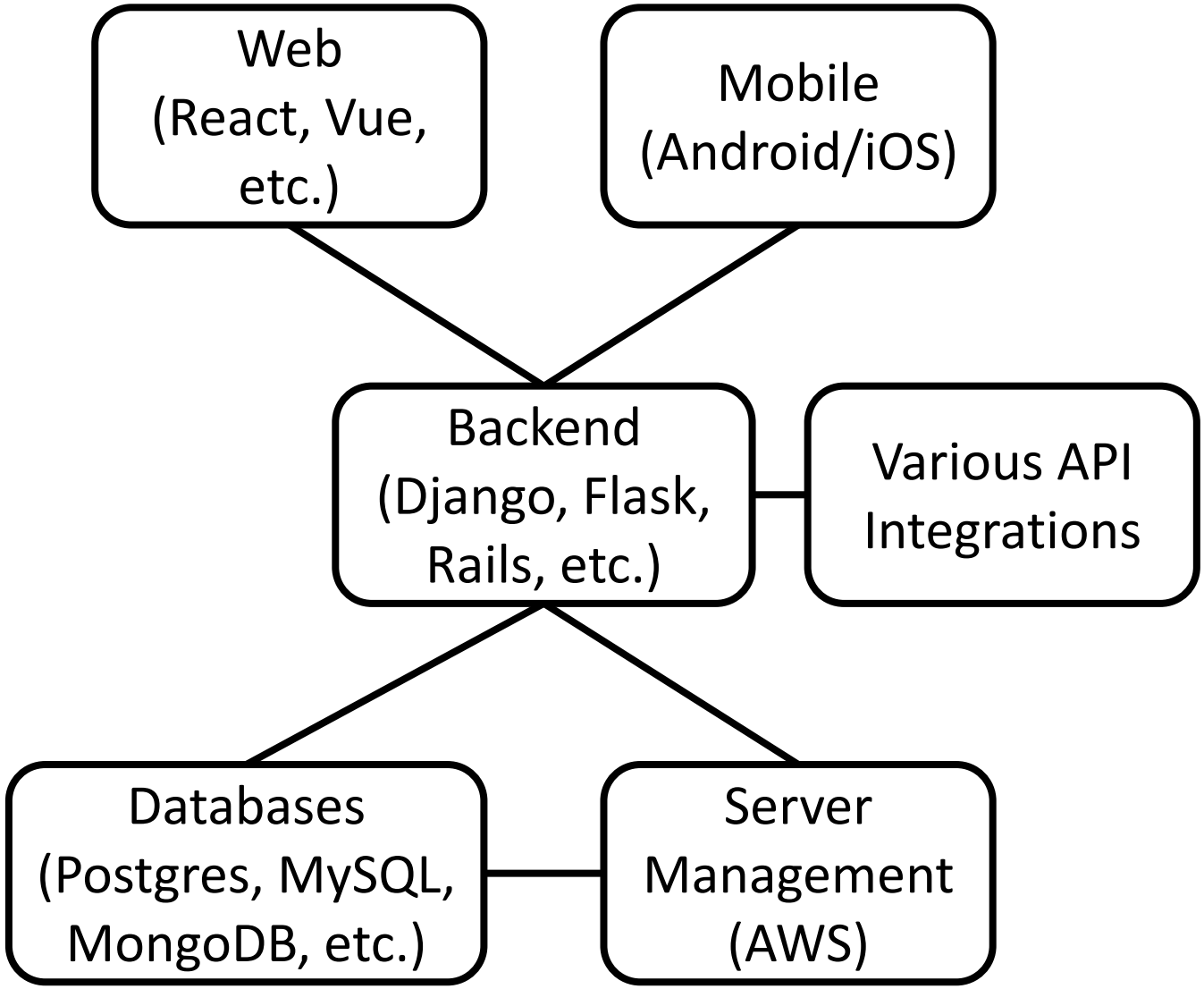
We build full stack software systems for other companies.

About my company

ZAGORSKY MCCARTHY FAN

ZAG----- ---AR--- -AN

Custom software development



Motivation

- What is today's date? The answer depends on where you are
- Whenever you're working with dates and times, you're working with timezones
- Even if the Sunshine Protection Act of 2021 passes, DST lives on in the form of historic timestamps and other countries
- Python 3.9: zoneinfo added to Python standard library
- Django 4: zoneinfo replaces pytz as the default timezone provider

Outline

- Handling Timezones Wrong
- Handling Timezones Right
- Common Recipes
- Django Extras

5 Cardinal Sins of Timezones

Don’t use naïve datetimes

```
from datetime import datetime, timezone
# naive datetime
datetime(2022, 1, 1, 12, 0)
# aware datetime
datetime(2022, 1, 1, 12, 0, tzinfo=timezone.utc)
```

- An aware datetime is a point in time
- A naïve datetime is the concept of a time and date (and therefore probably not what you want)
- Naïve datetimes have 26 hours of ambiguity (UTC-12 to UTC+14)

Don’t use naïve datetimes

- Don’t use `datetime.now()` because the result depends on system time
- Don’t use `datetime.utcnow()` because it’s missing tzinfo

```
# Local, system time US Eastern
datetime.now()
>>> datetime.datetime(2022, 4, 15, 16, 15, 22, 402186)

# Server, system time UTC
datetime.now()
>>> datetime.datetime(2022, 4, 15, 20, 15, 22, 402186)
```

Don’t use dates without a timezone

- While dates don’t have timezone-awareness, they still exist within a timezone
- Don’t use `date.today()` as it still depends on system time
- Can lead to bugs that only show up between 7pm and midnight

```
# Local, 10pm US Eastern
date.today()
>>> datetime.date(2022, 4, 15)
# Server, 2am UTC
date.today()
>>> datetime.date(2022, 4, 16)
```

Don’t do duration arithmetic outside of UTC

```
tz = ZoneInfo("America/New_York")
dt = datetime(2022, 3, 13, 1, 30, tzinfo=tz)
dt == dt.astimezone(timezone.utc)
>>> True

delta = timedelta(hours=2)
dt + delta == dt.astimezone(timezone.utc) + delta
>>> False
```

Don’t do duration math outside of UTC

```
tz = ZoneInfo("America/New_York")
dt = datetime(2022, 3, 13, 1, 30, tzinfo=tz)
dt.isoformat()
>>> '2022-03-13T01:30:00-05:00'

(dt + timedelta(hours=2)).isoformat()
>>> '2022-03-13T03:30:00-04:00'
```

Don’t use pytz

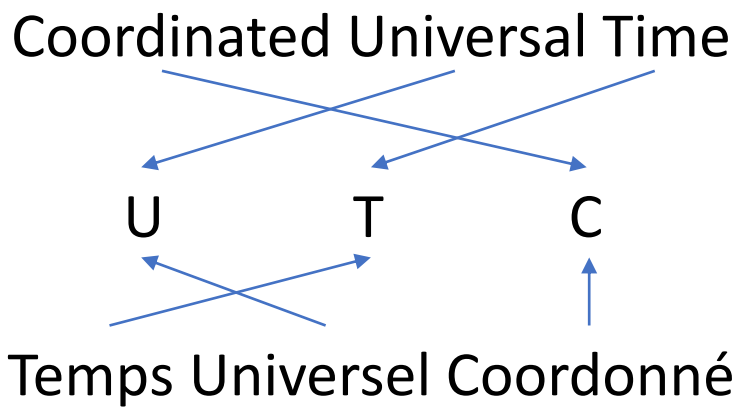
```
import pytz
tz = pytz.timezone("America/New_York")
dt = datetime(2022, 4, 6, 12, 0, tzinfo=tz)
dt.isoformat()
>>> '2022-04-06T12:00:00-04:56'

dt.astimezone(timezone.utc)
>>> datetime.datetime(2022, 4, 6, 16, 56, tzinfo=<UTC>)
```

Don’t do aware_datetime.replace(tzinfo=xxx)

```
tz = ZoneInfo("America/New_York")
dt = datetime(2022, 4, 6, 12, 0, 0, tzinfo=tz)
dt == dt.replace(tzinfo=timezone.utc)
>>> False

dt.replace(tzinfo=timezone.utc)
>>> datetime(2022, 4, 6, 12, 0, tzinfo=<UTC>)
```

Doing it Right

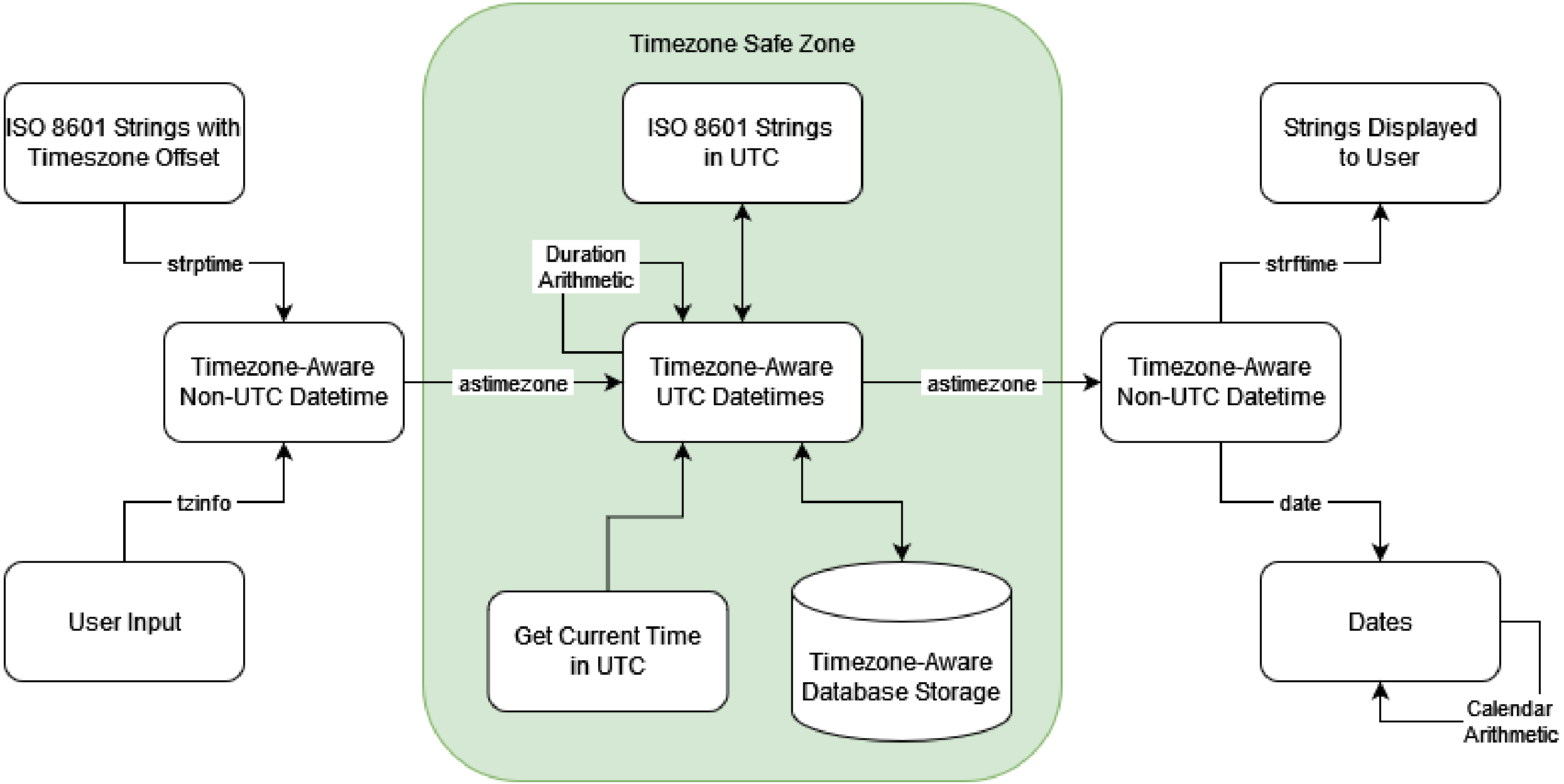
UTC: “Coordinated Universal Time”



Bad acronym but a great standard:

- UTC has no temporal discontinuities*
- All other timezones are defined in relation to UTC
- It’s the official timezone of the International Space Station

*Except for leap seconds


Timezone Safe Zone
ISO 8601 Strings with Timeszone Offset
strptime
Timezone-Aware Non-UTC Datetime
tzinfo
User Input
astimezone
ISO 8601 Strings in UTC
Duration Arithmetic
Timezone-Aware UTC Datetimes
Get Current Time in UTC
Timezone-Aware Database Storage
astimezone
Strings Displayed to User
strftime
Timezone-Aware Non-UTC Datetime
date
Dates
Calendar Arithmetic

Use Zoneinfo (instead of pytz)

```
from zoneinfo import Zoneinfo
tz = ZoneInfo("America/New_York")
dt = datetime(2022, 4, 6, 12, 0, tzinfo=tz).isoformat()
>>> '2022-04-06T12:00:00-04:00'
```

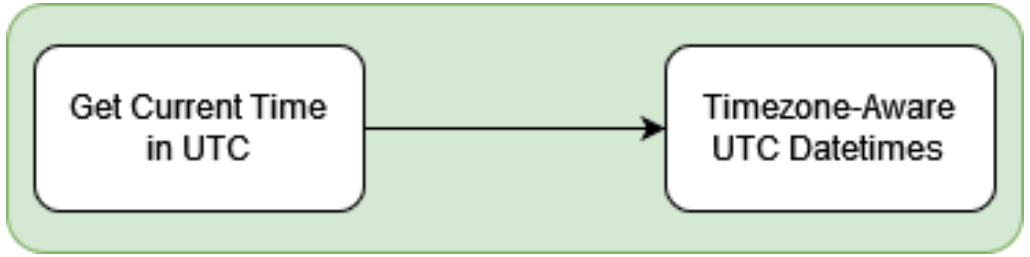
```
import pytz
tz = pytz.timezone("America/New_York")
dt = datetime(2022, 4, 6, 12, 0, tzinfo=tz).isoformat()
>>> '2022-04-06T12:00:00-04:56'
```

Maximum compatibility

```
# requirements.txt
backports.zoneinfo;python_version<"3.9"  # Python <= 3.8
tzdata  # Systems without timezone data (ex. Windows)

# code.py
try:
    from zoneinfo import ZoneInfo
except ImportError:
    from backports.zoneinfo import ZoneInfo
```

Recipes for Success

Get current time

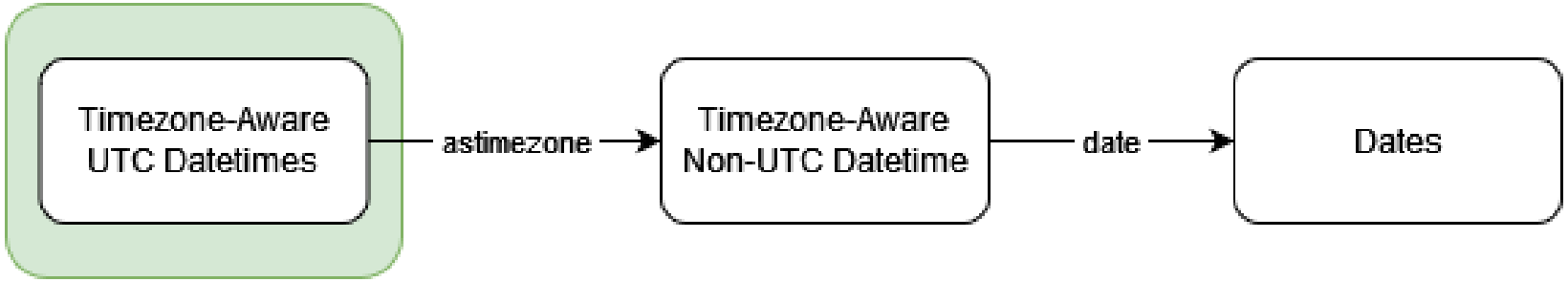



```
from datetime import datetime, timezone

datetime.now(timezone.utc) # Good
>>> datetime(2022, 4, 14, 21, 37, 22, tzinfo=<UTC>)

datetime.utcnow() # Bad
>>> datetime(2022, 4, 14, 21, 37, 22)
```

Get current date in a timezone

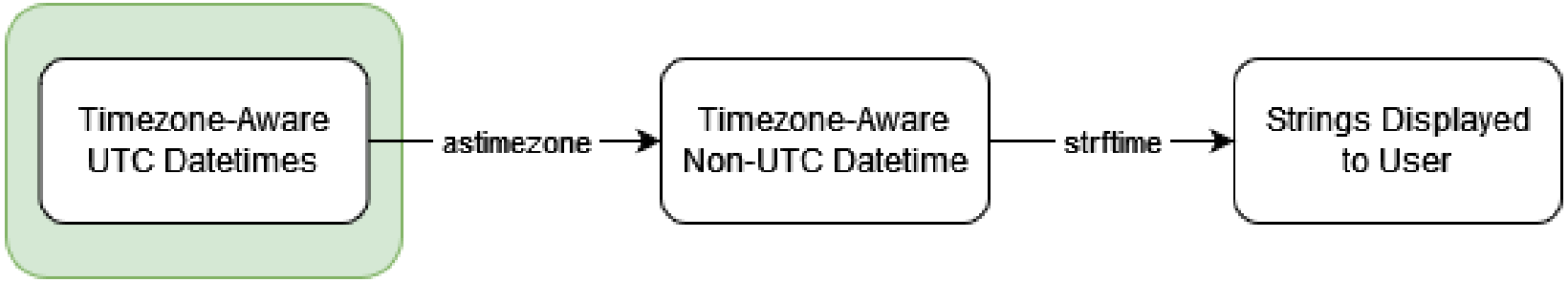
```
from datetime import datetime, date
from zoneinfo import ZoneInfo

datetime.now(ZoneInfo("America/New_York")).date()
```

Convert datetime to a date in a timezone



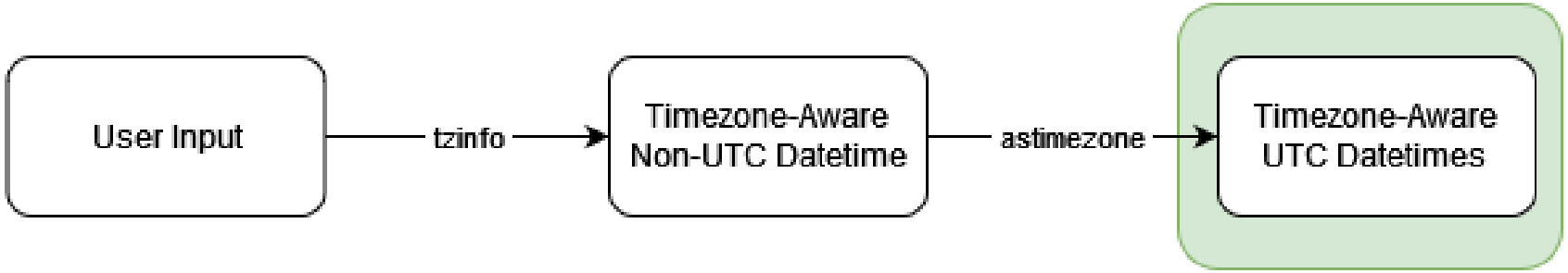

```
dt = datetime(2022, 4, 6, 2, 0, tzinfo=timezone.utc)

dt.astimezone(ZoneInfo("America/New_York")).date()
>>> date(2022, 4, 5)
```

Display UTC time as localtime (i.e. output)



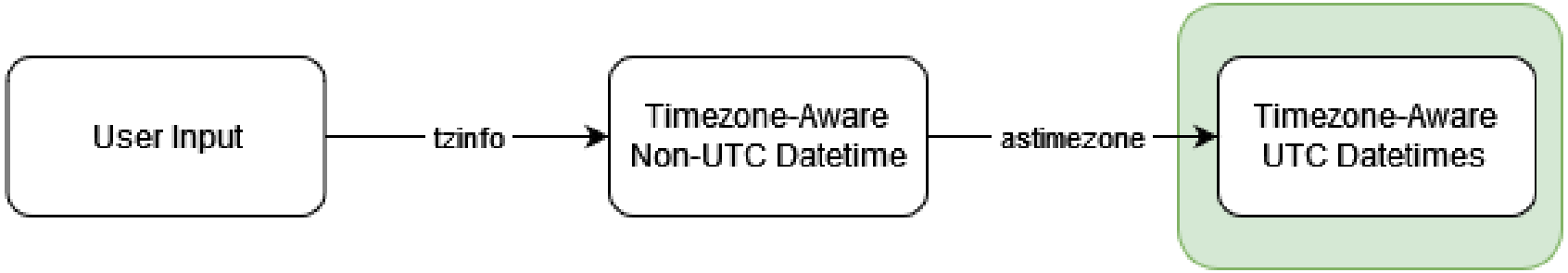

```
dt = datetime(2022, 4, 6, 2, 0, tzinfo=timezone.utc)
tz = ZoneInfo("America/New_York")
dt.astimezone(tz).strftime("%x %X")
>>> '04/05/22 22:00:00'
```

Convert naïve datetime to UTC (i.e. input)



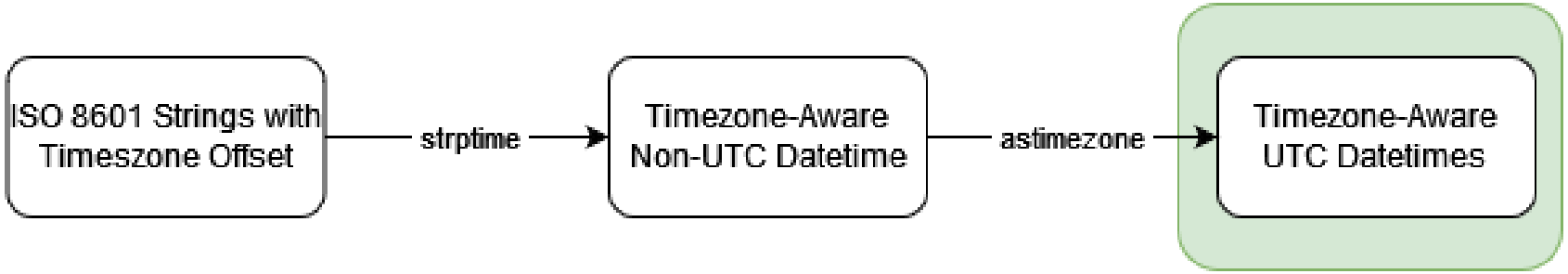

```
# Build a datetime with the local timezone
tz = ZoneInfo("America/New_York")
local_dt = datetime(2022, 4, 6, 12, 0, 0, tzinfo=tz)
local_dt.astimezone(timezone.utc)
```

Convert naïve datetime to UTC (i.e. input)



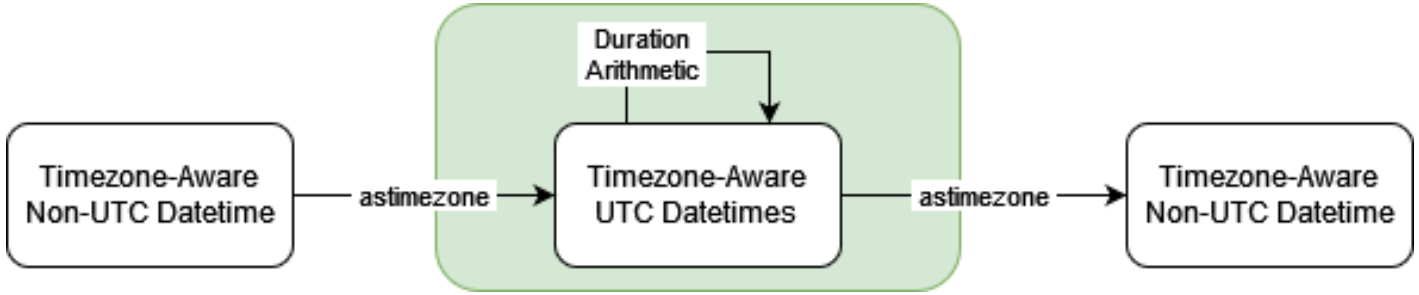

```
# Alternatively, add a timezone to a naieve datetime
tz = ZoneInfo("America/New_York")
naive_dt = datetime(2022, 4, 6, 12, 0, 0)
assert naive_dt.tzinfo is None
naive_dt.replace(tzinfo=tz).astimezone(timezone.utc)
```

Convert ISO-8601 string to UTC datetime



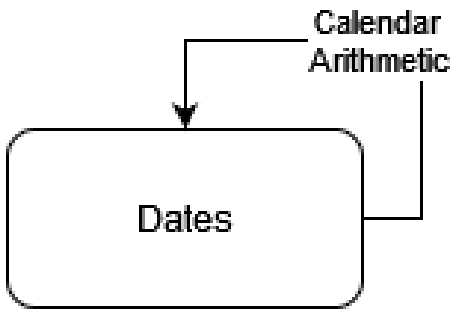

```
dt_str = "2022-03-13T03:30:00-04:00"
datetime.fromisoformat(dt_str).astimezone(timezone.utc)
>>> datetime(2022, 3, 13, 7, 30, tzinfo=<UTC>)

# Don't stop half-way
datetime.fromisoformat(dt_str)
>>> datetime(…, tzinfo=timezone(timedelta(hours=-4)))
```

Duration math on UTC datetimes




```
tz = ZoneInfo("America/New_York")
local_dt = datetime(2022, 3, 13, 1, 30, tzinfo=tz)
local_dt.isoformat()
>>> '2022-03-13T01:30:00-05:00'

new_dt = local_dt.astimezone(timezone.utc) + timedelta(hours=2)
new_dt.astimezone(tz).isoformat()
>>> '2022-03-13T04:30:00-04:00'
```

Calendar math on dates




```
tz = ZoneInfo("America/New_York")
local_dt = datetime(2022, 8, 1, 0, 0, tzinfo=tz)

# Good
local_dt.date() + timedelta(days=120)
>>> date(2022, 11, 29)

# Bad
new_dt = local_dt.astimezone(timezone.utc) + timedelta(days=120)
new_dt.astimezone(tz)
>>> datetime(2022, 11, 28, 23, 0, tzinfo='America/New_York')
```

Django

Django timezone setup

```
# settings.py
USE_TZ = True  # Enabled by default in Django 5+
TIME_ZONE = "UTC"

# code.py
from django.utils import timezone
timezone.activate("America/New_York")
timezone.get_current_timezone()
>>> <DstTzInfo 'America/New_York' ...>
```

Templates automatically convert to localtime

```
from django.template import Template, Context

timezone.activate("America/New_York")
dt = datetime(2022, 4, 6, 2, 0, tzinfo=timezone.utc)

Template("{{ dt }}").render(Context({"dt": dt}))
>>> 'April 5, 2022, 10 p.m.'
```

Getting current time and date

```
from django.utils import timezone

# Get the current time in UTC
timezone.now()
>>> datetime(..., tzinfo=<UTC>)

# Get the current date in the active timezone
timezone.activate("America/New_York")
timezone.localdate()
>>> date(2022, 4, 18)
```

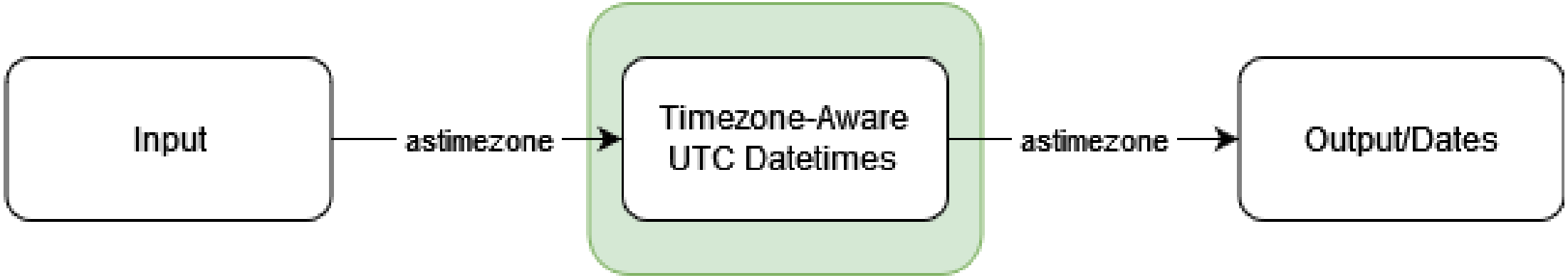
Converting to localtime

```
from django.utils import timezone
timezone.activate("America/New_York")
dt = datetime(2022, 4, 6, 2, 0, tzinfo=timezone.utc)

# Convert a datetime to a date in a timezone
timezone.localdate(dt)
>>> date(2022, 4, 5)

# Display a UTC time as localtime (i.e. output)
timezone.localtime(dt).strftime("%x %X")
>>> '04/05/22 22:00:00'
```

Attaching a timezone to a naïve datetime

```
from django.utils import timezone
timezone.activate("America/New_York")
naive_dt = datetime(2022, 4, 6, 12, 0, 0)

# Convert naive datetime to UTC (i.e. input)
# The `is_dst` flag is deprecated in Django 4
local_dt = timezone.make_aware(naive_dt, is_dst=False)
local_dt.astimezone(timezone.utc)
>>> datetime.datetime(2022, 4, 6, 16, 0, tzinfo=<UTC>)
```

Summary


Input
astimezone
Timezone-Aware
UTC Datetimes
astimezone
Output/Dates

Thank You

Thank you to Paul Ganssle

If you have questions for me or need contract software work:

Benjamin “Zags” Zagorsky
zags@zagaran.com

Bonus Material

ZoneInfo and DST

```
tz = ZoneInfo("America/New_York")

# Spring Forwards
dt = datetime(2022, 3, 13, 2, 30, tzinfo=tz)
dt.astimezone(timezone.utc).astimezone(tz)
>>> datetime(2022, 3, 13, 3, 30, tzinfo='America/New_York')

# Fall back
datetime(2022, 11, 6, 1, 30, tzinfo=tz)  # First occurence
datetime(2022, 11, 6, 1, 30, tzinfo=tz, fold=1)  # Second occurence
```

Leap Seconds

```
# Python ignores leap seconds and relies on NTP to update server time
datetime(2016, 12, 31, 23, 59, 59, 0, timezone.utc) + timedelta(seconds=1)
>>> datetime(2017, 1, 1, 0, 0, tzinfo=timezone.utc)
```

Zagaran Software | zags@zagaran.com

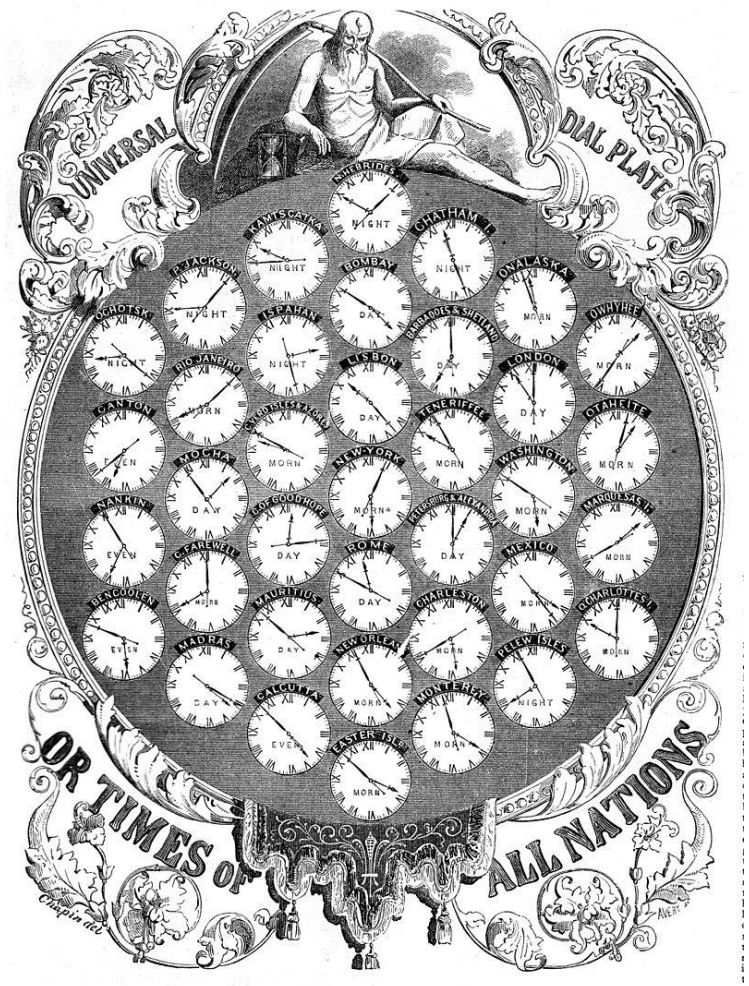
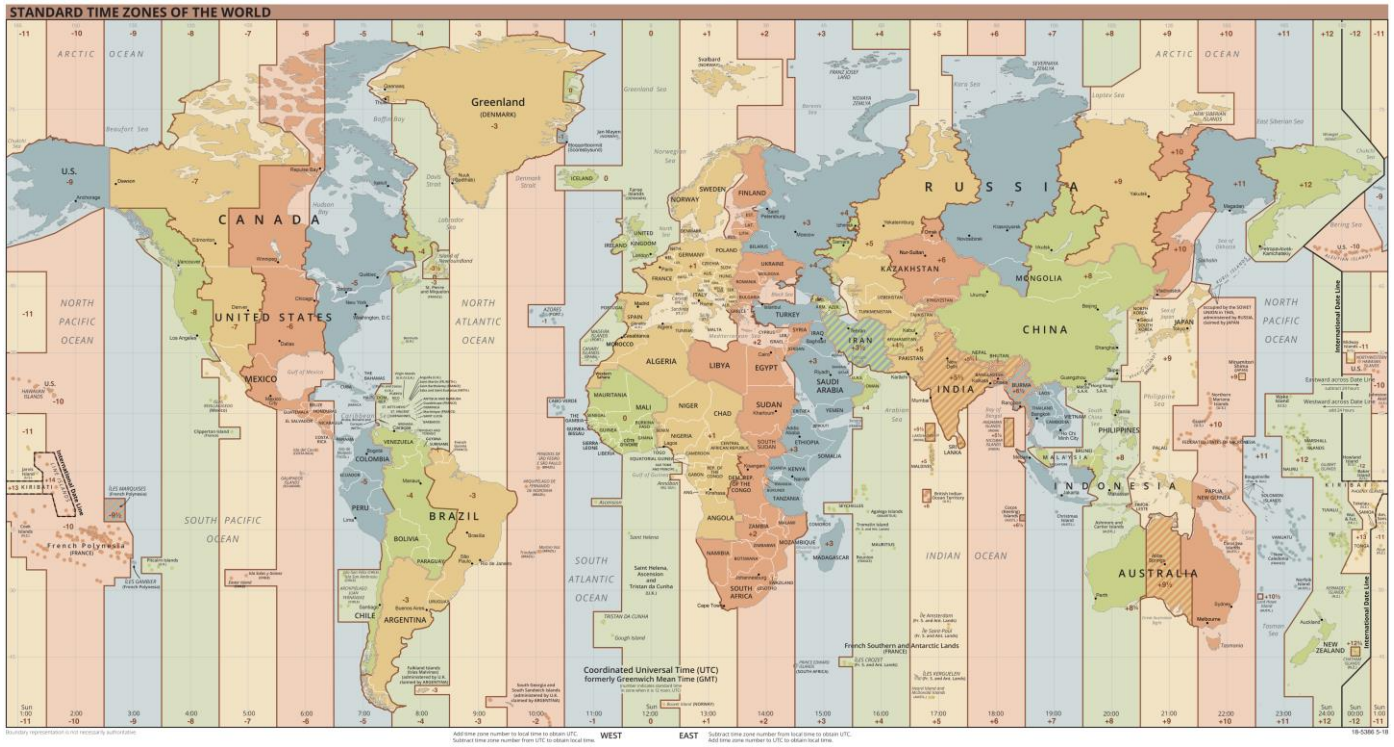
Using Local System Time

```
datetime(2022, 4, 6, 12, 0, tzinfo=timezone.utc).astimezone()
>>> datetime(2022, 4, 6, 8, 0,
    tzinfo=timezone(timedelta(hours=-4), 'EDT'))

datetime(2022, 4, 6, 12, 0, tzinfo=timezone.utc).astimezone().tzname()
>>> 'EDT'
```

What if you don’t know the offset?

- Libraries may return naïve datetimes
- Third-party APIs may send you ISO-8601 strings without a UTC offset

Things to do:

- Check the docs
- Run some experiments
- Comment your code

UTC: before and after