

Programming with Dates and Times in Java 8

INTRODUCTION AND OVERVIEW OF JAVA.TIME



Maurice Naftalin

@mauricenaftalin

Introduction and Overview of java.time



Maurice Naftalin
@mauricensaftalin

Summary

Summary

Summary

Why we need an API for date and time

Summary

Why we need an API for date and time

Domain assumptions behind `java.time`

Summary

Why we need an API for date and time

Domain assumptions behind `java.time`

- Human time vs. Machine time

Summary

Why we need an API for date and time

Domain assumptions behind `java.time`

- Human time vs. Machine time

Overview of the Java date/time API

Summary

Why we need an API for date and time

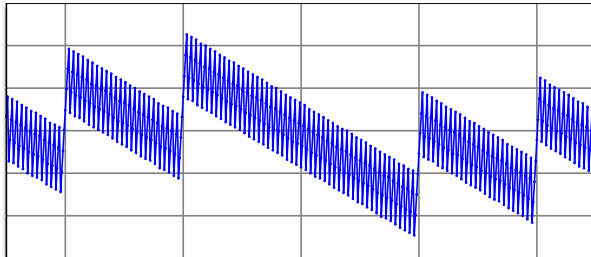
Domain assumptions behind `java.time`

- Human time vs. Machine time

Overview of the Java date/time API

- Design Goals, Core Classes

What's so Hard About Dates and Times?



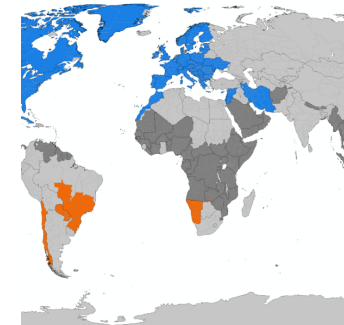
Leap Years

The solar year is
about 365.242 days
long



Time Zones

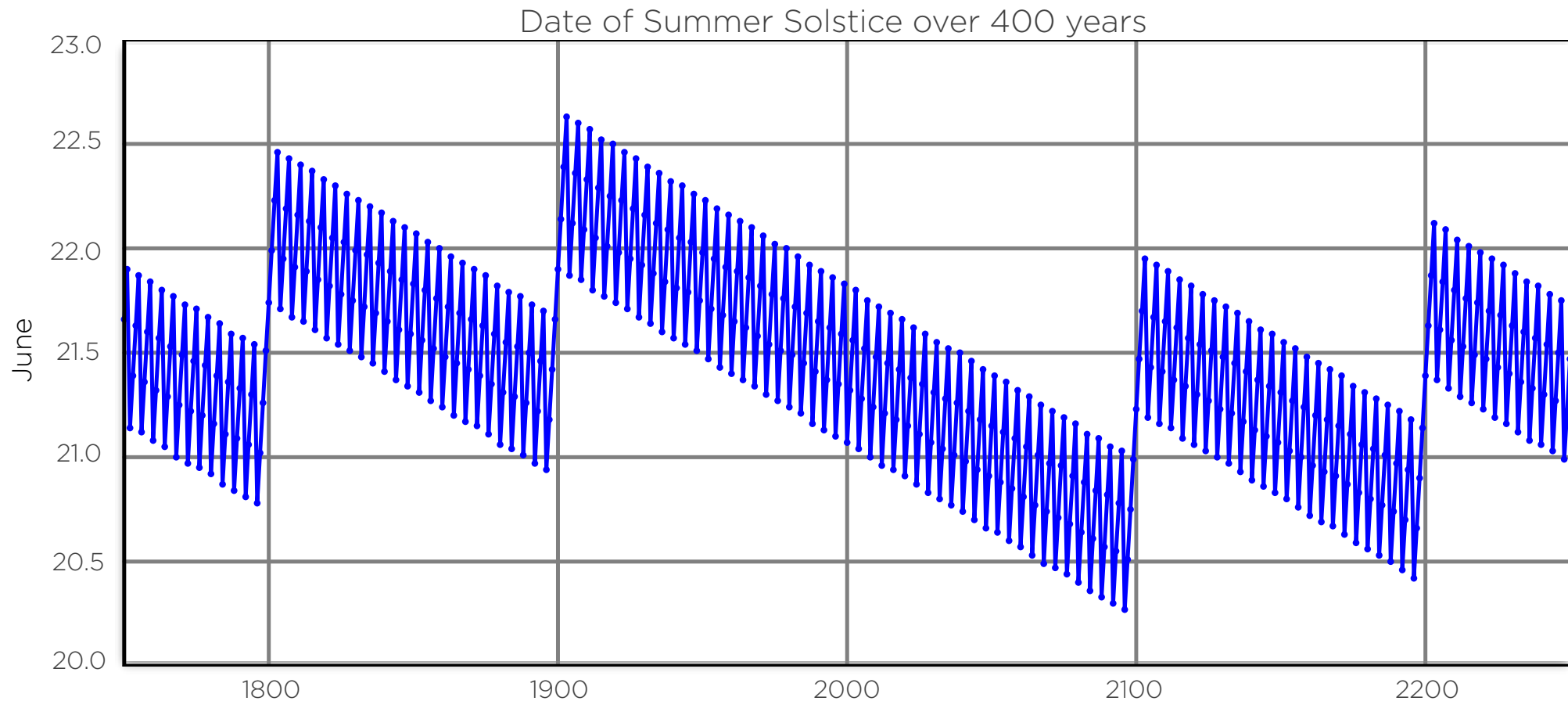
Countries or regions
need a uniform
standard time



Daylight Saving Time

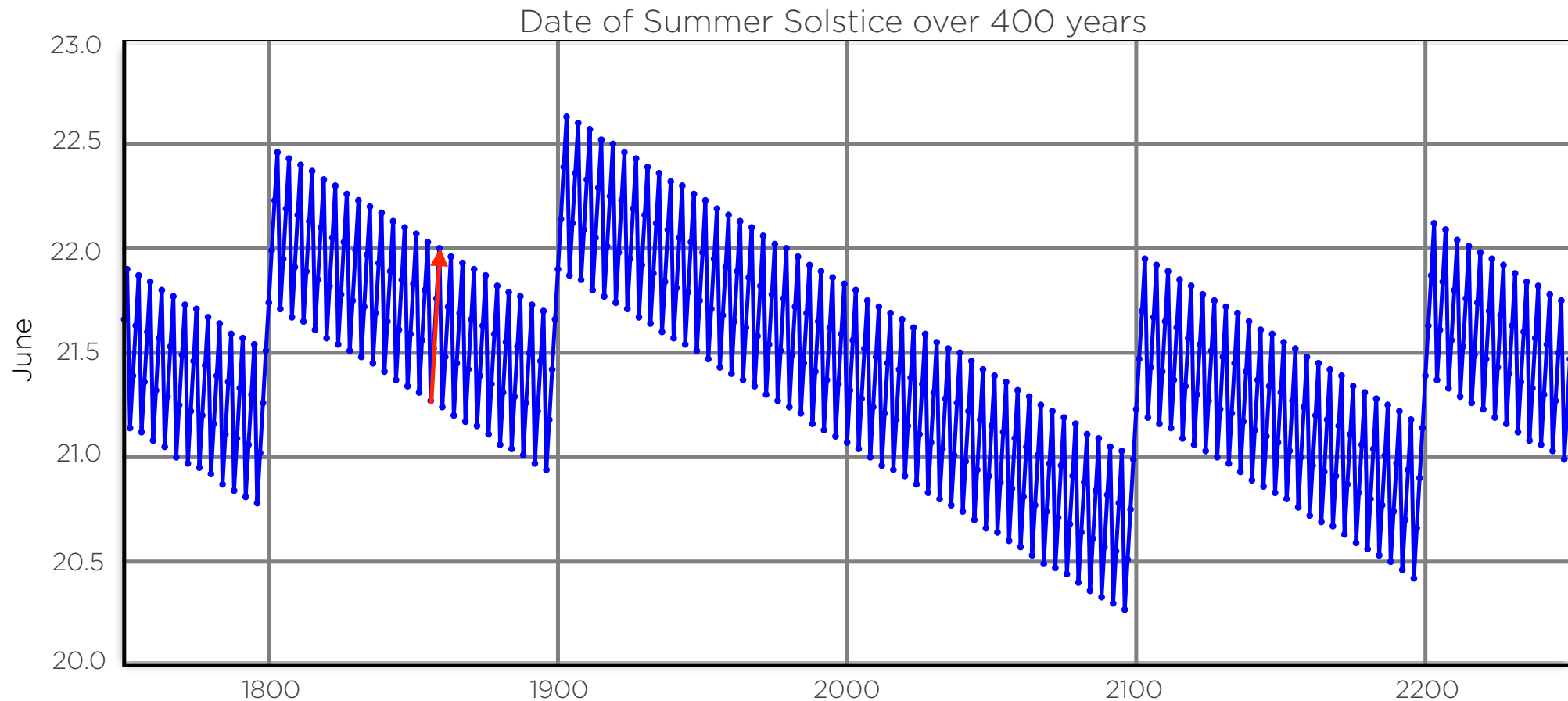
People like to match
their waking time to
the hours of daylight

What's so Hard About Dates and Times?



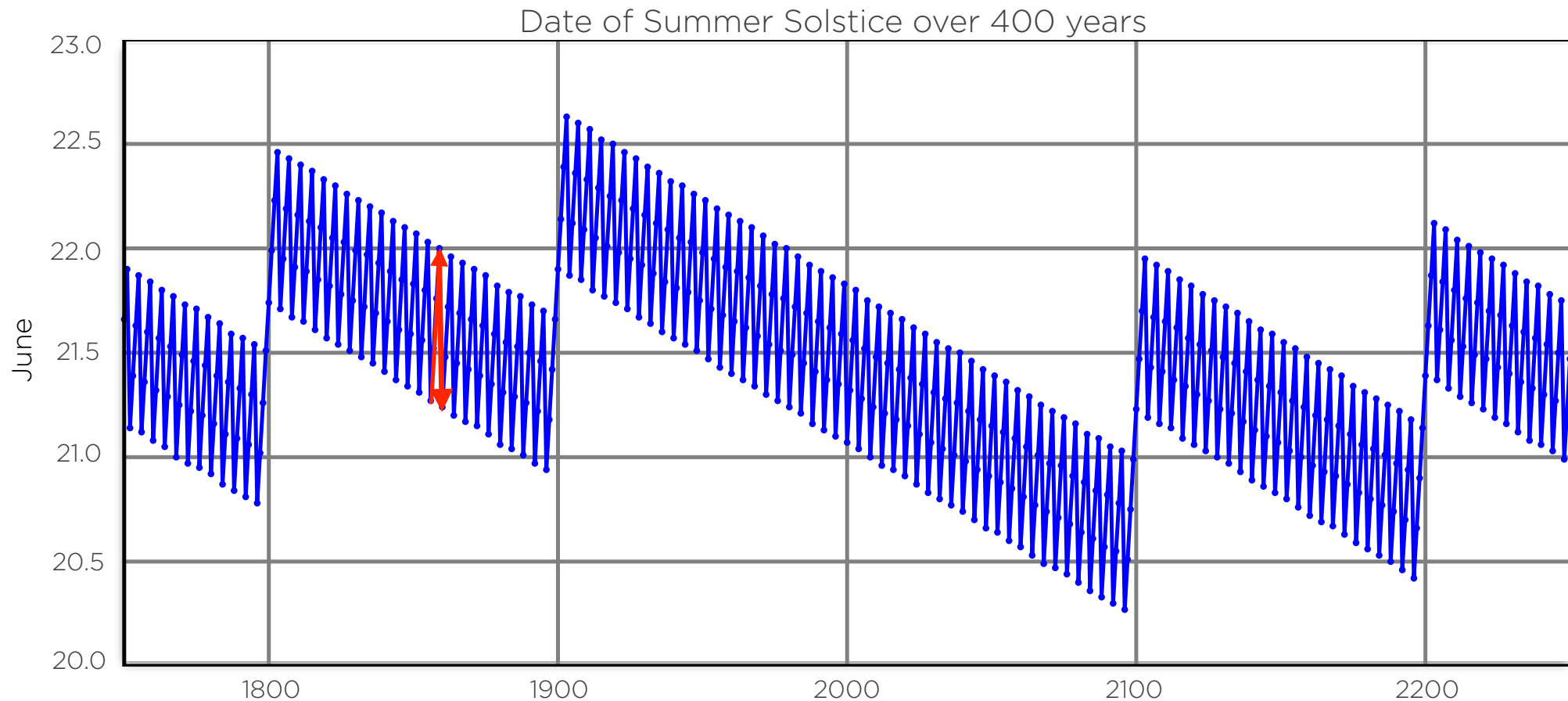
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



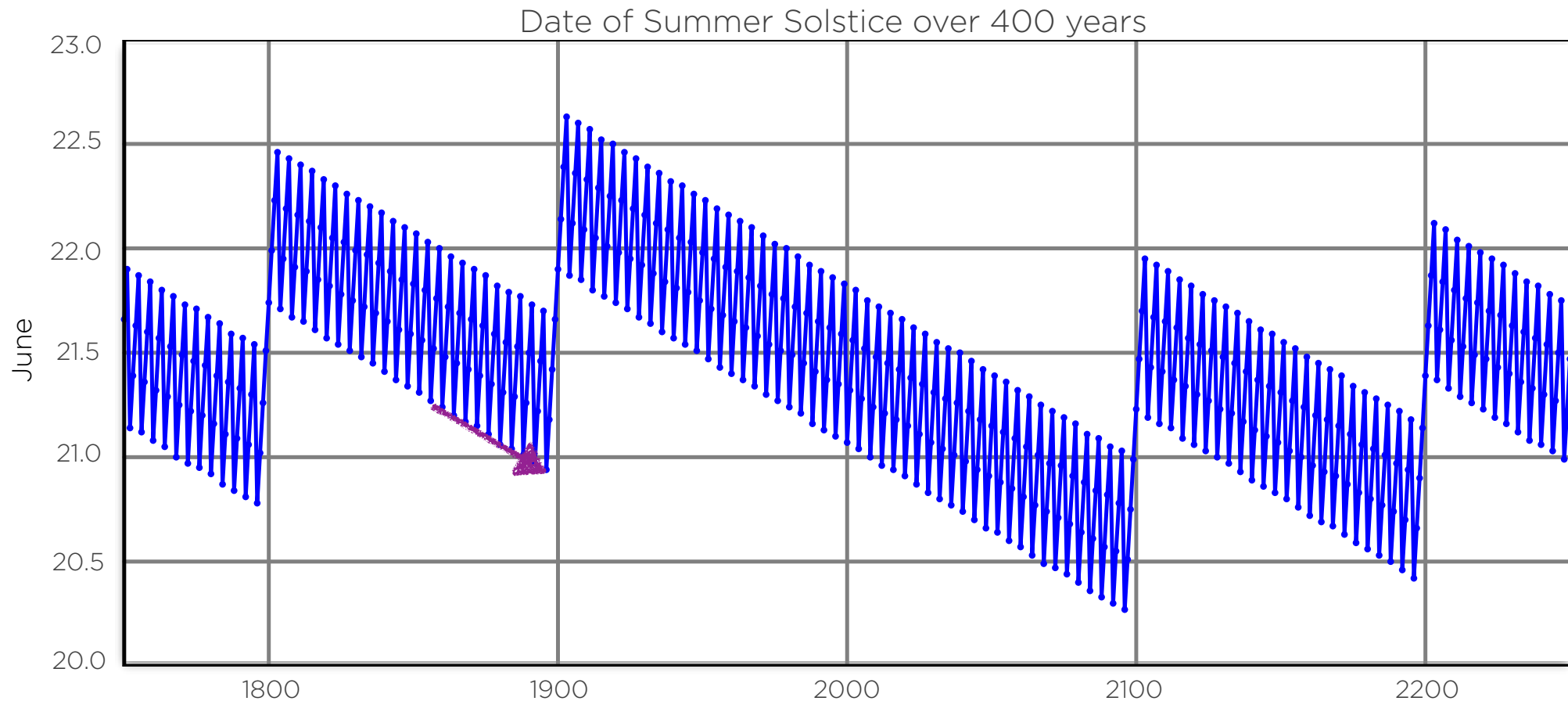
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



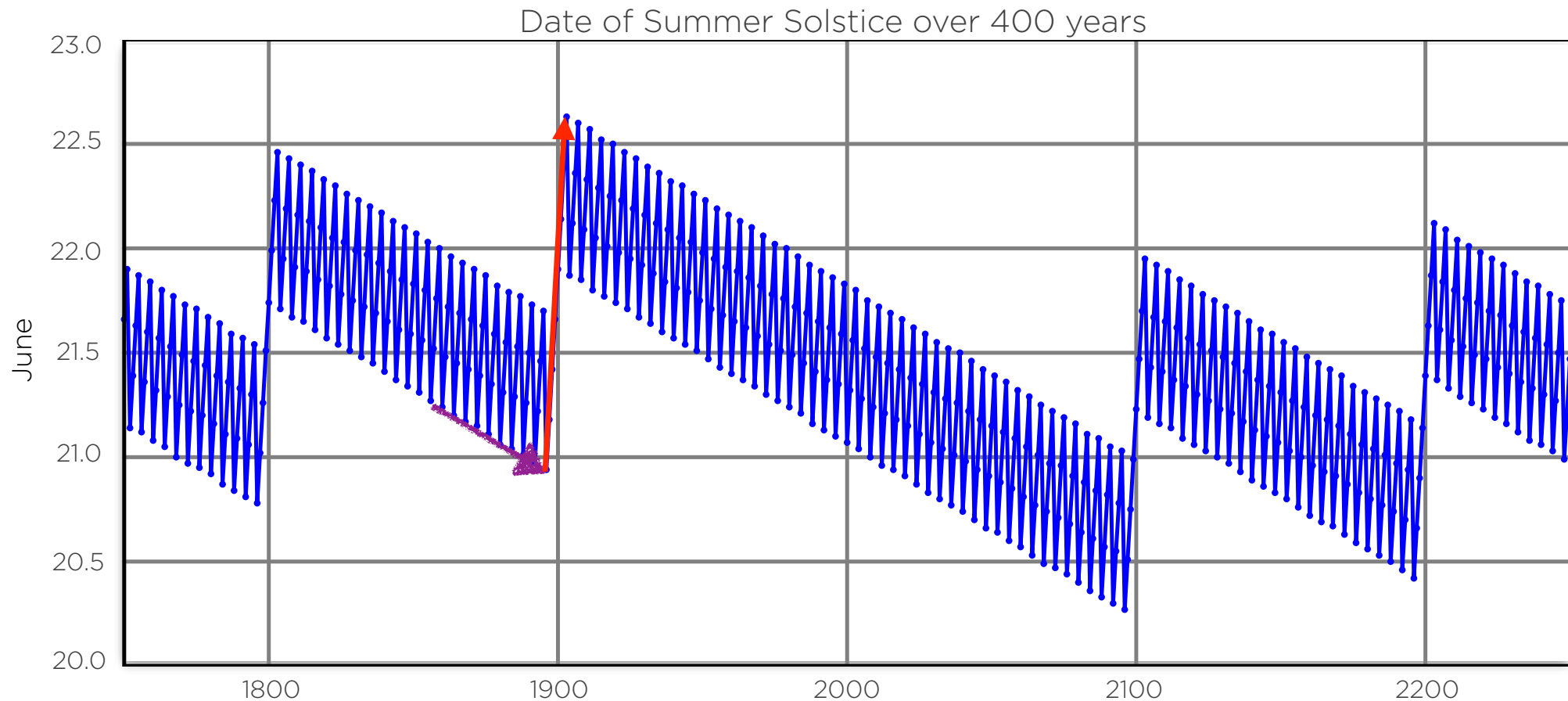
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



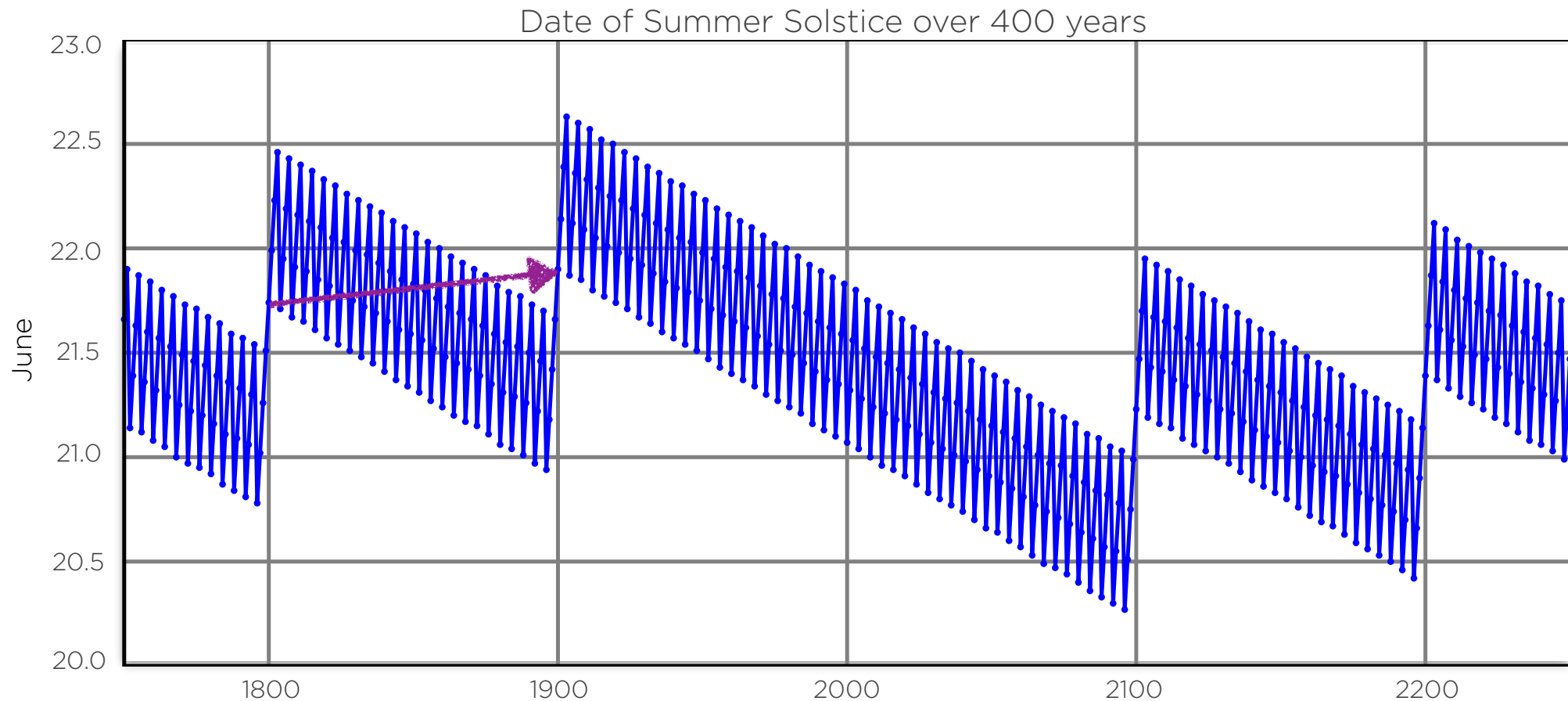
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



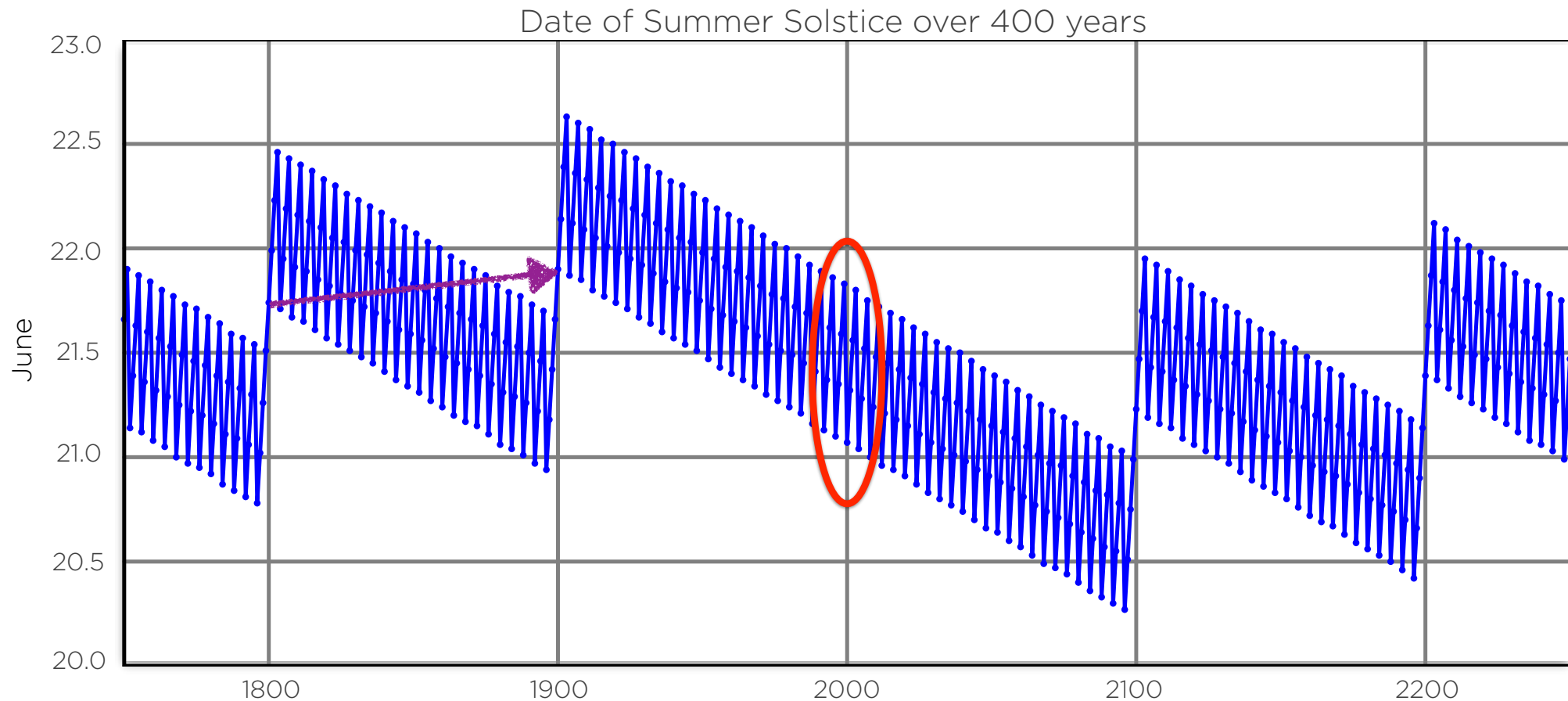
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



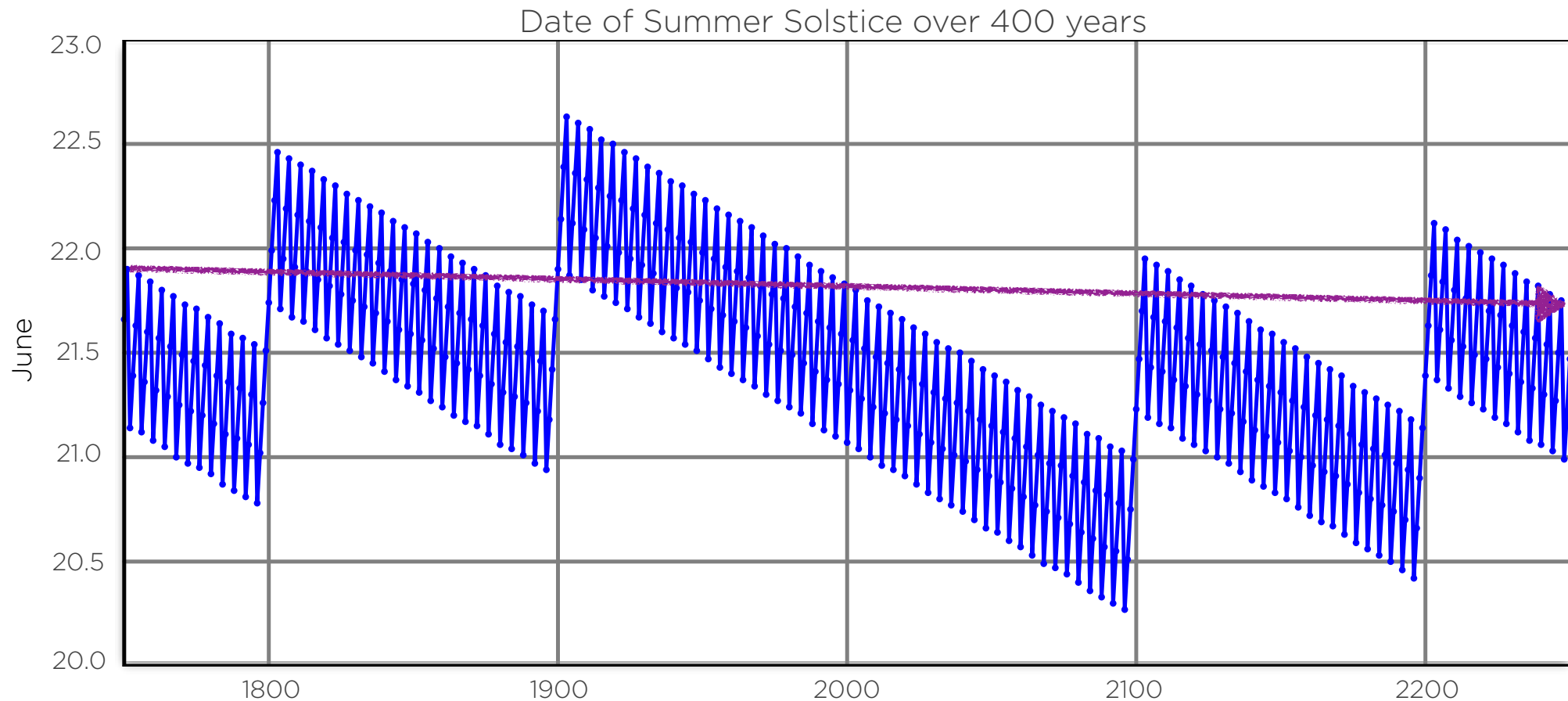
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



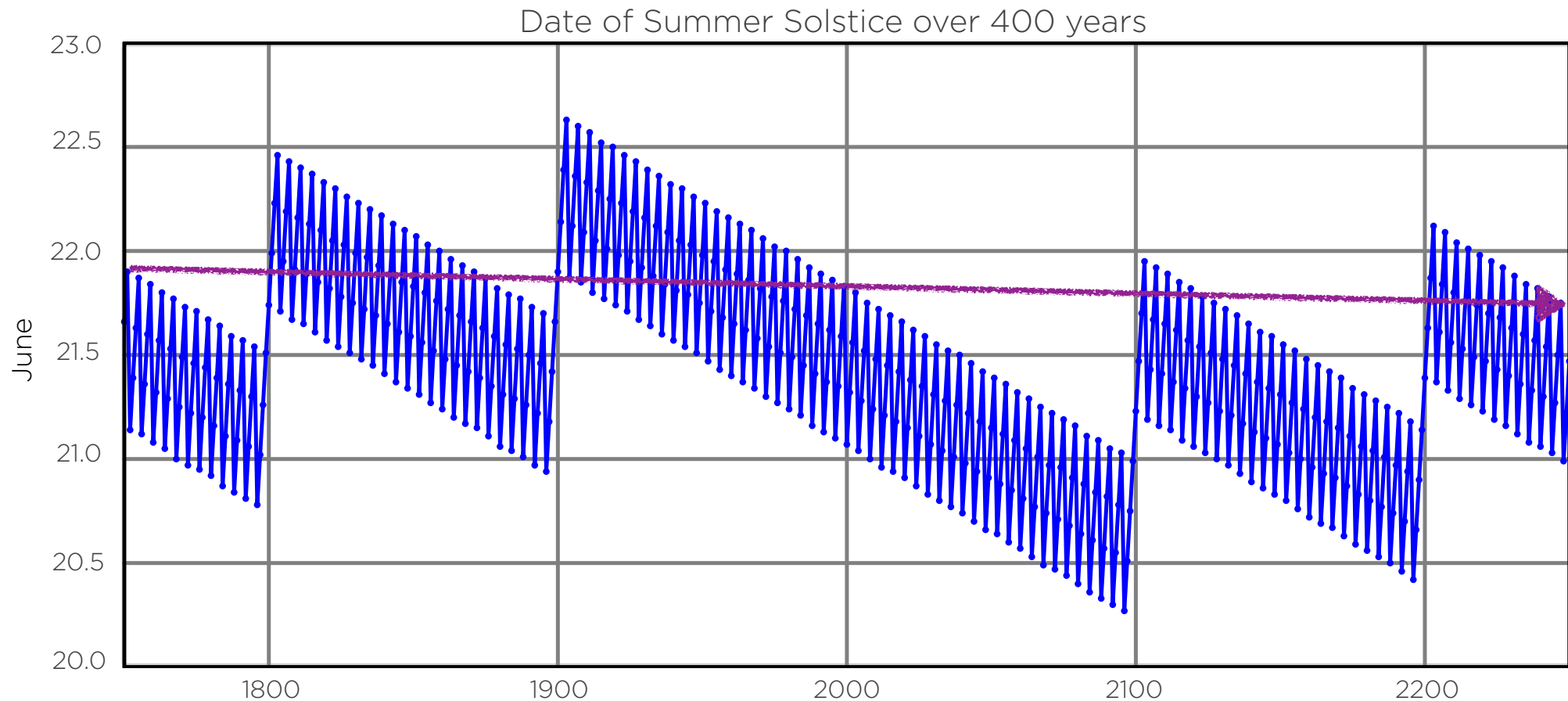
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



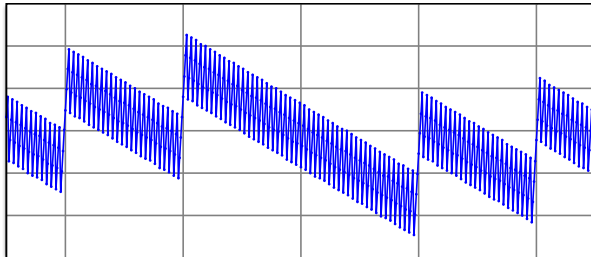
By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



By BasZoetekouw; spelling corrections and revision of subtitle by User:Gerry Ashton on 14 September 2008. - Own work; The data was generated by Astrolabe ([http://astrolabe.sourceforge.net/\[dead link\]](http://astrolabe.sourceforge.net/[dead link])), which uses the algorithms described in Jean Meeus's "Astronomical Algorithms" (ISBN 978-0943396613). The data have an error of less than 2.6 minutes., CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=4730687>

What's so Hard About Dates and Times?



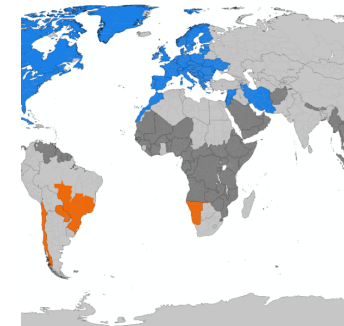
Leap Years

The solar year is about 365.242 days long



Time Zones

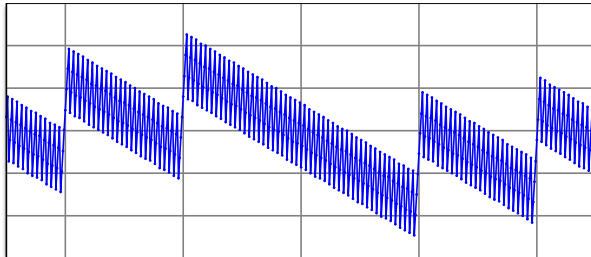
Countries or regions need a uniform standard time



Daylight Saving Time

People like to match their waking time to the hours of daylight

What's so Hard About Dates and Times?



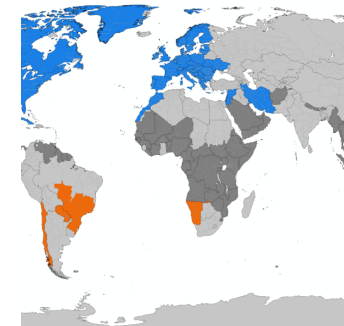
Leap Years

The solar year is about 365.242 days long



Time Zones

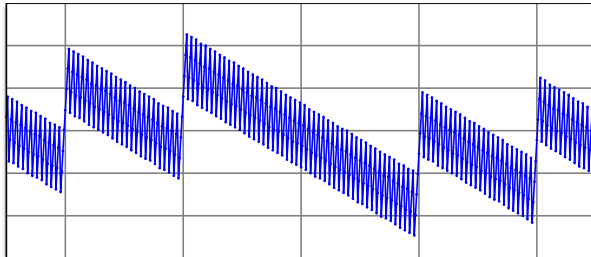
Countries or regions need a uniform standard time



Daylight Saving Time

People like to match their waking time to the hours of daylight

What's so Hard About Dates and Times?



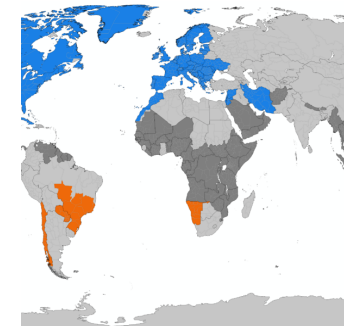
Leap Years

The solar year is
about 365.242 days
long



Time Zones

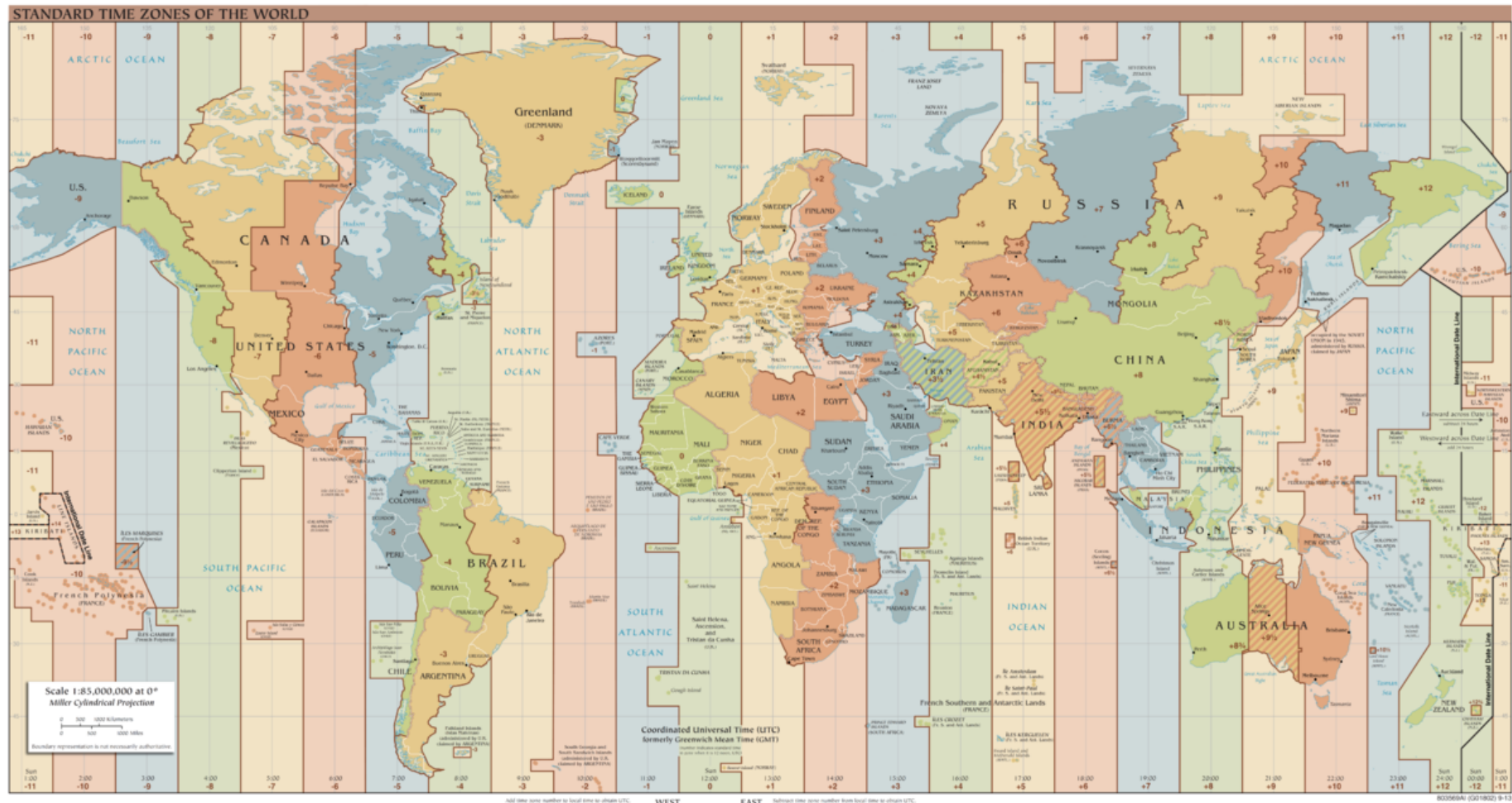
Countries or regions
need a uniform
standard time



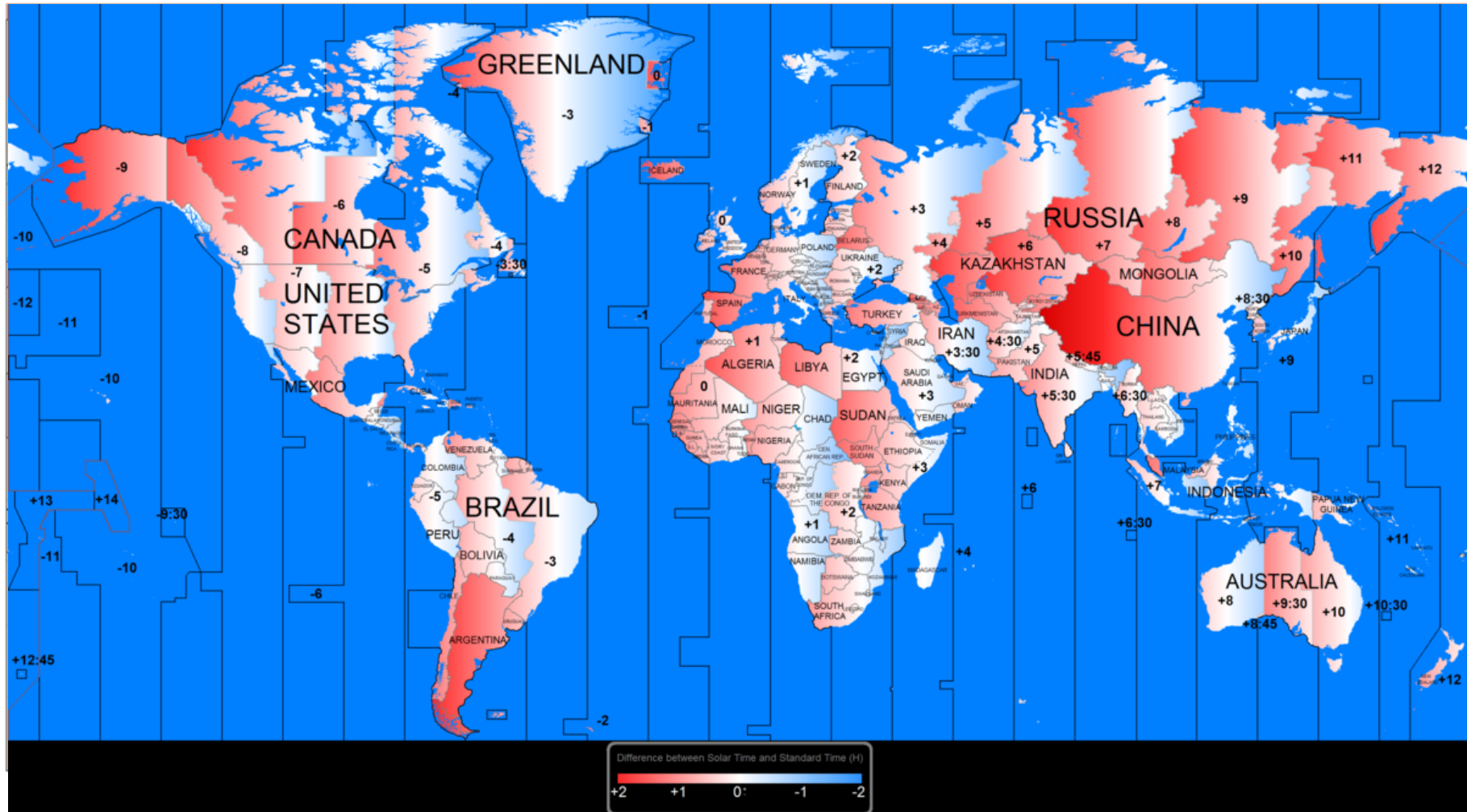
Daylight Saving Time

People like to match
their waking time to
the hours of daylight

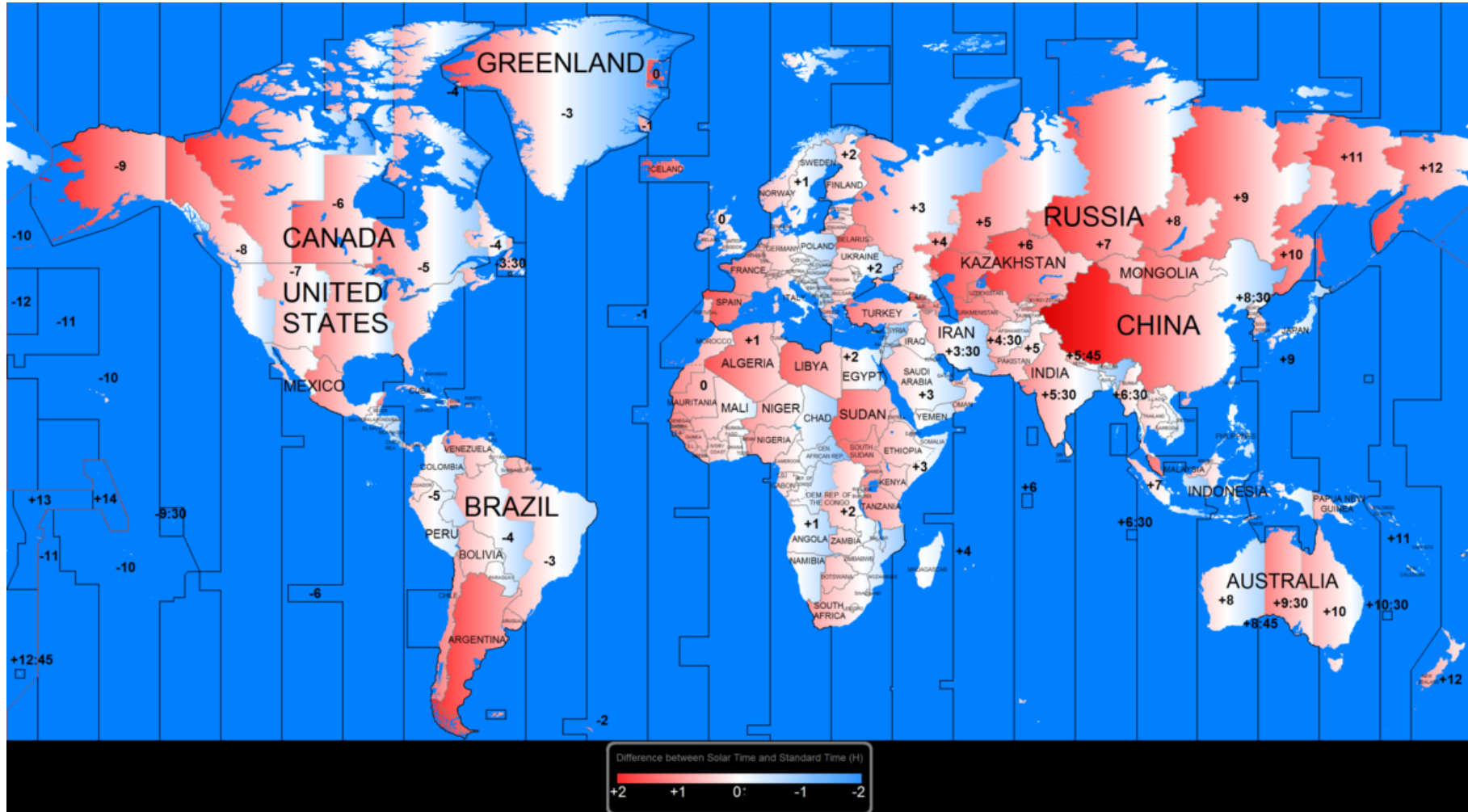
What's so Hard About Dates and Times?



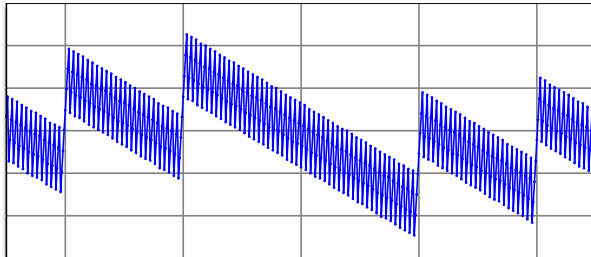
What's so Hard About Dates and Times?



What's so Hard About Dates and Times?

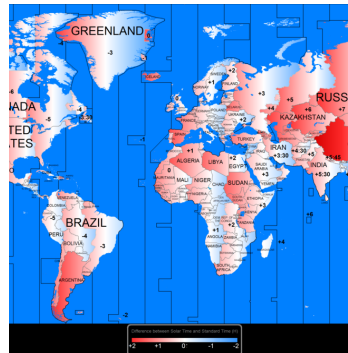


What's so Hard About Dates and Times?



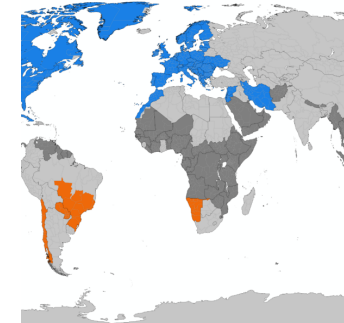
Leap Years

The solar year is
about 365.242 days
long



Time Zones

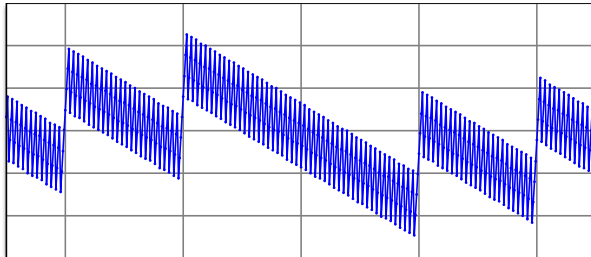
Countries or regions
need a uniform
standard time



Daylight Saving Time

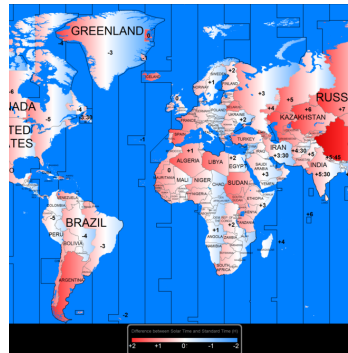
People like to match
their waking time to
the hours of daylight

What's so Hard About Dates and Times?



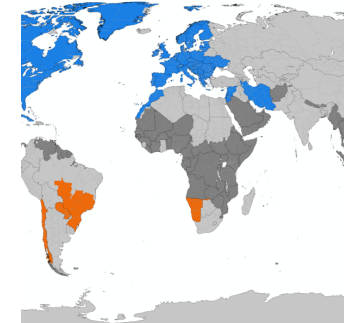
Leap Years

The solar year is
about 365.242 days
long



Time Zones

Countries or regions
need a uniform
standard time

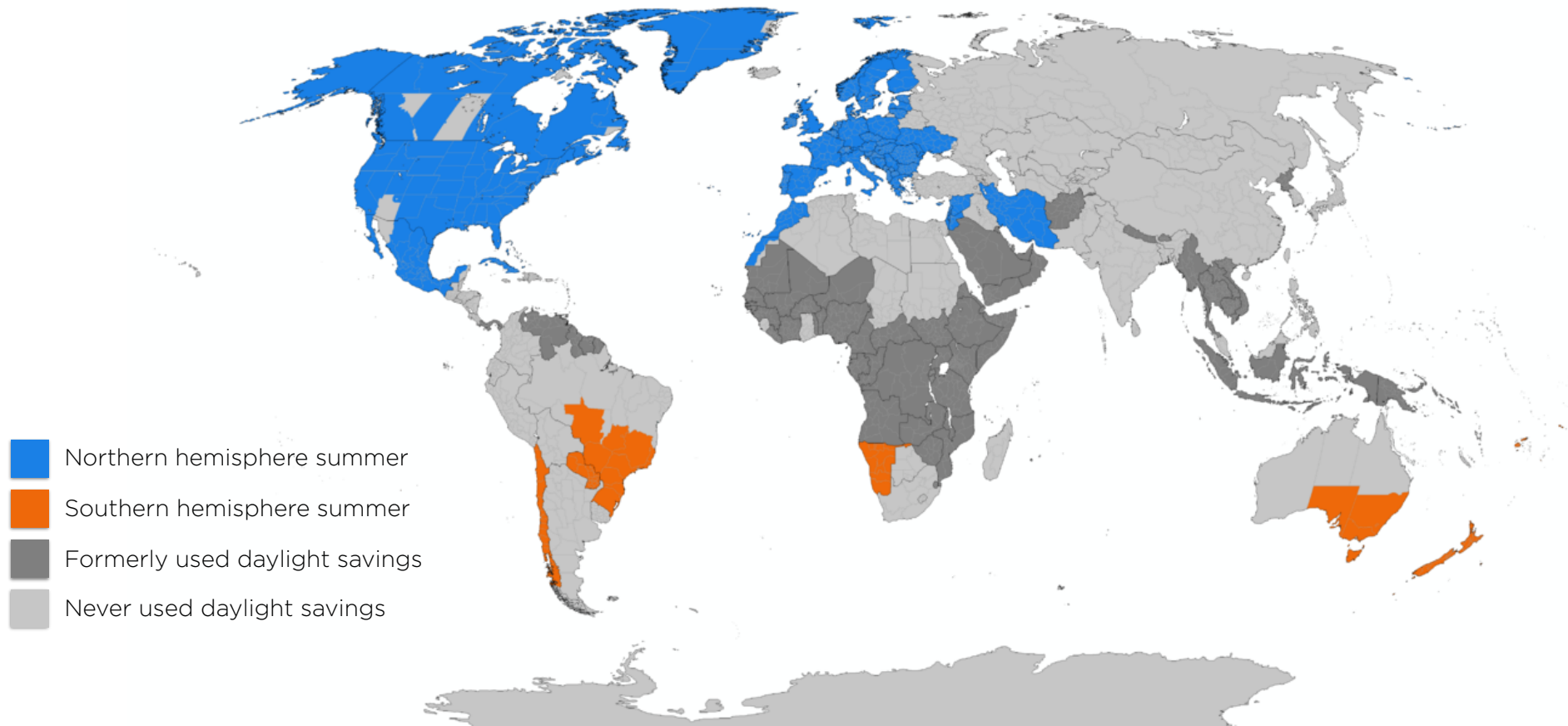


Daylight Saving Time

People like to match
their waking time to
the hours of daylight

What's so Hard About Dates and Times?

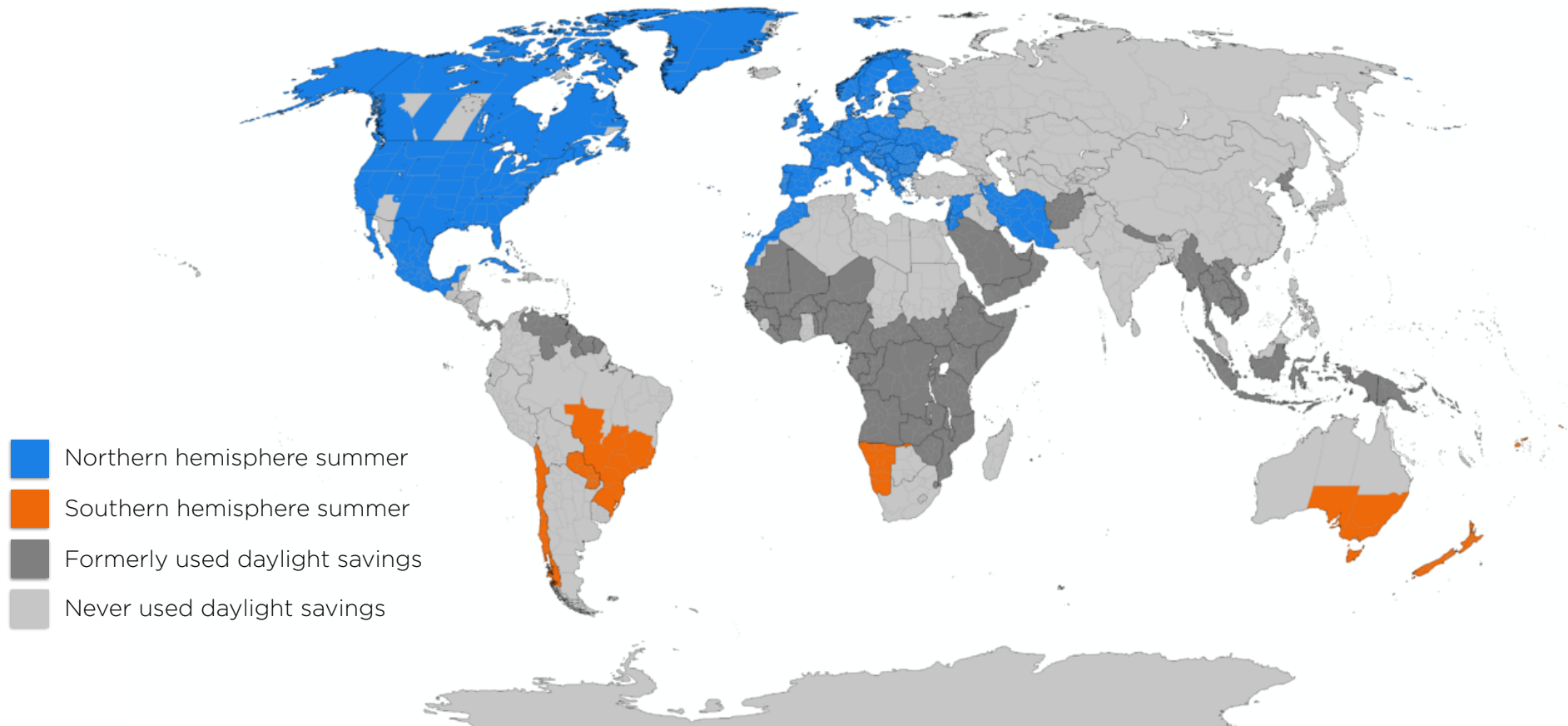
World daylight saving time regions



By TimeZonesBoy - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=17593495>

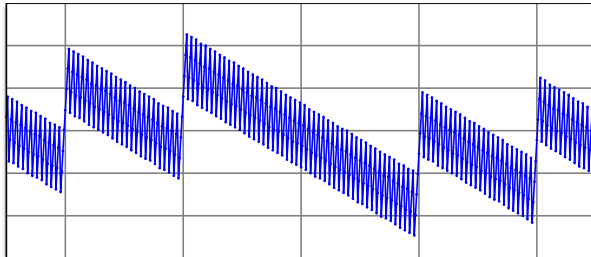
What's so Hard About Dates and Times?

World daylight saving time regions



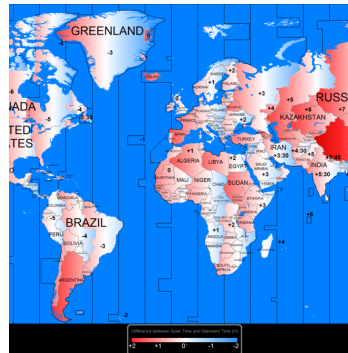
By TimeZonesBoy - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=17593495>

What's so Hard About Dates and Times?



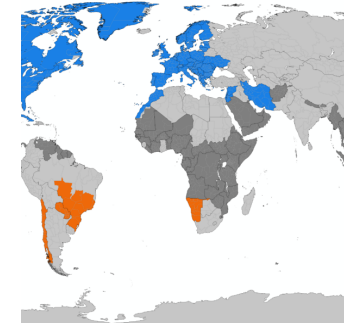
Leap Years

Some information
goes here; three lines
or fewer is best



Time Zones

Some information
goes here; three lines
or fewer is best



Daylight Saving Time

Some information
goes here; three lines
or fewer is best

Java Date and Time

Machine/Solar Time

Java Date and Time

Machine/Solar Time



Java Date and Time

Machine/Solar Time

Instant.EPOCH



Java Date and Time

Machine/Solar Time

`Instant.EPOCH`



`Instant.ofEpochSecond(9961199)`

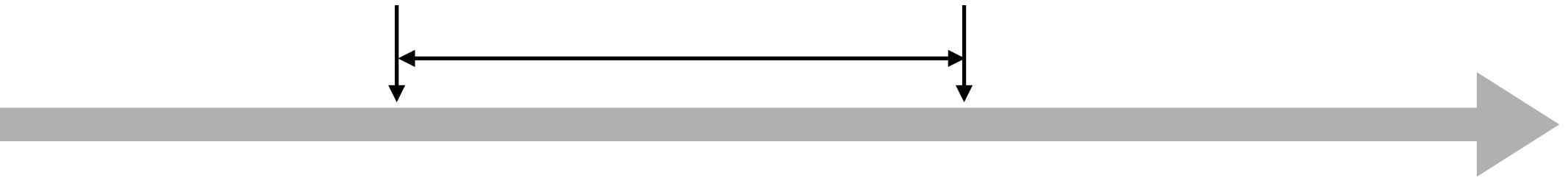


Java Date and Time

Machine/Solar Time

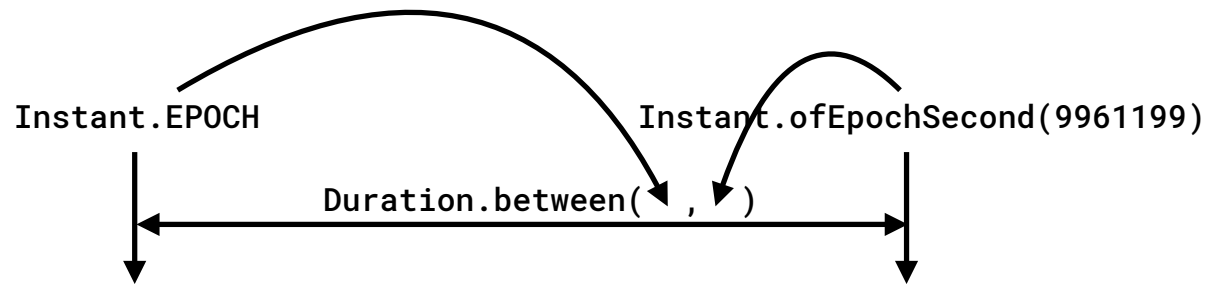
`Instant.EPOCH`

`Instant.ofEpochSecond(9961199)`



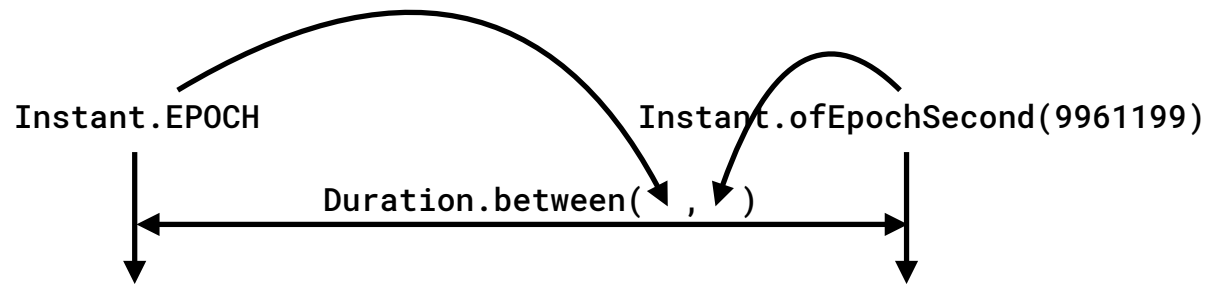
Java Date and Time

Machine/Solar Time



Java Date and Time

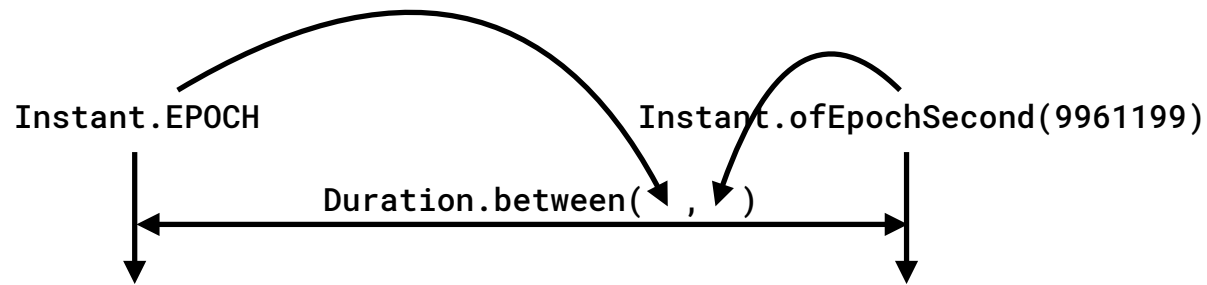
Machine/Solar Time



UTC

Java Date and Time

Machine/Solar Time

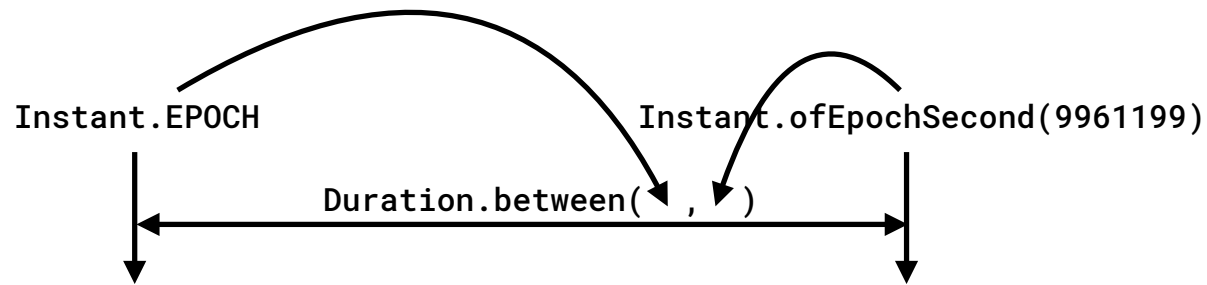


UTC

1970-01-01T00:00:00Z

Java Date and Time

Machine/Solar Time



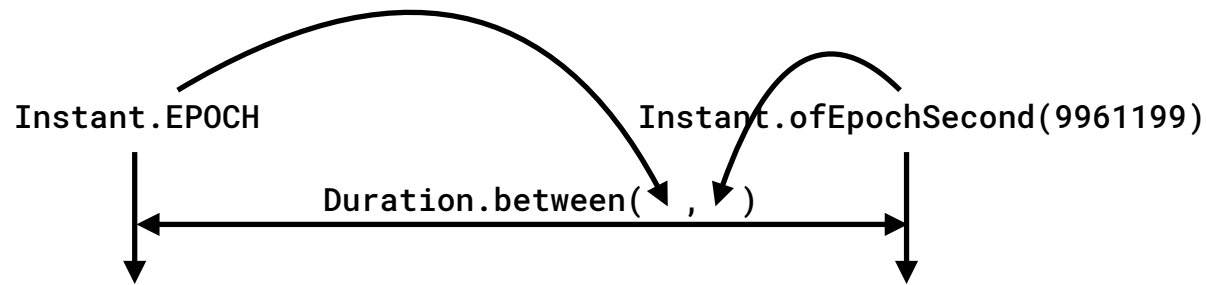
UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

Java Date and Time

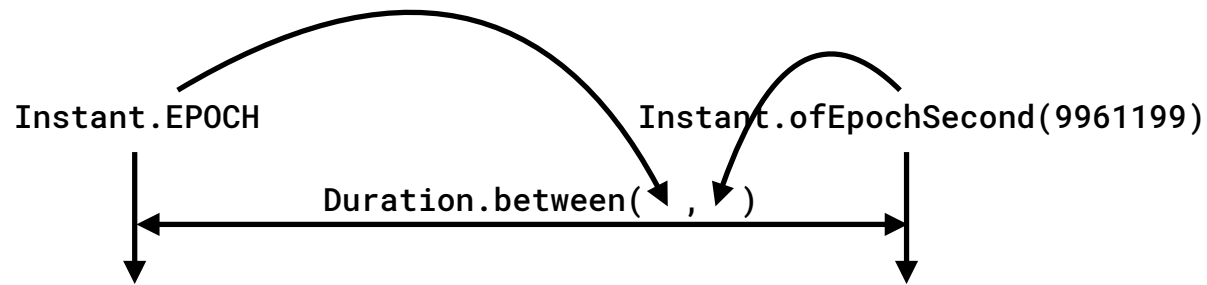
Machine/Solar Time



UTC 1970-01-01T00:00:00Z 1970-04-26T06:59:59Z

Java Date and Time

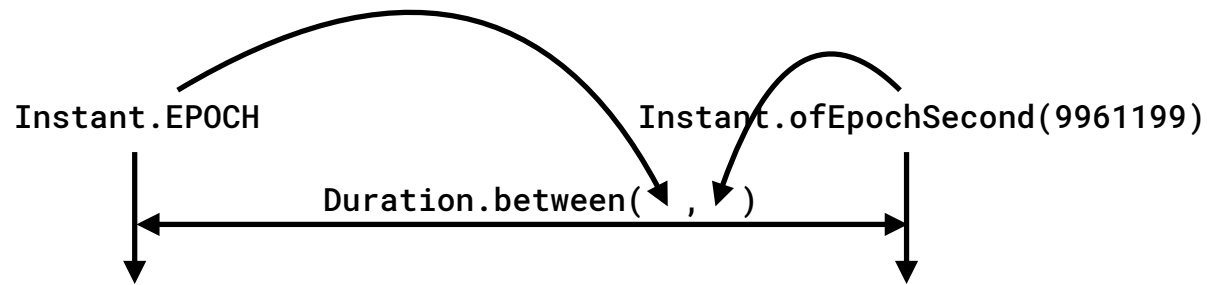
Machine/Solar Time



UTC 1970-01-01T00:00:00Z 1970-04-26T06:59:59Z

Java Date and Time

Machine/Solar Time



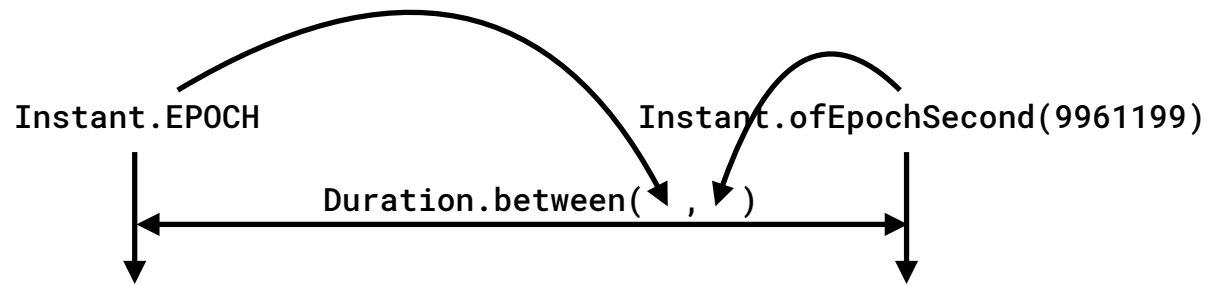
UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

Java Date and Time

Machine/Solar Time



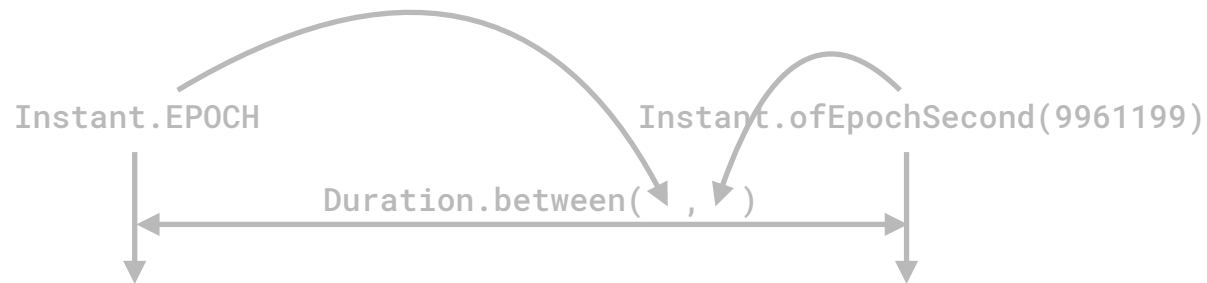
UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

Java Date and Time

Machine/Solar Time



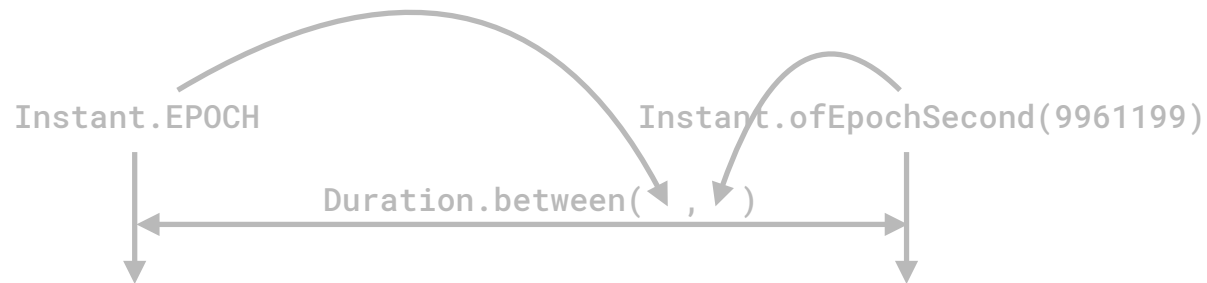
UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

Java Date and Time

Machine/Solar Time

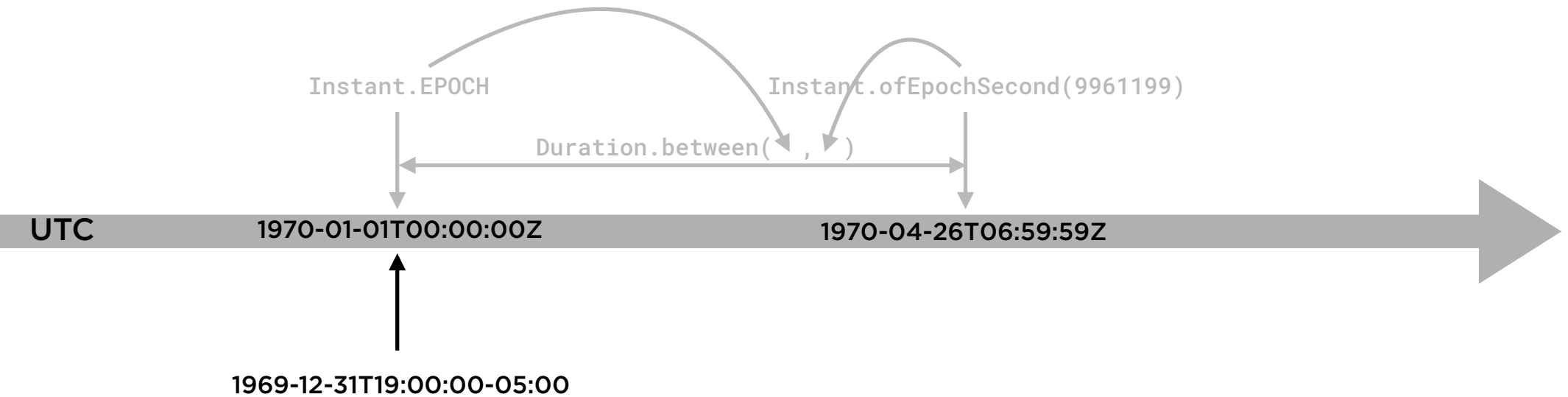


UTC 1970-01-01T00:00:00Z 1970-04-26T06:59:59Z

Human Time

Java Date and Time

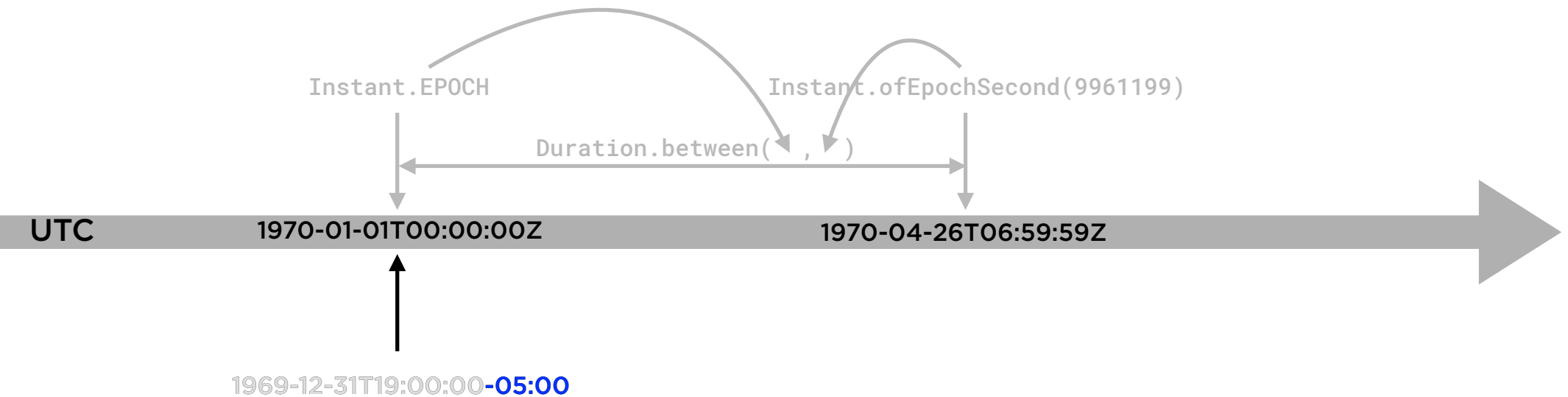
Machine/Solar Time



Human Time

Java Date and Time

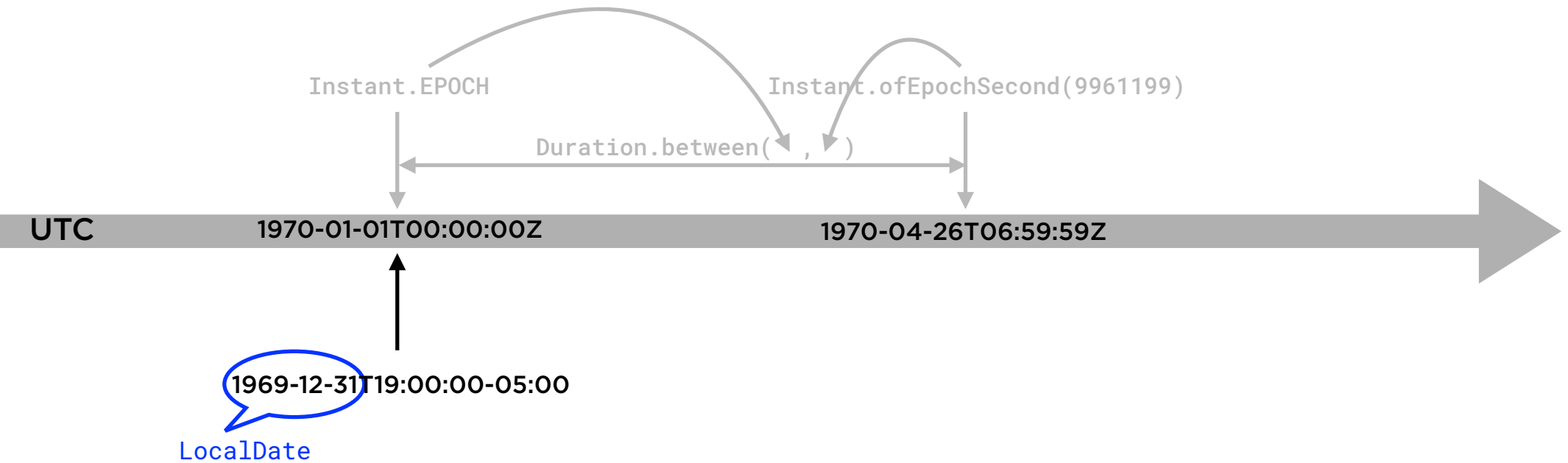
Machine/Solar Time



Human Time

Java Date and Time

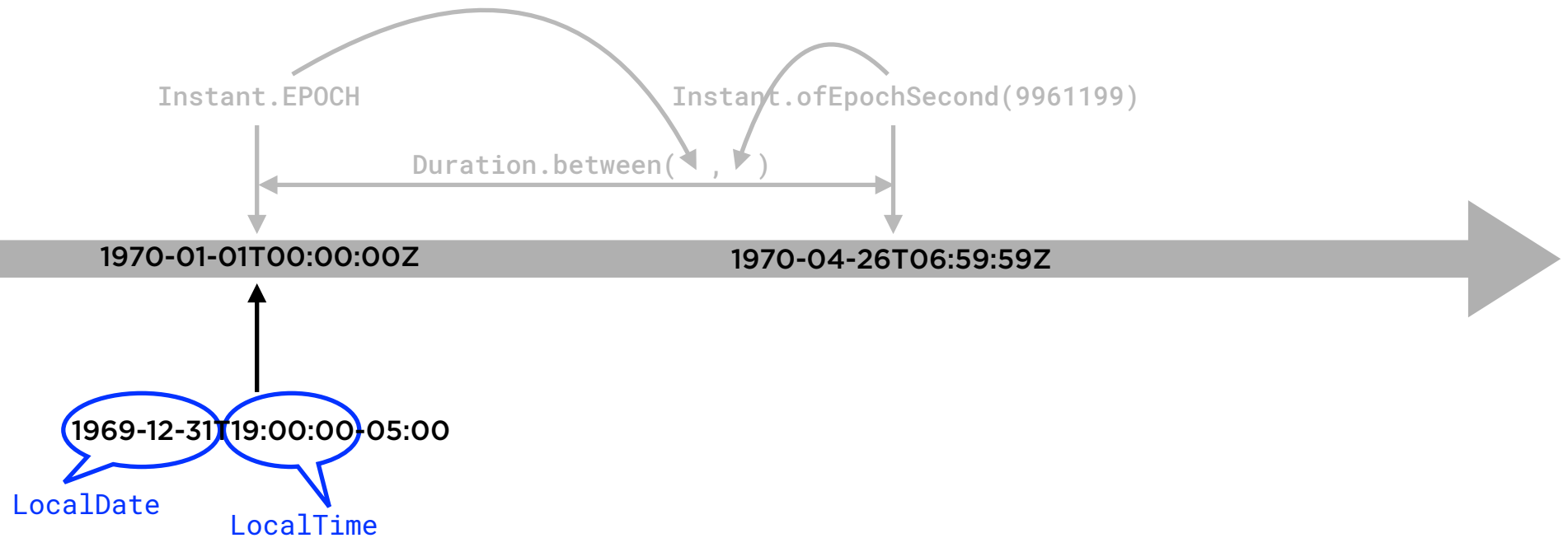
Machine/Solar Time



Human Time

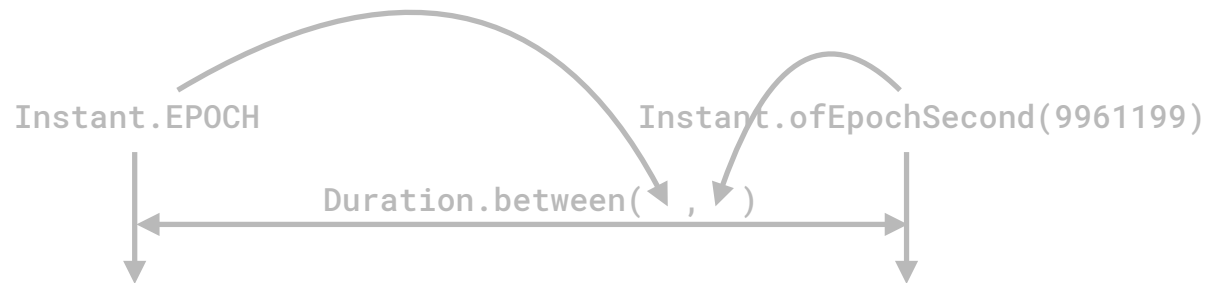
Java Date and Time

Machine/Solar Time



Java Date and Time

Machine/Solar Time



UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

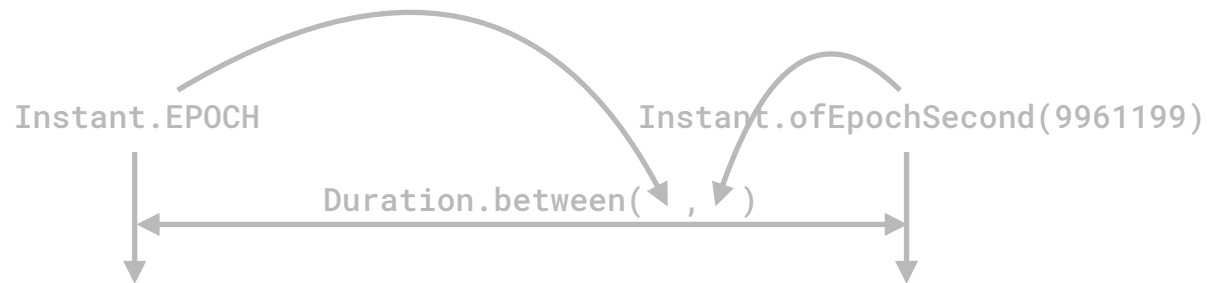
1969-12-31T19:00:00-05:00

`LocalDateTime`

Human Time

Java Date and Time

Machine/Solar Time



UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

1969-12-31T19:00:00-05:00

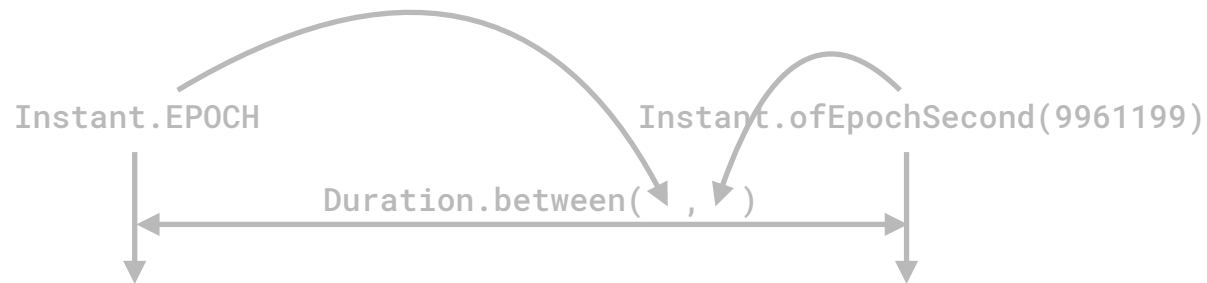
ZoneOffset

LocalDateTime

Human Time

Java Date and Time

Machine/Solar Time



UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

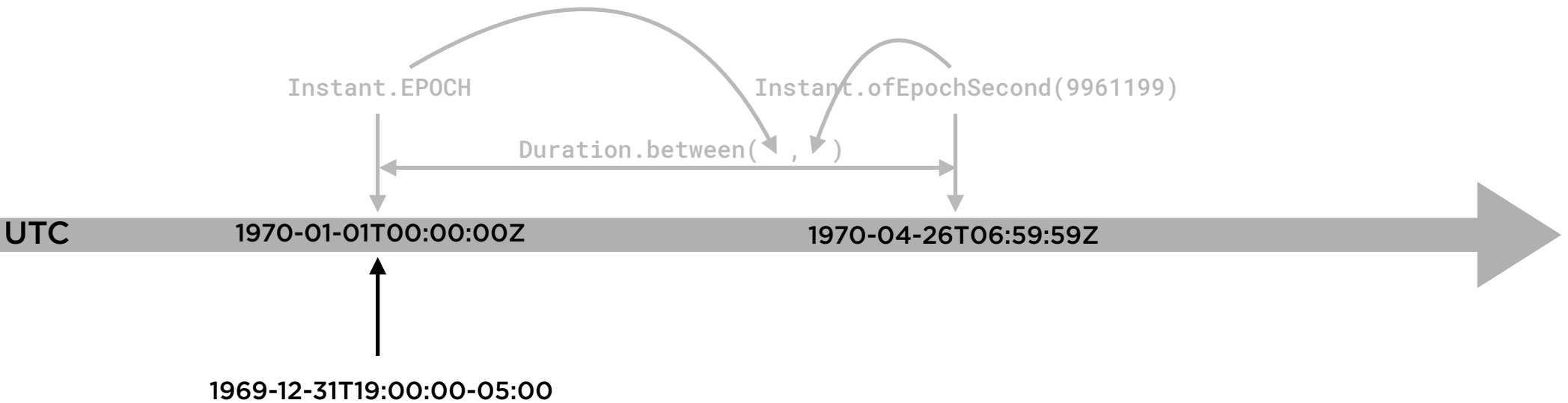
1969-12-31T19:00:00-05:00

ZonedDateTime

Human Time

Java Date and Time

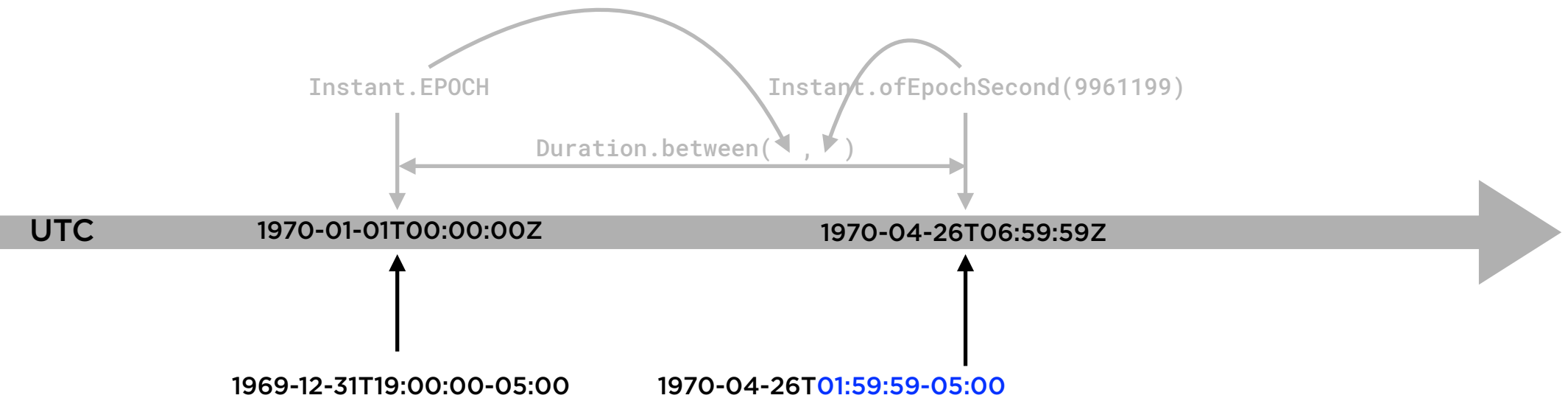
Machine/Solar Time



Human Time

Java Date and Time

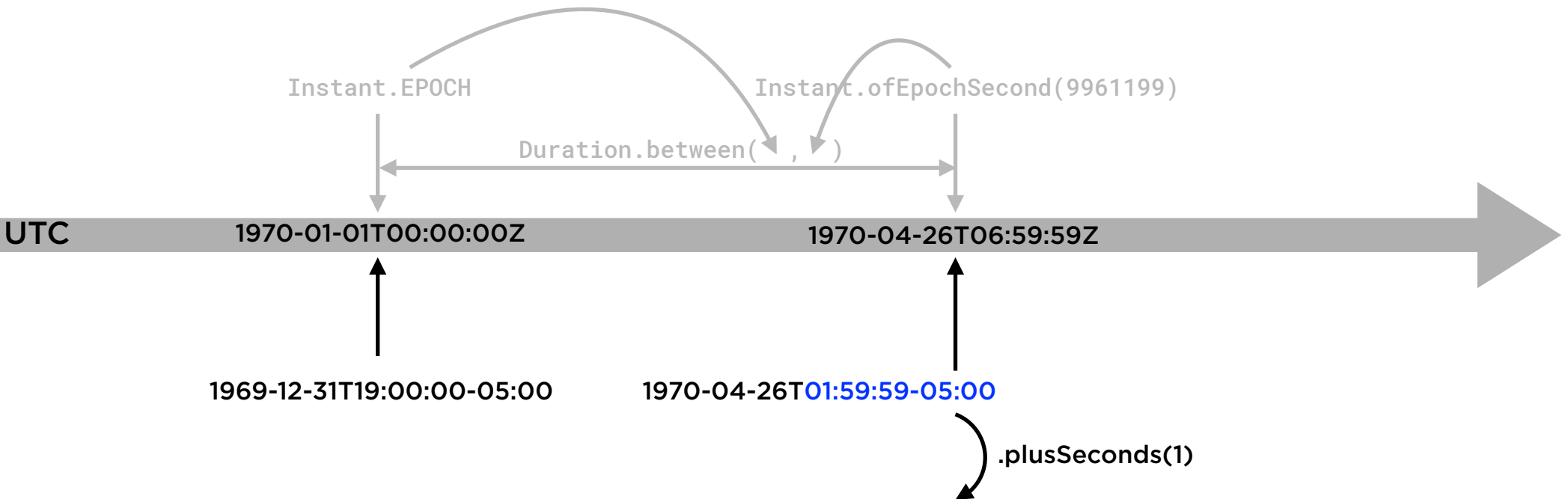
Machine/Solar Time



Human Time

Java Date and Time

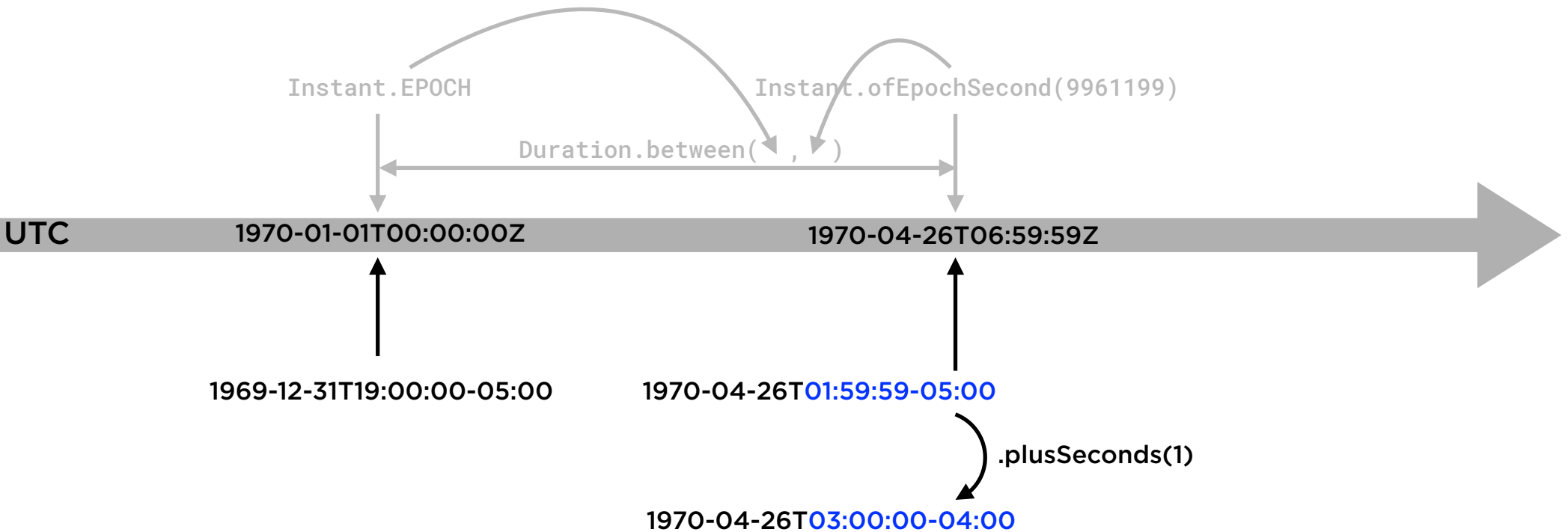
Machine/Solar Time



Human Time

Java Date and Time

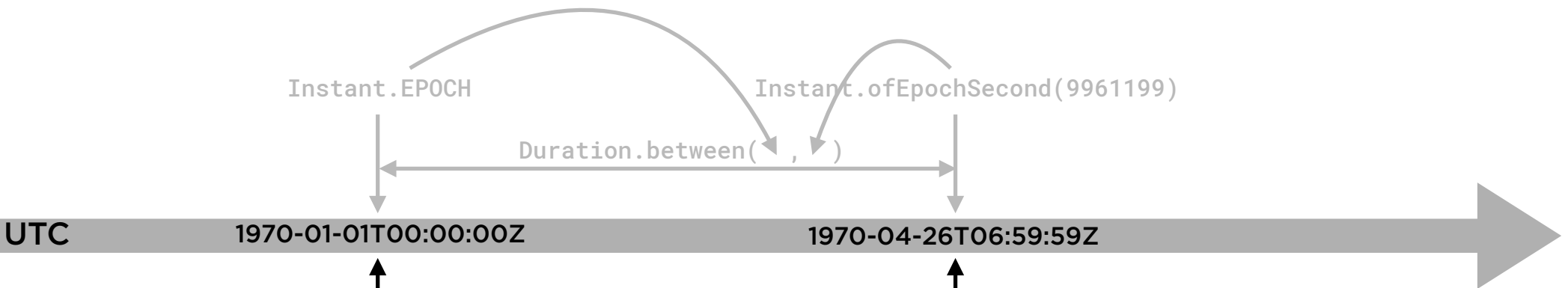
Machine/Solar Time



Human Time

Java Date and Time

Machine/Solar Time



UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

1969-12-31T19:00:00-05:00

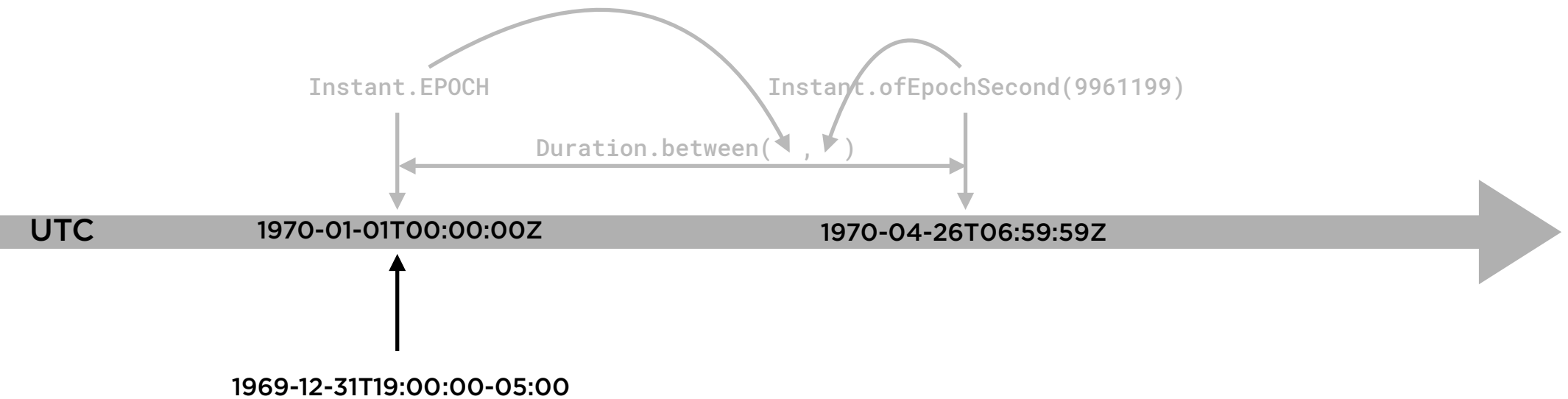
1970-04-26T01:59:59-05:00[America/New_York]

1970-04-26T03:00:00-04:00[America/New_York]

Human Time

Java Date and Time

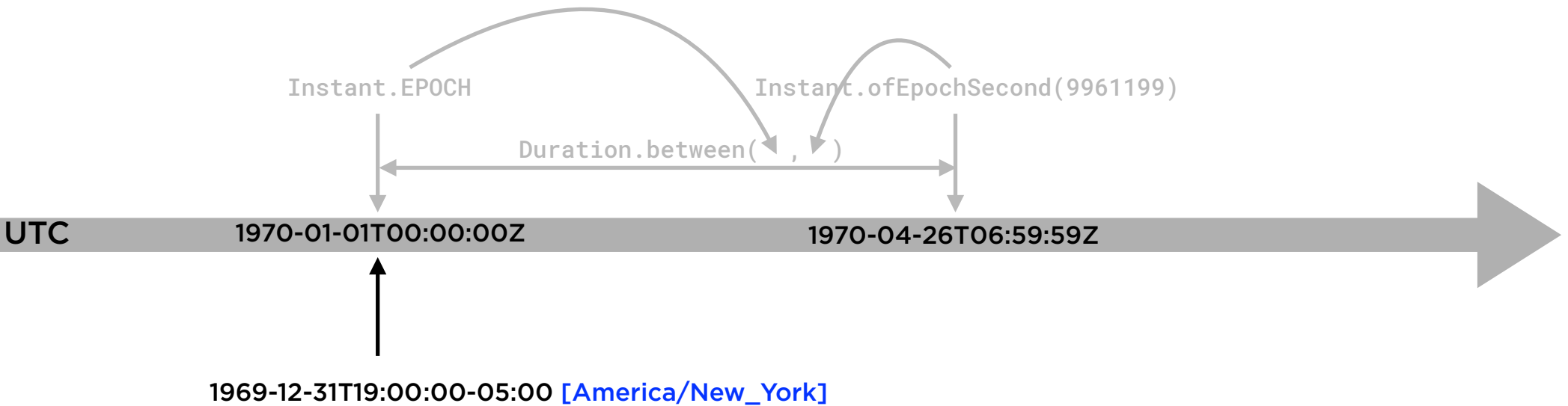
Machine/Solar Time



Human Time

Java Date and Time

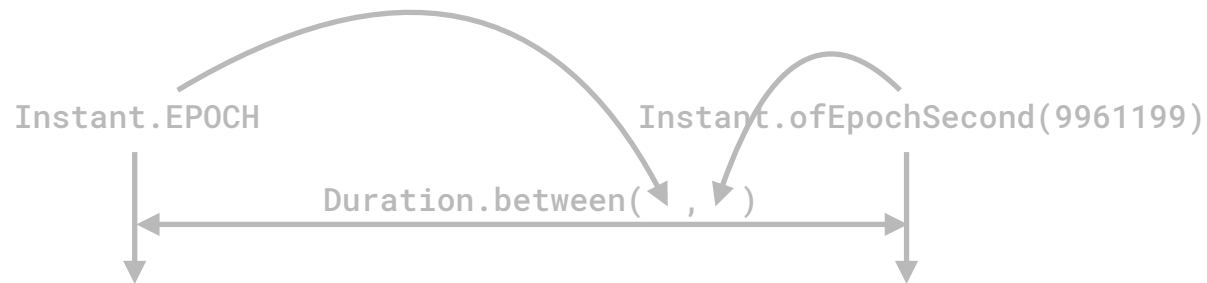
Machine/Solar Time



Human Time

Java Date and Time

Machine/Solar Time



UTC 1970-01-01T00:00:00Z 1970-04-26T06:59:59Z

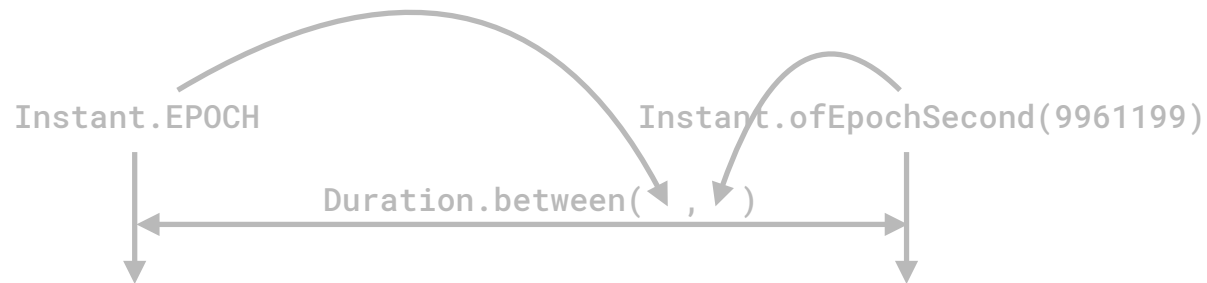
1969-12-31T19:00:00-05:00 [America/New_York]

`Duration.ofDays(116).addTo()`

Human Time

Java Date and Time

Machine/Solar Time



UTC 1970-01-01T00:00:00Z 1970-04-26T06:59:59Z

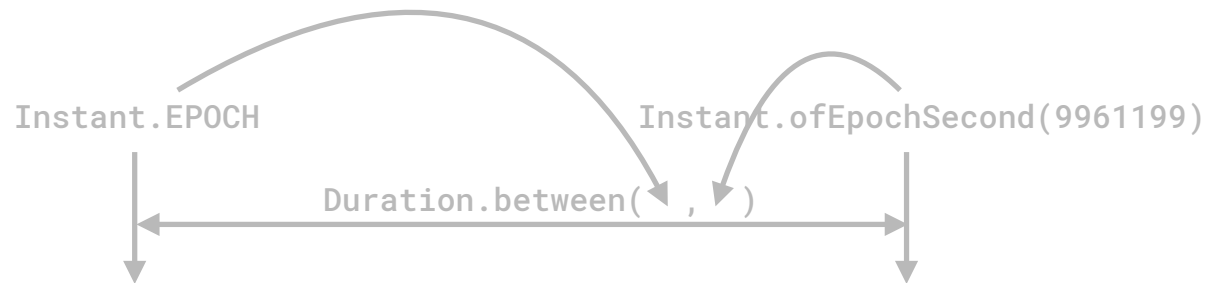
1969-12-31T19:00:00-05:00 [America/New_York]

`Duration.ofDays(116).addTo()` → 1970-04-26T20:00-04:00[America/New_York]

Human Time

Java Date and Time

Machine/Solar Time



UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

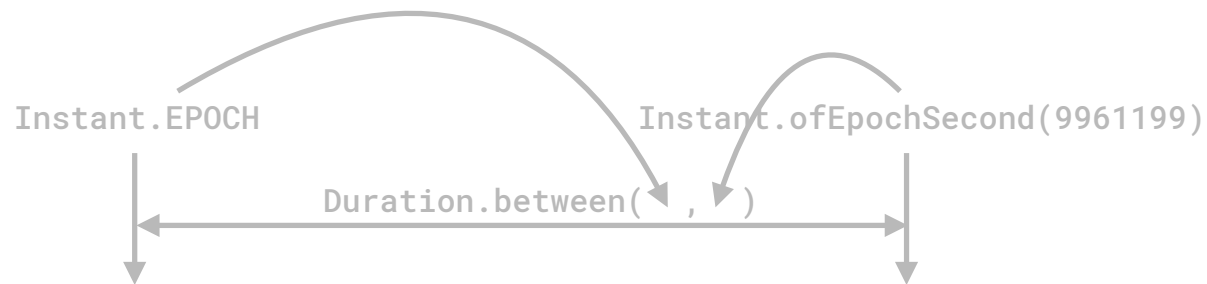
1969-12-31T19:00:00-05:00 [America/New_York]

`Duration.ofDays(116).addTo()` → 1970-04-26T20:00-04:00[America/New_York]
`Period.of(0, 0, 116).addTo()`

Human Time

Java Date and Time

Machine/Solar Time



UTC

1970-01-01T00:00:00Z

1970-04-26T06:59:59Z

1969-12-31T19:00:00-05:00 [America/New_York]

`Duration.ofDays(116).addTo()` → 1970-04-26T20:00-04:00[America/New_York]

`Period.of(0, 0, 116).addTo()` → 1970-04-26T19:00-04:00[America/New_York]

Human Time

java.time

Design Goals

Immutable

`java.time`

Design Goals

java.time

Design Goals

Immutable
- thread-safe

java.time

Design Goals

Immutable

- thread-safe
- allows caching

java.time

Design Goals

Immutable

- thread-safe
- allows caching
- works with streams and lambdas

java.time

Design Goals

Immutable

- thread-safe
- allows caching
- works with streams and lambdas

Fluent

java.time

Design Goals

Immutable

- thread-safe
- allows caching
- works with streams and lambdas

Fluent

- `flightDepartureTime`
 - `.withZoneSameInstant(arrivingZone)`
 - `.plusHours(10)`
 - `.plusMinutes(50);`

java.time

Design Goals

Immutable

- thread-safe
- allows caching
- works with streams and lambdas

Fluent

- `flightDepartureTime`
 `.withZoneSameInstant(arrivingZone)`
 `.plusHours(10)`
 `.plusMinutes(50);`

Type-safe

java.time

Design Goals

Immutable

- thread-safe
- allows caching
- works with streams and lambdas

Fluent

- `flightDepartureTime`
 `.withZoneSameInstant(arrivingZone)`
 `.plusHours(10)`
 `.plusMinutes(50);`

Type-safe

Extensible

`java.time`

Design Goals

Immutable

- thread-safe
- allows caching
- works with streams and lambdas

Fluent

- `flightDepartureTime`
 `.withZoneSameInstant(arrivingZone)`
 `.plusHours(10)`
 `.plusMinutes(50);`

Type-safe

Extensible

- more features in ThreeTen-Extra

`java.time`

Core Classes

java.time

Core Classes

java.time	Meaning	Example
-----------	---------	---------

java.time

Core Classes

java.time	Meaning	Example
Instant	instant of time	timestamp

java.time

Core Classes

java.time	Meaning	Example
Instant	instant of time	timestamp
ZonedDateTime	date-time with time zone information	start of a conference call

java.time

Core Classes

java.time	Meaning	Example
Instant	instant of time	timestamp
ZonedDateTime	date-time with time zone information	start of a conference call
LocalDate	date without time zone information	birthday

java.time

Core Classes

java.time	Meaning	Example
Instant	instant of time	timestamp
ZonedDateTime	date-time with time zone information	start of a conference call
LocalDate	date without time zone information	birthday
Duration	time between two instants	length of a conference call

java.time

Core Classes

java.time	Meaning	Example
Instant	instant of time	timestamp
ZonedDateTime	date-time with time zone information	start of a conference call
LocalDate	date without time zone information	birthday
Duration	time between two instants	length of a conference call
Period	amount of time in years, months, and days	length of a prison sentence

java.time

Formatting
and
Parsing

java.time

Formatting
and
Parsing

DateTimeFormatter

java.time

Formatting
and
Parsing

DateTimeFormatter

- .parse(String)

java.time

Formatting
and
Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME

```
1969-12-31T19:00:00-05:00[America/New_York]
```

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME

```
1969-12-31T19:00:00-05:00[America/New_York]
```

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME

```
1969-12-31T19:00:00-05:00[America/New_York]
```

- .ofLocalizedTime()

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME
- `1969-12-31T19:00:00-05:00[America/New_York]`
- .ofLocalizedTime(FormatStyle.SHORT)

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME
1969-12-31T19:00:00-05:00[America/New_York]
- .ofLocalizedTime(FormatStyle.SHORT)
US locale 7:00 PM

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME
1969-12-31T19:00:00-05:00[America/New_York]
- .ofLocalizedTime(FormatStyle.SHORT)
 - US locale 7:00 PM
 - UK locale 19:00

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME
1969-12-31T19:00:00-05:00[America/New_York]
- .ofLocalizedTime(FormatStyle.SHORT)
US locale 7:00 PM
UK locale 19:00
- .ofPattern()

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME
1969-12-31T19:00:00-05:00[America/New_York]
- .ofLocalizedTime(FormatStyle.SHORT)
US locale 7:00 PM
UK locale 19:00
- .ofPattern("E")

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME
1969-12-31T19:00:00-05:00[America/New_York]
- .ofLocalizedTime(FormatStyle.SHORT)
US locale 7:00 PM
UK locale 19:00
- .ofPattern("E")
Wed

java.time

Formatting and Parsing

DateTimeFormatter

- .parse(String)
- .format(TemporalAccessor)
- .ISO_DATE_TIME
1969-12-31T19:00:00-05:00[America/New_York]
- .ofLocalizedTime(FormatStyle.SHORT)
US locale 7:00 PM
UK locale 19:00
- .ofPattern("E")
Wed

`java.time`

Interoperation

`java.time`

Interoperation

`java.util` classes

java.time

Interoperation

java.util classes

-Date.from(Instant)

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()
- Calendar.toInstant()

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()
- Calendar.toInstant()
- timeZone.toZoneId()

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()
- Calendar.toInstant()
- timeZone.toZoneId()
- TimeZone.getTimeZone(ZoneId)

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()
- Calendar.toInstant()
- timeZone.toZoneId()
- TimeZone.getTimeZone(ZoneId)

java.sql classes

`java.time`

Interoperation

`java.util` classes

- `Date.from(Instant)`
- `date.toInstant()`
- `Calendar.toInstant()`
- `timeZone.toZoneId()`
- `TimeZone.getTimeZone(ZoneId)`

`java.sql` classes

- `Date <-> LocalDate`

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()
- Calendar.toInstant()
- timeZone.toZoneId()
- TimeZone.getTimeZone(ZoneId)

java.sql classes

- Date <-> LocalDate
- TimeStamp <-> LocalDateTime

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()
- Calendar.toInstant()
- timeZone.toZoneId()
- TimeZone.getTimeZone(ZoneId)

java.sql classes

- Date <-> LocalDate
- TimeStamp <-> LocalDateTime
- TimeStamp <-> Instant

java.time

Interoperation

java.util classes

- Date.from(Instant)
- date.toInstant()
- Calendar.toInstant()
- timeZone.toZoneId()
- TimeZone.getTimeZone(ZoneId)

java.sql classes

- Date <-> LocalDate
- TimeStamp <-> LocalDateTime
- TimeStamp <-> Instant

JPA - AttributeConverter

A Personal Task Scheduler

A Personal Task Scheduler

A Personal Task Scheduler

Demo application for the course

A Personal Task Scheduler

Demo application for the course

Like a todo list, but with a difference

A Personal Task Scheduler

Demo application for the course

Like a todo list, but with a difference

Starting point is a work schedule

A Personal Task Scheduler

Demo application for the course

Like a todo list, but with a difference

Starting point is a work schedule

- Including fixed events (like meetings)

A Personal Task Scheduler

Demo application for the course

Like a todo list, but with a difference

Starting point is a work schedule

- Including fixed events (like meetings)

User guesses the duration of each task

A Personal Task Scheduler

Demo application for the course

Like a todo list, but with a difference

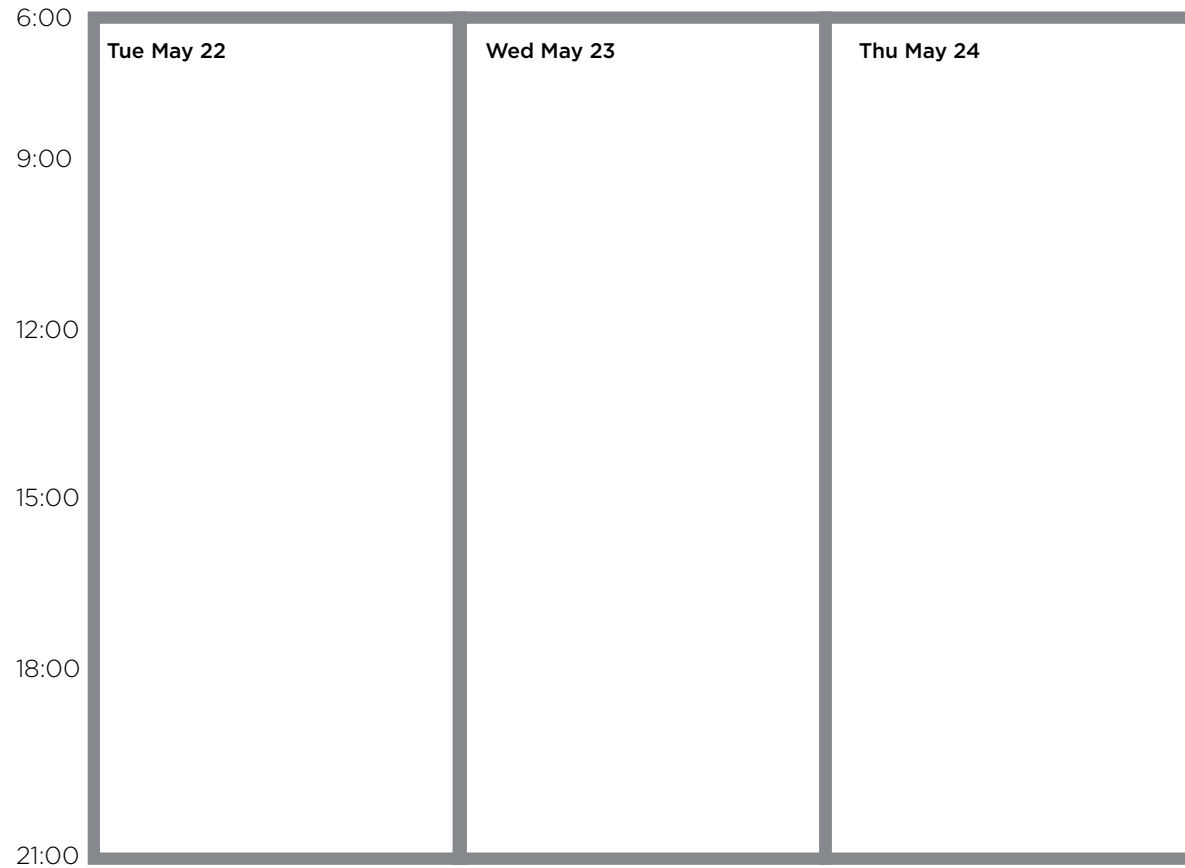
Starting point is a work schedule

- Including fixed events (like meetings)

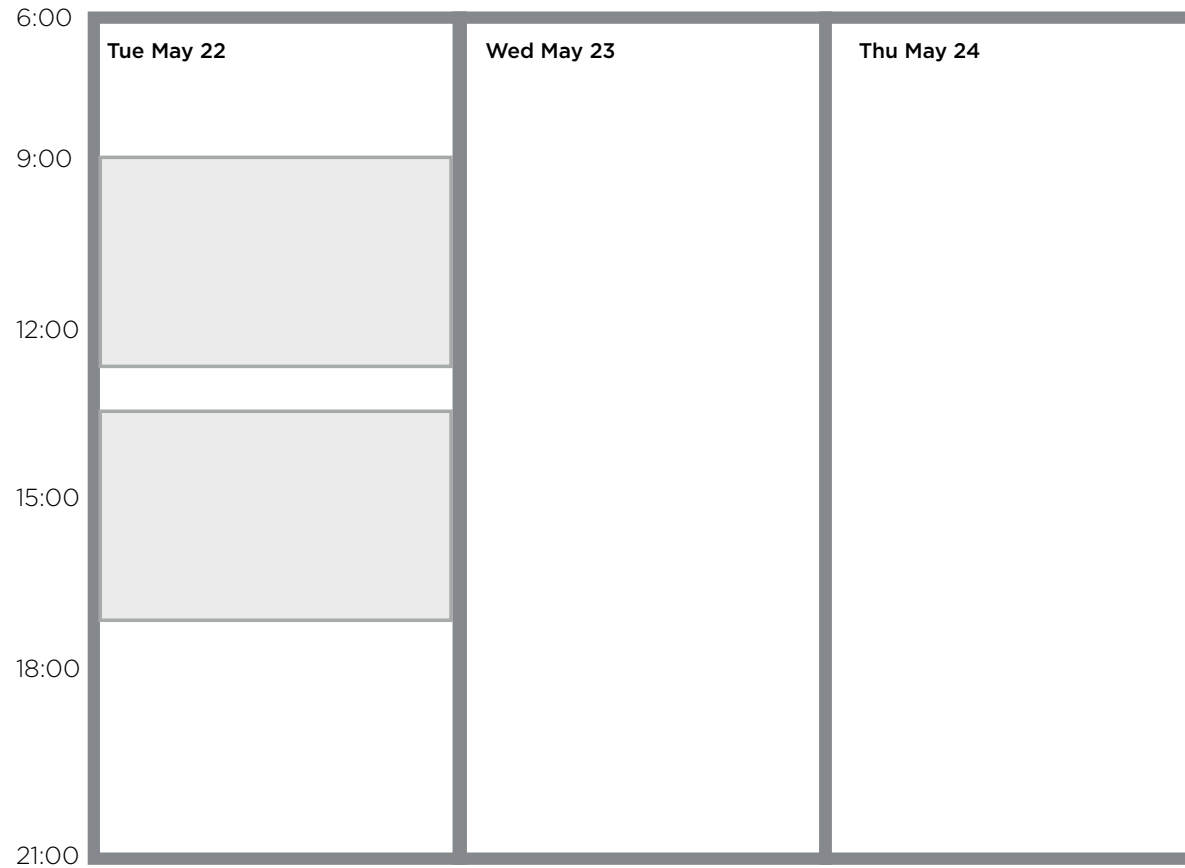
User guesses the duration of each task

System allocates task to work periods

A Task Scheduler



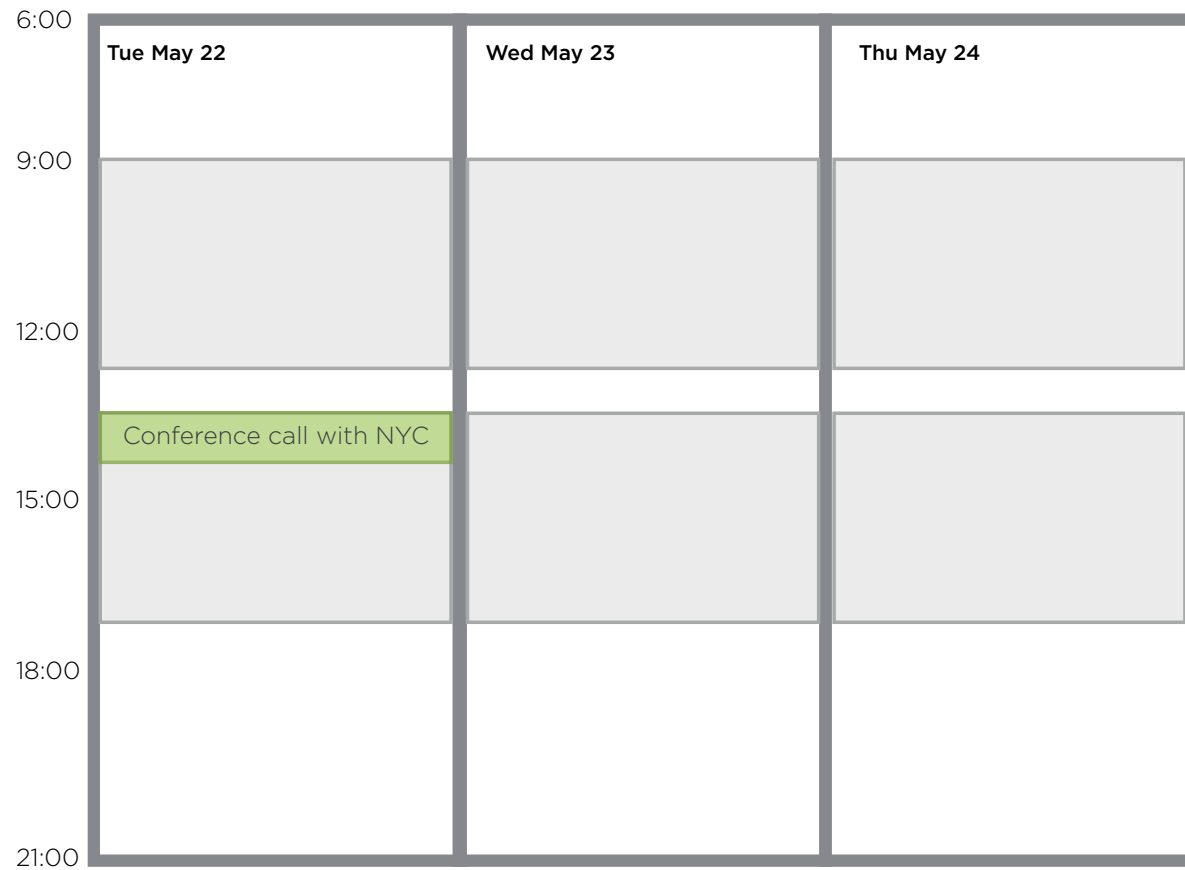
A Task Scheduler



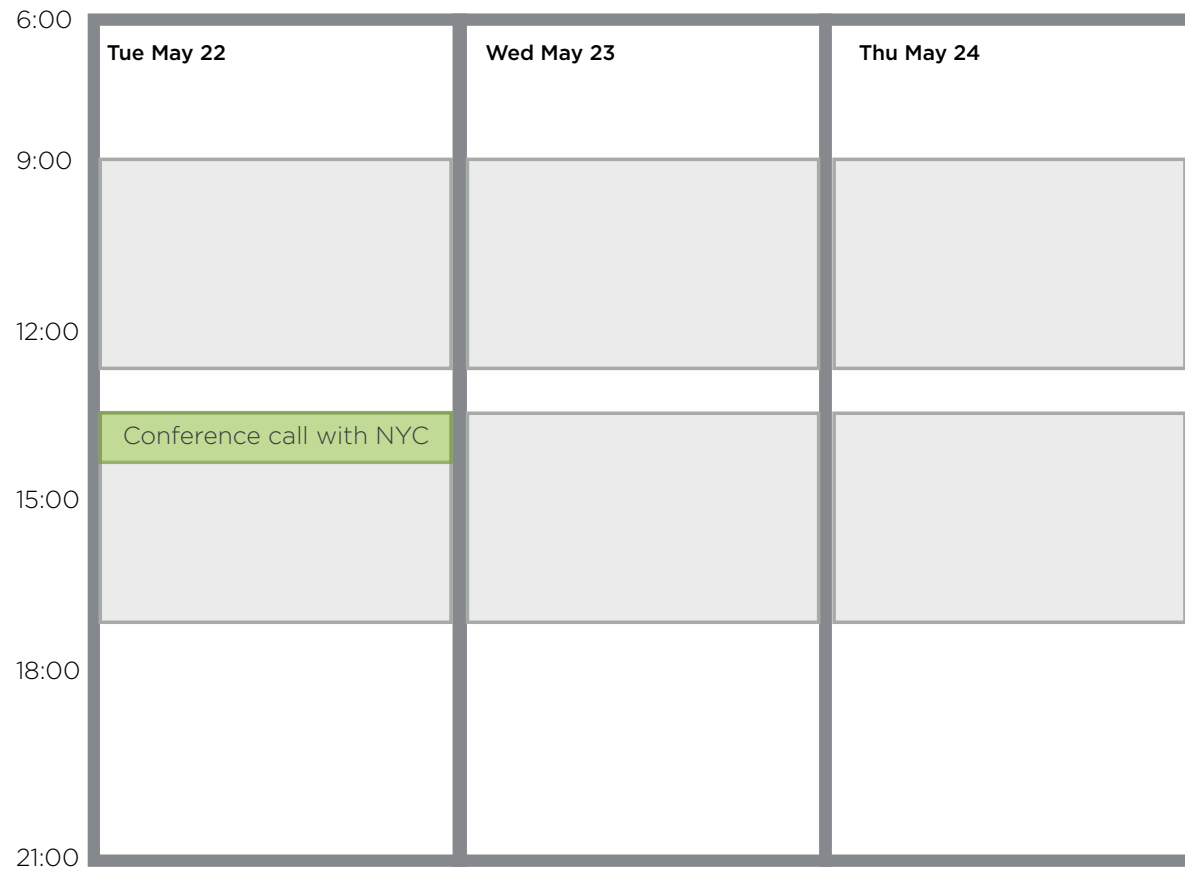
A Task Scheduler

6:00	Tue May 22	Wed May 23	Thu May 24
9:00			
12:00			
15:00			
18:00			
21:00			

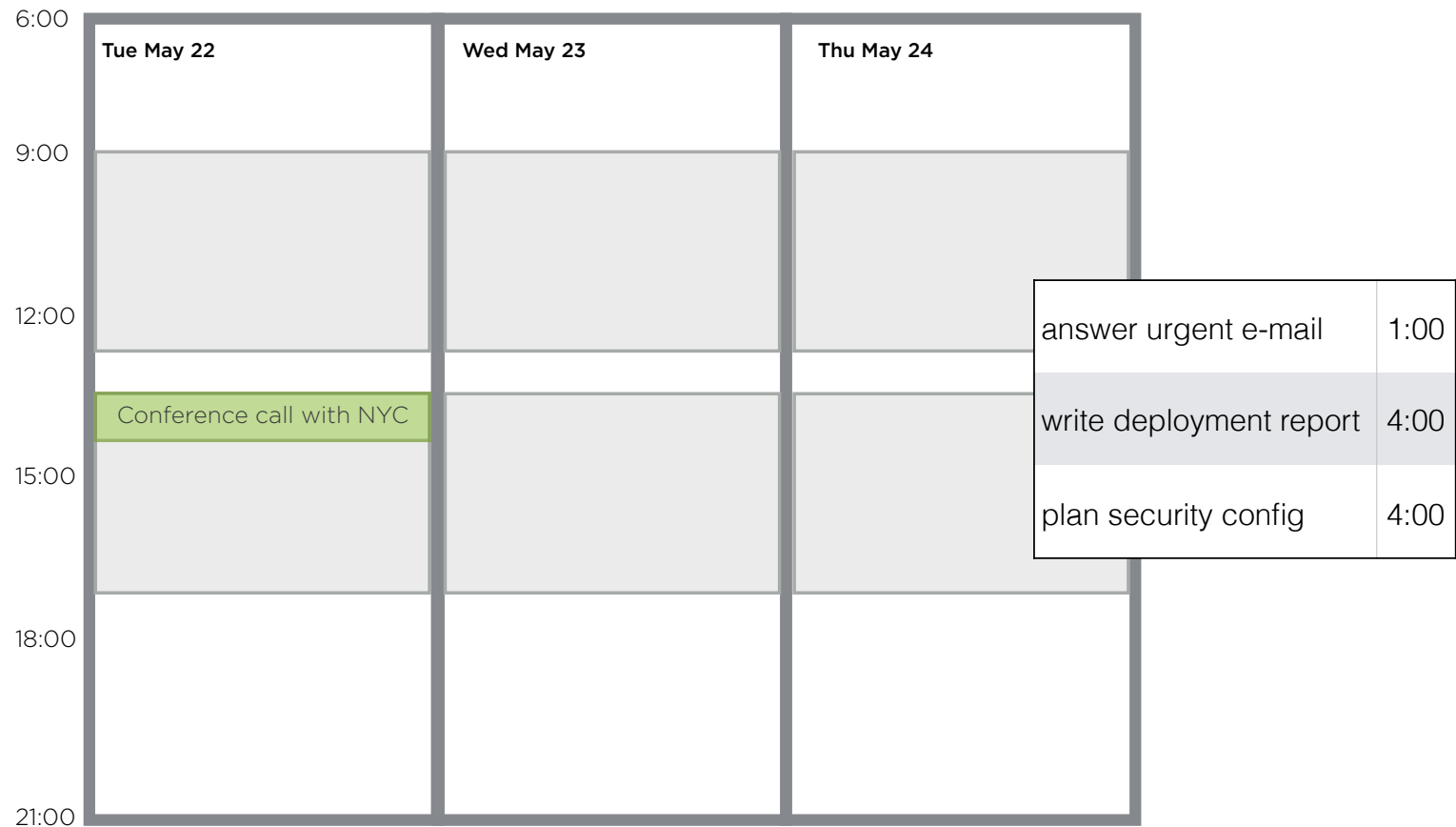
A Task Scheduler



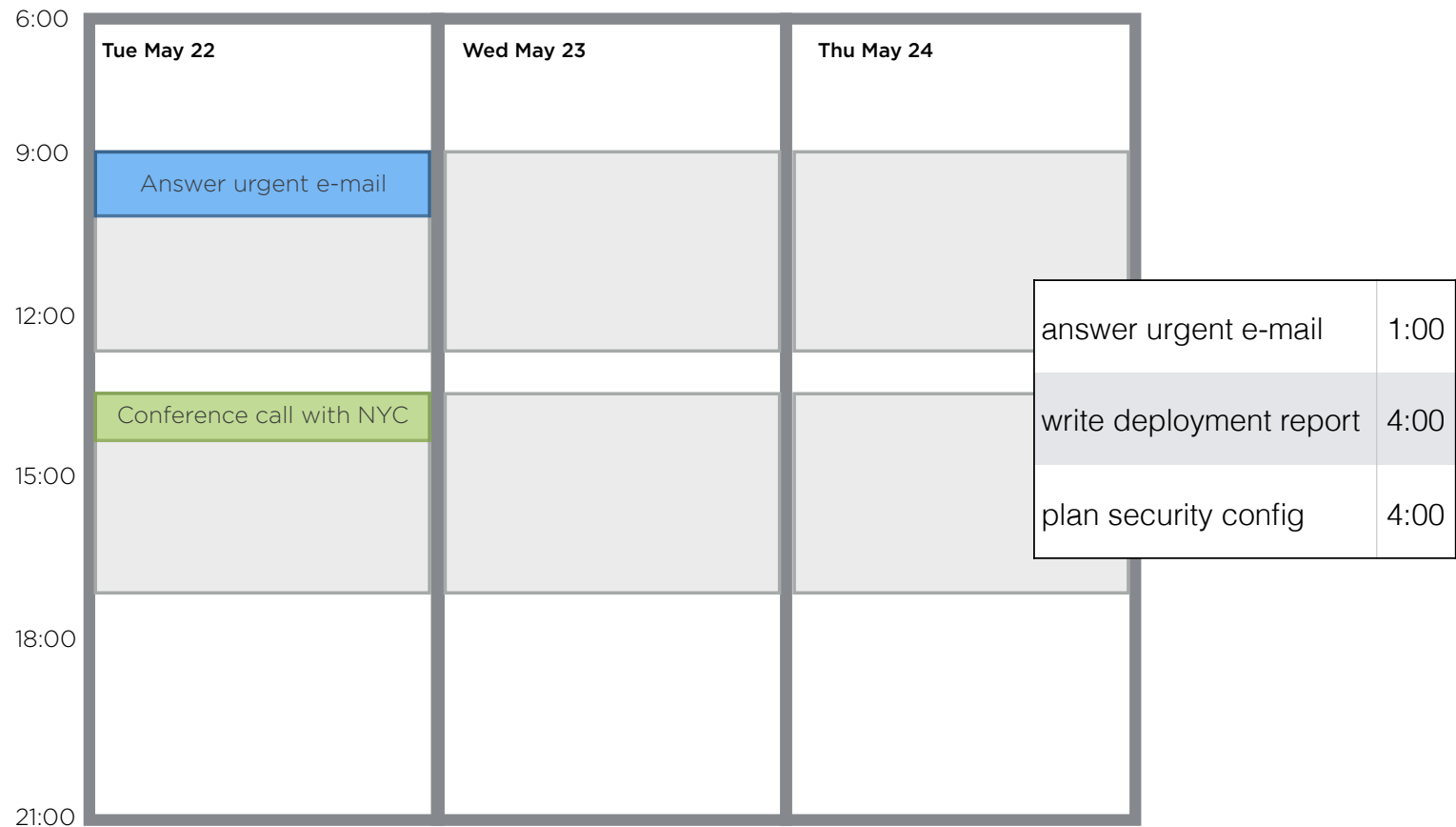
A Task Scheduler



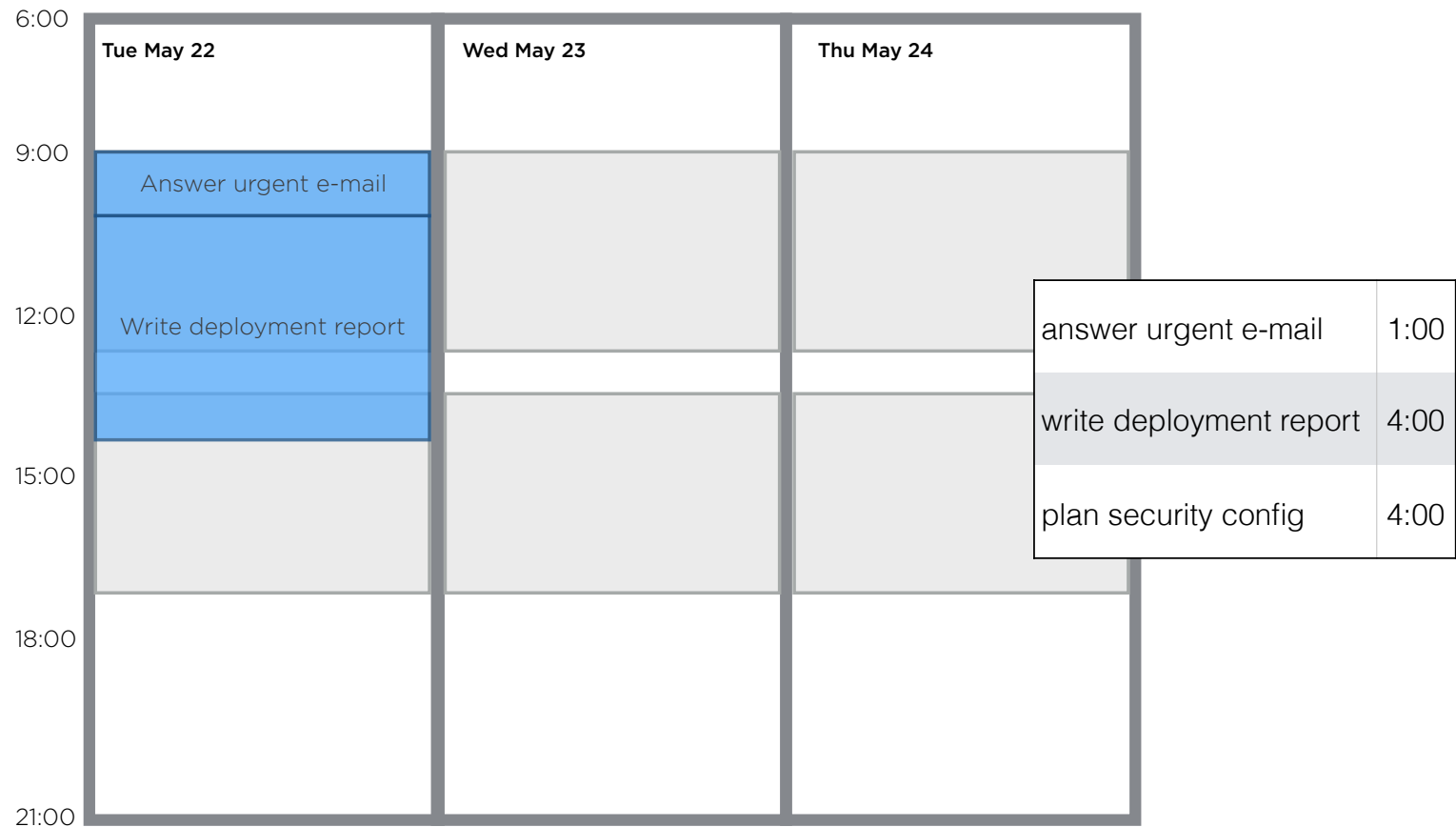
A Task Scheduler



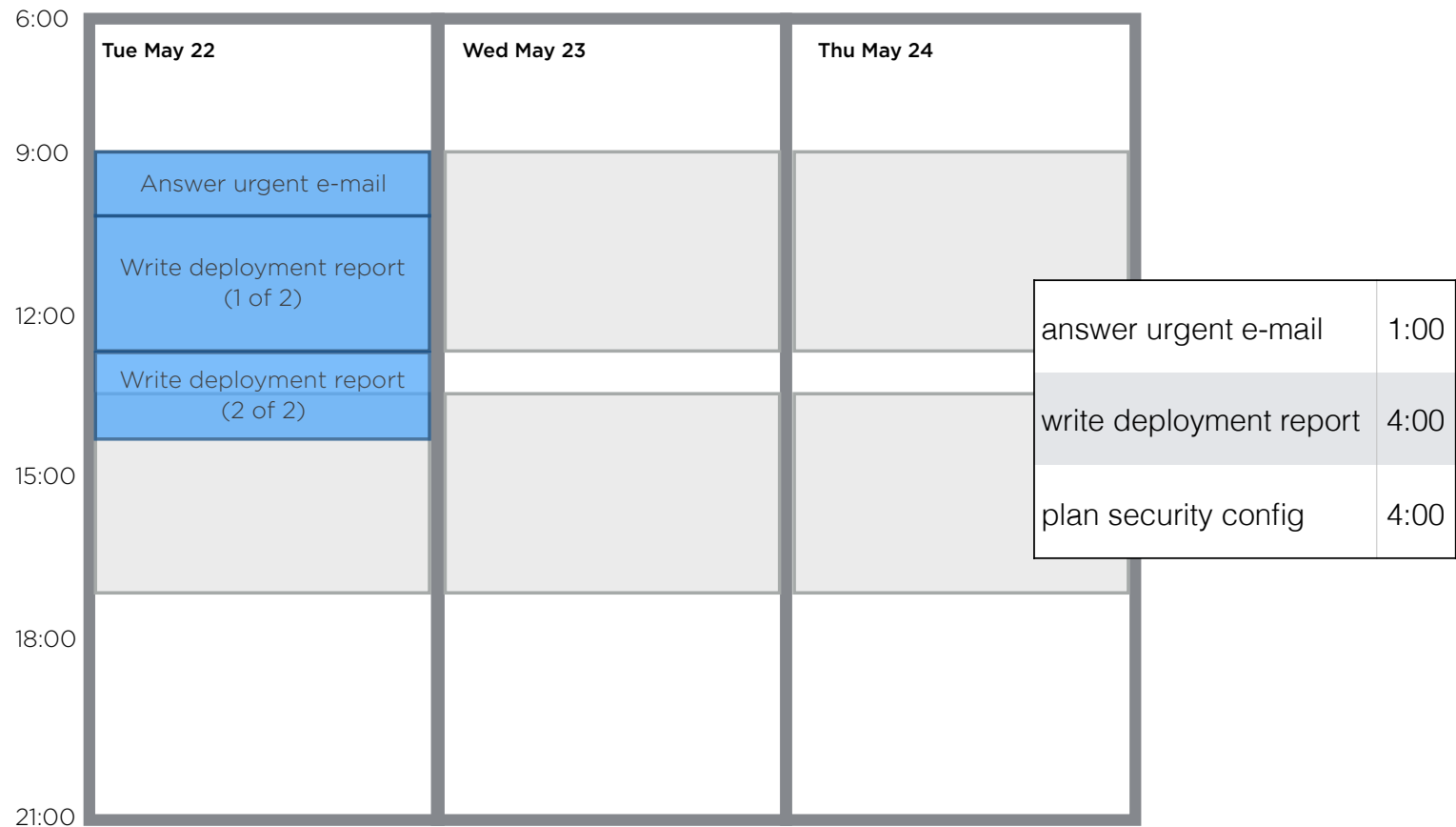
A Task Scheduler



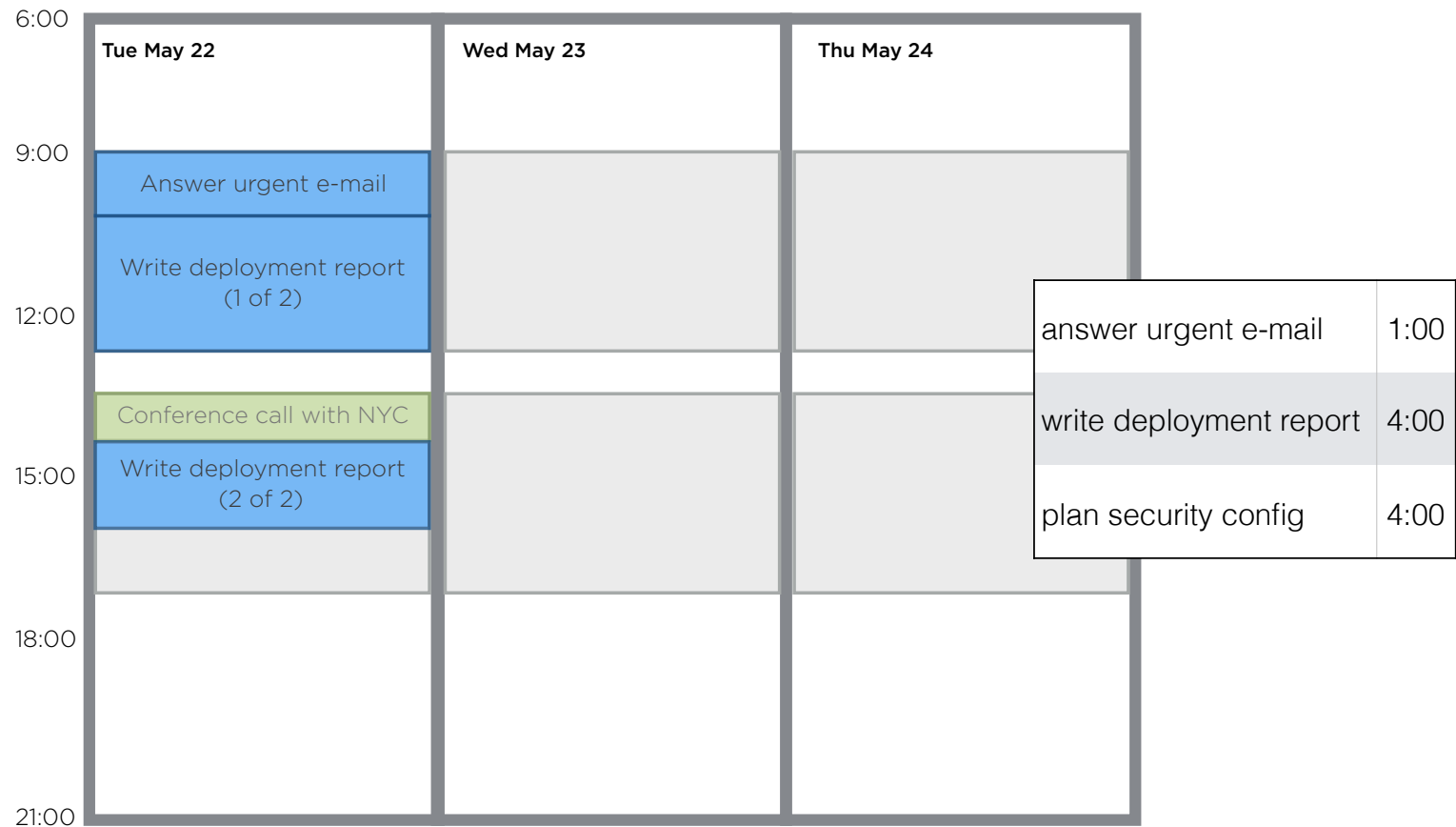
A Task Scheduler



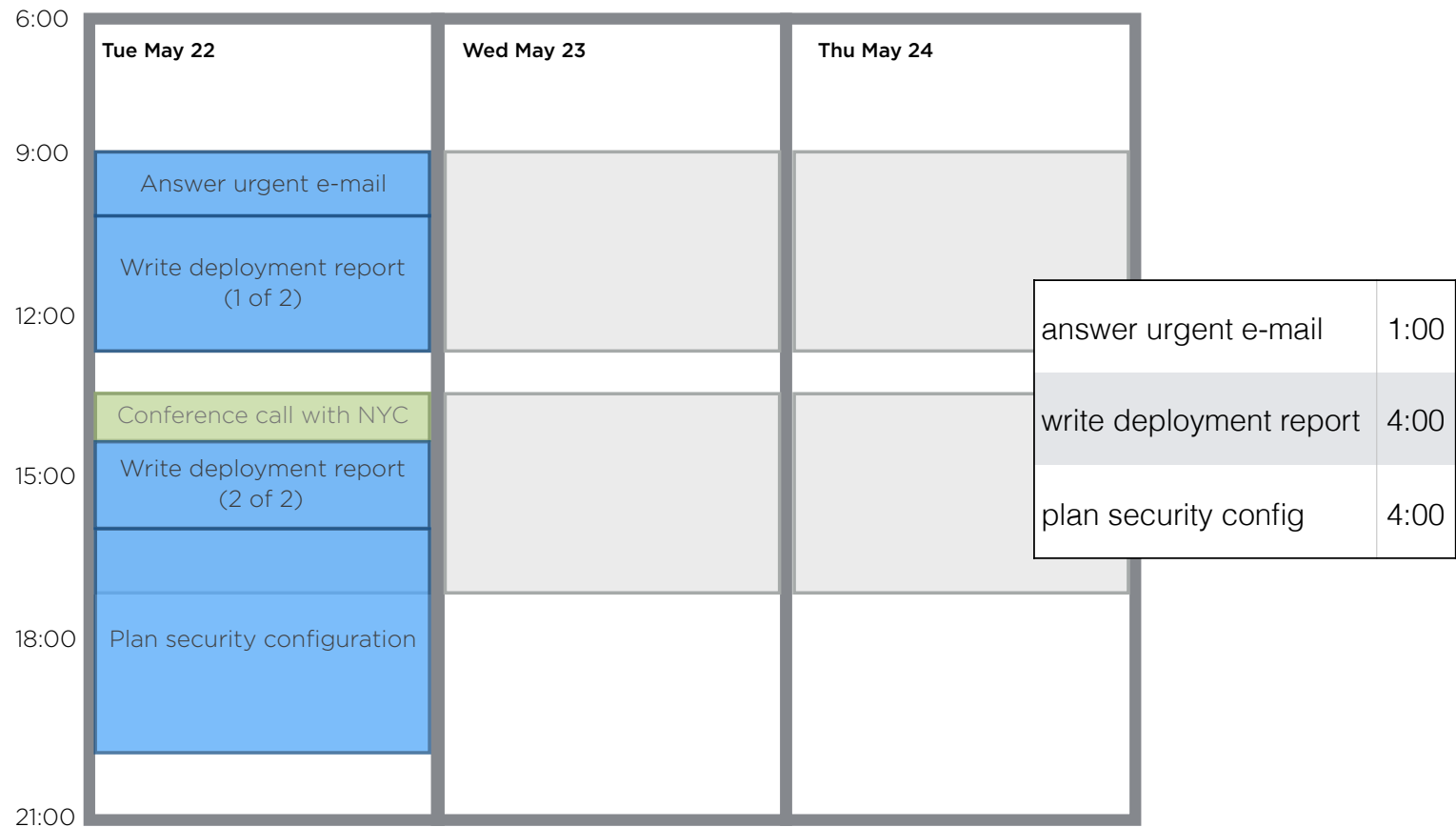
A Task Scheduler



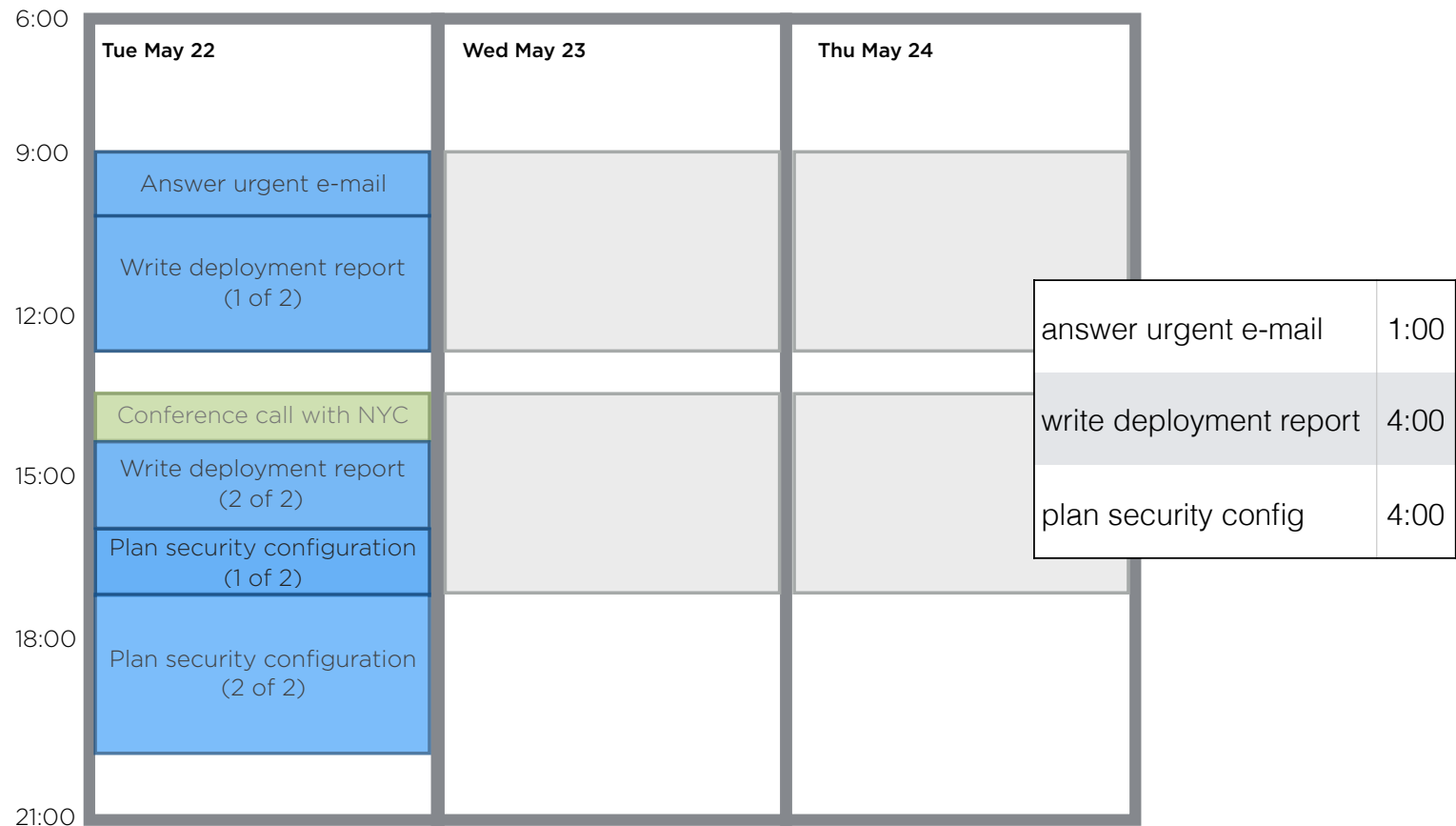
A Task Scheduler



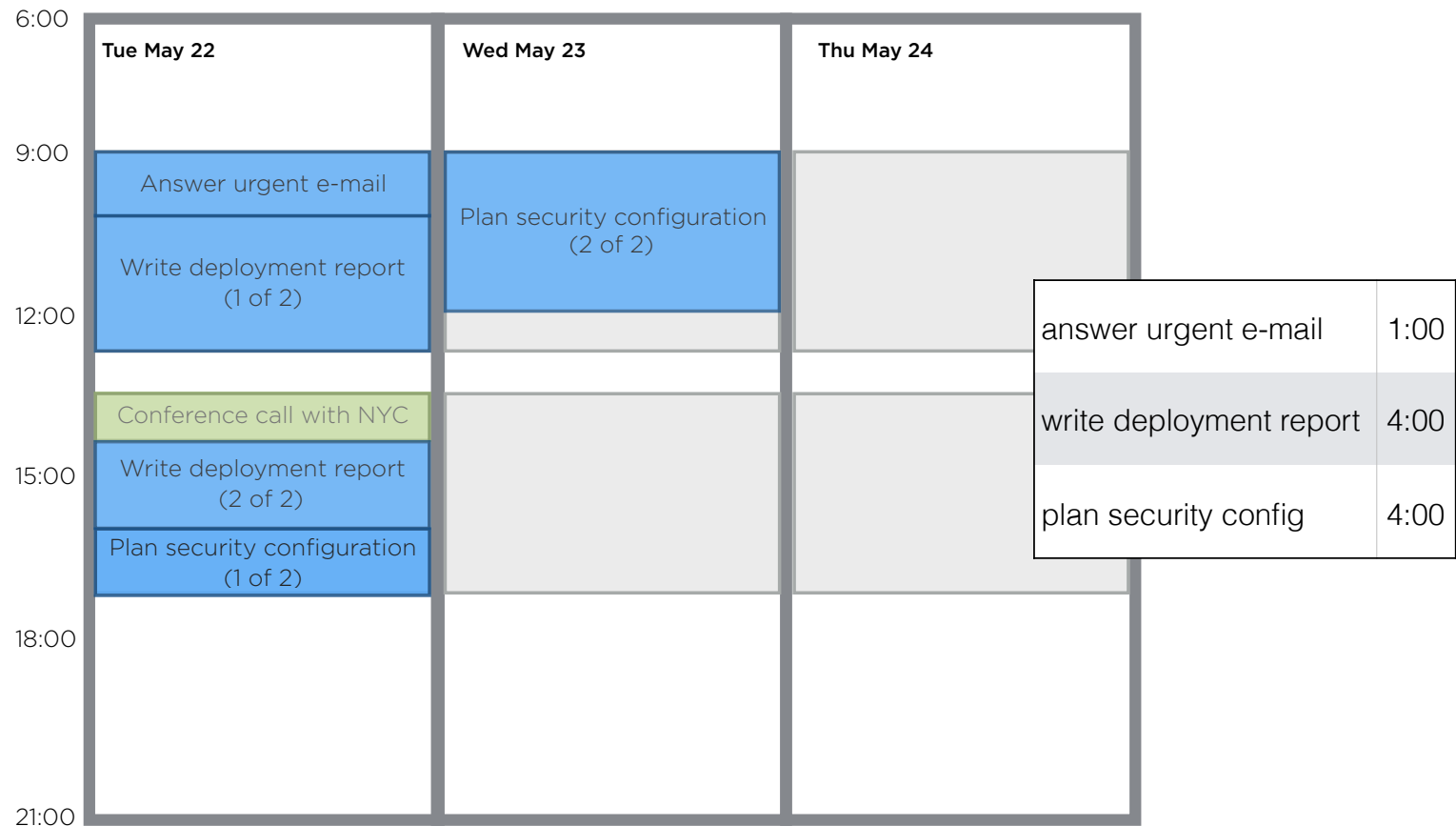
A Task Scheduler



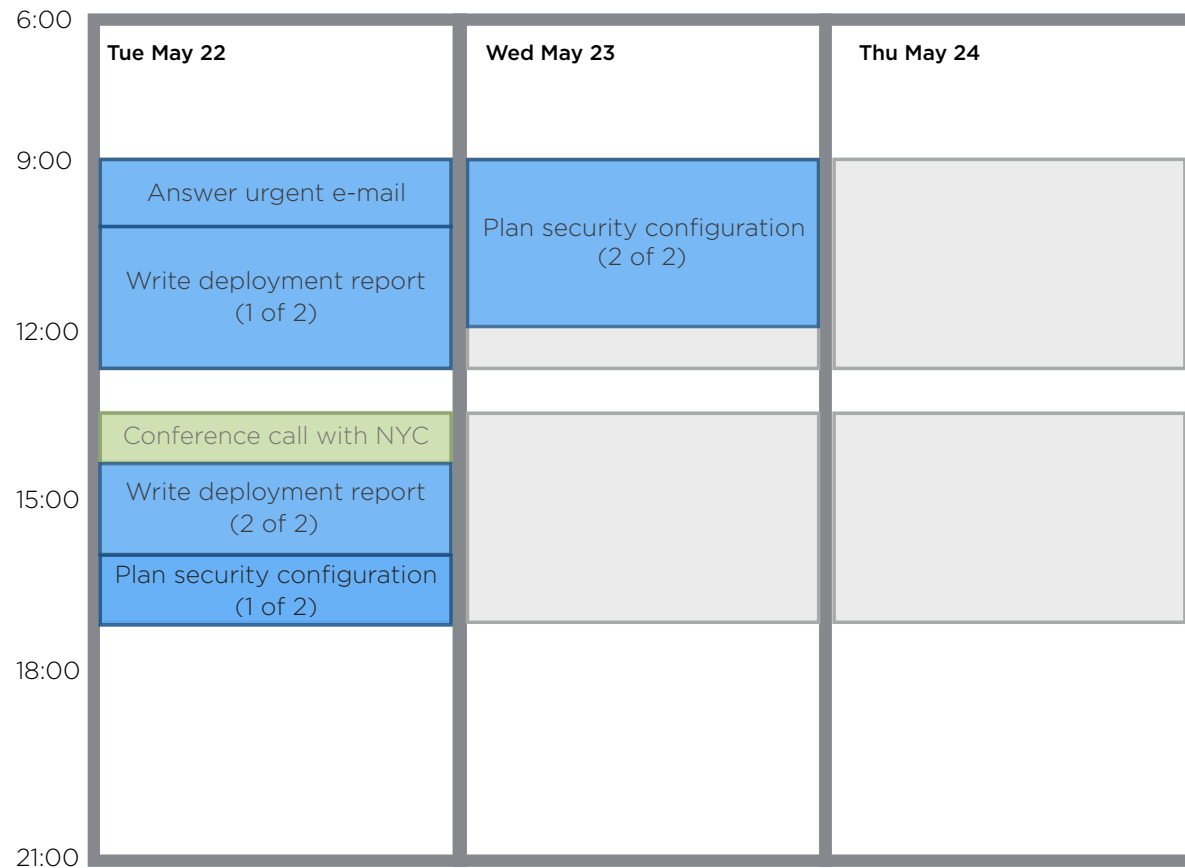
A Task Scheduler



A Task Scheduler



A Task Scheduler



Summary

Summary

Summary

Why we need an API for date and time

Summary

Why we need an API for date and time

Domain assumptions behind `java.time`

Summary

Why we need an API for date and time

Domain assumptions behind `java.time`

- Human time vs. Machine time

Summary

Why we need an API for date and time

Domain assumptions behind `java.time`

- Human time vs. Machine time

Overview of the Java date/time API

Summary

Why we need an API for date and time

Domain assumptions behind `java.time`

- Human time vs. Machine time

Overview of the Java date/time API

- Design Goals, Core Classes