CUBEDYNAMICS • RC3 BETA TESTER SUPERSHOWCASE

The pipe is the scientific sentence

This notebook is built to show what **CubeDynamics 0.1.0rc3** enables that ordinary analysis code usually hides: environmental nouns, source choices, ordered analytical verbs, semantic states, events, relationships, traces, and evidence.

noun → source → pipe → verb → state → trace → evidence

Four stories, one grammar

1 · Working Lands
When and where did unusual warmth and little or no rain coincide?

2 · Boulder Cold Snap
Can observations become conditions, events, synchrony, and lagged relationships?

3 · Multivariate Weather
Can the same grammar move across several environmental nouns without erasing their differences?

4 · Remote Sensing
Can the same vocabulary extend from climate cubes to repeated Sentinel-2 vegetation observations?

How to use this notebook in a room

Read the **pipe** aloud before running it. Ask the audience what scientific

> object they expect next. Then run the cell and let CubeDynamics explain what it thinks the sentence means.

0 · Install the exact release candidate

In [1]:
```
%pip install -q --user --upgrade "cubedynamics==0.1.0rc3"
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import sys, warnings, html, traceback
import numpy as np
import pandas as pd
import xarray as xr
import matplotlib.pyplot as plt
from IPython.display import display, HTML

import cubedynamics as cd
from cubedynamics import data, pipe, verbs as v

warnings.filterwarnings("ignore", category=FutureWarning)

display(HTML("""
<style>
:root{
  --cd-ink:#203039;
  --cd-muted:#5c6d75;
  --cd-blue:#2f7f9b;
  --cd-blue-bg:#eef6f8;
  --cd-green:#4f8b63;
  --cd-green-bg:#f5faf6;
  --cd-gold:#c8942f;
  --cd-gold-bg:#fffaf0;
  --cd-warm:#c65a37;
  --cd-warm-bg:#fff7f2;
  --cd-red:#b94a48;
  --cd-red-bg:#fff6f6;
}
.jp-Notebook,.notebook-container{max-width:1180px}
.cd-kicker{
  text-transform:uppercase;letter-spacing:.12em;font-weight:800;
  color:#55717d;font-size:12px;margin:4px 0 8px
}
.cd-hero{
  border:1px solid #d6e2e7;border-radius:16px;padding:24px 26px;
  margin:12px 0 22px;background:linear-gradient(135deg,#f7fbfc,#fff)
}
.cd-hero h1{margin:0 0 8px;font-size:36px;line-height:1.05;color:var(-
.cd-hero p{font-size:17px;line-height:1.5;color:#41545e;margin:8px 0}
.cd-card{
  border:1px solid #d9e3e8;border-left:6px solid var(--cd-blue);
```

```
  border-radius:12px;padding:14px 18px;margin:12px 0;background:#f8fbf
}
.cd-card.warm{border-left-color:var(--cd-warm);background:var(--cd-war
.cd-card.green{border-left-color:var(--cd-green);background:var(--cd-g
.cd-card.gold{border-left-color:var(--cd-gold);background:var(--cd-gol
.cd-card.red{border-left-color:var(--cd-red);background:var(--cd-red-b
.cd-title{font-weight:800;font-size:17px;margin-bottom:4px;color:var(-
.cd-question{
  padding:16px 18px;border-radius:12px;margin:14px 0;
  background:var(--cd-blue-bg);border:1px solid #cde1e7;font-size:16px
}
.cd-grammar{
  text-align:center;font-size:20px;font-weight:800;padding:16px;
  border-radius:12px;background:#f7f9fa;border:1px solid #dde5e8;margi
}
.cd-pipe{
  font-family:ui-monospace,SFMono-Regular,Menlo,monospace;
  font-size:16px;line-height:1.65;padding:14px 18px;margin:12px 0 18px
  border-radius:12px;background:#f7fbfc;border:1px solid #d7e6eb
}
.cd-pipe .noun{font-weight:800;color:#1f6f8b}
.cd-pipe .verb{font-weight:800;color:#7a5d12}
.cd-pipe .state{font-weight:800;color:#3f7d54}
.cd-pipe .arrow{padding:0 8px;color:#7f8f96}
.cd-scroll{
  max-height:250px;overflow:auto;padding:12px 14px;border:1px solid #d
  border-radius:10px;background:#fbfcfd;
  font-family:ui-monospace,SFMono-Regular,Menlo,monospace;font-size:12
}
</style>
"""))

assert cd.__version__ == "0.1.0rc3", f"Expected 0.1.0rc3, got {cd.__ve

def panel(title, body="", tone="green"):
    display(HTML(
        f'<div class="cd-card {tone}"><div class="cd-title">{html.esca
        f'<div>{body}</div></div>'
    ))

def text_panel(title, text, tone=""):
    display(HTML(
        f'<div class="cd-card {tone}"><div class="cd-title">{html.esca
        f'<div class="cd-scroll">{html.escape(str(text))}</div></div>'
    ))

def explain(p):
    text_panel("What CubeDynamics says this sentence means", p.explain
    try:
        text_panel("Semantic validation", p.validate(), "gold")
    except Exception:
        pass
```

```
def state_field(obj):
    if isinstance(obj, xr.Dataset):
        if "state" in obj:
            return obj["state"]
        if len(obj.data_vars) == 1:
            return obj[next(iter(obj.data_vars))]
    return obj

def compact_attrs(obj):
    keys = [
        "scientific_noun","source","source_product","source_variable",
        "units","semantic_kind","provider","crs"
    ]
    rows = [{"attribute":k,"value":obj.attrs.get(k)} for k in keys if
    if rows:
        display(
            pd.DataFrame(rows)
            .style
            .hide(axis="index")
            .set_caption("Selected source + semantic metadata")
        )

# ---------- plotting helpers: presentation plumbing only ----------

def show_working_lands_observations(temperature, precipitation):
    inds = [0, len(temperature.time)//2, -1]
    labels = ["July 1", "Mid-July", "July 31"]

    fig, axes = plt.subplots(1, 3, figsize=(15, 4.5), constrained_layo
    for ax, idx, label in zip(axes, inds, labels):
        im = temperature.isel(time=idx).plot(ax=ax, add_colorbar=False
        plt.colorbar(im, ax=ax, label=temperature.attrs.get("units", "
        ax.set_title(label)
    fig.suptitle("PRISM maximum temperature through July", fontsize=17
    plt.show()

    fig, axes = plt.subplots(1, 2, figsize=(12, 4.7), constrained_layo
    precipitation.isel(time=0).plot(ax=axes[0])
    axes[0].set_title("One precipitation observation")
    precipitation.sum("time").plot(ax=axes[1])
    axes[1].set_title("Accumulated July precipitation")
    plt.show()

def show_spatial_summary(p, title):
    field = state_field(p.unwrap())
    fig, ax = plt.subplots(figsize=(8.5, 5.2))
    field.plot(ax=ax)
    ax.set_title(title)
    plt.show()

def show_conditions(warm, dry):
```

```
    warm_state = state_field(warm.unwrap())
    dry_state = state_field(dry.unwrap())
    day = min(15, warm_state.sizes["time"] - 1)

    fig, axes = plt.subplots(1, 3, figsize=(15, 4.4), constrained_layo
    warm_state.isel(time=day).astype(int).plot(ax=axes[0], vmin=0, vma
    axes[0].set_title("Warm")
    dry_state.isel(time=day).astype(int).plot(ax=axes[1], vmin=0, vmax
    axes[1].set_title("Dry")
    warm_state.mean("time").plot(ax=axes[2], vmin=0, vmax=1)
    axes[2].set_title("Warm frequency")
    plt.show()
    return day

def show_joint(joint, frequency, day):
    joint_state = state_field(joint.unwrap())
    freq = state_field(frequency.unwrap())

    fig, axes = plt.subplots(1, 2, figsize=(12, 4.8), constrained_layo
    joint_state.isel(time=day).astype(int).plot(ax=axes[0], vmin=0, vm
    axes[0].set_title("Warm ∩ dry on one day")
    (freq * 100).plot(ax=axes[1], vmin=0, vmax=100)
    axes[1].set_title("Warm + dry frequency (%)")
    plt.show()
    return freq

def show_frequency_comparison(relative_freq, absolute_freq):
    fig, axes = plt.subplots(1, 2, figsize=(12, 4.8), constrained_layo
    (relative_freq * 100).plot(ax=axes[0], vmin=0, vmax=100)
    axes[0].set_title("Relative warm + dry")
    (absolute_freq * 100).plot(ax=axes[1], vmin=0, vmax=100)
    axes[1].set_title(">35 °C + dry")
    plt.show()

def show_regional_weather(temp_pipe, ppt_pipe):
    t = temp_pipe.unwrap().compute()
    p = ppt_pipe.unwrap().compute()

    fig, ax = plt.subplots(figsize=(12, 4.6))
    t.plot(ax=ax, marker="o", label="maximum temperature")
    ax.axhline(0, linestyle="--", linewidth=1.2, label="0 °C")
    ax2 = ax.twinx()
    p.plot(ax=ax2, alpha=.45)
    ax.set_title("January 2024: Boulder-region temperature and precipi
    ax.set_ylabel("temperature")
    ax2.set_ylabel("precipitation")
    ax.legend(loc="upper left")
    plt.show()

def show_condition(condition, day=13, title="Condition"):
    s = state_field(condition.unwrap())
    fig, axes = plt.subplots(1, 2, figsize=(12, 4.5), constrained_layo
```

```
    s.isel(time=day).astype(int).plot(ax=axes[0], vmin=0, vmax=1)
    axes[0].set_title(title)
    s.mean("time").plot(ax=axes[1], vmin=0, vmax=1)
    axes[1].set_title("Fraction of days true")
    plt.show()

def show_occurrence_sync(p):
    ds = p.unwrap()
    if isinstance(ds, xr.Dataset) and "occurrence_synchrony" in ds:
        z = ds["occurrence_synchrony"].squeeze()
        fig, ax = plt.subplots(figsize=(8.5, 5.2))
        z.plot(ax=ax, vmin=0, vmax=1)
        ax.set_title("Freezing-day synchrony with the center")
        plt.show()

def show_event_sync(timing, duration):
    fig, axes = plt.subplots(1, 2, figsize=(12, 4.8), constrained_layo
    any_plot = False

    if timing is not None:
        ds = timing.unwrap()
        if isinstance(ds, xr.Dataset) and "timing_synchrony" in ds:
            z = ds["timing_synchrony"]
            if "time_window_end" in z.dims:
                z = z.isel(time_window_end=-1)
            z.squeeze().plot(ax=axes[0], vmin=0, vmax=1)
            axes[0].set_title("Event timing synchrony")
            any_plot = True

    if duration is not None:
        ds = duration.unwrap()
        if isinstance(ds, xr.Dataset) and "duration_similarity" in ds:
            z = ds["duration_similarity"]
            if "time_window_end" in z.dims:
                z = z.isel(time_window_end=-1)
            z.squeeze().plot(ax=axes[1], vmin=0, vmax=1)
            axes[1].set_title("Event duration similarity")
            any_plot = True

    if any_plot:
        plt.show()
    else:
        plt.close(fig)

def show_two_conditions(a, b, a_title, b_title, day=13):
    fig, axes = plt.subplots(1, 2, figsize=(11, 4.4), constrained_layo
    state_field(a.unwrap()).isel(time=day).astype(int).plot(ax=axes[0]
    axes[0].set_title(a_title)
    state_field(b.unwrap()).isel(time=day).astype(int).plot(ax=axes[1]
    axes[1].set_title(b_title)
    plt.show()
```

```
def show_lag(p, title, xlabel):
    ds = p.unwrap()
    if isinstance(ds, xr.Dataset) and "coupling_score" in ds:
        score = ds["coupling_score"]
        spatial_dims = [d for d in score.dims if d != "lag"]
        curve = score.median(dim=spatial_dims, skipna=True)
        fig, ax = plt.subplots(figsize=(8.5, 4.5))
        ax.plot(ds["lag"].values, curve.values, marker="o")
        ax.set_title(title)
        ax.set_xlabel(xlabel)
        ax.set_ylabel("median same-pixel coupling")
        ax.grid(alpha=.25)
        plt.show()

def load_gridmet_suite(bbox, start, end):
    specs = [
        ("temperature", {"statistic":"maximum"}),
        ("precipitation", {}),
        ("vpd", {}),
        ("humidity", {"statistic":"maximum"}),
        ("wind", {}),
        ("radiation", {}),
    ]
    cubes, rows = {}, []

    for noun, kwargs in specs:
        try:
            cube = getattr(data, noun)(
                source="gridmet",
                bbox=bbox,
                start=start,
                end=end,
                **kwargs,
            )
            cubes[noun] = cube
            rows.append({
                "noun":noun,
                "status":"loaded",
                "shape":" × ".join(map(str, cube.shape)),
                "units":cube.attrs.get("units",""),
            })
        except Exception as exc:
            rows.append({
                "noun":noun,
                "status":"failed",
                "shape":"",
                "units":str(exc)[:80],
            })

    display(
        pd.DataFrame(rows)
        .style
```

```
        .hide(axis="index")
        .set_caption("Real gridMET noun retrieval")
    )
    return cubes

def show_gridmet_suite(cubes):
    preferred = [
        ("temperature","Temperature"),
        ("precipitation","Precipitation"),
        ("vpd","VPD"),
        ("humidity","Humidity"),
        ("wind","Wind"),
        ("radiation","Radiation"),
    ]
    available = [x for x in preferred if x[0] in cubes]
    if not available:
        return

    n_rows = int(np.ceil(len(available)/3))
    fig, axes = plt.subplots(
        n_rows, 3, figsize=(15, 4.2*n_rows), constrained_layout=True
    )
    axes = np.atleast_1d(axes).ravel()

    for ax, (noun, title) in zip(axes, available):
        cube = cubes[noun]
        day = min(15, cube.sizes["time"]-1)
        cube.isel(time=day).plot(ax=ax)
        ax.set_title(title)

    for ax in axes[len(available):]:
        ax.axis("off")

    fig.suptitle("One landscape, multiple gridMET nouns", fontsize=18)
    plt.show()

def zscore_suite(cubes):
    out, rows = {}, []
    for noun, cube in cubes.items():
        try:
            out[noun] = pipe(cube) | v.zscore(dim="time")
            rows.append({"noun":noun,"verb":"zscore(time)","status":"P
        except Exception as exc:
            rows.append({"noun":noun,"verb":"zscore(time)","status":f"

    display(
        pd.DataFrame(rows)
        .style
        .hide(axis="index")
        .set_caption("Same CubeDynamics verb, different nouns")
    )
    return out
```

```
def show_zscore_suite(zpipes):
    items = list(zpipes.items())
    if not items:
        return

    n_rows = int(np.ceil(len(items)/3))
    fig, axes = plt.subplots(
        n_rows, 3, figsize=(15, 4.2*n_rows), constrained_layout=True
    )
    axes = np.atleast_1d(axes).ravel()

    for ax, (noun, transformed) in zip(axes, items):
        np.abs(transformed.unwrap()).max("time").plot(ax=ax)
        ax.set_title(noun)

    for ax in axes[len(items):]:
        ax.axis("off")

    fig.suptitle("Strongest standardized July departure by noun", font
    plt.show()

def combine_standardized_weather(z_temperature, z_vpd, z_wind):
    z_temperature, z_vpd, z_wind = xr.align(
        z_temperature, z_vpd, z_wind, join="inner"
    )
    out = ((z_temperature + z_vpd + z_wind)/3).rename("weather_extreme
    out.attrs.update({
        "units":"1",
        "description":"Teaching index: mean z-score of temperature, VP
    })
    return out

def show_extreme_weather(condition):
    s = state_field(condition.unwrap())
    fig, axes = plt.subplots(1, 2, figsize=(12, 4.7), constrained_layo
    s.mean("time").plot(ax=axes[0], vmin=0, vmax=1)
    axes[0].set_title("Where was unusual weather frequent?")
    s.mean(("y","x")).plot(ax=axes[1], marker="o")
    axes[1].set_title("When was unusual weather widespread?")
    plt.show()

def show_source_comparison(prism_pipe, gridmet_pipe):
    p = prism_pipe.unwrap().compute()
    g = gridmet_pipe.unwrap().compute() - 273.15
    g.attrs["units"] = "degC"

    common = np.intersect1d(p.time.values, g.time.values)
    p, g = p.sel(time=common), g.sel(time=common)

    fig, axes = plt.subplots(1, 2, figsize=(12.5, 4.7), constrained_la
    p.plot(ax=axes[0], marker="o", label="PRISM")
```

```
    g.plot(ax=axes[0], marker="s", label="gridMET converted to °C")
    axes[0].set_title("Regional daily trajectories")
    axes[0].legend()

    axes[1].scatter(p.values, g.values)
    lo = np.nanmin([p.values.min(), g.values.min()])
    hi = np.nanmax([p.values.max(), g.values.max()])
    axes[1].plot([lo,hi],[lo,hi], linestyle="--")
    axes[1].set_xlabel("PRISM °C")
    axes[1].set_ylabel("gridMET °C")
    axes[1].set_title("Same noun ≠ identical product")
    plt.show()

def rank_ndvi_coverage(ndvi, n=3):
    coverage = ndvi.notnull().mean(("y","x")).compute()
    table = pd.DataFrame({
        "date":pd.to_datetime(ndvi.time.values),
        "map_coverage":np.asarray(coverage.values, dtype=float),
    }).sort_values("map_coverage", ascending=False)

    display(
        table.head(10)
        .style
        .format({"map_coverage":"{:.1%}"})
        .hide(axis="index")
        .set_caption("Sentinel-2 July scenes ranked by map coverage")
    )

    indices = np.argsort(np.asarray(coverage.values))[::-1][:n]
    return np.sort(indices), coverage

def show_ndvi_scenes(ndvi, indices, coverage):
    fig, axes = plt.subplots(
        1, len(indices), figsize=(5*len(indices), 4.8), constrained_la
    )
    axes = np.atleast_1d(axes)

    for ax, idx in zip(axes, indices):
        idx = int(idx)
        scene = ndvi.isel(time=idx)
        fraction = float(coverage.isel(time=idx))
        scene.plot(ax=ax, vmin=-0.2, vmax=1)
        date = str(pd.Timestamp(ndvi.time.values[idx]).date())
        ax.set_title(f"{date}\n{fraction:.0%} map coverage")

    fig.suptitle("Sentinel-2 NDVI: best-covered July observations", fo
    plt.show()

print("Python:", sys.version.split()[0])
print("CubeDynamics:", cd.__version__)
print("Imported from:", cd.__file__)
```

```
Python: 3.11.4
CubeDynamics: 0.1.0rc3
Imported from: /home/jovyan/.local/lib/python3.11/site-packages/cubedyn
amics/__init__.py
```

How to read the code

The visible code is intentionally constrained to a small number of forms:

```
noun = data.temperature(...)
state = (
    pipe(noun)
    | v.threshold_state(...)
)
relationship = (
    pipe(state.unwrap())
    | v.occurrence_synchrony(...)
)
```

Plotting loops, dataframe assembly, and layout code are hidden in helper functions because they are not the scientific grammar being demonstrated.

Visual legend

- **NOUN** = what environmental thing are we observing?
- **SOURCE** = which public product supplies that noun?
- **VERB** = what analytical operation are we authoring?
- **STATE** = what scientific object exists now?
- `|` = “then do this”
- `.unwrap()` = temporarily return to ordinary xarray/Python
- `.explain()` = ask CubeDynamics to translate the pipe back into science

> **Presentation test**
>
> If a scientist cannot read a visible cell aloud as an analysis, the cell is too complicated.

Chapter 1 — Discover the grammar

CubeDynamics begins by naming environmental concepts directly. The same noun may have several source flavors, but sharing a noun does not imply that two products are scientifically interchangeable.

In [3]:

```
sources = data.list_sources()

noun_table = pd.DataFrame(
    [
        {"noun": noun, "source": source}
        for noun, source_list in sources.items()
        for source in source_list
    ]
)

display(
    noun_table
    .sort_values(["noun","source"])
    .style
    .hide(axis="index")
    .set_caption("Built-in environmental nouns and source flavors")
)

print("Selected semantic verbs:")
print([
    name
    for name in (
        "anomaly",
        "variance",
        "zscore",
        "threshold_state",
        "quantile_state",
        "overlap",
        "detect_events",
        "occurrence_synchrony",
        "timing_synchrony",
        "duration_synchrony",
        "sync_with",
        "plot",
    )
    if hasattr(v, name)
])
```

Built-in environmental nouns and source flavors

noun	source
humidity	gridmet
precipitation	gridmet
precipitation	prism
radiation	gridmet
surface_reflectance	sentinel2
temperature	gridmet
temperature	prism
vegetation_index	sentinel2
vpd	gridmet
wind	gridmet

```
Selected semantic verbs:
['anomaly', 'variance', 'zscore', 'threshold_state', 'quantile_state',
'overlap', 'detect_events', 'occurrence_synchrony', 'timing_synchrony',
'duration_synchrony', 'sync_with', 'plot']
```

Story 1 — Working Lands

Scientific question

Where and when did unusually warm July days and days with little or no measured precipitation coincide in central South Dakota?

Interpretation boundary

This is a screening story. The result is exactly the condition we author below, not automatically a drought classification, ecological impact estimate, or causal model.

1A · Fix the scientific scope

In [4]:
```
BBOX_SD = [-101.2, 43.7, -100.4, 44.3]
START_SD = "2024-07-01"
```

```
END_SD = "2024-07-31"
```

temperature → PRISM observation cube
precipitation → PRISM observation cube

In [5]:
```
temperature = data.temperature(
    source="prism",
    statistic="maximum",
    bbox=BBOX_SD,
    start=START_SD,
    end=END_SD,
)

precipitation = data.precipitation(
    source="prism",
    bbox=BBOX_SD,
    start=START_SD,
    end=END_SD,
)

compact_attrs(temperature)
compact_attrs(precipitation)
```

Selected source + semantic metadata

attribute	value
scientific_noun	temperature
source	prism_streaming
source_product	PRISM AN81d/AN91d daily time series
units	degC
semantic_kind	continuous_field
crs	EPSG:4326

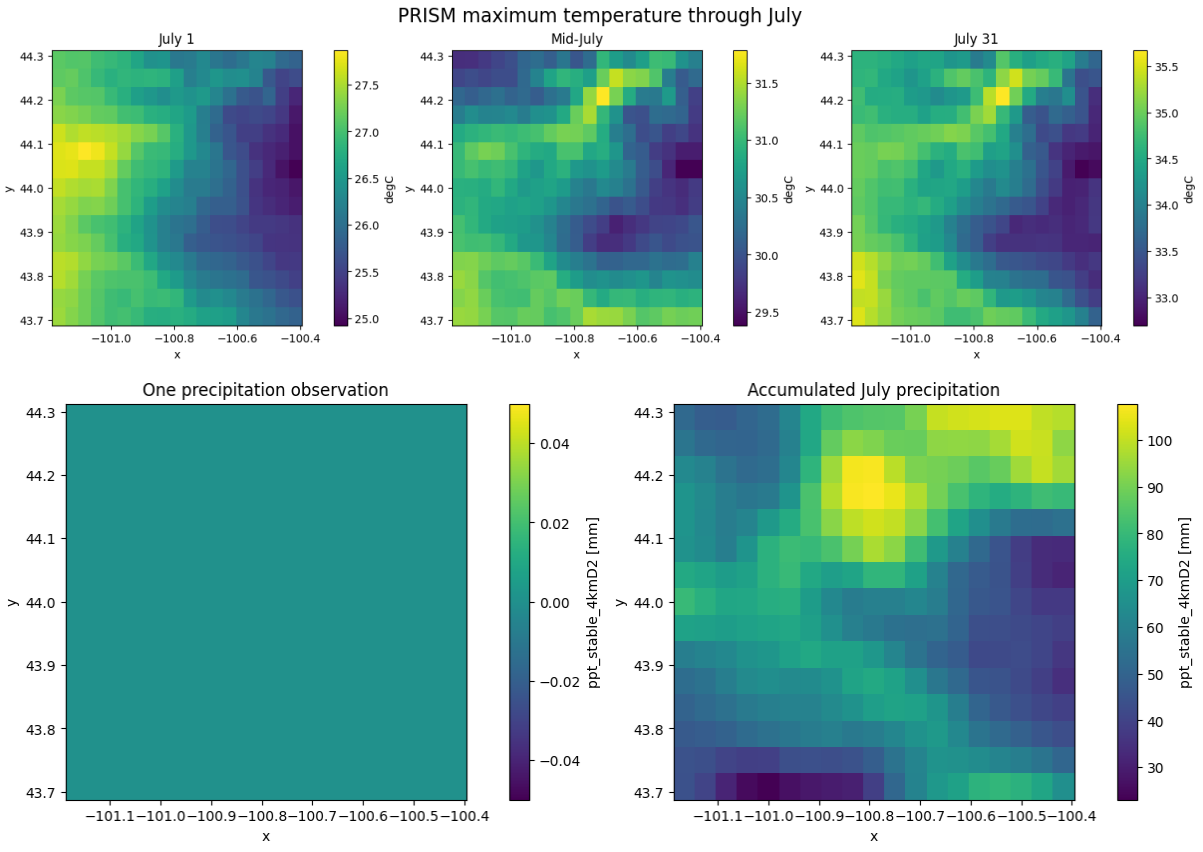
Selected source + semantic metadata

attribute	value
scientific_noun	precipitation
source	prism_streaming
source_product	PRISM AN81d/AN91d daily time series
units	mm
semantic_kind	continuous_field
crs	EPSG:4326

1B · Observe before classifying

Before calling anything *warm* or *dry*, look at the actual fields. These are the measurements every later state depends on.

In [6]:
```
show_working_lands_observations(
    temperature,
    precipitation,
)
```



1C · Make the cube visible

A single map is only one slice. The CubeDynamics renderer lets the audience see the underlying **x × y × time** object.

In [7]:
```
v.plot(
    temperature,
    title="PRISM maximum temperature · July 2024",
)
```

```
Preparing cube… 100%
```

Out[7]:

1D · Transform the observation

```
temperature → anomaly(time) → variance(time) → spatial summary
```

Read it aloud: **start with temperature, remove each cell's July mean, then summarize temporal variability.**

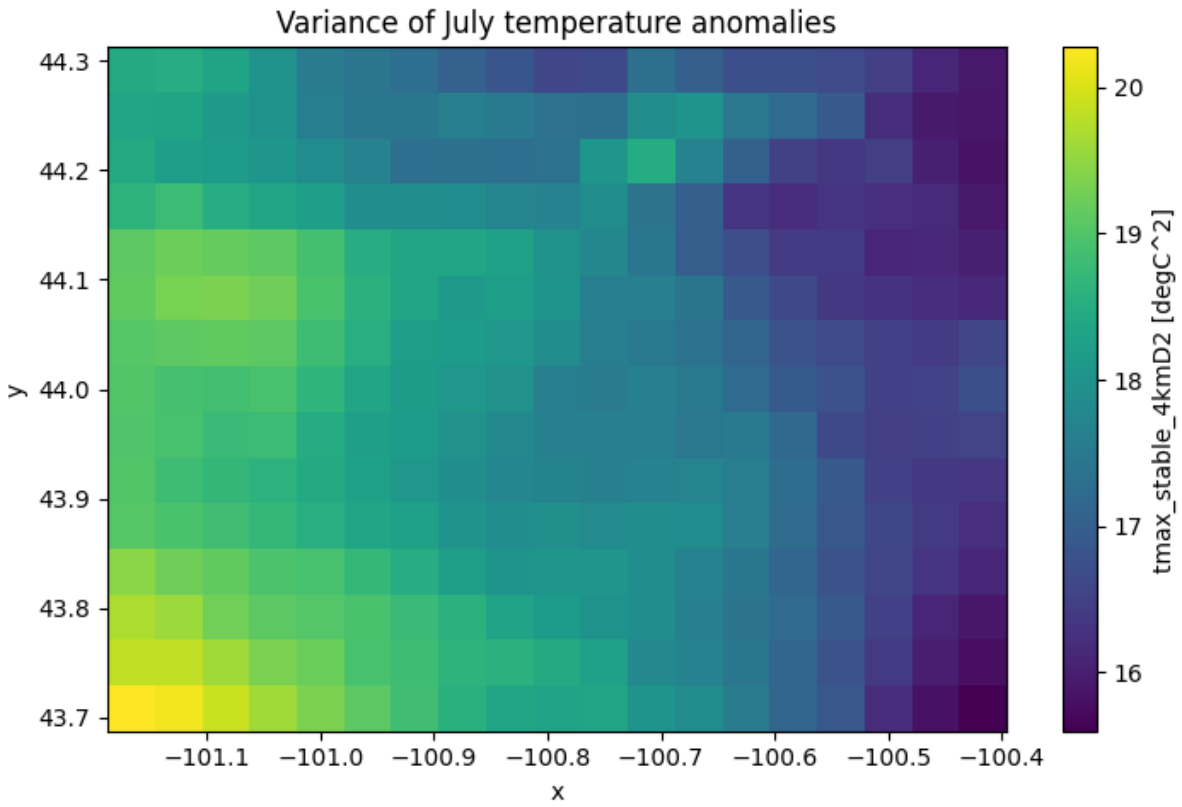
In [8]:
```
temperature_variability = (
    pipe(temperature)
    | v.anomaly(dim="time")
    | v.variance(
        dim="time",
        keep_dim=False,
    )
)

explain(temperature_variability)

show_spatial_summary(
    temperature_variability,
    "Variance of July temperature anomalies",
)
```

What CubeDynamics says this sentence means

```
Your analysis
1. Start with temperature (continuous field) from prism.
2. Calculate departures from the mean over time.
3. Measure variation in values over time.

Current result
- Semantic kind: summary
- Dimensions: y, x
- Units: degC^2

CubeDynamics executed the verbs exactly in the order written;
no step was rewritten.
```

Semantic validation

```
Semantic validation: ready
- INFO: Current result is classified as summary.
- INFO: Tracked dimensions: y, x.
- INFO: Spatial CRS is EPSG:4326.
- INFO: Units are degC^2.
- INFO: Source provenance metadata is present.
```



1E · Turn observations into declared conditions

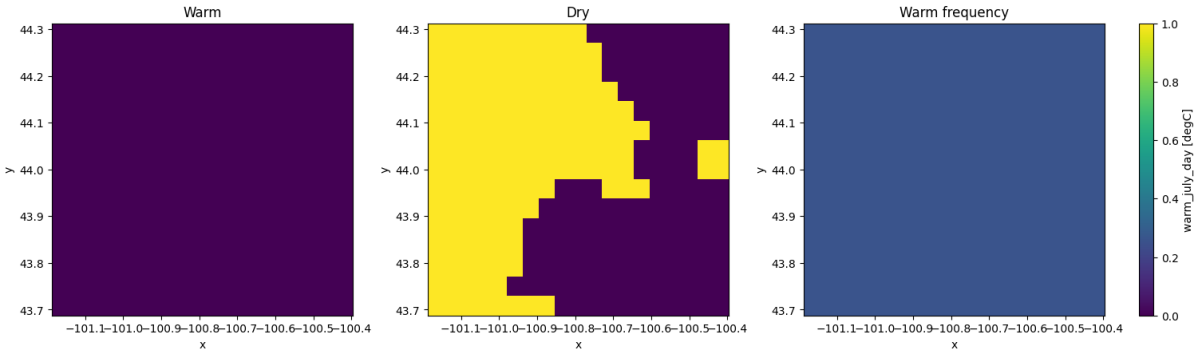
temperature → quantile_state(0.75) → warm condition

precipitation → threshold_state(≤ 0.1 mm) → dry condition

The important move is semantic: the observations did not arrive labeled *warm* or *dry*. We authored those definitions.

In [9]:
```
warm = (
    pipe(temperature)
    | v.quantile_state(
        quantile=0.75,
        direction="above",
        name="warm_july_day",
    )
)

dry = (
```

```
    pipe(precipitation)
    | v.threshold_state(
        threshold=0.1,
        direction="below",
        name="trace_or_no_rain",
    )
)

day = show_conditions(
    warm,
    dry,
)

text_panel("Warm condition", warm.explain())
text_panel("Dry condition", dry.explain())
```



Warm condition

```
Your analysis
1. Start with temperature (continuous field) from prism.
2. Define a condition using quantile 0.75. Reference
population: all 31 selected observations pooled across 'time',
estimated independently at each remaining coordinate.

Current result
- Semantic kind: condition
- Dimensions: time, y, x
- Units: boolean
- Temporal support: interval (day_ending)
- Quantile reference population: all 31 selected observations
pooled across 'time', estimated independently at each remaining
coordinate
```

Dry condition

```
Your analysis

1. Start with precipitation (continuous field) from prism.
2. Define a condition for values below 0.1.

Current result
- Semantic kind: condition
- Dimensions: time, y, x
- Units: boolean
- Temporal support: interval (day_ending)

CubeDynamics executed the verbs exactly in the order written;
no step was rewritten.
```

1F · Let the grammar refuse ambiguity

The daily labels match, but rc3 knows that the declared PRISM temperature and precipitation observation intervals differ. CubeDynamics refuses to silently collapse that distinction.

In [10]:
```
try:
    (
        pipe(warm.unwrap())
        | v.overlap(
            dry.unwrap(),
            name="warm_and_dry",
        )
    )
except Exception as exc:
    panel(
        "Expected temporal-support guardrail",
        f"{type(exc).__name__}: {html.escape(str(exc))}",
        "red",
    )
```

Expected temporal-support guardrail

ValueError: overlap inputs have exact time labels but different known observation intervals. Choose temporal_alignment='labels' to pair the existing labels with a recorded caveat, or 'require_exact_support' to reject the mismatch. CubeDynamics will not shift or resample either input.

1G · Make the scientific choice explicitly

```
warm condition → overlap(dry) → warm ∩
dry → mean(time) → frequency summary
```

For this screening analysis we choose `temporal_alignment="labels"`. CubeDynamics records the caveat and does not shift or resample either input.

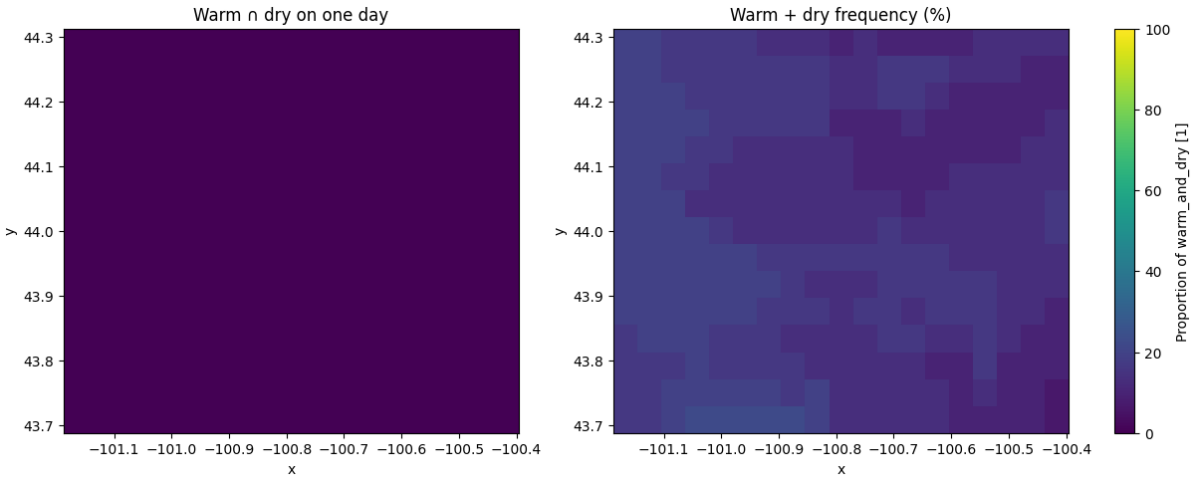
In [11]:
```
joint = (
    pipe(warm.unwrap())
    | v.overlap(
        dry.unwrap(),
        name="warm_and_dry",
        temporal_alignment="labels",
    )
)

hot_dry_frequency = (
    joint
    | v.mean(
        dim="time",
        keep_dim=False,
    )
)

explain(hot_dry_frequency)

frequency_map = show_joint(
    joint,
    hot_dry_frequency,
    day,
)
```

What CubeDynamics says this sentence means

```
Your analysis
1. Start with warm july day (condition) from prism.
2. Identify coincident truth in two exactly aligned conditions.
3. Average values over time.

Current result
- Semantic kind: summary
- Dimensions: y, x
- Units: proportion
- Quantile reference population: all 31 selected observations
pooled across 'time', estimated independently at each remaining
coordinate

Temporal alignment
```

Semantic validation

```
Semantic validation: ready
- INFO: Current result is classified as summary.
- INFO: Tracked dimensions: y, x.
- INFO: Spatial CRS is EPSG:4326.
- INFO: Units are proportion.
- INFO: Source provenance metadata is present.
- WARNING: prism and prism have matching date labels but
different declared observation intervals; policy 'labels' does
not make those intervals equal.
- WARNING: prism and prism have matching date labels but
different declared observation intervals; policy 'labels' does
not make those intervals equal.
```



1H · The derived state is also a cube

The voxels now mean something different from the raw temperature cube. Each voxel answers: **was this place both warm and dry on this day?**

In [12]:
```
v.plot(
    state_field(joint.unwrap()).astype(float),
    title="Warm and dry condition through July",
)
```

```
Preparing cube… 100%
```

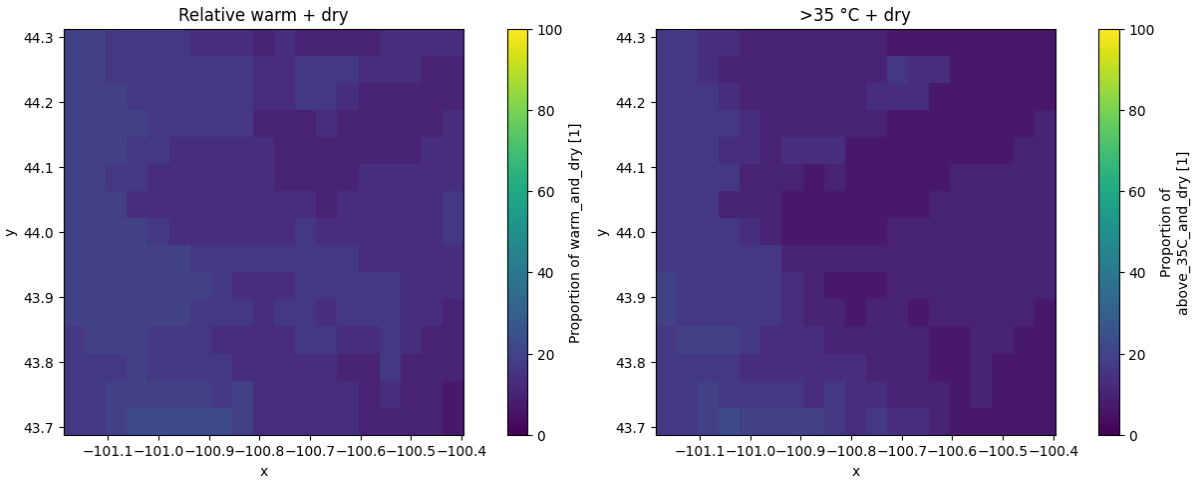
Out[12]:

1l · Change one verb parameter, change the scientific question

temperature → threshold_state(> 35 °C) → absolute hot condition → overlap(dry) → mean(time)

In [13]:
```
absolute_hot = (
    pipe(temperature)
    | v.threshold_state(
```

```
        threshold=35,
        direction="above",
        name="above_35C",
    )
)

absolute_hot_dry = (
    pipe(absolute_hot.unwrap())
    | v.overlap(
        dry.unwrap(),
        name="above_35C_and_dry",
        temporal_alignment="labels",
    )
    | v.mean(
        dim="time",
        keep_dim=False,
    )
)

show_frequency_comparison(
    frequency_map,
    state_field(absolute_hot_dry.unwrap()),
)
```



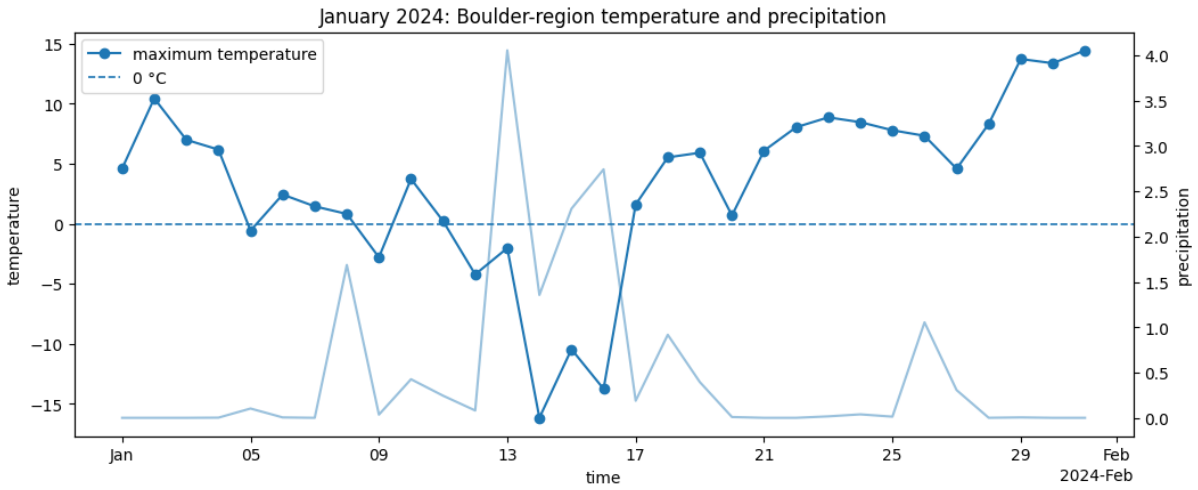
Story 2 — Boulder Cold Snap

Scientific question

Can a real cold spell move cleanly from observation → condition → event → synchrony → lagged relationship?

2A · Observe the region through time

In [14]:
```
BBOX_CO = [-105.55, 39.75, -104.85, 40.35]
START_CO = "2024-01-01"
END_CO = "2024-01-31"

cold_temperature = data.temperature(
    source="prism",
    statistic="maximum",
    bbox=BBOX_CO,
    start=START_CO,
    end=END_CO,
)

cold_precipitation = data.precipitation(
    source="prism",
    bbox=BBOX_CO,
    start=START_CO,
    end=END_CO,
)

regional_temperature = (
    pipe(cold_temperature)
    | v.mean(
        dim=("y","x"),
        keep_dim=False,
    )
)

regional_precipitation = (
    pipe(cold_precipitation)
    | v.mean(
        dim=("y","x"),
        keep_dim=False,
    )
)

show_regional_weather(
    regional_temperature,
    regional_precipitation,
)
```



2B · See the cold snap as a cube

In [15]:
```
v.plot(
    cold_temperature,
    title="Boulder-region daily maximum temperature · January 2024",
)
```

```
Preparing cube… 100%
```

Out[15]:

2C · Measurement → condition

temperature → threshold_state(< 0 °C) → freezing condition

In [16]:
```
cold = (
    pipe(cold_temperature)
    | v.threshold_state(
        threshold=0,
        direction="below",
        name="freezing_day",
    )
)
```

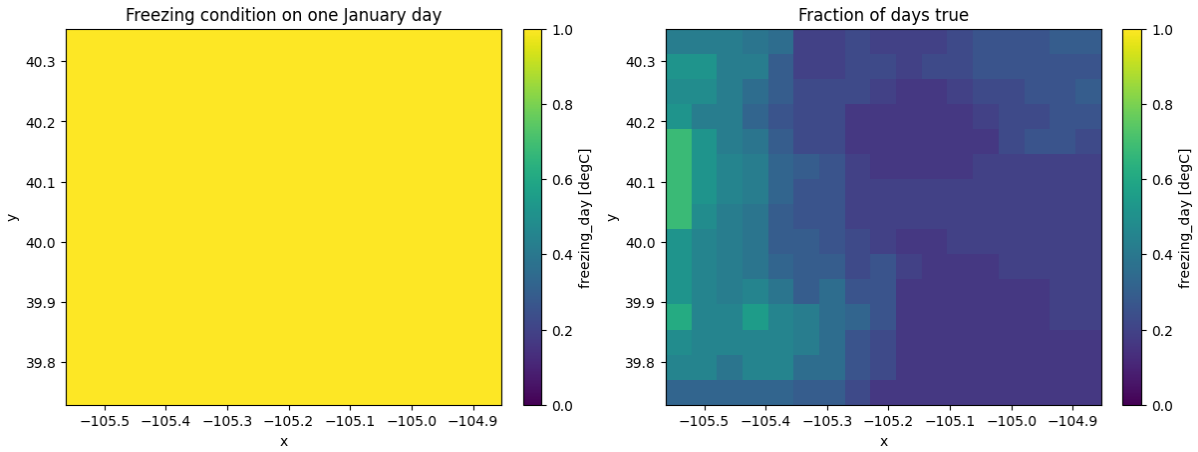
```
explain(cold)

show_condition(
    cold,
    day=13,
    title="Freezing condition on one January day",
)
```

What CubeDynamics says this sentence means

```
Your analysis
1. Start with temperature (continuous field) from prism.
2. Define a condition for values below 0.

Current result
- Semantic kind: condition
- Dimensions: time, y, x
- Units: boolean
- Temporal support: interval (day_ending)

CubeDynamics executed the verbs exactly in the order written;
no step was rewritten.
```

Semantic validation

```
Semantic validation: ready
- INFO: Current result is classified as condition.
- INFO: Tracked dimensions: time, y, x.
- INFO: Time coordinates are ordered.
- INFO: Observation temporal support is declared as interval
(day_ending).
- INFO: Spatial CRS is EPSG:4326.
- INFO: Units are boolean.
- INFO: Source provenance metadata is present.
```



2D · Condition → spatial relationship

freezing
condition → occurrence_synchrony → spatial relationship

Now the result is not temperature and not a condition. It is a relationship describing how often other locations share freezing conditions with the center.

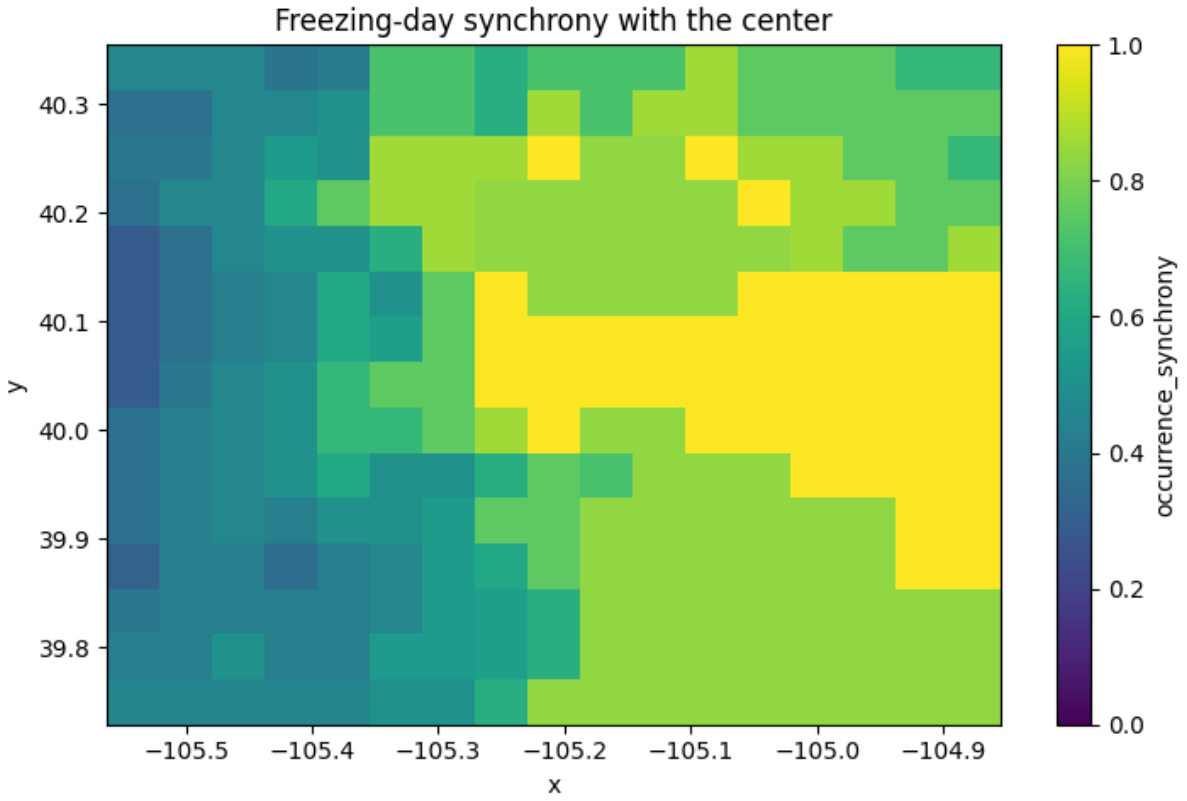
In [17]:
```
cold_occurrence = (
    pipe(cold.unwrap())
    | v.occurrence_synchrony(
        spatial_mode="reference",
        reference="center",
        method="jaccard",
    )
)

explain(cold_occurrence)

show_occurrence_sync(
    cold_occurrence,
)
```

What CubeDynamics says this sentence means

```
Your analysis
1. Start with freezing day (condition) from prism.
2. Compare co-occurrence among state cubes.

Current result
- Semantic kind: relationship
- Dimensions: time_window_end, y, x
- Units: not declared
- Temporal support: not verified

CubeDynamics executed the verbs exactly in the order written;
no step was rewritten.
```

Semantic validation

```
Semantic validation: ready
- INFO: Current result is classified as relationship.
- INFO: Tracked dimensions: time_window_end, y, x.
- INFO: Time coordinates are ordered.
- CHECK: A time coordinate is present, but observation temporal
support is not verified.
- CHECK: Spatial dimensions are present, but CRS metadata is
not declared.
- CHECK: Units are not declared for the current result.
- CHECK: Source provenance metadata is not declared.
```



2E · Let CubeDynamics render the relationship

In [18]:
```
v.plot(
    cold_occurrence.unwrap()["occurrence_synchrony"],
    title="Freezing-day occurrence synchrony",
)
```

```
Preparing cube… 100%
```

Out[18]:

2F · Condition → event

```
freezing condition → detect_events → cold events
```

A condition is true or false at each time and place. An event groups consecutive

true periods into episodes with start, end, duration, and event-level metrics.

In [19]:
```
cold_events = (
    pipe(cold.unwrap())
    | v.detect_events(
        min_duration=2,
        max_gap=0,
    )
)

explain(cold_events)

event_result = cold_events.unwrap()
display(event_result.catalog.head(12))
```

What CubeDynamics says this sentence means

```
Your analysis
1. Start with freezing day (condition) from prism.
2. Group consecutive true periods into events (minimum duration
2).

Current result
- Semantic kind: event
- Dimensions: time, y, x
- Units: boolean
- Temporal support: interval (day_ending)
- Event scope: local_cell
- One event row means one contiguous event instance at one
spatial grid cell.
```

Semantic validation

```
Semantic validation: ready
- INFO: Current result is classified as event.
- INFO: Tracked dimensions: time, y, x.
- INFO: Time coordinates are ordered.
- INFO: Observation temporal support is declared as interval
(day_ending).
- INFO: Spatial CRS is EPSG:4326.
- INFO: Units are boolean.
- INFO: Source provenance metadata is present.
- INFO: Event catalog scope is local_cell; one row means one
contiguous event instance at one spatial grid cell.
```

	event_id	y	x	y_index	x_index	start	end	duration
0	1	40.333333	-105.541667	0	0	2024-01-05	2024-01-17	13
1	2	40.333333	-105.500000	0	1	2024-01-05	2024-01-17	13
2	3	40.333333	-105.458333	0	2	2024-01-05	2024-01-17	13
3	4	40.333333	-105.416667	0	3	2024-01-06	2024-01-17	12
4	5	40.333333	-105.375000	0	4	2024-01-06	2024-01-09	4
5	6	40.333333	-105.375000	0	4	2024-01-11	2024-01-17	7
6	7	40.333333	-105.333333	0	5	2024-01-12	2024-01-16	5
7	8	40.333333	-105.291667	0	6	2024-01-12	2024-01-16	5
8	9	40.333333	-105.250000	0	7	2024-01-12	2024-01-16	5
9	10	40.333333	-105.208333	0	8	2024-01-14	2024-01-16	3
10	11	40.333333	-105.166667	0	9	2024-01-14	2024-01-16	3
11	12	40.333333	-105.125000	0	10	2024-01-14	2024-01-16	3

2G · Events support different relationship questions

cold events → timing_synchrony → timing relationship

cold events → duration_synchrony → duration relationship

In [ ]:
```
timing_synchrony = (
    pipe(cold_events.unwrap())
    | v.timing_synchrony(
        spatial_mode="neighbors",
        radius_km=75,
        match_tolerance="3D",
    )
)

duration_synchrony = (
    pipe(cold_events.unwrap())
    | v.duration_synchrony(
        spatial_mode="neighbors",
        radius_km=75,
        match_tolerance="3D",
        min_matched_events=1,
    )
)

show_event_sync(
    timing_synchrony,
    duration_synchrony,
)
```

2H · Add a second condition and ask about lag

precipitation → threshold_state(> 1 mm) → wet condition

freezing condition → sync_with(wet, lags) → lagged relationship

In [ ]:
```
wet = (
    pipe(cold_precipitation)
    | v.threshold_state(
        threshold=1.0,
        direction="above",
        name="wet_day",
    )
)

show_two_conditions(
    cold,
    wet,
    "Cold",
    "Wet",
    day=13,
)

cold_wet = (
    pipe(cold.unwrap())
    | v.sync_with(
        wet.unwrap(),
        synchrony="occurrence",
        spatial_relation="same_pixel",
        lags=["0D","1D","2D","3D"],
    )
)

explain(cold_wet)

show_lag(
    cold_wet,
    "Cold–wet occurrence coupling by lag",
    "wet-condition lag",
)
```

Story 3 — Multivariate Weather

Scientific question

Can the same analytical grammar move across several environmental nouns while preserving units, source identity, and interpretation?

> **Why this story matters**
>
> A reusable verb should mean the same analytical thing when applied to different nouns. The nouns themselves should remain distinct.

3A · Load a source-qualified environmental vocabulary

In [ ]:
```
gridmet = load_gridmet_suite(
    bbox=BBOX_SD,
    start=START_SD,
    end=END_SD,
)

show_gridmet_suite(
    gridmet,
)
```

3B · Reuse one CubeDynamics verb across several nouns

each gridMET noun → zscore(time) → standardized continuous field

In [ ]:
```
standardized = zscore_suite(
    gridmet,
)

show_zscore_suite(
    standardized,
)
```

3C · One intentional interoperability bridge

CubeDynamics does not need to own every equation. We use ordinary xarray once to combine three already-standardized fields into a teaching index, then immediately return to the grammar.

multivariate field → quantile_state(0.90) → extreme-weather condition

In [ ]:
```
weather_extremeness = combine_standardized_weather(
    standardized["temperature"].unwrap(),
    standardized["vpd"].unwrap(),
    standardized["wind"].unwrap(),
)

extreme_weather = (
    pipe(weather_extremeness)
    | v.quantile_state(
        quantile=0.90,
        direction="above",
        name="upper_decile_weather_extremeness",
    )
)

explain(extreme_weather)

show_extreme_weather(
    extreme_weather,
)
```

3D · The derived continuous field is also a cube

In [ ]:
```
v.plot(
    weather_extremeness,
    title="Temperature + VPD + wind · standardized extremeness",
)
```

3E · Same noun, different source flavor

PRISM temperature → mean(space) → regional series
gridMET temperature → mean(space) → regional series

We compare the two only after reducing each on its own native grid. Sharing the noun `temperature` does not declare equivalence.

In [ ]:
```
prism_temperature_series = (
    pipe(temperature)
    | v.mean(
        dim=("y","x"),
        keep_dim=False,
    )
)

gridmet_temperature_series = (
    pipe(gridmet["temperature"])
    | v.mean(
        dim=("y","x"),
        keep_dim=False,
    )
)

show_source_comparison(
    prism_temperature_series,
    gridmet_temperature_series,
)
```

Story 4 — Remote Sensing

Scientific question

Can the same vocabulary move from meteorological cubes to repeated satellite observations of vegetation?

Presentation choice

We use July 2024 and rank Sentinel-2 acquisitions by map coverage so the figure shows the three most spatially complete scenes rather than arbitrary catalog order.

`vegetation_index` → Sentinel-2 NDVI cube

In [ ]:
```
ndvi = data.vegetation_index(
    source="sentinel2",
    index="ndvi",
    lat=40.015,
    lon=-105.2705,
```

```
    start="2024-07-01",
    end="2024-07-31",
)

compact_attrs(ndvi)

best_scenes, coverage = rank_ndvi_coverage(
    ndvi,
    n=3,
)

show_ndvi_scenes(
    ndvi,
    best_scenes,
    coverage,
)
```

4A · Let the same renderer expose the satellite cube

In [ ]:
```
v.plot(
    ndvi,
    title="Sentinel-2 NDVI through July 2024",
)
```

Epilogue — Four paths through one grammar

Working Lands

`observation → condition → overlap → summary`

Cold Snap

`observation → condition → event → synchrony → lagged relationship`

Multivariate Weather

`multiple nouns → shared transformation → composition → condition → source comparison`

Remote Sensing

`source-qualified noun → repeated satellite observation cube`

> **What CubeDynamics is adding**

> The package is not replacing xarray. It is making the scientific structure of the analysis visible enough to inspect, explain, validate, and discuss.

Final takeaway

The goal is not less science. It is less hidden science.

The most important recurring visual in this notebook is:

```
result = (
    pipe(noun_or_state)
    | v.verb(...)
    | v.verb(...)
)
```

A good CubeDynamics pipe should let another scientist answer:

- What environmental thing did we start with?
- Which source supplied it?
- What decisions were made, and in what order?
- What kind of scientific object exists now?
- Can I inspect the trace and evidence?

noun → source → pipe → verb → state → trace → evidence