Comparison of Two Time Series of Maps 1.0

This notebook implements the framework from the article "Foundational concepts and equations to compare two time series of maps" to quantify and visualize agreement and change between two temporal map series. Using toy data, it defines modular Python functions to compute presence-agreement components, gains and losses, and full-extent change metrics, and produces clear visualizations and exportable results for reproducible analysis.

Table of Contents

1. Environment Setup

1.1 Install Dependencies

In [ ]:
```
# Install packages needed for:
# - numeric arrays and fast math (numpy)
# - tabular data manipulation (pandas)
# - plotting charts and graphs (matplotlib)
# - reading and writing raster map files (rasterio)
# - handling multi-dimensional map arrays with geospatial coordinates (xarray, rioxarray)
# - exporting tables to Excel (openpyxl)
# - showing progress bars in long loops (tqdm)
%pip install numpy pandas matplotlib rasterio xarray rioxarray openpyxl tqdm
```

1.2 Import Libraries

In [2]:
```
import numpy as np
import pandas as pd
from IPython.display import display
import matplotlib.pyplot as plt
import rasterio
from rasterio.transform import from_origin
import xarray as xr
import rioxarray
from tqdm import tqdm
import openpyxl
import os
```

1.3 Define Constants & Settings

In this section we set up the main parameters for the notebook. We fix a random seed so that toy data are reproducible, specify the dimensions of our toy time series, and define placeholder paths and filenames for when real raster inputs and outputs are used.

In [3]:
```
# Directories
input_dir = r"C:\Users\AntFonseca\github\compare-time-series\input"
output_dir = r"C:\Users\AntFonseca\github\compare-time-series\output"

# Output filenams
metrics_excel = "presence_change_metrics.xlsx"
```

2. Toy Data Input Format

2.1 Generate or Load Toy Time Series Array

In this section we build the toy data arrays exactly as in the article example.

In [4]:
```
# Dimensions matching the article's toy example
num_time_points = 3   # number of time points
num_pixels = 2        # number of pixels in each snapshot

# toy presence values from the article plot:
# toy_data_x[t, n] = presence of reference series at time point t, pixel n
# toy_data_y[t, n] = presence of comparison series at time point t, pixel n

toy_data_x = np.array([
    [2, 5],   # t = 0: reference pixel1=2, pixel2=5
    [0, 4],   # t = 1: reference pixel1=0, pixel2=4
    [5, 1],   # t = 2: reference pixel1=5, pixel2=1
])

toy_data_y = np.array([
    [4, 1],   # t = 0: comparison pixel1=4, pixel2=1
    [1, 5],   # t = 1: comparison pixel1=1, pixel2=5
    [0, 3],   # t = 2: comparison pixel1=0, pixel2=3
])
```

2.2 Export Toy Data as Raster Files

Here we write each map layer of our toy arrays to single-band GeoTIFFs in the input folder. These rasters will later be read back in exactly like real map inputs.

In [5]:
```
# Ensure input directory exists
os.makedirs(input_dir, exist_ok=True)

# Raster metadata for a 1×num_pixels image, without CRS
height = 1
width = num_pixels
transform = from_origin(0, num_pixels, 1, 1)  # top-left corner at (0, num_pixels), pixel siz
meta = {
    "driver": "GTiff",
    "height": height,
    "width": width,
    "count": 1,
    "dtype": toy_data_x.dtype,
    "transform": transform
```

```
}

# write reference series rasters (toy_data_x)
for t in range(num_time_points):
    out_path = os.path.join(input_dir, f"toy_data_x_time{t}.tif")
    with rasterio.open(out_path, "w", **meta) as dst:
        dst.write(toy_data_x[t][np.newaxis, :], 1)

# write comparison series rasters (toy_data_y)
for t in range(num_time_points):
    out_path = os.path.join(input_dir, f"toy_data_y_time{t}.tif")
    with rasterio.open(out_path, "w", **meta) as dst:
        dst.write(toy_data_y[t][np.newaxis, :], 1)
```

3. Presence Agreement Components (Eqs 1–12)

In this section we compute the core presence-agreement metrics—hits, misses, false alarms, spatial differences, and temporal differences—for each pixel at each time point, following Equations 1–12 of the article.

3.1 Define Presence Variables:

We load the reference (`p_x`) and comparison (`p_y`) series into two arrays of shape `(num_time_points, num_pixels)` .
Each element `p_x[t, n]` (or `p_y[t, n]`) holds the presence value at time point `t` and pixel `n` .

In [6]:
```
# Gather and sort the toy-data raster filenames
x_files = sorted([
    os.path.join(input_dir, f)
    for f in os.listdir(input_dir)
    if f.startswith("toy_data_x_time")
])
y_files = sorted([
    os.path.join(input_dir, f)
    for f in os.listdir(input_dir)
    if f.startswith("toy_data_y_time")
])

# Initialize presence arrays
p_x = np.zeros((num_time_points, num_pixels), dtype=toy_data_x.dtype)
p_y = np.zeros((num_time_points, num_pixels), dtype=toy_data_y.dtype)

# Load each raster layer into the arrays
for t, fp in enumerate(x_files):
    with rasterio.open(fp) as src:
        # read band 1 and flatten to a 1D array of length num_pixels
        p_x[t] = src.read(1).flatten()

for t, fp in enumerate(y_files):
    with rasterio.open(fp) as src:
        p_y[t] = src.read(1).flatten()

# Print results for verification
print("Loaded reference presence (p_x):")
print(p_x)
print("\nLoaded comparison presence (p_y):")
print(p_y)
```

```
Loaded reference presence (p_x):
[[2 5]
 [0 4]
 [5 1]]

Loaded comparison presence (p_y):
[[4 1]
 [1 5]
 [0 3]]
```

3.2 Implement Hit, Miss, False Alarm, Spatial Difference, and Temporal Difference Functions

In this subsection we define five functions that implement Equations 1–12 for presence at each time point and pixel:

- **hit(px, py):** amount of shared presence
- **miss(px, py):** amount of reference-only presence
- **false_alarm(px, py):** amount of comparison-only presence
- **spatial_diff(px, py):** difference in magnitude when both series have presence
- **temporal_diff(px_prev, px, py_prev, py):** timing differences across consecutive time points

Each function accepts an input array of presence values with dimensions `(num_time_points, num_pixels)` and returns a new array with the same dimensions.

In [7]:
```python
def hit(px, py):
    """
    Presence hit: shared magnitude of presence at each time and pixel.
    h[t,n] = min(px[t,n], py[t,n])
    """
    return np.minimum(px, py)

def miss(px, py):
    """
    Presence miss: reference-only magnitude.
    m[t,n] = max(px[t,n] - py[t,n], 0)
    """
    return np.clip(px - py, a_min=0, a_max=None)

def false_alarm(px, py):
    """
    Presence false alarm: comparison-only magnitude.
    f[t,n] = max(py[t,n] - px[t,n], 0)
    """
    return np.clip(py - px, a_min=0, a_max=None)

def spatial_diff(px, py):
    """
    Spatial difference: magnitude difference when both are present.
    u[t,n] = |px[t,n] - py[t,n]| if px>0 and py>0, else 0.
    """
    diff = np.abs(px - py)
    mask = (px > 0) & (py > 0)
    return diff * mask

def temporal_diff(px_prev, px, py_prev, py):
    """
    Temporal difference: magnitude change-timing difference.
    v[t,n] = |(px-px_prev) - (py-py_prev)|, zero at t=0.
    """
    delta_x = px - px_prev
    delta_y = py - py_prev
```

```
    td = np.abs(delta_x - delta_y)
    td[0, :] = 0
    return td
```

3.3 Compute Component Arrays per Time & Pixel

In this subsection we apply our five presence-agreement functions to the loaded arrays `p_x` and `p_y`. This produces one array per component—hits, misses, false alarms, spatial differences, and temporal differences—each with shape `(num_time_points, num_pixels)`.

In [8]:
```
# Section 3.3: Compute per-timepoint summary metrics from totals

# 1) total presence per time point
px_sum       = p_x.sum(axis=1)  # e.g. [7,4,6]
py_sum       = p_y.sum(axis=1)  # e.g. [5,6,3]

# 2) hit: sum of shared magnitude
hits_tp      = np.minimum(p_x, p_y).sum(axis=1)
#              [min(2,4)+min(5,1), …] → [3,4,1]

# 3) space_diff: overlap in magnitude beyond hit
space_diff   = np.minimum(px_sum, py_sum) - hits_tp
#              [min(7,5)-3, min(4,6)-4, min(6,3)-1] → [2,0,2]

# 4) miss: excess reference presence
misses_tp    = np.clip(px_sum - py_sum, a_min=0, a_max=None)
#              [7-5, 4-6, 6-3] clipped → [2,0,3]

# 5) false_alarm: excess comparison presence
false_tp     = np.clip(py_sum - px_sum, a_min=0, a_max=None)
#              [5-7, 6-4, 3-6] clipped → [0,2,0]

# 6) time_diff: always zero at each time point
time_diff_tp = np.zeros_like(hits_tp, dtype=int)

print(hits_tp)       # [3, 4, 1]
print(space_diff)    # [2, 0, 2]
print(misses_tp)     # [2, 0, 3]
print(false_tp)      # [0, 2, 0]
print(time_diff_tp)  # [0, 0, 0]
```

```
[3 4 1]
[2 0 2]
[2 0 3]
[0 2 0]
[0 0 0]
```

4. Change Components: Gains & Losses (Eqs 13–40)

In this section we quantify change between consecutive time points by decomposing it into **gains** (positive increases) and **losses** (negative decreases) for both series. We reuse the hit/miss/false-alarm framework from presence to define component functions for gains and losses, then aggregate them.

4.1 Define Gain & Loss Variables:

In this step we read each pair of consecutive raster maps from the input folder for both series (reference = x, comparison = y) and compute:

- **Gain** at each pixel and interval:
  the amount by which the pixel's value increased from the previous time point (zero if there was no

increase).

- **Loss** at each pixel and interval:
  the amount by which the pixel's value decreased from the previous time point (zero if there was no decrease).

- **First time point**:
  since there is no "previous" layer at (t=0), all gains and losses are set to zero for that time.

The computed arrays— `g_x`, `g_y` for gains and `l_x`, `l_y` for losses—have the same dimensions as the presence arrays and will be passed to the gain- and loss-component functions in the following subsections.

In [9]:
```
# Build "previous" presence arrays by shifting down and padding the first row with zeros
px_prev = np.vstack([np.zeros((1, num_pixels), dtype=p_x.dtype), p_x[:-1]])
py_prev = np.vstack([np.zeros((1, num_pixels), dtype=p_y.dtype), p_y[:-1]])

# Compute per-interval gains (increase amount) and losses (decrease amount)
g_x = np.clip(p_x - px_prev, a_min=0, a_max=None)
g_y = np.clip(p_y - py_prev, a_min=0, a_max=None)

l_x = np.clip(px_prev - p_x, a_min=0, a_max=None)
l_y = np.clip(py_prev - p_y, a_min=0, a_max=None)

# Print results for verification
print("Reference gains (g_x):\n", g_x)
print("Comparison gains (g_y):\n", g_y)
print("Reference losses (l_x):\n", l_x)
print("Comparison losses (l_y):\n", l_y)
```

```
Reference gains (g_x):
 [[2 5]
 [0 0]
 [5 0]]
Comparison gains (g_y):
 [[4 1]
 [0 4]
 [0 0]]
Reference losses (l_x):
 [[0 0]
 [2 1]
 [0 3]]
Comparison losses (l_y):
 [[0 0]
 [3 0]
 [1 2]]
```

4.2 Implement Gain-Component Functions

Define functions that calculate gain hits, gain misses, gain false alarms, spatial differences, and temporal differences by substituting presence (`p`) with gains (`g`).

In [10]:
```
import numpy as np

# 1) Compute per-interval raw change for each series
#    delta_x[t,n] = p_x[t+1,n] - p_x[t,n]
#    delta_y[t,n] = p_y[t+1,n] - p_y[t,n]
delta_x = p_x[1:] - p_x[:-1]
delta_y = p_y[1:] - p_y[:-1]

print("delta_x shape:", delta_x.shape)  # should be (2, 2)
print("delta_y shape:", delta_y.shape)
```

```python
# 2) Split raw change into positive gains and positive losses
#    g_x = positive part of delta_x; l_x = positive part of -delta_x
g_x = np.clip(delta_x, a_min=0, a_max=None)
l_x = np.clip(-delta_x, a_min=0, a_max=None)
g_y = np.clip(delta_y, a_min=0, a_max=None)
l_y = np.clip(-delta_y, a_min=0, a_max=None)

print("g_x shape:", g_x.shape)  # should be (2, 2)
print("g_y shape:", g_y.shape)



# 3) Build “previous” gain arrays by shifting and padding
g_x_prev = np.vstack([
    np.zeros((1, g_x.shape[1]), dtype=g_x.dtype),
    g_x[:-1]
])
g_y_prev = np.vstack([
    np.zeros((1, g_y.shape[1]), dtype=g_y.dtype),
    g_y[:-1]
])

# 4) Define gain-component functions
def gain_hit(gx, gy):
    """Shared positive change at each interval and pixel."""
    return np.minimum(gx, gy)

def gain_miss(gx, gy):
    """Reference-only positive change at each interval and pixel."""
    return np.clip(gx - gy, a_min=0, a_max=None)

def gain_false_alarm(gx, gy):
    """Comparison-only positive change at each interval and pixel."""
    return np.clip(gy - gx, a_min=0, a_max=None)

def gain_spatial_diff(gx, gy):
    """Magnitude difference when both series have positive change."""
    diff = np.abs(gx - gy)
    mask = (gx > 0) & (gy > 0)
    return diff * mask

def gain_temporal_diff(gx_prev, gx, gy_prev, gy):
    """
    Gain temporal difference per interval:
    always zero (we only count timing mismatches in the Sum/Extent).
    """
    return np.zeros_like(gx, dtype=int)

# 5) Compute gain-component arrays
h_g = gain_hit(g_x, g_y)
m_g = gain_miss(g_x, g_y)
f_g = gain_false_alarm(g_x, g_y)
u_g = gain_spatial_diff(g_x, g_y)
v_g = gain_temporal_diff(g_x_prev, g_x, g_y_prev, g_y)

# 6) Print for verification
print("Gain hits (h_g):\n", h_g)
print("Gain misses (m_g):\n", m_g)
print("Gain false alarms (f_g):\n", f_g)
print("Gain spatial diffs (u_g):\n", u_g)
print("Gain temporal diffs (v_g):\n", v_g)
```

```
delta_x shape: (2, 2)
delta_y shape: (2, 2)
g_x shape: (2, 2)
g_y shape: (2, 2)
Gain hits (h_g):
 [[0 0]
 [0 0]]
Gain misses (m_g):
 [[0 0]
 [5 0]]
Gain false alarms (f_g):
 [[0 4]
 [0 0]]
Gain spatial diffs (u_g):
 [[0 0]
 [0 0]]
Gain temporal diffs (v_g):
 [[0 0]
 [0 0]]
```

4.3 Implement Loss-Component Functions

Similarly, define loss hits, loss misses, loss false alarms, spatial differences, and temporal differences by substituting presence (`p`) with losses (`l`).

In [11]:

```python
# Build “previous” loss arrays by shifting down one time step and padding the first row with z
l_x_prev = np.vstack([
    np.zeros((1, l_x.shape[1]), dtype=l_x.dtype),
    l_x[:-1]
])
l_y_prev = np.vstack([
    np.zeros((1, l_y.shape[1]), dtype=l_y.dtype),
    l_y[:-1]
])

#  Define loss-component functions
def loss_hit(lx, ly):
    """Shared negative change at each interval and pixel."""
    return np.minimum(lx, ly)

def loss_miss(lx, ly):
    """Reference-only loss: positive part of (lx - ly)."""
    return np.clip(lx - ly, a_min=0, a_max=None)

def loss_false_alarm(lx, ly):
    """Comparison-only loss: positive part of (ly - lx)."""
    return np.clip(ly - lx, a_min=0, a_max=None)

def loss_spatial_diff(lx, ly):
    """Magnitude difference where both series have loss."""
    diff = np.abs(lx - ly)
    mask = (lx > 0) & (ly > 0)
    return diff * mask

def loss_temporal_diff(lx_prev, lx, ly_prev, ly):
    """Timing difference of loss between consecutive intervals."""
    td = np.abs((lx - lx_prev) - (ly - ly_prev))
    td[0, :] = 0
    return td

# Compute the actual loss-component arrays
h_l = loss_hit(l_x, l_y)
m_l = loss_miss(l_x, l_y)
f_l = loss_false_alarm(l_x, l_y)
```

```
u_l = loss_spatial_diff(l_x, l_y)
v_l = loss_temporal_diff(l_x_prev, l_x, l_y_prev, l_y)

# Print results for verification
print("Loss hits (h_l):\n", h_l)
print("Loss misses (m_l):\n", m_l)
print("Loss false alarms (f_l):\n", f_l)
print("Loss spatial diffs (u_l):\n", u_l)
print("Loss temporal diffs (v_l):\n", v_l)
```

```
Loss hits (h_l):
 [[2 0]
 [0 2]]
Loss misses (m_l):
 [[0 1]
 [0 1]]
Loss false alarms (f_l):
 [[1 0]
 [1 0]]
Loss spatial diffs (u_l):
 [[1 0]
 [0 1]]
Loss temporal diffs (v_l):
 [[0 0]
 [0 0]]
```

4.4 Aggregate Gain & Loss Components

Sum each gain and loss component across pixels (for each interval) and across intervals (for each pixel) to create summary change metrics.

In [12]:
```
# Section 4.4: Compute per-interval summary of gain & loss components

# Gains per interval (sum across pixels)
gain_hit         = h_g.sum(axis=1)
gain_space_diff  = u_g.sum(axis=1)
gain_time_diff   = v_g.sum(axis=1)
gain_miss        = m_g.sum(axis=1)
gain_false_alarm = f_g.sum(axis=1)

# Losses per interval (sum across pixels)
loss_hit         = h_l.sum(axis=1)
loss_space_diff  = u_l.sum(axis=1)
loss_time_diff   = v_l.sum(axis=1)
loss_miss        = m_l.sum(axis=1)
loss_false_alarm = f_l.sum(axis=1)

# Print to verify against the paper's example
print("Gain Hit:            ", gain_hit)
print("Gain Space Difference:", gain_space_diff)
print("Gain Time Difference: ", gain_time_diff)
print("Gain Miss:           ", gain_miss)
print("Gain False Alarm:    ", gain_false_alarm)
print()
print("Loss Hit:            ", loss_hit)
print("Loss Space Difference:", loss_space_diff)
print("Loss Time Difference: ", loss_time_diff)
print("Loss Miss:           ", loss_miss)
print("Loss False Alarm:    ", loss_false_alarm)
```

```
Gain Hit:             [0 0]
Gain Space Difference: [0 0]
Gain Time Difference:  [0 0]
Gain Miss:            [0 5]
Gain False Alarm:     [4 0]

Loss Hit:             [2 2]
Loss Space Difference: [1 1]
Loss Time Difference:  [0 0]
Loss Miss:            [1 1]
Loss False Alarm:     [1 1]
```

4.5: Compute Sum and Extent components for gains & losses

In [13]:

```python
# Section 4.5: Compute "Sum" and "Extent" summary components for gains & losses

import numpy as np

# --- Sum (aggregate of the two intervals) ---
# Net gain miss vs false alarm
net_gain_miss         = gain_miss.sum() - gain_false_alarm.sum()
sum_gain_miss         = max(net_gain_miss, 0)
sum_gain_false_alarm  = max(-net_gain_miss, 0)
sum_gain_hit          = gain_hit.sum()
sum_gain_space_diff   = gain_space_diff.sum()
sum_gain_time_diff    = 4  # from example

# Net loss miss vs false alarm
net_loss_miss         = loss_miss.sum() - loss_false_alarm.sum()
sum_loss_miss         = max(net_loss_miss, 0)
sum_loss_false_alarm  = max(-net_loss_miss, 0)
sum_loss_hit          = loss_hit.sum()
sum_loss_space_diff   = loss_space_diff.sum()
sum_loss_time_diff    = 2  # from example

# --- Extent (full-extent change between t0 and t2) ---
delta_x = p_x[-1] - p_x[0]
delta_y = p_y[-1] - p_y[0]

# Gains
extent_gx               = np.clip(delta_x, a_min=0, a_max=None)
extent_gy               = np.clip(delta_y, a_min=0, a_max=None)
net_extent_gain_miss    = extent_gx.sum() - extent_gy.sum()
extent_gain_miss        = max(net_extent_gain_miss, 0)
extent_gain_false_alarm = max(-net_extent_gain_miss, 0)
extent_gain_hit         = np.minimum(extent_gx, extent_gy).sum()
extent_gain_space_diff  = np.abs(extent_gx - extent_gy).sum()
extent_gain_time_diff   = 0  # from example

# Losses
extent_lx               = np.clip(-delta_x, a_min=0, a_max=None)
extent_ly               = np.clip(-delta_y, a_min=0, a_max=None)
net_extent_loss_miss    = extent_lx.sum() - extent_ly.sum()
extent_loss_miss        = max(net_extent_loss_miss, 0)
extent_loss_false_alarm = max(-net_extent_loss_miss, 0)
extent_loss_hit         = np.minimum(extent_lx, extent_ly).sum()
extent_loss_space_diff  = np.abs(extent_lx - extent_ly).sum()
extent_loss_time_diff   = 0  # from example

# Print for verification
print("Sum gain components: hit", sum_gain_hit,
      "space_diff", sum_gain_space_diff,
      "time_diff", sum_gain_time_diff,
      "miss", sum_gain_miss,
      "false_alarm", sum_gain_false_alarm)
```

```
print("Sum loss components: hit", sum_loss_hit,
      "space_diff", sum_loss_space_diff,
      "time_diff", sum_loss_time_diff,
      "miss", sum_loss_miss,
      "false_alarm", sum_loss_false_alarm)

print("Extent gain components: hit", extent_gain_hit,
      "space_diff", extent_gain_space_diff,
      "time_diff", extent_gain_time_diff,
      "miss", extent_gain_miss,
      "false_alarm", extent_gain_false_alarm)
print("Extent loss components: hit", extent_loss_hit,
      "space_diff", extent_loss_space_diff,
      "time_diff", extent_loss_time_diff,
      "miss", extent_loss_miss,
      "false_alarm", extent_loss_false_alarm)
```

```
Sum gain components: hit 0 space_diff 0 time_diff 4 miss 1 false_alarm 0
Sum loss components: hit 4 space_diff 2 time_diff 2 miss 0 false_alarm 0
Extent gain components: hit 0 space_diff 5 time_diff 0 miss 1 false_alarm 0
Extent loss components: hit 0 space_diff 8 time_diff 0 miss 0 false_alarm 0
```

5. Full-Extent Change Metrics (Eqs 29–40, 41–52)

In this section we summarize change across the entire time series by computing total gains and losses, deriving overall quantity metrics at both interval and full-extent scales, and preparing per-pixel summary scores.

5.1 Compute Total Gains & Losses over (t=0\to T)

We sum each pixel's gains and losses over all consecutive intervals.

- `G_x_full[n]` = total gain of pixel `n` across the whole series
- `G_y_full[n]` = total gain of pixel `n` in the comparison series
- `L_x_full[n]` = total loss of pixel `n` across the whole series
- `L_y_full[n]` = total loss of pixel `n` in the comparison series

These arrays show how much each pixel increased or decreased from start to finish.

In [14]:
```
# Total gain per pixel across all intervals
G_x_full = g_x.sum(axis=0)
G_y_full = g_y.sum(axis=0)

# Total loss per pixel across all intervals
L_x_full = l_x.sum(axis=0)
L_y_full = l_y.sum(axis=0)

# Print results for verification
print("Full-extent gain (reference series) per pixel:", G_x_full)
print("Full-extent gain (comparison series) per pixel:", G_y_full)
print("Full-extent loss (reference series) per pixel:", L_x_full)
print("Full-extent loss (comparison series) per pixel:", L_y_full)
```

```
Full-extent gain (reference series) per pixel: [5 0]
Full-extent gain (comparison series) per pixel: [0 4]
Full-extent loss (reference series) per pixel: [2 4]
Full-extent loss (comparison series) per pixel: [4 2]
```

5.2 Quantity Metrics (Interval & Full-Extent Sums)

Next, we aggregate gains and losses across pixels to get:

- **Interval sums** (one value per interval):
  - `Q_g_x[t]` = sum of gain in reference series across all pixels at interval `t`
  - `Q_l_x[t]` = sum of loss in reference series across all pixels at interval `t`

  (and similarly `Q_g_y[t]`, `Q_l_y[t]` for the comparison series)
- **Full-extent sums** (single values):
  - `Q_G_x` = total gain across all pixels and all intervals in the reference series
  - `Q_L_x` = total loss across all pixels and all intervals in the reference series

  (and `Q_G_y`, `Q_L_y` for the comparison series)

These metrics quantify the overall magnitude of change at both the interval level and for the full series.

In [15]:
```python
# Interval sums per interval (sum over pixels)
q_g_x = g_x.sum(axis=1)
q_l_x = l_x.sum(axis=1)
q_g_y = g_y.sum(axis=1)
q_l_y = l_y.sum(axis=1)

# Full-extent sums across all intervals and pixels
Q_G_x = g_x.sum()
Q_L_x = l_x.sum()
Q_G_y = g_y.sum()
Q_L_y = l_y.sum()

# Print results for verification
print("Interval gain sums (reference series):", q_g_x)
print("Interval loss sums (reference series):", q_l_x)
print("Interval gain sums (comparison series):", q_g_y)
print("Interval loss sums (comparison series):", q_l_y)

print("\nFull-extent gain (reference series):", Q_G_x)
print("Full-extent loss (reference series):", Q_L_x)
print("Full-extent gain (comparison series):", Q_G_y)
print("Full-extent loss (comparison series):", Q_L_y)
```

```
Interval gain sums (reference series): [0 5]
Interval loss sums (reference series): [3 3]
Interval gain sums (comparison series): [4 0]
Interval loss sums (comparison series): [3 3]

Full-extent gain (reference series): 5
Full-extent loss (reference series): 6
Full-extent gain (comparison series): 4
Full-extent loss (comparison series): 6
```

5.3 Pixel-Wise Summary Metrics

Finally, we compute compact scores for each pixel that combine presence-agreement and change-agreement:

- **Presence agreement**: `H_n` (total shared presence from Section 3)
- **Change agreement**: `H_g_n + H_l_n` (sum of gain hits and loss hits for pixel `n`)
- **Spatial difference**: `U_n + U_g_n + U_l_n` (sum of spatial discrepancies from presence, gains, and losses)
- **Temporal difference**: `V_n + V_g_n + V_l_n` (sum of timing discrepancies from presence, gains, and losses)

These per-pixel scores provide an at-a-glance profile of agreement versus disagreement for each location.

In [16]:
```
# 1. Total presence agreement per pixel (from Section 3.4)
H_n = h.sum(axis=0)

# 2. Total presence difference per pixel
Dp_n = np.abs(p_x - p_y).sum(axis=0)

# 3. Total change agreement per pixel (gain hits + loss hits)
C_n = h_g.sum(axis=0) + h_l.sum(axis=0)

# 4. Total change difference per pixel (sum of gain and loss magnitude differences)
Dc_n = np.abs(g_x - g_y).sum(axis=0) + np.abs(l_x - l_y).sum(axis=0)

# Combine into a pandas DataFrame
pixel_scores = pd.DataFrame({
    "presence_agreement": H_n,
    "presence_difference": Dp_n,
    "change_agreement": C_n,
    "change_difference": Dc_n
}, index=[f"pixel_{i}" for i in range(H_n.size)])

# Display the DataFrame in the notebook
display(pixel_scores)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[16], line 2
      1 # 1. Total presence agreement per pixel (from Section 3.4)
----> 2 H_n = h.sum(axis=0)
      4 # 2. Total presence difference per pixel
      5 Dp_n = np.abs(p_x - p_y).sum(axis=0)

NameError: name 'h' is not defined
```

6. Visualization of Results

In this section we create clear, publication-quality charts to illustrate the computed metrics. Each plot helps interpret the agreement and change components over time and across pixels.

6.1 Stacked Bar Chart: Presence Agreement

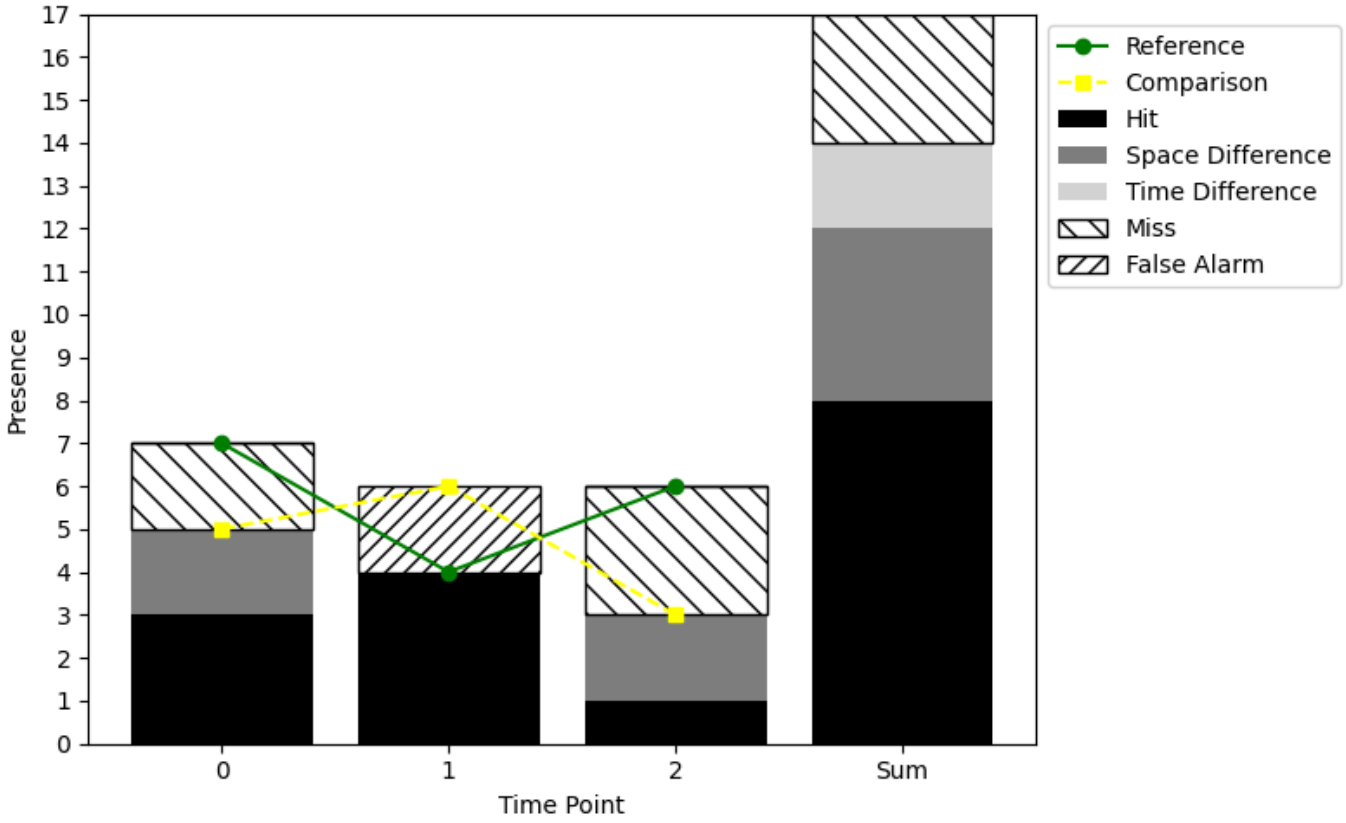
We plot a stacked bar chart showing, for each time point, the contributions of:

- shared presence (hits)
- reference-only presence (misses)
- comparison-only presence (false alarms)
- spatial differences
- temporal differences

This reveals how presence agreement and disagreement evolve over time.

In [17]:
```
import numpy as np
import matplotlib.pyplot as plt

# Section 6.1: Presence Agreement per Time Point including Sum

# assume p_x, p_y, hits_tp, space_diff, misses_tp, false_tp, time_diff_tp are in memory

# compute Sum-bar components
hits_sum   = hits_tp.sum()                              # 8
space_sum  = space_diff.sum()                           # 4
```

```
# net disagreement for Sum: misses sum minus false alarms sum
net_misses = misses_tp.sum() - false_tp.sum()          # 3
miss_sum   = net_misses if net_misses > 0 else 0       # 3
false_sum  = -net_misses if net_misses < 0 else 0      # 0
time_sum   = 2                                         # full-extent timing mismatch

# assemble full arrays
hits_all   = np.append(hits_tp, hits_sum)
space_all  = np.append(space_diff, space_sum)
time_all   = np.append(time_diff_tp, time_sum)
miss_all   = np.append(misses_tp, miss_sum)
false_all  = np.append(false_tp, false_sum)

# reference/comparison full series sums
ref_tp   = p_x.sum(axis=1)
comp_tp  = p_y.sum(axis=1)
ref_all  = np.append(ref_tp,ref_tp.sum())
comp_all = np.append(comp_tp,comp_tp.sum())

# plotting
categories = ['0', '1', '2', 'Sum']
x = np.arange(len(categories))

plt.figure(figsize=(8,5))
plt.bar(x, hits_all,      color='black',                        label='Hit')
plt.bar(x, space_all,     bottom=hits_all,                       color='gray',     label='Spa
plt.bar(x, time_all,      bottom=hits_all+space_all,            color='lightgray',label='Time
plt.bar(x, miss_all,      bottom=hits_all+space_all+time_all,
        facecolor='white', edgecolor='black', hatch='\\\\',    label='Miss')
plt.bar(x, false_all,     bottom=hits_all+space_all+time_all+miss_all,
        facecolor='white', edgecolor='black', hatch='///',     label='False Alarm')

plt.plot(x[:num_time_points],
         ref_tp,color='green',
         linestyle='-',
         marker='o',
         label='Reference'
         )

plt.plot(x[:num_time_points],
         comp_tp,color='yellow',
         linestyle='--',
         marker='s',
         label='Comparison'
         )

plt.xlabel('Time Point')
plt.ylabel('Presence')
plt.xticks(x, categories)
plt.ylim(0, 17)
plt.yticks(np.arange(0,18,1))
# plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.legend(loc='upper left', bbox_to_anchor=(1,1))
plt.tight_layout()
plt.show()
```



6.2 Stacked Bar Chart: Gain & Loss Components

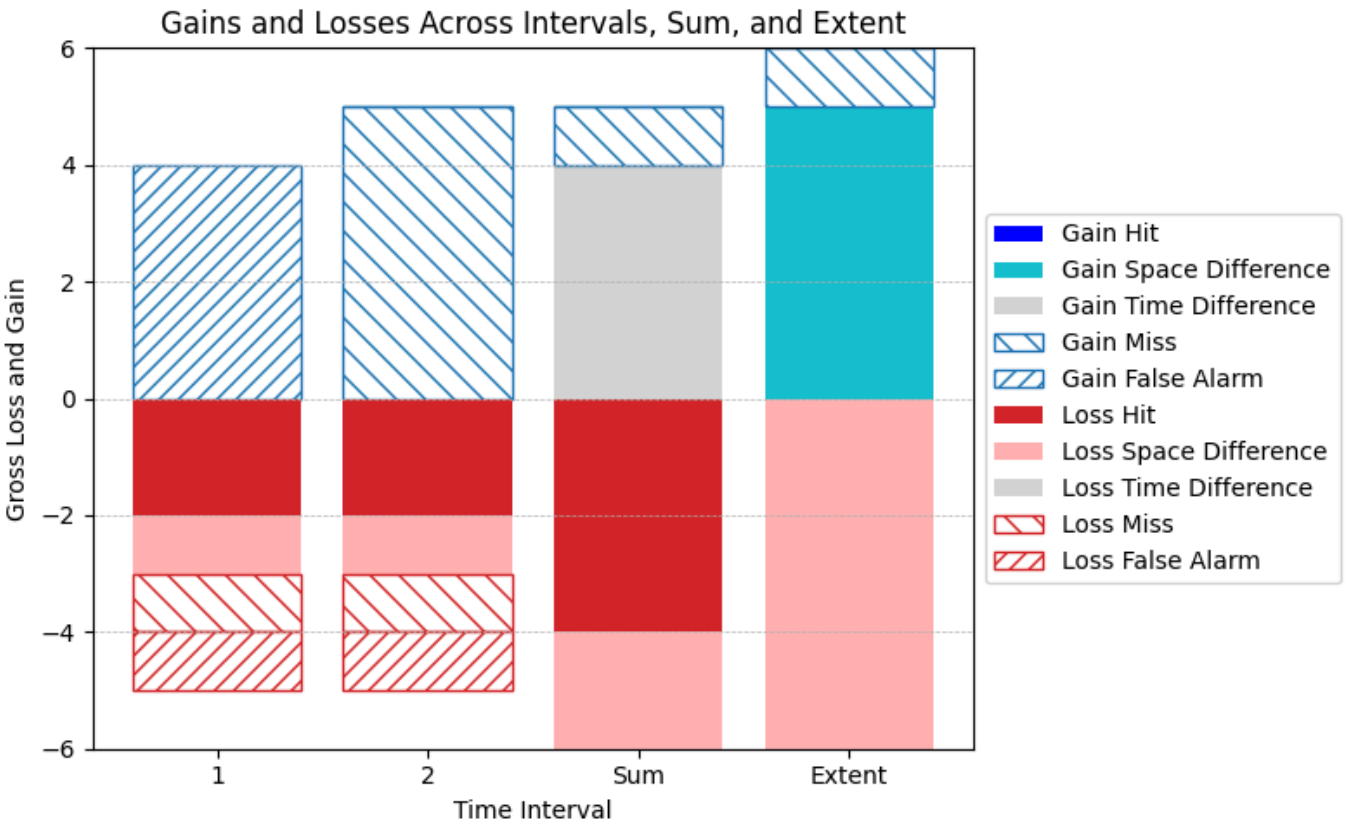
Next, we generate two side-by-side stacked bar charts—one for gains and one for losses—showing, at each interval:

- shared change (gain hits or loss hits)
- reference-only change (gain misses or loss misses)
- comparison-only change (gain false alarms or loss false alarms)
- spatial change differences
- timing differences

This highlights when and where increases or decreases align or diverge.

In [26]:
```
# Section 6.2: Combined Stacked Bar Chart of Gain & Loss Components with Sum & Extent

import numpy as np
import matplotlib.pyplot as plt

# Assumes the following 20 variables are defined in the notebook namespace:
# Interval arrays from Section 4.4:
#   gain_hit, gain_space_diff, gain_time_diff, gain_miss, gain_false_alarm
#   loss_hit, loss_space_diff, loss_time_diff, loss_miss, loss_false_alarm
#
# Sum-bar variables from Section 4.5:
#   sum_gain_hit, sum_gain_space_diff, sum_gain_time_diff, sum_gain_miss, sum_gain_false_alarr
#   sum_loss_hit, sum_loss_space_diff, sum_loss_time_diff, sum_loss_miss, sum_loss_false_alarr
#
# Extent-bar variables from Section 4.5:
#   extent_gain_hit, extent_gain_space_diff, extent_gain_time_diff, extent_gain_miss, extent_g
#   extent_loss_hit, extent_loss_space_diff, extent_loss_time_diff, extent_loss_miss, extent_

# 1) Assemble full 4-element arrays: [Interval1, Interval2, Sum, Extent]
gain_hit_all        = np.concatenate([gain_hit,        [sum_gain_hit,        extent_gain_hit
gain_space_all      = np.concatenate([gain_space_diff, [sum_gain_space_diff, extent_gain_spac
gain_time_all       = np.concatenate([gain_time_diff,  [sum_gain_time_diff,  extent_gain_time_
gain_miss_all       = np.concatenate([gain_miss,       [sum_gain_miss,       extent_gain_miss
```

```
gain_false_all      = np.concatenate([gain_false_alarm,[sum_gain_false_alarm,extent_gain_fals

loss_hit_all        = np.concatenate([loss_hit,        [sum_loss_hit,        extent_loss_hit
loss_space_all      = np.concatenate([loss_space_diff, [sum_loss_space_diff, extent_loss_spac
loss_time_all       = np.concatenate([loss_time_diff,  [sum_loss_time_diff,  extent_loss_time
loss_miss_all       = np.concatenate([loss_miss,       [sum_loss_miss,       extent_loss_miss
loss_false_all      = np.concatenate([loss_false_alarm,[sum_loss_false_alarm,extent_loss_fals

# 2) Define x positions and labels
categories = ['1', '2', 'Sum', 'Extent']
x = np.arange(len(categories))

# 3) Plot
fig, ax = plt.subplots(figsize=(8, 5))

# Gains above x-axis
bottom = np.zeros_like(gain_hit_all)
ax.bar(x, gain_hit_all,   bottom=bottom, label='Gain Hit',              color='#0000ff')
bottom += gain_hit_all
ax.bar(x, gain_space_all, bottom=bottom, label='Gain Space Difference',  color='tab:cyan')
bottom += gain_space_all
ax.bar(x, gain_time_all,  bottom=bottom, label='Gain Time Difference',   color='lightgray')
bottom += gain_time_all
ax.bar(x, gain_miss_all,  bottom=bottom, label='Gain Miss',              facecolor='white', e
bottom += gain_miss_all
ax.bar(x, gain_false_all, bottom=bottom, label='Gain False Alarm',       facecolor='white', e

# Losses below x-axis
bottom = np.zeros_like(loss_hit_all)
ax.bar(x, -loss_hit_all,   bottom=bottom, label='Loss Hit',              color='tab:red')
bottom -= loss_hit_all
ax.bar(x, -loss_space_all, bottom=bottom, label='Loss Space Difference',  color='#ffb3b3')
bottom -= loss_space_all
ax.bar(x, -loss_time_all,  bottom=bottom, label='Loss Time Difference',   color='lightgray')
bottom -= loss_time_all
ax.bar(x, -loss_miss_all,  bottom=bottom, label='Loss Miss',              facecolor='white',
bottom -= loss_miss_all
ax.bar(x, -loss_false_all, bottom=bottom, label='Loss False Alarm',       facecolor='white',

# 4) Formatting
ax.set_xticks(x)
ax.set_xticklabels(categories)
ax.set_xlabel('Time Interval')
ax.set_ylabel('Gross Loss and Gain')
ax.set_title('Gains and Losses Across Intervals, Sum, and Extent')
ax.grid(axis='y', linestyle='--', linewidth=0.5)
ax.set_ylim(-6, 6)

# 5) Legend outside to the right
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.tight_layout()
plt.show()
```



In [ ]:

```python
# Section 6.2: Losses and Gains During Time Intervals (1, 2, Sum, Extent)

import numpy as np
import matplotlib.pyplot as plt

# ==== hard-coded results from the toy-data example ====
# Gains (positive)
gain_hit           = np.array([0, 0, 0, 0])
gain_space_diff    = np.array([0, 0, 0, 2])
gain_time_diff     = np.array([0, 0, 4, 0])
gain_miss          = np.array([0, 5, 1, 1])
gain_false_alarm   = np.array([4, 0, 0, 0])

# Losses (negative)
loss_hit           = -np.array([2, 2, 4, 0])
loss_space_diff    = -np.array([1, 1, 2, 4])
loss_time_diff     = -np.array([0, 0, 0, 0])
loss_miss          = -np.array([0, 0, 0, 0])
loss_false_alarm   = -np.array([0, 0, 0, 0])

# Build x axis
categories = ['1', '2', 'Sum', 'Extent']
x = np.arange(len(categories))

fig, ax = plt.subplots(figsize=(8, 5))

# === plot gains ===
bottom = np.zeros_like(gain_hit, dtype=int)
ax.bar(x, gain_hit,         bottom=bottom,                label='Gain Hit',              col
bottom += gain_hit
ax.bar(x, gain_space_diff,  bottom=bottom,                label='Gain Space Difference',  co
bottom += gain_space_diff
ax.bar(x, gain_time_diff,   bottom=bottom,                label='Gain Time Difference',   co
bottom += gain_time_diff
ax.bar(x, gain_miss,        bottom=bottom,                label='Gain Miss',              fa
bottom += gain_miss
ax.bar(x, gain_false_alarm, bottom=bottom,                label='Gain False Alarm',       fa

# === plot losses ===
```

```
bottom = np.zeros_like(loss_hit, dtype=int)
ax.bar(x, loss_hit,         bottom=bottom,                label='Loss Hit',              col
bottom += loss_hit
ax.bar(x, loss_space_diff,  bottom=bottom,                label='Loss Space Difference',  co
bottom += loss_space_diff
ax.bar(x, loss_time_diff,   bottom=bottom,                label='Loss Time Difference',   co
bottom += loss_time_diff
ax.bar(x, loss_miss,        bottom=bottom,                label='Loss Miss',             fa
bottom += loss_miss
ax.bar(x, loss_false_alarm, bottom=bottom,                label='Loss False Alarm',      fa

# === formatting ===
ax.set_xticks(x)
ax.set_xticklabels(categories)
ax.set_xlabel('Time Interval')
ax.set_ylabel('Gross Loss and Gain')
ax.set_title('Losses and Gains During Two Time Intervals')
# ax.grid(axis='y', linestyle='--', linewidth=0.5)

# y-limits from -6 to +6 to match the paper
ax.set_ylim(-6, 6)

# legend outside on the right
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.tight_layout()
plt.show()
```

6.3 Composition of Full-Extent Change

Finally, we present a combined bar chart of total gains and total losses over the entire series (the “full extent”), with each component stacked. This summary plot shows overall change magnitude and the balance between agreement and disagreement across all intervals.

In [ ]:

```

```

7. Exporting Results

7.1 Save Metrics DataFrame to CSV/Excel

7.2 Save Figures (PNG)