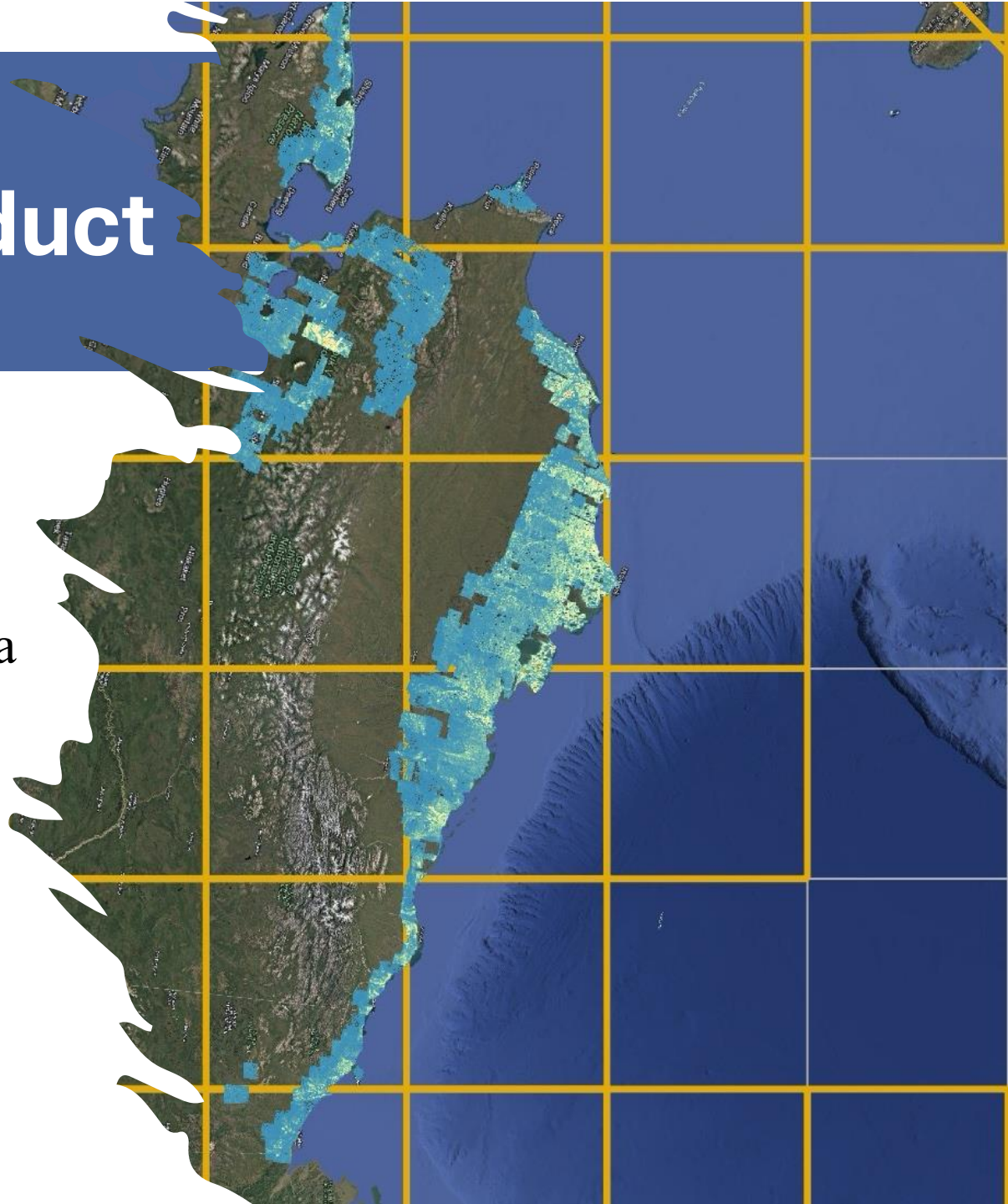
# IWP big statistics data product



A step-by-step tutorial to generate IWP big stats data

Cyberinfrastructure and Computational Intelligence (CICI) Lab
Arizona State University

# Introduction

**Ice Wedge (IW)** is a crack in the ground formed by a narrow or thin piece of ice that measures up to 3–4 meters in length at ground level. As IW gets deeper, **Ice Wedge Polygons (IWP)** are formed.

The large number of IWPs across the entire Pan-Arctic region were extracted and organized by PDG (either in geopackage or shapefile format in an arctic projection EPSG:3413).

# Introduction

The goal of this project is to analysis IWPs, and generate data products to describe different statistical profiles of IWP. This include:

- IWP count maps.
- Area sum
- Length(diameter) sum/min/max/median/mean/std
- Perimeter sum
- Width sum
- LCP count

(We also generate some other side products, such as heatmap maps.)

This tutorial will walk you through how to download these Ice Wedge Polygon (IWP), how to process the data in batch, and finally generate big statistics data products

# Workflow of the big statistics data mapping

**Step 1**: Create 230 grids to cover the entire Pan-Arctic region

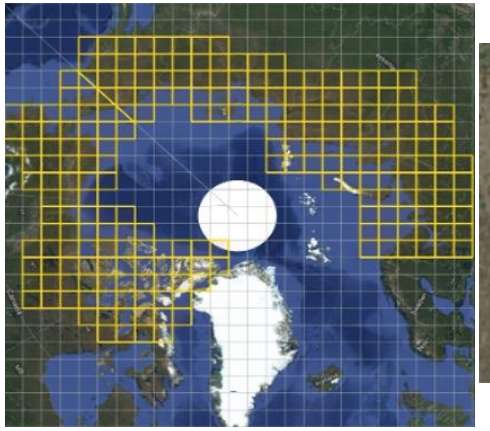
**Step 2**: Set the pixel of each grid to 1km

**Step 3**: (Download geopackages first if needed) Get BBOX of each pixel, and search for IWP under Tiling system
(refer to Juliet's code: https://github.com/PermafrostDiscoveryGateway/viz-info/blob/main/helpful-code/prepocessing/bounding_box_tiles.ipynb)

**Step 4**: Calculate all IWPs within a pixel, and process pixels in batch

**Step 5**: Validation

**Step 6**: Map all pixels within a grid, and slice all grids together



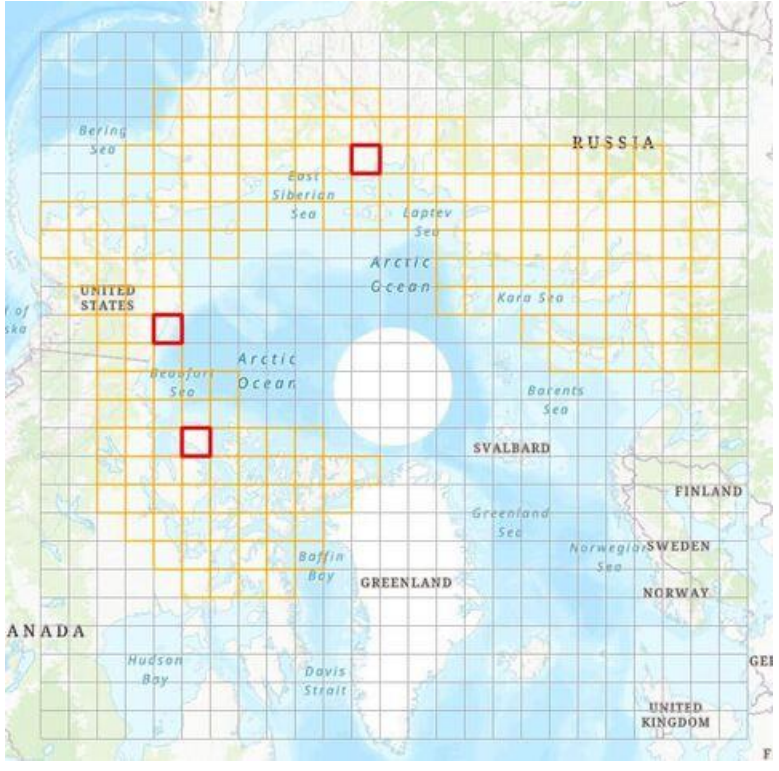
(230 grids)

(IWPs)

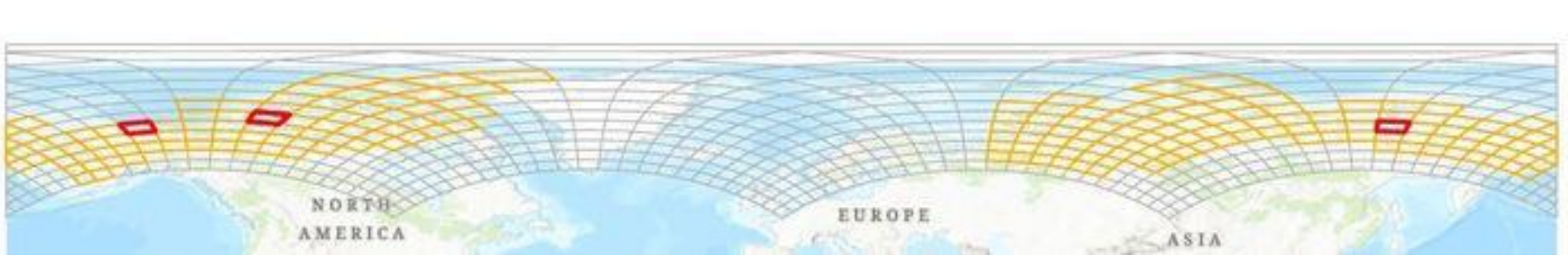
# Step 1: Create Grids (256km * 256km)

Create 230 grids in ArcGIS

- Projection: 3413
- The extent of each grid is 256km*256km




EPSG: 3413



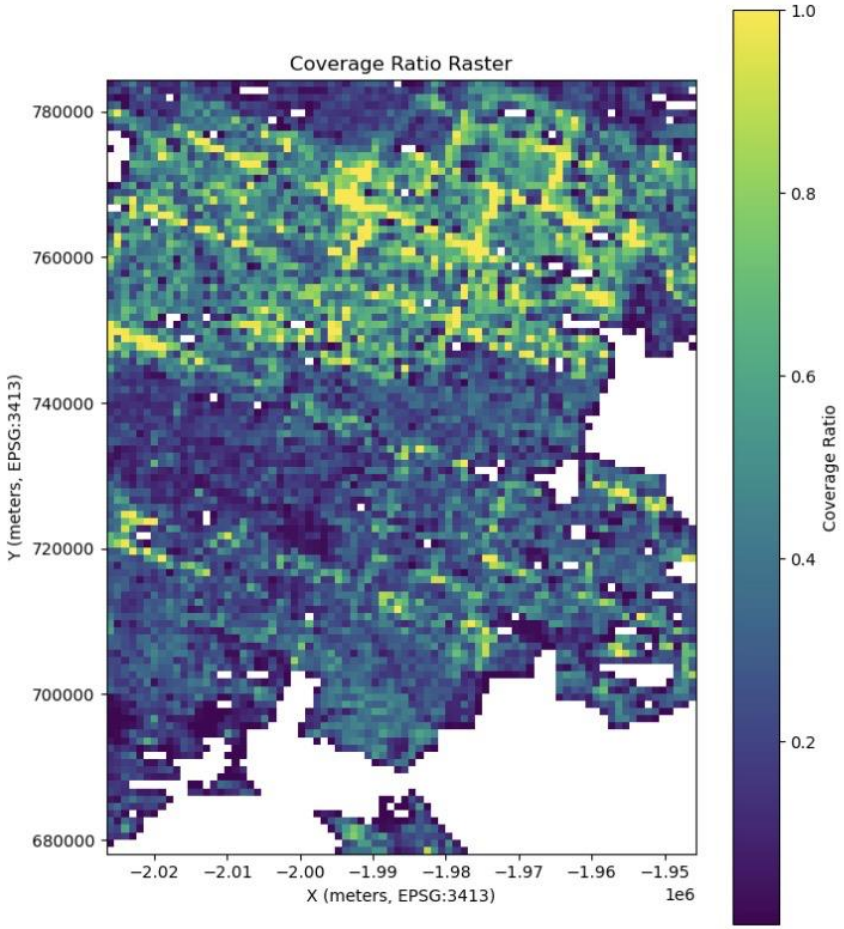

EPSG: 4326

# Step 2: Create pixels (1km * 1km)

As shown in Github Repo

Code location:
src> gpkg > statistics_mapping_noDatabase.py

```
def gen_pixel_bounds(cell_bounds):
    minx, miny, maxx, maxy = cell_bounds
    x = range(round(minx), round(maxx), SIZE_PIXEL)
    y = range(round(miny), round(maxy), SIZE_PIXEL)
    assert len(x) == N_PIXELS and len(y) == N_PIXELS, "Invalid cell bounds."
```



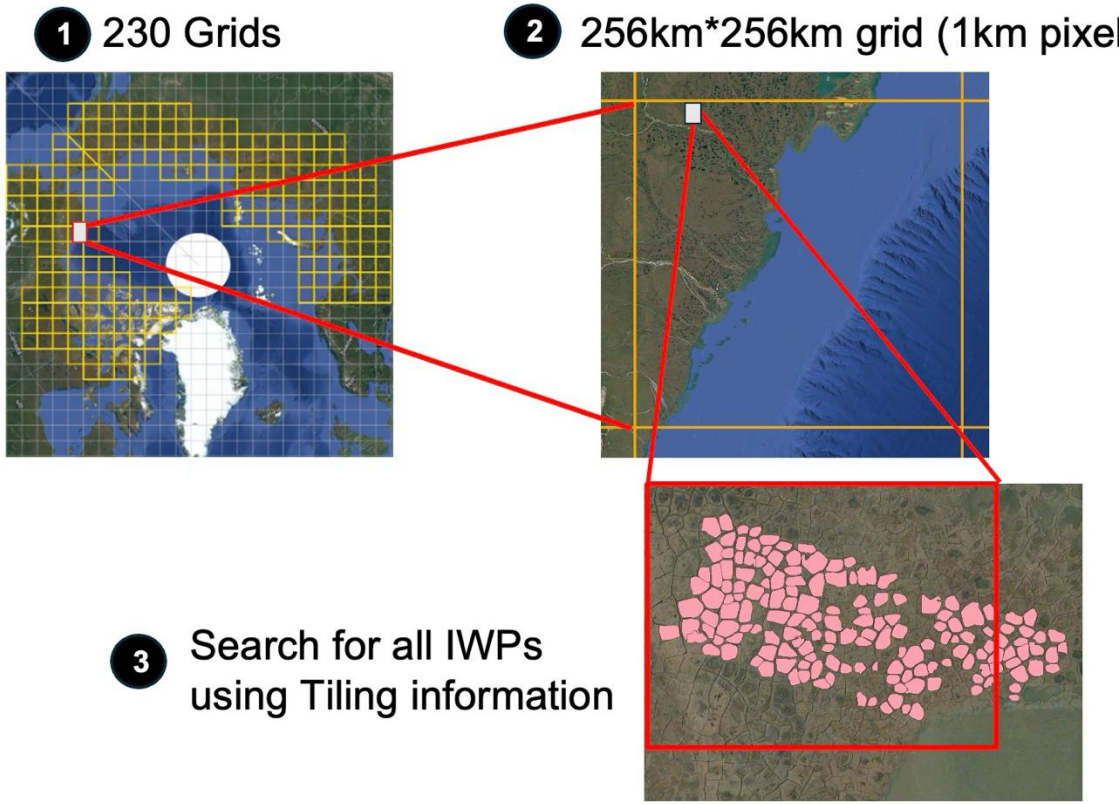
# Step 3: (Download geopackages first if needed) Get BBOX of each pixel, and search for IWPs under Tiling system

a) Introduction to Tiling system

b) Download geopackges (GPKGs)

c) IWP deduplication

d) Get intersected geopackages within pixels



# a) Tiling system

Our IWP data is stored in geopackages or shapefiles, and these vector files are organized under Tiling system.

We can search for IWPs from desired region based on their Tiling information (Z-X-Y).

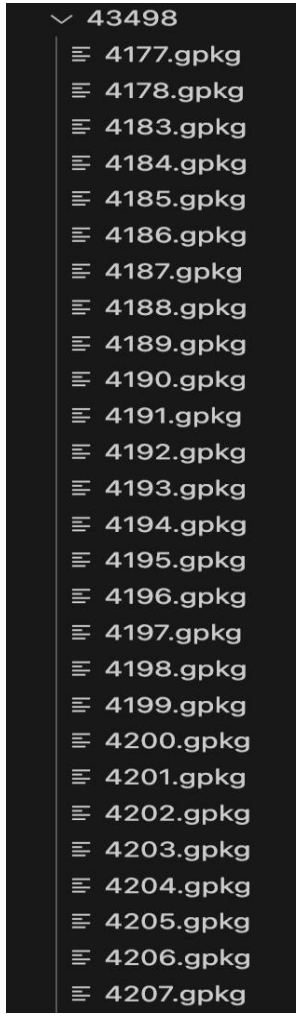
For example, all IWPs are stored on zoom level 15 (Z).

In the picture, 43498 is IWPs' X tile index,

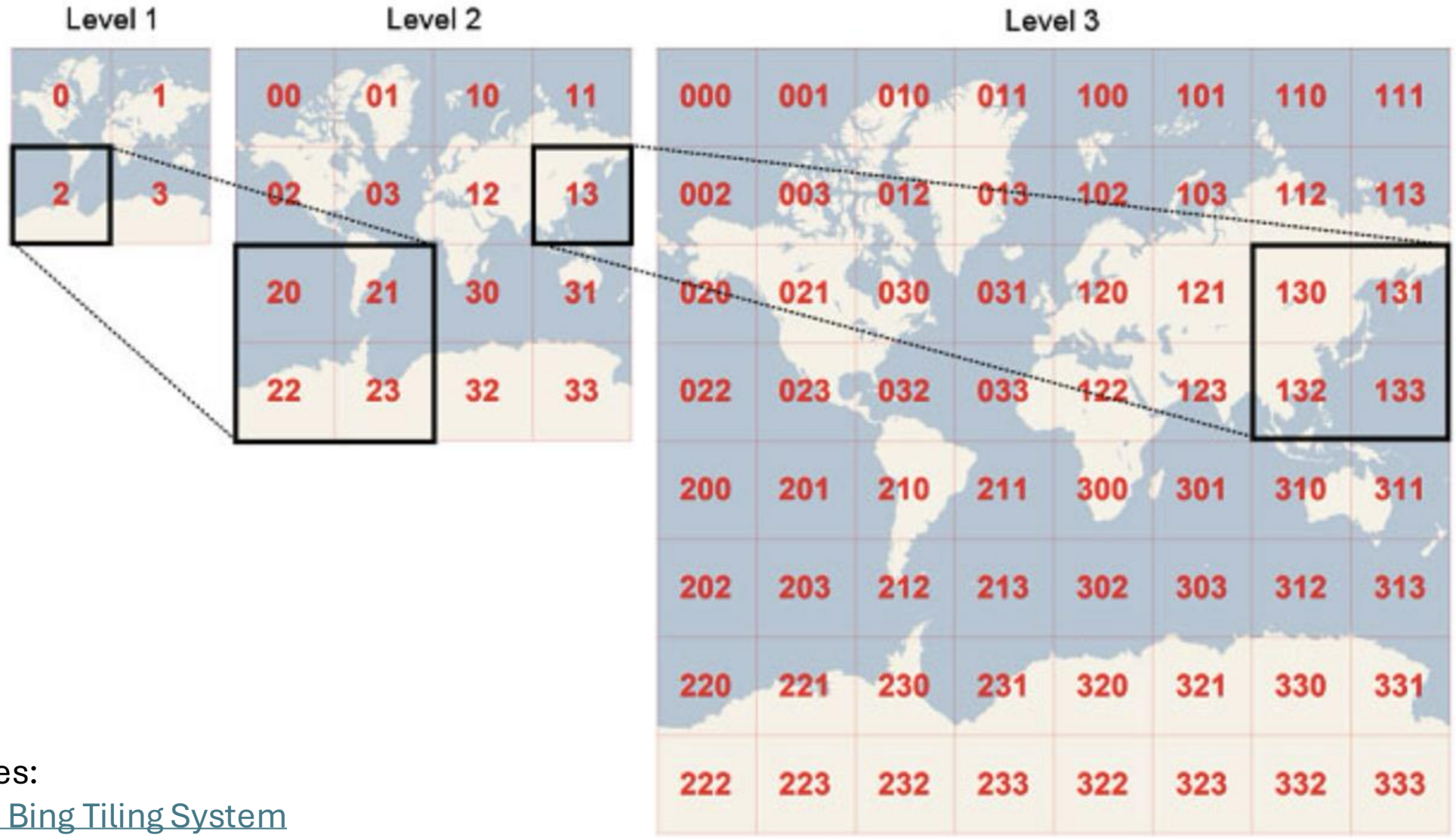
4177/4178/... is IWPs' Y tile index.

One Z-X-Y tile index corresponds to one geopakcage (GPKGs) file.

One geopackage file contains multiple IWPs.



# Tile Matrix Set




References:

[Microsoft Bing Tiling System]

[Open Geospatial Consortium]

# b) Download geopackages

```
(base) xchen@cici2labasuedu:~/data/geopackages/iwp_geopackage_high/WGS1984Quad/15$ ls
100    11599  13200  14786  16434  18029  19727  2408  3994  43410  44996  4658  4950  6536  8375
1000   116    13201  14787  16435  1803   19728  2409  3995  43411  44997  46580 4951  6537  8376
10000  11600  13202  14788  16436  18030  19729  241   3996  43412  44998  46581 4952  6538  8377
10001  11601  13203  14789  16437  18031  19730  2410  3997  43413  44999  46582 4953  6539  8378
10002  11602  13204  1479   16438  18032  19731  2411  3998  43414  45     46583 4954  654   8379
10003  11603  13205  14790  16439  18033  19732  2412  3999  43415  450    46584 4955  6540  8380
10004  11604  13206  14791  1644   18034  19733  2413  40    43416  4500   46585 4956  6541  8381
10005  11605  13207  14792  16440  18035  19734  2414  400   43417  45000  46586 4957  6542  8382
10006  11606  13208  14793  16441  18036  19735  2415  4000  43418  45001  46587 4958  6543  8383
10007  11607  13209  14794  16442  18037  19736  2416  4001  43419  45002  46588 4959  6544  8384
10008  11608  1321   14795  16443  18038  19737  2417  4002  4342   45003  46589 496   6545  8385
10009  11609  13210  14796  16444  18039  19738  2418  4003  43420  45004  4659  4960  6546  8386
1001   11610  13211  14797  16445  1804   19739  2419  4004  43421  45005  46590 4961  6547  8387
10010  11611  13212  14798  16446  18040  19740  242   4005  43422  45006  46591 4962  6548  8388
10011  11612  13213  14799  16447  18041  19741  2420  4006  43423  45007  46592 4963  6549  8389
10012  11613  13214  148    16448  18042  19742  2421  4007  43424  45008  46593 4964  655   8390
10013  11614  13215  1480   16449  18043  19743  2422  4008  43425  45009  46594 4965  6550  8391
10014  11615  13216  14800  1645   18044  19744  2423  4009  43426  4501   46595 4966  6551  8392
10015  11616  13217  14801  16450  18045  19745  2424  401   43427  45010  46596 4967  6552  8393
10016  11617  13218  14802  16451  18046  19746  2425  4010  43428  45011  46597 4968  6553  8394
10017  11618  13219  14803  16452  18047  19747  2426  4011  43429  45012  46598 4969  6554  8395
10018  11619  1322   14804  16453  18048  19748  2427  4012  4343   45013  46599 497   6555  8396
10019  11620  13220  14805  16454  18049  19749  2428  4013  43430  45014  466   4970  6556  8397
1002   11621  13221  14806  16455  1805   19750  2429  4014  43431  45015  4660  4971  6557  8398
10020  11622  13222  14807  16456  18050  19751  243   4015  43432  45016  46600 4972  6558  8399
10021  11623  13223  14808  16457  18051  19752  2430  4016  43433  45017  46601 4973  6559  84
10022  11624  13224  14809  16458  18052  19753  2431  4017  43434  45018  46602 4974  656   8400
10023  11625  13225  1481   16459  18053  19754  2432  4018  43435  45019  46603 4975  6560  8401
10024  11626  13226  14810  1646   18054  19755  2433  4019  43436  4502   46604 4976  6561  8402
10025  11627  13227  14811  16460  18055  19756  2434  402   43437  45020  46605 4977  6562  8403
10026  11628  13228  14812  16461  18056  19757  2435  4020  43438  45021  46606 4978  6563  8404
10027  11629  13229  14813  16462  18057  19758  2436  4021  43439  45022  46607 4979  6564  8405
10028  11630  1323   14814  16463  18058  19759  2437  4022  4344   45023  46608 498   6565  8406
10029  11631  13230  14815  16464  18059  19760  2438  4023  43440  45024  46609 4980  6566  8407
1003   11632  13231  14816  16465  1806   19761  2439  4024  43441  45025  4661  4981  6567  8408
10030  11633  13232  14817  16466  18060  19762  244   4025  43442  45026  46610 4982  6568  8409
```

If GPKGs are stored in our running environment, we don't need to install them.
If GPKGs are stored in other servers, like NCEAS datateam server, then we need to download it to our running environment first.

You could check which tiles intersect with Alaska, e.g. in python you could use morecantile and and an overlay operation in geopandas.

Download code can be found in Github Repo:
src> gpkg > statistics_mapping_noDatabase.py

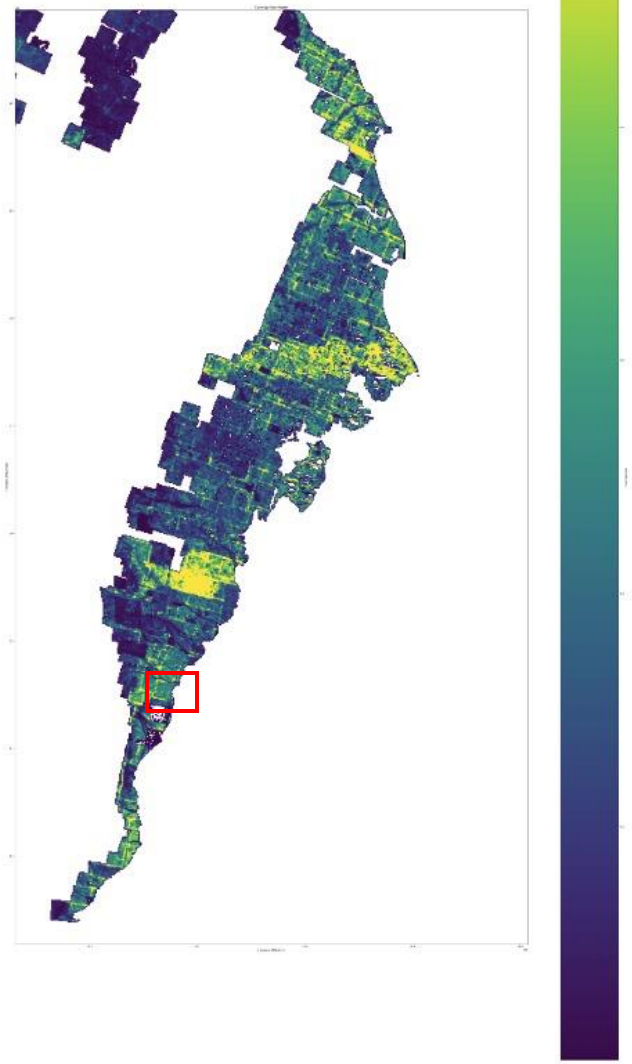
download_tile(tile, download_root='downloads')

```
(alaska_shpMapping2) (base) xchen@cici2labasuedu:~/data/geopackages/iwp_geopackage_high/WGS1984Quad/15/100$ ls
3358.gpkg  3368.gpkg  3378.gpkg  3388.gpkg  3398.gpkg  3408.gpkg  3418.gpkg  3428.gpkg  3452.gpkg
3359.gpkg  3369.gpkg  3379.gpkg  3389.gpkg  3399.gpkg  3409.gpkg  3419.gpkg  3429.gpkg  3455.gpkg
3360.gpkg  3370.gpkg  3380.gpkg  3390.gpkg  3400.gpkg  3410.gpkg  3420.gpkg  3430.gpkg  3465.gpkg
3361.gpkg  3371.gpkg  3381.gpkg  3391.gpkg  3401.gpkg  3411.gpkg  3421.gpkg  3431.gpkg  3470.gpkg
3362.gpkg  3372.gpkg  3382.gpkg  3392.gpkg  3402.gpkg  3412.gpkg  3422.gpkg  3432.gpkg  3471.gpkg
3363.gpkg  3373.gpkg  3383.gpkg  3393.gpkg  3403.gpkg  3413.gpkg  3423.gpkg  3433.gpkg  index.html
3364.gpkg  3374.gpkg  3384.gpkg  3394.gpkg  3404.gpkg  3414.gpkg  3424.gpkg  3434.gpkg  'index.html?C=D;O=A'
3365.gpkg  3375.gpkg  3385.gpkg  3395.gpkg  3405.gpkg  3415.gpkg  3425.gpkg  3435.gpkg  'index.html?C=M;O=A'
3366.gpkg  3376.gpkg  3386.gpkg  3396.gpkg  3406.gpkg  3416.gpkg  3426.gpkg  3436.gpkg  'index.html?C=N;O=D'
3367.gpkg  3377.gpkg  3387.gpkg  3397.gpkg  3407.gpkg  3417.gpkg  3427.gpkg  3440.gpkg  'index.html?C=S;O=A'
```

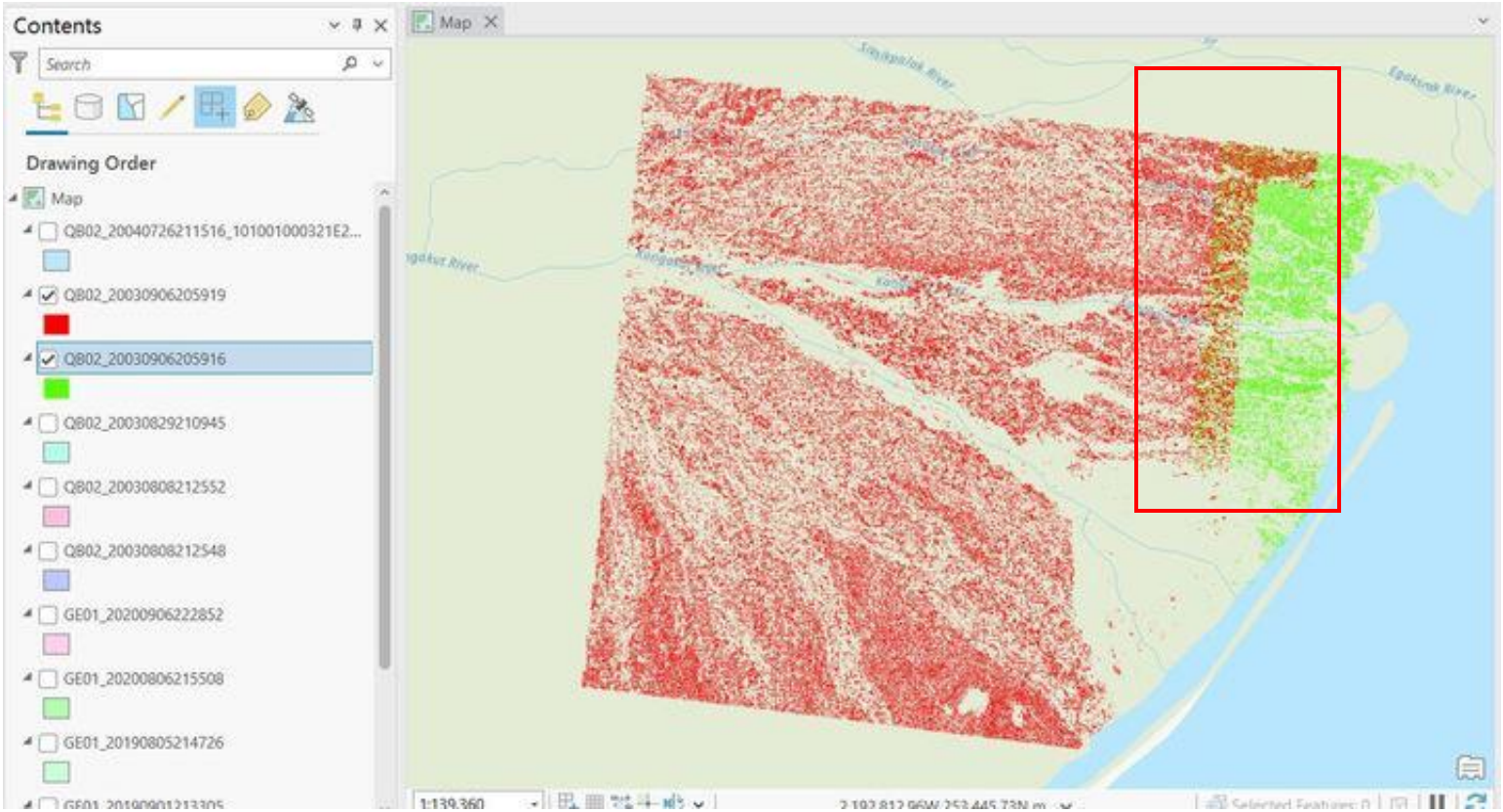
~ 90 geopackages under this folder

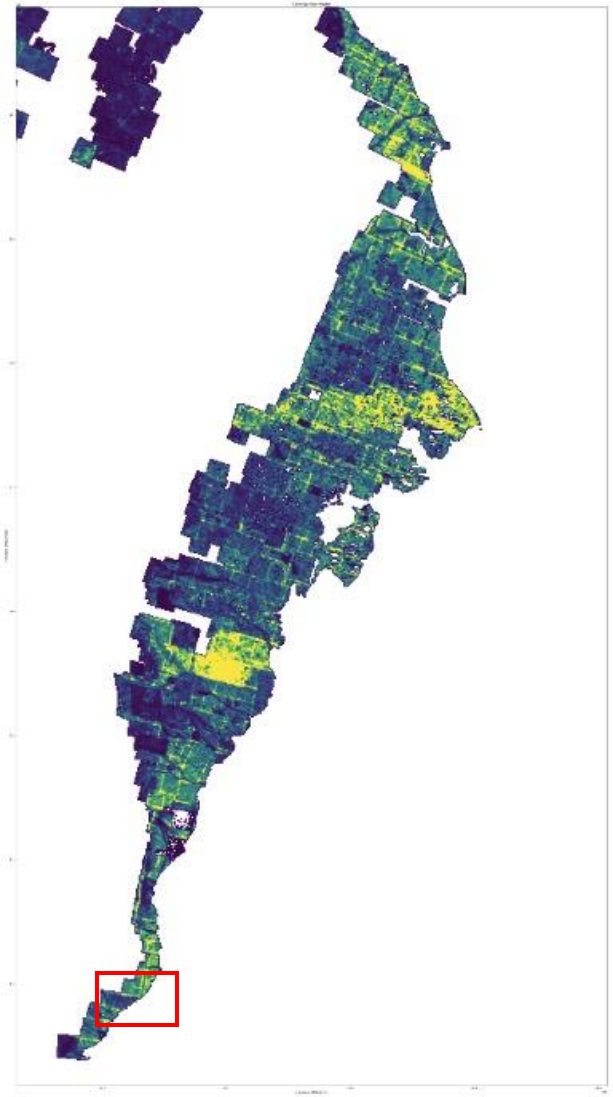
# c) IWP deduplication

Due to the remote sensing image footprint, images are partially overlapped, causing segmentation of IWPs from neighboring RS images are duplicated near the boundary.

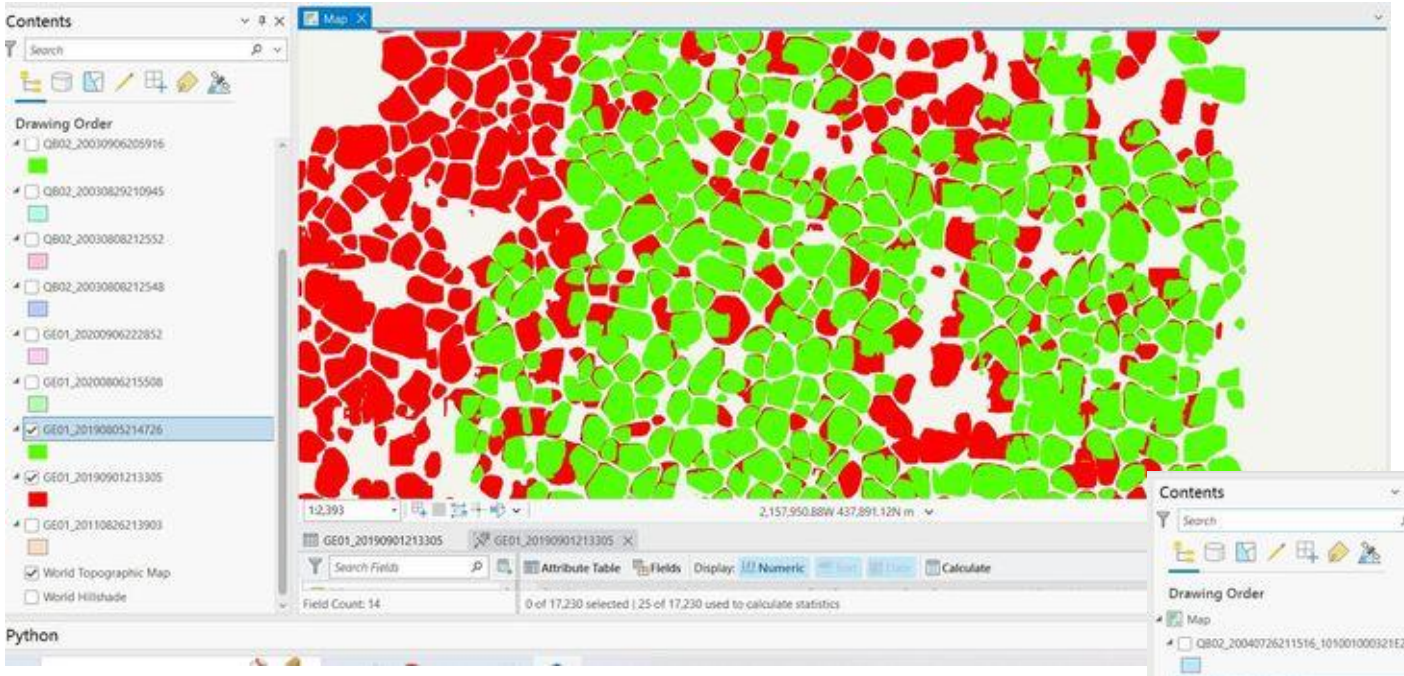



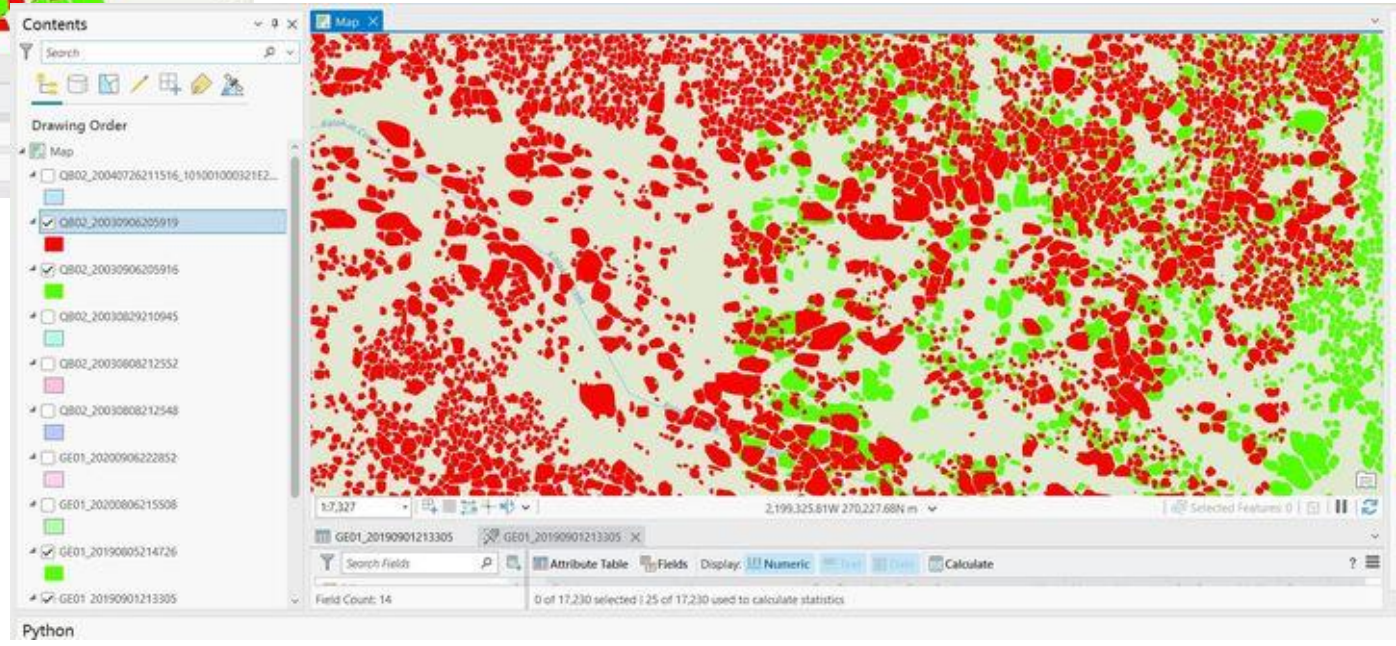
# Overlapped area



# Overlapped area details



# Deduplication

IWP dataset has a staging info to record if IWP has been deduplicated
We will select those IWP which have been selected

Deduplication code can be found here
src> gpkg > statistics_mapping_noDatabase.py

```
_gdf = gpd.read_file(tile_path)
dedup_gdf = _gdf[_gdf['staging_duplicated'] == False]
```

# d) Get intersected geopackages within pixels

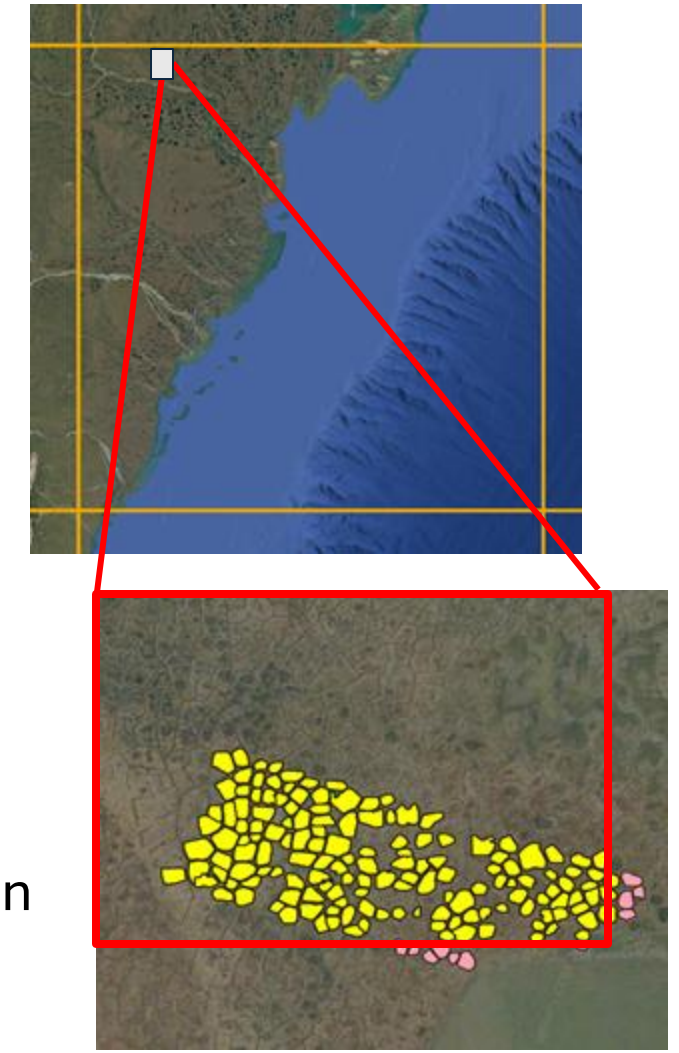
**Prerequisite:**

- One pixel may contain multiple geopakcages as we mentioned before
- Not all geopackages totally fall within one pixel

IWPs fall within the current pixel is what we desire and calculate.

This can be found in the same file in Github Repo:
get_intersected_tiles(bounds, tms, zoom=15)

256km*256km grid (1km pixel)



Search for all IWPs using Tiling information

# Step 4: Calculate all IWPs within a pixel , and process pixels in batch

- Each grid has 256 *256 pixels
- Each pixel has multiple geopacakges overlayed
- Each geopackages contains hundreds of IWPs
- We only calculate IWP calling within the current processing pixel

# Calculation

```
# fileter the data based on the column 'centroidX' and 'centroidY' to get the data within
inbox_gdf = gdf[
    gdf['CentroidX'].between(bounds[0], bounds[2]) &
    gdf['CentroidY'].between(bounds[1], bounds[3])
]

# get the skeleton of the IWP
inbox_gdf = inbox_gdf.to_crs(crs)
```

```
# print(inbox_gdf.columns)
stats = [
    len(inbox_gdf),
    inbox_gdf['Area'].sum(),
    inbox_gdf['Length'].sum(),
    inbox_gdf['Length'].min(),
    inbox_gdf['Length'].max(),
    inbox_gdf['Length'].median(),
    inbox_gdf['Perimeter'].sum(),
    inbox_gdf['Width'].sum(),
    (inbox_gdf['Class'].astype(int) == 1).sum(),
    IWP_skeleton.sum()
]
```

# Batch processing

```
# Process the pixel in parallel
mapper = Parallel(n_jobs=n_workers)
process = delayed(process_pixel)
results = mapper(process(i, pixel, tms) for i, pixel in enumerate(tqdm(pixel_bounds)))
```

# Step 5: Validation

- Reliability evaluation (BBOX validation & query validation)
- Data installation completeness validation
- Cross valiation with Elias coverage ratio dataset
- Scientific validation (compared with field ground truth data)

# Reliability evaluation

- Statistics value
  - data downloading & querying: randomly choose 3× pixels in Alaska, Canada, and Russia. Compare results from our workflow and the manually downloaded gpkg files and calculated in ArcGIS.

# Process validation

From 256*256 1km pixels, we will select the two pixels

In our workflow

Automatically generated 256*256 1km pixels
Choose two of pixels

Automatically generated BBOX

Query 1

1) get area sum in database

Two pixels manually generated via ArcGIS tools

Automatically generated BBOX query

Query 2

2) get area sum in database

Manually downloaded GPKGs within the pixel

3) Get area sum in ArcGIS

1) Compare data

2) Compare data

Conclusion: Results from 1), 2), 3) are all the same → Results generated from our code is consistent with data true value.

# Data installation

Uninterrupted setting

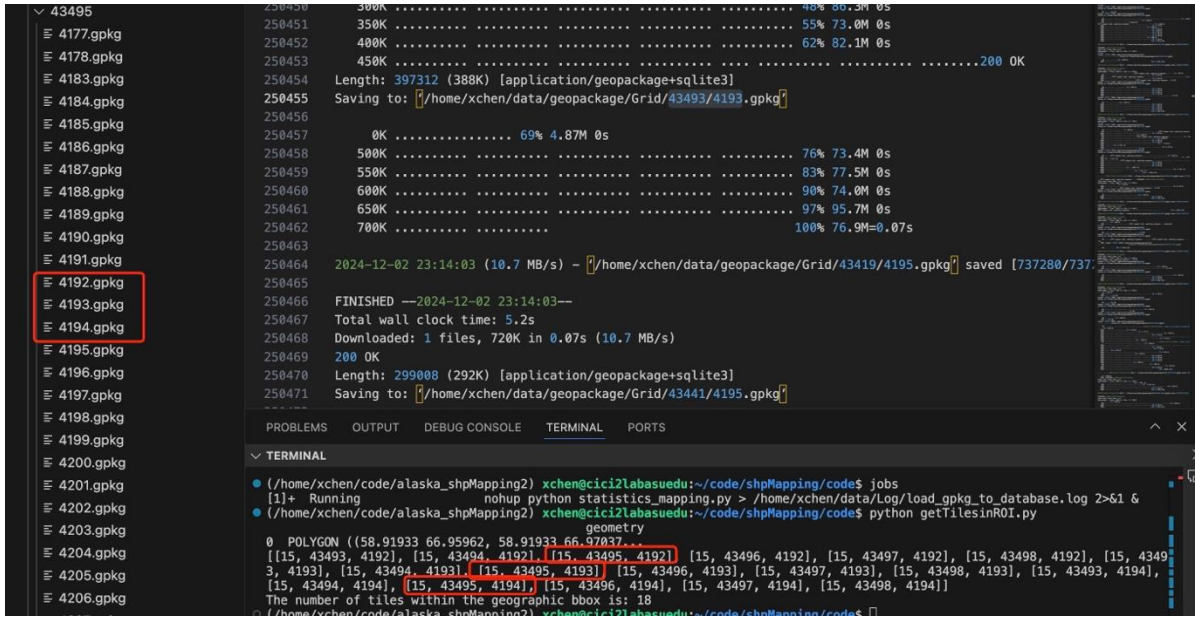
- Retry up to 30 times
- Timeout: 120 seconds

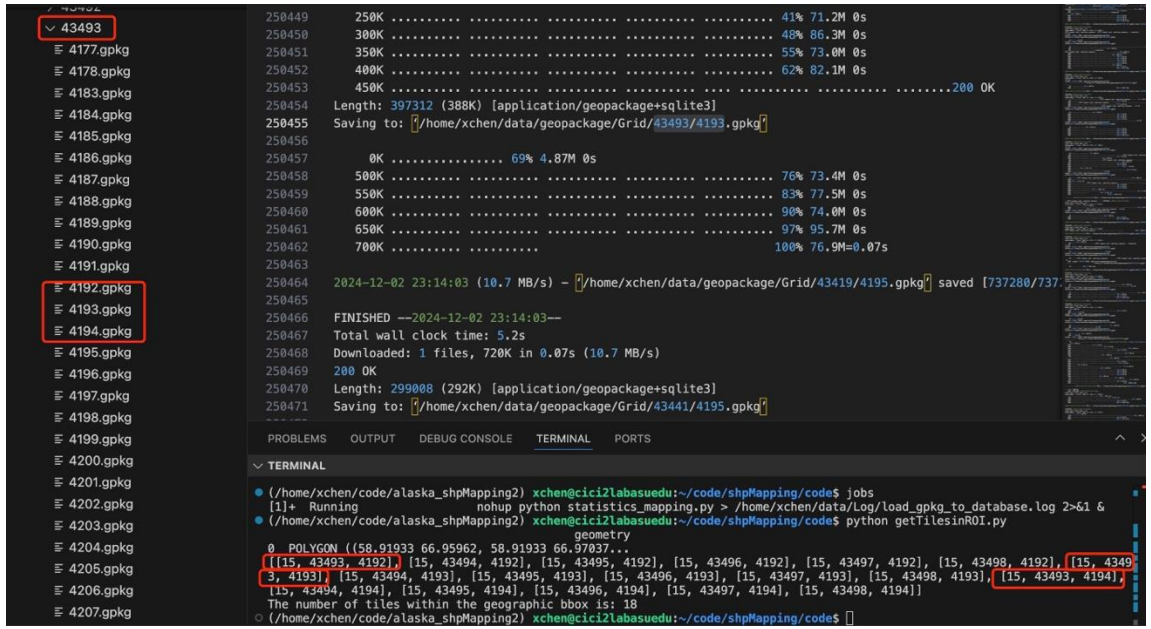
Results:

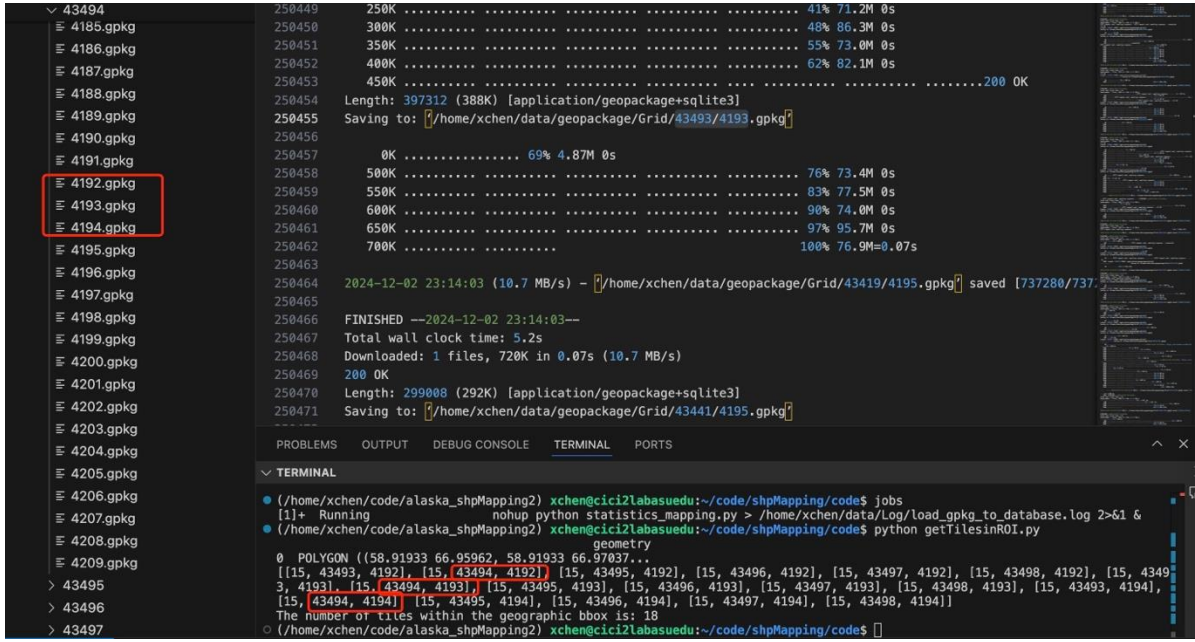
- No fetch failures due to various network connection issues (Connection, timeout, SSL, etc.)
- Total runtime for this grid: 5719 seconds (1.5h)

# Data installation

- Validation
  - Randomly select a pixel: Compare the tiles that the pixel actually overlapped with our installed GPKGs

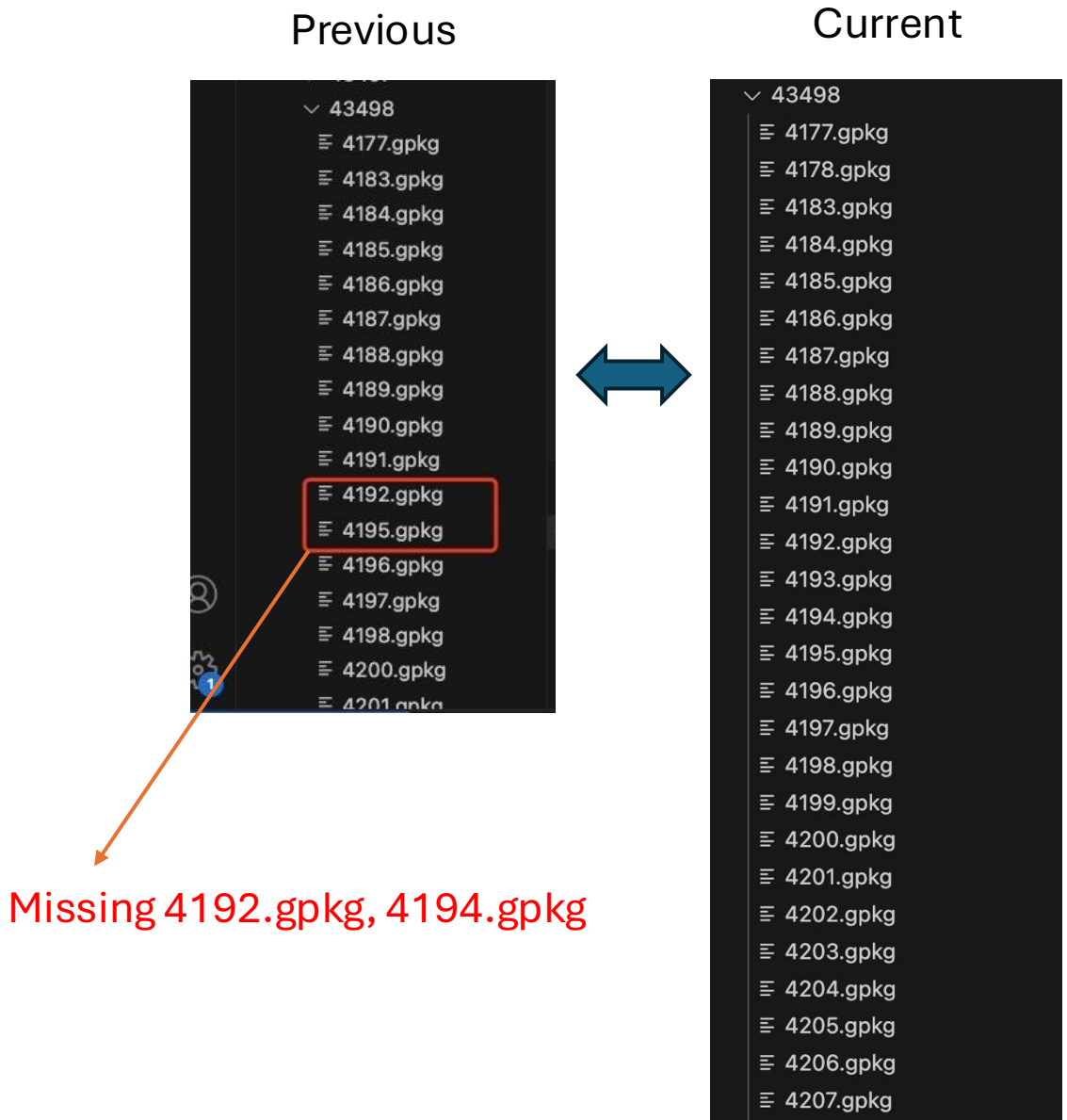


# Data installation

- Validation
  - Randomly selected a Tile X: Check if surrounding tiles are missing



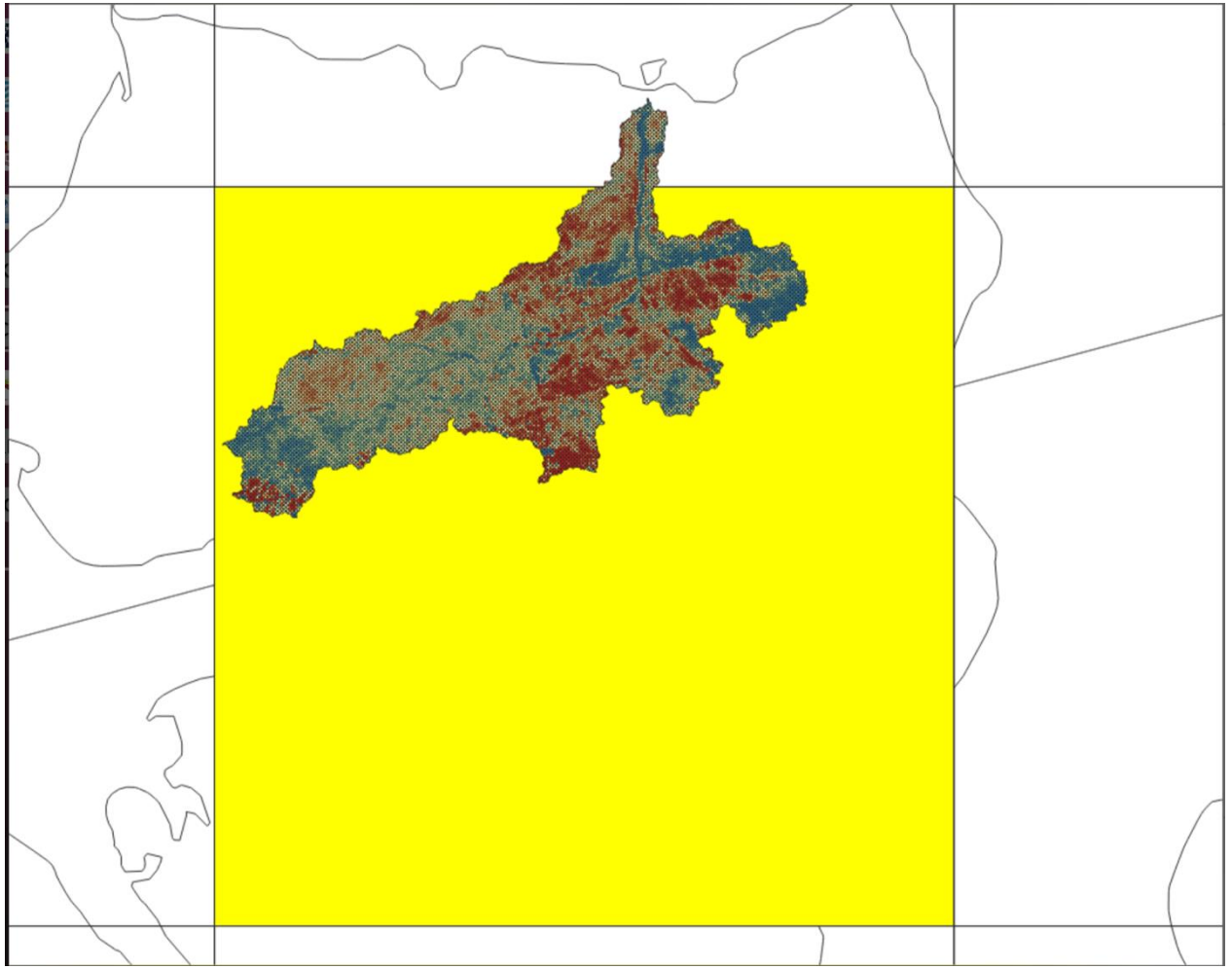
# Data installation

- Validation
  - Randomly selected a Tile X: Check continuous tiles

| Previous | | Current |
| --- | --- | --- |
| 18,826 GPKG<br>Within a grid |  | 22,157 GPKG<br>Within a grid |

# Cross validation

- Cross validation on percentage coverage calculation with Elias' results
  - Yellow is one 256km by 256km at 1km resolution tile in our processing
  - The other is the watershed in Bank island that Elias processed.



# Our results vs. Elias result on % coverage

- Computational efficiency improvement:
  - If we don’t consider data downloading time, each 256km^2 will take 20-30min on average.
  - So far, we are no longer using database
  - Processing the 240 tiles (a week)

# Scientific validation

Their IWP dataset starts from 2001 to 2021, if our IWP dataset get trained on the images (2005-2008) within one of their 5 sites, then it should be fine. I'm not sure anything about their field sites now, so couldn't know if the datasets are consistent (edited)

Diameters of IWP (max) and total area of IWP. The smaller the IWP, the more ice wedges we have on ground.

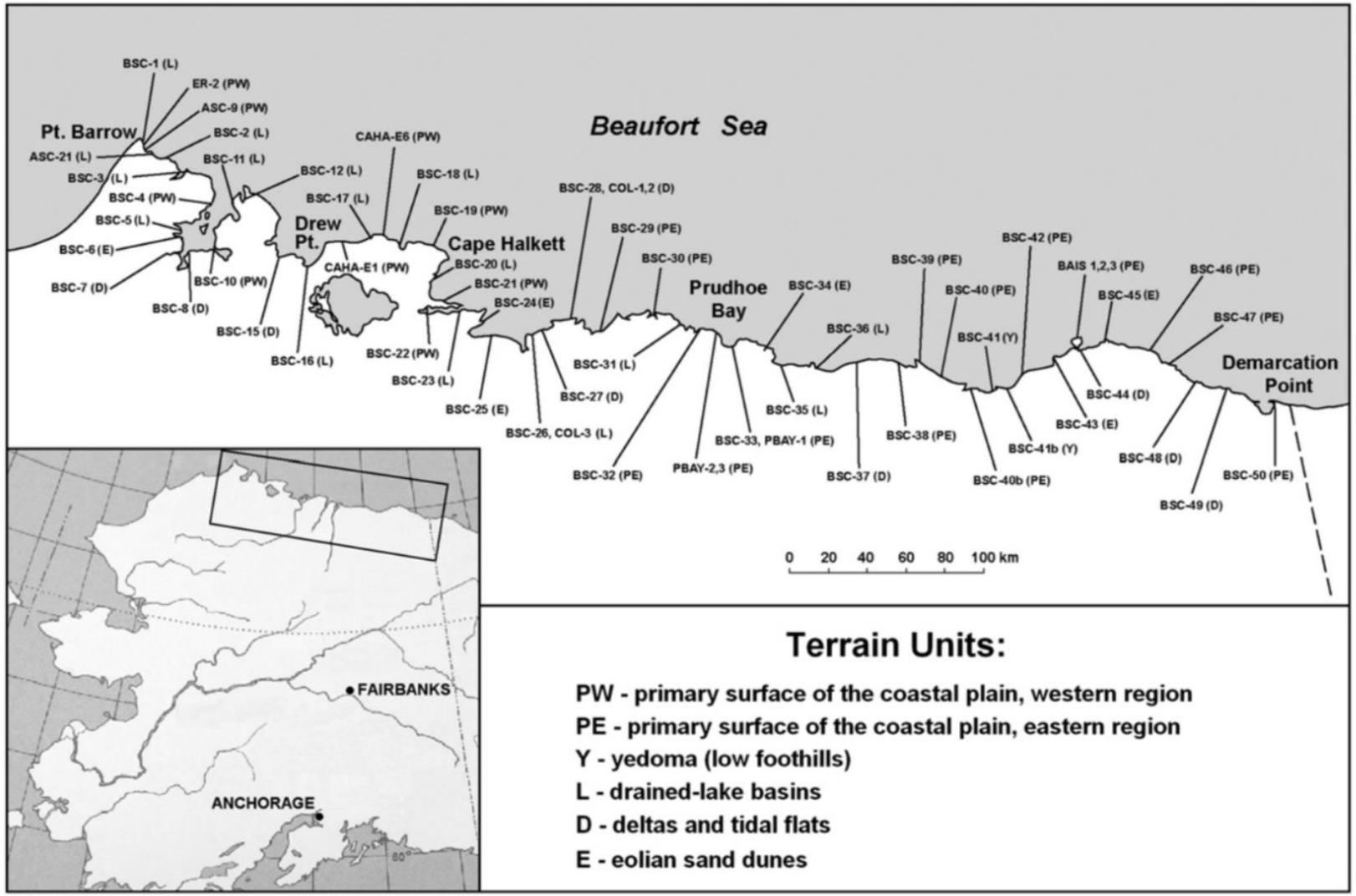
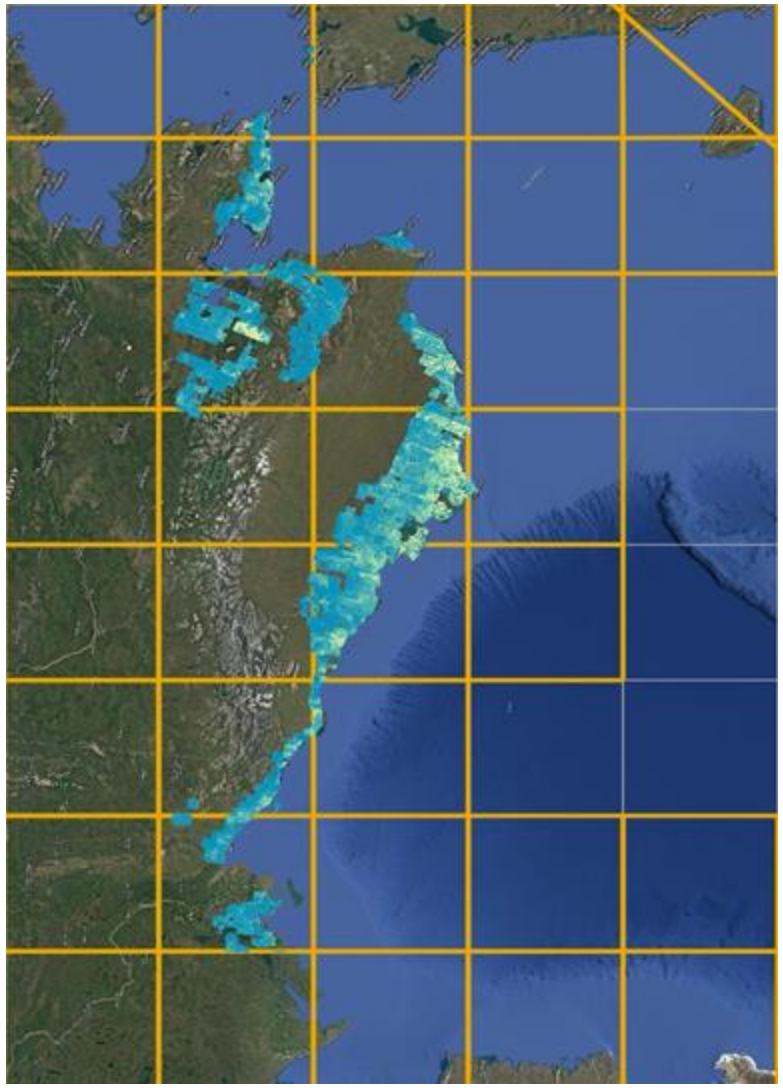



**Fig. 1.** Location of field sites studied in 2005–2008 along the Beaufort Sea coast from Point Barrow to the Canadian border.

https://www.sciencedirect.com/science/article/abs/pii/S0165232X12001644

# Step 6: Map all pixels within a grid, and slice all grids together



Current results in Alaska