# PRIOGRID Ingestion: WSF (Built Up Area)

## Processing Steps

### Retrieve WSF Evolution Data:

**FUNCTION: *summarize_wsf_tile_coverage*.**

1. **Summary:** The summarize_wsf_tile_coverage() function automatically retrieves and processes metadata from the WSF Evolution and WSF 2019 public directories hosted by DLR. It parses available GeoTIFF tile filenames, extracts their spatial identifiers, and compares the spatial coverage between the two datasets. It generates a summary report of shared and unique tiles and returns structured download information for downstream analysis.

2. **Utility:** This function is useful for establishing interoperability between two complementary remote sensing datasets by identifying spatial overlap. It supports automated data acquisition workflows, ensuring that analysis is performed only on matched tiles. This enables scalable, reproducible comparisons of urban change across time without relying on manually maintained file inventories.

**FUNCTION: *load_and_plot_tile_pair*.**

1. **Summary:** The load_and_plot_tile_pair() function downloads and loads a pair of raster tiles—one from the WSF Evolution dataset and one from the WSF 2019 dataset—based on a user-supplied tile_id. It constructs the appropriate download URLs, saves each tile to a temporary location, reads them as terra raster objects, and optionally displays them side by side for visual inspection. It returns a named list containing both rasters and the tile ID.

2. **Utility:** This function serves as a modular building block for tile-by-tile urban change analysis using WSF data. It abstracts away the mechanics of file retrieval, local storage, and raster loading, enabling users to work seamlessly with specific geographic tiles. Its optional plotting feature supports visual quality control, while its design makes it ideal for integration into larger workflows that loop over multiple tiles for batch processing.

### Preprocess select WSF Scenes:

**FUNCTION: *process_tile_pair*.**

1. **Summary:** The process_tile_pair() function prepares and compares raster layers from the WSF Evolution and WSF 2019 datasets for a single geographic tile. It first masks and recodes raw values in both rasters to isolate meaningful built-up data. Then, it resamples the WSF Evolution raster to match the resolution and grid of the 2019 raster, enabling pixel-level alignment. Using logical masks, the function identifies newly built-up areas in 2019 that were

not present in the historical record and separates them from areas already built. It returns only the resampled evolution raster and the isolated new 2019 development raster.

2. **Utility:** This function is essential for detecting and isolating urban expansion at fine resolution. It enables spatially consistent comparison between legacy (1985–2015) and recent (2019) built-up surfaces by harmonizing resolution and masking overlap. Designed for integration into tile-based workflows, it supports downstream operations such as merging, statistical summarization, and map generation while remaining memory-efficient. Its minimal return structure ensures only essential outputs are carried forward, avoiding unnecessary memory accumulation in batch processing pipelines.

## PG processes

**FUNCTION: *filter_pg_cells_within_raster*.**

1. **Summary:** The filter_pg_cells_within_raster() function filters a set of spatial polygons—specifically PRIO-GRID cells (pg_sf)—to retain only those that fall entirely within the spatial extent of a given raster layer. It first reprojects the polygon geometries to match the coordinate reference system (CRS) of the raster, then constructs a bounding polygon from the raster's extent. It uses spatial containment logic to return only those polygons that are fully enclosed by the raster boundary.

2. **Utility:** This function ensures that spatial analyses such as zonal statistics or raster masking are applied only to polygons that are completely covered by a raster scene. This is crucial for maintaining consistency in per-polygon metrics, avoiding edge effects, and preventing partial raster coverage from skewing results. Designed for use in tile-based raster processing workflows, it improves computational efficiency and data reliability by pre-filtering valid analysis zones.

**FUNCTION: .**

1. **Summary:** summarize_built_up_by_pg_cell() computes pixel-based zonal statistics from a merged WSF raster (r_combined) within a set of PRIO-GRID polygons. It calculates the total built-up area, the proportional built-up coverage, and the proportion of area built within defined time intervals. All measurements are normalized by the actual area of each PG cell, ensuring geospatial accuracy.

2. **Utility:** This function is a critical step for extracting interpretable metrics of urban change from raster data. It enables fine-grained temporal and spatial analysis by quantifying not just how much land was developed, but when it occurred, per PG cell. Its use of projected coordinate systems ensures accurate area calculations, making the output directly usable for comparative studies, statistical modeling, and policy-relevant spatial analysis.

## Compile WSF dataset at PG resolution

**FUNCTION: *process_all_tiles_summary*.**

1. **Summary:** process_all_tiles_summary() is a high-level wrapper function that automates the end-to-end processing of multiple WSF Evolution and WSF 2019 raster tile pairs. For each tile, it loads, processes, merges, filters relevant PRIO-GRID cells, computes zonal statistics,

and appends results to a cumulative data frame. It also times each iteration and cleans memory to ensure efficient large-scale execution.

2. **Utility:** This function is ideal for scaling urban expansion analysis across large spatial extents. By integrating all core processing steps—including tile loading, raster alignment, zonal masking, and statistical summarization—it streamlines the generation of built-up area metrics at the PG cell level. It is memory-efficient, extensible, and compatible with parallelized or batch workflows, making it suitable for integration into reproducible spatial data pipelines.