CS 205 Project - M3

Parallelizing Crop Phenology reports via NDVI datasets

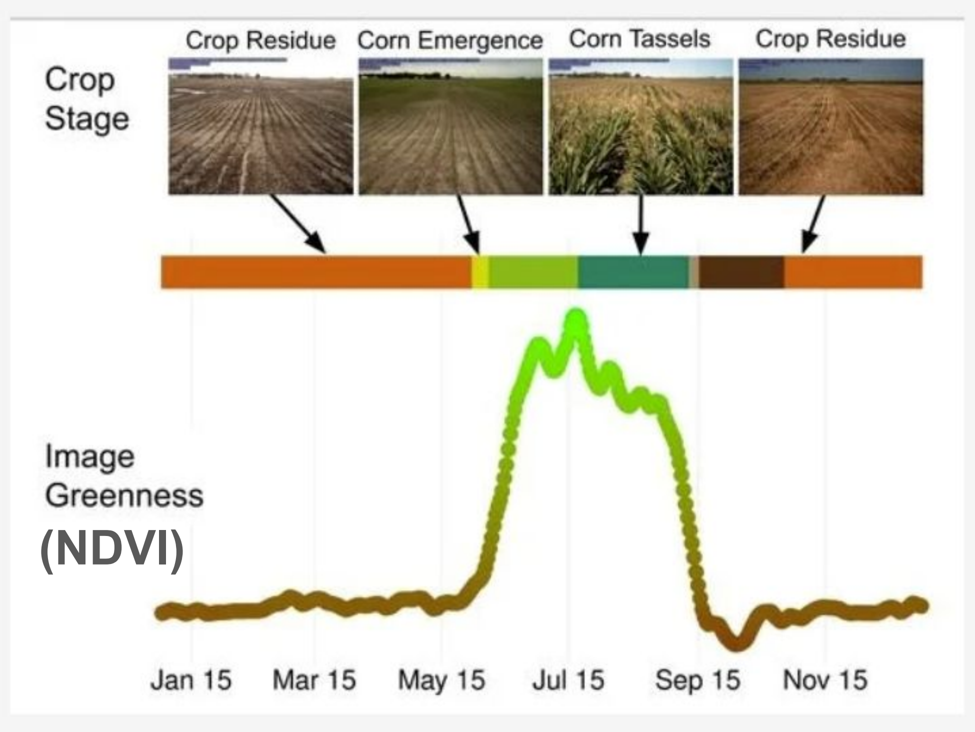
Team 18:

Elaine Swanson, Alex Johnson, Saketh Mynampati, Alyssa Ross

Problem: Crop phenology using NDVI data

- What is crop phenology (crop stages)?
- How do we get crop phenology using the Normalized Difference Vegetation Index (NDVI = greenness)?

$$NDVI = \frac{NIR - RED}{NIR + RED}$$



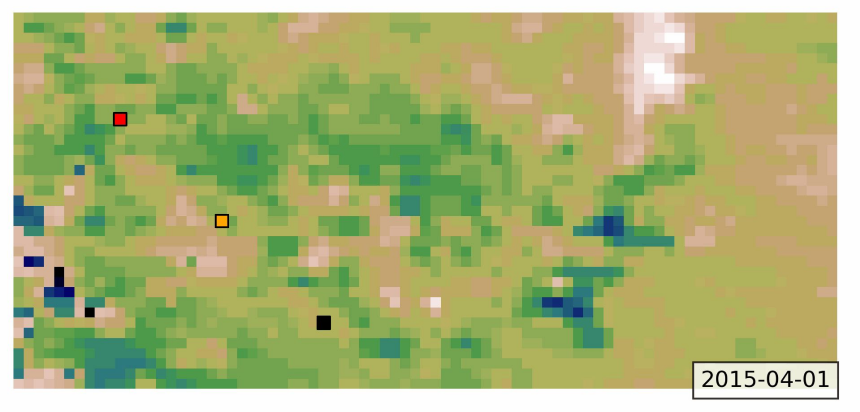
Source: https://www.mdpi.com/2072-4292/14/2/286

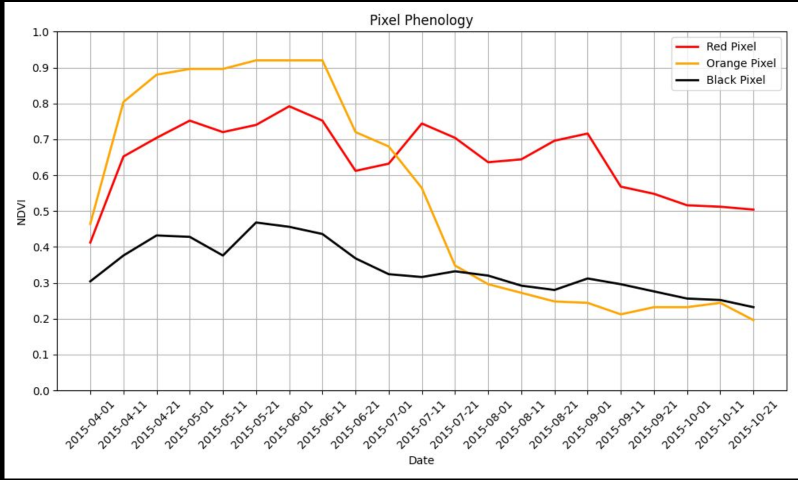
- How do we do this efficiently over ~380,000 km² of land and over two decades? So far that is ~ 4.2 billion potential reports

Data and Model:

- **Dataset name:**
  CGLS Collection NDVI Version 2
- Obtained by the Sentinel-3 Satellite (European Space Agency) data products via the **openEO** API and **VITO** Backend
- **Temporal information:**
  Three products per month
  dated day **1**, day **11** and day **21** of each month
- **Spatial Information:**
  Global coverage, but just pulling and focusing on the North West USA, 300 meter resolution
- Forms hypercube $\mathbf{I}(x,y,t,\omega)$
- **Limitations:**
  Can't differentiate between intended agricultural crops and natural vegetation (ex. forests, marshlands)
  - Need to overlap with actual crop parcel data. Crop delineation dataset obtained via U-Net

NDVI of Baker Valley, OR in 2015





Need for Shared Parallelization:

- Create NDVI dataset, create parcel dataset, create crop phenology reports
  - **Crop Parcels → Pixels → Tiles**
- A tiling scheme with many small tiles increases overhead and can introduce extra latency.
- → multi-threading and tiling must be combined
- **Goal:** maximize a version of **Amdahl's Speedup Law**:
  - For **P** = parallelization, **N** = number of tiles, **n** = subset of tiles due to lack of cores/memory, **T** = number of threads, **C** = number of cores
  -

$$S = S_{tiling}(N) \cdot S_{threading}\left(\left\lfloor \frac{C}{n} \right\rfloor\right) \cdot \frac{n}{N}$$

Where $\left\lfloor \frac{C}{n} \right\rfloor = T$ and $\frac{n}{N}$ corrects for latency

Draft Plan:

- Formulate our spatial aggregation workflow using the openEO VITO backend
- Using main package:

pyjeo

  - largely depends of the **JIPlib** library, which is implemented in **C++** and contains three main classes: Jim, JimList and VectorOgr. Other dependency in **C++** is **miallib.**
    - **Jim** is the main class to represent raster data objects

  - **Have to almost rebuild this open source package due to backend restrictions by Italian government** 🤌

  - Then use Python interface via **Simplified Wrapper and Interface Generator (SWIG)**

- **pyjeo** supports multi-threading using OpenMP API and setting CPU affinity