CWI

Centrum Wiskunde & Informatica

Assignment 1: Querying a Social Graph

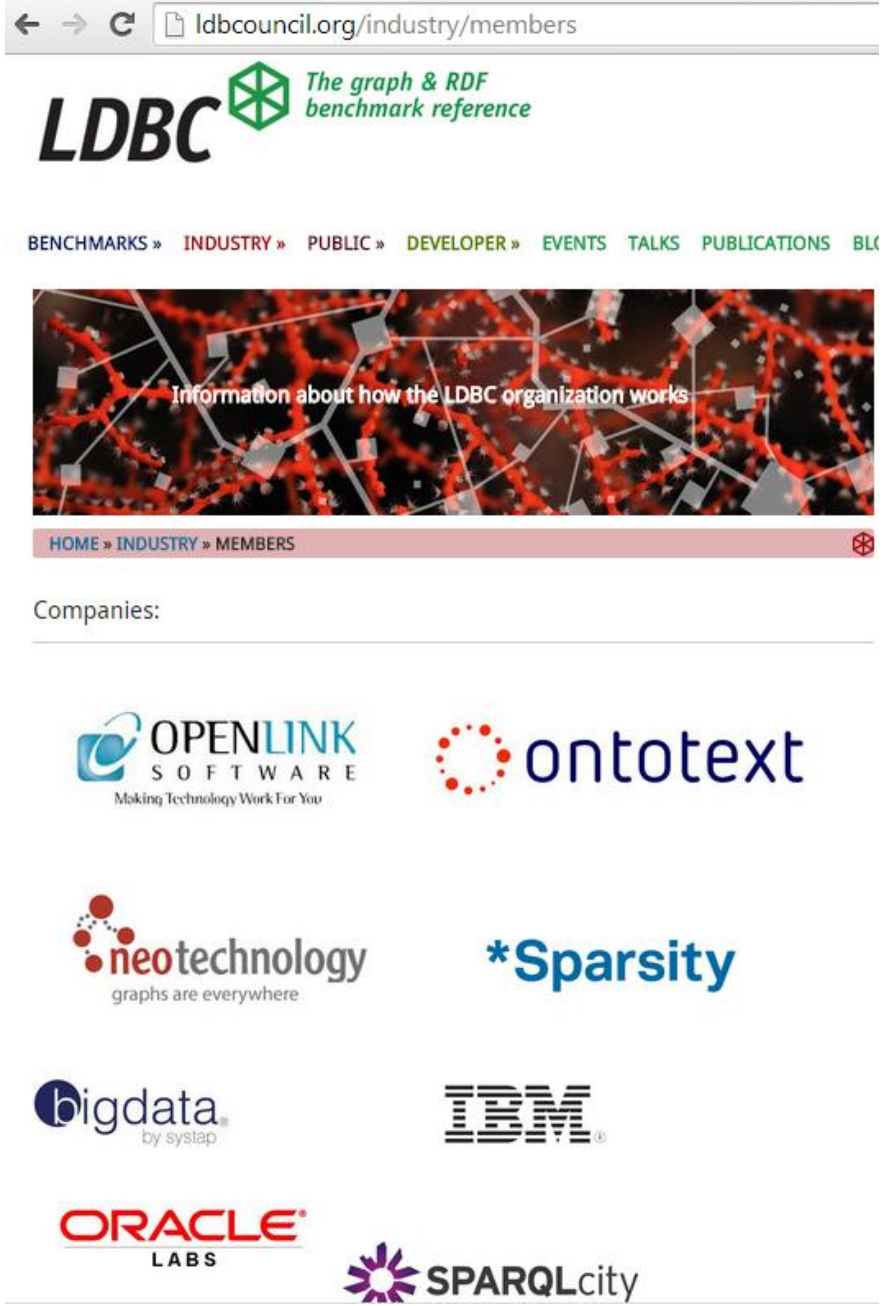
CWI
Centrum Wiskunde & Informatica

LDBC Data generator

- Synthetic dataset available in different scale factors
  - SF100 🡐 for quick testing
  - SF3000 🡐 the real deal
- Very complex graph
  - Power laws (e.g. degree)
  - Huge Connected Component
  - Small diameter
  - Data correlations

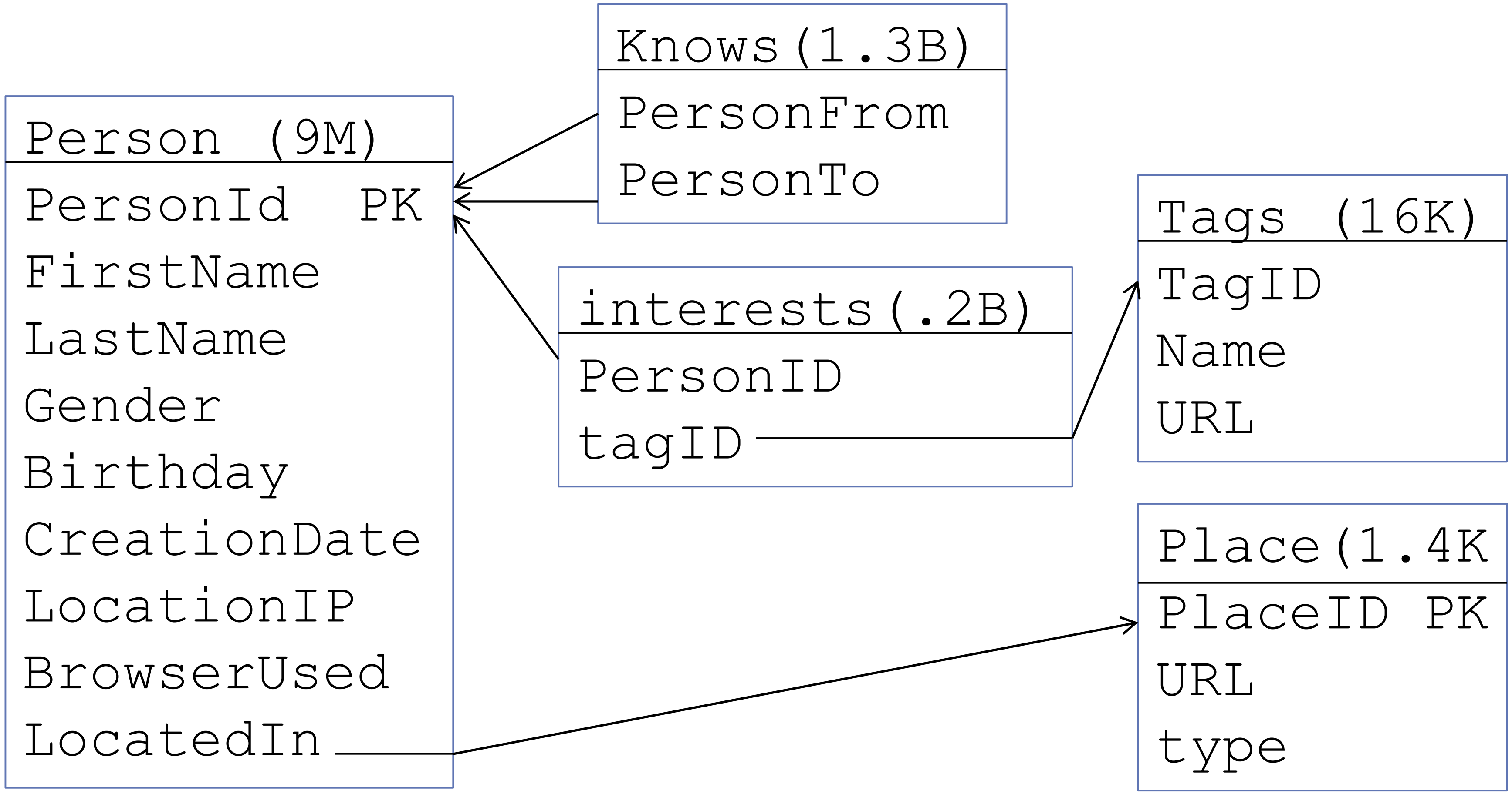
    Chinese have more Chinese names
  - Structure correlations

    Chinese have more Chinese friends



CSV file schema

- See: http://wikistats.ins.cwi.nl/lsde-data/practical_1
- Counts for sf3000 (total 37GB)



The Query

- The marketeers of a social network have been data mining the musical preferences of their users. They have built statistical models which predict given an interest in say artists A2 and A3, that the person would also like A1 (i.e. rules of the form: A2 and A3 ➔ A1). Now, they are commercially exploiting this knowledge by selling targeted ads to the management of artists who, in turn, want to sell concert tickets to the public but in the process also want to expand their artists' fanbase.
- The ad is a suggestion for people who already are interested in A1 to buy concert tickets of artist A1 (with a discount!) as a birthday present for a friend ("who we know will love it" - the social network says) who lives in the same city, who is not yet interested in A1 yet, but is interested in other artists A2, A3 and A4 that the data mining model predicts to be correlated with A1.

The Query

For all persons P :

- *who have their birthday on or in between D1..D2*
- *who do not like A1 yet*

  we give a score of

  - *1 for liking any of the artists A2, A3 and A4 and*
  - *0 if not*

  the final score, the sum, hence is a number between 0 and 3.

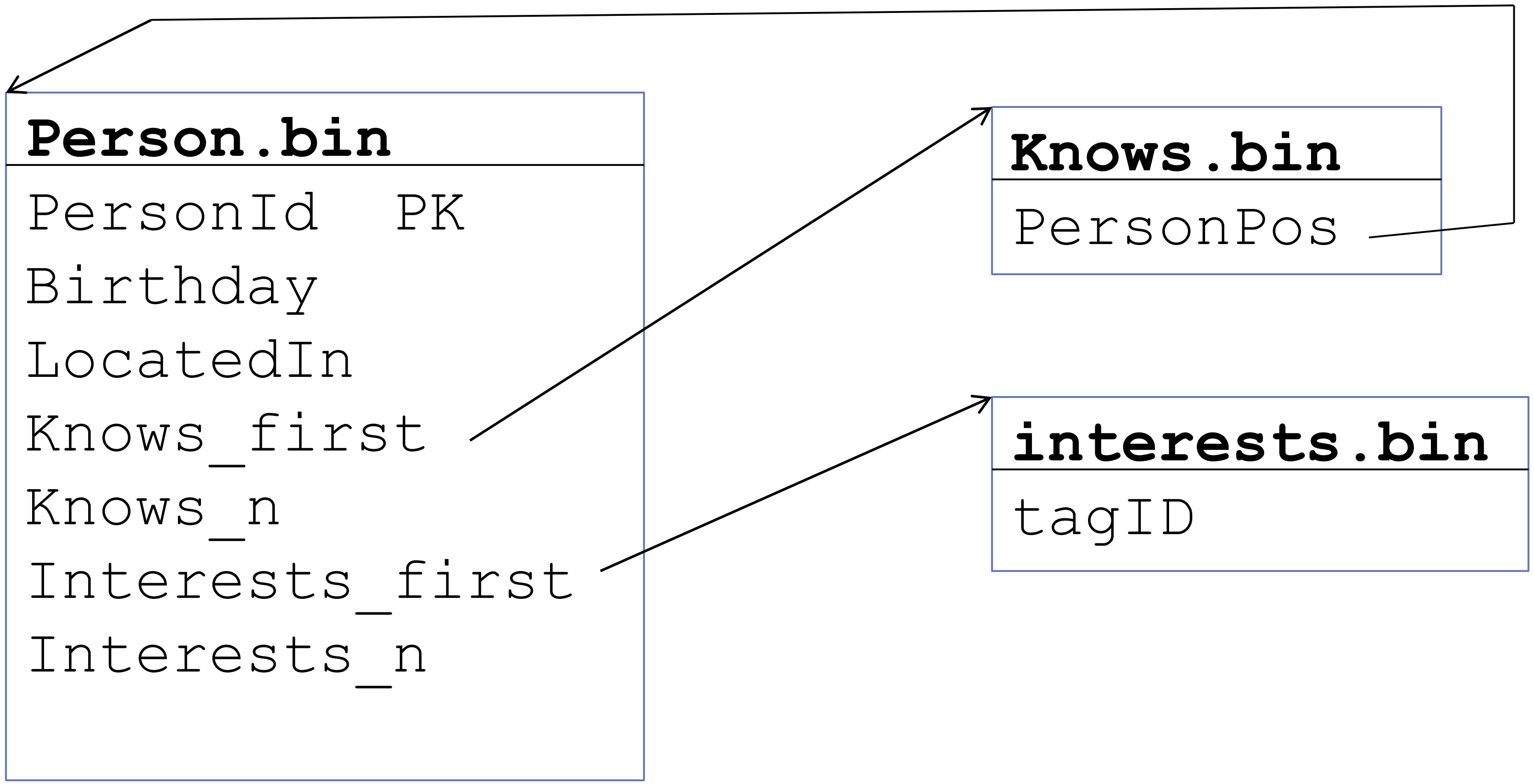
  Further, we look for friends F:

  - *Where P and F who know each other mutually*
  - *Where P and F live in the same city and*
  - *Where F already likes A1*

    The answer of the query is a table (score, P, F) with only scores > 0

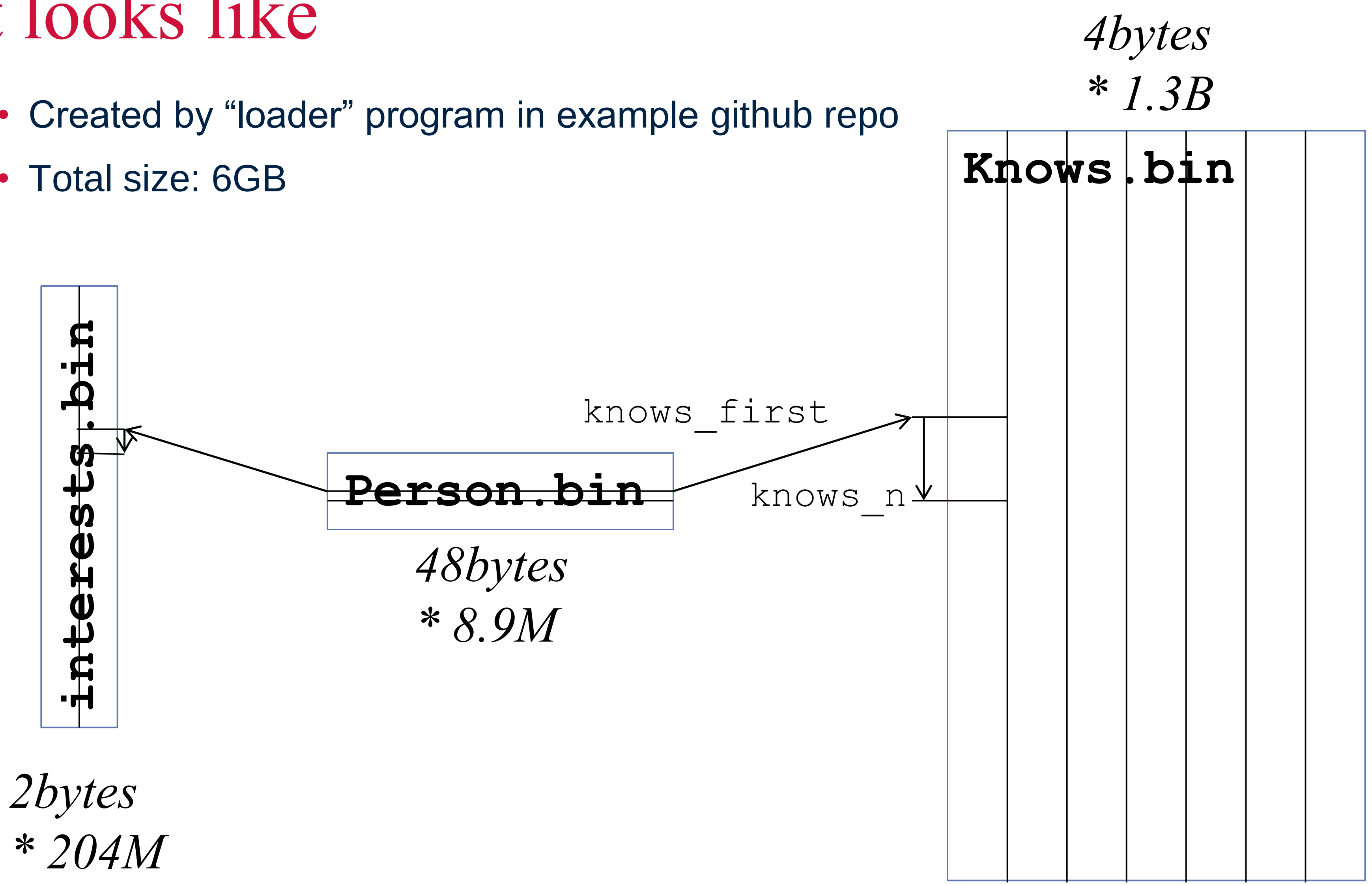
Binary files

- Created by “loader” program in example github repo
- Total size: 6GB



What it looks like

- Created by “loader” program in example github repo
- Total size: 6GB



The Naïve Implementation

The "cruncher" program

Go through the persons P sequentially

- *counting how many of the artists A2,A3,A4 are liked as the score*

  for those with score>0:
  - *visit all persons F known to P.*

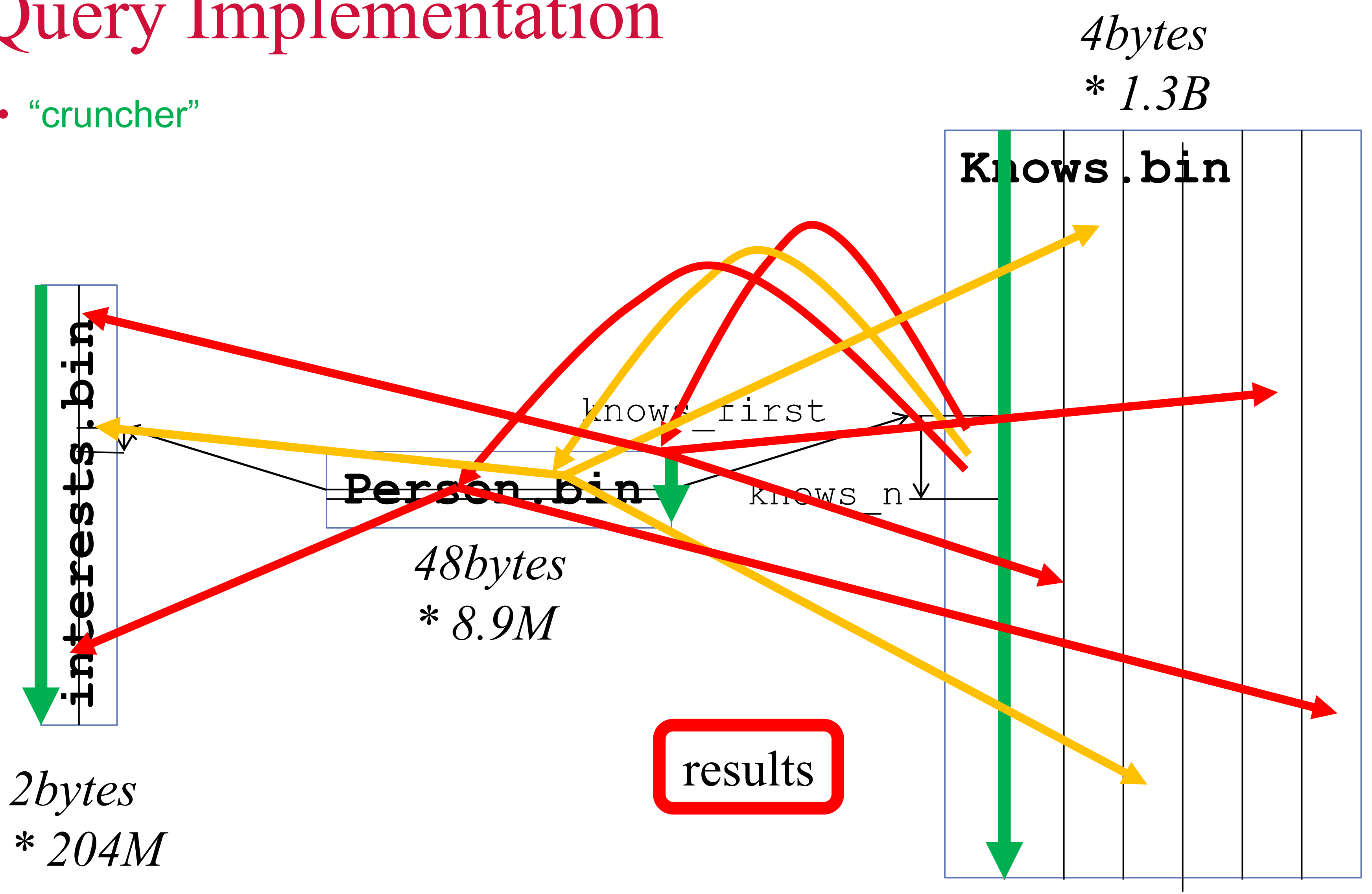
    For each F:
    - *checks on equal location*
    - *check whether F already likes A1*
    - *check whether F also knows P*

    if all this succeeds (score,P,F) is added to a result table.

CWI
Centrum Wiskunde & Informatica

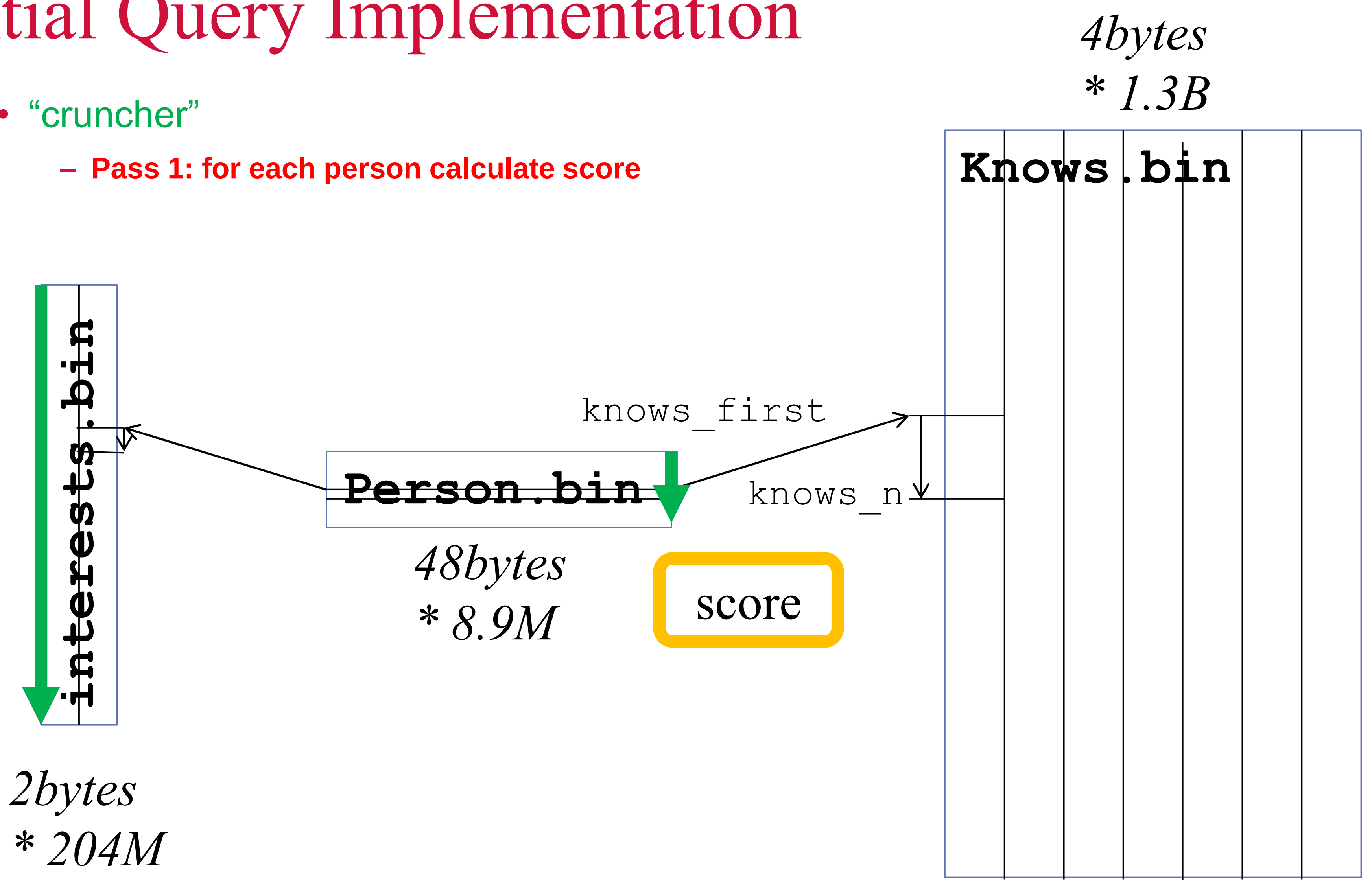
Naïve Query Implementation

- "cruncher"



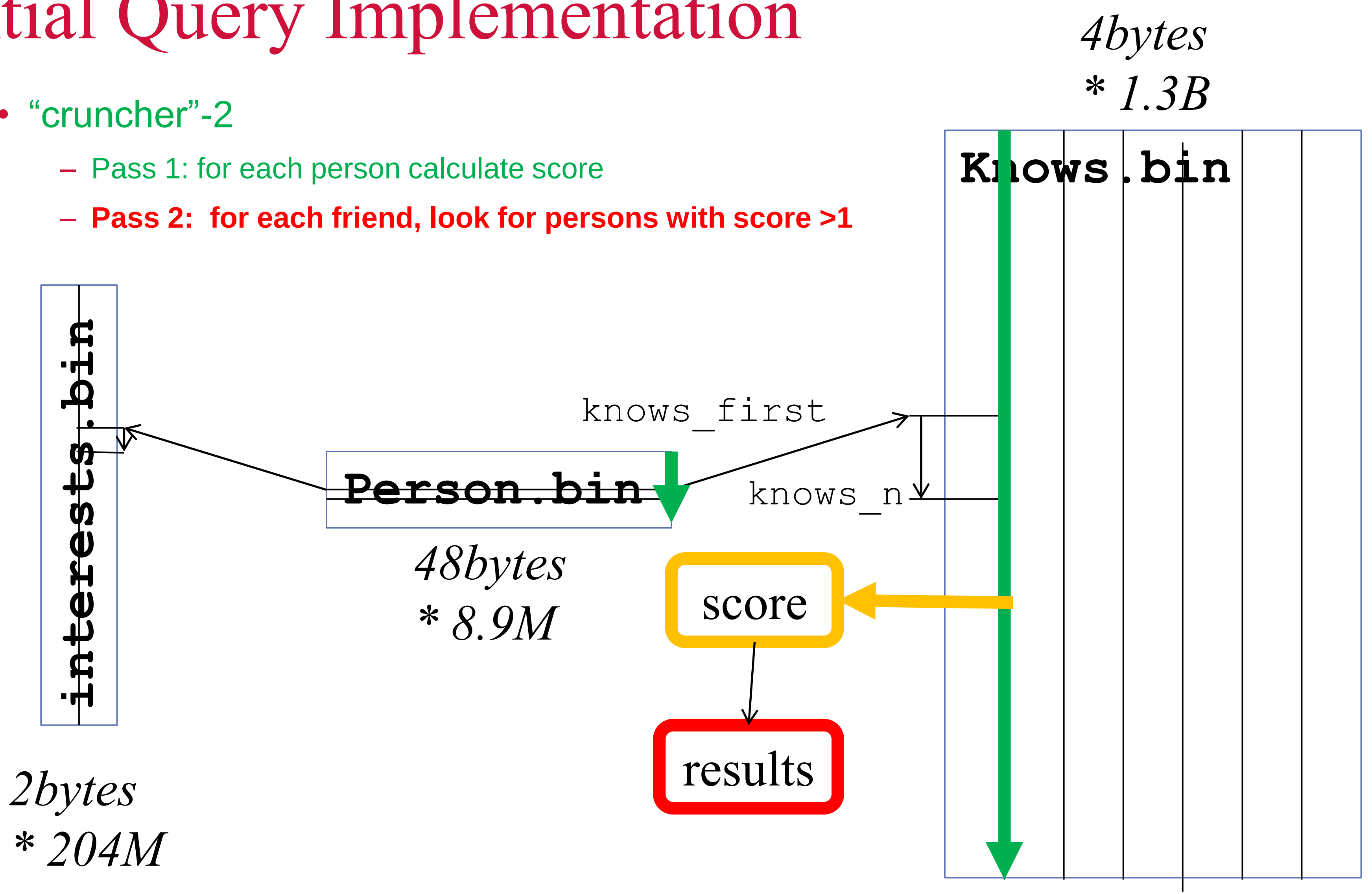
Sequential Query Implementation

- "cruncher"
  - **Pass 1: for each person calculate score**



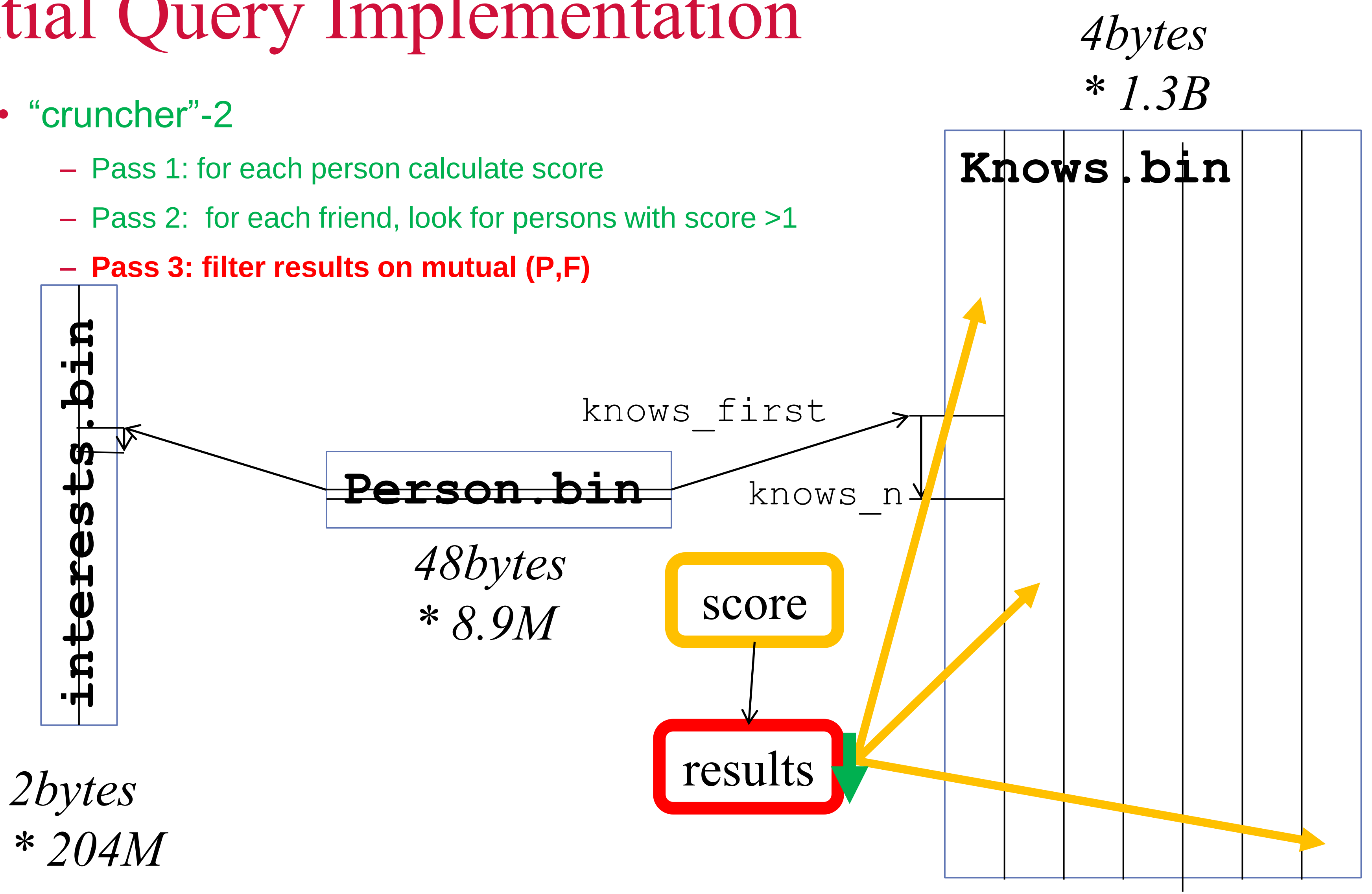
Sequential Query Implementation

- "cruncher"-2
  - Pass 1: for each person calculate score
  - **Pass 2: for each friend, look for persons with score >1**



Sequential Query Implementation

- "cruncher"-2
  - Pass 1: for each person calculate score
  - Pass 2: for each friend, look for persons with score >1
  - **Pass 3: filter results on mutual (P,F)**



Improving Bad Access Patterns

- Minimize Random Memory Access
  - Apply filters first. Less accesses is better.
- Denormalize the Schema
  - Remove joins/lookups, add looked up stuff to the table (but.. makes it bigger)
- Trade Random Access For Sequential Access
  - perform a 100K random key lookups in a large table

    ➔ put 100K keys in a hash table, then

    scan table and lookup keys in hash table
- Try to make the randomly accessed region smaller
  - Remove unused data from the structure
  - Apply data compression
  - Cluster or Partition the data (improve locality) …hard for social graphs
- If the random lookups often fail to find a result
  - Use a Bloom Filter