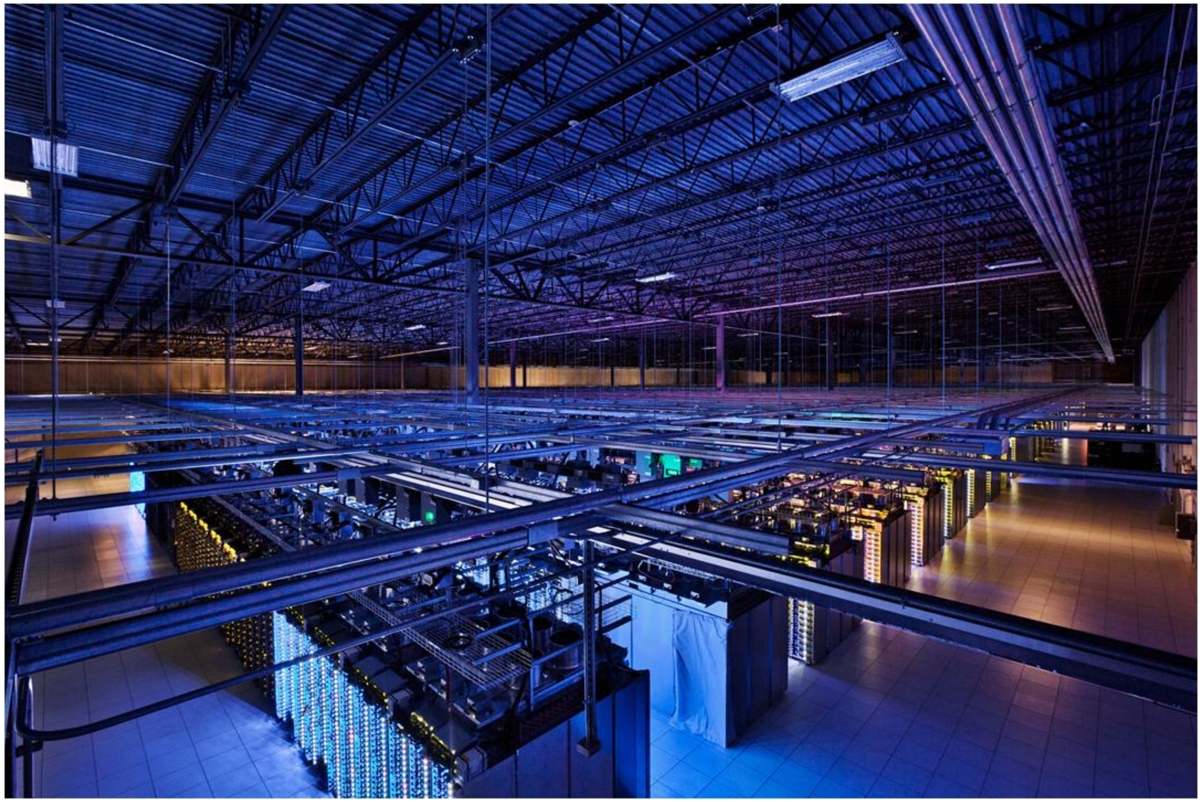
CWI
Centrum Wiskunde & Informatica

Large-Scale Data Engineering

Some notes on Access Patterns, Latency, Bandwidth



+ Tips for practical

event.cwi.nl/lsde

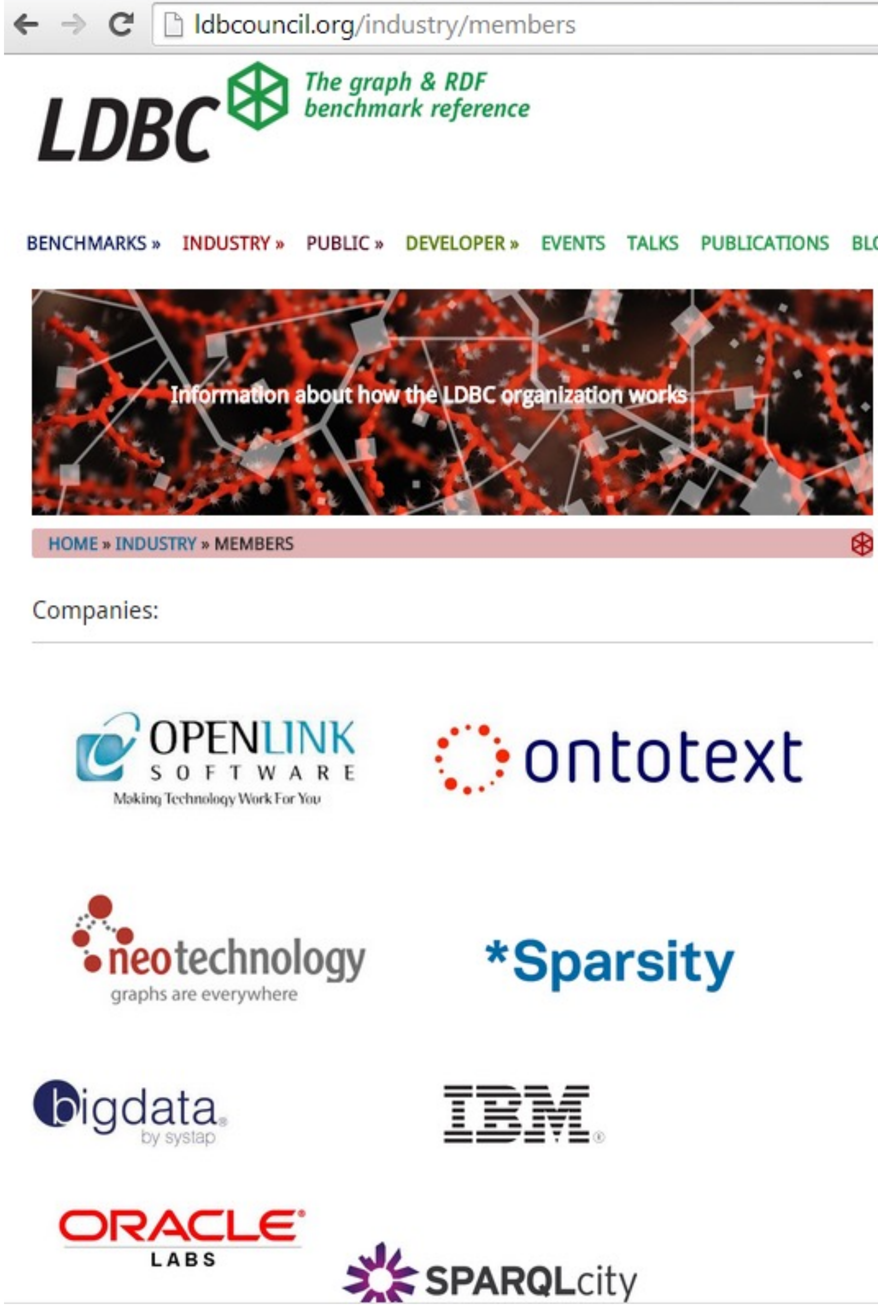
Assignment 1: Querying a Social Graph

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

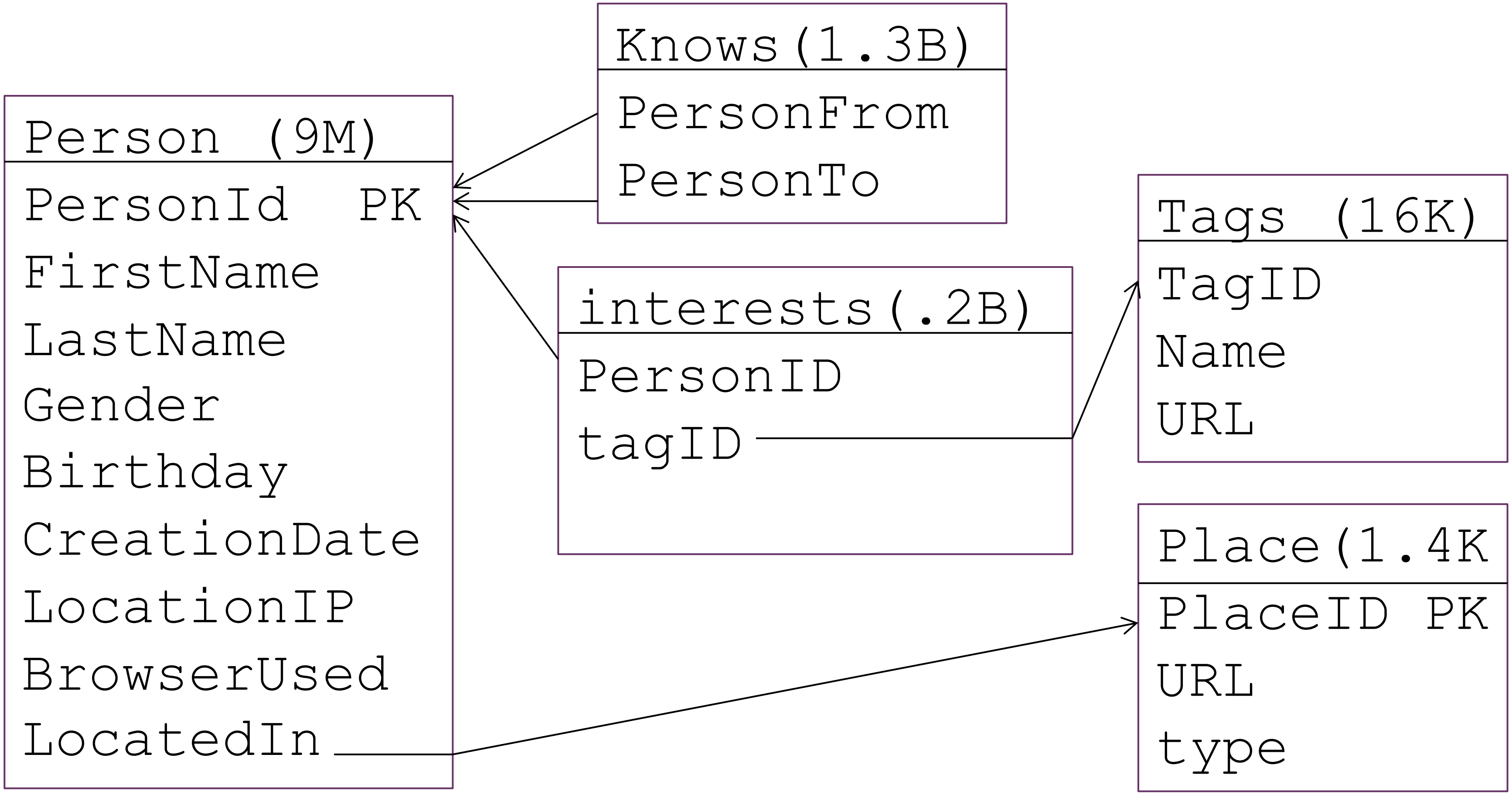
    Chinese have more Chinese friends



Centrum Wiskunde & Informatica

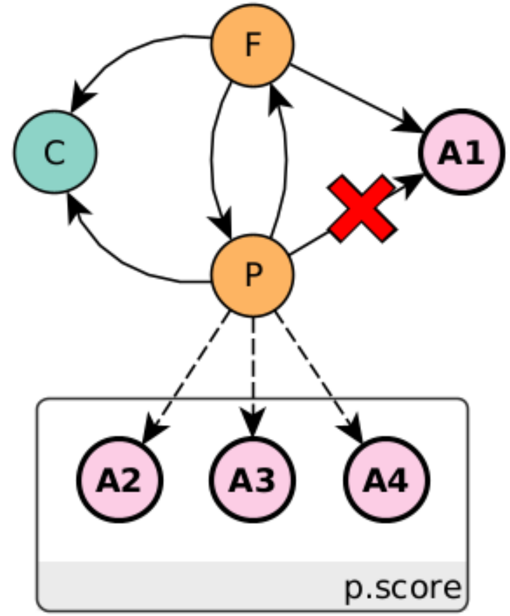
CSV file schema

- See: https://event.cwi.nl/lsde/data (sf100 only)
- Counts for sf3000 (total size: 37GB CSV, 7GB bz2 compressed)



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 (3 for the price of 2!) for your birthday celebration birthday to invite two of your friends ("who we know will love it" - the social network says), who are also friends themselves, who live in the same city, who are not yet interested in A1 yet, because they are interested in other artists A2, A3 and A4 that the data mining model predicts to be correlated with A1.



Person P's birthday (month/day) is in [D1, D2]

The Query

For all persons P :

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

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:

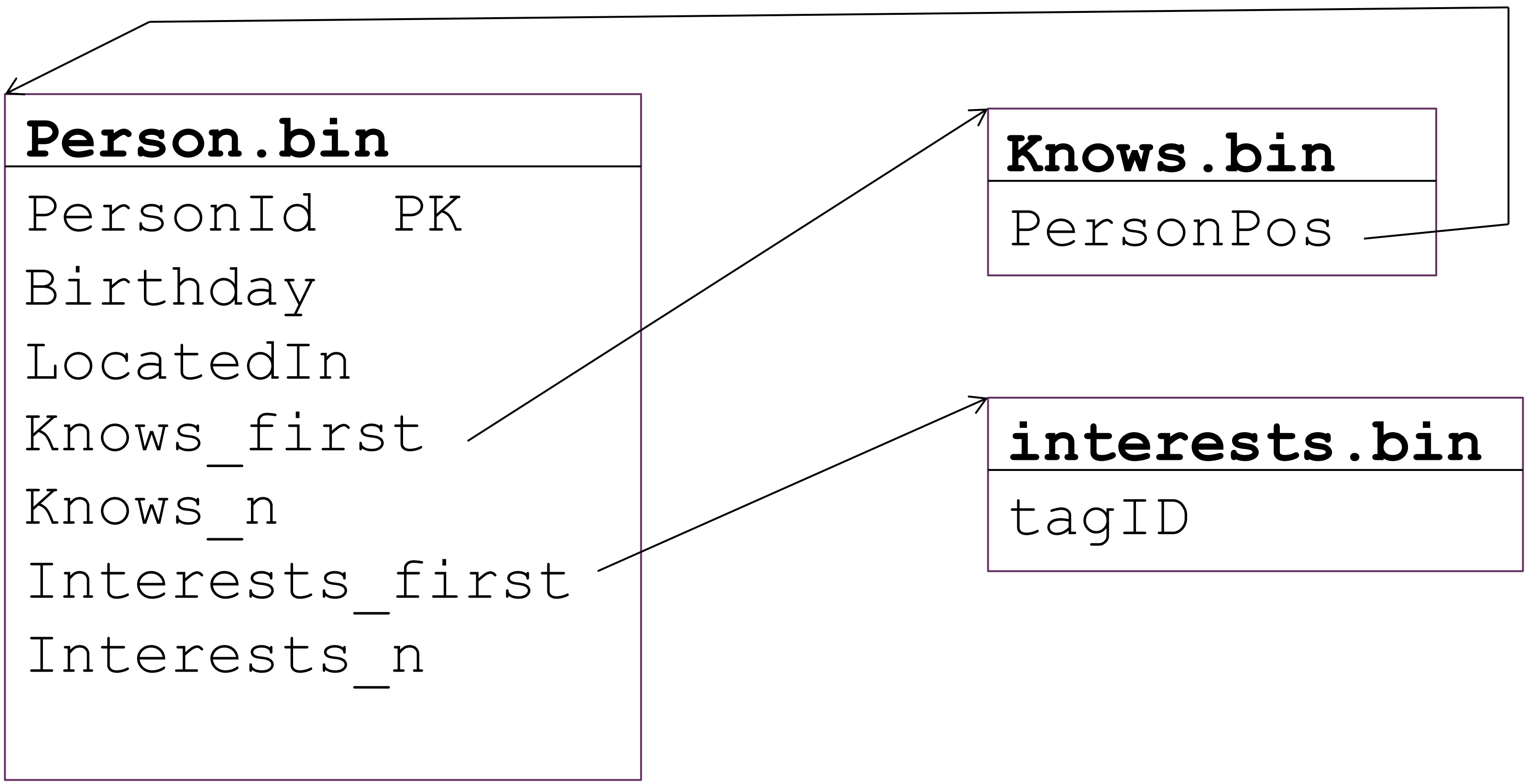
- *Where P and F who know each other mutually*
- *Where P and F live in the same city and*
- *Where F already likes A1*

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

Centrum Wiskunde & Informatica

Binary files

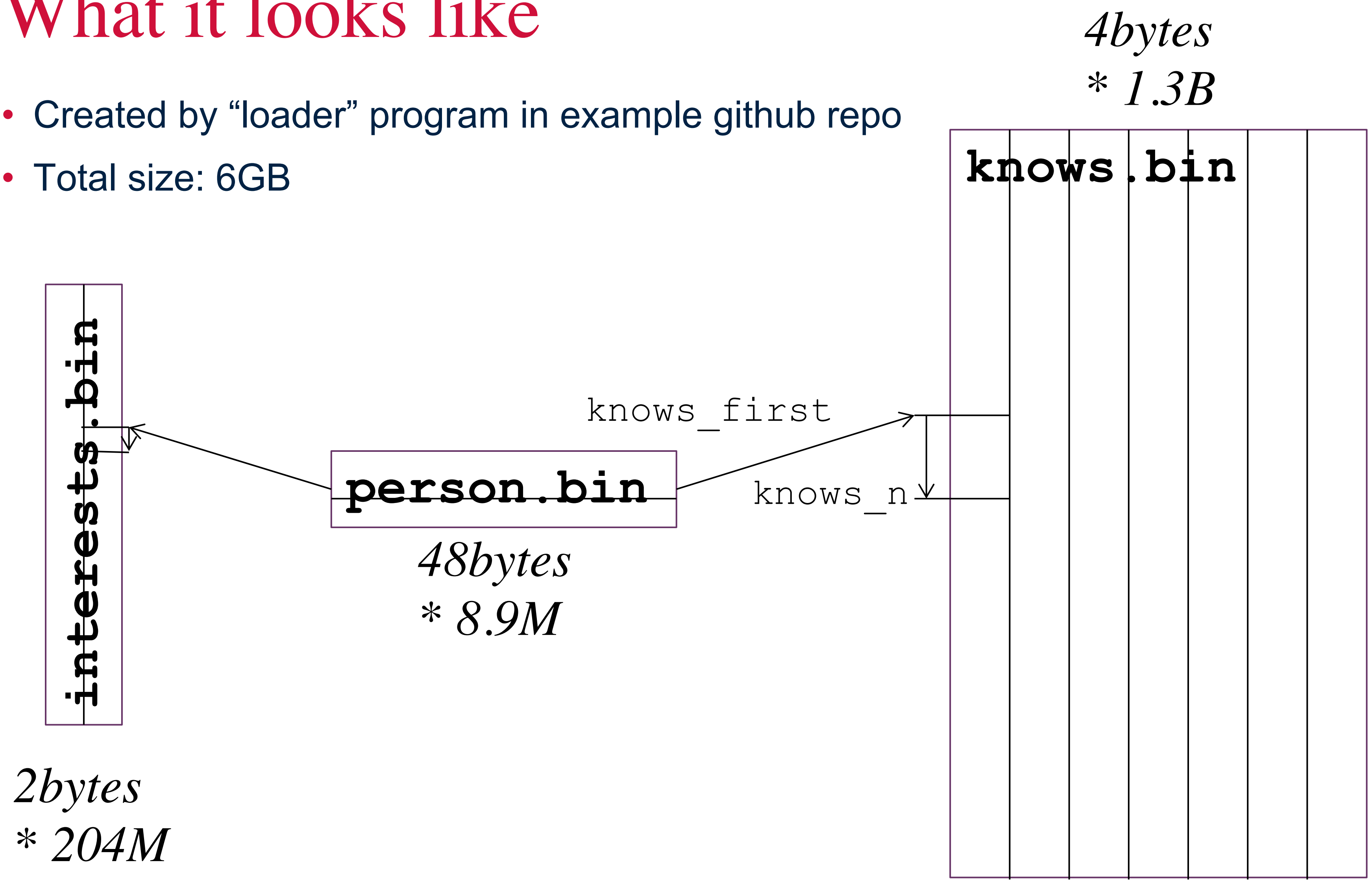
- Created by “loader” program in example github repo
- Total size: 6GB



Centrum Wiskunde & Informatica

What it looks like

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



The Naïve Implementation

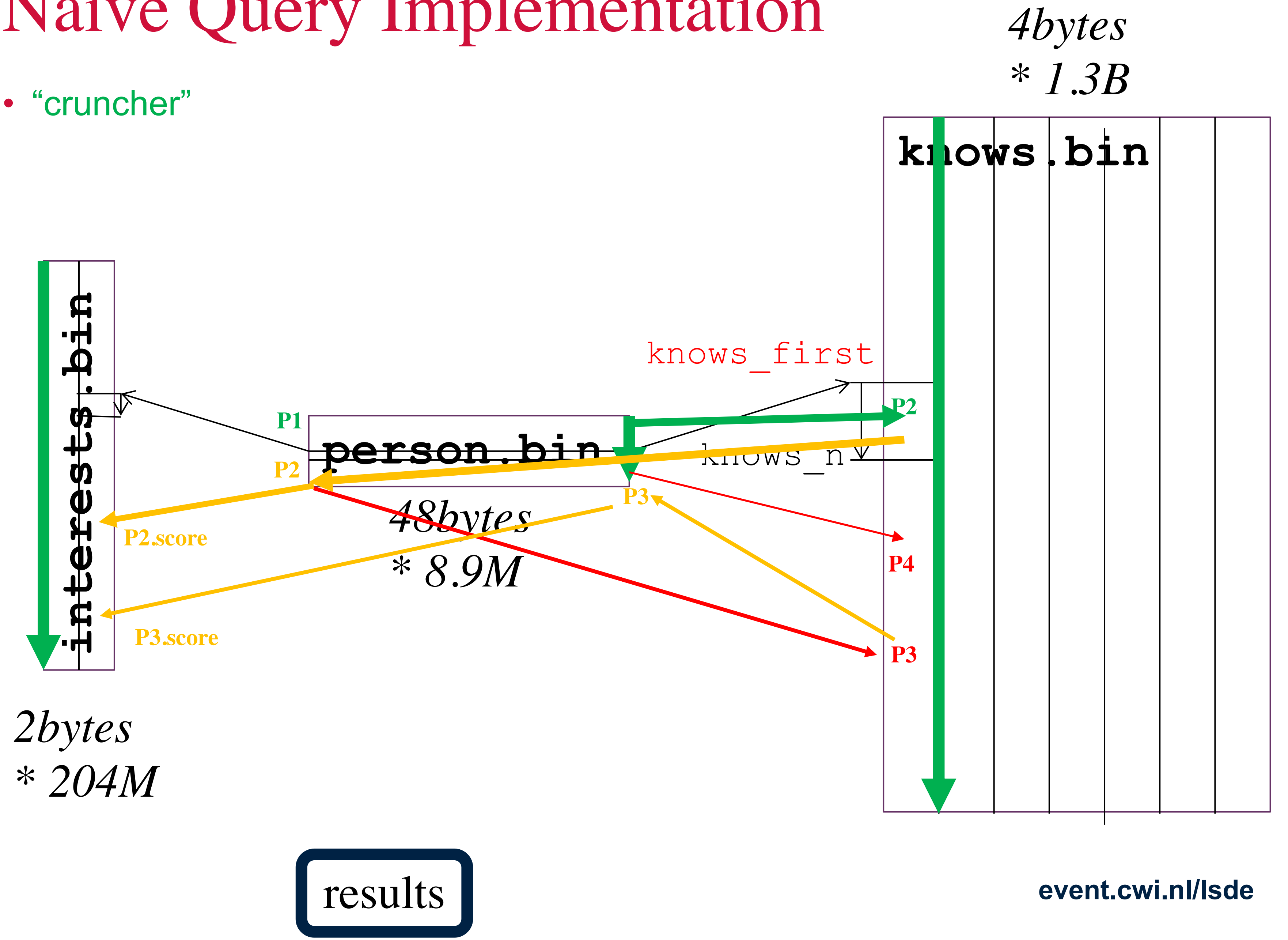
The "cruncher" program

Go through the persons P1 sequentially

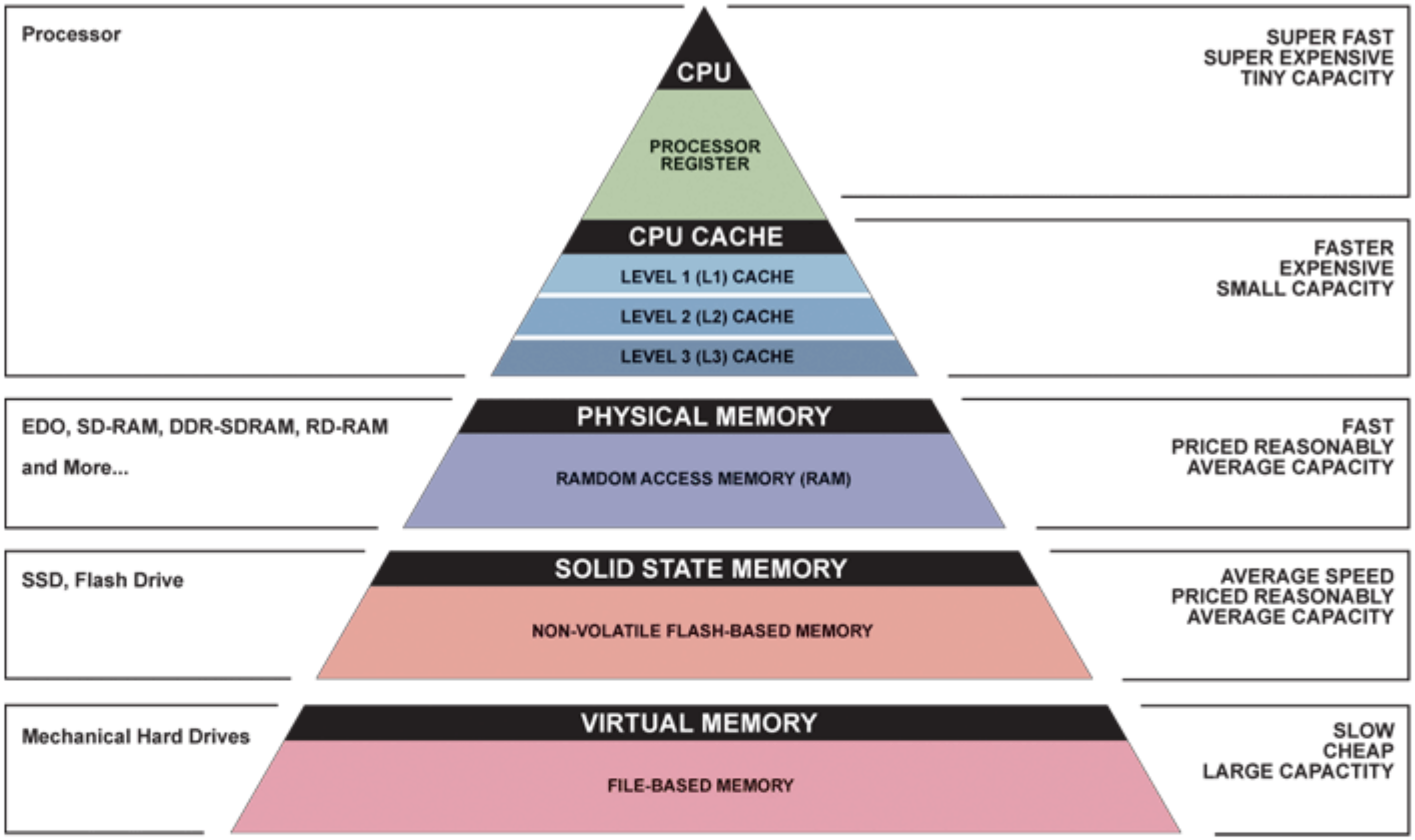
- *check whether P1's birthdate is in range D1..D2*
- *check in interests whether this person likes A1, if so*
- *visit all friends P2 of P1, for each:*
    - *Check in the person data that P2 lives in the same places as P1*
    - *Compute in interests the score for P2 (likes A2,A3,A4?)*
    - *If the P2.score >= 2, visit all friends P3 of P2, for each:*
        - *Check in the person data that P3 lives in the same places as P1*
        - *Compute in interests the score for P3 (likes A2,A3,A4?)*
        - *If the P3.score >= 2, see if P1 is among the friends of P3, if so*
            - *We have a result (P2.score+P3.score,P1,P2,P3)*

Naïve Query Implementation

- “cruncher”

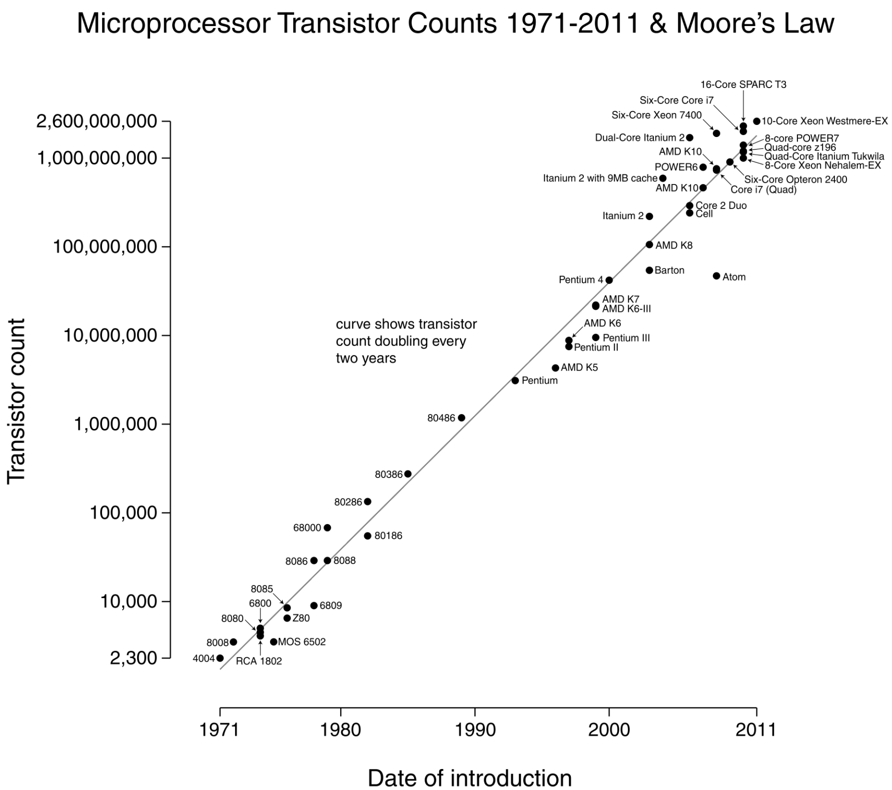


Memory Hierarchy

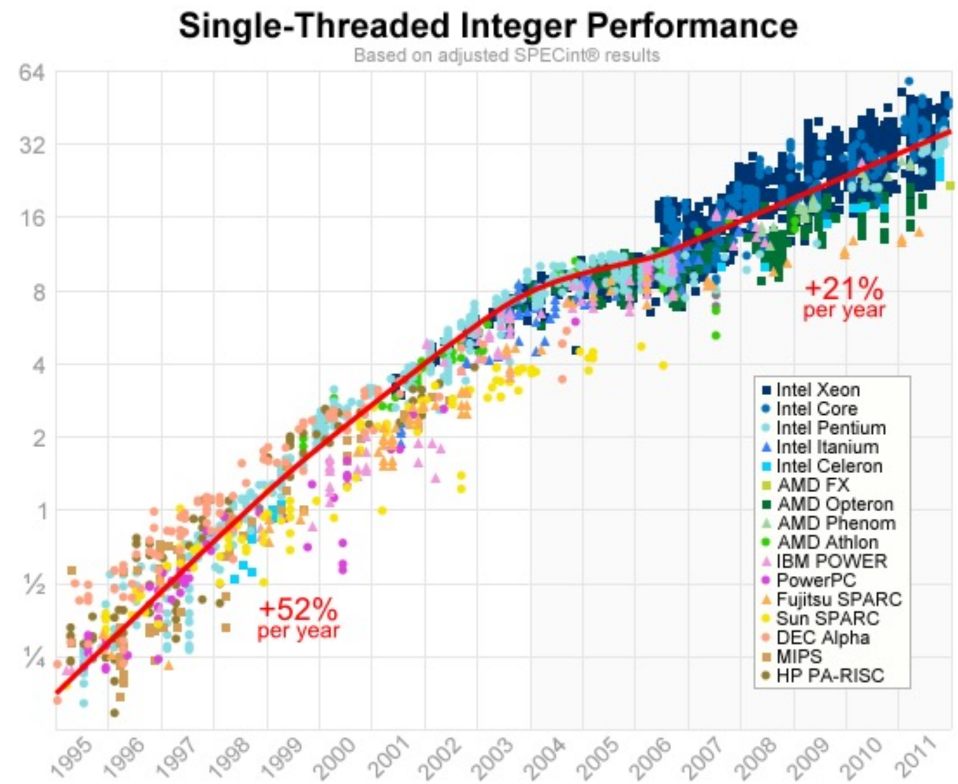


▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Hardware Progress

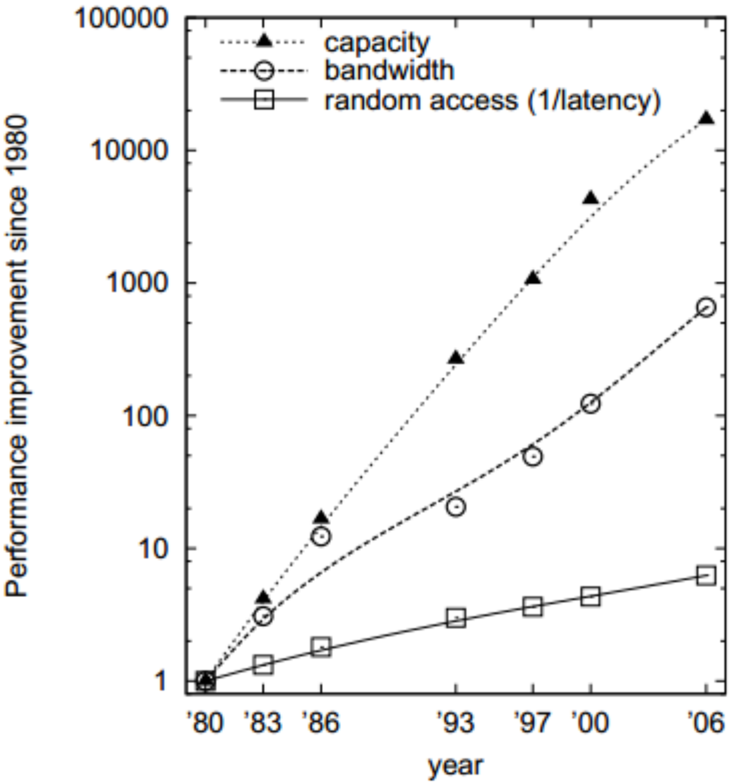


Transistors

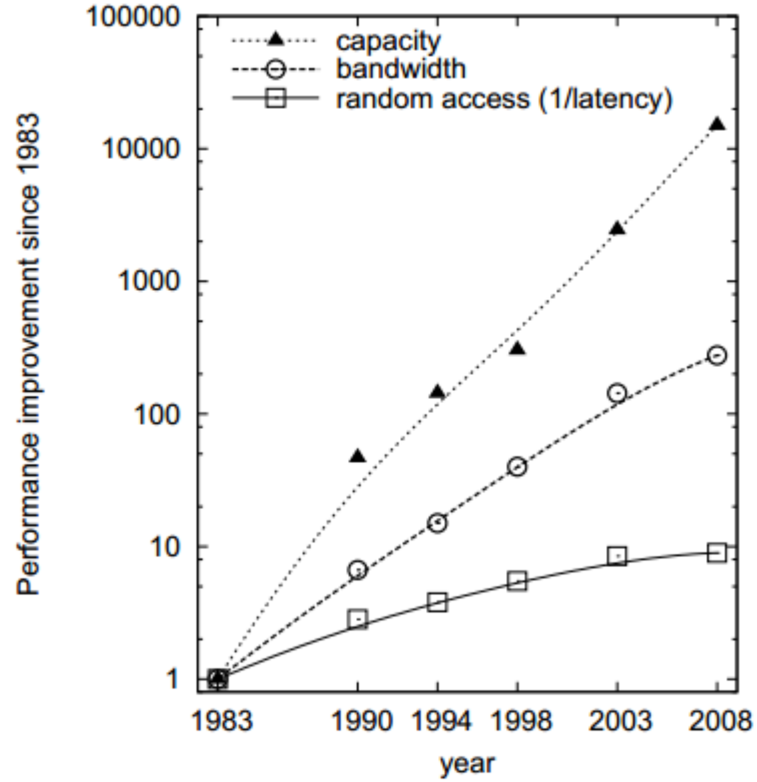


CPU performance

RAM,Disk Improvement Over the Years



RAM



Magnetic Disk

Latency Lags Bandwidth

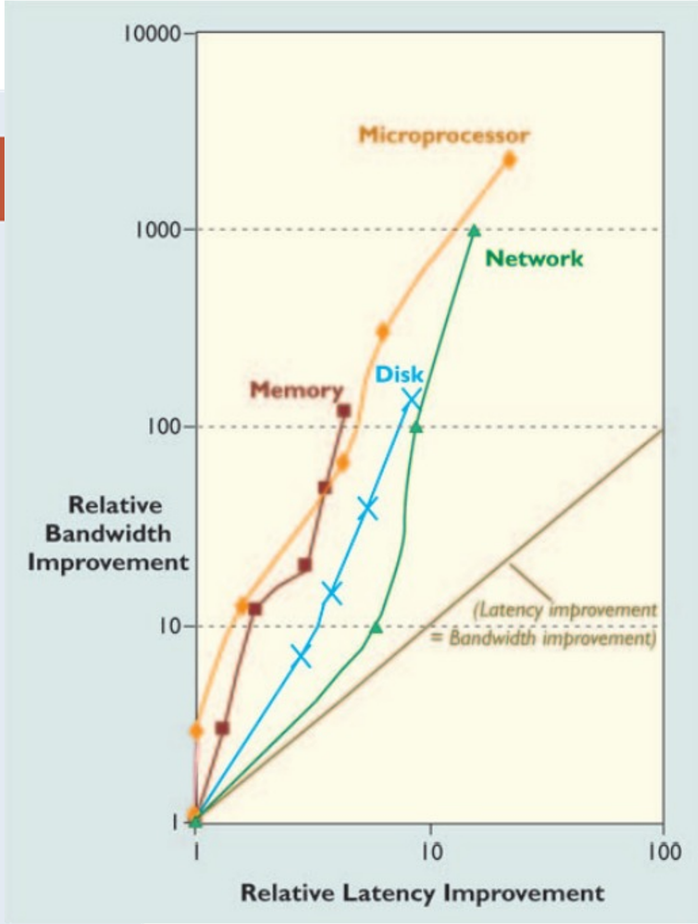
- Communications of the ACM, 2004

By David A. Patterson

LATENCY LAGS BANDWITH

Recognizing the chronic imbalance between bandwidth and latency, and how to cope with it.

As I review performance trends, I am struck by a consistent theme across many technologies: bandwidth improves much

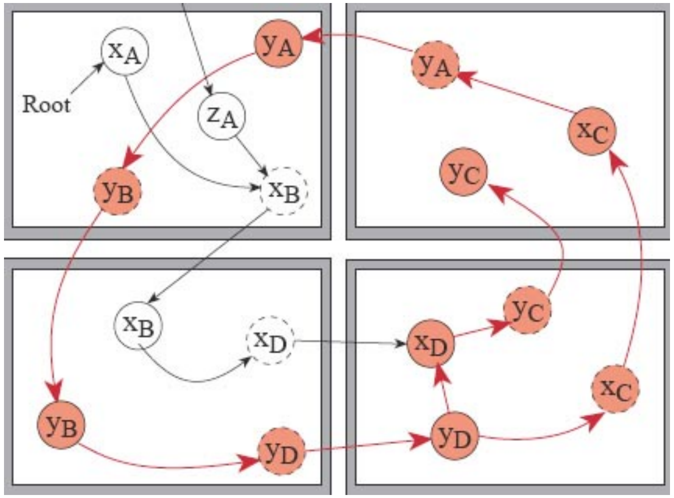


Sequential Access Hides Latency

- Sequential RAM access
  - CPU prefetching: multiple consecutive cache lines being requested concurrently
- Sequential Magnetic Disk Access
  - Disk head moved once
  - Data is streamed as the disk spins under the head
- Sequential Network Access
  - Full network packets
  - Multiple packets in transit concurrently

Consequences For Algorithms

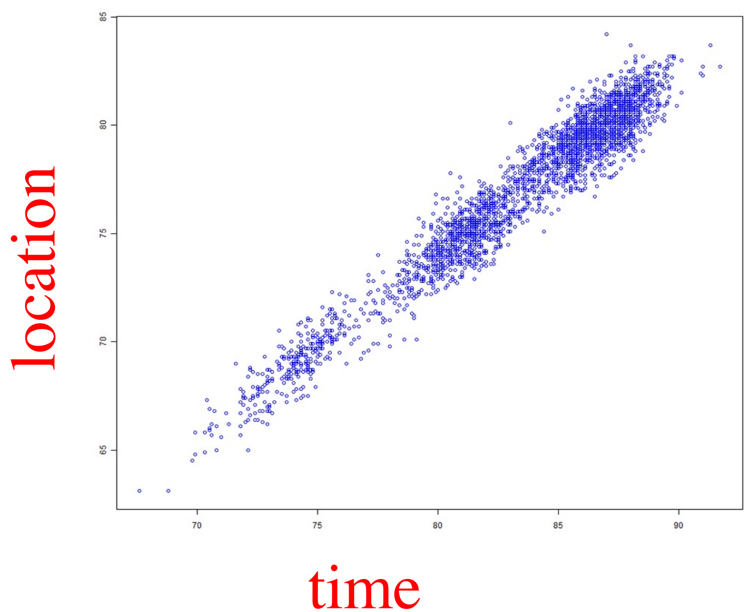
- Analyze the main data structures
  - How big are they?
    - Are they bigger than RAM?
    - Are they bigger than CPU cache (a few MB)?
  - How are they laid out in memory or on disk?
    - One area, multiple areas?

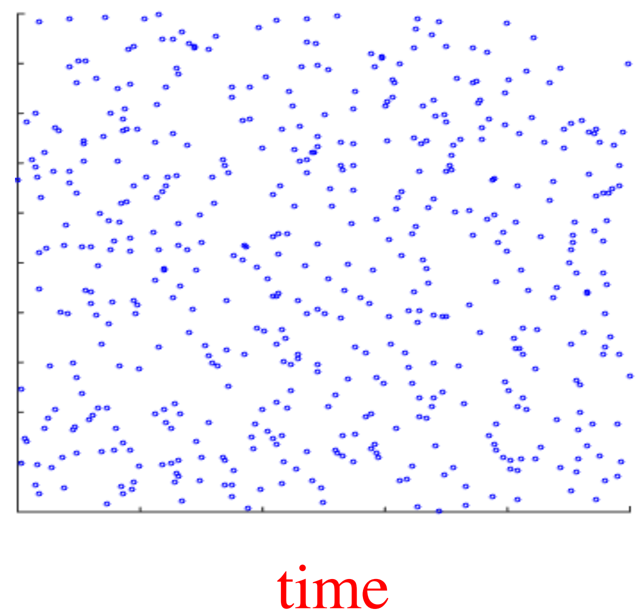


Java Object Data Structure
vs
memory pages (or cache lines)

Consequences For Algorithms

- Analyze your access patterns
  - Sequential: you’re OK
  - Random: it better fit in cache!
    - What is the access granularity?
    - Is there temporal locality?
    - Is there spatial locality?





Improving Data Access Patterns

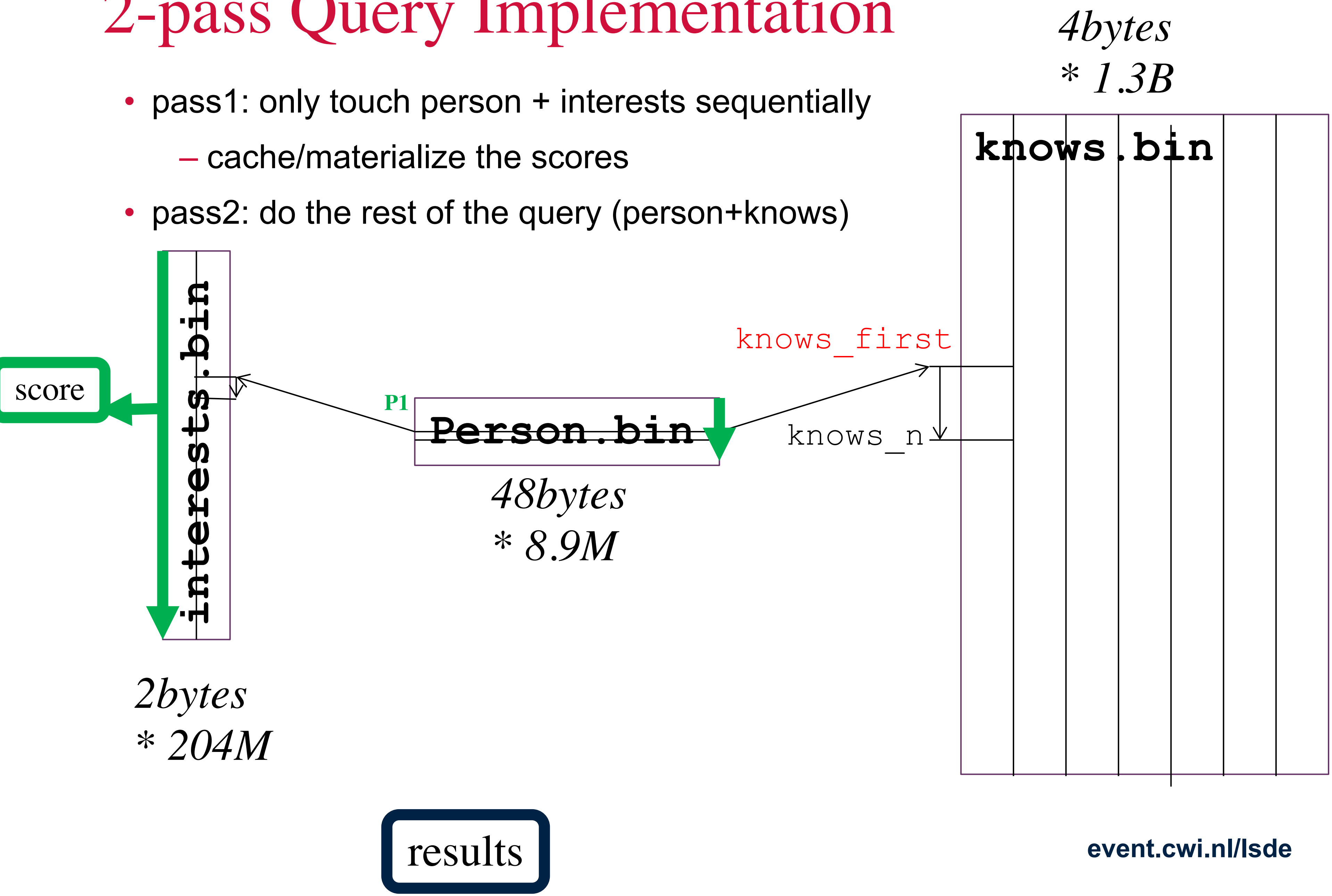
- **Make the data smaller**
  - Remove unused data from the structure
  - Apply data compression (of some kind)
    - If random access is needed, gzip does not work
    - zero surpression ➔ use the smallest datatype possible
- **Do Not Access All Data**
  - Apply filters as soon as possible
  - Cluster or Partition the data
    - Only access data in particular clusters/partitions
  - Build an index
    - Avoid full access to the main table by identifying useful regions using an index
- **Trade Random Access For Sequential Access**
  - Make more passes over the data. Separate access to different regions into different phases.
- **Try Denormalizing the Schema**
  - Remove joins/lookups, add looked up stuff to the table
    - Does not help if the join explodes the size (this is the case with friends!)

Improving Data Access Patterns

- **Make the data smaller**
  - Remove unused data from the structure
  - Apply data compression (of some kind)
    - If random access is needed, gzip does not work
    - zero surpression ➔ use the smallest datatype possible
- **Do Not Access All Data**
  - Apply filters as soon as possible
  - Cluster or Partition the data
    - Only access data in particular clusters/partitions
  - Build an index
    - Avoid full access to the main table by identifying useful regions using an index
- **Trade Random Access For Sequential Access**
  - Make more passes over the data. Separate access to different regions into different phases.
- **Try Denormalizing the Schema**
  - Remove joins/lookups, add looked up stuff to the table
    - Does not help if the join explodes the size (this is the case with friends!)

2-pass Query Implementation

- pass1: only touch person + interests sequentially
  - cache/materialize the scores
- pass2: do the rest of the query (person+knows)



2-pass Query Implementation

- pass1: only touch person + interests sequentially
  - cache/materialize the scores
- pass2: do the rest of the query (person+knows)