*Urban Sensing

Designing the Big Data architecture

Peter Bednár

*Urban Sensing project

- FP7 research project
- SMEs partners
  - urban architects, design and marketing company, data science and technology company
- R&D partners
  - TUK + 2 research and development SMEs
- http://urban-sensing.eu/

*Urban Sensing project

1. Integrate content from social media sites and open data sources
2. Extract entities and perform sentiment analysis on textual content
3. Aggregate data according to various dimensions
4. Interactively visualize aggregated data in „real-time“

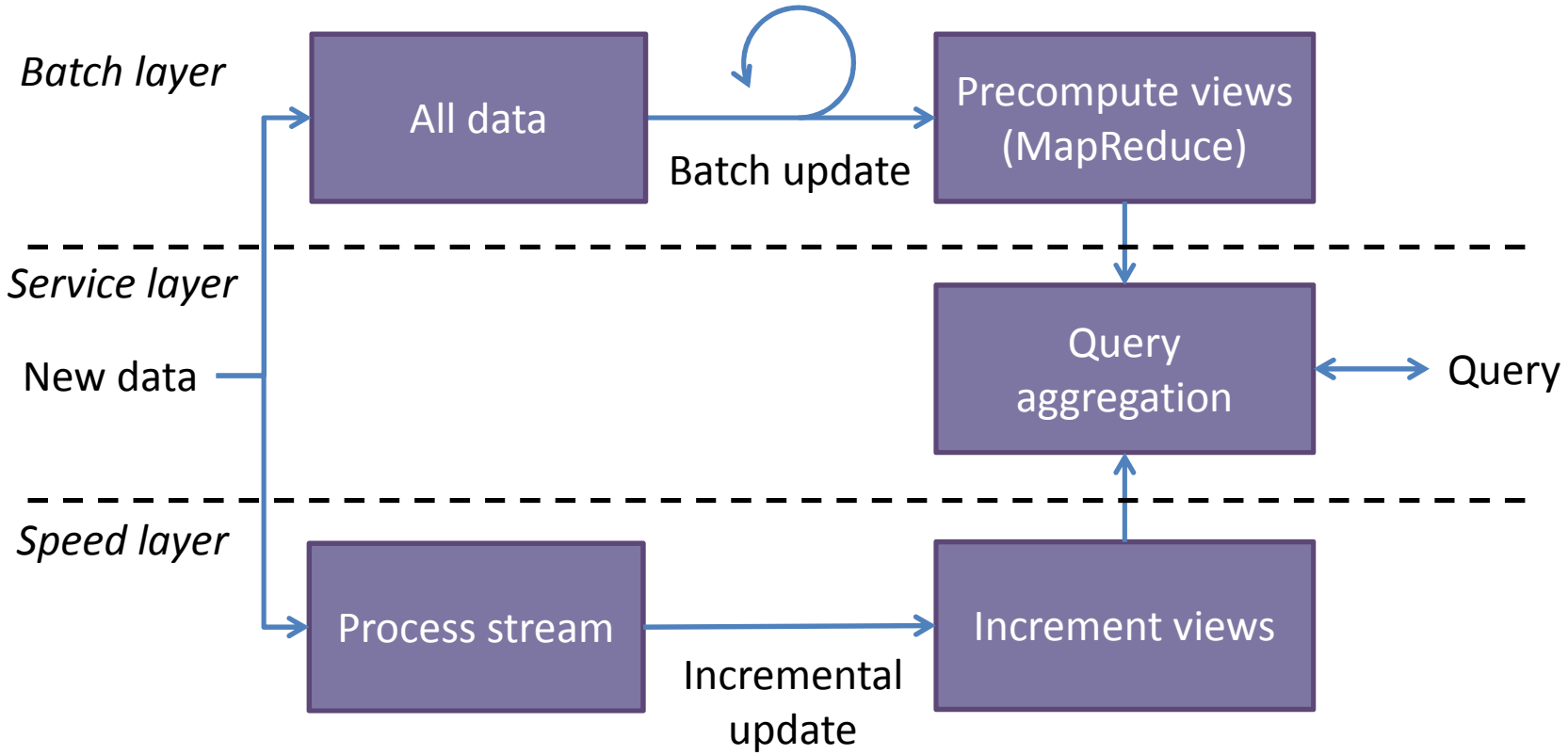
- scalable up to 10000 contributions per s
- volume 1T/year/area of interest

Technological context

- Distributed databases
  - SQL/NoSQL
- Batch processing
  - Apache Hadoop ecosystem (HDFS, MapReduce, Pig, Hive)
- Stream processing
  - Apache Storm

- but before…
  - Micro-batch processing (i.e. Apache Spark)
  - λ-architecture

λ-architecture



Design steps

so how we have designed *US architecture?

1. User requirements
2. Definition of basic building blocks
3. Specification of functional layers
4. Composition
5. Mapping to implementation

User requirements

- Indicator scenarios - "business" questions aka:
  - *Which are the places from where people contribute the most?"*
  - *Where do people name brands and products?*
  - *Is there a relation between the place where people experience or name a brand and where the brand is advertised/present somehow?*
  - *From where people talk about the event the most?*
  - *Where are specific ethnic groups contributing the most?*
- Visualization strategies
  - visualization atlas
  - UI Mock-ups

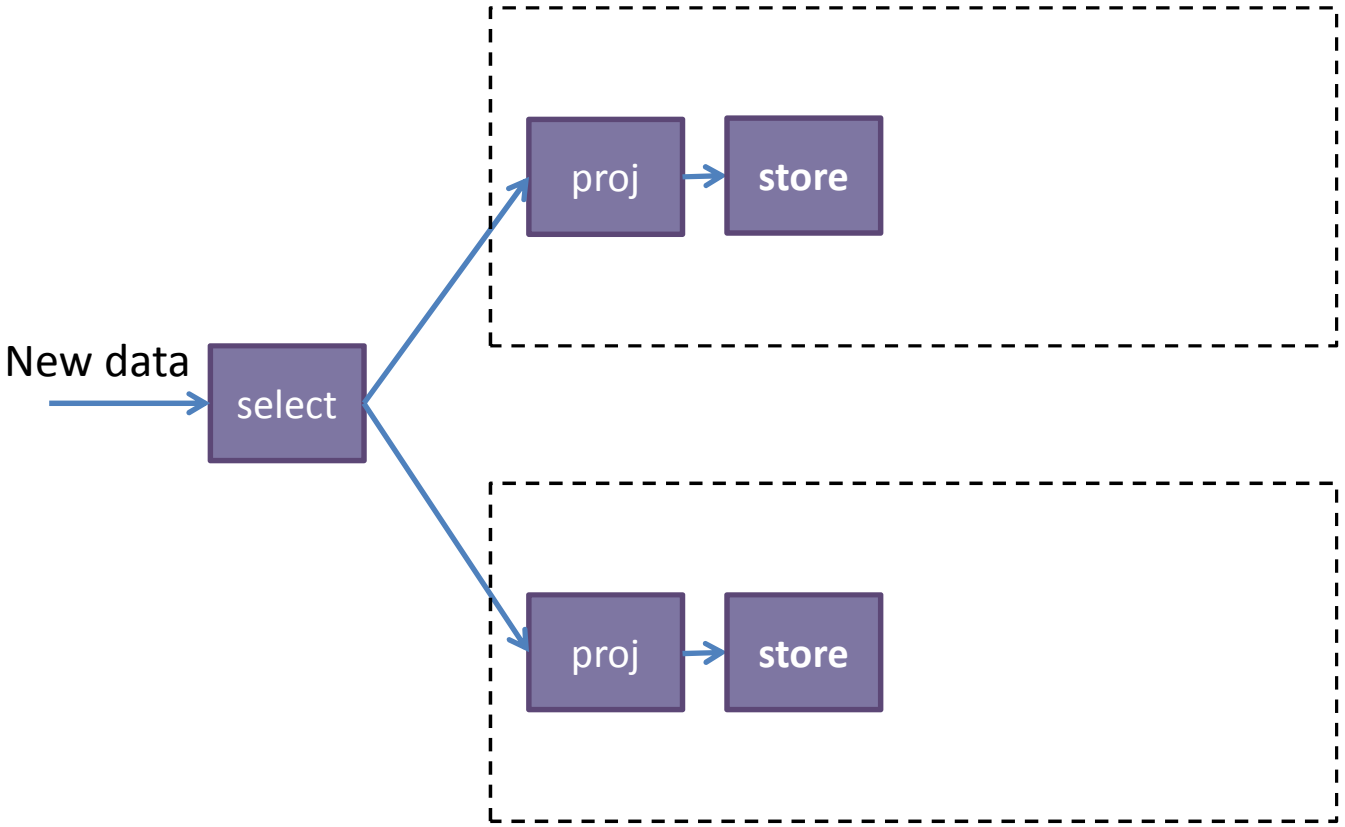
Basic building blocks

- composition of “atomic” functions
- **selection**
  - true/false = select(O)
- **projection**
  - O = proj(O)
  - O[] = proj(O)
- **aggregation**
  - A = agg(O, A) – increment
  - A = agg(A, A)
  - e.g. sort – special type of aggregation S[] = sort(S_1[], S_2[])

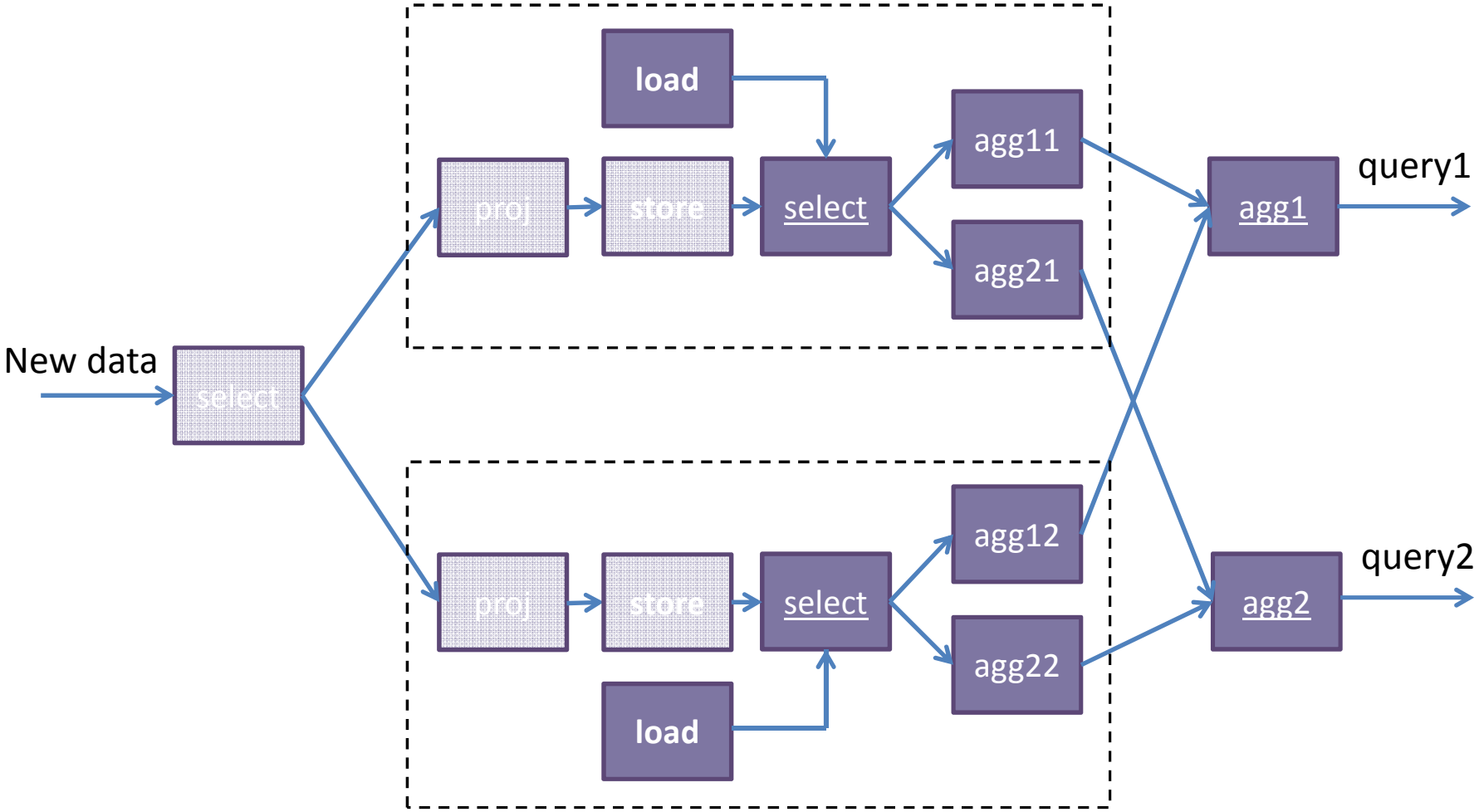
State management

- **State-less functions**
  - select(O,θ)
  - proj(O,θ)
  - θ – “hidden” parameters independent on updates
- **State-full functions**
  - A = agg(O, A) – A is managed state
  - A = load(K), {A, K} = store(A) – in-memory storage
  - A = **load**(K), {A, K} = **store**(A) – “deep” persistent storage

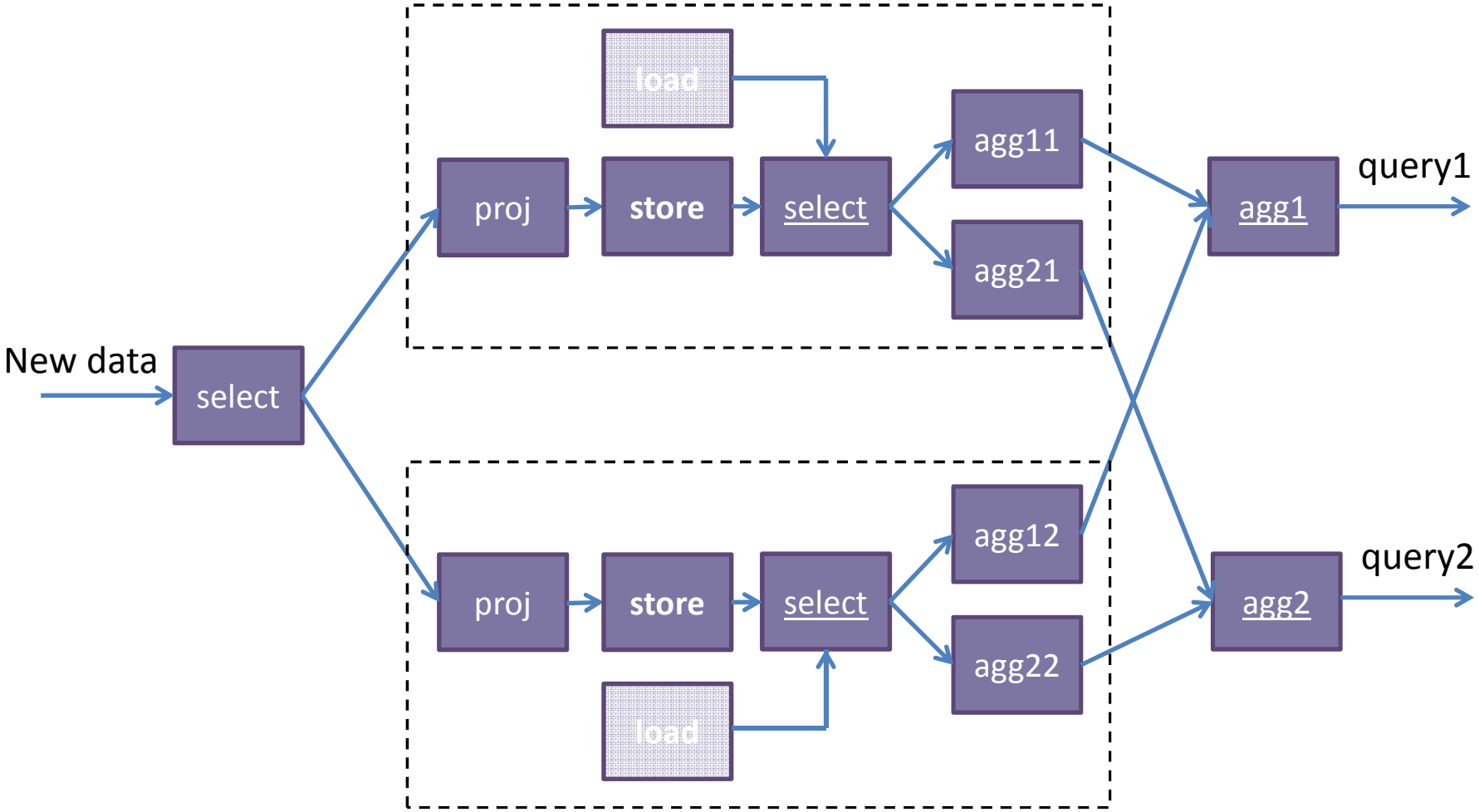
Composition



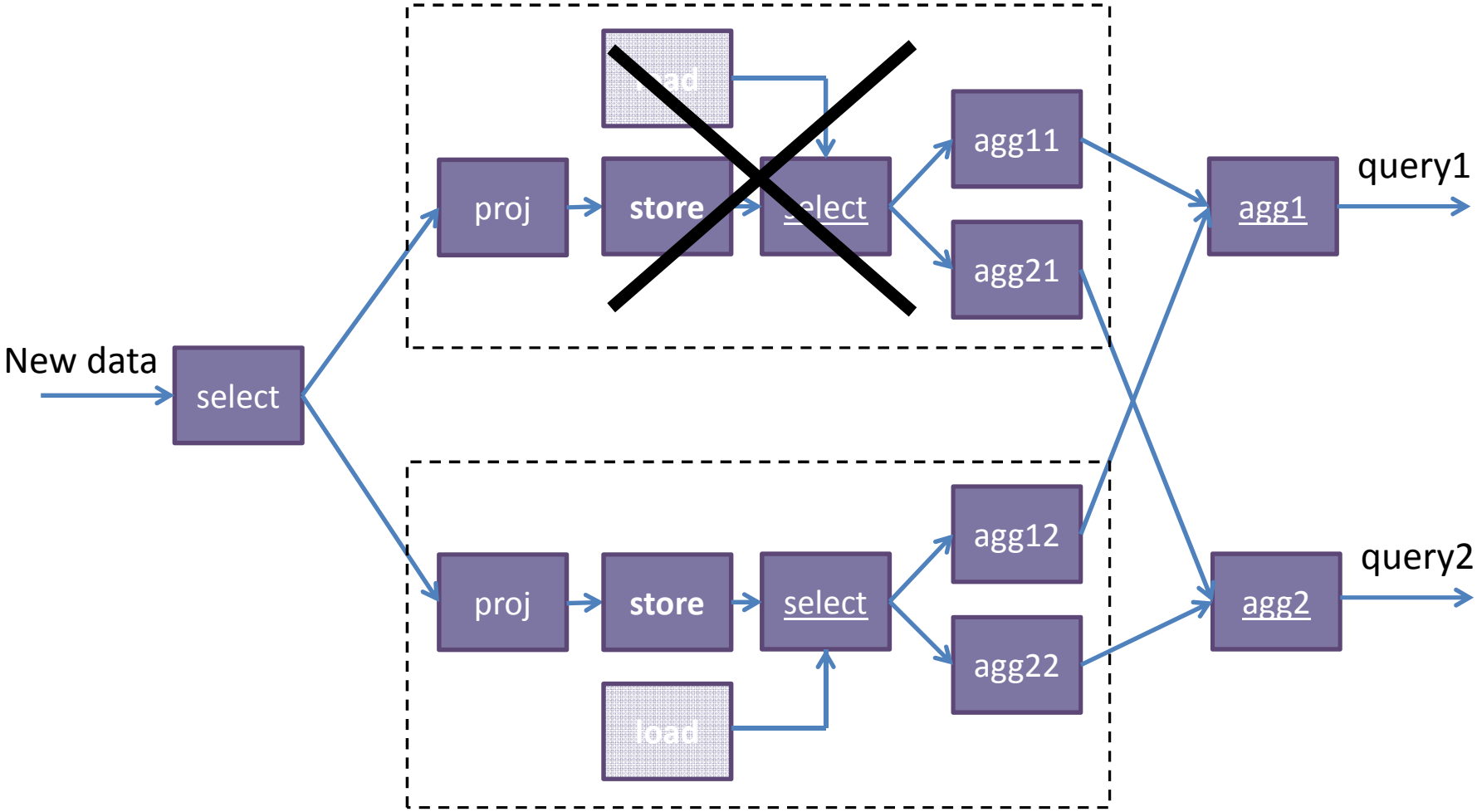
Query = select + aggregate



Query = select + aggregate



I will not talk about the consistency...



Architecture functional layers

- **Data integration**
  - connectors and mediators
- **Information extraction**
  - NER + sentiment analysis + emotion detection + sarcasm detection
  - machine learning + rule-based methods
- **Aggregation and visualization**
  - ad-hoc queries
  - geographical maps, time series

Data integration

- **Connectors**
  - handling various communication protocols and get “raw” data
  - pull/push methods
  - Twitter, Facebook, Foursqure, Flickr, Instagram, Panoramio
  - RSS, XML, OGC for geo-located data
- **Mediators**
  - convert “raw” data to Common Data Model
  - extract text content + context + metadata
  - proj() (but not for complex context)

Common Data Model

- timestamp
- location
- text title and content
- context – {timestamp, text, context}
- languages
- tags
- contributor – {id, gender, birth date, locales, home address}
- resources

Information extraction

- **Language identification**
- **Extract named entities**
  - Places
  - People
  - Events
  - Products
- Rule-based methods
  - Tokenization
  - Gazetteer lookup
  - POS
  - Regular expression over annotations

Information extraction

- **Sentiment analyses and emotion detection**
  - overall sentiment + sentiment about the subject (entity)
  - anger, joy, disgust, fear, sadness
- Rule-based methods + Machine learning methods (SVM)

- proj() chaining – NO online methods

Precomputed aggregations

- **geo-area classifications**
  - e.g. to state boundaries, cities, city blocks, etc.
- **people paths**
  - in-memory store for previous path with time eviction (time window)
  - for 1h up to 5 nodes
  - pre-computed “bearings” – incoming/outgoing angles

Implementation mapping

- How to leverage existing distributed databases for efficient „re-playing“ of historical data?
  - highly optimized by indexing/caching
  - but less control – mixed selection, projection, aggregation
  - aggregation API in MongoDB and Elasticsearch

- Separate layers with message queues
- Don‘t forget about monitoring
- Quite difficult to track consistency

other lessons learned from Big Data project

- be prepared for new technologies with lower readiness level – you are on cutting edge
- allocate resources for Ops – maintenance is crucial already for developing

Conclusions

- Decompose algorithms to functional block
- Be state-less… if possible
- Distribute state-full functions
- Reflect about the consistency

and don’t panic during the deployment and testing

"Your recent Amazon purchases, Tweet score and location history makes you 23.5% welcome here."