MUNI
ÚVT

PV177 – DataScience seminář (ELK stack and Graph DBs)

Tomáš Rebok
Ústav výpočetní techniky MU

MUNI
ÚVT

Data analysis with ELK framework

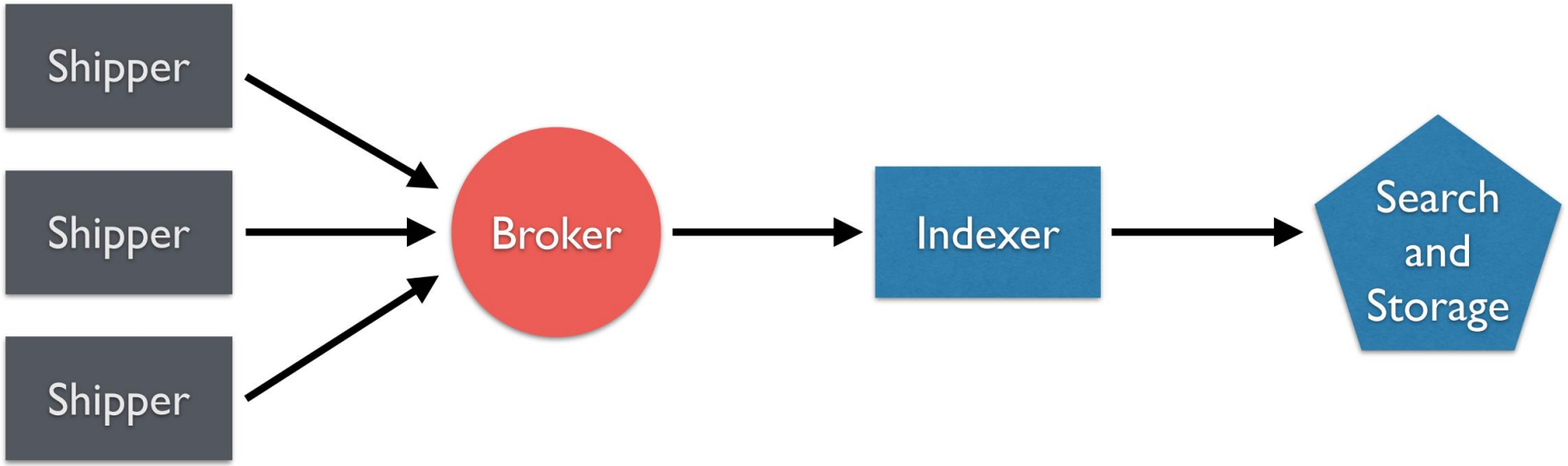
Elasticsearch ELK Software Stack

- **ELK consists of three open source software products provided by the company “Elastic” (formerly Elasticsearch)**
  - **E** => **E**lasticsearch
    (Highly scalable search index server)
  - **L** => **L**ogstash
    (Tool for the collection, enrichment, filtering and forwarding of data, e.g. log data)
  - **K** => **K**ibana
    (Tool for the exploration and visualization of data)

Log Logstash Kibana

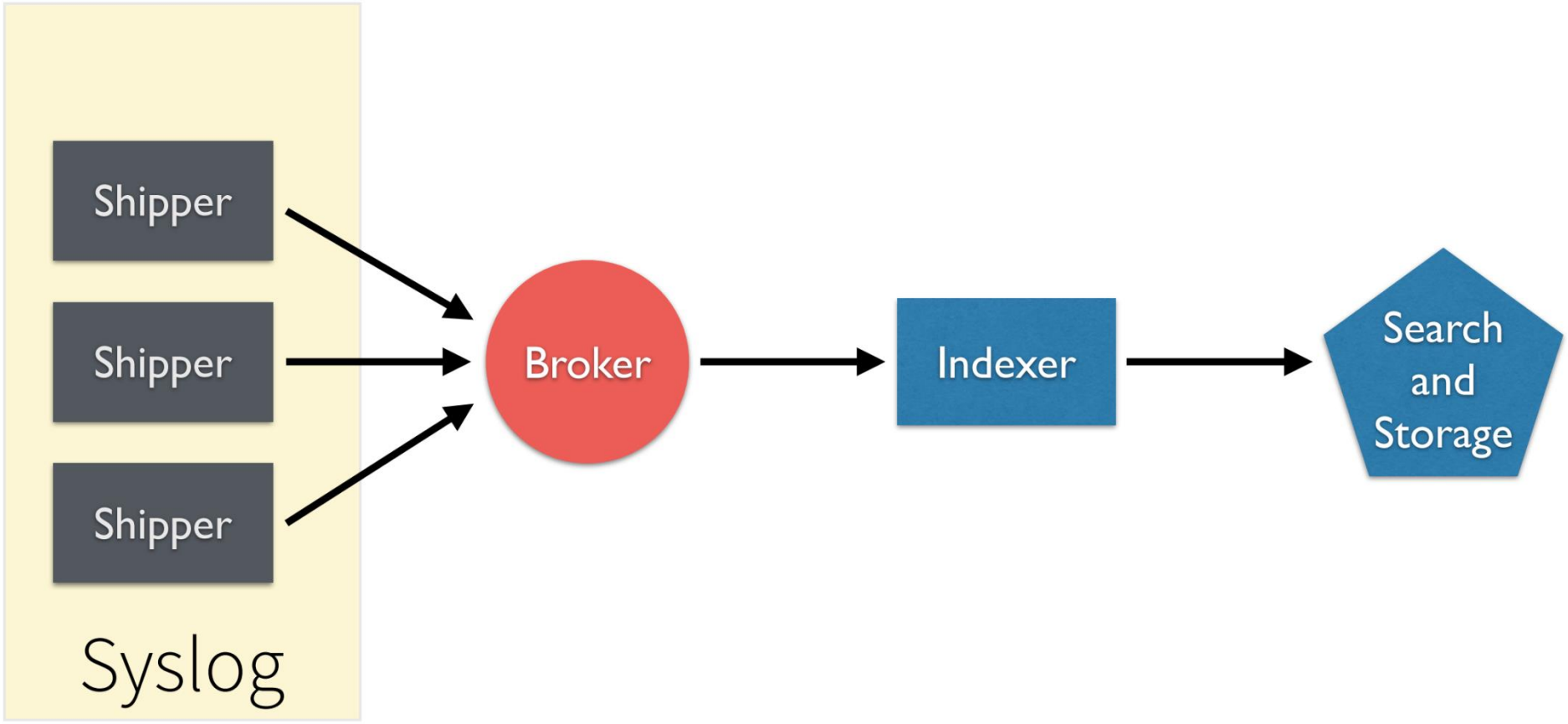
MUNI ÚVT

ELK Architecture



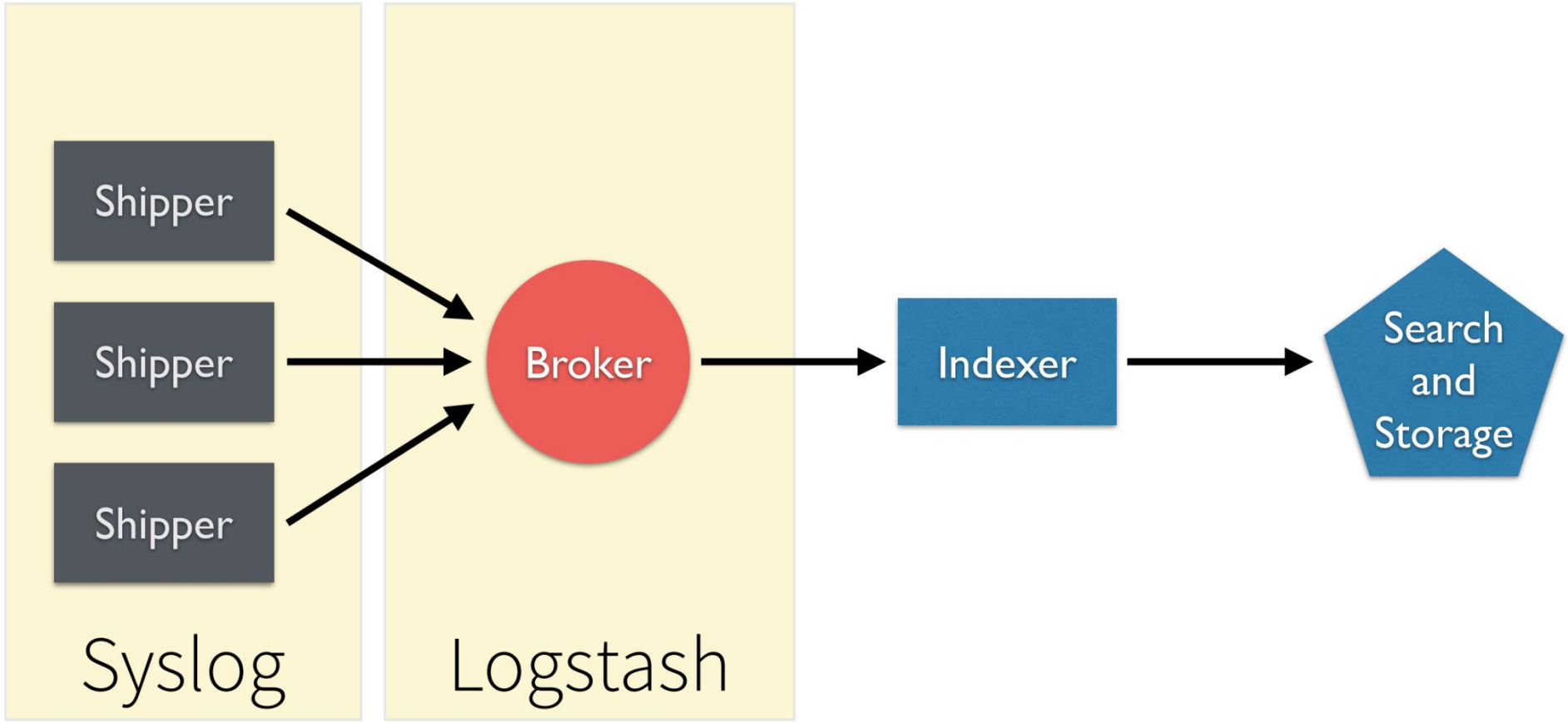
MUNI ÚVT

ELK Architecture



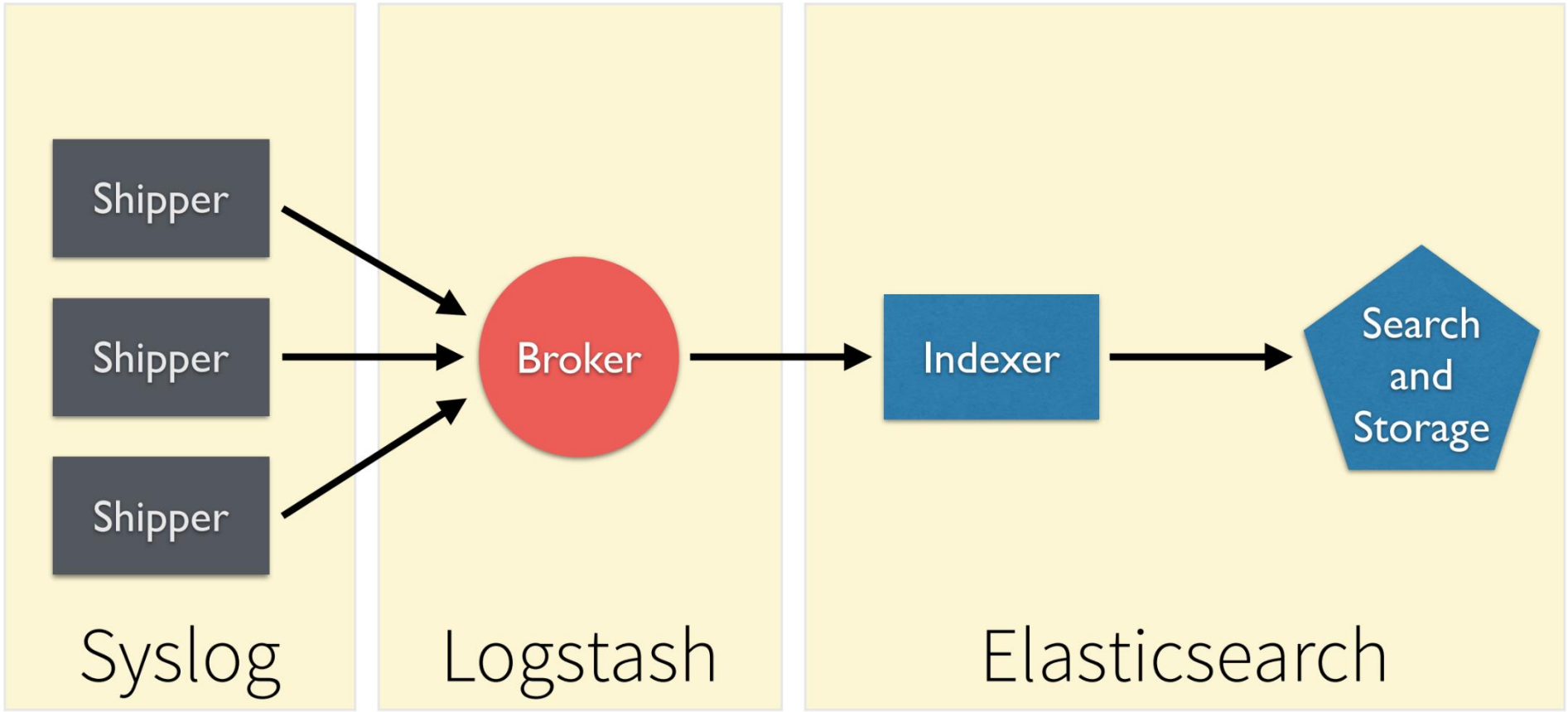
MUNI ÚVT

ELK Architecture



MUNI ÚVT

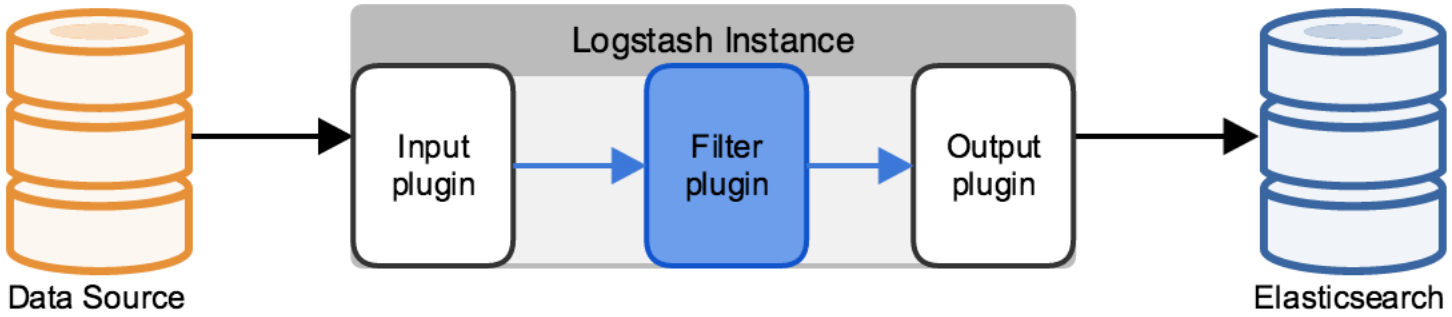
ELK Architecture



MUNI ÚVT

Logstash

- Open source software to collect, transform, filter and forward data (e.g. log data) from input sources to output sources (e.g. Elasticsearch)
- Implemented in JRuby and runs on a JVM (Java Virtual Machine)
- Simple message-based architecture
- Extendable by plugins (e.g. input, output, filter plugins)



MUNI ÚVT

Configuration

Multiple inputs of different types

Conditionally filter and transform data; some common formats are already known

Forward to multiple outputs

```
input {
  file {
    path => "/tmp/access_log"
    start_position => "beginning"
  }
}

filter {
  if [path] =~ "access" {
    mutate { replace => { "type" => "apache_access" } }
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
  }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

output {
  elasticsearch {
    host => localhost
  }
  stdout { codec => rubydebug }
}
```

Console output processing Apache log files

```
{
        "message" => "127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] \"GET
        /xampp/status.php HTTP/1.1\" 200 3891 \"http://cadenza/xampp/navi.php\"
        \"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101
        Firefox/25.0\"",
     "@timestamp" => "2013-12-11T08:01:45.000Z",
       "@version" => "1",
           "host" => "cadenza",
       "clientip" => "127.0.0.1",
          "ident" => "-",
           "auth" => "-",
      "timestamp" => "11/Dec/2013:00:01:45 -0800",
           "verb" => "GET",
        "request" => "/xampp/status.php",
    "httpversion" => "1.1",
       "response" => "200",
          "bytes" => "3891",
       "referrer" => "\"http://cadenza/xampp/navi.php\"",
          "agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0)
          Gecko/20100101 Firefox/25.0\""
}
```

```
input {
  tcp {
    port => 5000
    type => syslog
  }
  udp {
    port => 5000
    type => syslog
  }
}

filter {
  if [type] == "syslog" {
    grok {
      match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_h
_program}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
      add_field => [ "received_at", "%{@timestamp}" ]
      add_field => [ "received_from", "%{host}" ]
    }
    syslog_pri { }
    date {
      match => [ "syslog_timestamp", "MMM  d HH:mm:ss", "MMM dd HH:mm:ss" ]
    }
  }
}

output {
  elasticsearch { host => localhost }
  stdout { codec => rubydebug }
}
```

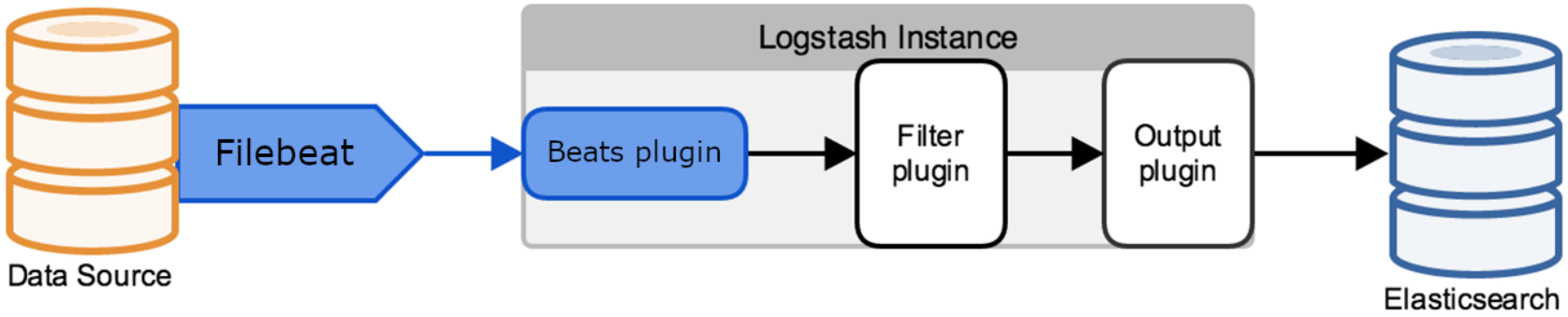
Console output processing Syslog messages

```
{
                 "message" => "Dec 23 14:30:01 louis CRON[619]: (www-data) CMD (php
                 /usr/share/cacti/site/poller.php >/dev/null
                 2>/var/log/cacti/poller-error.log)",
               "@timestamp" => "2013-12-23T22:30:01.000Z",
                 "@version" => "1",
                     "type" => "syslog",
                     "host" => "0:0:0:0:0:0:0:1:52617",
          "syslog_timestamp" => "Dec 23 14:30:01",
           "syslog_hostname" => "louis",
            "syslog_program" => "CRON",
                "syslog_pid" => "619",
            "syslog_message" => "(www-data) CMD (php /usr/share/cacti/site/poller.php
            >/dev/null 2>/var/log/cacti/poller-error.log)",
               "received_at" => "2013-12-23 22:49:22 UTC",
             "received_from" => "0:0:0:0:0:0:0:1:52617",
      "syslog_severity_code" => 5,
      "syslog_facility_code" => 1,
           "syslog_facility" => "user-level",
           "syslog_severity" => "notice"
}
```

Input Plugins

- **file -> for processing files**
- **tcp, udp, unix -> reading directly from network sockets**
- **http -> for processing HTTP POST requests**
- **http_poller -> for polling HTTP services as input sources**
- **imap -> accessing and processing imap mail**
- **Different input plugins to access MOM („*Message-Oriented Middleware*“, message queues)**
  - Rabbitmq, stomp, …
- **Different plugins for accessing database systems**
  - jdbc, elasticsearch, …
- **Plugins to read data from system log services and from command line**
  - syslog, eventlog, pipe, exec
- **and more …**

Elastic Beats framework + Beats plugin



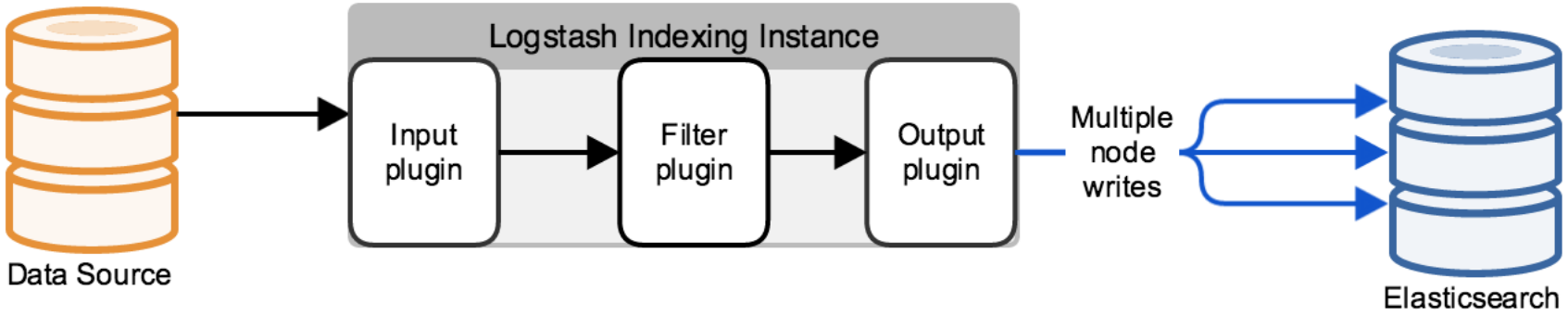
- The "**Elastic Beats**" framework allows to forward input from a set of "data sources" to a Logstash instance for processing
  - Filebeat, Packetbeat, Winlogbeat, Metricbeat, Functionbeat, etc.
- The "**Beats plugin**" can then be configured to consume messages from "Elastic Beats"
- Transfer can be secured by security certificate and encrypted transmission
  - authentication and confidentiality

MUNI ÚVT

Output plugins

- **stdout, pipe, exec -> show output on console, feed to a command**
- **file -> store output in file**
- **email -> send output as email**
- **tcp, udp, websocket -> send output over network connections**
- **http -> send output as HTTP request**
- **Different plugins for sending output to database systems, index server or cloud storage**
  - elasticsearch, solr_http, mongodb, google_bigquery, google_cloud_storage, opentsdb
- **Different output plugins to send output to MOM (message queues)**
  - Rabbitmq, stomp, …
- **Different output plugins for forwarding messages to metrics applications**
  - graphite, graphtastic, ganglic, metriccatcher

Multiple node writes



- The Logstash output plugin can write to multiple Elasticsearch nodes
- It will distribute output objects to different nodes ("*load balancing*")
- A Logstash instance can also be part of a Elasticsearch cluster and write data through the cluster protocol

MUNI ÚVT

Filter plugins

- **grok -> parse and structure arbitrary text: best generic option to interpret text as (semi-)structured objects**
  - alternative: **dissect** (faster, but does not use regular expressions)
- **filter for parsing different data formats**
  - csv, json, kv (key-valued paired messages), xml, …
- **multiline -> collapse multiline messages to one logstash event**
- **split -> split multiline messages into several logstash events**
- **aggregate -> aggregate several separate message lines into one Logstash event**
- **mutate -> perform mutations of fields (rename, remove, replace, modify)**
- **dns -> lookup DNS entry for IP address**
- **geoip -> find geolocation of IP address**
- **and more**

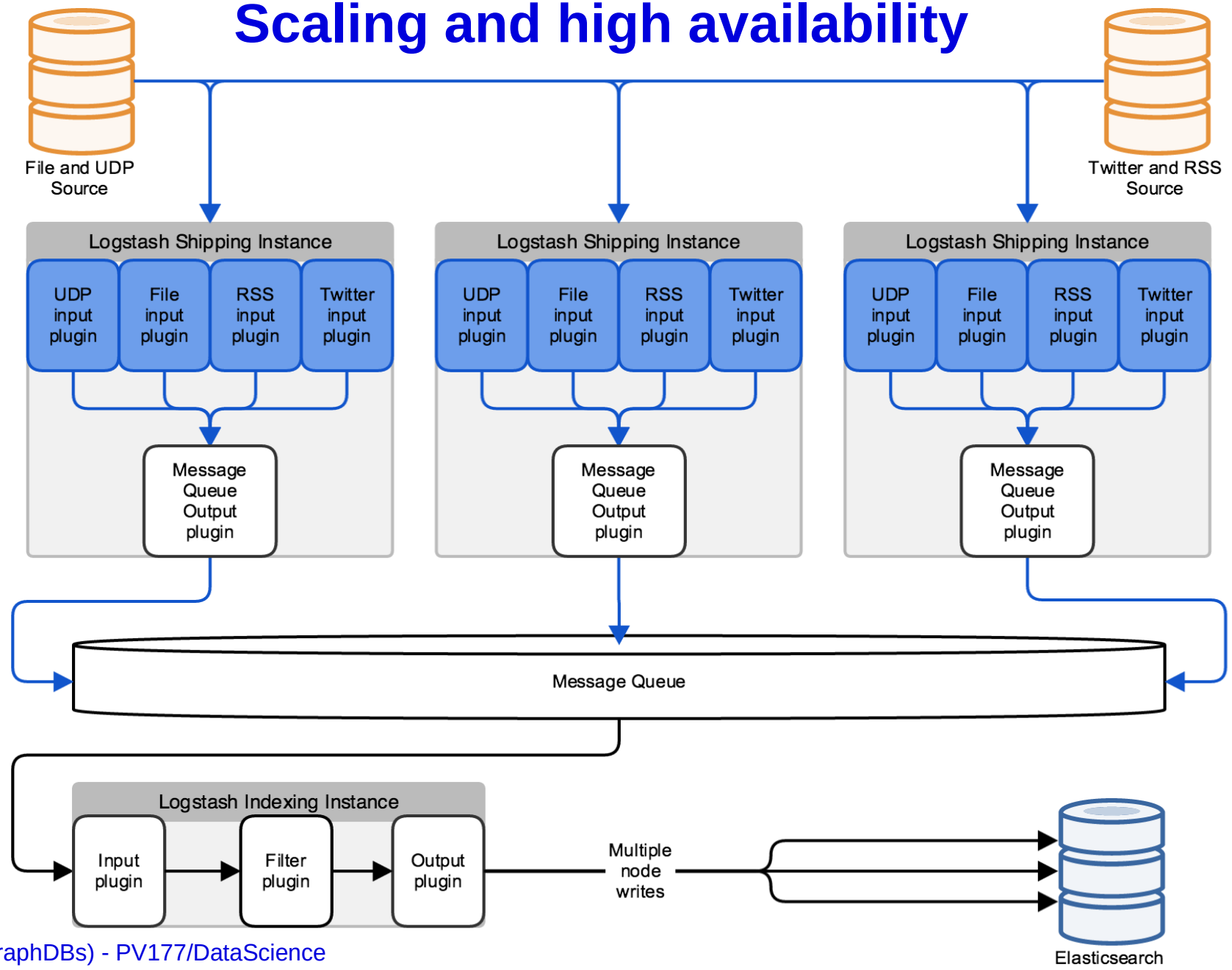
grok usage example

- **Input: 55.3.244.1 GET /index.html 15824 0.043**
- **grok filter**

```
filter {
    grok { match => { "message" => "%{IP:client} %{WORD:method}
%{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration}" }
}
```

- **Then the output will contain fields like:**
  - client: 55.3.244.1
  - method: GET
  - request: /index.html
  - bytes: 15824
  - duration: 0.043

Scaling and high availability



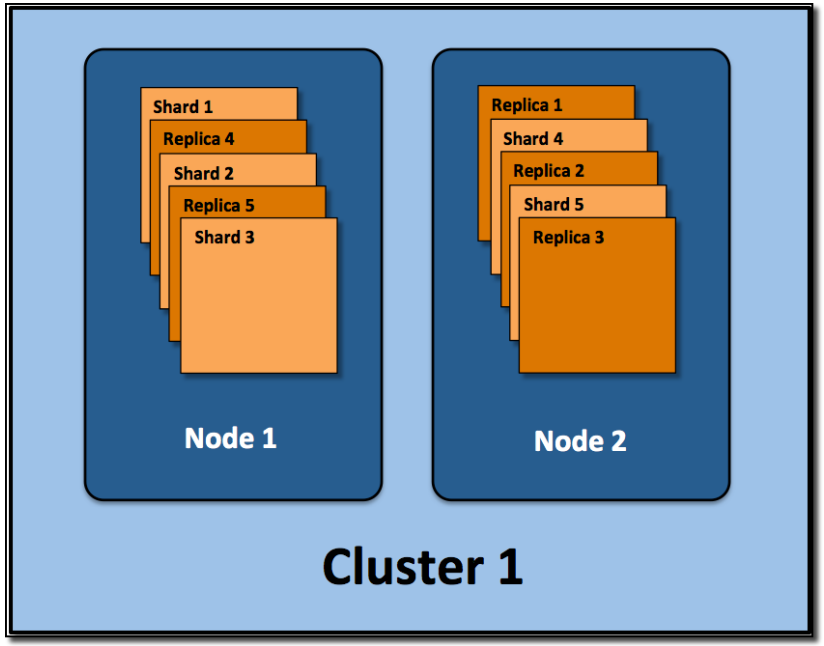
MUNI ÚVT

ElasticSearch

- **Server environment for storing large-scale structured index entries and query them**
  - Written in Java
  - Based on Apache Lucene
    - Uses Lucene for index creation and management
  - Document-oriented (structured) index entries which can (but must not) be associated with a schema
  - Combines “full text”-oriented search options for text fields with more precise search options for other types of fields, like date + time fields, geolocation fields, etc.
  - Near real-time search and analysis capabilities
- **Provides Restful API as JSON over HTTP**

Scalability of Elasticsearch

- **Elasticsearch can run as one integrated application on multiple nodes of a cluster**
- **Indexes are stored in Lucene instances called “Shards” which can be distributed over several nodes**
  - Ability to subdivide your (large) index into multiple pieces
  - Each shard is in itself a fully-functional and independent "index" that can be hosted on any node in the cluster
- **There a two types of “Shards”**
  - Primary Shards
  - Replica
- **Replicas of “Primary Shards” provide**
  - Failure tolerance and therefore protect data
  - Make queries (searches) faster



MUNI ÚVT

Indexing data with Elasticsearch

- **Send JSON documents to server, e.g. use REST API**
  - No schema necessary => ElasticSearch determines types of attributes
  - But it‘s possible to explicitly specify schema, i.e. types for attributes
    - Like string, byte, short, integer, long, float, double, boolean, date
- **Analysis of text attributes for fulltext-oriented search**
  - Word extraction, reduction of words to their base form (stemming)
  - Stop words
  - Support for multiple languages (including czech, but not slovak yet)
- **Automatically generates identifiers for data sets or allows to specify them while indexing**

Indexing data using the REST API

```
PUT /megacorp/employee/1
{
    "first_name" : "John",
    "last_name" :  "Smith",
    "age" :        25,
    "about" :      "I love to go rock climbing",
    "interests": [ "sports", "music" ]
}
```

- PUT request inserts the JSON payload into the index with name “megacorp” as object of type “employee”
- Schema for type can be explicitly defined (at time of index creation or automatically determined)
- Text field (e.g. “about”) will be analyzed if analyzers are configured for that field
- Request URL specifies the identifier “1” for the index entry

Retrieval of an index entry

GET /megacorp/employee/1

```
{
  "_index" :   "megacorp",
  "_type" :    "employee",
  "_id" :      "1",
  "_version" : 1,
  "found" :    true,
  "_source" :  {
      "first_name" :  "John",
      "last_name" :   "Smith",
      "age" :         25,
      "about" :       "I love to go rock climbing",
      "interests":  [ "sports", "music" ]
  }
}
```

- **A “GET” REST API call with “/megacorp/employee/1” will retrieve the entry with id 1 as JSON object**

GET /megacorp/employee/_search

Simple Query

- GET request with “_search” at the end of the URL performs query
- Search results are returned in JSON response as “hits” array
- Further metadata specifies count of search results (“total”) and max_score


```
{
   "took":      6,
   "timed_out": false,
   "_shards": { ... },
   "hits": {
      "total":      2,
      "max_score":  1,
      "hits": [
         {
            "_index":          "megacorp",
            "_type":           "employee",
            "_id":             "3",
            "_score":          1,
            "_source": {
               "first_name":  "Douglas",
               "last_name":   "Fir",
               "age":         35,
               "about":       "I like to build cabinets",
               "interests": [ "forestry" ]
            }
         },
         {
            "_index":          "megacorp",
            "_type":           "employee",
            "_id":             "1",
            "_score":          1,
            "_source": {
               "first_name":  "John",
               "last_name":   "Smith",
               "age":         25,
               "about":       "I love to go rock climbing",
               "interests": [ "sports", "music" ]
            }
         }
      ]
   }
```


MUNI ÚVT

Simple Query with search string

GET /megacorp/employee/_search?q=last_name:Smith

```
{
   ...
   "hits": {
      "total":      2,
      "max_score":  0.30685282,
      "hits": [
         {
            ...
            "_source": {
               "first_name":  "John",
               "last_name":   "Smith",
               "age":         25,
               "about":       "I love to go rock climbing",
               "interests": [ "sports", "music" ]
            }
         },
         {
            ...
            "_source": {
               "first_name":  "Jane",
               "last_name":   "Smith",
               "age":         32,
               "about":       "I like to collect rock albums",
               "interests": [ "music" ]
            }
         }
      ]
   }
}
```


MUNI ÚVT

More complex queries with Query DSL

```
GET /megacorp/employee/_search
{
    "query" : {
        "match" : {
            "last_name" : "Smith"
        }
    }
}
```

- Query DSL is a JSON language for more complex queries
- Will be sent as payload with the search request
- Match clause has the same semantics as in simple query

More complex queries with Query DSL

- Consist of a query and a filter part
- Query part matches all entries with last_name “smith” (2)
- Filter will then only select entries which fulfill the range filter (1)

  “age”: {“gt” : 30 }

```
GET /megacorp/employee/_search
{
    "query" : {
        "filtered" : {
            "filter" : {
                "range" : {
                    "age" : { "gt" : 30 } (1)
                }
            },
            "query" : {
                "match" : {
                    "last_name" : "smith" (2)
                }
            }
        }
    }
}
```

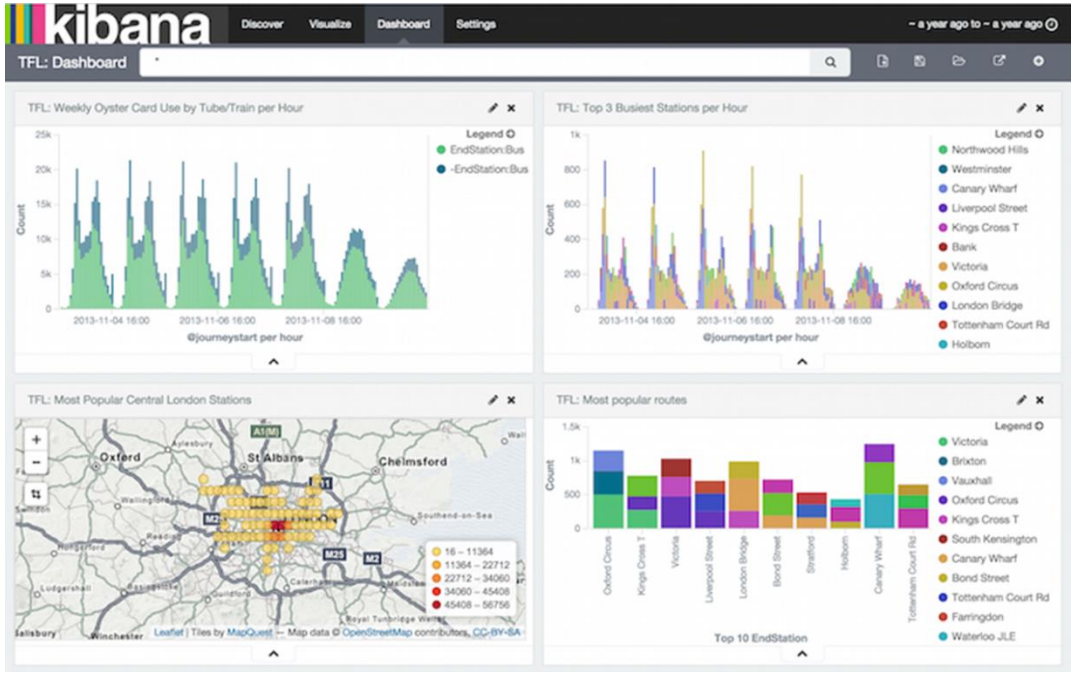

MUNI ÚVT

Some query possibilities

- **Combined search on different attributes and different indices**
  - Many possibilities for full-text search on attribute values
    - Exact, non-exact, proximity (phrases), partial match
  - Support well-known logical operators (And / or, …)
  - Range queries (i.e. date ranges)
  - …
- **Control relevance and ranking of search results, sort them**
  - Boost relevance while indexing
  - Boost or ignore relevance while querying
  - Different possibilities to sort search results otherwise

Kibana

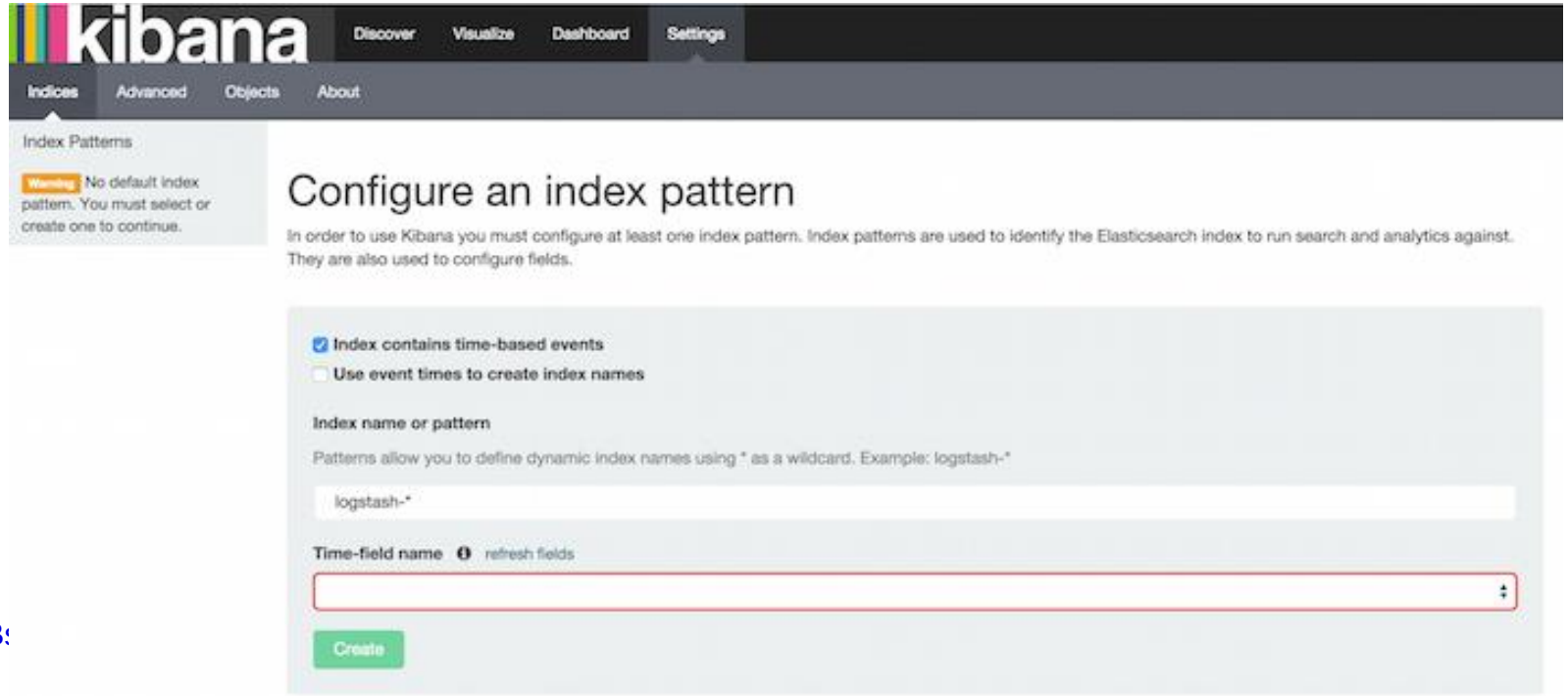
- Web-based application for exploring and visualizing data
- Modern Browser-based interface (HTML5 + JavaScript)
- Ships with its own web server for easy setup
- Seamless integration with Elasticsearch



MUNI ÚVT

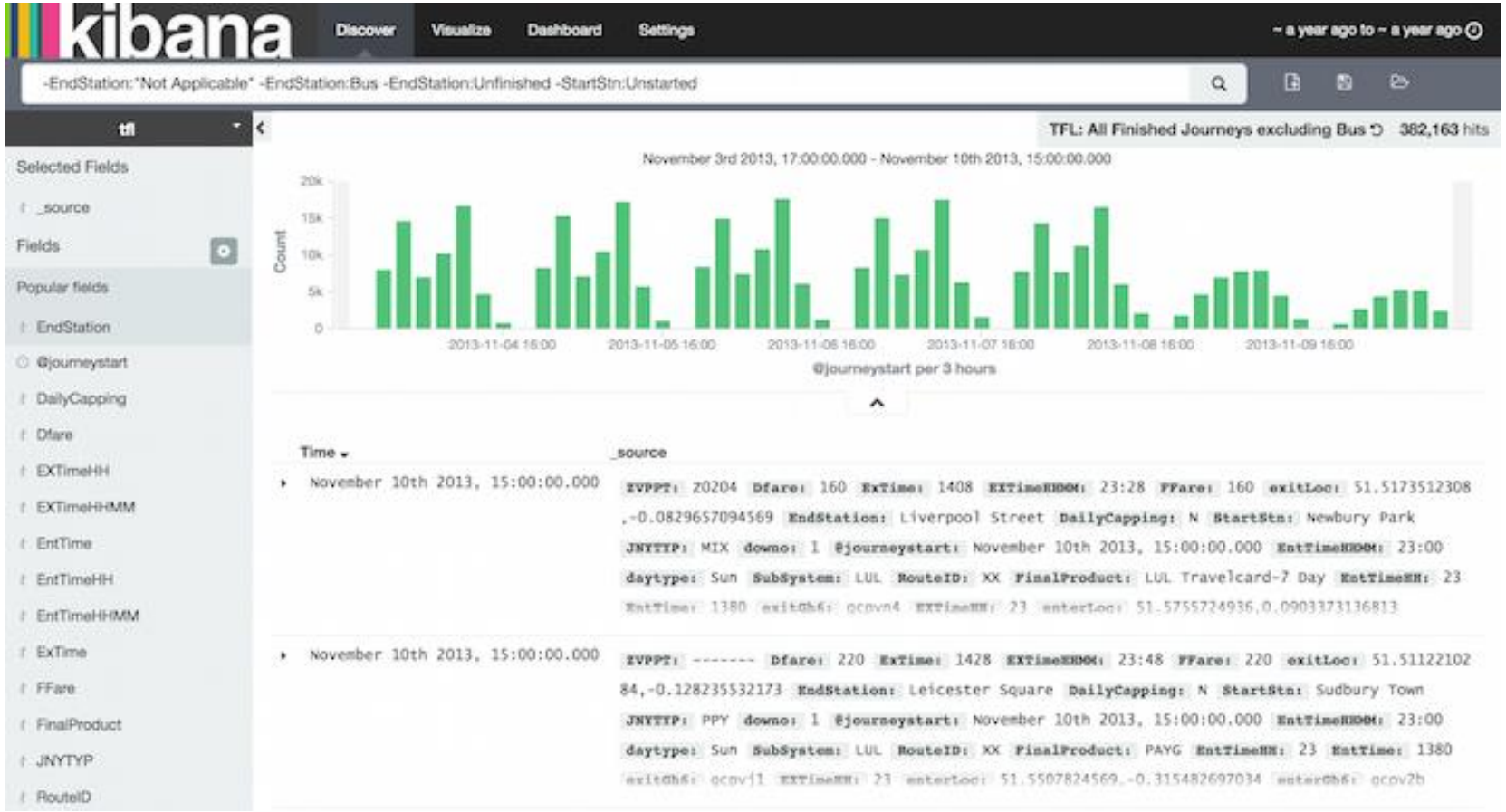
Configure Kibana

- **After installation first configure Kibana to access Elasticsearch server(s)**
  - Should be done by editing the Kibana config file
- **Then use web UI to configure indexes to use**



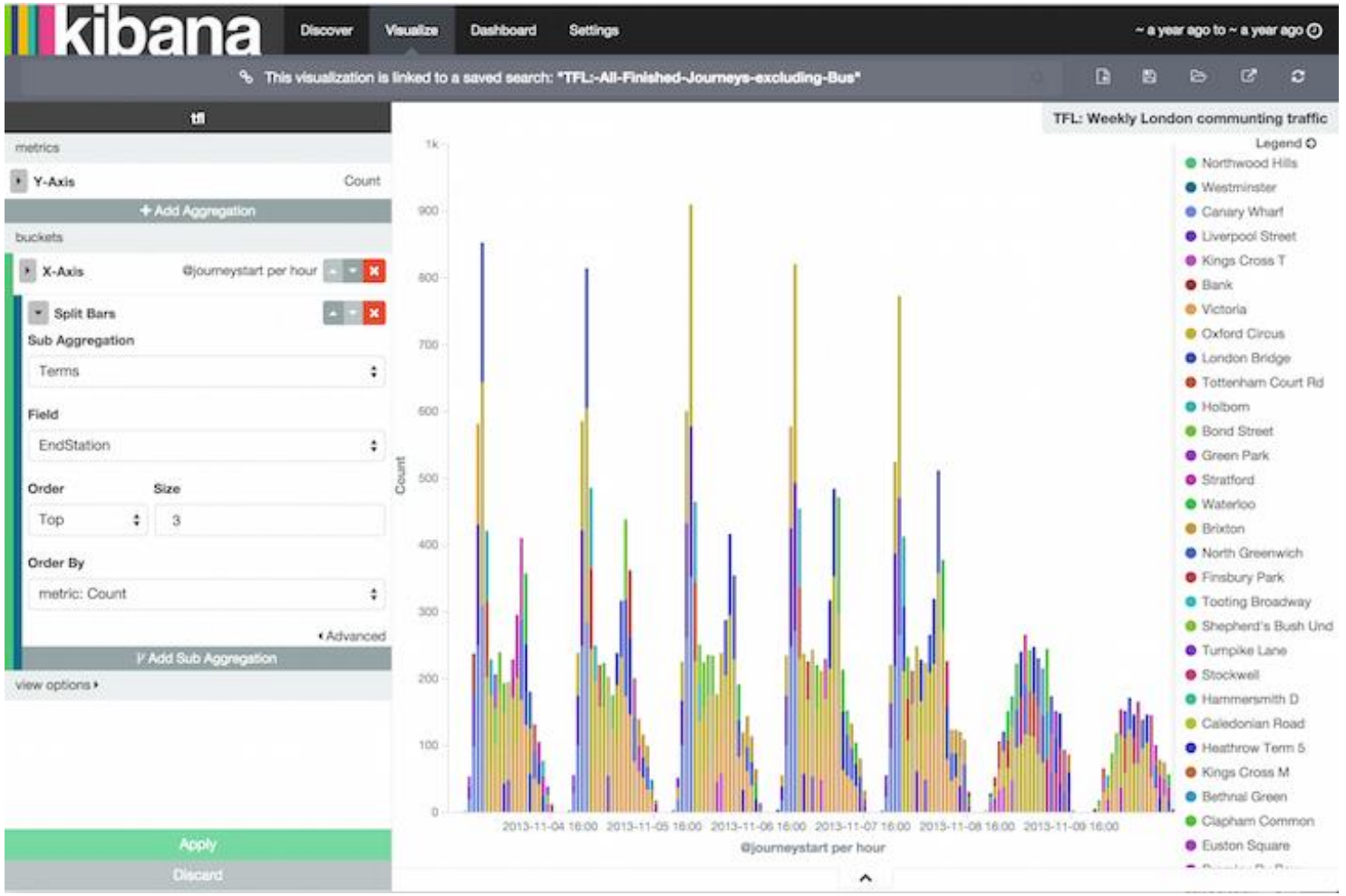
MUNI ÚVT

Discover data



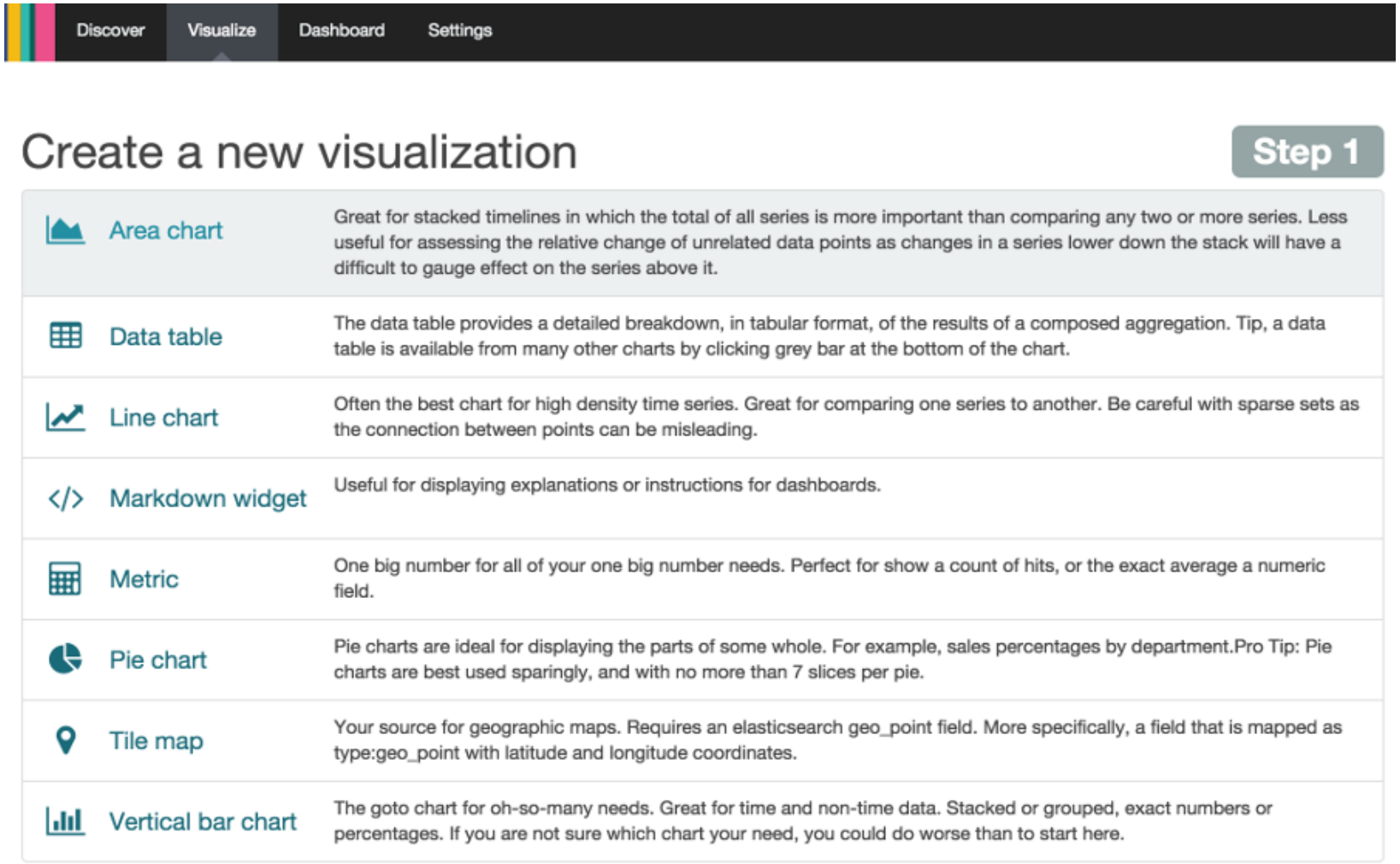
MUNI ÚVT

Create a visualization



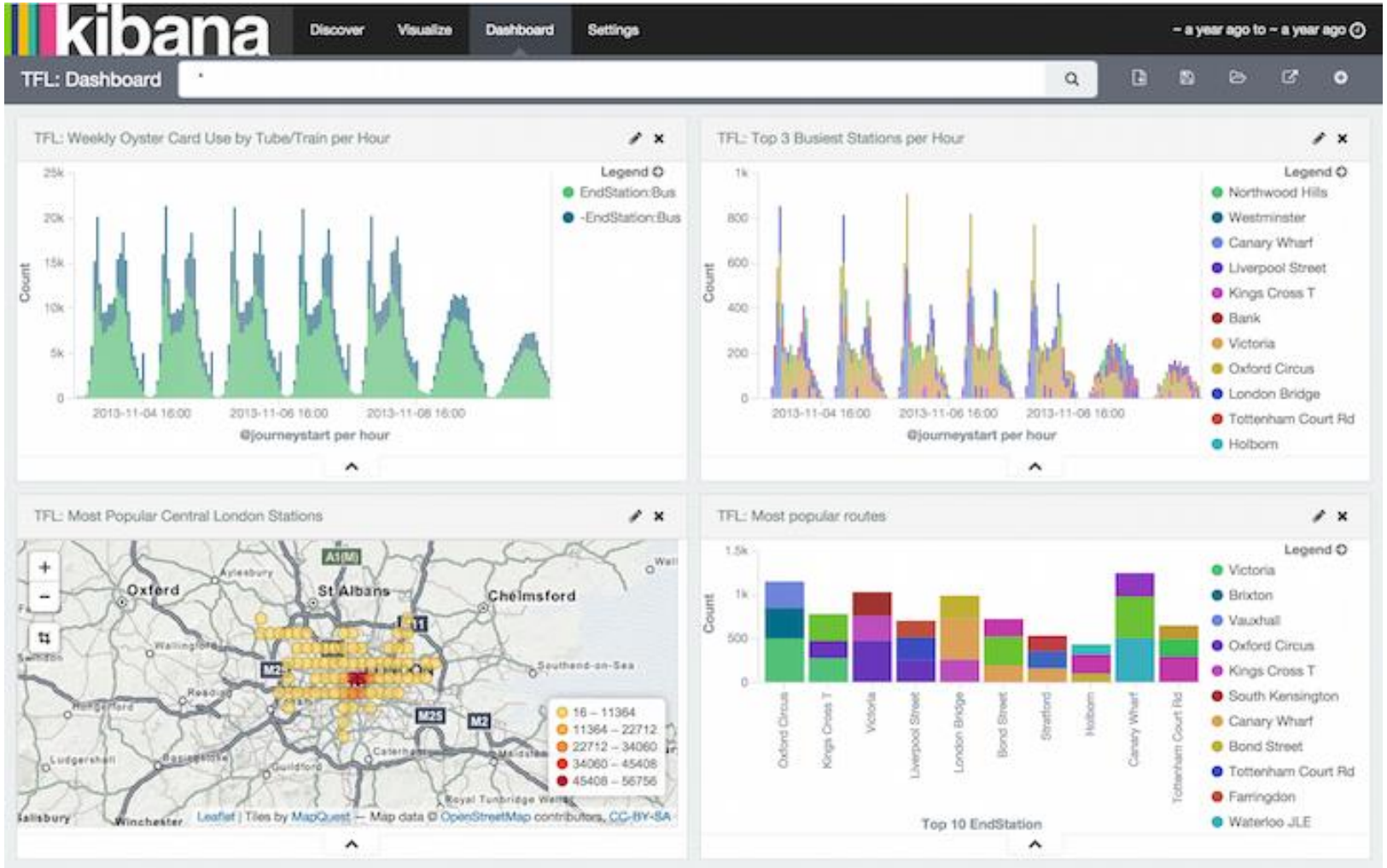
MUNI ÚVT

Different types of visualizations



MUNI ÚVT

Combine visualizations to a Dashboard



MUNI ÚVT

MUNI
ÚVT

Typical ELK use cases

Some use cases of the ELK stack

- **Log data management and analysis**
- **Monitor systems and/or applications and notify operators about critical events**
- **Collect and analyze other (mass) data**
  - i.e. business data for business analytics
  - Energy management data or event data from smart grids
  - Environmental data
- **Use the ELK stack for search driven access to mass data in web-based information systems**

Log data management and analysis

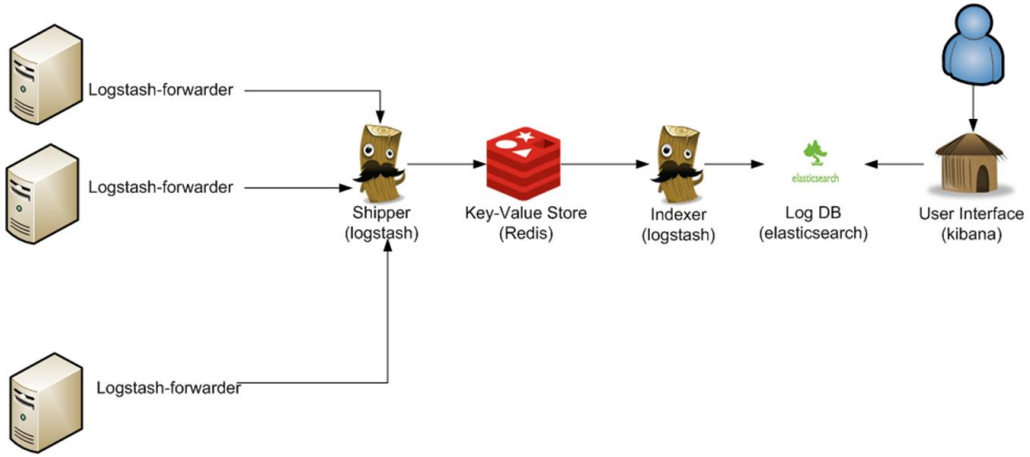
- **Many different types of logs**
  - Application logs
  - Operating system logs
  - Network traffic logs from routers, etc.
- **Different goals for analysis**
  - Detect errors at runtime or while testing applications
  - Find and analyze security threats
  - Aggregate statistical data / metrics

Problems of log data analysis

- **No centralization**
  - Log data could be everywhere
    - on different servers and different places within the same server
- **Accessibility Problems**
  - Logs can be difficult to find
  - Access to server / device is often difficult for analyst
  - High expertise for accessing logs on different platforms necessary
  - Logs can be big and therefore difficult to copy
  - SSH access and grep on logs doesn’t scale or reach
- **No Consistency**
  - Structure of log entries is different for each app, system, or device
  - Specific knowledge is necessary for interpreting different log types
  - Variation in formats makes it challenging to search
    - Many different types of time formats

The ELK stack provides solutions

- **Logstash allows to collect all log entries at a central place (e.g. Elasticsearch)**
  - End users don’t need to know where the log files are located
  - Big log files will be transferred continuously in smaller chunks
- **Log file entries can be transformed into harmonized event objects**
- **Easy access for end users via Browser-based interfaces (e.g. Kibana)**
- **Elasticsearch / Kibana provide advanced functionality for analyzing and visualizing the log data**



Monitoring

- **The ELK stack also provides good solutions for monitoring data and alerting users**
  - Logstash can check conditions on log file entries and even aggregated metrics
  - And conditionally sent notification events to certain output plugins if monitoring criteria are met
    - E.g. forward notification event to email output plugin for notifying user (e.g. operators) about the condition
    - Forwarding notification event to a dedicated monitoring application
  - Elasticsearch in combination with Watcher (another product of Elastic)
    - Can instrument arbitrary Elasticsearch queries to produce alerts and notifications
    - These queries can be run at certain time intervals
    - When the watch condition happens, actions can be taken (sent an email or forwarding an event to another system)

Log analysis examples from the Internet

- Logging and analyzing network traffic
  https://operational.io/elk-stack-for-network-operations-reloaded/
- How to Use ELK to Monitor Performance
  http://logz.io/blog/elk-monitor-platform-performance/
- How Blueliv Uses the Elastic Stack to Combat Cyber Threats
  https://www.elastic.co/blog/how-blueliv-uses-the-elastic-stack-to-combat-cyber-threats
- Centralized System and Docker Logging with ELK Stack
  http://www.javacodegeeks.com/2015/05/centralized-system-and-docker-logging-with-elk-stack.html

Summary

- **The ELK stack is easy to use and has many use cases**
  - Log data management and analysis
  - Monitor systems and / or applications and notify operators about critical events
  - Collect and analyze other (mass) data
  - Providing access to big data in large scale web applications
- **Thereby solving many problems with these types of use cases compared to “handmade”-solutions**
- **Because of its service orientation and cluster readiness it fits nicely into bigger service-oriented applications**

MUNI
ÚVT

ELK deployment made easy

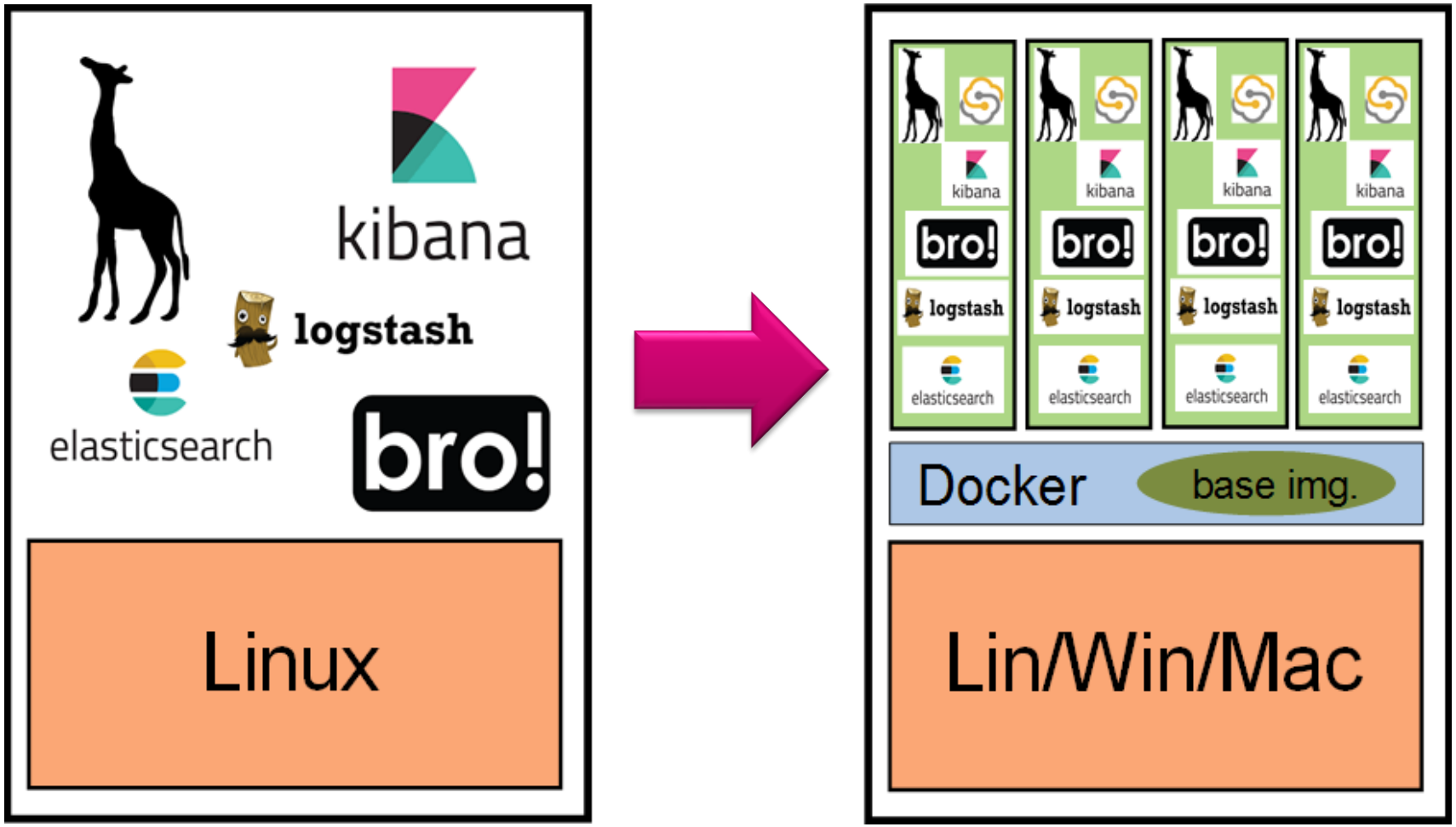
Introducing CopAS

CopAS – *Cops Analytic System*

- fine-tuned production-ready framework running Elastic Platform developed in collaboration with Police CR (PCR)
- Bro, LogStash, ElasticSearch, and Kibana
- graphical user interface (Neck)
- a set of pre-prepared dashboards and visualizations

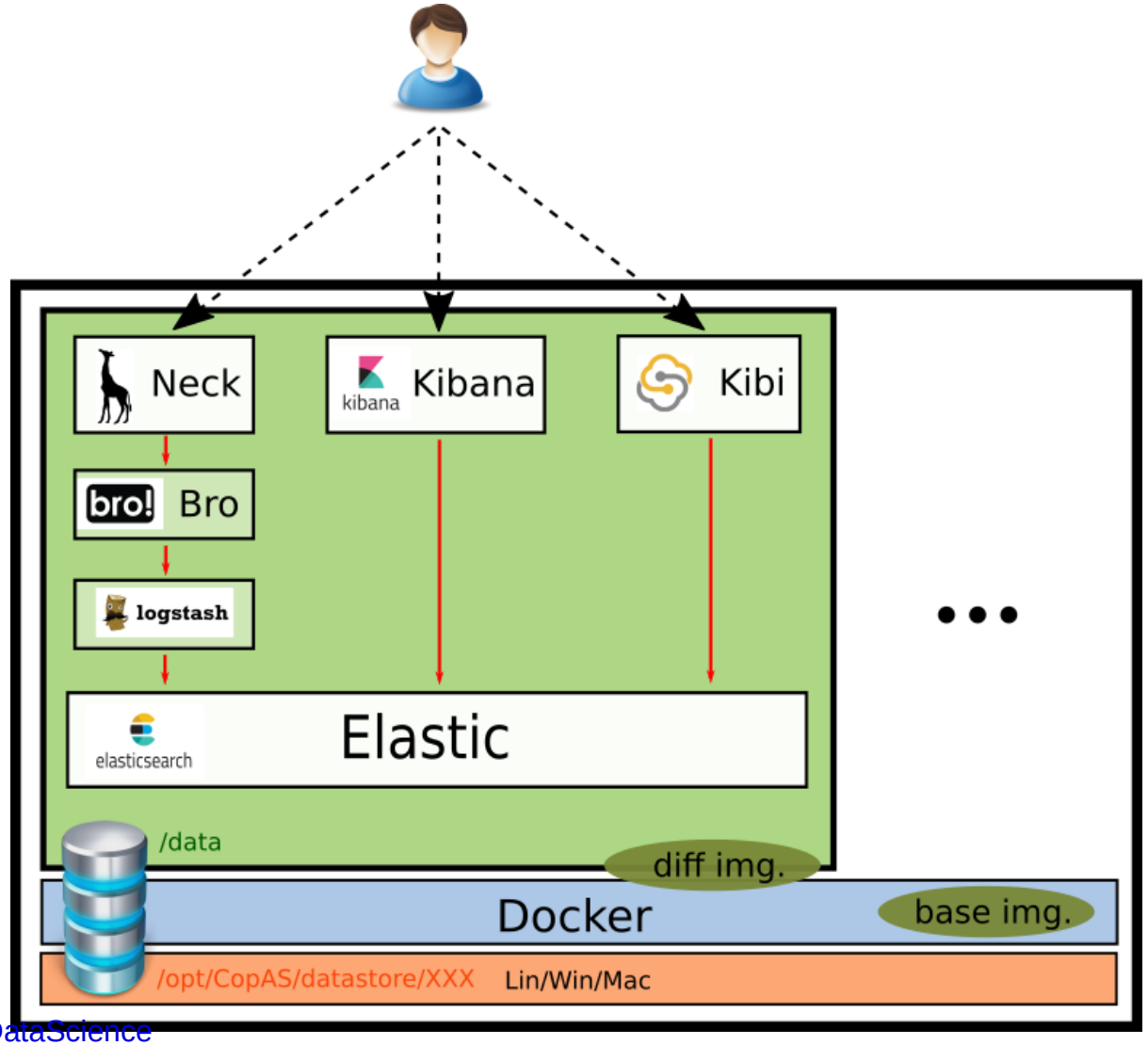
- main emphasis on user-friendliness and ease of deployment & use
  - employs Docker for easier deployment
  - runs on all systems with Docker available (Windows, Linux, MacOS, …)

KIBANA vs. CopAS



MUNI ÚVT

CopAS container



MUNI ÚVT

CopAS – container management

`copas ACTION [container name]`

- *a tool for CopAS container management*

```
[jeronimo@caine /home/jeronimo]$ copas -h
**************************************************************************
* CopAS (Cops Analytic System) -- a system for data analyses using Elastic stack *
*       Created by Institute of Computer Science, Masaryk University, 2017       *
**************************************************************************

Usage: copas ACTION [container_name]
       Available actions:
          create  ... creates a CopAS container (named 'container_name', if provided)
          start   ... starts a CopAS container (named 'container_name', if provided)
          stop    ... stops a CopAS container (named 'container_name', if provided)
          destroy ... destroys a CopAS container (named 'container_name', if provided)
          info    ... shows information about available CopAS containers
          monitor ... monitors the resource usage of CopAS containers
                      (if -l|--live option provided, shows live resource usage)
          enter   ... enters a CopAS container (named 'container_name', if provided)
          update  ... updates the CopAS base image
                      if a filename is provided, updates from the local image
```


MUNI ÚVT

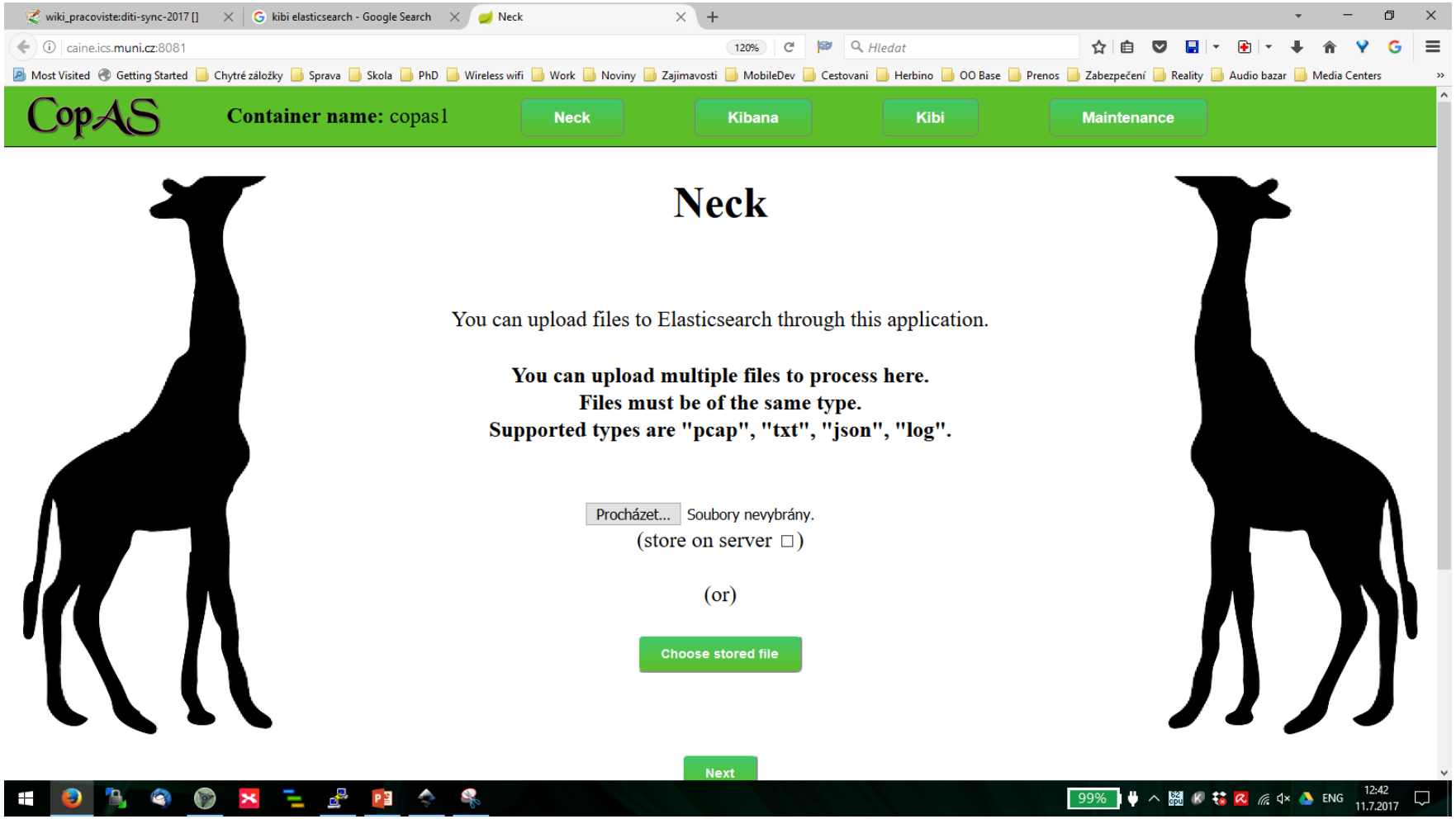
CopAS – example

Example:

- `$ copas info`

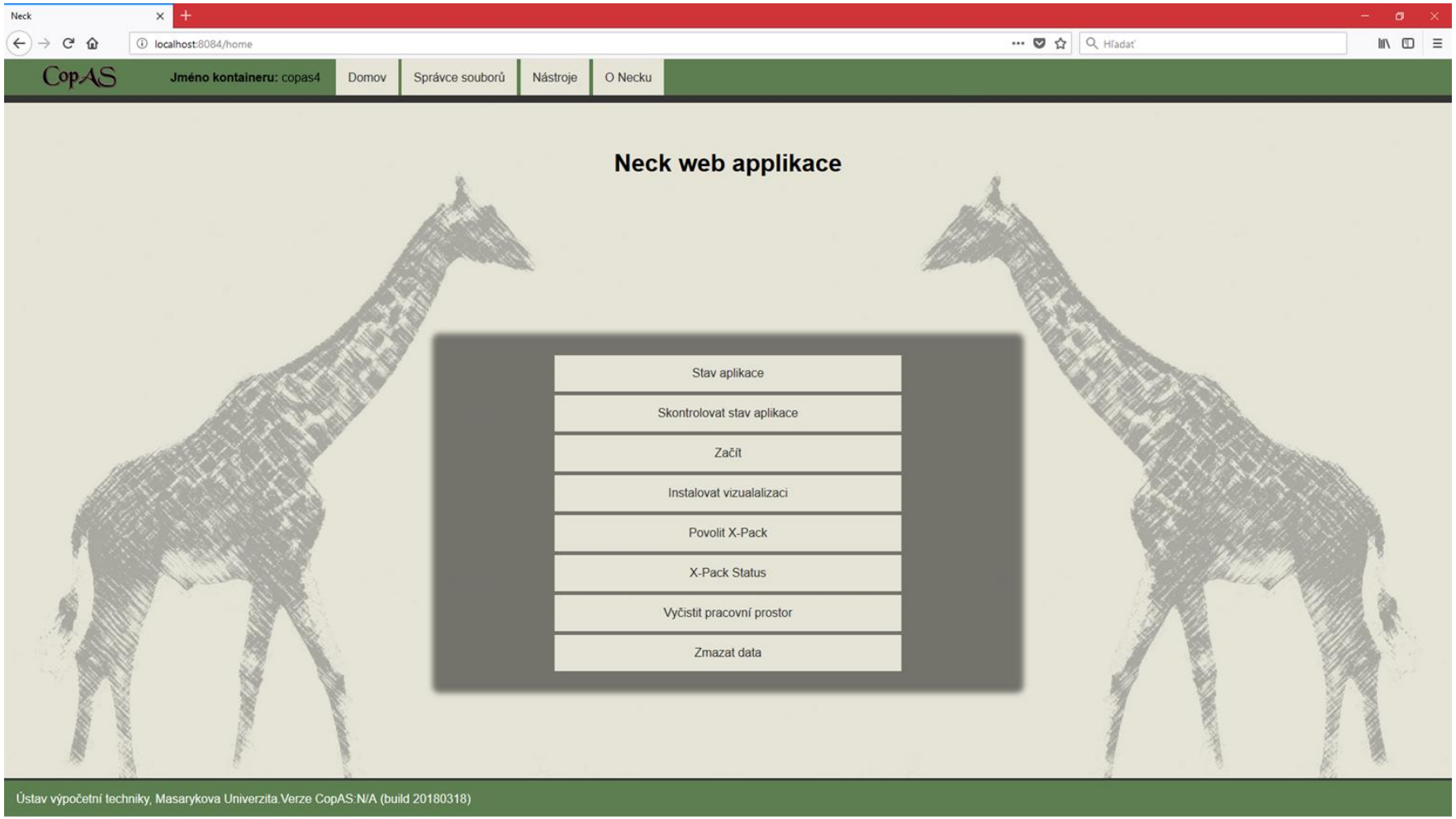
```
[jeronimo@caine /home/jeronimo]$ copas info
Listing all CopAS containers:

    Name            User         URL                                    State
    ================================================================================
    kajinek         jeronimo     http://caine.ics.muni.cz:8083          stopped
    copas2          jeronimo     http://caine.ics.muni.cz:8082          stopped
    copas1          jeronimo     http://caine.ics.muni.cz:8081          RUNNING
```

CopAS – old user environment (version 1.0)



MUNI ÚVT

CopAS – old user environment (version 2.0)



MUNI ÚVT

CopAS version 3.1

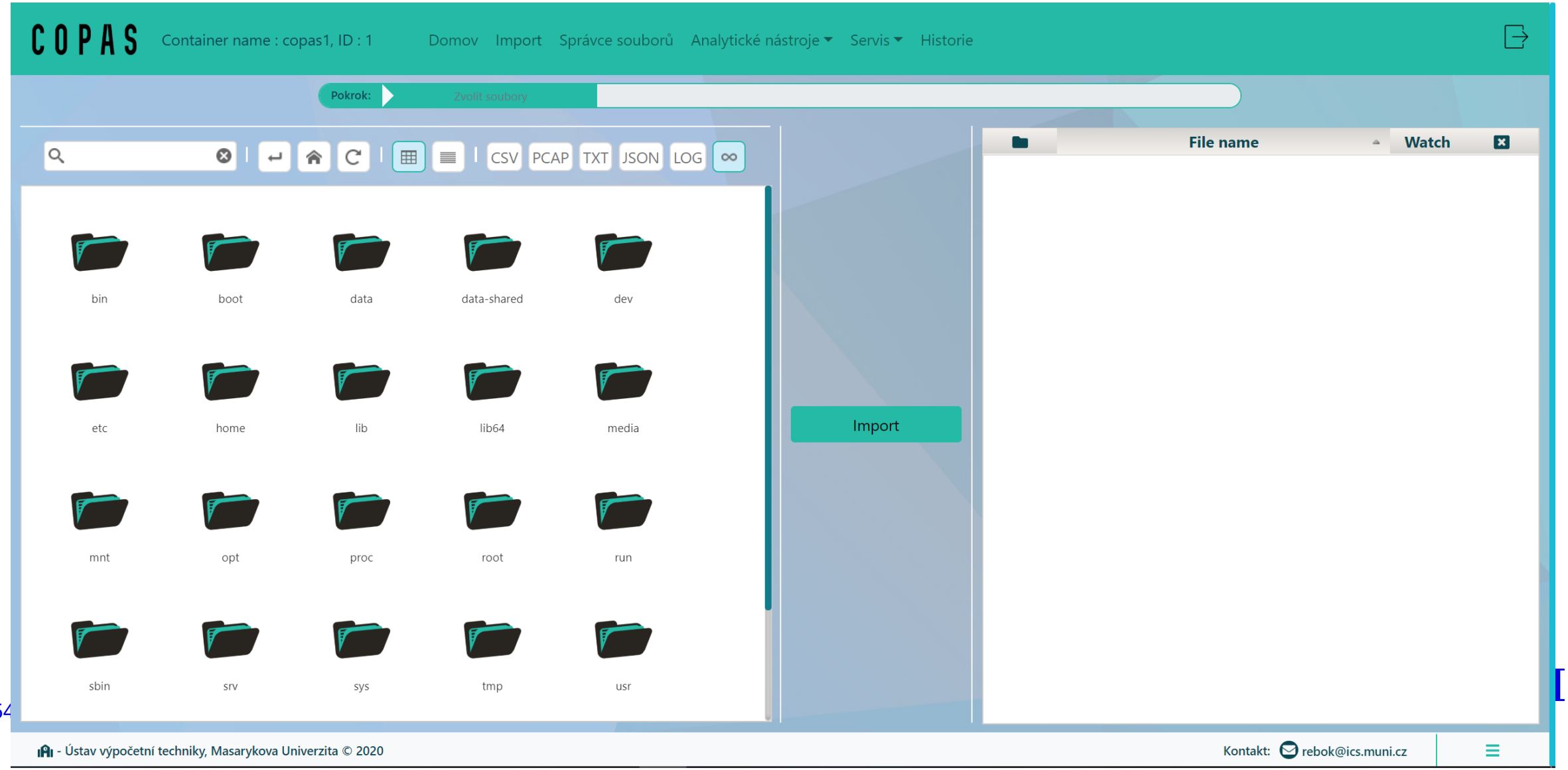
Main changes in workflow and GUI

New functionality

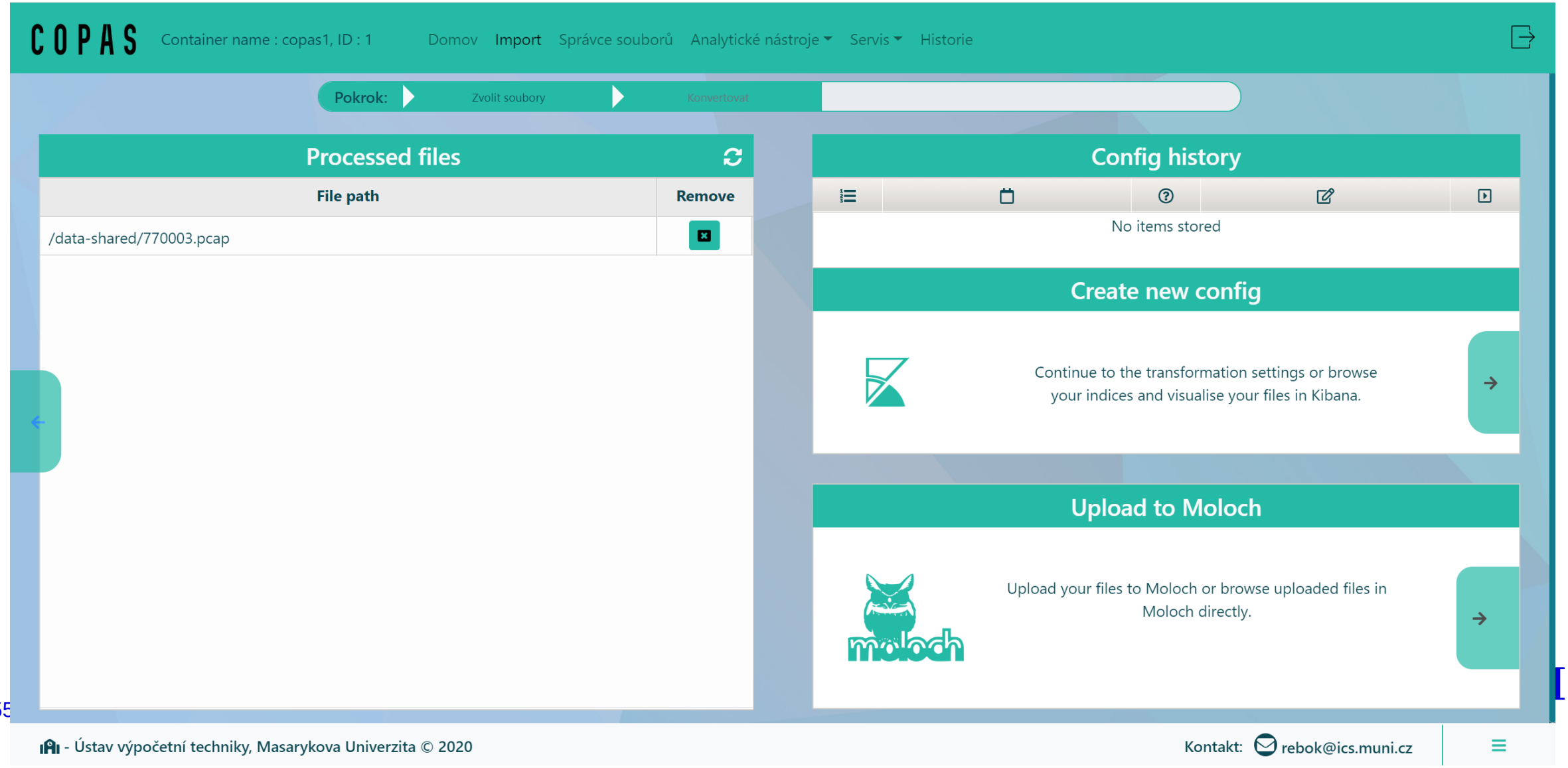
- large files analysis support
  - limited only by available resources
- local files analysis support
- (g)zipped files support
- support for PCAPs and CSVs
- automated files import – *CopAS WatchDog*
  - one can define monitored directories
- backup and restore of containers
  - `copas backup` and `copas import`
  - ability to move containers among different analytical systems
- extended Logstash configuration
- integrated Molo.ch analytic tool (PCAPs only)

CopAS – user environment (version 3.1)

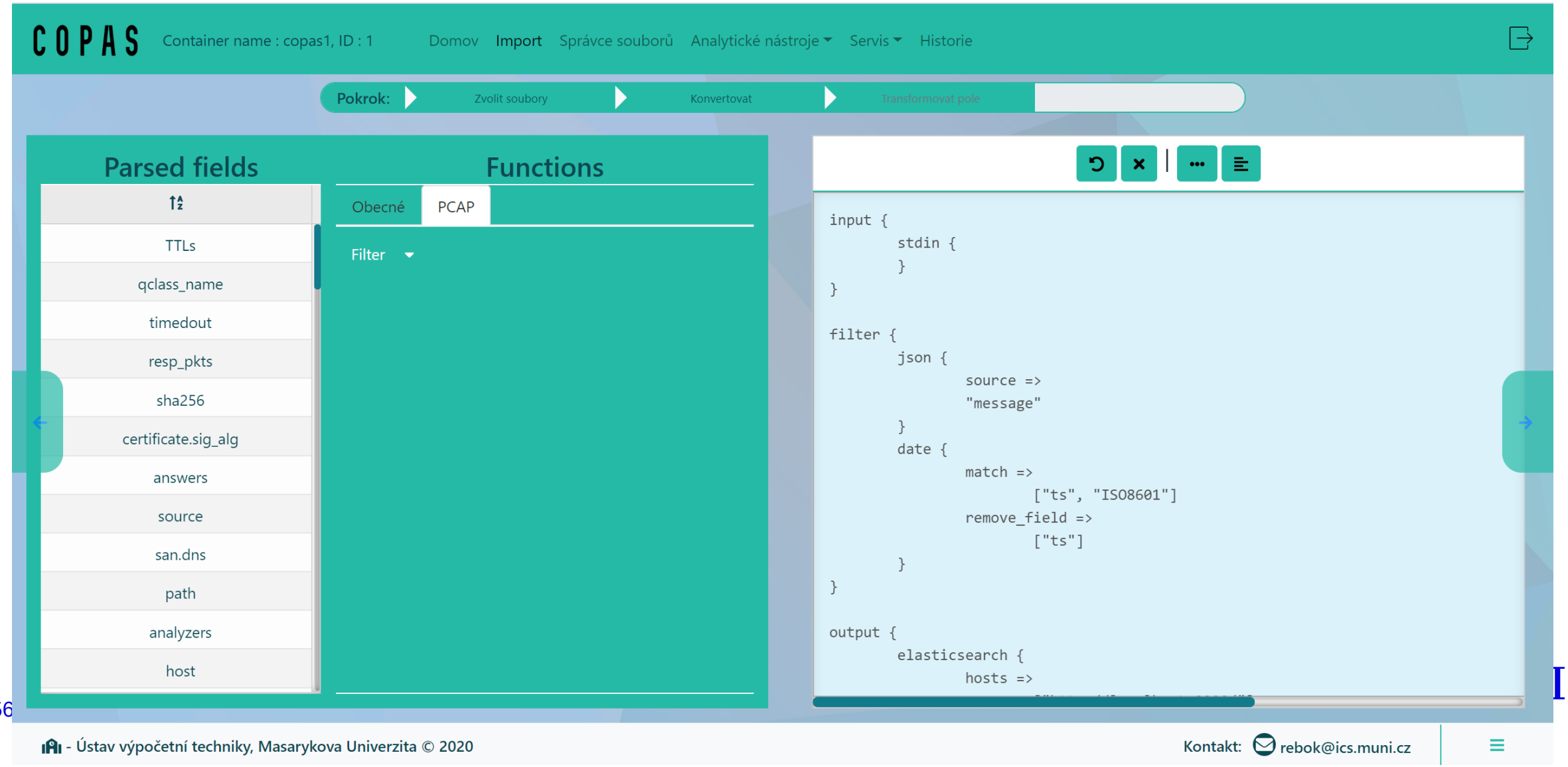
CopAS – user environment (version 3.1)



CopAS – user environment (version 3.1)

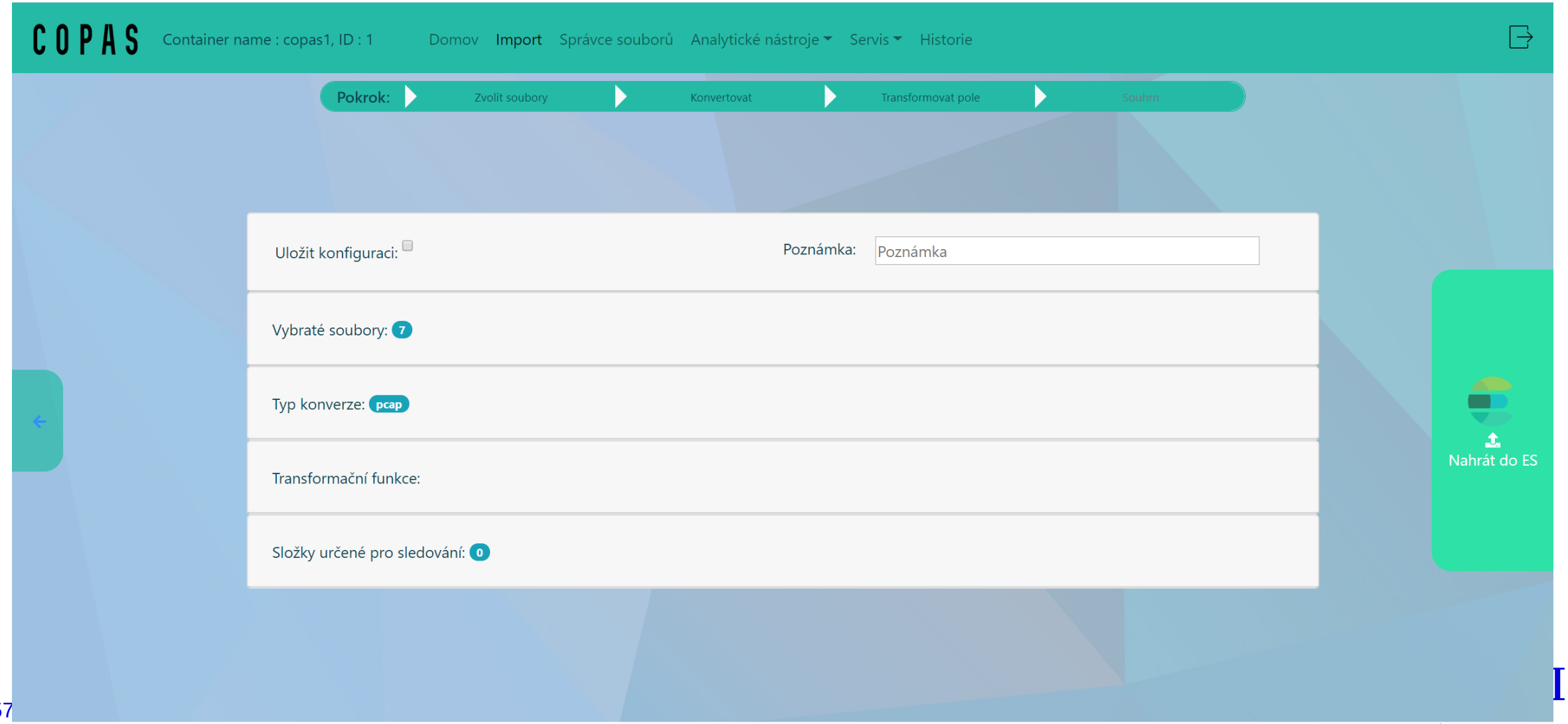


CopAS – user environment (version 3.1)



Kontakt: rebok@ics.muni.cz

CopAS – user environment (version 3.1)



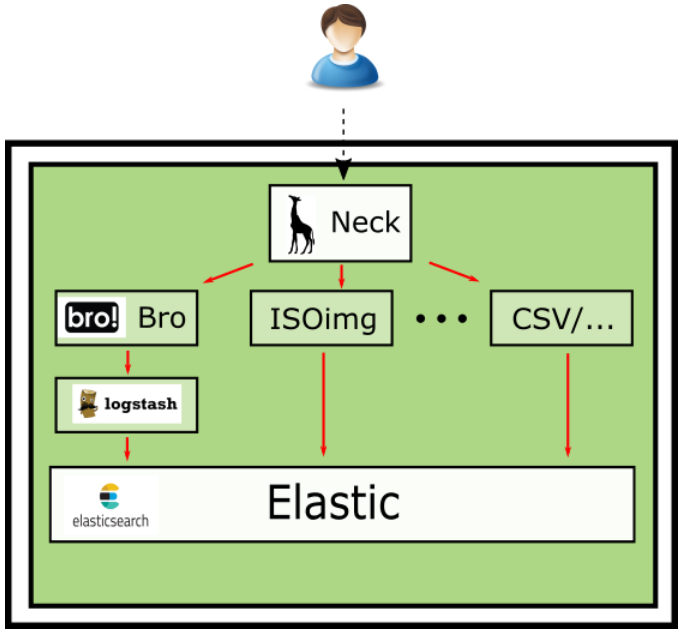
CopAS development and future

CopAS – main development

- great work made by previous PV177/DataScience students
  - K. Gutič and V. Lazárik
- **CopAS v. 4.0 alpha** – several improvements ongoing (O. Machala)
  - modular design, unified GUI

CopAS – not only PCR tool

- PCR specifics are just pre-defined visualizations, dashboards, searches, etc.
  - without specific addons, it is a **generic ES-based data analytic tool**
- assumes multiple input formats support in Neck GUI
  - (proposals for input formats welcomed)



MUNI ÚVT

CopAS availability

CopAS v. 3.1 installation (Linux OS)

- https://frakira.fi.muni.cz/~jeronimo/PV177/copas-install.tgz

CopAS v. 3.1 offline image

- 5,8 GB – not necessary, but easier to deploy
- https://frakira.fi.muni.cz/~jeronimo/PV177/copasimg-20200915.tgz

CopAS v. 4.0 alpha

- https://frakira.fi.muni.cz/~jeronimo/PV177/v4.0/copas-src.tar (2,2 GB)

Testing datasets:

- PCAPs: https://tcpreplay.appneta.com/wiki/captures.html

MUNI
ÚVT

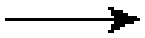
Graph databases

What is a Graph?

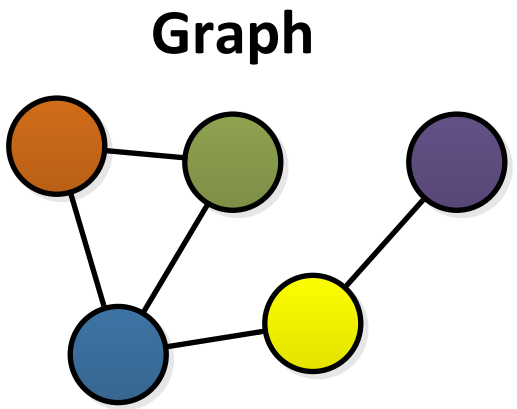
- Formally, a graph is a collection of vertices and edges

- **Less Formally Defined:**
  - A graph is a set of nodes, relationships, and properties
  - A network of connected objects

Object (Vertex, Node)



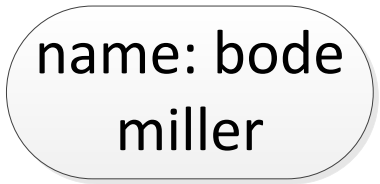
Link (Edge, Arc, Relationship)



MUNI ÚVT

Nodes

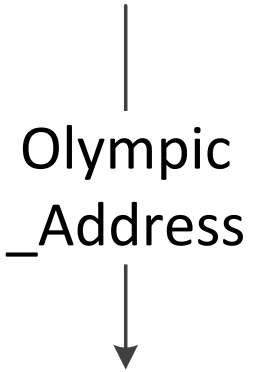
- Nodes represent entities and complex types
- Nodes can contain properties
- Each node can have different properties



Think of nodes as documents that store properties in the form of arbitrary key-value pairs.

Relationships

- Every relationship has a name and direction
- Relationships can contain properties, which can further clarify the relationship
- Must have a start and end node

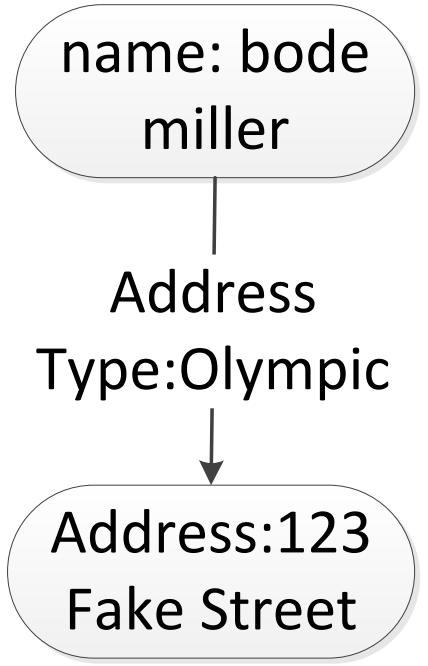


Relationships connect and structure nodes.

MUNI ÚVT

Properties

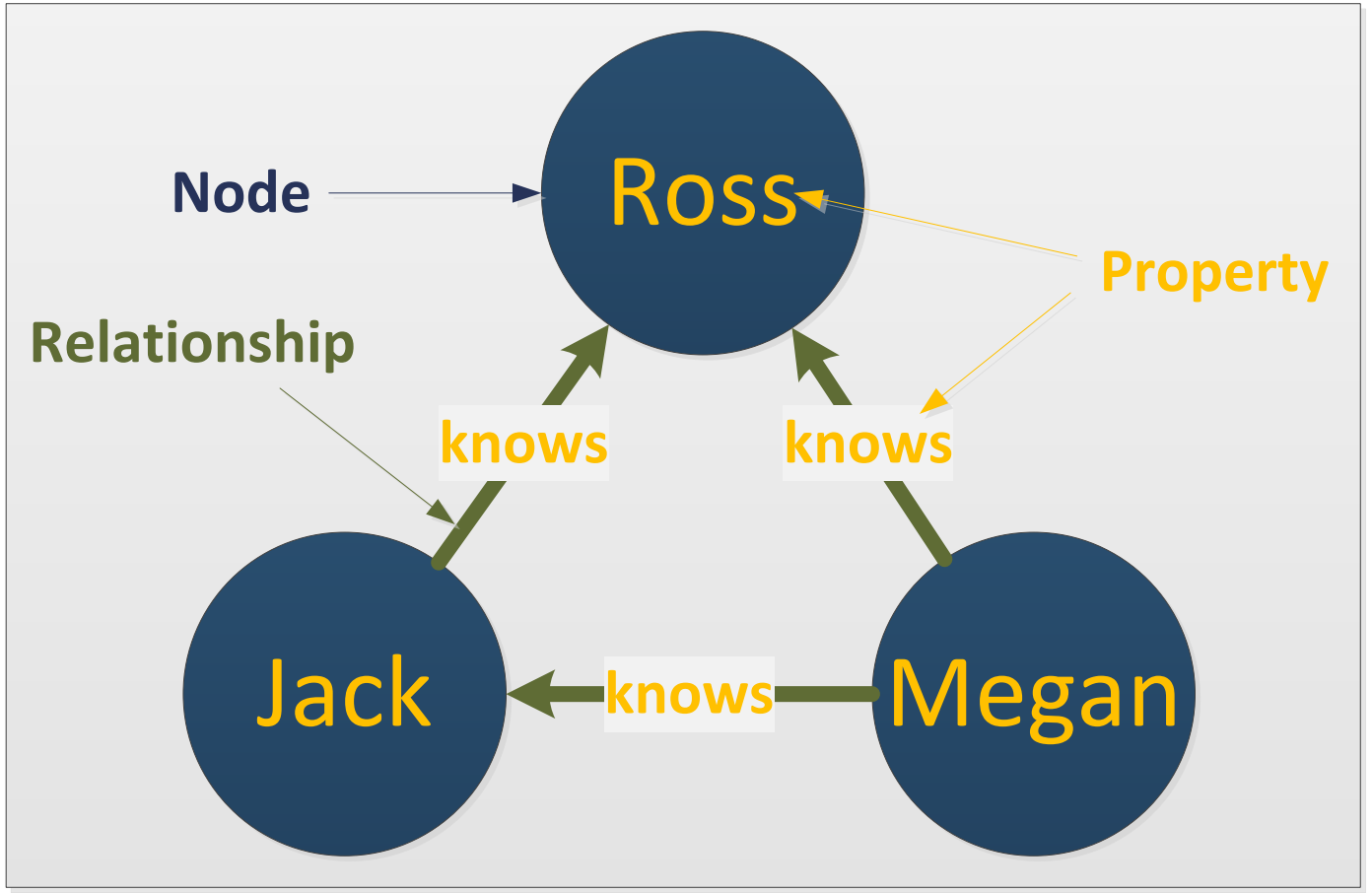
- Key value pairs used for nodes and relationships
- Adds metadata to your nodes and relationships
- Entity attributes
- Relationship qualities



Allows you to create additional semantics to entities and relationships.

MUNI ÚVT

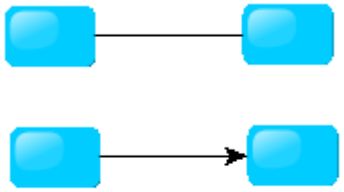
Basic Graph



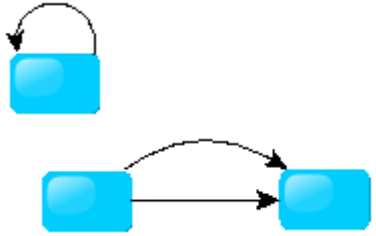
MUNI ÚVT

Different Kinds of Graphs

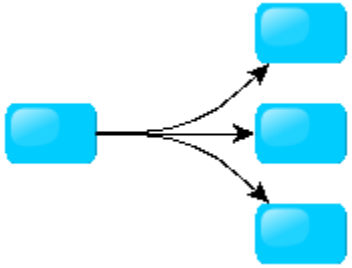
- Undirected Graph
- Directed Graph



- Pseudo Graph
- Multi Graph

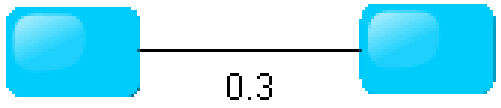


- Hyper Graph

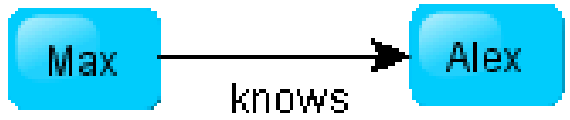


More Kinds of Graphs

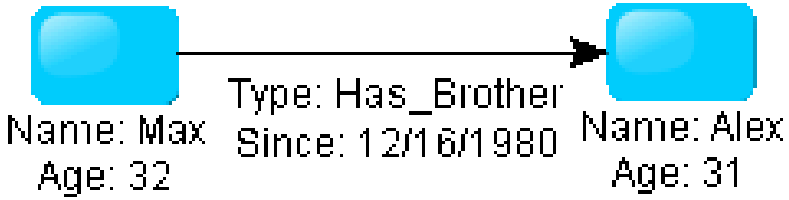
- **Weighted Graph**



- **Labeled Graph**



- **Property Graph**

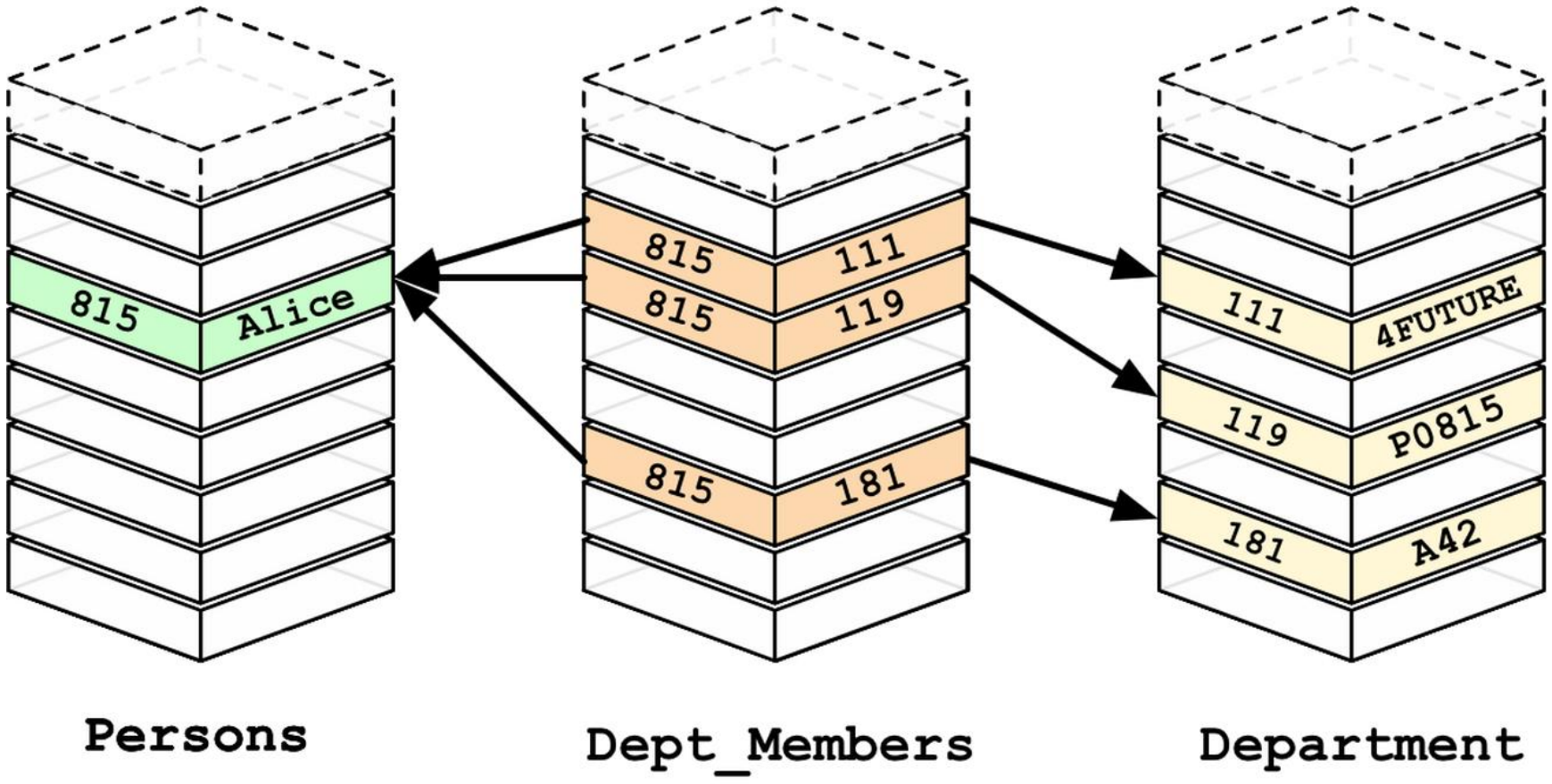


MUNI ÚVT

What is a Graph Database?

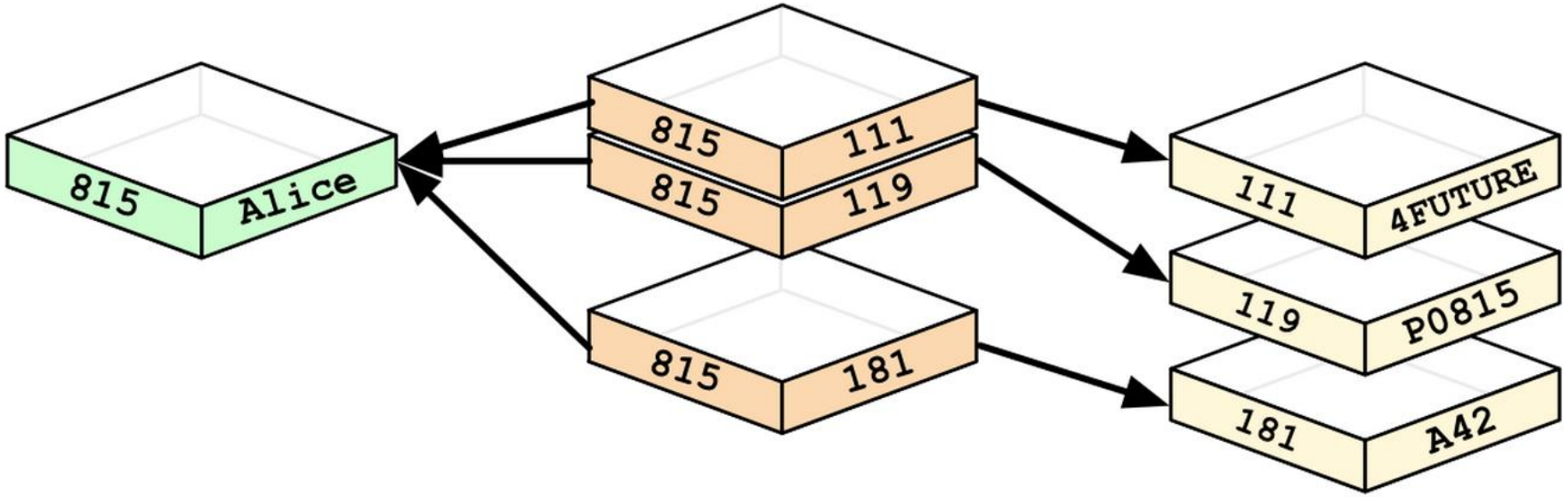
- A database with an explicit graph structure
- Each node knows its adjacent nodes
- As the number of nodes increases, the cost of a local step (or hop) remains the same
- Plus an Index for lookups

Relational Databases



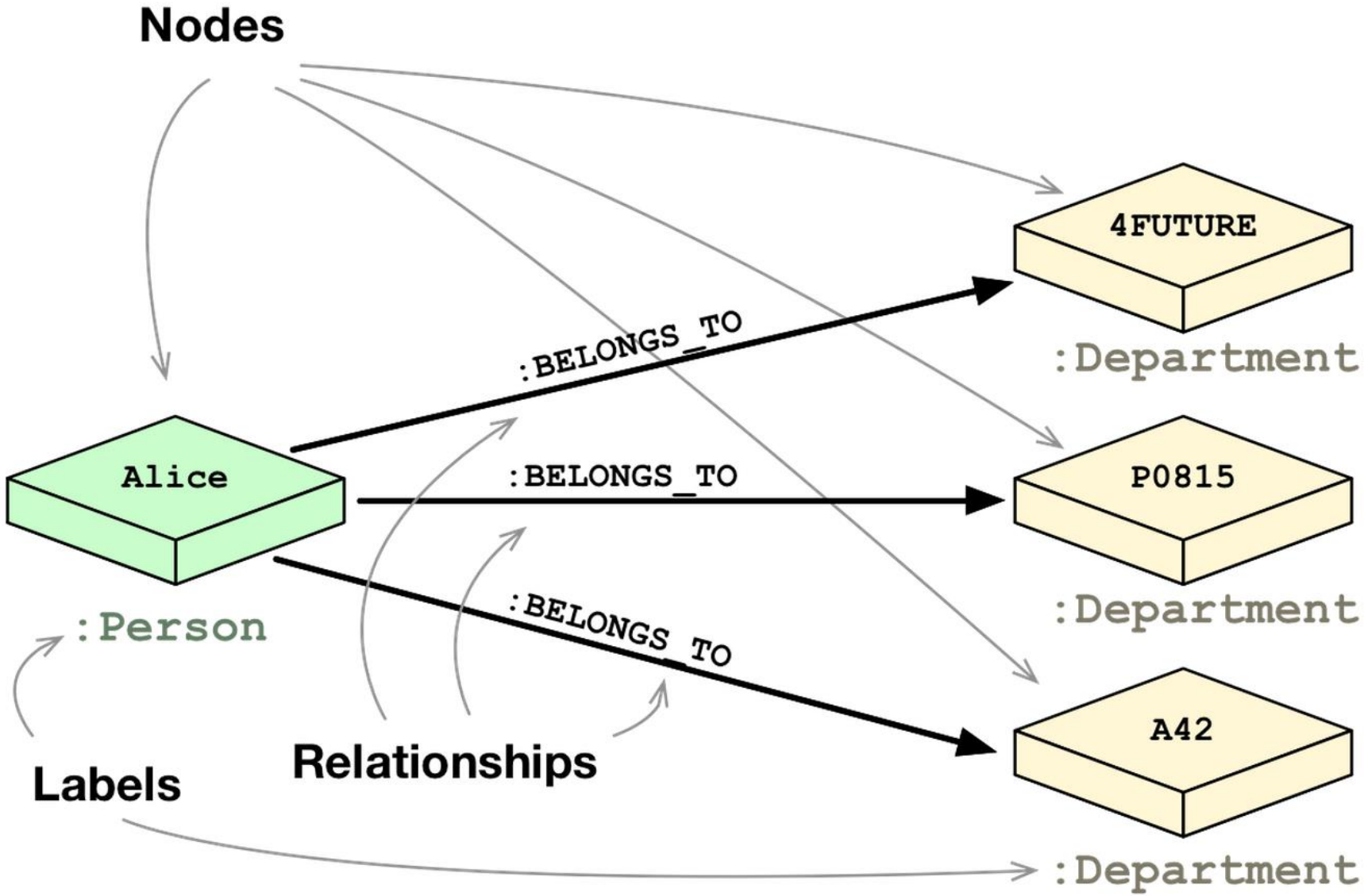
MUNI ÚVT

Relational To Graph Databases …



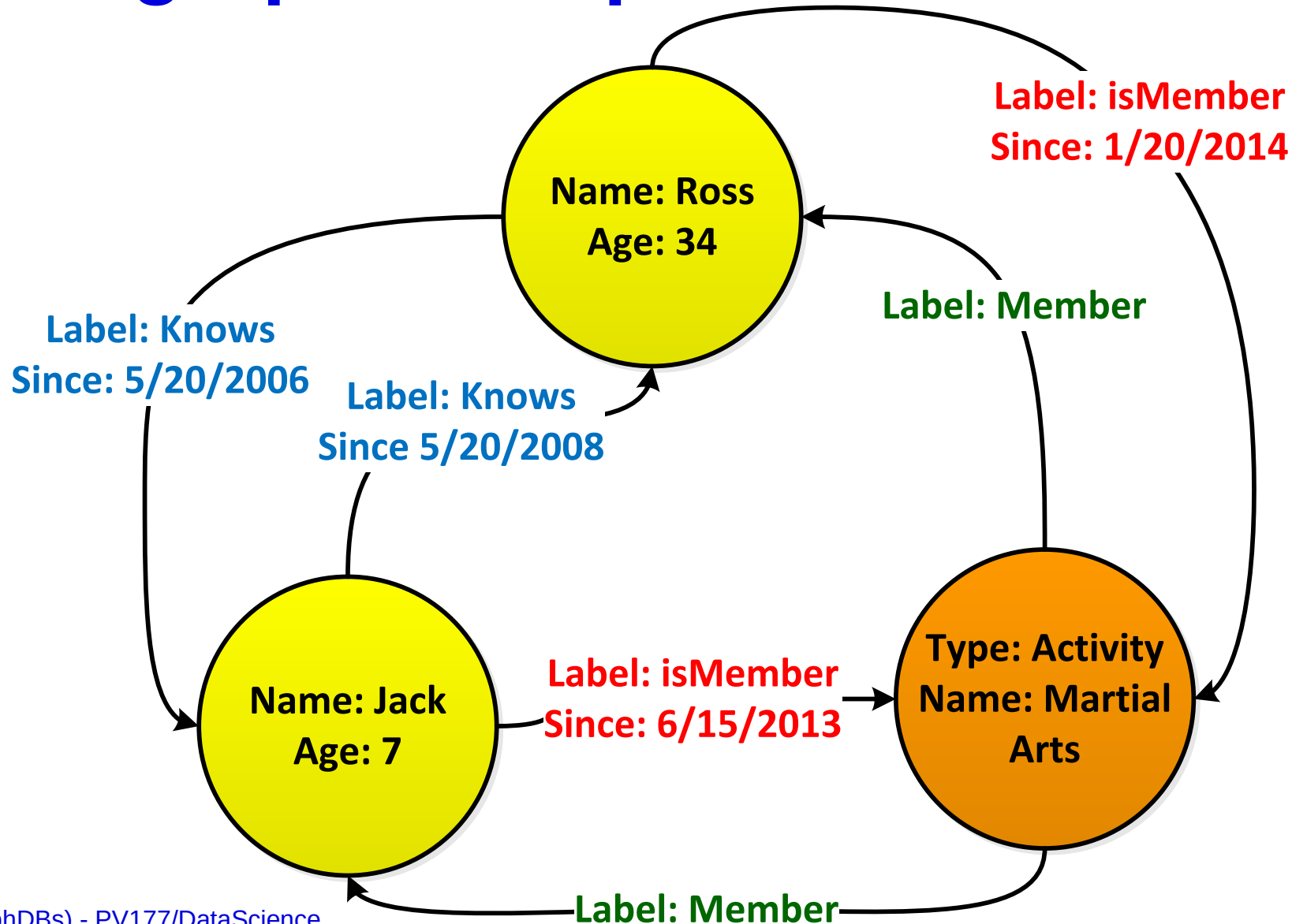
MUNI ÚVT

Graph Databases



MUNI ÚVT

Another graph example



MUNI ÚVT

GRAPH DB VS RELATIONAL DB

- **Each entity table is represented by a label on nodes**
- **Each row in an entity table is a node**
- **Columns on those tables become node properties**
- **Join tables are transformed into relationships, columns on those tables become relationship properties**

GRAPH DATABASES: PROS AND CONS

Pros:

- ✓Easy to query
- ✓Ability to connect disparate data easily without needing a common data model

Cons:

- Requires a different way to think about data
- No single graph query language

When to use / not use Graph DBs?

Graph DBs are great for:

- data, which are connected and/or where relationships matter
- data, which you want to query using various graph algorithms

but not ideal for:

- not optimized for massive graph traversing
  - *MATCH (n) WHERE n.name=`Jenifer` RETURN n*
    - but great for particular graph traversing like
      MATCH (n:Person {name: `Jenifer`})-[r:KNOWS]->(p:Person) RETURN p
  - it will work, but the performance will not be very good

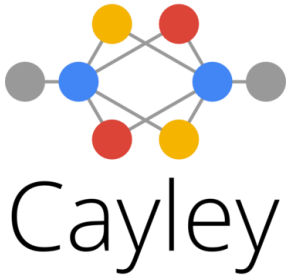
Neo4j vs. RDBMS (book „*Neo4j in action*“)

***Example:* in a social network, find all the friends of a user's friends. Even more so, for friends of friends of friends.**

- 1.000.000 users, query for 1.000 users
- max. time 1 hour

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Popular Graph DB Engines



Pros:

- Open-source version available
- Runs complex distributed queries
- Scales out through sharded storage
- Returns data natively in JSON, making it ideally suited for web development
- Written on top of GraphQL

Cons:

- No native windows installation
  - Docker could be used

Pros:

- Multi model DB – both graph and document DB
- Easily add users/roles
- Supports multiple databases

Cons:

- Requires more schema design up front

Pros:

- Open-source version available
- Steep learning curve, more user-friendly
- Runs on Windows natively - in either a console or as a service
- Large and active user community

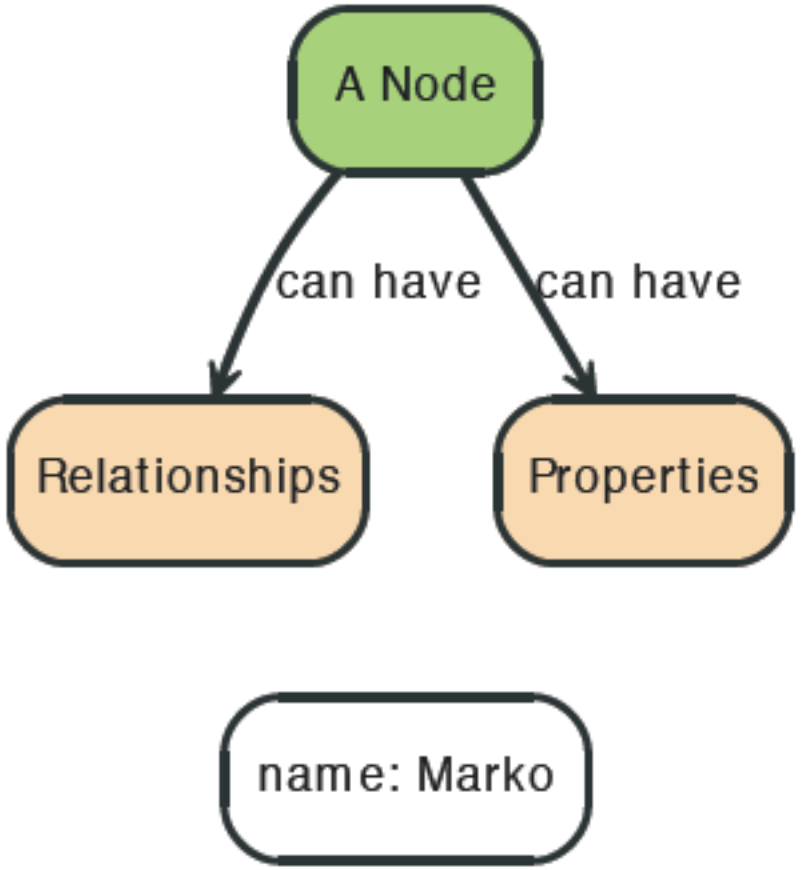
Cons:

- Only one DB can be running on a single port at a time

NEO4J – WHAT DOES IT PROVIDE?

- ✓Full ACID (atomicity, consistency, isolation, durability)
- ✓REST API
- ✓Property Graph
- ✓Lucene Full-Text Index
- ✓High Availability (with Enterprise Edition)

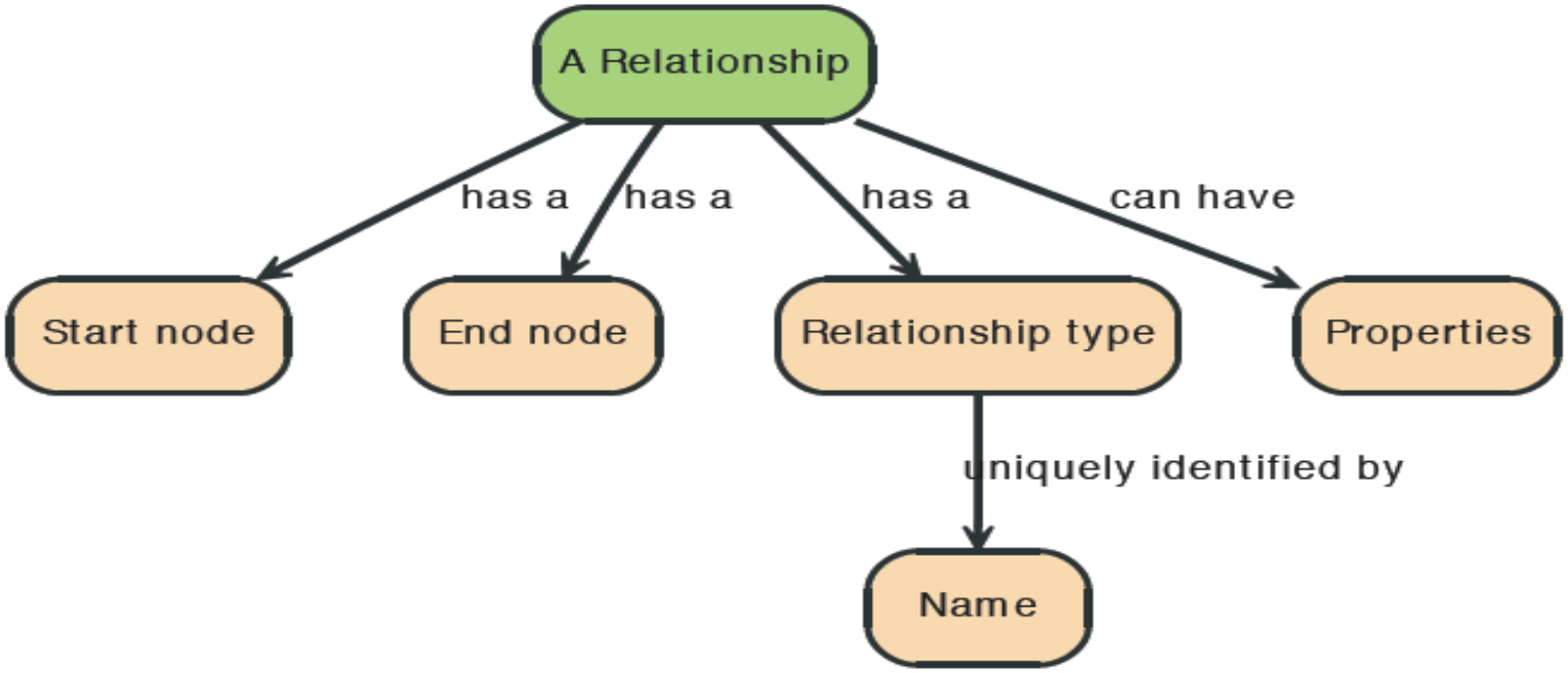
Node in Neo4j



MUNI ÚVT

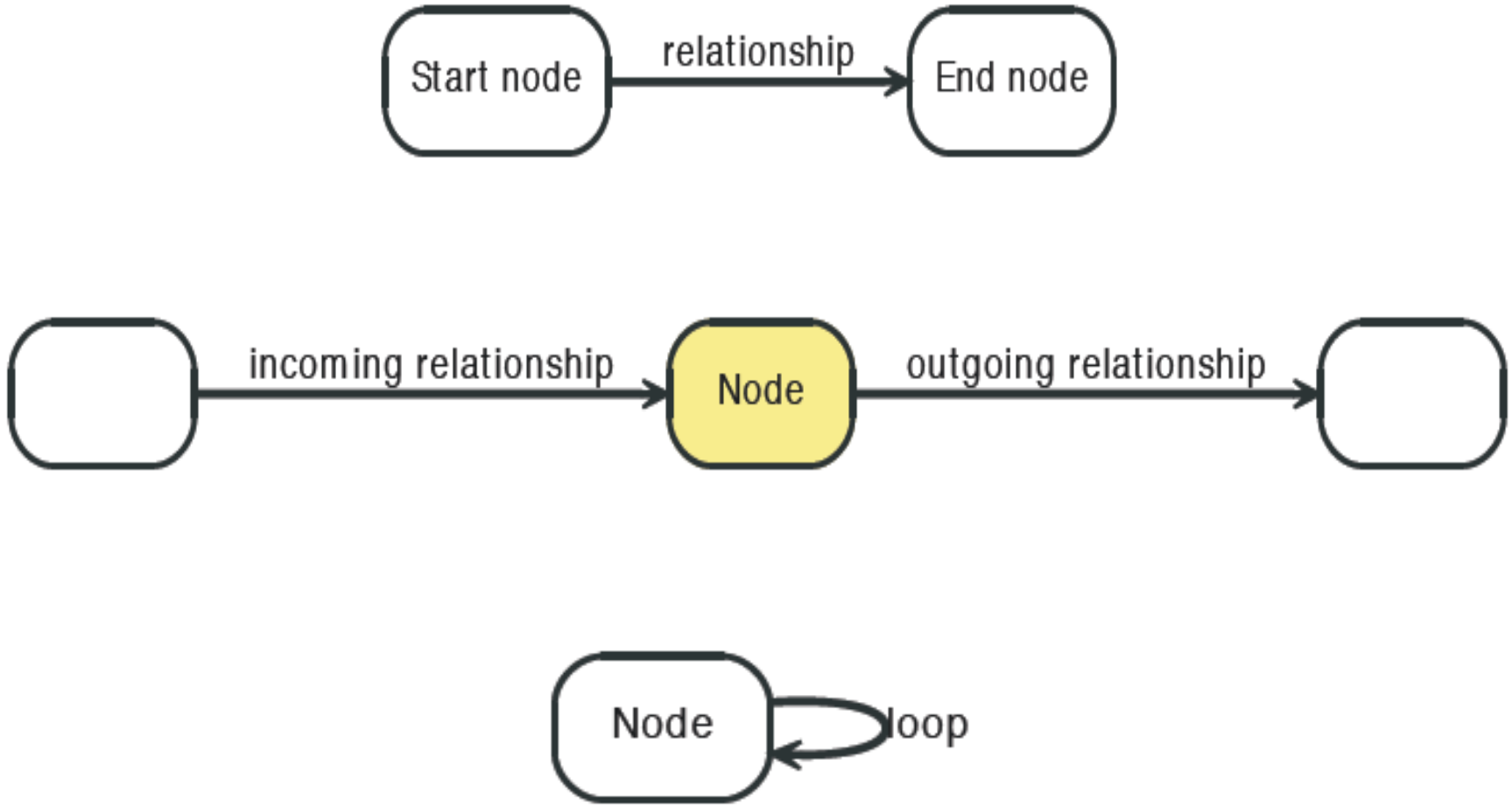
Relationships in Neo4j

- **Relationships between nodes are a key part of Neo4j**



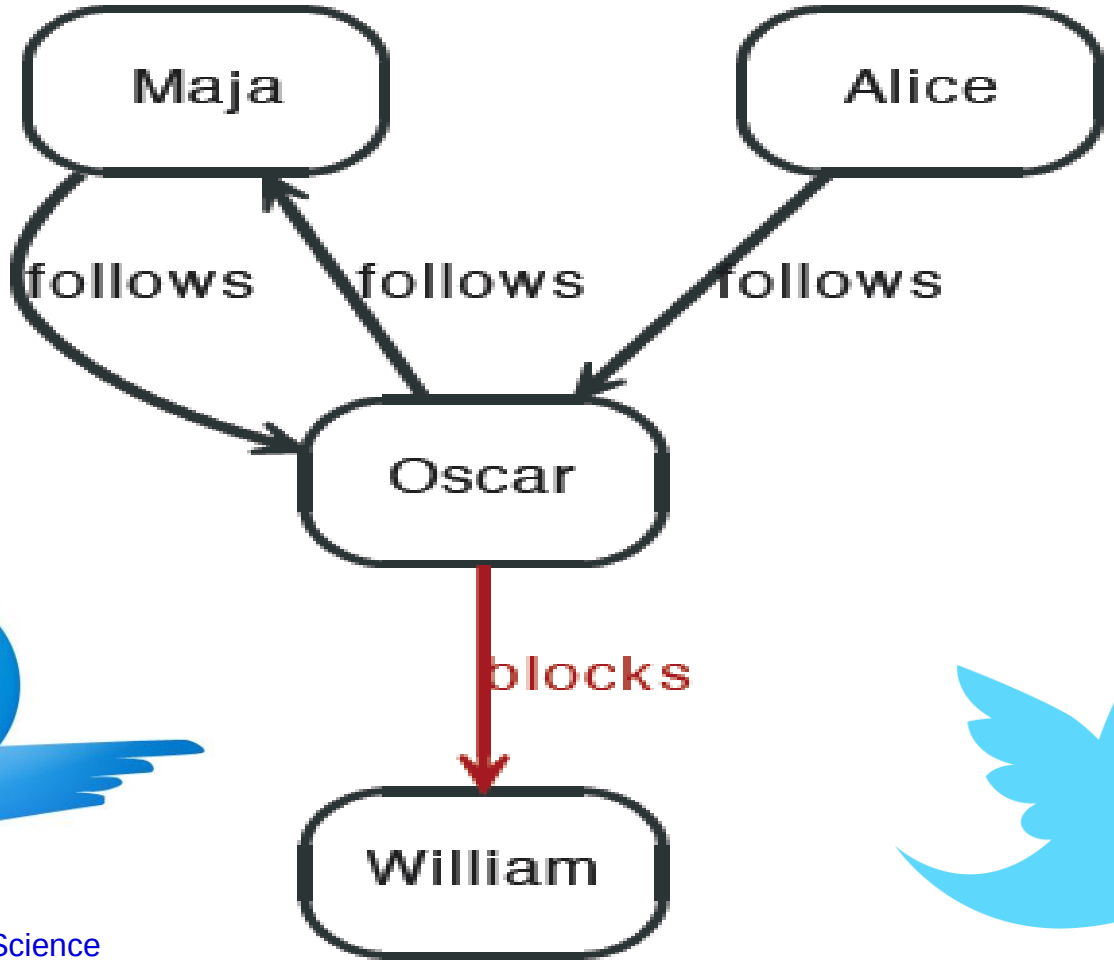
MUNI ÚVT

Relationships in Neo4j



MUNI ÚVT

Twitter and relationships



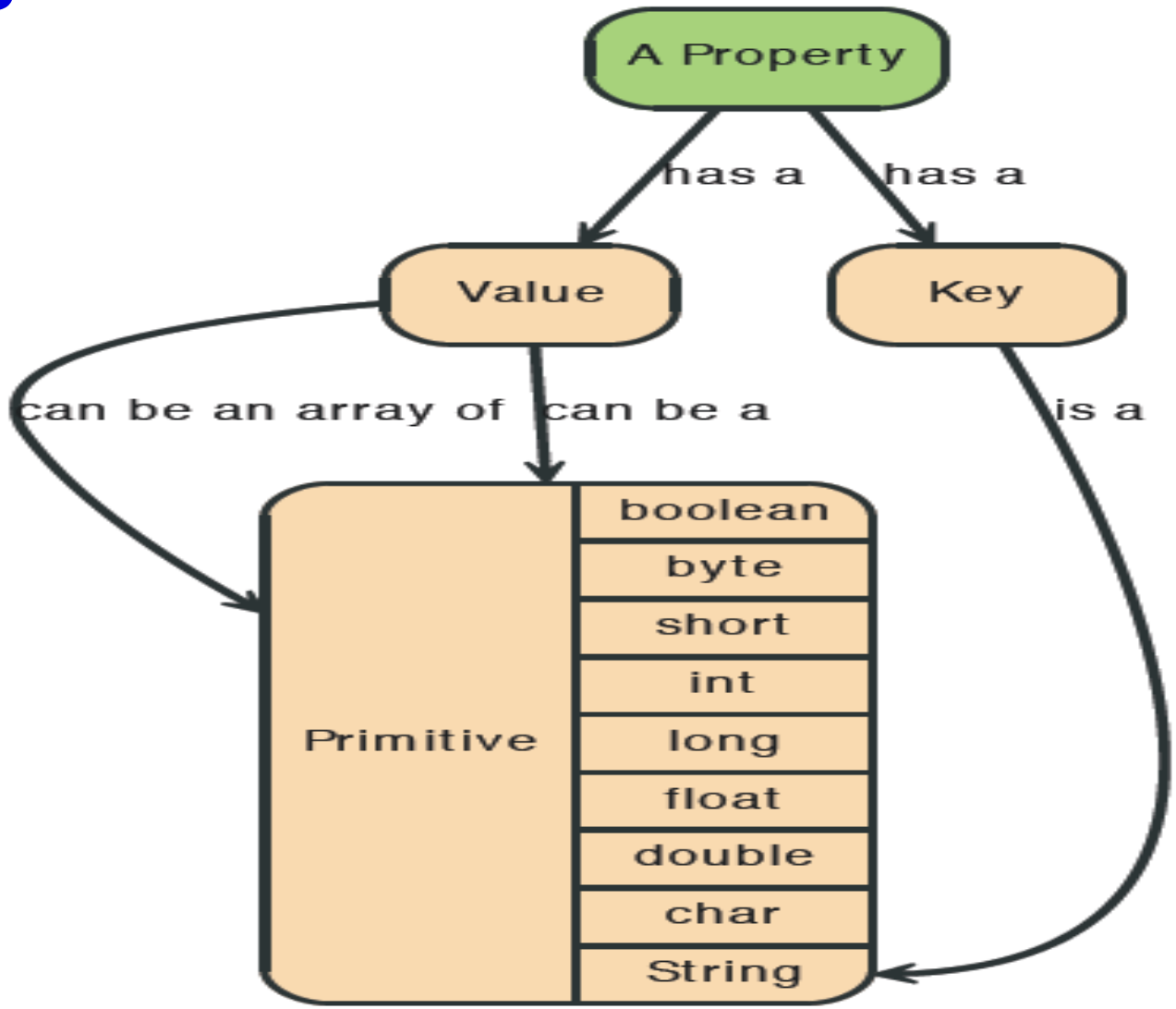
MUNI ÚVT

Properties

- **Both nodes and relationships can have properties**
- **Properties are key-value pairs where the key is a string**
- **Property values can be either a primitive or an array of one primitive type**

  For example String, int and int[] values are valid for properties

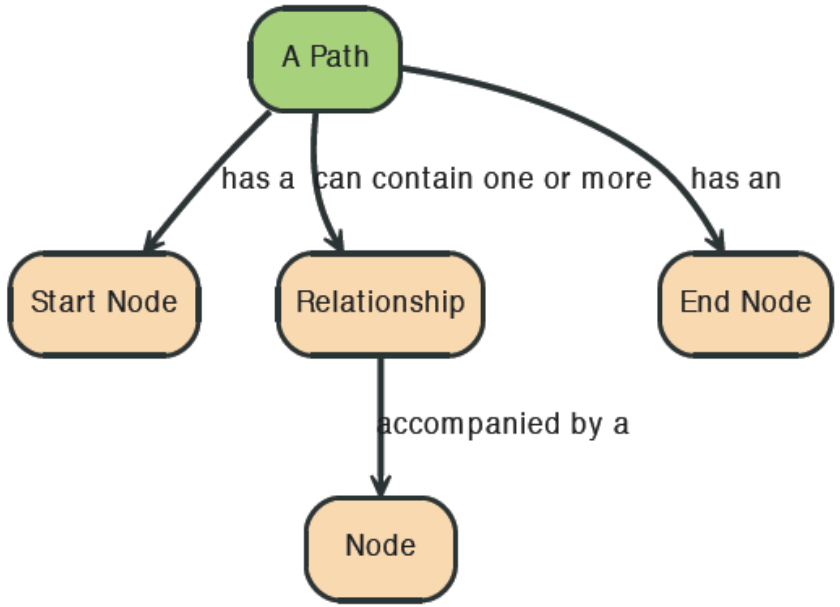
Properties

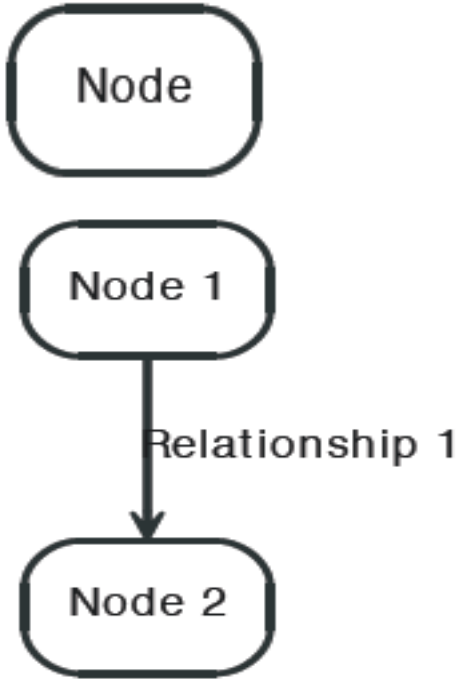


MUNI ÚVT

Paths in Neo4j

- **A path is one or more nodes with connecting relationships, typically retrieved as a query or traversal result**

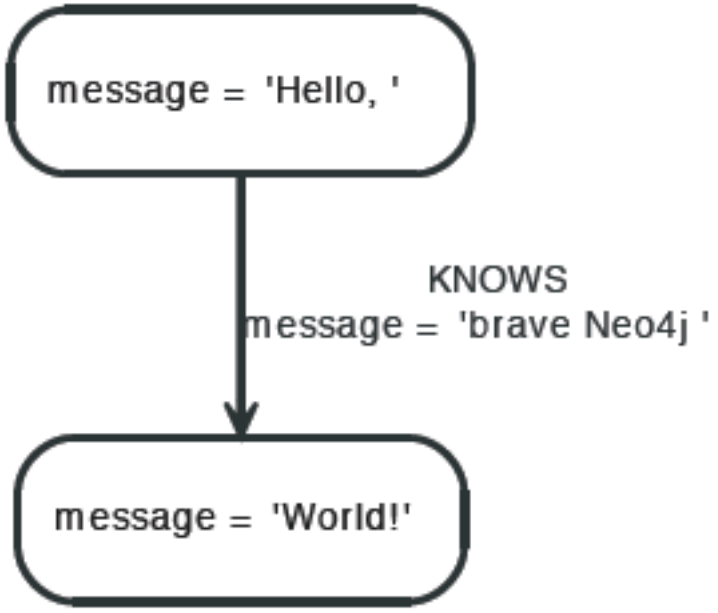




MUNI ÚVT

Creating a small graph

```
firstNode = graphDb.createNode();
firstNode.setProperty( "message", "Hello, " );
secondNode = graphDb.createNode();
secondNode.setProperty( "message", "World!" );

relationship = firstNode.createRelationshipTo( secondNode, RelTypes.KNOWS );
relationship.setProperty( "message", "brave Neo4j " );
```



MUNI ÚVT

Print the data

```
System.out.print( firstNode.getProperty( "message" ) );
System.out.print( relationship.getProperty( "message" ) );
System.out.print( secondNode.getProperty( "message" ) );
```

Remove the data

```
firstNode.getSingleRelationship( RelTypes.KNOWS, Direction.OUTGOING ).delete();
firstNode.delete();
secondNode.delete();
```

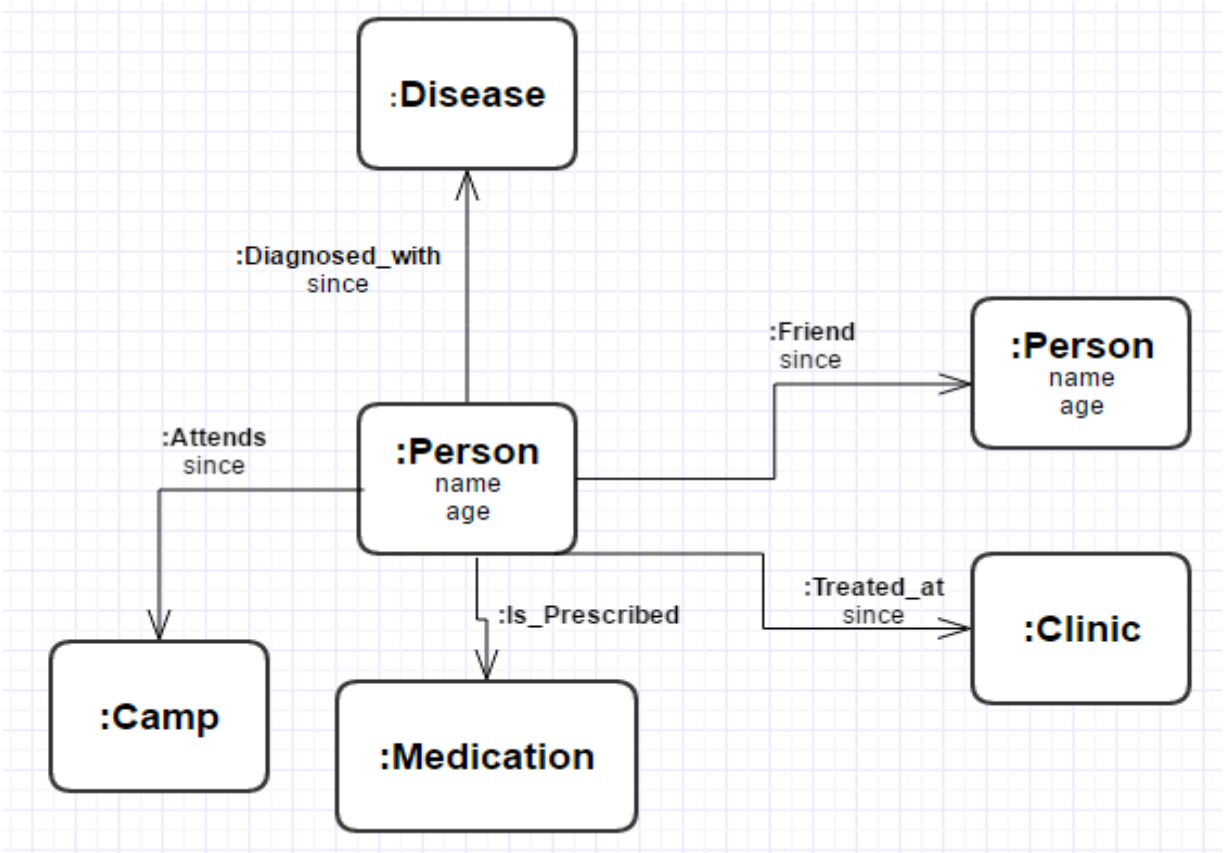
MUNI
ÚVT

Graph DB example

FIND FRIENDS OF FRIENDS THAT HAVE TYPE 1 DIABETES – RDBMS

```
SELECT
    Me.PersonId                    AS MeId,
    Me.Name,
    FriendOfFriend.RelatedPersonId    AS SuggestedFriendId,
    FriendOfAFriend.Name
FROM
    Person         AS Me
INNER JOIN
      PersonRelationship    AS MyFriends
      ON MyFriends.PersonId = Me.PersonId
INNER JOIN
    PersonRelationship    AS FriendOfFriend
    ON MyFriends.RelatedPersonId = FriendOfFriend.PersonId
INNER JOIN
    Person AS FriendOfAFriend
    ON FriendOfFriend.RelatedPersonId = FriendOfAFriend.PersonId
LEFT JOIN
      PersonRelationship    AS FriendsWithMe
      ON  Me.PersonId = FriendsWithMe.PersonId
      AND FriendOfFriend.RelatedPersonId = FriendsWithMe.RelatedPersonId
INNER JOIN
  PersonDisease
  ON PersonDisease.PersonId = FriendOfAFriend.PersonId
WHERE
    FriendsWithMe.PersonId IS NULL
AND Me.PersonId <> FriendOfFriend.RelatedPersonId
AND Me.Name = 'Bill'
AND PersonDisease.DiseaseId = 1
```

NEO4J MODEL



MUNI ÚVT

FIND FRIENDS OF FRIENDS THAT HAVE TYPE 1 DIABETES – GRAPHDB

```
MATCH (user:Person {name:'Bill'})-[:FRIENDS_WITH*2..5]->(fof)-
[:DIAGNOSED_WITH]->(disease)
return fof
```


thank you
danke
謝謝
gracias
merci
obrigado
bedankt
dziękuję
go raibh maith agat
terima kasih
kop khun krap
sukriya
teşekkür ederim
ngiyabonga
спасибо
рахмат
Баярлалаа
dank je
tapadh leat
mochchakkeram
dakujem
arigatô
grazie
takk
taiku
sagolun
hvala
mauruuru
dhanyavad
kiitos
dankie
faafetai lava
blagodaram
matondo
misaotra
paldies
grazzi
mahalo
barka
welalin
tack
mersi
spas
vinaka
спасибі
akun
dankon
ačiū
köszönöm
nanni
nandri
bayarlalaa
grâçie
enkosi
djiere dieuf
tau
asante
manana
obrigada
tenki
murakoze
chokrane
хвала
дякую
mamnun
chnorakaloutioun
gratias ago
gràcies
sulpáy
sobodi
dēkuji
mèsi
didi madloba
kam sah hamnida
rahmat
najis tuke
ありがとう
tanemirt
rahmet
diolch
dhanyavadagalu
shukriya
merci
мерси
trugarez
ευχαριστώ
xièxie
감사합니다
তোমাকে ধন্যবাদ

MUNI ÚVT

MUNI

ÚVT