FONDAZIONE BRUNO KESSLER

NewsReader
POST HOC ERGO PROPTER HOC

SEVENTH FRAMEWORK PROGRAMME

RDF$_{pro}$ an Extensible Tool for Building Stream-Oriented RDF Processing Pipelines

Riva del Garda, 19 October 2014

Marco Rospocher[1], Marco Amadori[2], Michele Mostarda[2], Francesco Corcoglioniti

[1] Data and Knowledge Management Unit, FBK-Irst, http://dkm.fbk.eu/
[2] Web of Data Unit, FBK-Irst http://wod.fbk.eu/

http://fracor.bitbucket.org/rdfpro

The problem

FONDAZIONE BRUNO KESSLER

perform simple RDF processing tasks

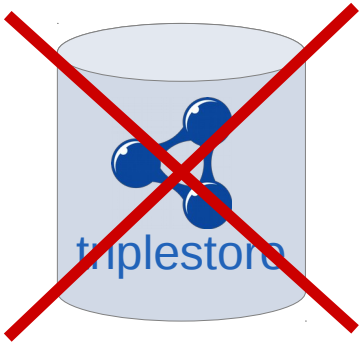
- filtering and transformation (quad-level)
- basic inference (RDFS)
- dataset merging → deduplication, owl:sameAs **smushing**
- simple statistics extraction (VOID+)
- ...

on large datasets

- LOD-sized: 100M+ triples
- quads, not just triples

on a single commodity machine

- no cluster / distributed computing
- no triplestore or other data index



The solution

FONDAZIONE BRUNO KESSLER

RDF$_{pro}$

pro = processor (and not 'professional'!)

~ Java command line tool ~

~ embeddable Java library ~

~ public domain code ~

http://fracor.bitbucket.org/rdfpro/

RDF$_{pro}$ ingredients

FONDAZIONE BRUNO KESSLER

① streaming

realized via the RDF processor abstraction

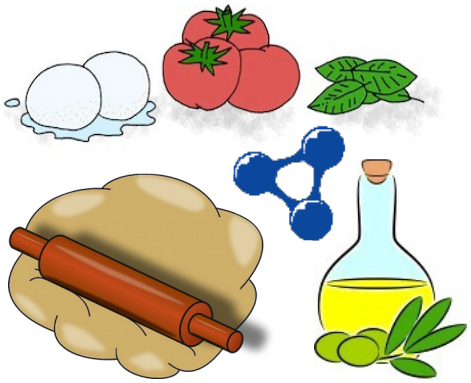
invocation syntax: rdfpro @P args

pro:

- natural model for many tasks
- O(n) time complexity
  → fast, also due to sequential data access
- O(1) space complexity (usually)
  → copes with arbitrarily large datasets

cons:

- restrictive model!



RDF$_{pro}$ ingredients

FONDAZIONE BRUNO KESSLER

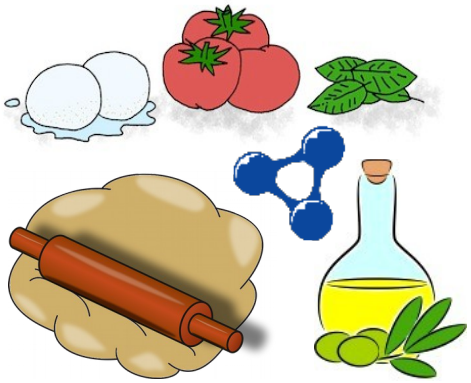
① streaming

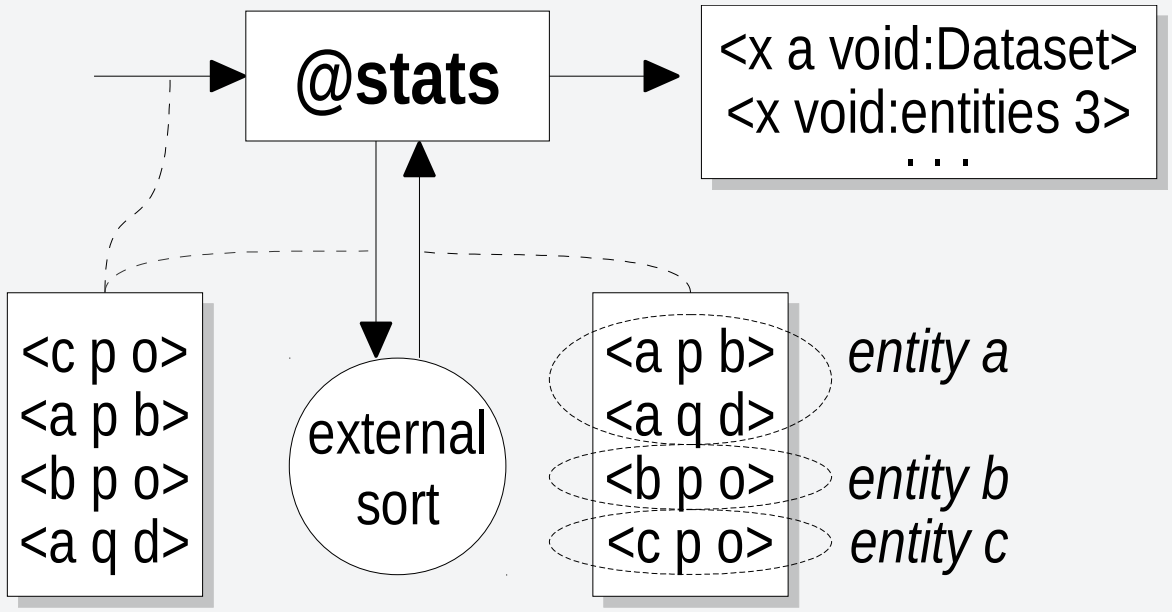
② sorting

realized via external sorting (sort utility)

allows tasks not doable with pure streaming

- duplicate removal
- set operations (quad union, intersection, diff.)
- VOID statistics extraction
- ...

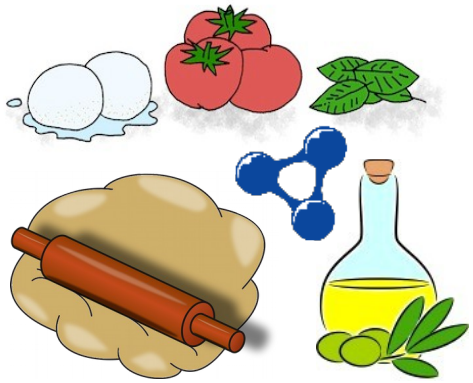




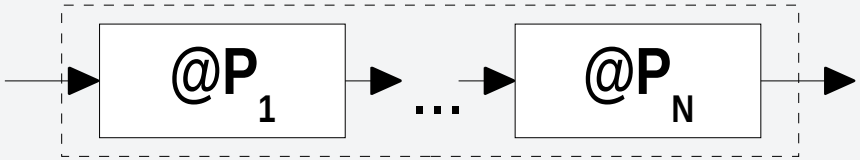
RDF$_{pro}$ ingredients

FONDAZIONE BRUNO KESSLER

1. streaming
2. sorting
3. pipelining

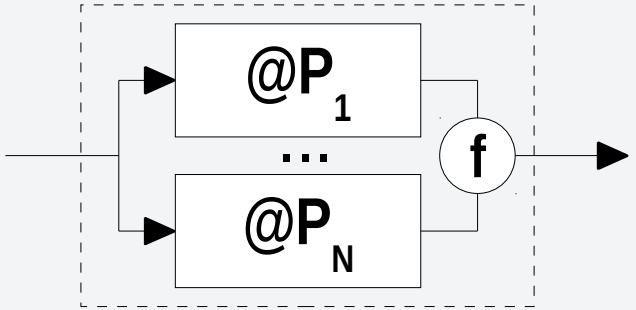


1 sequence composition



rdfpro @P$_1$ args$_1$ ... @P$_N$ args$_N$

2 parallel composition



rdfpro { @P$_1$ args$_1$, ... , @P$_N$ args$_N$ }f

pro:

- reduced I/O costs (less temporary files)
- reduced execution time (parallelism)

RDF$_{pro}$ ingredients

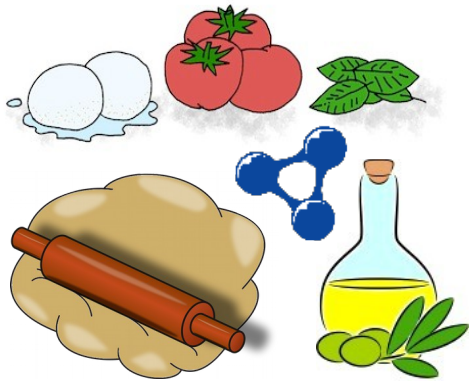
FONDAZIONE BRUNO KESSLER

① streaming

② sorting

③ pipelining

④ multi-threading



① inter-processor parallelism

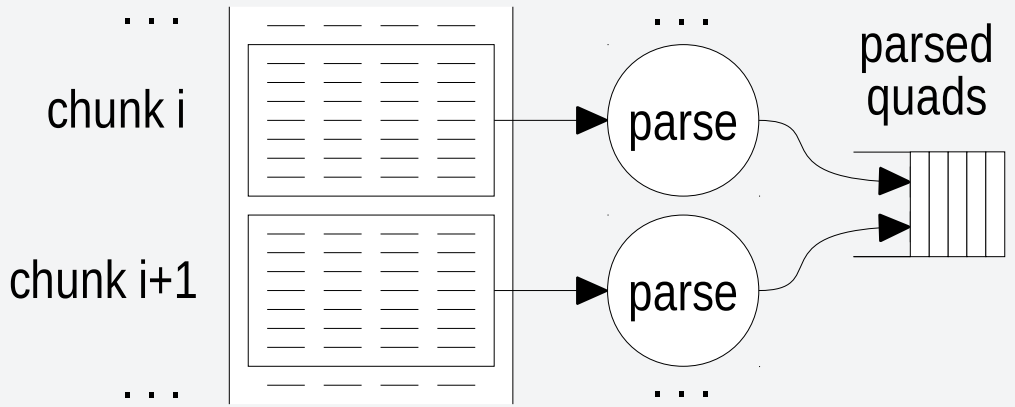
- multiple processors run in parallel

② intra-processor parallelism

- handleStatement() called concurrently

③ I/O parallelism

- multiple files read/written in parallel
- single files split in chunks processed in parallel (line-oriented RDF formats only)



Putting all together, you can ...

FONDAZIONE BRUNO KESSLER

move data around

- @read / @write files
- @download from / @upload to SPARQL endpoints

transform data

- general purpose data @transform using Groovy
- @infer the RDFS closure
- @smush data, replacing URI aliases with canonical URIs
- extract @tbox and VOID @stats

compose these tasks freely

- also via set operations

A simple use case

FONDAZIONE BRUNO KESSLER

integrate:

- Freebase (2014/07/10 dump, 2623 MQuads)
- GeoNames (2013/08/27 dump 125 MQuads)
- DBpedia EN, ES, IT, NL (subset of ver. 3.9, 271 MQuads)

performing:

- filtering (remove redundant quads & quads in unwanted languages)
- smushing (based on owl:sameAs links in DBpedia)
- inference (excluding <X rdf:type rdfs:Resource> stuff)
- statistics extraction (VOID with class & property partitions)

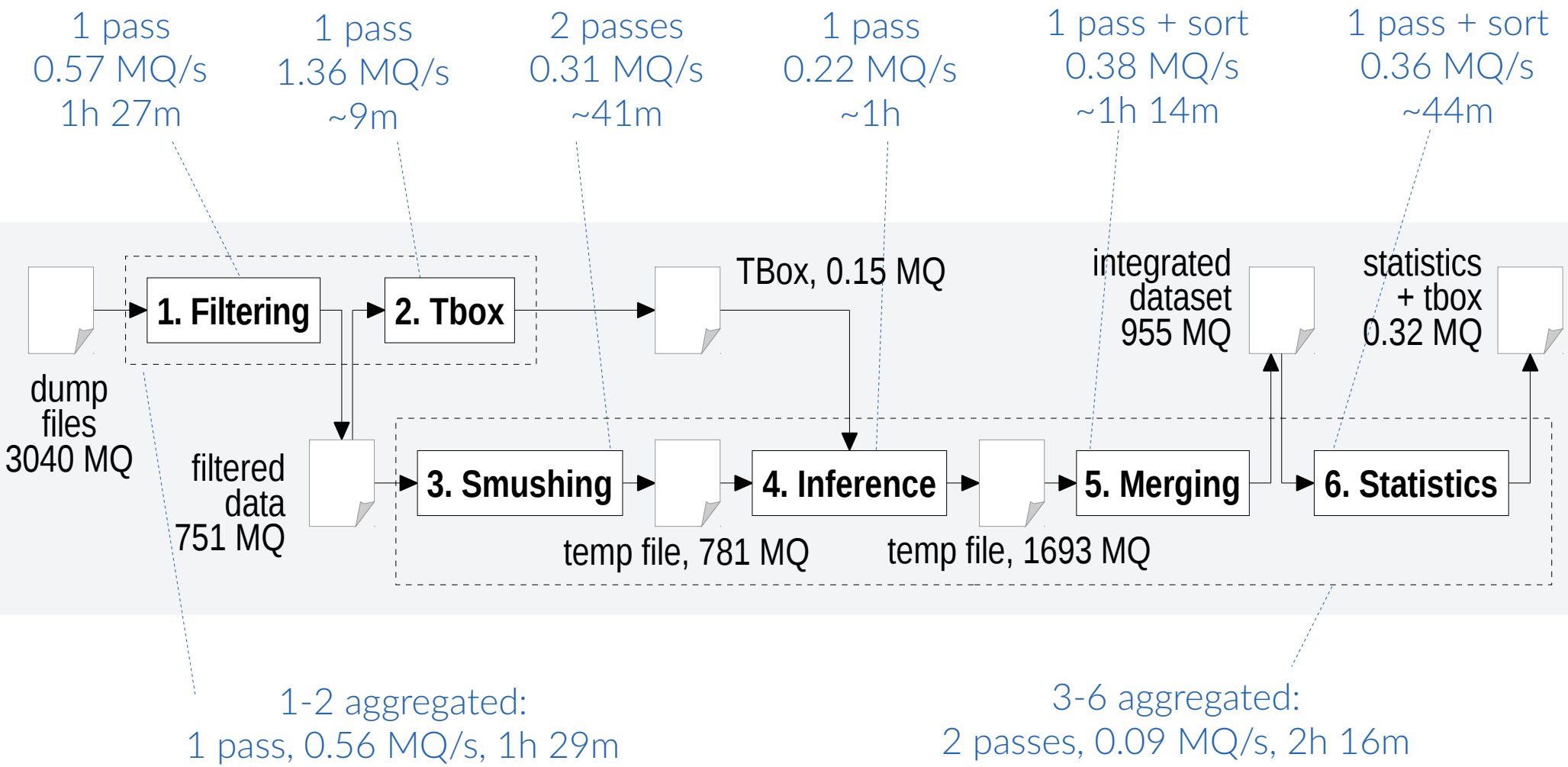
using:

- a small workstation (I7 860, 16 GB ram, 500 GB 7200 rpm hd)
- RDF_{pro} + parallel sort + pigz + pbzip2

A simple use case

FONDAZIONE BRUNO KESSLER

tasks performed individually - 5h 16m total



1-2 aggregated:
1 pass, 0.56 MQ/s, 1h 29m

3-6 aggregated:
2 passes, 0.09 MQ/s, 2h 16m

aggregated tasks – 3h 46m total (-28%)

A simple use case

FONDAZIONE BRUNO KESSLER

individual tasks

Task	Input size		Output size		Throughput		Time
	[MQuad]	[GB]	[MQuad]	[GB]	[MQuad/s]	[MB/s]	[hh:mm:ss]
1. Filtering	3019.89	29.31	750.78	9.68	0.57	5.70	1:27:46
2. TBox extraction	750.78	9.68	0.15	0.01	1.36	18.00	9:11
3. Smushing	750.78	9.68	780.86	10.33	0.31	4.04	40:53
4. Inference	781.01	10.34	1693.59	15.56	0.22	2.91	1:00:30
5. Deduplication	1693.59	15.56	954.91	7.77	0.38	3.61	1:13:33
6. Statistics	954.91	7.77	0.32	0.01	0.36	3.02	44:00
whole processing	3019.89	29.31	955.23	7.78	0.16	1.58	5:15:53

aggregated tasks

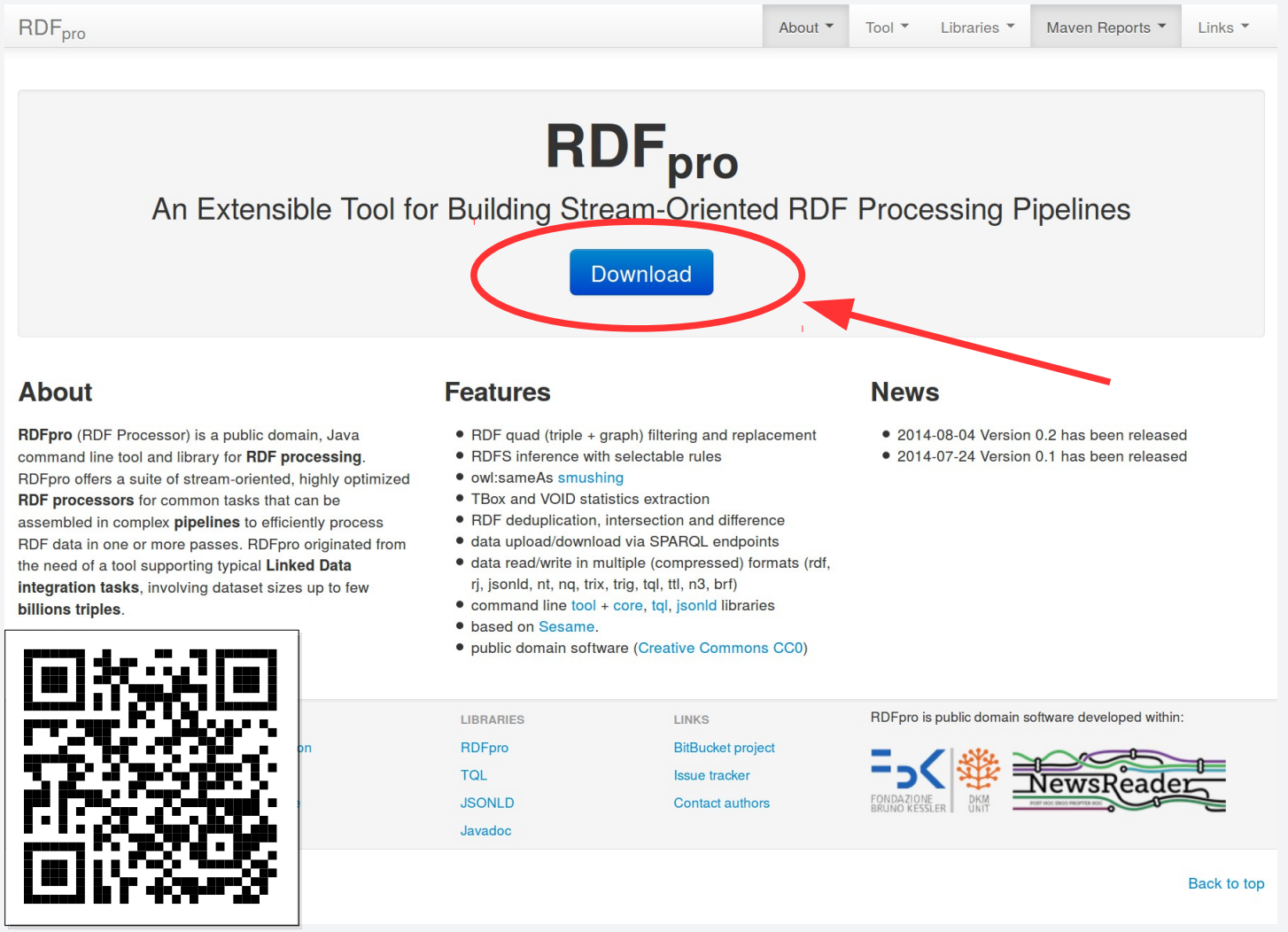
Task	Input size		Output size		Throughput		Time
	[MQuad]	[GB]	[MQuad]	[GB]	[Mquad/s]	[MB/s]	[hh:mm:ss]
1-2 aggregated	3019.89	29.31	750.92	9.69	0.56	5.60	1:29:23
3-6 aggregated	750.92	9.69	955.23	7.78	0.09	1.21	2:16:08
whole processing	3019.89	29.31	955.23	7.78	0.22	2.22	3:45:31

RDF$_{pro}$ cookbook

FONDAZIONE BRUNO KESSLER

① download

http://fracor.bitbucket.org/rdfpro (or Google for it!)



RDF$_{pro}$ cookbook

FONDAZIONE BRUNO KESSLER

① download

② install

check requirements:

- Java 1.7+ (Oracle, OpenJDK, whatever)
- gzip, bzip2, sort utilities available on PATH

extract the download tarball:

```
$ tar tf rdfpro-0.3.tar.gz
```

check that everything works:

```
$ cd rdfpro
$ ./rdfpro -v
RDF Processor Tool (RDFpro) 0.3
Java 64 bit (Oracle Corporation) 1.7.0_67
This is free software released into the public domain
```

suggestions:

- add rdfpro directory to PATH
- install and configure pigz and pbzip2 (see web site)

RDF$_{pro}$ cookbook

FONDAZIONE BRUNO KESSLER

1. download
2. install
3. try it out!

let's get and process some data from Dbpedia:

```
$ ./rdfpro \
>     @read http://dbpedia.org/resource/Riva_del_Garda \
>           http://it.dbpedia.org/resource/Riva_del_Garda \
>     @smush \
>     @infer http://downloads.dbpedia.org/3.9/dbpedia_3.9.owl.bz2 \
>     @transform "emitIf(t == rdf:type)" \
>     @unique \
>     @write riva_del_garda.ttl.gz
```

FONDAZIONE
BRUNO KESSLER

That's all:
enjoy cooking triples with RDF_{pro} and...
happy eating !!

for any question about the ~~menu~~ RDF_{pro}, contact
Francesco Corcoglioniti <corcoglio@fbk.eu>