NeverBlink

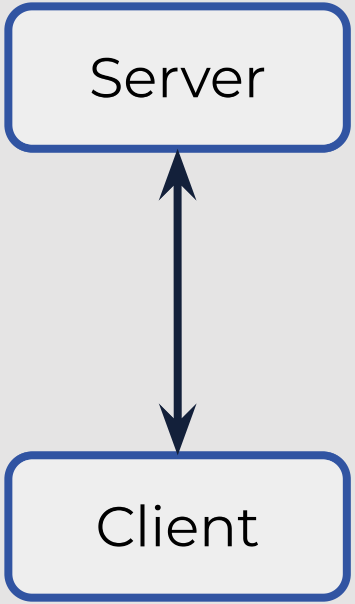
September 5, 2025

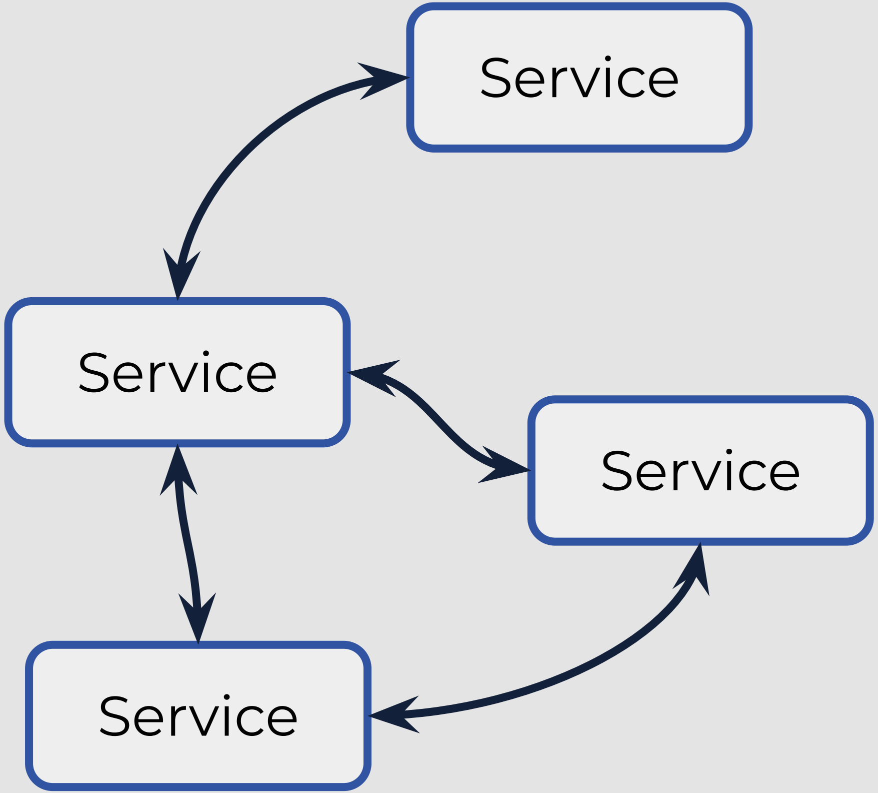
SEMANTiCS, Vienna, Austria

Tackling inter-service RDF communication bottlenecks in the Nanopublication network with Jelly

Piotr Sowiński[1], Tobias Kuhn[2], Karolina Bogacka[1]

[1] *NeverBlink*
[2] *Knowledge Pixels*


Server
Client


Service
Service
Service
Service

Google invented Protobuf to solve it...

But **5%** of their datacenter CPU cycles are still spent on ser/des!

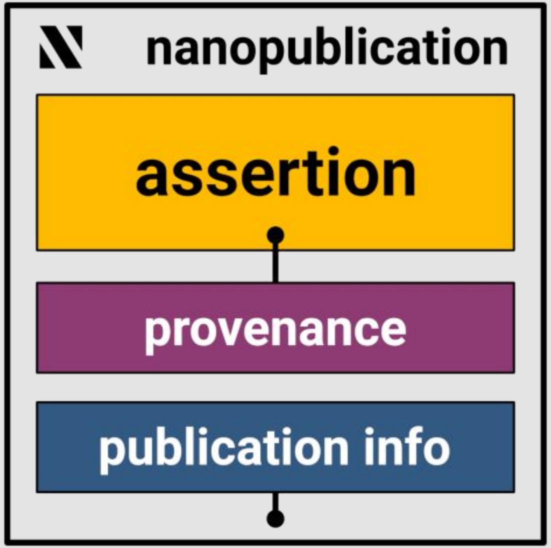
Kanev, S., Darago, J. P., Hazelwood, K., Ranganathan, P., Moseley, T., Wei, G. Y., & Brooks, D. (2015, June). Profiling a warehouse-scale computer. In *Proceedings of the 42nd annual international symposium on computer architecture* (pp. 158-169).

Can your serialization keep up with the rest of the system?

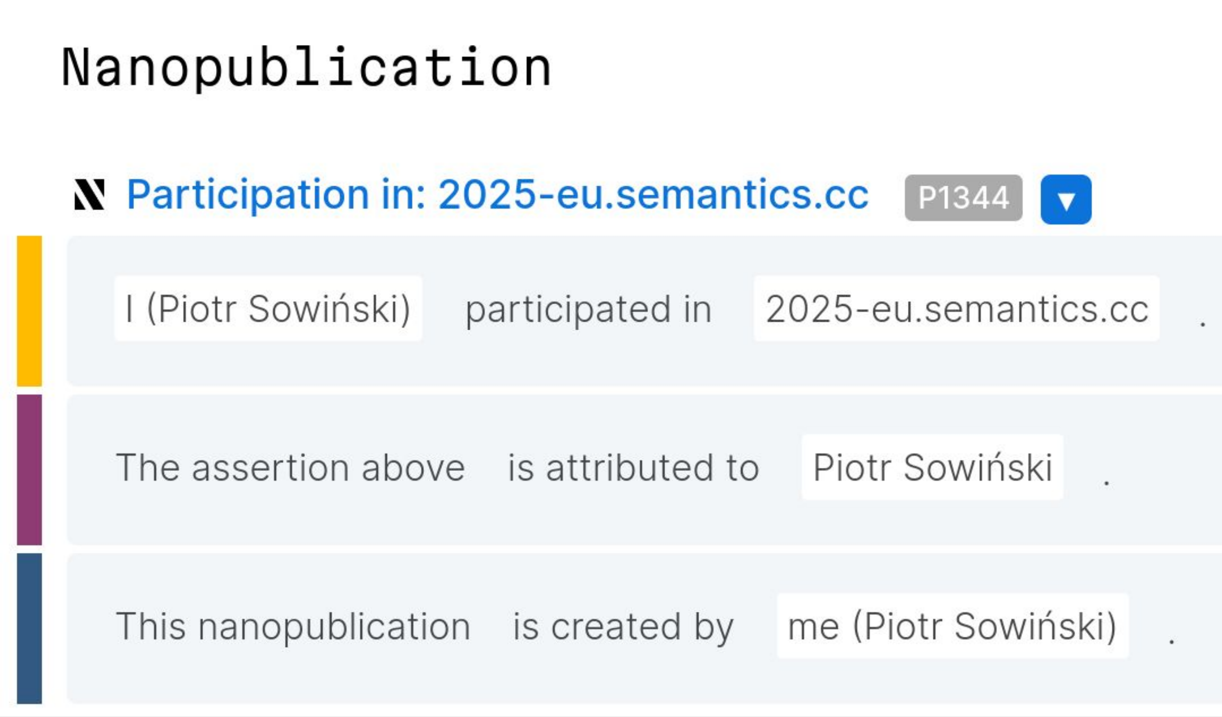
NeverBlink

Nanopublication network

Anatomy of a nanopublication



- FAIR by design
- 1 nanopub = 1 RDF dataset (4 named graphs)
- ~50–200 triples
- Lots of them!



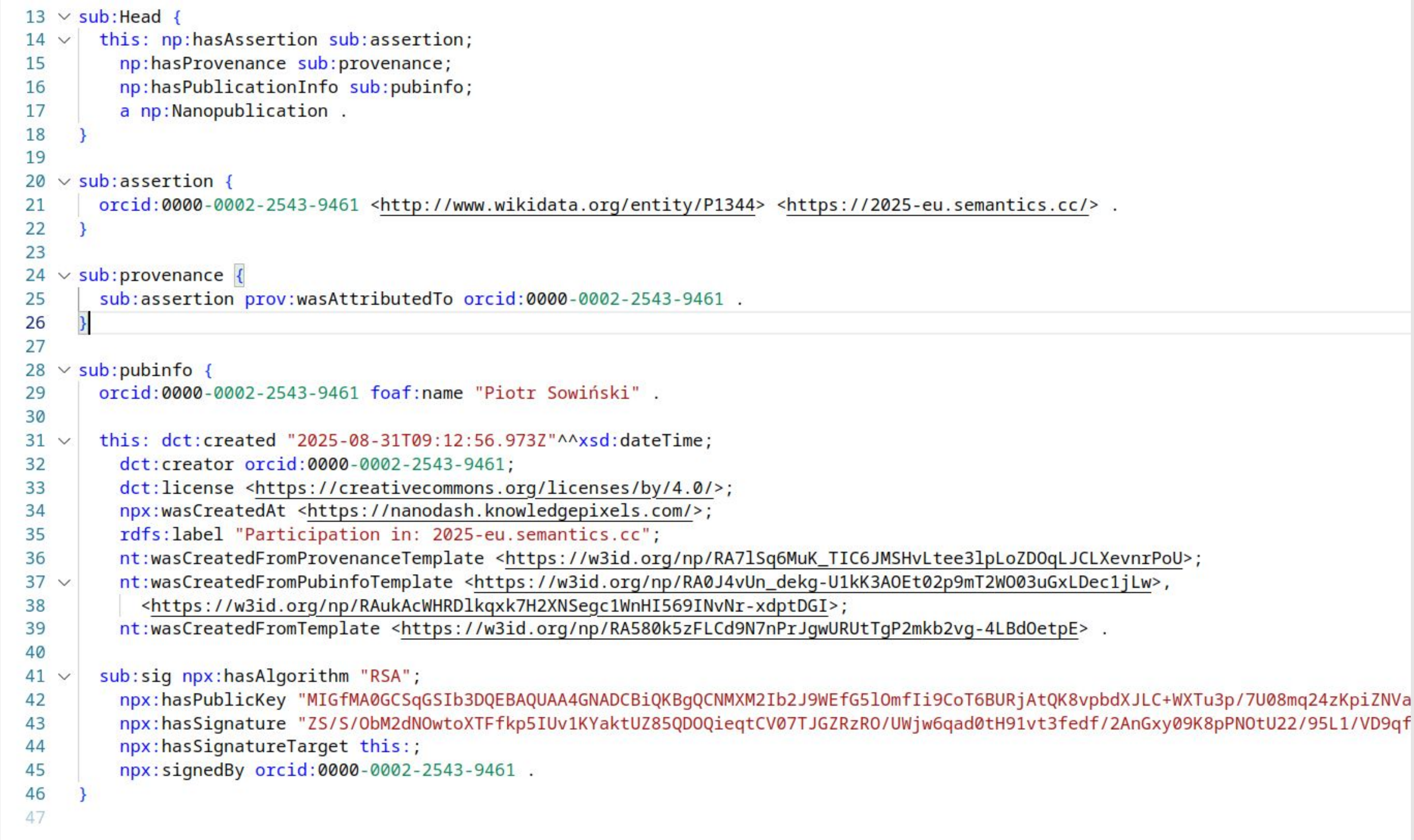
Nanopublication

Participation in: 2025-eu.semantics.cc P1344

I (Piotr Sowiński) participated in 2025-eu.semantics.cc .

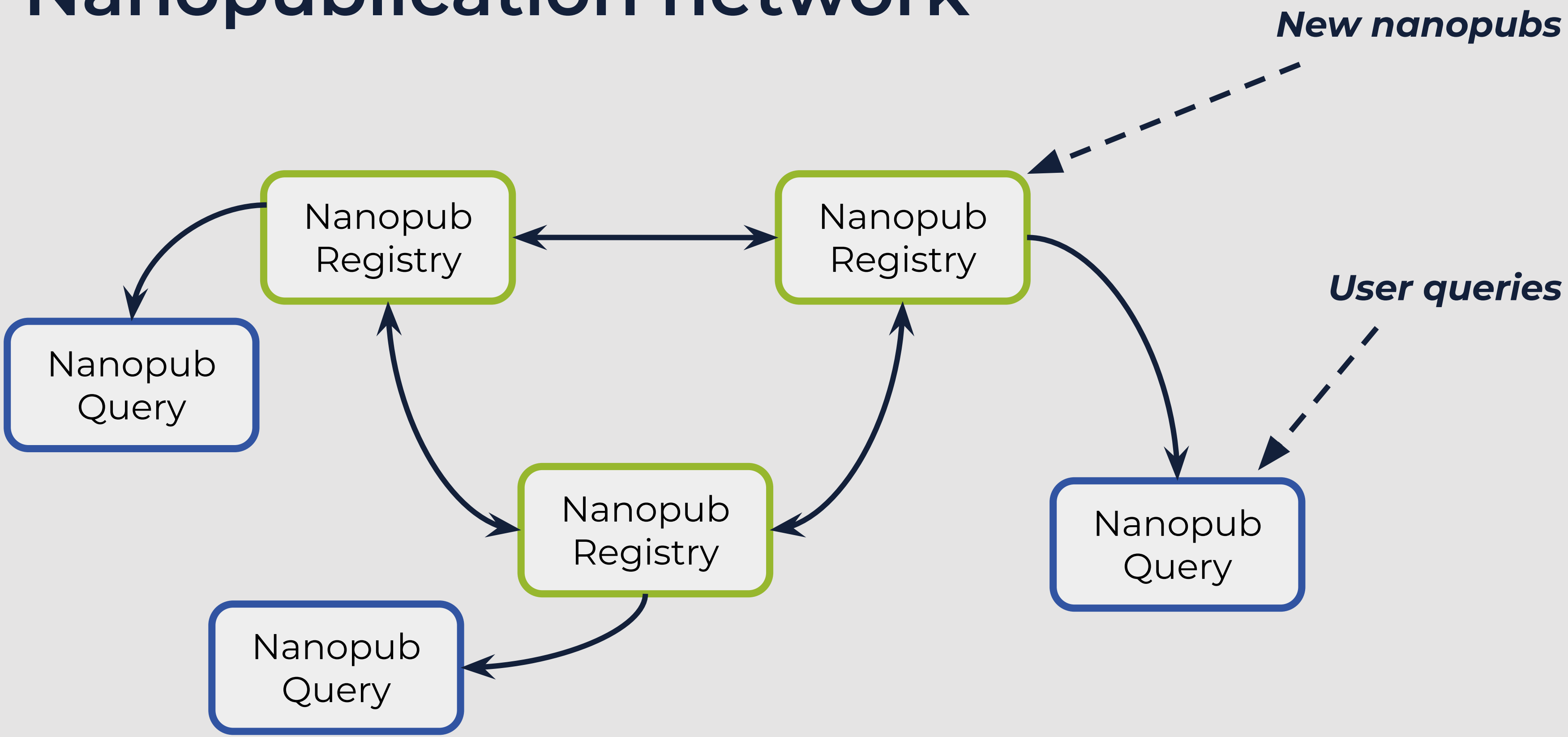
The assertion above is attributed to Piotr Sowiński .

This nanopublication is created by me (Piotr Sowiński) .

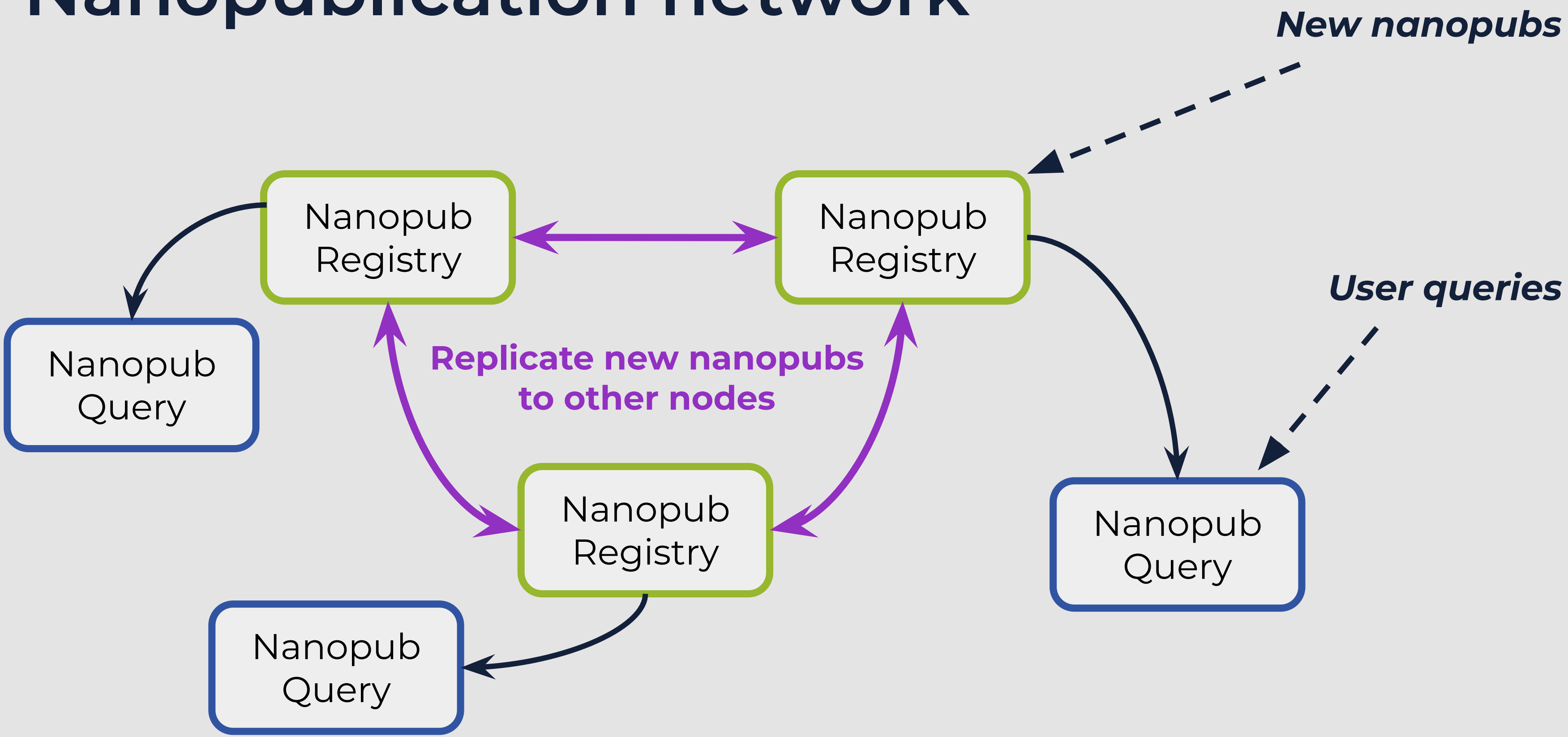


```
sub:Head {
  this: np:hasAssertion sub:assertion;
    np:hasProvenance sub:provenance;
    np:hasPublicationInfo sub:pubinfo;
    a np:Nanopublication .
}

sub:assertion {
  orcid:0000-0002-2543-9461 <http://www.wikidata.org/entity/P1344> <https://2025-eu.semantics.cc/> .
}

sub:provenance {
  sub:assertion prov:wasAttributedTo orcid:0000-0002-2543-9461 .
}

sub:pubinfo {
  orcid:0000-0002-2543-9461 foaf:name "Piotr Sowiński" .

  this: dct:created "2025-08-31T09:12:56.973Z"^^xsd:dateTime;
    dct:creator orcid:0000-0002-2543-9461;
    dct:license <https://creativecommons.org/licenses/by/4.0/>;
    npx:wasCreatedAt <https://nanodash.knowledgepixels.com/>;
    rdfs:label "Participation in: 2025-eu.semantics.cc";
    nt:wasCreatedFromProvenanceTemplate <https://w3id.org/np/RA7lSq6MuK_TIC6JMSHvLtee3lpLoZDOqLJCLXevnrPoU>;
    nt:wasCreatedFromPubinfoTemplate <https://w3id.org/np/RA0J4vUn_dekg-U1kK3AOEt02p9mT2WO03uGxLDec1jLw>,
      <https://w3id.org/np/RAukAcWHRDlkqxk7H2XNSegc1WnHI569INvNr-xdptDGI>;
    nt:wasCreatedFromTemplate <https://w3id.org/np/RA580k5zFLCd9N7nPrJgwURUtTgP2mkb2vg-4LBdOetpE> .

  sub:sig npx:hasAlgorithm "RSA";
    npx:hasPublicKey "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNMXM2Ib2J9WEfG5lOmfIi9CoT6BURjAtQK8vpbdXJLC+WXTu3p/7U08mq24zKpiZNVa
    npx:hasSignature "ZS/S/ObM2dNOwtoXTFfkp5IUv1KYaktUZ85QDOQieqtCV07TJGZRzRO/UWjw6qad0tH91vt3fedf/2AnGxy09K8pPNOtU22/95L1/VD9qf
    npx:hasSignatureTarget this:;
    npx:signedBy orcid:0000-0002-2543-9461 .
}
```

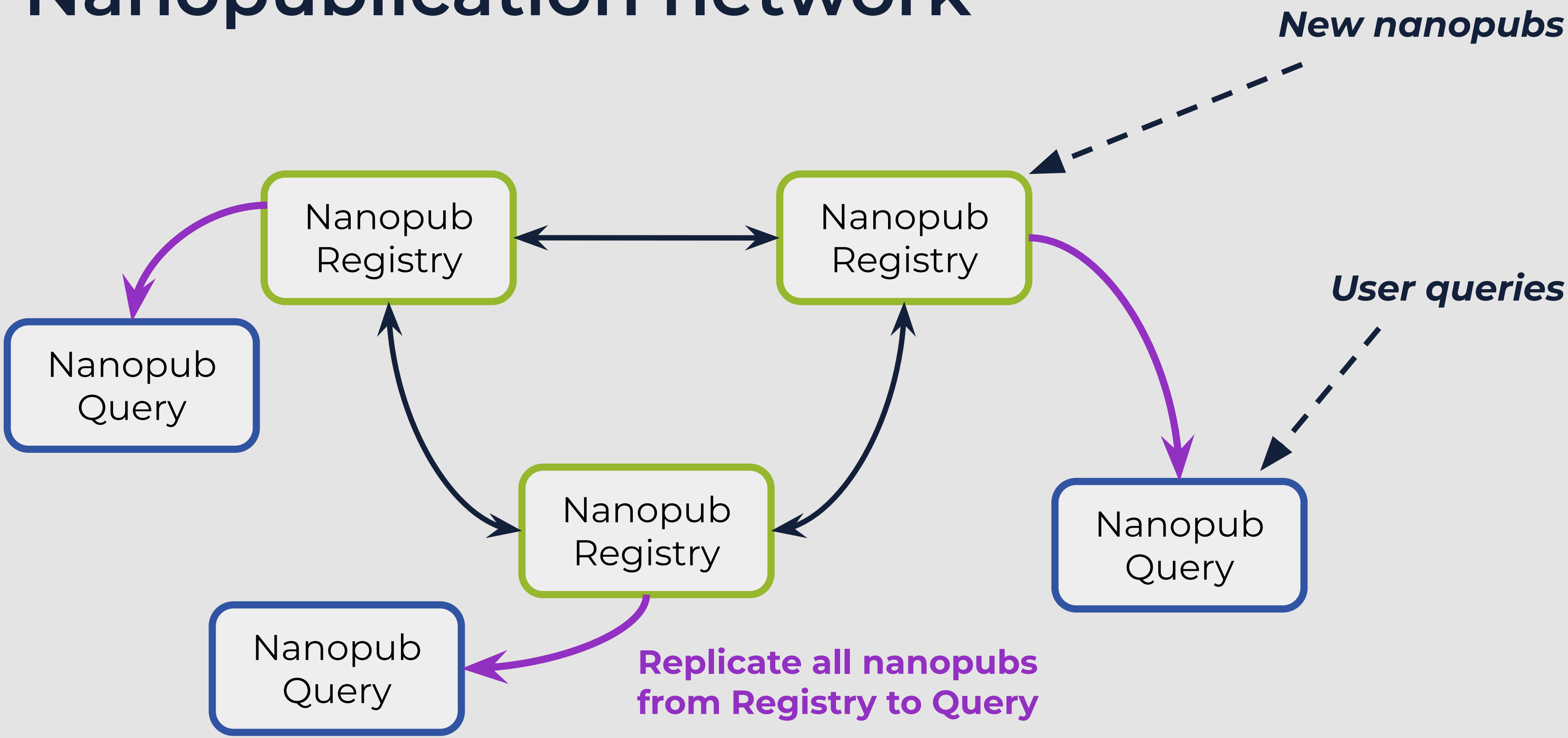
Nanopublication network



Nanopublication network

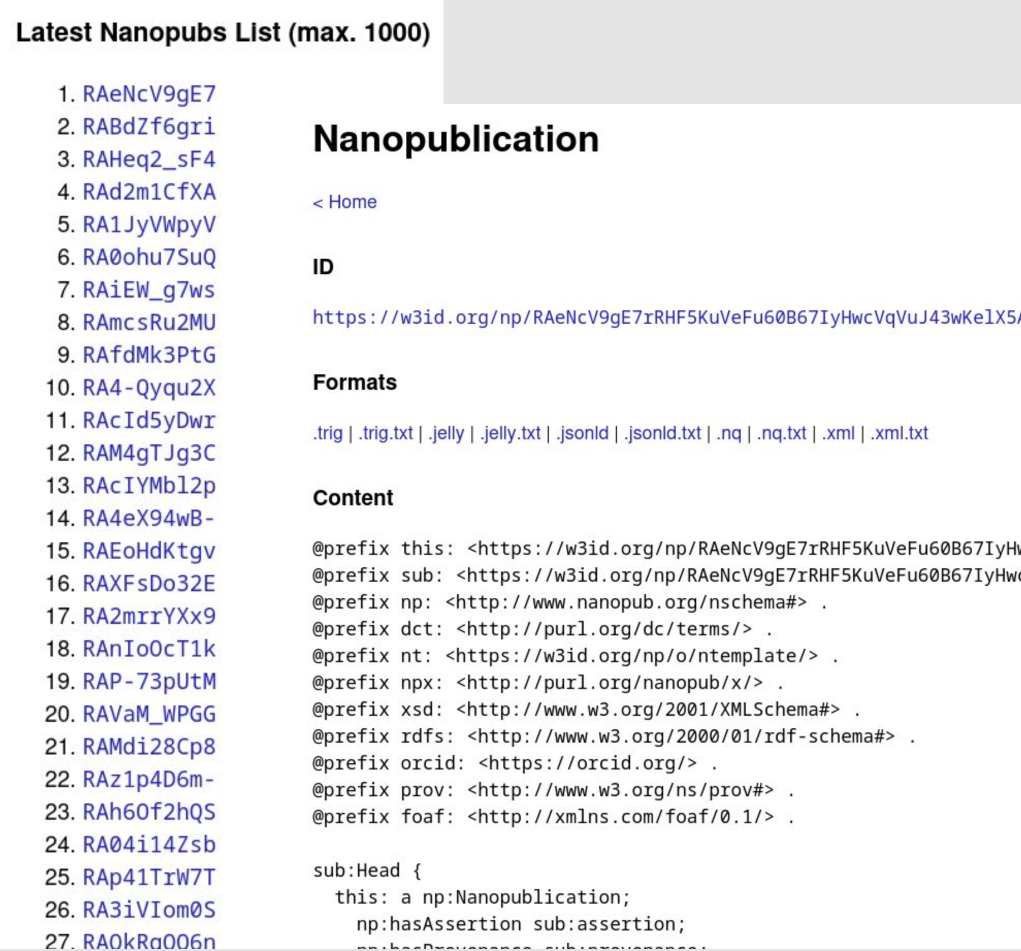


Nanopublication network



Starting situation

- HTML / JSON list pages with links to individual nanopubs
- Individual nanopubs served as TriG files
- Accessing 60k nanopubs = 60k+ HTTP requests



Latest Nanopubs List (max. 1000)

1. RAeNcV9gE7
2. RABdZf6gri
3. RAHeq2_sF4
4. RAd2m1CfXA
5. RA1JyVWpyV
6. RA0ohu7SuQ
7. RAiEW_g7ws
8. RAmcsRu2MU
9. RAfdMk3PtG
10. RA4-Qyqu2X
11. RAcId5yDwr
12. RAM4gTJg3C
13. RAcIYMbl2p
14. RA4eX94wB-
15. RAEoHdKtgv
16. RAXFsDo32E
17. RA2mrrYXx9
18. RAnIoOcT1k
19. RAP-73pUtM
20. RAVaM_WPGG
21. RAMdi28Cp8
22. RAz1p4D6m-
23. RAh6Of2hQS
24. RA04i14Zsb
25. RAp41TrW7T
26. RA3iVIom0S
27. RAOkRgOO6n

Nanopublication

< Home

ID

https://w3id.org/np/RAeNcV9gE7rRHF5KuVeFu60B67IyHwcVqVuJ43wKelX5

Formats

.trig | .trig.txt | .jelly | .jelly.txt | .jsonld | .jsonld.txt | .nq | .nq.txt | .xml | .xml.txt

Content

```
@prefix this: <https://w3id.org/np/RAeNcV9gE7rRHF5KuVeFu60B67IyHw
@prefix sub: <https://w3id.org/np/RAeNcV9gE7rRHF5KuVeFu60B67IyHwc
@prefix np: <http://www.nanopub.org/nschema#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix nt: <https://w3id.org/np/o/ntemplate/> .
@prefix npx: <http://purl.org/nanopub/x/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix orcid: <https://orcid.org/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

sub:Head {
  this: a np:Nanopublication;
    np:hasAssertion sub:assertion;
    np:hasProvenance sub:provenance;
```

Starting situation – issues

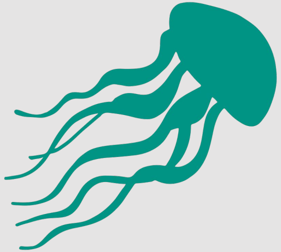
- TriG format is very slow to parse
- Repeated HTTP requests add a lot of overhead

The result:

- Very slow replication throughput
- Additional latency (1 round-trip for list, then 1 for nanopub)

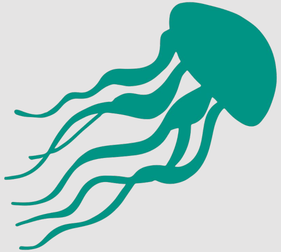
Solution: Jelly

Jelly in a nutshell



- Binary RDF format based on Protobuf
- 100% open spec & open source (https://w3id.org/jelly)
- Very fast to write (**2x** faster than N-Triples in Jena)
- Very, *very* fast to read (**12x** faster than N-Triples)
- Reasonably well-compressed (**6x** smaller than N-Triples)

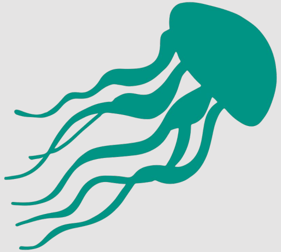
Jelly in a nutshell



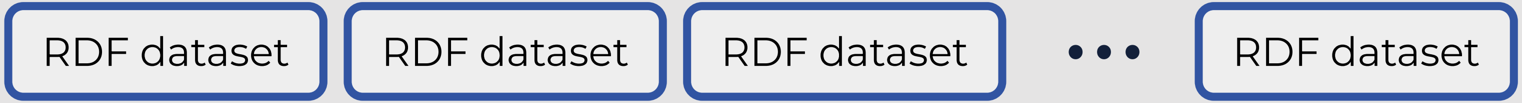
Works with:

- Java (Apache Jena, RDF4J, Titanium)
- Python (RDFLib or no library)
- *Rust (Sophia)* – experimental, community-led
- Neo4j
- CLI application

How does Jelly work?



- Lightweight streaming compression algorithm
- For ***n*** triples:
  - O(1) memory complexity
  - O(n) time complexity
- Max supported triple count = **∞**
- 1 file can contain 1 RDF document (graph or dataset)…
- …or 1 file can contain **many** RDF documents (**!**)



RDF dataset

RDF dataset

RDF dataset

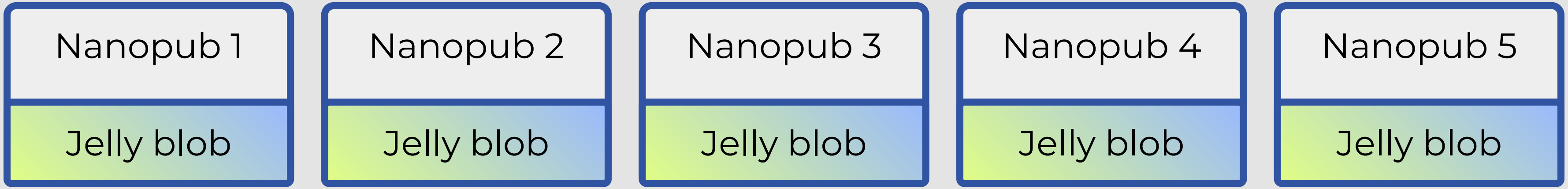
• • •

RDF dataset

Clients

Nanopub Registry API app (Java)

Nanopub Registry DB (MongoDB)



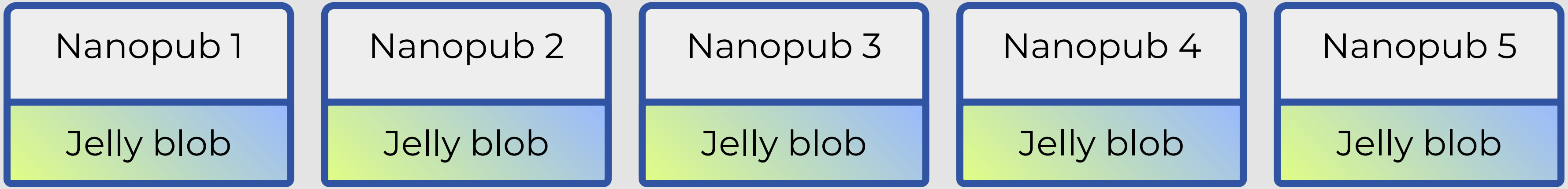
Nanopub 1	Nanopub 2	Nanopub 3	Nanopub 4	Nanopub 5
Jelly blob	Jelly blob	Jelly blob	Jelly blob	Jelly blob

Clients

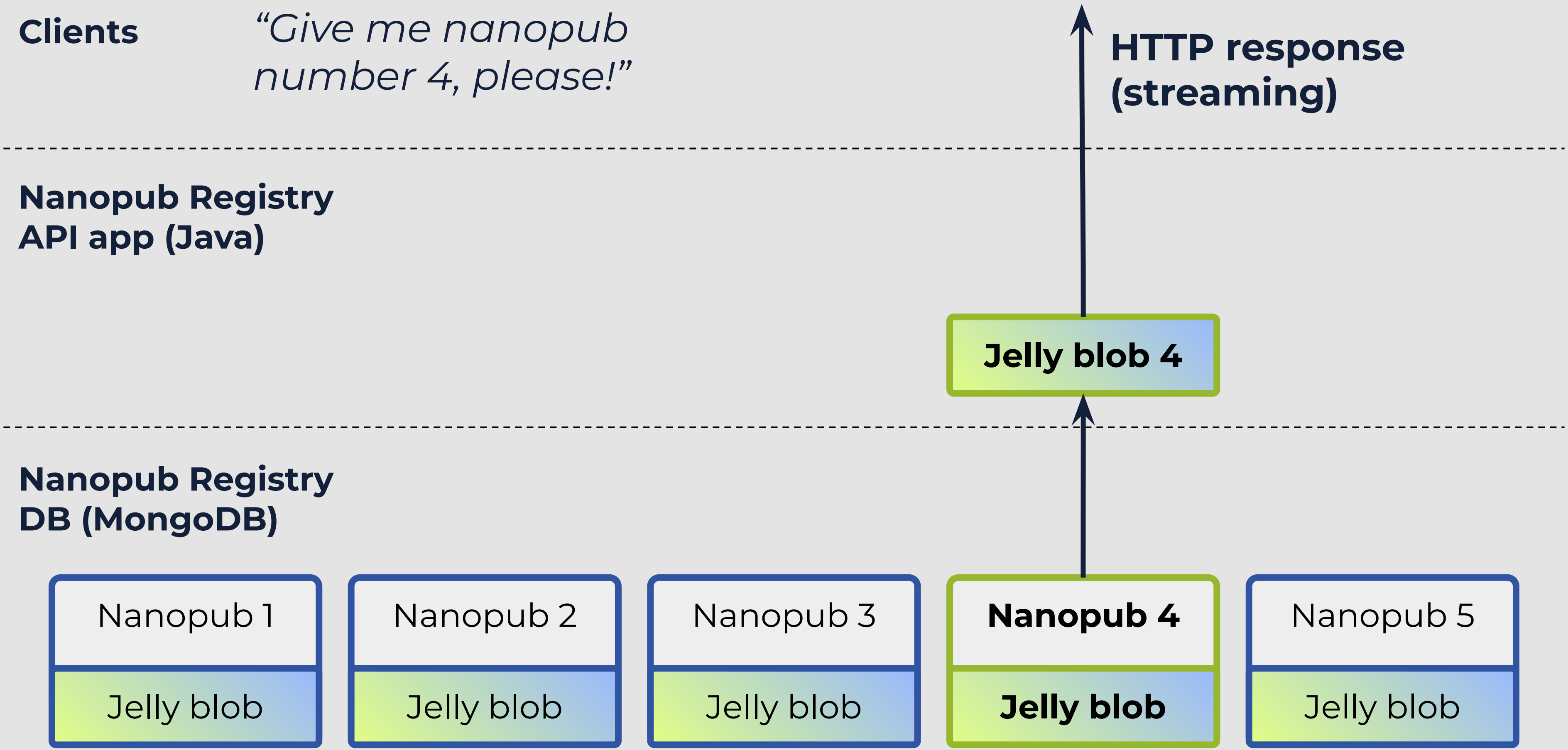
“Give me nanopub number 4, please!”

Nanopub Registry API app (Java)

Nanopub Registry DB (MongoDB)



Nanopub 1	Nanopub 2	Nanopub 3	Nanopub 4	Nanopub 5
Jelly blob	Jelly blob	Jelly blob	Jelly blob	Jelly blob



Clients

“Give me nanopub number 4, please!”

HTTP response (streaming)

Nanopub Registry API app (Java)

Jelly blob 4

Nanopub Registry DB (MongoDB)

Nanopub 1	Nanopub 2	Nanopub 3	**Nanopub 4**	Nanopub 5
Jelly blob	Jelly blob	Jelly blob	**Jelly blob**	Jelly blob

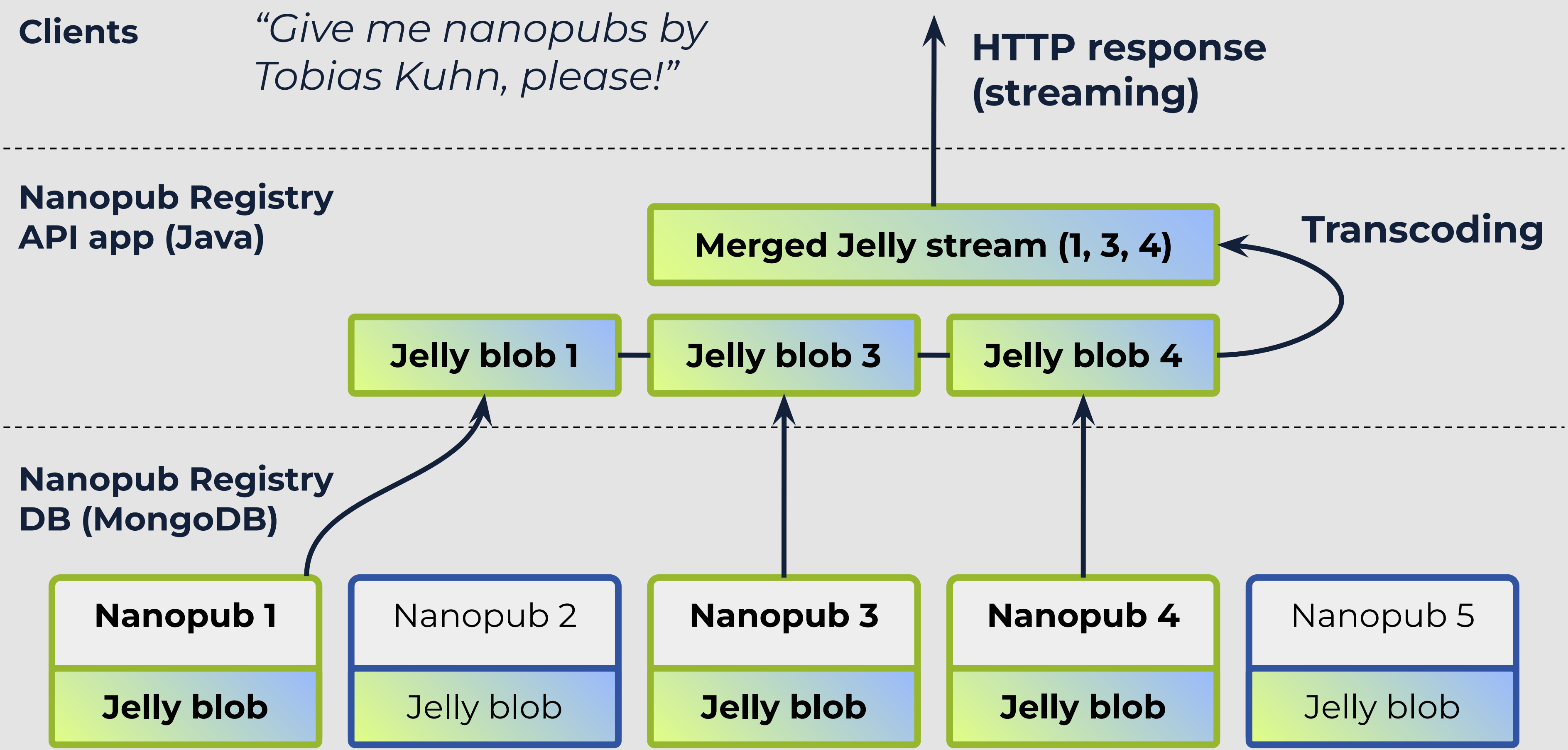
Clients

“Give me nanopubs by
Tobias Kuhn, please!”

Nanopub Registry
API app (Java)

Nanopub Registry
DB (MongoDB)

Nanopub 1	Nanopub 2	**Nanopub 3**	**Nanopub 4**	Nanopub 5
Jelly blob	Jelly blob	**Jelly blob**	**Jelly blob**	Jelly blob



Clients

“Give me nanopubs by Tobias Kuhn, please!”

HTTP response (streaming)

Nanopub Registry API app (Java)

Merged Jelly stream (1, 3, 4)

Transcoding

Jelly blob 1

Jelly blob 3

Jelly blob 4

Nanopub Registry DB (MongoDB)

Nanopub 1

Jelly blob

Nanopub 2

Jelly blob

Nanopub 3

Jelly blob

Nanopub 4

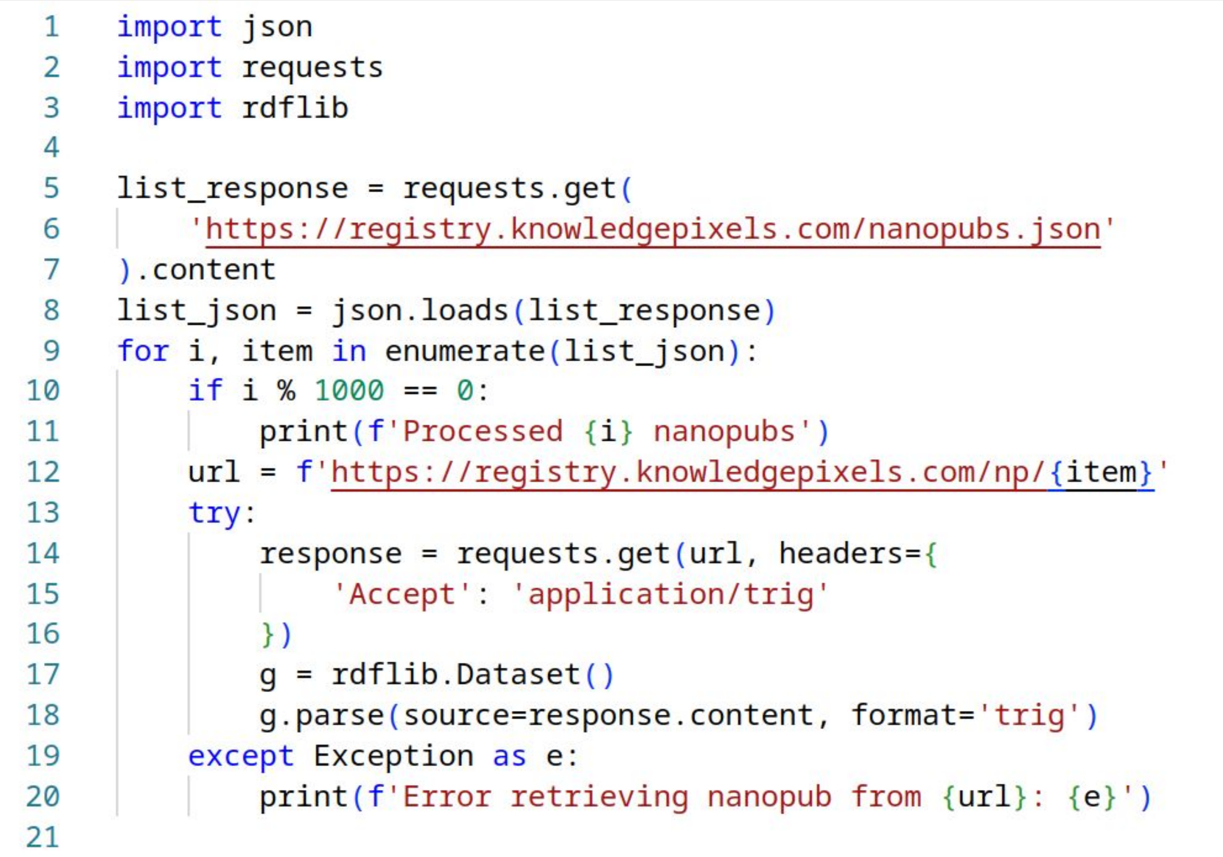
Jelly blob

Nanopub 5

Jelly blob

Results

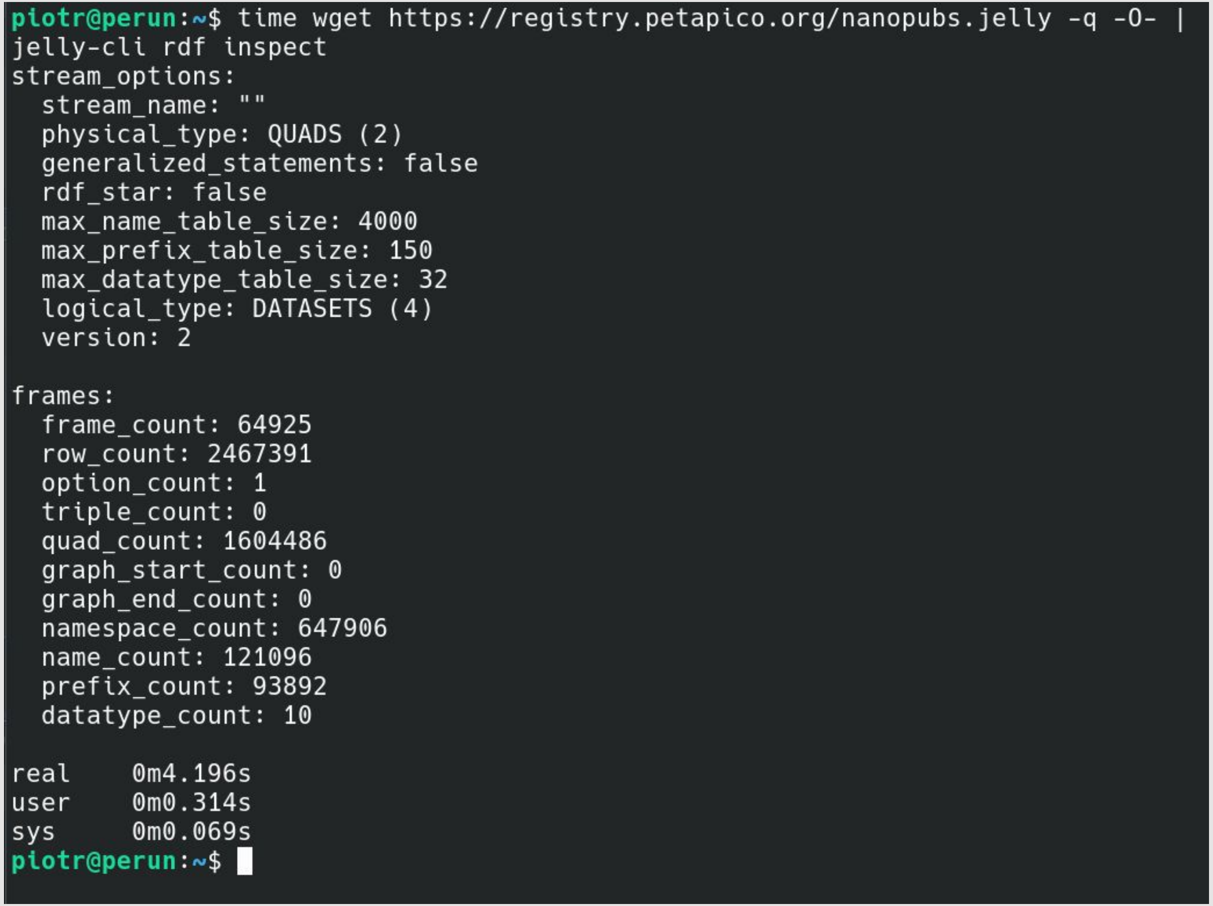
Naïve comparison: original



```python
import json
import requests
import rdflib

list_response = requests.get(
    'https://registry.knowledgepixels.com/nanopubs.json'
).content
list_json = json.loads(list_response)
for i, item in enumerate(list_json):
    if i % 1000 == 0:
        print(f'Processed {i} nanopubs')
    url = f'https://registry.knowledgepixels.com/np/{item}'
    try:
        response = requests.get(url, headers={
            'Accept': 'application/trig'
        })
        g = rdflib.Dataset()
        g.parse(source=response.content, format='trig')
    except Exception as e:
        print(f'Error retrieving nanopub from {url}: {e}')

```

Takes >3 hours to complete.

Naïve comparison: Jelly

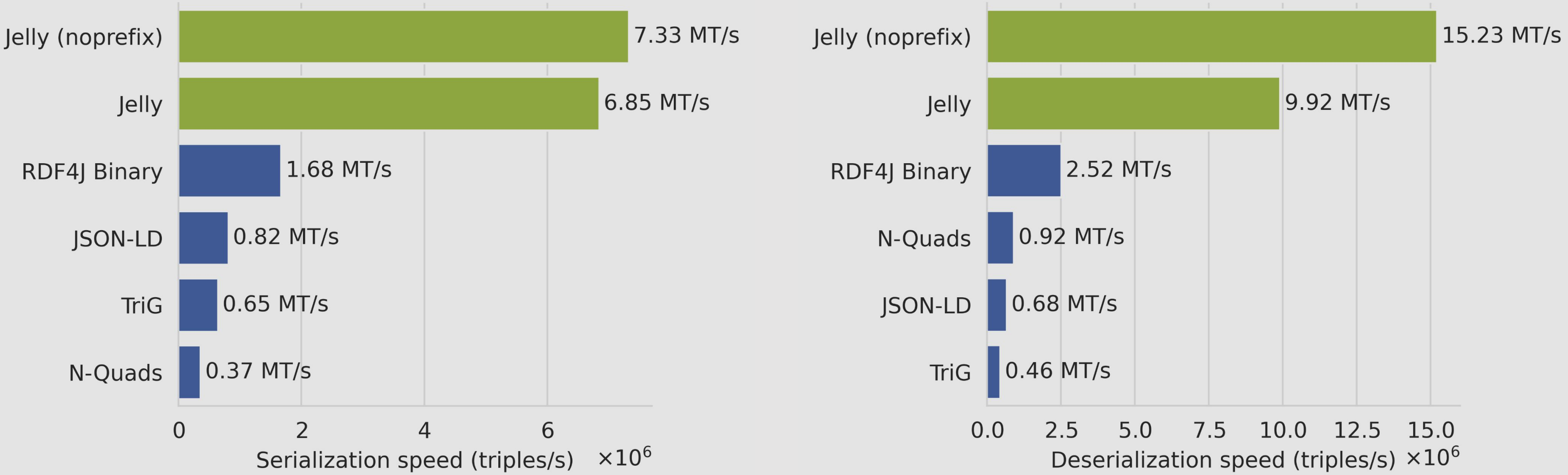


```
piotr@perun:~$ time wget https://registry.petapico.org/nanopubs.jelly -q -O- |
jelly-cli rdf inspect
stream_options:
  stream_name: ""
  physical_type: QUADS (2)
  generalized_statements: false
  rdf_star: false
  max_name_table_size: 4000
  max_prefix_table_size: 150
  max_datatype_table_size: 32
  logical_type: DATASETS (4)
  version: 2

frames:
  frame_count: 64925
  row_count: 2467391
  option_count: 1
  triple_count: 0
  quad_count: 1604486
  graph_start_count: 0
  graph_end_count: 0
  namespace_count: 647906
  name_count: 121096
  prefix_count: 93892
  datatype_count: 10

real    0m4.196s
user    0m0.314s
sys     0m0.069s
piotr@perun:~$
```

Takes ~4 seconds to complete.

Raw ser/des throughput comparison

(no HTTP overhead)



Platform: Oracle GraalVM 24+36.1, RDF4J 5.1.4, Jelly-JVM 2.10.3, Ryzen 9 7900 5.0 GHz, 64 GB RAM
Dataset: 10M nanopublications (RiverBench: nanopubs)

Why not
pipelining, parallelization, caching…?

- More complex = more costly
- Hidden resource usage – overhead still largely exists!
- Depends on the client to “do things right”
- Jelly can also compress across nanopublication boundaries
- Caches don’t help!
  - Cache is usually completely cold

Conclusion

- Communication went **from a bottleneck to a non-issue** thanks to Jelly
- **Live** on the nanopublication network: https://nanopub.net
- **Large potential for transferability:**
  - Mature tooling & documentation
  - **Use cases:** client-server communication, microservices, database dumps, streaming ingest, database replication, and more...
  - **Open community – anyone can contribute and use Jelly!**
- 100% open-source

https://w3id.org/jelly

⭐ **Star us on GitHub!**

Backup slides

Solution summary

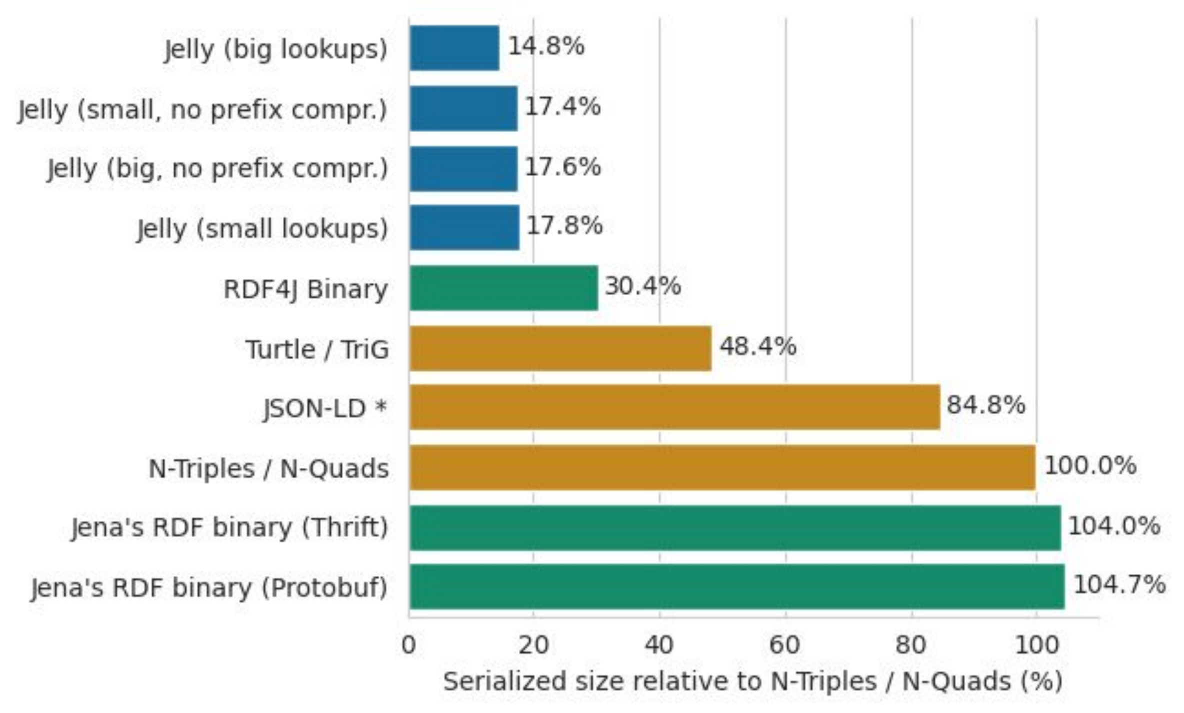
- Registry serves arbitrary subsets of nanopubs as a single streaming HTTP response
- Query & Registry consume the stream, unpack it, and process each nanopub individually

To retrieve 60k nanopubs:

- Original: 60k+ requests
- Jelly: **exactly 1 request**

Size comparison

RiverBench dataset: nanopubs, obtained with Apache Jena 5.1.0


Jelly (big lookups) 14.8%
Jelly (small, no prefix compr.) 17.4%
Jelly (big, no prefix compr.) 17.6%
Jelly (small lookups) 17.8%
RDF4J Binary 30.4%
Turtle / TriG 48.4%
JSON-LD * 84.8%
N-Triples / N-Quads 100.0%
Jena's RDF binary (Thrift) 104.0%
Jena's RDF binary (Protobuf) 104.7%
Serialized size relative to N-Triples / N-Quads (%)