Pipelining your music
Jyrki Pulliainen
Content team
Spotifier since January

Pythonista since 2.3
@nailor
Spotify?
Josh Groban

Pop and classical singer Josh Groban made his debut in the seventh grade, but then put music aside for a few years...

TOP HITS

01 You Raise Me Up

Machu Picchu

The Strokes
Number fun

- 10M monthly active users
- 18M tracks
- 100 years of music
- 20k added every day
The Music Pipeline
100s of TBs of data

Load of deliveries daily

Malformed data every day
Right tools for the right job

CC-BY raindrift http://www.flickr.com/photos/raindrift/7095238893/in/set-72157629492908038/
XPath extensions
```python
>>> def formerlify(_, name):
...     return 'The artist formerly known as %s' %name

>>> # Namespace stuff
>>> from lxml import etree
>>> ns = etree.FunctionNamespace('http://my.org/myfunctions')
>>> ns['hello'] = hello
>>> ns.prefix = 'f'

>>> root = etree.XML('<a><b>Prince</b></a>')
>>> print(root.xpath('f:hello(string(b))'))
```

... The artist formerly known as Prince
Fun(?) facts

- 10 different XML formats
  - Majors vs our own (indies)
  - One industry "standard"

Biggest XML 3.3M lines (350MB)
  - Bible apparently fits in 3MB of XML

- lxml ftw
>>> min(timeit.repeat('etree.parse("huge.xml")', setup="from lxml import etree", number=1, repeat=5))
2.309144973754883

>>> min(timeit.repeat('etree.parse("huge.xml")', setup="from xml.etree import cElementTree as etree", number=1, repeat=5))
3.0681779384613037

>>> min(timeit.timeit('etree.parse("huge.xml")', setup="from xml.etree import ElementTree as etree", repeat=5, number=1))
Killed

>>> # (with PyPy 1.9)
>>> min(timeit.repeat('etree.parse("huge.xml")', setup="from xml.etree import ElementTree as etree, number=1, repeat=5))
23.186518907546997
Merging
Fun(?) facts

- Artists don't have any global or even label specific IDs
  - Multiple artists with same name
  - Even spelling differs inside a single label
- Multiple versions of the same album
- Enormous search space!
  - \((18 \times 10^{6})^2 == \text{huge number}\)
Insufficient data

Machine learning!
>>> from unicodedata import normalize
>>> key = ''.join(normalize('NFD', char)[0].lower() for char in title)[5]

Side note: Levenshtein is expensive
=> use other edit distances too
(or use PyPy, 4x speed increase ftw)
Pro-tips

● If data is relational, use relational database (duh)

● Don't over-normalize yourself, BCNF is rarely beneficial

● Weight between denormalize vs. moar indices

● Let the DB do the hard lifting, query planner is your friend!
● Asynchronous!

● RabbitMQ + amqplib

● One master, 49 slaves

● Isilon storage => 8Gbit/s throughput!
Index building

....with Java
Why not Python?

- Not powerful enough for computationally intensive stuff
- We use Lucene for Search, so Java is a natural choice

...but I'd like to try PyPy here.
The Music Distribution
TTL one day
time to live (laɪv)
Publishing an index
SCPing around, moving hundreds of GBs daily
Future == BitTorrent

...not totally free of issues either
Index format?

Read only K/V (mostly)
Keep your eye on it
Mind the speed!
Experiment
Ditch your code
Thank you!

spoti.fi/ep_2012