ZeroMQ

MAKE YOUR WEB APP DO HEAVY SHIT

#SYPY

@CHEWXY

Message Queues – A Primer

- Facilitates inter-process communications
- Often used in decoupling heavy processing from live requests
- Poor man's distributed processing

Message Queues – A Primer

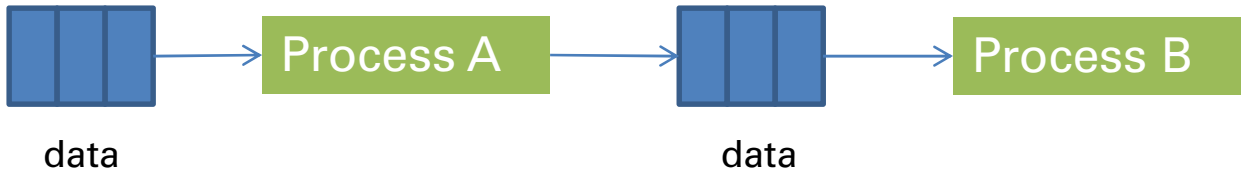
- Take this: `[[1,2,3],[4,5,6],[7,8,9]]`
- Transform into: `[1, 4, 9, 16, 25, 36, 49, 64, 81]`

Message Queues – A Primer

```
def processA(data):
    return map(lambda x: [y ** 2 for y in x], data)

def processB(data):
    return reduce(list.__add__, data, [])

def main(data):
    data = processA(data)
    print processB(data)

>>> data = [[1,2,3],[4,5,6],[7,8,9]]
>>> main(data)
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

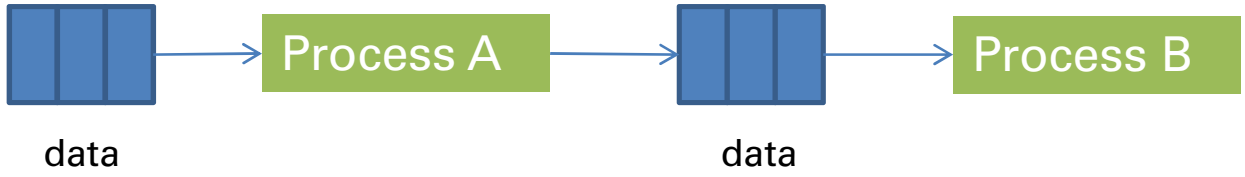
Sure it's easy with functional programming tools.
Also it looks fancy and complicated, which is why it's in this slide

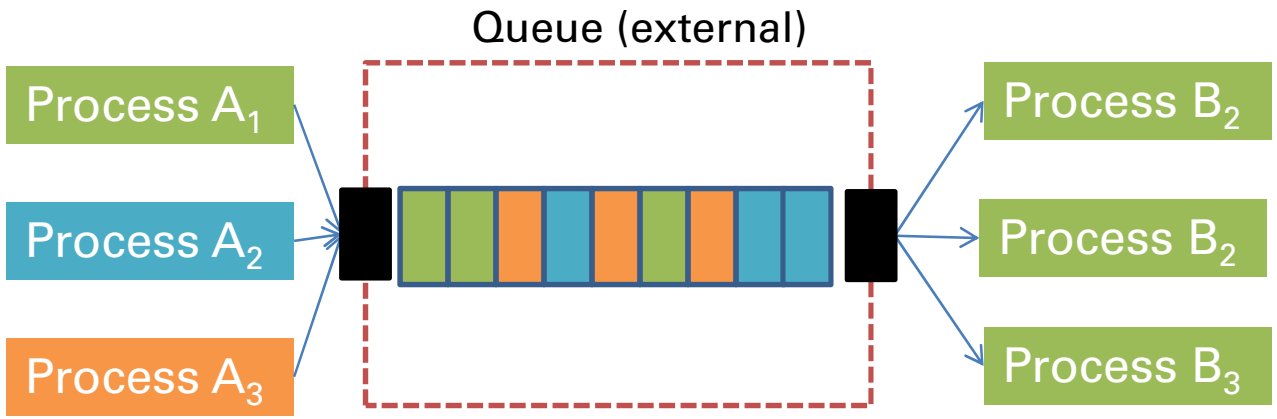
Message Queues – A Primer



From the example above, the function `processA` has to perform its map on every element in the `data` list first before passing on `data` to `processB`

Message Queues – A Primer





Message Queues – A Primer

- RabbitMQ
- IBM Websphere *
- Beanstalk'd
- Celery (Python based) *

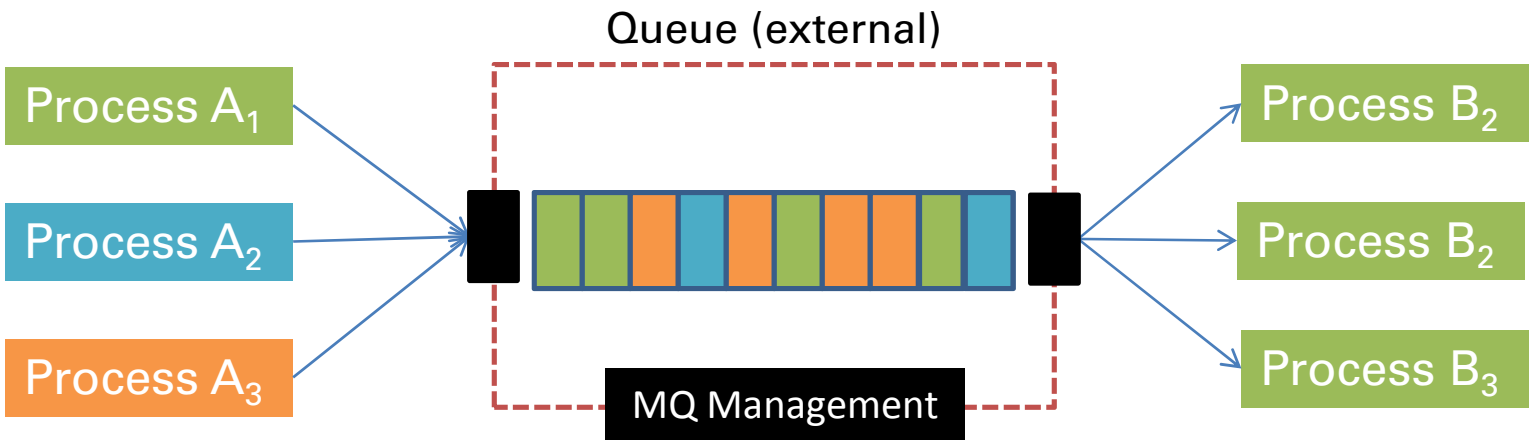
Some are technically not message queues but for the purposes of SOA and the like, but for the purposes of this talk, let's consider them MQs

ZeroMQ

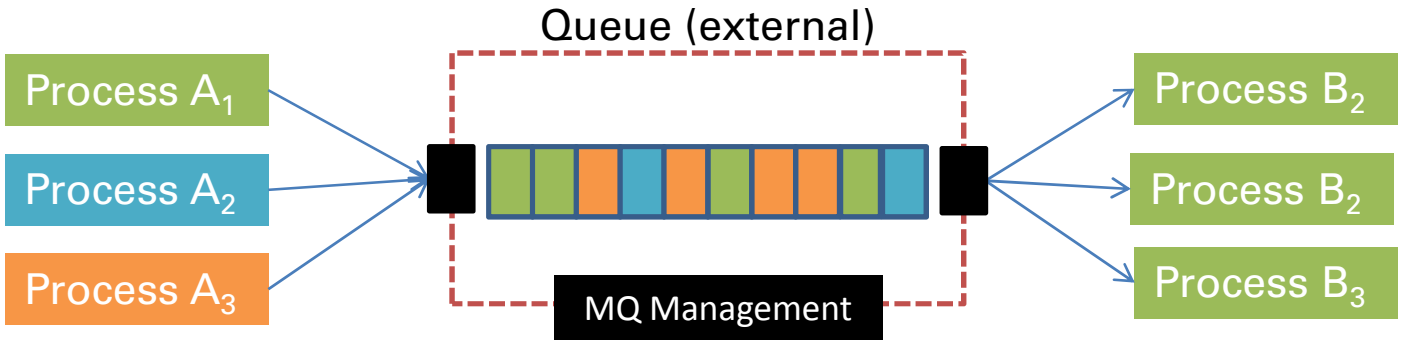
- Literally, 0 MQ (or rather, no broker)
- ZeroMQ uses sockets
- I think of ZeroMQ as a network stack with built in MQ

ZeroMQ

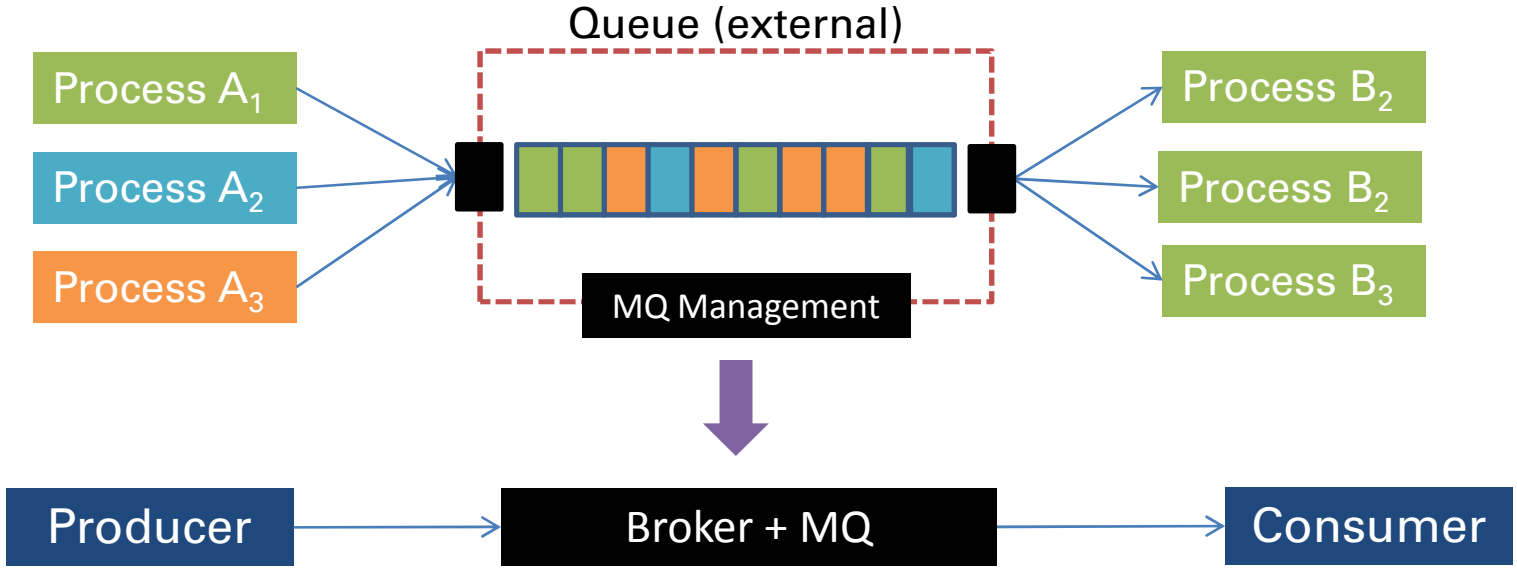
- Literally, 0 MQ (or rather, no broker)
- ZeroMQ uses sockets
- I think of ZeroMQ as a network stack with built in MQ



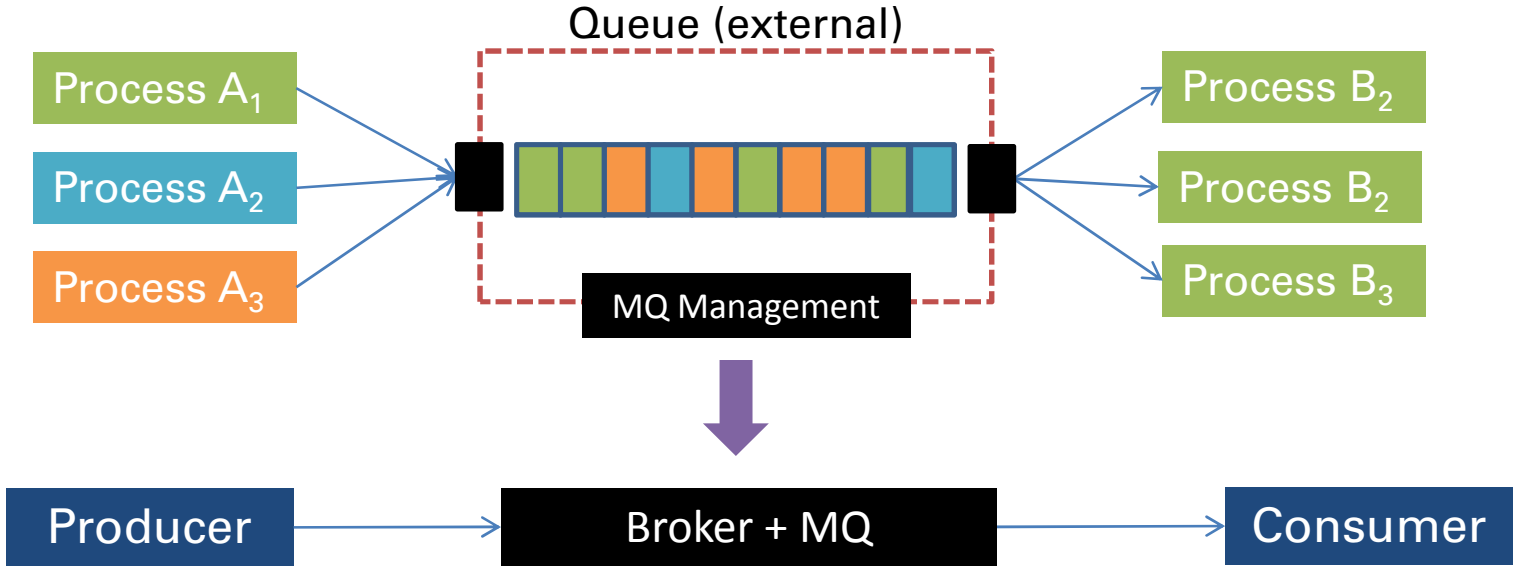
ZeroMQ


Queue (external)
Process A_1
Process A_2
Process A_3
Process B_2
Process B_2
Process B_3
MQ Management

ZeroMQ


Queue (external)
Process A1
Process A2
Process A3
Process B2
Process B2
Process B3
MQ Management
Producer
Broker + MQ
Consumer

ZeroMQ



The Genius of ZeroMQ

ZeroMQ

```
import zmq
context = zmq.Context()
socket = context.socket(zmq.PUSH)
socket.bind('tcp://*:1234')

>>> socket.send('test')
```

```
import zmq
context = zmq.Context()
socket = context.socket(zmq.PULL)
socket.connect('tcp://*:1234')

while True:
    msg = socket.recv()
    print msg

test
```

If you are reading this during the presentation that means the live coding didn't go well

ZeroMQ

- Many patterns supported:
  - Push/Pull (one way)
  - Request/Reply (two way)
  - Pub/Sub (one to many)
  - Many More!
- Many protocols supported:
  - TCP (as example)
  - IPC (inter-process communication: UNIX only)
  - InProc (inter-thread communication)
  - PGM (multicast)
- Did I mention… automatic load balancing?
- ZeroMQ is easy:
  - Wrote a push to talk chat app in < 10 minutes
  - ~ 60 lines of code with just one of the above pattern
  - See Examples

ZeroMQ in Action

- Typical usage: As a superfast version of a traditional message queue
- Also: Service Oriented Architecture
- Personally, I like the UNIX Philosophy better.

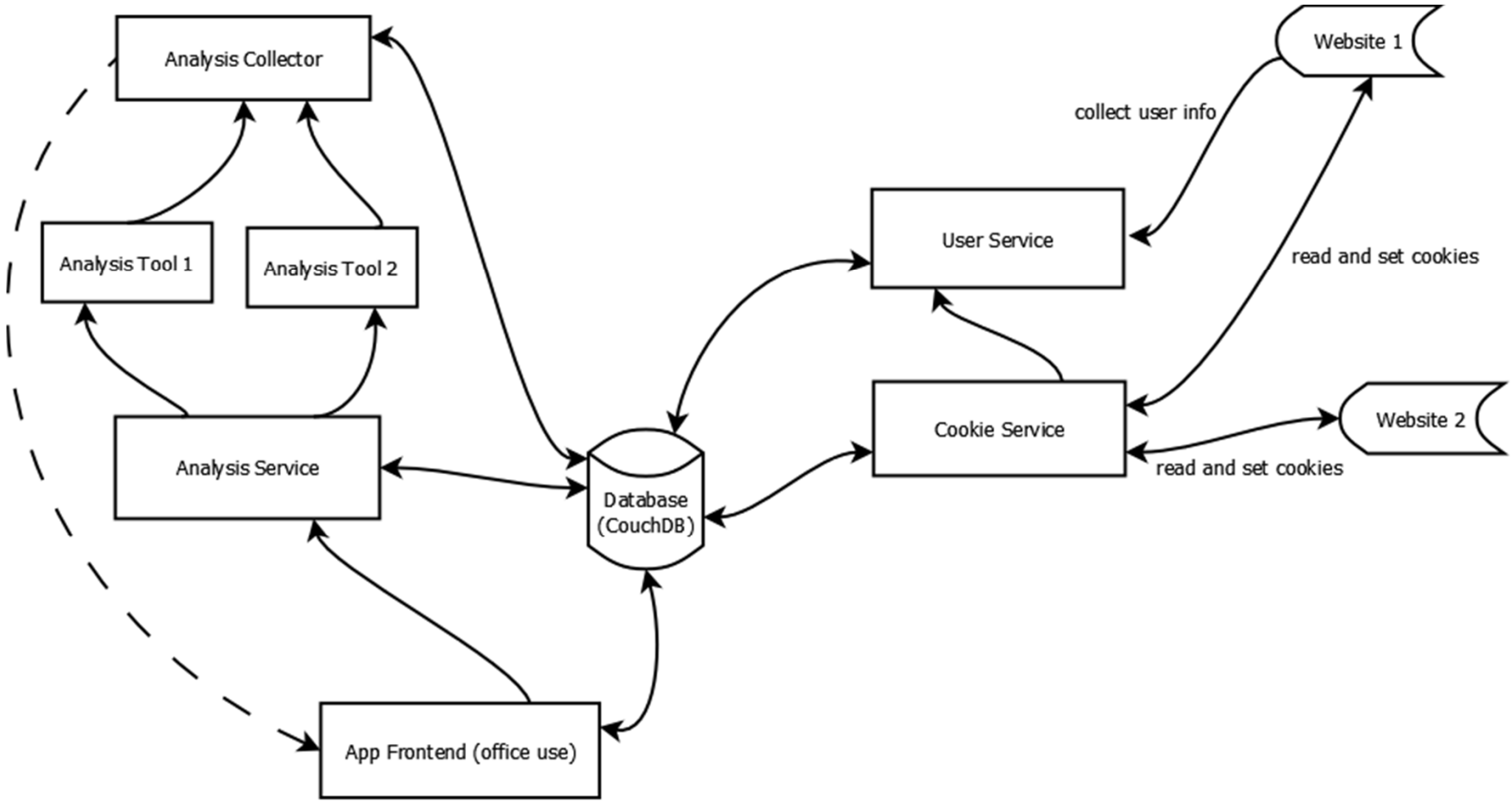
Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

ZeroMQ in Action

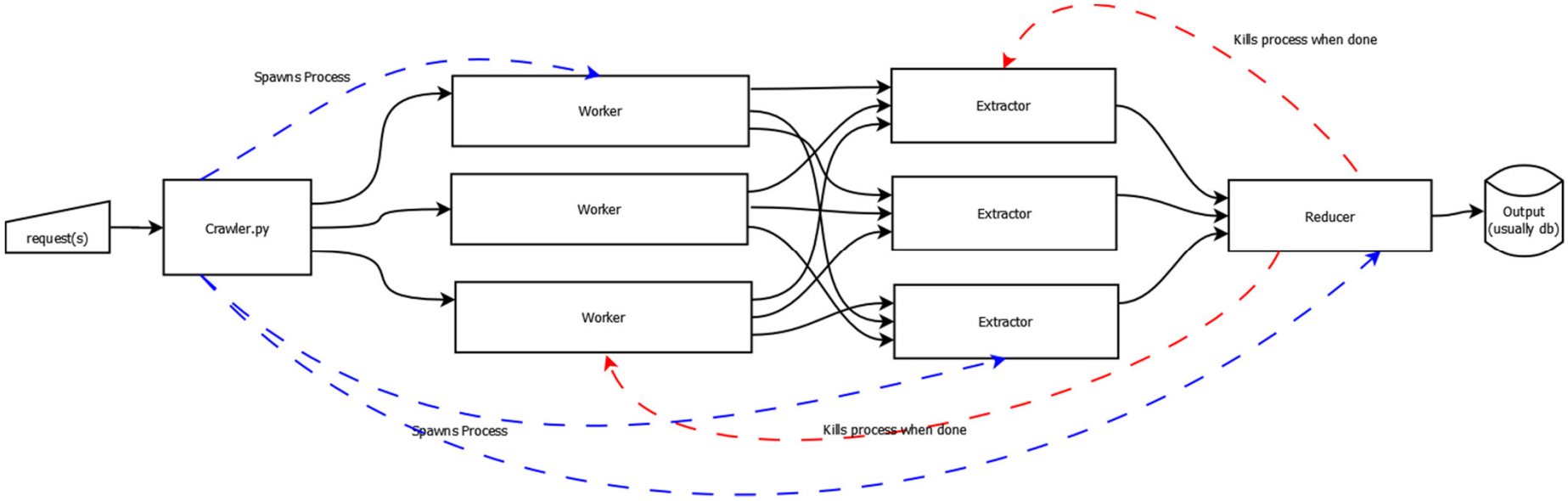
Projects I've worked on that uses ZeroMQ:

- edgeyo (http://edgeyo.com) – *project shelved*
- Strangers for Dinner (http://strangersfordinner.com) – *pivoting*
- A Brand Safety Product
- A Data Management Platform
- A Media Analysis Toolkit

ZeroMQ in Action



ZeroMQ in Action



Do Heavy Shit

- Plan, plan, plan
- You cannot tack on ZeroMQ like you tack on RabbitMQ
- Spread your load across many processes
- Not a poor man's distributed processing

Make Your Site Fast

- POST Handlers
- When POSTs are made, reply the user instantly with cached data.
- Handle the POST information separately in the back end (by PUSHing your data to your POST handler app)
- If you need to communicate results with the user, *socket.io* is a good idea.

THANK YOU

- Examples: http://github.com/chewxy/SyPyOct2012
- Email: chewxy@pressyo.com
- Twitter: @chewxy

p/s: follow me now and a unicorn will present bacon and whiskey to you tonight