Seminario Informática “Mirian Andrés”

El arte de la ciencia de datos: buenas prácticas y tendencias

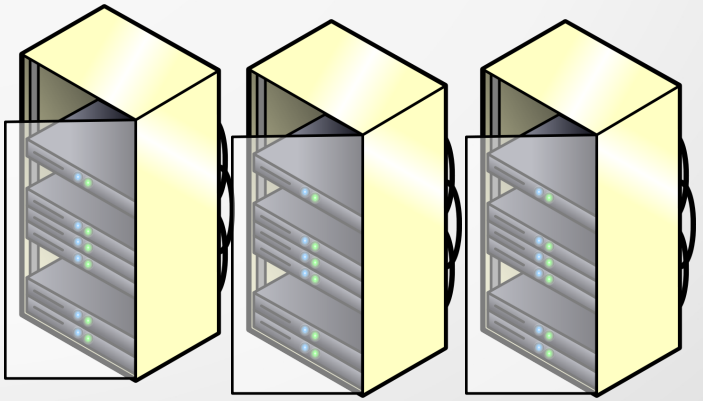
Felipe Ortega.
Dpto. de Informática y Estadística
Universidad Rey Juan Carlos
e-mail: felipe.ortega@urjc.es Twitter: @jfelipe

Ideas centrales

Big data ≠ hadoop

Big data ≠ SQL

Big data ≠



Definición de *big data*

- Conjuntos de datos que, por su excesivo **tamaño** o **complejidad**, no se pueden procesar usando métodos o herramientas (computacionales) **convencionales**.
- Usualmente definido por tres características [1] :
  - **Volumen**
  - **Variedad**
  - **Velocidad**

Implicaciones *big data*

- Los requisitos de *big data* obligan a buscar métodos y herramientas **alternativas** para trabajar de forma **eficiente**.
  - Sin embargo esto **no implica** que ciertas **tecnologías existentes ya no sean útiles** (ej. RDBMS).
- Claves del éxito en *big data*:
  - Entender los requisitos del problema.
  - Aprovechar nuevos métodos y tecnologías junto con otros ya existentes.
  - **Optimizar**, **personalizar**, **innovar**.

El *arte* de la ciencia de datos

- En la práctica, la ciencia de datos exige
  - **comprender** mejor el problema o proyecto;
  - perfiles/equipos **multidisciplinares**;
  - considerar **múltiples alternativas** de implementación;
  - tener **paciencia** para probar y probar alternativas hasta encontrar la mejor solución;
    - Error #1: "*Quiero trabajar con Hadoop a toda costa*" (sin valorar las implicaciones que conlleva).
    - Error #2: "*Hay que instalar NoSQL sí o sí*" (porque es lo que "todo el mundo" está usando ahora).

Ciencia de datos *pragmática*

- Si es posible, comenzar por **optimizar la infraestructura existente**.
- Identificar requisitos del escenario:
  - Volumen: espacio en disco, compresión de datos, particionado de datos, formatos de representación
  - Variedad: métodos y herramientas estadísticas
  - Velocidad: algoritmos, lenguaje/compilador, paralelización, conectividad de red, datos en memoria, tecnología estado sólido.
- Plantear soluciones de computación distribuida sólo cunado sea necesario.

Ciencia de datos *pragmática*

- Evaluar implicaciones de coste (dinero, tiempo, esfuerzo, cambio).
- Ejemplo: Aplicación intensiva en base de datos.
  - **Vía 1**: Cluster + Hadoop + HDFS + Cassandra
    - Para descubrir que **no se puede hacer JOIN**
  - **Vía 2**: Mejora esquema, creación índices, particionado de datos, optimización del servidor, optimización sistema de ficheros, usar SSDs.
    - Mejora de rendimiento (hasta un 70%), sistema conocido, cambios mínimos para usuarios.

Caso: AEROSPIKE

- Base de datos escalable de alto rendimiento.
  - Objetivo: análisis de datos en **tiempo real** (baja latencia).
  - Modo RDBMS (adquisición de AlchemyDB en 2012).
  - Modo NoSQL (clave-valor, datos complejos).
- Arquitectura flexible.
  - Datos en memoria (sin persistencia).
  - Datos en memoria con persistencia en disco.
  - Datos en SSD, índices en memoria (**default**).
  - Un servidor, múltiples servidores, múltiples data centers.

Tecnología SSD

Comparando costes

	Disco convencional	Disco de estado sólido SATA	Estado sólido PCIe	DRAM
COSTE PROMEDIO POR GB	€0,04 – €0,09	€0,40 – €0,80	€1,50 – €2,30	€12 – €25
CAPACIDAD MÁXIMA POR DISPOSITIVO	Hasta 6 TB	Hasta 1 TB	Hasta 4 TB	Hasta 8 GB

Algunas alternativas implementación

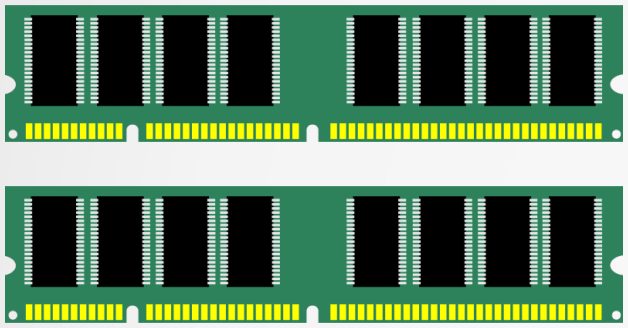
- Datos en memoria.
  - ✔ Opción más rápida (latencia ~ **nanosegundos**).
  - ✗ Elevado coste, capacidad limitada.
- RAID discos convencionales.
  - ✔ Mejor relación coste/capacidad.
  - ✔ Capacidad escalable (hasta Exabytes).
  - ✗ Velocidad de acceso limitada
    - Los discos convencionales están sujetos a latencias debido a componentes mecánicos móviles.
    - Latencia de acceso del orden de **milisegundos**.

Algunas alternativas implementación

- SSD.
  - ✔ Segunda opción más rápida (latencia ~ **10 nanosegundos**).
  - ✔ Buena capacidad, coste asequible.
  - ✗ Durabilidad limitada (aunque mejorando).
  - ✗ Soporte capas intermedias (kernel, sistemas de ficheros).
- RAID SSDs.
  - ✔ Mayor escalabilidad y velocidad
  - ✗ Precisa soporte específico de la controladora y ancho de banda suficiente para altas tasas de transferencia.

Estrategias con SSD

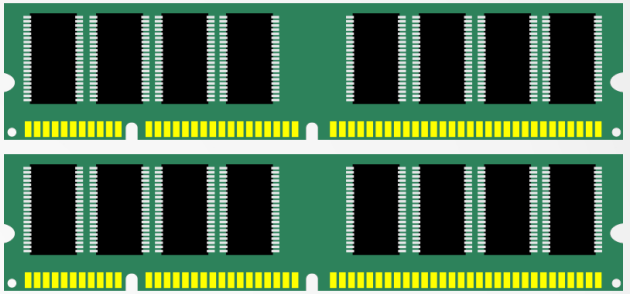
ESCENARIO 1



8 GB RAM

- x8 Capacidad memoria
- Aceleramos S.O. y aplicaciones
- Por una fracción del coste de ampliación de RAM

ESCENARIO 2

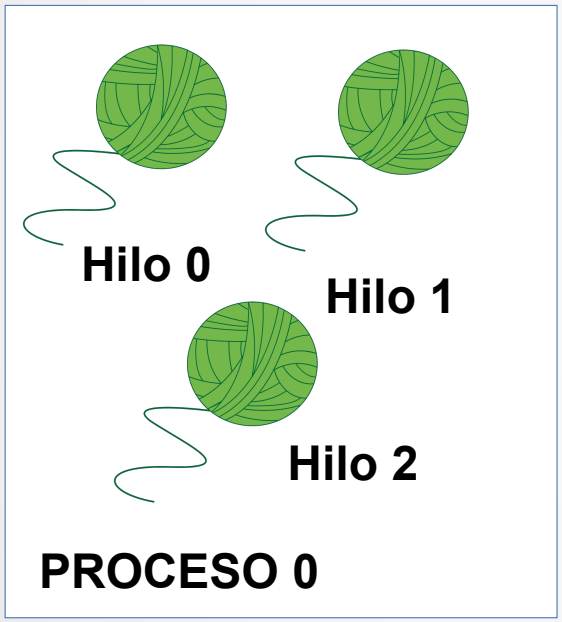


8 GB RAM + 64 GB swap SSD

Multiproceso y comunicación

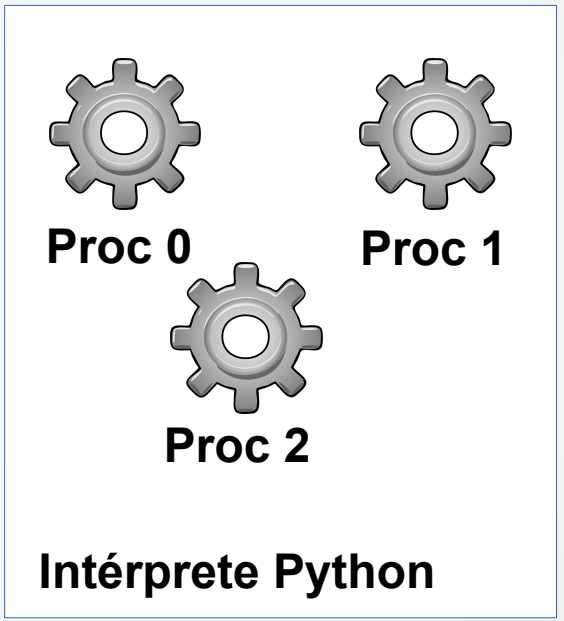
Concurrencia y paralelismo

Multihilo



threading

Multiproceso

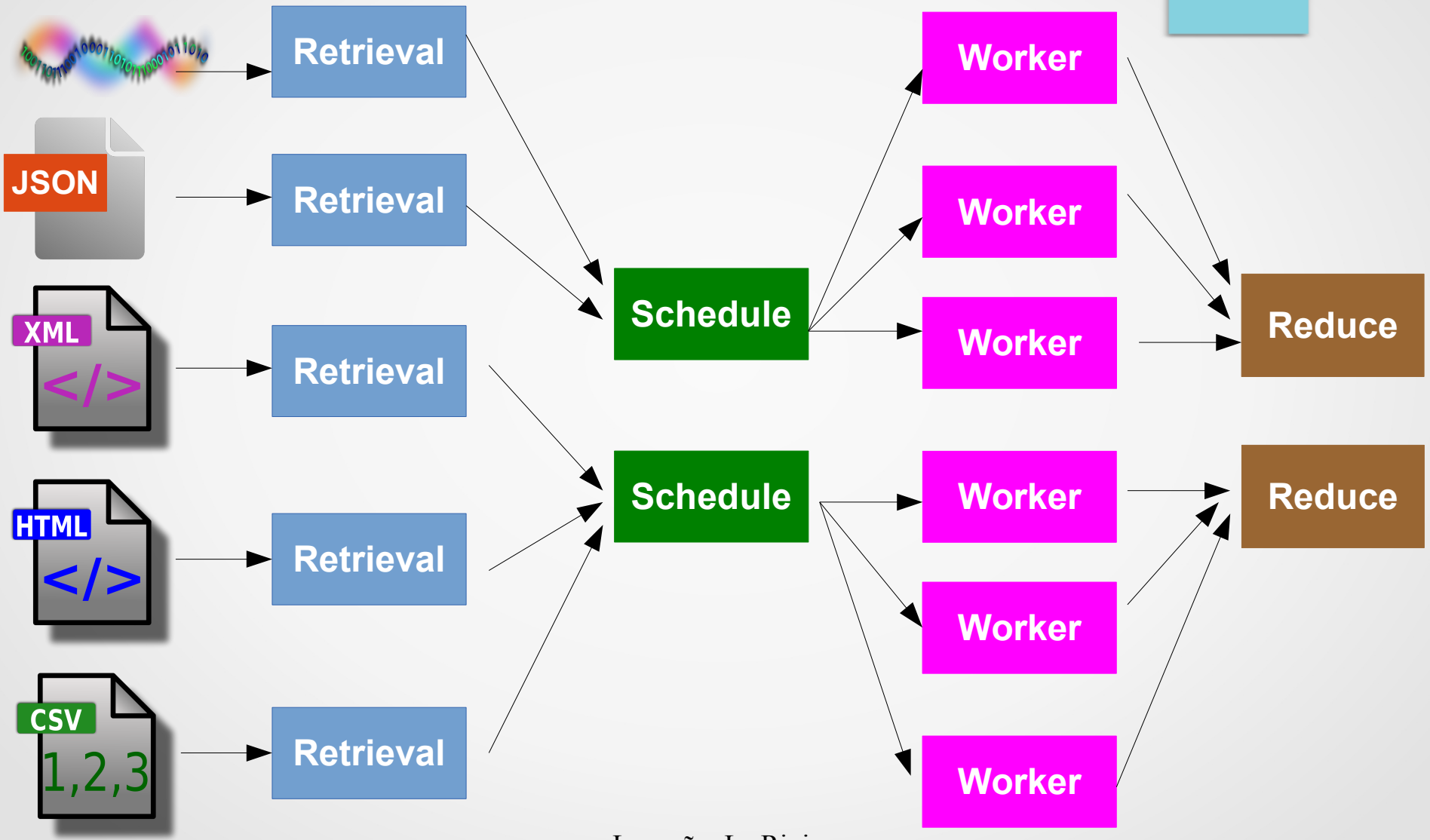


multiprocessing

Concurrencia y paralelismo

- Programación multihilo en Python
  - Módulo *threading*.
  - Sistemas típicos de sincronización.
    - Event, Lock, Semaphore.
  - Grave problema con el GIL.
    - Nunca habrá más de un hilo en ejecución (!!).
- Programación multiproceso en Python.
  - Módulo *multiprocessing*.
  - Reescritura de *threading* como procesos ligeros.

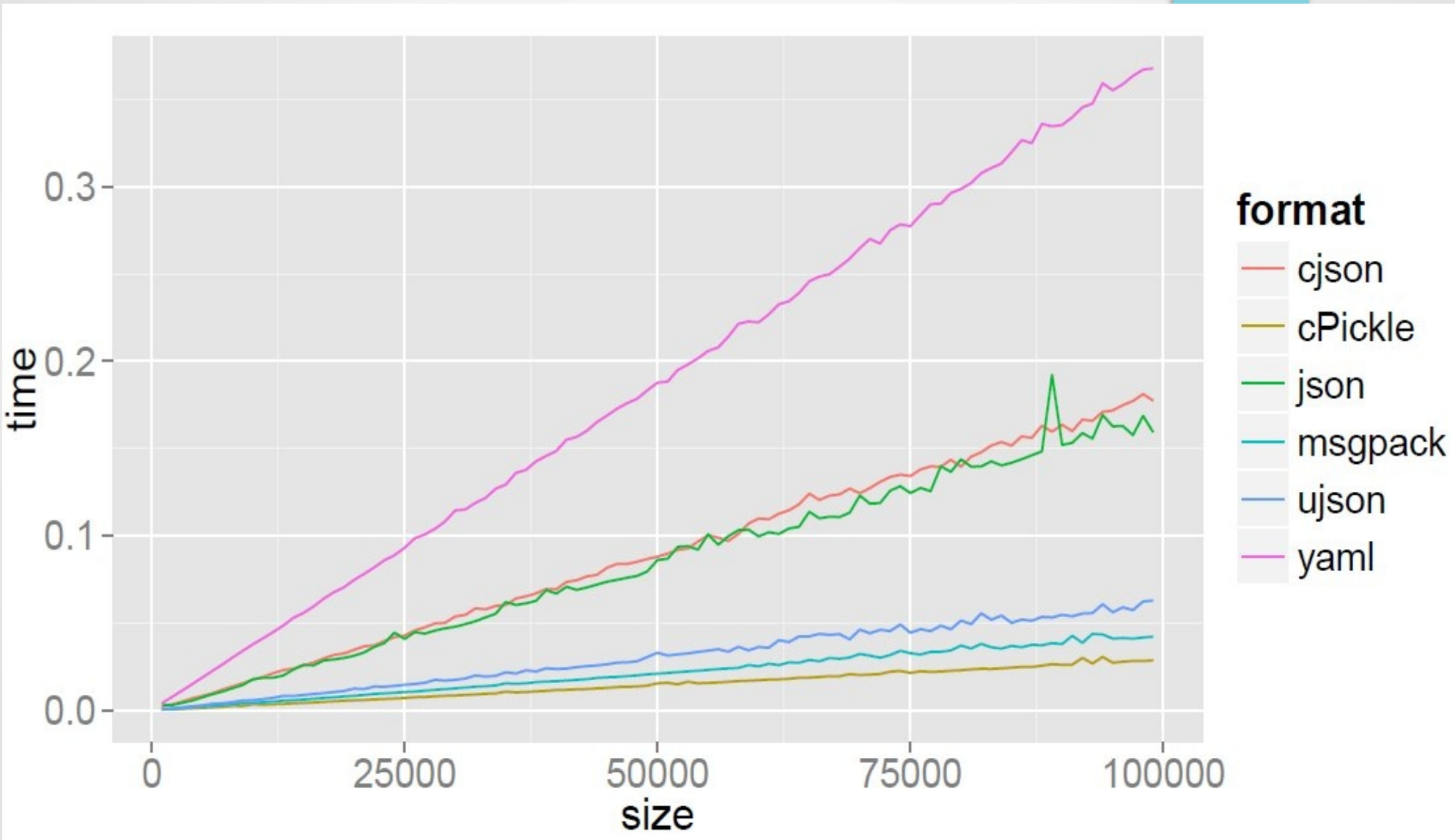
Python *multiprocessing*



Importación y transferencia datos

- Usar módulos alto rendimiento.
  - *ujson* (importanción de datos en JSON).
  - *lxml* (importanción de datos en XML).
  - *beautifulsoup4* (importación de datos en HTML).
  - *csv* (lectura de ficheros CSV).
- Preferir formatos ligeros frente a sobrecargados.
  - Ligeros: JSON, CSV.
  - Sobrecargados: XML (aunque potentes).

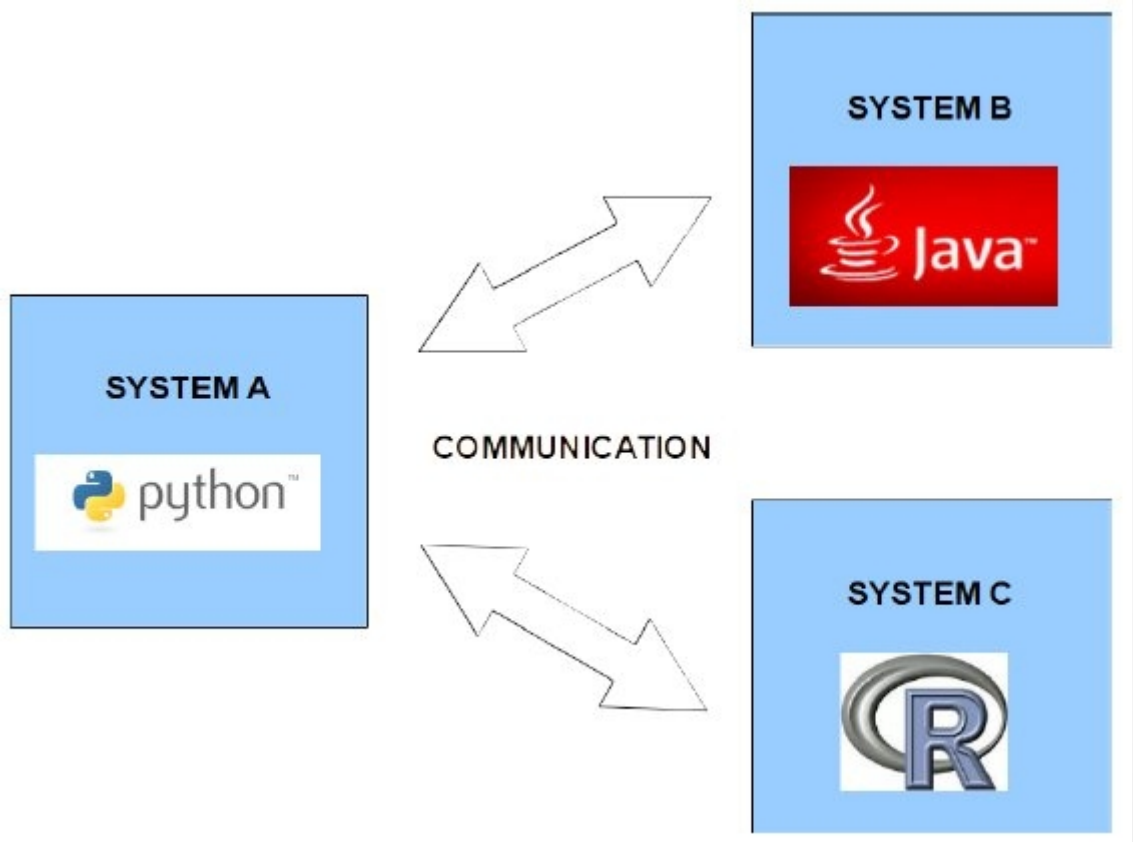
Importación y transferencia datos



Transferencia datos RDBMS

- SQL Toolkit & ORM.
  - Modo ORM.
    - Oculta complejidad, más lento.
  - Modo Core.
    - Acceso a ajustes finos para rendimiento.
- Otras opciones:
  - Utilización de archivos intermedios.
    - Cargar desde CSV.
  - *Executemany(...)*
    - *Estandar DBI, soportado por multiples BD y conectores.*

Comunicación inteligente



0MQ (ZeroMQ)

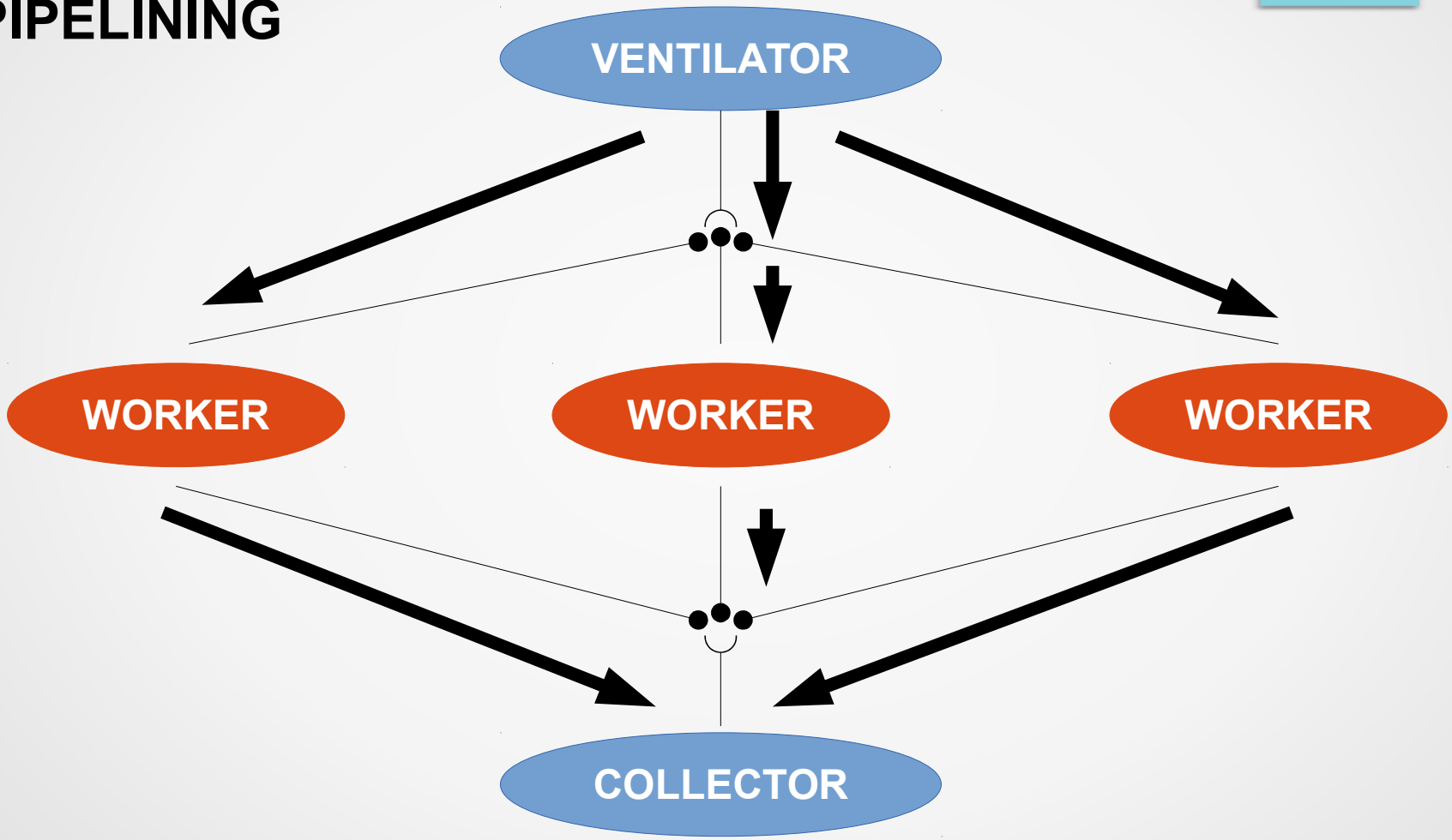
- Sistema de mensajería escalable, alto rendimiento.
  - Comunicación por red o IPC (local file device).
  - Válido para multiproceso o cómputo distribuido.
  - Bibliotecas de conexión en más de 20 lenguajes.
- Patrones de comunicación.
  - REQ/REP.
  - PUB/SUB.
  - PUSH/PULL.
  - Paired communication.
- Dispositivos de conexión predefinidos.
  - Queue, Forwarder, Streamer.

0MQ (ZeroMQ)

- Ventajas
  - Multiplataforma.
  - Muy baja latencia de envío, gran escalabilidad.
  - Extremadamente sencillo de utilizar [2].
- Inconvenientes
  - Soporte para autenticación y cifrado añadido muy recientemente.
  - El problema de persistencia de datos queda del lado del programador.
    - Estándar AMPQ lo implementa, pero mucho más lento.

Ejemplo 0MQ

PIPELINING



Ejemplo ZeroMQ: *Ventilator*

```
60      def run(self):
61          target = self.target
62
63          context = zmq.Context()
64          # Set up sending channel for page elements
65          channel_pages_send = context.socket(zmq.PUSH)
66          channel_pages_send.bind("tcp://127.0.0.1:%s" % self.push_pages_port)
67
68          # Set up sending channel for revision elements
69          channel_revs_send = context.socket(zmq.PUSH)
70          channel_revs_send.bind("tcp://127.0.0.1:%s" % self.push_revs_port)
71
72          channel_control = context.socket(zmq.PUB)
73          channel_control.bind("tcp://127.0.0.1:%s" % self.control_port)
74
75          # Wait a second to wake up and connect
76          time.sleep(1)
77
78          for item in target(*self.args, **self.kwargs):
79              # Classify outcome elements in their corresponding queue
80              # for later processing
81              if isinstance(item, Page):
82                  send_ujson(channel_pages_send, item)
83
84              elif isinstance(item, Revision):
85                  send_ujson(channel_revs_send, item)
        # Send STOP message to all workers and quit
        if self.page_consumers > 0 and self.rev_consumers > 0:
            channel_control.send('STOP')
```

Ejemplo ZeroMQ: *Forwarder*

```
170     def items(self):
171         context = zmq.Context()
172         data_recv = context.socket(zmq.PULL)
173         data_recv.connect("tcp://127.0.0.1:%s" % self.pull_port)
174 
175         control_sub = context.socket(zmq.SUB)
176         control_sub.connect("tcp://127.0.0.1:%s" % self.control_port)
177         control_sub.setsockopt(zmq.SUBSCRIBE, "STOP")
178 
179         # Wait a second to wake up and connect
180         time.sleep(1)
181 
182         # Initialize poll set
183         poller = zmq.Poller()
184         poller.register(data_recv, zmq.POLLIN)
185         poller.register(control_sub, zmq.POLLIN)
```

Ejemplo ZeroMQ: *Forwarder*

```
while self.producers > 0:
    # Work on requests from pipelining and control channel
    while True:
        socks = dict(poller.poll())
        if data_recv in socks and socks[data_recv] == zmq.POLLIN:
            yield(recv_ujson(data_recv))

        if control_sub in socks and socks[control_sub] == zmq.POLLIN:
            message = control_sub.recv()
            if message == "STOP":
                print "Recieved exit command, client %s stop recieving messages" % self.name
                break  # Exit poll loop

    self.producers -= 1
```

Ejemplo ZeroMQ: *Forwarder*

```
def run(self):
    target = self.target
    context = zmq.Context()
    channel_send = context.socket(zmq.PUSH)
    channel_send.connect("tcp://127.0.0.1:" + str(self.push_port))

    # Wait a second to wake up and connect
    time.sleep(1)

    for item in target(self.items(), **self.kwargs):
        send_ujson(channel_send, item)

    for x in range(self.consumers):
        send_ujson(channel_send, 'STOP')
```

Ejemplo ZeroMQ: *Collector*

```
125     def items(self):
126         context = zmq.Context()
127         data_recv = context.socket(zmq.PULL)
128         data_recv.bind("tcp://127.0.0.1:"+str(self.pull_port))
129 
130         # Wait a second to wake up and connect
131         time.sleep(1)
132 
133         while self.producers > 0:
134             while True:
135                 item = recv_ujson(data_recv)
136                 if item == 'STOP':
137                     break
138                 yield item
139             self.producers -= 1
140 
141         time.sleep(1)
142         #data_recv.close()
143 
144     def run(self):
145         target = self.target
146         target(self.items(), **self.kwargs)
```

Reproducibilidad

Reproducible vs. replicable

- La distinción tiene gran trascendencia y fuertes implicaciones [3].
  - **Replicabilidad**: Se puede llevar a cabo el mismo experimento, en las mismas condiciones, obteniendo los mismos resultados ("*reproduce exactly*").
    - Ej: Mismo hardware, SO, software, bibliotecas, conjunto de datos...
  - **Reproducibilidad**: Realizar un experimento con el mismo objetivo pero condiciones diferentes, obteniendo resultados que concuerdan o confirman las conclusiones del experimento original.
    - Hardware distinto, otro SO, diferentes bibliotecas, otros conjuntos de datos

...¿Y en la ciencia de datos?

- Elementos necesarios:
  - Conjuntos de **datos** utilizados.
  - **Infraestructura** (recursos computacionales).
  - Software:
    - **Código** para llevar a cabo el análisis.
    - **Dependencias** (otros programas, bibliotecas).
    - **Configuración** original.
  - Metodología.
    - Explicación detallada del **proceso** (limpieza y preparación de datos, análisis, resultados, conclusiones).

Espectro de reproducibiliad

Sólo publicación **Elementos adicionales** **Replicación total**

Código

Código y datos

Entorno de ejecución y datos enlazados

No Reproducible

Gold standard

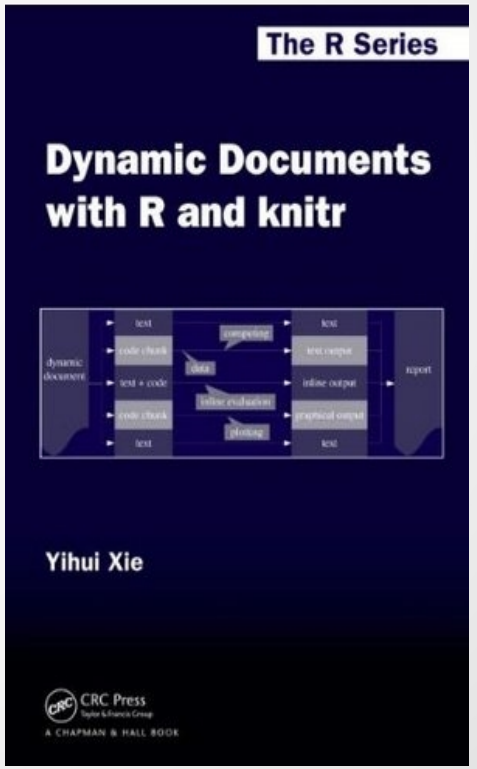
Grados de replicación

En la práctica...

- Resultados de investigación no reproducibles.
  - **Oncología** [4]: Dpto. Biotecnología de la firma Amgen (Thousand Oaks) sólo confirmó 6 de un total de 53 artículos emblemáticos. Bayer HealthCare (Alemania) pudo validar un 25% de estudios.
  - **Psicología** [5]: De un total de 249 artículos de la APA, el 73% de los autores no respondieron sobre sus datos en 6 meses.
  - **Economía, finanzas** [6]: Diferentes paquetes software producen resultados muy distintos con técnicas estadísticas directas aplicadas sobre datos idénticos a los originales.
  - **Ing. Software** [7]: 171 artículos analizados, la mayoría sin software fuentes de datos u otros elementos que permitan replicación.

Automatización de publicaciones

- Sweave.
  - Paquete original de programación literaria con LaTeX y R.
- Knittr.
  - Otro paquete más moderno y sencillo de utilizar para producción de documentos con LaTeX y R.
  - Se pueden migrar documentos desde Sweave con `Sweave2knittr()`.
  - Integrado con IDEs (RStudio).

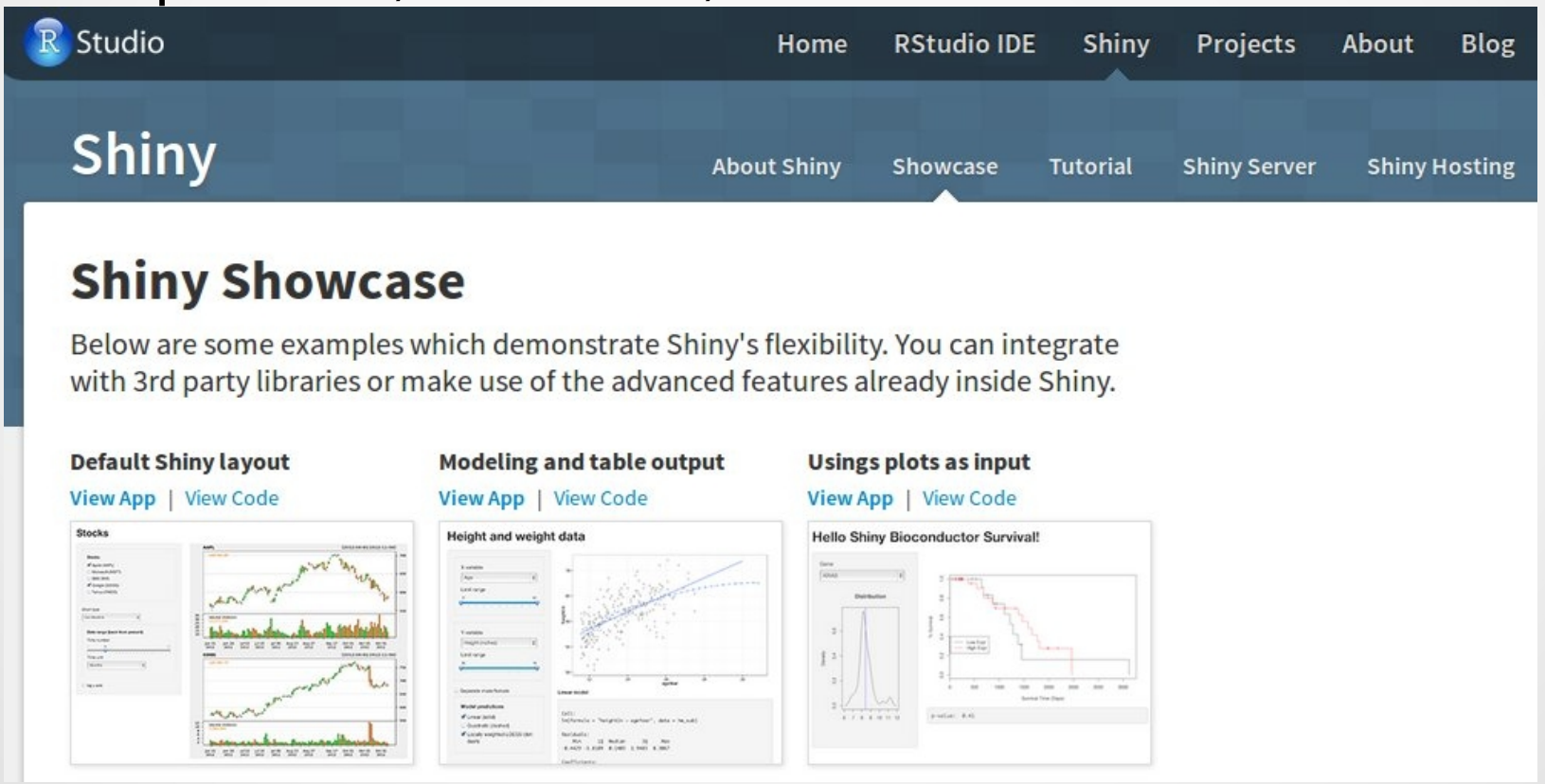


Publicación (web)

- REST: Django + Tastypie.
  - Aplicacones web con Django reutilizables para creación de servicios/APIs REST.
  - Protocolos sin estado (evita abstracciones tipo SOAP).
  - Operaciones bien definidas (estándar HTTP).
  - Acceso a través de la URI (sintaxis universal).
    - http://127.0.0.1:8000/api/entry/1/?format=json
    - http://127.0.0.1:8000/api/entry/schema/?format=json
    - http://127.0.0.1:8000/api/entry/set/1;3/?format=json
  - Múltiples formatos de representación: HTML, XML, JSON, YAML...

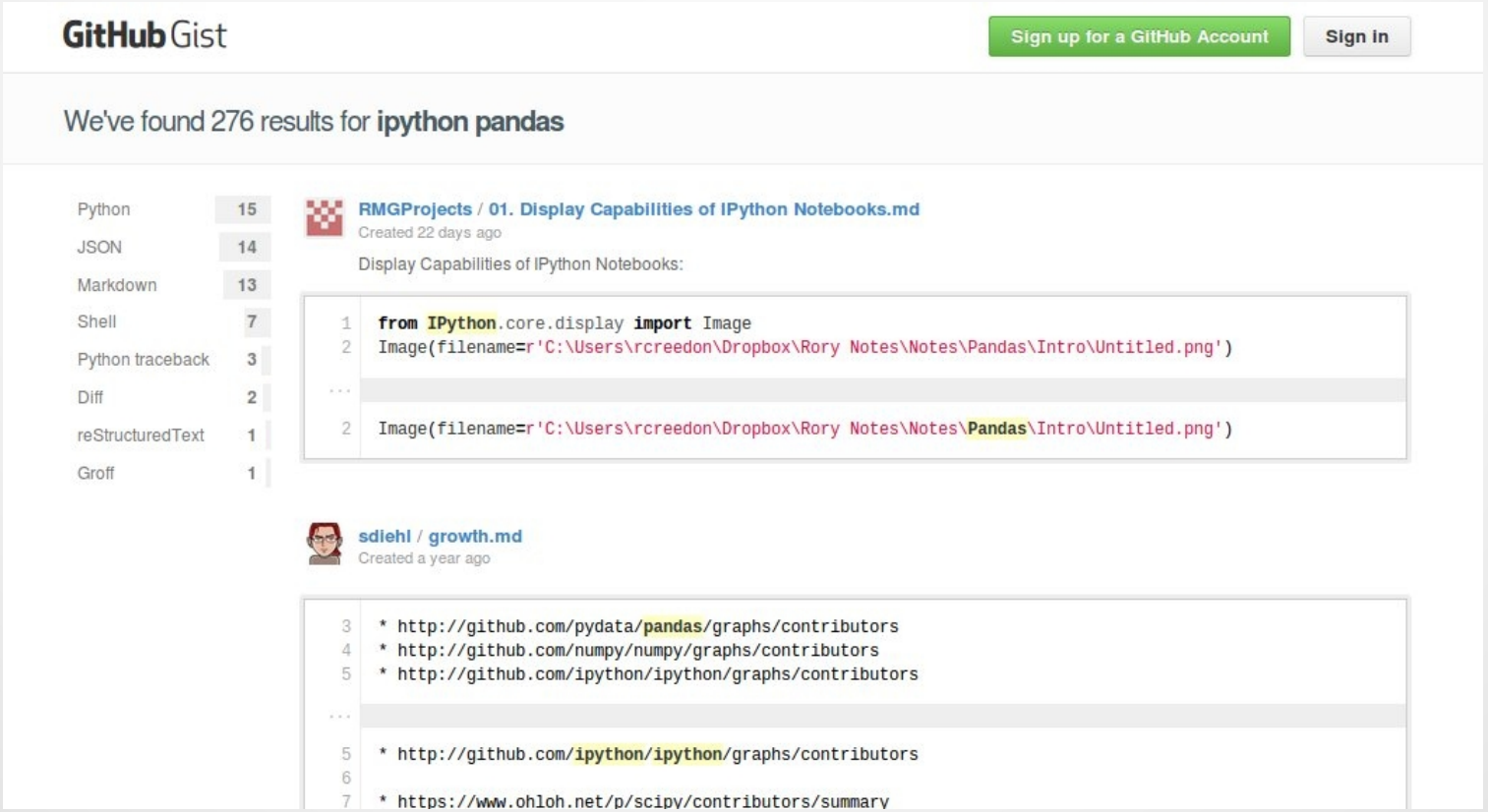
Publicación (web)

- Interfaces web (Rstudio + Shiny).
  - Replicación, educación, demostradores.



Compartiendo código

- Gist: Forma fácil de compartir piezas de código.
  - Cada Gist es un repositorio Git (versionable, admite fork).



Compartiendo código

- Shiny admite la ejecución de Gist remotos.
  - Podemos compartir código de aplicaciones Shiny.
  - Descargar y ejecutar localmente en pocos segundos.

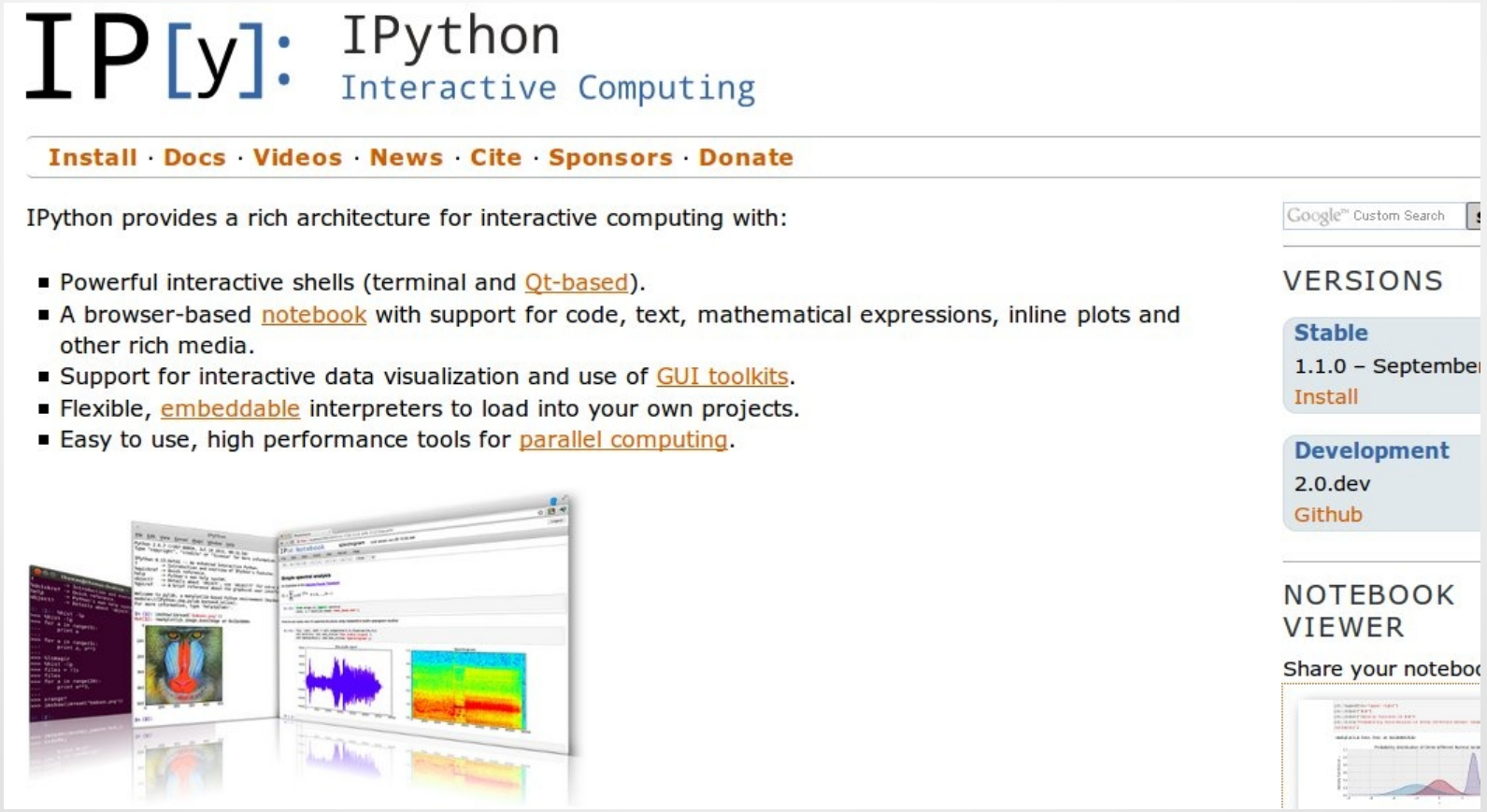
```
> install.packages("shiny")
# Siguiente paquete para ejecutar ejemplo
> install.packages("PerformanceAnalytics")
> library(shiny)
> runGist(5081906)
```

Documentando el proceso

- Documentando el código.
  - Vignettes, Sweave, Knittr, roxygen (en R).
  - Markdown.
  - Sphinx (originalmente Python, también C y C++).
- Cuadernos de notas.
  - Wikis (semáticos, si es posible).
    - Múltiples motores: Tiki Wiki (CMS), DokuWiki, Mediawiki.
  - RStudio puede crear “cuadernos” HTML básicos a partir de archivos fuente R.

Documentando el proceso

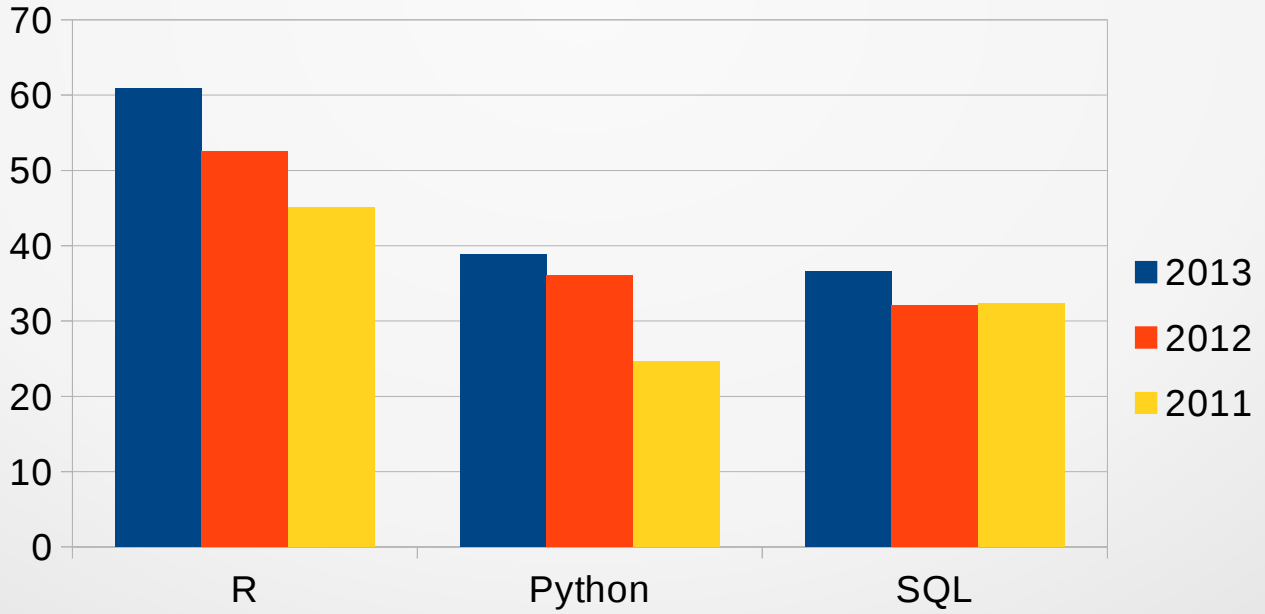
- IPython.
  - Entorno interactivo avanzado para programación y documentación con Python.



Conclusiones

Encuesta de popularidad

- Programación estadística y científica.
  - R + CRAN + Bioconductor.
  - Python + SciPy + NumPy + Pandas.
- Ranking the popularidad (encuesta 2013 KDnuggets).



Conclusiones

- Evitar restricción de opciones.
  - Big data no es solo Hadoop.
  - Big data no es solo NoSQL.
- Soluciones creativas, comprendiendo los requisitos del problema y aspectos como el coste o las implicaciones de cambios.
- No implementar soluciones ciegamente, sólo porque otros también lo hagan.
  - Principio de **parsimonia** (soluciones **simples** y escalables).

Dan Ariely

Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it...

6 de enero de 2013 a la(s) 8:02 ·

Referencias

[1] Doug Laney, 3d Data managment: controlling data volume, velocity and variety, Appl. Delivery Strategies Meta Group (949) (2001).

[2] Nicholas Piël. *ZeroMQ: An introduction*. http://nichol.as/zeromq-an-introduction

[3] Drummond, C. 2009. *Replicability is not reproducibility: nor is it good science*. Proc. Eval. Methods Mach. Learn. Workshop 26th ICML, Montreal, Quebec, Canada. http://www.csi.uottawa.ca/~cdrummon/pubs/ICMLws09.pdf.

[4] Begley, C. Glenn, and Lee M. Ellis. "Drug development: Raise standards for preclinical cancer research." Nature 483.7391 (2012): 531-533.

[5] Wicherts, Jelte M., et al. "The poor availability of psychological research data for reanalysis." American Psychologist 61.7 (2006): 726.

[6] Burman, Leonard E., W. Robert Reed, and James Alm. "A call for replication studies." Public Finance Review 38.6 (2010): 787-793.

[7] Robles, Gregorio. "Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings." Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 2010.