Lo Zen e l'Arte della Manutenzione di Astrazioni

http://www.aleax.it/itpyc_abst.pdf

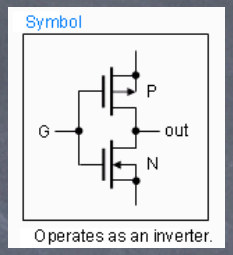
I "livelli" di questo talk

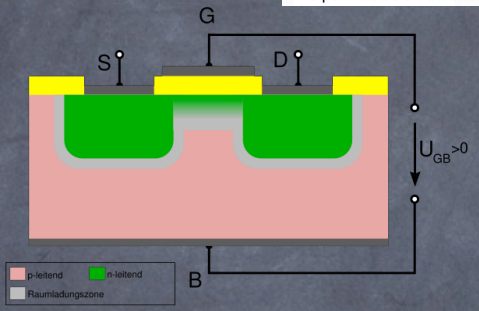
15' Q & A alla fine

(+: parliamone dopo!)

Google

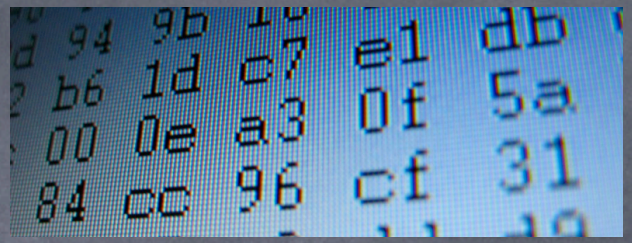
Una Torre di Astrazioni

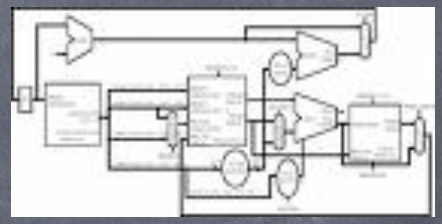


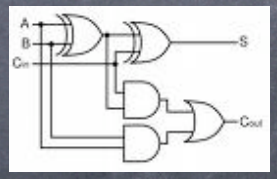


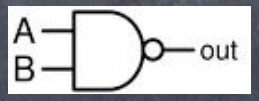
$$-\sum_i \frac{\hbar^2 \nabla_i^2 \Psi}{2m_i} + \sum_{i>j} V_{i,j} \Psi = E\Psi$$

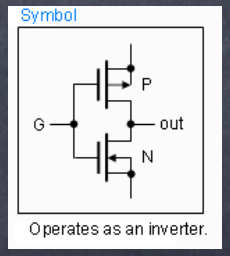
```
push    ebp
mov     ebp, esp
movzx   ecx, [ebp+arg_0]
pop     ebp
movzx   dx, cl
```





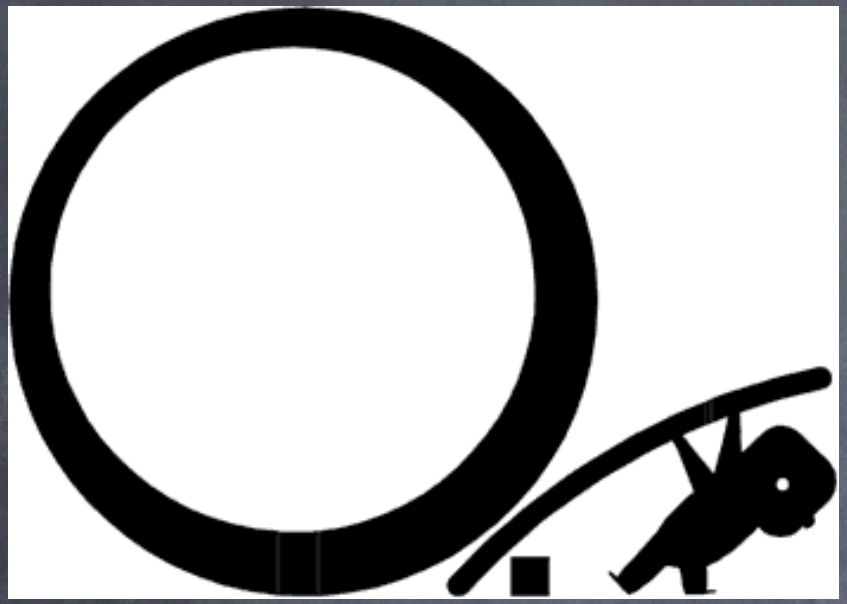






Google

Datemi un punto d'appoggio...



...e potró fare tanto di piú...

...ma puó essere un guaio se le cose vanno male!

Google

Non possiamo farne senza...

- programmare (& tanto altro "knowledge work")
  - USA, sempre, vari strati di astrazione,
  - e spesso ne PRODUCE altri sovrapposti

Google

...ma come sopravvivere?

- le astrazioni "FAN ACQUA" (Legge di Spolsky)

  - ...bug, overload, attacchi alla sicurezza...
  - ...dunque DEVI capire qualche livello sotto!
- piú, DEVONO "far acqua" (a volte;-)
  - in modo progettato e architettato

  e: l'astrazione *puó rallentarti*!

Google

Astrarre -> Procrastinare!

- McCrea, S. M., Liberman, N., Trope, Y., & Sherman, S. J. -- Construal level and procrastination. Psychological Science, Volume 19, Number 12, December 2008, pp. 1308-1314(7)
- gli eventi remoti nel tempo si rappresentano mentalmente con + astrazione di quelli vicini
- McCrea &c provano che vale il contrario: livelli di percezione + astratti portano a maggior probabilitá di procrastinare
- (almeno x gli studenti di psicologia, come al solito uniche cavie sperimentali disponibili;-)

Google

Per FARE, pensa CONCRETO

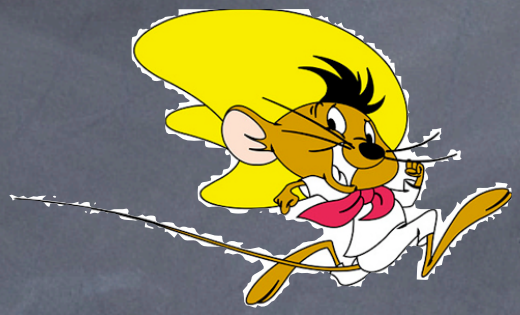
- Allen, "Getting Things Done":
  - qual é la SINGOLA PROSSIMA AZIONE?
- progetto d'interazione (e user-centered):
  - NON "l'utente", MA "Giovanni, mercante inesperto, con gran pratica di videogame" e/o "Marco, mercante stagionato che inizió al tempo di Hammurabi e ANCORA preferisce il cuneiforme, ..."
- "prefer action to abstr-action" (J. Fried, fondatore di "37 signals")

Google

Penalitá x l'Astrazione

- quando un linguaggio permette approcci a bassa E ad alta astrazione, puó esserci una penalitá x l'astrazione (Stepanov, http://std.dkuug.dk/JTC1/SC22/WG21/docs/PDTR18015.pdf & tanta ricerca successiva)
- questione di qualitá d'implementazione, non sempre vera: in Python siamo abituati a un *premio* di astrazione, non una penalitá
- spesso grazie a itertools, ma non sempre...

Itertools é una scheggia!



```
$ python -mtimeit 'for x in range(42): pass'
100000 loops, best of 3: 5.13 usec per loop

$ python -mtimeit 'for x in xrange(42): pass'
100000 loops, best of 3: 4.17 usec per loop

$ python -mtimeit -s'import itertools' \
> 'for x in itertools.repeat(None, 42): pass'
100000 loops, best of 3: 3.4 usec per loop
```

Google

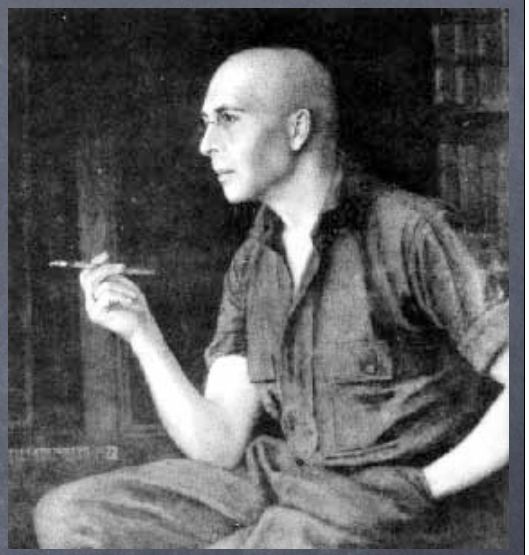
Ma anche il Martian Smilie!

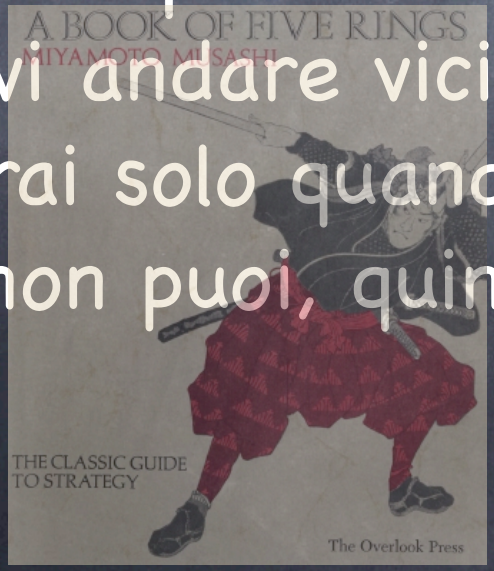
```
$ python -mtimeit -s'x="abracadabra"' \
> 'y="".join(reversed(x))'
100000 loops, best of 3: 5.96 usec per loop
$ python -mtimeit -s'x="abracadabra"' \
> 'y=x[::-1]'
1000000 loops, best of 3: 0.597 usec per loop
```

Google

Le Astrazioni Fan Acqua

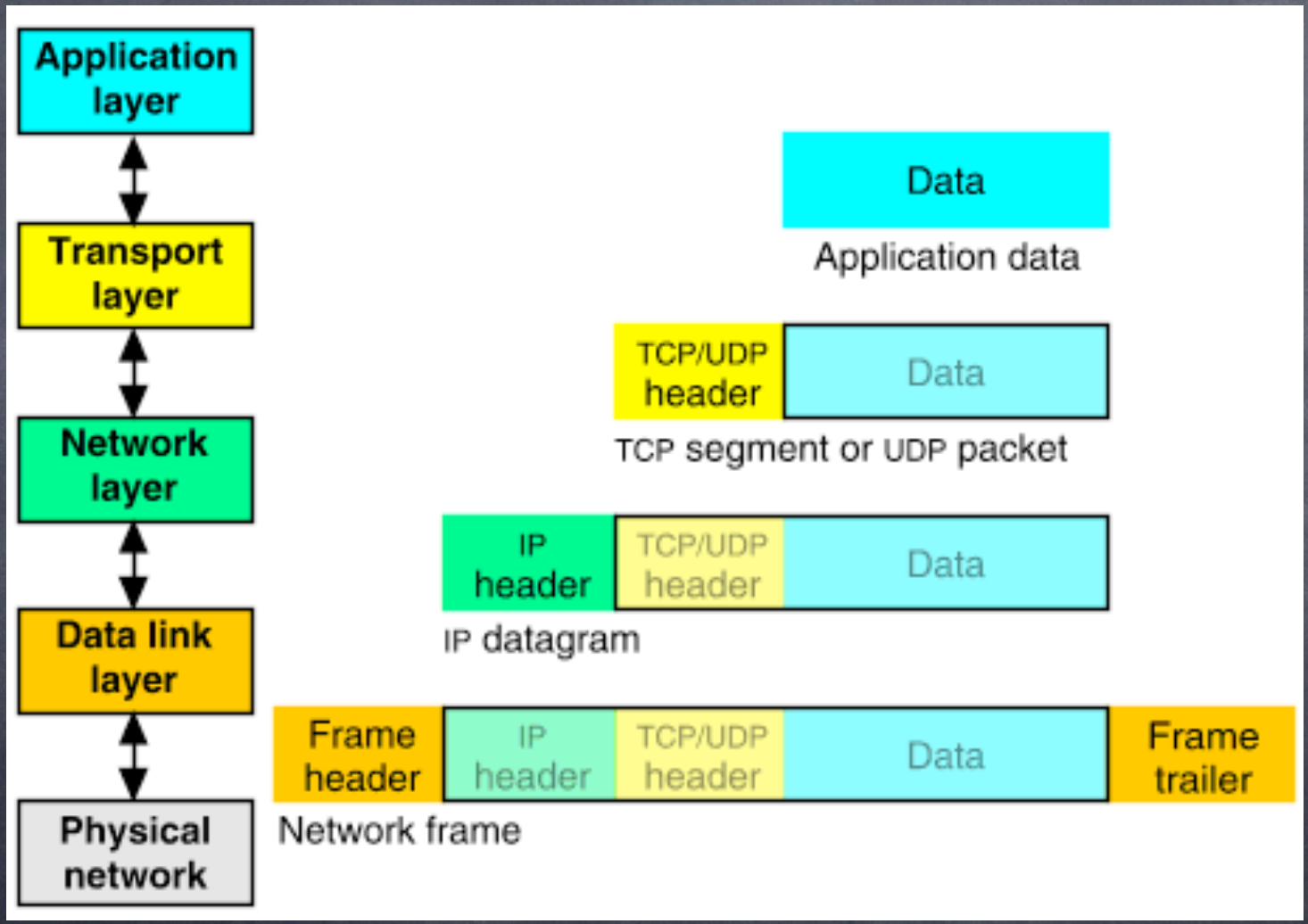
- le astrazioni fan acqua, perché...
  - ...*le astrazioni MENTONO*!
  - la mappa non é il territorio
- prima di potere astrarre,
  - devi capire i dettagli
- prima di poter arretrare,
  - devi andare vicino
- abstrai solo quando conosci tutti i dettagli
  - e non puoi, quindi, umiltá e flessibilitá!





Google

Splendida astrazione: TCP/IP

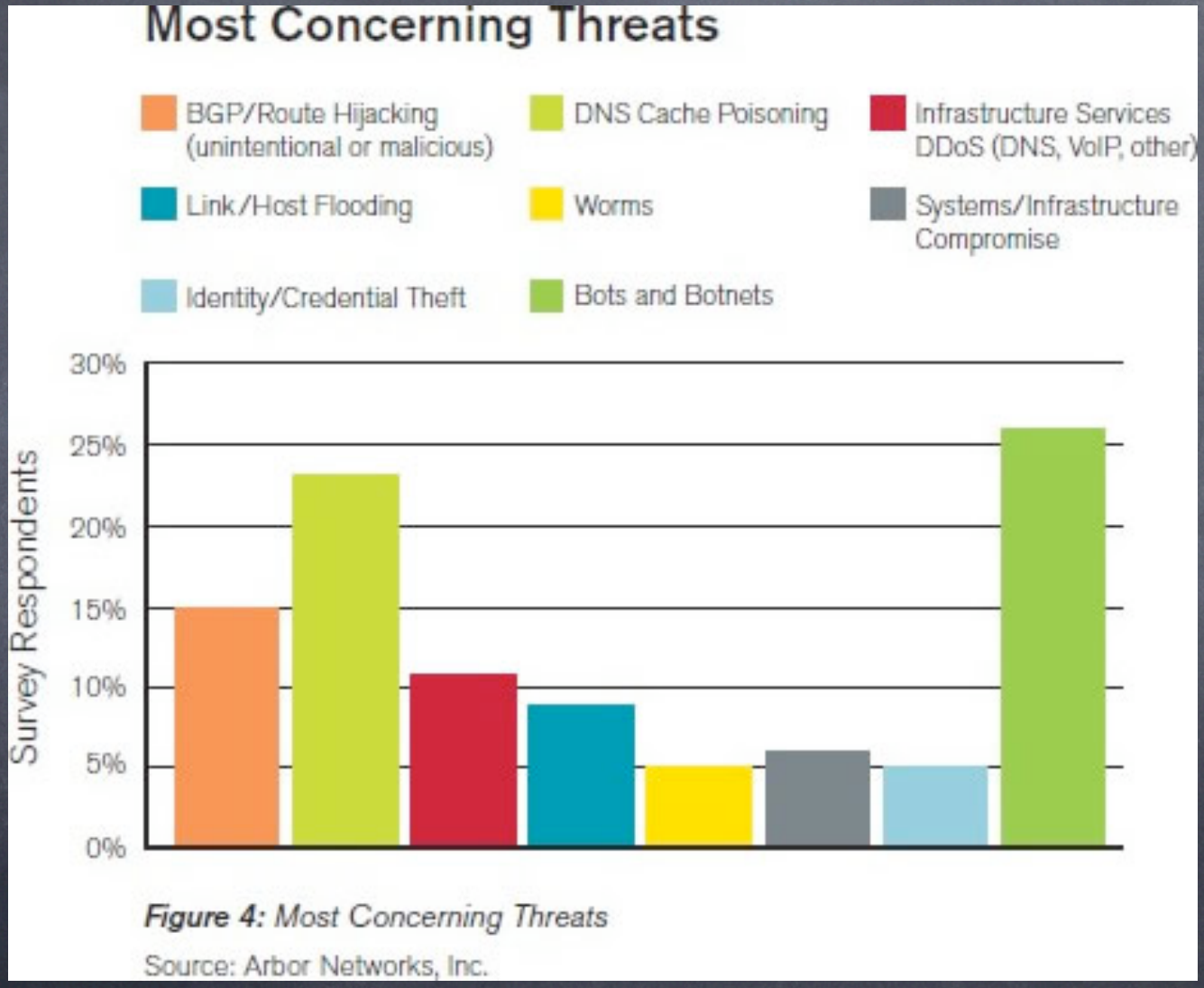


Google

Falle di TCP/IP: la FIDUCIA!

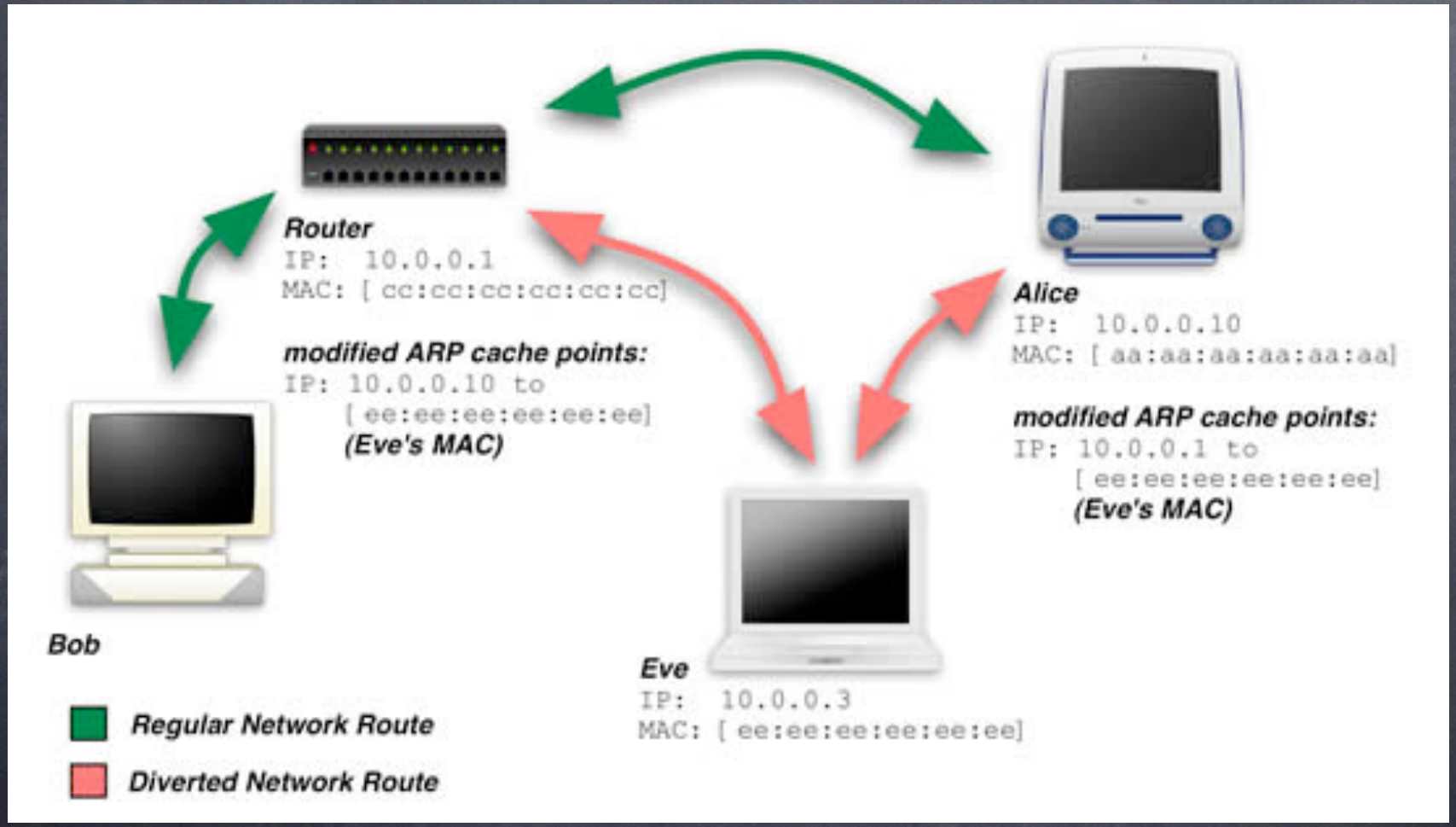
- TCP/IP, splendido stack d'astrazioni, MA...
- ...progettato in un'antica epoca di fiducia!
- L'intero stack "fa acqua" da ogni parte in termini di rischi di sicurezza:
  - "sotto" (avvelenamento di cache ARP)
  - "sopra" (avvelenamento di cache DNS),
  - "di fianco" (BGP menzognero),
  - "dentro" (sniffing, pwd FTP/Telnet, ...)
  - ...ecc, ecc...

TCP/IP oggi…:-(



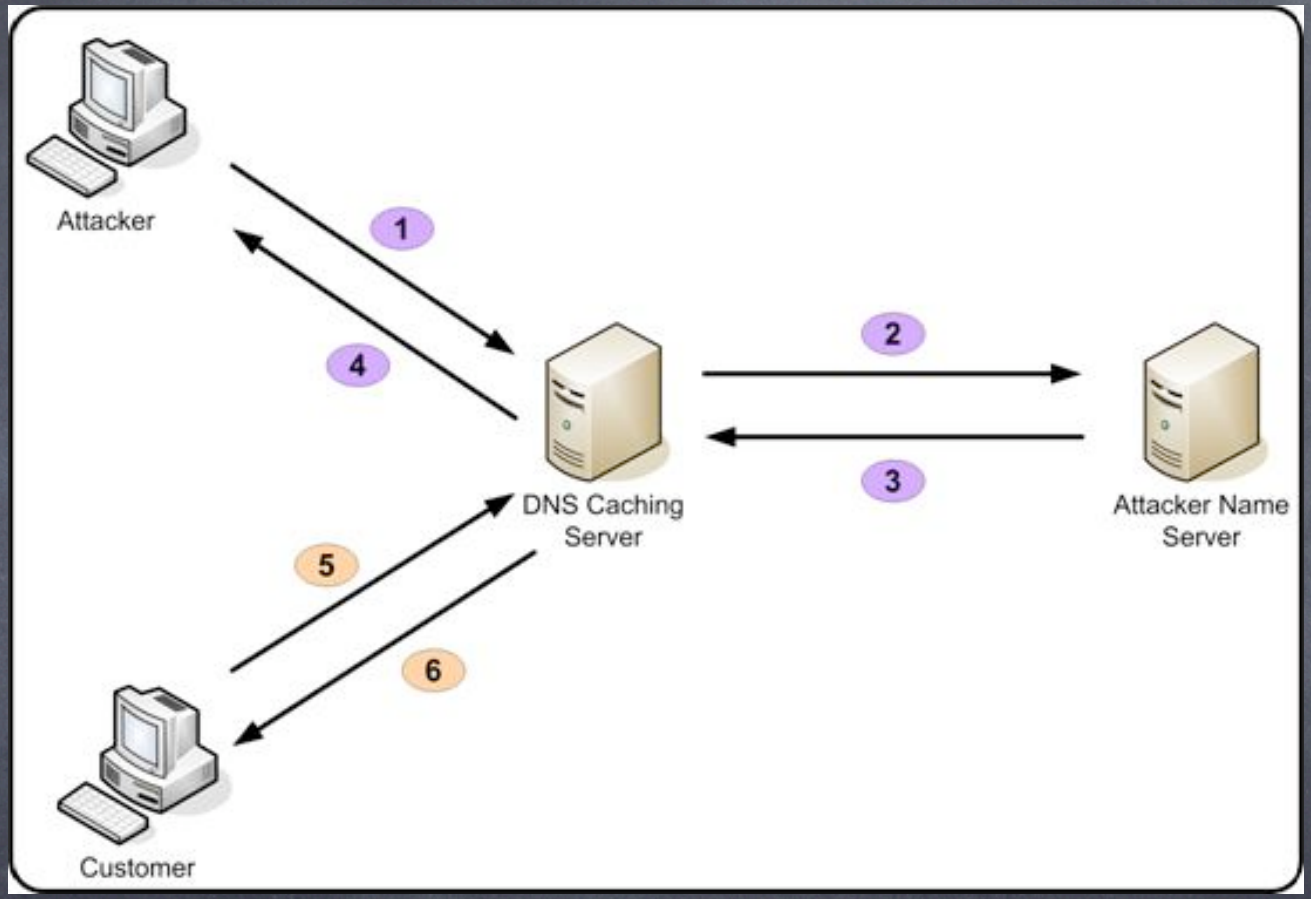
Google

Una "falla": veleno in ARP



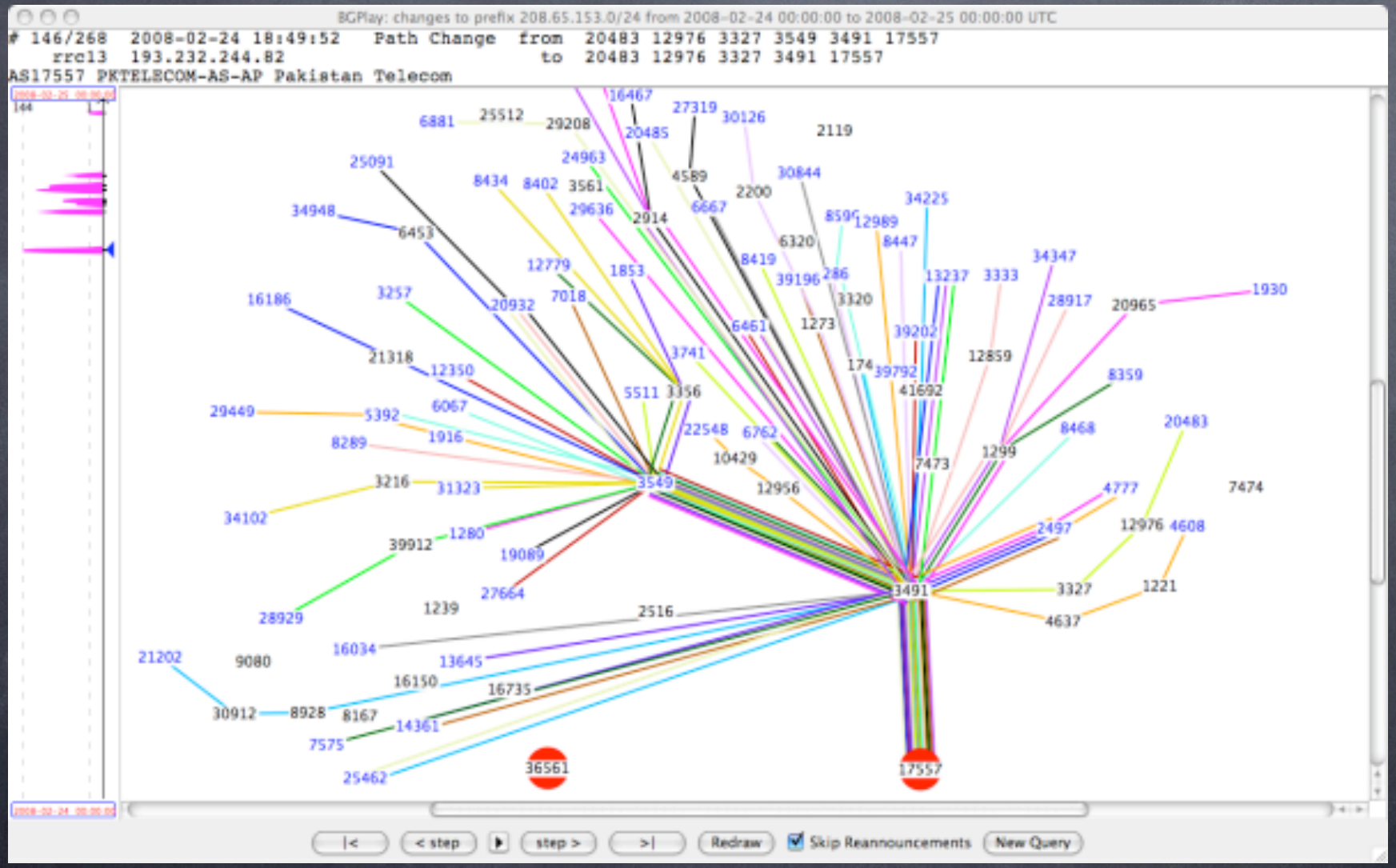
Google

Altra "falla": veleno in DNS



Google

La peggio: BGP Hijacking



...ma a volte CI VUOLE!

- esempio: filesystem remoti/distribuiti
  - di solito cercano di "emulare" quelli locali
  - meno locali sono, + costa "l'astrazione"
    - semantica, locking, affidabilitá, ...
  - "filesystem" é una splendida astrazione...
  - ma "filesystem locale" decisamente NO!
    - "mai sottoclassi di classi concrete" [Haahr]
- non vuol dire "l'astrazione é un male"
  - ma non BASTA avere SOLO l'astrazione
  - servono SISTEMATICHE "falle" da usare!

Mala Astrazione

- nel piccolo: 1 classe -> 1 interfaccia
  - inevitabilmente "galleggiano" dettagli di implementazione privi di reale rilevanza!
- nel medio: "ereditá da classi concrete"
  - una classe concreta (implementazione) non é MAI la base giusta da cui ereditare
- nel medio: errori di incapsulazione
  - e.g., finestre vs toolbar in MFC 4.*
- nel grande: il framework galleggiante
  - un "framework" con 1 sola applicazione...

Google

Usare BENE l'astrazione

- DEVI capire bene almeno 1-2 livelli SOTTO
- e x PROGETTARE un'ottima astrazione:
  - GRANDE familiaritá con VARIE sue possibili implementazioni ("strati sotto")
  - GRANDE familiaritá con VARI usi previsti (gli "strati superiori" che la USANO)
  - niente paraocchi, niente scorciatoie!
- TU puoi essere un implementatore o utente!
  - la Regola d'Oro é dunque un must;-)
- http://c2.com/cgi/wiki?TooMuchAbstraction

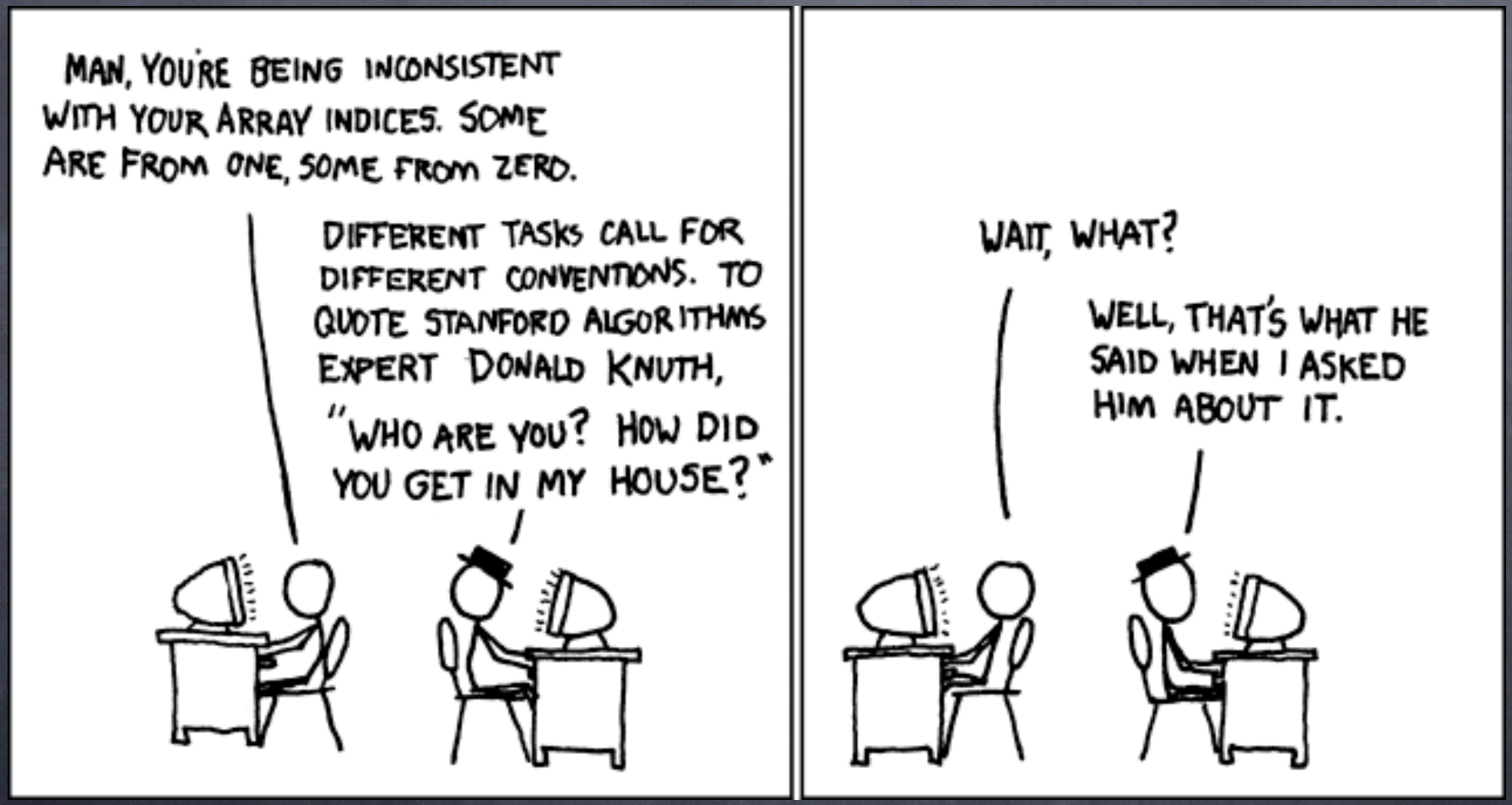
Che ne dice Donald Knuth

- the psychological profiling [[of the programmer]] is mostly the ability to shift levels of abstraction, from low level to high level. To see something in the small and to see something in the large. [[...]]
- Computer scientists see things simultaneously at the low level and the high level [[of abstraction]]

http://www.ddj.com/184409858

Google

Che altro dice Knuth...



Google

Che ne dice Jason Fried

- "Here's the problem with copying:
  - Copying skips understanding.
  - Understanding is how you grow.
  - You have to understand why something works or why something is how it is.
  - When you copy it, you miss that.
  - You just repurpose the last layer instead of understanding the layers underneath."
- Fare '%s/copy/use existing high-level abstractions blindly/g' ...;-)

http://www.37signals.com/svn/posts/1561-why-you-shouldnt-copy-us-or-anyone-else

Google

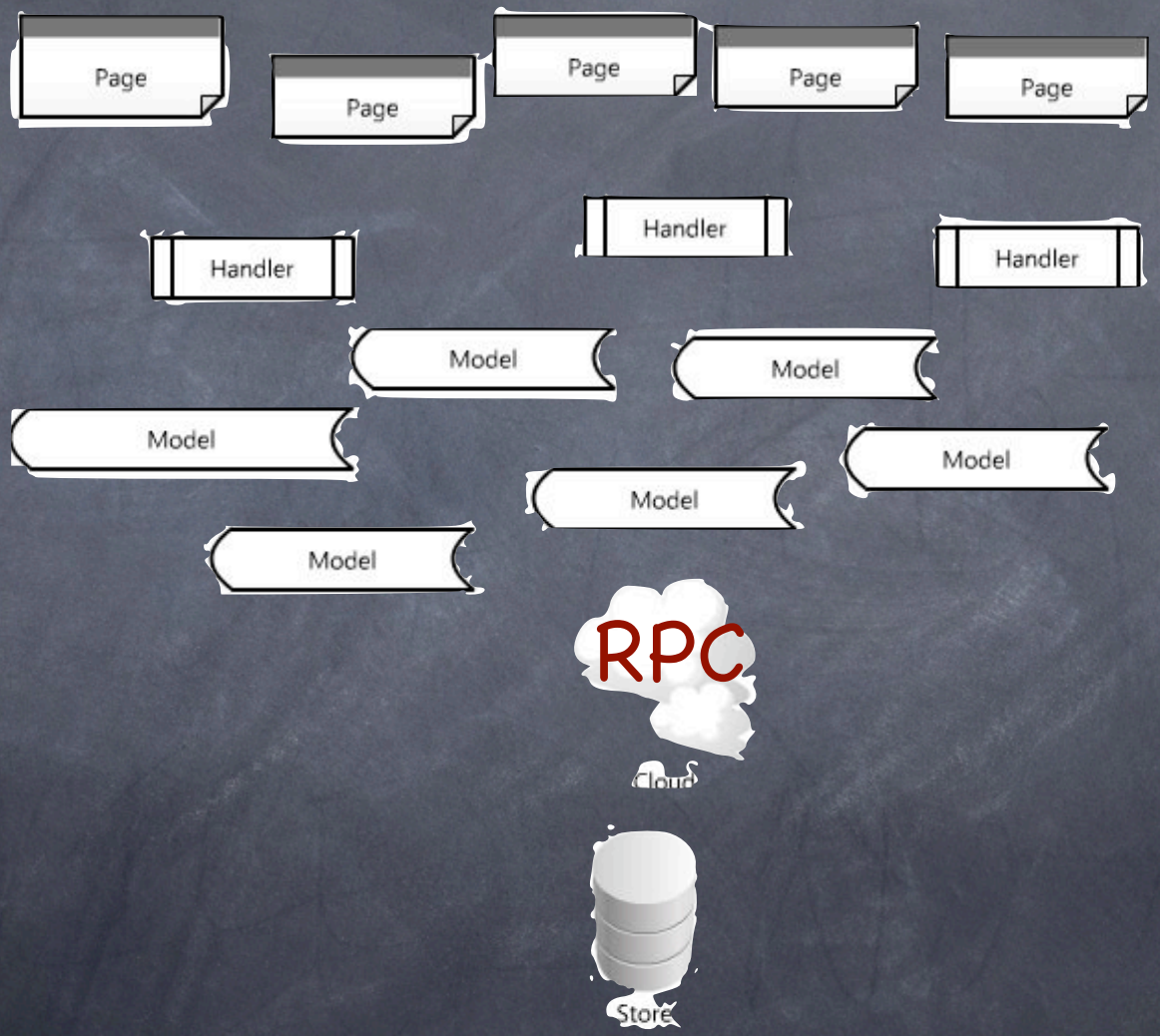
Che ne dice Jeff Attwood

- "don't reinvent the wheel,
  - unless you plan on learning more about wheels!"

http://www.codinghorror.com/blog/archives/001145.html

Google

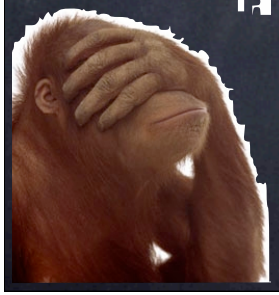
"Hack" di App Engine



Hack con Monkey-patch

- le operazioni vanno via uno strato RPC, apiproxy_stub_map.MakeSyncCall
- non consigliabile: il *monkey-patch*...:

```
from google.appengine.api import \
    apiproxy_stub_map
_org = apiproxy_stub_map.MakeSyncCall
def fake(svc, cal, req, rsp):
  ...
  x = _org(svc, cal, req, rsp)
  ...
apiproxy_stub_map.MakeSyncCall = fake
```



Google

Perché la scimmia é triste

```
class Client(object):
"""Memcache client object... """

def __init__(self, servers=None, debug=0,
     pickleProtocol=pickle.HIGHEST_PROTOCOL,
     pickler=pickle.Pickler,
     unpickler=pickle.Unpickler,
     pload=None,
     pid=None,
     make_sync_call=apiproxy_stub_map.MakeSyncCall):
 """Create a new Client object.... """
 ...
 self._make_sync_call = make_sync_call
```

Meglio: "Ganci" appositi

http://blog.appenginefan.com/2009/01/hacking-google-app-engine-part-1.html (e un GRAZIE a Jens Scheffler!)

```
from google.appengine.api import apiproxy_stub_map
def prehook(svc, cal, req, rsp):
  ...
apiproxy_stub_map.apiproxy.GetPreCallHooks(
  ).Append('unique_name', prehook, 'opt_api_id')
```

Google

Ma come fornire "ganci"?

- ...se non c'é un "fulcro naturale" tipo RPC?
- "colli di bottiglia" semanticamente cruciali
  - e.g.: se il sistema fa delle query SQL
    - pre-ganci sull'SQL, post- sui risultati
- approcci "a eventi" (Qt signal/slot)
- design patterns
  - ganci pre/post & eventi ~ Observer
  - Template Method (e.g. Queue.Queue)
  - Dependency Injection

Fornire ganci: scheduler

```
class ss(object):
  def __init__(self):
    self.i = itertools.count().next
    self.q = somemodule.PriorityQueue()
  def add_event(self, when, c, *a, **k):
    self.q.push((when, self.i(), c, a, k))
  def run(self):
    while self.q:
      when, n, c, a, k = self.q.pop()
      time.sleep(when - time.time())
      c(*a, **k)
```

(la PQ é "ovvia"...):

```
class PriorityQueue(object):
  def __init__(self):
    self.l = []
  def __len__(self):
    return len(self.l)
  def push(self, obj):
    heapq.heappush(self.l, obj)
  def pop(self):
    return heapq.heappop(self.l)
```

Bella astrazione, ma...

- ...come testare ss senza lunghe attese?
- ...come integrarlo con event-loop di altri sistemi, simulazioni, ecc...?

Problema: ss "dipende concretamente" da oggetti specifici (time.sleep e time.time).

Per "bucare l'astrazione", vediamo...:

1. lasciarlo al "Monkey Patching"
2. design pattern Dependency Injection

Monkey-patching...

```
import ss
class faker(object): pass
fake = faker()
ss.time = fake
fake.sleep = ...
fake.time = ...
```

- utile nelle emergenze, ma...
  - ...troppo spesso é una scusa x la pigrizia!-)
- sottile, nascosta "comunicazione" per vie oscure (explicit is better than implicit!-)
- si rompe con ottimizzazioni &c...

Google

Dependency Injection

```
class ss(object):
  def __init__(self, tm=time.time,
                     sl=time.sleep):
    self.tm = tm
    self.sl = sl
  ...
      self.sl(when - self.tm())
```

- ovvero, proprio come sched in libr.standard!-)

DI é un utile "gancio"

```
class faketime(object):
  def __init__(self, t=0.0): self.t = t
  def time(self): return self.t
  def sleep(self, t): self.t += t

f = faketime()
s = ss(f.time, f.sleep)
...
```


Google

Esempio di DI (app engine:-)

```
class Client(object):
"""Memcache client object... """

def __init__(self, servers=None, debug=0,
     pickleProtocol=pickle.HIGHEST_PROTOCOL,
     pickler=pickle.Pickler,
     unpickler=pickle.Unpickler,
     pload=None,
     pid=None,
     make_sync_call=apiproxy_stub_map.MakeSyncCall):
 """Create a new Client object.... """
 ...
 self._make_sync_call = make_sync_call
```

Q & A

http://www.aleax.it/itpyc_abst.pdf

Google