In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from pylab import *
from IPython.display import Image
import matplotlib.ticker as mtick

import scipy.stats as stats
import statsmodels.stats.weightstats as wstats
from collections import OrderedDict

from __future__ import print_function
from __future__ import division
%matplotlib inline
```

We want as many people as possible to sign-up with Facebook accounts, so that we can connect them with friends and also get more information about those users.

Therefore, we want to test two Sign-up with Facebook forms and see which one will have higher number of people actually connected with Facebook

In [2]:
```
Image('https://cloud.githubusercontent.com/assets/5244286/7023268/c8ac9664-dd2b-11e4-8c44-b9302f781a25.png', retina=True)
```

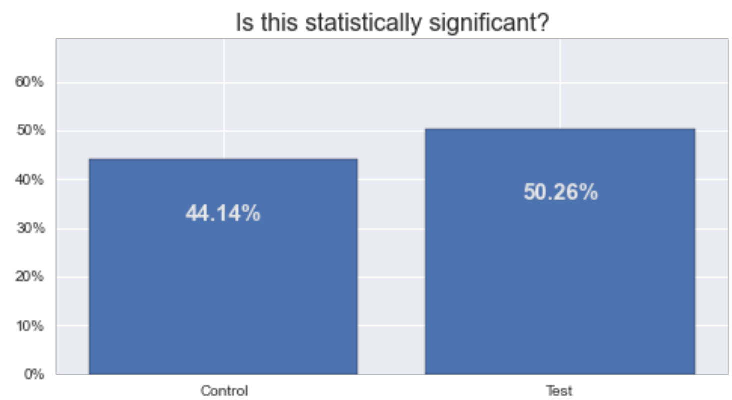
Out[2]:

Result of fake A/B test

In [4]:
```
# Generating fake data
control_installs = 2501
control_connected = 1104
test_installs = 2141
test_connected = 1076

print(' {}: installs = {} \t connected = {} \t prop = {}'
      .format('A', control_installs, control_connected, control_connected/control_installs))
```

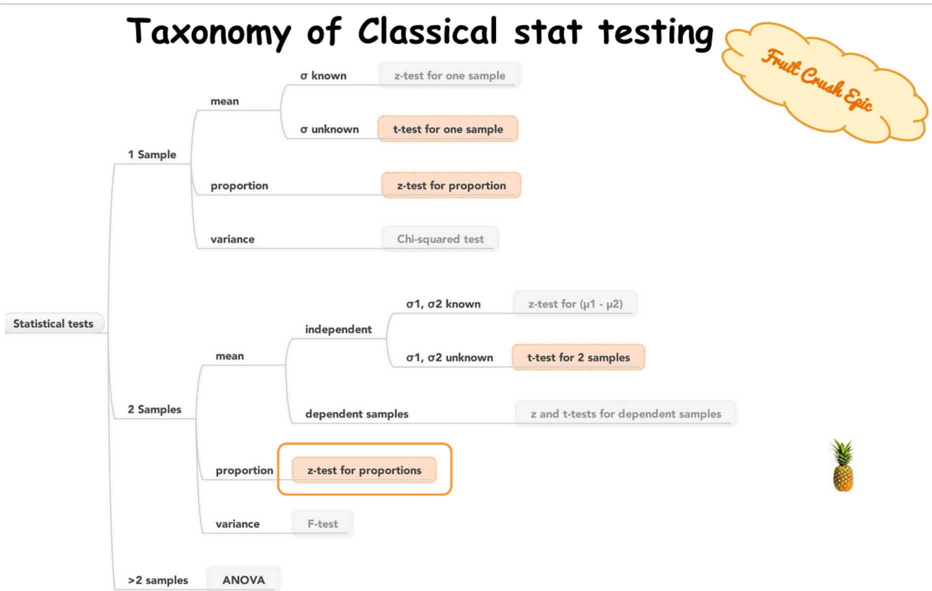
```
      .format('A', control_installs, control_connected, control_connected/control_installs))

print(' {}: installs = {} \t connected = {} \t prop = {}'
      .format('B', test_installs, test_connected, test_connected/test_installs))


fig, ax = plt.subplots(figsize=(8,4))

x = [0,1]
y = [control_connected/control_installs, test_connected/test_installs]
ax.bar(x, y, align='center', width=.8)
ax.set_xticks(x)
ax.set_xticklabels(['Control', 'Test'])
xlim(-.5,1.5)
ylim(0, .69)
for xx, yy in zip(x,y):
    ax.text(xx, yy*.7, '%0.2f%%'%(100*yy),ha='center', va='bottom', fontdict={'size':15,'weight':'
bold','color':(0.9,.9,.9)})
# ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.0f%%'))
# def perc(x, pos=0):
#     return '%0.0f%%'%(100*x)
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, pos=0: '%0.0f%%'%(100.0*x)))
title('Is this statistically significant?', fontdict={'size':16})
pass
# fig.savefig('03.01 two samples.png', bbox_inches='tight', pad_inches=0.2 ,dpi=200)
```

```
 A: installs = 2501      connected = 1104        prop = 0.441423430628
 B: installs = 2141      connected = 1076        prop = 0.502568893041
```



In [3]:
```
Image('https://cloud.githubusercontent.com/assets/5244286/7023317/3e8c68c8-dd2c-11e4-8735-51edd4b8
bfcb.png', retina=True)
```

Out[3]:



Two Samples z-test for Proportions

$z = \frac{\hat{p}_1 - \hat{p}_2}{\ldots}$

$$\sqrt{\hat{p}(1-\hat{p})(\frac{1}{n_1}+\frac{1}{n_2})}$$

where

$$\hat{p}_1 = \frac{x_1}{n_1},\ \hat{p}_2 = \frac{x_2}{n_2}$$

$$\hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$$

x_1, x_2 - number of successes in group 1 and 2

n_1, n_2 - number of observations in group 1 and 2

In [5]:
```
# implementation from scratch
def ztest_proportion_two_samples(x1, n1, x2, n2, one_sided=False):
    p1 = x1/n1
    p2 = x2/n2

    p = (x1+x2)/(n1+n2)
    se = p*(1-p)*(1/n1+1/n2)
    se = sqrt(se)

    z = (p1-p2)/se
    p = 1-stats.norm.cdf(abs(z))
    p *= 2-one_sided # if not one_sided: p *= 2
    return z, p

z,p = ztest_proportion_two_samples(control_connected, control_installs, test_connected, test_insta
lls, one_sided=False)
print(' z-stat = {z} \n p-value = {p}'.format(z=z,p=p))
```

```
 z-stat = -4.16111492042
 p-value = 3.16697658287e-05
```

In [6]:
```
# using statsmodels
from statsmodels.stats.proportion import proportions_ztest
count = np.array([control_connected,test_connected])
nobs = np.array([control_installs, test_installs])
z,p = proportions_ztest(count, nobs, value=0, alternative='two-sided')
print(' z-stat = {z} \n p-value = {p}'.format(z=z,p=p))
```

```
 z-stat = -4.16111492042
 p-value = 3.16697658288e-05
```

We can reject the null-hypothesis. Our second flow results in a better Facebook connection rate. We can even quantify the uplift!

So what is the uplift?

$$CI = (\hat{p}_1 - \hat{p}_2)\text{Unknown character}\text{Unknown character}z_{critical} \cdot SE$$

$$SE = \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$$

where

$\hat{p}_1$, $\hat{p}_2$ - proportion in group 1 and 2

n_1, n_2 - number of observations in group 1 and 2

In [7]:
```
def compute_standard_error_prop_two_samples(x1, n1, x2, n2, alpha=0.05):
    p1 = x1/n1
    p2 = x2/n2
    se = p1*(1-p1)/n1 + p2*(1-p2)/n2
    return sqrt(se)

def zconf_interval_two_samples(x1, n1, x2, n2, alpha=0.05):
    p1 = x1/n1
    p2 = x2/n2
    se = compute_standard_error_prop_two_samples(x1, n1, x2, n2)
    z_critical = stats.norm.ppf(1-0.5*alpha)
```

```
    return p2-p1-z_critical*se, p2-p1+z_critical*se

ci_low,ci_upp = zconf_interval_two_samples(control_connected, control_installs,
                                           test_connected, test_installs)
print(' 95% Confidence Interval = ( {0:.2f}% , {1:.2f}% )'
      .format(100*ci_low, 100*ci_upp))
```

```
 95% Confidence Interval = ( 3.24% , 8.99% )
```

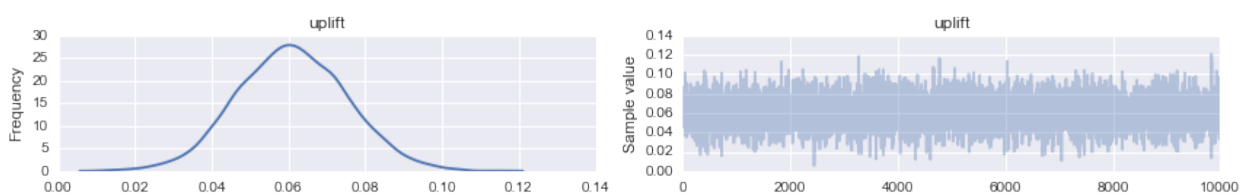
What's with all that Baesian stuff?

In [8]:
```
# using pymc3
import pymc as pm

control   = [1]*control_connected + [0]*(control_installs - control_connected)
treatment = [1]*test_connected + [0]*(test_installs - test_connected)
control = np.asarray(control)
treatment = np.asarray(treatment)

start = {}

start['p_C'] = (control).sum()/len(control)
start['p_T'] = (treatment).sum()/len(treatment)
```

In [9]:
```
with pm.Model() as model:
    p_C = pm.Uniform('p_C', 0.00001, .9)
    p_T = pm.Uniform('p_T', 0.00001, .9)
    # Record the difference between P1 and P2
    uplift = pm.Deterministic('uplift',p_T - p_C)

    # Set of observations, in this case we have two observation datasets
    obs_C = pm.Bernoulli("obs_C", p_C, observed=control)
    obs_T = pm.Bernoulli("obs_T", p_T, observed=treatment)

    # Inference
    step = pm.NUTS(scaling=start) # Start from a good starting point
    trace = pm.sample(10000, step, start=start, progressbar=True)

pass
```

```
 [-----------------100%-----------------] 10000 of 10000 complete in 7.6 sec
/Users/volodymyrkazantsev/anaconda/lib/python2.7/site-packages/theano/gof/cmodule.py:293: RuntimeW
arning: numpy.ndarray size changed, may indicate binary incompatibility
  rval = __import__(module_name, {}, {}, [module_name])
```

In [15]:
```
pm.traceplot(trace,['uplift']);
```



In [16]:
```
pm.stats.hpd(trace['uplift'], alpha=0.05)
```

Out[16]:
```
array([ 0.03415476,  0.09029491])
```

In [12]:
```
print( trace['p_C'].mean() )
print( trace['p_T'].mean() )
print( trace['uplift'].mean() )
```

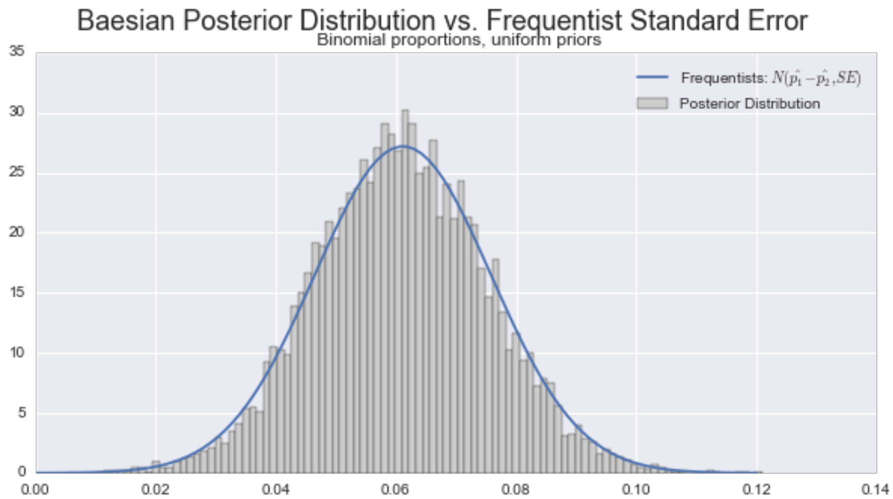
```
0.441487368335
0.502446401476
0.0609590331411
```

In [13]:
```
# Let's compare it against 95% CI calculated by Classical Statistics
ci_low,ci_upp = zconf_interval_two_samples(control_connected, control_installs, test_connected, te
st_installs)
print(' 95% Confidence Interval = ( {0:.2f}% , {1:.2f}% )'.format(100*ci_low, 100*ci_upp))
```

```
 95% Confidence Interval = ( 3.24% , 8.99% )
```

In [14]:
```
import matplotlib.mlab as mlab
```

In [14]:
```python
import matplotlib.mlab as mlab

mean = test_connected/test_installs - control_connected/control_installs
sigma = compute_standard_error_prop_two_samples(test_connected, test_installs, control_connected,
control_installs)

fig, ax = plt.subplots(figsize=(10,5))
x = np.linspace(0,.12,100)
plt.plot(x,mlab.normpdf(x,mean,sigma), label='Frequentists: $ N(\hat{p_1}-\hat{p_2}, SE)$')
hist(trace['uplift'], bins=100, normed=True, color='0.8', label='Posterior Distribution');
legend()
suptitle ('Baesian Posterior Distribution vs. Frequentist Standard Error', fontsize=18)
title(' Binomial proportions, uniform priors' )
pass
# fig.savefig('03.03 Bayesian CrI vs CI.png', dpi=200)
```



In [21]:
```python
# What is the probability that we gained less than +5% uplift in Facebook sign-ups?
(trace['uplift']<0.05).sum()/len(trace)
```

Out[21]: 0.2253