In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from qutip import *
import time

font = {'size'   : 18}

plt.rc('font', **font)
```

In [2]:
```
"""
bloch_vector: bloch vector of spin-1/2 moment
r: vector from spin to target position in nm

returns field in units of GHz * hbar/mu_B
"""
def dipole_field(bloch_vector, r):
    dist = np.linalg.norm(r)
    direction = r/dist

    mu0 = 4*np.pi*pow(10,-7) #vacuum permeability
    muB = 9.274*pow(10,-24) #bohr magneton SI
    hbar = 1.054571817*pow(10,-34) #hbar SI
    mag = -1*muB*bloch_vector #magnetic moment of a single electron

    return muB * mu0/(4*np.pi) * 1/hbar * (3*np.dot(mag,direction)*direction - mag)/(pow(dist,3) * pow(10,-27)) * pow(10,-9)
```

In [3]:
```
"""
Bext: external field strength (G)
times: list of times (1/GHz) over which to run experiment **todo - check if you're off by 2pi
modes: list of [[k, amplitude]] of magnon modes
a: inter-site magnon spacing (nm)
dist: sensor distance to index = 0 magnon site
indices: integer multiples of a at which we consider magnon field
max_magnon: maximum point in field-free exchange magnon dispersion (GHz)
mw_rabi: Rabi frequency associated with MW drive
mw_frequencies: list of CW frequencies over which we can sweep to look for the resonance (GHz)
D: zero field splititng in GHZ (2.87 for bare nv center)
"""
def run_expt(Bext, times, modes, a, dist, indices, max_magnon, mw_rabi, mw_frequencies, D = 2.87, output = False):

    hbar = 1.054571817*pow(10,-34) #hbar SI
    gamma = -28 #electron gyromagnetic ratio GHz/T

    def magnon_single_mode_field(Bext,t,k,a,dist,amplitude,indices,max_magnon):
        B_total = np.array([0.,0.,0.])
        #get frequency in ghz from magnon dispersion relation
        omega = max_magnon/2*(1 - np.cos(k*a)) + 2.8*pow(10,-3)*Bext*np.sqrt(1-amplitude**2)
        for index in indices:
            #bloch vector precession
            bloch_vector = np.array([amplitude*np.cos(index*k*a - omega*t),amplitude*np.sin(index*k*a - omega*t), np.sqrt(1-amplitude**2)])
            direction = np.array([-1*index*a, 0, dist])
            B_total += dipole_field(bloch_vector, direction)

        return B_total

    def total_coupling_field(t, Bext, a, dist, indices, modes, max_magnon):
        total_field = np.array([0.,0.,0.])

        for mode in modes:
            total_field += magnon_single_mode_field(Bext,t,mode[0],a,dist,mode[1],indices, max_magnon)

        return total_field

    B_diag = Bext * 2* 1.4 * pow(10,-3)

    H0 = Qobj([[0,0,0],[0,D-B_diag,0],[0,0,D+B_diag]])

    nv_init = Qobj([0,1,0])
    states = []

    """
    args to pass: mw_rabi, mw_frequency, Bext, a, dist, indices, modes, max_magnon
    """

    def spc(t, args):
        f = total_coupling_field(t, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        return f[0]/2+f[1]/(2j) + args['mw_rabi']*np.cos(args['mw_frequency']*t)

    def smc(t,args):
        f = total_coupling_field(t, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        return f[0]/2-f[1]/(2j) + args['mw_rabi']*np.cos(args['mw_frequency']*t)

    def szc(t, args):
        f = total_coupling_field(t, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        return f[2]

    for omega in mw_frequencies:
        Sp = Qobj([[0,np.sqrt(2),0],[0,0,0],[np.sqrt(2),0,0]])
        Sm = Qobj([[0,0,np.sqrt(2)],[np.sqrt(2),0,0],[0,0,0]])
        Sz = Qobj([[0,0,0],[0,-1,0],[0,0,1]])


        H = [H0,[Sp,spc],[Sm,smc], [Sz, szc]]
        st = time.time()

        out = sesolve(H, nv_init, times,args={'mw_rabi':mw_rabi, 'mw_frequency':omega, 'Bext':Bext, 'a':a, 'dist':dist, 'indices':indices, 'modes':modes, 'max_magnon':max_mag
        end = time.time()
        args = {'mw_rabi':mw_rabi, 'mw_frequency':omega, 'Bext':Bext, 'a':a, 'dist':dist, 'indices':indices, 'modes':modes, 'max_magnon':max_magnon}
        f = total_coupling_field(0, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        if output:
            print(omega, f, end-st)
        states.append(out.states)
    return states
```

In [4]:
```
#Total magnon + MW field used to drive system leading to off diagonal matrix element
def drive_only(t, args):
    mw_rabi = 0.2 #100 MHz Rabi frequency of drive field
    mw_frequency = args['omega']

    return mw_rabi*np.cos(mw_frequency*t)
```

In [5]:
```
def magnon_single_mode_field(Bext,t,k,a,dist,amplitude,indices,max_magnon):
        B_total = np.array([0.,0.,0.])
        #get frequency in ghz from magnon dispersion relation
        omega = max_magnon/2*(1 - np.cos(k*a)) + 2.8*pow(10,-3)*Bext*np.sqrt(1-amplitude**2)
        for index in indices:
            #bloch vector precession
            bloch_vector = np.array([amplitude*np.cos(index*k*a - omega*t),amplitude*np.sin(index*k*a - omega*t), np.sqrt(1-amplitude**2)])
            direction = np.array([-1*index*a, 0, dist])
            B_total += dipole_field(bloch_vector, direction)

        return B_total

def total_coupling_field(t, Bext, a, dist, indices, modes, max_magnon):
    total_field = np.array([0.,0.,0.])

    for mode in modes:
        total_field += magnon_single_mode_field(Bext,t,mode[0],a,dist,mode[1],indices, max_magnon)

    return total_field
```

In [8]:
```
mcc_dist_k_results = {}
for k in np.linspace(np.pi/3,2*np.pi,10)[2:5]:
    print(k)
    omega_res = 100*(1 - np.cos(k/2))
    out = run_expt(20, np.linspace(0,100,500), [[k,0.2]], 0.5,1, np.linspace(-5,5,11), 200, 0.5, np.linspace(omega_res +1,omega_res +7,50), D = 3.63)
    mcc_dist_k_results[k] = [max([abs(out[i][j][0])**2 for j in range(len(out[i]))]) for i in range(len(out))]
```

```
2.210750385859484
2.7925268031909276
3.3743032205223704
```

In [13]:
```
nv_dist_k_results = {}
for k in np.linspace(np.pi/3,2*np.pi,10)[2:5]:
    print(k)
    omega_res = 100*(1 - np.cos(k/2))
    out = run_expt(20, np.linspace(0,100,500), [[k,0.2]], 0.5,10, np.linspace(-5,5,11), 200, 0.5, np.linspace(omega_res +1,omega_res +7,50))
    nv_dist_k_results[k] = [max([abs(out[i][j][0])**2 for j in range(len(out[i]))]) for i in range(len(out))]
```

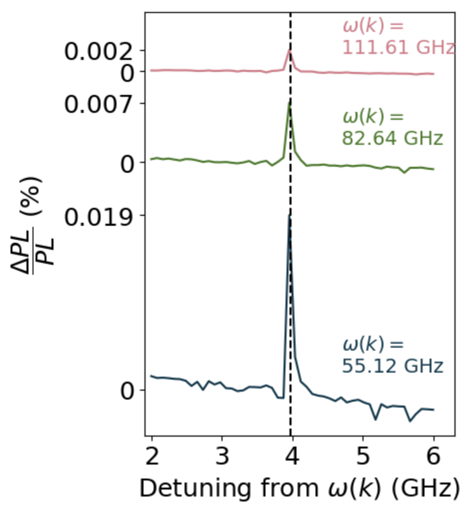
```
2.210750385859484
2.7925268031909276
3.3743032205223704
```

In [36]:
```
fig,ax = plt.subplots(figsize = (4,5.5))
ki = 0

cmap = plt.get_cmap('cubehelix')
my_colors = [cmap(int(i)) for i in np.linspace(50,150,3)]
y_off_total = 0
curr_y_off = 0.025
yticks = []
ylabels = []
for key in list(mcc_dist_k_results.keys()):
    omega_res = 100*(1 - np.cos(key/2))
    mcc_dist_results = mcc_dist_k_results[key]
    mcc_contrast = [(i-np.mean(mcc_dist_results))/(1-np.mean(mcc_dist_results))*100 + y_off_total for i in mcc_dist_results]
    plt.text(4.7,y_off_total+0.002,"$\omega(k) = $" + "\n" + str(round(omega_res,2)) + " GHz",color = my_colors[ki],size = 14)
    yticks.append(y_off_total)
    yticks.append(max(mcc_contrast)[0])
    ylabels.append(0)
    ylabels.append(round(max(mcc_contrast)[0]-y_off_total, 3))
    y_off_total += curr_y_off
    curr_y_off = 0.01
    plt.plot(np.linspace(2,6,50), mcc_contrast, label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz",color = my_colors[ki])
#     plt.plot(np.linspace(2,6,50), nv_contrast, "--",label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz",color = my_colors[ki])

    ki += 1
#plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xlim(1.9,6.3)
plt.xticks([2,3,4,5,6])
plt.ylabel("$\dfrac{\Delta PL}{PL}$ (%)")
plt.plot([3.63+0.35,3.63+0.35],[-0.1,0.18],"--",color = "black")
#plt.plot([2.81,2.81],[-0.1,0.18],"--",color = "black")
plt.ylim([-0.005,0.0415])
plt.yticks(yticks, labels = ylabels)
plt.xlabel("Detuning from $\omega(k)$ (GHz)")
```

Out[36]:
```
Text(0.5, 0, 'Detuning from $\\omega(k)$ (GHz)')
```

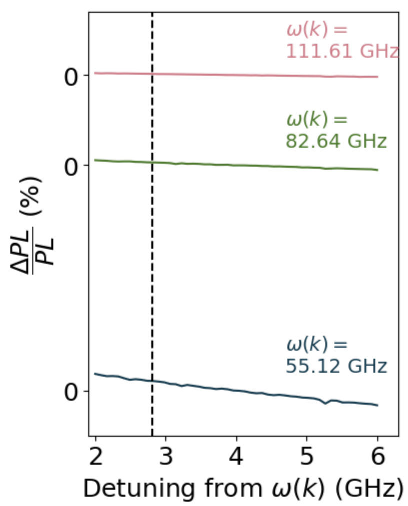


In [33]:
```
fig,ax = plt.subplots(figsize = (4,5.5))
ki = 0

cmap = plt.get_cmap('cubehelix')
my_colors = [cmap(int(i)) for i in np.linspace(50,150,3)]
y_off_total = 0
curr_y_off = 0.025
yticks = []
ylabels = []
for key in list(mcc_dist_k_results.keys()):
    omega_res = 100*(1 - np.cos(key/2))
    mcc_dist_results = nv_dist_k_results[key]
    mcc_contrast = [(i-np.mean(mcc_dist_results))/(1-np.mean(mcc_dist_results))*100 + y_off_total for i in mcc_dist_results]
    plt.text(4.7,y_off_total+0.002,"$\omega(k) = $" + "\n" + str(round(omega_res,2)) + " GHz",color = my_colors[ki],size = 14)
    yticks.append(y_off_total)
    #yticks.append(max(mcc_contrast)[0])
    ylabels.append(0)
    #ylabels.append(round(max(mcc_contrast)[0]-y_off_total, 3))
    y_off_total += curr_y_off
    curr_y_off = 0.01
    plt.plot(np.linspace(2,6,50), mcc_contrast, label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz",color = my_colors[ki])
#     plt.plot(np.linspace(2,6,50), nv_contrast, "--",label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz",color = my_colors[ki])

    ki += 1
#plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xlim(1.9,6.3)
plt.xticks([2,3,4,5,6])
plt.ylabel("$\dfrac{\Delta PL}{PL}$ (%)")
#plt.plot([3.63+0.35,3.63+0.35],[-0.1,0.18],"--",color = "black")
plt.plot([2.81,2.81],[-0.1,0.18],"--",color = "black")
plt.ylim([-0.005,0.042])
plt.yticks(yticks, labels = ylabels)
plt.xlabel("Detuning from $\omega(k)$ (GHz)")
```

Out[33]: Text(0.5, 0, 'Detuning from $\\omega(k)$ (GHz)')



In [ ]: