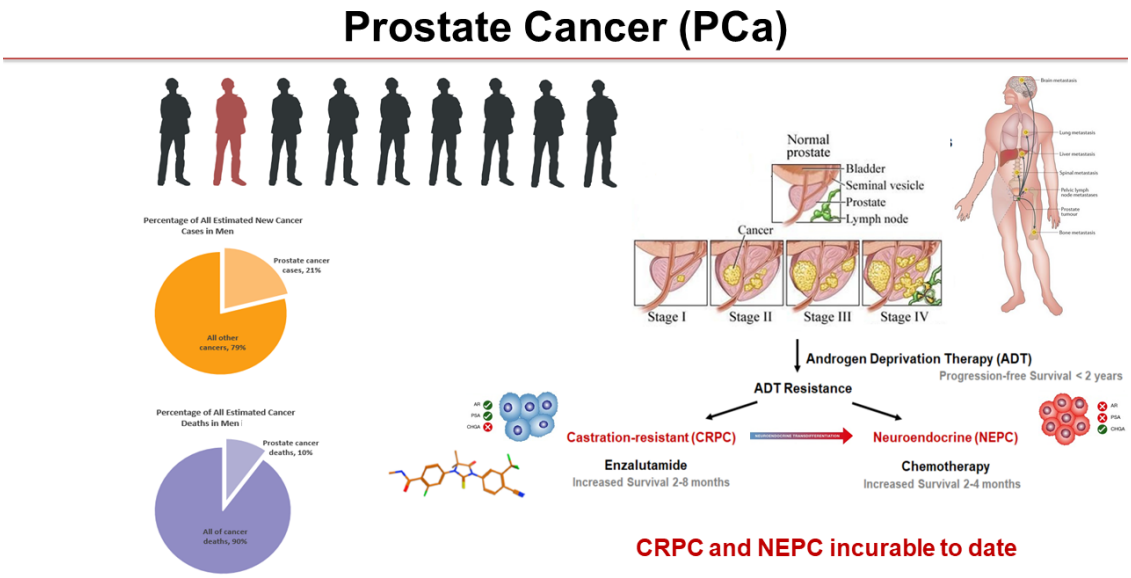
Myc-Max_Structural_Data

April 8, 2022

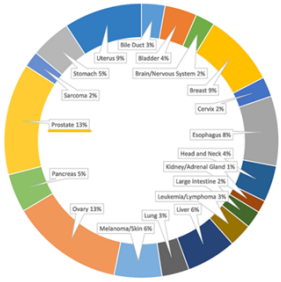


0.0.1 Urgent unmet clinical need for innovative therapeutics for management of lethal CRPC and NEPC

0.0.2 Strategy: Target the oncogenic activity of Myc - the most sought after drug targets, causally implicated in most if not all human cancers, often correlated with disease aggressiveness

Myc – Major oncogenic transcription factors

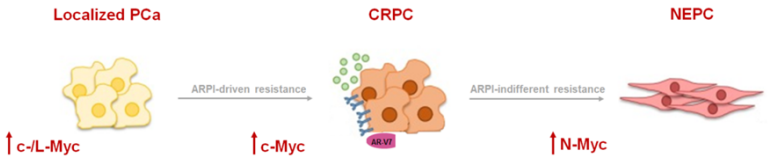
Myc amplifications



Amplifications of Myc
75% all human cancers; 13% prostate

Pathogenesis
Progression
Therapy-resistance
Lethal outcomes

Ideal therapeutic targets in advanced PCa



c-Myc -> knockdown positively regulates AR/AR-V7 expression & stability

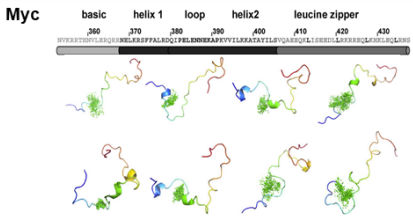
c-Myc -> inhibition alleviates enzalutamide-resistance

c-Myc -> stimulates AR-V7 alternative splicing

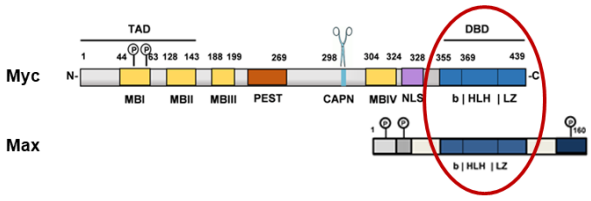
N-Myc -> induces neuroendocrine differentiation

Challenges in drugging the "undruggable" Myc

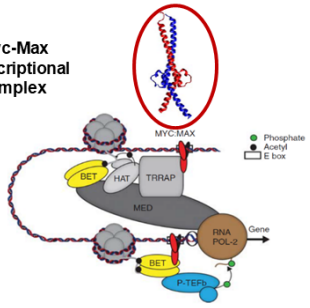
- Intrinsically disordered protein (IDP).
- No effective binding pockets on Myc surface alone.



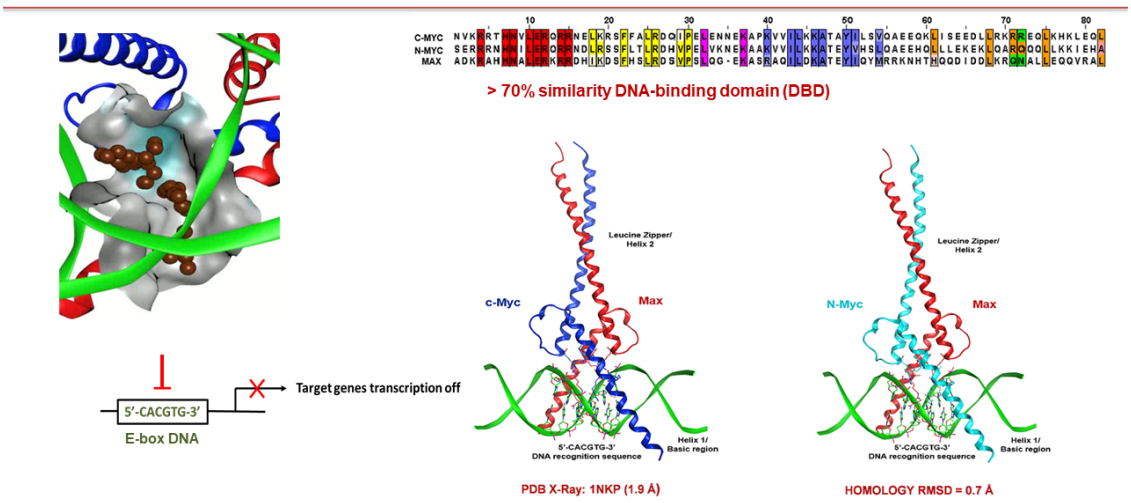
- MUST FORM Myc-Max functional sequence-specific DNA-binding domain for transcriptional activation and oncogenic transformation.



Myc-Max transcriptional complex



- Lack of high-quality structures of Myc-Max complex ligated with small molecule inhibitors.



0.1 X-ray structure of c-Myc-Max heterodimer in complex with DNA

0.1.1 PDB ID: 1NKP

[1]:
```
import Bio.PDB.PDBParser
import itertools
```

[2]:
```
fn = '1NKP.pdb'
```

[3]:
```
parser = Bio.PDB.PDBParser(QUIET=True)
structure = parser.get_structure(fn.split('.')[0], fn)
```

[4]:
```
import nglview as nv

view = nv.show_biopython(structure)
display(view)
```

```
NGLWidget()
```

[5]:
```
name = structure.header["name"]
print(name)

keywords = structure.header["keywords"]
print(keywords)

[chain.get_id() for chain in structure[0]]
```

```
crystal structure of myc-max recognizing dna
transcription, dna, bhlhz, oncogene, heterodimer, transcription-dna complex
```

[5]:
```
['F', 'G', 'A', 'B']
```

0.2 c-Myc-Max heterodimer

[6]:
```
from Bio.PDB import PDBIO, Select

class PolypeptideSelect(Select):
    def accept_chain(self, chain):
        if chain.get_id() in ['A', 'B']:
            return 1
        else:
            return 0

io = PDBIO()
io.set_structure(structure)
io.save('1nkp_ab.pdb', PolypeptideSelect())
```

[7]:
```
fn = '1nkp_ab.pdb'
structure = Bio.PDB.PDBParser(QUIET=True).get_structure(fn.split('.')[0], fn)

view = nv.show_biopython(structure)
display(view)
```

```
NGLWidget()
```

0.3 Residues

[8]:
```
class ResidueSelect(Select):
    def accept_residue(self, residue):
        if residue.get_resname() != 'HOH':
            return 1
        else:
            return 0

io = PDBIO()
io.set_structure(structure)
io.save('1nkp_ab_noHOH.pdb', ResidueSelect())
```

[9]:
```
fn1 = '1NKP_ab_noHOH.pdb'
structure1 = Bio.PDB.PDBParser(QUIET=True).get_structure(fn1.split('.')[0], fn1)
residues1 = structure1[0]["A"] #a generator
residues1_list = list(residues1)
print('c-Myc residues: ' + str(len(residues1_list)))
[res.get_resname() + str(res.get_id()[1]) for res in residues1_list]
```

```
    c-Myc residues: 88
```

```
[9]: ['GLY897',
      'HIS898',
      'MET899',
      'ASN900',
      'VAL901',
      'LYS902',
      'ARG903',
      'ARG904',
      'THR905',
      'HIS906',
      'ASN907',
      'VAL908',
      'LEU909',
      'GLU910',
      'ARG911',
      'GLN912',
      'ARG913',
      'ARG914',
      'ASN915',
      'GLU916',
      'LEU917',
      'LYS918',
      'ARG919',
      'SER920',
      'PHE921',
      'PHE922',
      'ALA923',
      'LEU924',
      'ARG925',
      'ASP926',
      'GLN927',
      'ILE928',
      'PRO929',
      'GLU930',
      'LEU931',
      'GLU932',
      'ASN933',
      'ASN934',
      'GLU935',
      'LYS936',
      'ALA937',
      'PRO938',
      'LYS939',
      'VAL940',
      'VAL941',
```

```
 'ILE942',
 'LEU943',
 'LYS944',
 'LYS945',
 'ALA946',
 'THR947',
 'ALA948',
 'TYR949',
 'ILE950',
 'LEU951',
 'SER952',
 'VAL953',
 'GLN954',
 'ALA955',
 'GLU956',
 'GLU957',
 'GLN958',
 'LYS959',
 'LEU960',
 'ILE961',
 'SER962',
 'GLU963',
 'GLU964',
 'ASP965',
 'LEU966',
 'LEU967',
 'ARG968',
 'LYS969',
 'ARG970',
 'ARG971',
 'GLU972',
 'GLN973',
 'LEU974',
 'LYS975',
 'HIS976',
 'LYS977',
 'LEU978',
 'GLU979',
 'GLN980',
 'LEU981',
 'GLY982',
 'GLY983',
 'CYS984']
```

[10]:
```
residues2 = structure1[0]["B"]
residues2_list = list(residues2)
print('Max residues: ' + str(len(residues2_list)))
```

```
[res.get_resname() + str(res.get_id()[1]) for res in residues2_list]
```

```
Max residues: 83
```

[10]:
```
['ASP202',
 'LYS203',
 'ARG204',
 'ALA205',
 'HIS206',
 'HIS207',
 'ASN208',
 'ALA209',
 'LEU210',
 'GLU211',
 'ARG212',
 'LYS213',
 'ARG214',
 'ARG215',
 'ASP216',
 'HIS217',
 'ILE218',
 'LYS219',
 'ASP220',
 'SER221',
 'PHE222',
 'HIS223',
 'SER224',
 'LEU225',
 'ARG226',
 'ASP227',
 'SER228',
 'VAL229',
 'PRO230',
 'SER231',
 'LEU232',
 'GLN233',
 'GLY234',
 'GLU235',
 'LYS236',
 'ALA237',
 'SER238',
 'ARG239',
 'ALA240',
 'GLN241',
 'ILE242',
 'LEU243',
 'ASP244',
```

```
 'LYS245',
 'ALA246',
 'THR247',
 'GLU248',
 'TYR249',
 'ILE250',
 'GLN251',
 'TYR252',
 'MET253',
 'ARG254',
 'ARG255',
 'LYS256',
 'ASN257',
 'HIS258',
 'THR259',
 'HIS260',
 'GLN261',
 'GLN262',
 'ASP263',
 'ILE264',
 'ASP265',
 'ASP266',
 'LEU267',
 'LYS268',
 'ARG269',
 'GLN270',
 'ASN271',
 'ALA272',
 'LEU273',
 'LEU274',
 'GLU275',
 'GLN276',
 'GLN277',
 'VAL278',
 'ARG279',
 'ALA280',
 'LEU281',
 'GLY282',
 'GLY283',
 'CYS284']
```

0.4 N-Myc-Max homology model

[11]:
```
# MODELLER script used to generate N-Myc homology models
# using c-Myc-Max structure as template

'''
```

```
#Homology modeling by the Modeller automodel class
from modeller import *              # Load standard Modeller classes
from modeller.automodel import *    # Load the automodel class

log.verbose()    # request verbose output
env = environ()  # create a new MODELLER environment to build this model in

#directories for input atom files
env.io.atom_files_directory = ['.', '../atom_files']

#env.io.hetatm = True

a = automodel(env,
              alnfile  = 'alignment.ali',          # alignment filename
              knowns   = '1nkp',                   # codes of the template(s)
              sequence = 'n_homology_model',       # code of the target
              assess_methods = (assess.DOPE))      # energy score
a.starting_model= 1                                # index of the first model
a.ending_model  = 10                               # index of the last model␣
 ↪(how many models to calculate)


#Through VTFM optimization:
a.library_schedule=autosched.slow
a.max_var_iterations = 300

#Through MD optimization:
a.md_level = refine.slow

a.make()                                           # do the actual homology␣
 ↪modeling
'''
```

[11]: "\n#Homology modeling by the Modeller automodel class \nfrom modeller import *
Load standard Modeller classes\nfrom modeller.automodel import *   # Load the
automodel class\n\nlog.verbose()   # request verbose output\nenv = environ() #
create a new MODELLER environment to build this model in\n\n#directories for
input atom files\nenv.io.atom_files_directory = ['.',
'../atom_files']\n\n#env.io.hetatm = True\n\na = automodel(env,\n
alnfile = 'alignment.ali',   # alignment filename\n   knowns
= '1nkp',   # codes of the template(s)\n   sequence =
'n_homology_model',   # code of the target\n   assess_methods =
(assess.DOPE))   # energy score\na.starting_model= 1
index of the first model\na.ending_model = 10
index of the last model (how many models to calculate)\n
\n\n#Through VTFM
optimization:\na.library_schedule=autosched.slow\na.max_var_iterations =

```
300\n\n#Through MD optimization:\na.md_level = refine.slow\n\na.make()
# do the actual homology modeling\n"
```

[12]:
```
# Content of the structure-sequence alignment file

with open('alignment.ali') as f:
    aln = f.read()
    print(aln)
```

```
>P1;1NKP
structureX:1NKP:6    :A:88   :A::::

GHMNVKRRTHNVLERQRRNELKRSFFALRDQIPELENNEKAPKVVILKKATAYILSVQAEEQKLISEEDLLRKRREQLKH
KLEQLGGC*

>P1;MYCN

sequence:MYCN:6    : :88   : ::::
SEDSERRRNHNILERQRRNDLRSSFLTLRDHVPELVKNEKAAKVVILKKATEYVHSLQAEEHQLLLEKEKLQARQQQLLK
KIEHARTC*
```

[13]:
```
fn2 = 'n_homology_model.pdb'
structure2 = Bio.PDB.PDBParser(QUIET=True).get_structure(fn2.split('.')[0], fn2)

view = nv.show_biopython(structure2)
view
```

```
NGLWidget()
```

[14]:
```
residues2 = structure2[0][" "]

residues2_list = list(residues2)
print('N-Myc residues: ' + str(len(residues2_list)))
[res.get_resname() + str(res.get_id()[1]) for res in residues2_list]
```

```
N-Myc residues: 88
```

[14]:
```
['SER1',
 'GLU2',
 'ASP3',
 'SER4',
 'GLU5',
 'ARG6',
 'ARG7',
 'ARG8',
 'ASN9',
```

```
'HIS10',
'ASN11',
'ILE12',
'LEU13',
'GLU14',
'ARG15',
'GLN16',
'ARG17',
'ARG18',
'ASN19',
'ASP20',
'LEU21',
'ARG22',
'SER23',
'SER24',
'PHE25',
'LEU26',
'THR27',
'LEU28',
'ARG29',
'ASP30',
'HIS31',
'VAL32',
'PRO33',
'GLU34',
'LEU35',
'VAL36',
'LYS37',
'ASN38',
'GLU39',
'LYS40',
'ALA41',
'ALA42',
'LYS43',
'VAL44',
'VAL45',
'ILE46',
'LEU47',
'LYS48',
'LYS49',
'ALA50',
'THR51',
'GLU52',
'TYR53',
'VAL54',
'HIS55',
'SER56',
```

```
 'LEU57',
 'GLN58',
 'ALA59',
 'GLU60',
 'GLU61',
 'HIS62',
 'GLN63',
 'LEU64',
 'LEU65',
 'LEU66',
 'GLU67',
 'LYS68',
 'GLU69',
 'LYS70',
 'LEU71',
 'GLN72',
 'ALA73',
 'ARG74',
 'GLN75',
 'GLN76',
 'GLN77',
 'LEU78',
 'LEU79',
 'LYS80',
 'LYS81',
 'ILE82',
 'GLU83',
 'HIS84',
 'ALA85',
 'ARG86',
 'THR87',
 'CYS88']
```

0.5 Quality of the N-Myc-Max homology model

0.5.1 Superimpose the structures and calculate RMS deviation (RMSD) between atomic coordinates

[15]:
```
ppb = Bio.PDB.PPBuilder()

query = ppb.build_peptides(structure1)[0][:]
target = ppb.build_peptides(structure2)[1][:]

query_atoms = [r['CA'] for r in query]
target_atoms = [r['CA'] for r in target]

superimposer = Bio.PDB.Superimposer()
```

```
superimposer.set_atoms(query_atoms, target_atoms)

print("Query and target superimposed, RMSD", round(superimposer.rms, 2),␣
 ↪"Angstroms")
```

```
Query and target superimposed, RMSD 0.77 Angstroms
```

0.6 Myc-Max in complex with 70551 lead inhibitor

[16]:
```
from rdkit import Chem

fn = 'Myc-Max_70551.pdb'
structure = Chem.rdmolfiles.MolFromPDBFile(fn)

view = nv.show_rdkit(structure)
display(view)
```

```
NGLWidget()
```

[ ]:
```

```