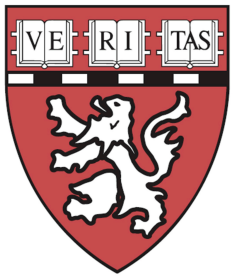
Exploring Apoptosis Signaling Using an Executable Rules-based Approach

Carlos F. Lopez
Jeremy L. Muhlich, John A. Bachman, Peter K. Sorger
Harvard Medical School



The battle ahead…

SUMMARY STATEMENT
(Privileged Communication) ***Release Date:*** **03/23/2012**

Approach: 5

Weaknesses
There is no literature evidence that rule-based modeling, which has been developed for almost a decade now, provides any real benefits over other mechanistic modeling approaches.

Outline

Biology of extrinsic apoptosis

- Alternate mechanisms of Bcl-2 biology
- Challenges modeling alternative topologies

Motivation for executable models

Executable biological models

Modularity, expandability, shareability, transparency

Exploring mechanism hypotheses

- Instantiating model topologies
- Exploring topologies
- Beyond topologies (simulation, calibration)

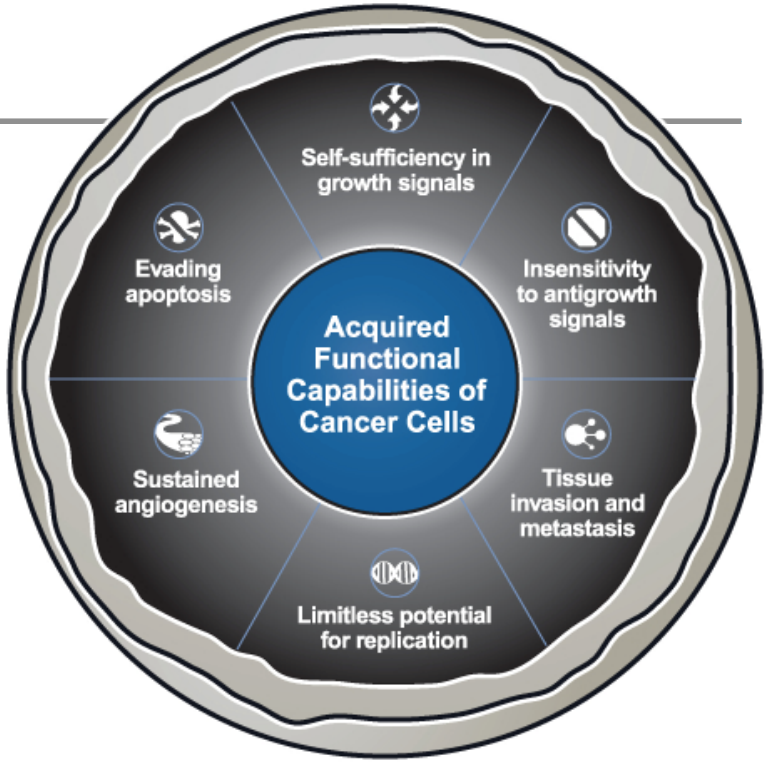
Ongoing + Future work

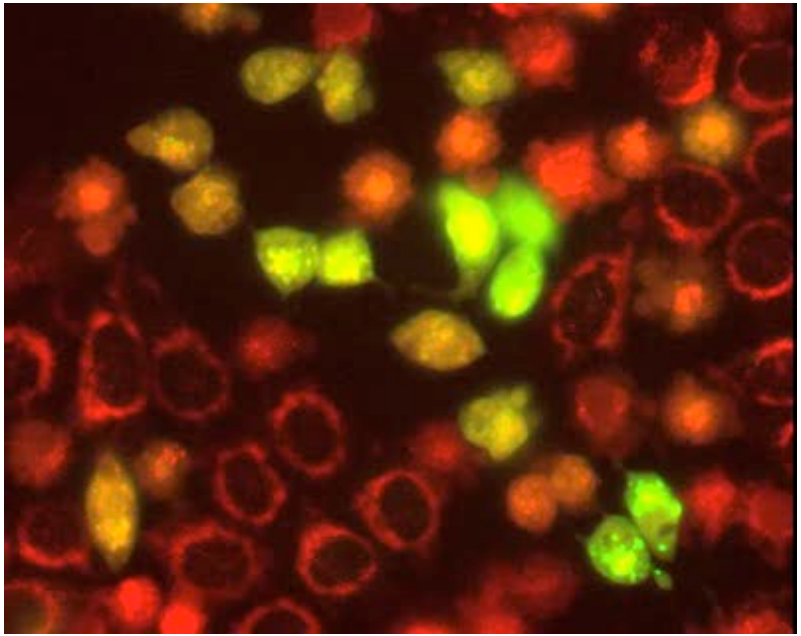
- Model sensitivity to topologies
- Bringing molecular simulation to bio modeling

Apoptosis

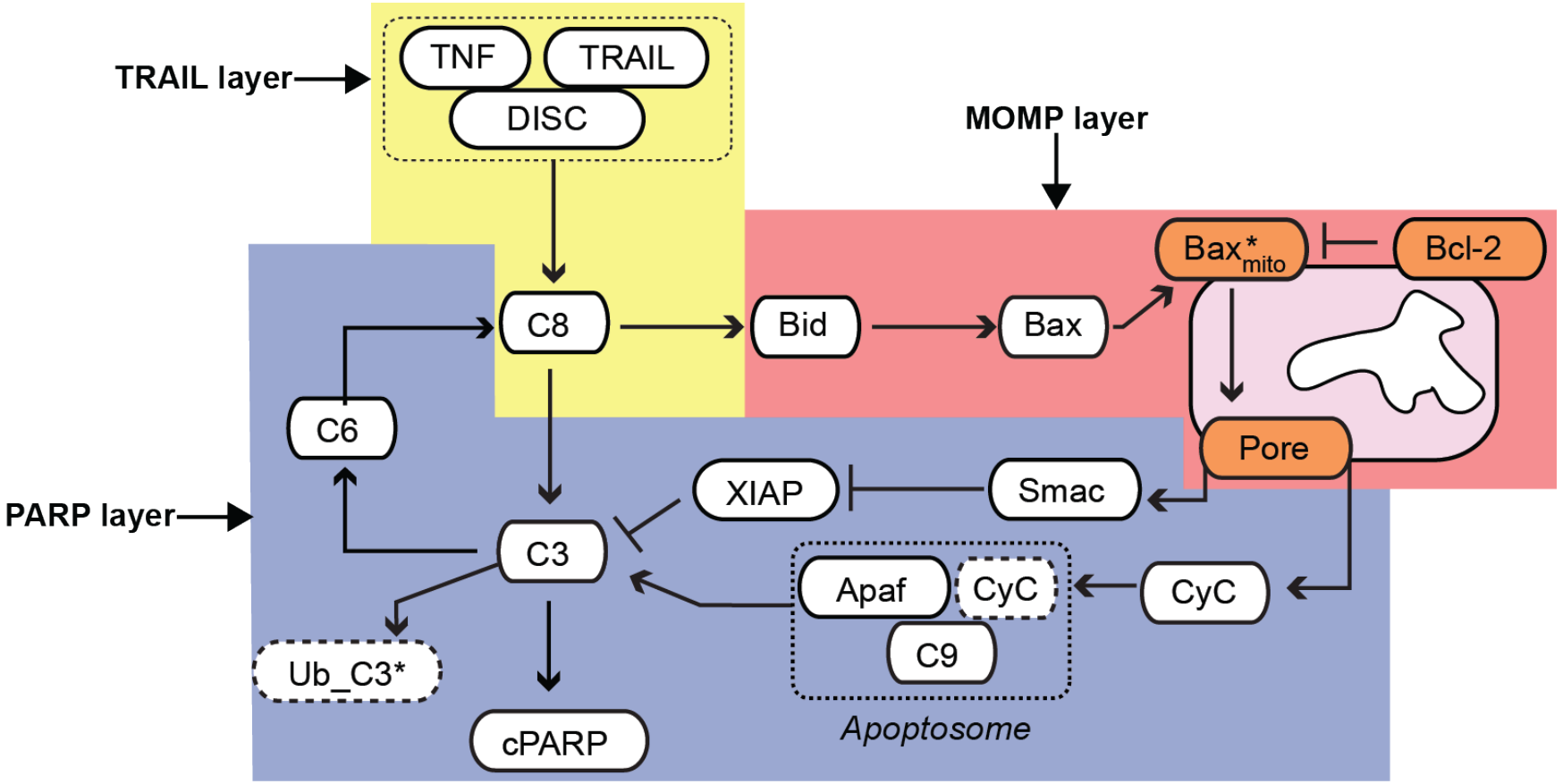
Apoptosis in cancer

- Programmed cell death is necessary to maintain balance in healthy tissue.
- Self-destruction and removal of cell material.
- Apoptosis activation possible strategy for cancer treatment.
- Tightly regulated by the interactions among members of Bcl-2 protein family.
- Almost all cancer cells evade apoptosis.
- Focus on extrinsic apoptosis.





Extrinsic apoptosis



TRAIL layer: Ligand binding to IC activation
MOMP layer: Bid activation to pore assembly
PARP layer: Cytochrome C release to PARP cleavage

Bid to MOMP mechanisms

MOMP is the commitment step in apoptosis.

Alternative hypotheses difficult to prove/disprove.

Diagrams represent many chemical interactions.

Can we use kinetic modeling to distinguish differences between hypotheses?

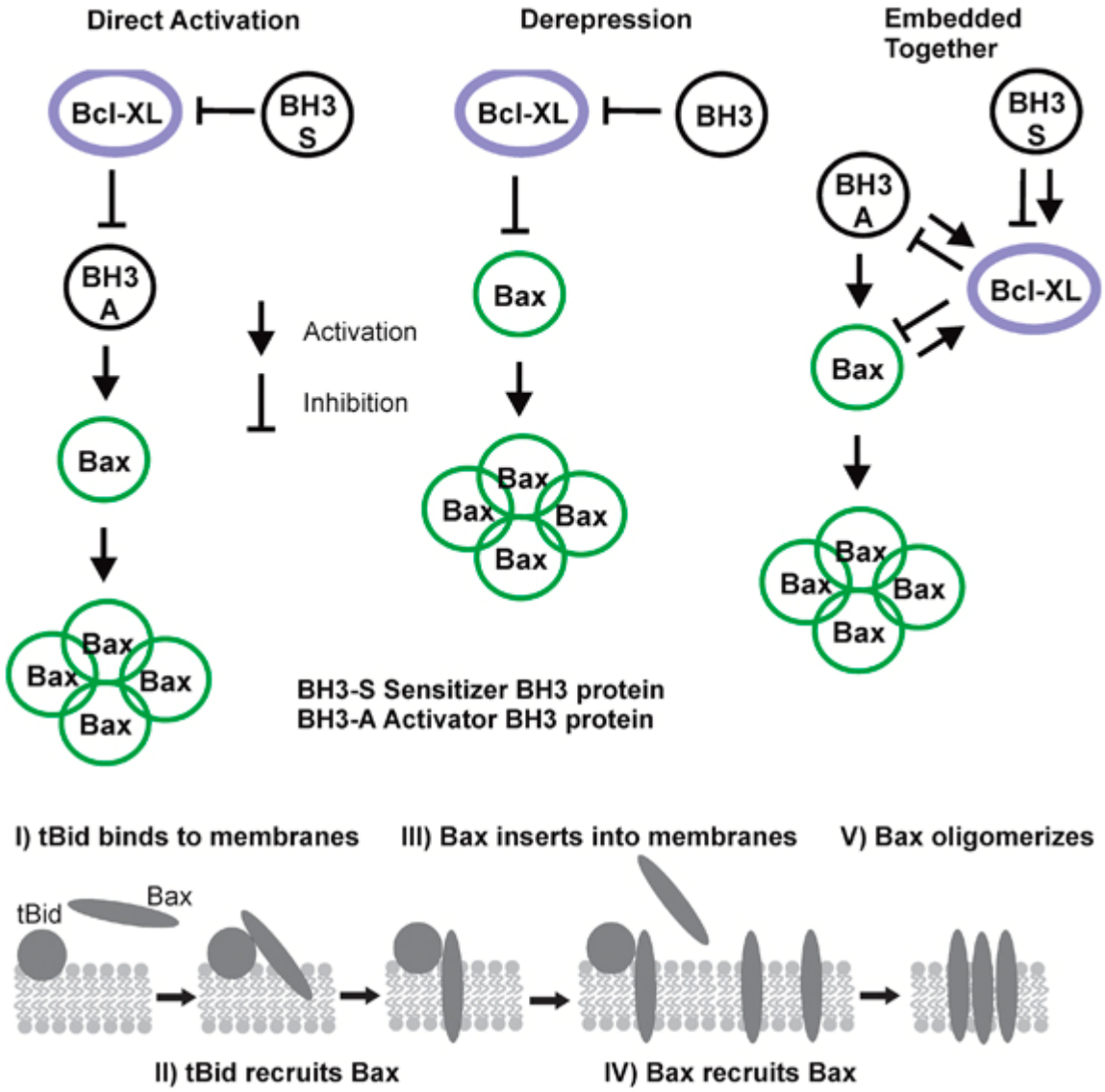


Figure: Andrews et al, Oncogene, 2010

Bcl-2 Family of Proteins

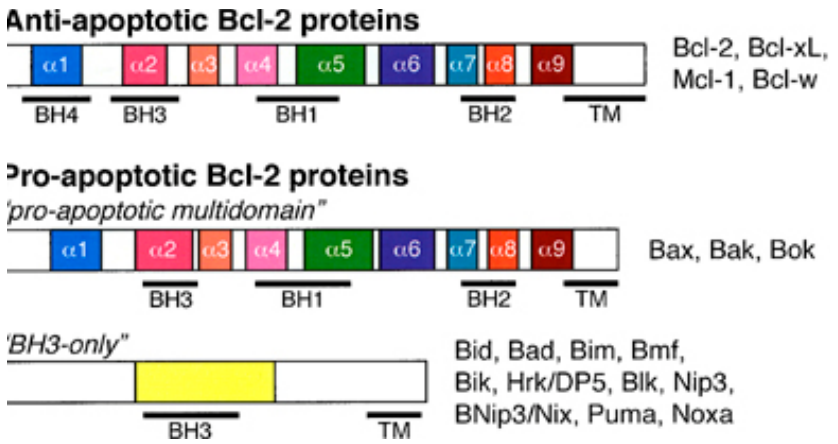
Interactions are further complicated by affinities.

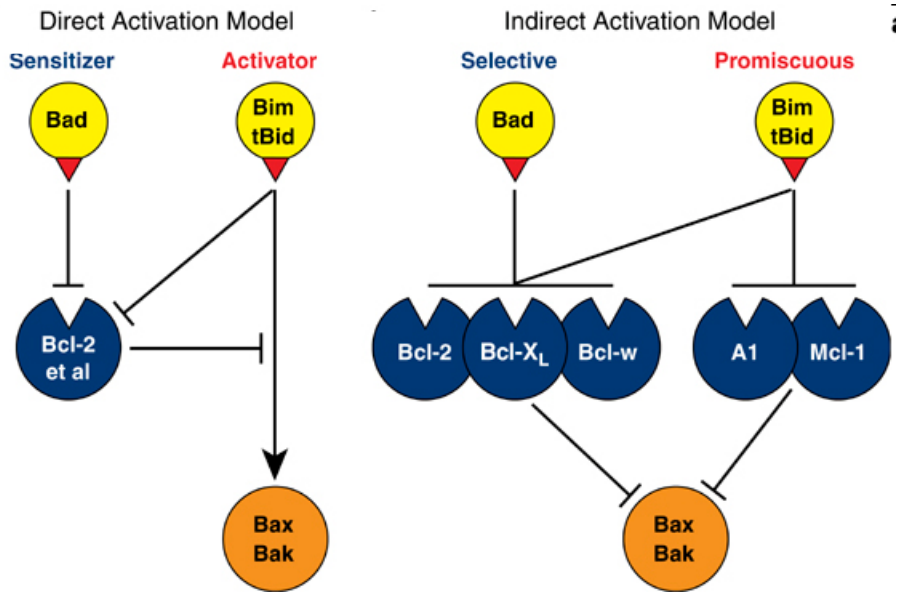
Different interactions in different proposed models.

Bax vs. Bak?

Activators (PUMA?)

Role of Sensitizers?



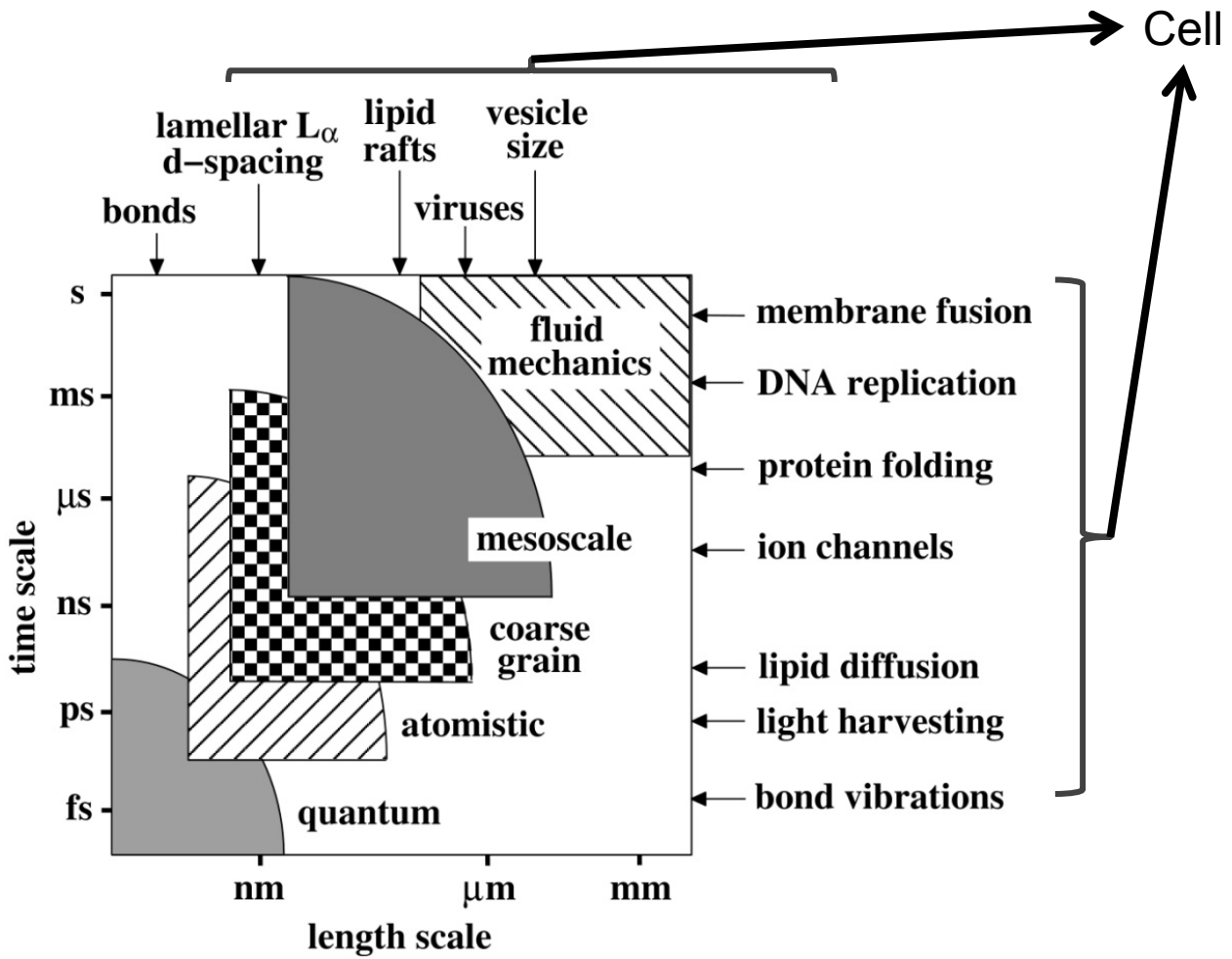


Figures: Green Oncogene (2006)

Cory Oncogene (2007)

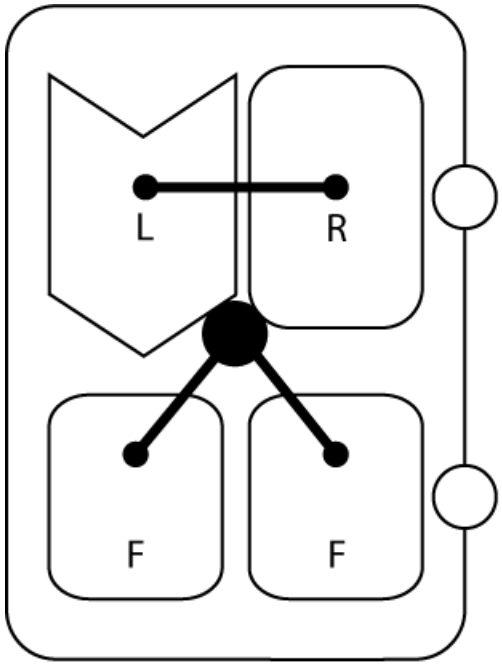
Modeling challenge

Crossing multiple spatiotemporal scales



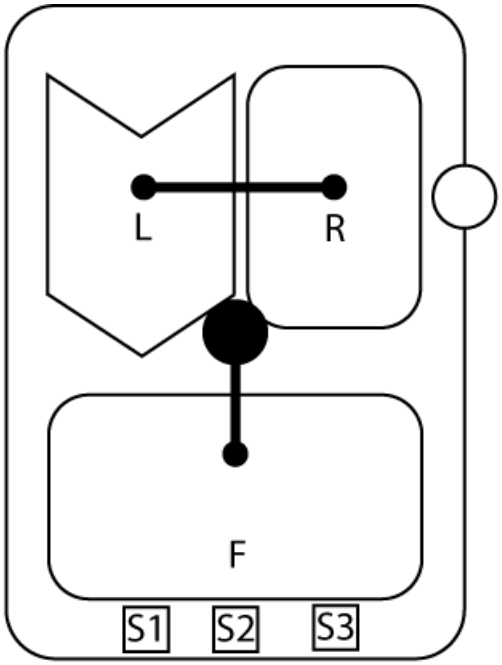
Cell is heterogeneous across spatiotemporal scales

Multiple ways to represent reality



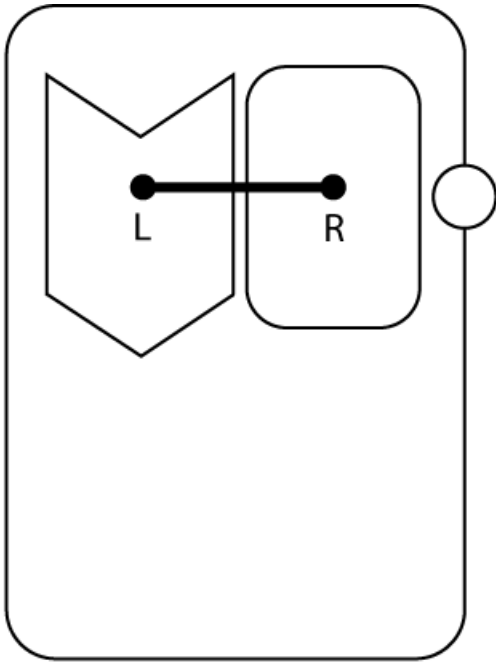
	DISC	MODEL
SPECS	4	13
RXNS	3	22

Fusseneger (2000)



	DISC	MODEL
SPECS	6	41
RXNS	5	32

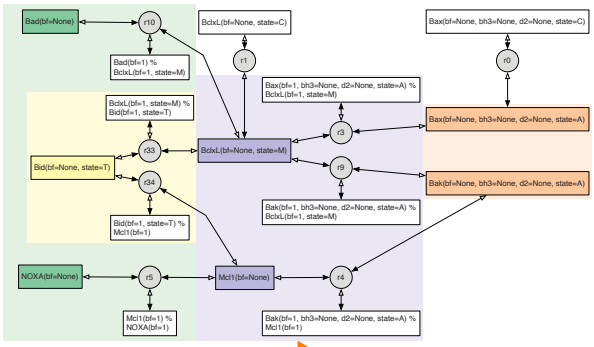
Bentele (2004)

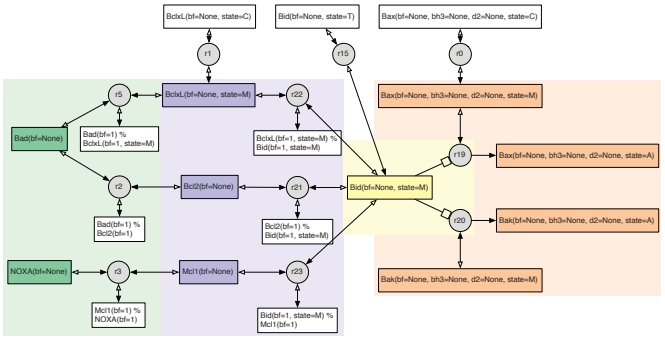


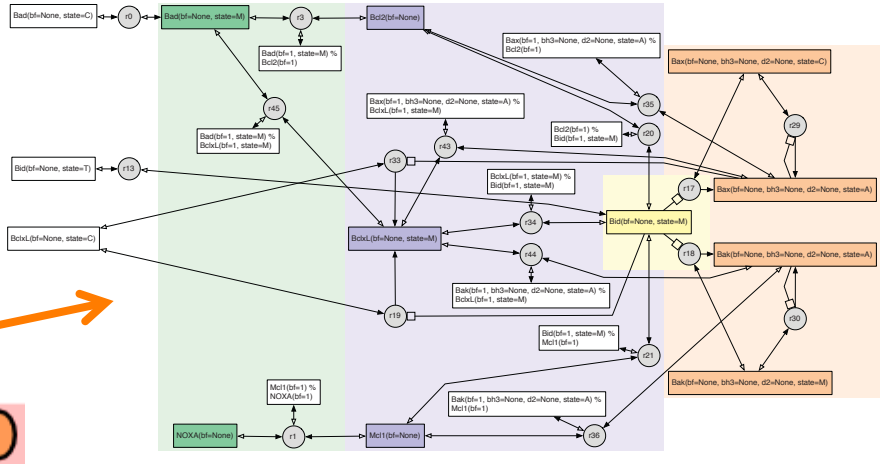
	DISC	MODEL
SPECS	2	58
RXNS	1	28

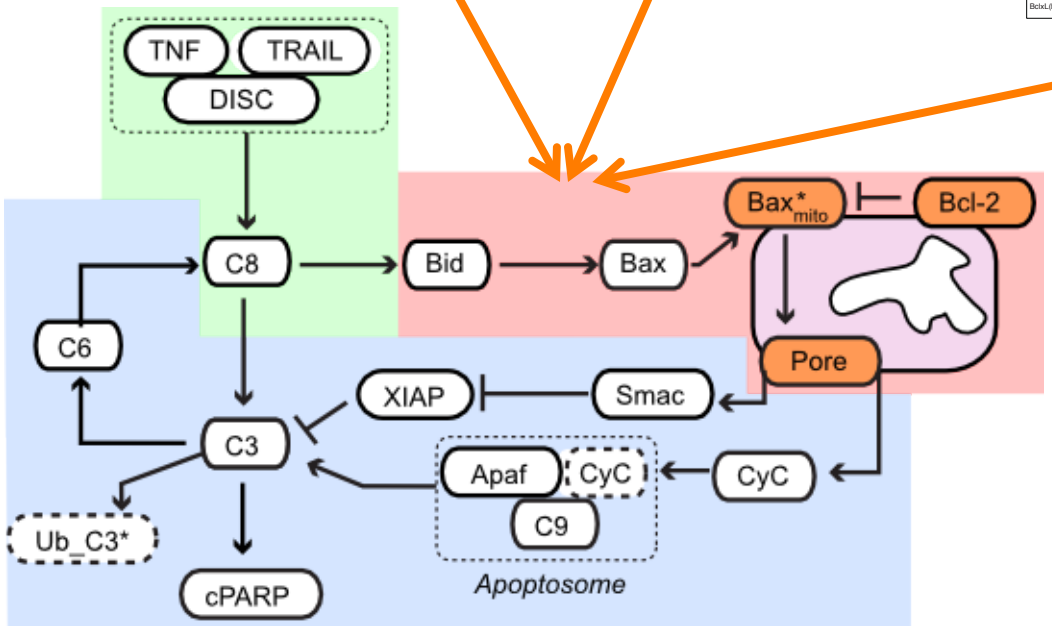
Albeck (2008)

Representing biology knowledge









Combinatorics issues arise quickly

```
Bax + Bax <-> Bax2
Bax2 + Bax2 <-> Bax4
Bax + Bcl-2 <-> Bax:Bcl-2
Bax + Bcl-XL <-> Bax:Bcl-XL
...

dBax/dt = ...
dBax2/dt = ...
dBax4/dt = ...
dBcl-2/dt = ...
dBax:Bcl-2/dt = ...
```

Source of Combinatorial Complexity:

- Oligomerization
- Phosphorylation
- Ubiquination
- Molecular Machines
- …

Altering model topology...

```
$c(456) = c(855)*c(476)*k(74)/k(20)+c(853)*c(733)*k(70)/k(3)-c(456)*k(70)-c(456)*k(74);
$c(457) = c(836)*c(477)*k(69)/k(20)+c(424)*k(69)+c(422)*k(70)+c(423)*k(74)+c(835)*c(738)*k(71)/k(3)+c(537)*k(72)+c(587)*c(834)*k(72)/k(20)+c(495)*k(73)-c(853)*c(457)*k(70)/k(3)-c(855)*c(457)*k(74)/k(20)-c(836)*c(457)*k(69)/k(20)-c(457)*k(73)-c(457)*k(72)-c(835)*c(457)*k(71)/k(3)-c(457)*k(69)-c(834)*c(457)*k(72)/k(20);
$c(458) = c(407)*k(69)+c(853)*c(477)*k(70)/k(3)+c(426)*k(70)+c(425)*k(74)+c(538)*k(72)+c(835)*c(739)*k(71)/k(3)+c(496)*k(73)+c(588)*c(834)*k(72)/k(20)-c(836)*c(458)*k(69)/k(20)-c(855)*c(458)*k(74)/k(20)-c(853)*c(458)*k(70)/k(3)-c(834)*c(458)*k(72)/k(20)-c(458)*k(73)-c(458)*k(70)-c(835)*c(458)*k(71)/k(3)-c(458)*k(72);
$c(459) = c(408)*k(69)+c(409)*k(70)+c(427)*k(74)+c(855)*c(477)*k(74)/k(20)+c(497)*k(73)+c(589)*c(834)*k(72)/k(20)+c(539)*k(72)+c(835)*c(740)*k(71)/k(3)-c(836)*c(459)*k(69)/k(20)-c(853)*c(459)*k(70)/k(3)-c(855)*c(459)*k(74)/k(20)-c(459)*k(73)-c(459)*k(72)-c(834)*c(459)*k(72)/k(20)-c(835)*c(459)*k(71)/k(3)-c(459)*k(74);
$c(460) = c(429)*k(70)+c(853)*c(478)*k(70)/k(3)+c(428)*k(74)+c(836)*c(739)*k(69)/k(20)+c(492)*k(73)+c(541)*k(72)+c(591)*c(834)*k(72)/k(20)+c(499)*k(69)-c(855)*c(460)*k(74)/k(20)-c(853)*c(460)*k(70)/k(3)-c(835)*c(460)*k(71)/k(3)-c(836)*c(460)*k(69)/k(20)-c(460)*k(70)-c(460)*k(72)-c(834)*c(460)*k(72)/k(20)-c(460)*k(69);
$c(461) = c(410)*k(70)+c(855)*c(478)*k(74)/k(20)+c(430)*k(74)+c(493)*k(73)+c(500)*k(69)+c(542)*k(72)+c(836)*c(740)*k(69)/k(20)+c(592)*c(834)*k(72)/k(20)-c(853)*c(461)*k(70)/k(3)-c(855)*c(461)*k(74)/k(20)-c(461)*k(74)-c(836)*c(461)*k(69)/k(20)-c(835)*c(461)*k(71)/k(3)-c(461)*k(69)-c(834)*c(461)*k(72)/k(20)-c(461)*k(72);
$c(462) = c(431)*k(74)+c(855)*c(479)*k(74)/k(20)+c(853)*c(740)*k(70)/k(3)+c(501)*k(70)+c(498)*k(69)+c(544)*k(72)+c(594)*c(834)*k(72)/k(20)+c(494)*k(73)-c(855)*c(462)*k(74)/k(20)-c(853)*c(462)*k(70)/k(3)-c(836)*c(462)*k(69)/k(20)-c(462)*k(74)-c(835)*c(462)*k(71)/k(3)-c(462)*k(72)-c(462)*k(70)-c(834)*c(462)*k(72)/k(20);
$c(463) = c(432)*k(69)+c(433)*k(70)+c(434)*k(74)+c(435)*k(73)+c(834)*c(582)*k(72)/k(20)+c(662)*k(72)+c(835)*c(481)*k(71)/k(3)+c(834)*c(737)*k(72)/k(20)-c(836)*c(463)*k(69)/k(20)-c(853)*c(463)*k(70)/k(3)-c(855)*c(463)*k(74)/k(20)-c(835)*c(463)*k(71)/k(3)-c(463)*k(73)-c(834)*c(463)*k(72)/k(20)-2*c(463)*k(72);
$c(464) = c(432)*k(73)+c(436)*k(70)+c(437)*k(74)+c(438)*k(69)+c(834)*c(583)*k(72)/k(20)+c(836)*c(481)*k(69)/k(20)+c(663)*k(72)+c(834)*c(738)*k(72)/k(20)-c(835)*c(464)*k(71)/k(3)-c(853)*c(464)*k(70)/k(3)-c(855)*c(464)*k(74)/k(20)-c(836)*c(464)*k(69)/k(20)-2*c(464)*k(72)-c(834)*c(464)*k(72)/k(20)-c(464)*k(69);
$c(465) = c(433)*k(73)+c(436)*k(69)+c(439)*k(74)+c(440)*k(70)+c(853)*c(481)*k(70)/k(3)+c(834)*c(739)*k(72)/k(20)+c(834)*c(584)*k(72)/k(20)+c(664)*k(72)-c(835)*c(465)*k(71)/k(3)-c(836)*c(465)*k(69)/k(20)-c(855)*c(465)*k(74)/k(20)-c(853)*c(465)*k(70)/k(3)-c(834)*c(465)*k(72)/k(20)-c(465)*k(70)-2*c(465)*k(72);
$c(466) = c(434)*k(73)+c(437)*k(69)+c(439)*k(70)+c(441)*k(74)+c(834)*c(740)*k(72)/k(20)+c(665)*k(72)+c(855)*c(481)*k(74)/k(20)+c(834)*c(585)*k(72)/k(20)-c(835)*c(466)*k(71)/k(3)-c(836)*c(466)*k(69)/k(20)-c(853)*c(466)*k(70)/k(3)-c(855)*c(466)*k(74)/k(20)-2*c(466)*k(72)-c(834)*c(466)*k(72)/k(20)-c(466)*k(74);
$c(467) = c(443)*k(70)+c(482)*c(853)*k(70)/k(3)+c(442)*k(74)+c(527)*k(72)+c(551)*k(73)+c(836)*c(743)*k(69)/k(20)+c(503)*k(69)+c(835)*c(591)*k(71)/k(3)-c(467)*c(855)*k(74)/k(20)-c(853)*c(467)*k(70)/k(3)-c(835)*c(467)*k(71)/k(3)-c(467)*k(73)-c(467)*k(70)-c(467)*k(69)-c(467)*c(834)*k(72)/k(20)-c(836)*c(467)*k(69)/k(20);
$c(468) = c(411)*k(70)+c(444)*k(74)+c(482)*c(855)*k(74)/k(20)+c(504)*k(69)+c(836)*c(744)*k(69)/k(20)+c(552)*k(73)+c(835)*c(592)*k(71)/k(3)+c(528)*k(72)-c(853)*c(468)*k(70)/k(3)-c(855)*c(468)*k(74)/k(20)-c(468)*c(834)*k(72)/k(20)-c(835)*c(468)*k(71)/k(3)-c(468)*k(69)-c(836)*c(468)*k(69)/k(20)-c(468)*k(74)-c(468)*k(73);
$c(469) = c(483)*c(855)*k(74)/k(20)+c(445)*k(74)+c(530)*k(72)+c(502)*k(69)+c(835)*c(594)*k(71)/k(3)+c(554)*k(73)+c(505)*k(70)+c(853)*c(744)*k(70)/k(3)-c(855)*c(469)*k(74)/k(20)-c(469)*k(74)-c(469)*k(73)-c(835)*c(469)*k(71)/k(3)-c(853)*c(469)*k(70)/k(3)-c(469)*c(834)*k(72)/k(20)-c(469)*k(70)-c(836)*c(469)*k(69)/k(20);
$c(470) = c(446)*k(70)+c(447)*k(74)+c(448)*k(69)+c(835)*c(742)*k(71)/k(3)+c(836)*c(485)*k(69)/k(20)+c(683)*k(73)+c(638)*k(72)+c(835)*c(587)*k(71)/k(3)-c(470)*c(853)*k(70)/k(3)-c(470)*c(855)*k(74)/k(20)-c(836)*c(470)*k(69)/k(20)-2*c(470)*k(73)-c(835)*c(470)*k(71)/k(3)-c(470)*c(834)*k(72)/k(20)-c(470)*k(69);
$c(471) = c(446)*k(69)+c(449)*k(74)+c(450)*k(70)+c(835)*c(588)*k(71)/k(3)+c(485)*c(853)*k(70)/k(3)+c(835)*c(743)*k(71)/k(3)+c(639)*k(72)+c(684)*k(73)-c(836)*c(471)*k(69)/k(20)-c(471)*c(855)*k(74)/k(20)-c(853)*c(471)*k(70)/k(3)-c(471)*c(834)*k(72)/k(20)-c(835)*c(471)*k(71)/k(3)-2*c(471)*k(73)-c(471)*k(70);
$c(472) = c(447)*k(69)+c(449)*k(70)+c(451)*k(74)+c(485)*c(855)*k(74)/k(20)+c(640)*k(72)+c(835)*c(589)*k(71)/k(3)+c(685)*k(73)+c(835)*c(744)*k(71)/k(3)-c(836)*c(472)*k(69)/k(20)-c(853)*c(472)*k(70)/k(3)-c(855)*c(472)*k(74)/k(20)-c(472)*c(834)*k(72)/k(20)-c(835)*c(472)*k(71)/k(3)-2*c(472)*k(73)-c(472)*k(74);
$c(473) = c(452)*k(74)+c(486)*c(855)*k(74)/k(20)+c(853)*c(747)*k(70)/k(3)+c(558)*k(69)+c(506)*k(70)+c(836)*c(594)*k(69)/k(20)+c(533)*k(72)+c(547)*k(73)-c(855)*c(473)*k(74)/k(20)-c(473)*c(834)*k(72)/k(20)-c(473)*k(70)-c(853)*c(473)*k(70)/k(3)-c(835)*c(473)*k(71)/k(3)-c(473)*k(69)-c(836)*c(473)*k(69)/k(20)-c(473)*k(74);
$c(474) = c(453)*k(74)+c(454)*k(70)+c(645)*k(72)+c(670)*k(73)+c(836)*c(746)*k(69)/k(20)+c(836)*c(591)*k(69)/k(20)+c(694)*k(69)+c(488)*c(853)*k(70)/k(3)-c(474)*c(855)*k(74)/k(20)-c(853)*c(474)*k(70)/k(3)-c(474)*c(834)*k(72)/k(20)-c(836)*c(474)*k(69)/k(20)-c(835)*c(474)*k(71)/k(3)-c(474)*k(70)-2*c(474)*k(69);
$c(475) = c(453)*k(70)+c(455)*k(74)+c(671)*k(73)+c(488)*c(855)*k(74)/k(20)+c(836)*c(592)*k(69)/k(20)+c(836)*c(747)*k(69)/k(20)+c(646)*k(72)+c(695)*k(69)-c(853)*c(475)*k(70)/k(3)-c(855)*c(475)*k(74)/k(20)-c(475)*c(834)*k(72)/k(20)-c(475)*k(74)-2*c(475)*k(69)-c(835)*c(475)*k(71)/k(3)-c(836)*c(475)*k(69)/k(20);
$c(476) = c(456)*k(74)+c(649)*k(72)+c(853)*c(594)*k(70)/k(3)+c(699)*k(70)+c(853)*c(749)*k(70)/k(3)+c(688)*k(69)+c(855)*c(490)*k(74)/k(20)+c(674)*k(73)-c(855)*c(476)*k(74)/k(20)-c(836)*c(476)*k(69)/k(20)-2*c(476)*k(70)-c(835)*c(476)*k(71)/k(3)-c(476)*k(74)-c(853)*c(476)*k(70)/k(3)-c(476)*c(834)*k(72)/k(20);
$c(477) = c(412)*k(69)+c(413)*k(70)+c(414)*k(74)+c(457)*k(69)+c(458)*k(70)+c(459)*k(74)+c(712)*k(72)+c(615)*k(73)+c(753)*c(834)*k(72)/k(20)+c(568)*k(72)+c(835)*c(752)*k(71)/k(3)+c(510)*k(73)-2*c(836)*c(477)*k(69)/k(20)-2*c(853)*c(477)*k(70)/k(3)-2*c(855)*c(477)*k(74)/k(20)-2*c(834)*c(477)*k(72)/k(20)-c(477)*k(72)-2*c(835)*c(477)*k(71)/k(3)-c(477)*k(73);
$c(478) = c(415)*k(70)+c(416)*k(74)+c(460)*k(70)+c(461)*k(74)+c(507)*k(73)+c(713)*k(72)+c(754)*c(834)*k(72)/k(20)+c(569)*k(72)+c(618)*k(69)+c(836)*c(752)*k(69)/k(20)+c(513)*k(69)+c(612)*k(73)-2*c(853)*c(478)*k(70)/k(3)-2*c(855)*c(478)*k(74)/k(20)-c(478)*k(69)-2*c(834)*c(478)*k(72)/k(20)-c(478)*k(72)-2*c(836)*c(478)*k(69)/k(20)-2*c(835)*c(478)*k(71)/k(3);
$c(479) = c(417)*k(74)+c(462)*k(74)+c(620)*k(70)+c(511)*k(69)+c(714)*k(72)+c(515)*k(70)+c(853)*c(752)*k(70)/k(3)+c(508)*k(73)+c(755)*c(834)*k(72)/k(20)+c(570)*k(72)+c(613)*k(73)+c(616)*k(69)-2*c(855)*c(479)*k(74)/k(20)-2*c(853)*c(479)*k(70)/k(3)-c(479)*k(72)-2*c(835)*c(479)*k(71)/k(3)-2*c(834)*c(479)*k(72)/k(20)-c(479)*k(70)-2*c(836)*c(479)*k(69)/k(20);
$c(480) = c(855)*c(752)*k(74)/k(20)+c(512)*k(69)+c(509)*k(73)+c(621)*k(74)+c(514)*k(70)+c(715)*k(72)+c(617)*k(69)+c(619)*k(70)+c(756)*c(834)*k(72)/k(20)+c(571)*k(72)+c(516)*k(74)+c(614)*k(73)-2*c(853)*c(480)*k(70)/k(3)-c(480)*k(72)-2*c(855)*c(480)*k(74)/k(20)-c(480)*k(74)-2*c(836)*c(480)*k(69)/k(20)-2*c(835)*c(480)*k(71)/k(3)-2*c(834)*c(480)*k(72)/k(20);
$c(481) = c(463)*k(73)+c(464)*k(69)+c(465)*k(70)+c(466)*k(74)+c(834)*c(752)*k(72)/k(20)+c(716)*k(72)-c(835)*c(481)*k(71)/k(3)-c(836)*c(481)*k(69)/k(20)-c(853)*c(481)*k(70)/k(3)-c(855)*c(481)*k(74)/k(20)-c(834)*c(481)*k(72)/k(20)-c(481)*k(72);
$c(482) = c(418)*k(70)+c(419)*k(74)+c(467)*k(70)+c(468)*k(74)+c(702)*k(72)+c(835)*c(754)*k(71)/k(3)+c(753)*c(836)*k(69)/k(20)+c(723)*k(73)+c(624)*k(69)+c(575)*k(73)+c(562)*k(72)+c(519)*k(69)-2*c(482)*c(853)*k(70)/k(3)-2*c(482)*c(855)*k(74)/k(20)-2*c(836)*c(482)*k(69)/k(20)-2*c(482)*c(834)*k(72)/k(20)-c(482)*k(73)-c(482)*k(69)-2*c(835)*c(482)*k(71)/k(3);
$c(483) = c(420)*k(74)+c(469)*k(74)+c(576)*k(73)+c(622)*k(69)+c(724)*k(73)+c(753)*c(853)*k(70)/k(3)+c(835)*c(755)*k(71)/k(3)+c(626)*k(70)+c(703)*k(72)+c(517)*k(69)+c(521)*k(70)+c(563)*k(72)-2*c(483)*c(855)*k(74)/k(20)-c(483)*k(73)-2*c(836)*c(483)*k(69)/k(20)-2*c(483)*c(834)*k(72)/k(20)-c(483)*k(70)-2*c(835)*c(483)*k(71)/k(3)-2*c(853)*c(483)*k(70)/k(3);
$c(484) = c(522)*k(74)+c(704)*k(72)+c(518)*k(69)+c(520)*k(70)+c(627)*k(74)+c(753)*c(855)*k(74)/k(20)+c(564)*k(72)+c(623)*k(69)+c(725)*k(73)+c(625)*k(70)+c(835)*c(756)*k(71)/k(3)+c(577)*k(73)-2*c(484)*c(834)*k(72)/k(20)-2*c(836)*c(484)*k(69)/k(20)-2*c(835)*c(484)*k(71)/k(3)-2*c(853)*c(484)*k(70)/k(3)-c(484)*k(74)-c(484)*k(73)-2*c(855)*c(484)*k(74)/k(20);
$c(485) = c(470)*k(69)+c(471)*k(70)+c(472)*k(74)+c(835)*c(753)*k(71)/k(3)+c(726)*k(73)+c(705)*k(72)-c(836)*c(485)*k(69)/k(20)-c(485)*c(853)*k(70)/k(3)-c(485)*c(855)*k(74)/k(20)-c(835)*c(485)*k(71)/k(3)-c(485)*c(834)*k(72)/k(20)-c(485)*k(73);
$c(486) = c(421)*k(74)+c(473)*k(74)+c(572)*k(73)+c(579)*k(69)+c(565)*k(72)+c(629)*k(70)+c(730)*k(69)+c(836)*c(755)*k(69)/k(20)+c(717)*k(73)+c(706)*k(72)+c(524)*k(70)+c(754)*c(853)*k(70)/k(
```

CANNOT edit this by hand…

Challenges to study topology

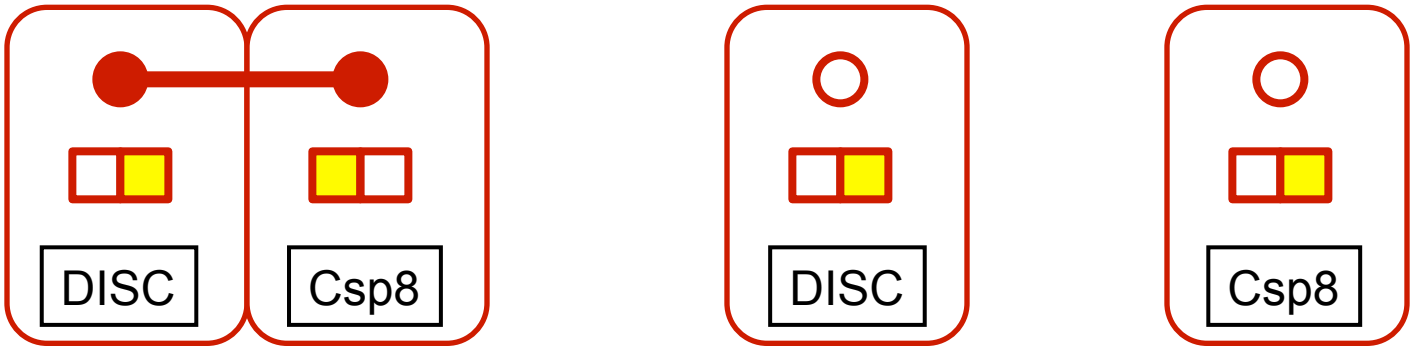
- Traditional modelling approaches:
  - Prescriptive and database-driven.
  - Do not allow direct model exploration/manipulation.
  - Difficult to explore multiple topologies systematically.

- Also want knowledge annotation (not today, please ask if interested)
  - Model:
    - Tracking
    - Revision
    - Sharing
  - Ontology + Wiki based model tracking approaches.

The rules “language”

DISC:Csp8 → DISC + Csp8*

```
Rule('Csp8_Activation',
       DISC(b=1, state='A') % Csp8(b=1, state='I') >
       DISC(b=None, state='A') + Csp8(b=None, state='A')
       KDISCCSP8C)
```



Syntactic complexity increases w problem size

Keyword

Parameter Name

Value

Initial Conditions

```
Parameter('TRAIL_0',      3000) # molecules/cell original: 3000
Parameter('DReceptor_0',   200) # molecules/cell
Parameter('Csp8_0',         2E4) # molecules/cell
Parameter('Csp3_0',         1E4) # molecules/cell
Parameter('Csp6_0',         1E4) # molecules/cell
Parameter('Csp9_0',         1E5) # molecules/cell
```

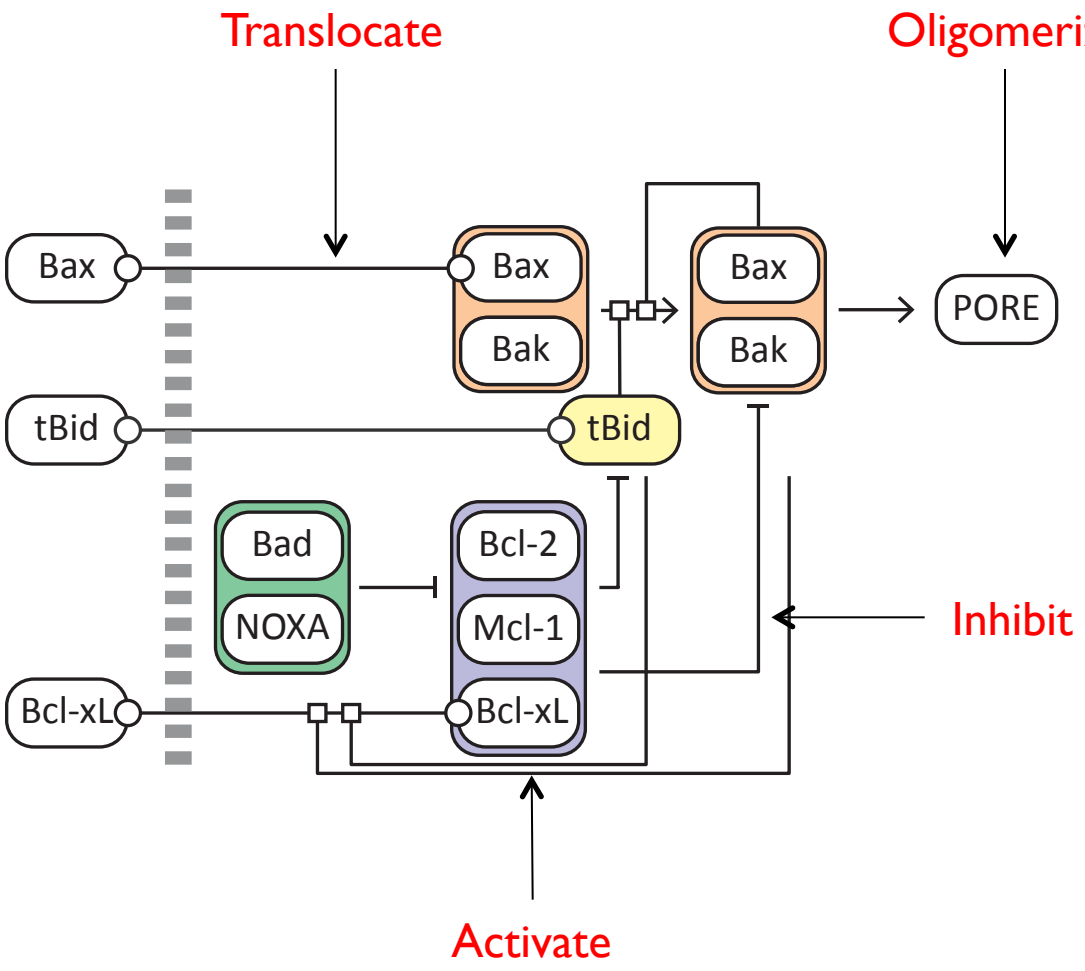
Reaction Rates

```
Parameter('KBIDCCSP8F',          3.0E-8)   # (#/cell)^-1 sec^-1
Parameter('KBIDCCSP8R',          1.0E-3)   # sec^-1
Parameter('KBIDCCSP8C',          1.0    )   # sec^-1
Parameter('KBIDCBAXCF',          4.5E-8)   # (#/cell)^-1 sec^-1
```

Rules

```
egfr(Y1148~pY) + Shc(PTB,Y317~Y)    <-> egfr(Y1148~pY!1).Shc(PTB!1,Y317~Y)   kp13,km13
egfr(Y1148~pY) + Shc(PTB,Y317~pY)   <-> egfr(Y1148~pY!1).Shc(PTB!1,Y317~pY)  kp15,km15
egfr(Y1148~pY) + Shc(PTB,Y317~pY!1).Grb2(SH2!1,SH3)   <-> \
                      egfr(Y1148~pY!2).Shc(PTB!2,Y317~pY!1).Grb2(SH2!1,SH3)  kp18,km18
egfr(Y1148~pY) + Shc(PTB,Y317~pY!1).Grb2(SH2!1,SH3!3).Sos(dom!3)   <-> \
          egfr(Y1148~pY!2).Shc(PTB!2,Y317~pY!1).Grb2(SH2!1,SH3!3).Sos(dom!3)  kp20,km20
```

Want to express "actions"



Logic operations implied by each action.

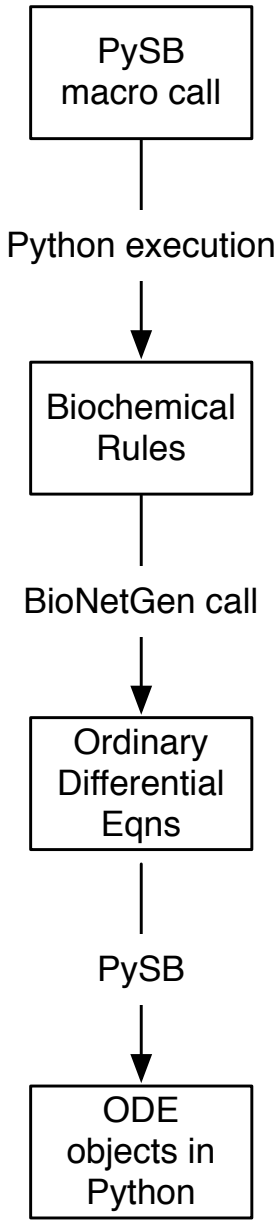
Executable modeling

pySB – executable model approach

- Key goal: write biological knowledge.
- Avoid long lists of parameters/formalisms.
- Enumerating math by hand is cumbersome.
- Transparency of a model is essential (sharing, revising).
- Make expanding models routine.
- Leave the chemical and mathematical procedures to the computer.

- Build an environment where model exploration is allowed.
- Build platform where multiple tools can be used (i.e. not just Matlab).
- Open source, Python language

Biochemical functions in Python




```
catalyze(Casp8, Bid, 'trunc', 'N', 'Y', kf, kr, kc)
```

```
Casp8(b) + Bid(b,trunc~N) <-> Casp8(b!1).Bid(b!1,trunc~N)   kf, kr
Casp8(b!1).Bid(b!1,trunc~N) -> Casp8(b) + Bid(b,trunc~Y)    kc
```

```
r1 = kf[y1][y2]
r2 = kr[y3]
r3 = kc[y3]
d[y1]/dt = kc[y3] - kf[y1][y2] + kr[y3]
d[y2]/dt = -kf[y1][y2] + kr[y3]
d[y3]/dt = -kc[y3] + kf[y1][y2] - kr[y3]
d[y4]/dt = kc[y3]
```

```
model.odes()
```

Defining simple functions

```
def catalyze(enz, sub, prod, kf, kr, kc):
    """2-step catalytic process"""
    r1_name = 'bind_%s_%s' % (sub.name, enz.name)
    r2_name = 'produce_%s_via_%s' % (prod.name, enz.name)
    E = enz(b=None)
    S = sub(b=None)
    ES = enz(b=1) * sub(b=1)
    P = prod(b=None)
    Rule(r1_name, E + S <> ES, kf, kr)
    Rule(r2_name, ES >> E + P, kc)


def catalyze_convert(s1, s2, p, kf, kr, kc):
    """2-step catalytic-type process, but the "catalyst" is effectively consumed"""
    r1_name = 'bind_%s_%s' % (s1.name, s2.name)
    r2_name = 'produce_%s' % p.name
    A = s1(b=None)
    B = s2(b=None)
    AB = s1(b=1) * s2(b=1)
    C = p(b=None)
    Rule(r1_name, A + B <> AB, kf, kr)
    Rule(r2_name, AB >> C, kc)


def inhibit(targ, inh, kf, kr):
    """inhibition by complexation/sequestration"""
    r_name = 'inhibit_%s_by_%s' % (targ.name, inh.name)
    T = targ(b=None)
    I = inh(b=None)
    TI = targ(b=1) * inh(b=1)
    Rule(r_name, T + I <> TI, kf, kr)
```

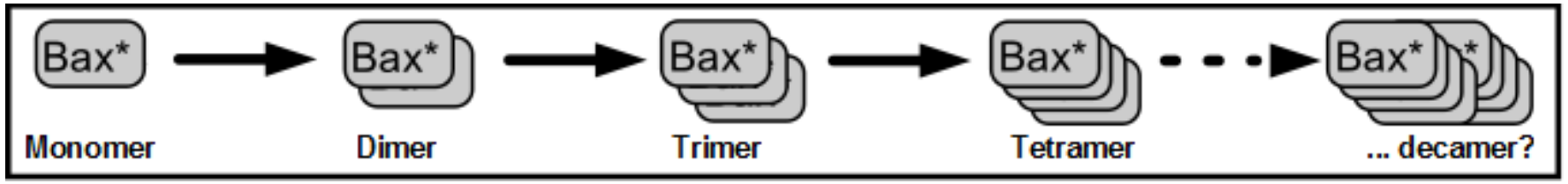
- Functions allow us to generalize chemical/biological/physical patterns
- Patterns abstracted to function
- Functions can be reused

Using functions

```
# L + pR <--> L:pR --> DISC
Parameter('kf1', 4e-07)
Parameter('kr1', 1e-03)
Parameter('kc1', 1e-05)
catalyze_convert(L, pR, DISC, kf1, kr1, kc1)

# flip + DISC <-->  flip:DISC
Parameter('kf2', 1e-06)
Parameter('kr2', 1e-03)
inhibit(DISC, flip, kf2, kr2)

# pC8 + DISC <--> DISC:pC8 --> C8 + DISC
Parameter('kf3', 1e-06)
Parameter('kr3', 1e-03)
Parameter('kc3', 1e+00)
catalyze(DISC, pC8, C8, kf3, kr3, kc3)

# C8 + BAR <--> BAR:C8
Parameter('kf4', 1e-06)
Parameter('kr4', 1e-03)
inhibit(BAR, C8, kf4, kr4)

# pC3 + C8 <--> pC3:C8 --> C3 + C8
Parameter('kf5', 1e-07)
Parameter('kr5', 1e-03)
Parameter('kc5', 1e+00)
catalyze(C8, pC3, C3, kf5, kr5, kc5)
```

- We write what we mean
- Easy to revise
- Annotate/Track

Higher level complexity: oligomerization



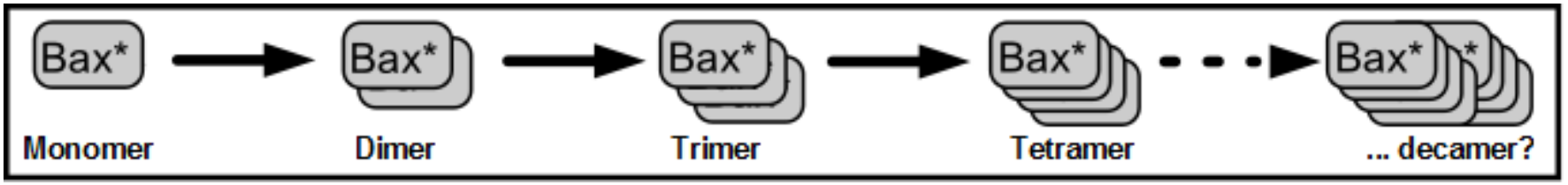
Make model functional rather than prescriptive:

```
def pore_assembly(Subunit, size, rates):
    for poresize in range(2, size + 1):
        basepore = pore_species(Subunit, 1)
        addunit = pore_species(Subunit, i-1)
        newpore = pore_species(Subunit, i)
        rules.append(Rule('%s_pore_assembly_%d'
                          % (Subunit.monomer.name, poresize),
                          basepore + addunit <> newpore, *rates))
    return rules
```

Call for rule generation:

```
pore_assembly(Bax(bf=None, state='A'), 4, kd['BAX_PORE'])
```

pore_assembly() execution



BNG Rules for Bax/Bak tetramer generation

```
Bax(bf,bh3,d2,state~A) + Bax(bf,bh3,d2,state~A) <-> Bax(bf,bh3!1,d2,state~A).Bax(bf,bh3,d2!1,state~A) kbaxdimf, kbaxdimr
Bax(bf,bh3,d2,state~A) + Bax(bf,bh3!1,d2,state~A).Bax(bf,bh3,d2!1,state~A) <->
{MatchOnce}Bax(bf,bh3!1,d2!2,state~A).Bax(bf,bh3!2,d2!3,state~A).Bax(bf,bh3!3,d2!1,state~A) kbaxtrimf, kbaxtrimr
Bax(bf,bh3,d2,state~A) + {MatchOnce}Bax(bf,bh3!1,d2!2,state~A).Bax(bf,bh3!2,d2!3,state~A).Bax(bf,bh3!3,d2!1,state~A) <->
{MatchOnce}Bax(bf,bh3!1,d2!2,state~A).Bax(bf,bh3!2,d2!3,state~A).Bax(bf,bh3!3,d2!4,state~A).Bax(bf,bh3!4,d2!1,state~A)
kbaxtetf, kbaxtetr
Bak(bf,bh3,d2,state~A) + Bak(bf,bh3,d2,state~A) <-> Bak(bf,bh3!1,d2,state~A).Bak(bf,bh3,d2!1,state~A) kbakdimf, kbakdimr
Bak(bf,bh3,d2,state~A) + Bak(bf,bh3!1,d2,state~A).Bak(bf,bh3,d2!1,state~A) <->
{MatchOnce}Bak(bf,bh3!1,d2!2,state~A).Bak(bf,bh3!2,d2!3,state~A).Bak(bf,bh3!3,d2!1,state~A) kbaktrimf, kbaktrimr
Bak(bf,bh3,d2,state~A) + {MatchOnce}Bak(bf,bh3!1,d2!2,state~A).Bak(bf,bh3!2,d2!3,state~A).Bak(bf,bh3!3,d2!1,state~A) <->
{MatchOnce}Bak(bf,bh3!1,d2!2,state~A).Bak(bf,bh3!2,d2!3,state~A).Bak(bf,bh3!3,d2!4,state~A).Bak(bf,bh3!4,d2!1,state~A)
kbaktetf, kbaktetr
```

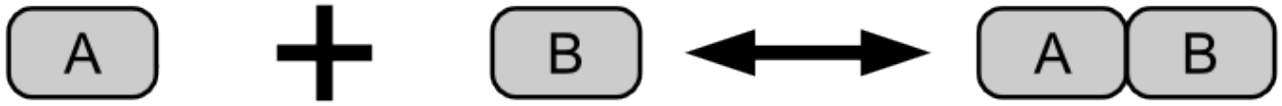
~45 BNG rules (one shown)

Equations…

```
S7:  kbidbakr*s46 - kbidbakf*s41*s7
S9:  bakbclxdr*s61 + baxbclxdr*s59 + kbadbclxdr*s25 + kbclxlCbclxlMr*s23 + knoxabcl2r*s26 - kbclxlCbclxlMf*s9 - bakbclxlf*s53*s9 - baxbclxlf*s52*s9 -
     kbadbclxlf*s11*s9 - knoxabcl2f*s12*s9
S10: bakmcl1r*s63 + knoxamcl1r*s27 - bakmcl1f*s10*s53 - knoxamcl1f*s10*s12
S13: 0.25*kbakcytocMCr*s70 + 0.25*kbaxcytocMCr*s68 - 0.25*kbakcytocMCf*s13*s67 - 0.25*kbaxcytocMCf*s13*s66
S14: 0.25*kbaksmacCAr*s71 + 0.25*kbaxsmacCAr*s69 - 0.25*kbaksmacCAf*s14*s67 - 0.25*kbaxsmacCAf*s14*s66
S23: bakbclxdr*s62 + baxbclxdr*s60 + kbadbclxdr*s30 + kbclxlCbclxlMf*s9 + knoxabcl2r*s31 - kbclxlCbclxlMr*s23 - bakbclxlf*s23*s53 - baxbclxlf*s23*s52 -
     kbadbclxlf*s11*s23 - knoxabcl2f*s12*s23
S41: kbidCbidMf*s39 + kbidbakc*s46 + kbidbakr*s46 + kbidbaxc*s45 + kbidbaxr*s45 + kbidbcl2r*s47 + kbidbcl2r*s48 - kbidCbidMr*s41 - kbidbakf*s41*s7 -
     kbidbaxf*s21*s41 - kbidbcl2f*s22*s41 - kbidbcl2f*s41*s8
S46: -kbidbakc*s46 - kbidbakr*s46 + kbidbakf*s41*s7
S53: bakbclxdr*s61 + bakbclxdr*s62 + bakmcl1r*s63 + kbaktetr*s67 + kbaktrimr*s65 + kbidbakc*s46 + 2*kbakdimr*s56 - bakbclxlf*s23*s53 - bakbclxlf*s53*s9 -
```

~150 ODEs (9 shown)

Higher level complexity: interaction families

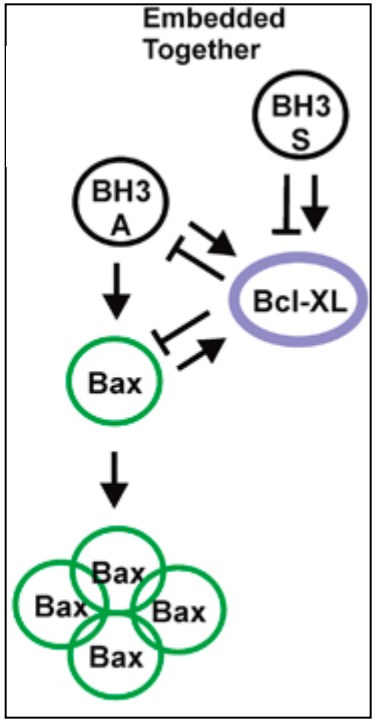



```
def simple_bind(name(opt), sub1, sub2, kf, kr):
    rule_name="complex_formation_"+sub1.name+sub2.name
    A = sub1(site=None)
    B = sub2(site=None)
    AB = sub1(site=1) % sub2(site=1)
    return Rule(rule_name, A + B <> AB, kf, kf)
```

```
def simple_bind_table(table, paramlist):
    rules = []
    reactants0 = table(first-row)
    reactants1 = table(first-column)
    interactions = table(rest-row, rest-column)
    for pair(i,j) in interactions:
        if intrxn[i,j] is True:
            kf = paramlist[i,j]
            kr = paramlist[i,j]
            rule_name = "complex_"+ i.name + j.name
            rules.append(simple_bind(rule_name, i, j, kf, kr))
    return rules
```

Inhibitor/sensitizer tables

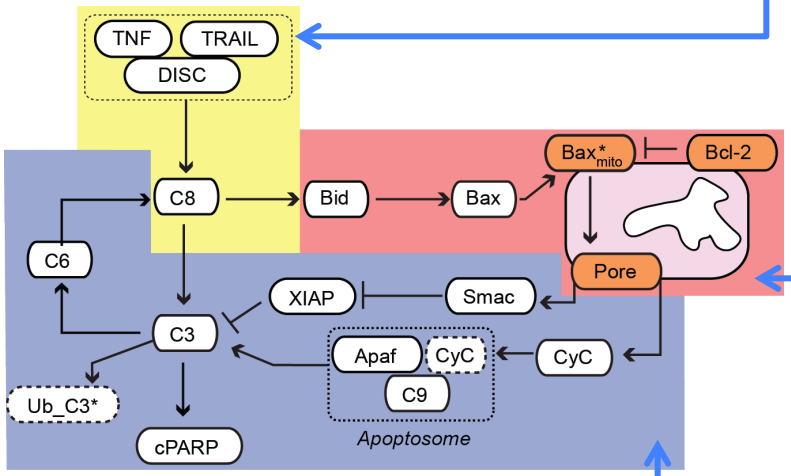
```
# MOMP Inhibition
simple_bind_table([[                      Bcl2, BclxL,  Mcl1],
                   [                        {},    {},    {}],
                   [Bid, {'state':'M'},  True,  False, False],
                   [Bax, {'state':'A'},  True,   True, False],
                   [Bak, {'state':'A'}, False,   True,  True]],
                  kd['BID_BAX_BAK_inh'], model)
# Bcl-2 Sensitizers
simple_bind_table([[           Bcl2, BclxL,  Mcl1],
                   [             {},    {},    {}],
                   [Bad,  {},  True,  True, False],
                   [NOXA, {},  False, True,  True]],
                  kd['BCLs_sens'], model)
```

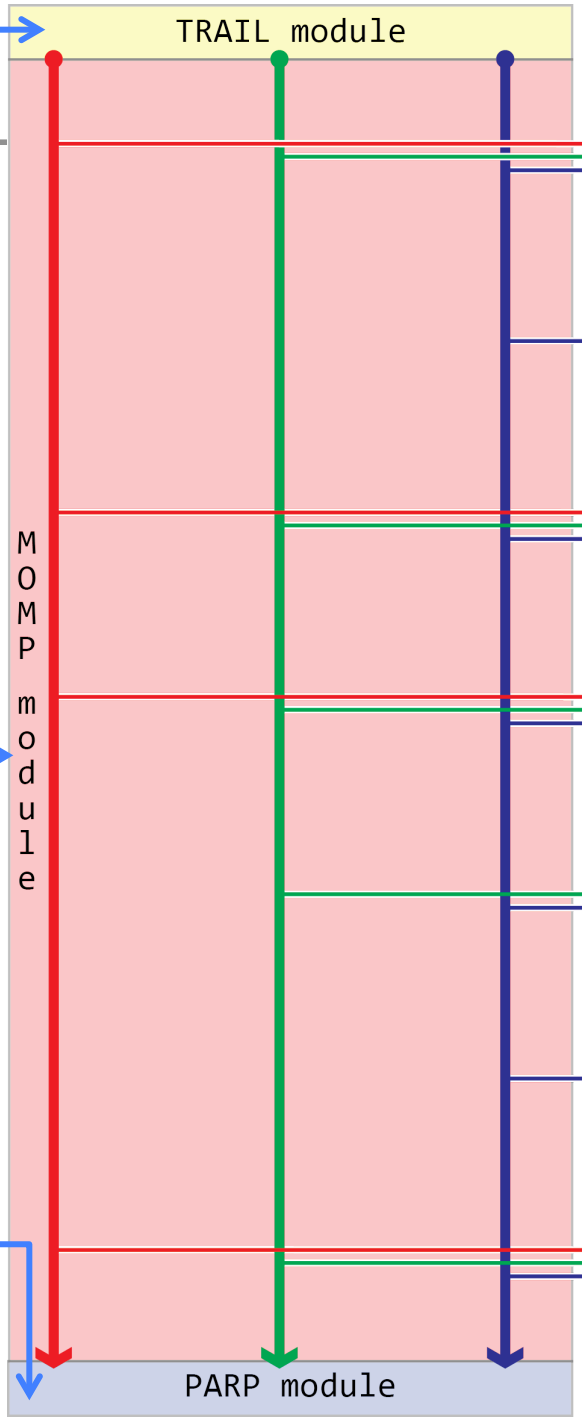


15 potential interactions shown
~50 total potential interactions in MOMP mechanisms

Exploring apoptosis biology

Modules





Function calls

translocate()
Bax, Bcl-xL
Bid, Bax, Bcl-xL
Bid, Bad

assisted_transport()
Bax (activation), Bcl-xL (inhibition)

bind_table(sensitizers)
Bad, NOXA
Bad, NOXA
Bad, NOXA

bind_table(inhibitors)
Bcl-xL, Mcl-1
Bcl-2, Bcl-xL, Mcl-1
Bcl-2, Bcl-xL, Mcl-1

catalytic_activation
Bax (cyto), Bax (mito), Bak
Bax, Bak

auto_activation
Bax, Bak

pore_assembly
Bax(tetramer), Bak(tetramer)
Bax(tetramer), Bak(tetramer)
Bax(tetramer), Bak(tetramer)

Indirect topology code

```
from earm_indirect_mem_parms import parameter_dict as kd
import earm_1_0modules
translocate('Bax_to_mem', Bax, state = 'T', state = 'M'), kbaxtf, kbaxtr)
translocate('Bclxl_to_mem', Bclxl, state = 'T', state = 'M'), kbclxlt, kbclxltr)
pore_assembly(Bax(bf=None, state='A'), 4, kd['BAX_PORE'])
pore_assembly(Bak(bf=None, state='A'), 4, kd['BAK_PORE'])
inhib_table([[                                    BclxL,   Mcl1],
             [                        {'state': 'M'},      {}],
             [Bid,  {'state': 'T'},                True,   True],
             [Bax,  {'state': 'A'},                True,  False],
             [Bak,  {'state': 'A'},                True,   True]],
             kd['BID_BAX_BAK_inh'], model)
sensi_table([[                          BclxL,   Mcl1],
             [             {'state': 'M'},      {}],
             [Bad,   {},                 True,  False],
             [NOXA,  {},                 True,   True]],
             kd['BCLs_sens'], model)
earm_1_0modules.rec_to_bid(model, kd)
earm_1_0modules.pore_to_parp(model, kd)
```

6 lines of function calls
Reusable modules / parameters called from main program
(actual python code)

Direct topology code

```
from earm_direct_mem_parms import parameter_dict as kd
import earm_1_0modules
translocate('Bid_to_mem', Bid, state = 'T', state = 'M'), kbidtf, kbidtr)
translocate('Bax_to_mem', Bax, state = 'T', state = 'M'), kbaxtf, kbaxtr)
translocate('Bclxl_to_mem', Bclxl, state = 'T', state = 'M'), kbclxlt, kbclxltr)
activate(Bid(state = 'T'), Bax(state = 'M'), Bax(bf = None, state = 'A'), kd['BIDt_BAX'])
activate(Bid(state = 'M'), Bax(state = 'M'), Bax(bf = None, state = 'A'), kd['BIDm_BAX'])
activate(Bid(state = 'T'), Bak(state = 'M'), Bak(bf = None, state = 'A'), kd['BIDt_BAK'])
pore_assembly(Bax(bf=None, state='A'), 4, kd['BAX_PORE'])
pore_assembly(Bak(bf=None, state='A'), 4, kd['BAK_PORE'])
inhib_table([[                          Bcl2,          BclxL,   Mcl1],
             [                            {},   {'state':'M'},     {}],
             [Bid, {'state':'M'}, True,           True, False]],
             kd['BID_BAX_BAK_inh'], model)
sensi_table([[                          Bcl2,          BclxL,   Mcl1],
             [                            {},   {'state':'M'},     {}],
             [Bad,   {},                True,           True, False],
             [NOXA,  {},               False,           True,  True]],
             kd['BCLs_sens'], model)
earm_1_0modules.rec_to_bid(model, kd)
earm_1_0modules.pore_to_parp(model, kd)
```

10 lines of function calls
Reusable modules / parameters called from main program
(actual python code)

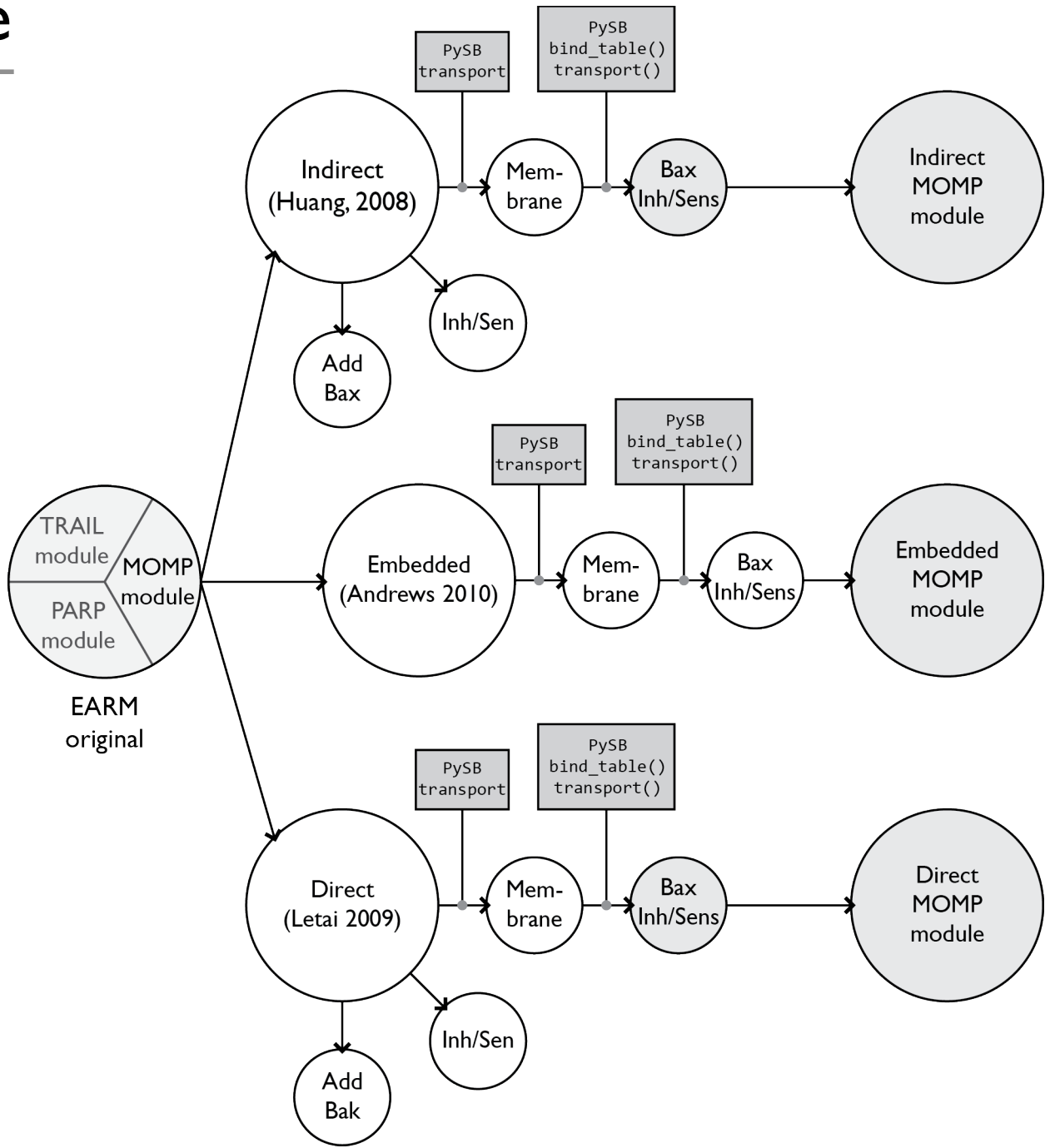
Embedded topology code

```
from earm_2_0_parms import parameter_dict as kd
import earm_1_0modules
translocate('Bid_to_mem', Bid, state = 'T', state = 'M'), kbidtf, kbidtr)
translocate('Bad_to_mem', Bid, state = 'T', state = 'M'), kbadtf, kbadtr)
activate(Bid(state = 'M'), Bax(state='C'), Bax(bf = None, state = 'A'), kd['BID_BAX'])
activate(Bid(state = 'M'), Bak(state='M'), Bak(bf = None, state = 'A'), kd['BID_BAK'])
activate(Bax(state = 'A'), Bax(state='C'), Bax(bf = None, state = 'A'), kd['BAX_BAX'])
activate(Bak(state = 'A'), Bak(state='M'), Bak(bf = None, state = 'A'), kd['BAK_BAK'])
pore_assembly(Bax(bf=None, state='A'), 4, kd['BAX_PORE'])
pore_assembly(Bak(bf=None, state='A'), 4, kd['BAK_PORE'])
activemod(Bid(state='M'), BclxL(state='C'), Bid(bf=1, state='M')%BclxL(bf=1, state='M'), kd['Bid_BclxL_RA'])
activemod(Bax(state='A'), BclxL(state='C'), Bax(bf=1, state='A')%BclxL(bf=1, state='M'), kd['Bax_BclxL_RA'])
inhib_table([ [                    Bcl2,          BclxL,   Mcl1],
              [                      {},   {'state': 'M'},     {}],    #STATES
              [Bid,  {'state': 'M'},   True,           True,  False],
              [Bax,  {'state': 'A'},   True,           True,  False],
              [Bak,  {'state': 'A'},  False,           True,   True]],
              kd['BID_BAX_BAK_inh'], model)
sensi_table([[                     Bcl2,          BclxL,   Mcl1],
              [                      {},   {'state': 'M'},     {}],    #STATES
              [Bad,   {'state': 'M'},   True,           True,  False],
              [NOXA,  {},              False,           True,   True]],
              kd['BCLs_sens'],  model)
earm_1_0modules.rec_to_bid(model, kd)
earm_1_0modules.pore_to_parp(model, kd)
```

12 lines of function calls
Reusable modules / parameters called from main program
(actual python code)

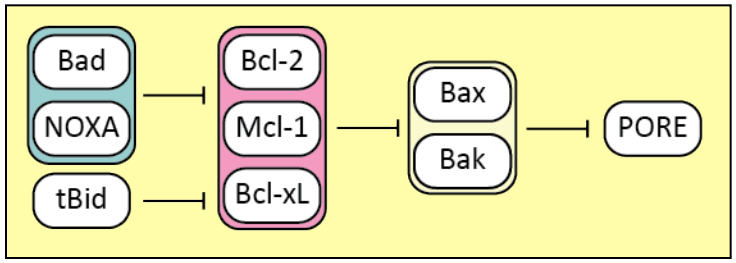
Expand + explore

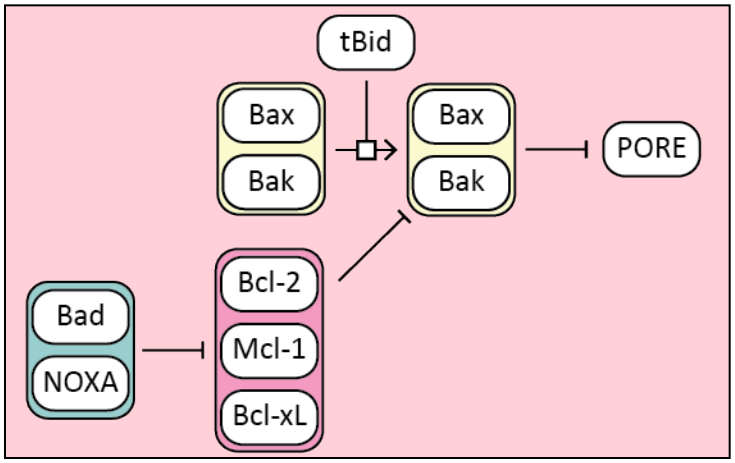
- Explore model biochemistry.
- Update/correct previous models.
- Generate model topologies for numerical testing.

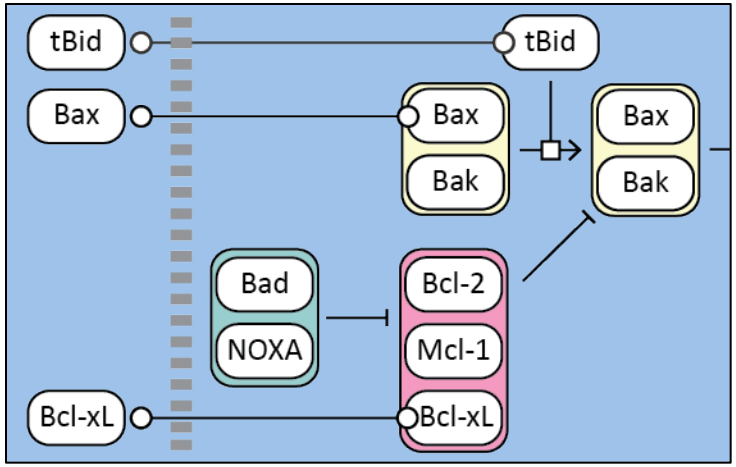


PySB generated models

Attribute / Model	PySB statements	BNG-Rules	Species/ODEs	Reactions	Parameters	Non-zero Initial
Indirect	6	51	81	128	108	20
Direct	10	56	72	95	116	20
Embedded	12	65	197	659	137	21







Cross platform access

PySB Model

```
# Instantiate the model
Model()

Monomer('tBid', ['b'])
Monomer('Bax', ['b', 'state'], {'state': ['I','A']})

two_step_mod(tBid(b=None), Bax(b=None, state='I'),
             Bax(b=None, state='A'), site='b')
Rule('Bax_inactivation',
      Bax(b=None, state='A') >> Bax(b=None, state='I'),
      Parameter('k_rev', 1e-1))

# Set the initial conditions
Initial(tBid(b=None), Parameter('tBid_0', 1000))
Initial(Bax(b=None, state='I'), Parameter('Bax_0', 1000))

# Model output: activated Bax
Observe('aBax', Bax(state='A'))
```

```
# Run simulation and calibrate
output= odesolve(model, 20000)
annealout = annlodesolve(model...)
```

Sundials/CVODE Integrators

```
# Solve for st. state aBax using SymPy
cons_eqns = parse_expr('s0 + s1  ...')
solve(model.odes + cons_eqns, s5)
```

SymPy

```
# Run a deterministic ODE simulation
det_time = linspace(0, tmax, numpoints)
det_data = odesolve(model, det_time)
```

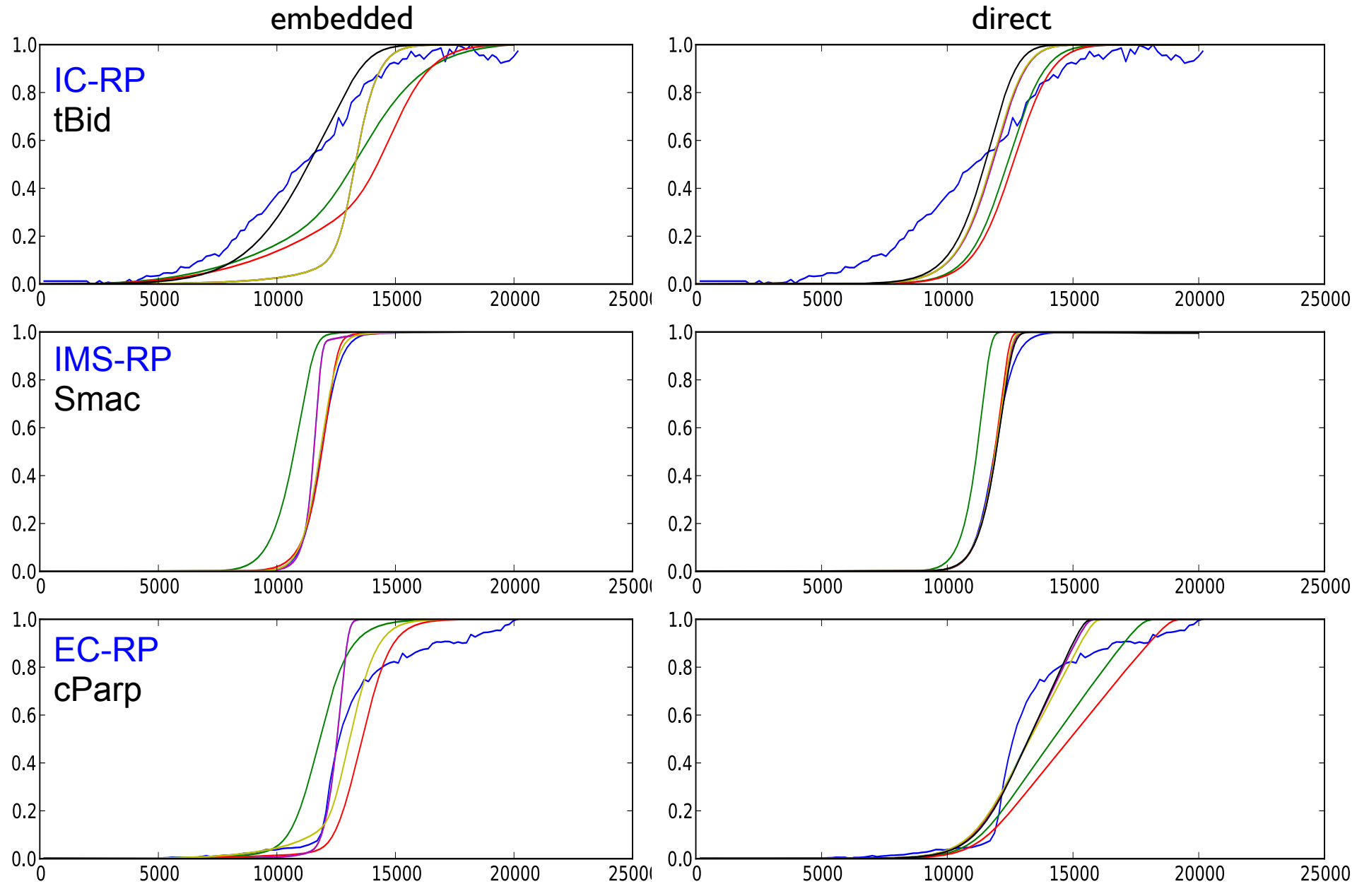
BNG,

```
# Run a stochastic simulation using kappa
stoch_data = kappa.get_kasim_data(model,
             time=tmax, points=numpoints)
```

Kappa Stochastic Simulator

Figure: John A. Bachman

Calibration: SciPy, PySundials, BNG, Pylab, SymPy…



Summary

- Discerning among topologies in a systematic manner: develop metrics to compare models.
- Exploit modularity further among different cell lines.
- Transparency, reuse, revise, share models online.
- Integration with existing and new technologies.
- Interactive/ systematic hypothesis testing.

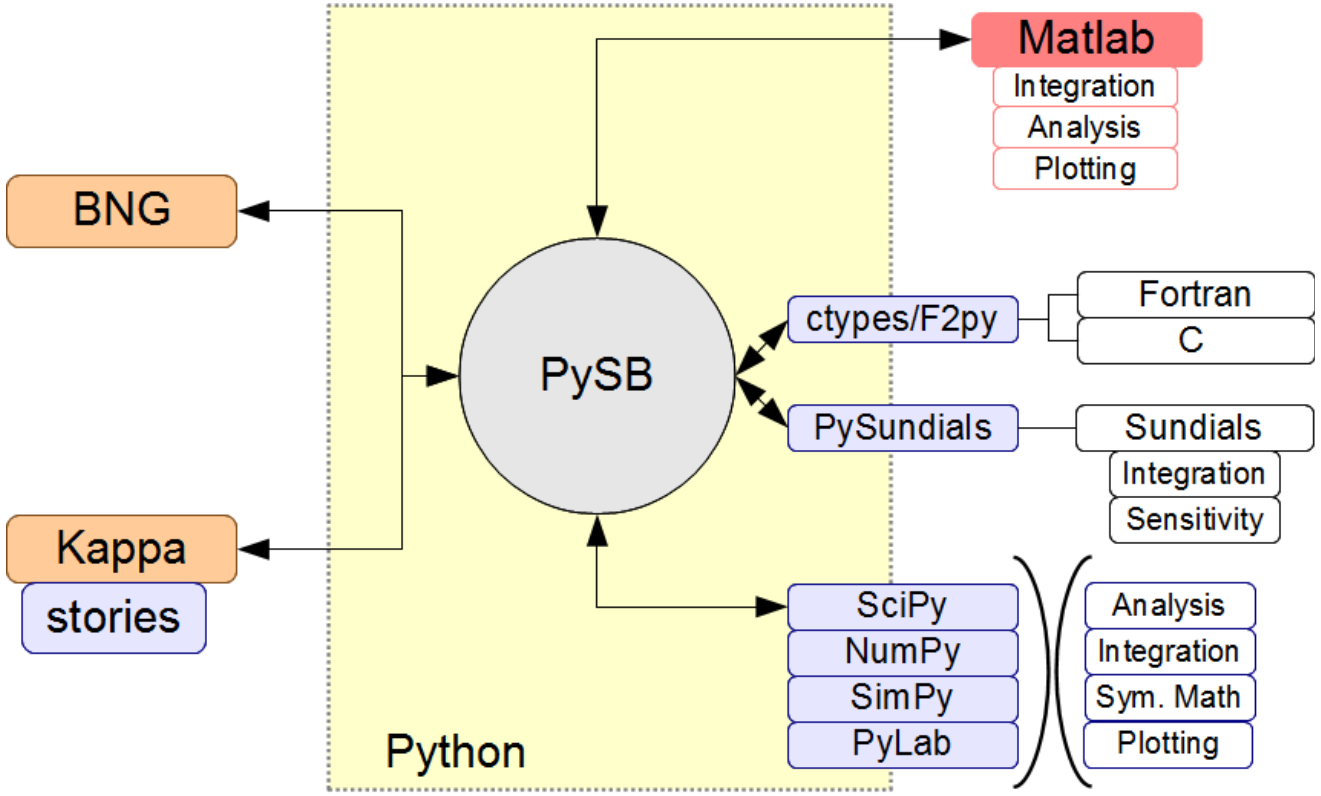


Figure: Andrews et al, Oncogene, 2010

Acknowledgements

SBFM:

- Organizers:
- Dr. Patrick Cousot, Dr. Jim Faeder, Dr. Jerome Feret
- NYU Staff

Harvard Medical School

- Peter K Sorger
- Jeremy Muhlich (PySB, SBWiki)
- John Bachman (PySB)

Funding:

- NIH Transition to Independence (K22CA151918)
- King Trust (Med Foundation) HWP Fellowship
- NIH Minority

Coming this Fall: