Two Techniques for Software Engineering:

Reactive Animation and Smart Play-Out

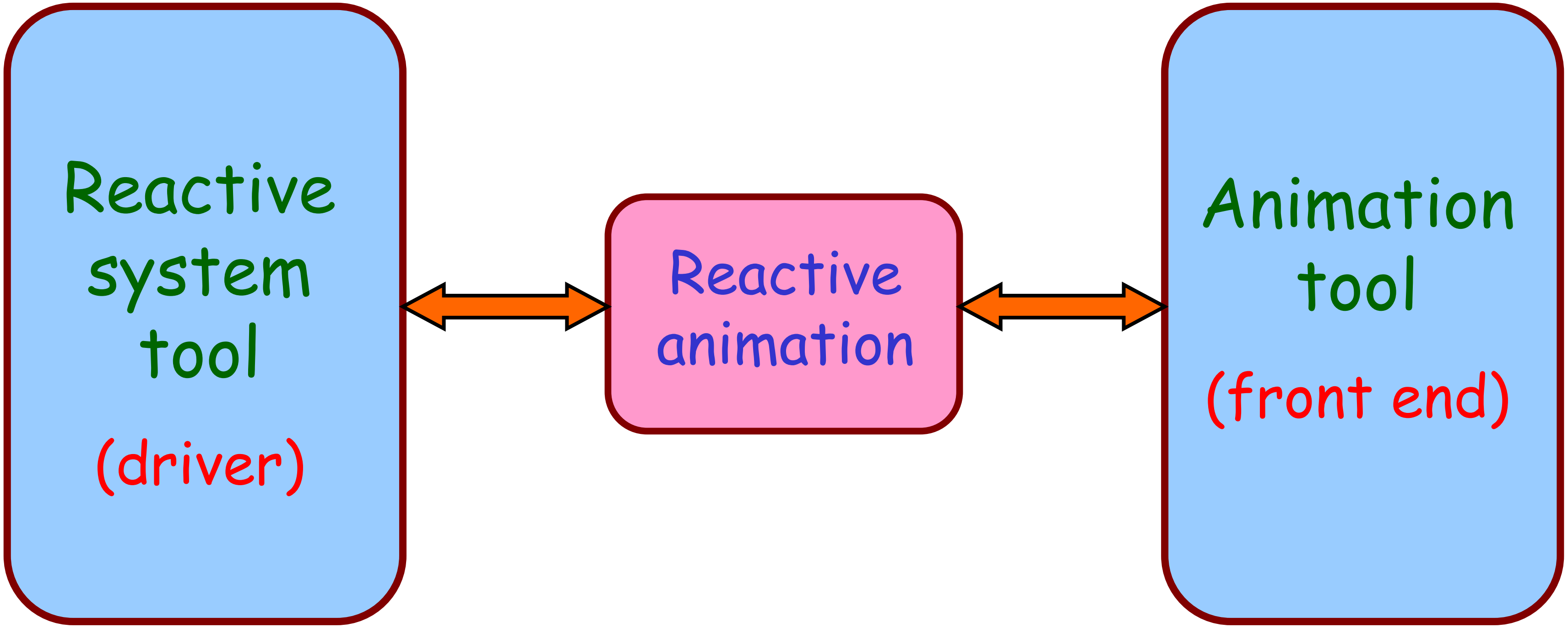
David Harel
The Weizmann Institute of Science

I. Reactive animation

Linking a state-of-the-art reactive system engine with a state-of-the-art animation system

Motivation: complex reactive systems with numerous objects, for which standard kinds of GUIs are inadequate as a front end

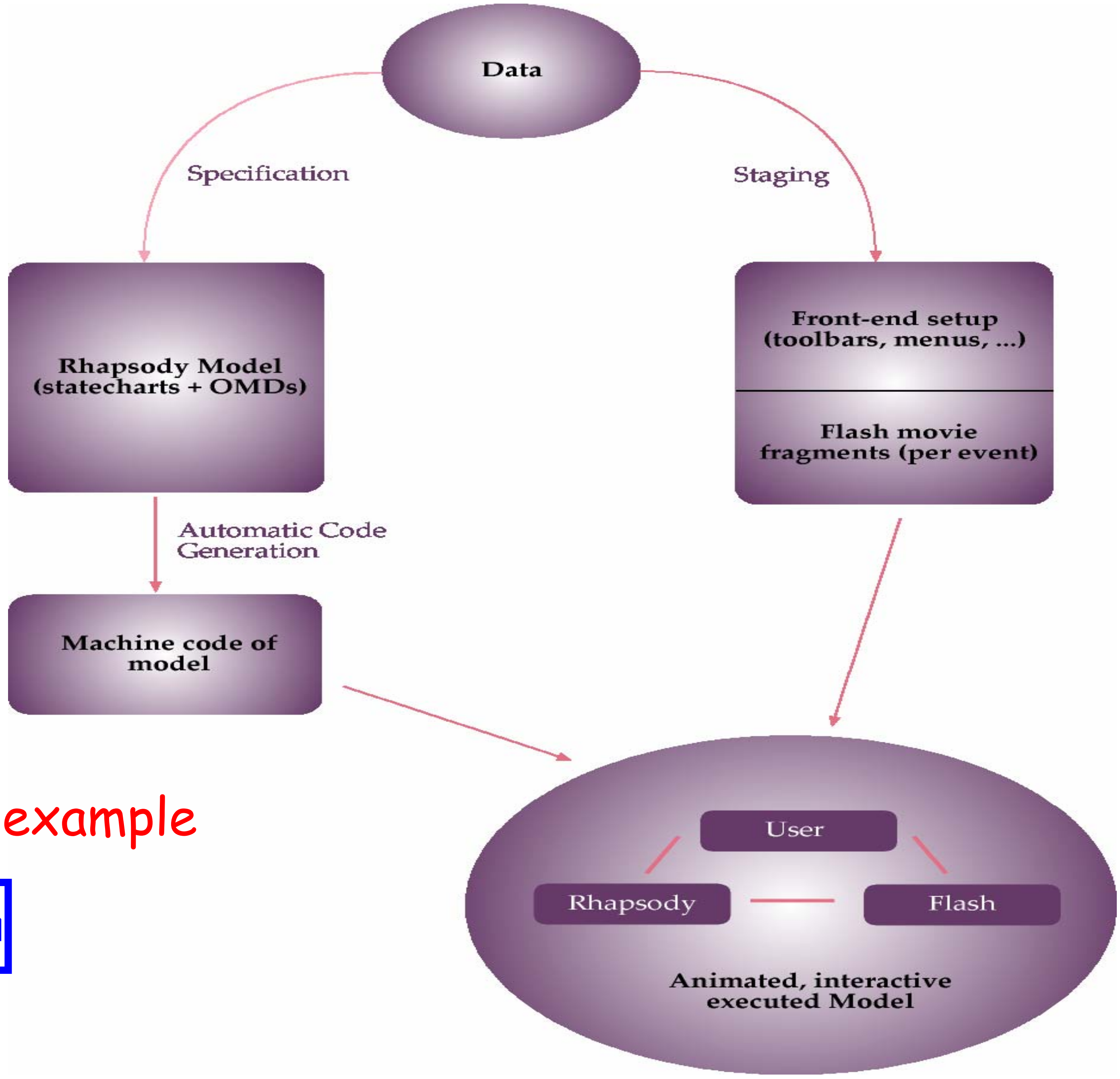
Benefits: relevant to a wide variety of application areas; flexible and realistic; the best of both worlds


Reactive system tool
(driver)
Reactive animation
Animation tool
(front end)

Our main example so far:

Connecting the Rhapsody tool supporting statecharts, with Flash from Macromedia

(with S. Efroni and I. Cohen)



Traffic example

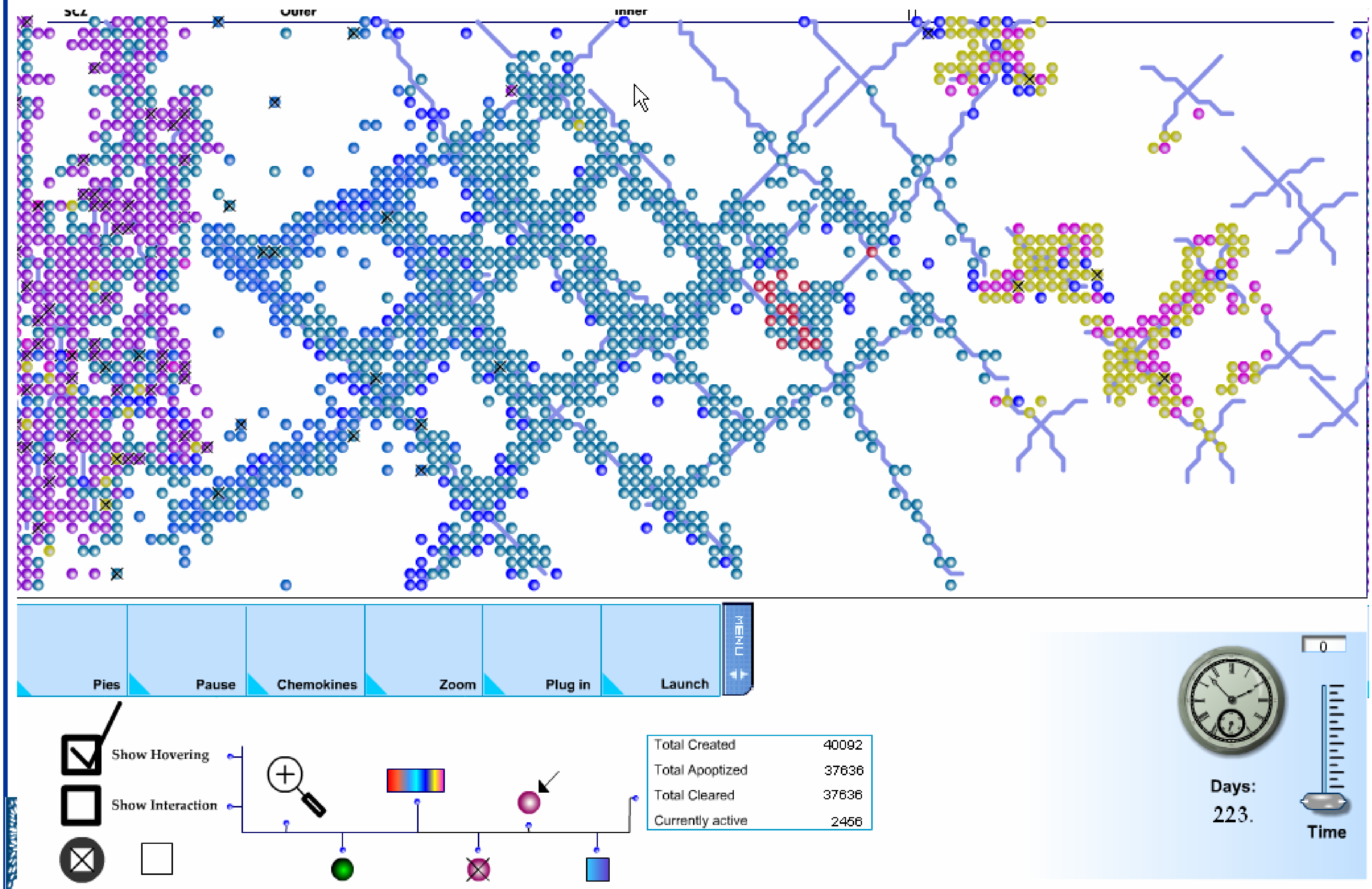
The British humor view on the ability to predict behavior by simulation…

Biological example

(with S. Efroni and I. Cohen)

- T-cell (thymocyte) behavior in the thymus.
- Many cells of few types, internal behavior, complex interaction, geometric movement.
- An enormous amount of biological data assimilated, assessed and modeled (around 300 papers).

Flash front-end of entire lobule at runtime



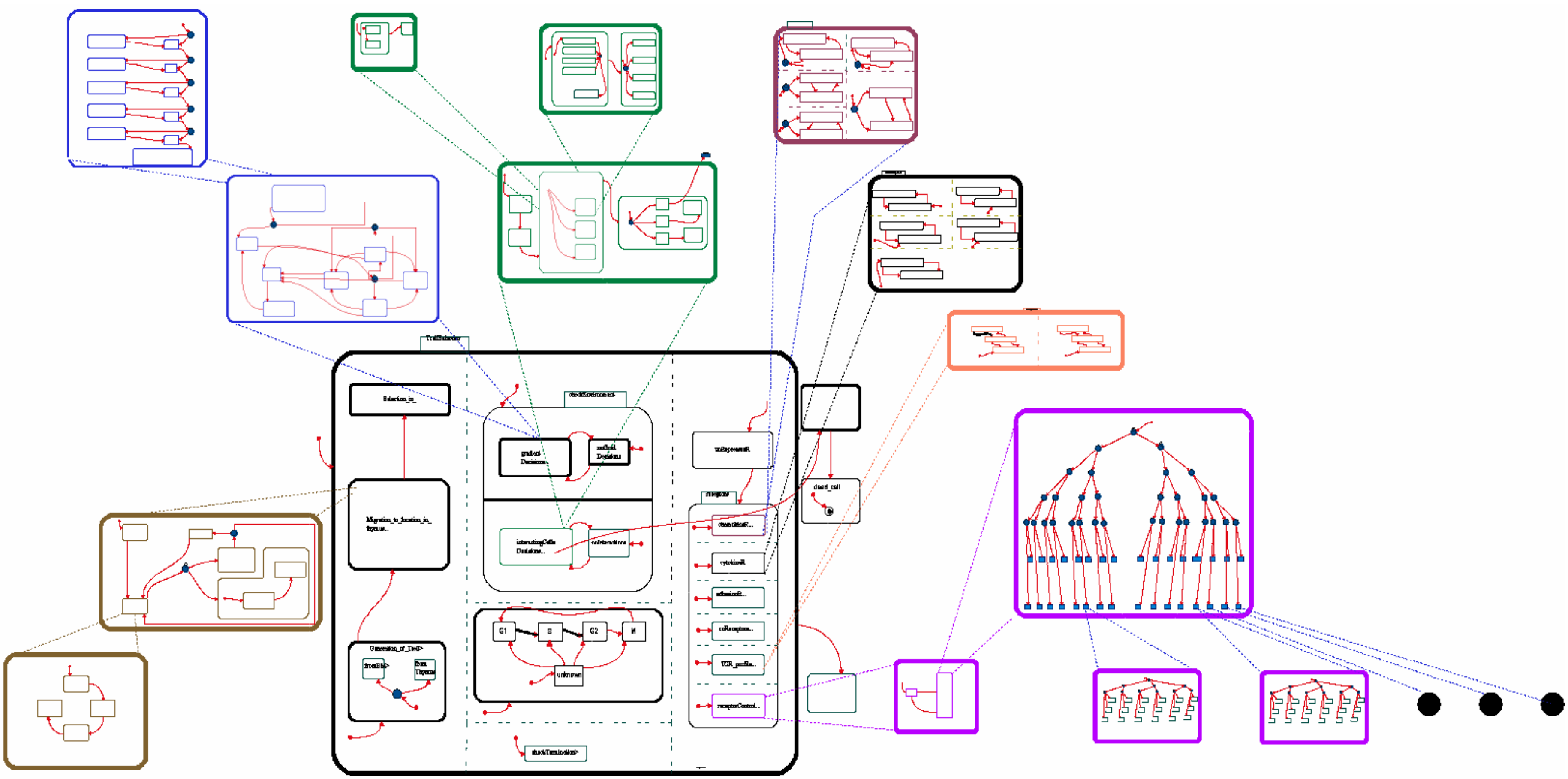
Pre-recorded demos

T cells in the thymus

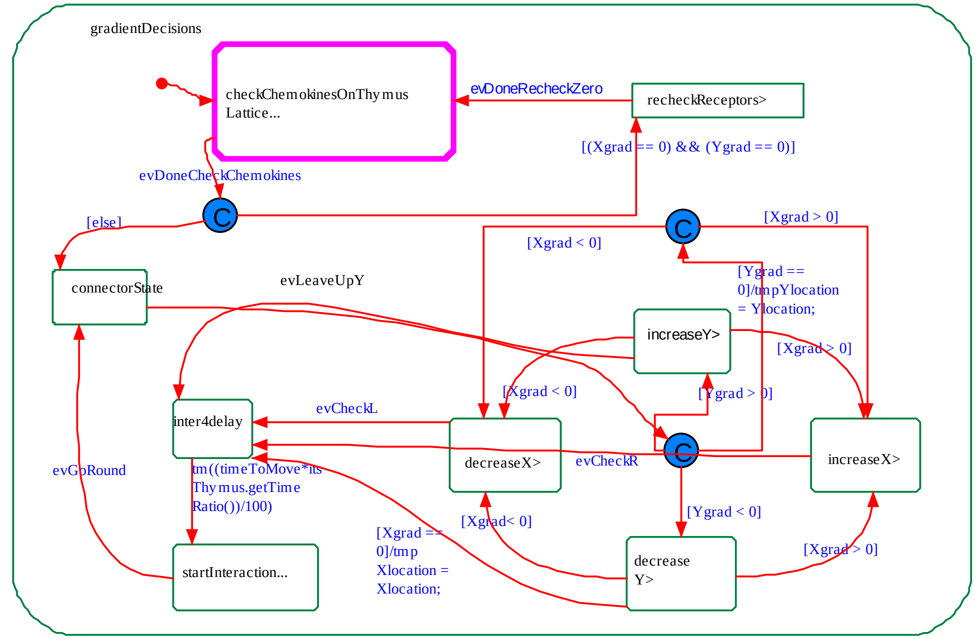
days 9 to 13

27 days

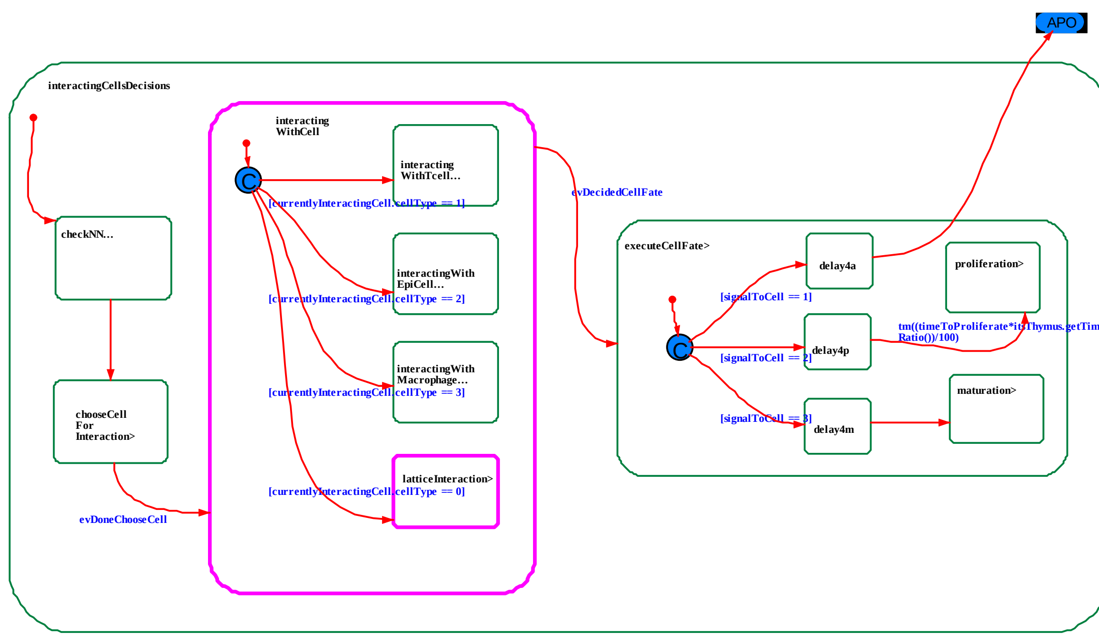
Pseudo statechart of one T-cell



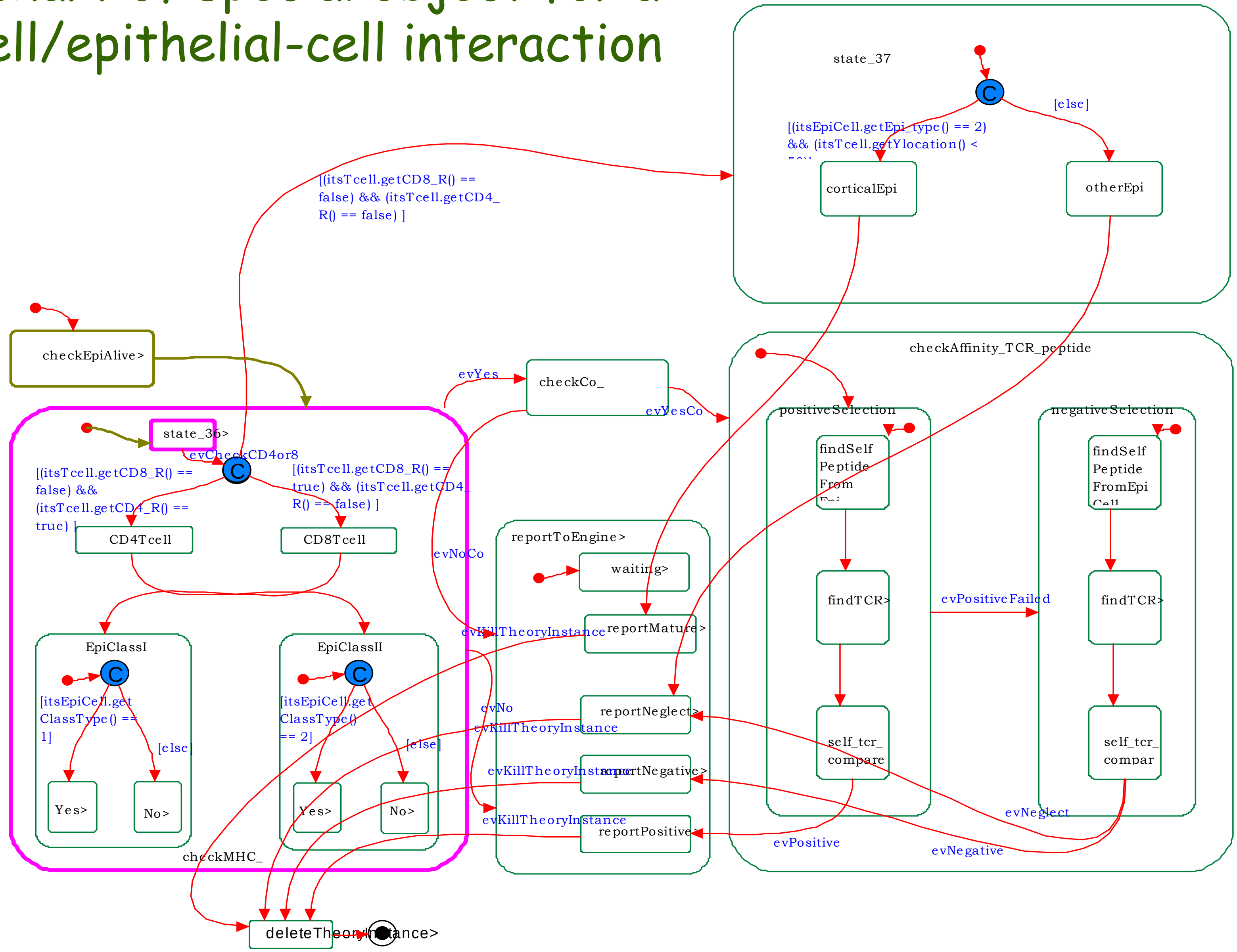
Sub-chart responsible for movement



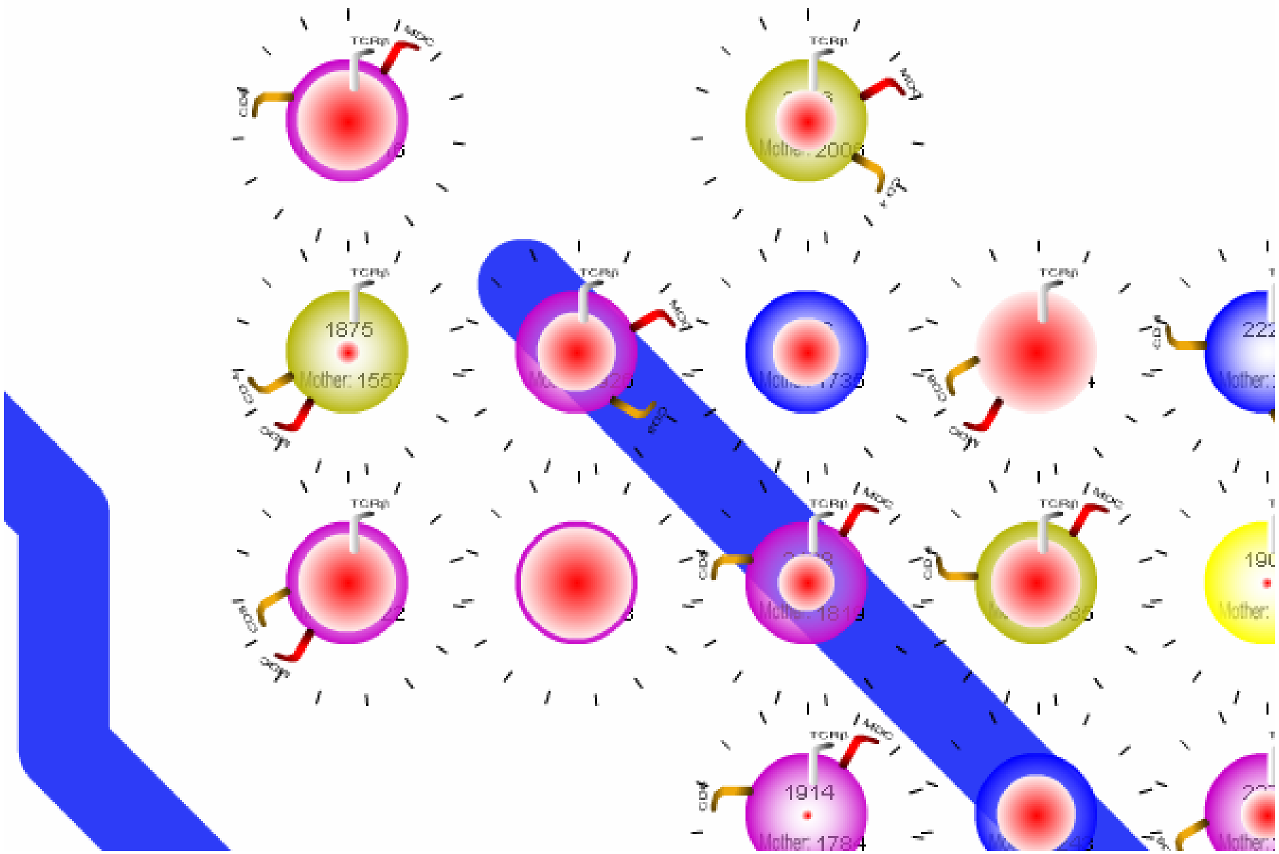
Sub-chart responsible for interactions with other cells



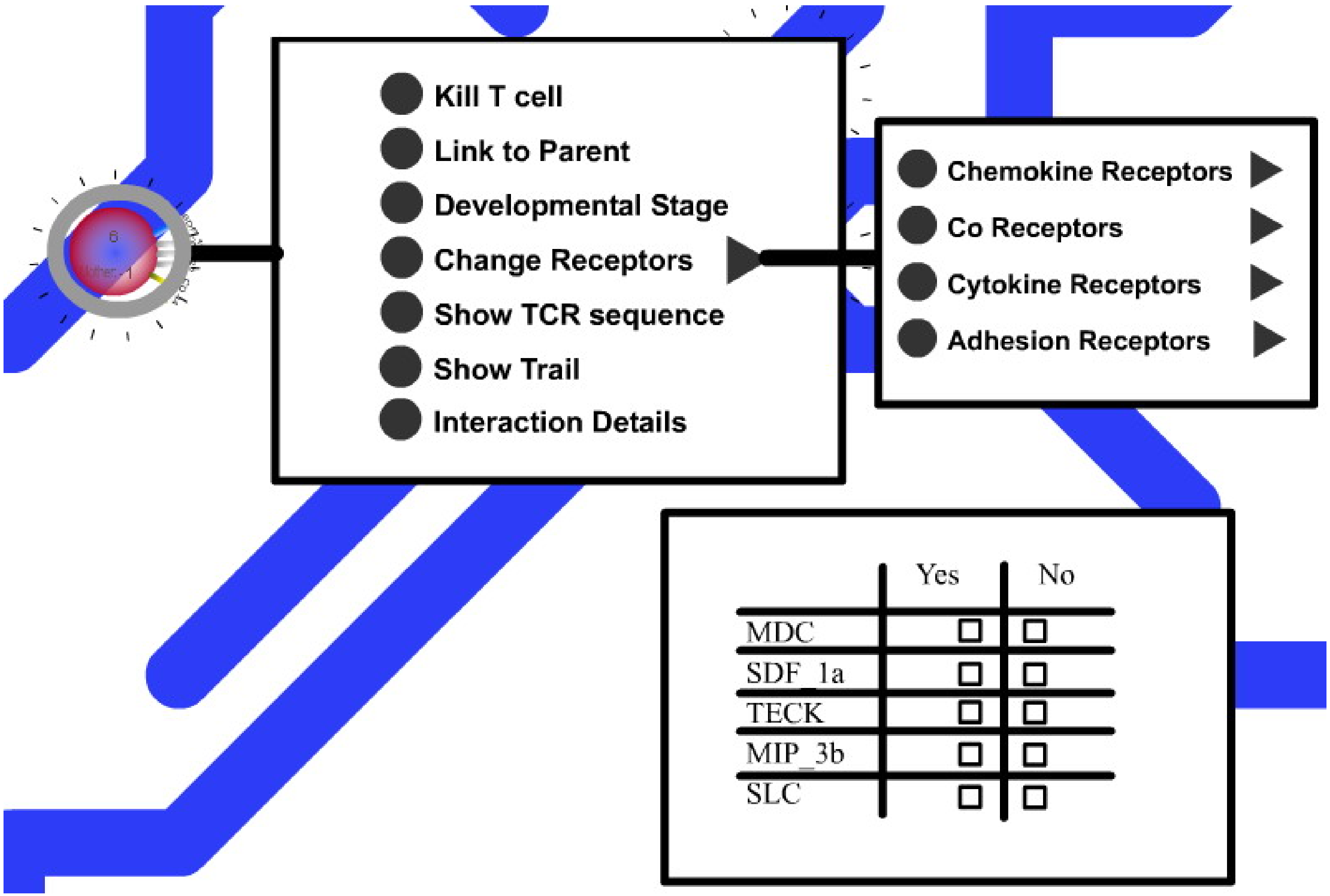
Statechart of special object for a T-cell/epithelial-cell interaction



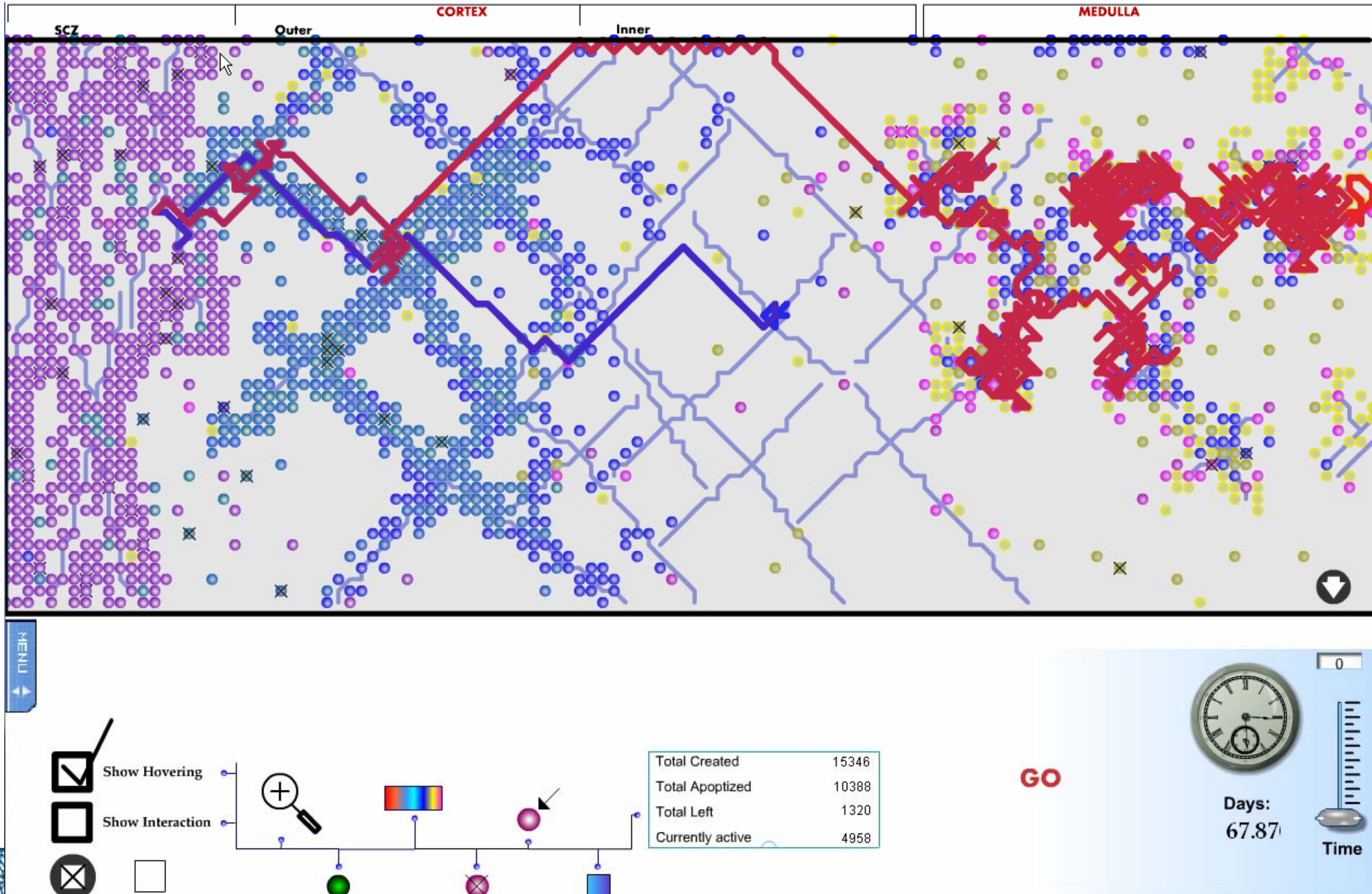
Zooming in



Experimentation in silico



Cell migration



Some additional goodies

- We are not committed to a particular theory (e.g., of cell interaction or movement). Several such are modeled, and are selectable at pre-run or at run-time.
- We have associated the source publications with the model and the front-end in a way that facilitates easy retrieval.

Pre-recorded demos

T cells in the thymus

statecharts running

zooming

show paper

mid-run changes

reactive animation illustrated

We envision numerous applications of reactive animation

Are working on a second example -- linking the Play-Engine with Maya , e.g., incorporating 3D -- and on making the idea technically generic

II. Smart Play-Out

Using hard-core verification tools to run real-world programs, rather than to prove properties thereof

Motivation: declarative, logical or constraint-based languages, whose inherent execution mechanism is highly nondeterministic

Benefits: predictive scenario-based programming, smart executable requirements and use cases, powerful testing, etc.

Our main example so far:

Using model-checking to run live sequence charts (LSCs) in the Play-Engine environment

(with H. Kugler, R. Marelly and A. Pnueli)

Live sequence charts (LSC's)

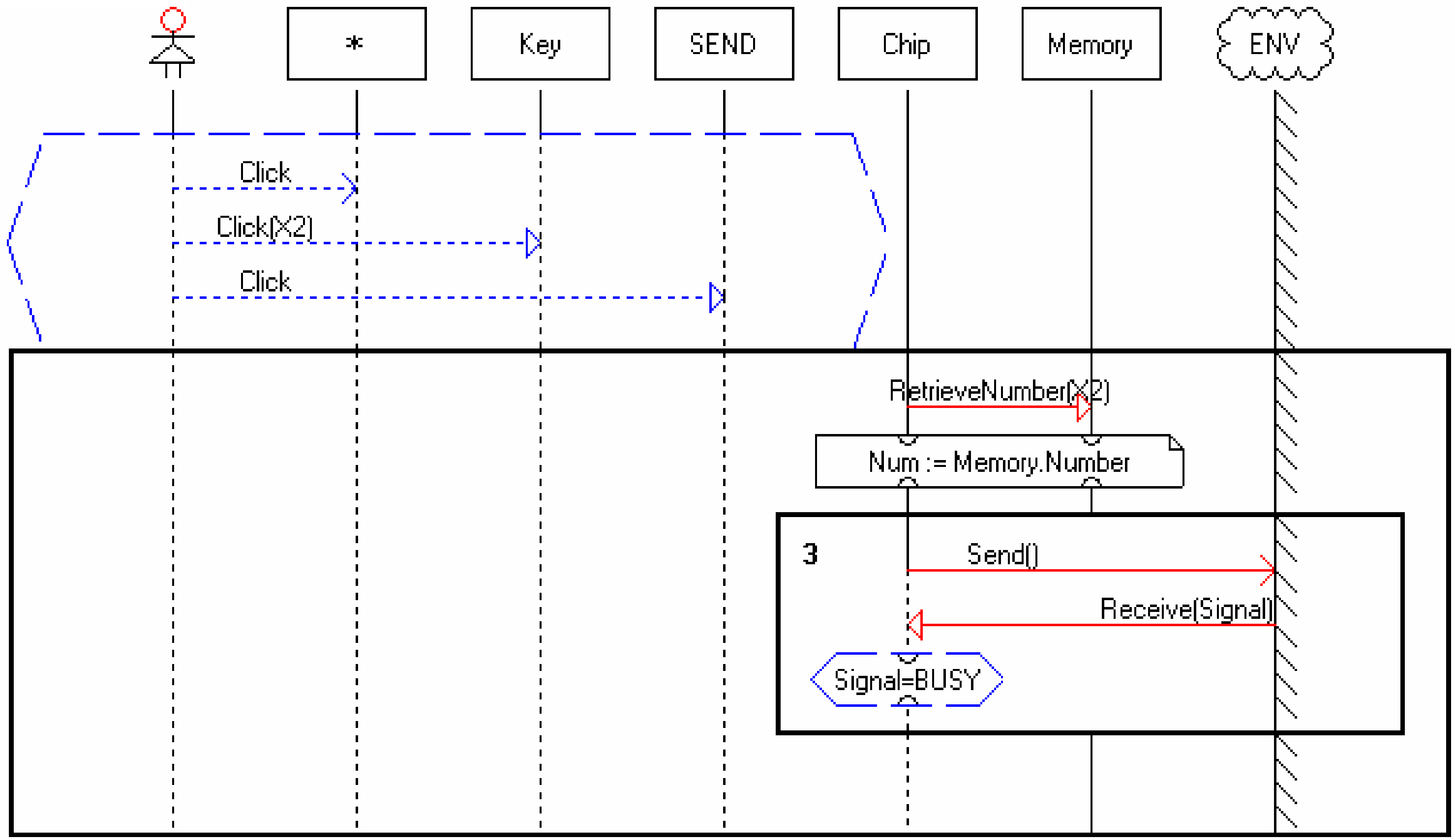
(Damm & H, '98)

A natural extension of classical MSCs, with **modalities** (universal/existential, hot/cold, etc.) and **structure** (subcharts, conditionals, loops, etc.)

Basic form of a (universal) LSC

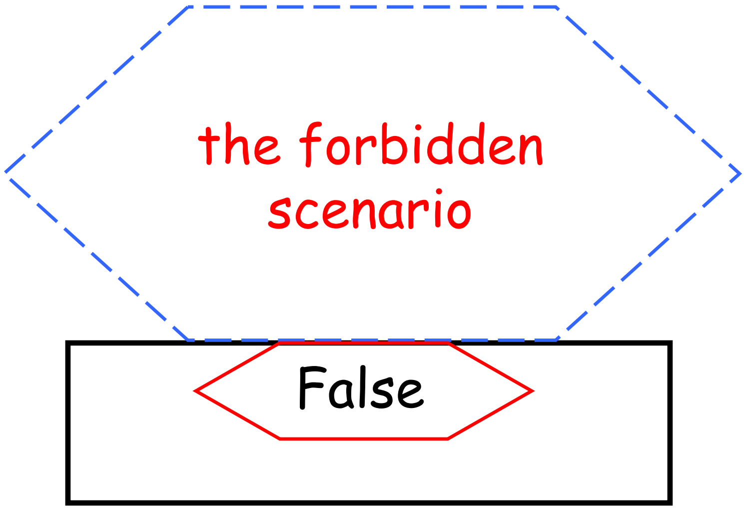
precharto
(if)

main chart
(then)



(similar to [a]<b> in dynamic logic)

- Subcharts
- Loops
- Cold conditions enable control structures
- Hot conditions enable anti-scenarios:



Play-in/Play-out
(H & Marelly '99-'03)

- Extensive strengthening of the 1998 version of LSCs (e.g., symbolic instances, time & real-time, weighted choice, forbidden elements,…)
- Play-In (friendly & convenient GUI-based capture)
- Play-Out (execution techniques & algorithms)

The Play-Engine: Play-Out

Play-out works like an over-obedient, but strictly minimalistic citizen, zealously adhering to the Book of Rules.

- Universal charts drive the execution; relevant chart copies started and monitored continuously; instances & variables bound on the fly.

(external event; step*; stable?) = superstep

Hot stuff *will* be done, cold stuff *might*.

Play-out demo

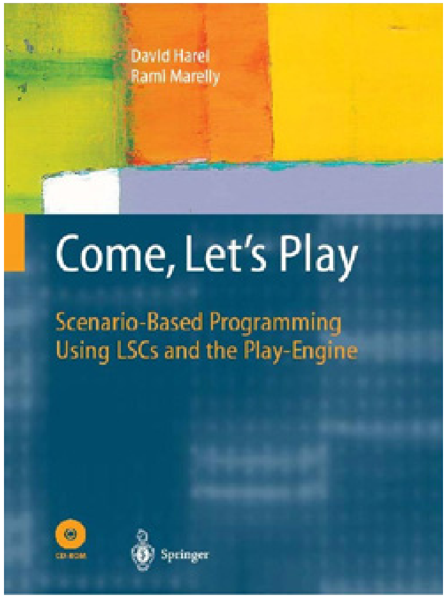
At the very least, this enhances many aspects of the standard system design process:

executable requirements,
"deep" prototyping,
runnable test suites,
solid basis for synthesis, etc.

But why not be a lot more ambitious??

Can use LSCs and the Play-Engine to program a system as a final implementation

Recent book attempts to describe it all:



Come, Let's Play:

Scenario-Based Programming Using LSCs and the Play-Engine

D. Harel and R. Marelly

Springer, June 2003

(includes the Play-Engine software and formal operational semantics:)

“Smart” Play-out

- LSCs may give rise to different legal runs, even within supersteps, due to partial order within a chart, and multiple charts interleaving.
- Play-engine takes a practical approach: implements policies and heuristics to execute system runs, not controllable by the user (except by explicit acts programmed into the LSCs themselves).
- Applying powerful methods taken from program verification can help find the “correct” run or identify inconsistencies.

- Goal 1 : compute a superstep; that is, figure out a "good" series of responses of the system to an action from the user or the environment, and drive the play-engine's execution.

- Goal 2 : compute a way to satisfy a full existential chart; that is, figure out a "good" sequence of events that will drive the engine to satisfy a test scenario.

- Approach :

- Formulate the goal as a generic verification problem.
- Perform model-checking (TLV, CMU-SMV…).
- Model-checker produces a desired super-step (if there is one), or the sought-after run of the entire existential chart (if there is one).

The translation

Variables:

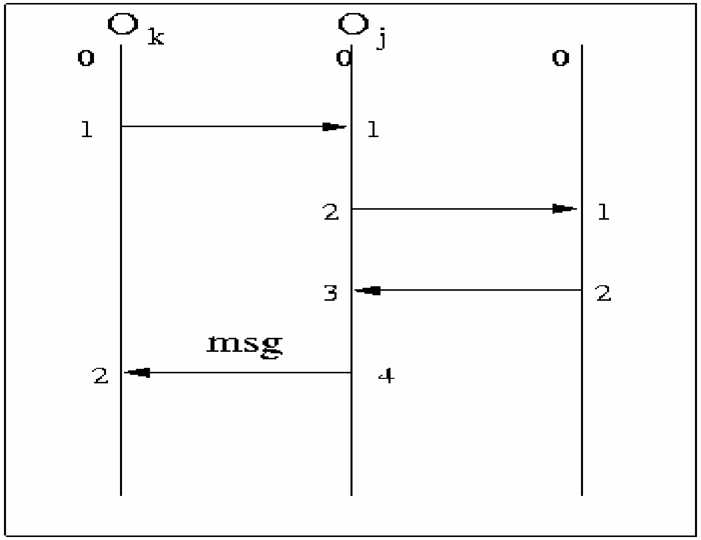
act_{m_i} - chart m_i is active (in main chart)

$msg^{s}_{O_j \rightarrow O_k}$ - O_j sends msg to O_k

$msg^{r}_{O_j \rightarrow O_k}$ - O_k receives msg from O_j

l_{m_i, O_j} - O_j's location ($0 \ldots l_{max}$)

Translation relation



$$l'_{m_i,O_j} = \begin{cases} l & \text{if } l_{m_i,O_j} = l - 1 \wedge msg^{s}_{O_j \to O_k}{}' = 1 \\ l - 1 & \text{if } l_{m_i,O_j} = l - 1 \wedge msg^{s}_{O_j \to O_k}{}' = 0 \end{cases}$$

Translation relation (cont.)

There is an active chart causing msg, and all active charts must agree on msg

$$msg^{s}_{O_j \to O_k}{}' = \begin{cases} 1 & \text{if } \phi_1 \wedge \phi_2 \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_1 = \bigvee_{m_i \in M^U \wedge msg^{s}_{O_j \to O_k} \in Messages(m_i)} act_{m_i} = 1$$

$$\phi_2 = \bigwedge_{m_i \in M^U \wedge msg^{s}_{O_j \to O_k} \in Messages(m_i)} (act_{m_i} = 0 \vee \psi(m_i))$$

$$\psi(m_i) = \bigvee_{l_t \text{ s.t. } f(l_t) = msg^{s}_{O_j \to O_k}} (l_{m_i, O_j} = l_t - 1 \wedge l'_{m_i, O_j} = l_t)$$

Chart activation

Chart is active when the prechart reaches maximal locations, and is deactivated when the main chart reaches maximal locations.

$$act'_{m_i} = \begin{cases} 1 & \text{if } \phi(pch(m_i)) \\ 0 & \text{if } \phi(m_i) \\ act_{m_i} & \text{otherwise} \end{cases}$$

$$\phi(m_i) = \bigwedge_{O_j \in Obj(m_i)} (l'_{m_i,O_j} = l^{max}_{m_i,O_j})$$

Model checking for super-step execution

There is an eventual point where none of the universal charts is active:

$$\neg \Box \left(\bigvee_{m_i \in M^U} act_{m_i} = 1 \right)$$

If this is true, the model-checker finds a satisfying run, which is a desired superstep

This is then fed automatically into the Play-Engine for execution

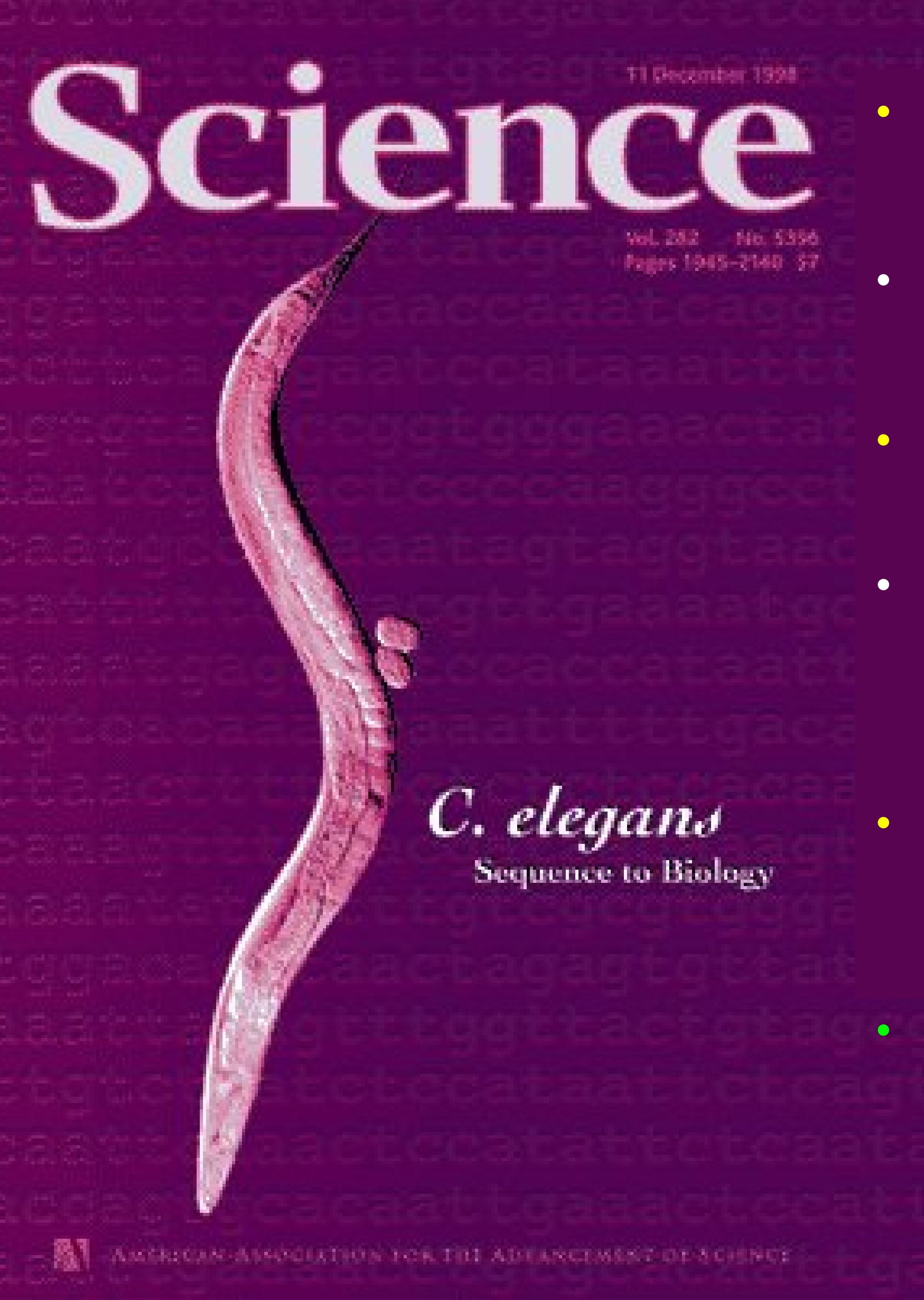
Pre-recorded demo

Being smart helps

Biological example

(with N. Kam, M. Stern, J. Hubbard, A. Pnueli)

Modeling vulval precursor cell fate in *C. elegans*

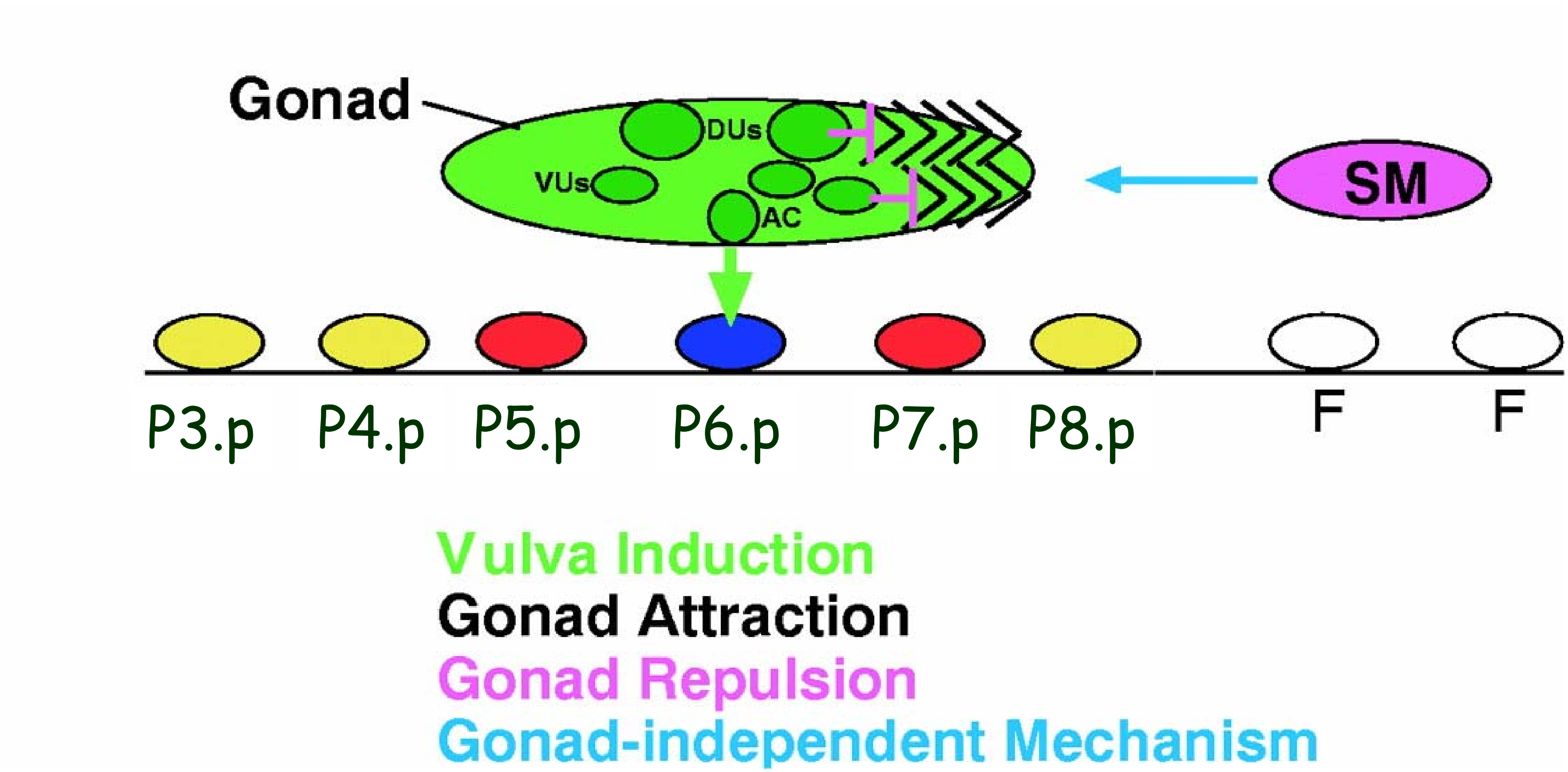


- Small (1mm long) and transparent.
- The most completely described creature ever.
- Studied in about 450 labs worldwide.
- Extremely resilient (survived Feb. '03 Columbia crash)
- Its pioneers (e.g., Sydney Brenner) rec'd 2002 Nobel Prize
- Fixed development (→ wildtype has fixed number of cells with fixed roles).

Development of the Egg-Laying System

Bird's eye

A biologist's "GUI"



(Jungblut et al., 2001)

Pre-recorded demos

Smart Let-23
(make anti-scenario)

Smart P7pAlone
(use anti-scenario)

We envision an increasingly broader potential for smart play-out

Are working hard on strengthening the technique, extending the language features it covers, and linking it to notions of consistency and synthesis

Thank you for listening