THE UNIVERSITY OF ARIZONA

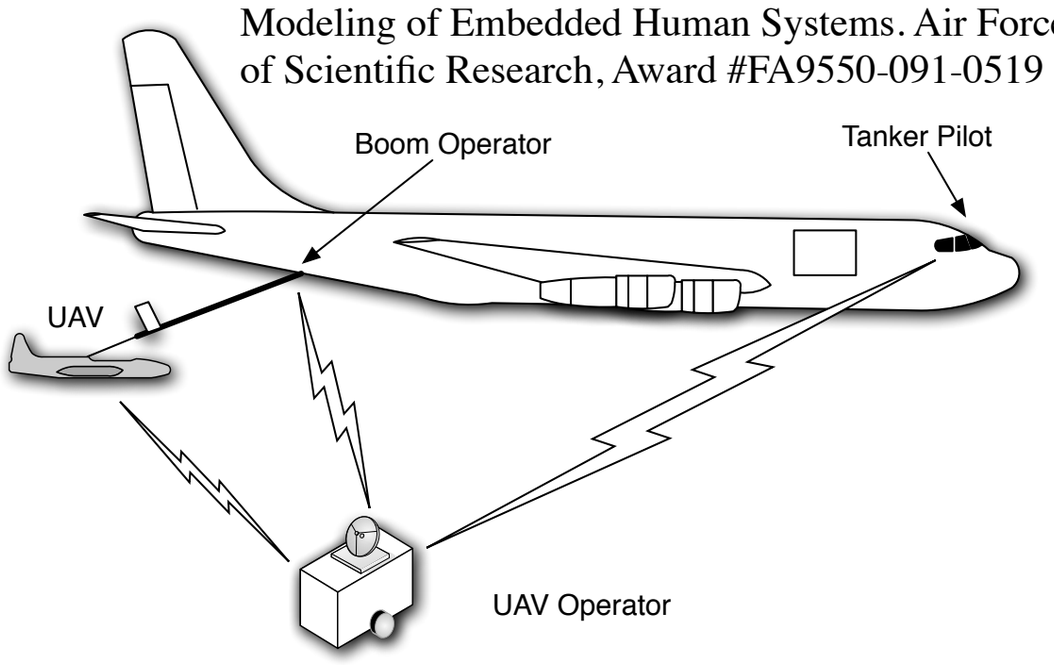
ece

Domain-Specific Modeling: *When Real-World Models Attack!!!*

Jonathan Sprinkle

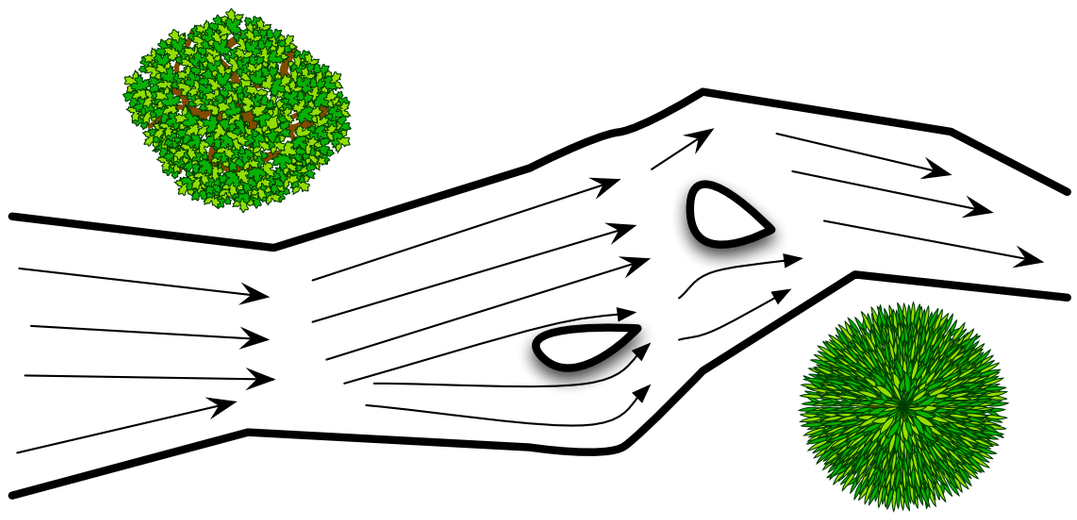
Domain-Models ... why they matter?

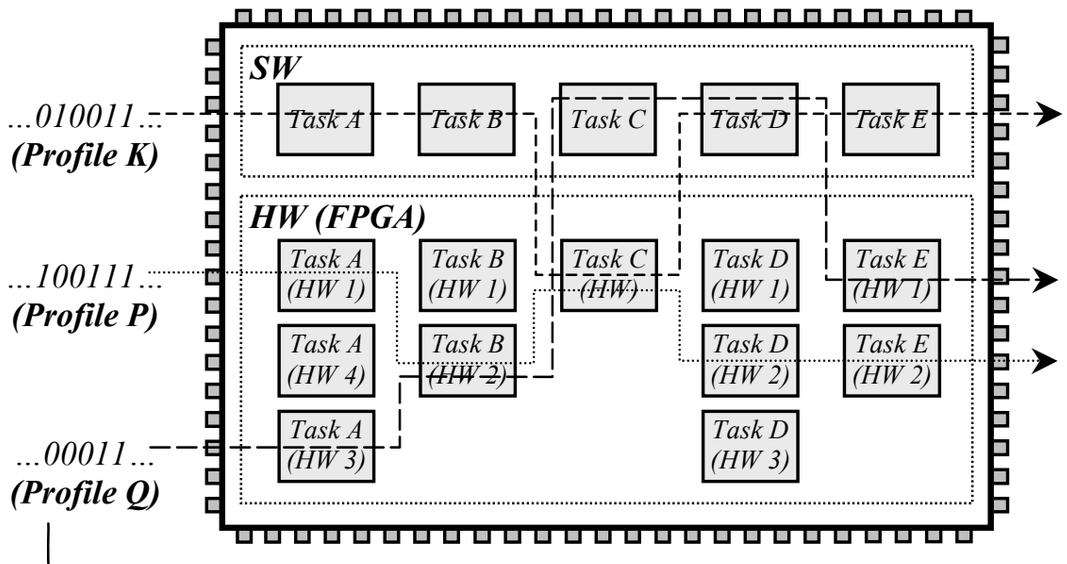
Modeling of Embedded Human Systems. Air Force Office of Scientific Research, Award #FA9550-091-0519



Autonomous Ground Vehicles: Software for Dependable Systems.

Physical Modeling and Software Synthesis for Self-Reconfigurable Sensors in River Environments, NSF CNS-0930919





Data-Adaptable Reconfigurable Embedded Systems (DARES), NSF CNS-0915010

- Fundamentally: Communication + Computation + Control
- Characteristics:
  - Physical and Software are intertwined
  - System operates an different spatial and temporal scales
  - Distributed computation and actuation---not just distributed sensing
- The Upshots
  - Exponential growth in complexity/expense/development
    - In many cases, certification effort is greater than 50% of total cost!!
  - Unpredictability or EPIC FAIL when used outside of conservative design
  - Advancements in a particular problem seldom give insight into other problems in the domain of CPS
- Open problems
  - Composition, robustness, dependability, certifiability, complexity, uncertainty management, abstractions, tools...

See Sprinkle et al., ISSE2006

$$\theta_d(x) = -\left(x_3 - \sin(\theta_{\mathcal{G}})\sqrt{x_1^2 + x_2^2 + x_3^2}\right) + \theta_{\mathcal{G}} \quad (29)$$

$$\psi_d(x) = -(x_2 - \tan(\psi_{\mathcal{G}})x_1) + \psi_{\mathcal{G}} \quad (30)$$

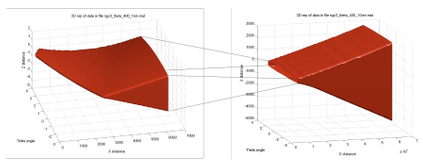
From this, and utilizing (12), we obtain the desired feedback control law:

$$P_{II}(x) = -U^1_{\max}\Gamma_\epsilon(\theta - \mathrm{Sat}^{\theta_{\max}}_\epsilon(\theta_d(x))) \quad (31)$$

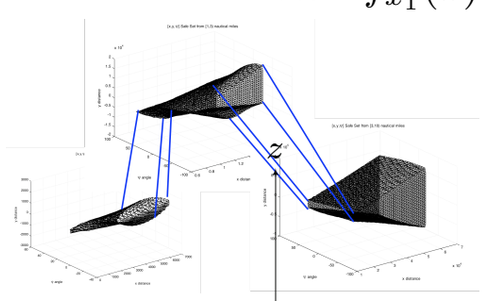
$$Q_{II}(x) = -U^2_{\max}\Gamma_\epsilon(\psi - \mathrm{Sat}^{\psi_{\max}}_\epsilon(\psi_d(x))) \quad (32)$$

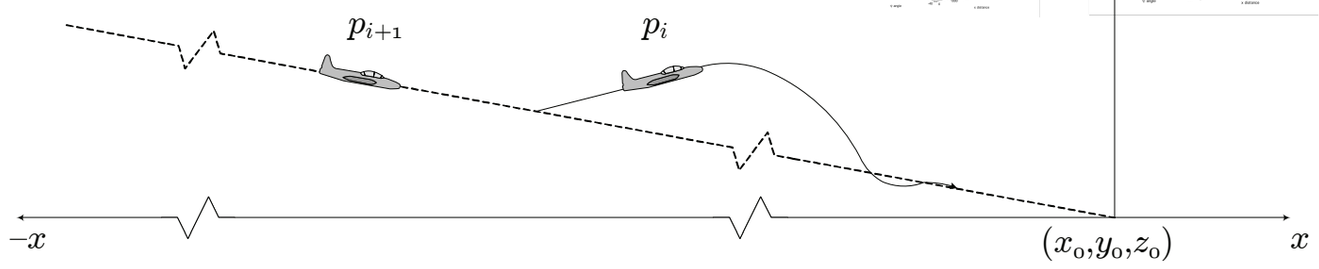
where θ_d and ψ_d are given as in (29) and (30).

$$\mathcal{G}_0 = \begin{cases} \theta_{\mathcal{G}} \in [2.85^\circ, 3.15^\circ], \\ \psi_{\mathcal{G}} \in [-0.2^\circ, +0.2^\circ], \\ x_2 \in [-100, +100] \text{ ft}, \\ x_3 \in [-15, +15] \text{ ft}, \\ x_1 = 0 \end{cases}$$

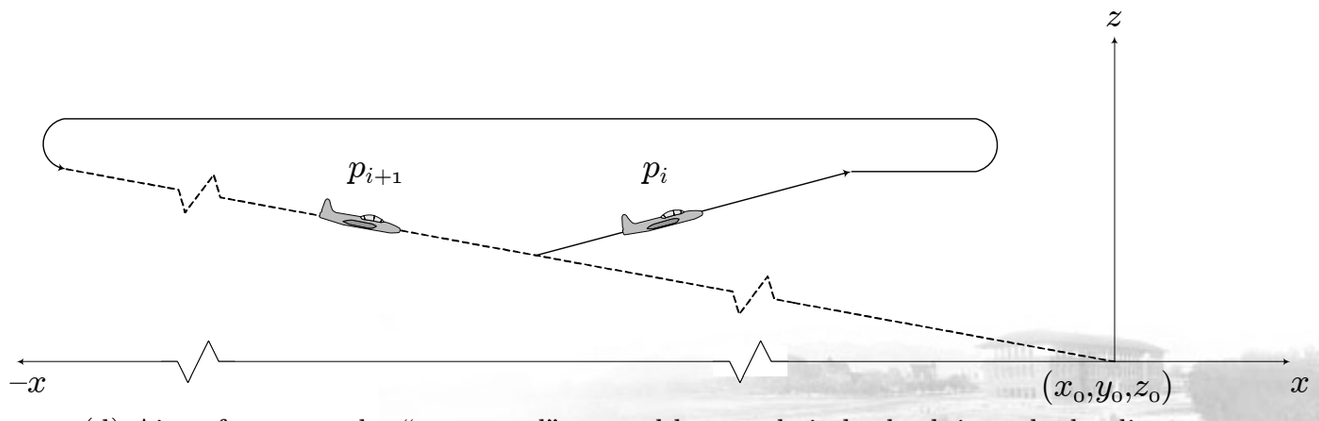


$$H(\tilde{x}, p) = \min_{u \in \mathcal{U}} \frac{p^T \tilde{f}(\tilde{x}, u)}{f_{x_1}(\tilde{x})}$$





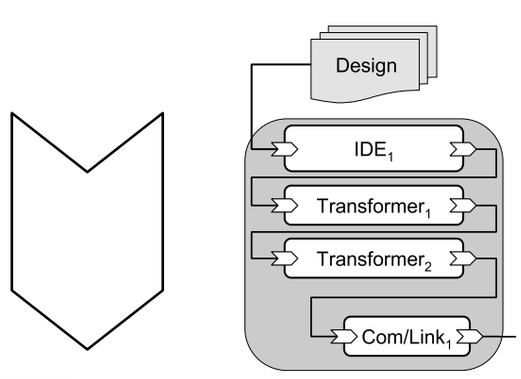
(c) Recapture of aircraft p_i to the glideslope, using the recapture controller according to the $\mathcal{P}_{next}$ formulation.



(d) Aircraft p_i uses the "go-around" control law, and circles back into the landing set.

#include
#include
#define

#include
#include
#define

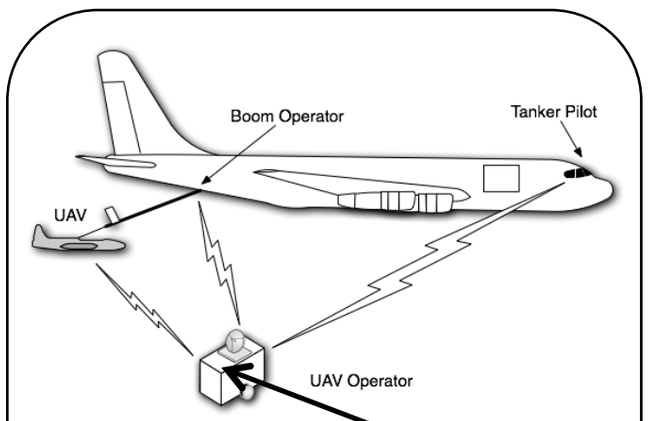


0101001
1001010
1010000
1110101
0100000

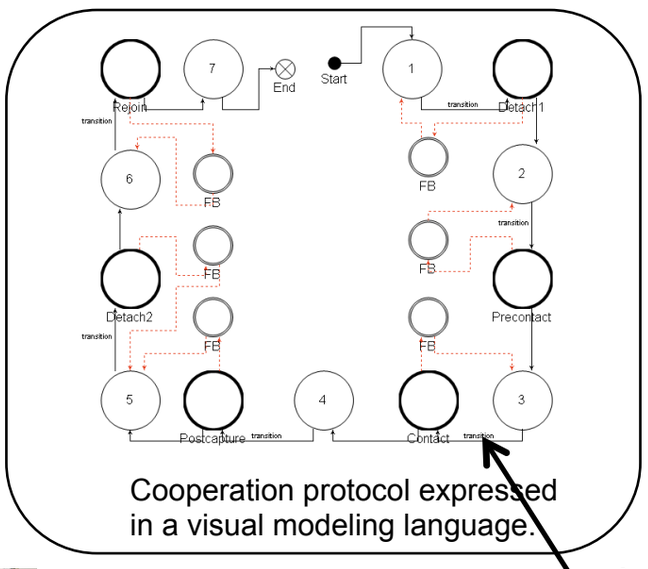
0101001
1001010
1010000
1110101
0100000

Modeling of Embedded Human Systems

THE UNIVERSITY OF ARIZONA
Electrical and Computer Engineering ece

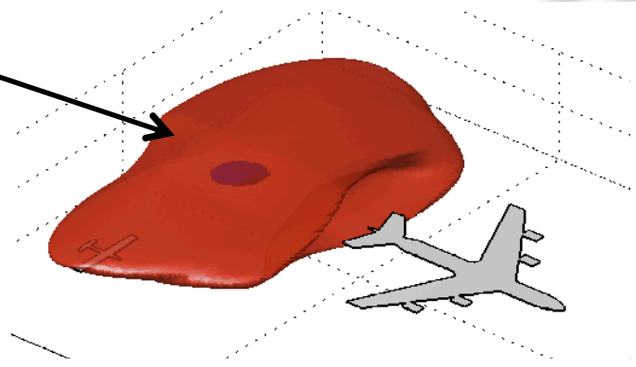


A scenario: cooperating vehicles with human controller in the loop

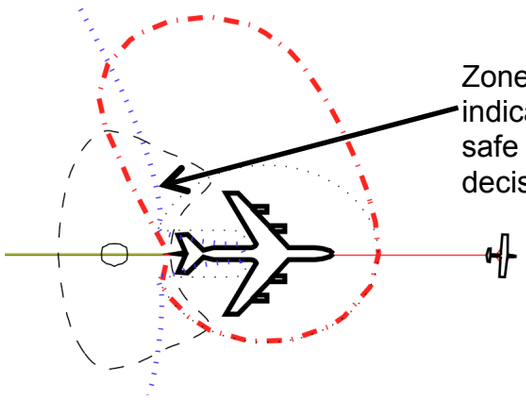


Cooperation protocol expressed in a visual modeling language.

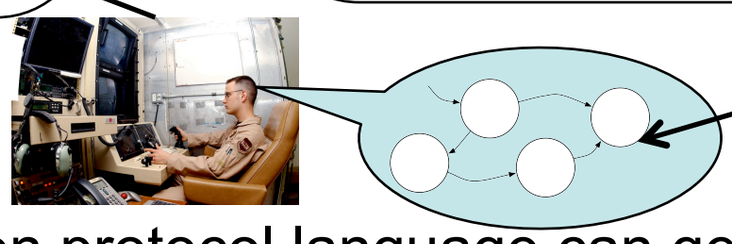
Every mode has its unique zone of safety, based on vehicle parameters and control laws



Synthesis of software



Zone overlap indicates multiple safe and unsafe decisions



Actions of human are received events in the protocol.

- Results:
  - Decision protocol language can generate MATLAB code that plugs into existing continuous reachability toolbox
- Anticipated Results:
  - Certify protocols for robustness to decisions under latency
  - Identify weaknesses in a protocol due to overlap of decision points
  - Incorporate protocol rewriting algorithms based on human factors results
  - Develop algorithms for finding initial conditions to test a protocol with a human in the loop, based on identified weaknesses
- Collaborations: C. Tomlin (UC Berkeley), I. Mitchell (UBC)

Accomplishments to date extend an existing scenario from previous research: refueling an unmanned vehicle that is controlled by a ground station operator.

We have developed an integrated modeling language through the Generic Modeling Environment (GME, from Vanderbilt University) that represents decision protocols for this embedded human system. The protocol explains how the vehicle will react to certain ground station commands, such as commands to transition to the next state, etc. Based on the behaviors in each state, continuous reachability analysis can be performed already. Our tool integrates these individual reachability calculations in a unique way, to compare the current state of the vehicle to the safe—and unsafe—regions of the continuous state space of potential discrete states.

With this formulation, we can now extend this language to analyze existing protocols for weaknesses, using criteria we will determine in future work. Some candidate examples include discovery of the largest latency that this protocol will support, or the amount of overlap of unsafe and safe zones that is permitted for these maneuvers.

A Fully Autonomous Controller

Let's say that the system controller is fully autonomous. Then, the state machine shown would contain *only* state information relevant to the UAV. A scenario might be, "start in state one, and transition to state 2. Wait for a few seconds, then transition to state 3. Wait a few seconds, then transition to state 4…" With the tools we developed so far, there are verification problems that we can address, such as minimum time to complete a maneuver in fully autonomous mode. All events in this model are "State-based Events."

An Embedded Human Controller

If the system requires human intervention to approve messages or react to changing conditions, then the state machine must now contain *human input* information (aka, "Human Operator Events"). The cooperation protocol (shown) encodes such information, by ensuring that to leave certain states, a human must send a message. Complications arise, when more than one message can be sent, or when the same message may apply to more than one state. However, there are still some transitions respond to "State based events" (such as "when you arrive in this area of the state space, wait for further instructions). This hybrid approach permits the embedded human to focus only on the commands that need human intervention, and leave the low-level autonomy up to the autonomous portion of the system.

The Model of the Embedded Human's Behavior

In our future work, there is another state model, which will consider the human behavior more formally. The actions produced by this model translate into the "Human Operator Events" received by the cooperation protocol. (More information shown on slide 8).

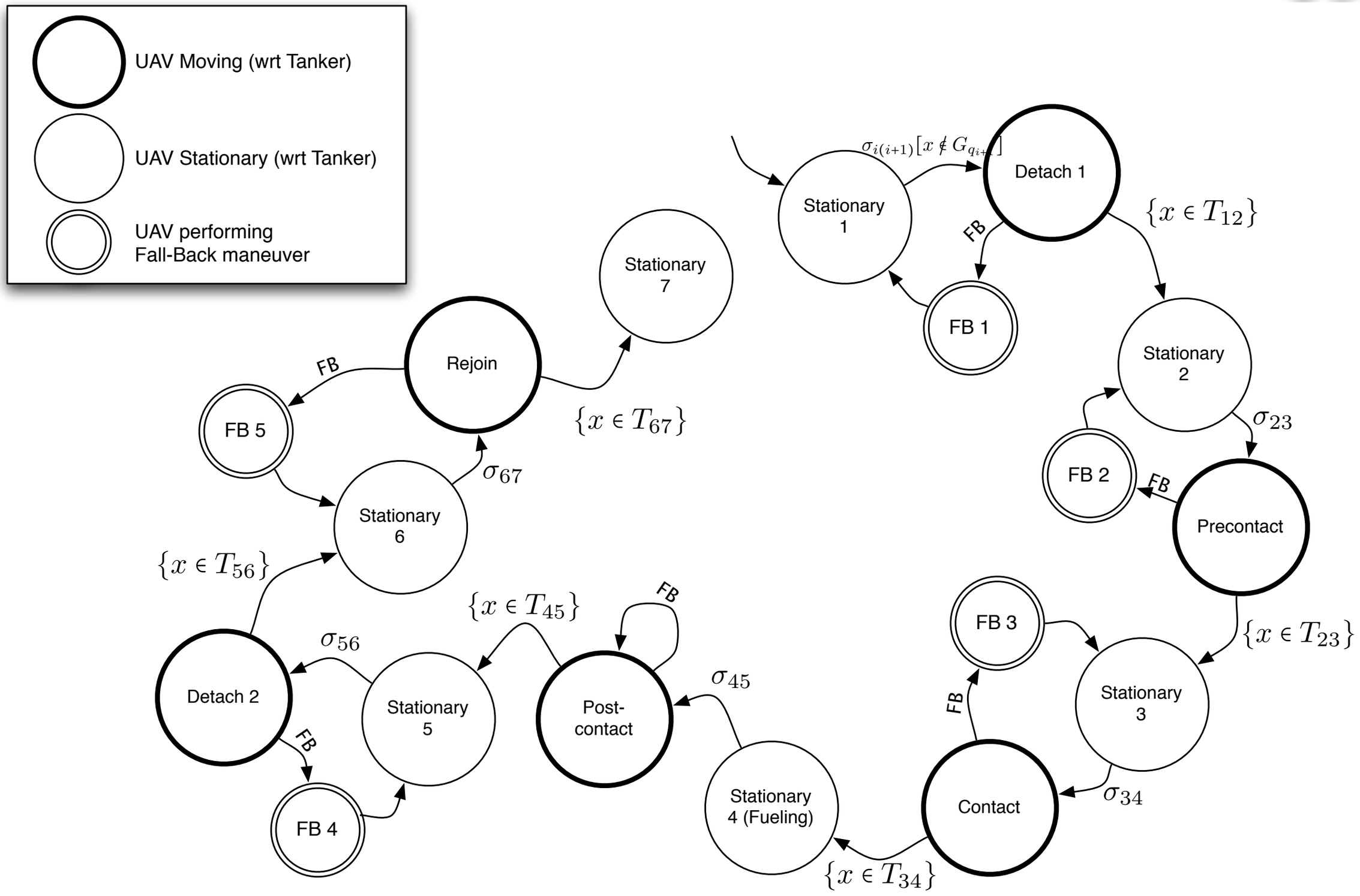
Sample of execution for the embedded human system (this clearly has elements from the work in progress, but we treat it as an expected result):

There are two defined state machines: (1) the cooperation protocol (shown above) and (2) the embedded human's behavior model (the work in progress). In State 1, the UAV will stay in its target neighborhood until receiving an event from the embedded human. The human behavioral model checks to see
 * how many decisions can I make
 * what decision (if any) am I being asked to make?
 * what information is provided for me to observe
 * what timing constraints (if any) exist on my decision

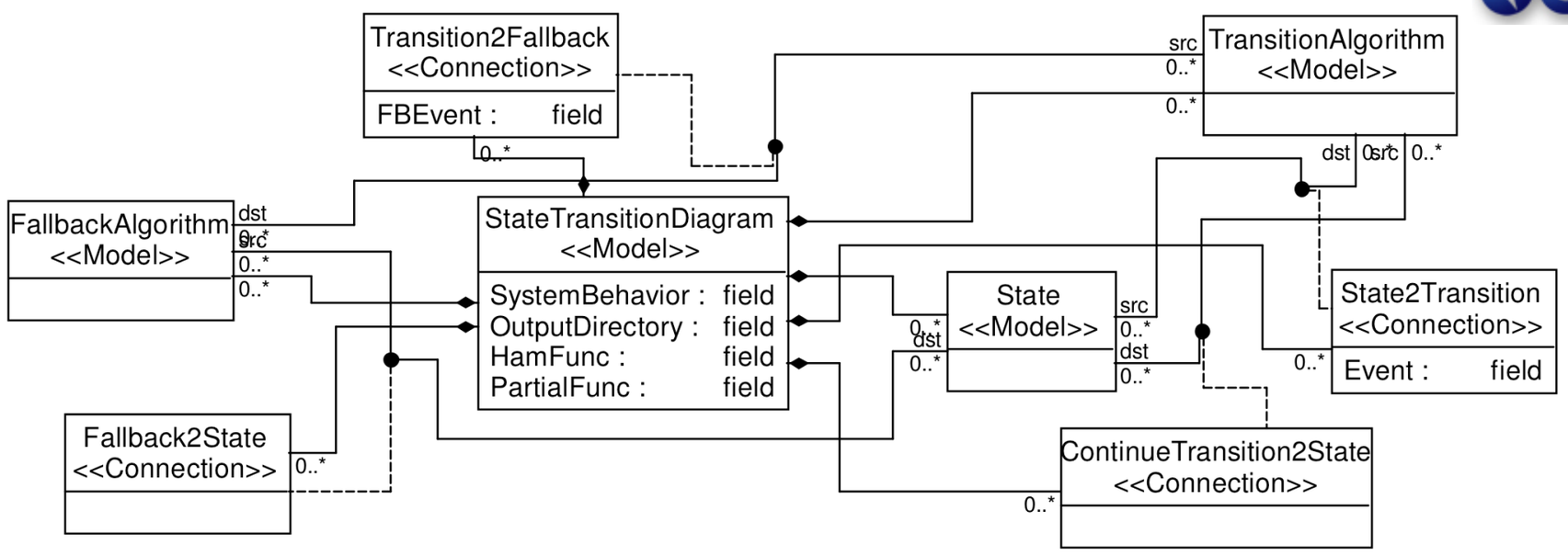
This human behavioral model is actually a function of the cooperation protocol (1). That is, depending on the topology of that protocol, more (or less) decisions may exist at certain times. Moreover, depending on any global constraints that may exist, certain decisions may be removed from the UI presented to the embedded human, and are thus not included in that behavioral model.

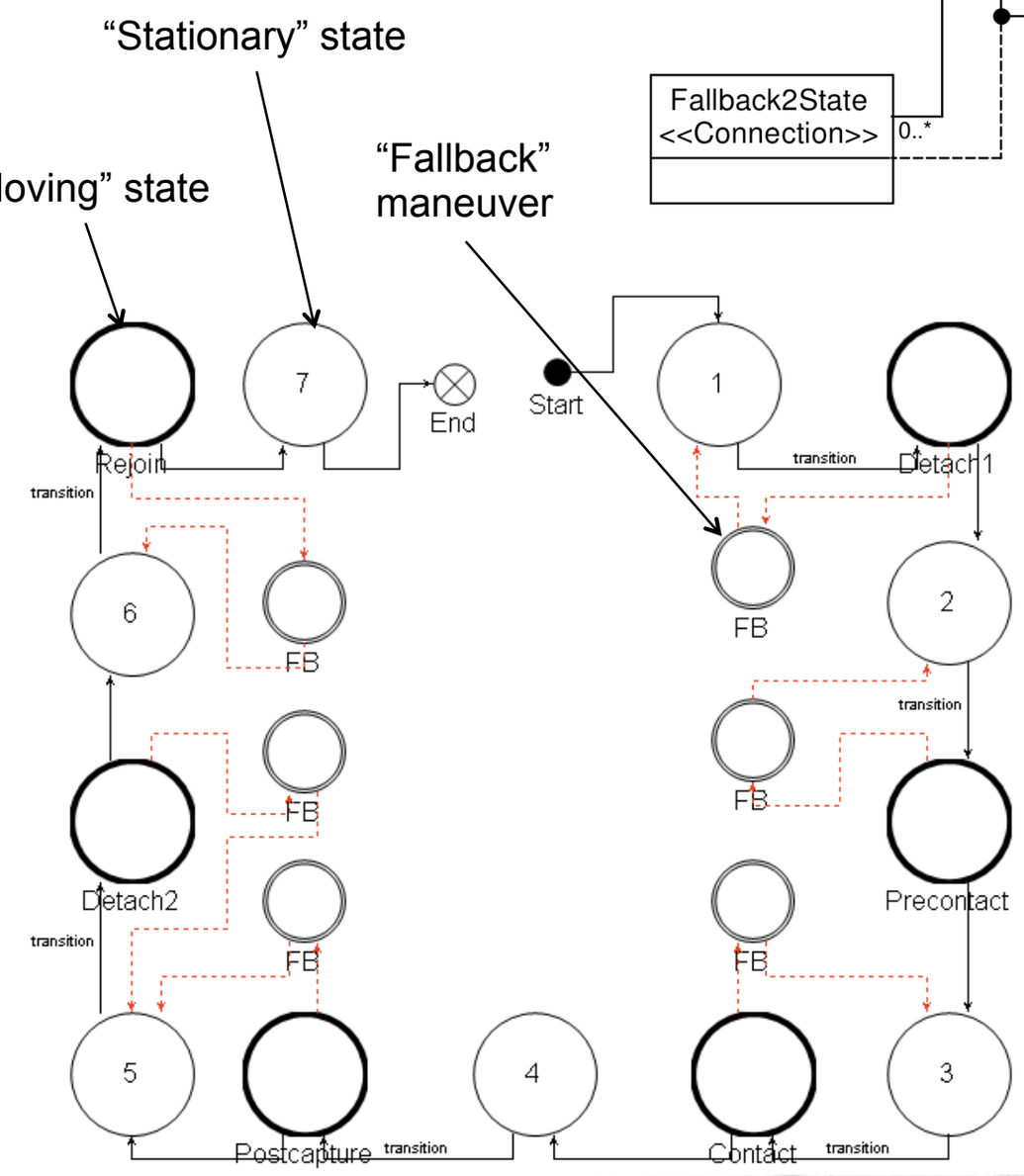
State 1 (being a "stationary" state in the metamodel) can only be exited through a human operator event. The next state (called "Detach 1" in the diagram) can be exited either through receipt of a human operator event (fall back) or by arriving in the target set of State 2. This pattern repeats itself, until the end of the protocol (as currently designed).

Now, we can apply our work from Year 1 that looks at the safety in certain zones of the statespace of certain decisions, from the perspective of the time it takes to make decisions. Protocols like the one we designed (where the human operator has essentially limitless time to make a decision from each stationary state) will be robust to long decision making times. However, there will be some occasions where the time to make the decision (combined with the latency of the network) may be near the time that was used to check the safety of decision making. We expect to examine protocols from these perspectives generically, by permitting users to design their own protocols, and then perform some analysis similar to what we describe before.


UAV Moving (wrt Tanker)
UAV Stationary (wrt Tanker)
UAV performing Fall-Back maneuver
Stationary 1
$\sigma_{i(i+1)}[x \notin G_{q_{i+1}}]$
Detach 1
FB
FB 1
$\{x \in T_{12}\}$
Stationary 2
σ_{23}
FB 2
FB
Precontact
$\{x \in T_{23}\}$
FB 3
FB
Stationary 3
σ_{34}
Contact
$\{x \in T_{34}\}$
Stationary 4 (Fueling)
σ_{45}
Post-contact
FB
$\{x \in T_{45}\}$
Stationary 5
σ_{56}
Detach 2
FB
FB 4
$\{x \in T_{56}\}$
Stationary 6
σ_{67}
Rejoin
FB
FB 5
$\{x \in T_{67}\}$
Stationary 7

Language constraints enforce a structured protocol design



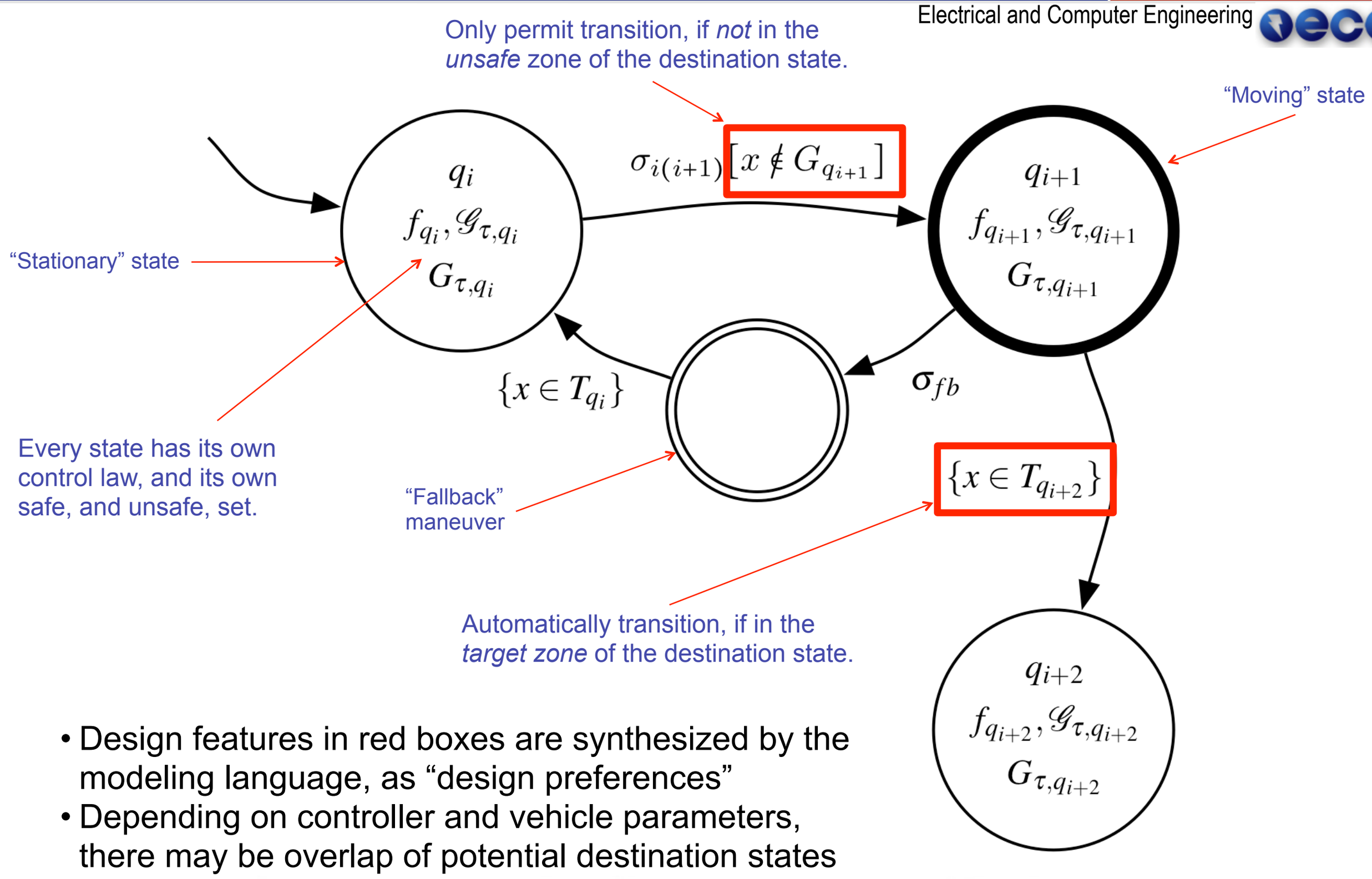


- The metamodel (above) is used to generate a language for encoding transitions between discrete states
- "Moving" states must be next to "stationary" states, to permit embedded human control under latency
- Moving states have "Fallback" maneuvers, to transition to safety if commanded by the human

Significant work has been performed to encode the decision protocol of an embedded human in a formal language that can generate design artifacts.

The toolchain flow is: (1) development of a decision protocol metamodel, to generate the decision language, (2) construction of existing, known protocols in this language, (3) synthesis of MATLAB code that plugs into an existing continuous reachability tool, namely Ian Mitchell's Toolbox of Level Set Methods.

Important features of the current language for cooperative maneuvers include:
* individual control preferences for each state
* states where relative motion is '0' (stationary) must be "between" states with relative motion
* structure of language permits the synthesis of:
** configurations for the level set toolbox
** automated simulations of the entire maneuver
** generation of simulator that takes user input (user can press buttons for events between states)

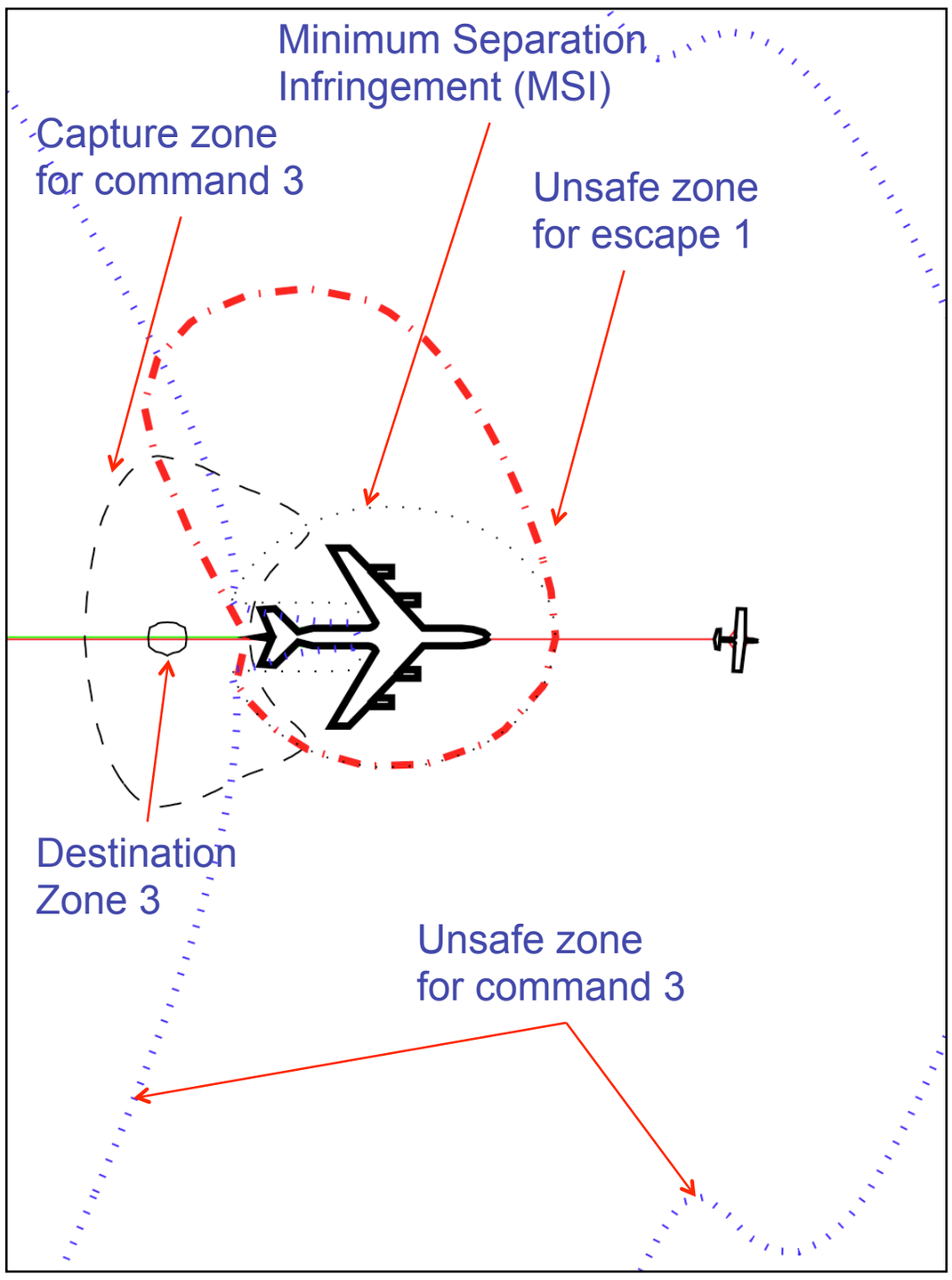


- Design features in red boxes are synthesized by the modeling language, as "design preferences"
- Depending on controller and vehicle parameters, there may be overlap of potential destination states
- Alternate design preferences are underway in current effort

This graph shows an arbitrary maneuver step. The language encodes all the \sigma_{i+1} events, which are commands from the operator to initiate the transition to the next state.

The red boxes in the graph are highlighted to show that a global design preference can add more information to the graph. For example, "do not enable this transition if we are in a KNOWN UNSAFE set of the next state." Since the language constraints us to be in a "stationary" state (i.e., relative motion of 0 between the vehicles) before we begin to move, we are always safe if we stay in that state, and the other vehicle stays within its assumed operational envelope.

Another design preference might be, "do not enable this transition if we are not within the KNOWN SAFE set of the next state". We are exploring these alternate design preferences, and global preferences for them, as our ongoing work.



This plot generated through work in collaboration with
J. Ding, C. Tomlin, and S. Sastry of UC Berkeley

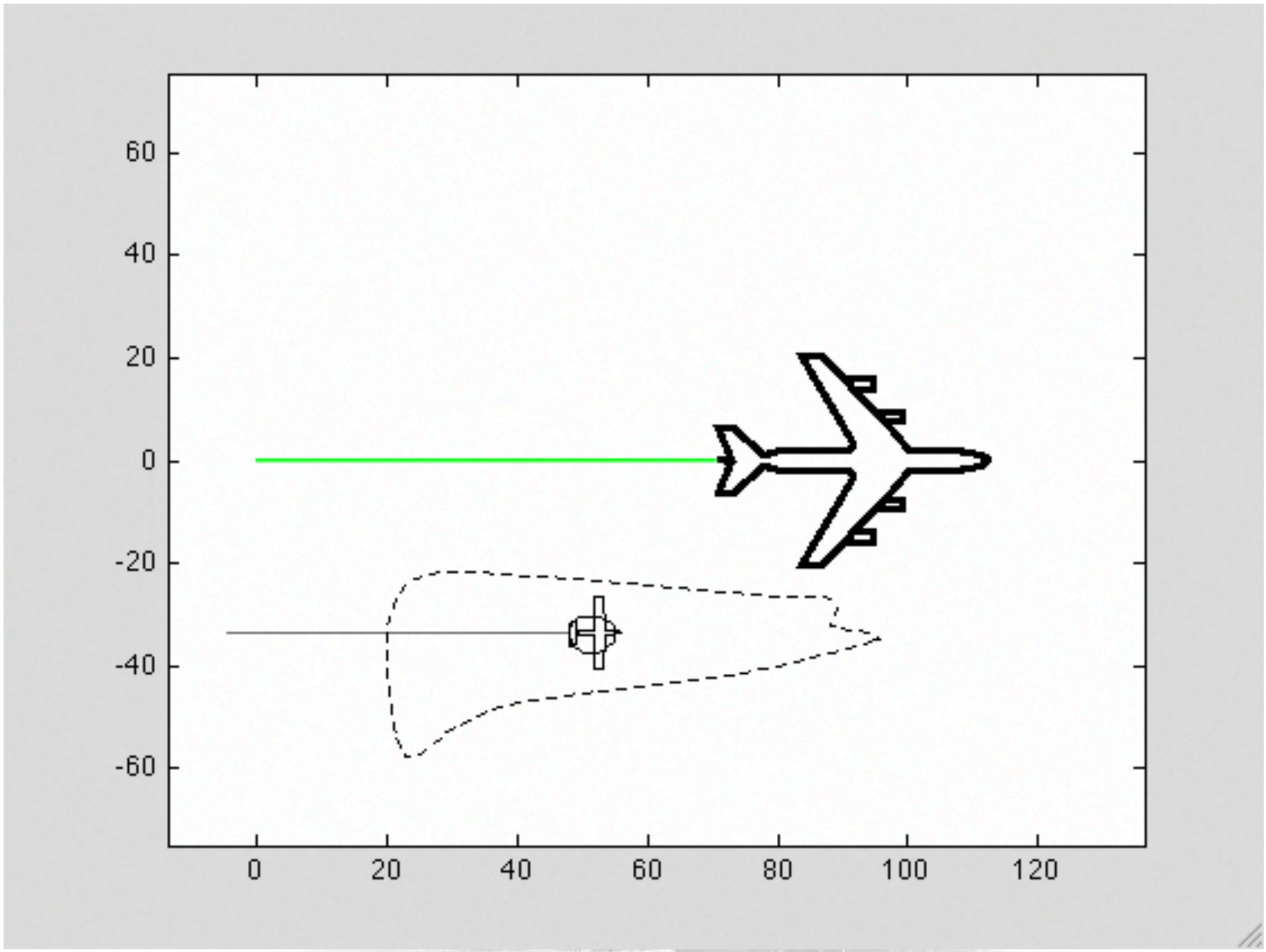
- Consider the scenario at left, where the UAV is trying to get to Zone 3
- Available commands are Command 3, and Escape 1
- Executing Command 3 would violate the MSI
- Execution of Escape 1, and then Command 3, is appropriate
- However, note that “Unsafe for Command 3” and “Capture for Zone 3” are not mutually exclusive: *this is a weakness in the existing design, and should be addressed!!!*
- We are working on automation for detecting weaknesses such as these

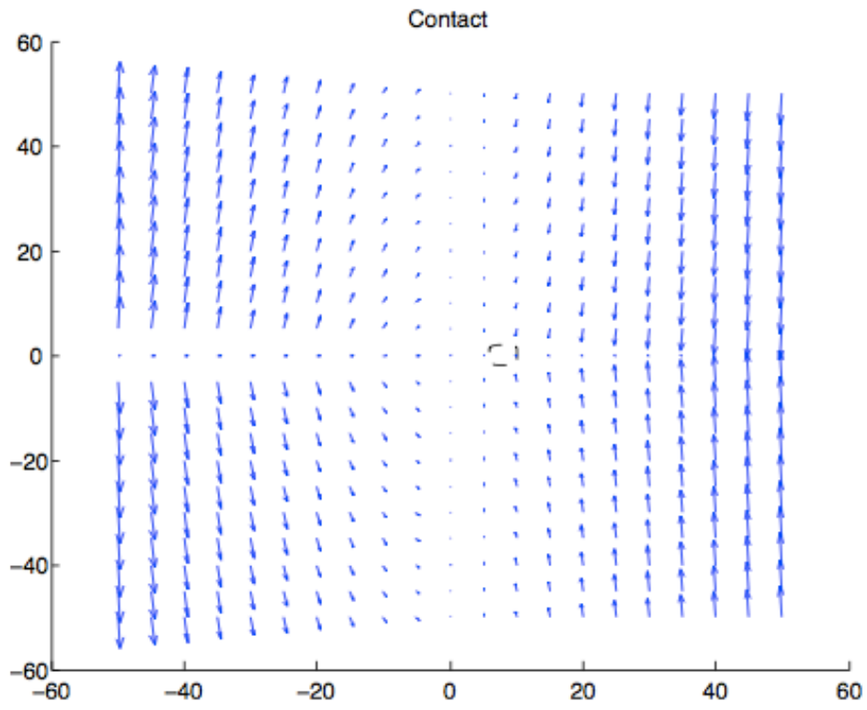
This represents a case where a UAV is issued a direct command, which is “Go to Destination Zone 3”, while out of position. This might be the result of a long-delayed message, or operator error. The problem is that if the UAV initiates this maneuver, then it will violate the Minimum Separation Infringement (MSI) boundary around the tanker.

This situation is the kind of situation we are exploring: what happens when strange messages are received, or when a received message will result in known violations of safety constraints. We examine this particular scenario because it is an extreme example.

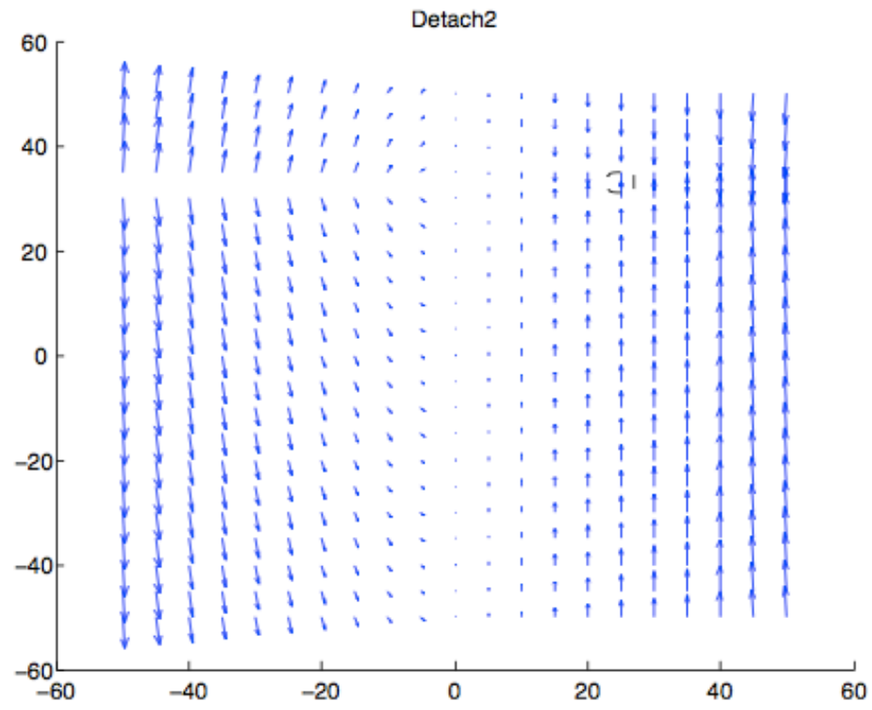
There are a few important things to note:
1) The unsafe zone for command 3 (the zone in which we MIGHT intersect with the MSI region in bounded time, under a pathological behavior by the tanker) is very large.
2) The unsafe zone for escape 1 (the zone in which we WILL intersect with the MSI region in bounded time if we issue that command) reflects that control law’s behavior of “TURN RIGHT, NOW”
3) The capture zone for command 3 (the zone in which we WILL converge on our destination zone 3 in bounded time time) INTERSECTS with both the previous zones, but it is not a subset
4) The available behaviors for the UAV are
   (a) sit still
   (b) escape 1
   (c) command 3

After offline analysis by a human expert, it is clear that this zone overlap is a weakness in the design. Our goal is to automate this discovery, and perhaps generate new control options that mitigate this design weakness, or recommend to the designer options such as set subtraction (where the overlap of sets removes any chance of ambiguity). Alternatives include generating a simulation where the initial conditions are IN THE INTERSECTION OF THESE SETS in order to attempt to discover whether embedded human operators appropriately determine that certain commands are unsafe. This ongoing work will generate algorithms, and interfaces for the integration of human factors experts in other efforts.

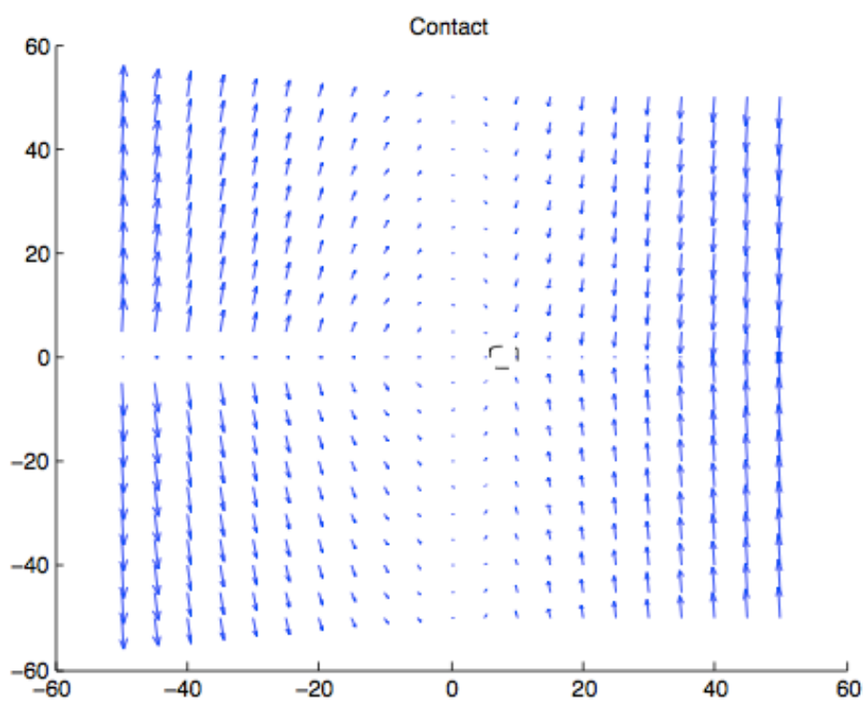

60
40
20
0
-20
-40
-60
0
20
40
60
80
100
120



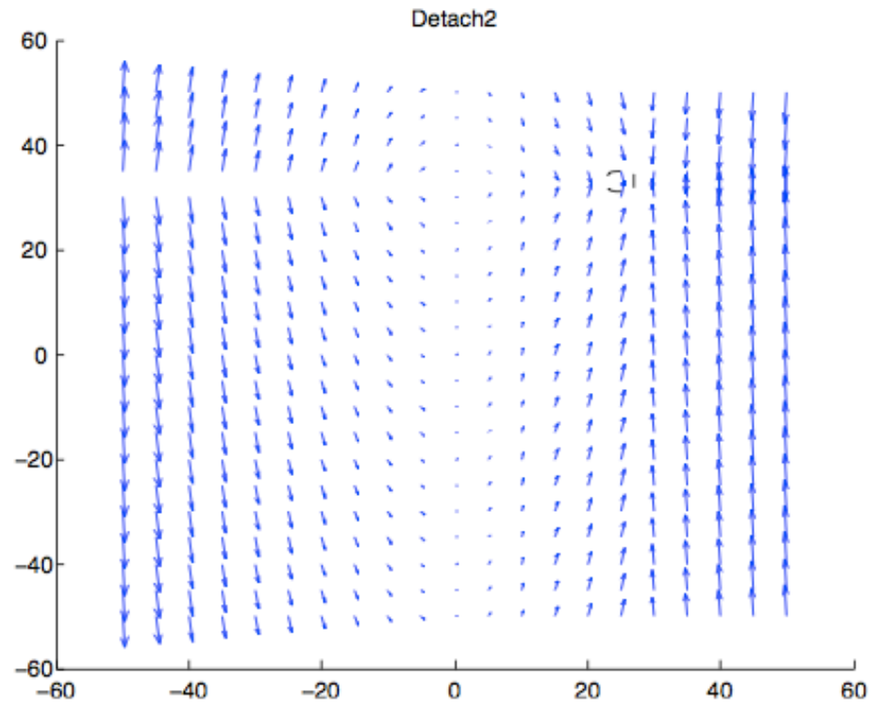
(a) Proportional Controller, Contact Mode



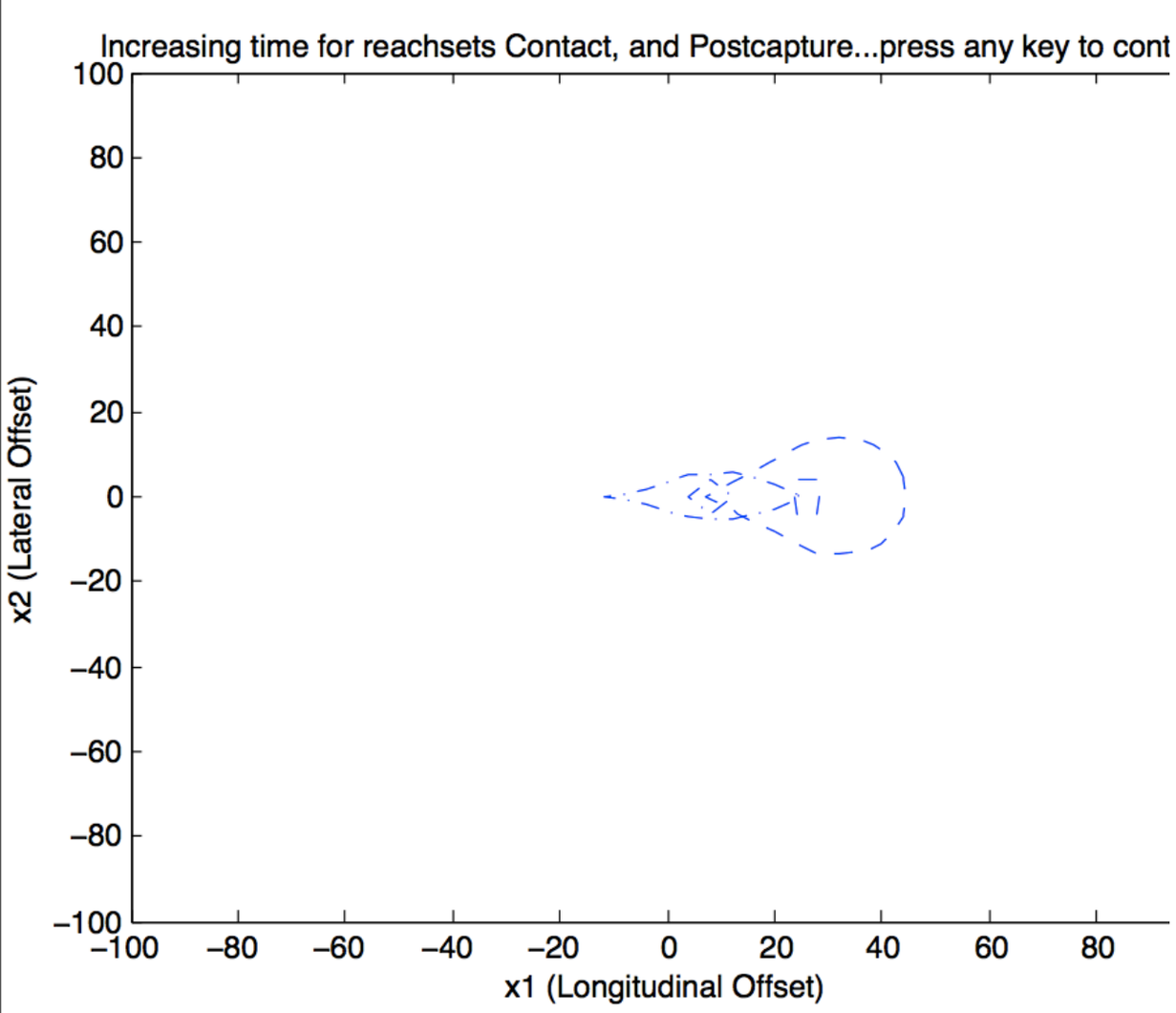
(b) Proportional Controller, Detach 2 Mode



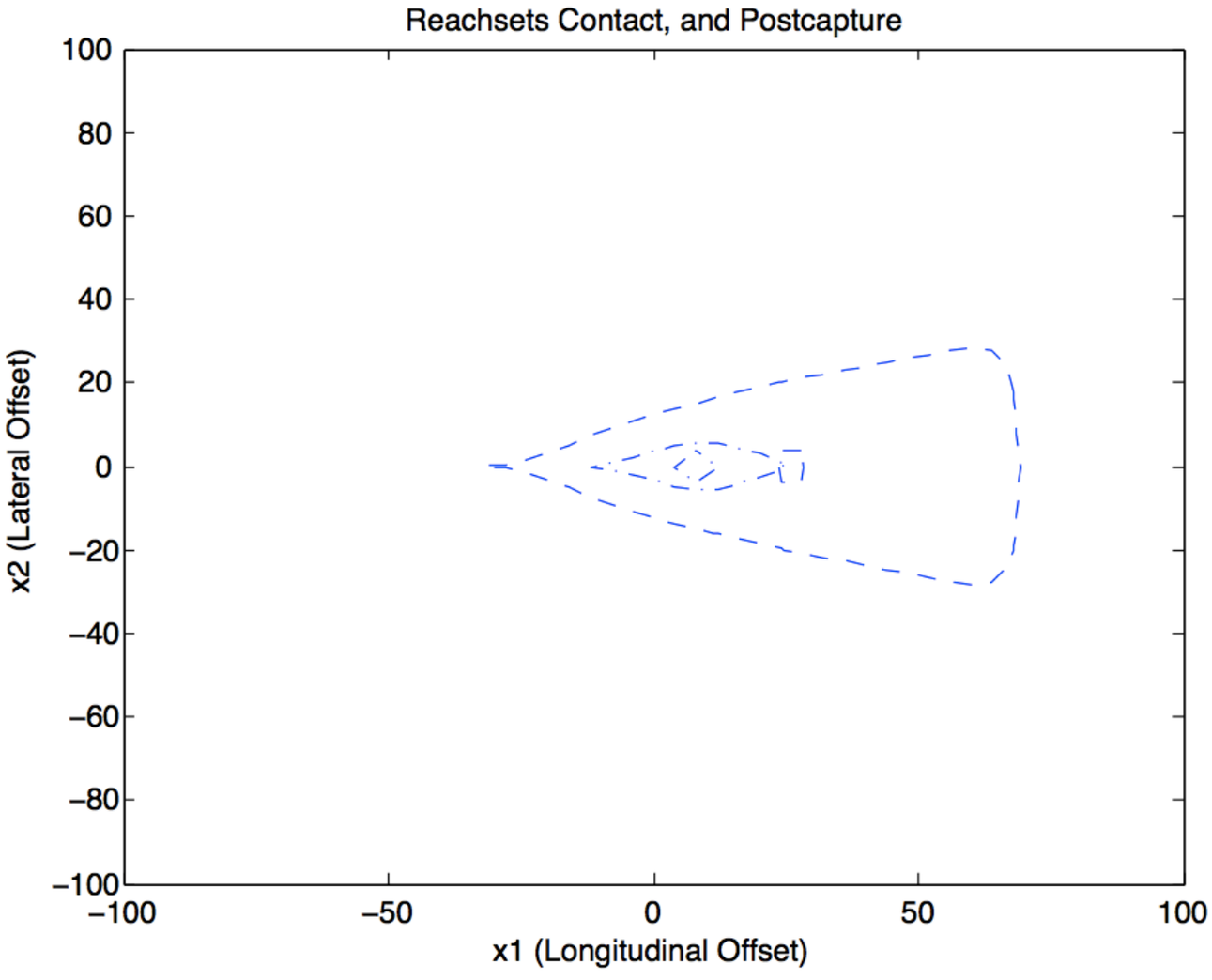
(c) Bang-Bang, Contact Mode



(d) Bang-Bang, Detach 2 Mode

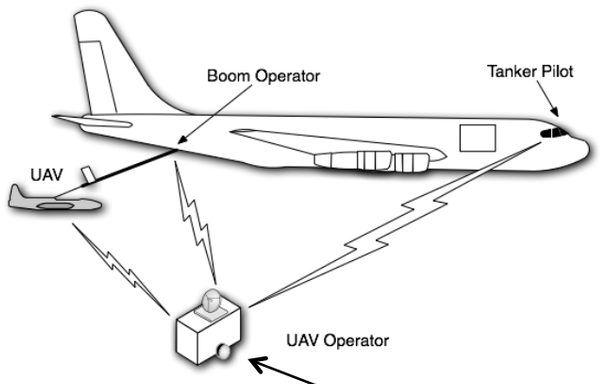


(a) Heuristically generated τ

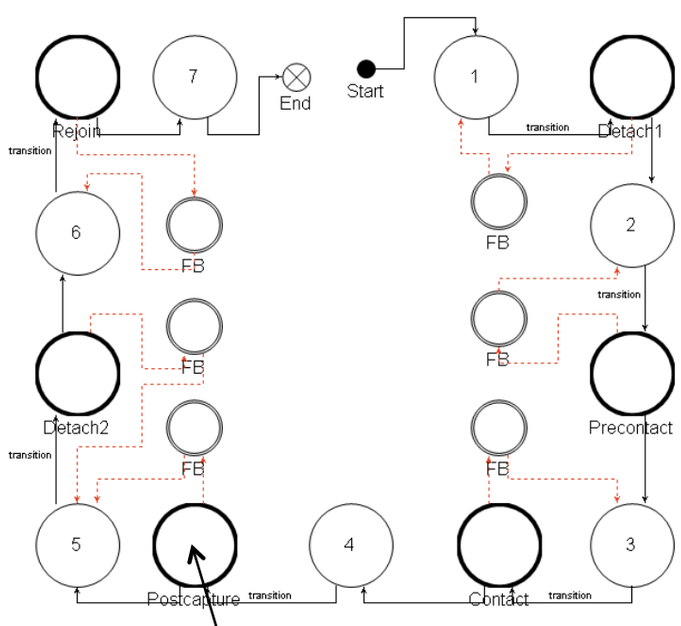


(b) Time τ calculation is automated.

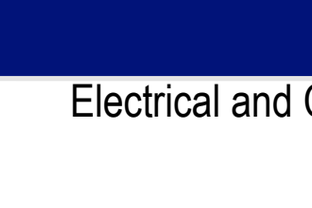
In future years on this effort, validation techniques will also look at the time it takes for humans to make decisions. This will use known results from human behavior modeling, and also estimate what human behavior models will be, based on the structure of the cooperation protocol.

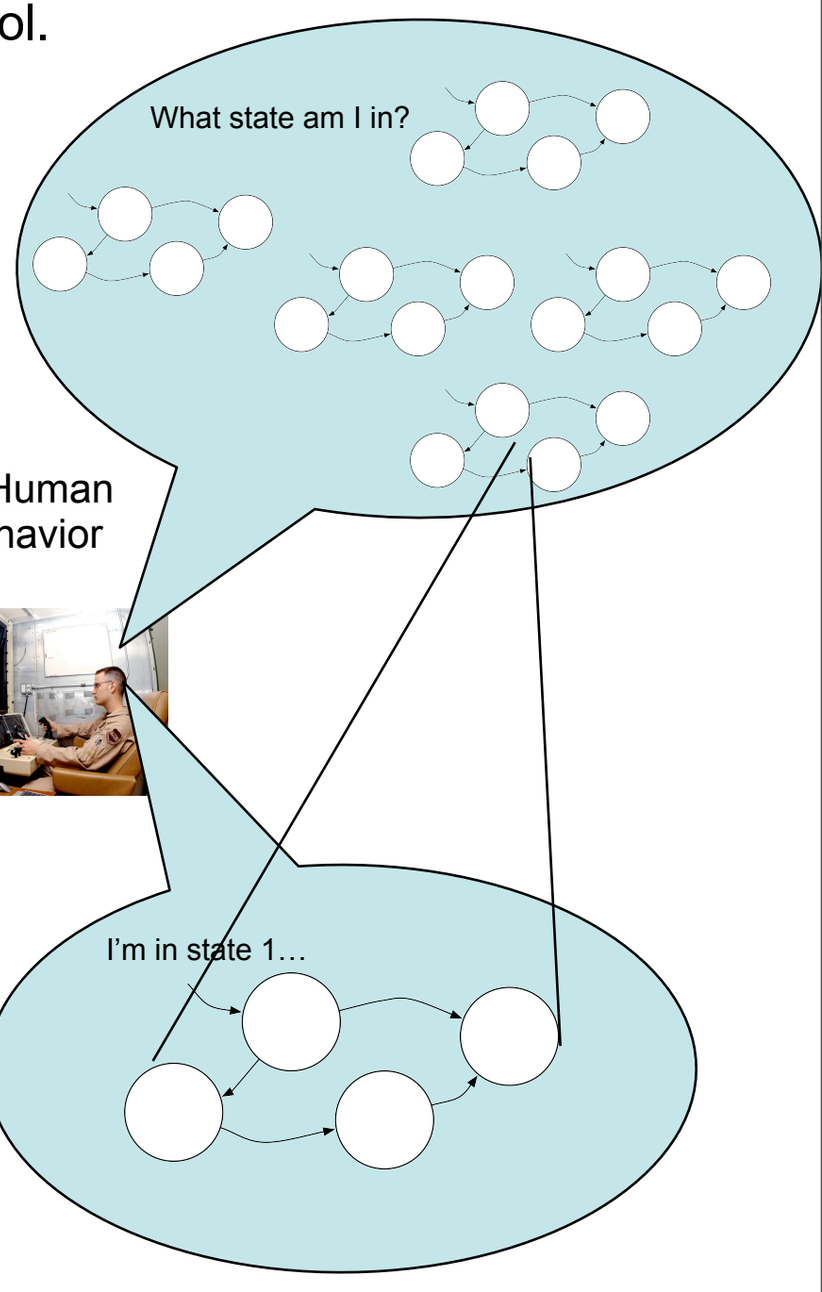


Cooperation protocol



Embedded Human Model of Behavior





"Events" in the cooperation protocol are "actions" from the human model.

Time-delay for a human to produce actions will impact the safety and effectiveness of the protocol design.

Depending on the topology of the cooperation protocol, the time-delay might be greater for the human, in terms of deciding correctly.

If a state has several different potential actions by a human, then the human behavior model will be different when responding to that state, than in other states.

Monday, April 12, 2010 13

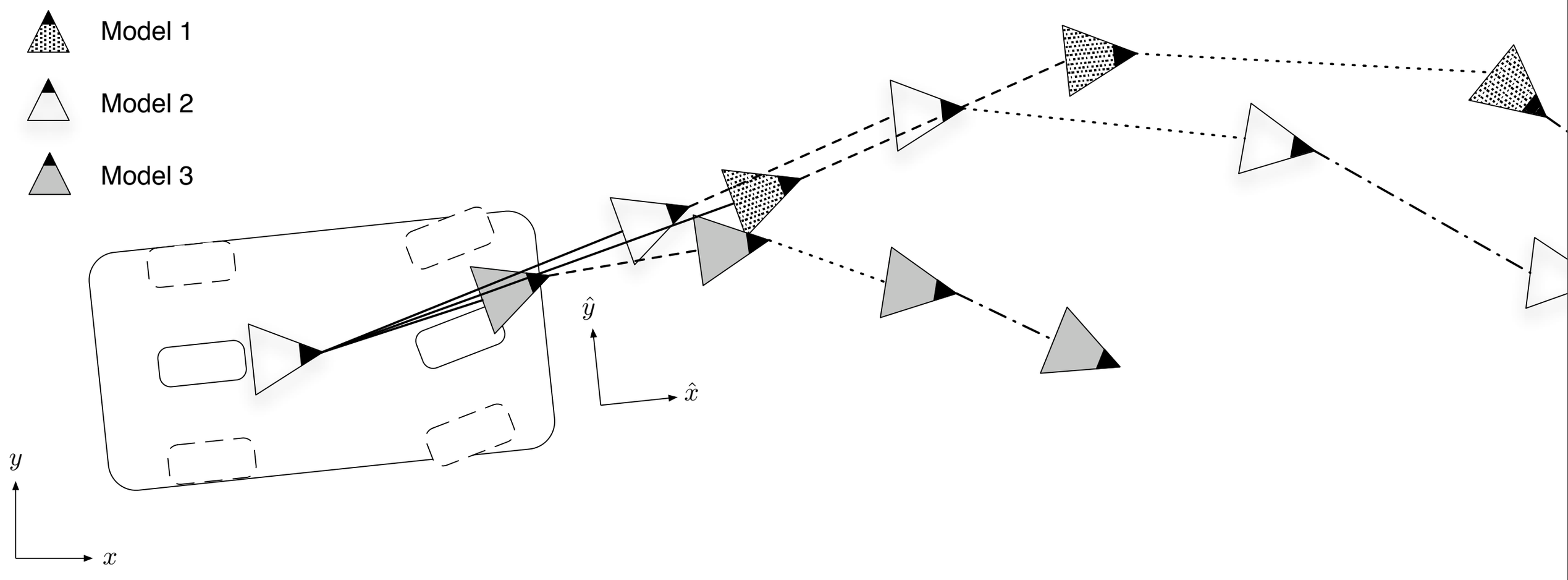
Using a separate state machine (from the cooperation protocol), we are working on expressing the human behavior as well. We are not yet decided whether we should integrate cognitive modeling tools such as ACT-R in this effort, or whether we should simply define human decisions as taking time, and being a function of the number of decisions that must be made, or data that must be processed to make a decision (we may choose to have our own model first, and later interface with more sophisticated tools). That second state model (aka, the human behavior model) will then produce the events that permit the UAV to transition to different modes.

Note that the image used for the Embedded Human Behavior Model in this slide is a repeat from the code synthesis slide of this deck; here, we use it to represent a state model of the embedded human's (not the UAV's) behavior.

These human behavior models may be a function of how the cooperation protocol itself is designed. For example, if there are 20 events that can be sent, the human behavior protocol will be more complicated, as it should take into account time to find the right button, press that button, etc. Having this model be generated from the cooperation protocol is a goal, probably in our final year.

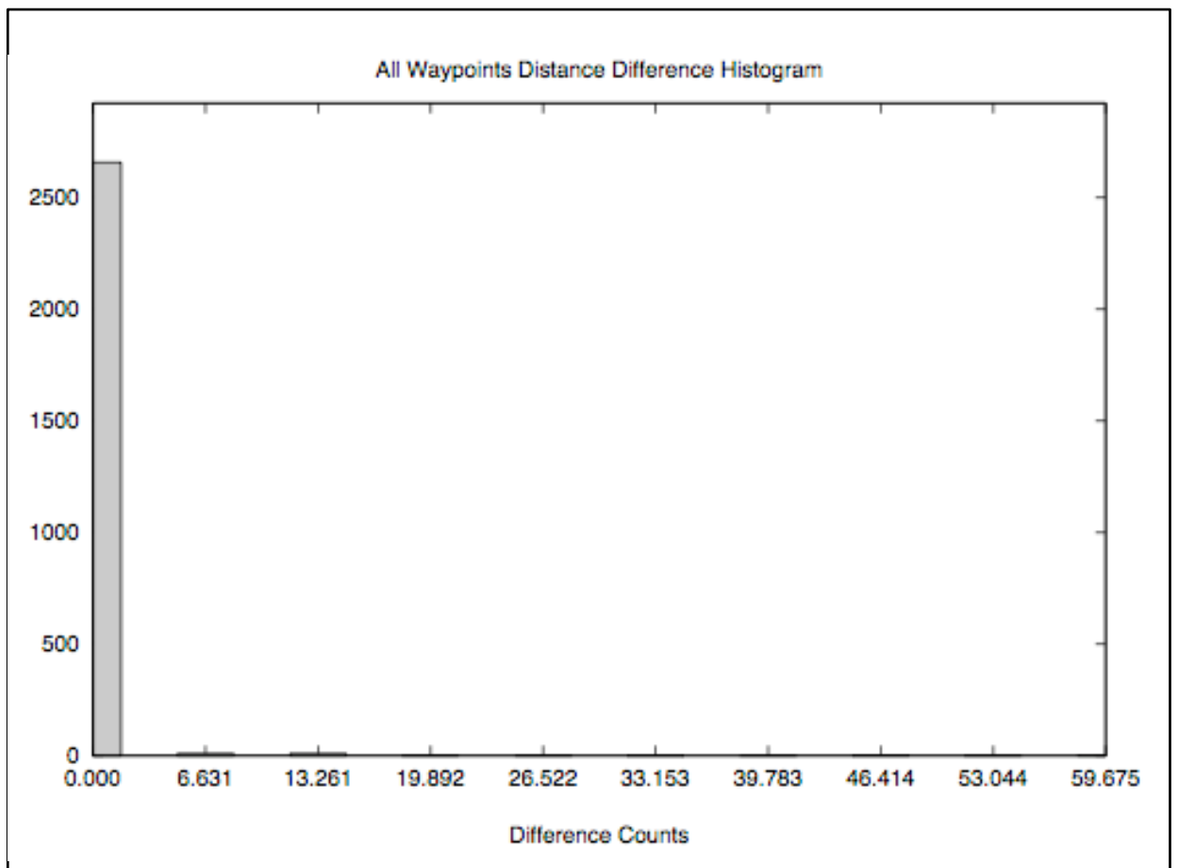
- The following slides are joint work with Brandon Eames of Utah State University/Sandia NL
- For copies of the paper under review, please email me

- Model 1: Rigid body dynamics
- Model 2: Kinematic bicycle model
- Model 3: Ackermann four-wheeled model

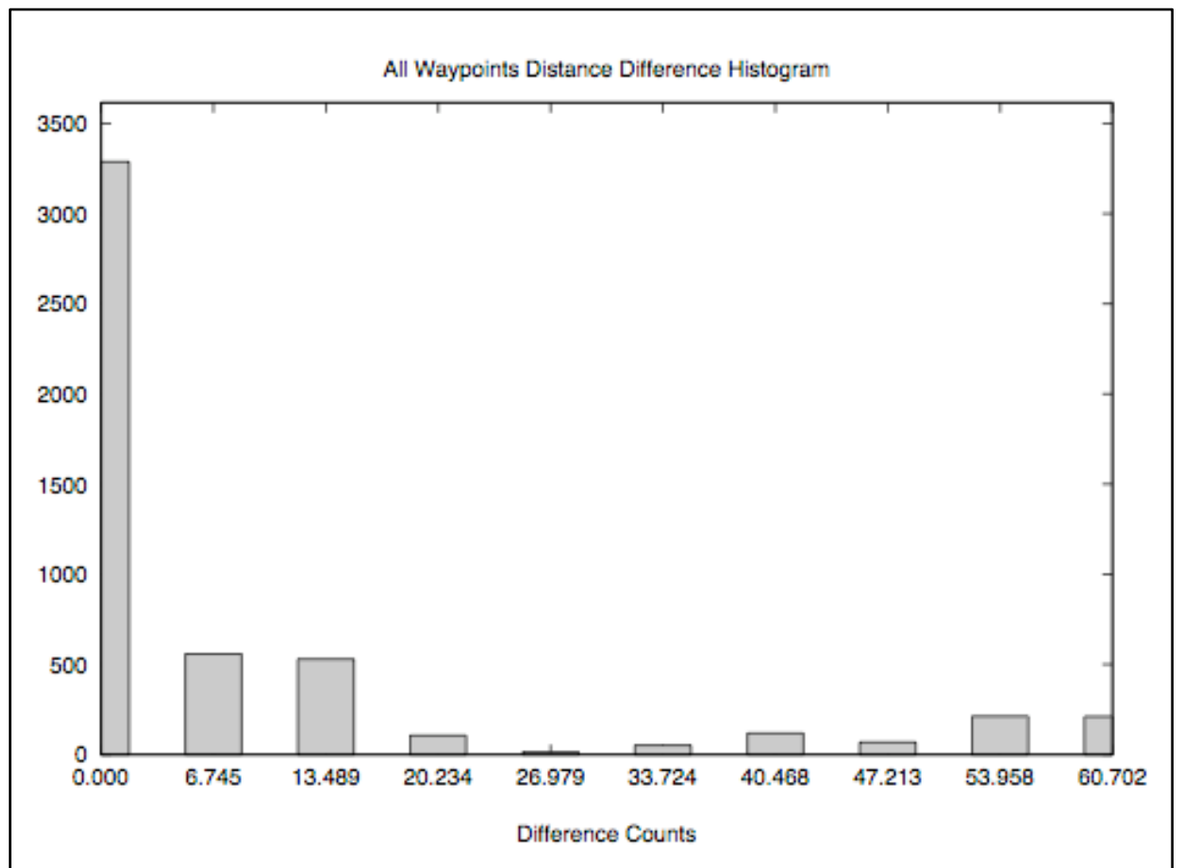


* Sample points not necessarily representative of models; points are exaggerated to show deviations in behavior.

- The goal of our experiment is to determine the robustness of a somewhat fragile cooperation between two data-driven components, capable of running on the same core, two different cores, or two different machines, when those components are deployed on various machines with various single or multicore processors.

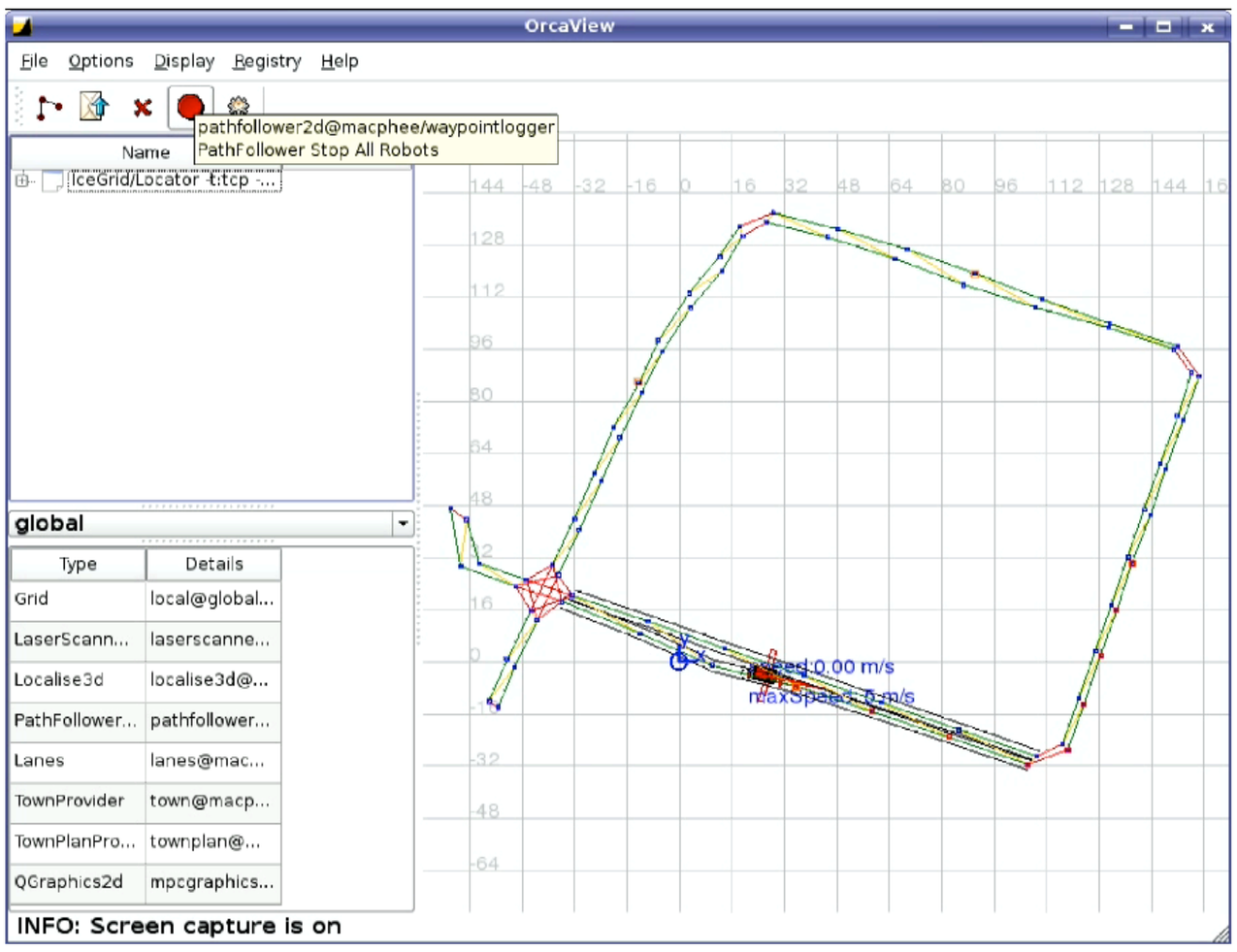


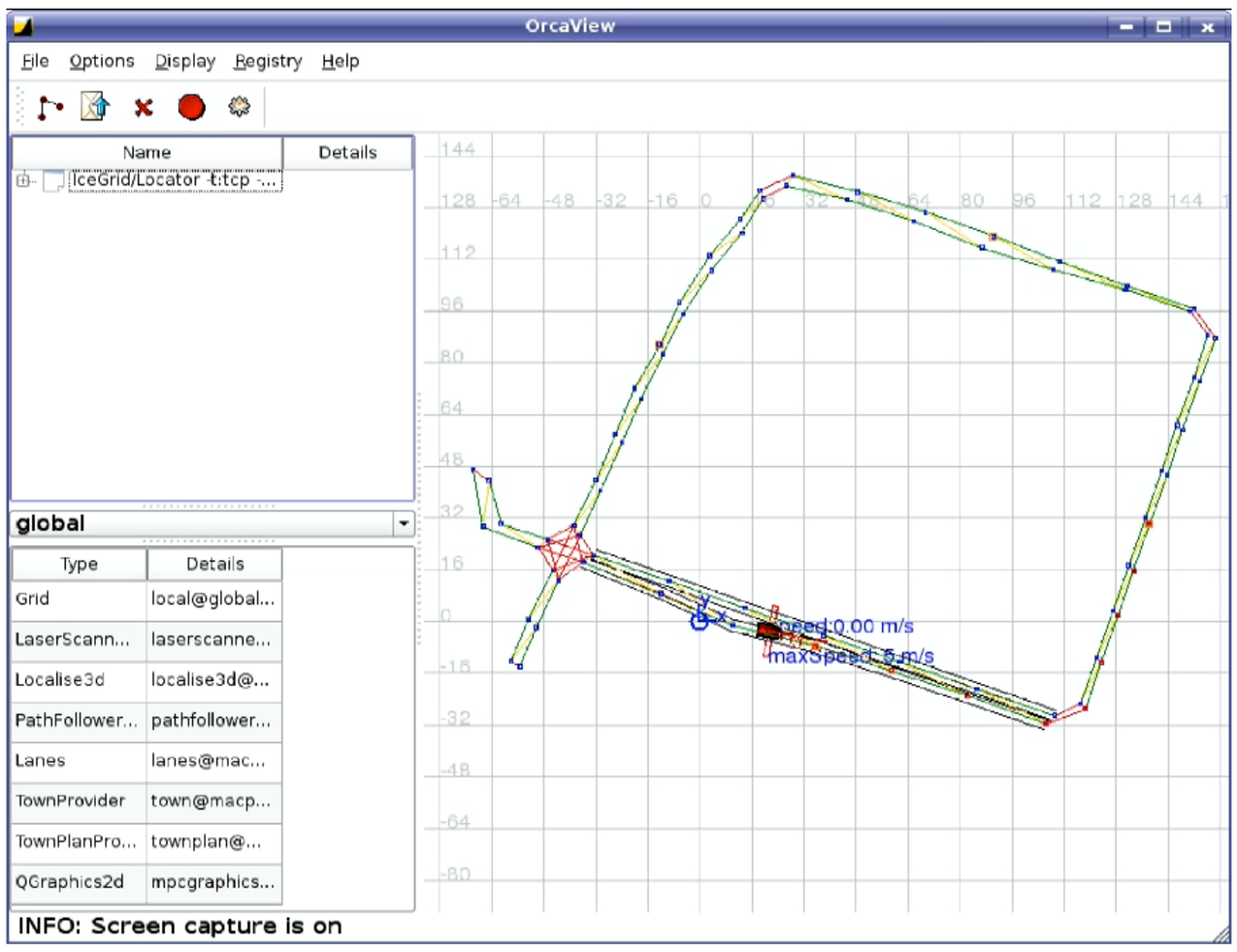
Dimble (core2 duo)



Be3 (quad core)

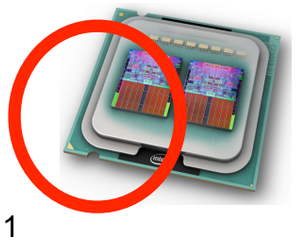
What ways are there to see fragility?



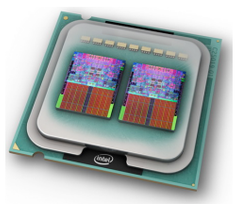


This discrete behavioral difference is akin to a condition number, that when accumulated over many different instances (from the same initial conditions) will indicate whether new hardware amplifies this fragility.

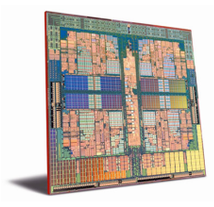
Machine Name	Processor Type	Processor Details	Operating System
macphee	Single Core	Allocated as one processor through VMWare. Hardware is Intel Core 2 Duo, 2.4 MHz	Linux Kubuntu 6.10 (Kernel 2.6.17-10-386), (VMWare running on Mac OS X Host)
dimble	Dual Core	Intel Core2 Duo CPU, E4500, 2.20GHz	Linux Kubuntu 6.10 (Kernel 2.6.17-10-386)
beames-desktop	Quad Core	Intel Q6600	Linux Kubuntu 6.10 (Kernel 2.6.17-10-386)
feverstone	Dual Processor, Quad Core	Intel Xeon CPU, X5460, 3.16GHz, 6MB Cache	Linux Kubuntu 6.10 (Kernel 2.6.17-10-386)
hardcastle	Dual Processor, Quad Core	Intel Xeon CPU L5420, 2.50GHz, 6MB Cache	Linux Kubuntu 6.10 (Kernel 2.6.17-10-386)



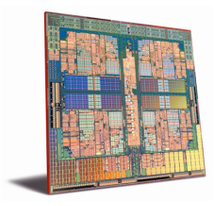
1

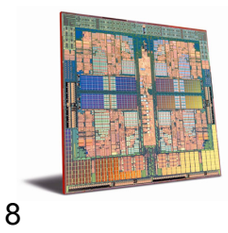


2



4

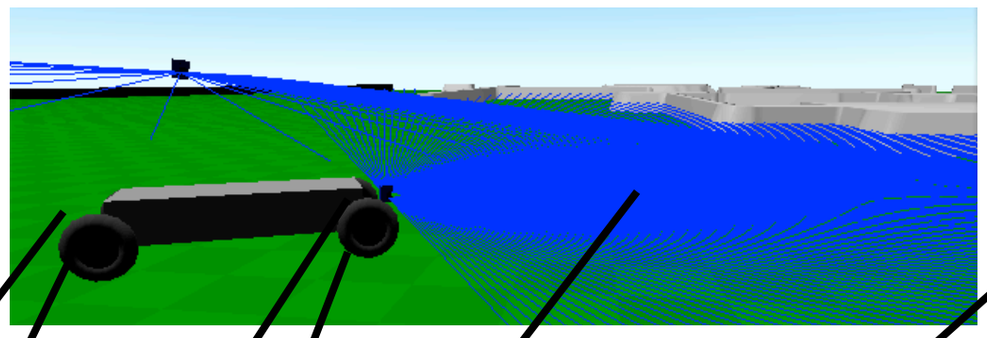


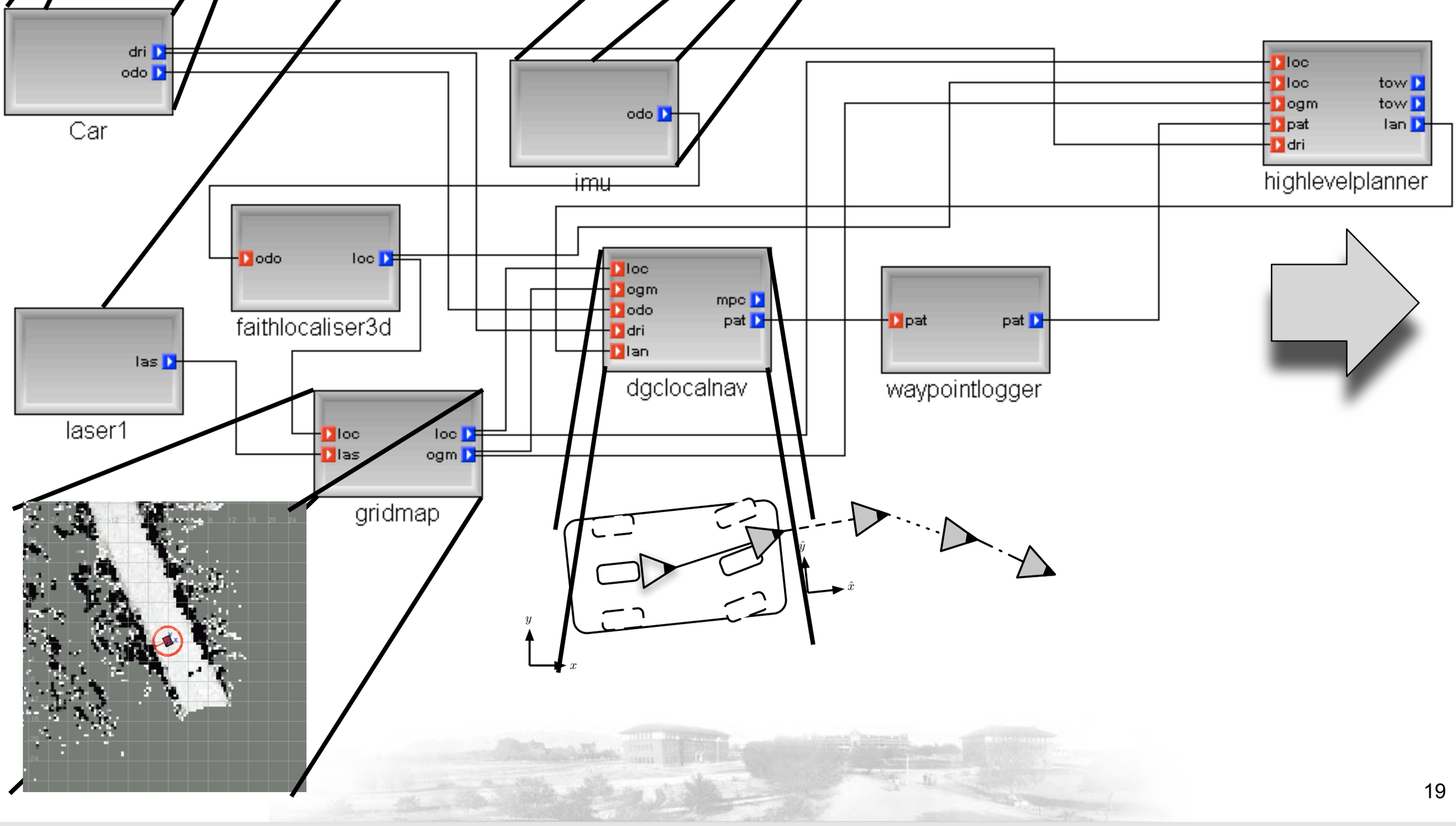


8

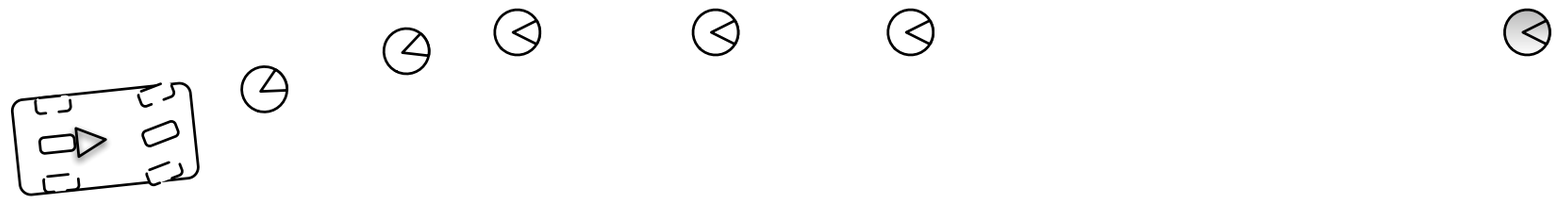
Basic simulation

Model of Computation:
Process Networks (PN)

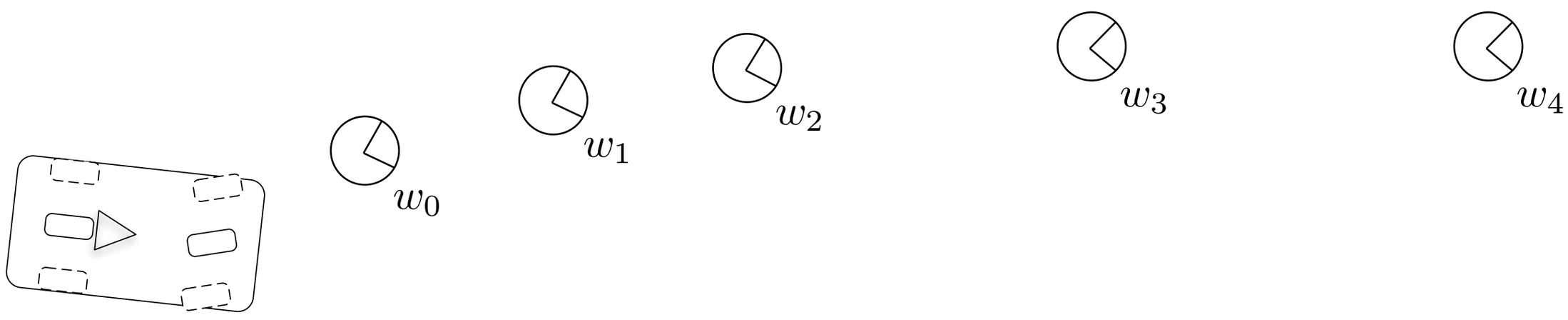




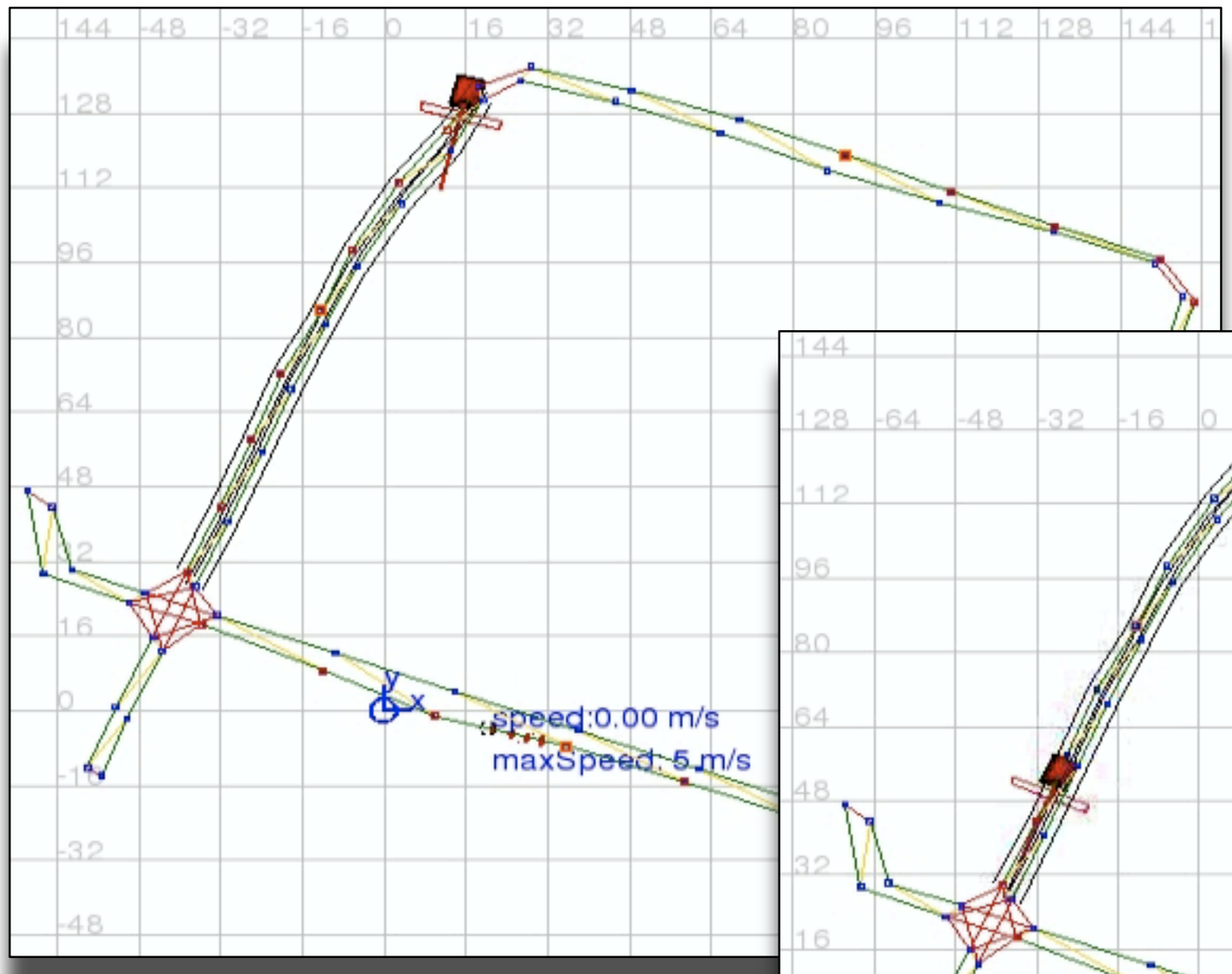
- Produces a series of (intermediate) waypoints, to guide the vehicle to some location, and to be in a particular heading at that location



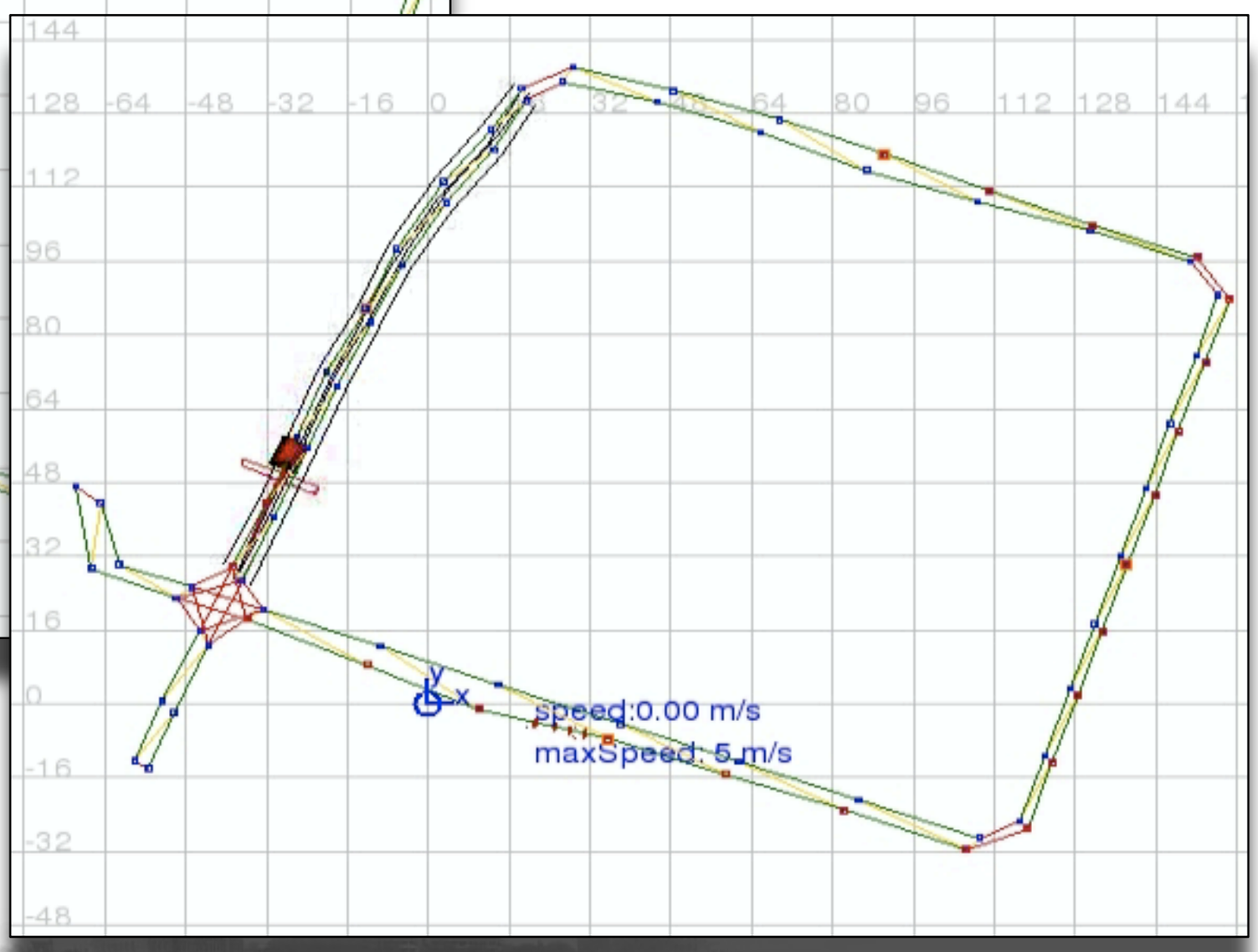
- Waypoints are not tracked, so *each time* highlevelplanner runs, it produces new waypoints for the vehicle
- Waypoints are calculated *backward* from the 'next' destination



Failure

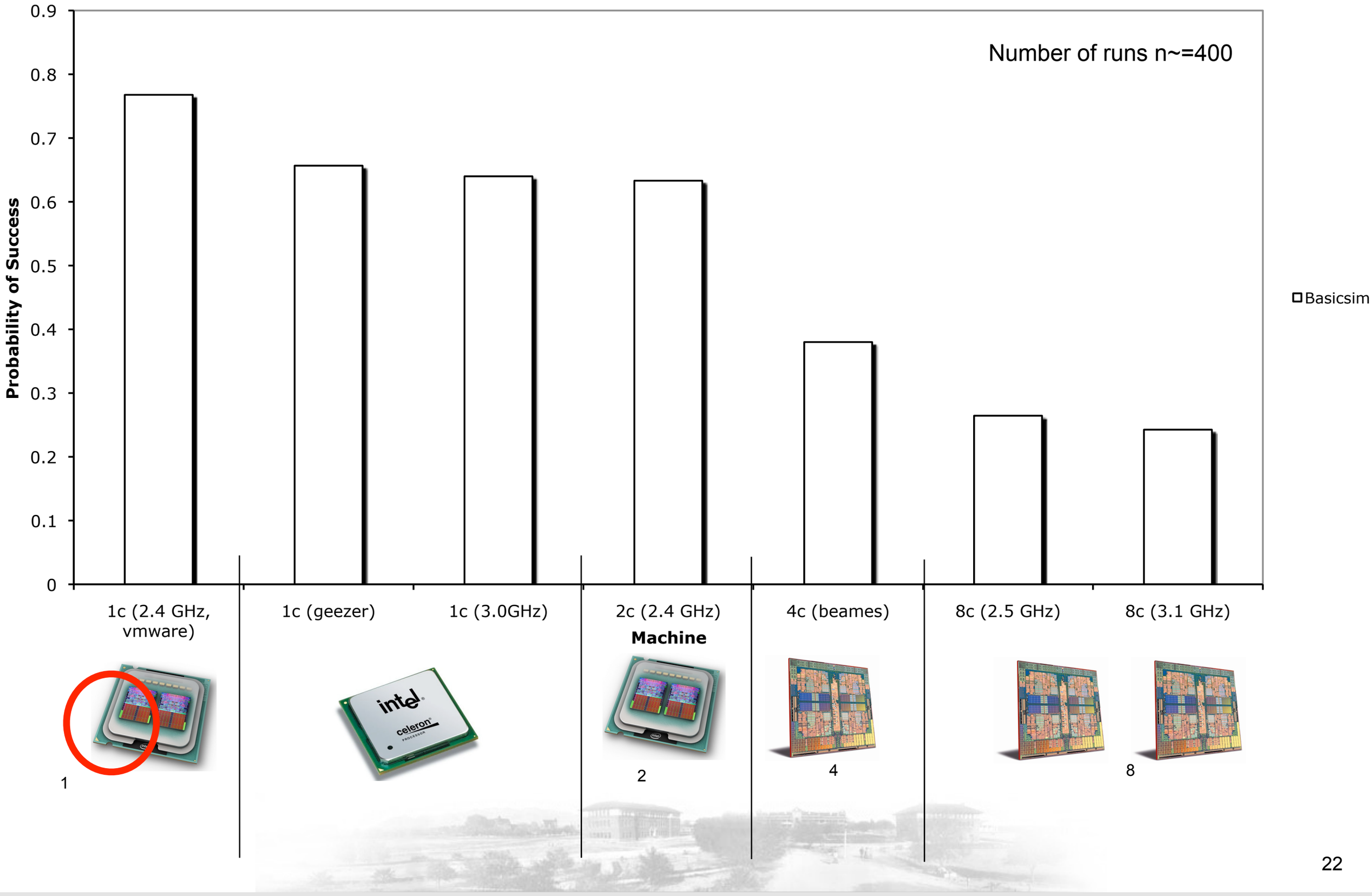


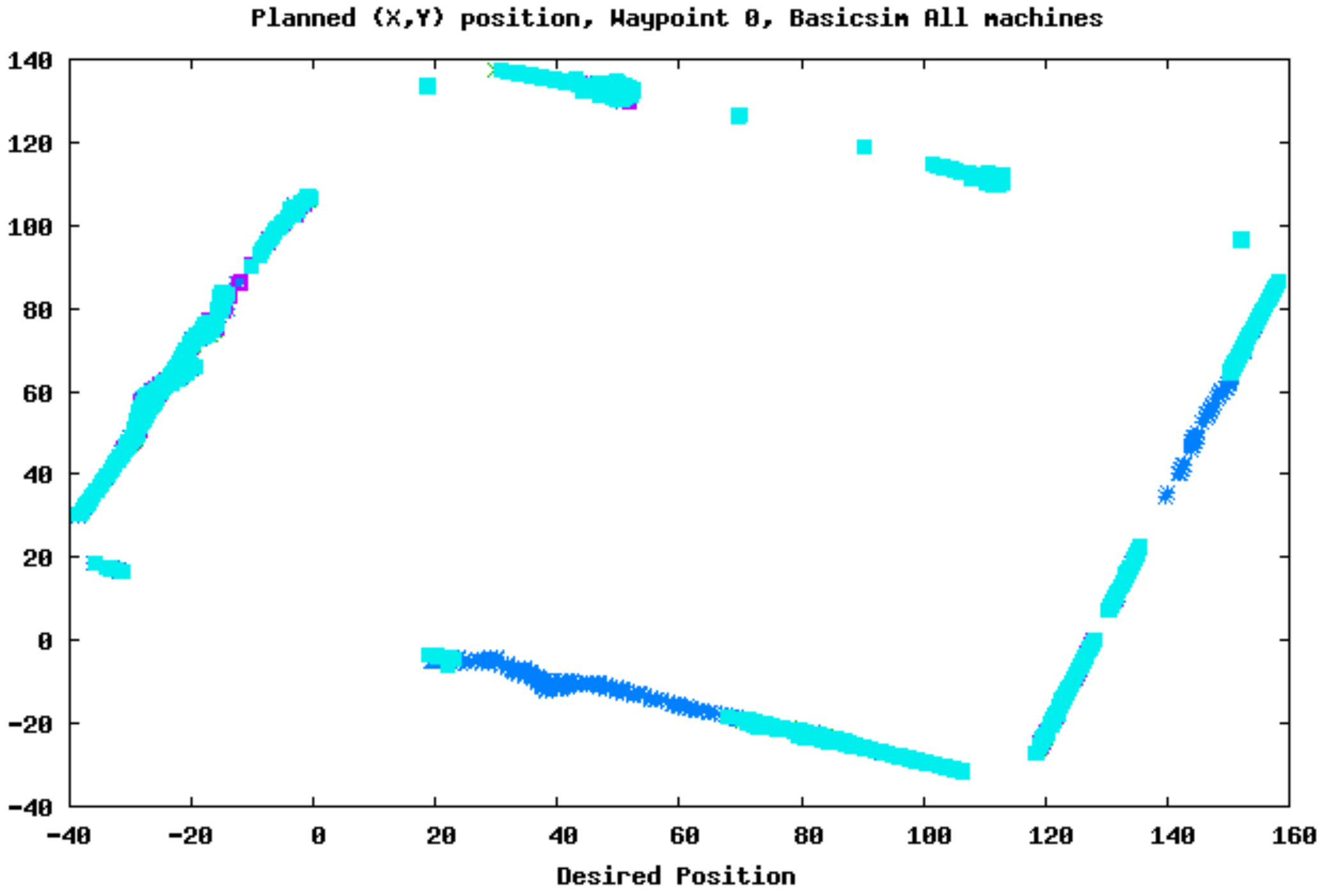
Success

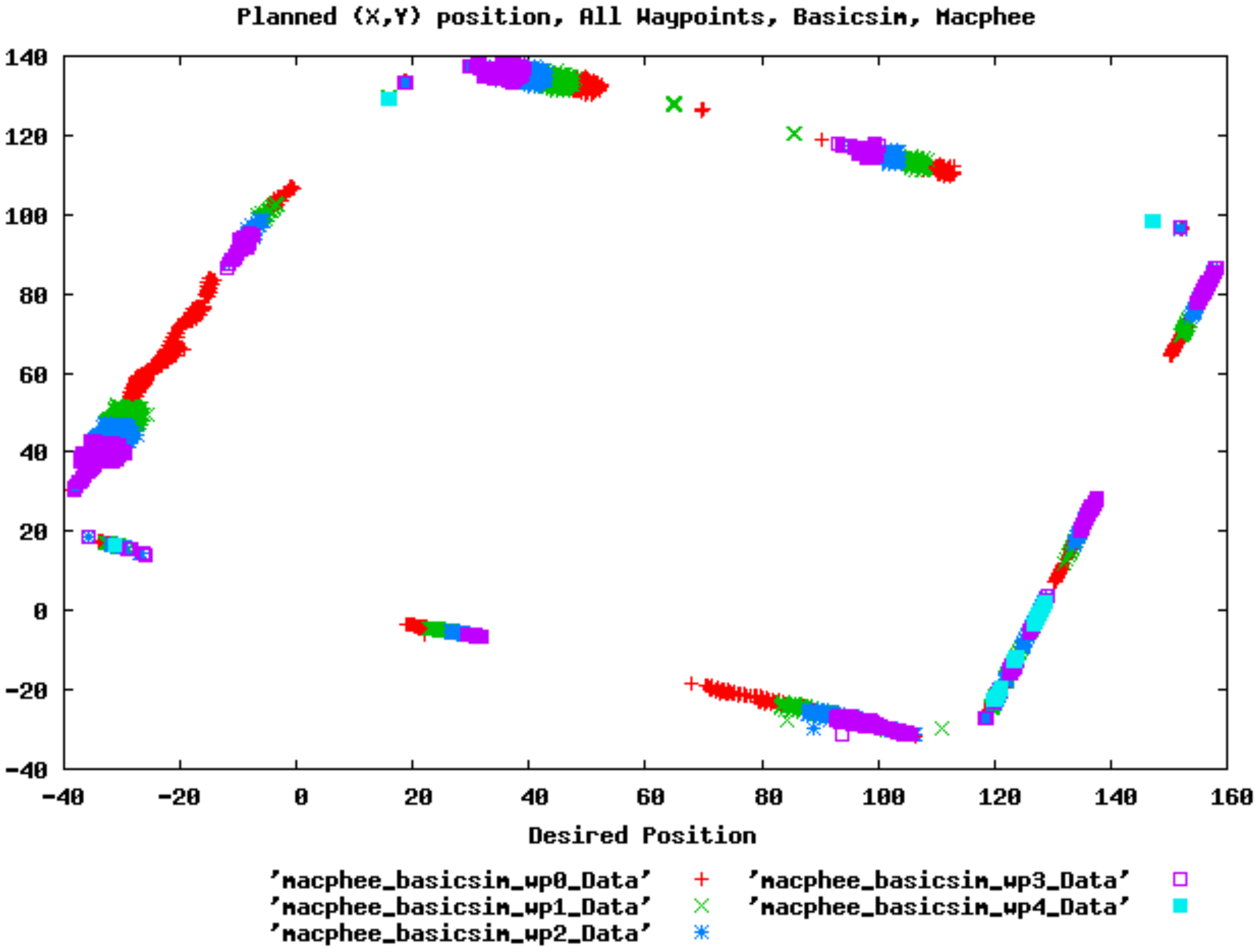


If in 200 seconds we have not passed “dead man’s curve” then we are not going to make it past.

Probability of Success (basic sim)

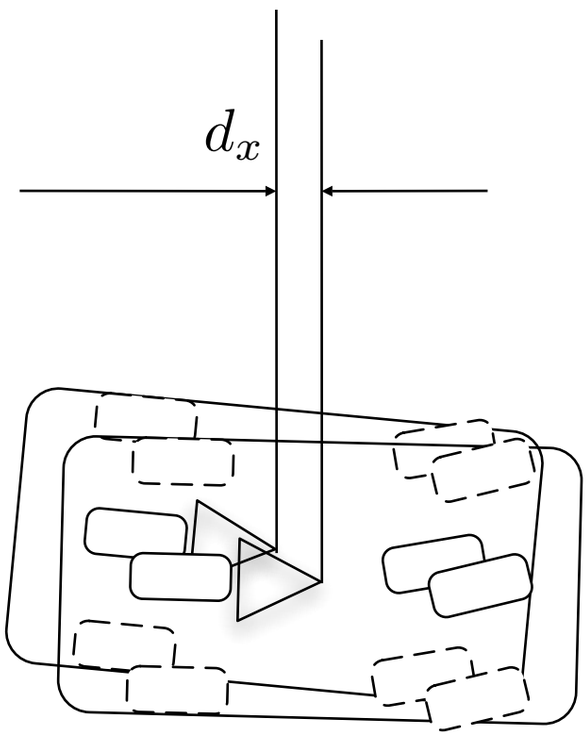


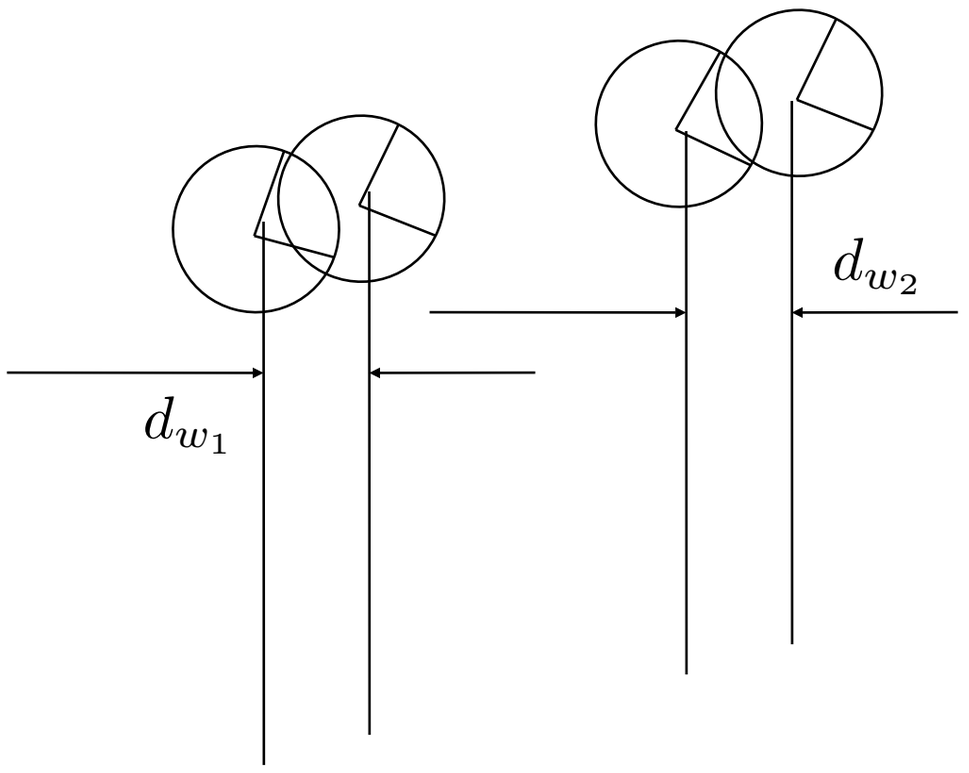

Planned (X,Y) position, Waypoint 0, Basicsim All machines
140
120
100
80
60
40
20
0
-20
-40
-40
-20
0
20
40
60
80
100
120
140
160
Desired Position
'dimble_basicsim_wp0_Data'
'feverstone_basicsim_wp0_Data'
'hardcastle_basicsim_wp0_Data'
'beames-desktop_basicsim_wp0_Data'
'macphee_basicsim_wp0_Data'


Planned (X,Y) position, All Waypoints, Basicsim, Macphee
140
120
100
80
60
40
20
0
-20
-40
-40
-20
0
20
40
60
80
100
120
140
160
Desired Position
'macphee_basicsim_wp0_Data'
'macphee_basicsim_wp1_Data'
'macphee_basicsim_wp2_Data'
'macphee_basicsim_wp3_Data'
'macphee_basicsim_wp4_Data'

- Highlevelplanner runs TOO FAST!
- We are providing waypoints so fast that the local navigator goes unstable

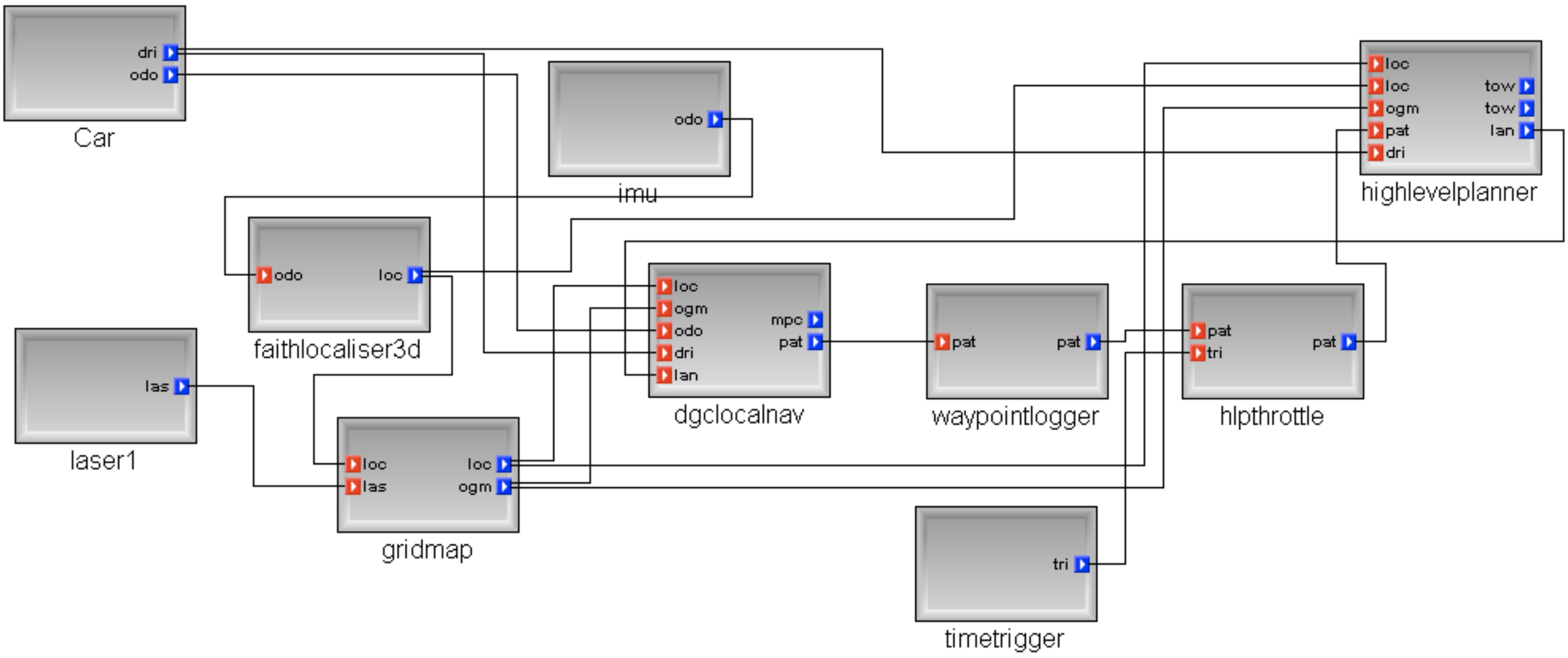
$$d_{w_1} >> d_x$$





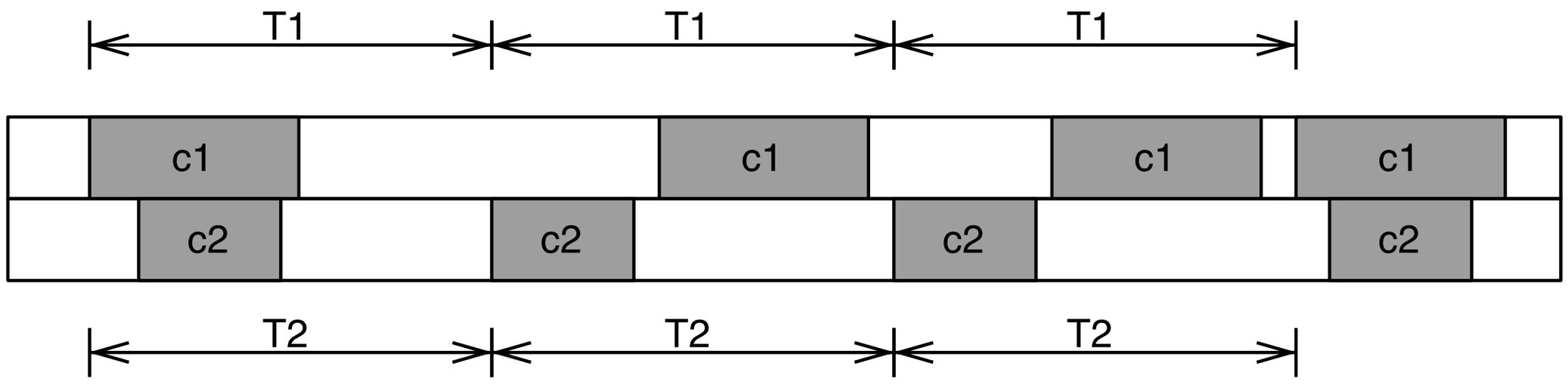
- We have some system where we have a “race condition” (no, not really, but...) that is exacerbated by an increase in the number of cores
- We see some functional (behavioral) difference, that is (somewhat) easily characterized

- Now:
  - What strategies can mitigate this issue?
  - Can we do this without changing existing code?
  - What experiments should we run to show that it is working?

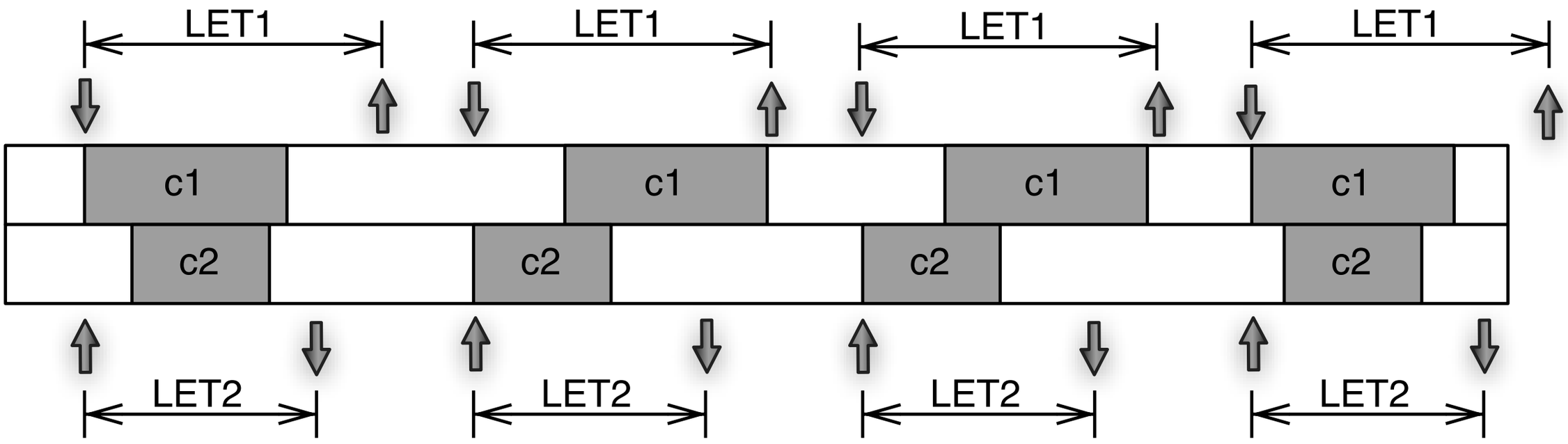

dri
odo
Car
odo
imu
loc
loc
ogm
pat
dri
tow
tow
lan
highlevelplanner
odo
loc
faithlocaliser3d
las
laser1
loc
ogm
odo
dri
lan
mpc
pat
dgclocalnav
pat
pat
waypointlogger
pat
tri
pat
hlpthrottle
loc
las
loc
ogm
gridmap
tri
timetrigger

- Do not depend on dataflow for firing rules
- Go to sleep, and wake up when they are fired (usually at some period, *T*)

- Can still lead to issues, if the start time is not consistent

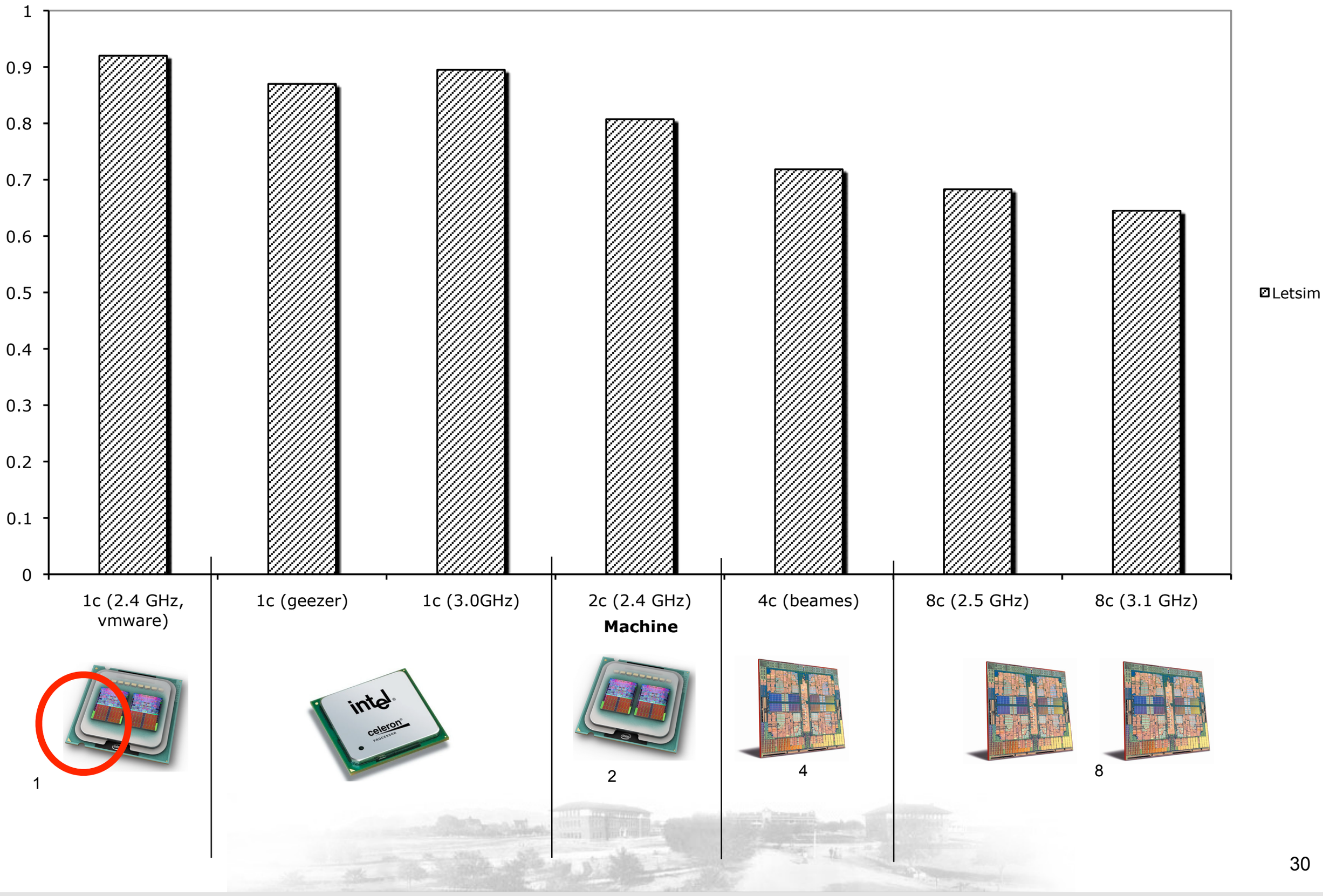


- Fix the scheduled time at which inputs are read, and outputs produced

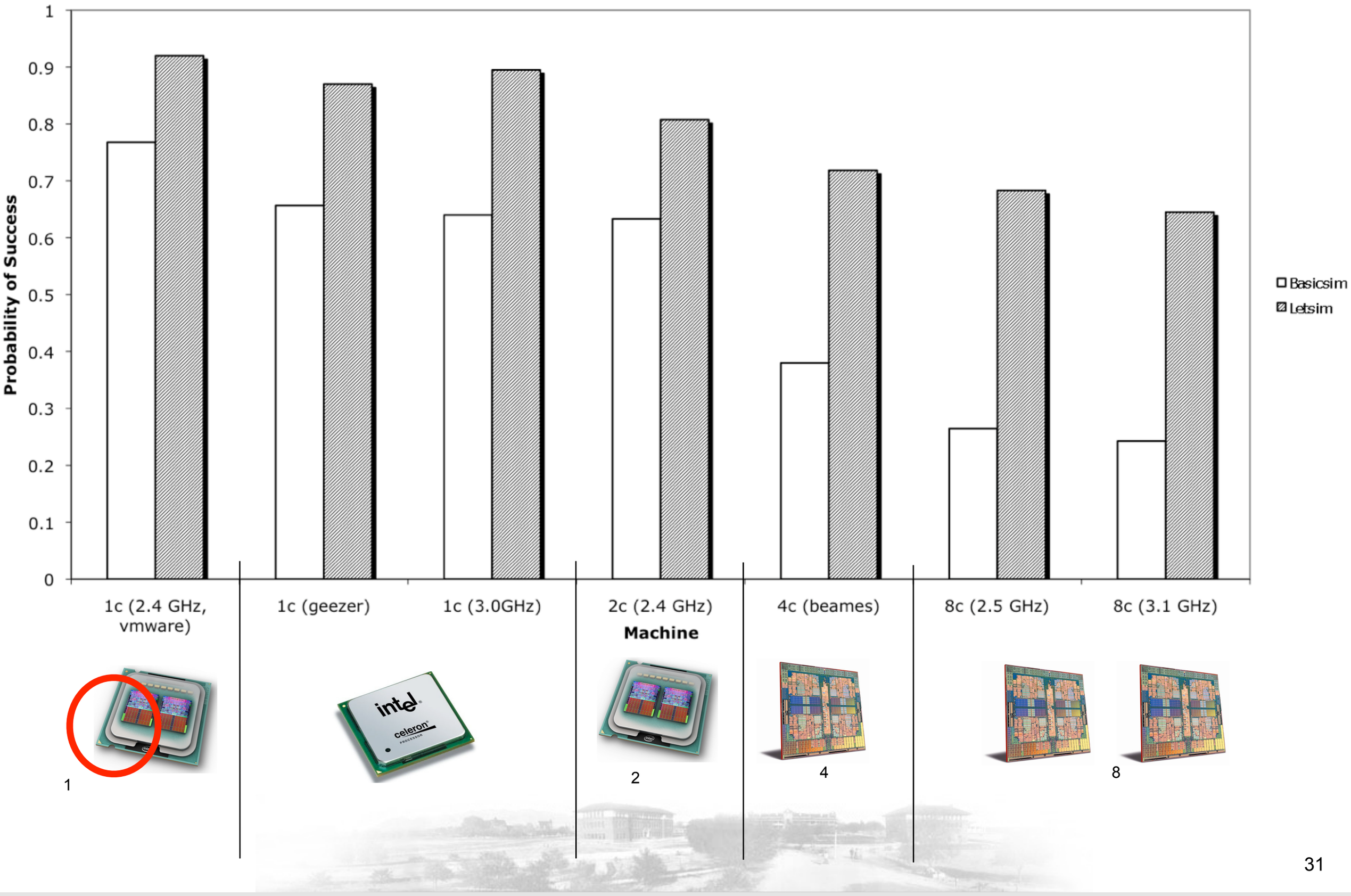


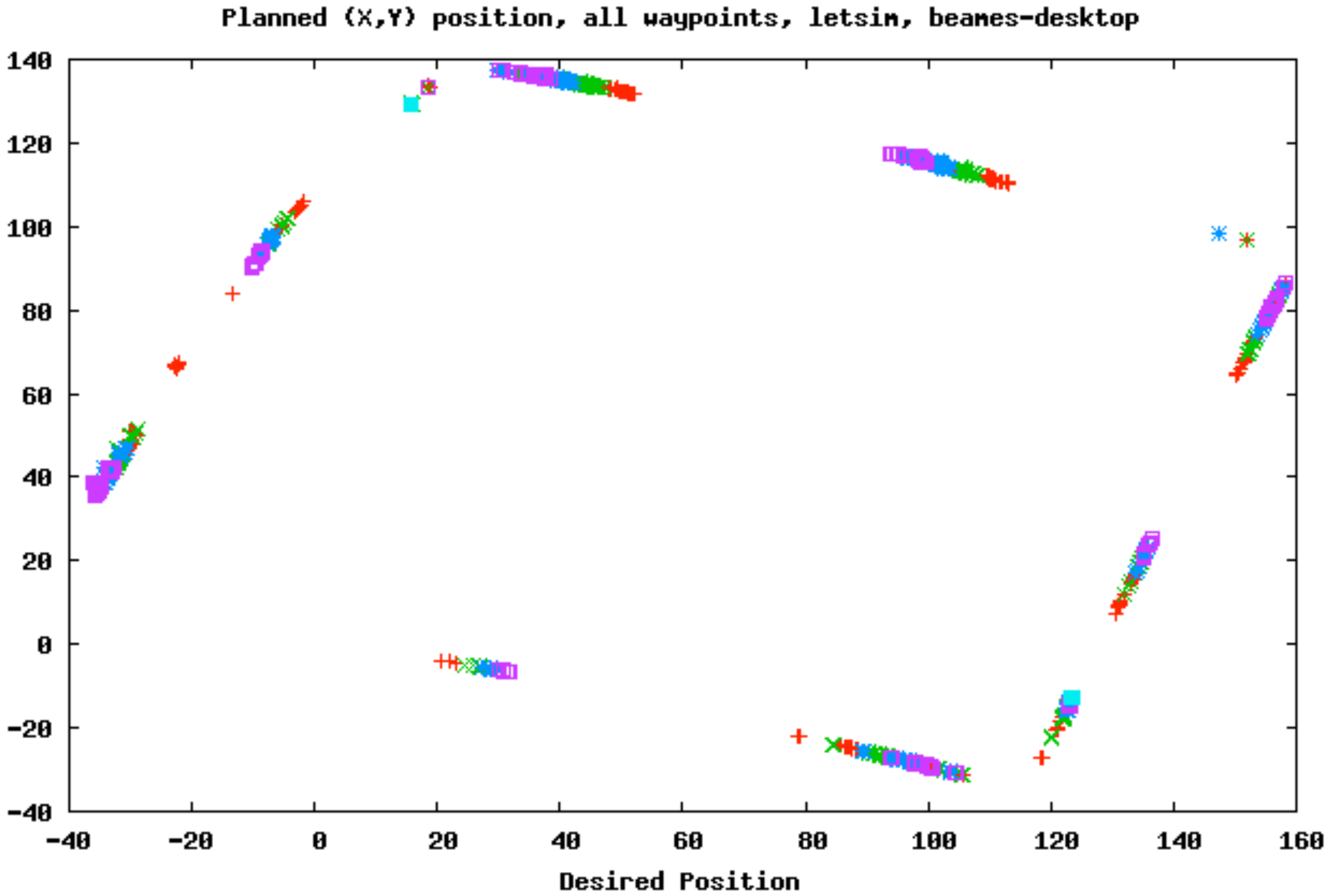
- As long as the component begins and ends between the logical beginning and end of the LET, the behavior is deterministic
- These kinds of schedules are robust to new processors (which execute faster)
- Even *lazy* versions of this work fairly well

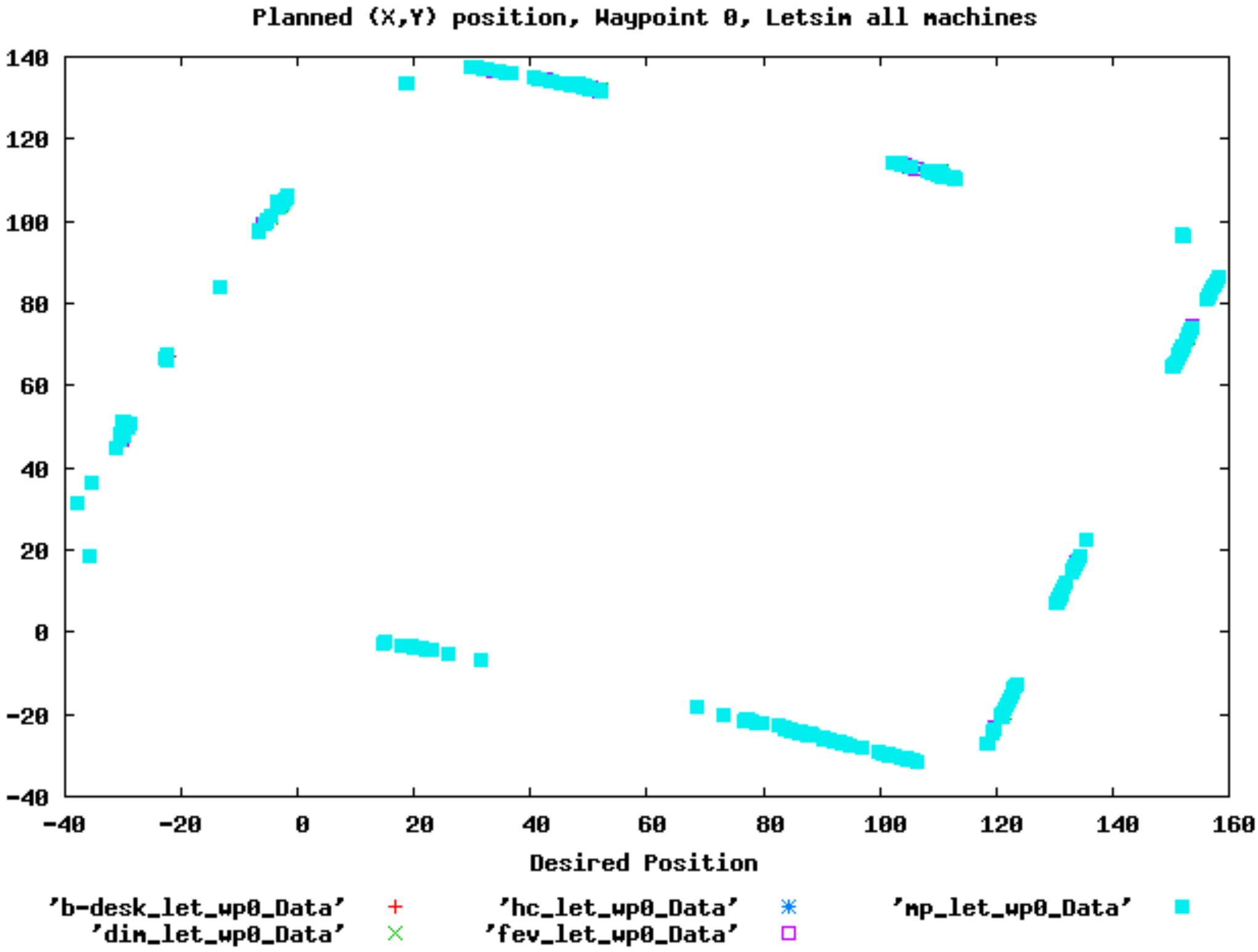
Probability of Success (letsim sim)

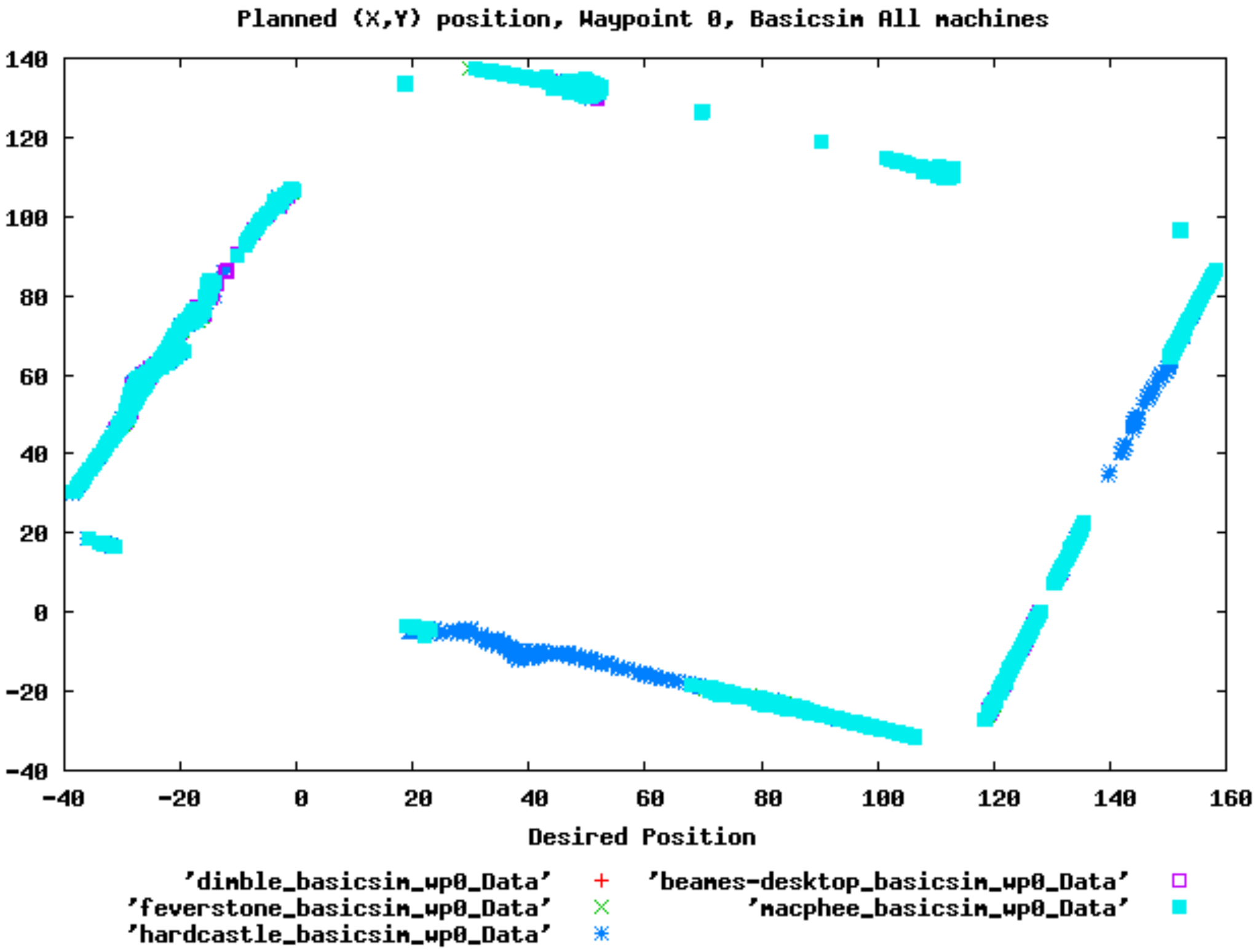


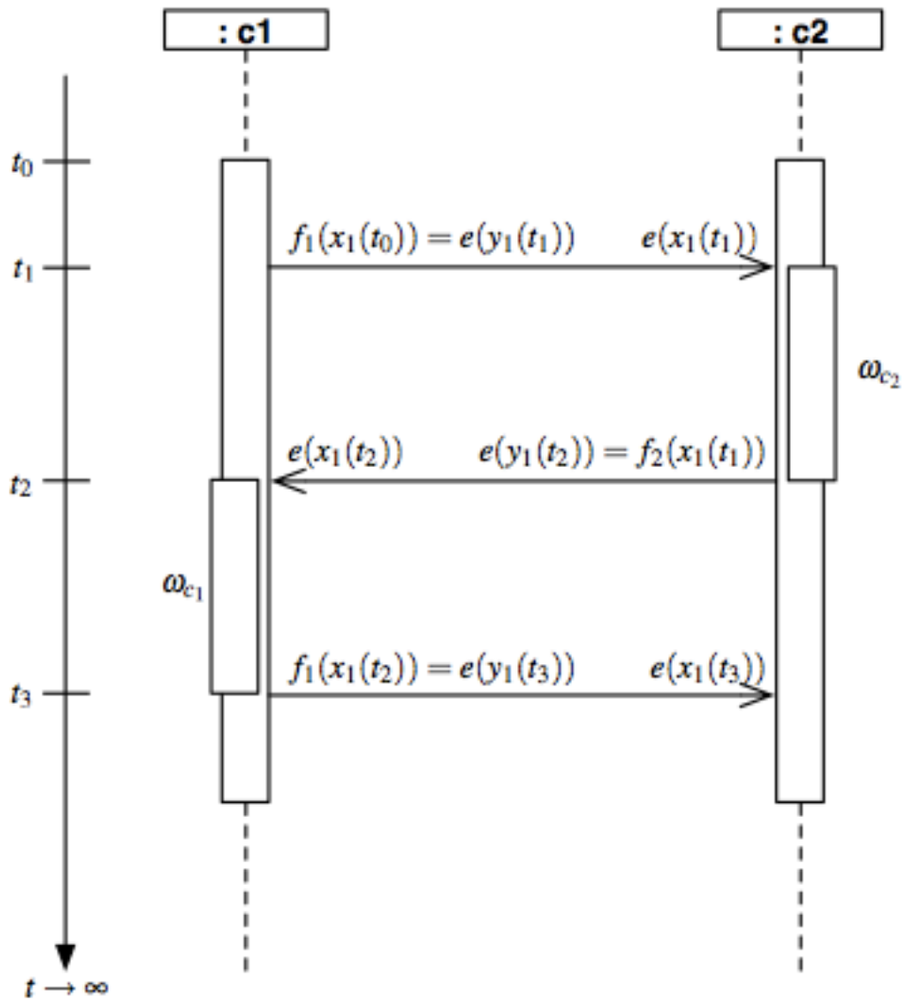
Comparison of simulation results



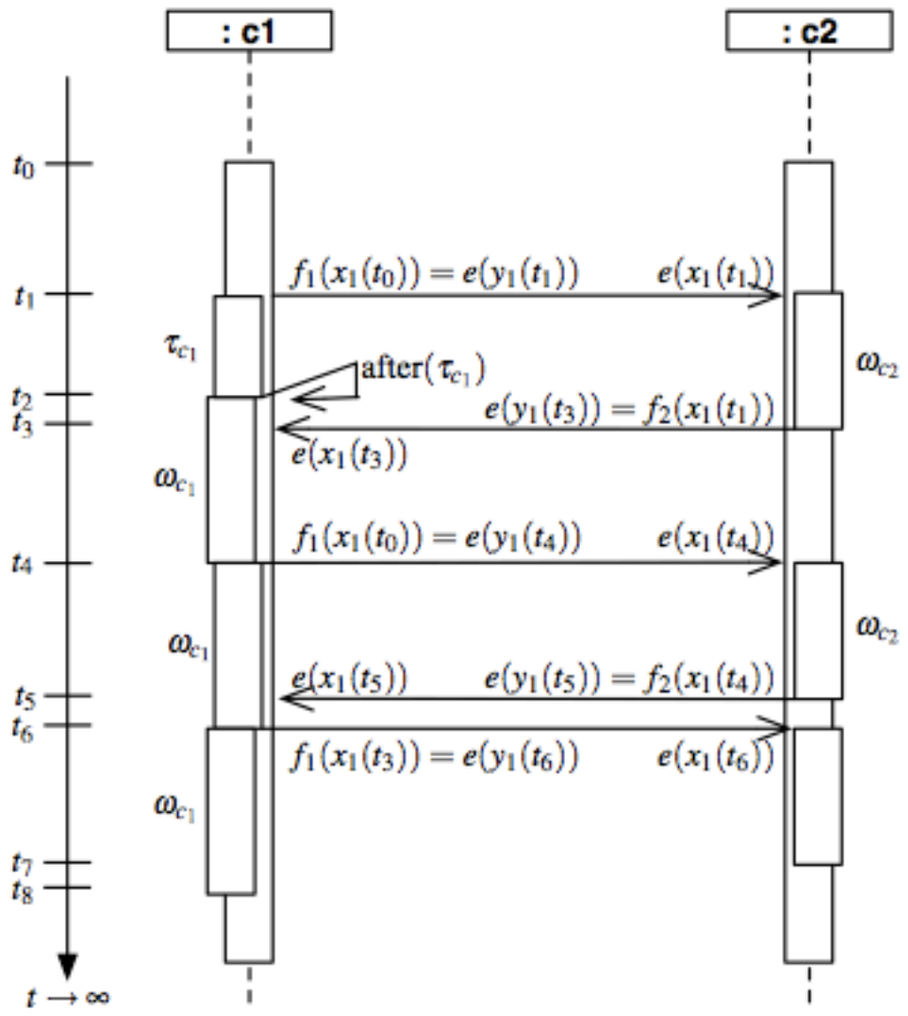

Planned (X,Y) position, all waypoints, letsim, beames-desktop
140
120
100
80
60
40
20
0
-20
-40
-40
-20
0
20
40
60
80
100
120
140
160
Desired Position
'beames-desktop_letsim_wp0_Data'
'beames-desktop_letsim_wp1_Data'
'beames-desktop_letsim_wp2_Data'
'beames-desktop_letsim_wp3_Data'
'beames-desktop_letsim_wp4_Data'


Planned (X,Y) position, Waypoint 0, Letsim all machines
140
120
100
80
60
40
20
0
-20
-40
-40
-20
0
20
40
60
80
100
120
140
160
Desired Position
'b-desk_let_wp0_Data'
'dim_let_wp0_Data'
'hc_let_wp0_Data'
'fev_let_wp0_Data'
'mp_let_wp0_Data'


Planned (X,Y) position, Waypoint 0, Basicsim All machines
140
120
100
80
60
40
20
0
-20
-40
-40
-20
0
20
40
60
80
100
120
140
160
Desired Position
'dimble_basicsim_wp0_Data'
'feverstone_basicsim_wp0_Data'
'hardcastle_basicsim_wp0_Data'
'beames-desktop_basicsim_wp0_Data'
'macphee_basicsim_wp0_Data'

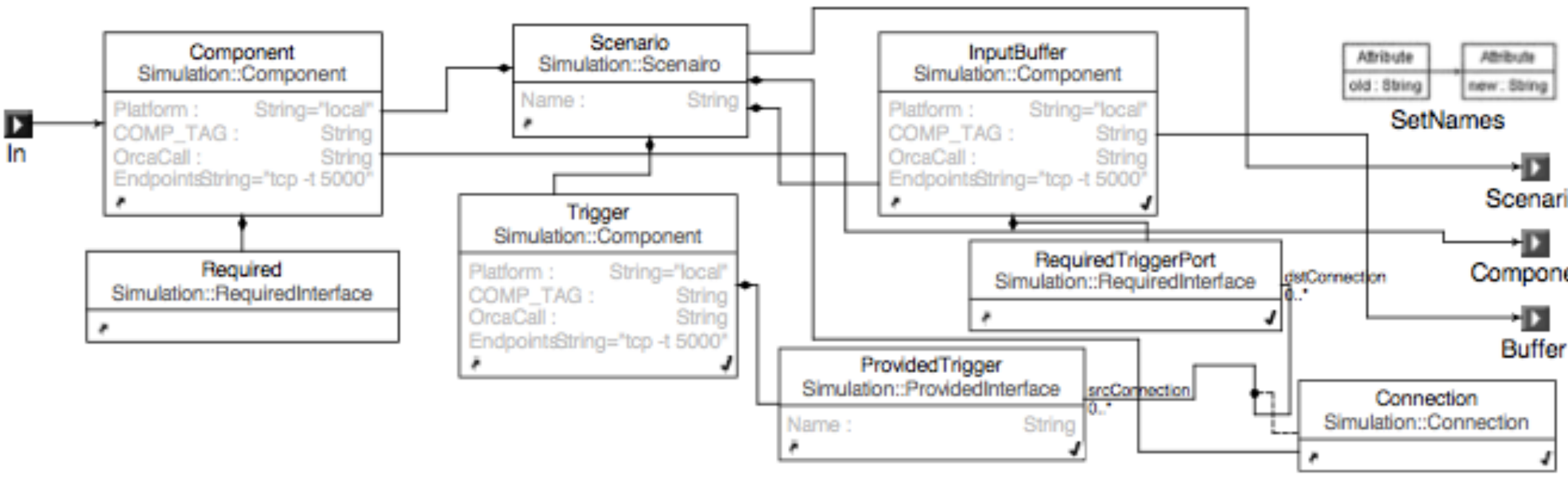


(a) Properly specified timeout values.

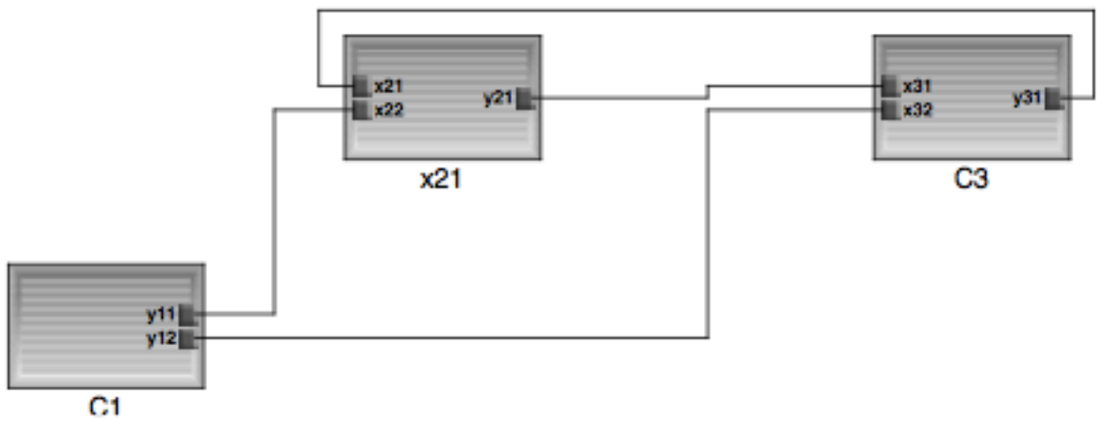


(b) Improperly specified timeout values.

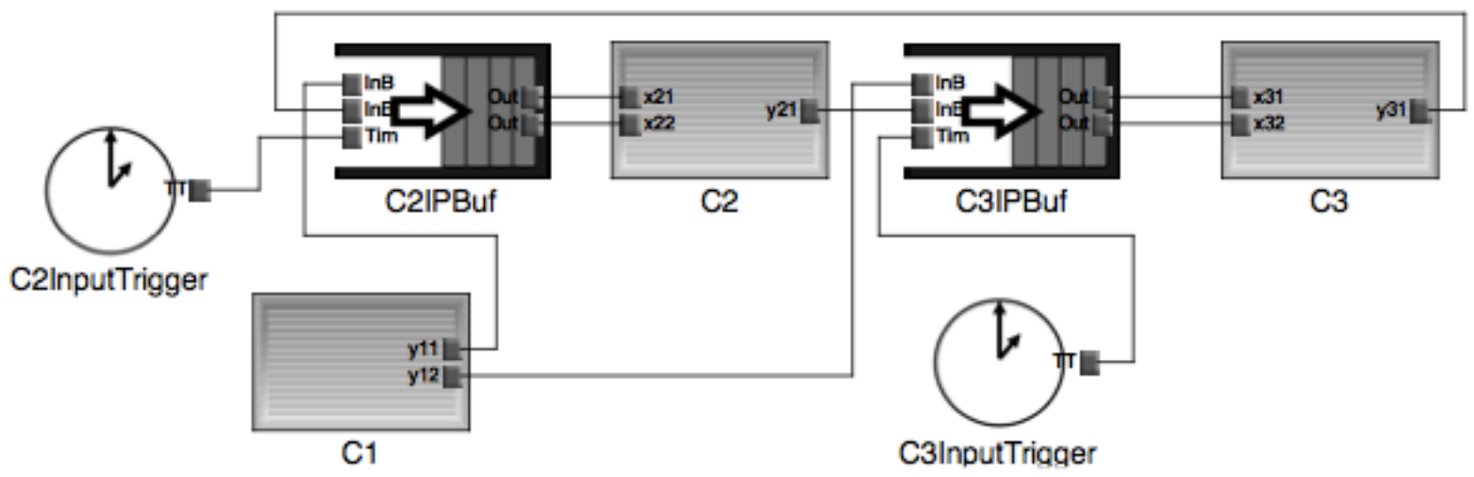
See Sprinkle, Eames, OOPSLA DSM 09



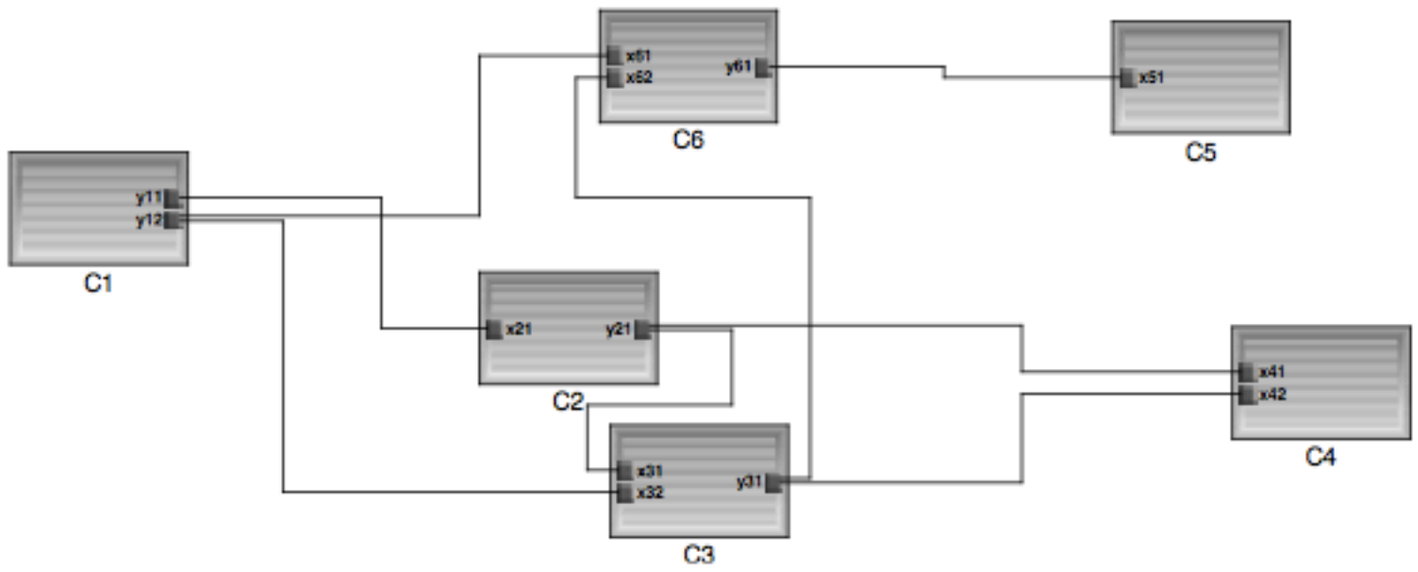
GReAT rule to create buffer components for triggered input reading based on time-triggered events. Note that the trigger scheduler is generated at the same step.



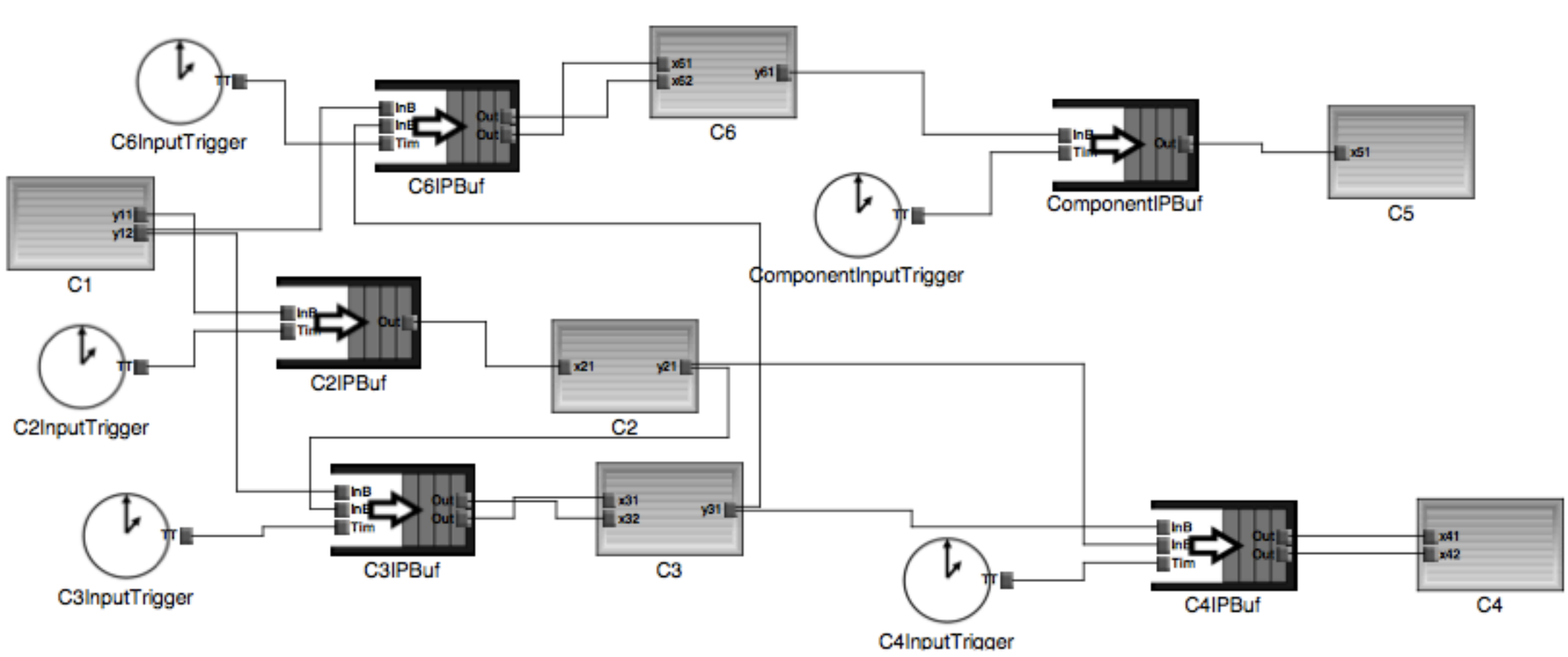
(a) Original model.



(b) Transformed model.

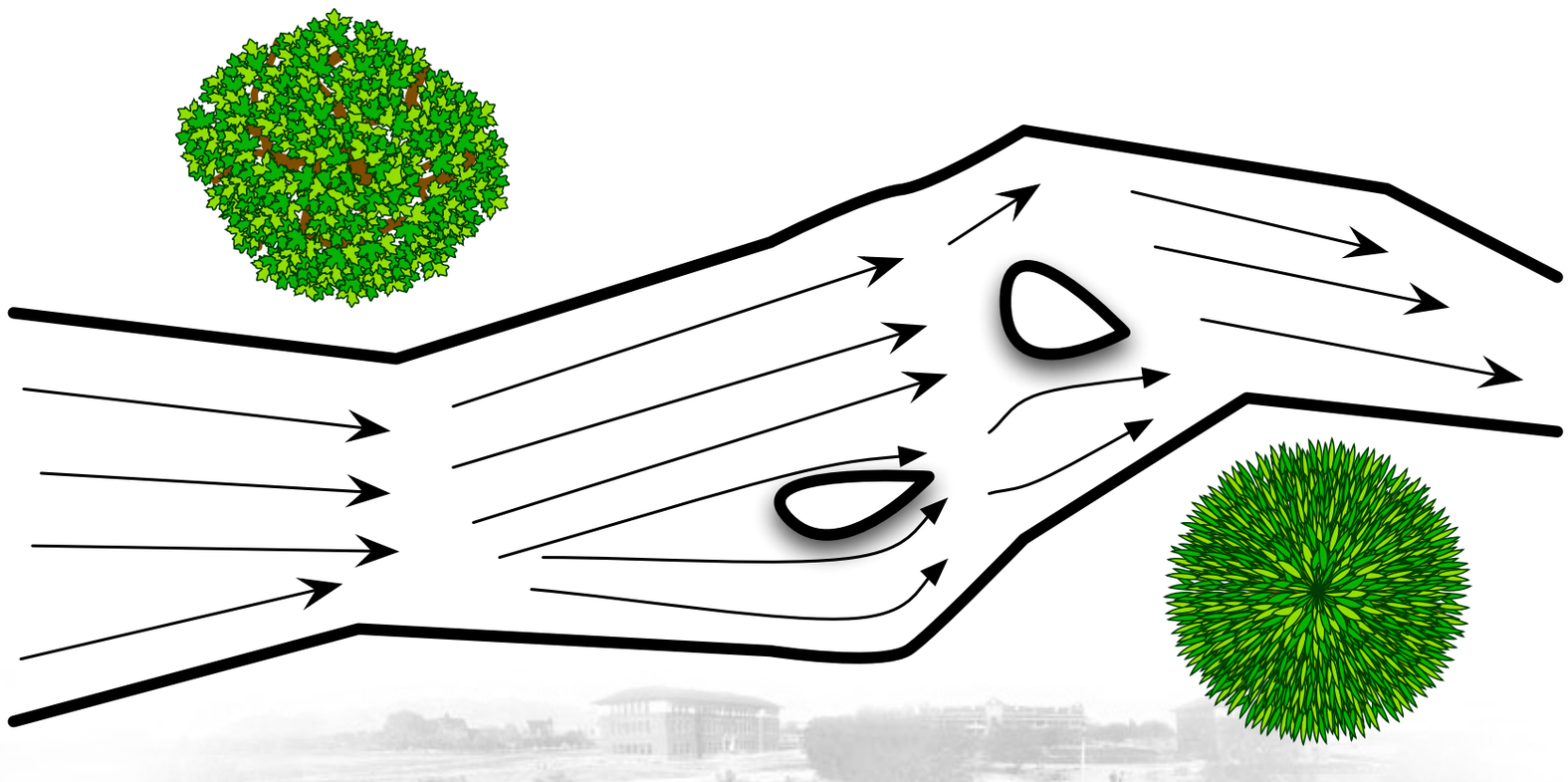


(a) Original event-based model



(b) Transformed model, now time-triggered.

- Sensors upload bursts of information, sometimes as often as every 5 minutes, sometimes as infrequent as once per day
- River flow model is built using work by A. Bayen's group at UC Berkeley for shallow water flow
- Vehicle control based on work by S. Martinez at UCSD
- Based on yesterday's river flow, today's optimization should enforce the direction of sensors that are still upstream
- Requires multiple timescales, and distributed computation, and embedded computation using (probably faulty) on-board sensors



THE UNIVERSITY OF ARIZONA

We are always looking for good graduate students.

http://ece.arizona.edu/