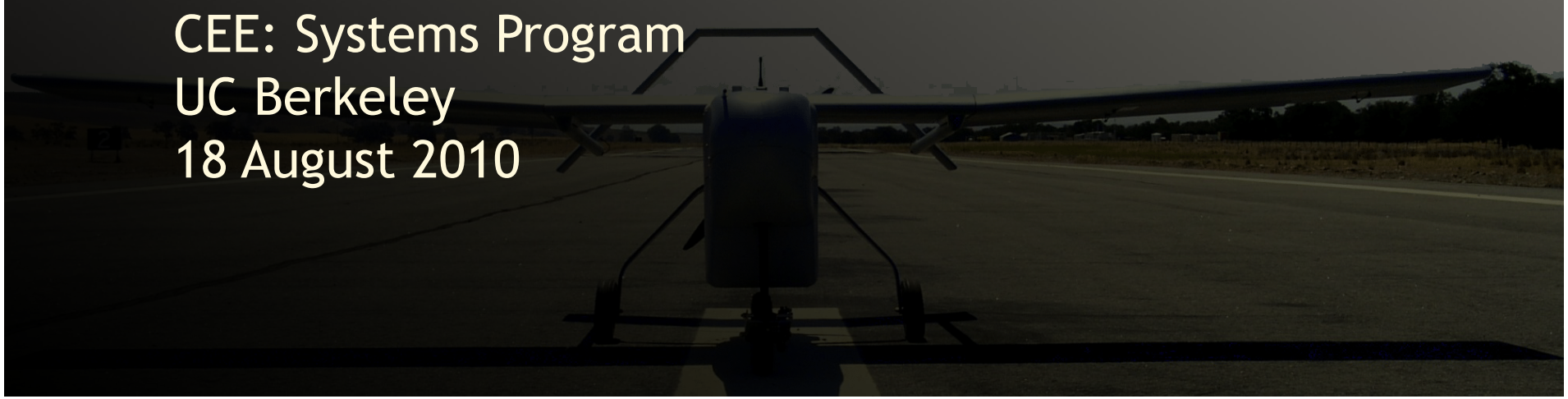
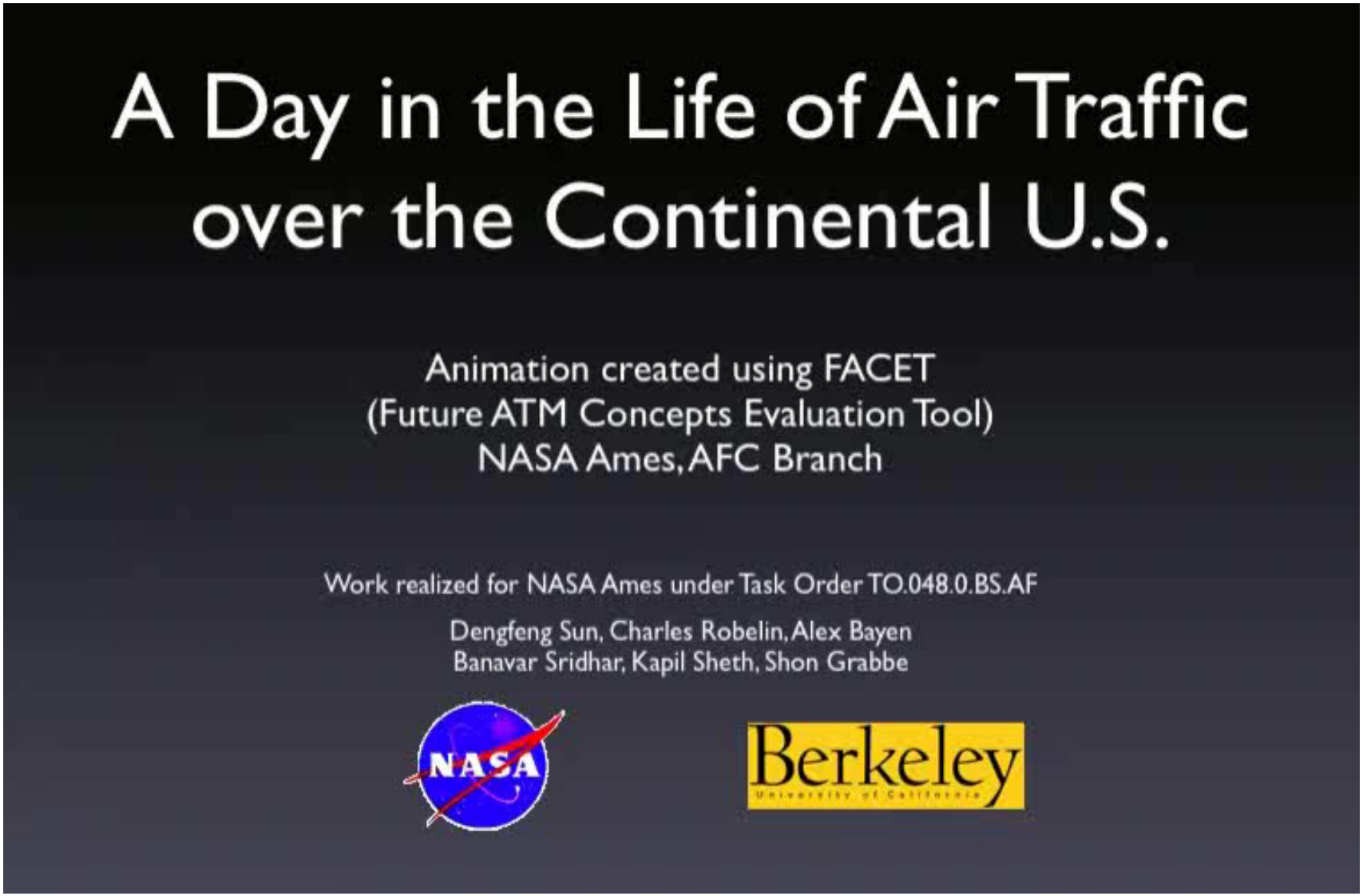
Control software for systems that change structure

Raja Sengupta, Eloi Pereira

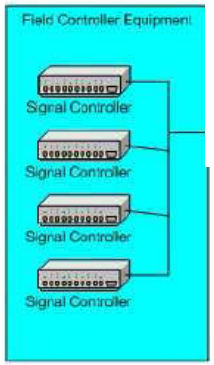
Associate Professor
CEE: Systems Program
UC Berkeley
18 August 2010



Can a Simulink Block Diagram describe this?



Another system with changing structure Los Angeles ATCS: 1 system, 3000 signals, 50 miles: Signals join and leave everyday

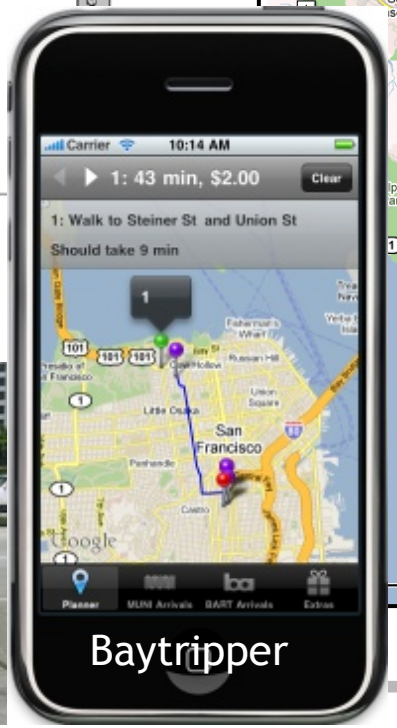


http://www.zennaro.net/projects/transportation.php

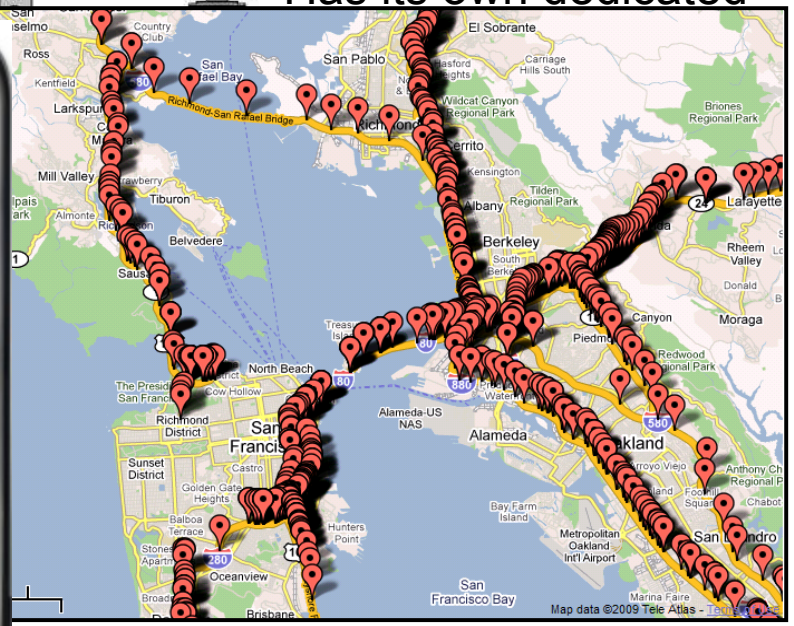
Marco Zennaro, Phd 2007, Berkeley 2070 ATCP

Los Angel
Central cl
command
signals ev

Has its own dedicated



Baytripper



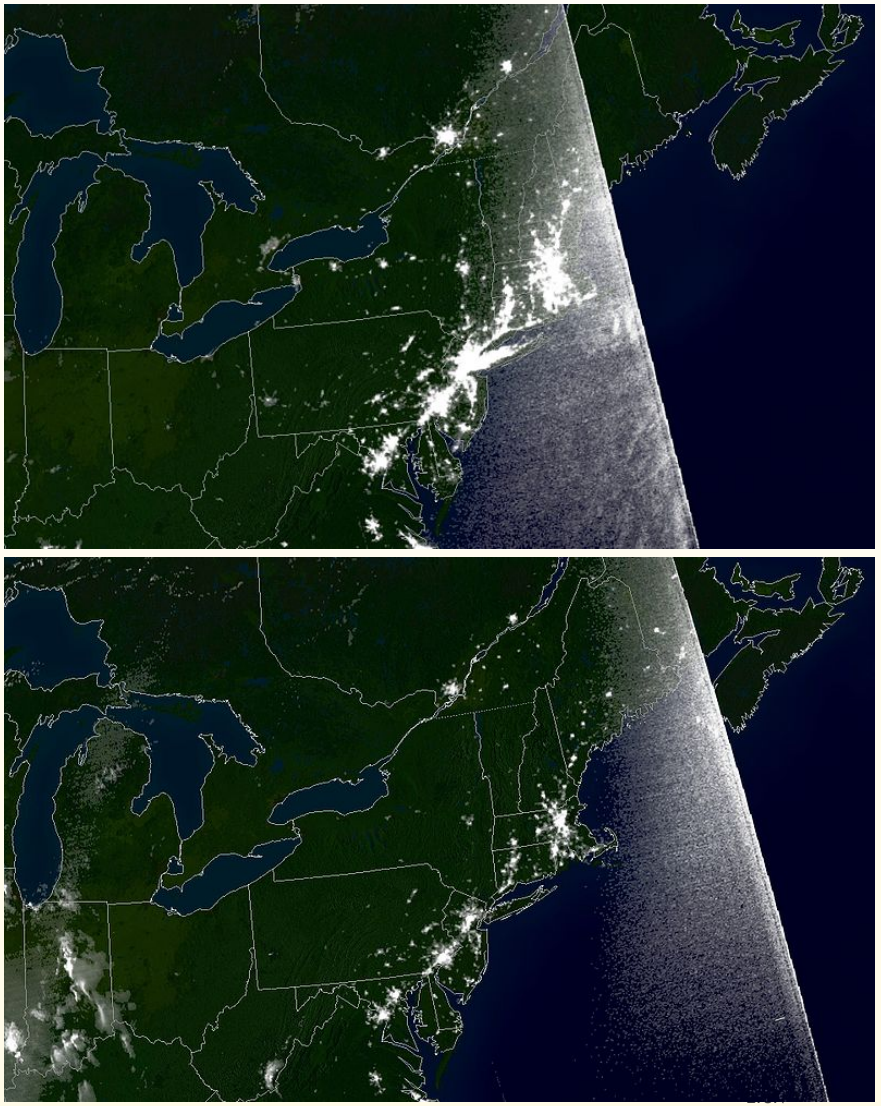
The thesis: Large systems change structure as part of normal operation

- Changes of structure
  - Components join or are created
  - Components leave or are deleted
  - New linkages are created
  - Old linkages are deleted
- When systems are sufficiently large

  these changes have to be part of normal business
- Designers keep system properties invariant to changes of structure
  - Safety, liveness, optimality, stability, …….

System Engineers successfully negotiate millions of changes. They fail rarely……

- Northeast Blackout of 2003 - “FirstEnergy’s EastLake plant shut down unexpectedly on August 14, 2003, triggering a series of problems on its transmission line that triggered a cascade effect that caused the cross-border blackout.” [CBC News]; According to NERC Final Report 256 power plants went off-line, most due to the action of automatic protective controls. 55 million people affected [Wikipedia];



System Engineers successfully negotiate millions of changes. They fail rarely……How come we succeed so often?

- January 1990, AT&T Long Distance Network Collapse - A crash in one switch due to a software bug propagated throughout the remaining 114 switches, blocking over 50 million calls in the nine hours it took to stabilize the system [CalPoly];
- April 2009, T-Mobile crash in Germany - 39 million customers were left unable to use their mobile phones to send calls or SMS text messages; The outage occurred due to a failure in a single system's component, the Home Location Register [bild.de].

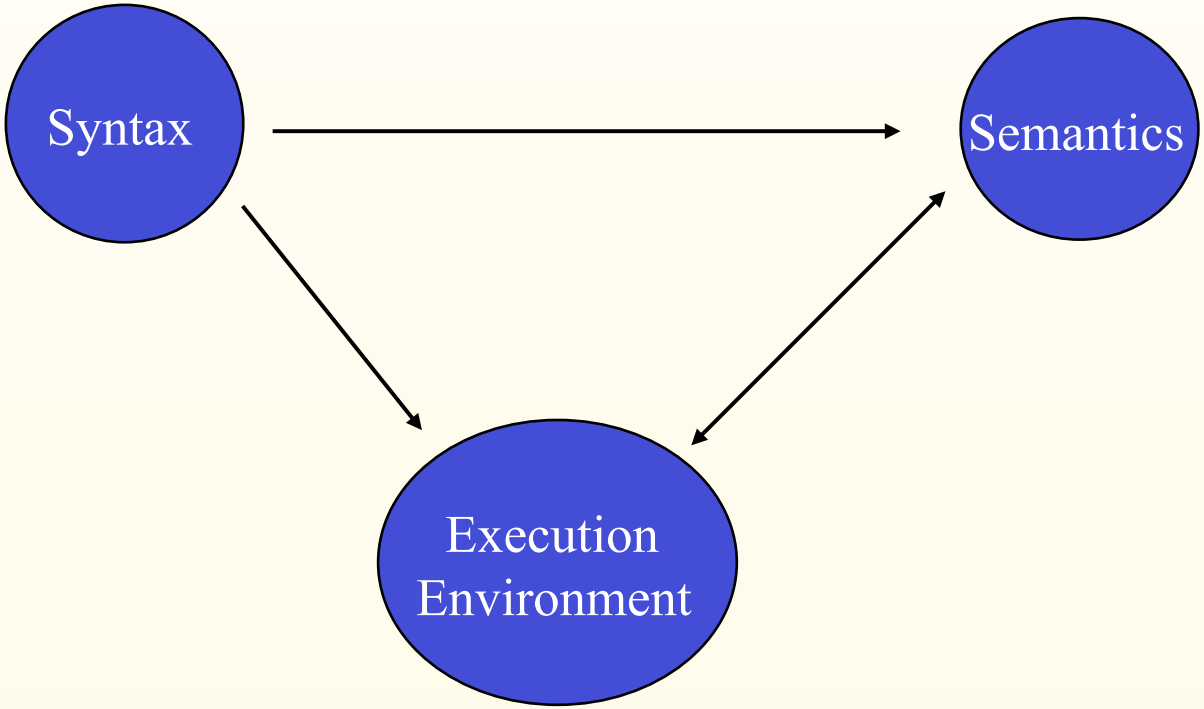
Designers keep system properties invariant to changes of structure
Safety, liveness, optimality, stability, …….

- The question I hope we can discuss today:
  - **What design support is available from the systems sciences?**
- Programming abstractions?
- Verification?
- Control theory?
- Run-time substrates? Real-time substrates?

What design support is available from the systems sciences?

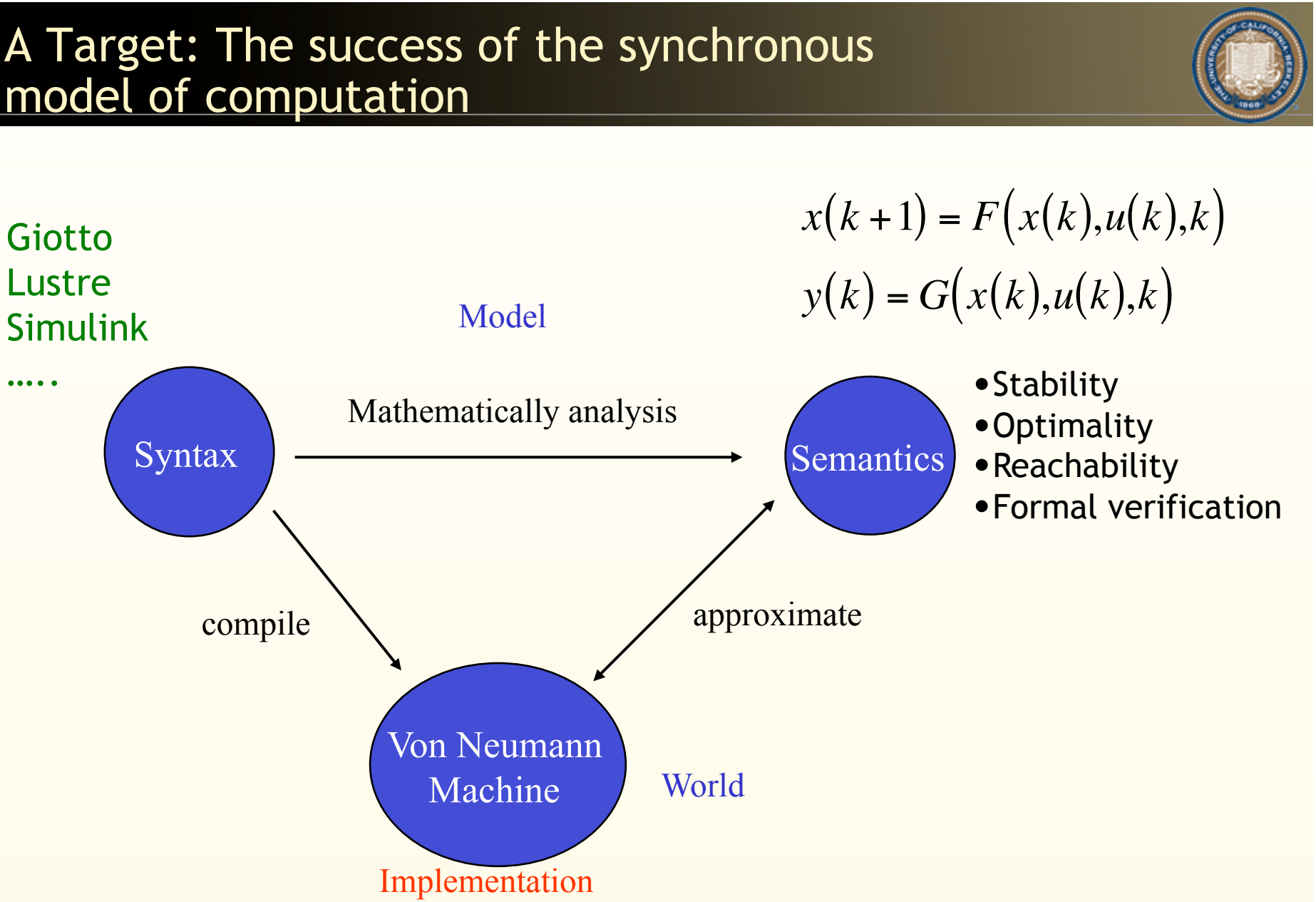
- Computer engineering community builds systems that change structure
  - **Internet, Cloud Computing, …..**
- Programming abstractions – Mainstream languages provide many abstractions
  - Run-time creation of objects
  - Dynamic heap management
  - Object serialization, persistence
  - Remote method invocation
  - Middleware, SOA - Service-oriented architecture: Loose coupling, notification
- Verification?
- Control theory?
- Run-time substrates? Real-time substrates?

What is design support from the systems sciences?



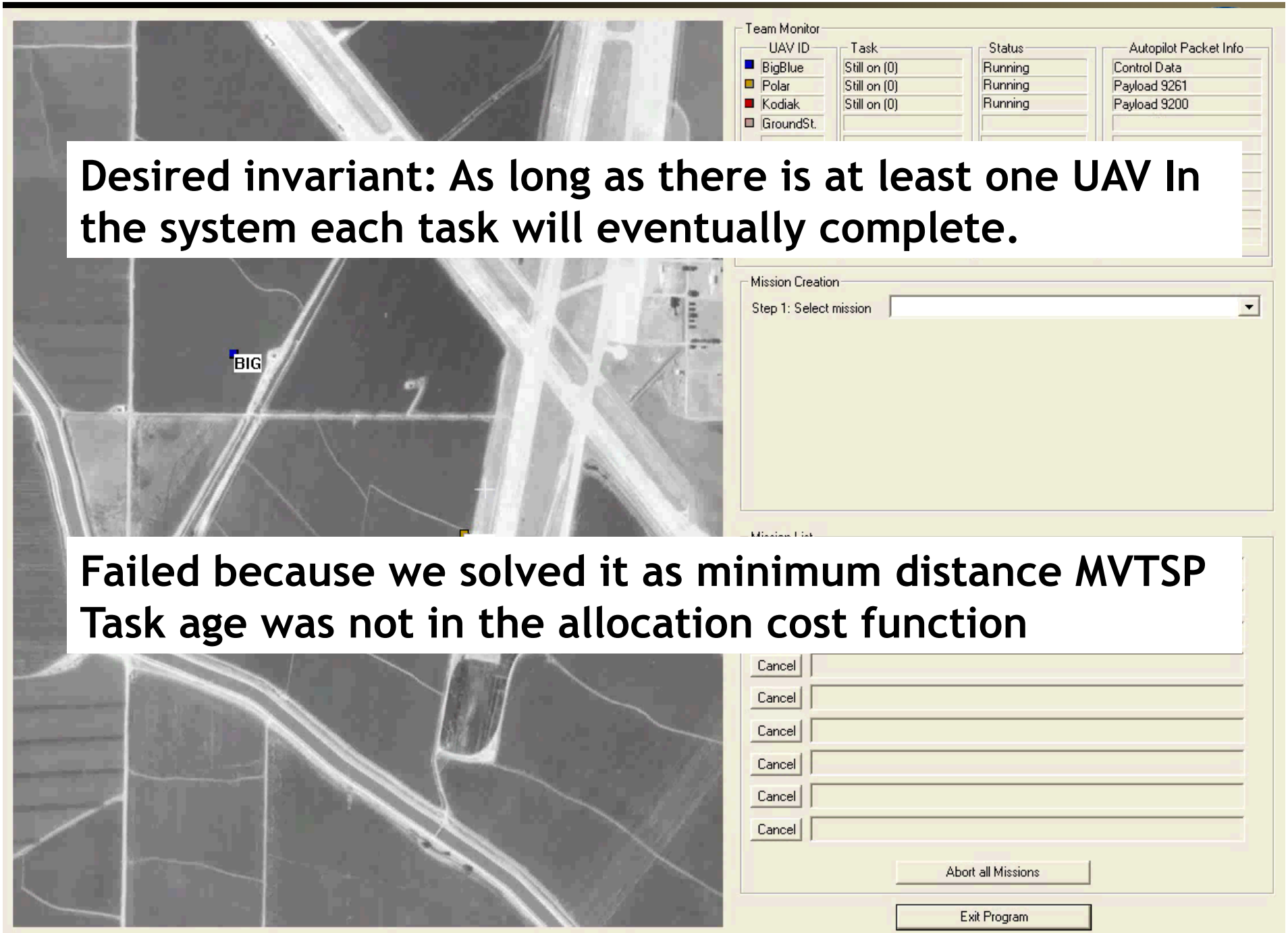
Varaiya 2001

A Target: The success of the synchronous model of computation



A laboratory: And an illustrative mistake

- http://c3uv.berkeley.edu
- Center for Collaborative Control of Unmanned Vehicles
- Fellow faculty J. Karl Hedrick, C. Kirsch, Y. Fallah

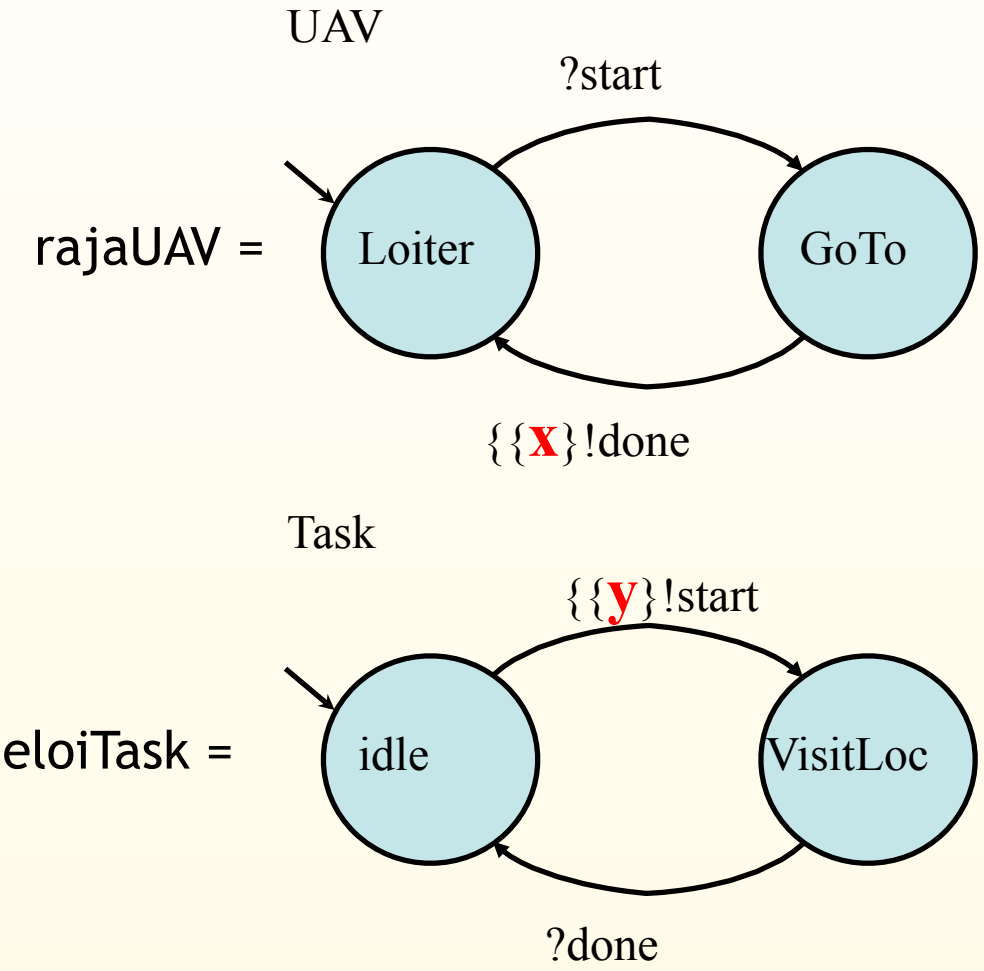


Desired invariant: As long as there is at least one UAV In the system each task will eventually complete.

Failed because we solved it as minimum distance MVTSP Task age was not in the allocation cost function

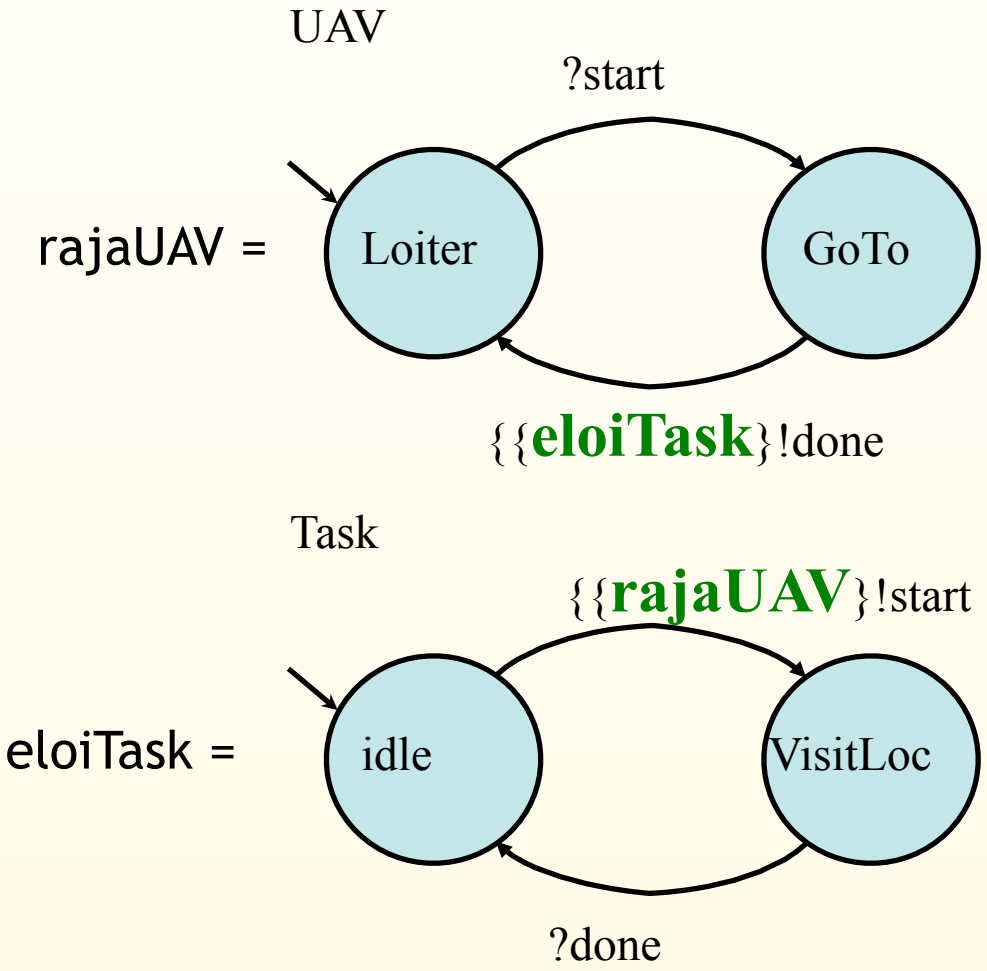
Can formal verification or control synthesis methods help?

Before Allocation: x, y free

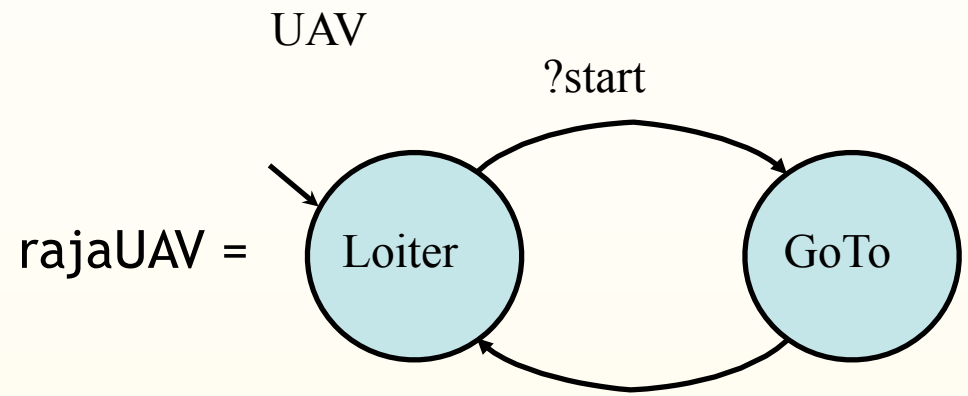


Can formal verification or control synthesis methods help?

After Allocation: x, y bound

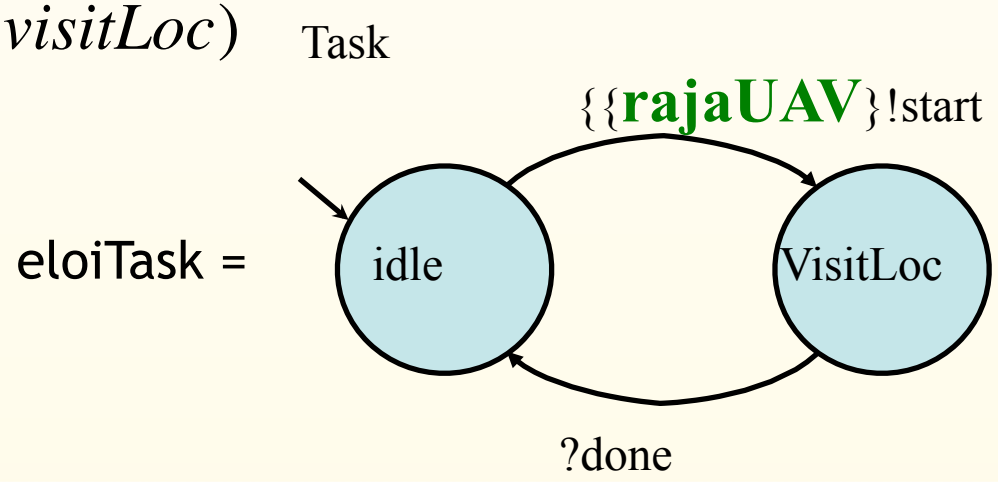


The bound system can be verified
Control synthesis as in SCT



$(loiter, idle)$

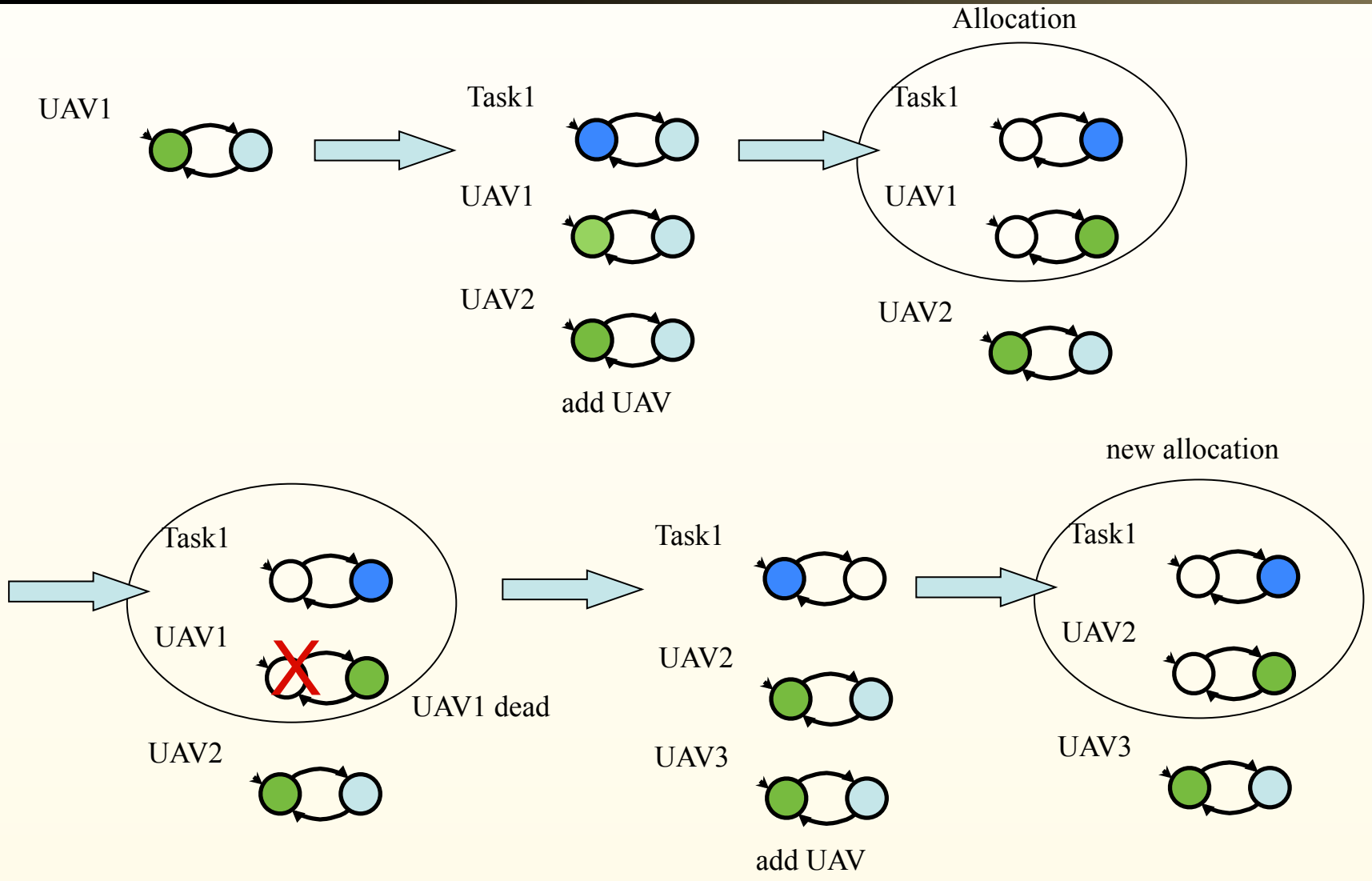
$\models eventually.(goTo, visitLoc)$



$$(loiter, idle) \xrightarrow{\{x,y\}} (GoTo, VisitXY) \xrightarrow{\{done\}} (loiter, idle) \xrightarrow{\{x,y\}} (GoTo, VisitXY)$$

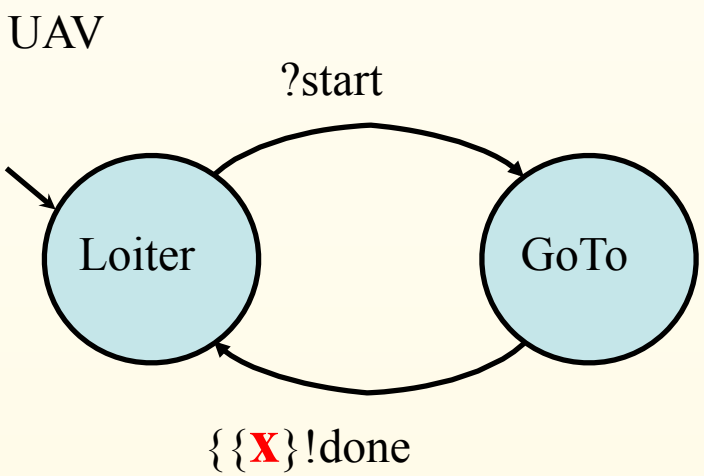
$$\xrightarrow{\{done\}} (loiter, idle) \xrightarrow{\{x,y\}!a} (GoTo, VisitXY) \ldots$$

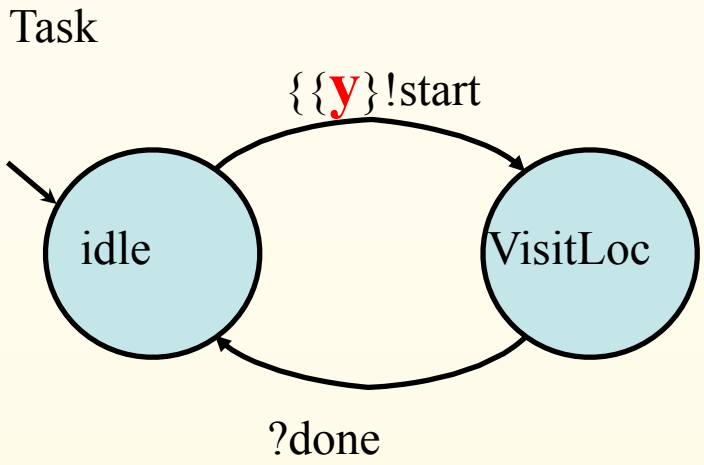
The system we would like to control and verify



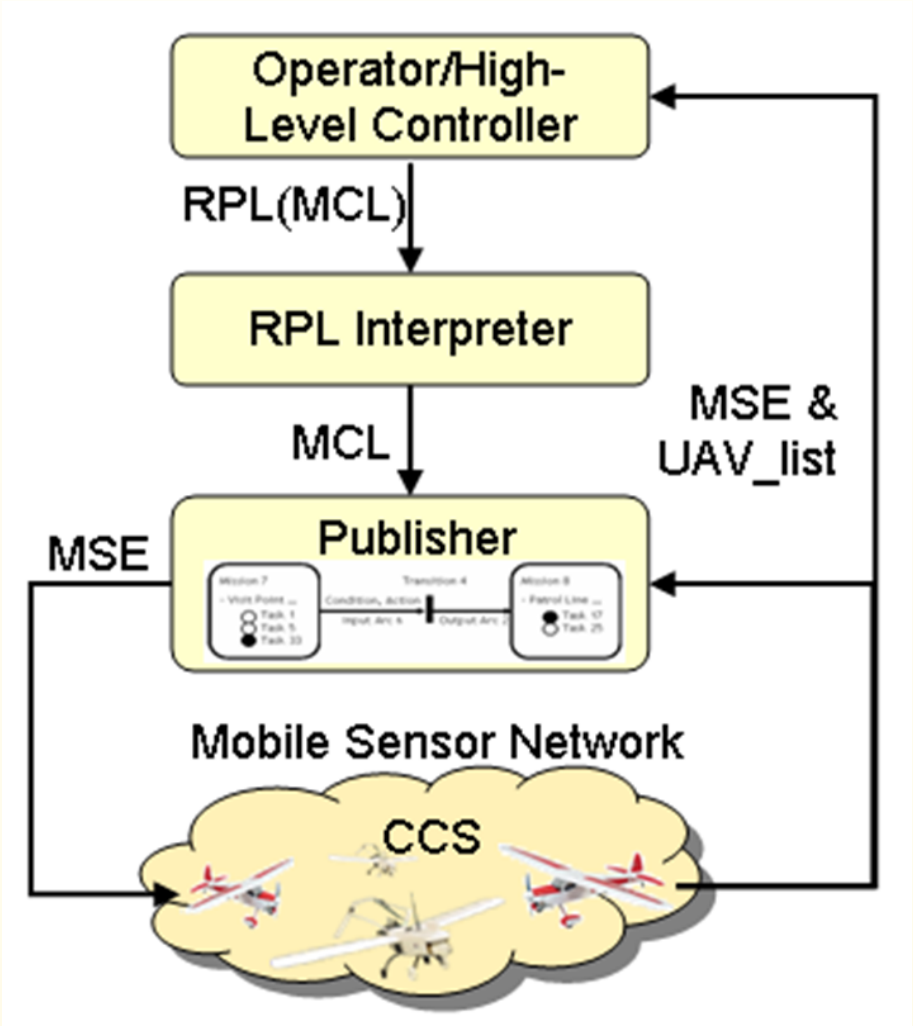
The system we would like to control and verify

always. As long as there exists a UAV x.
for all tasks y. eventually. state.y = visitLoc





Additional Complexities



Its not the same UAV that exists

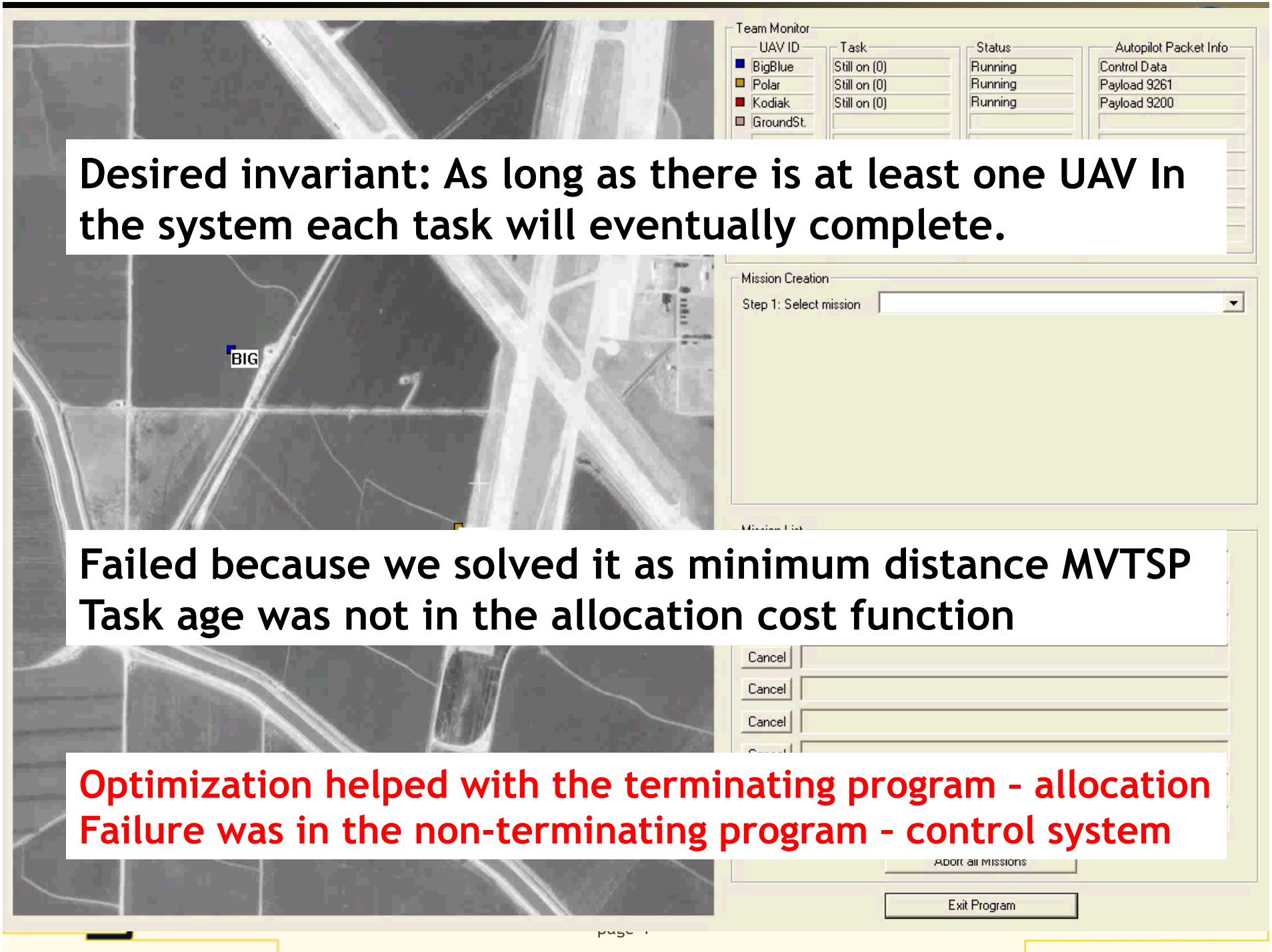
The communication channels also keep changing

Verification: Some things we have learnt about

- Distefano et al. introduced a logic where allocation/deallocation are captured as first-order concepts;
  - Allocation Temporal Logic (*All*TL) is an extension of Linear TL that allows existential quantification over variables which represent entities.
- German et al. present model checking of concurrent systems containing arbitrary number of finite-state processes;
  - The processes communicate through synchronous actions similar to Milner's Calculus of Communicating Systems;
  - The specification language used is Propositional Temporal Logic;
- Baukus et al. and Lesens et al. present methods for abstraction and formal verification of parameterized networks;
  - A parametrized network is the parallel composition of an arbitrary but finite number of identical processes;
  - Both references present methods for the abstraction of parameterized networks and semi-automatic Formal Verification using theorem proving;

Invariance to changes of structure
What design support is available from the systems sciences?

- Programming abstractions – Mainstream languages provide many abstractions
- Verification – Can systems with changing structure be verified?



Desired invariant: As long as there is at least one UAV In the system each task will eventually complete.

Failed because we solved it as minimum distance MVTSP Task age was not in the allocation cost function

Optimization helped with the terminating program – allocation Failure was in the non-terminating program – control system

Invariance to changes of structure
What design support is available from the systems sciences?

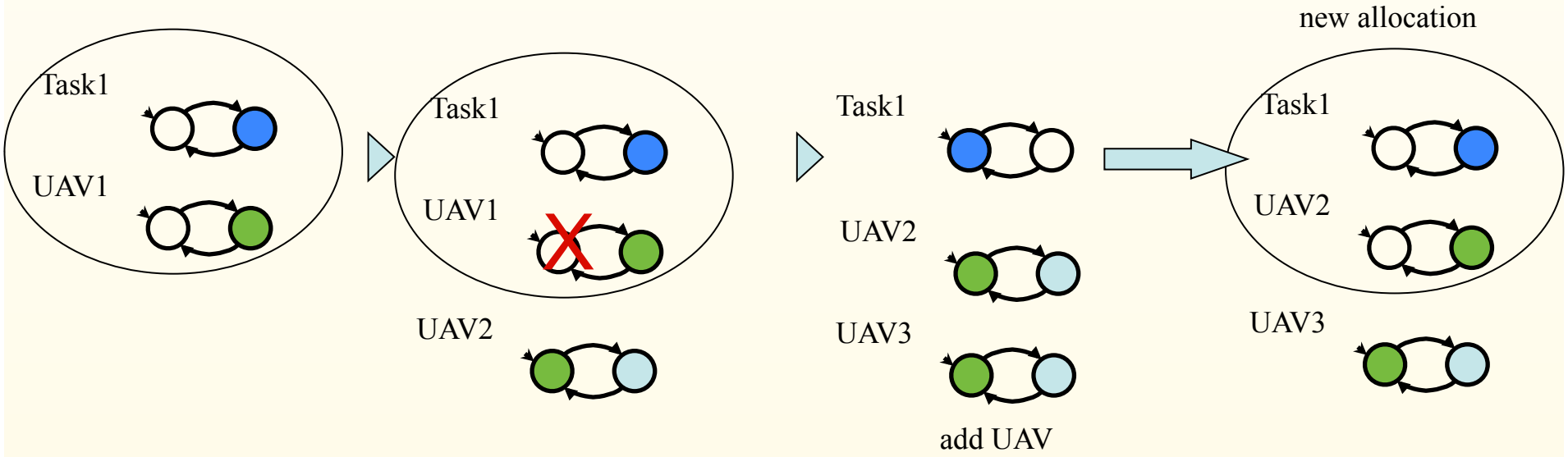
- Programming abstractions – Mainstream languages provide many abstractions
- Verification – Can systems with changing structure be verified?
- Can we have a Control Theory for systems with changing dimension?
  - Asymptotics - Flocking, graph labplacians, switched systems, string stability, ….
  - Finite-time?
    - Safety
    - Can these open systems have notions of finite-time performance?
      - Phase transitions
- Run-time substrates, Real-time substrates

Control software for fixed structure goes further

The ~~system~~ program we would like to verify

always. As long as there exists a UAV x.
for all tasks y. eventually. visitLoc

What formal syntaxes can such programs be written in?



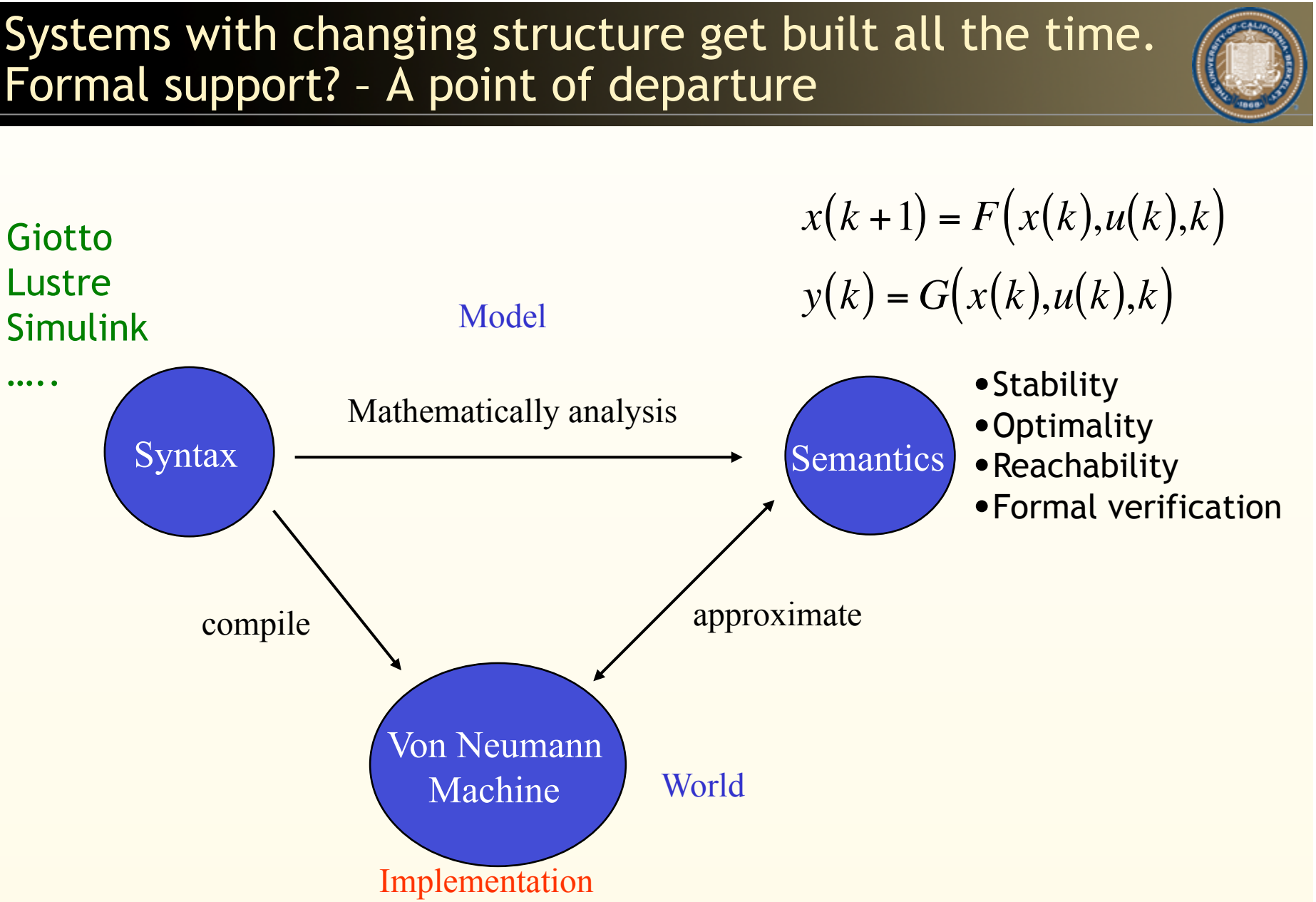
The way I think of it

$$
\begin{array}{l}
(\texttt{if } \exists task.published(task) \wedge \exists uav.available(uav) \\
\quad \texttt{then } allocate(uav, task)) \\
\| \\
(\texttt{if } \exists uav.\exists task.(dead(uav) \wedge allocated(uav, task)) \\
\quad \texttt{then } deallocate(uav, task))
\end{array}
$$

- Use the operators of temporal programming – sequential, parallel, conditional
  - A model of computation
- The conditional needs to generalize from propositional logic towards first-order logic
  - Horn logic?
- Recursion in the sequential and parallel operators

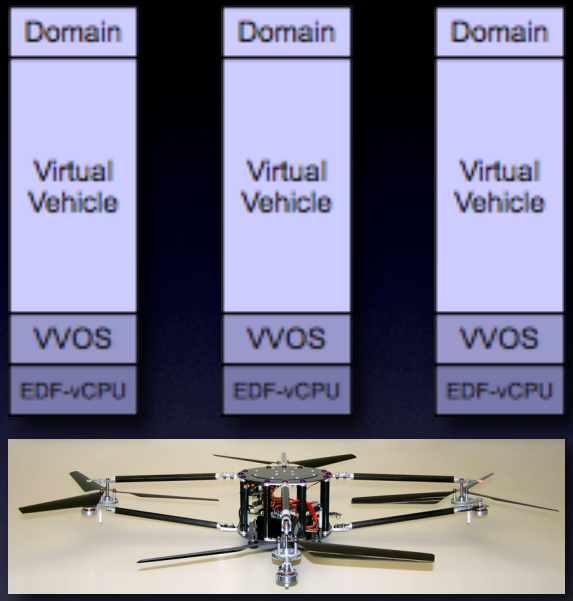
What of the pi-calculus, bi-graphs, SHIFT, Prolog, CPL, ……?

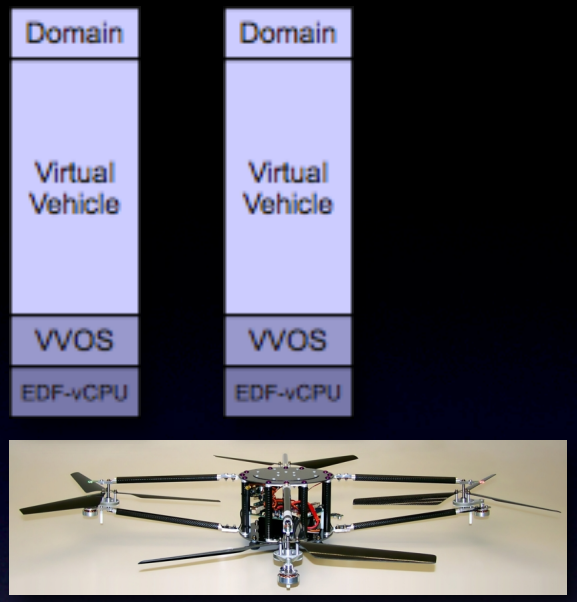
Systems with changing structure get built all the time. Formal support? – A point of departure



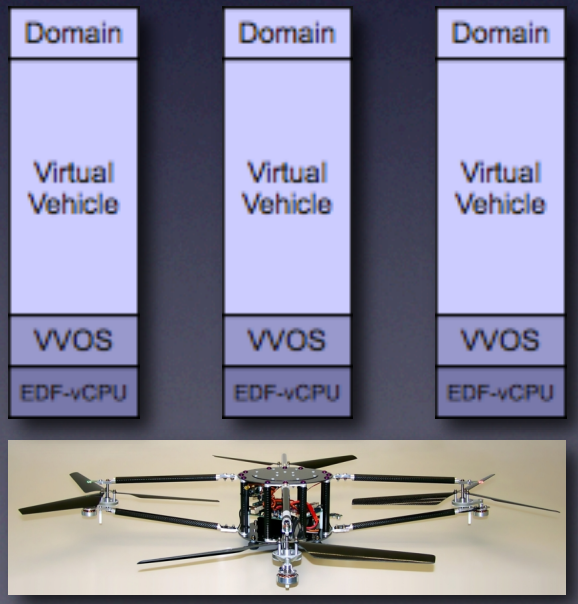
Control Software for systems with changing structure

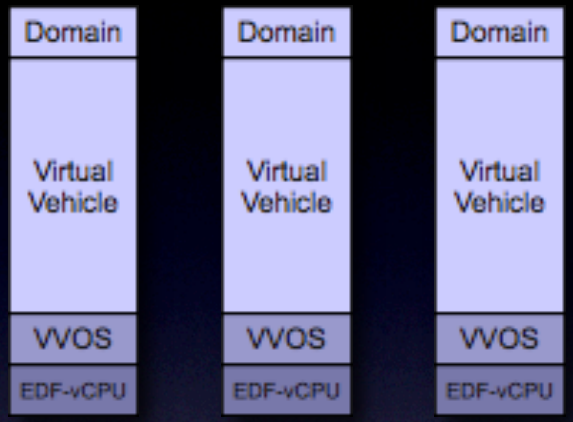
- Programming abstractions – Mainstream languages provide many abstractions
  - Syntaxes? Calculi?
- Verification – Can systems with changing structure be verified?
- Can we have a Control Theory for systems with changing dimension?
  - Asymptotics - Flocking, graph labplacians, switched systems, string stability, ….
  - Finite-time?
    - Safety
    - Can these open systems have notions of finite-time performance?
      - Phase transitions
- Run-time substrates, Real-time substrates
  - Virtualization?, Model Integrated Computing?

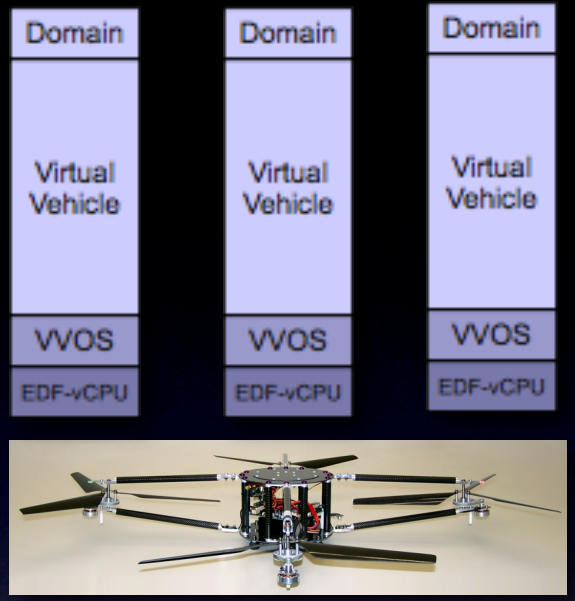




A Cyber-Physical Cloud

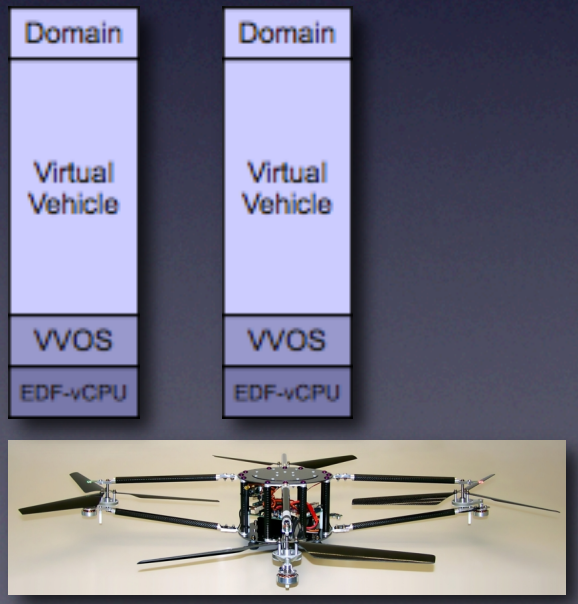






migration
=
flying

A Cyber-Physical Cloud



END