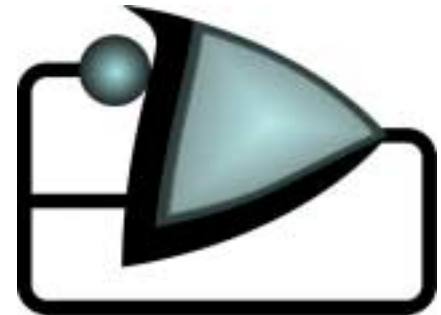


Reliability in highly-interconnected systems

Jason Hickey

Computer Science



Complexity in distributed systems

- The Internet has grown to over 100 million hosts; even more Internet appliances
- High connectivity, highly interdependent



Lucent Network Genome

- Design based on principles from the 70s
 - Individual nodes are powerful
 - Resources are tied to a physical location
 - Limited security and reliability

Formal design



- Use *math* to guide the design process
- Increases confidence:
 - **reliability**: we know what a system is supposed to do, and it does it
 - tools: specifications, code, and verification
- Adds automation:
 - code analysis, **synthesis**, and **optimization**
 - interactive design assistance
- High-confidence design requires systematic and structured approaches at all time scales (run time, design time)

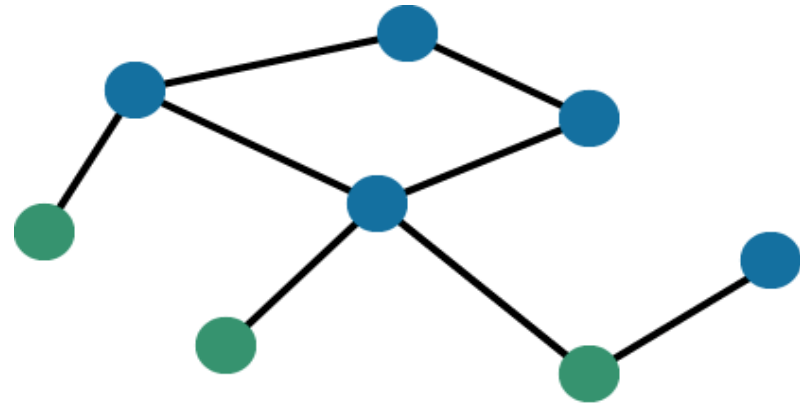
Logical Programming Environments



- Automate design for reliability
- A LPE includes:
 - A **logical library** where programs, proofs, and reasoning tools can be stored and shared in a collaborative development
 - A **formal compiler** that provides an open platform for producing executable code from programs, specifications, and proofs
 - An **automated reasoning system** that is used to develop formal proofs that programs meet their specification

An example: consensus in a network

- Goal: nodes must *agree*
- *Group communication*
 - Like a multi-point version of TCP
 - Communication is reliable



- Used in NY, Swiss stock exchanges
- French air-traffic control
- Navy's AEGIS command, control

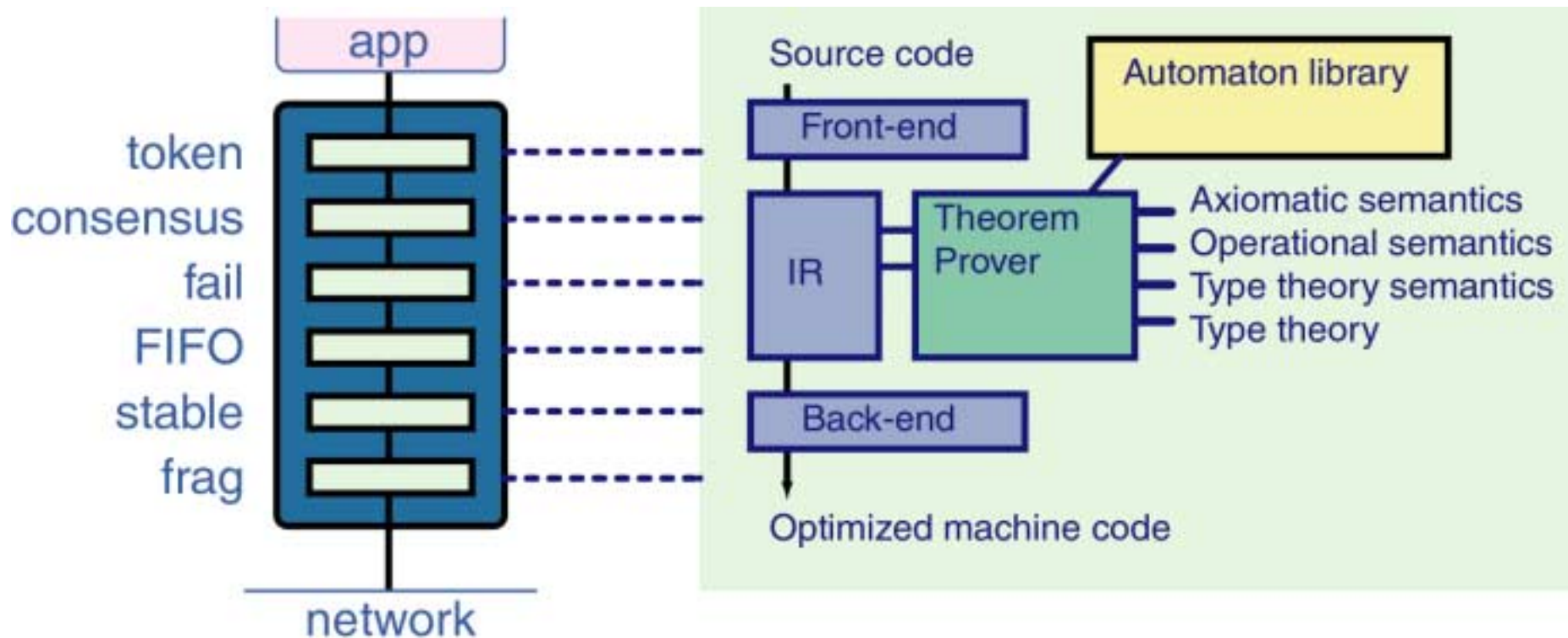
Step 1: compositional design models



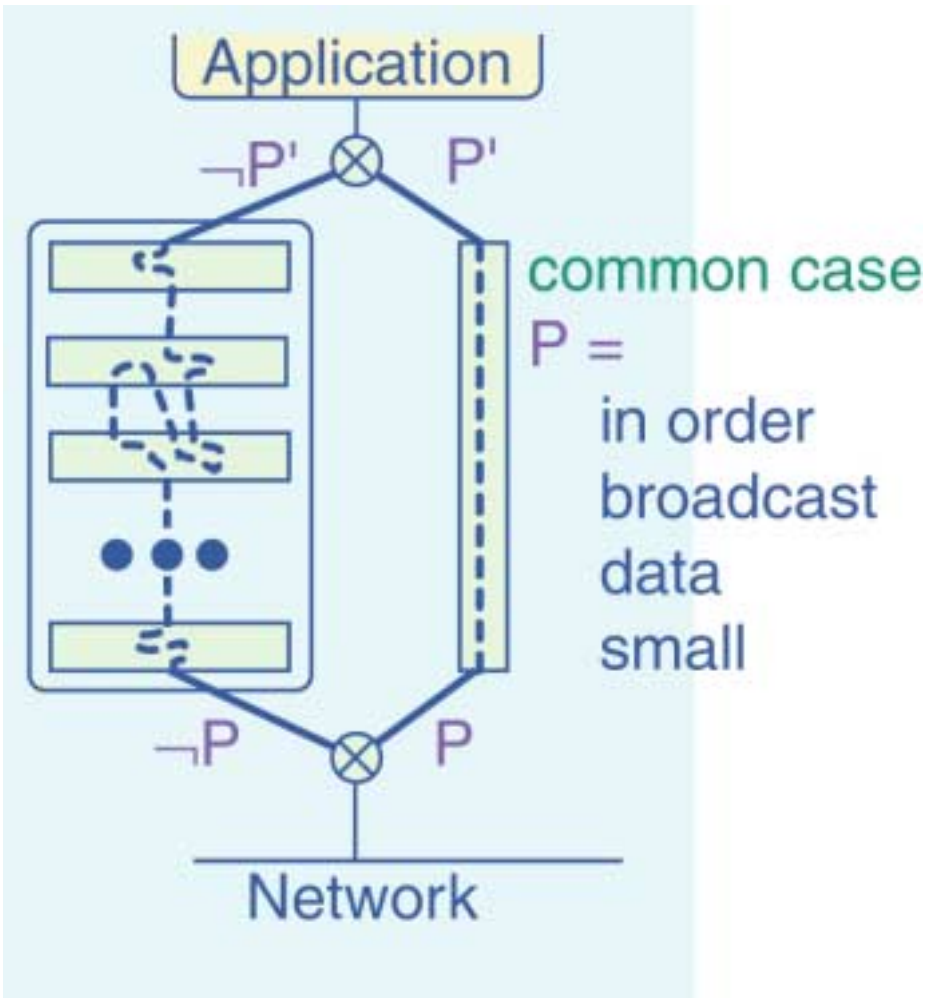
- Modular “pluggable” components
- Compositionality guaranteed by the model

Step 2: characterize components

- Nuprl Logical Programming Environment
- All properties (and meta-proofs of algebra) are formal



Step 3: assemble and optimize



- Protocols are pluggable components
- Protocol layers are in ML
- ~70 components, 1000s protocols
- About 30 layers in a protocol; roughly 300 lines of ML each
- Use refinement to verify/synthesize ML code

Problems and paths



- Formal methods provide tools
 - *models* to help understand problems
 - *compilers* to automatically generate code
 - *guarantees* about reliability
- But, the tools are hard to use
 - Deep knowledge of logic and semantics
- How do we apply the knowledge?
- How do we provide a migration path?

FC: a “functional” C compiler



- C programs are ubiquitous
 - But they are poorly understood
 - They have weak properties
 - Not compositional
- But they are everywhere
- FC: provide a formal foundation for C programs

Migration path for legacy code: FC

- Import C programs into a high-confidence, formal environment
- Allow *all* C programs
 - pointer arithmetic
 - arbitrary coercions
- Map to a **safe-**functional language
- Add: **transactions, migration**

```
e ::=f  let v : t f a in e
      j  let v f s in e
      j  let v : t f unop a in e
      j  let v : t f a binop a in e
      j  let type typdefs in e
      j  let fun fundefs in e
      j  let v : t f f „a1;:::;an...in e
      j  let closure v : t f f „a1;:::;an...in e
      j  let external v f „f : ty ...,q;:::;an...in e
      j  f „a1;:::;an...
      j  internal f „a1;:::;an...
      j  if a1 relop a2 then e1 else e2
```

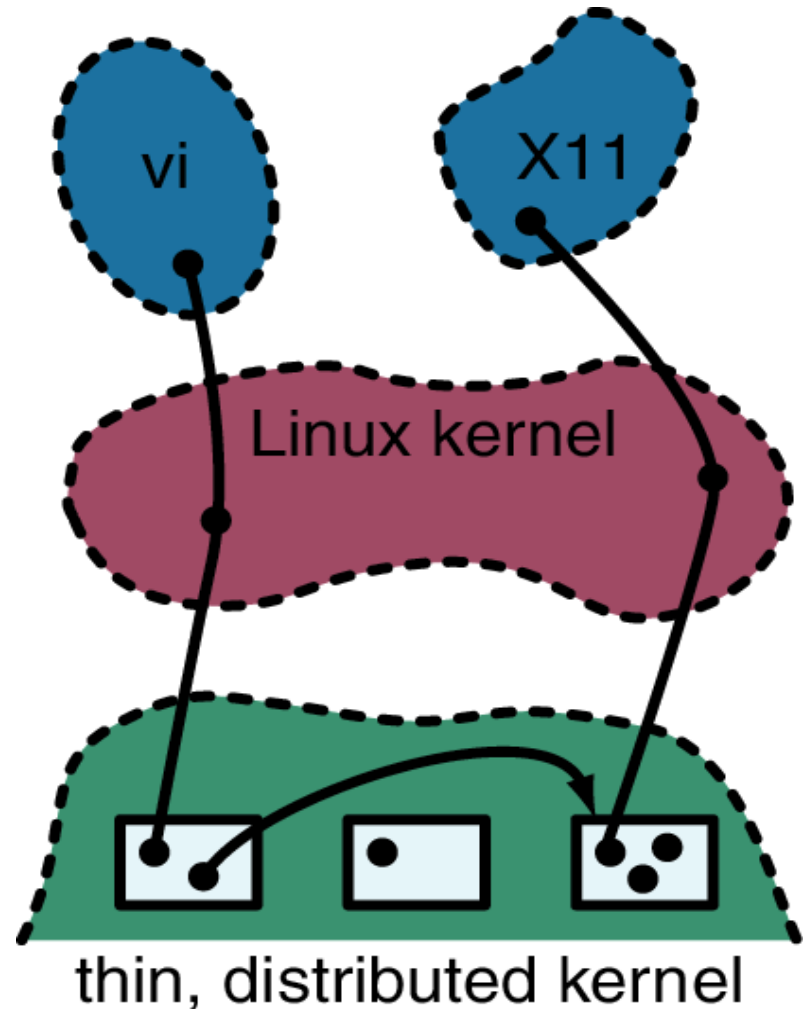
Extensions



- Programs are *safe*
 - No program accesses memory that it does not own,
 - No program executes code that it does not own
 - (Programs may still self-destruct)
- C allows
 - arbitrary coercion
 - pointer arithmetic
- The critical step is to introduce checking (some of it at run time)

Distributed operating systems

- Attack the problem of distributed, reliable operating systems
- **Safety** means there is no need for the kernel-user distinction
- OS can be stripped down all the way to the hardware
- Key tools:
 - process migration
 - distributed **atomic** operations



Testbed: cooperative control



- Systems must deliver even in the presence of uncertainty
- A UAV flock must be robust to changes in the environment, component failures, communication failures, as well as loss of entire vehicles
- Requires expertise both in controls and distributed systems

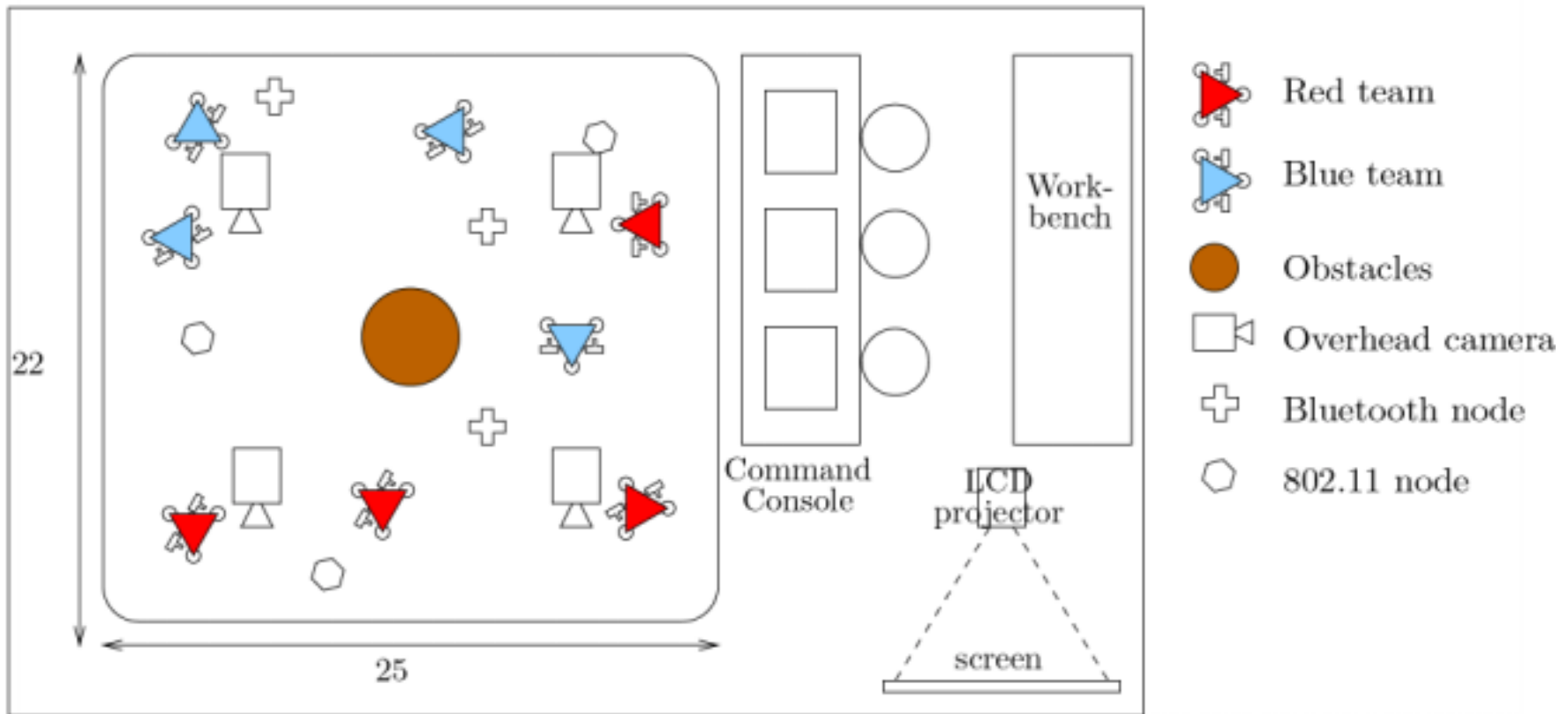
Multi-vehicle wireless testbed

- 8-10 vehicles, integrated computing and communications, including wireless Ethernet (802.11), and Bluetooth
- 2-4 fixed communication nodes, capable of broadcasting on multiple channels
- A set of overhead cameras that can be used to provide position information to the vehicles (perhaps simulating GPS)

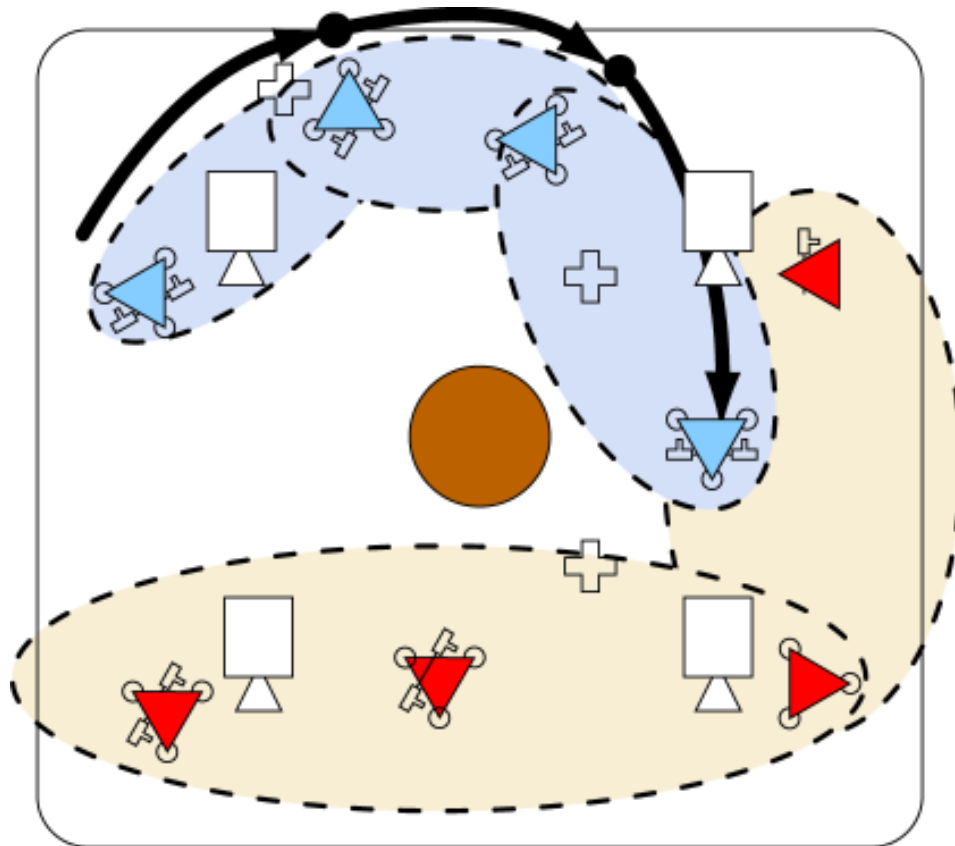


- A command console with computing and communication nodes

Multi-vehicle wireless testbed

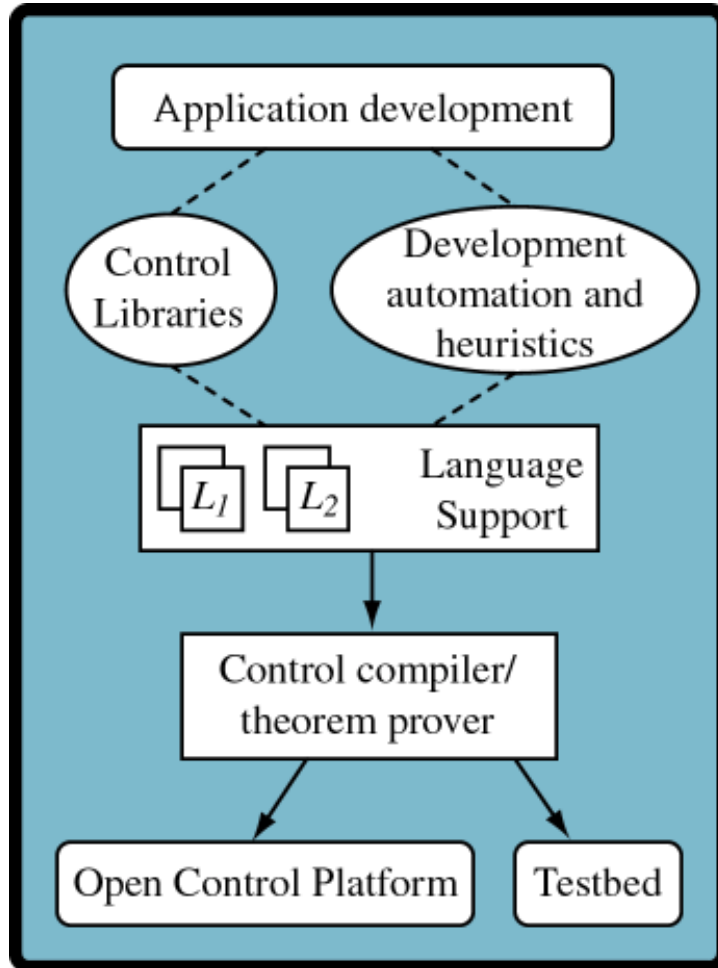


Multi-vehicle routing



- Network topology is rapidly changing
 - Consensus
 - Message routing
 - Real-time prioritized traffic
 - Make use of topology predictions

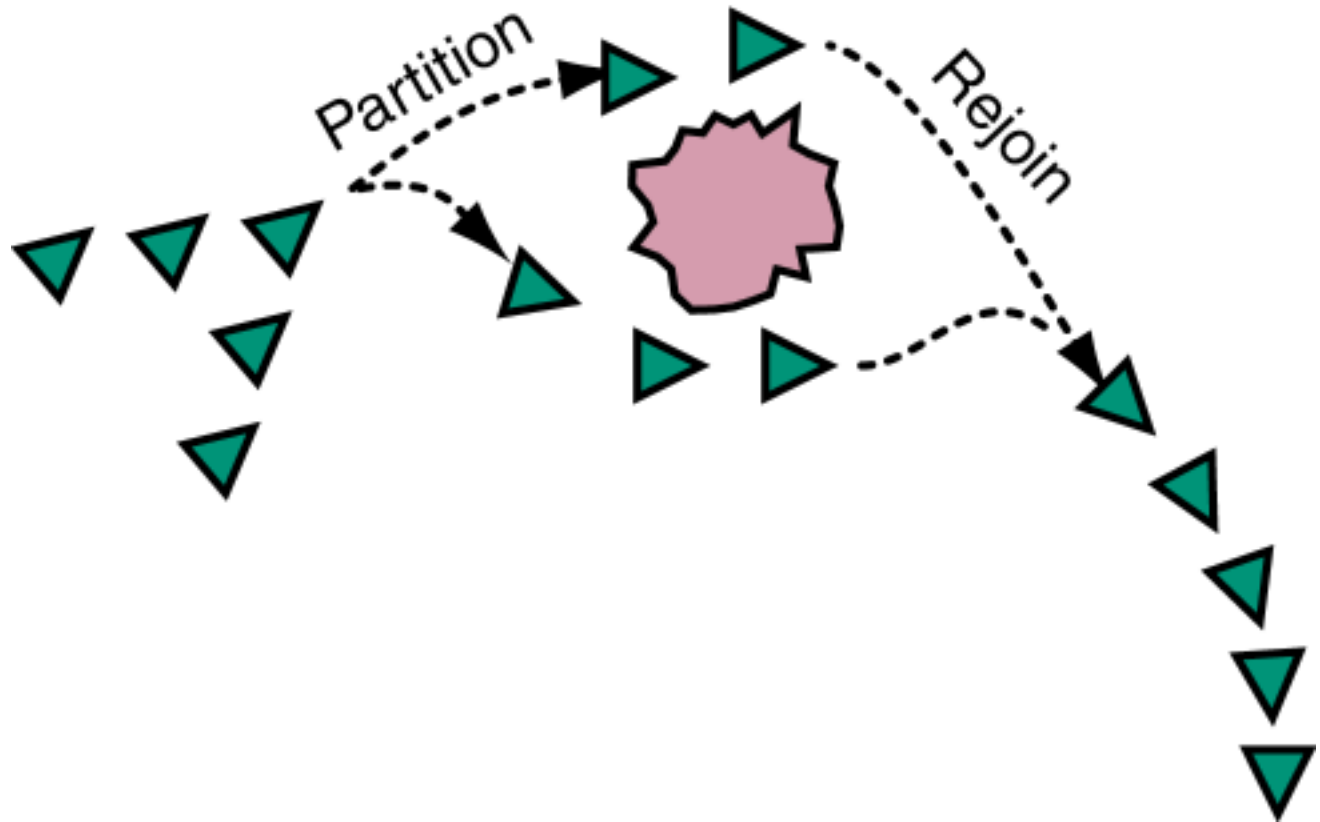
Logical Programming Environment



- The LPE is a framework for supporting formal design
 - *Type theory* is a common language for specification and synthesis
 - Enables *collaborative* development of verified control libraries and design automation tools
 - The *compiler* is an assistant, and the link to executable code

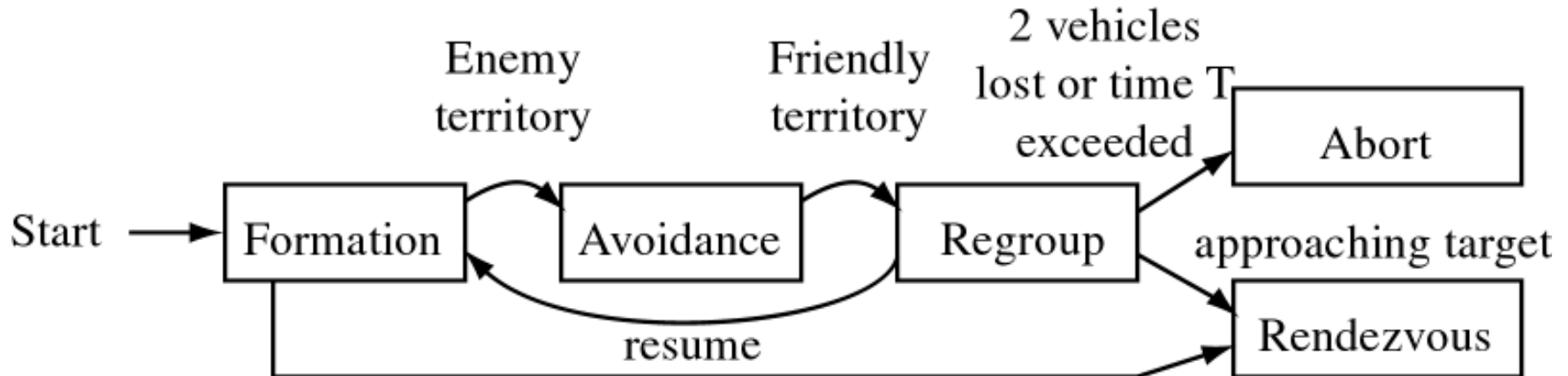
Problem formulation for UAV

- Formalize a rejoin



Flow-chart design style

- Second-level: specify computation as a reactive state machine
- Verify that the decomposition satisfies the spec



Summary



- Link together all aspects of reliable networked systems
- Design with *logical programming environments*
 - Libraries capture design knowledge
 - Compiler provides design automation
 - Prover provides system guarantees
- Provide a migration path for existing code base (FC)
- Use formal tools to create high-confidence distributed operating systems
- Demonstrate with high-confidence coordinated control systems