Problem Frame Transformations:
Deriving Specifications from Requirements

Robert Seater
Daniel Jackson

Software Design Group
Massachusetts Institute of Technology

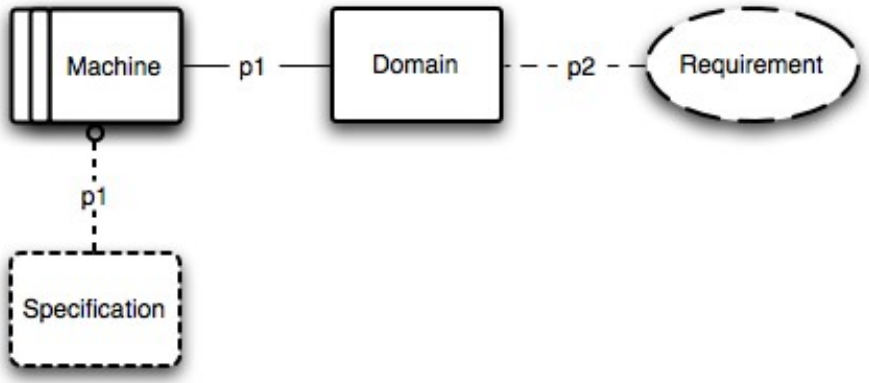
May 23rd, 2006
2nd International Workshop on Applications and Advances in Problem Frames
(IWAAPF'06, part of ICSE'06)

History

- tool for understanding proton therapy machine
- build-then-analyze vs. design-then-build
- local reasoning (local understanding)
- show history (tracability, communication)

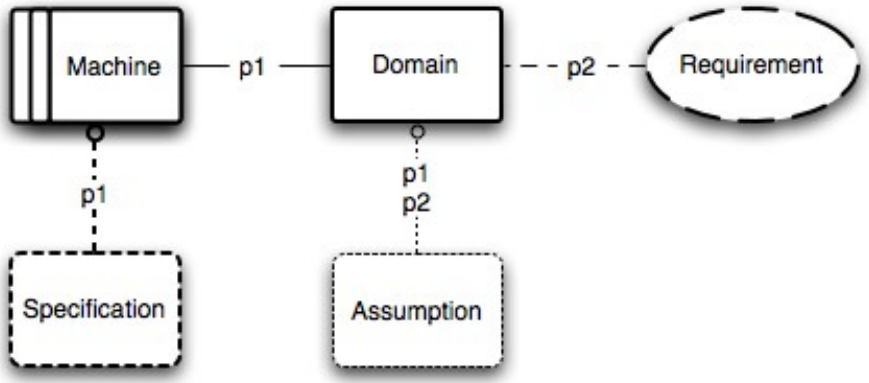
Requirements & Specifications

- does the spec enforce the requirement?



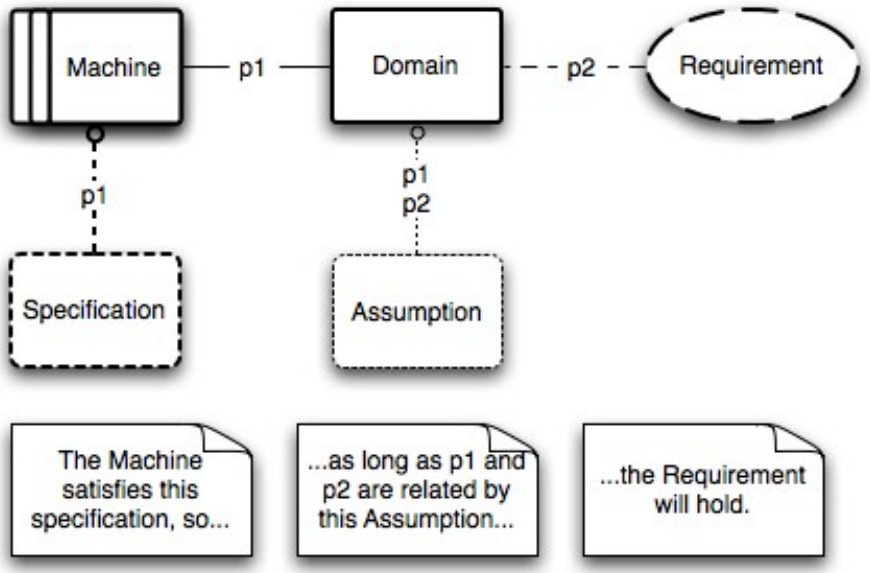
Requirements & Specifications

- does the spec enforce the requirement?
- relies on **domain assumptions**



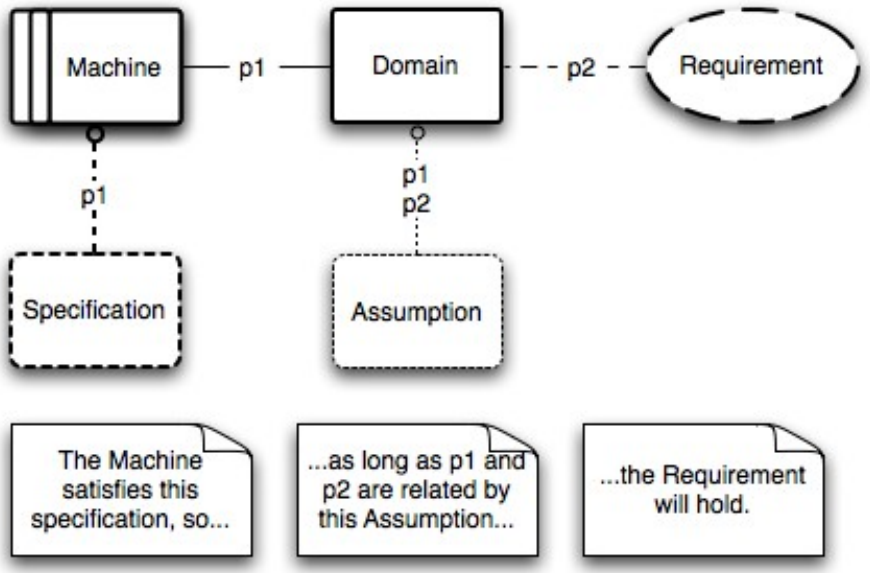
Requirements & Specifications

- does the spec enforce the requirement?
- relies on **domain assumptions**
- conventional solution: catalogue of **frame concerns** derived from prior experience
- template for correctness argument, list of relevant assumptions



Key Observations

- requirement is not a spec **only** because it references phenomena not controlled by the machine
- domain assumption justifies constraining p1 instead of p2
- can incrementally transform requirement into spec plus set of domain assumptions

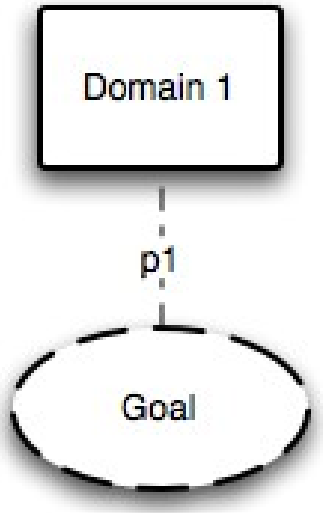


Transformation Toolkit

- **add** a breadcrumb
- **rephrase** the goal
- **push** an arc
- **split**/**merge** arcs
- **heuristic**: walk the requirement towards the machine

Transformation Toolkit

- **add** a breadcrumb
- **rephrase** the goal
- **push** an arc
- **split**/**merge** arcs
- **heuristic**: walk the requirement towards the machine

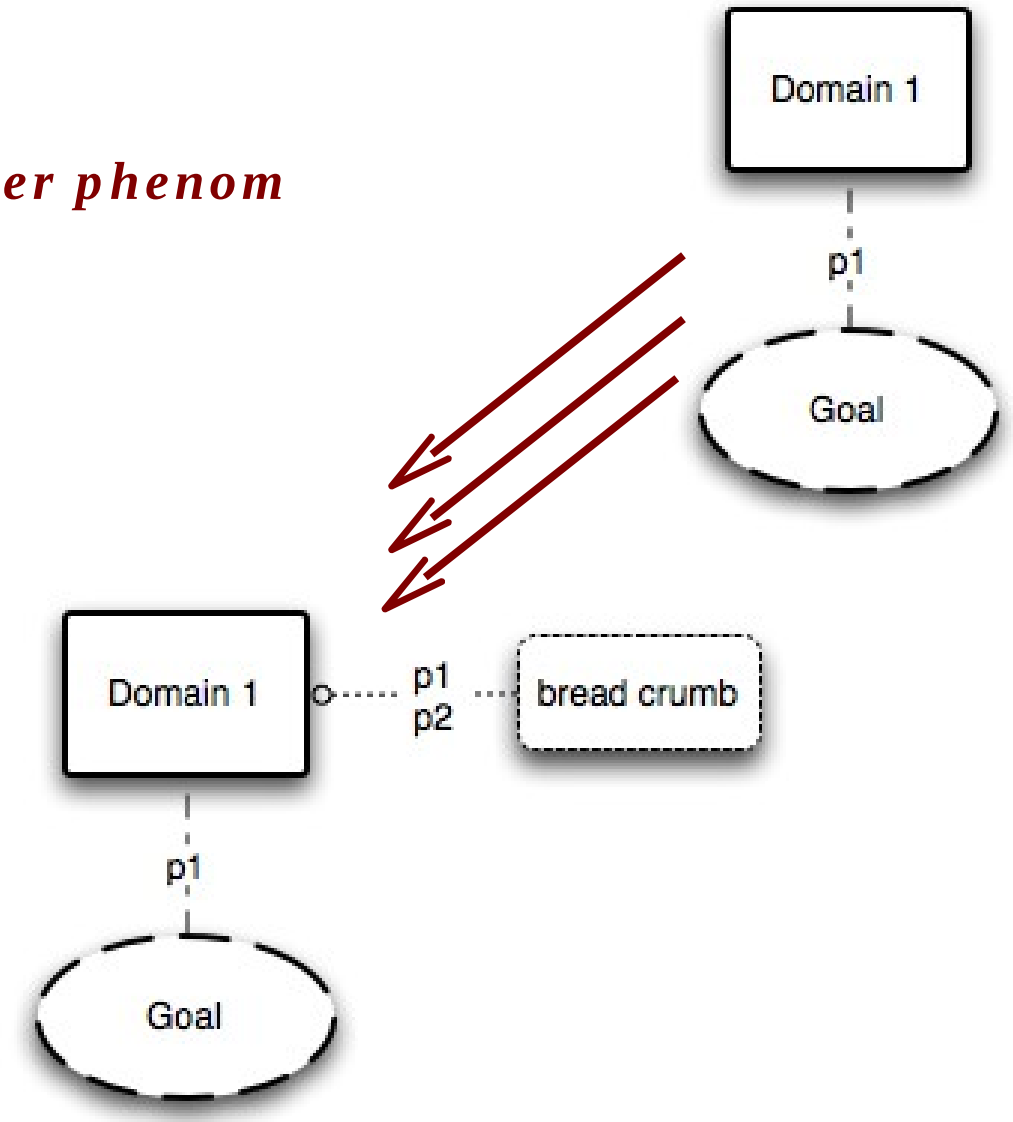


Transformation Toolkit

- **add** a breadcrumb

relate referenced phenom to other phenom

- **rephrase** the goal
- **push** an arc
- **split/merge** arcs
- **heuristic**: walk the requirement towards the machine



Transformation Toolkit

- **add** a breadcrumb

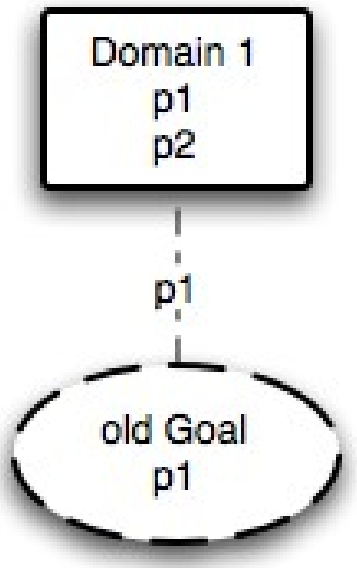
relate referenced phenom to other phenom

- **rephrase** the goal

- **push** an arc

- **split**/**merge** arcs

- **heuristic**: walk the requirement towards the machine



Transformation Toolkit

- **add** a breadcrumb

  relate referenced phenom to other phenom

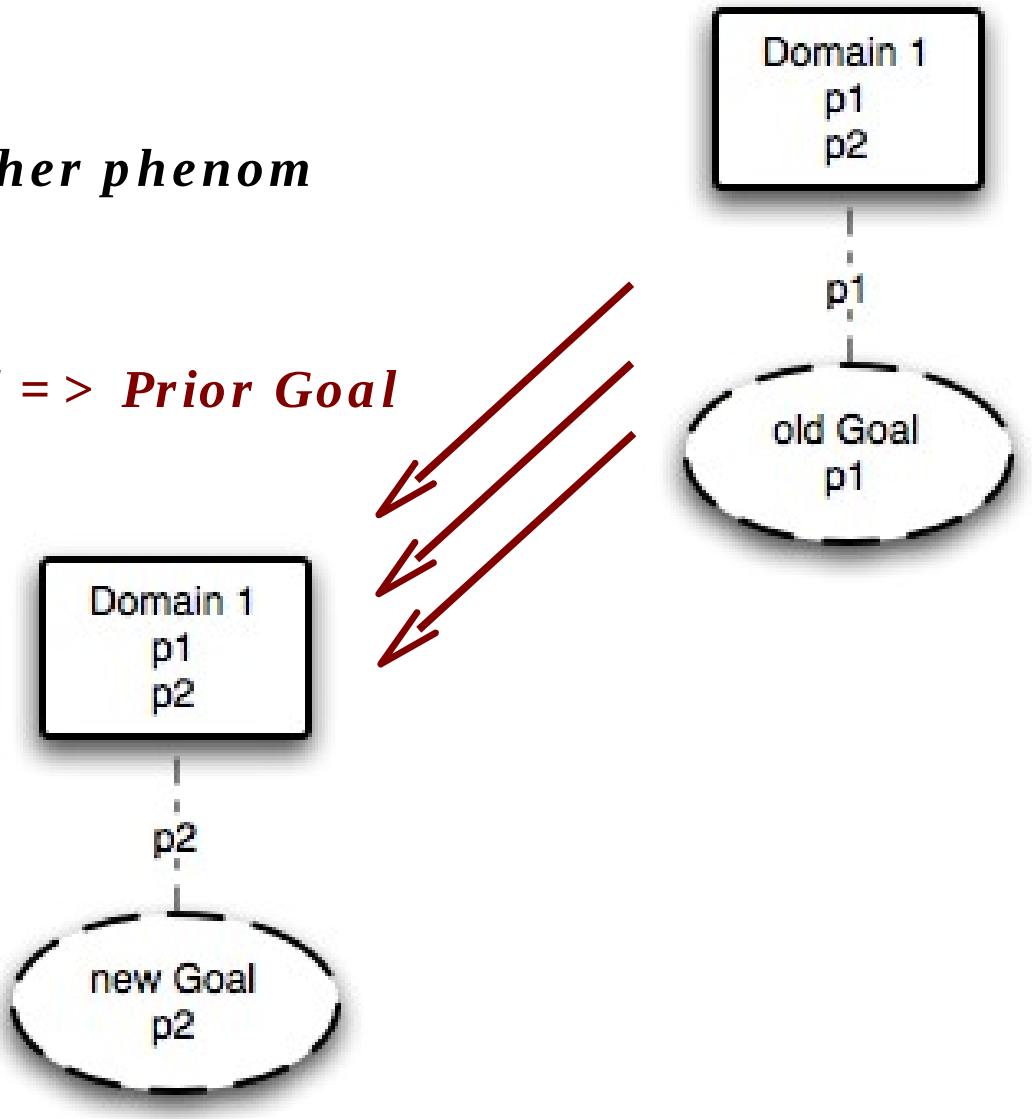
- **rephrase** the goal

  Breadcrumb ^ Rephrased Goal => Prior Goal

- **push** an arc

- **split/merge** arcs

- **heuristic**: walk the requirement towards the machine



Transformation Toolkit

- **add** a breadcrumb

relate referenced phenom to other phenom

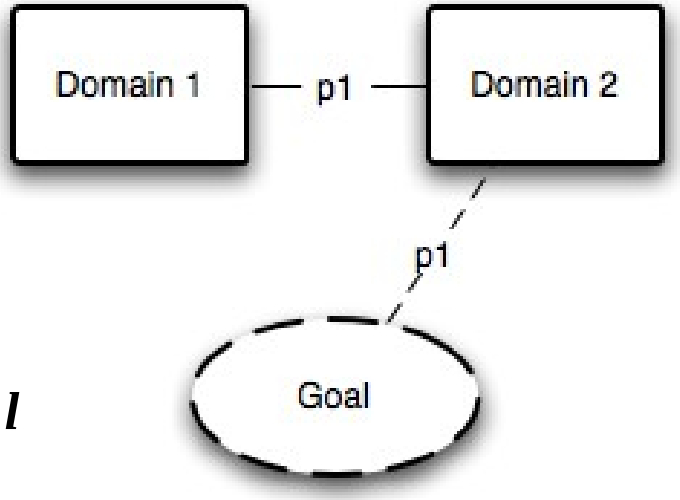
- **rephrase** the goal

Breadcrumb ^ Rephrased Goal => Prior Goal

- **push** an arc

- **split/merge** arcs

- **heuristic**: walk the requirement towards the machine



Transformation Toolkit

- **add** a breadcrumb

  relate referenced phenom to other phenom

- **rephrase** the goal

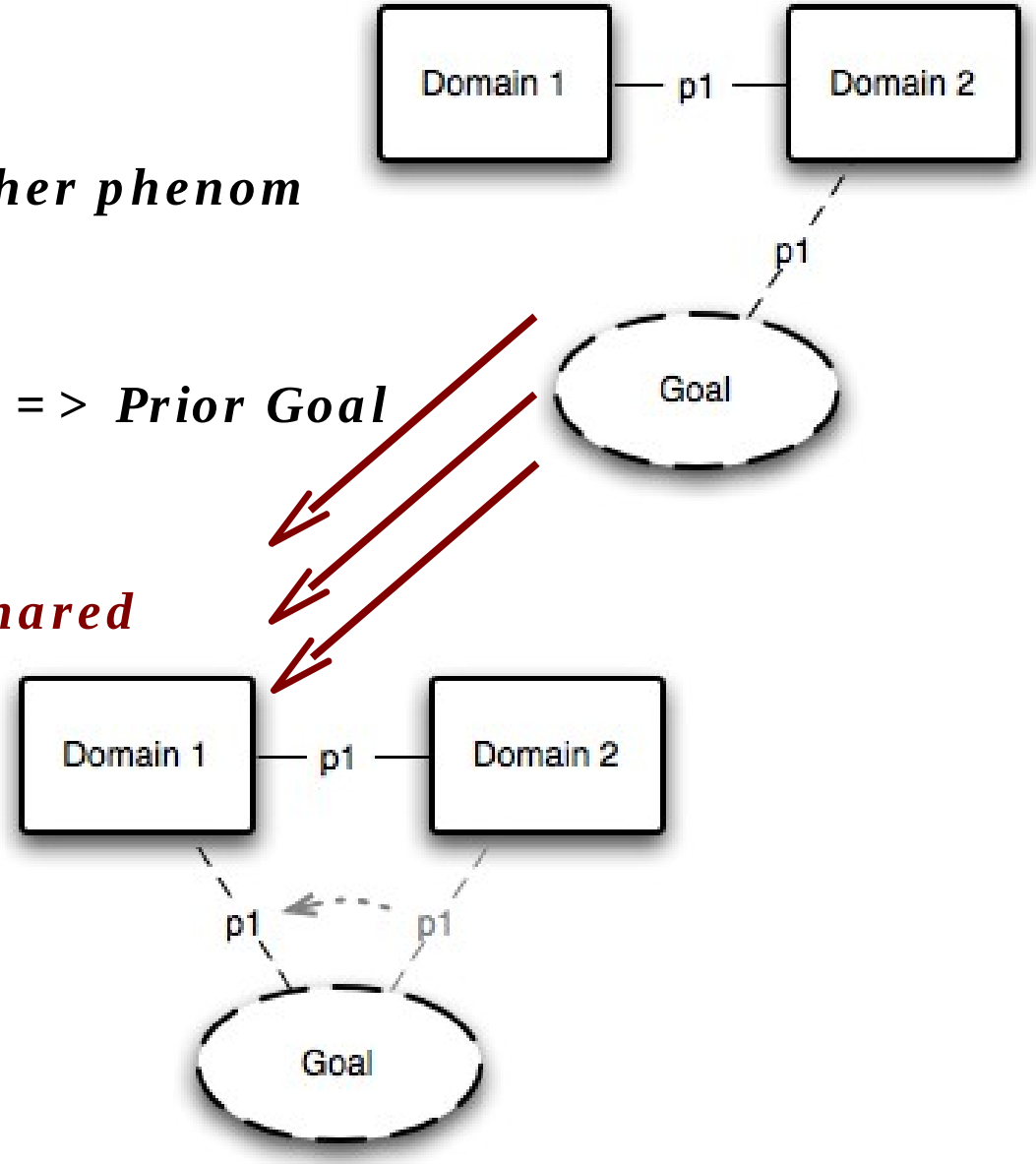
  Breadcrumb ^ Rephrased Goal => Prior Goal

- **push** an arc

  phenom on that arc must be shared

- **split/merge** arcs

- **heuristic**: walk the requirement towards the machine



Transformation Toolkit

- **add** a breadcrumb

  relate referenced phenom to other phenom

- **rephrase** the goal

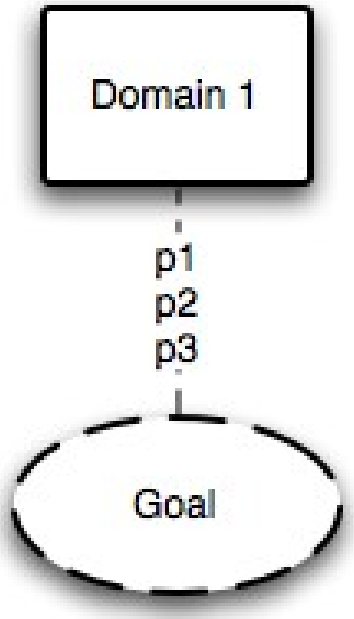
  Breadcrumb ^ Rephrased Goal => Prior Goal

- **push** an arc

  phenom on that arc must be shared

- **split/merge** arcs

- **heuristic**: walk the requirement towards the machine



Transformation Toolkit

- **add** a breadcrumb

  relate referenced phenom to other phenom

- **rephrase** the goal

  Breadcrumb ^ Rephrased Goal => Prior Goal

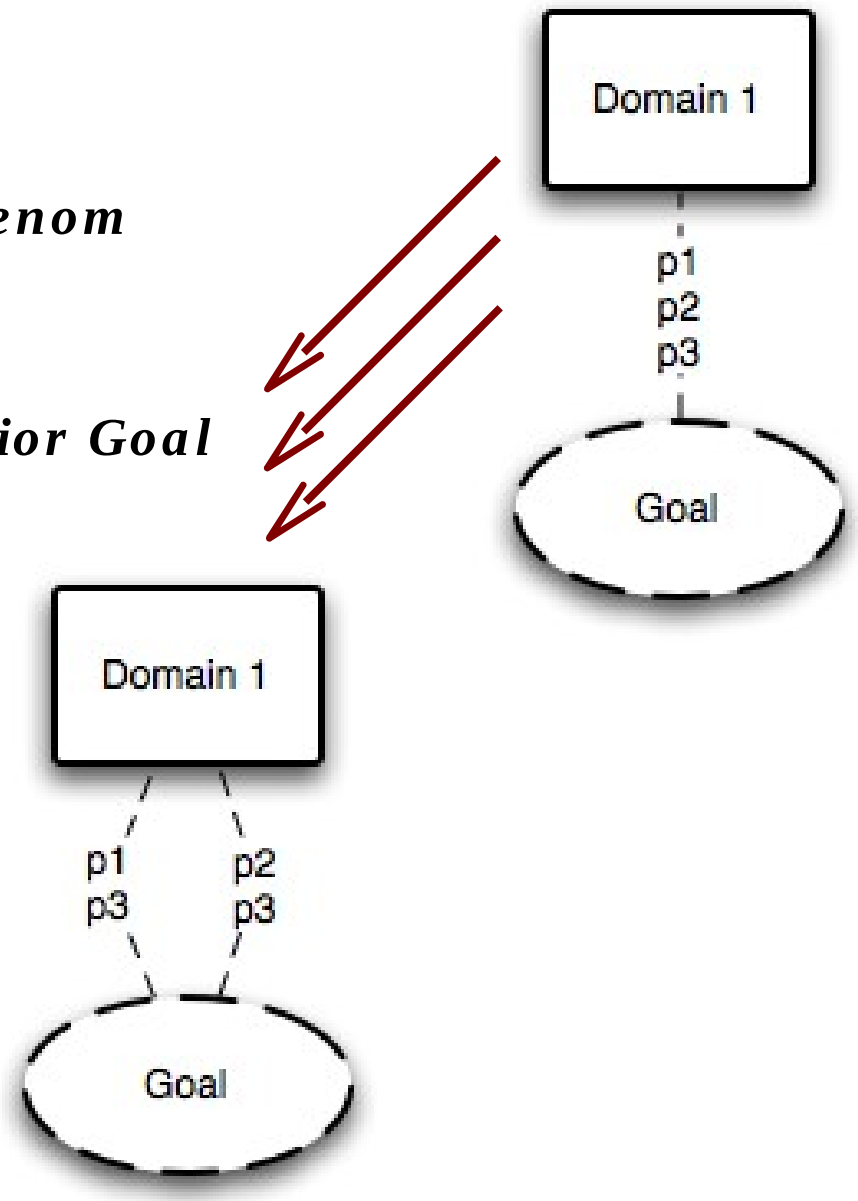
- **push** an arc

  phenom on that arc must be shared

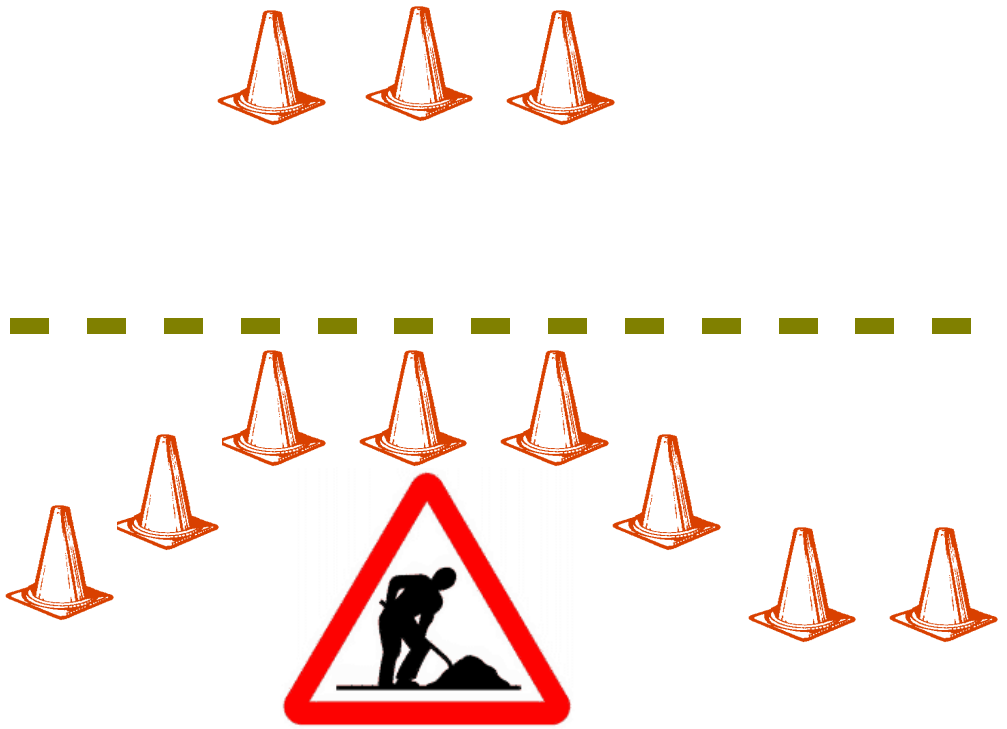
- **split/merge** arcs

  nothing else changes

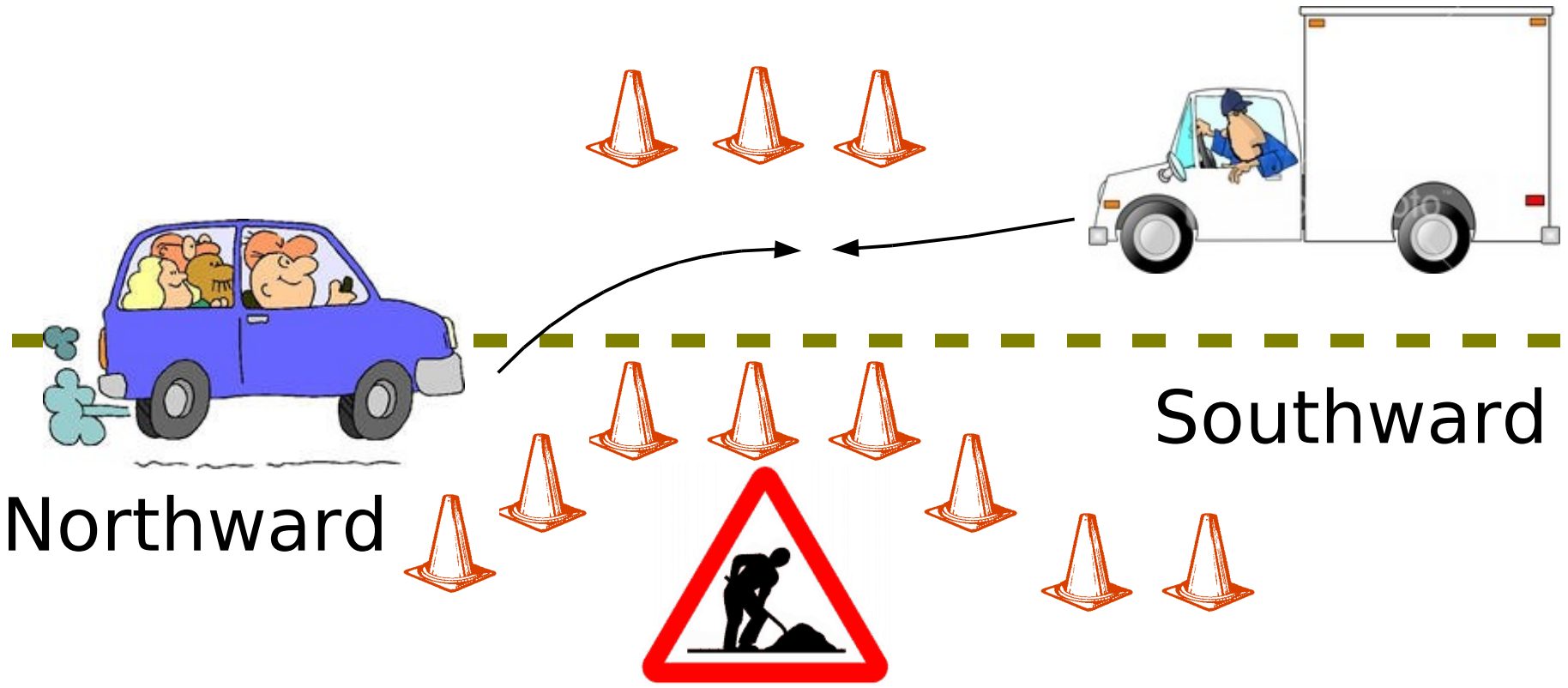
- **heuristic**: walk the requirement towards the machine



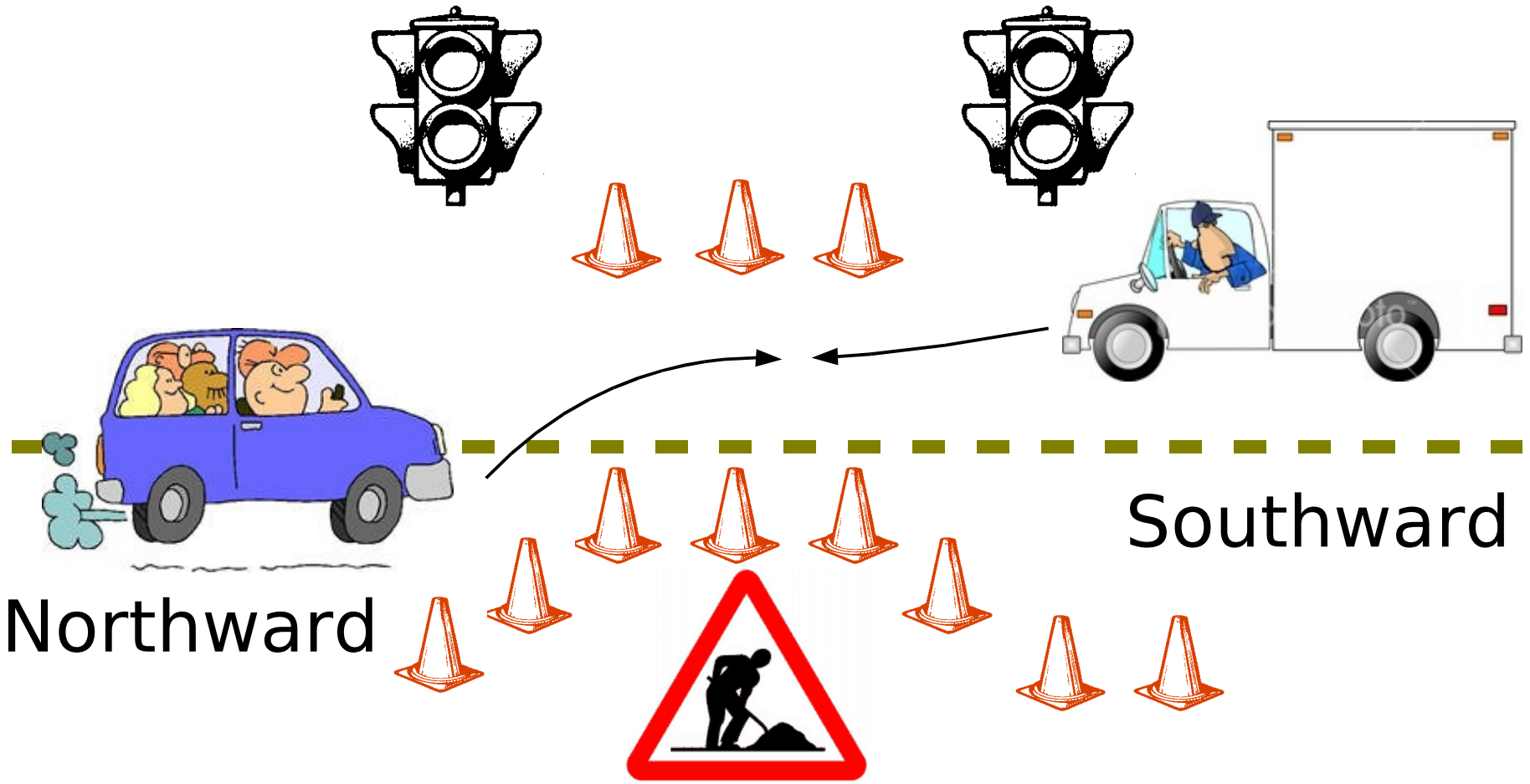
Two-Way Traffic Light



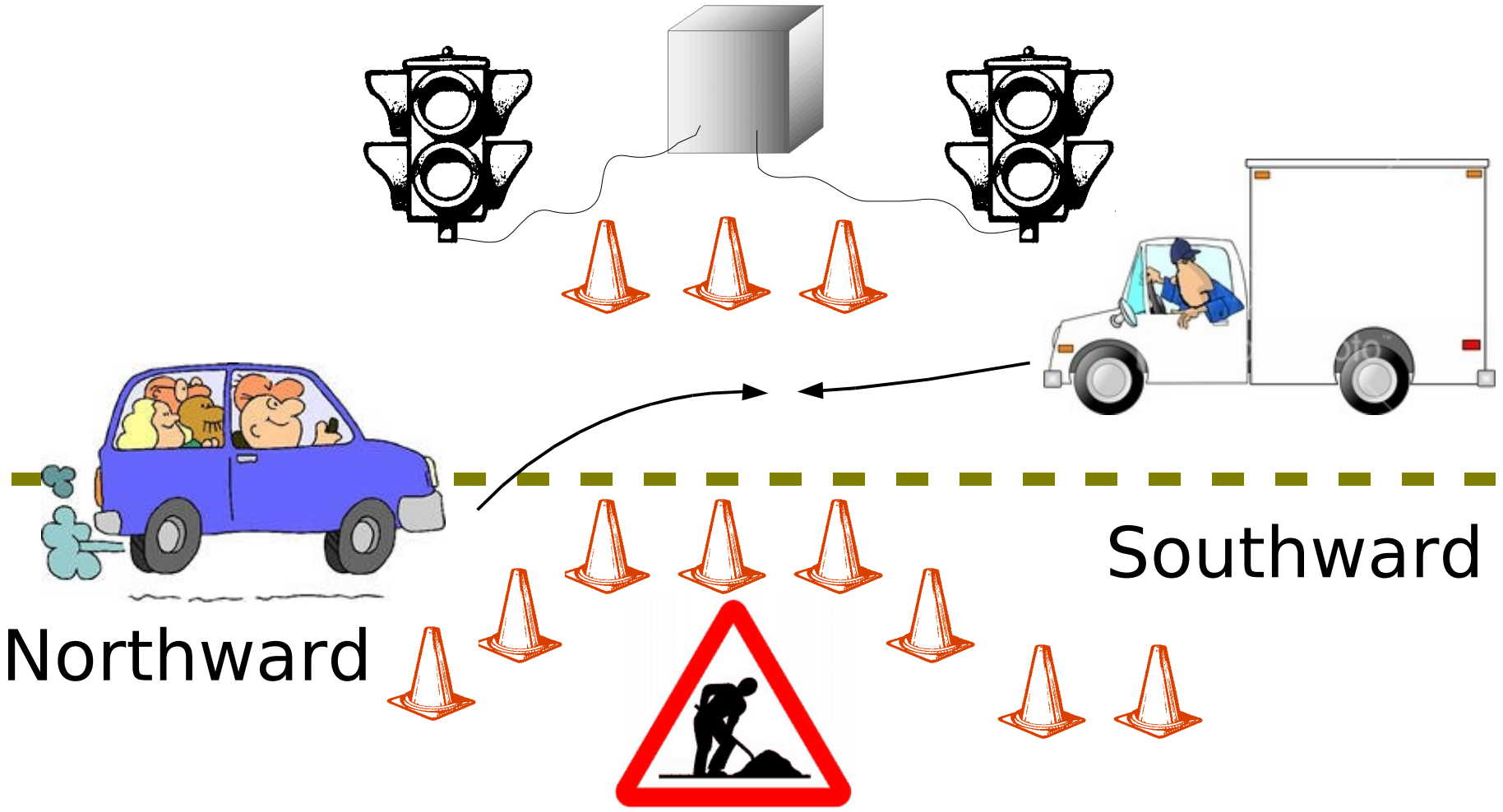
Two-Way Traffic Light



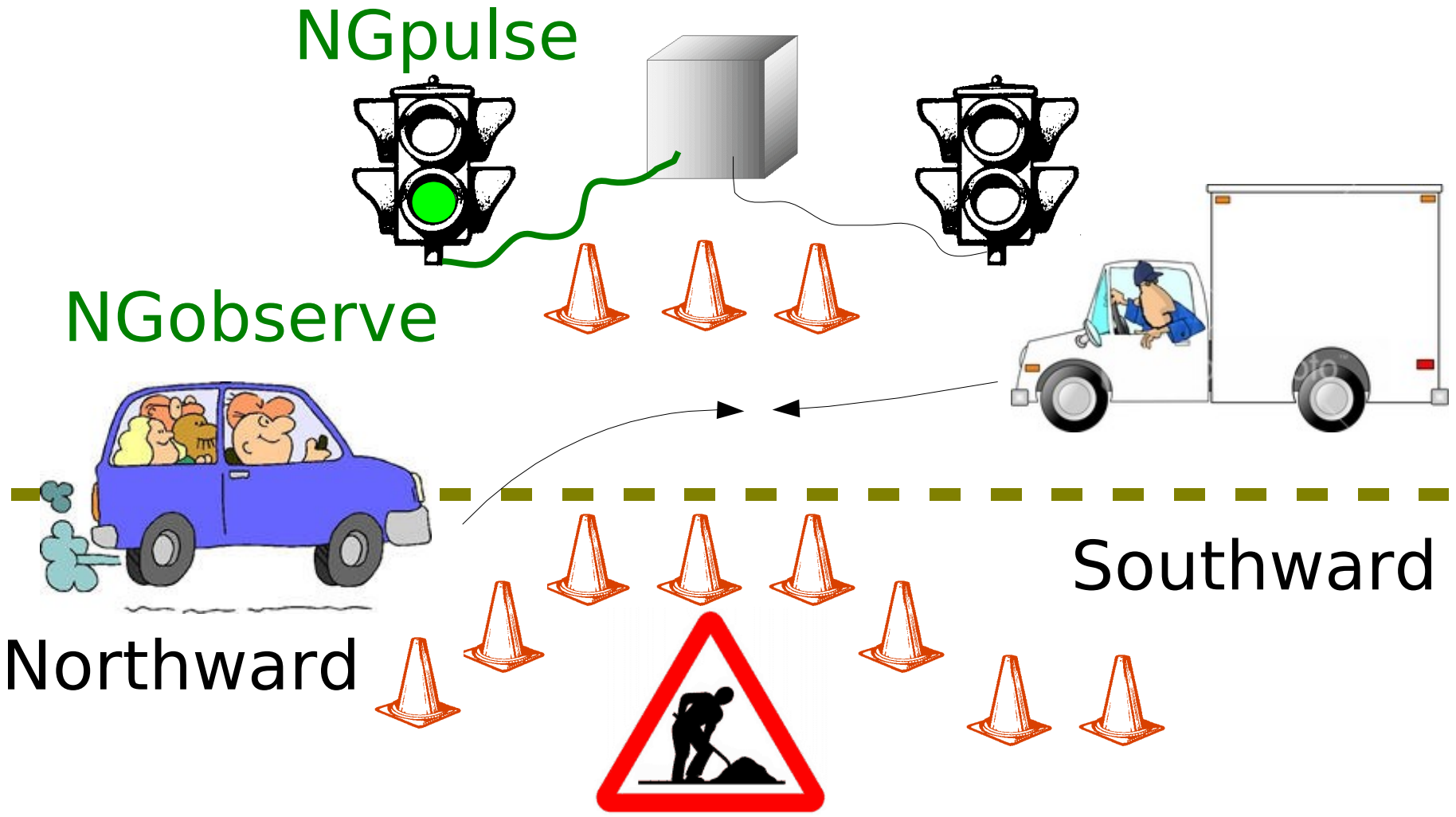
Two-Way Traffic Light



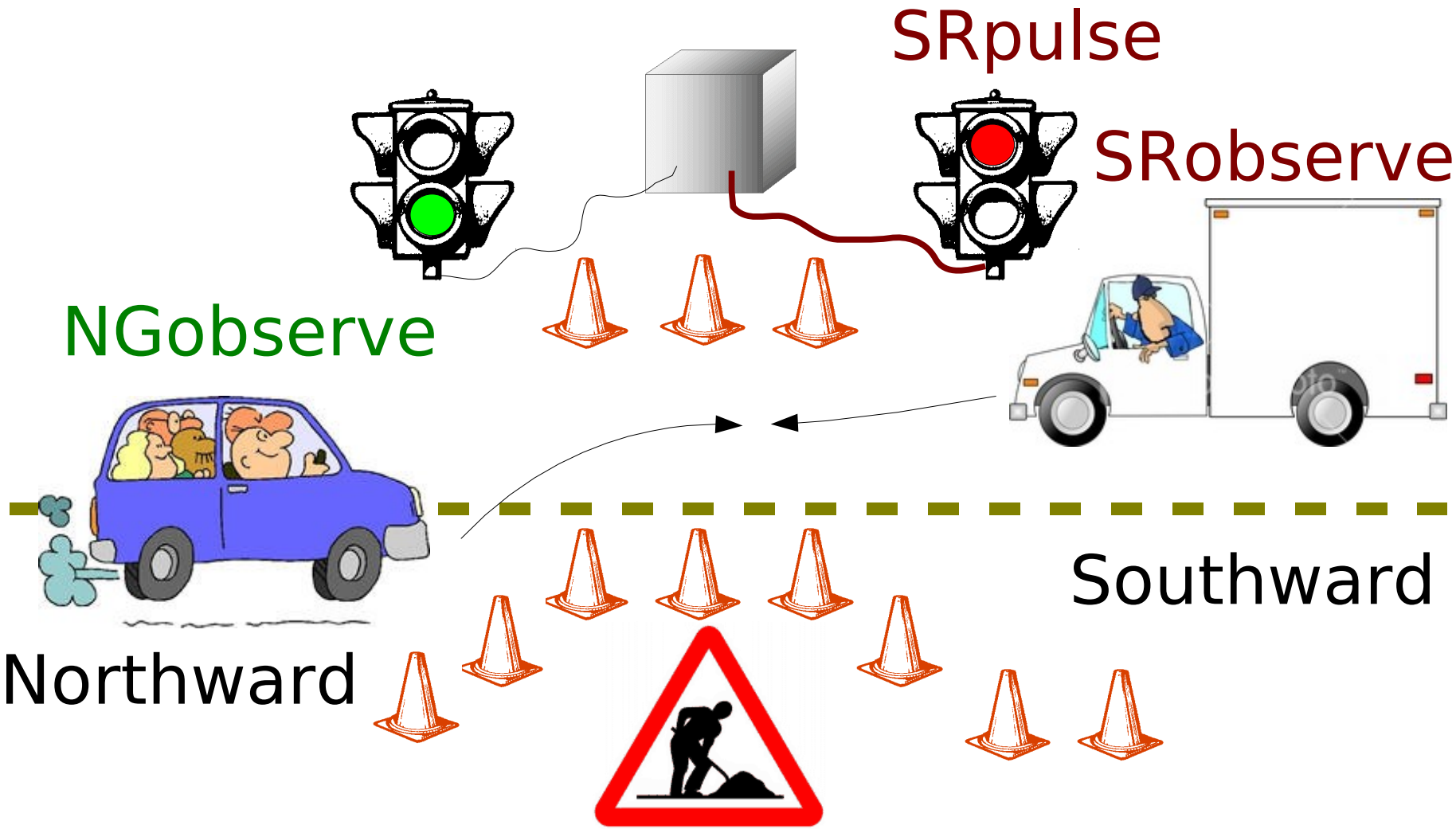
Two-Way Traffic Light



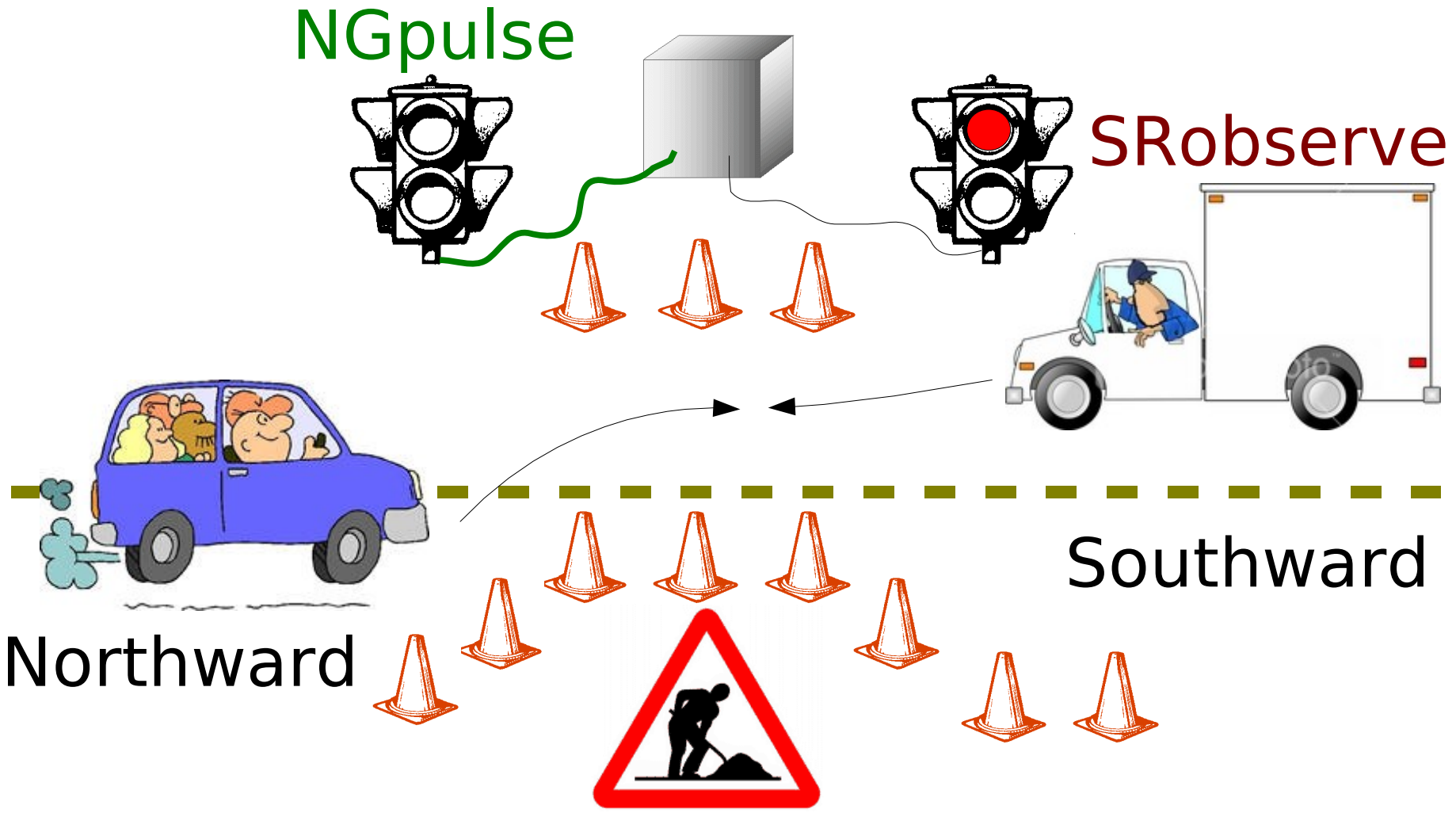
Two-Way Traffic Light



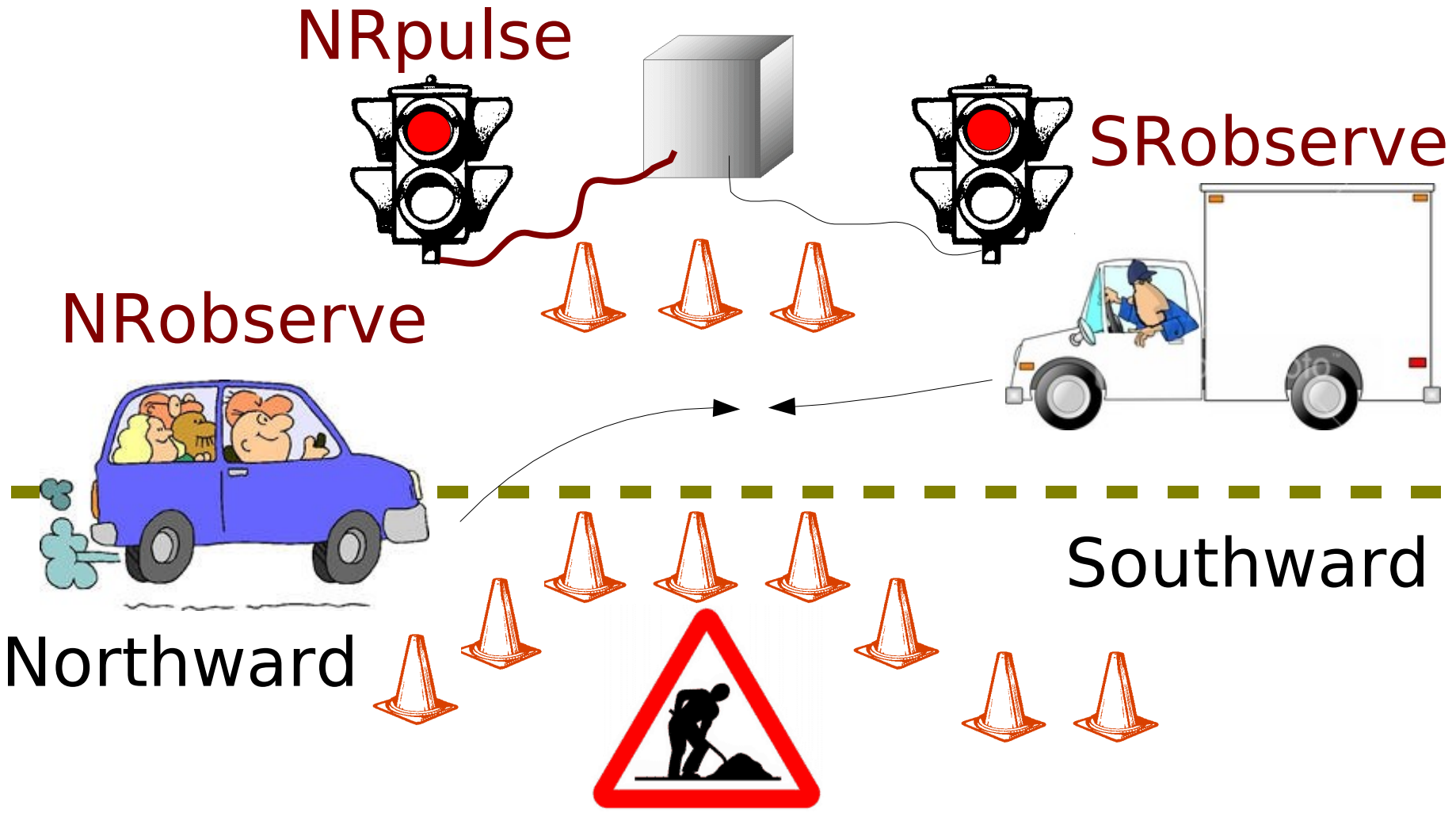
Two-Way Traffic Light



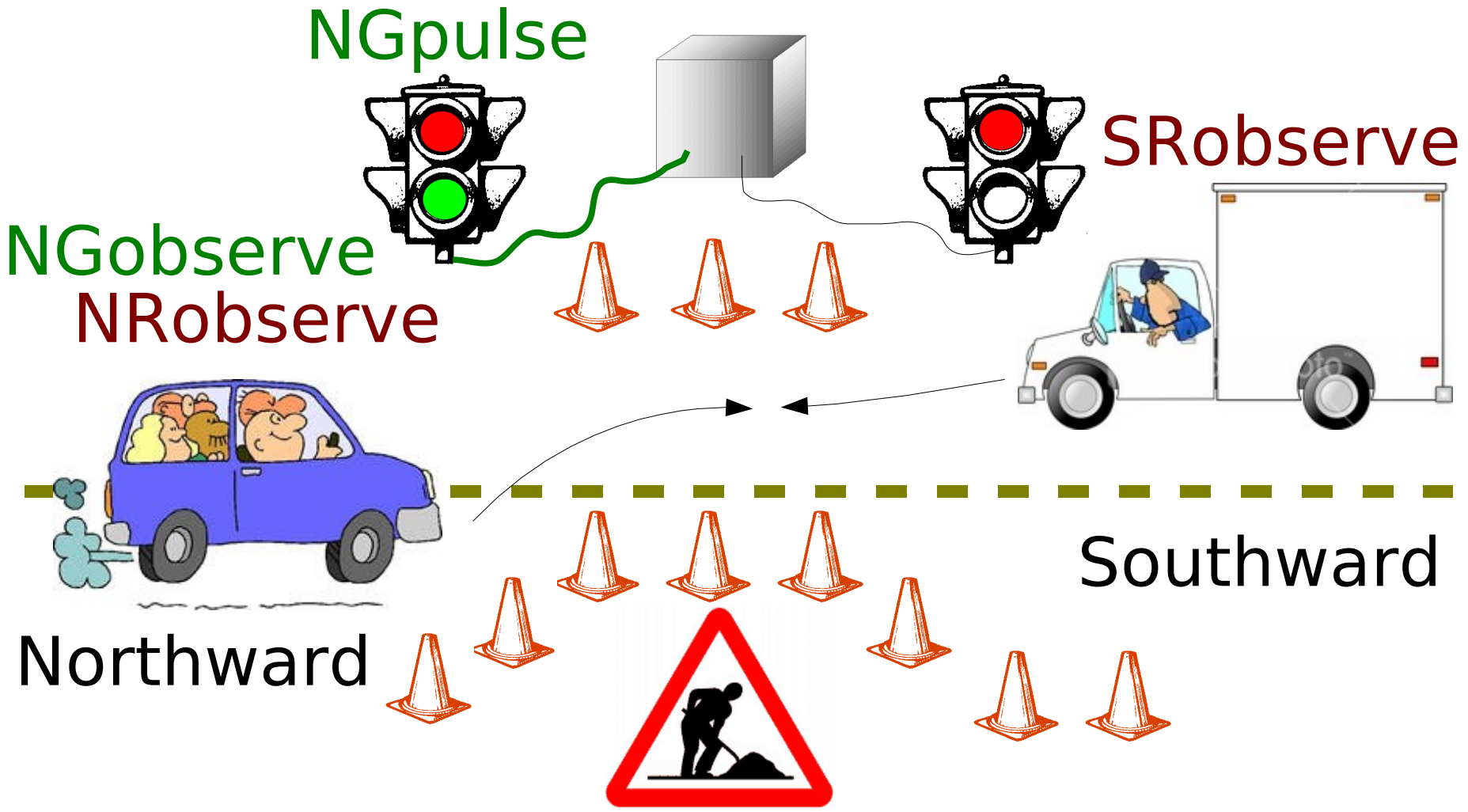
Two-Way Traffic Light



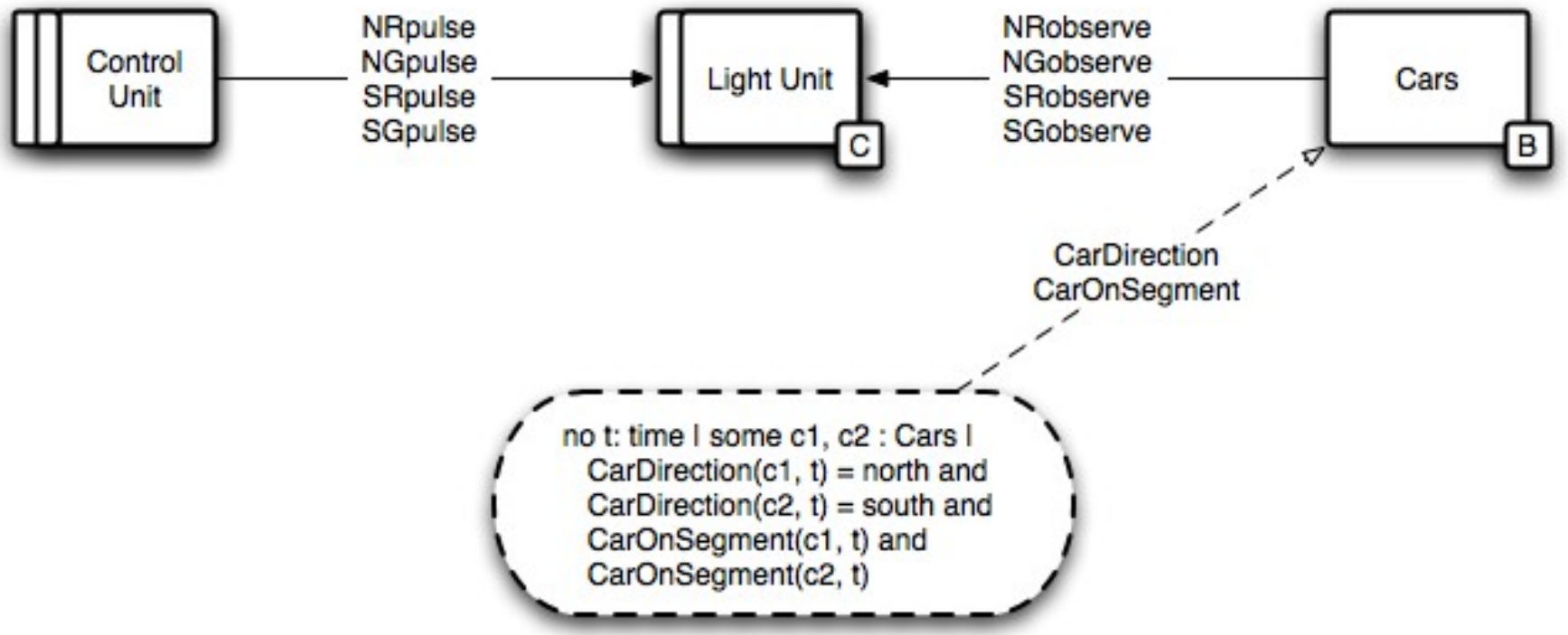
Two-Way Traffic Light



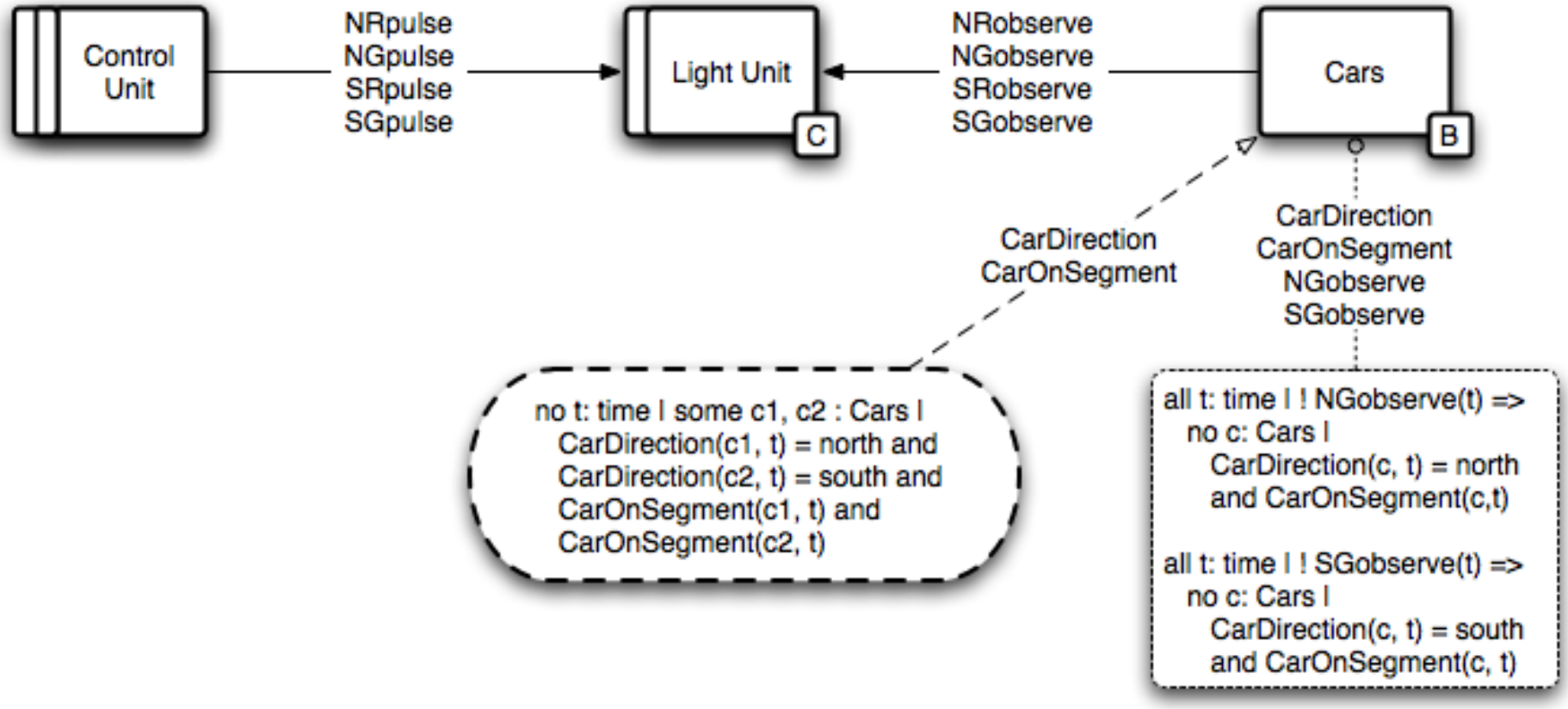
Two-Way Traffic Light



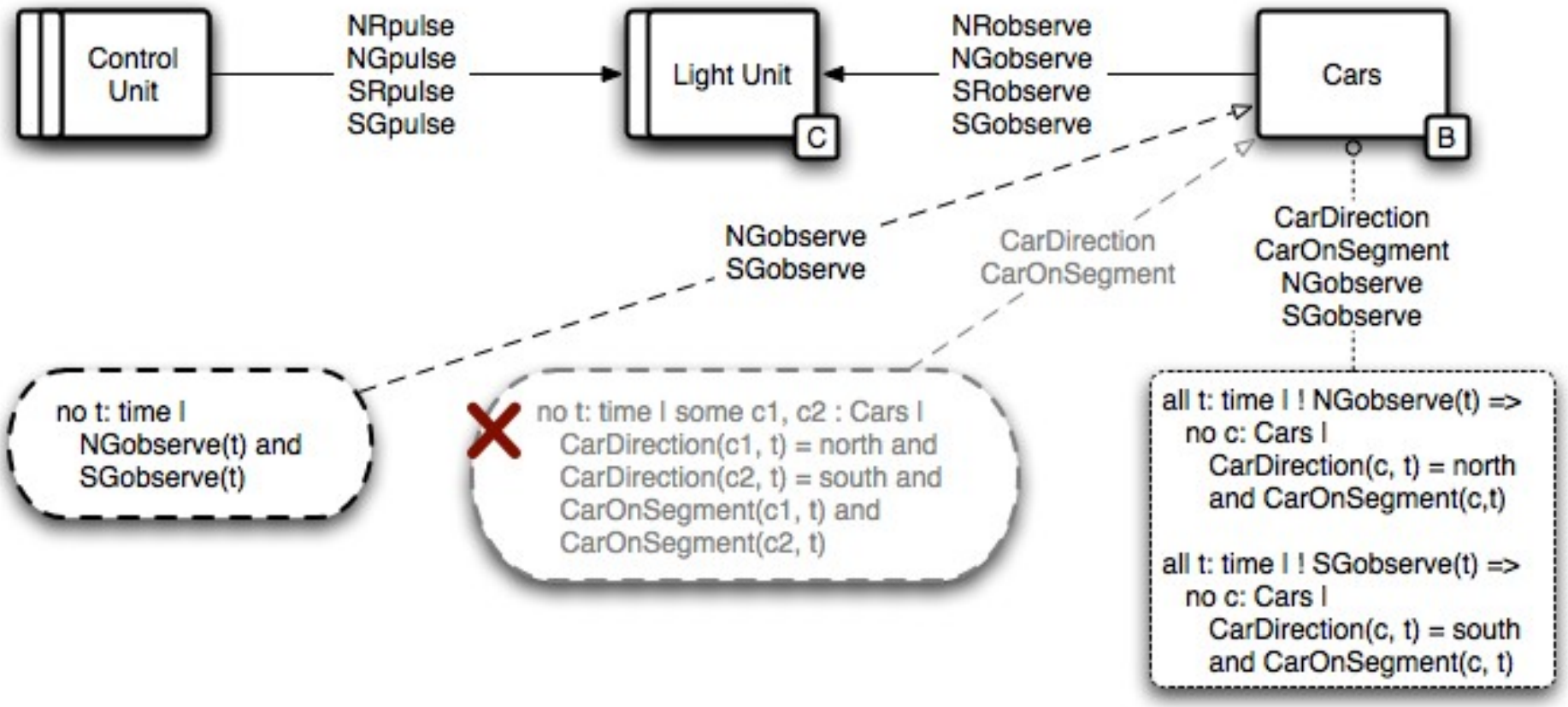
Problem Frame Description



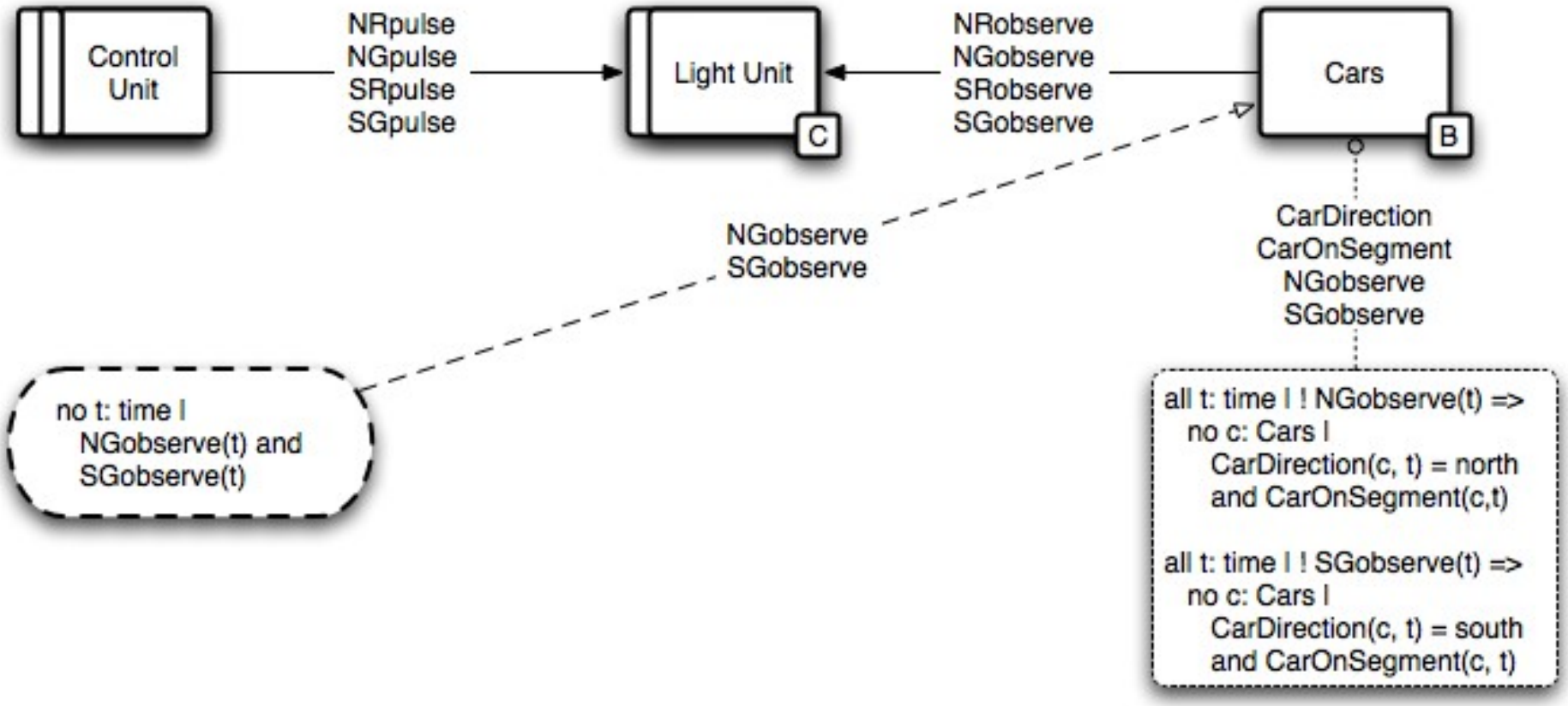
Breadcrumb 1



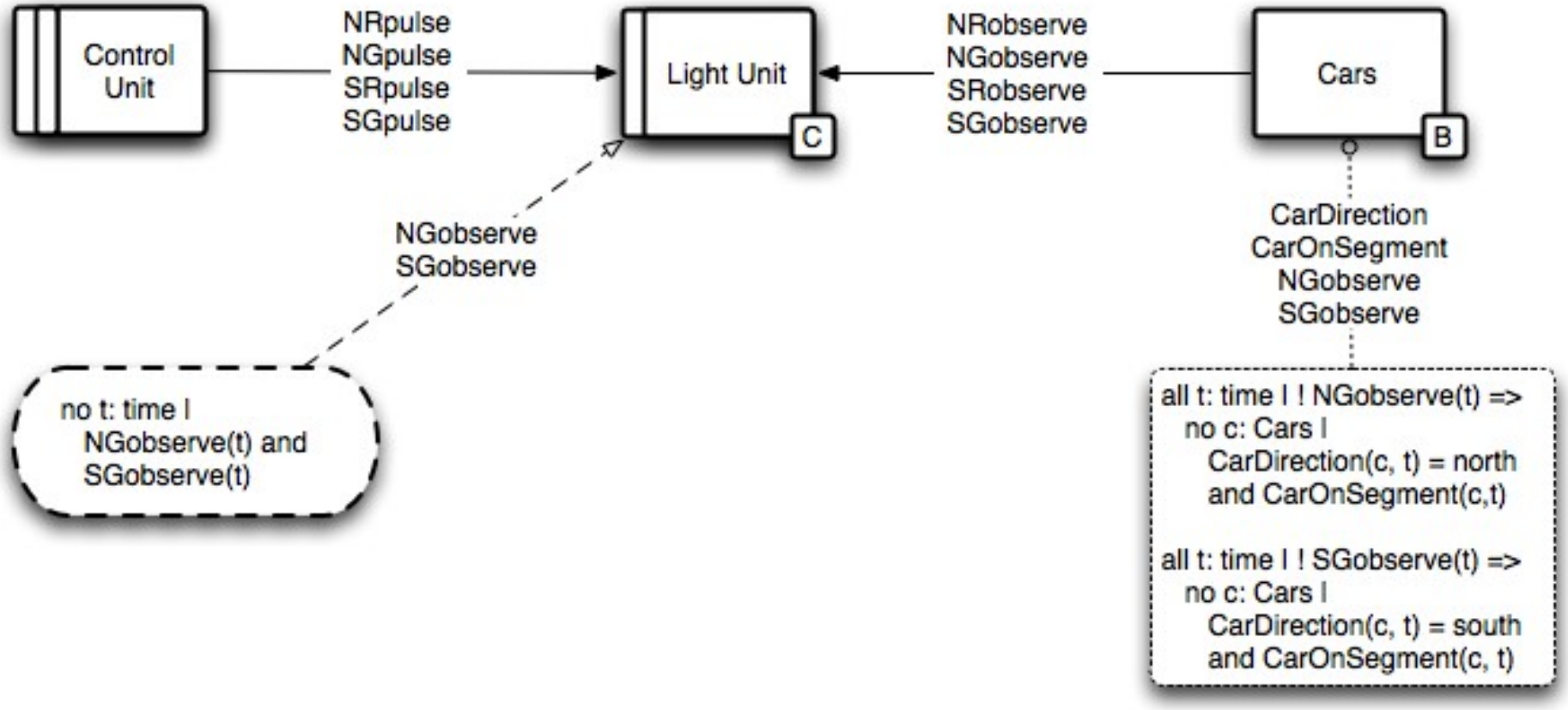
Rephrase 1


Control Unit
NRpulse
NGpulse
SRpulse
SGpulse
Light Unit
C
NRobserve
NGobserve
SRobserve
SGobserve
Cars
B
NGobserve
SGobserve
CarDirection
CarOnSegment
CarDirection
CarOnSegment
NGobserve
SGobserve
no t: time | NGobserve(t) and SGobserve(t)
no t: time | some c1, c2 : Cars | CarDirection(c1, t) = north and CarDirection(c2, t) = south and CarOnSegment(c1, t) and CarOnSegment(c2, t)
all t: time | ! NGobserve(t) => no c: Cars | CarDirection(c, t) = north and CarOnSegment(c,t)
all t: time | ! SGobserve(t) => no c: Cars | CarDirection(c, t) = south and CarOnSegment(c, t)

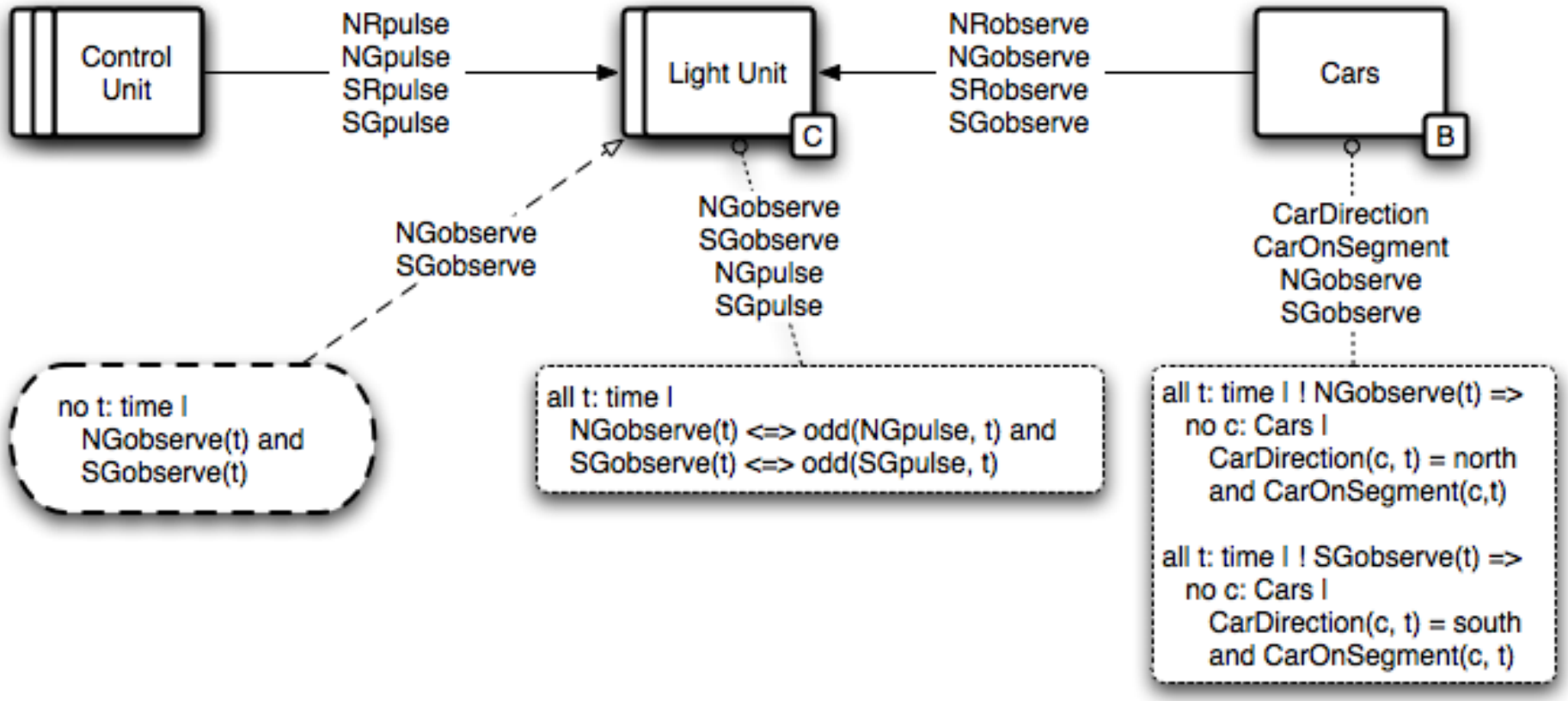
Rephrase 1


Control Unit
NRpulse
NGpulse
SRpulse
SGpulse
Light Unit
C
NRobserve
NGobserve
SRobserve
SGobserve
Cars
B
CarDirection
CarOnSegment
NGobserve
SGobserve
NGobserve
SGobserve
no t: time |
NGobserve(t) and
SGobserve(t)
all t: time | ! NGobserve(t) =>
no c: Cars |
CarDirection(c, t) = north
and CarOnSegment(c,t)
all t: time | ! SGobserve(t) =>
no c: Cars |
CarDirection(c, t) = south
and CarOnSegment(c, t)

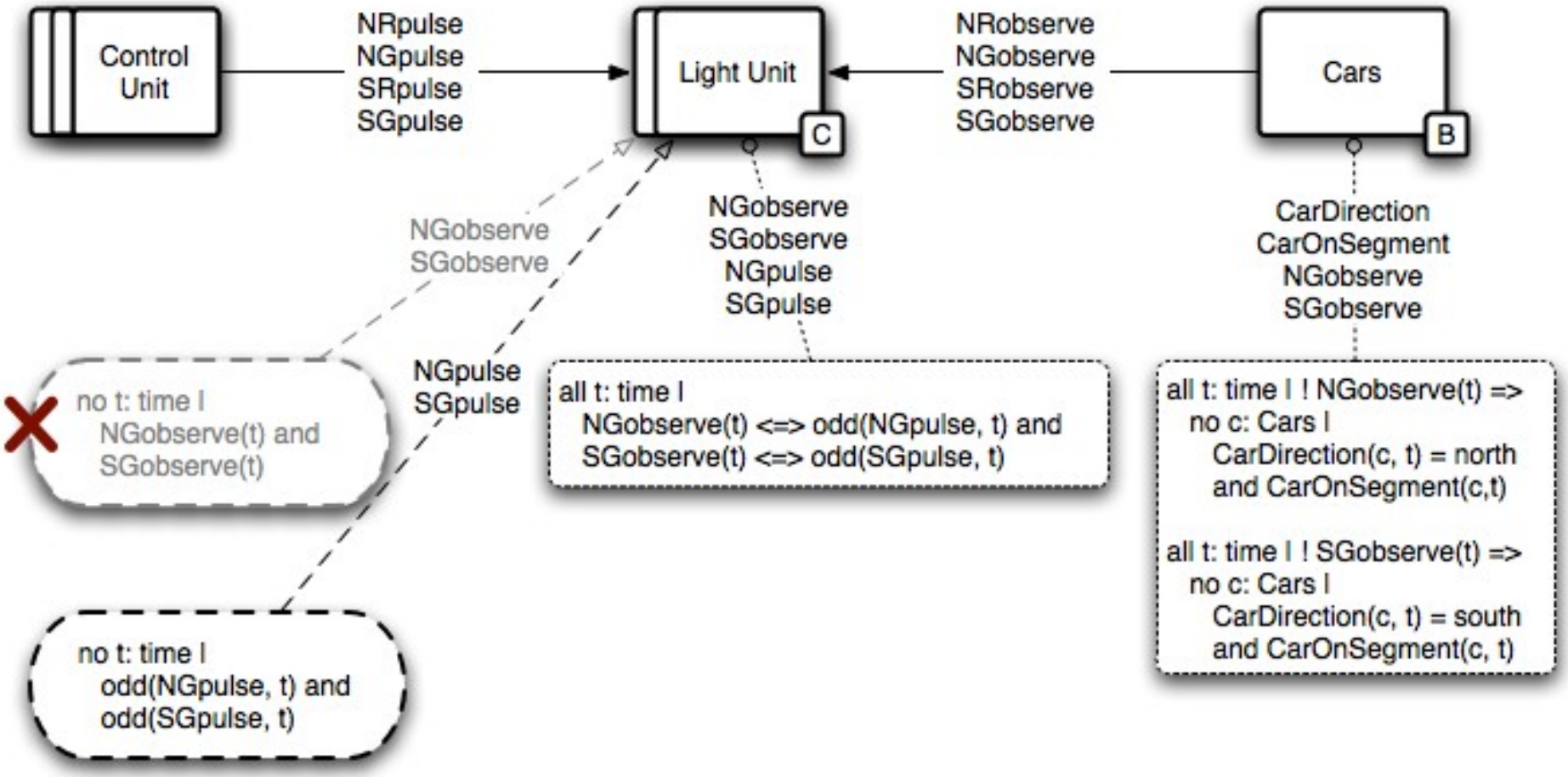
Push 1


Control Unit
NRpulse
NGpulse
SRpulse
SGpulse
Light Unit
C
NRobserve
NGobserve
SRobserve
SGobserve
Cars
B
CarDirection
CarOnSegment
NGobserve
SGobserve
NGobserve
SGobserve
no t: time |
NGobserve(t) and
SGobserve(t)
all t: time | ! NGobserve(t) =>
no c: Cars |
CarDirection(c, t) = north
and CarOnSegment(c,t)
all t: time | ! SGobserve(t) =>
no c: Cars |
CarDirection(c, t) = south
and CarOnSegment(c, t)

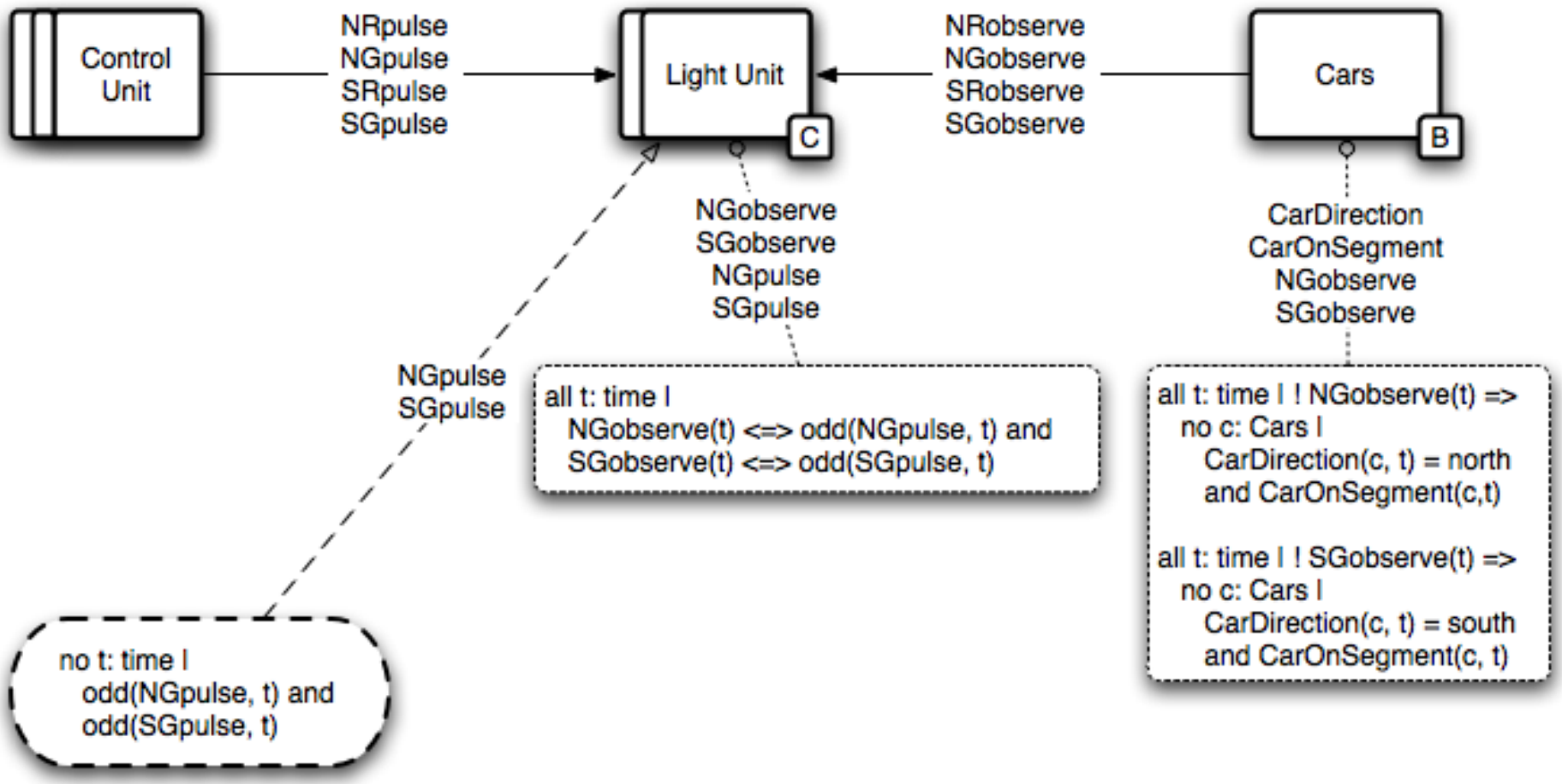
Breadcrumb 2



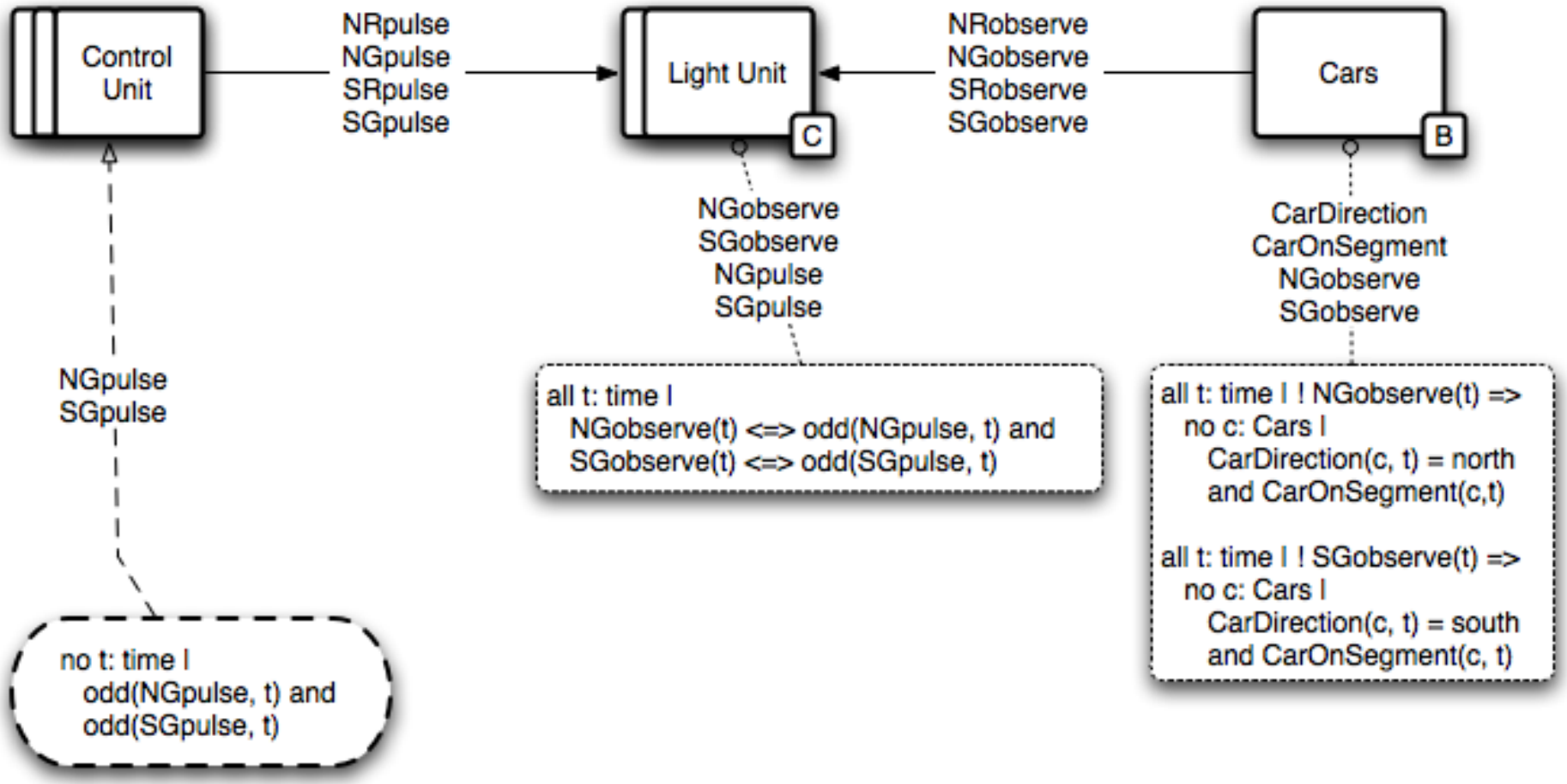
Rephrase 2


Control Unit
NRpulse
NGpulse
SRpulse
SGpulse
Light Unit
C
NRobserve
NGobserve
SRobserve
SGobserve
Cars
B
NGobserve
SGobserve
NGobserve
SGobserve
NGpulse
SGpulse
CarDirection
CarOnSegment
NGobserve
SGobserve
no t: time I
NGobserve(t) and
SGobserve(t)
NGpulse
SGpulse
all t: time I
NGobserve(t) <=> odd(NGpulse, t) and
SGobserve(t) <=> odd(SGpulse, t)
all t: time I ! NGobserve(t) =>
no c: Cars I
CarDirection(c, t) = north
and CarOnSegment(c,t)
all t: time I ! SGobserve(t) =>
no c: Cars I
CarDirection(c, t) = south
and CarOnSegment(c, t)
no t: time I
odd(NGpulse, t) and
odd(SGpulse, t)

Rephrase 2


Control Unit
NRpulse
NGpulse
SRpulse
SGpulse
Light Unit
C
NRobserve
NGobserve
SRobserve
SGobserve
Cars
B
NGobserve
SGobserve
NGpulse
SGpulse
CarDirection
CarOnSegment
NGobserve
SGobserve
NGpulse
SGpulse
all t: time l
NGobserve(t) <=> odd(NGpulse, t) and
SGobserve(t) <=> odd(SGpulse, t)
all t: time l ! NGobserve(t) =>
no c: Cars l
CarDirection(c, t) = north
and CarOnSegment(c,t)
all t: time l ! SGobserve(t) =>
no c: Cars l
CarDirection(c, t) = south
and CarOnSegment(c, t)
no t: time l
odd(NGpulse, t) and
odd(SGpulse, t)

Push 2


Control Unit
NRpulse
NGpulse
SRpulse
SGpulse
Light Unit
C
NRobserve
NGobserve
SRobserve
SGobserve
Cars
B
NGobserve
SGobserve
NGpulse
SGpulse
CarDirection
CarOnSegment
NGobserve
SGobserve
NGpulse
SGpulse
all t: time l
NGobserve(t) <=> odd(NGpulse, t) and
SGobserve(t) <=> odd(SGpulse, t)
all t: time l ! NGobserve(t) =>
no c: Cars l
CarDirection(c, t) = north
and CarOnSegment(c,t)
all t: time l ! SGobserve(t) =>
no c: Cars l
CarDirection(c, t) = south
and CarOnSegment(c, t)
no t: time l
odd(NGpulse, t) and
odd(SGpulse, t)

Provides

- systematic local reasoning

  $Breadcrumb \wedge Rephrased\ Goal \Rightarrow Prior\ Goal$

- global guarantee

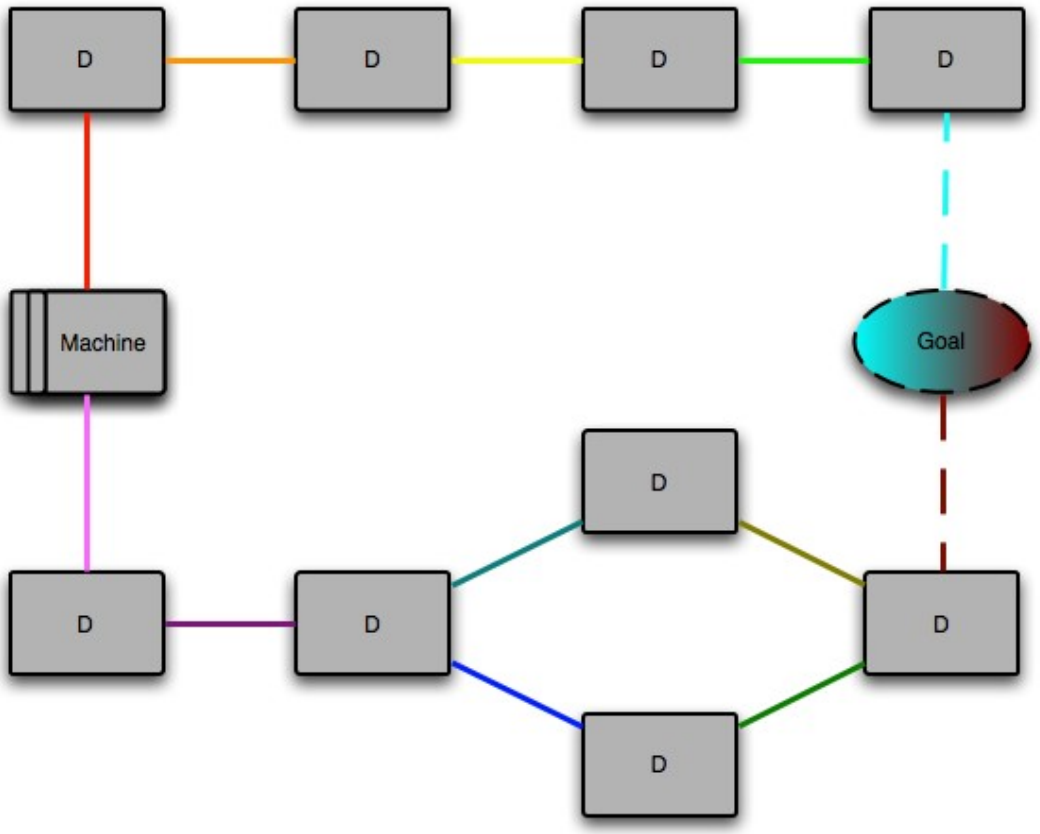
  $Breadcrumb\_0 \wedge \ldots \wedge Breadcrumb\_n \wedge Specification \Rightarrow Requirement$

- tracability: trail of breadcrumbs
- identify unused phenomena
- handle general topologies
- formalize use of frame concern
- local patterns replace global patterns

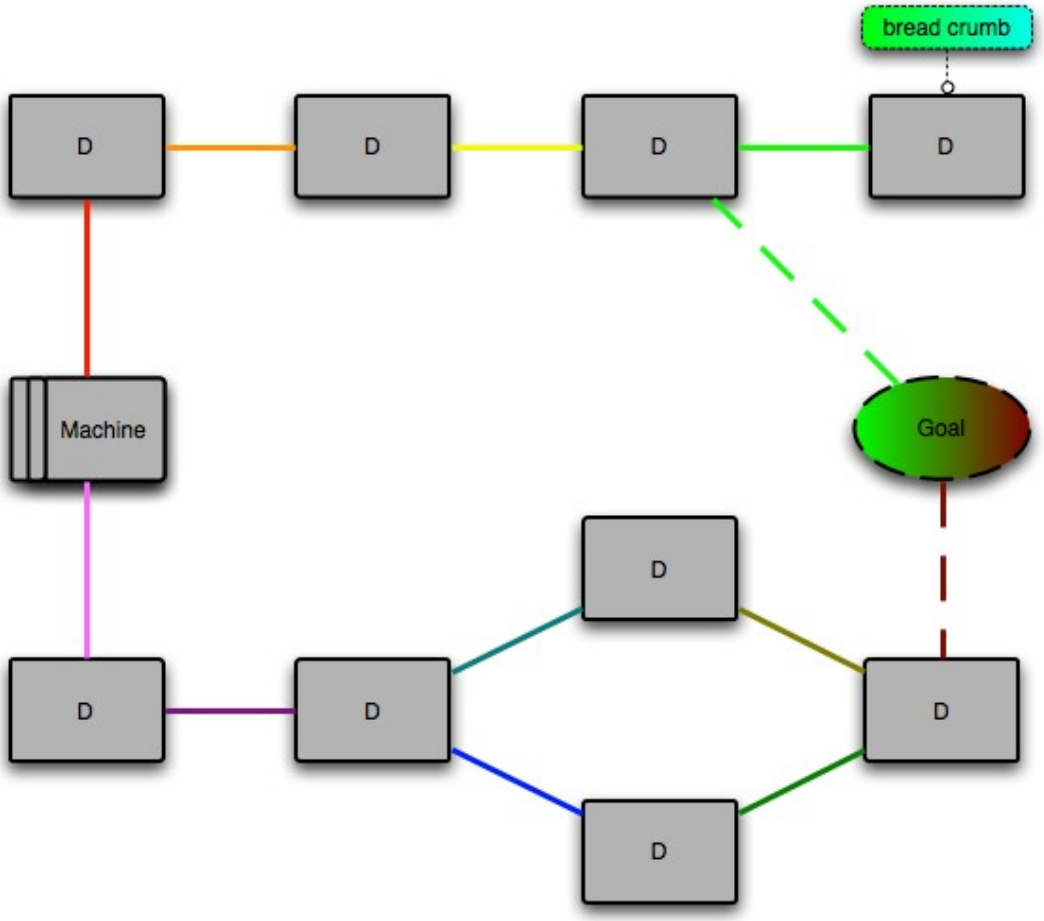
Difficulties

- systematic not automatic (inescapable)
- readability, implementability, consistency
- which breadcrumb/rewrite?
- which push? split?
- get stuck later on?

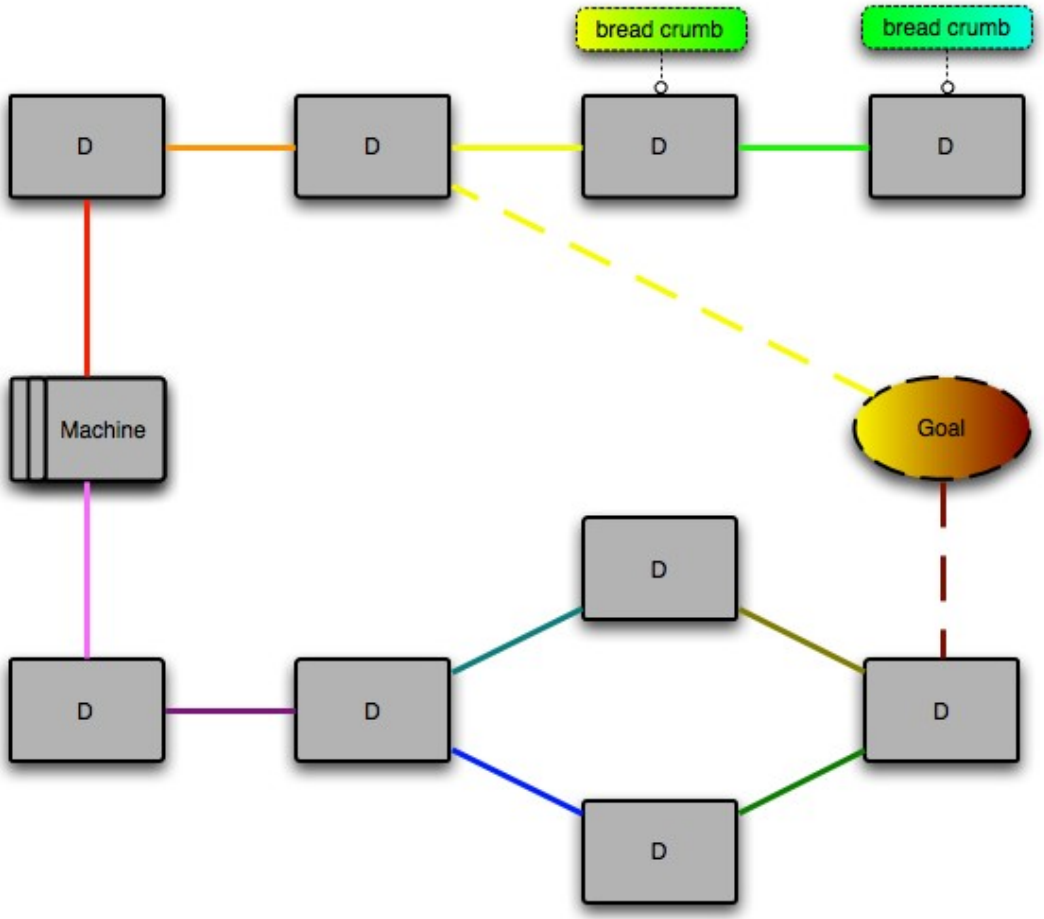
Cartoon of Big Example



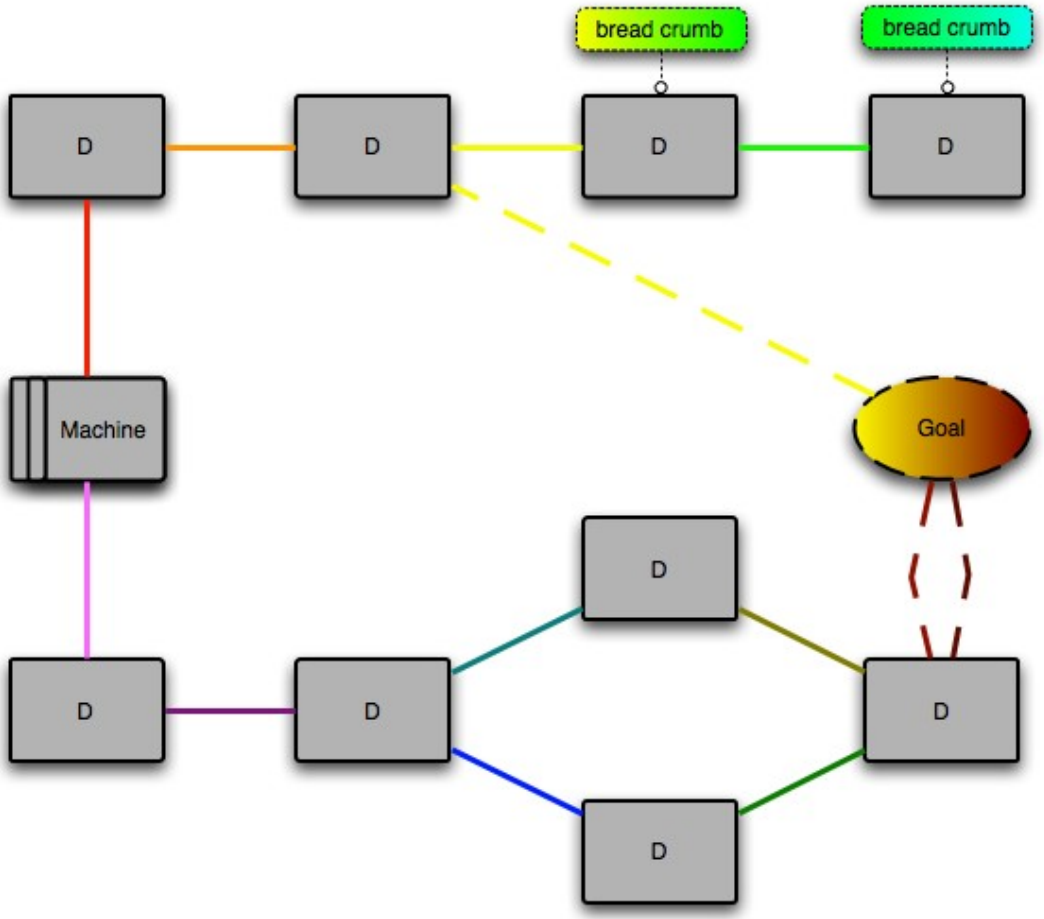
Cartoon of Big Example



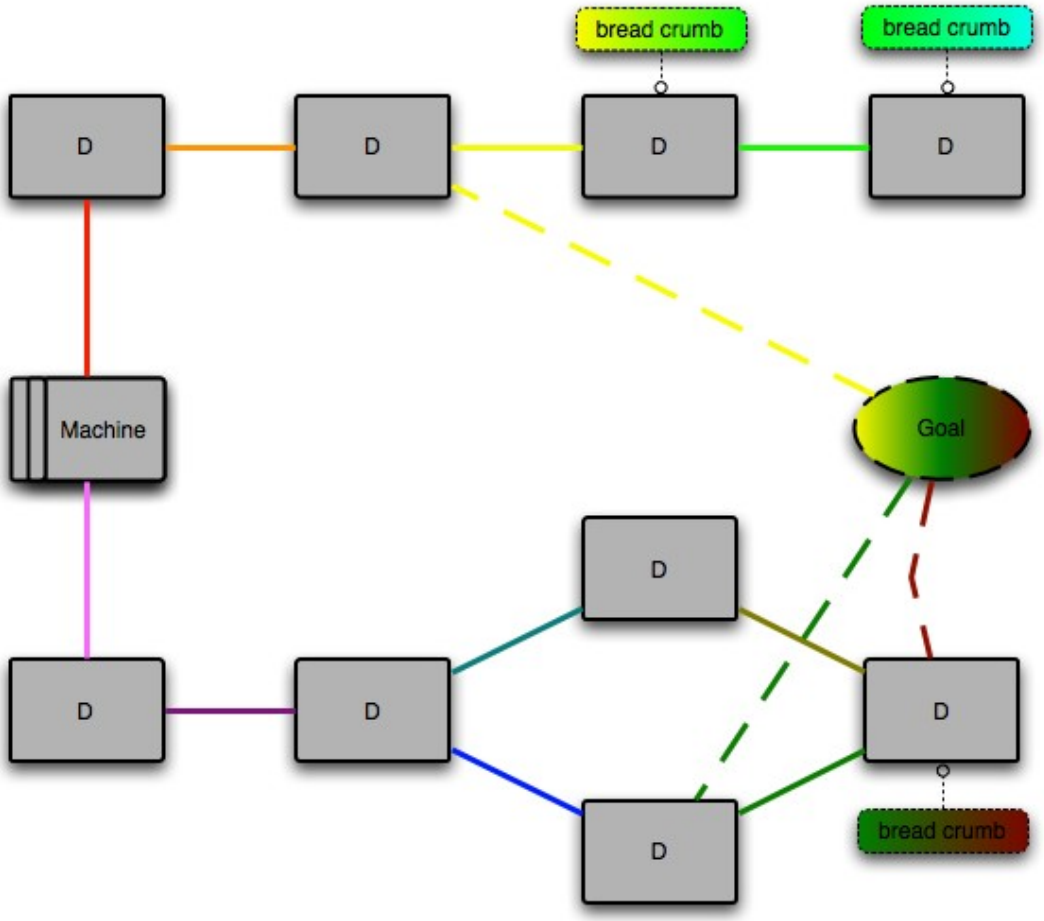
Cartoon of Big Example



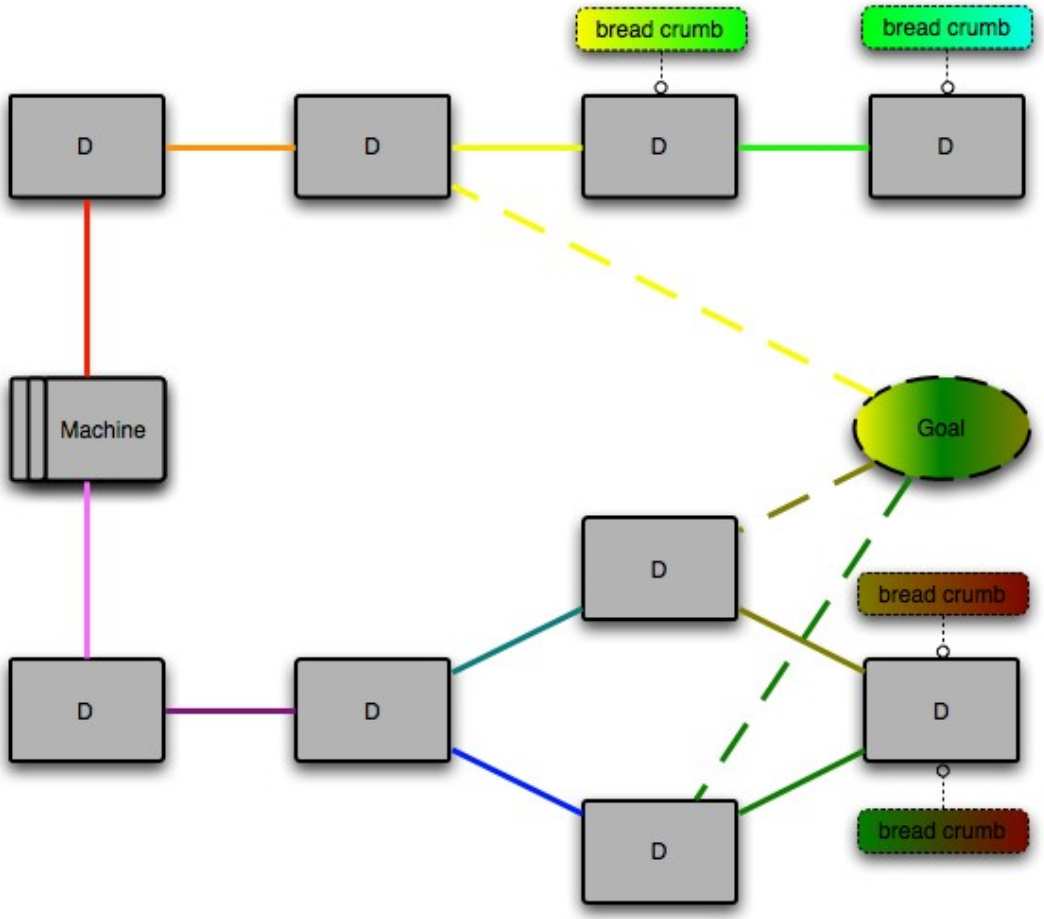
Cartoon of Big Example



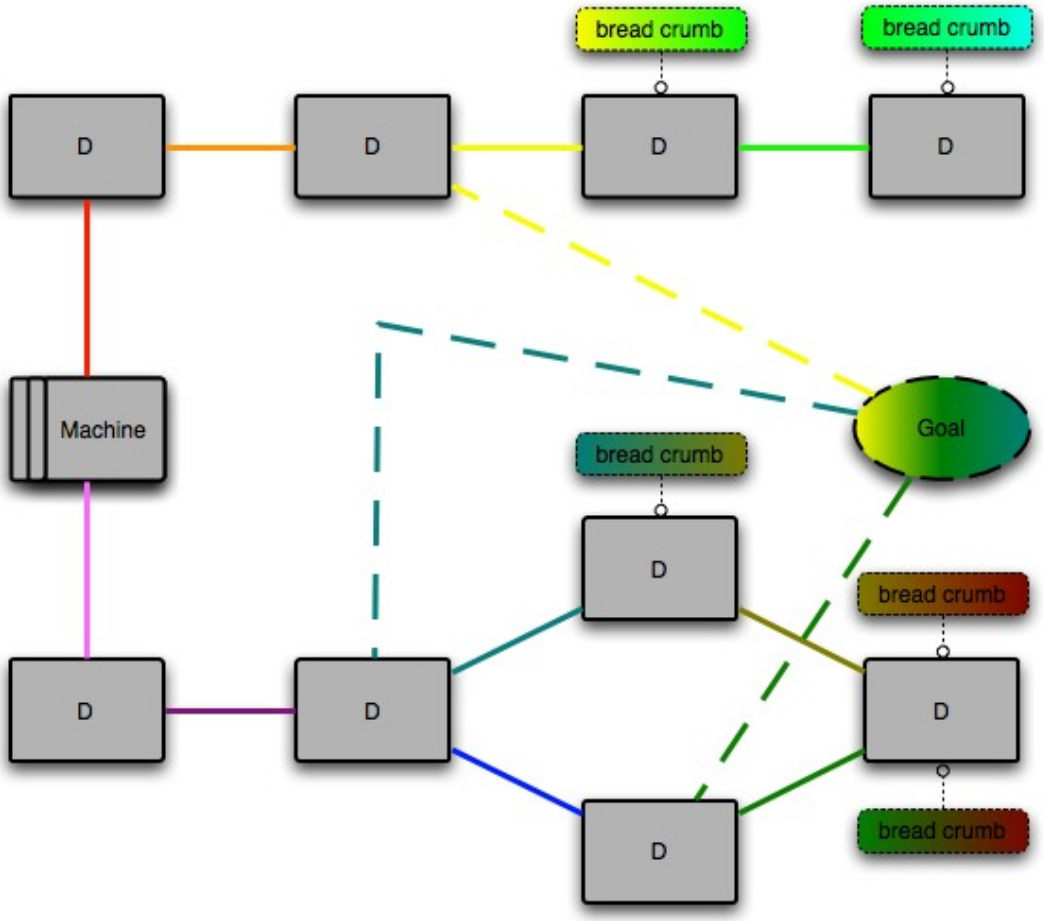
Cartoon of Big Example



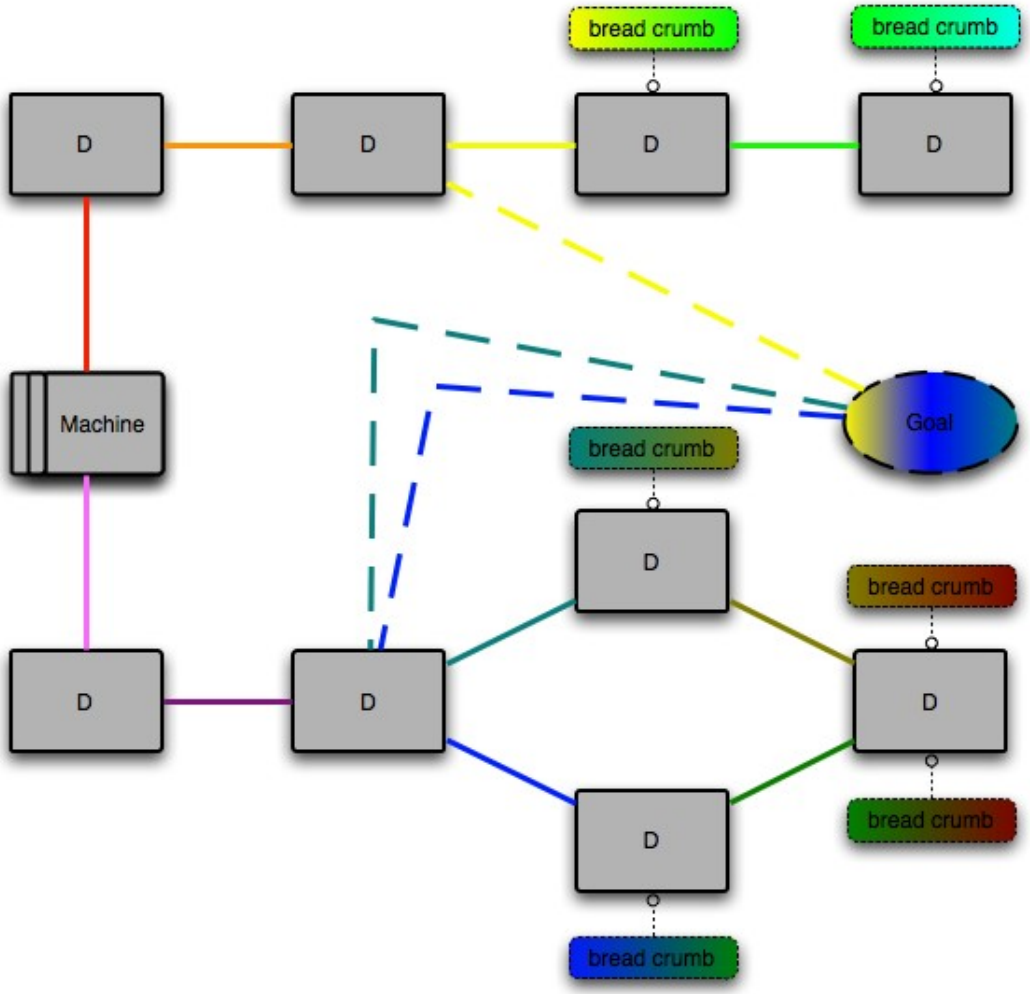
Cartoon of Big Example



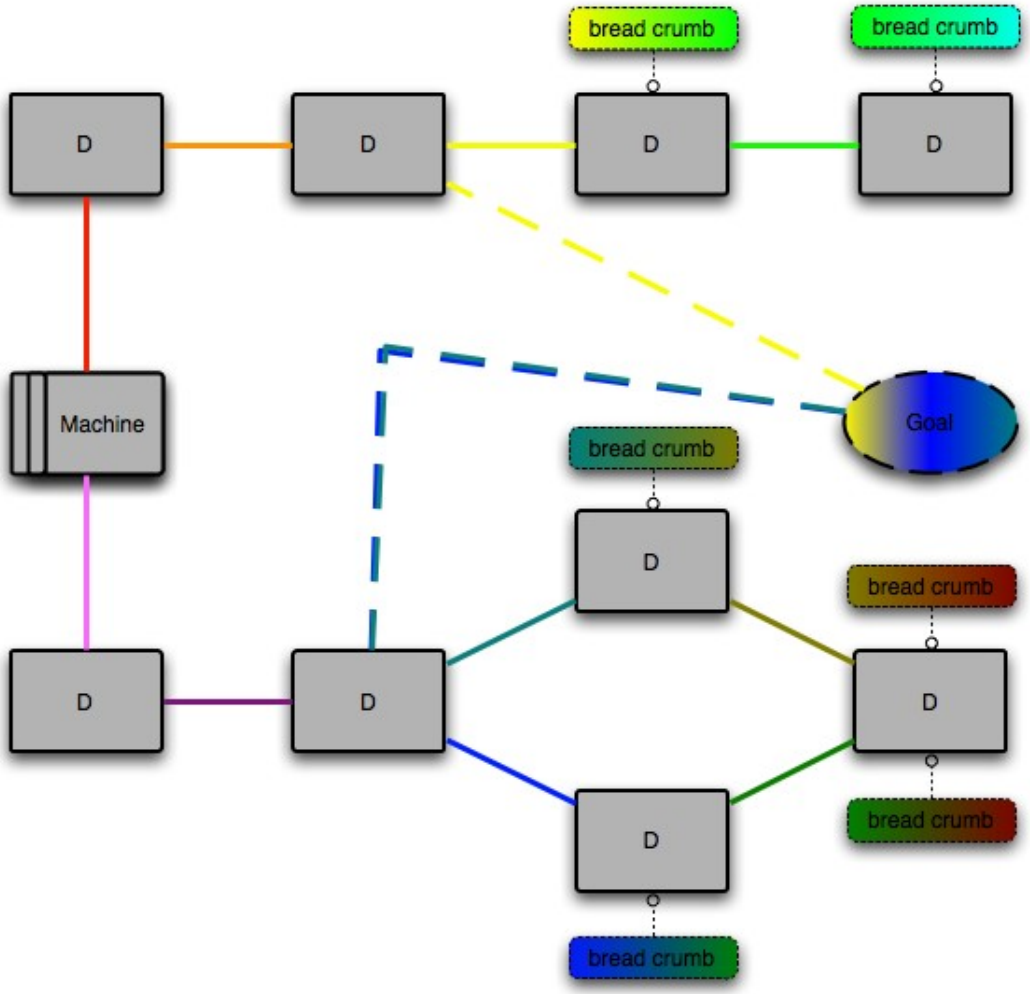
Cartoon of Big Example



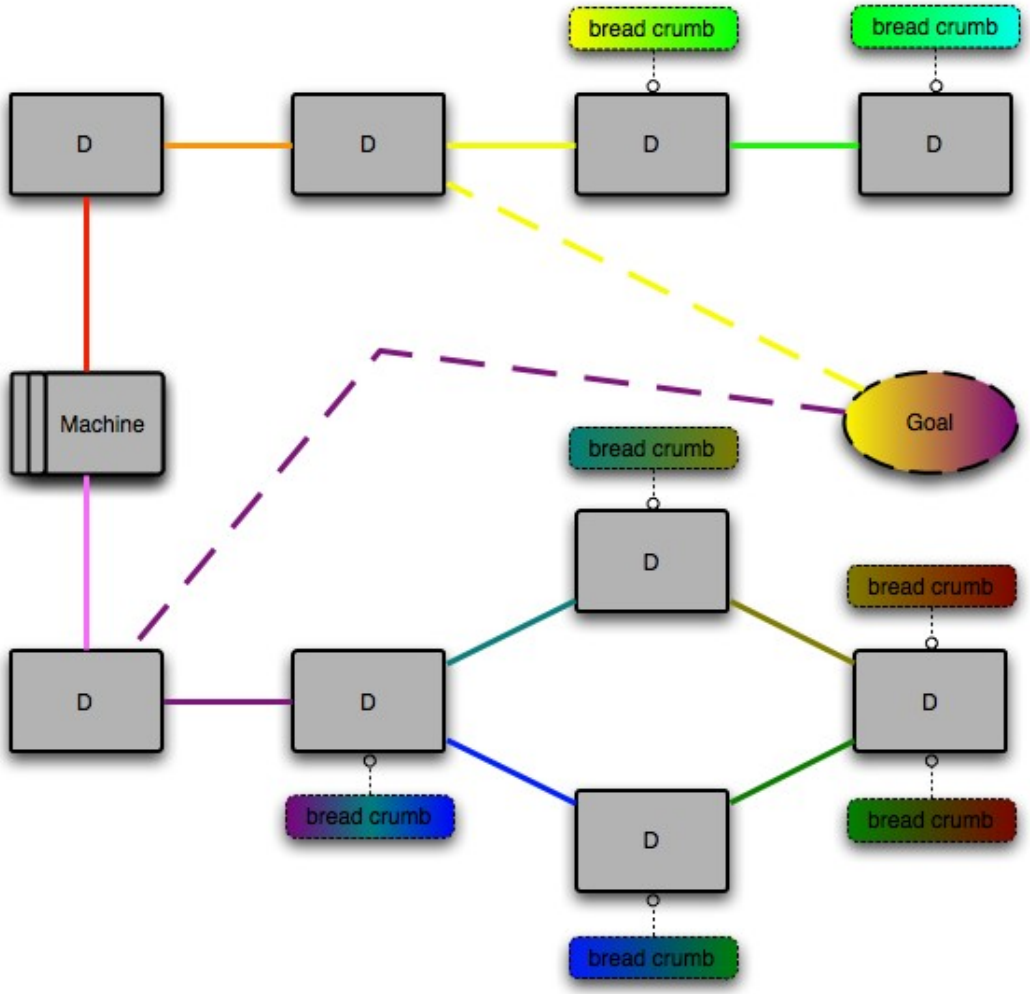
Cartoon of Big Example



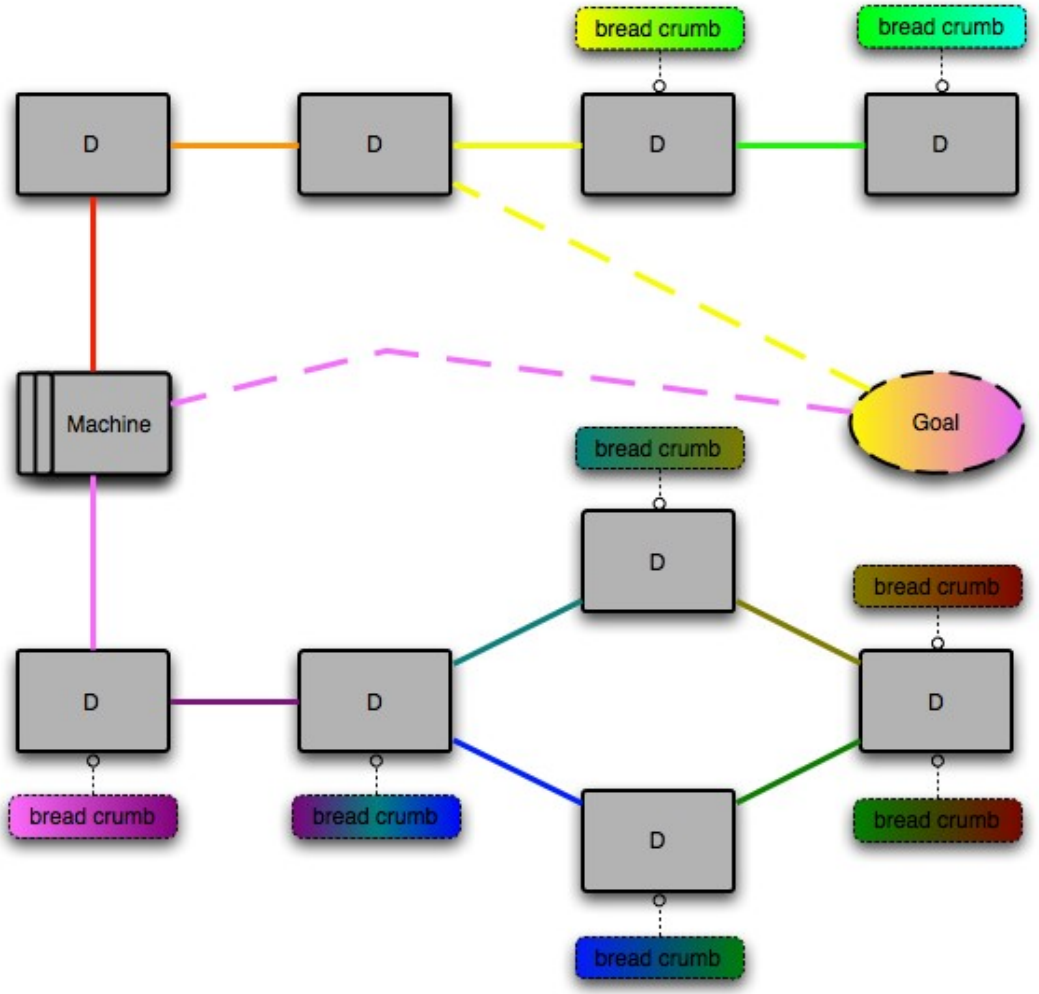
Cartoon of Big Example



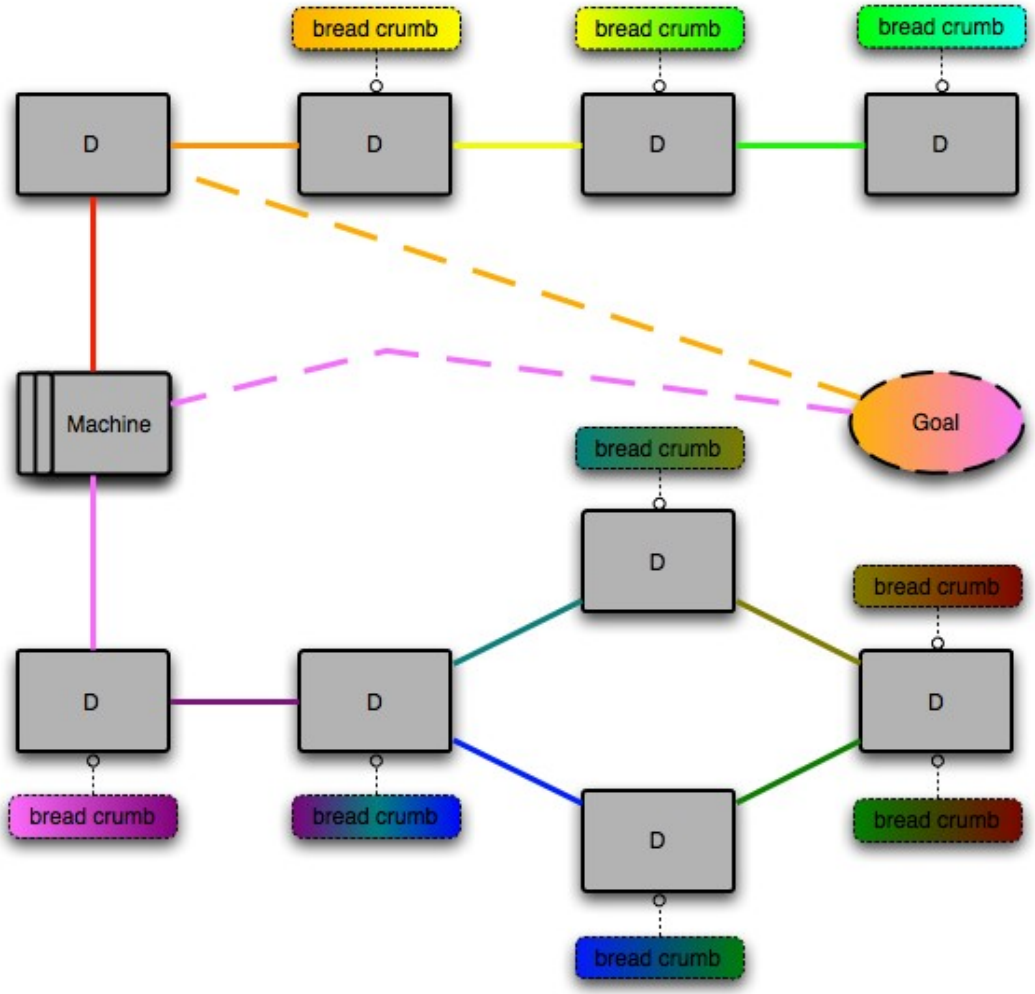
Cartoon of Big Example



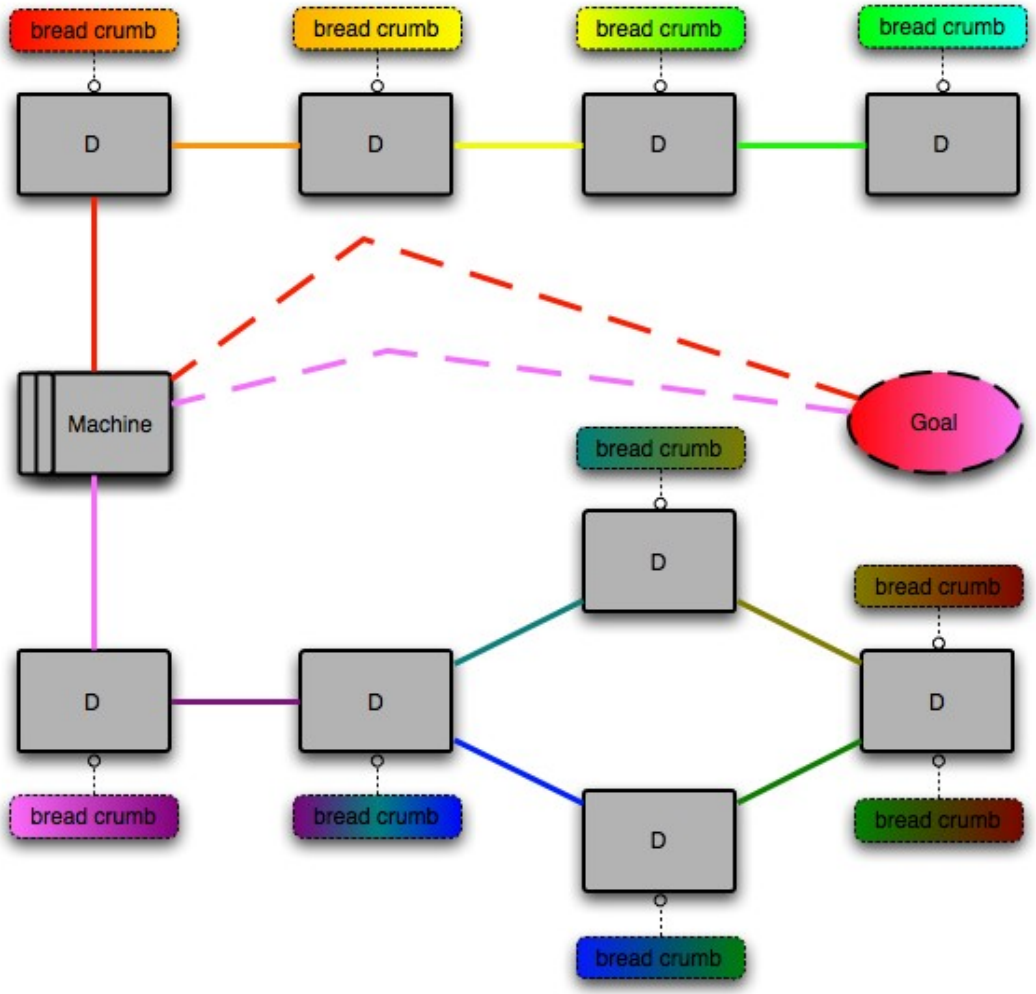
Cartoon of Big Example



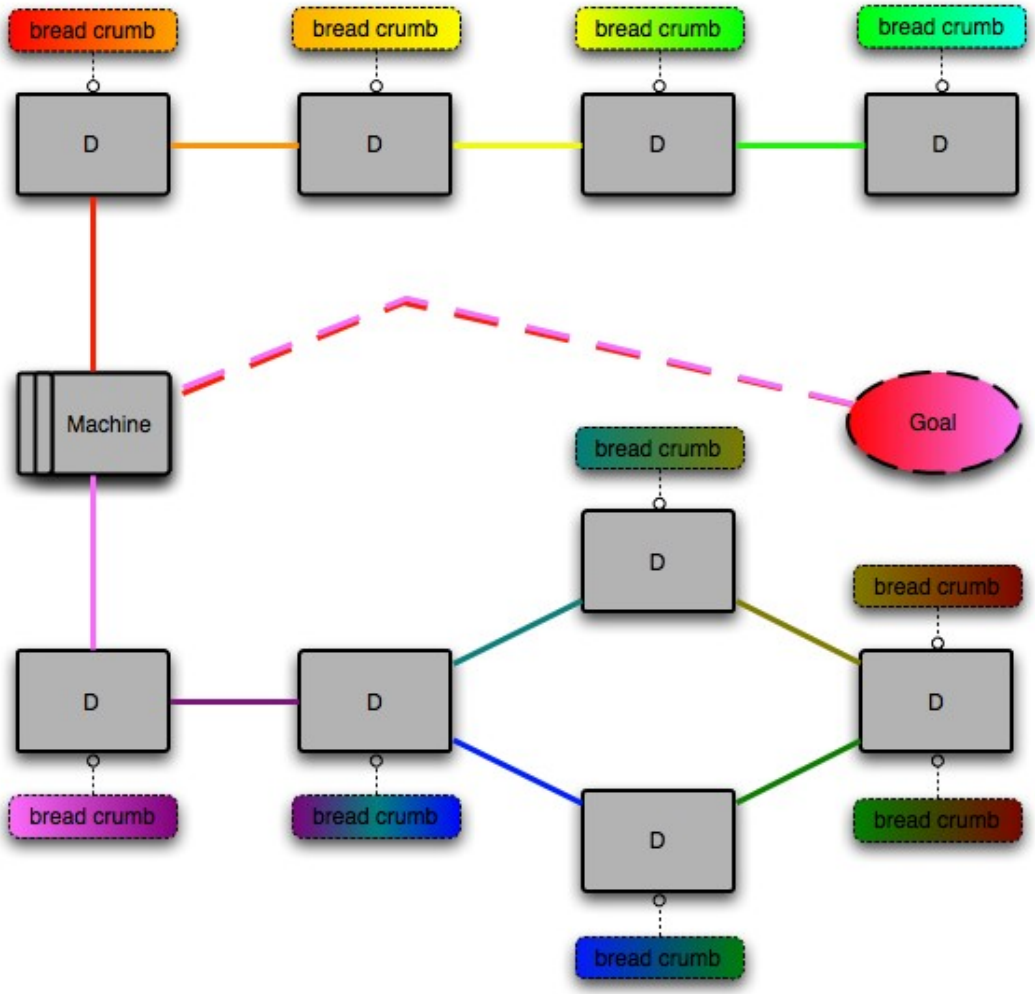
Cartoon of Big Example



Cartoon of Big Example



Cartoon of Big Example

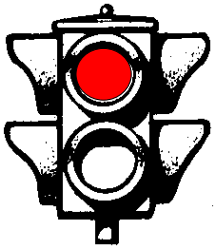


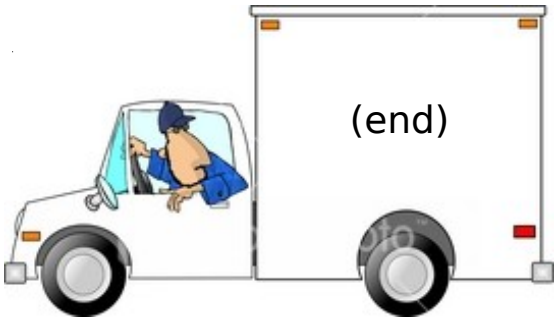
Future Work

- patterns for local steps, concurrent steps
- proton therapy case study / safety case
- example/error progression

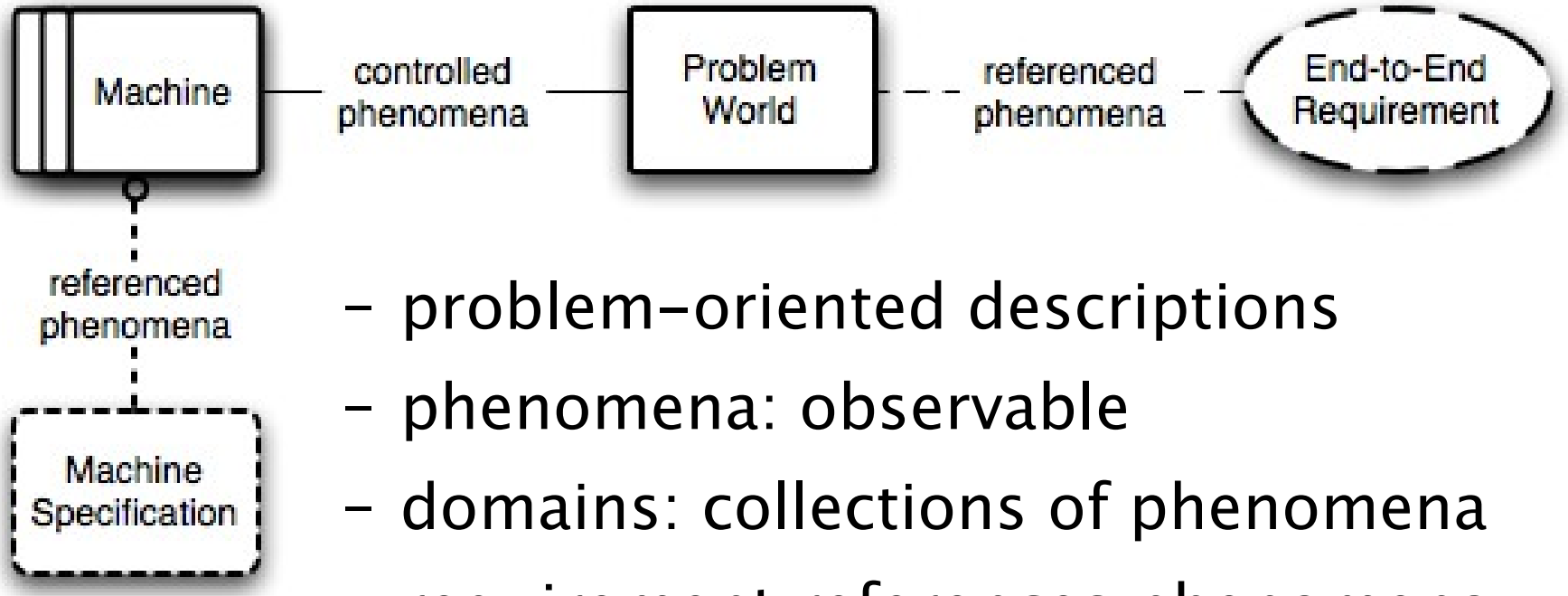
Related Work on Problem Frames

- Jackson, Zave (1995) turnstyle example
- Jackson (2001) problem progression
- Rapanotti, Hall, Li (2006) causal reasoning
- Hall, Rapanotti (2006) requirement progression
- Hall, Jackson, Laney, Nuseibeh, Rapanotti (2002, 2004) modeling architectural decisions




(end)

Problem Frames



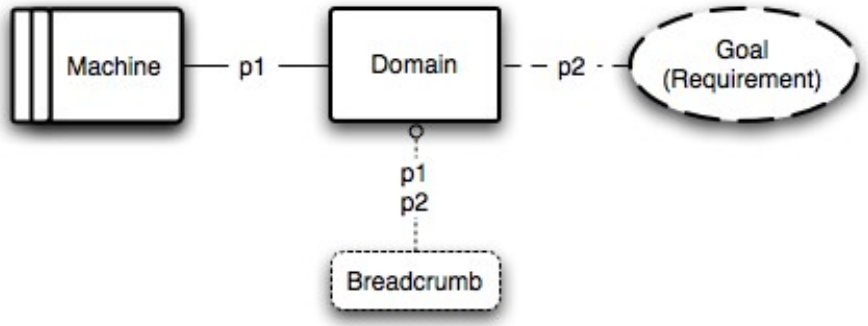
- problem-oriented descriptions
- phenomena: observable
- domains: collections of phenomena
- requirement references phenomena
- machine controls phenomena to enforce requirement
- specification references controlled phenomena

Typical Transformation

- need to constrain p1 instead of p2

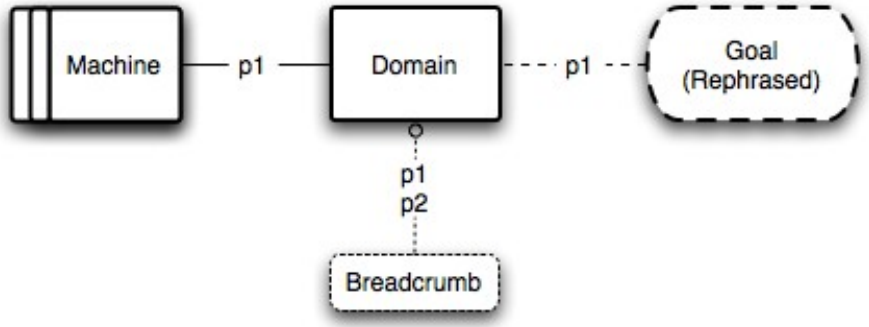
Typical Transformation

- need to constrain p1 instead of p2
- **add** a breadcrumb assumption relating p1 and p2



Typical Transformation

- need to constrain p1 instead of p2
- **add** a breadcrumb assumption relating p1 and p2
- **rephrase** the goal to reference p1 instead of p2 such that



$$Breadcrumb \wedge Rephrased\ Goal \Rightarrow Prior\ Goal$$

Typical Transformation

- need to constrain p1 instead of p2
- **add** a breadcrumb assumption relating p1 and p2
- **rephrase** the goal to reference p1 instead of p2 such that

$$Breadcrumb \wedge Rephrased\ Goal \Rightarrow Prior\ Goal$$

- **push** the goal towards the machine