The Application of Dependence Analysis to Software Architecture Descriptions

Presenter:

Alexander L. Wolf

University of Colorado

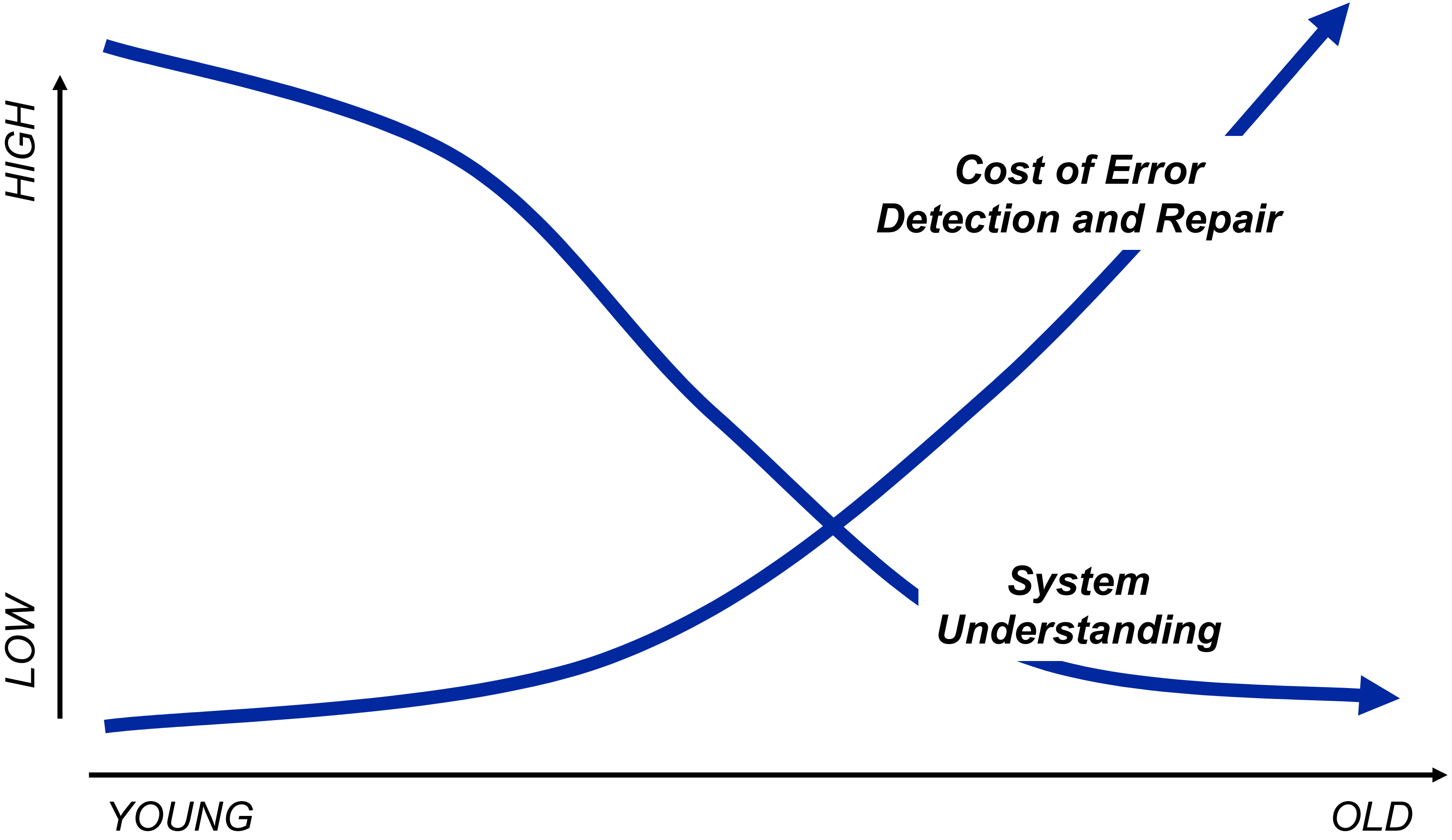
Boulder, CO USA

With contributions by:

Judith Stafford, Tufts University, USA

Mauro Caporuscio, Universitá dell'Aquila, Italy

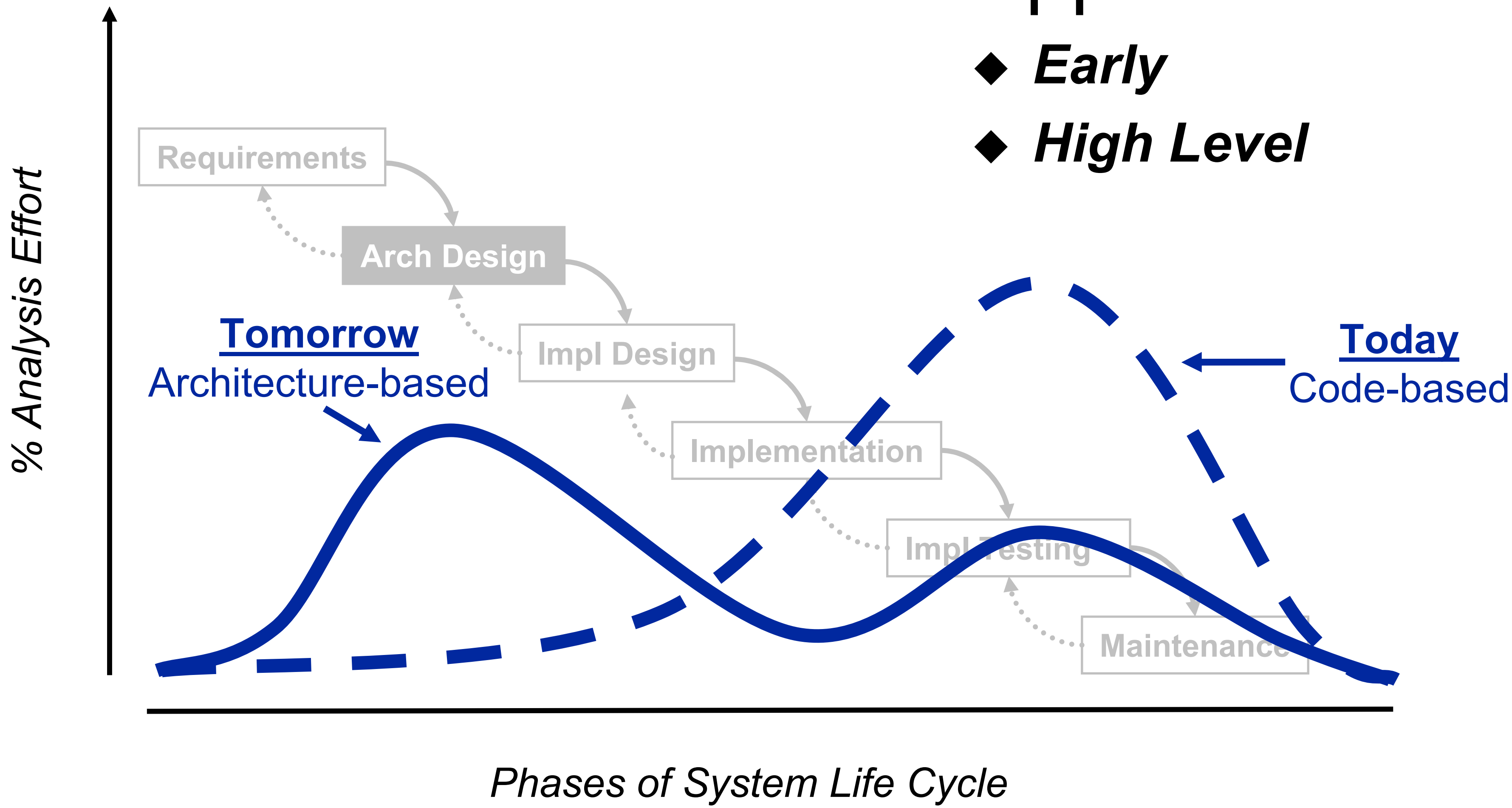
High Cost of System Maintenance



Architecture-Based Analysis

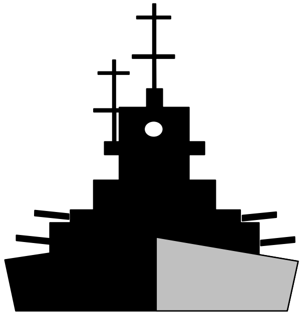
Approach

- ***Early***
- ***High Level***



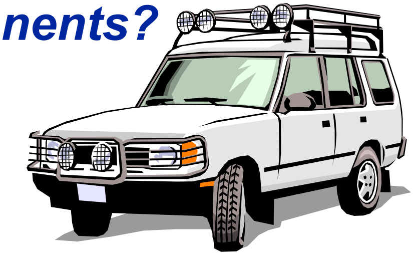
Example Architectural Questions

Why is the ignition never allowed to activate?



What could cause the system to go on alert?

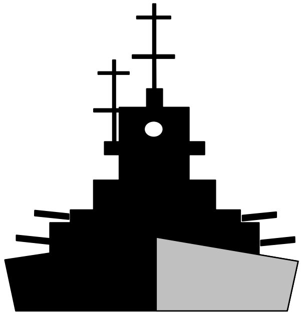
Can the braking system be affected by any less safety critical components?

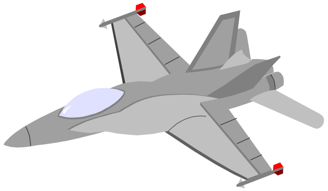


Will keystrokes be recognized in the order they were typed?

Example Architectural Relationships

State-based: The car must be in park when the ignition is activated.





Causal: When a plane comes within range, the system must be put on alert.

Safety level: The level 4 braking subsystem can be affected by the level 1 GPS.

Temporal: Keystrokes must be recognized in the order they were typed.

More Architectural Questions

- Which components make use of this particular state of a component?
- If this component uses a shared repository, with what other components does it communicate?
- What are the potential effects of dynamically replacing this component?
- If this component is to be reused in another system, which other components of the system are also required?

Still More Architectural Questions

- If a failure of the system occurs, what is the minimal set of components that must be inspected during the debugging process?
- If the source specification for a component is checked out into a workspace for modification, which other source specifications should also be checked out?
- If a change is made to this component, what is the minimal set of test cases that must be rerun?

Dependence Analysis

- Widely studied for *program analysis*
  - determines dependence relationships among code (i.e., implementation-level) elements

- Formal architecture description languages enable automated analyses

- Can we apply dependence analysis techniques to architectural descriptions?

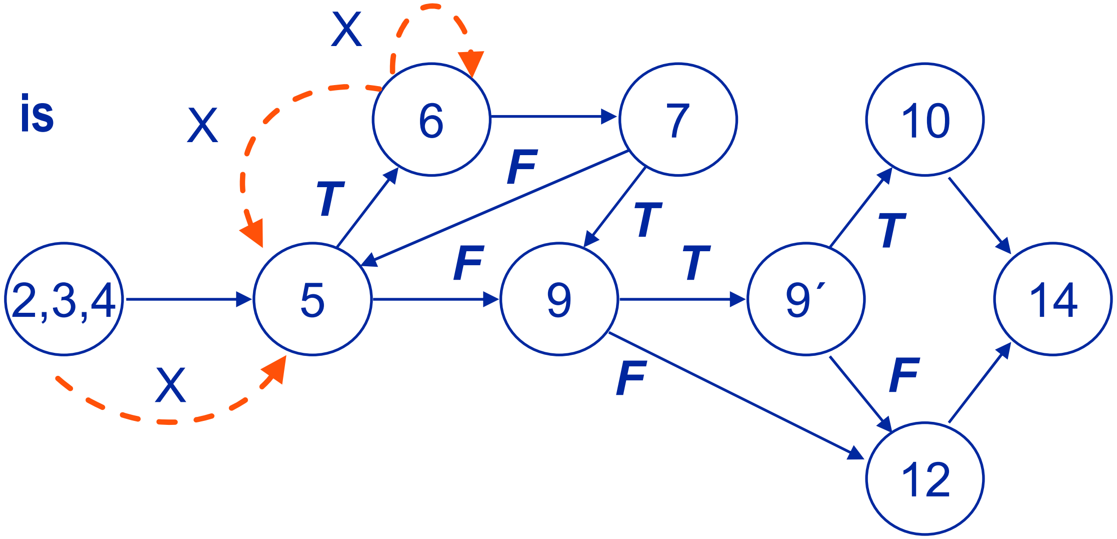
Foundations: Flow Graphs for Programs

Graph representation of control flow and data flow relationships

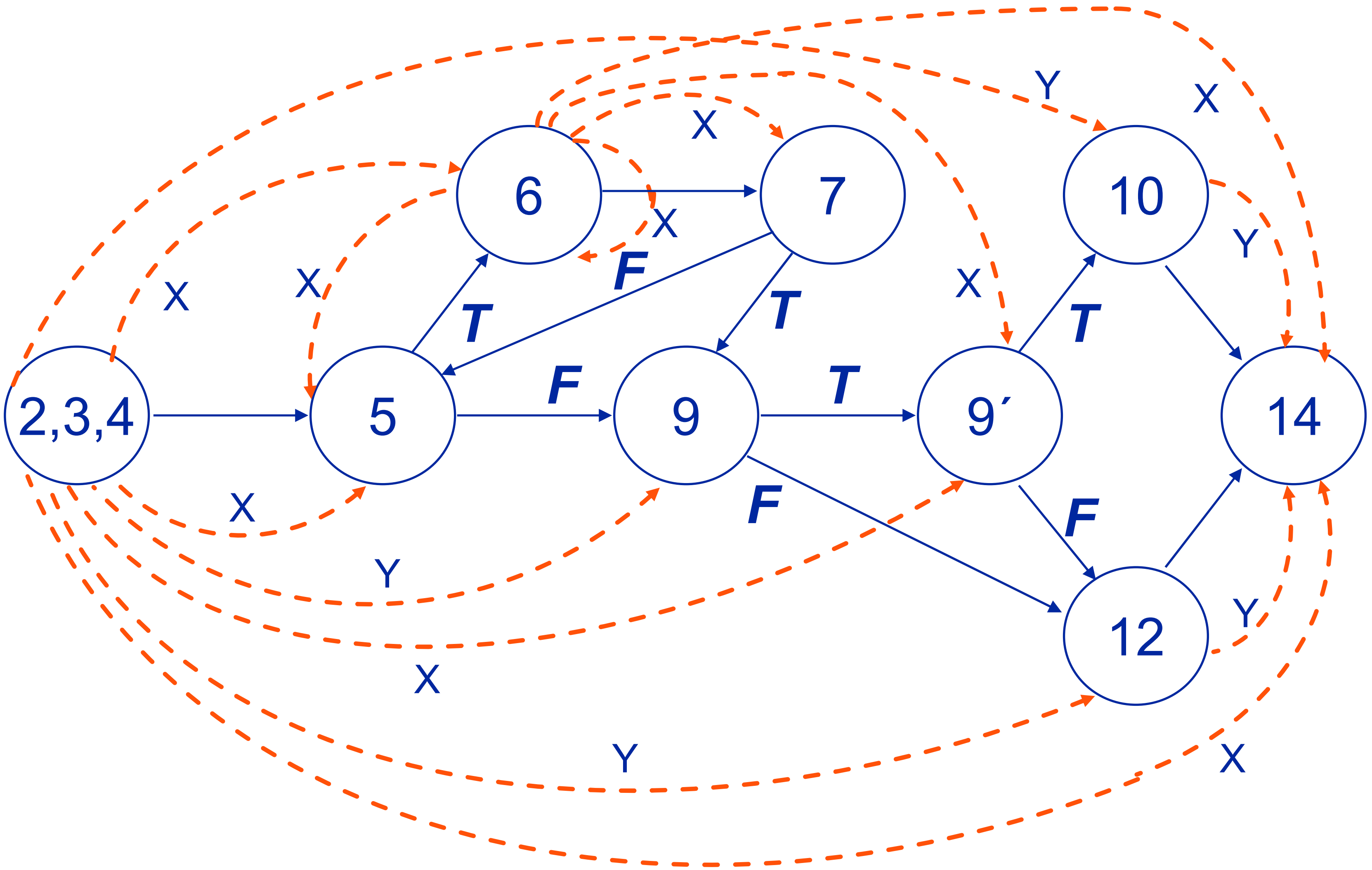
- Control flow
  - the partial order of statement execution, as defined by the semantics of the language
- Data flow
  - the flow of values from definitions of a variable to its uses

A Sample Program

```
 1  function P return INTEGER is
 2  begin
 3        X, Y: INTEGER;
 4        READ(X); READ(Y);
 5        while (X > 10) loop
 6              X := X – 10;
 7              exit when X = 10;
 8        end loop;
 9        if (Y < 20 and then X mod 2 = 0) then
10              Y := Y + 20;
11        else
12              Y := Y – 20;
13        end if;
14        return 2 ∗ X + Y;
15  end P;
```



P’s Control/Data Flow Graph



Program Dependence Graph (PDG)

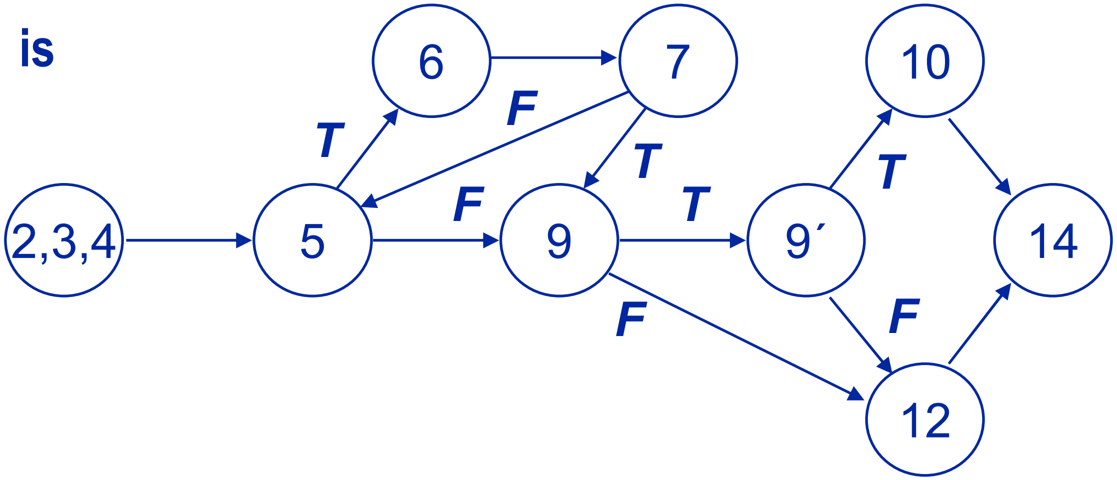
- Summary representation of “dependence”
- Nodes are either
  - statements
  - predicates
  - special “entry” node
- Two kinds of edges
  - control dependence edge
  - data dependence edge
- Two subgraphs induced by the edges

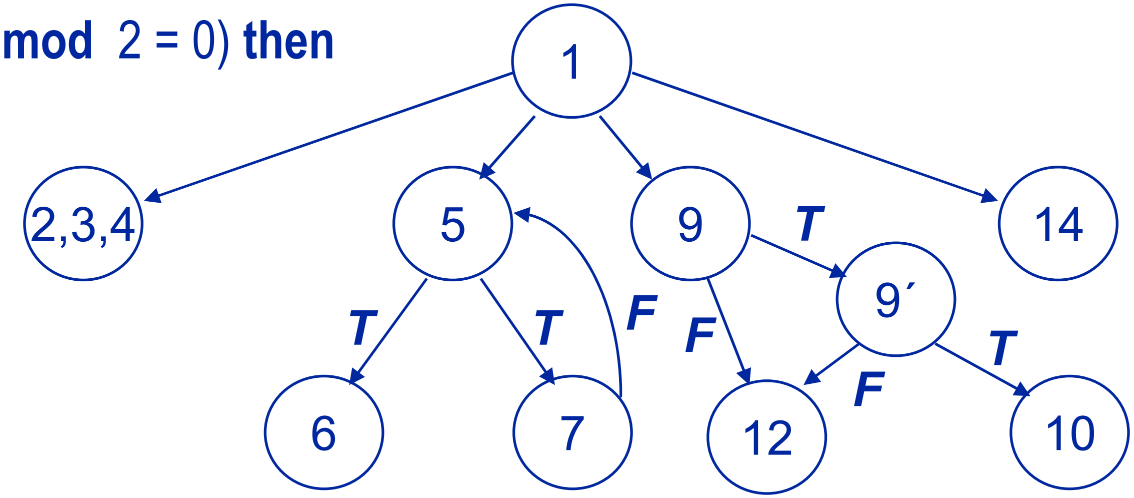
Control Dependence Graph (CDG)

- Informal definition
  - for nodes X and Y in a CFG, Y is control dependent on X if, during execution, X can directly affect whether Y is executed

A Sample Program

```
1  function P return INTEGER is
2  begin
3        X, Y: INTEGER;
4        READ(X); READ(Y);
5        while (X > 10) loop
6              X := X – 10;
7              exit when X = 10;
8        end loop;
9        if (Y < 20 and then X mod 2 = 0) then
10             Y := Y + 20;
11       else
12             Y := Y – 20;
13       end if;
14       return 2 ∗ X + Y;
15 end P;
```





Control Dependence Graph (CDG)

- Formal definition
  - let X and Y be nodes in a CFG
  - if Y appears on every path from X to the exit node, where Y≠X, then Y post-dominates X
  - there is a control dependence from X to Y with label L iff:
    - » there is a non-null path p from X to Y, starting with edge L, such that Y post-dominates every node strictly between X and Y on p

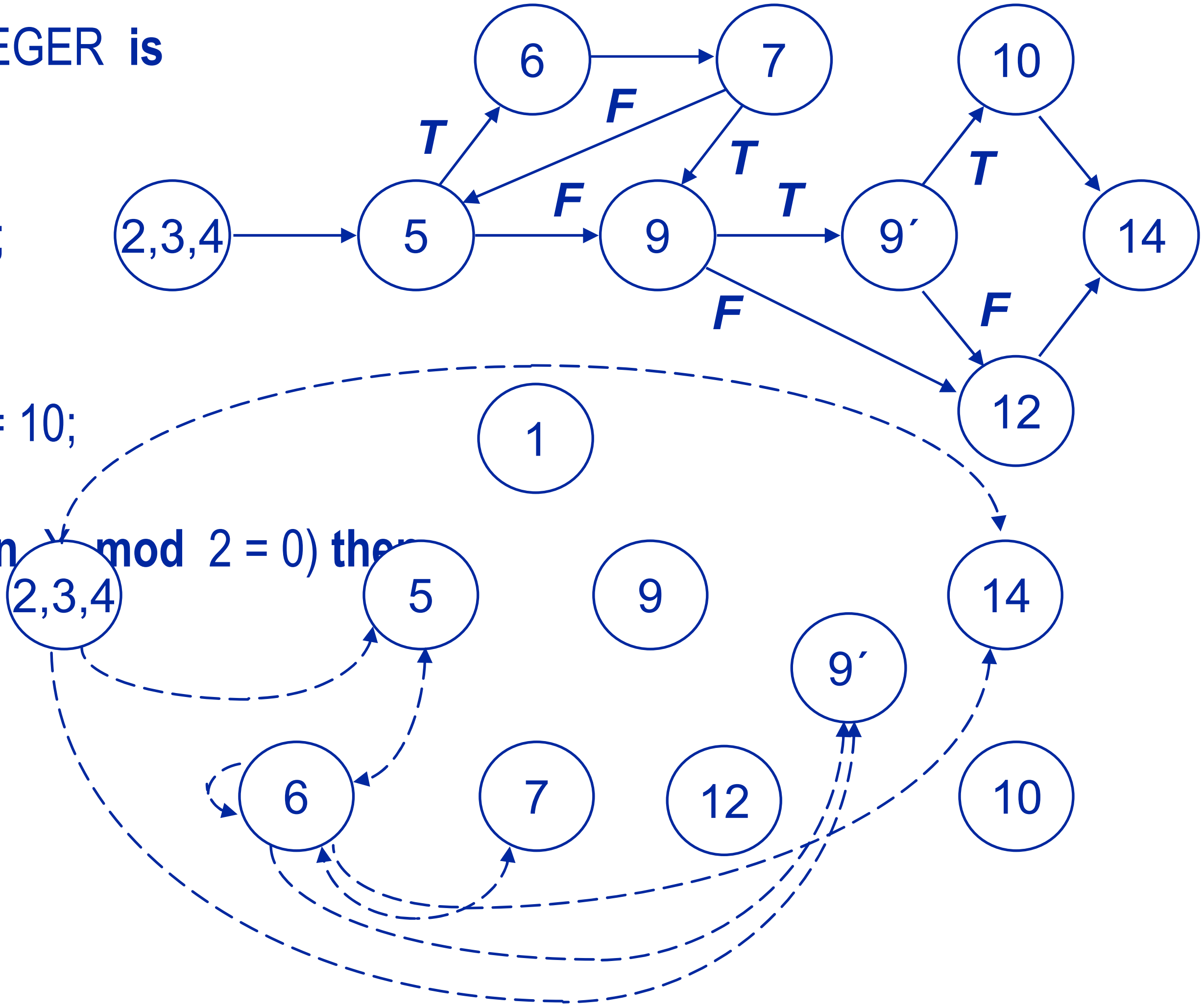
      and
    - » Y does not post-dominate X

Data Dependence Graph (DDG)

- Informal definition
  - two statements are data dependent if they might reference the same memory location and one of the references is an assignment to the memory location
  - *intuition:* if the statements cannot be switched without affecting the program, then they are data dependent

A Sample Program

```
1  function  P  return  INTEGER  is
2  begin
3         X, Y: INTEGER;
4         READ(X); READ(Y);
5         while (X > 10) loop
6                X := X – 10;
7                exit  when  X = 10;
8         end  loop;
9         if (Y <  20  and  then  Y  mod  2 = 0) then
10               Y := Y + 20;
11        else
12               Y := Y – 20;
13        end  if;
14        return  2 ∗ X + Y;
15 end  P;
```



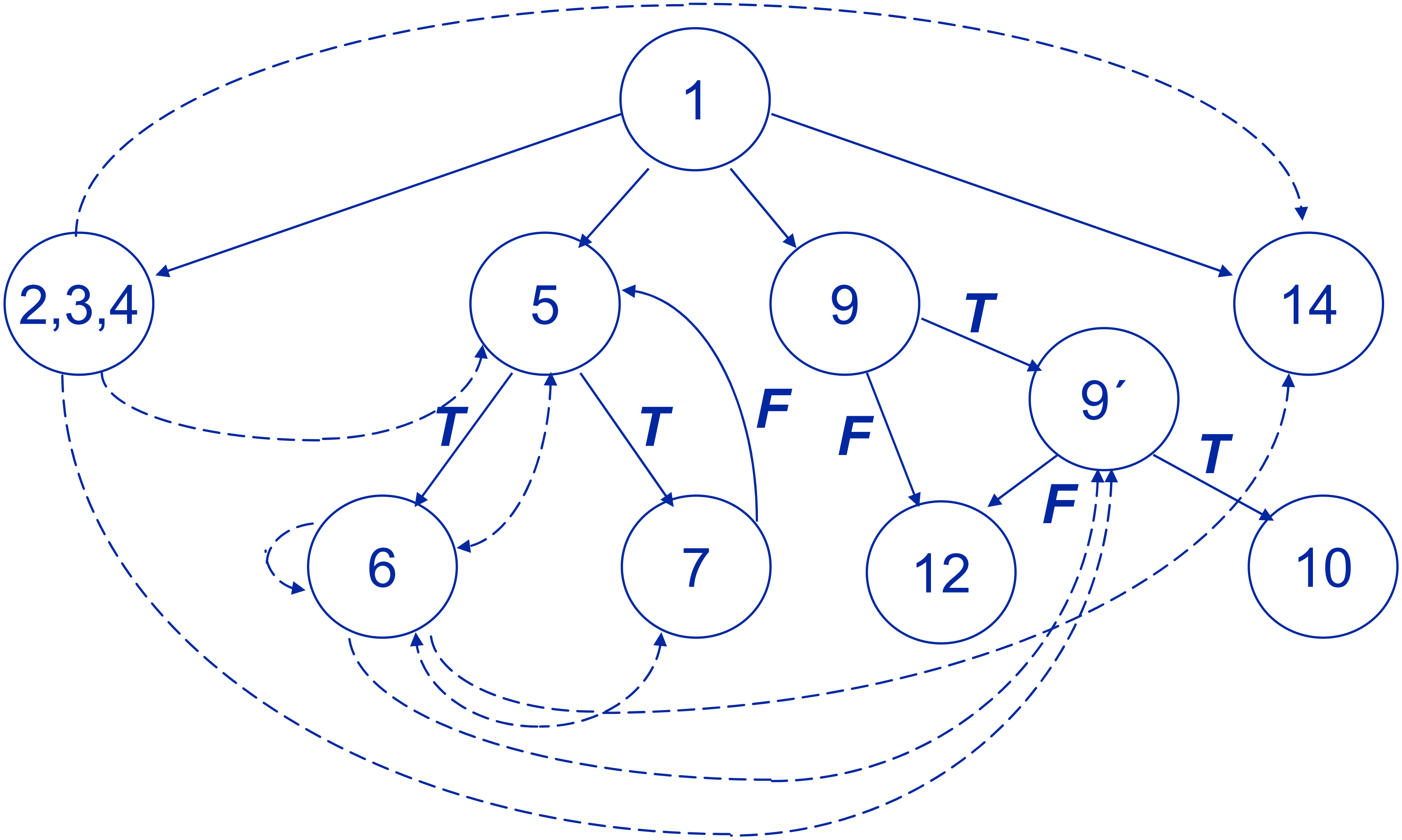
Data Dependence Graph (DDG)

- Formal definition
  - let X and Y be nodes in a CFG
  - there is a data dependence from X to Y with respect to a variable v iff there is a non-null path p from X to Y with no intervening definition of v and either:
    - » X contains a definition of v and Y a use of v

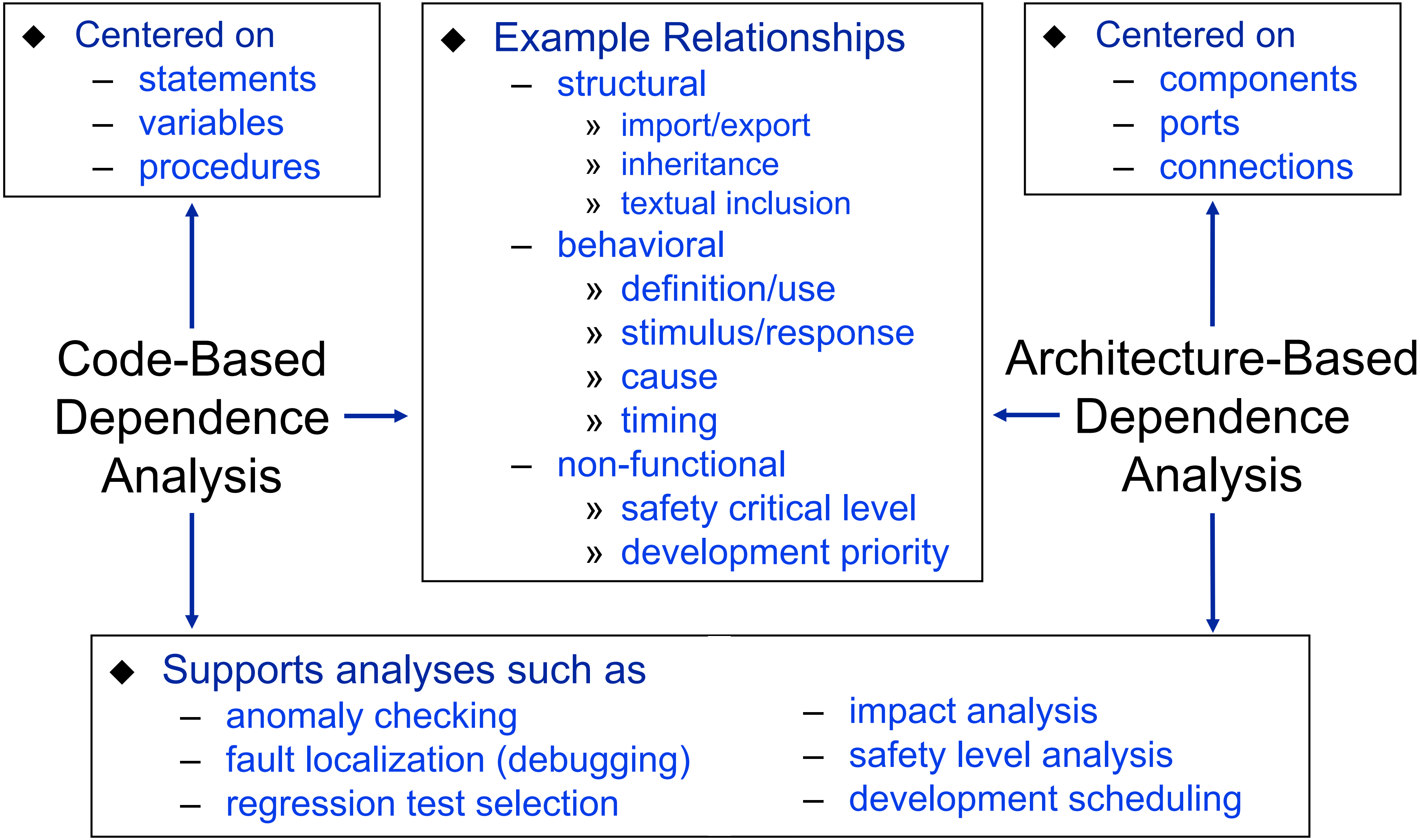
      or
    - » X contains a use of v and Y a definition of v

      or
    - » X contains a definition of v and Y a definition of v

P’s PDG (DDG for X Only)



Dependence Analysis Comparison



Code-Based Dependence Analysis

- Centered on
  - statements
  - variables
  - procedures

- Example Relationships
  - structural
    - » import/export
    - » inheritance
    - » textual inclusion
  - behavioral
    - » definition/use
    - » stimulus/response
    - » cause
    - » timing
  - non-functional
    - » safety critical level
    - » development priority

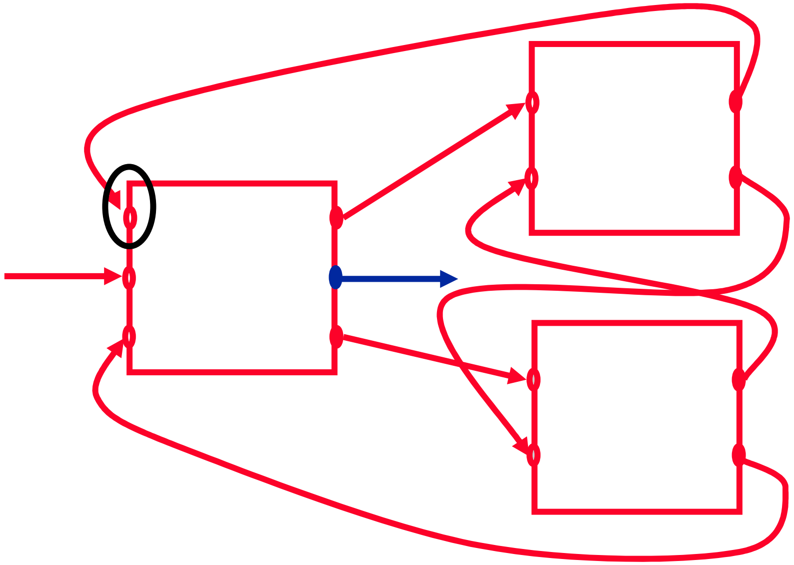
Architecture-Based Dependence Analysis

- Centered on
  - components
  - ports
  - connections

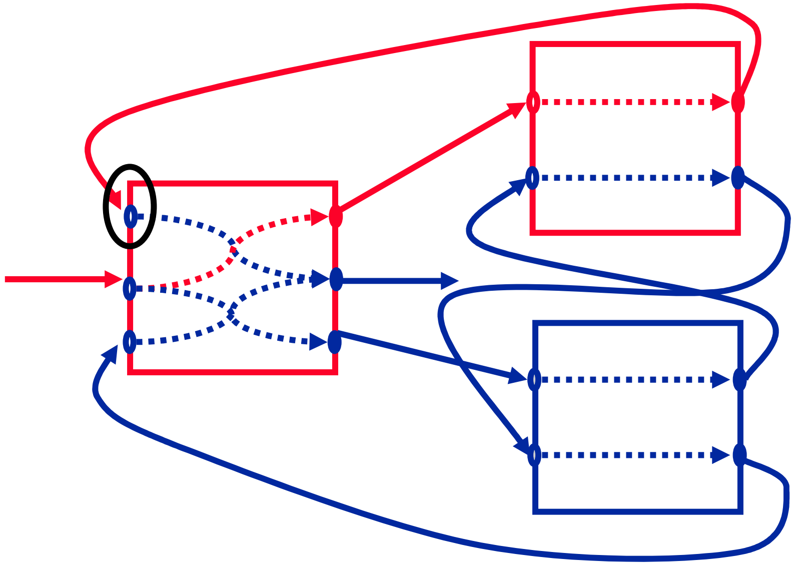
- Supports analyses such as
  - anomaly checking
  - fault localization (debugging)
  - regression test selection
  - impact analysis
  - safety level analysis
  - development scheduling

Refining Analysis of Architectures

- Strictly structural view



- Add behavioral connections



Conservative → **Precise**

Aladdin: A Tool for Architecture Analysis

- Implements a technique called *chaining*

- Supports architectural queries including:
  - are there ports that are ignored or neglected?
  - what ports could directly affect or be affected by a particular port?
  - what ports could indirectly affect or be affected by a particular port?

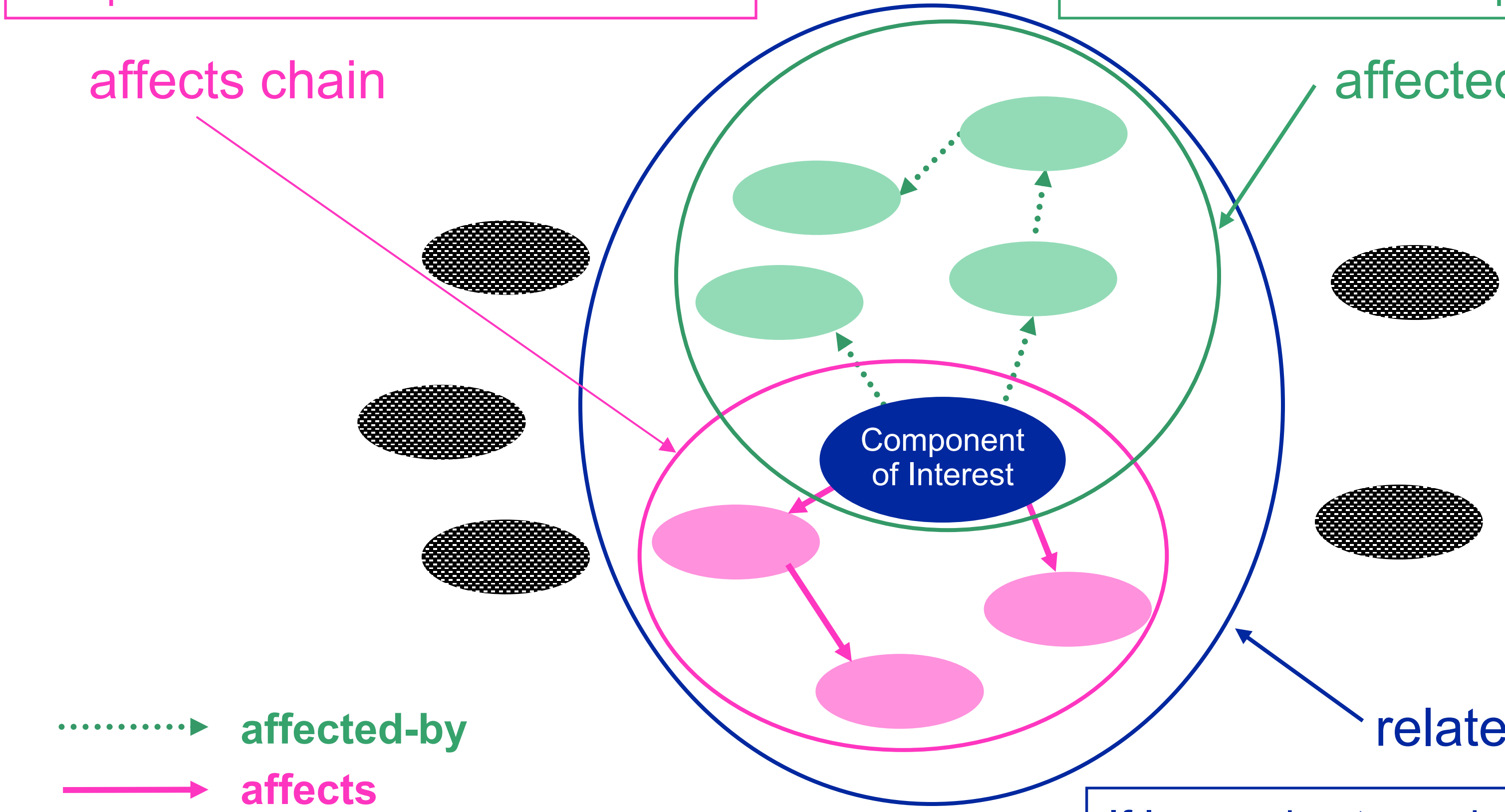
Chaining

- A ***link*** represents a *direct* dependence between two components
- A ***chain*** represents the *indirect* and *direct* relationships among components
- ***Chaining*** is...
  - the construction of chains to answer questions about software architectures
  - a means for performing software architecture dependence analysis

A Component-Centric View of Chains

If this component is replaced, what components will need to be retested?

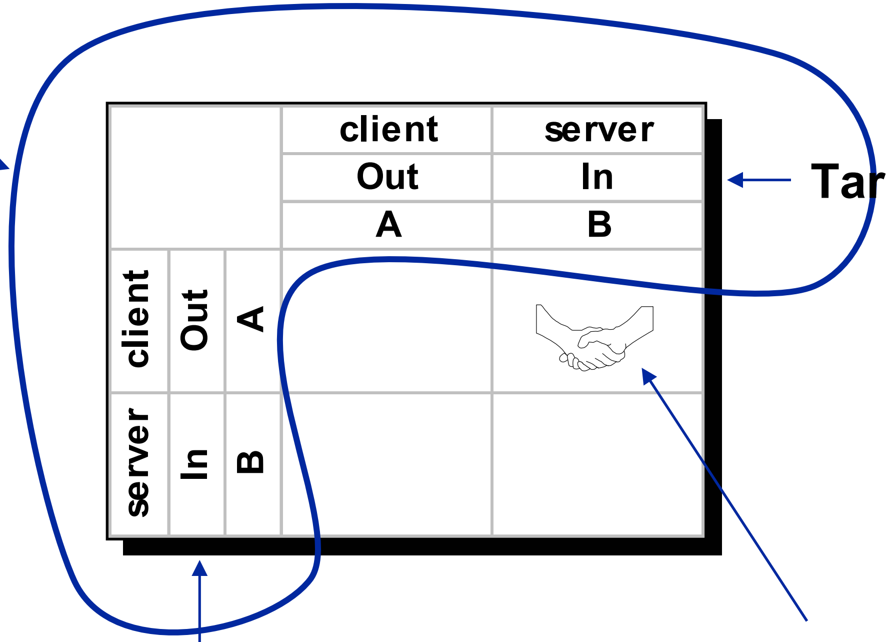
What components could have contributed to a failure in this component?



If I am going to work on this component, what other components do I want to have immediately available?

Tabular Representation

Table frame is built by recording the ports



Targets

Sources

Relationships are recorded in the cells

			client	server
			Out	In
			A	B
client	Out	A		🤝
server	In	B		

ADL Specification

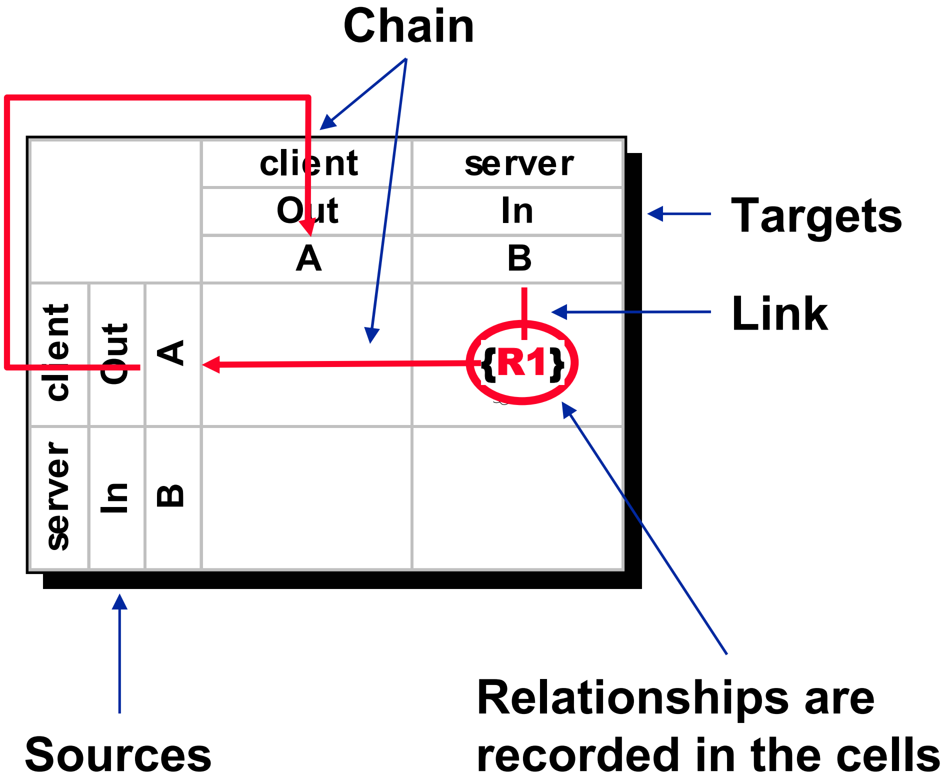
```
component Client
 { out: A;
   behavior
        send A; }

component Server
 { in: B;
   behavior
        when B then DOSOMETHING; }

architecture Client-Server {
  server: Server;
  client: Client;
  connect
    client.A => server.B; }
```

Links and Chains

- Choose a port and a relationship
- Perform transitive closure over the links

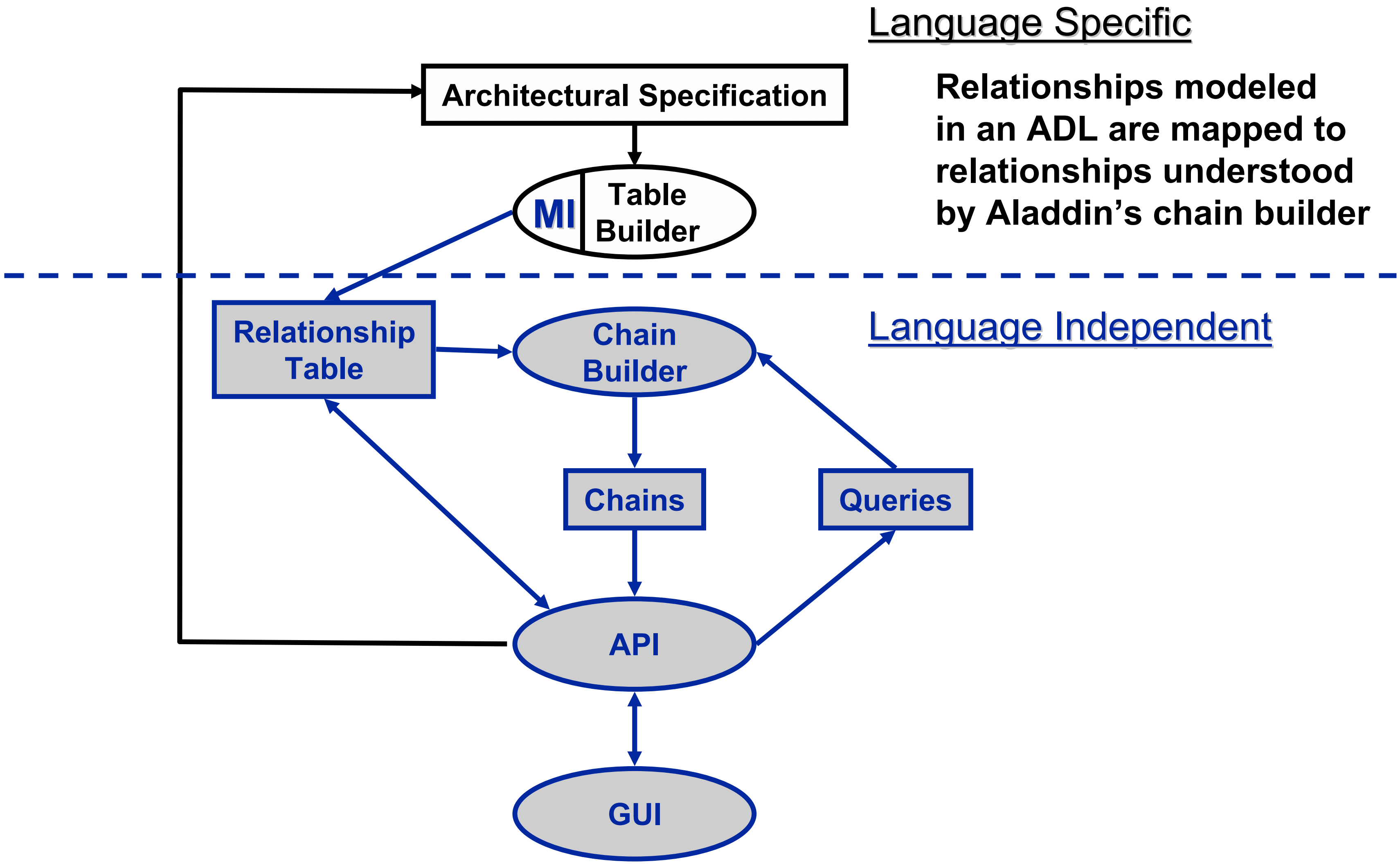
ADL Specification

```
component Client
 { out: A;
   behavior
      send A; }

component Server
 { in: B;
   behavior
      when B then DOSOMETHING; }

architecture Client-Server {
  server: Server;
  client: Client;
  connect
    client.A => server.B; }
```



			client	server
			Out	In
			A	B
client	Out	A		{R1}
server	In	B		

Aladdin Architecture



Language Specific

Relationships modeled in an ADL are mapped to relationships understood by Aladdin's chain builder

Language Independent

Example: Gas Station

- Rapide specification
  - 1 operator, 1 pump, and 2 customers
- Aladdin analyses
  - anomaly checking
    - are there any ports that are neglected or ignored?
  - fault localization
    - why can’t the second customer refuel?
  - impact analysis
    - which components could be affected by a change to the pump?

Rapide Specification for Gas Station

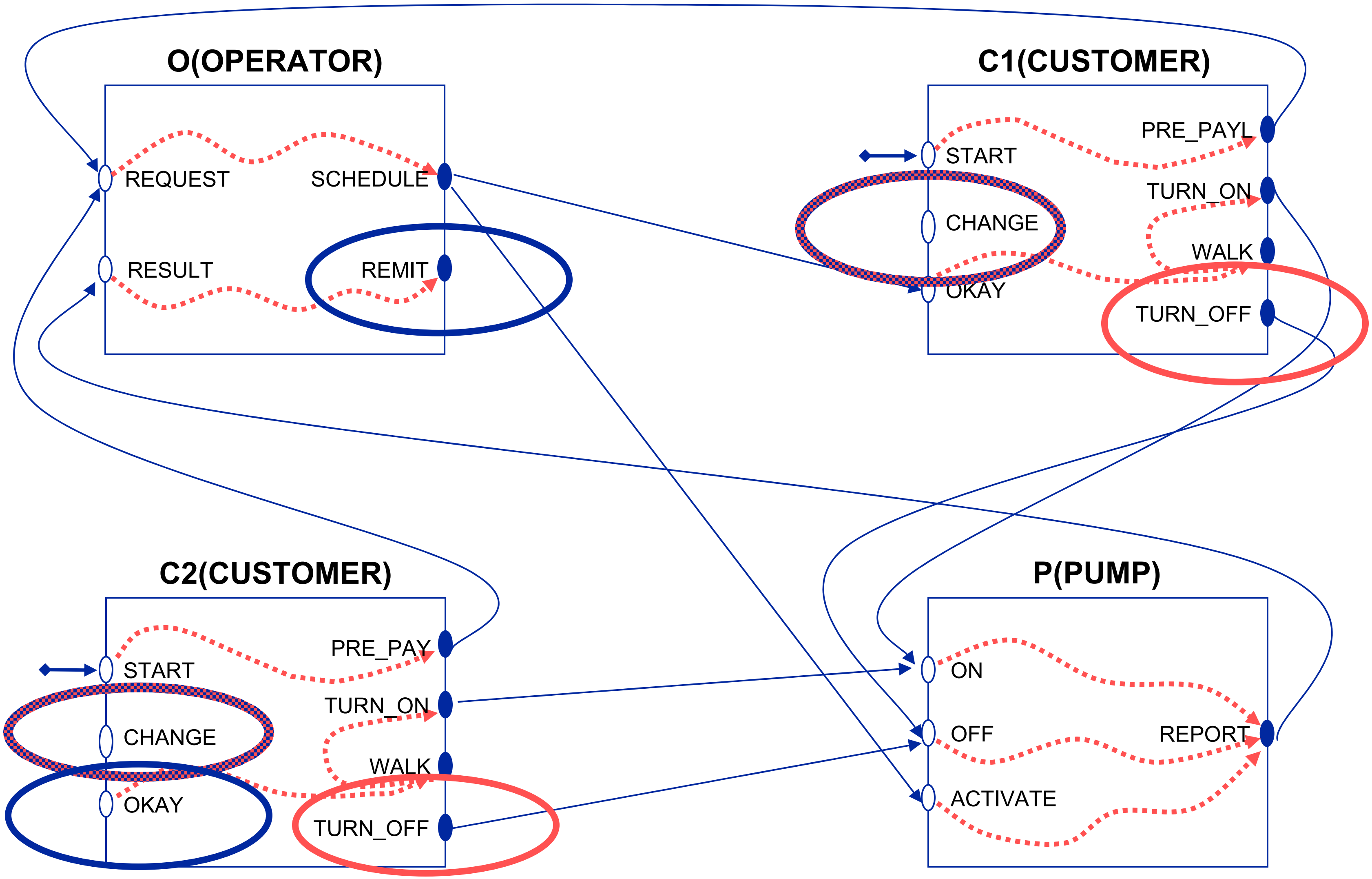
```
type Dollars is integer; - enum 0, 1, 2, 3 end enum;
type Gallons is integer; - enum 0, 1, 2, 3 end enum;

type Customer is interface
action in     Okay(), Change(Cost : Dollars);
         out   Pre_Pay(Cost : Dollars), Okay(), Turn_On(), Walk(), Turn_Off();
behavior
         D : Dollars is 10;
begin
         start   ||> Pre_Pay(D
         Okay ||> Walk;;
         Walk  ||> Turn_On;;
end Customer;
```

```
    type Operator is interface
```

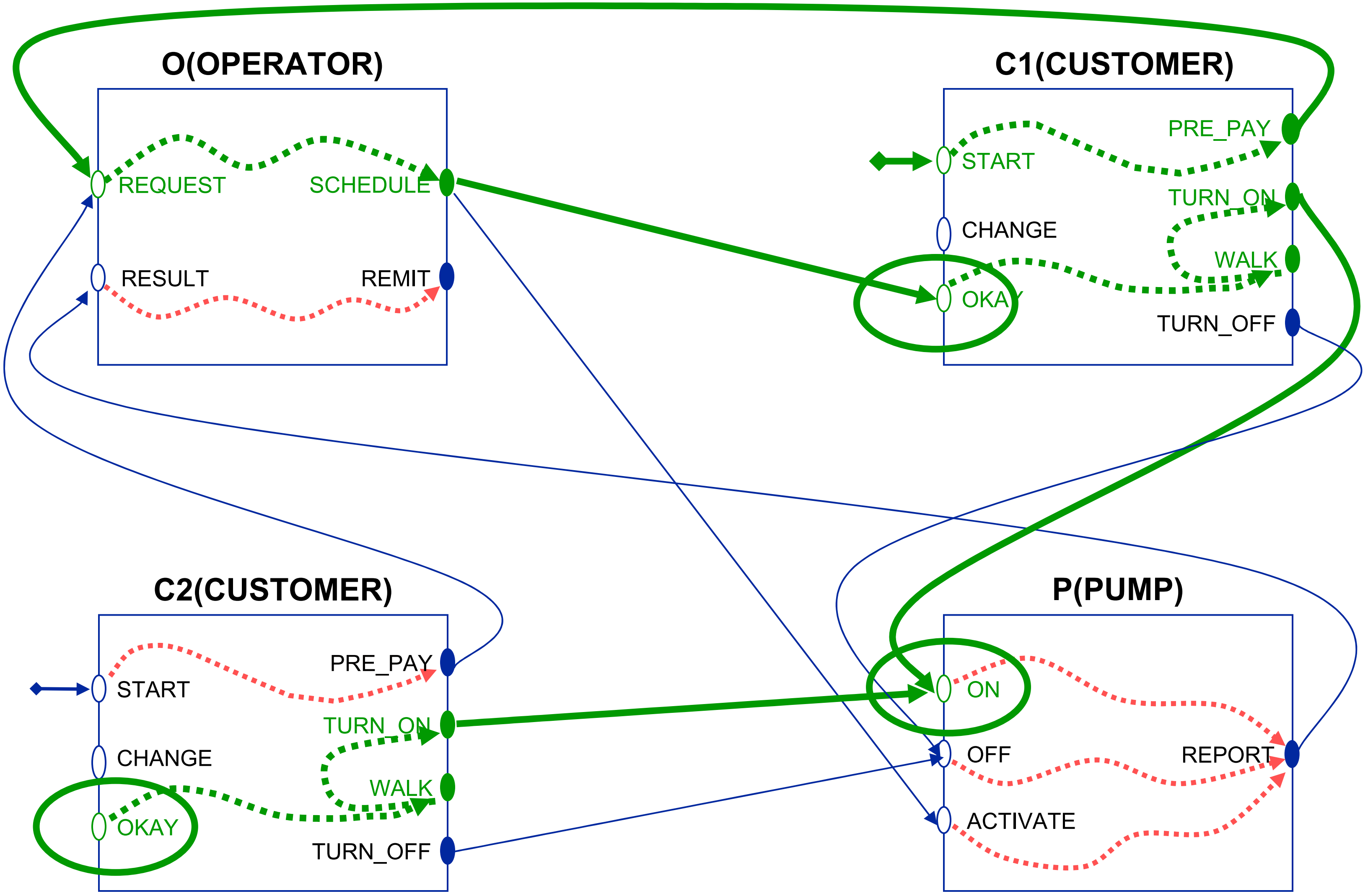
```
    type Pump is interface
```

```
architecture gas_station() return root is
    O : Operator;
    P : Pump;
    C1, C2 : Customer;
  connect
    (?C : Customer; ?X : Dollars) ?C.Pre_Pay(?X) ||> O.Request(?X);
    (?X : Dollars) O.Schedule(?X) ||> P.Activate(?X);
    (?X : Dollars) O.Schedule(?X) ||> C1.Okay;
    (?C : Customer) ?C.Turn_On ||> P.On;
    (?C : Customer) ?C.Turn_Off ||> P.Off;
    (?X : Gallons; ?Y : Dollars)P.Report(?X, ?Y) ||> O.Result(?Y);
  end gas_station;
```

Gas Station Anomalies



Gas Station Fault Localization



Summarizing Local Behavior

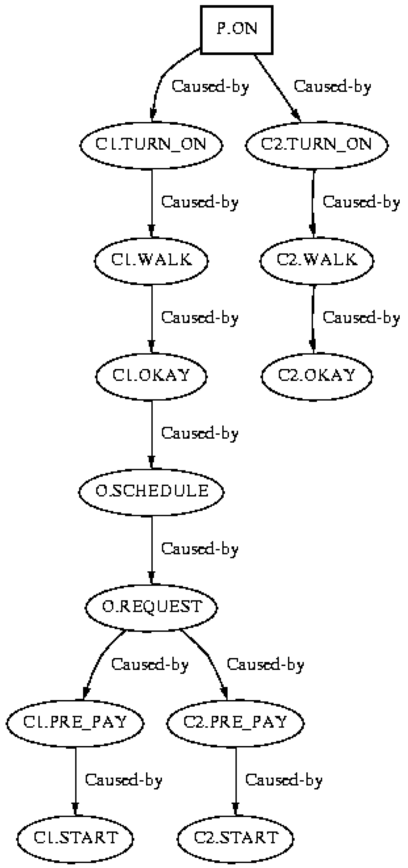
```
type Pump is interface
action in     On(), Off(), Activate(Cost : Dollars);
        out   Report(Amount: Gallons, Cost : Dollars), ;
behavior
        Free: var Boolean := True;
        Reading, Limit : var Dollars := 0;
        action In_Use(), Done();
begin
        (?X: Dollars)(On ~ Activate(?X)) where $Free ||> Free := False; Limit:= ?X; In_Use;;
        In_Use ||> Reading := $Limit; Done;;
        Off or Done ||> Free := True; Report($Reading);;

end Pump;
```

internal events

- Q: How can we ignore details of internal events?
- A: Conservatively relate internal stimulus events back to some external stimulus event, and internal stimulus event forward to external out actions

Architecture Debugging

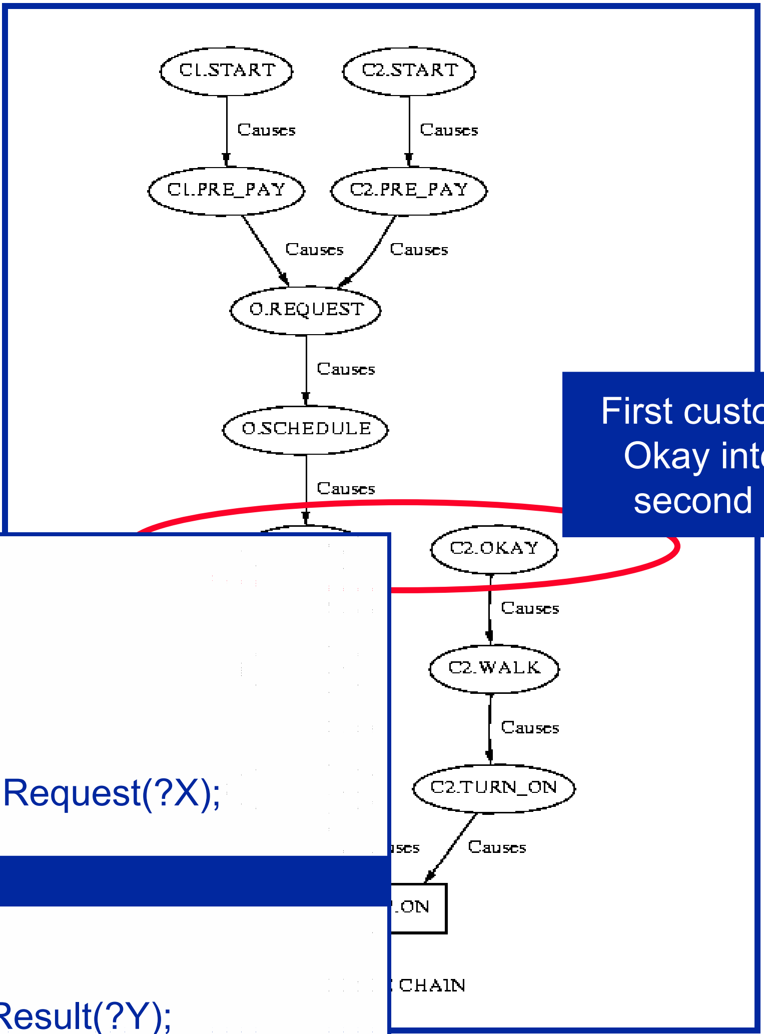
Why is it that the second customer can never pump gas?



AFFECTED-BY CHAIN

Architecture Debugging

Why is it that the second customer can never pump gas?



First customer gets Okay intended for second customer

```
architecture gas_station() return root is
    O : Operator;
    P : Pump;
    C1, C2 : customer;
  connect
    (?C : customer; ?X : Dollars) ?C.Pre_Pay(?X) ||> O.Request(?X);
    (?X : Dollars) O.Schedule(?X) ||> P.Activate(?X);
    (?X : Dollars) O.Schedule(?X) ||> C1.Okay;
    (?C : customer) ?C.Turn_On ||> P.On;
    (?C : customer) ?C.Turn_Off ||> P.Off;
    (?X : Gallons; ?Y : Dollars) P.Report(?X, ?Y) ||> O.Result(?Y);
  end gas_station;
```

Tabular Representation of Gas Station

			Operator				Pump				Customer1							Customer2						
			Out		In		Out	In			Out				In			Out				In		
			Sch	Rem	Req	Res	Rep	On	Off	Act	PP	T_On	Walk	T_Off	Okay	Chg	start	PP	T_On	Walk	T_Off	Okay	Chg	start
OPERATOR	Out	Schedule								||>					||>									
		Remit																						
	In	Request	||>																					
		Result		||>																				
Pump	Out	Report				||>																		
	In	On					||>																	
		Off					||>																	
		Activate					||>																	
Customer1	Out	Pre_Pay																						
		Turn_On						||>																
		Walk										||>												
		Turn_Off							||>															
	In	Okay											||>											
		Change																						
		Start									||>													
Customer2	Out	Pre_Pay			||>																			
		Turn_On						||>																
		Walk																	||>					
		Turn_Off							||>															
	In	Okay																		||>				
		Change																						
		Start																||>						

||> Rapide *agent* connection: Models new thread of control for each triggering

Connection Anomaly

			Operator				Pump				Customer1							Customer2						
			Out		In		Out	In			Out				In			Out				In		
			Sch	Rem	Req	Res	Rep	On	Off	Act	PP	T_On	Walk	T_Off	Okay	Chg	start	PP	T_On	Walk	T_Off	Okay	Chg	start
OPERATOR	Out	Schedule								||>					||>									
		Remit																						
	In	Request	||>																					
		Result		||>																				
Pump	Out	Report				||>																		
	In	On					||>																	
		Off					||>																	
		Activate					||>																	
Customer1	Out	Pre_Pay																						
		Turn_On						||>																
		Walk										||>												
		Turn_Off							||>															
	In	Okay											||>											
		Change																						
		Start									||>													
Customer2	Out	Pre_Pay			||>																			
		Turn_On						||>																
		Walk																	||>					
		Turn_Off							||>															
	In	Okay																		||>				
		Change																						
		Start																||>						

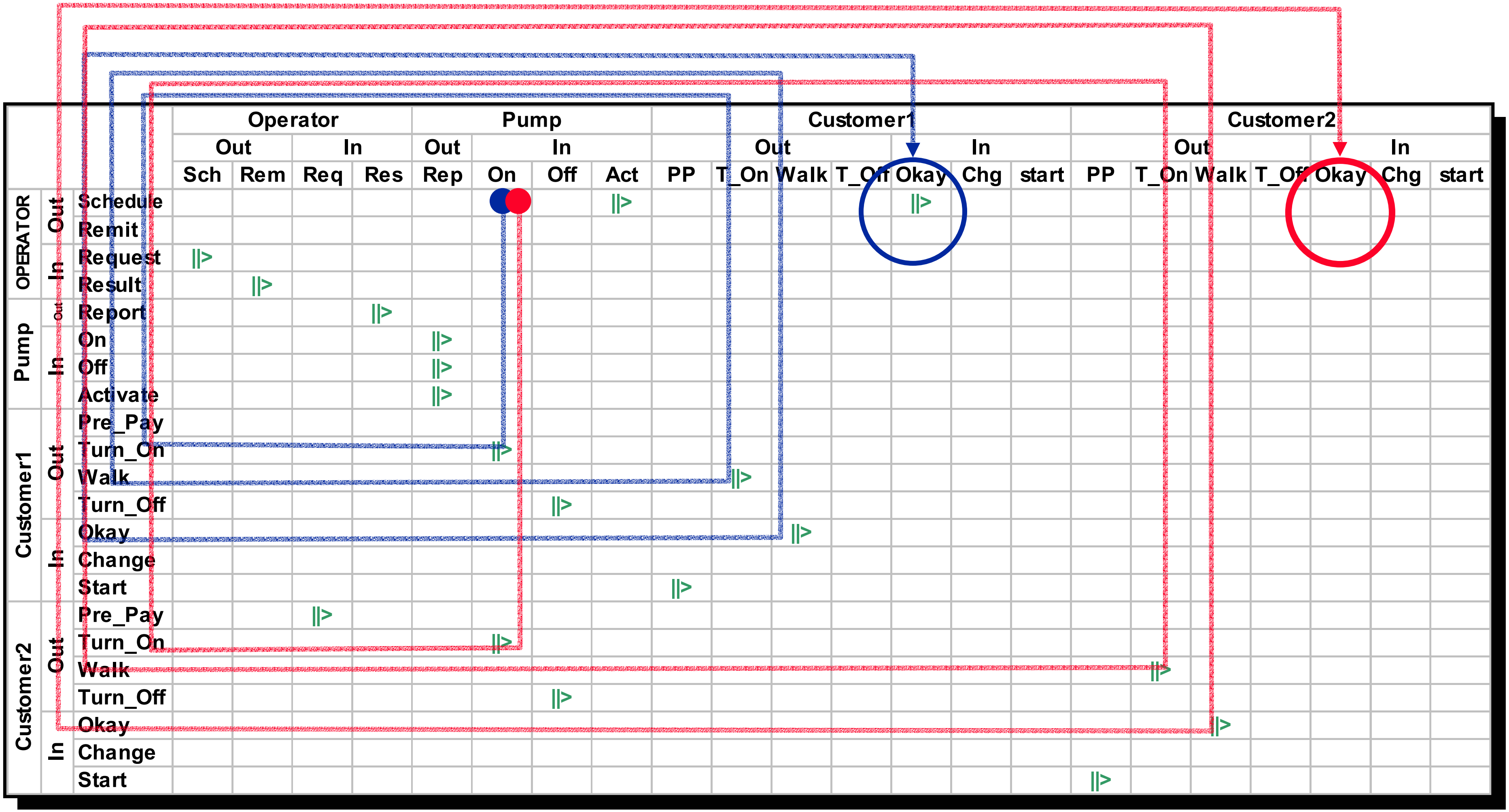
||> Rapide *agent* connection: Models new thread of control for each triggering

Impact Analysis

			Operator				Pump				Customer1							Customer2						
			Out		In		Out	In			Out				In			Out				In		
			Sch	Rem	Req	Res	Rep	On	Off	Act	PP	T_On	Walk	T_Off	Okay	Chg	start	PP	T_On	Walk	T_Off	Okay	Chg	start
OPERATOR	Out	Schedule								||>					||>									
		Remit																						
	In	Request	||>																					
		Result		||>																				
Pump	Out	Report ●				||>																		
	In	On					||>																	
		Off					||>																	
		Activate					||>																	
Customer1	Out	Pre_Pay																						
		Turn_On						||>																
		Walk										||>												
		Turn_Off							||>															
	In	Okay											||>											
		Change																						
		Start									||>													
Customer2	Out	Pre_Pay			||>																			
		Turn_On						||>																
		Walk																	||>					
		Turn_Off							||>															
	In	Okay																		||>				
		Change																						
		Start																||>						

||> Rapide *agent* connection: Models new thread of control for each triggering

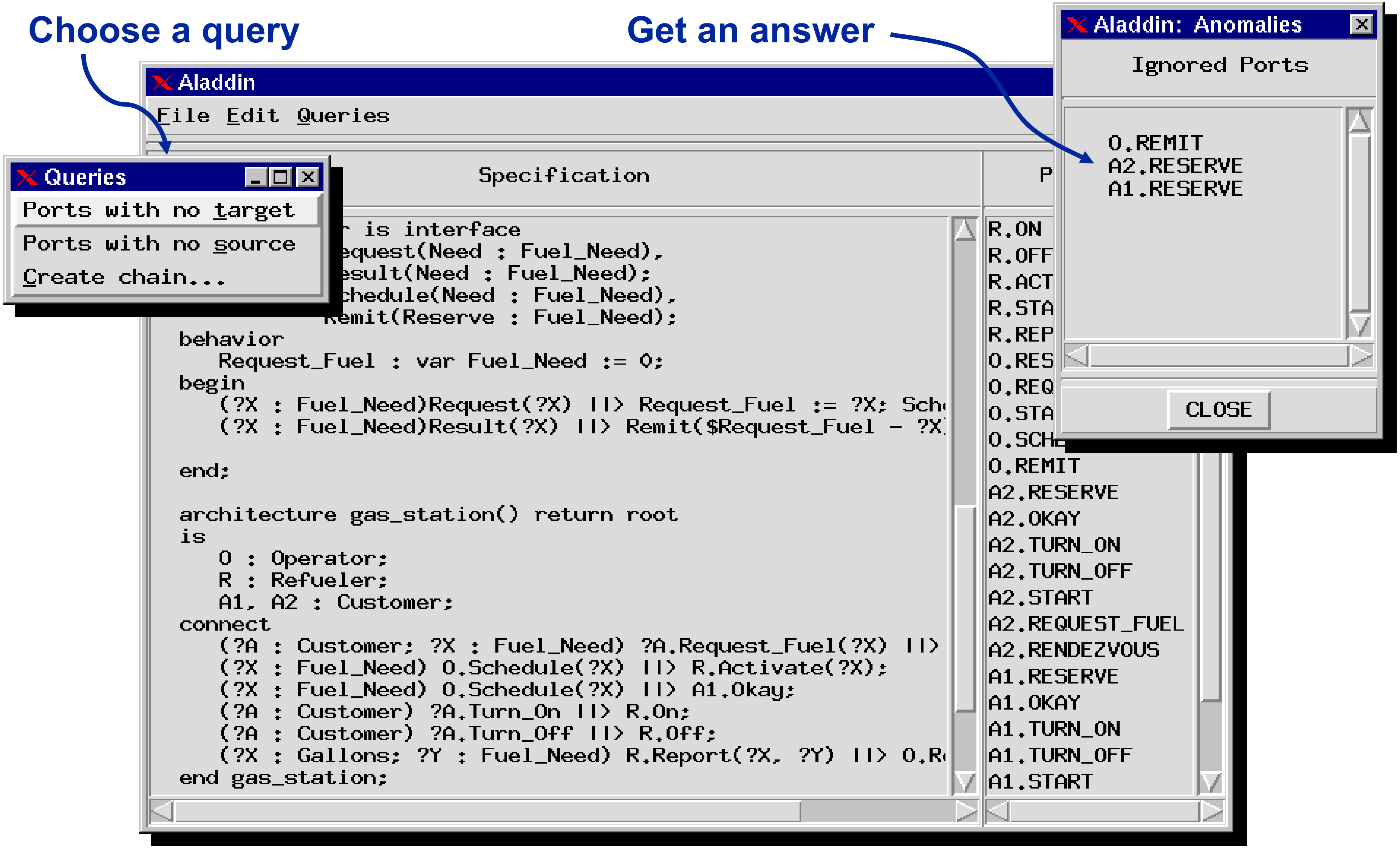
Architecture Debugging



			Operator				Pump				Customer1							Customer2						
			Out		In		Out	In			Out					In		Out					In	
			Sch	Rem	Req	Res	Rep	On	Off	Act	PP	T_On	Walk	T_Off	Okay	Chg	start	PP	T_On	Walk	T_Off	Okay	Chg	start
OPERATOR	Out	Schedule						●●		||>					||>									
		Remit																						
	In	Request	||>																					
		Result		||>																				
Pump	Out	Report				||>																		
	In	On					||>																	
		Off					||>																	
		Activate					||>																	
Customer1	Out	Pre_Pay																						
		Turn_On						||>																
		Walk										||>												
		Turn_Off							||>															
		Okay											||>											
	In	Change																						
		Start									||>													
Customer2	Out	Pre_Pay			||>																			
		Turn_On						||>																
		Walk																	||>					
		Turn_Off							||>															
		Okay																		||>				
	In	Change																						
		Start																||>						

||> Rapide *agent* connection: Models new thread of control for each triggering

Aladdin User Interface



```
                   r is interface
                   equest(Need : Fuel_Need),
                   esult(Need : Fuel_Need);
                   chedule(Need : Fuel_Need),
                  Remit(Reserve : Fuel_Need);
behavior
   Request_Fuel : var Fuel_Need := 0;
begin
   (?X : Fuel_Need)Request(?X) ||> Request_Fuel := ?X; Sch
   (?X : Fuel_Need)Result(?X) ||> Remit($Request_Fuel - ?X

end;

architecture gas_station() return root
is
   O : Operator;
   R : Refueler;
   A1, A2 : Customer;
connect
   (?A : Customer; ?X : Fuel_Need) ?A.Request_Fuel(?X) ||>
   (?X : Fuel_Need) O.Schedule(?X) ||> R.Activate(?X);
   (?X : Fuel_Need) O.Schedule(?X) ||> A1.Okay;
   (?A : Customer) ?A.Turn_On ||> R.On;
   (?A : Customer) ?A.Turn_Off ||> R.Off;
   (?X : Gallons; ?Y : Fuel_Need) R.Report(?X, ?Y) ||> O.R
end gas_station;
```

```
R.ON
R.OFF
R.ACT
R.STA
R.REP
O.RES
O.REQ
O.STA
O.SCH
O.REMIT
A2.RESERVE
A2.OKAY
A2.TURN_ON
A2.TURN_OFF
A2.START
A2.REQUEST_FUEL
A2.RENDEZVOUS
A1.RESERVE
A1.OKAY
A1.TURN_ON
A1.TURN_OFF
A1.START
```

```
O.REMIT
A2.RESERVE
A1.RESERVE
```

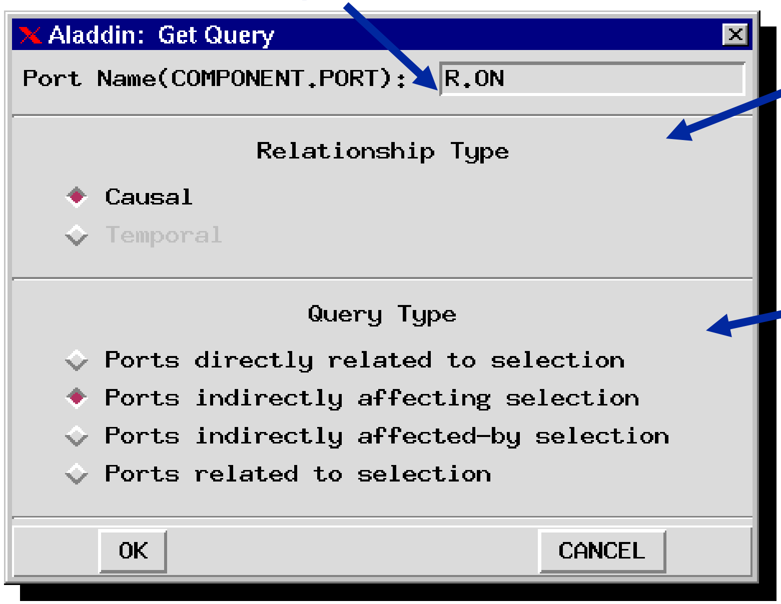
Result of Aladdin Query

Architecture Debugging:

- Second Customer is not allowed to refuel
- Aladdin helps locate the fault in the specification

Choose a port

Select a relationship type



Aladdin: Get Query

Port Name(COMPONENT.PORT): R.ON

Relationship Type

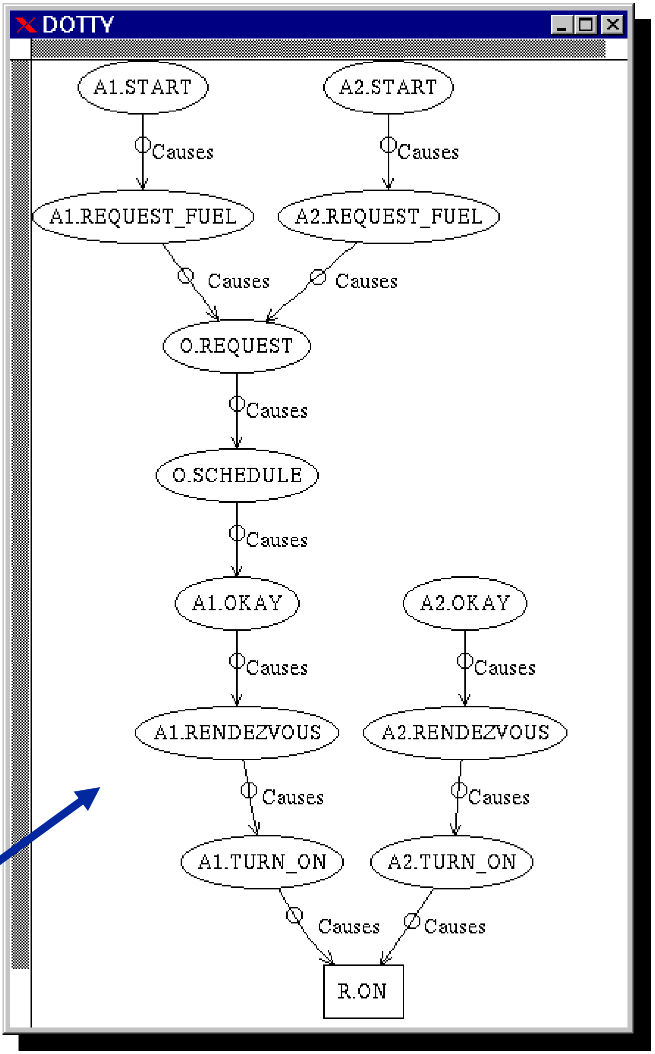
● Causal
○ Temporal

Query Type

○ Ports directly related to selection
● Ports indirectly affecting selection
○ Ports indirectly affected-by selection
○ Ports related to selection

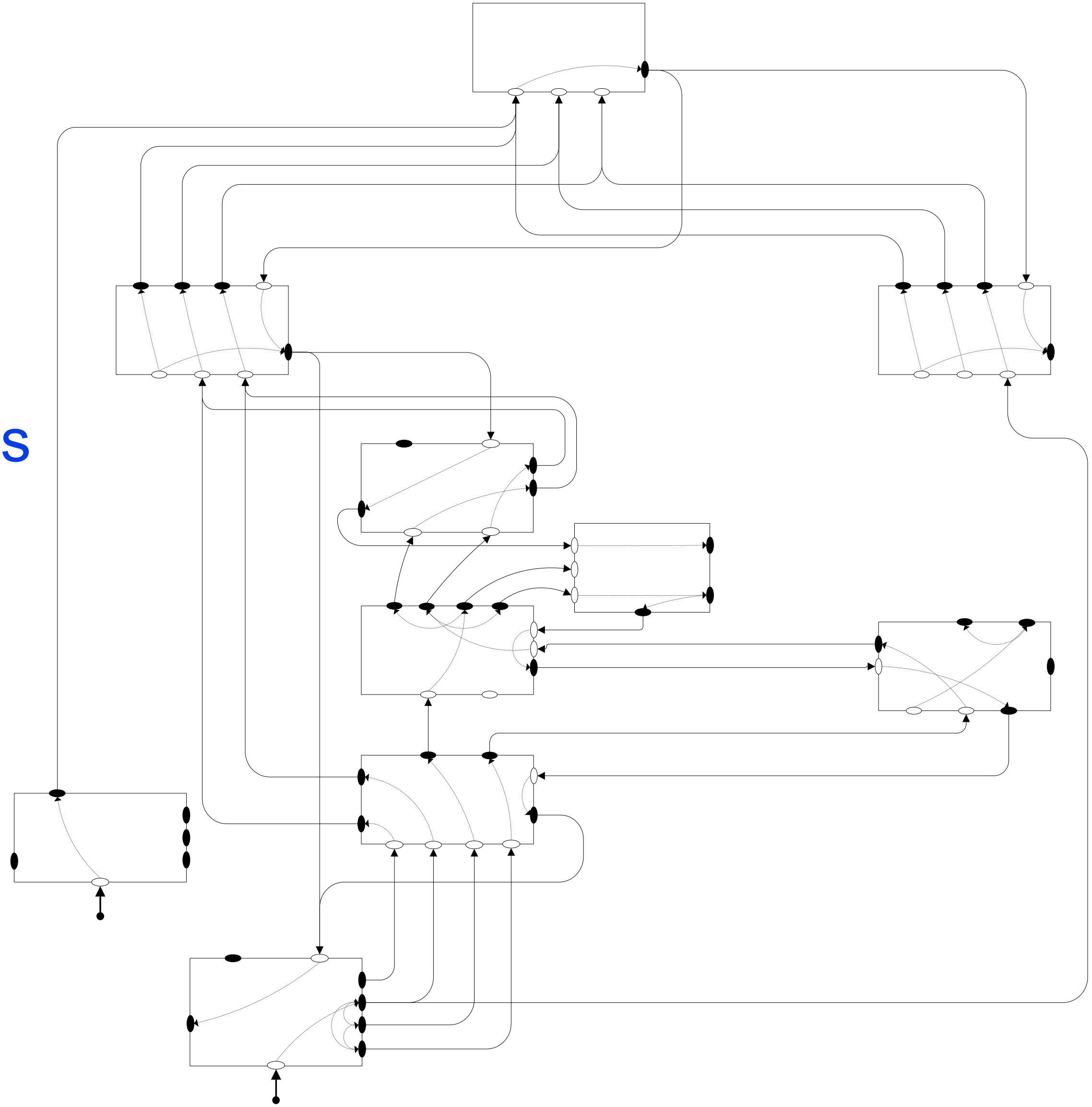
Select a query type

Study the resulting chain

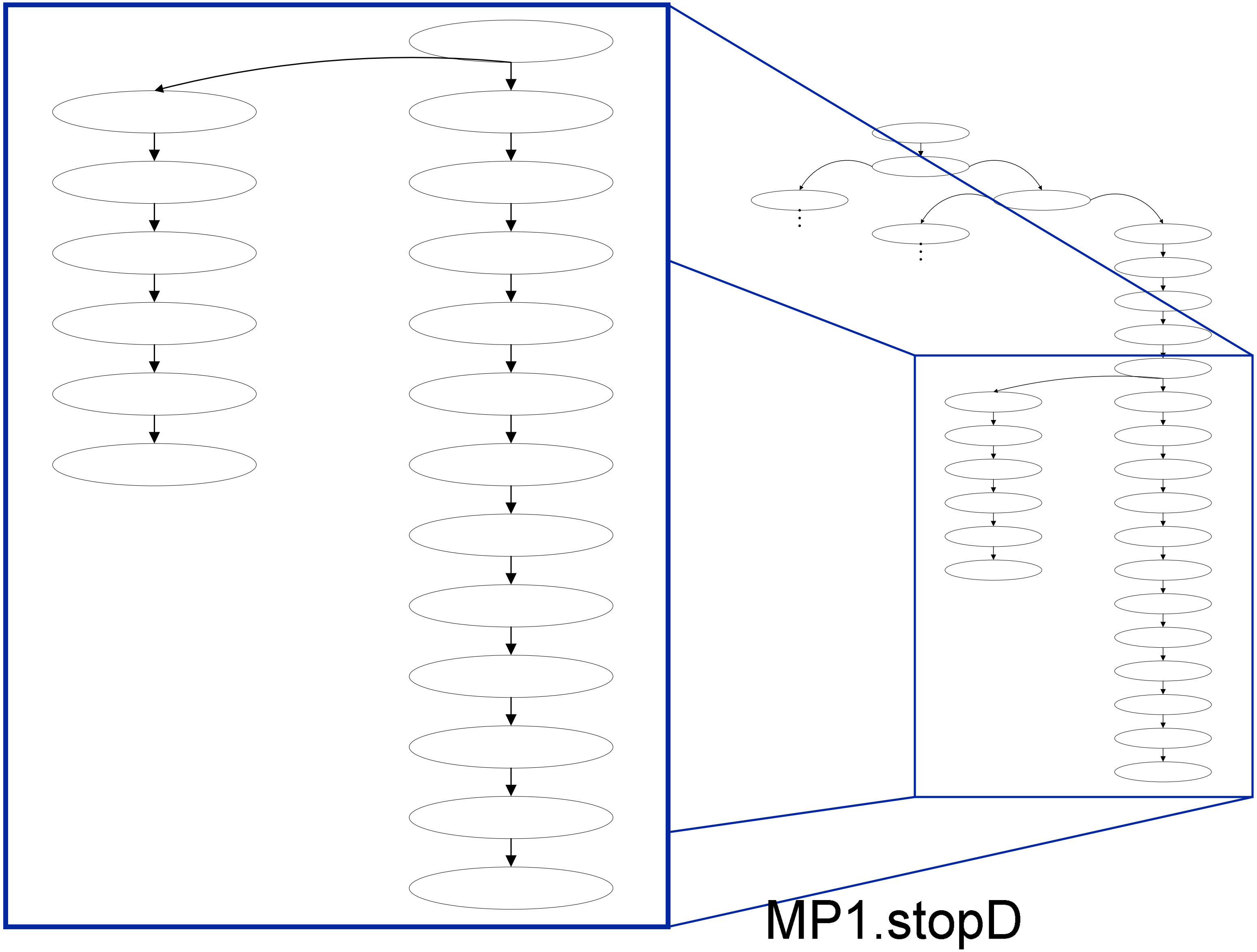


Another Example: MobiKit

- Publish/subscribe mobility service
- Architectural questions
  - Which components of the system contribute to the “event download” functionality?
  - Does the system behave as expected?



Chain Derived from Rapide Specification



MP1.stopD

Problem in MobiKit Specification

- Lack of coordination between “move out” and “move in” mobility operations
  - client can perform “move in” before “move out” completed
  - can be attributed to asynchronous behavior as shown by the parallel chains
- Components needing further examination
  - *Subscriber*, *MobiKit-Client*, *MobiKit-Proxy1*, *MobiKit-Proxy2*, *Queue*, *Dummy*, *S0,* and *S1*
  - revealed by affects chain

Aladdin Summary

- Current status
  - analyzes Rapide and informal specifications
  - provides a generic interface for other ADLs
  - performs analysis for causal relationships
- Future plans
  - make use of other relationships (e.g., timing)
  - leverage other features of Rapide to increase precision of chains (e.g., patterns, constraints)
  - include query for cycles in the architecture
  - incorporate table builders for other ADLs

Some Related Work

- A. Podgurski and L.A. Clarke
  - formalized program dependence analysis
- O. O'Malley and D.J. Richardson
  - program dependence analysis tools (ProDAG, TAOS)
- A.M. Sloane and J. Holdsworth
  - slicing of non-imperative programs
- J. Chang and D.J. Richardson
  - specification slicing
- G. Naumovich, G.S. Avrunin, L.A. Clarke, and L.J. Osterweil
  - software architecture concurrency analysis
- J. Zhao
  - software architecture slicing

Architectural Slicing [Zhao]

- An analysis technique applied to formal architectural specifications
- An architectural slice is a subset of behaviors
- Intended to isolate the behavior of a specific set of component or connectors

Architectural Slicing and Program Slicing

- Program slice
  - consists of those parts of a program that may directly or indirectly affect the values computed at some program point of interest
- Program slicing
  - a decomposition technique that extracts program elements related to a particular computation
  - an application of dependence analysis

More on Program Slicing

- Concerned with code written in conventional programming languages
  - applied to variables and statements
- Usual definition
  - a ***slicing criterion*** is a pair *(s,V)*, where *s* is a statement and *V* is a set of variables defined or used at *s*
  - a slice consists of only statements
- Expensive to compute and of questionable utility

Architectural Slicing as a Tool

- Takes as input a formal architectural specification *P* of a software system
- Removes from the specification those components and interconnections that are not necessary for maintaining the semantics of the software architecture
- Returns as output a "sub-architecture" *S*

Architectural Slicing: A Definition

- Given an architectural specification

$$P = (C_m, C_n, c_g)$$

where

C_m: set of components

C_n: set of connectors

c_g: configuration of P

an ***architectural slice*** $S_p = (C'_m, C'_n, c'_g)$ is a sub-architecture of P that partially preserves the semantics of P

Elements of a Design Entity

- Component entity
  - ports and computations
- Connector entity
  - roles and glue
- Configuration entity
  - instances and attachments

Reductions on Entities

- Let $P = (C_m, C_n, c_g)$ be an architectural specification and $c_m \in C_m$, $c_n \in C_n$
  - a ***reduced component*** of c_m is a component c'_m that is derived from c_m by removing zero or more elements from c_m
  - a ***reduced connector*** of c_c is a connector c'_n that is derived from c_n by removing zero or more elements from c_n
  - a ***reduced configuration*** of c_g is a configuration c'_g that is derived from c_g by removing zero or more elements from c_g

Reduced Architectural Specification

- Let $P = (C_m, C_n, c_g)$ and $P' = (C'_m, C'_n, c'_g)$ be two architectural specifications
  - P' is a ***reduced architectural specification*** of P if

    $C'_m=\{c'_{m_1}, c'_{m_2},\ldots, c'_{m_k}\}$ is a subset of $C_m=\{c_{m_1}, c_{m_2},\ldots, c_{m_k}\}$ such that for $i = 1,2,\ldots,k$, c'_{m_i} is a reduced component of c_{m_i}

    $C'_n=\{c'_{n_1}, c'_{n_2},\ldots, c'_{n_k}\}$ is a subset of $C_n=\{c_{n_1}, c_{n_2},\ldots, c_{n_k}\}$ such that for $i = 1,2,\ldots,k$, c'_{n_i} is a reduced component of c_{n_i}

    c'_g is a reduced configuration of c_g

Slicing Criterion

- Defines a starting point for the slice

- Let $P = (C_m, C_n, c_g)$ be an architectural specification
  - a ***slicing criterion*** for P is a pair (c, E) such that $c \in C_m$ and E is a set of port elements of c, or $c \in C_n$ and E is a set of roles elements of c

Backward Slicing

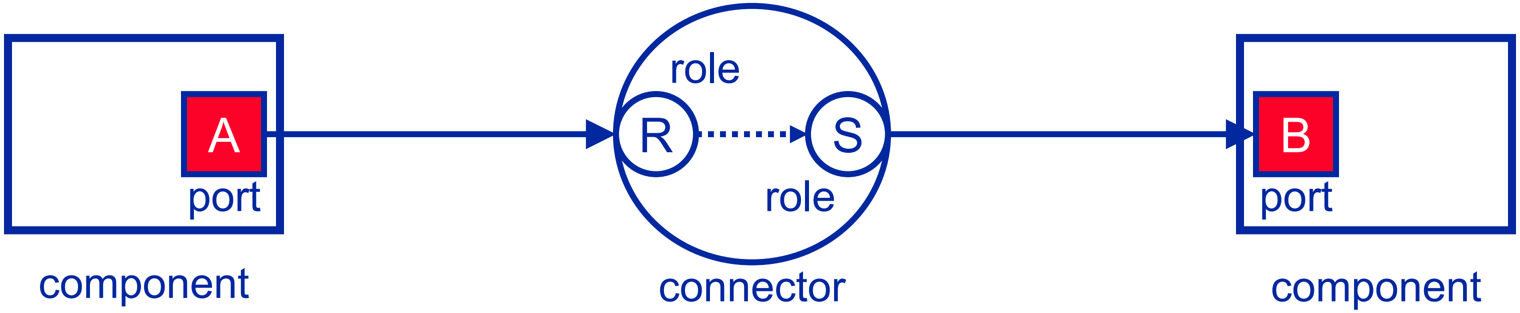
- Let $P = (C_m, C_n, c_g)$ be an architectural specification
  - a ***backward architectural slice*** $S_{b_p}=(C'_m,C'_n,c_g)$ of P using a give slicing criterion (c,E) is a reduced architectural specification of P that contains only those reduced components, connectors, and configuration that might directly or indirectly ***affect*** the behavior of c through elements in E

Forward Slicing

- Let $P = (C_m, C_n, c_g)$ be an architectural specification
  - a ***forward architectural slice*** $S_{f_p}=(C'_m,C'_n,c_g)$ of P using a give slicing criterion (c,E) is a reduced architectural specification of P that contains only those reduced components, connectors and configuration that might be directly or indirectly ***affected by*** the behavior of c through elements in E

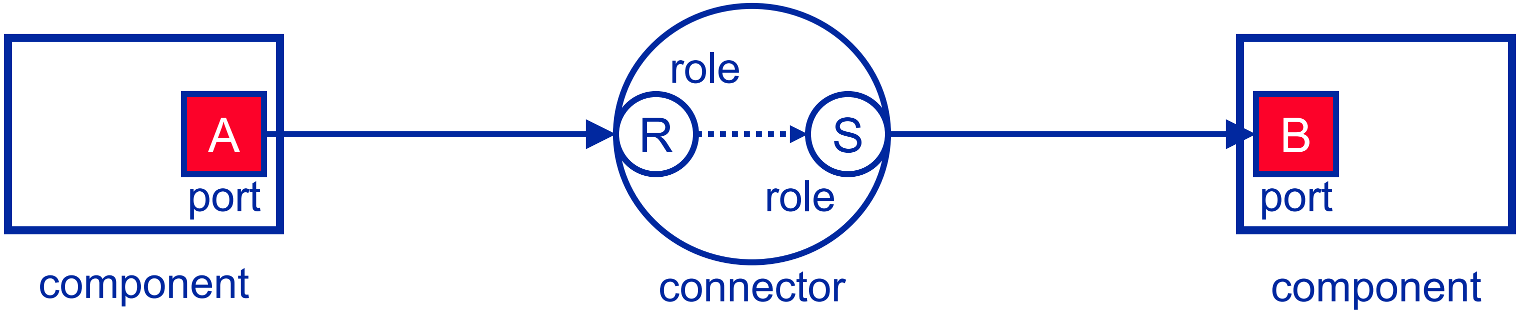
Architectural Slicing: Data Structure

- Architecture Information Flow Graph (AIFG)
  - a digraph whose vertices represent the ports of components and the roles of connectors in an architectural specification
  - arcs represents possible information flows between components and/or connectors in the specification



Data Structure Definition

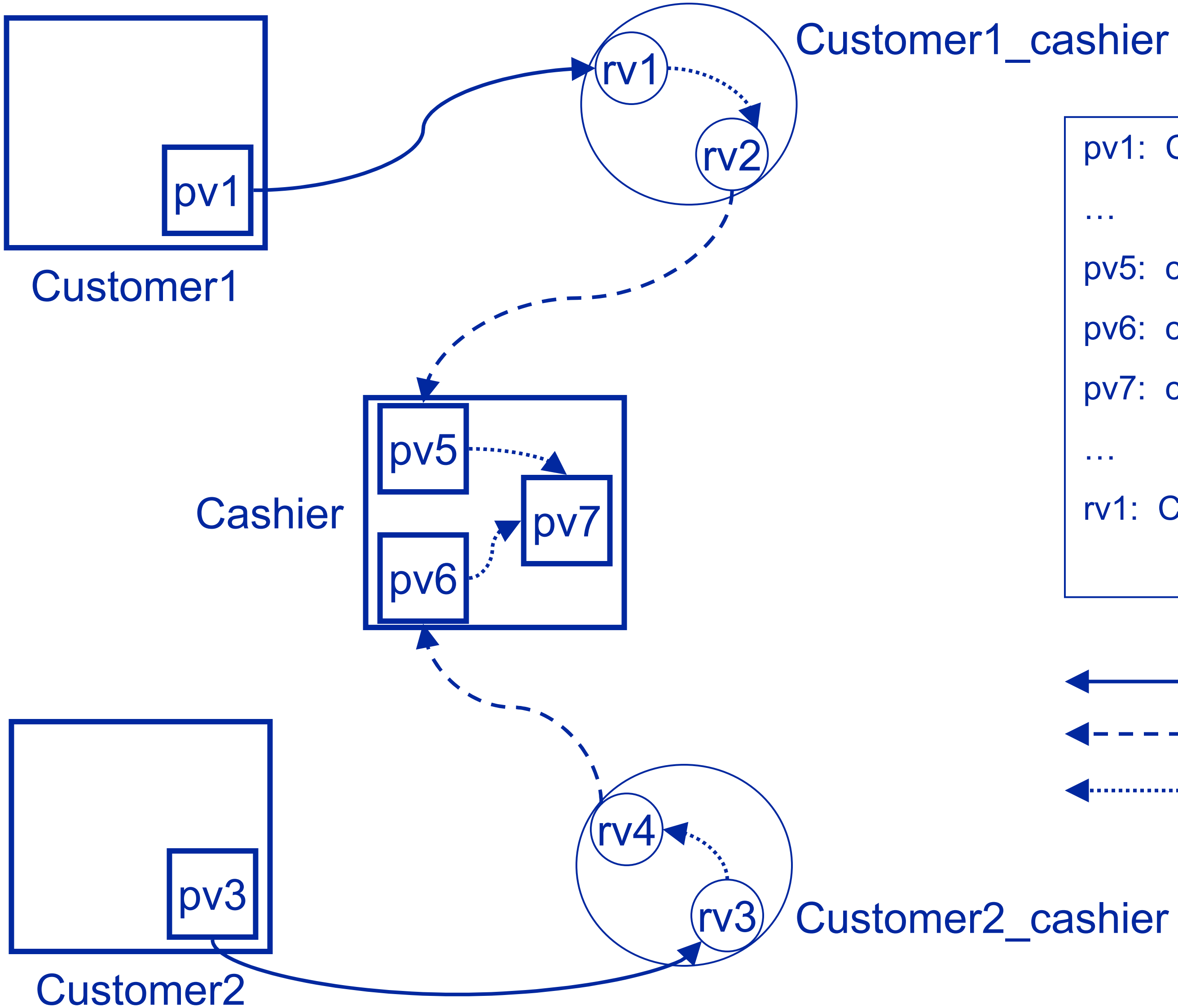
- The AIFG of architectural specification *P* is a digraph G=$(V_{com}, V_{con}, Com, Con, Int)$ where
  - V_{com} is the set of port vertices of *P*
  - V_{con} is the set of role vertices of *P*
  - *Com* is the set of component-connector flow arcs
  - *Con* is the set of connector-component flow arcs
  - *Int* is the set of internal flow arcs



Computing an Architectual Slice

- Amounts to walks over the AIFG
- Two steps
  1. compute forward and backward dependence relationships
  2. reduce the architectural description by removing non-dependent elements

Gas Station Example



pv1: Customer.Pay

…

pv5: cashier.Customer1

pv6: cashier.Customer2

pv7: cashier.ToPump

…

rv1: Customer_cashier.givemoney

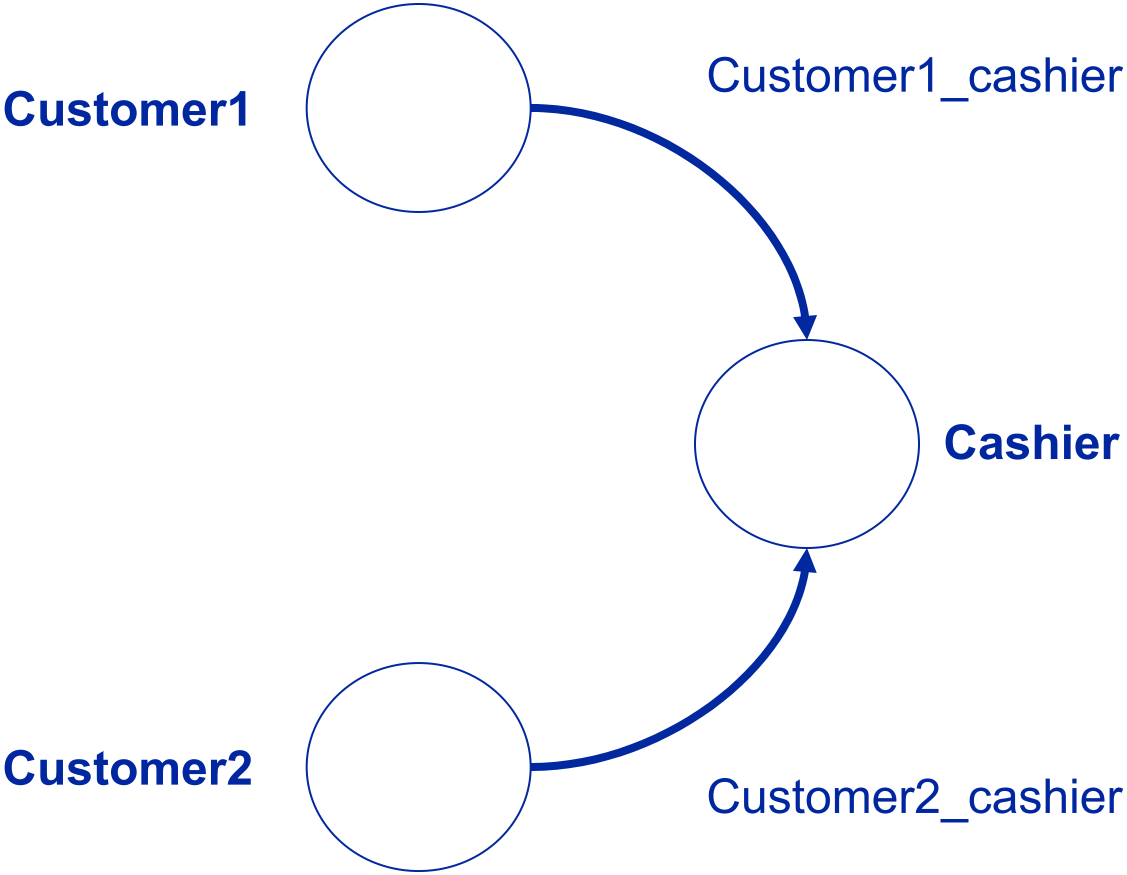
◀—— Component-connector

◀- - - Connector-component

◀········ Internal

Gas Station Slice

Slicing Criterion: (cashier, *E*)
such that *E*={Customer1, Customer2, ToPump}



Concluding Thoughts

- Dependence analysis is a powerful technique
- Formal architecture description lends itself to dependence analysis
- But what is the method that guides its use?
  - What ***relationships*** are really of interest?
  - What ***types of analyses*** can architecture-based dependence analysis support?
  - How can we create and maintain precise, bi-directional, inter-level ***mappings*** between the architecture and the implementation?

Concluding Thoughts (cont.)

- Architecture-level dependence analysis tends to be conceived in traditional terms
  - sequential, deterministic control and data flow
- Architecture description languages tend to be conceived in non-traditional terms
  - event interactions and patterns
  - concurrency and asynchronous communication
  - constraints on connections/state transitions
- How do we re-conceptualize dependence analysis to fit?