An Approach to Practical Validation of Control Software Specification

Tomoyuki Kaga
Toyota Motor Co.

Masakazu Adachi
Toyota Central R&D Labs

Contents of this talk

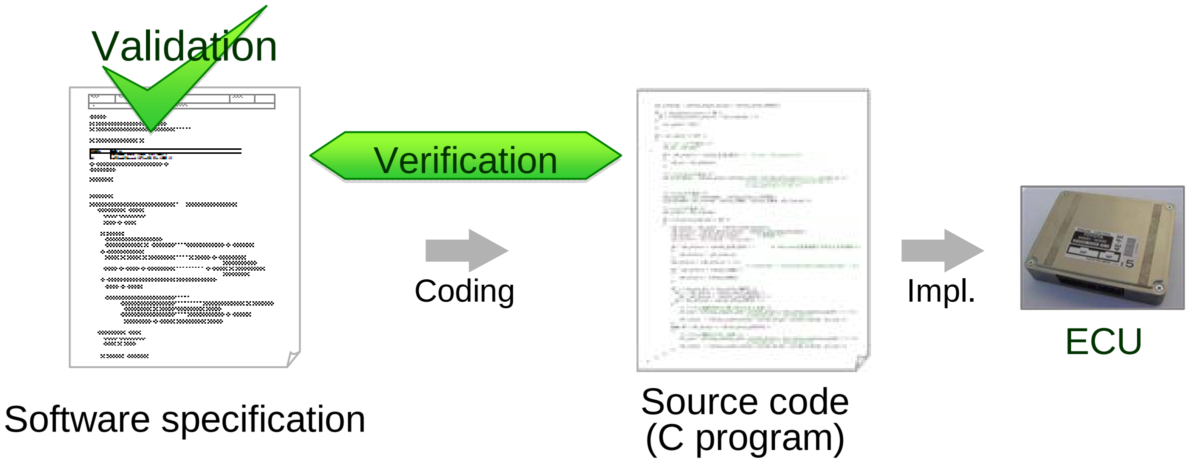
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction

Definition of Verification and Validation

Standard definitions

- ☐ Verification
  - Confirmation by examination and provision of objective evidence that specified requirements and are fulfilled
  - Ensuring implementation satisfies the requirements for that step. Can include testing, analysis and review
  - You built product right
- ☐ Validation
  - Confirmation by examination and provision of objective evidence that particular requirements for a specific intended use are fulfilled
  - Ensuring requirements are complete and correct
  - You built the right product

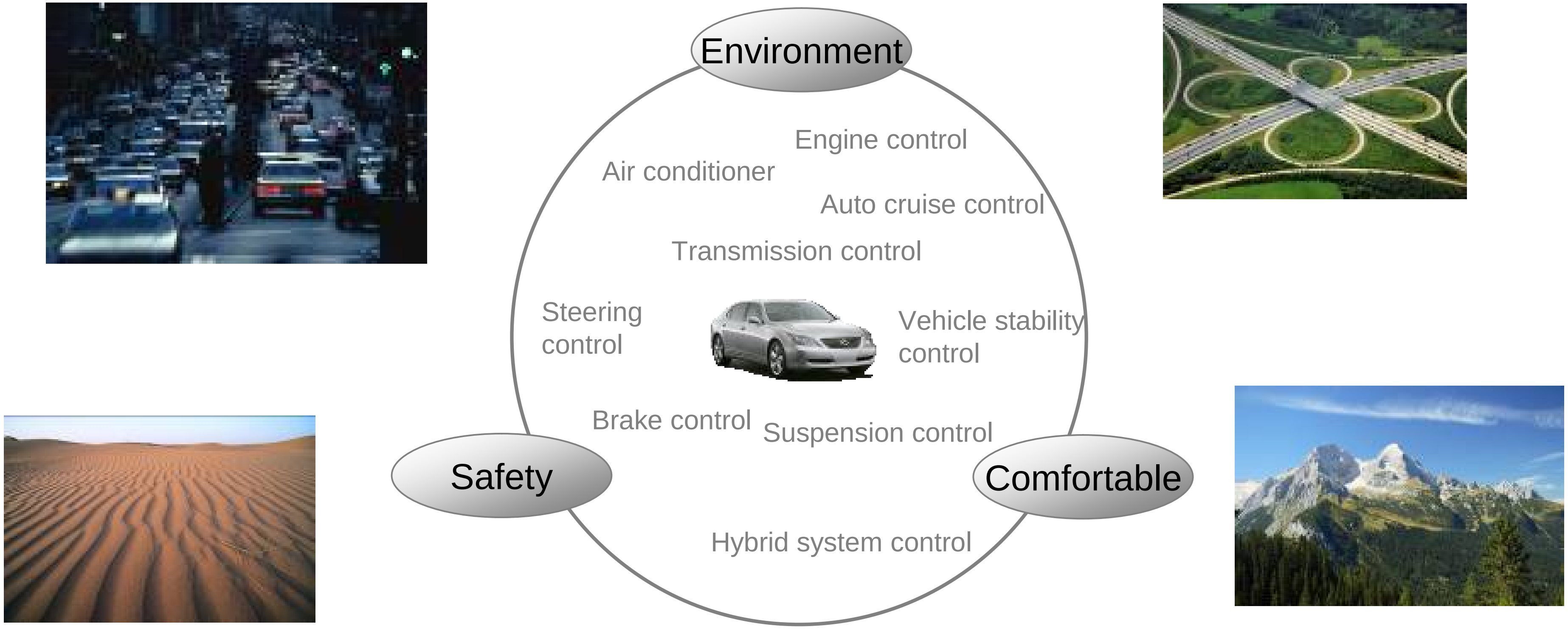
Definition in this talk



1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction

Automotive control system and software

Major automotive control systems



Environment

Safety

Comfortable

Engine control

Air conditioner

Auto cruise control

Transmission control

Steering control

Vehicle stability control

Brake control

Suspension control

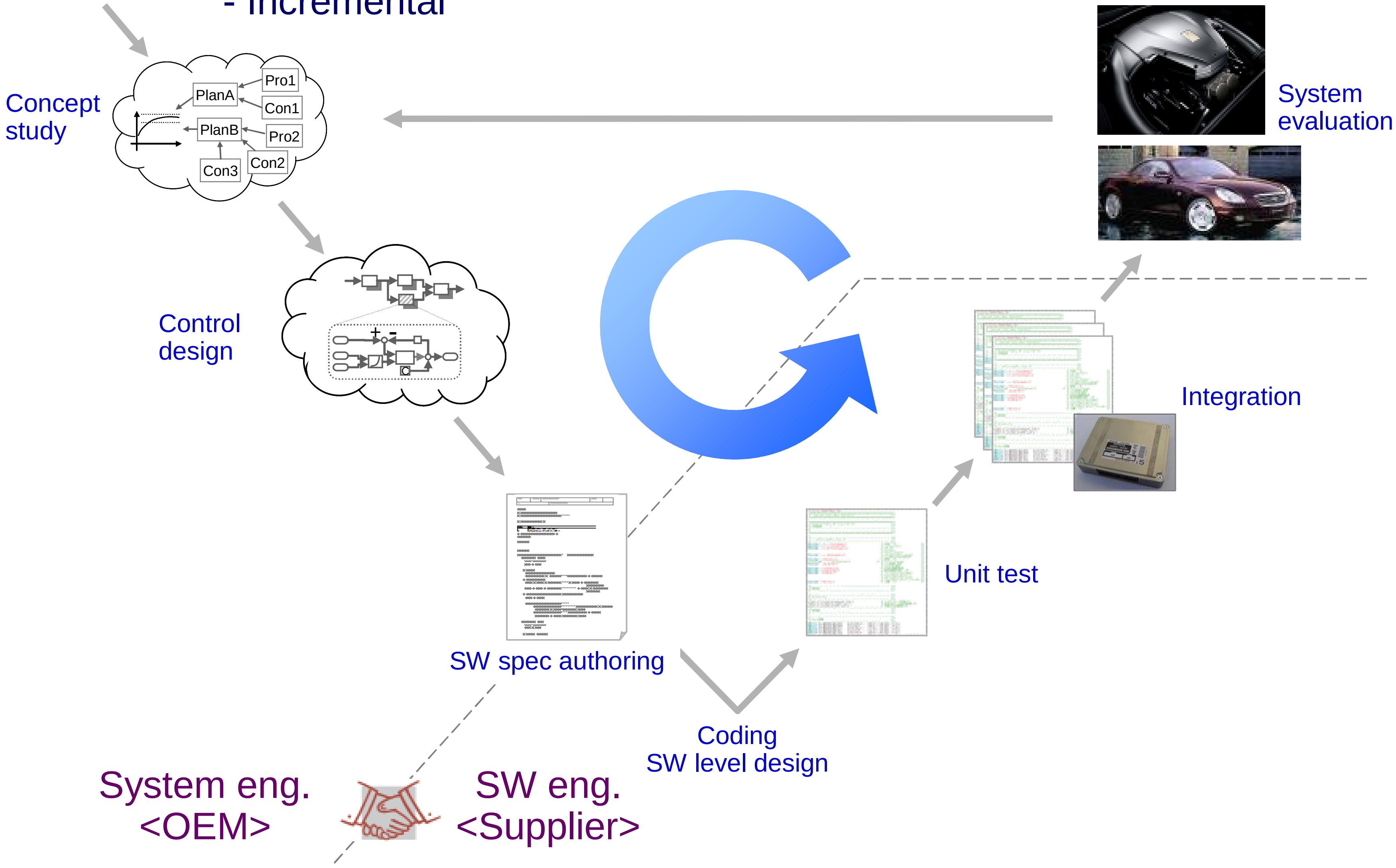
Hybrid system control

- Interacting with physical environment and various drivers
- Growing contribution of elaborate control systems to deliver attractive products and to meet regulations

→ Coping with growing complexity is a big challenge

Traditional Development Process

- Repetitive prototyping loops
- Requires different domain skills : system and software
- Incremental



Target

Concept study

Control design

SW spec authoring

Coding
SW level design

Unit test

Integration

System evaluation

System eng.
<OEM>

SW eng.
<Supplier>

Example : Developing Cold Start Control of Gasoline Engine

Target

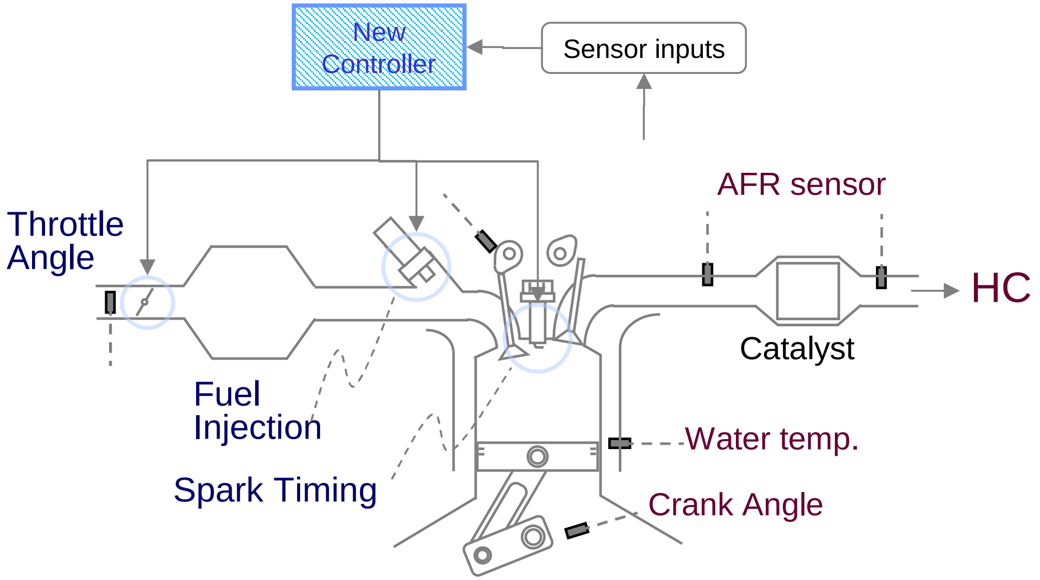
Reduce 5% Hydro Carbon emission

Concept

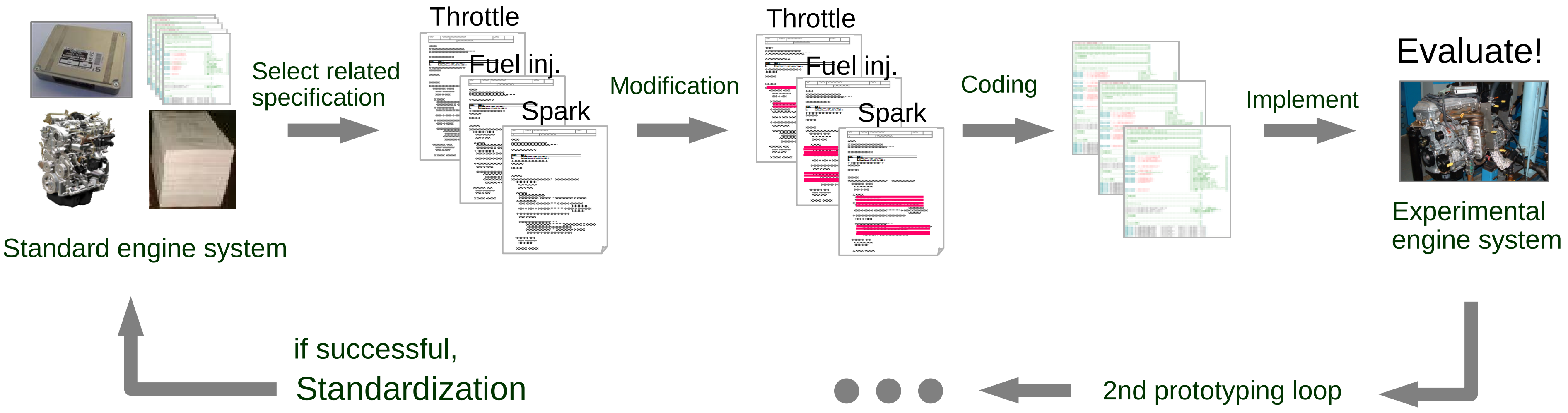
Activate catalyst converter as fast as possible

Control design

Optimize reference profiles of throttle, fuel injection and spark timing.



Supposed prototyping loops



Standard engine system

Select related specification

Modification

Coding

Implement

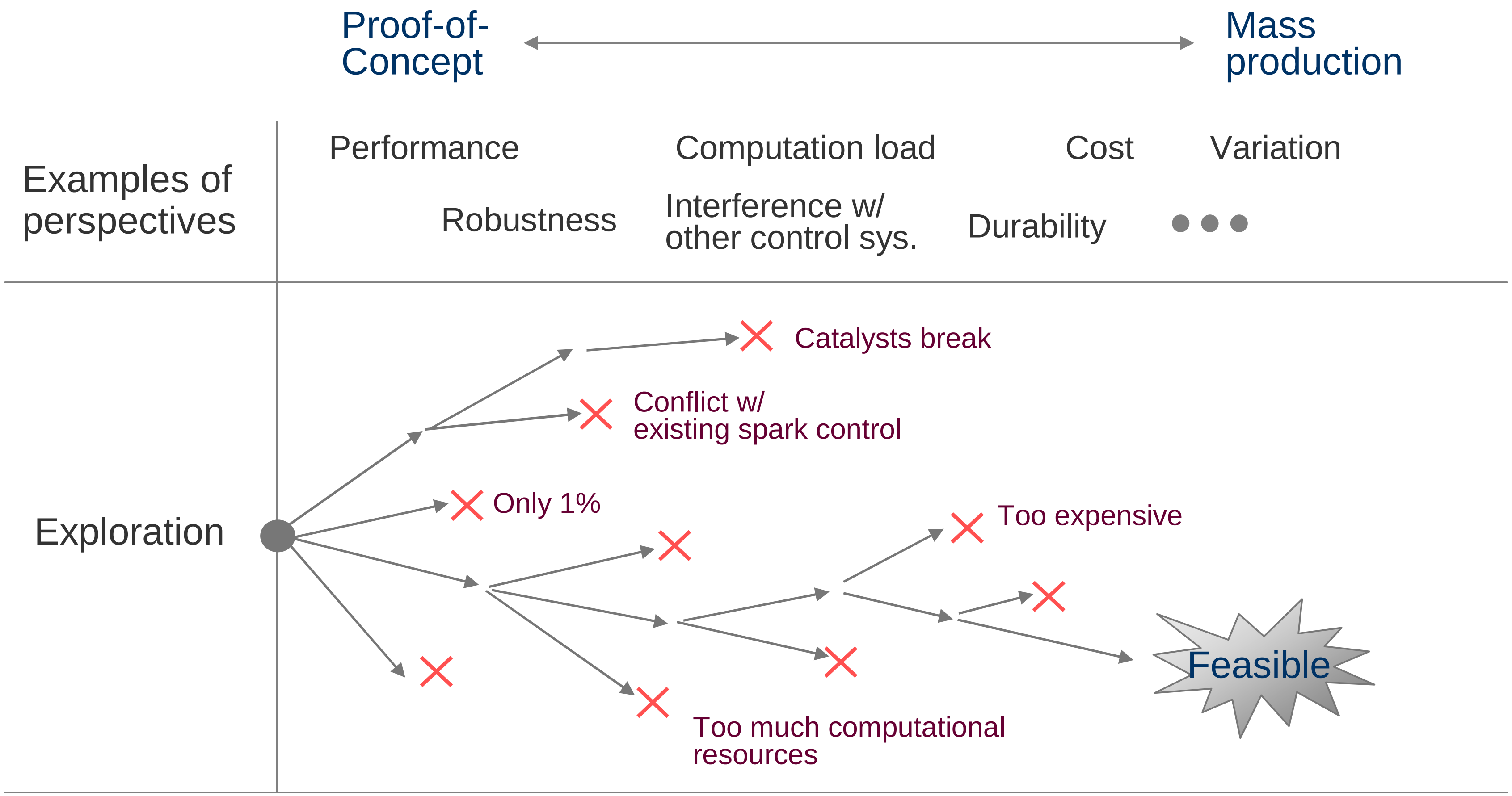
Evaluate!

Experimental engine system

if successful,
Standardization

2nd prototyping loop

System engineering is exploration of feasible solution



Proof-of-Concept ↔ Mass production

Examples of perspectives

Performance

Robustness

Computation load

Interference w/ other control sys.

Cost

Durability

Variation

● ● ●

Exploration

Catalysts break

Conflict w/ existing spark control

Only 1%

Too expensive

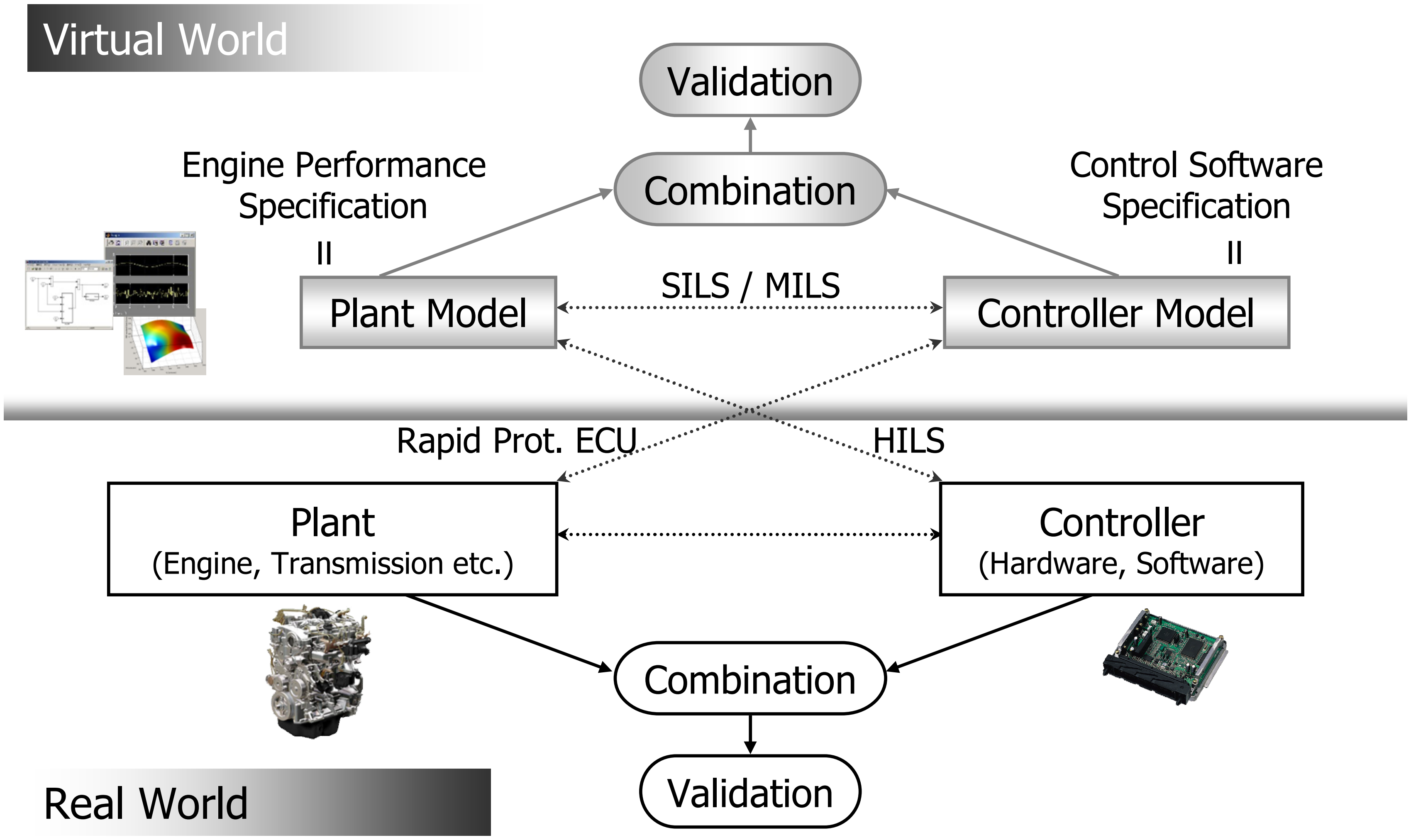
Too much computational resources

Feasible

- Due to growing complexity, hard to foresee exploration paths without prototyping
- Solution space is getting narrower

→ Improvement of prototyping loops is the key

Model-Based Development



Virtual World

Validation

Combination

Engine Performance
Specification
II
Plant Model

Control Software
Specification
II
Controller Model

SILS / MILS

Rapid Prot. ECU

HILS

Plant
(Engine, Transmission etc.)

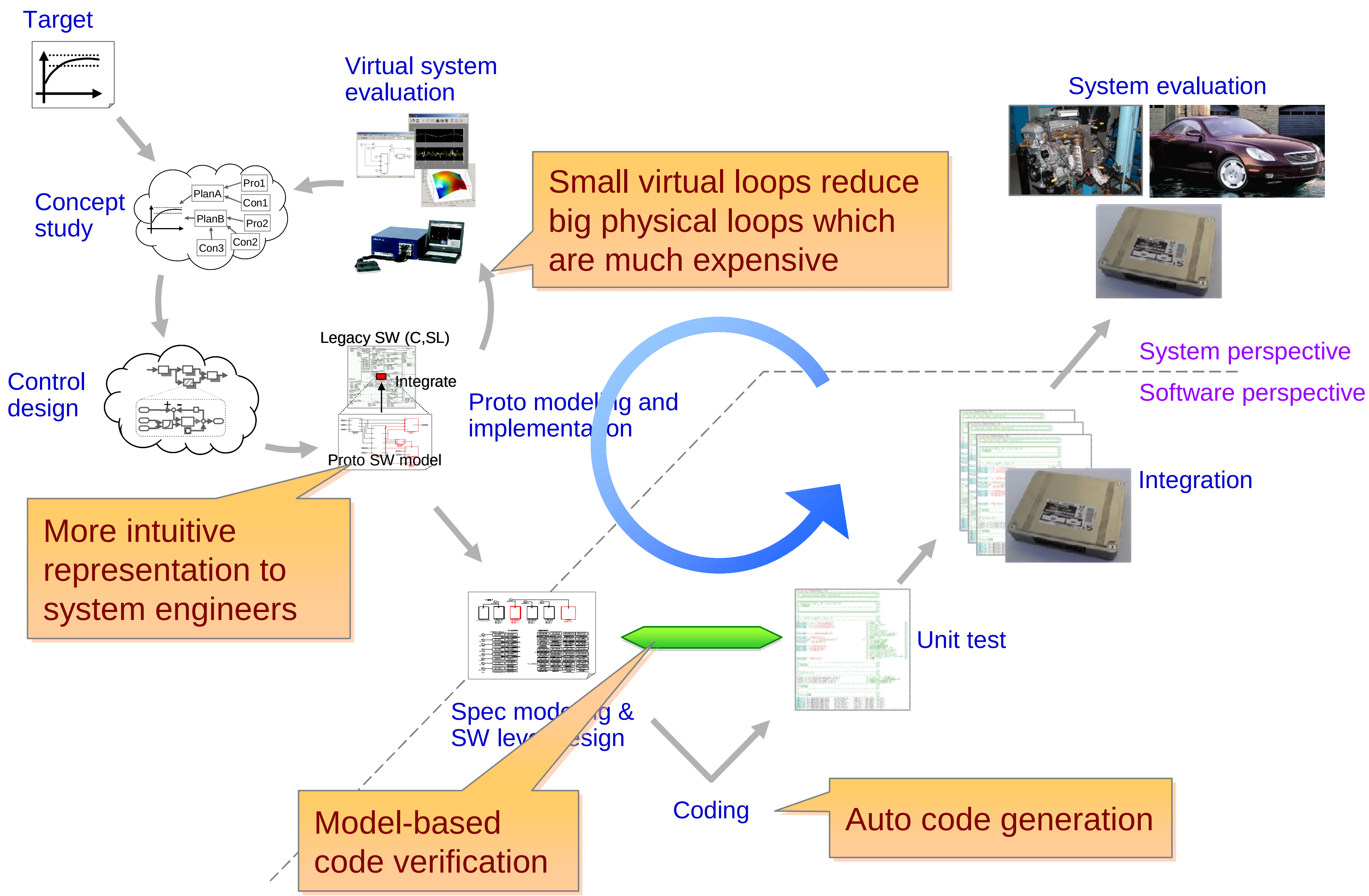
Controller
(Hardware, Software)

Combination

Validation

Real World

Model-based Development of Control Software



Target

Concept study

Virtual system evaluation

Control design

Small virtual loops reduce big physical loops which are much expensive

System evaluation

System perspective

Software perspective

Proto modeling and implementation

Integration

More intuitive representation to system engineers

Unit test

Spec modeling & SW level design

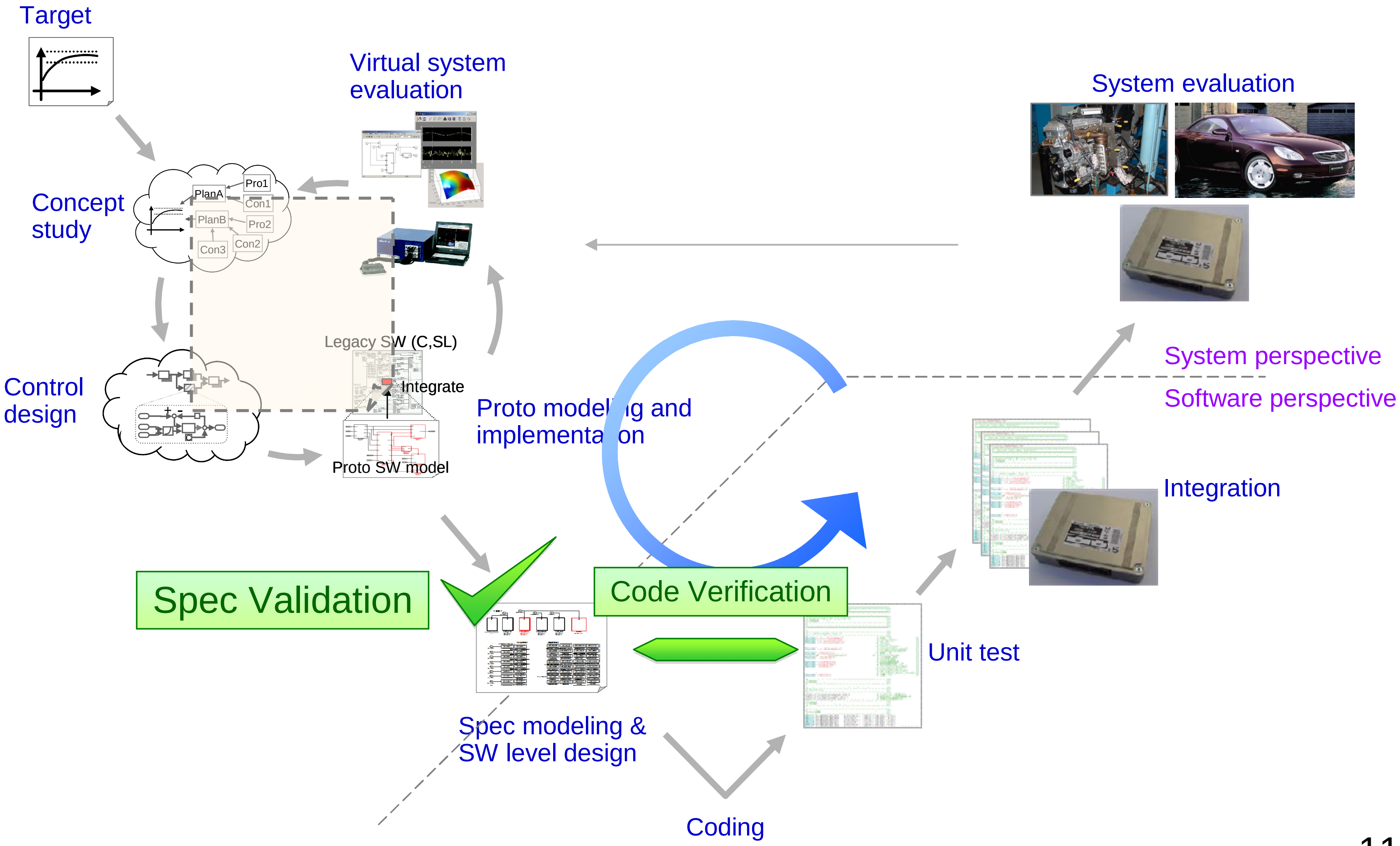
Coding

Model-based code verification

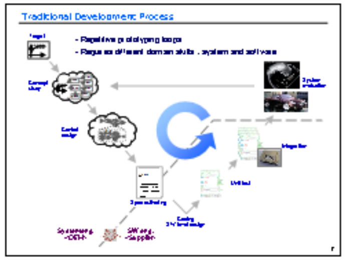
Auto code generation

Areas where further improvement is needed

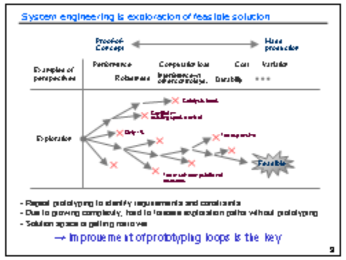
- Validation of code specification
- Lack of documents/evidence



Summary - Automotive control software development


Traditional Development Process

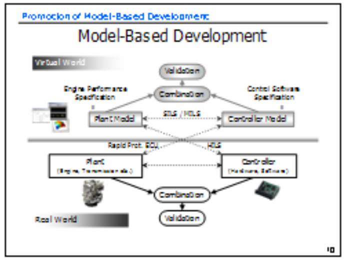
- Repetitive prototyping loops
- Requires different domain skills : system and software


System engineering is exploration of feasible solution

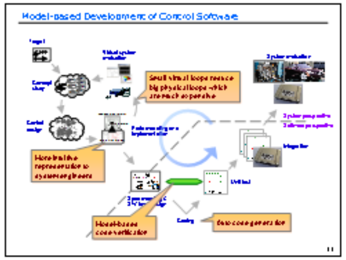
- Rapid prototyping to identify requirements and constraints
- Due to growing complexity, hard to foresee exploration paths without prototyping
- Solution space is getting narrower

→ Improvement of prototyping loops is the key

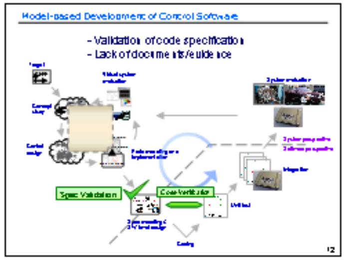
- Requires repetitive prototyping loops for feasible solution exploration


Promotion of Model-Based Development

Model-Based Development


Model-based Development of Control Software

- Promotion of Model-based Development to improve prototyping loop efficiency


Model-based Development of Control Software

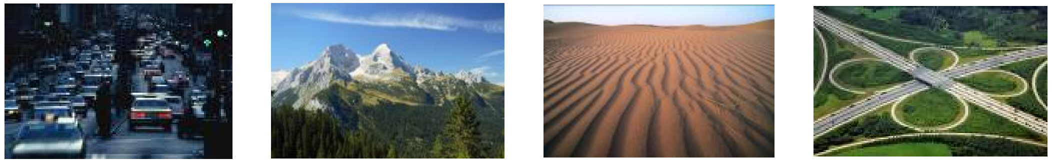
- Validation of code specification
- Lack of documents/evidence

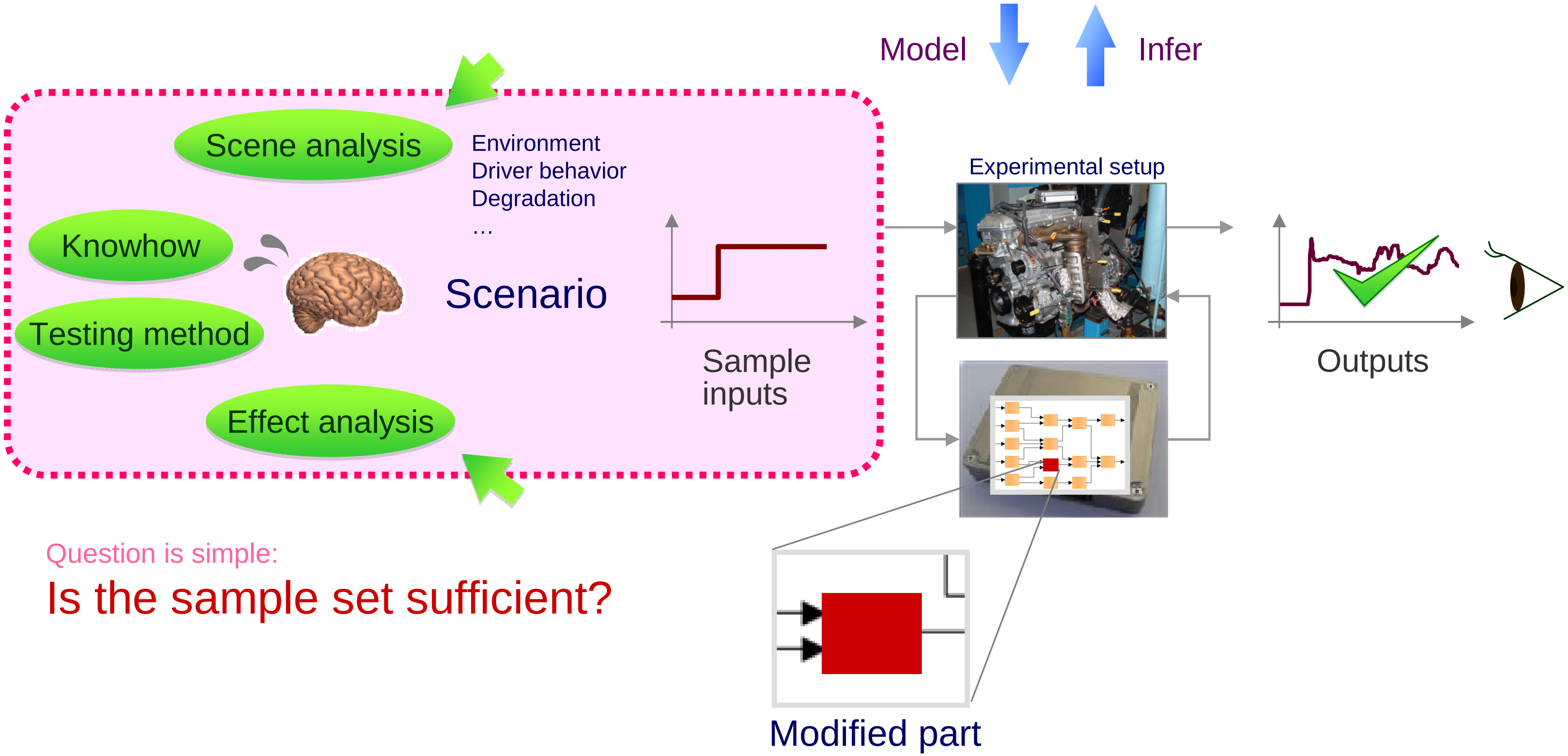
- Specification validation is a big challenge

1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction

Validation of code specification

Though true validity can only be confirmed in actual uses…





Model

Infer

Scene analysis

Environment
Driver behavior
Degradation
…

Knowhow

Testing method

Effect analysis

Scenario

Sample inputs

Experimental setup

Outputs

Modified part

Question is simple:

Is the sample set sufficient?

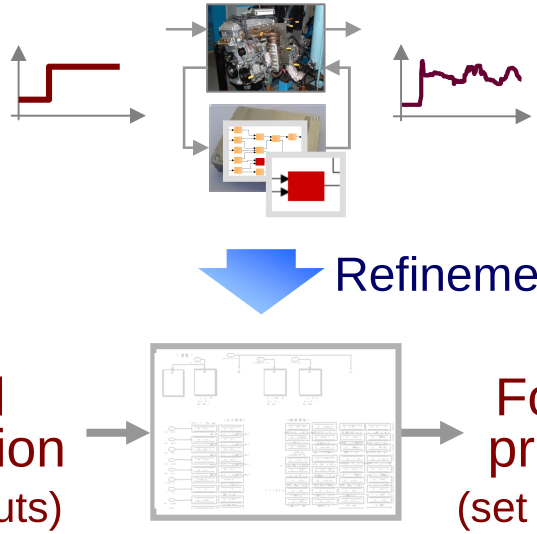
Making development process formal and accountable

☐ Systematic breakdown from system requirement to software spec

☐ Make design artifacts traceable to higher requirements as evidence

☐ Use formal models to apply advanced verification technology : property checking

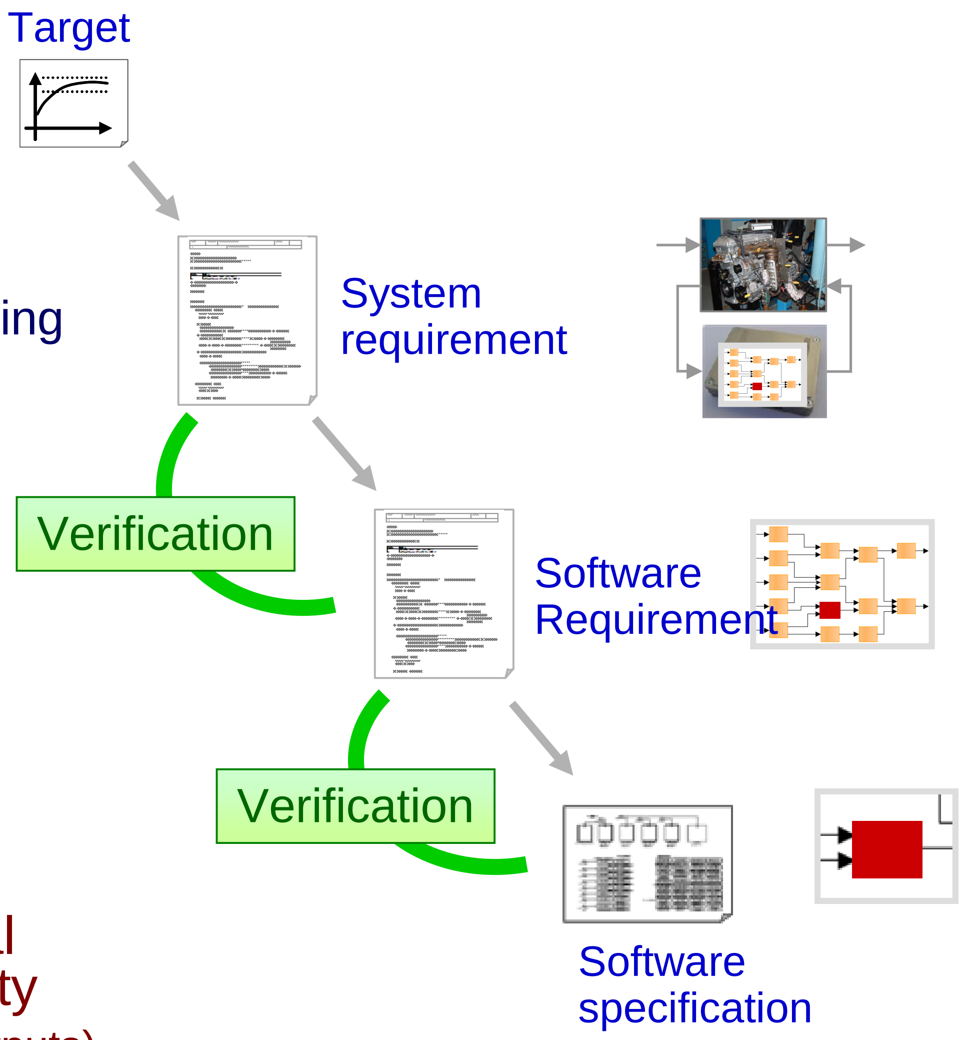
  - Model checking
  - Exhaustive testing



Refinement

Formal assumption (set of inputs)

Formal property (set of outputs)



Target

System requirement

Verification

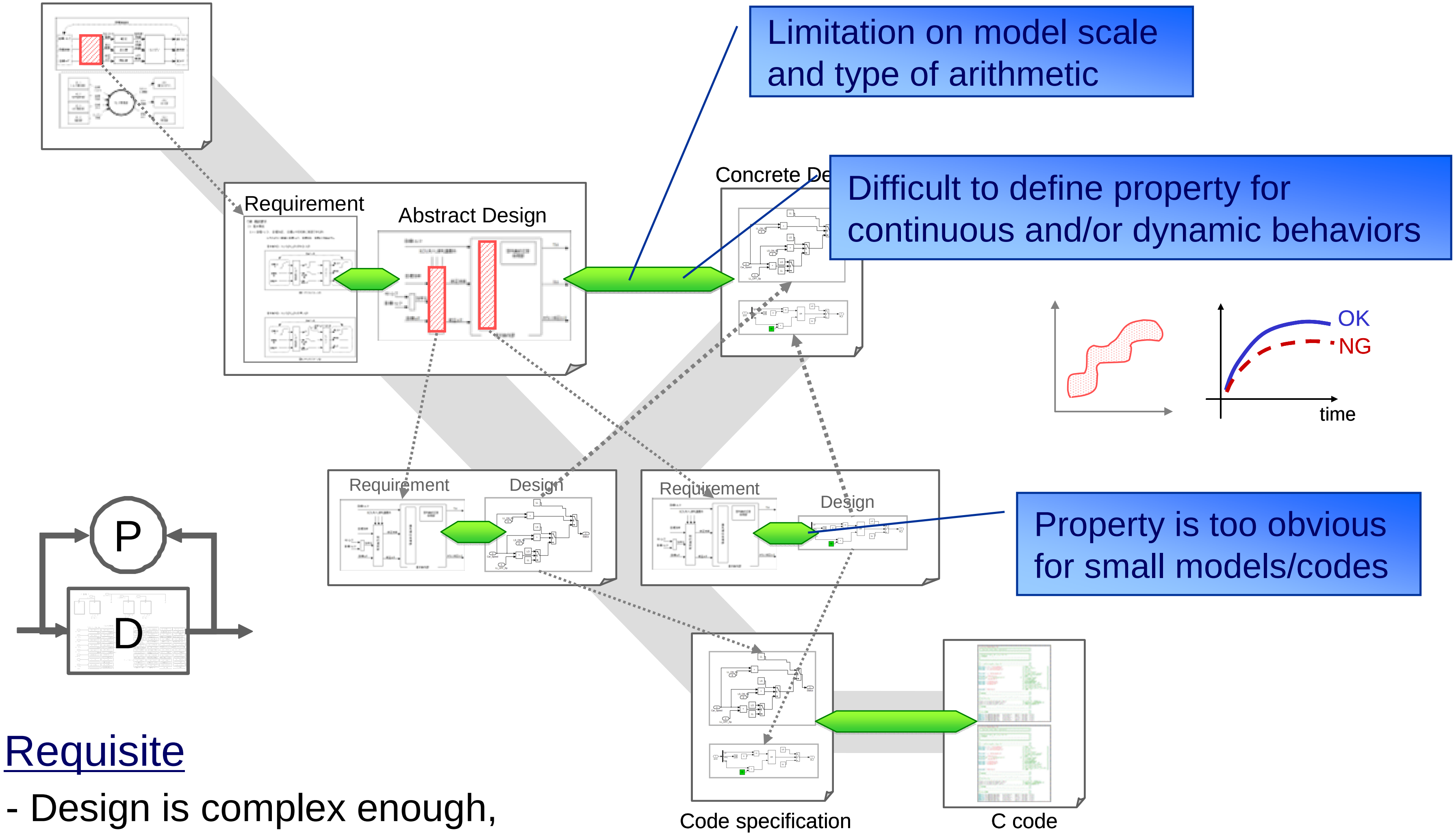
Software Requirement

Verification

Software specification

No need to care about samples! – exhaustiveness assured

Formal process and verification - lessons learned



Limitation on model scale and type of arithmetic

Difficult to define property for continuous and/or dynamic behaviors

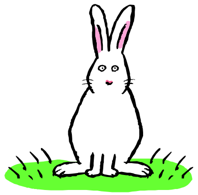
Property is too obvious for small models/codes

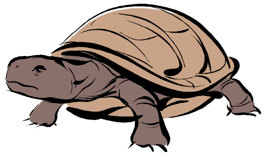
Requisite

- Design is complex enough, but not too much
- Property must be simply describable relative to the Design

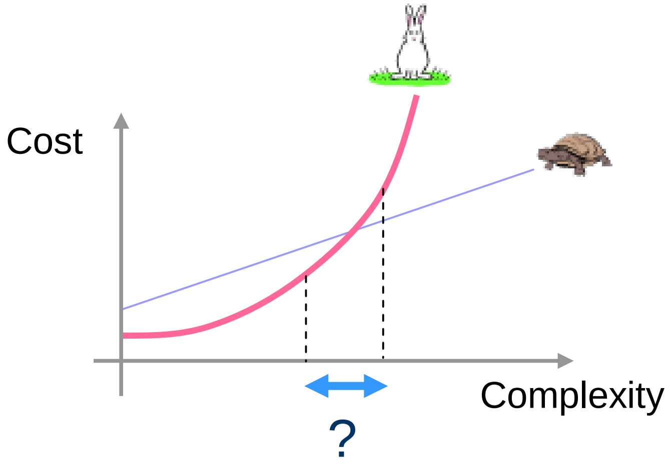
So far applicable area is limited.

Rabbit vs. Turtle





Agile	Waterfall
Informal	Formal
Small system	Large system
Not scalable	Scalable
Integral	Modular
Unaccountable	Accountable
Individual/Skill oriented	Team/Rule oriented
Efficient	Redundant



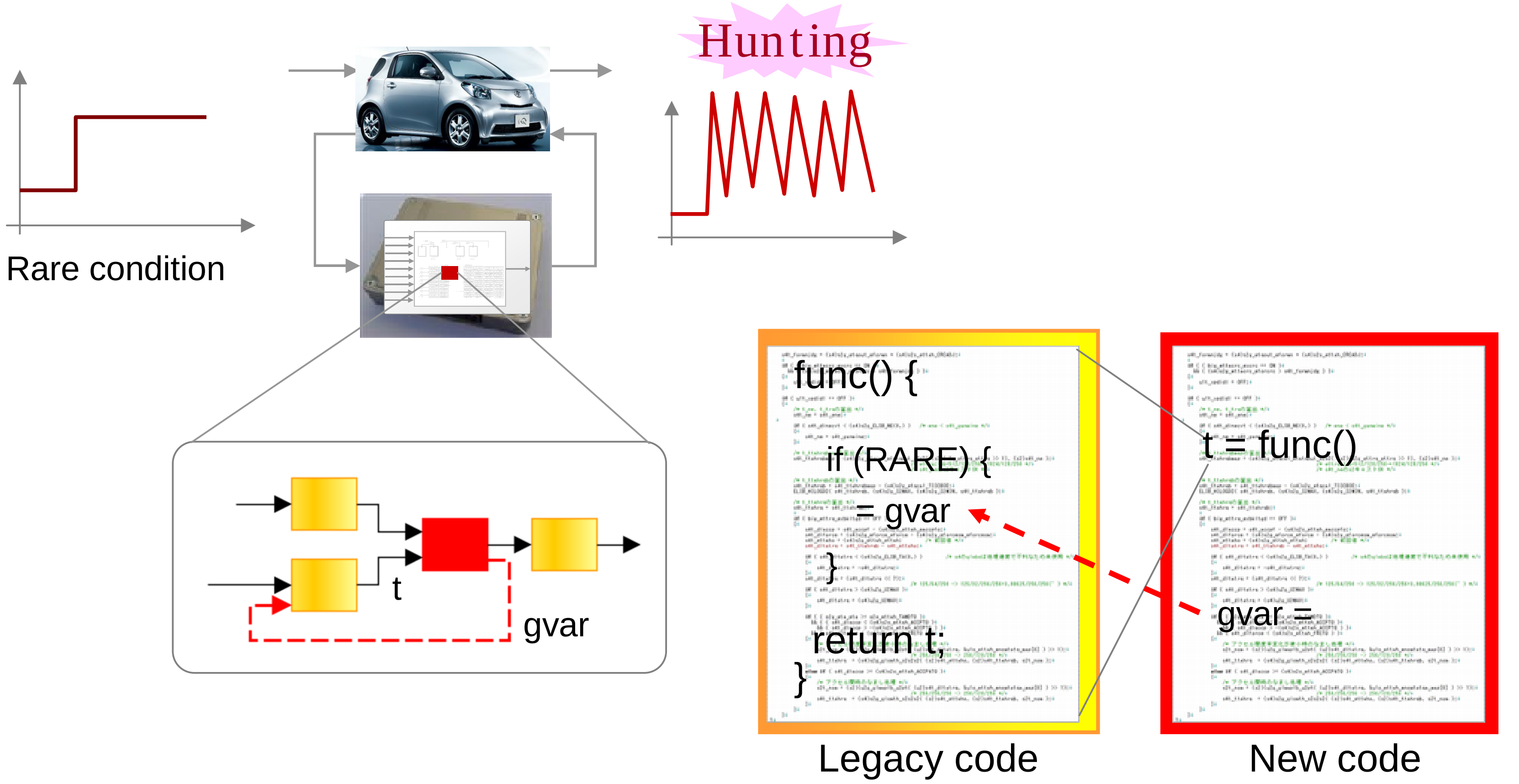
In Japan:

- Low communication cost
- Cultural strength
- …

On the way to finding the best level of formality

A practical problem - latent function

Under a rare condition, hunting behavior had occurred.
The cause was an unrecognized dependence loop.



```
func() {
  if (RARE) {
    = gvar
  }
  return t;
}
```

```
t = func()

gvar =
```

Due to complex dependence among function, especially via shared memory, one function was overlooked and the condition hadn’t been exercised due to its rareness.

→ C code is to blame for … ?

Another practical problem - latent function in a Simulink model

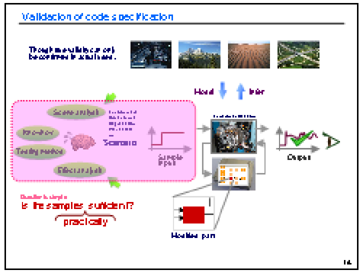
It seems there is no latency in models, but it is not as obvious as we suppose.

Redundant paths in a Simulink model authored by system engineer

Supposed reasons of incomprehensibility:

- Lack of software design skill or less care about model quality
- Essential paths are obscured by software level design details (e.g. type guard)
- Functional grouping is not an easy task

Summary - Specification validation and open issue

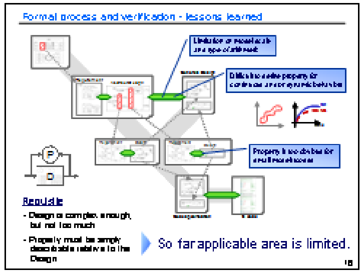


Validation of code specification

Is the samples sufficient?

practically

- “Is the sample set enough?”

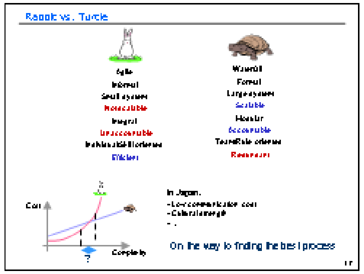


Formal process and verification - lessons learned

Requisite

- Design is complex enough, but not too much
- Property must be simply describable relative to the Design

So far applicable area is limited.

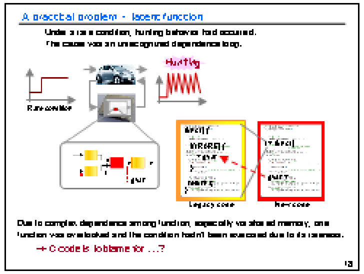


Rabbit vs. Turtle

In Japan,
- Low communication cost
- Cultural strength
- ...

On the way to finding the best process

- Current usage of formal verification is limited
- On the way to finding the best level of formality

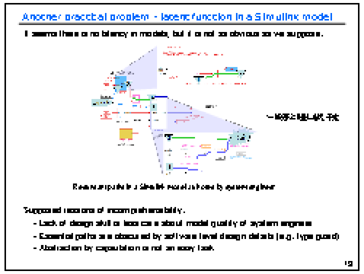


A practical problem - latent function

Under a rare condition, hunting behavior had occurred.
The cause was an unrecognized dependence loop.

Due to complex dependence among function, especially via shared memory, one function was overlooked and the condition hadn't been executed due to its rareness.

→ C code is to blame for …?



Another practical problem - latent function in a Simulink model

It seems there is no latency in models, but it is not so obvious as we suppose.

Suggested reasons of incomprehensibility.
- Abstraction by encapsulation is not an easy task.

- One of validation problem: Latent function

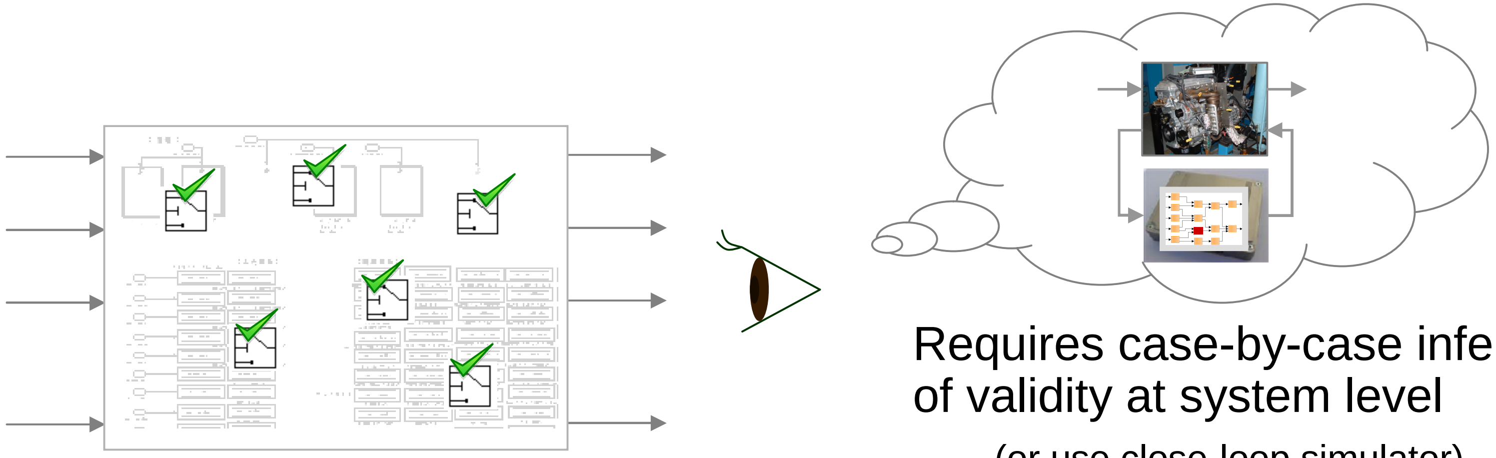
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction

A direction to go - whitebox

Testing with software coverage metrics

Make sure branches and conditions of each switch are exercised

Any

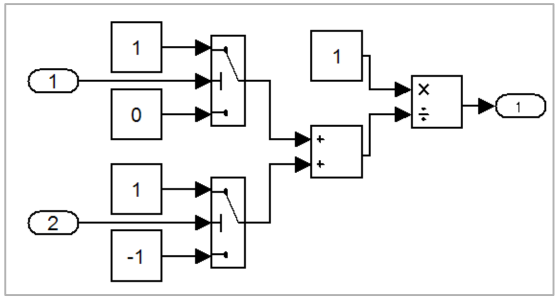


Requires case-by-case inference of validity at system level

(or use close-loop simulator)

Problems:

- Are functional components really covered?
- Bad S/N for functional coverage
  (importance of branches are not even)
- Hard to infer validity without knowing which function was stimulated

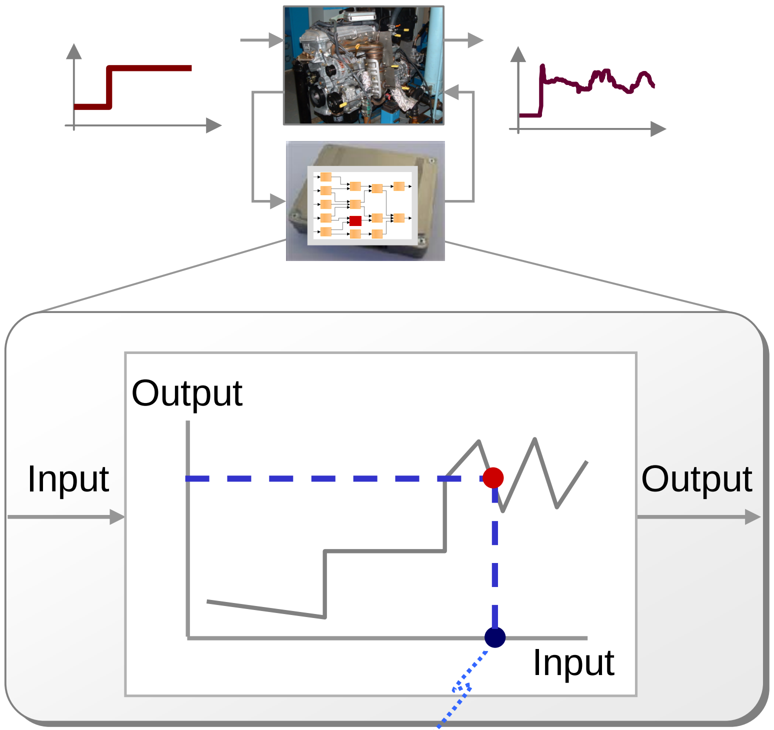


There is a chance of div by 0 error with 100% branch coverage

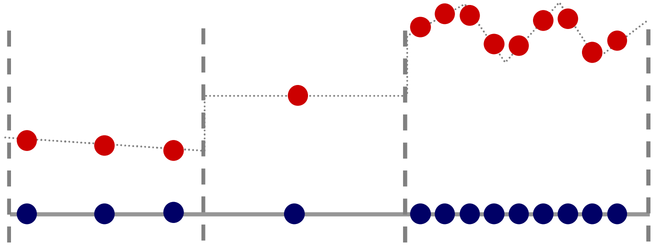
Hard to get a sense of functional coverage

Functional coverage

Example:
1 input, 1 output

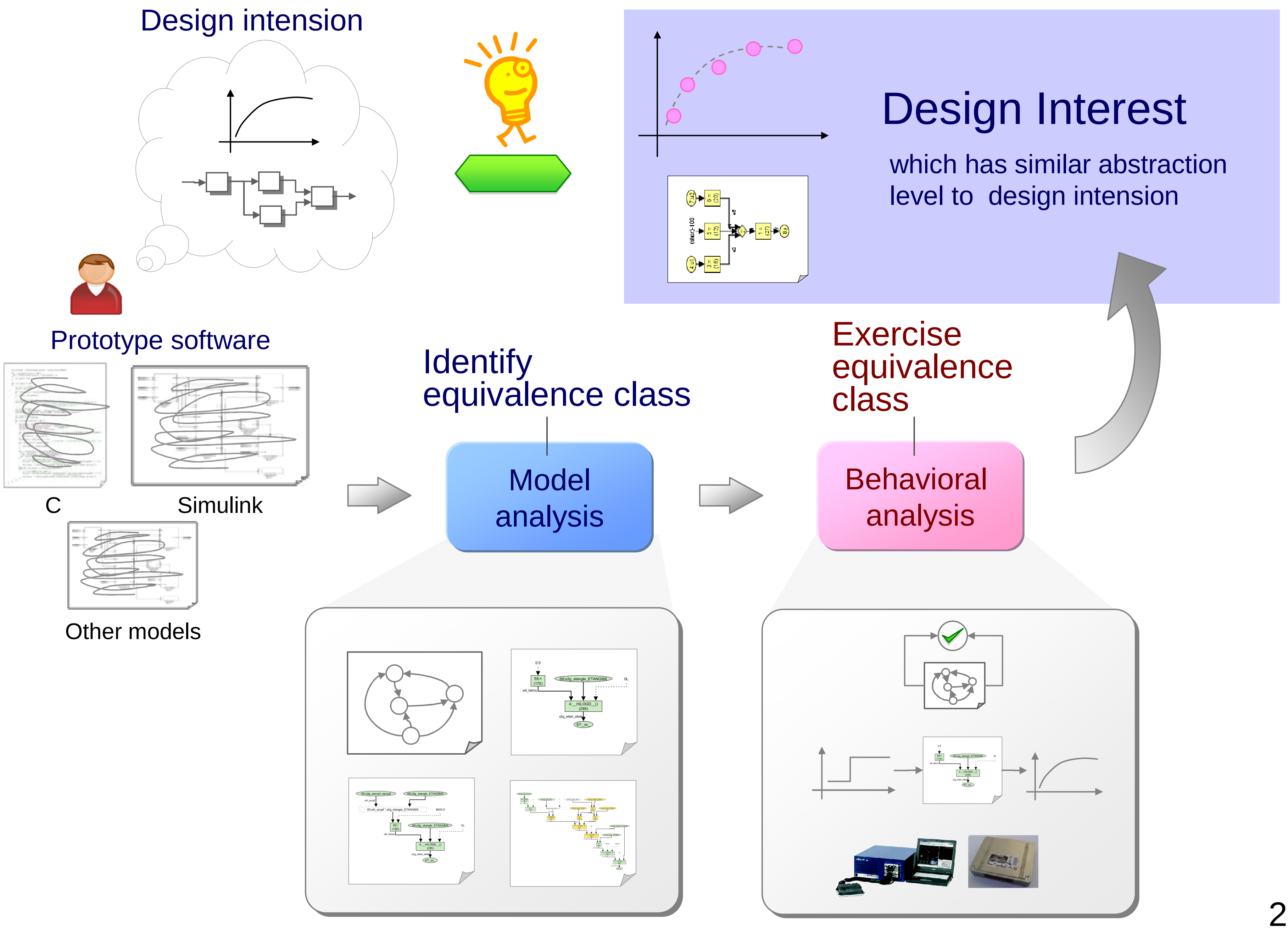


A point exercised by the sample
(trajectory if dynamic)



Coverage is supposed to be sufficient if each of **equivalence class** of function is exercised in proper manner respectively.

Specification validation by design interest extraction



Design intension

Design Interest

which has similar abstraction level to design intension

Prototype software

C

Simulink

Other models

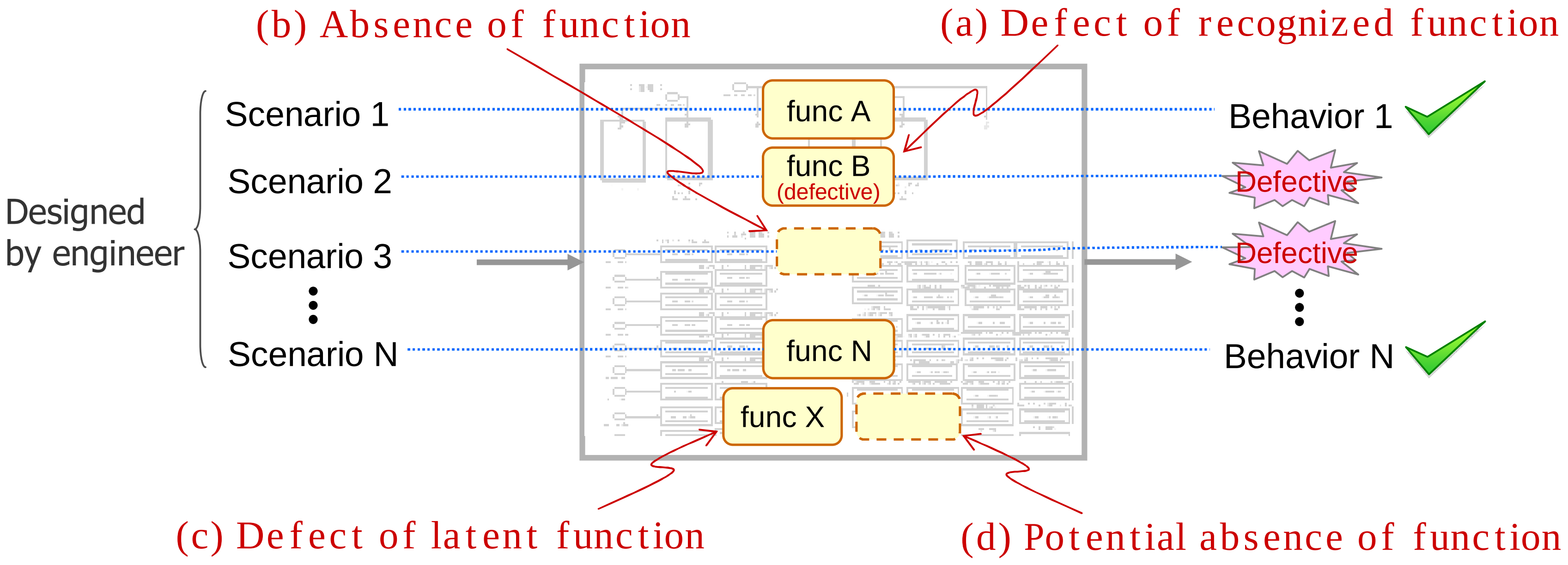
Identify equivalence class

Model analysis

Exercise equivalence class

Behavioral analysis

Defect category and our target



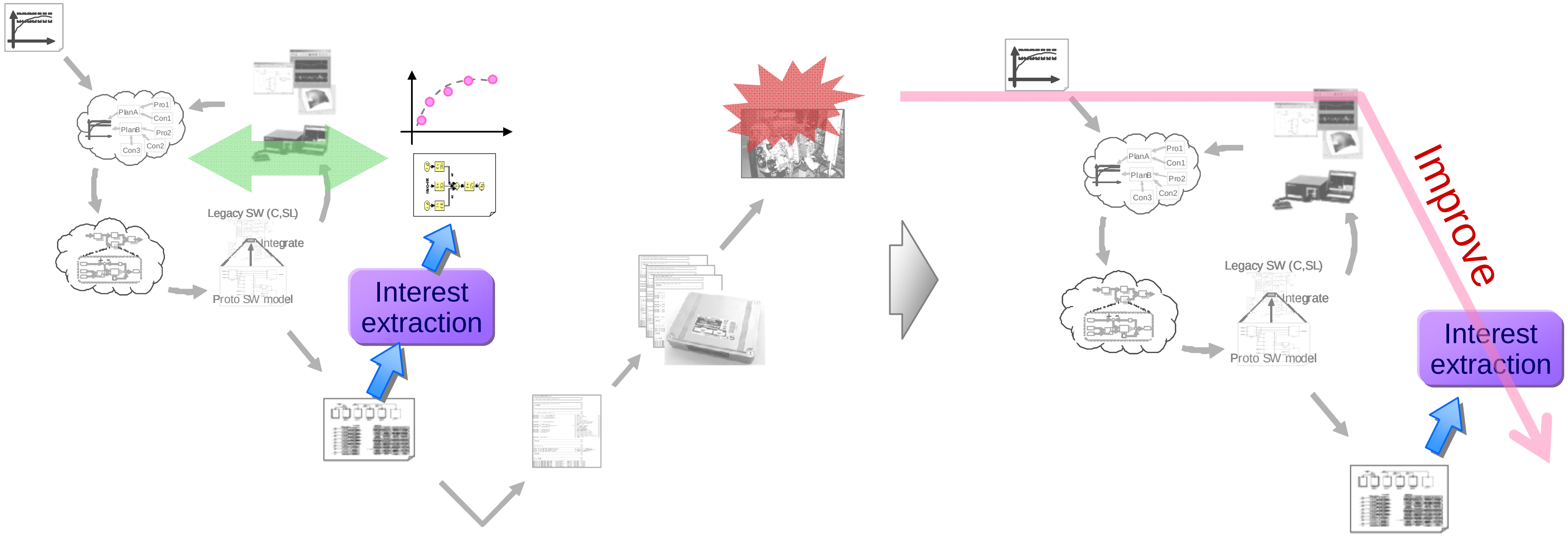
		Class of function	
		Exist	Not exist
Scenario	Specified	(a) Recognized	(b) Absent
	Unspecified	(c) Latent	(d) Potentially absent

Matter of requirement engineering

Our target : Are functional components covered?

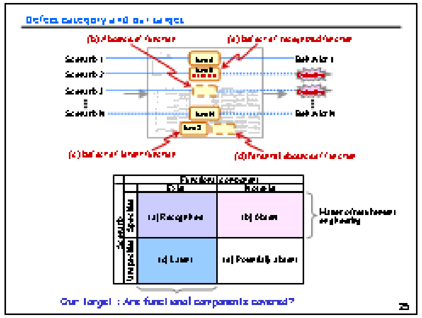
Expected benefits of the approach

- Interactive process with visual support stimulates engineer’s awareness
- Mechanized interest extraction serves as the baseline of coverage standard
- Quality of validation can be improved by tuning extraction mechanism



Next development cycle

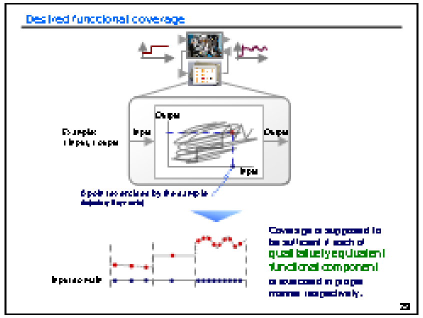
Summary - Concept of the practical approach



Defect category and our target

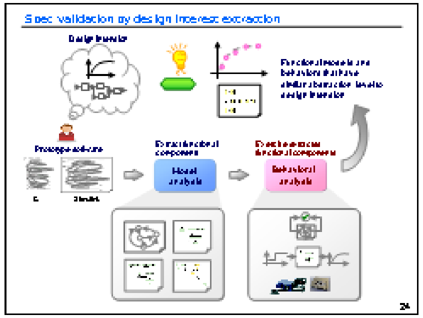
Our target : Are functional components covered?

- Our target is latency problem



Desired functional coverage

- Covering equivalence class of function



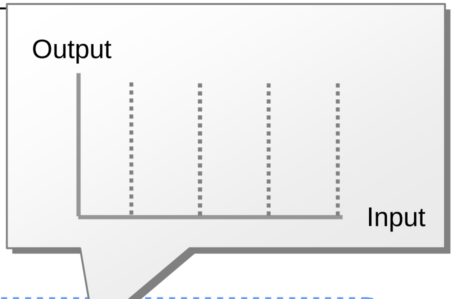
Spec validation by design interest extraction

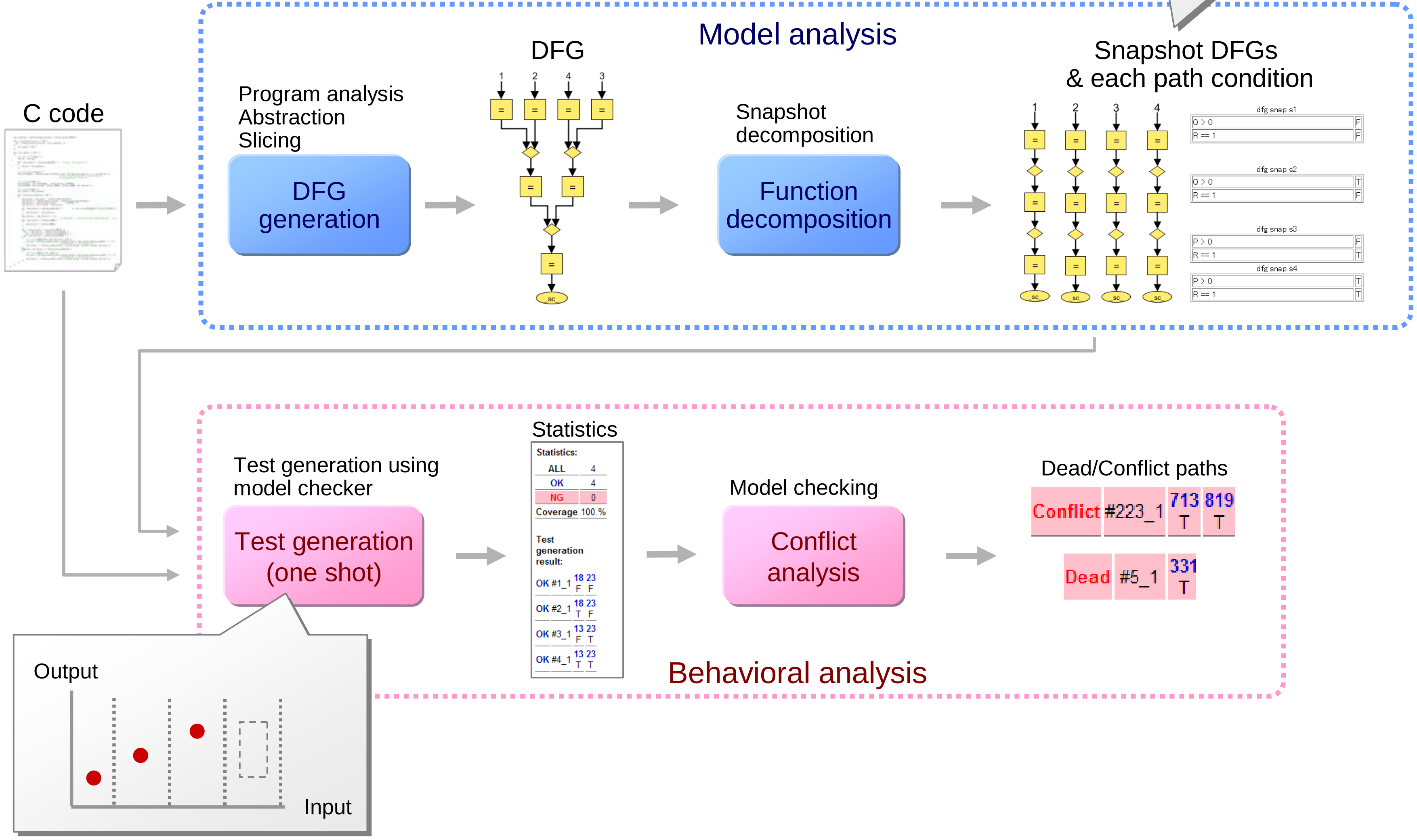
- Extract design interest : equivalence class of function and their behaviors
- Visual support enhances engineers' awareness

1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction

Outline of current tool implementation

- A kind of dataflow graph (DFG) as one of the abstract functional model of C code
- Snapshot DFG as one of the equivalence class
- Oneshot testing which stimulate the path corresponding to each DFG





Model analysis

C code

Program analysis
Abstraction
Slicing

DFG generation

DFG

Snapshot decomposition

Function decomposition

Snapshot DFGs
& each path condition

dfg snap s1

Q > 0	F
R == 1	F

dfg snap s2

Q > 0	T
R == 1	F

dfg snap s3

P > 0	F
R == 1	T

dfg snap s4

P > 0	T
R == 1	T

Behavioral analysis

Test generation using
model checker

Test generation (one shot)

Statistics

Statistics:

ALL	4
OK	4
NG	0
Coverage	100.%

Test generation result:

OK	#1_1	18 F	23 F
OK	#2_1	18 T	23 F
OK	#3_1	13 F	23 T
OK	#4_1	13 T	23 T

Model checking

Conflict analysis

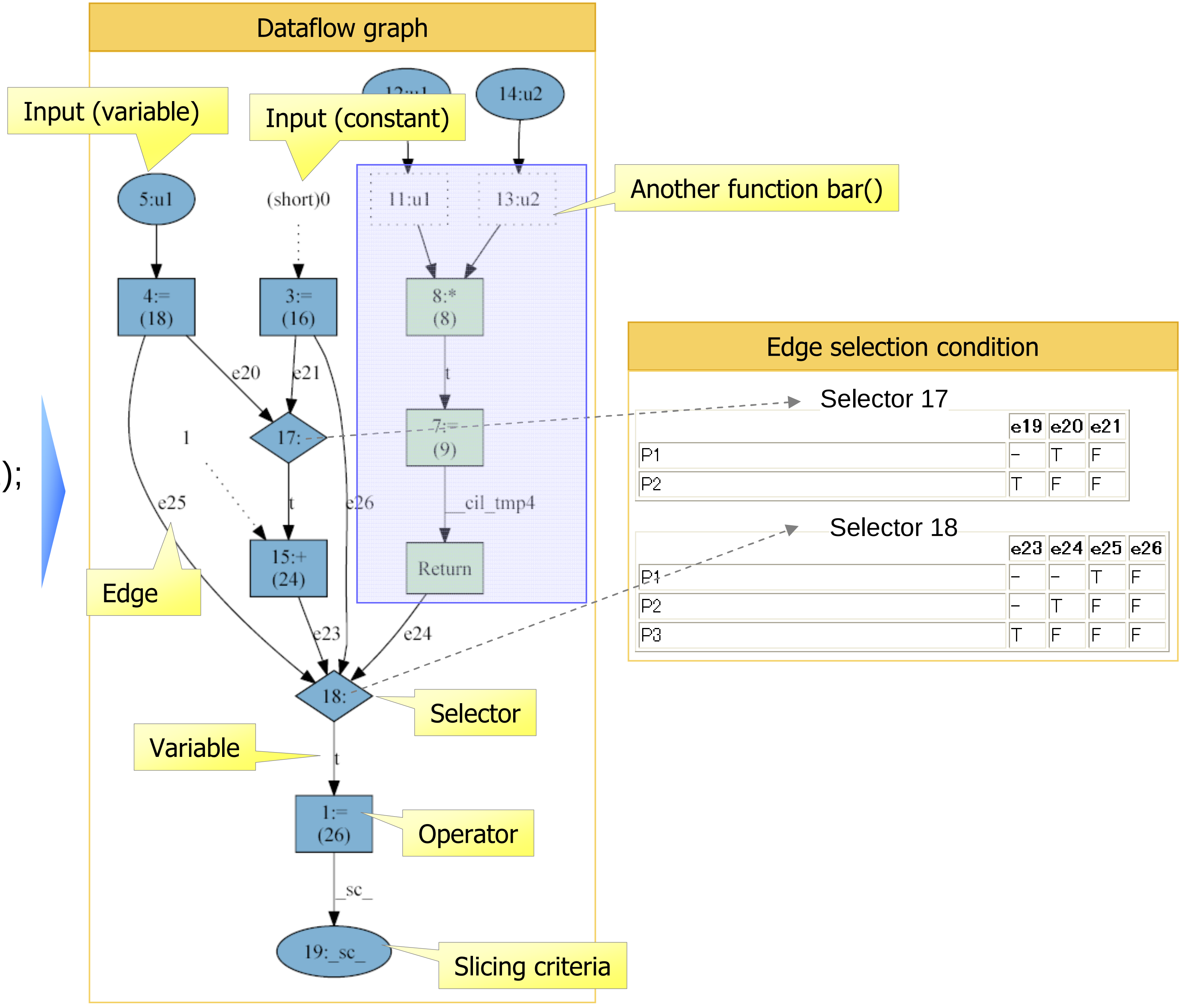
Dead/Conflict paths

Conflict	#223_1	713 T	819 T
Dead	#5_1	331 T	

DFG with edge selection condition

```
Line
15   foo() {
16     t = 0;
17     if (P1) {
18       t = u1;
19     }
20     if (P2) {
21       t = bar(u1, u2);
22     }
23     if (P3) {
24       t = t + 1;
25     }
26     _sc_ = t;
     }
```

Slicing criteria



Selector 17

	e19	e20	e21
P1	–	T	F
P2	T	F	F

Selector 18

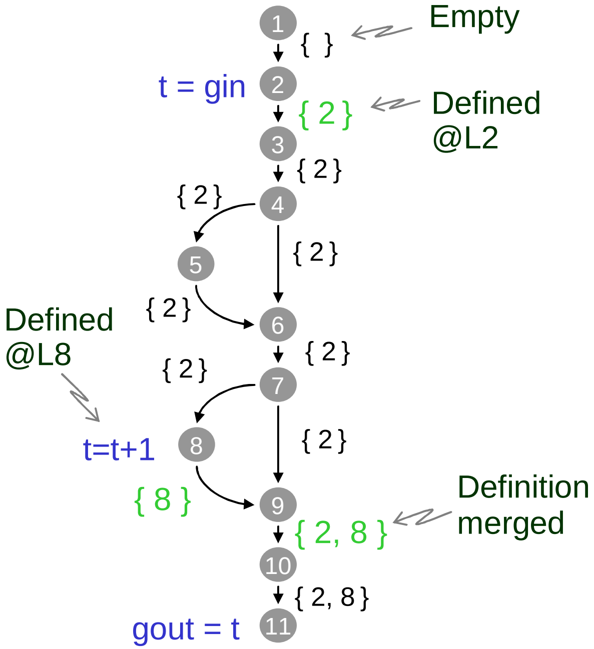
	e23	e24	e25	e26
P1	–	–	T	F
P2	–	T	F	F
P3	T	F	F	F

DFG generation algorithm

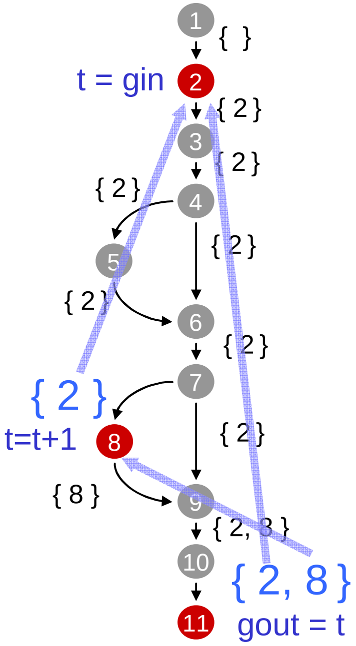
Code

```
1:   …
2:   t = gin
3:   ...
4:   If (P) {
5:     …
6:   }
7:   if (Q) {
8:     t = t+1
9:   }
10:  …
11:  gout = t
```

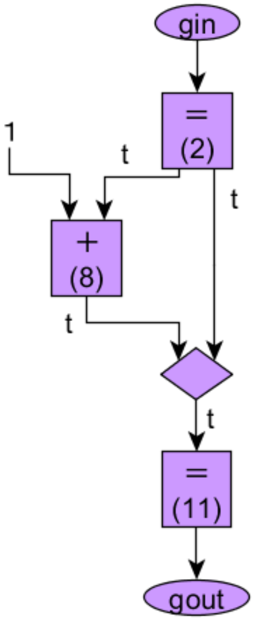
Reaching-definition



Backward tracing



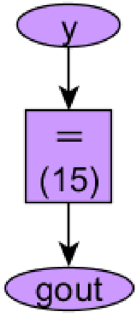
DFG



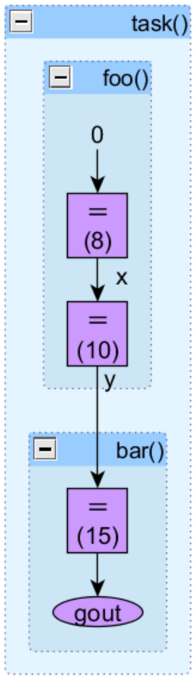
Points unique to embedded control software

```
1:   repetitive_task(){
2:    foo()
3:    bar()
4:   }
5:
6:   foo(){
7:     if (P) {
8:       x = 0
9:     }
10:    y = x
11:  }
12:
13:  bar(){
14:    x = gin
15:    gout = y
16:  }
```

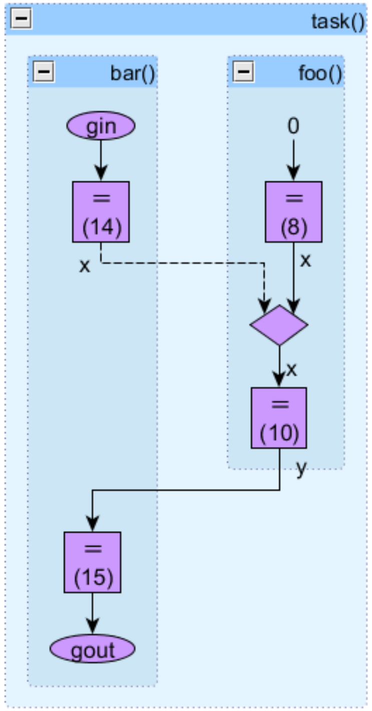
Slicing criteria



Inter-procedural dependence

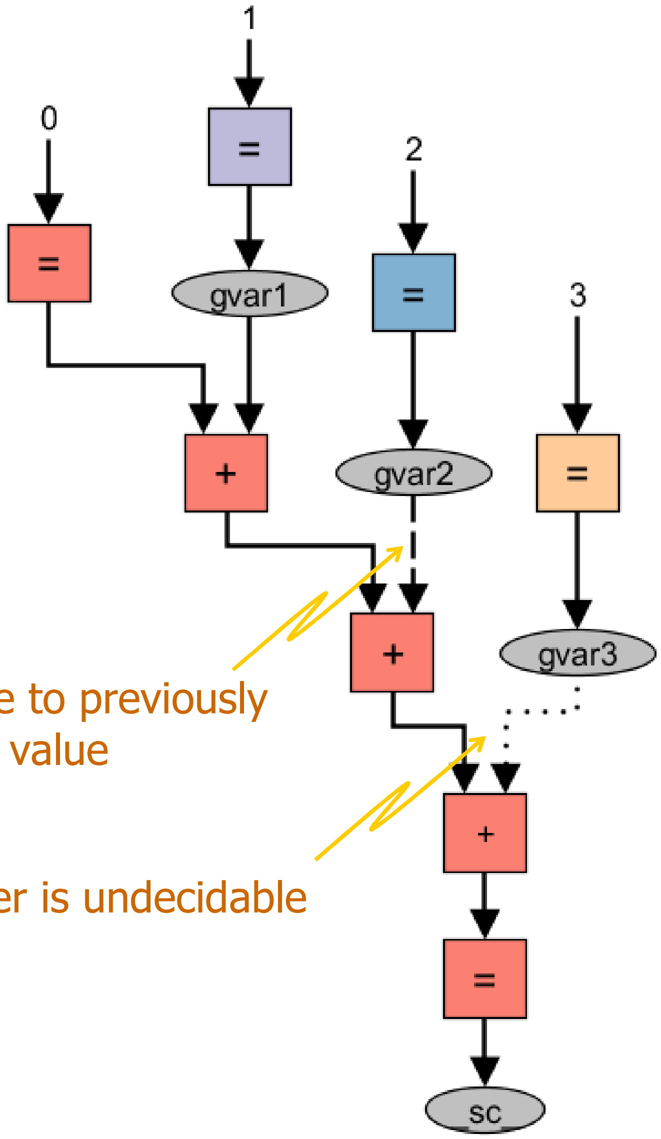


Dependence to previous definition



Example

```
int _sc_, gvar1, gvar2, gvar3;
int * const gvar_tbl[3]  = {&gvar1, &gvar2, &gvar3};

void timed_task(void){
  foo();
  bar();
  baz();
}

void foo(void){
  gvar1 = 1;
}

void bar(void)
{
  int i, P, sum;

  sum = 0;
  for ( i = 0 ; i < 3 ; i++ ) {
    sum += *(gvar_tbl[i]);
  }

  _sc_ = sum;
}

void baz(void){
  gvar2 = 2;
}

void occasional_event(void){
  gvar3 = 3;
}
```



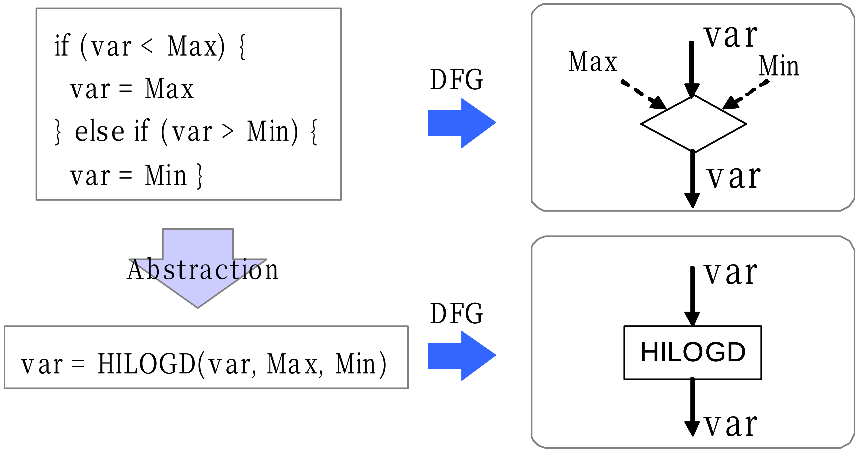
Reference to previously executed value

Order is undecidable

Model abstraction

Replace typical function patterns to compact representation:
- to omit trivial branches
- to help comprehension

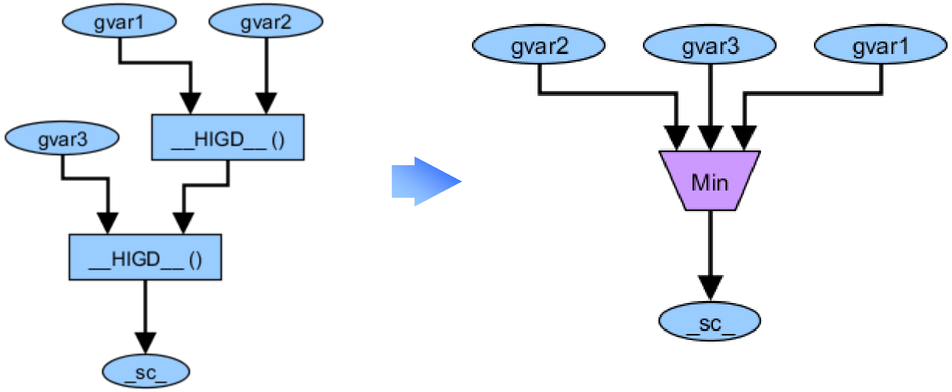
  - Type guard
  - Absolute
  - Rounding
  - Max/Min
  - Summation
  - Cast
  - …



```
if (var < Max) {
  var = Max
} else if (var > Min) {
  var = Min }
```

```
var = HILOGD(var, Max, Min)
```

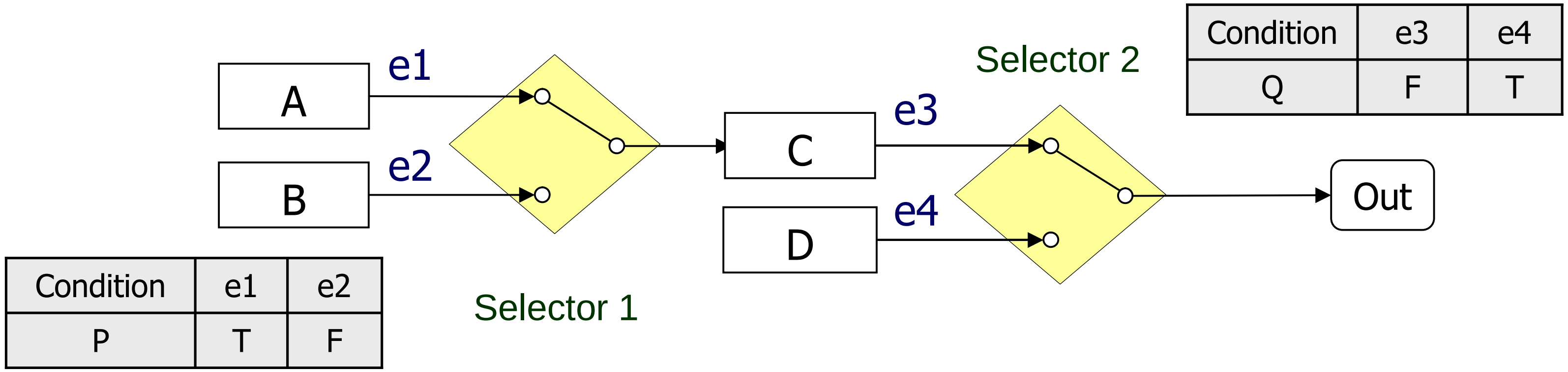
On Control flow graph



On Dataflow graph

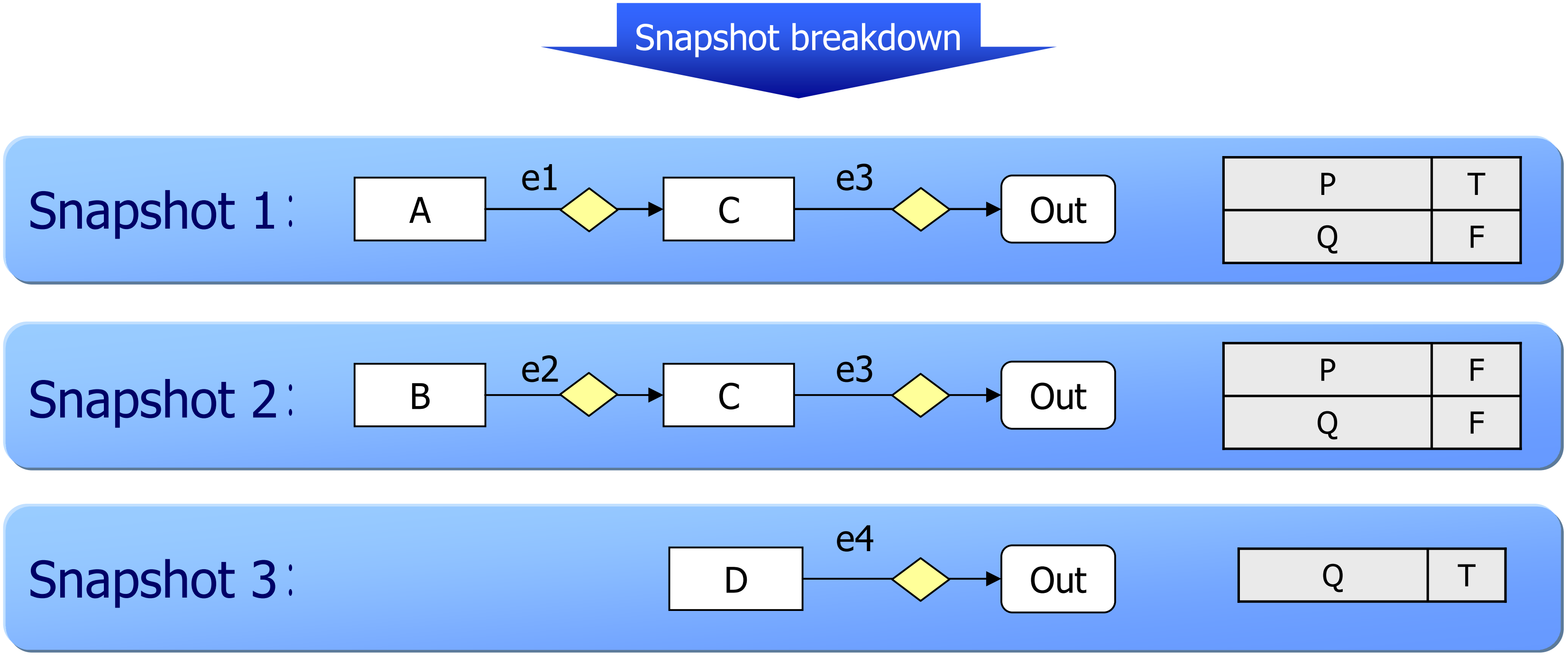
Snapshot DFG as a decomposed functional component

Enumerate possible dataflow patterns by taking edge combinations



Condition	e1	e2
P	T	F

Condition	e3	e4
Q	F	T


Snapshot breakdown

Snapshot 1 :

P	T
Q	F

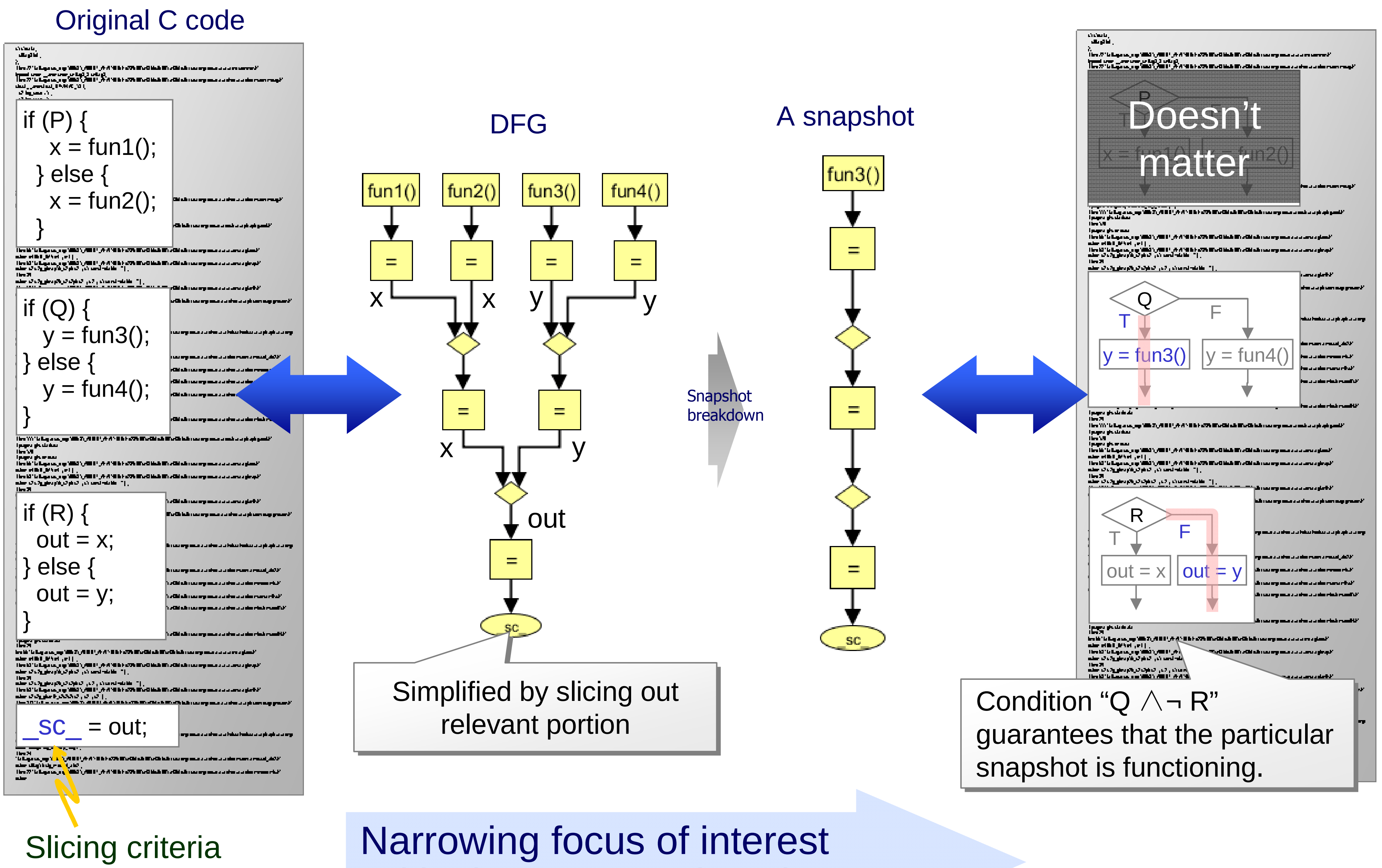
Snapshot 2 :

P	F
Q	F

Snapshot 3 :

Q	T

Meaning of the snapshot breakdown



Original C code

```
if (P) {
    x = fun1();
 } else {
    x = fun2();
 }
```

```
if (Q) {
   y = fun3();
} else {
   y = fun4();
}
```

```
if (R) {
  out = x;
} else {
  out = y;
}
```

```
_SC_ = out;
```

Slicing criteria

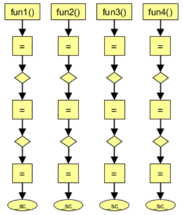
Simplified by slicing out relevant portion

Condition “Q ∧¬ R” guarantees that the particular snapshot is functioning.

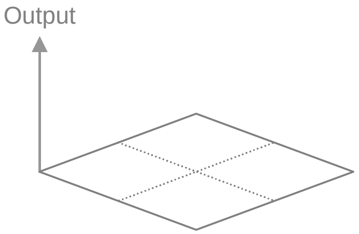
Narrowing focus of interest with visual comprehension

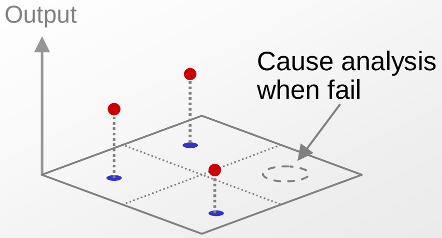
Behavioral analysis for extracted snapshot

Regarding the snapshots as functional components...

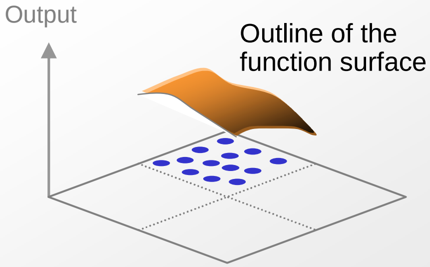


Snap1	P ∧ R
Snap2	¬P ∧ R
Snap3	Q ∧ ¬R
Snap4	¬ Q ∧ ¬R



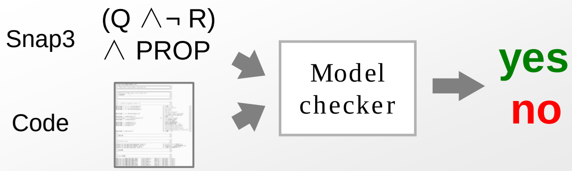


Oneshot test to cover snapshots (ATG1)

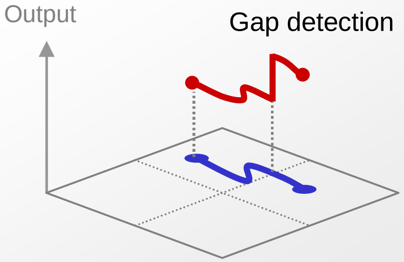


Grid test in a snapshot (ATG2)

Already implemented



Property checking in snapshots



Dynamic test w/ snap traverse (ATG3)

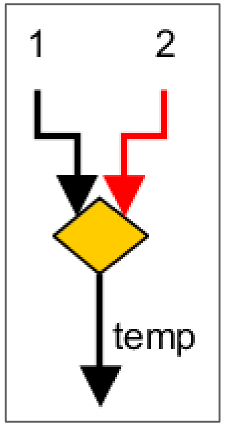
✓ Snap1	P ∧ R
✓ Snap2	¬P ∧ R
✓ Snap3	Q ∧ ¬R
✓ Snap4	¬ Q ∧ ¬R

Coverage monitoring for closed-loop test

⋮

ATGX

Instrument C code and find inputs which passes the target path.



```
int gvar1
int gvar2

void main (void){
  int temp;

  if ((gvar1 + gvar2) > 10) {
    temp = 1;
  } else {
    temp = 2;
  }

}
```

Ex. Target branch
(gvar1 + gvar2 <= 10)

Instrumentation

```
int gvar1
int gvar2

void main (void){
  int temp;

  if ((gvar1 + gvar2) > 10) {
    temp = 1;
  } else {
    flag = 1;
    temp = 2;
  }

  assert (!(flag == 1));

}
```

flag becomes 1 when this branch is executed

An assertion to find counter example that falsifies "!(flag == 1)"

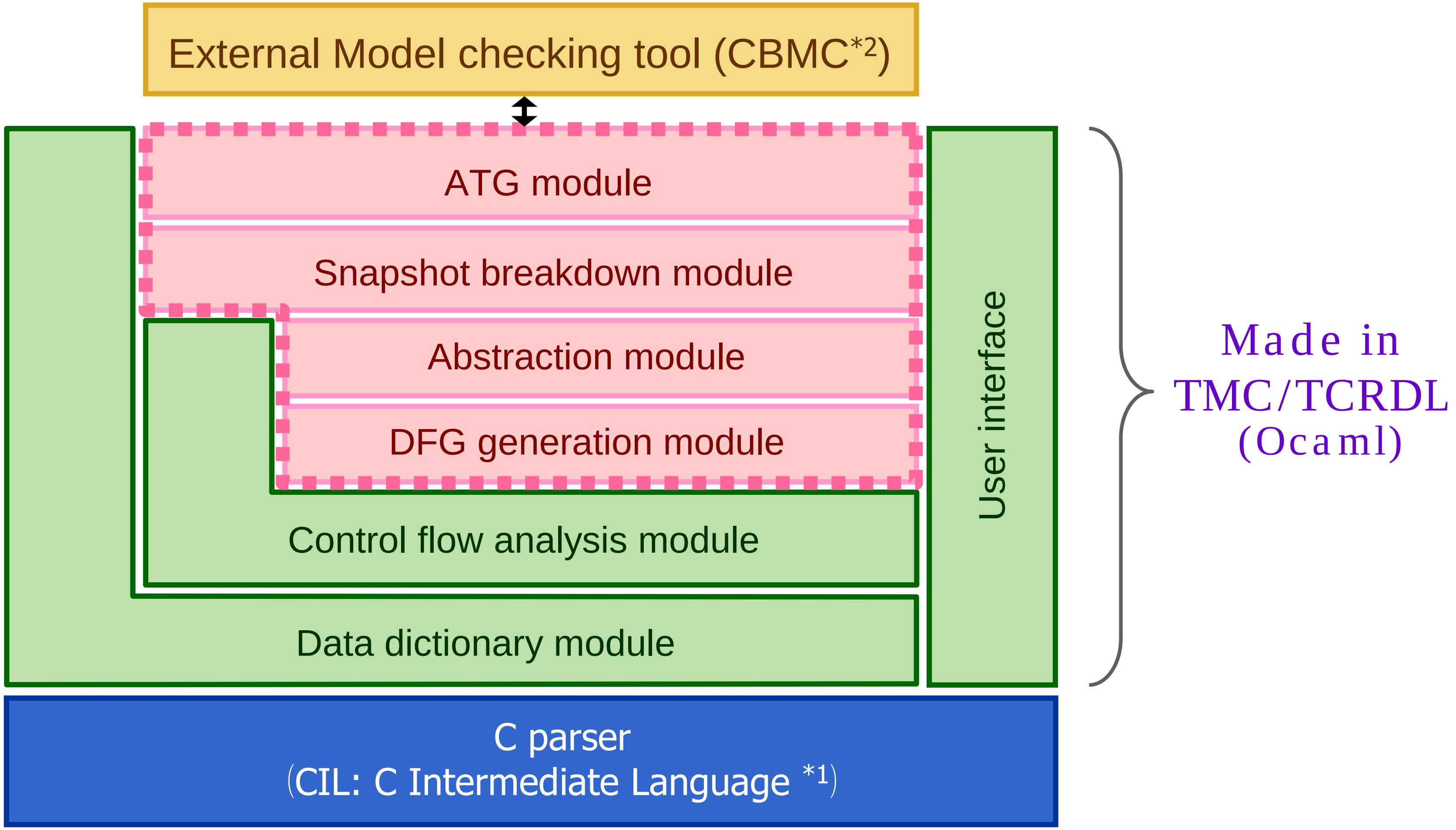
Model checking

gvar1 = 2
gvar2 = 4

||

Test inputs that stimulates the target branch!!

Prototype tool architecture



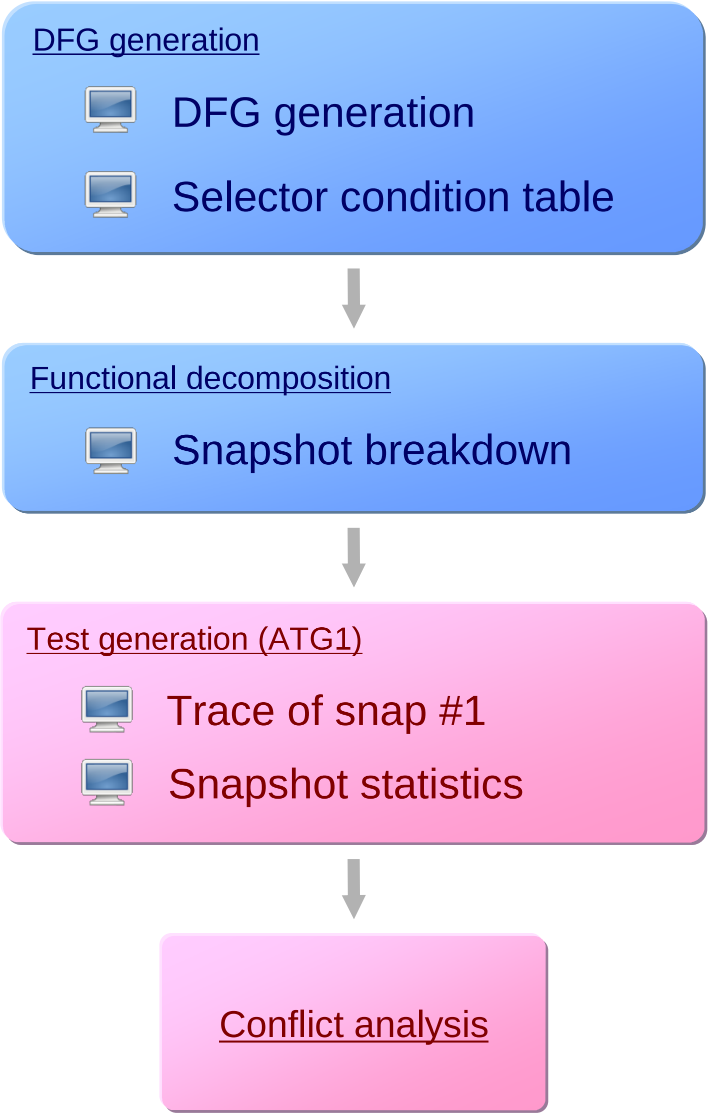
*1 "CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs" by George C. Necula, Scott McPeak, S.P. Rahul and Westley Weimer, in "Proceedings of Conference on Compilier Construction", 2002.

*2 http://www.cprover.org/cbmc/

Example

Input: gin1, gin2
Output: _sc_

```
int gin1, gin2, _sc_  ;

void foo (void)
{
 int P, Q, R, x, y, out ;

 P = (  gin1 == 10 );
 Q = (  gin1 * gin2 > 0 );
 R = ( (gin1 + gin2 < 0) & (gin2 > 5) );

 if (P) {
  x = 1;
 } else {
  x = 2;
 }
 if (Q) {
  y = 3;
 } else {
  y = 4;
 }
 if (R) {
  out = x;
 } else {
  out = y;
 }

 _sc_ = out;
}
```



DFG generation

- DFG generation
- Selector condition table

Functional decomposition

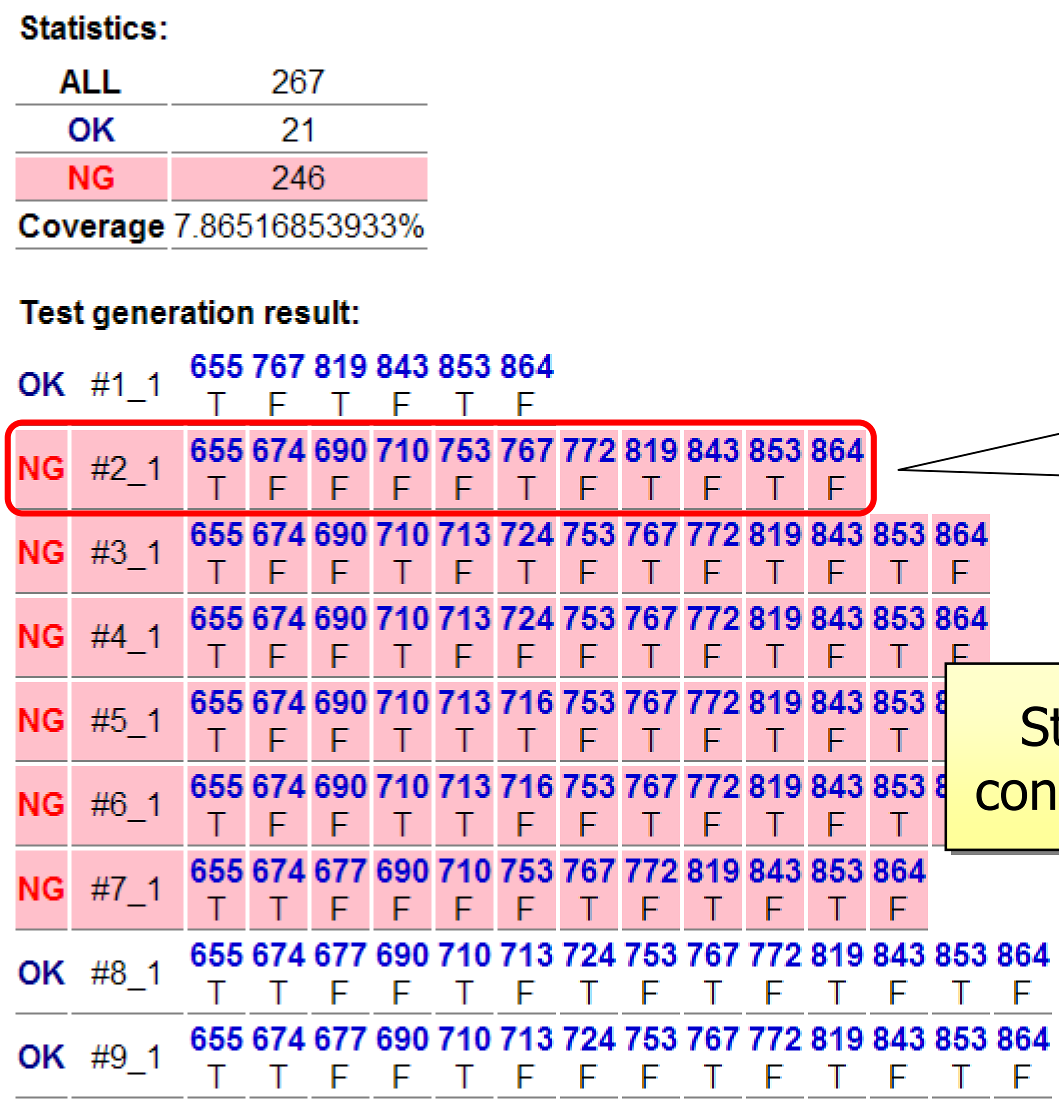
- Snapshot breakdown

Test generation (ATG1)

- Trace of snap #1
- Snapshot statistics

Conflict analysis

Conflict analysis



Statistics:

ALL	267
OK	21
NG	246
Coverage	7.86516853933%

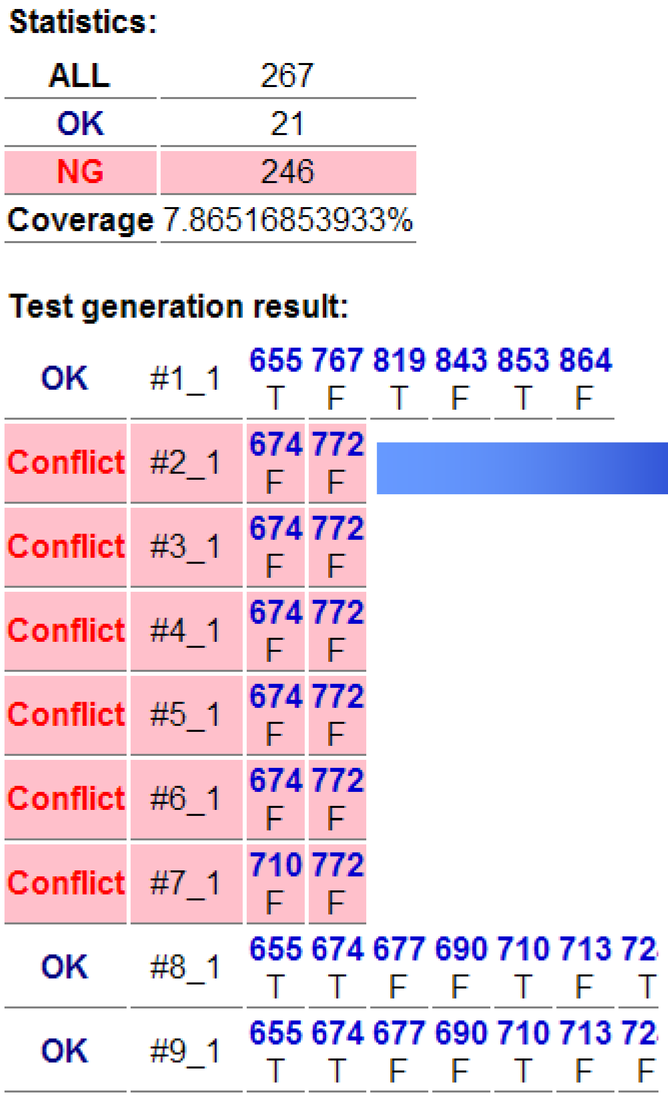
Test generation result:

OK	#1_1	655 T	767 F	819 T	843 F	853 T	864 F								
NG	#2_1	655 T	674 F	690 F	710 F	753 F	767 T	772 F	819 T	843 F	853 T	864 F			
NG	#3_1	655 T	674 F	690 F	710 T	713 F	724 T	753 F	767 T	772 F	819 T	843 F	853 T	864 F	
NG	#4_1	655 T	674 F	690 F	710 T	713 F	724 F	753 F	767 T	772 F	819 T	843 F	853 T	864 F	
NG	#5_1	655 T	674 F	690 F	710 T	713 T	716 T	753 F	767 T	772 F	819 T	843 F	853 T		
NG	#6_1	655 T	674 F	690 F	710 T	713 T	716 F	753 F	767 T	772 F	819 T	843 F	853 T		
NG	#7_1	655 T	674 T	677 F	690 F	710 F	753 F	767 T	772 F	819 T	843 F	853 T	864 F		
OK	#8_1	655 T	674 T	677 F	690 F	710 T	713 F	724 T	753 F	767 T	772 F	819 T	843 F	853 T	864 F
OK	#9_1	655 T	674 T	677 F	690 F	710 T	713 F	724 F	753 F	767 T	772 F	819 T	843 F	853 T	864 F

No test input activating snap#2 was found

Step-by-step identification of root conflict by solving relaxed constraints

Example of the root conflict



Statistics:

ALL	267
OK	21
NG	246
Coverage	7.86516853933%

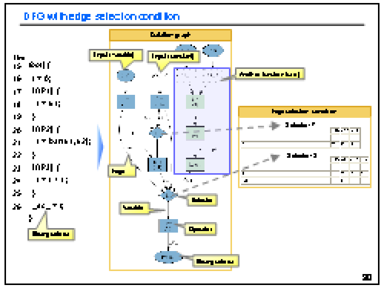
Test generation result:

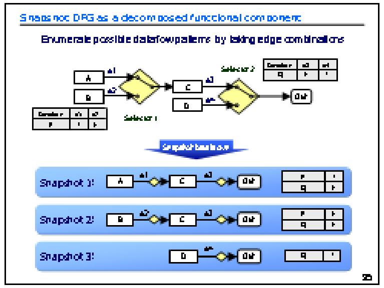
OK	#1_1	655 T	767 F	819 T	843 F	853 T	864 F	
Conflict	#2_1	674 F	772 F					
Conflict	#3_1	674 F	772 F					
Conflict	#4_1	674 F	772 F					
Conflict	#5_1	674 F	772 F					
Conflict	#6_1	674 F	772 F					
Conflict	#7_1	710 F	772 F					
OK	#8_1	655 T	674 T	677 F	690 F	710 T	713 F	72 T
OK	#9_1	655 T	674 T	677 F	690 F	710 T	713 F	72 F

```
#line 674
    if ( flag == 0) {

#line 772
    if ( flag == 1 ) {
```

Existence of conflict is fine.
Unrecognized conflict is the problem.

Summary - Tool implementation


DFG with edge selection condition


Snapshot DFG as a decomposed functional component

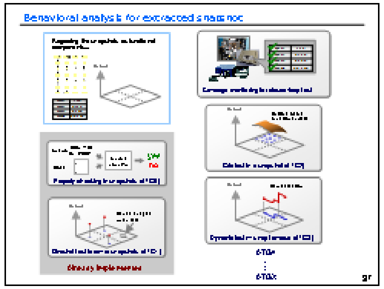
Enumerate possible dataflow patterns by taking edge combinations

Snapshot 1:

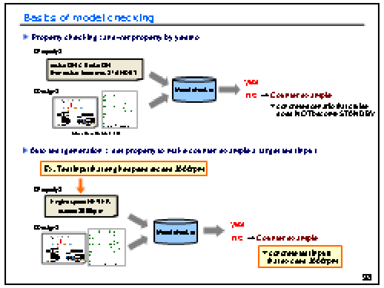
Snapshot 2:

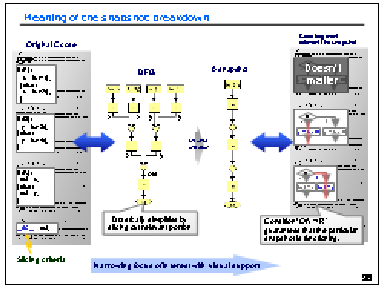
Snapshot 3:

- Dataflow graph and its snapshots as one of the model of functional component


Behavioral analysis for extracted snapshot

- Auto test generators for behavioral analysis


Basics of model checking


Meaning of one snapshot breakdown

Original Code

DFG

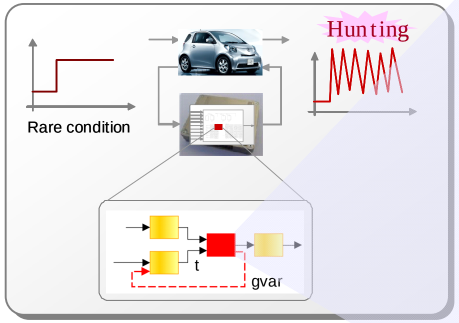
Doesn't matter

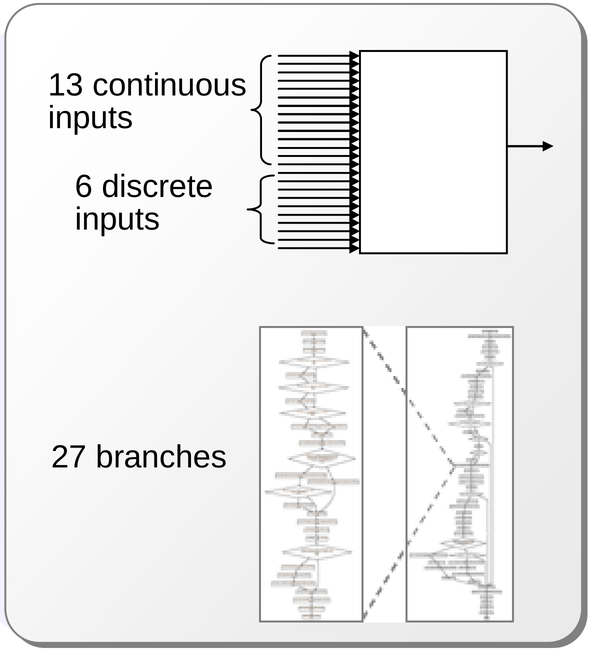
- Model checking based one shot test generator
- Tests pinpointing the particular snapshot
- Conflict analysis

1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction

Monitoring snapshot coverage on SILS

Hunting behavior

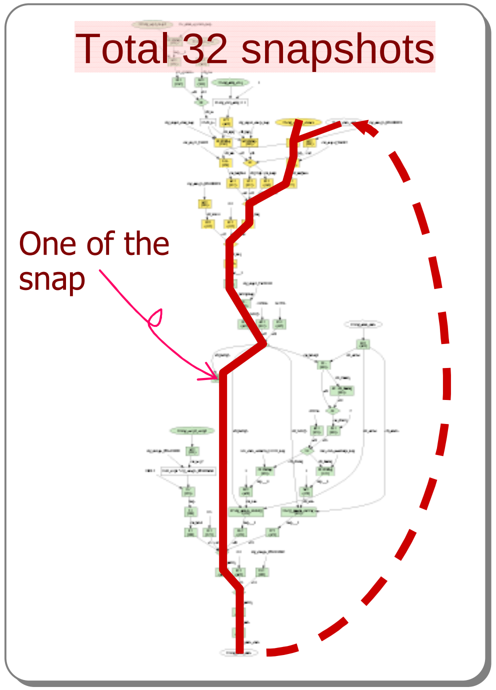




13 continuous inputs

6 discrete inputs

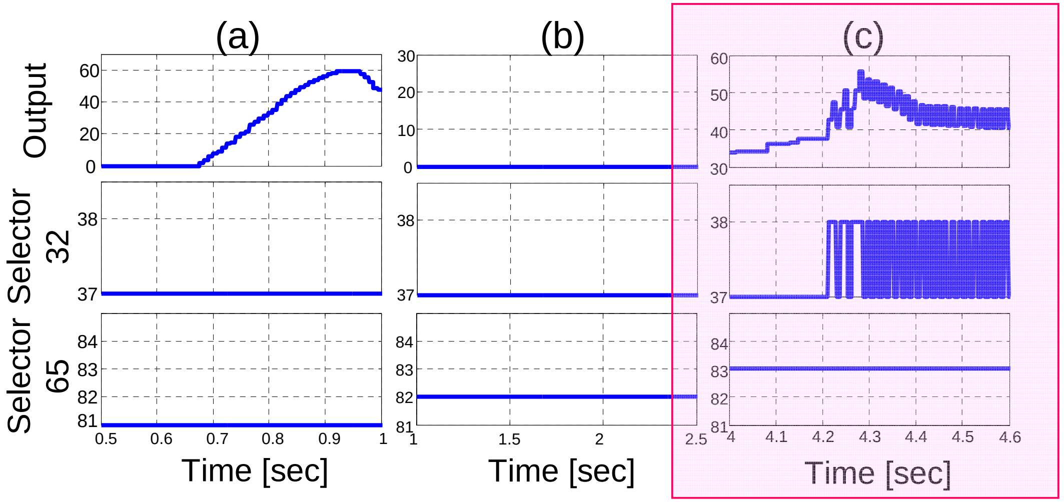
27 branches



Total 32 snapshots

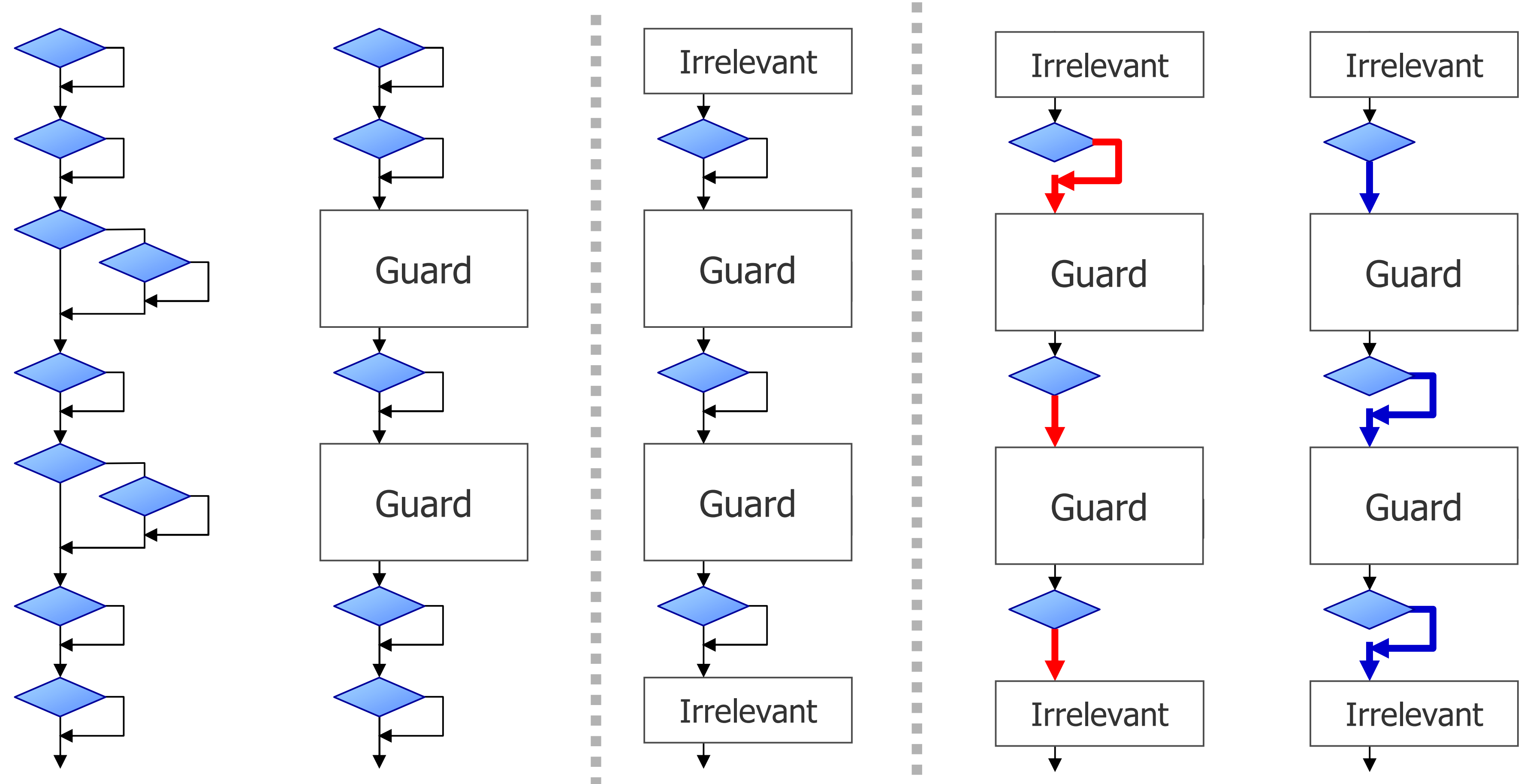
One of the snap

Criteria	# of cases
Input space coverage	$res^{13} * 2^{6}$
Full paths coverage	87000
Branch coverage	54
Condition coverage	66
DFG snapshots	116
DFG snapshots w/ abstraction	32



SILS simulation

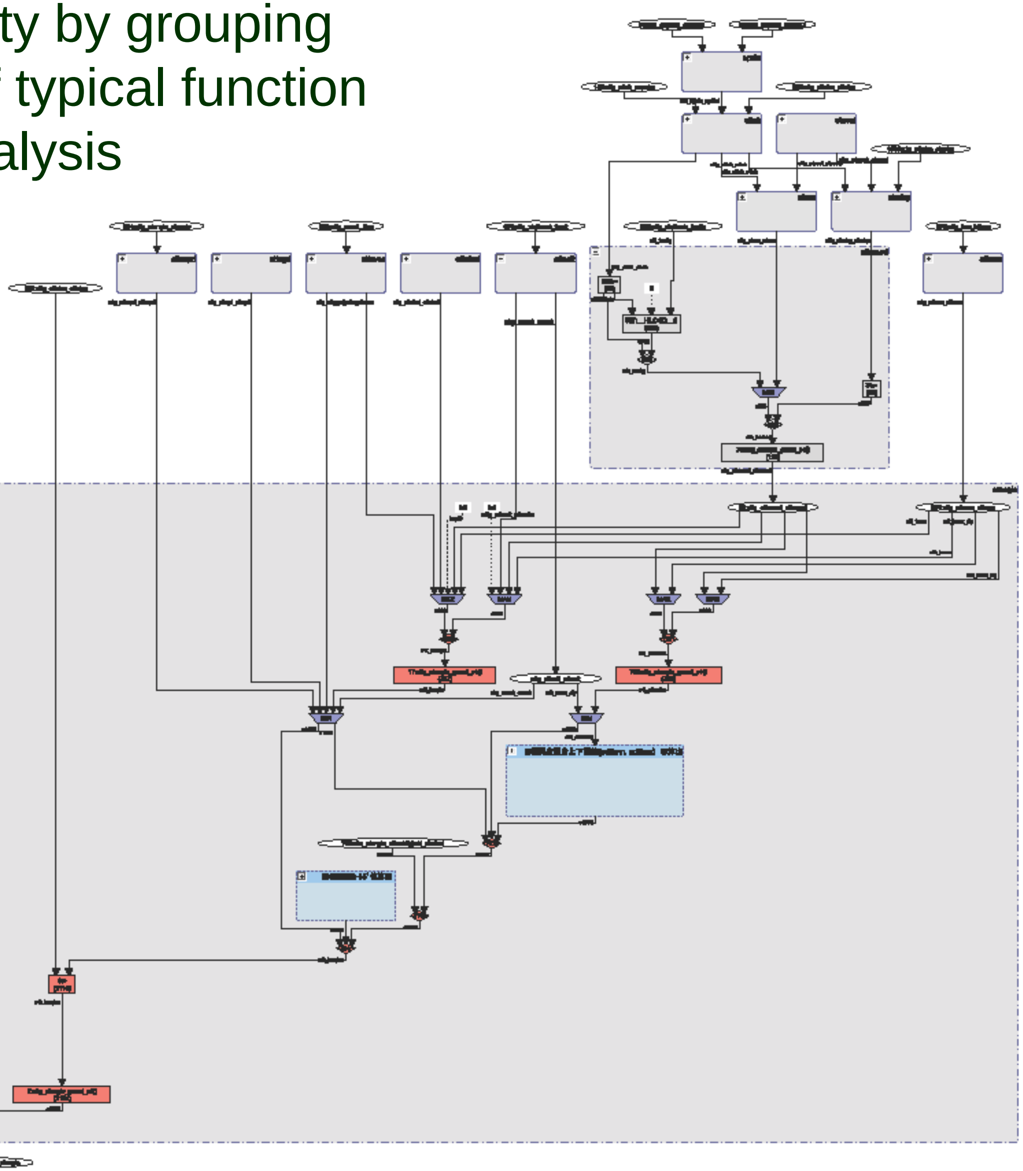
Paths reduction in a production code



CFG		DFG snapshot	ATG1 (one shot test)	
Source Code	Abstraction (trivial branches)	Snapshots	Not executable paths	Executable paths
$10^{8\sim9}$	27,122,688	267	246	21

Architecture analysis of large scale legacy code

- Extracted from C code : 52 files
- Controlling granularity by grouping
- Model abstraction of typical function
- 60hrs by manual analysis



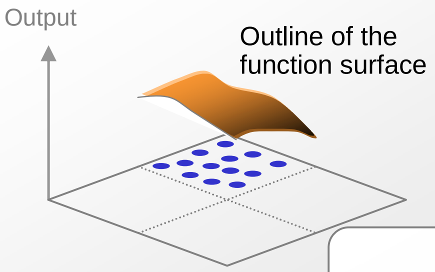
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction

Future direction

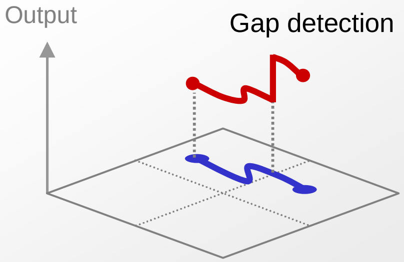
- Tool implementation
  - - DFG extraction from Simulink model
  - - Integration to SILS/HILS environment
- Abstraction level of equivalence class of function
  - - Extract more essential function for larger problems
  - - Other function models
- Auto test generators
- Software modeling

⋮

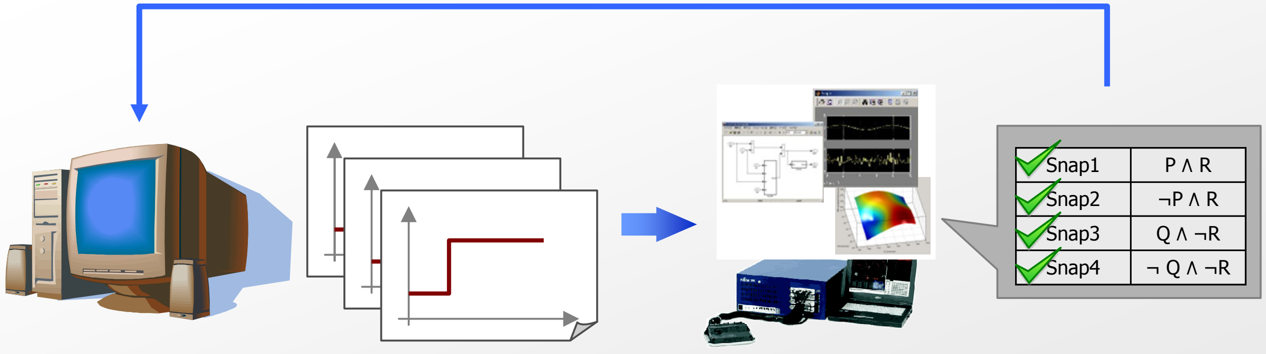
Auto test generators



Grid test in a snapshot (A



Dynamic test w/ snap traverse (ATG3)



✓ Snap1	P ∧ R
✓ Snap2	¬P ∧ R
✓ Snap3	Q ∧ ¬R
✓ Snap4	¬ Q ∧ ¬R

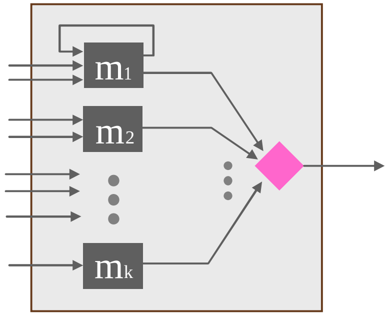
Coverage-guided test generator for closed-loop simulator (ATGx)

Software modeling

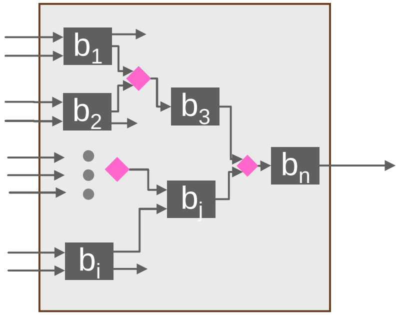
Demand for software modeling:

- Embeddable (can generate C code)
- Helps intuitive understanding of equivalence class of function
- Separation of concern
  - Implementation details ⇔ Essential function
- Unique (no manual synchronization among models)

Centralized mode control



Distributed mode control



+ Clear mode of operation

- Redundant description

+ Compact description

- Ambiguous mode of operation

Is it possible to describe as a static model??

End.