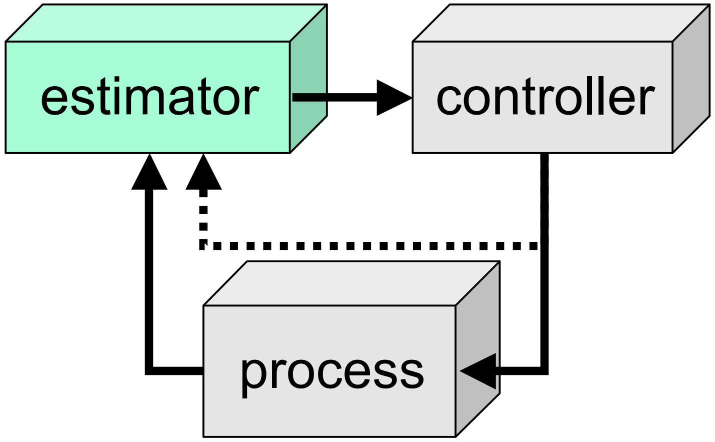
Guided Simulation of Autonomous Controllers

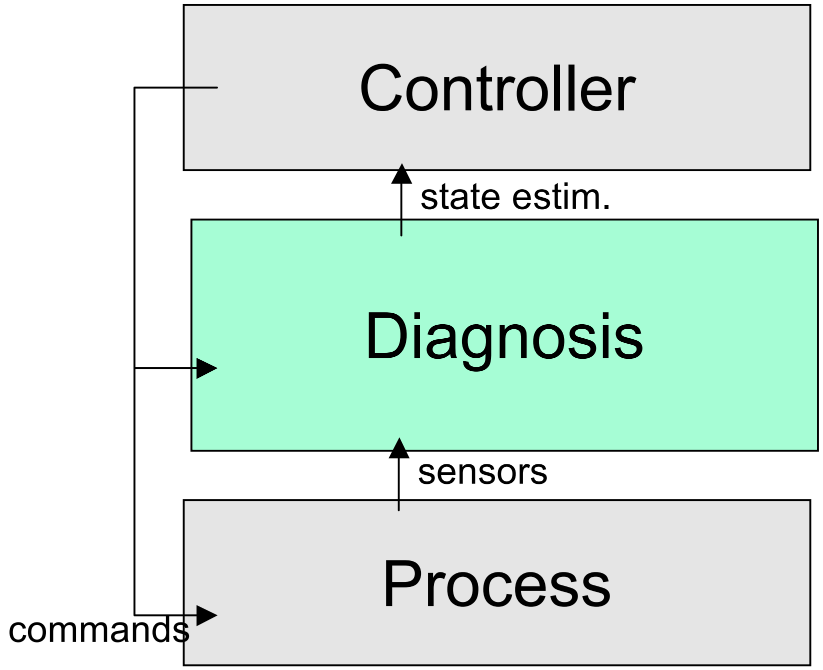
Charles Pecheur (UC Louvain)

with Tony Lindsey (QSS Group)
performed at NASA Ames Research Center

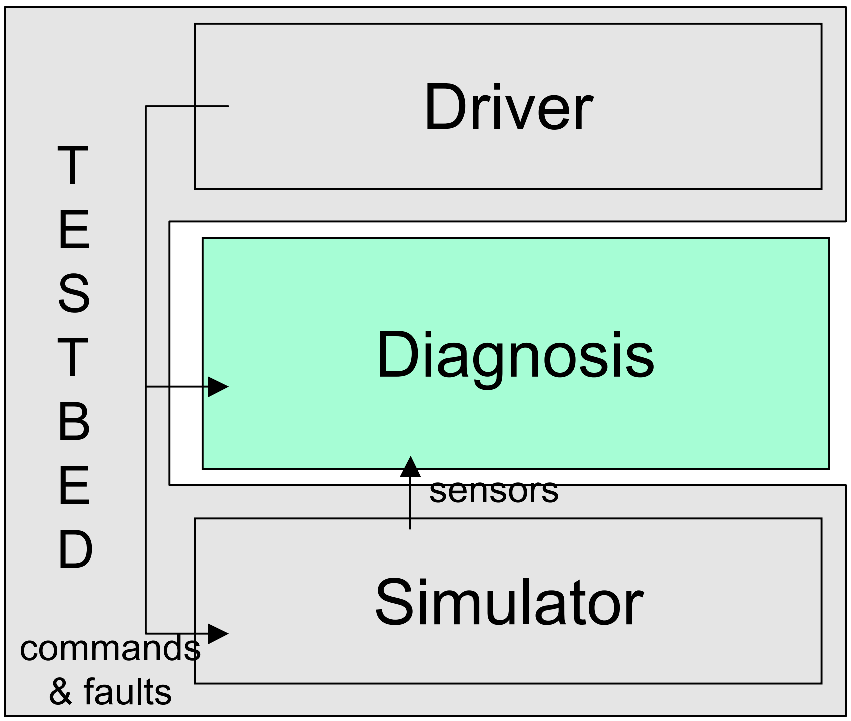
Ames Research Center

Diagnosis

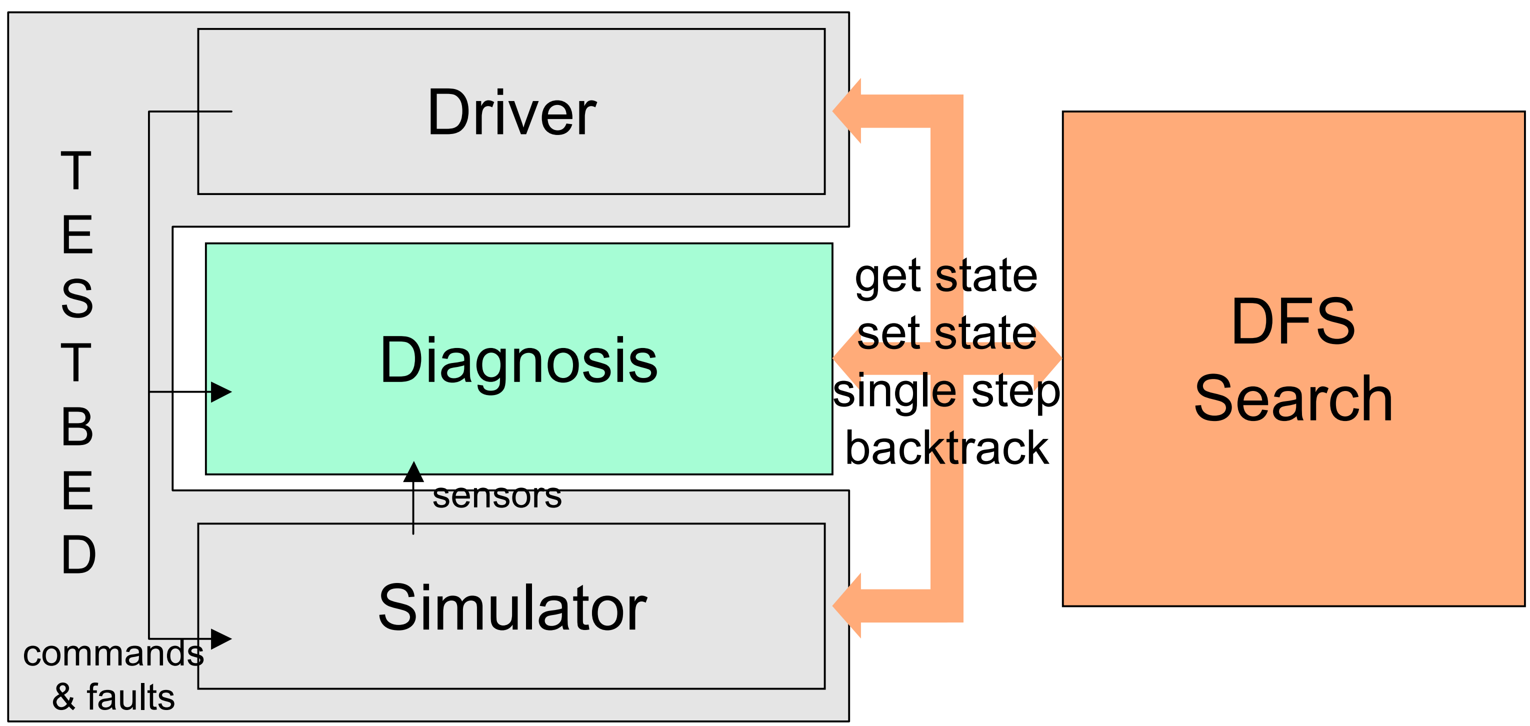

estimator
controller
process


Controller
state estim.
Diagnosis
sensors
Process
commands

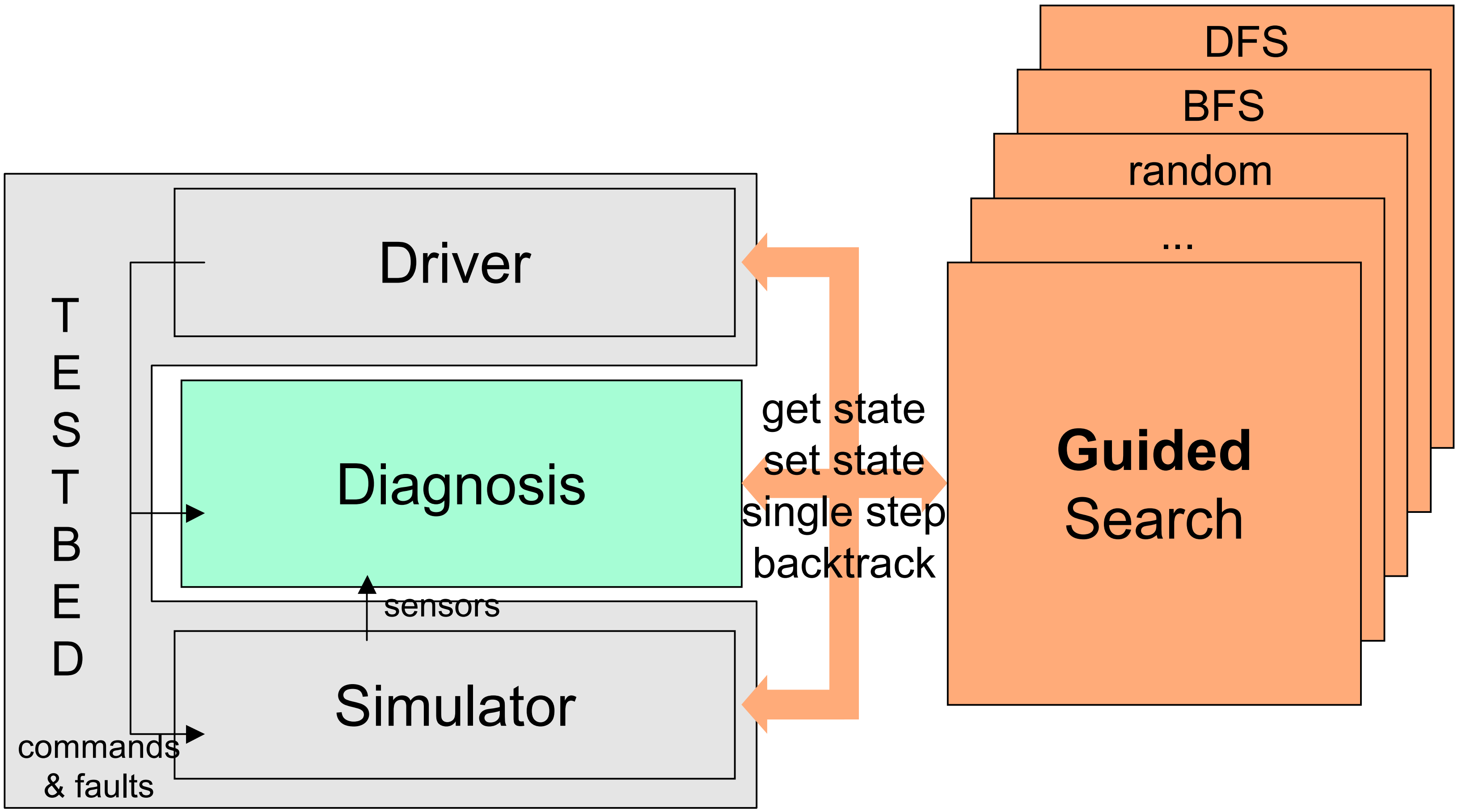
Diagnosis + Testbed



Diagnosis + Testbed + Search



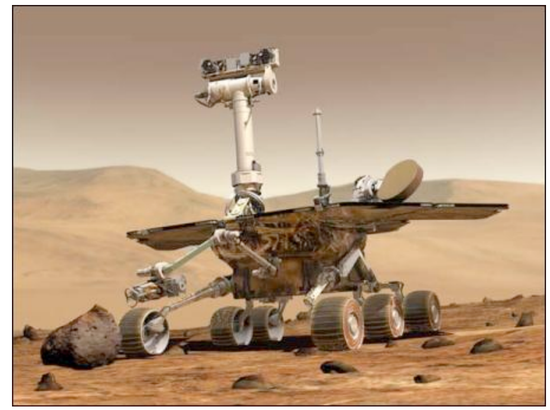
Diagnosis + Testbed + Searches

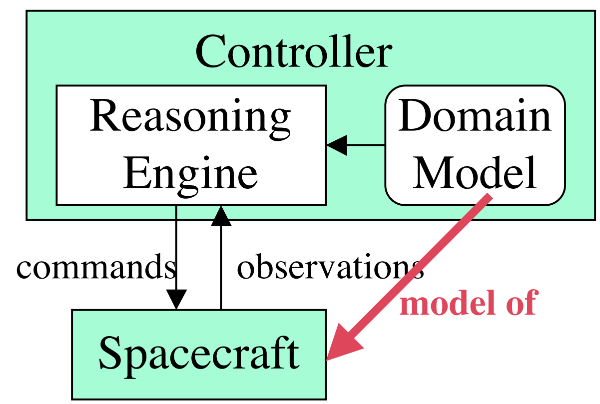


Autonomy (in Space)

Autonomous spacecraft = on-board intelligence (AI)

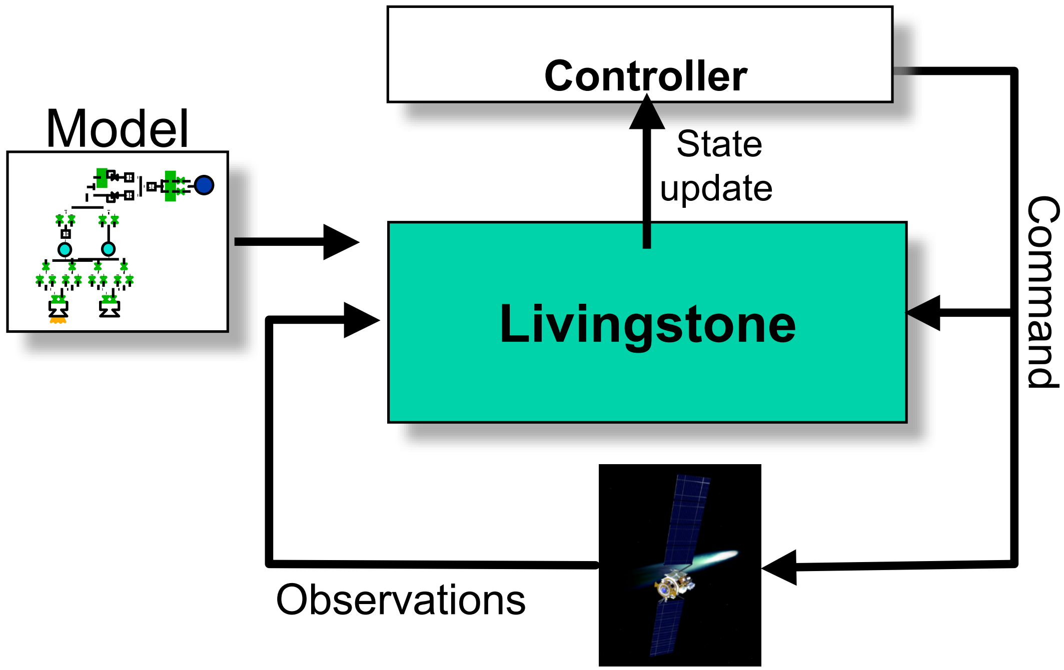
- **Goal:** Unattended operation in an unpredictable environment
- **Approach:** **model-based** reasoning
- **Pros**: smaller mission control crews, no communication delays/blackouts
- **Cons:** **Verification and Validation ???** Much more complex, huge state space
- Better verification is critical for adoption





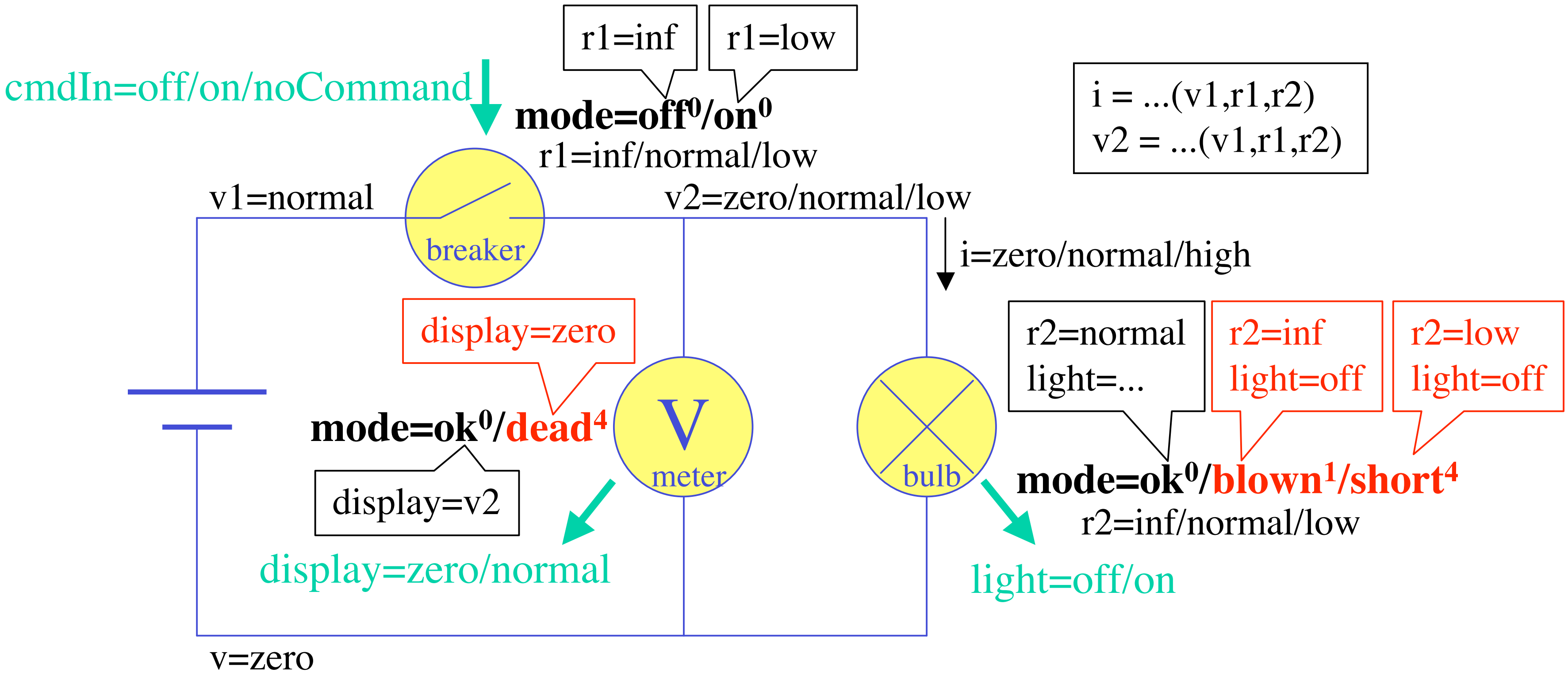
Model-Based Diagnosis

- Focus on **Livingstone** system from NASA Ames.
- Uses a discrete, qualitative model to reason about faults



Courtesy Autonomous Systems Group, NASA Ames

A Simple Diagnosis Model



Goal: determine **modes** from observations
Generates and tracks *candidates*

breaker	bulb	meter	rank
off^0	**ok^0**	**ok^0**	**0**
off^0	**ok^0**	**blown^1**	**1**
on^0	**dead^4**	**short^4**	**8**

Faults vs. Errors

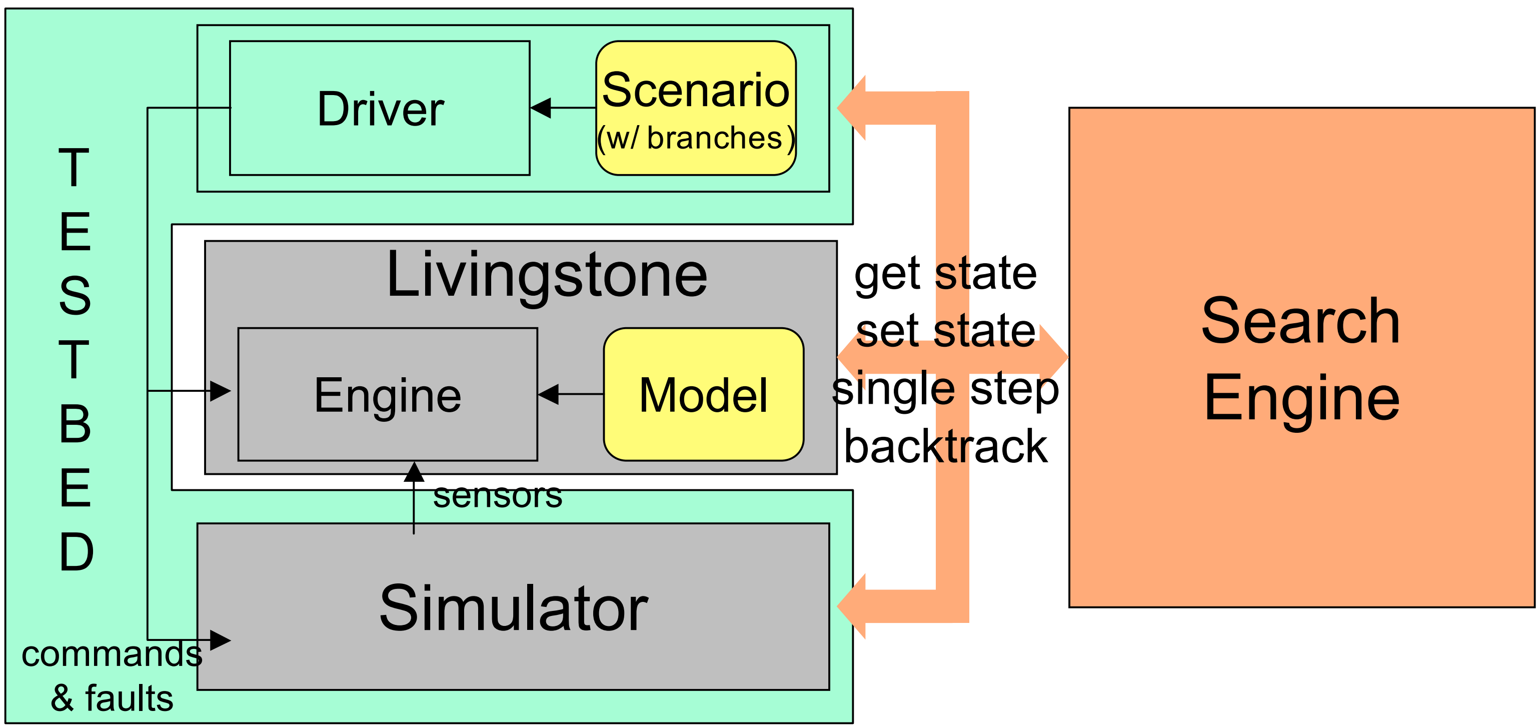
Faults	Errors
Ex: valve is stuck	Ex: fault not detected
in Process/Simulator	in Diagnosis/Design
Spontaneous physical event	Human design flaw
To de detected by Diagnosis	**To be detected by Verification**

Verification of Model-Based Autonomy

Two complementary approaches:

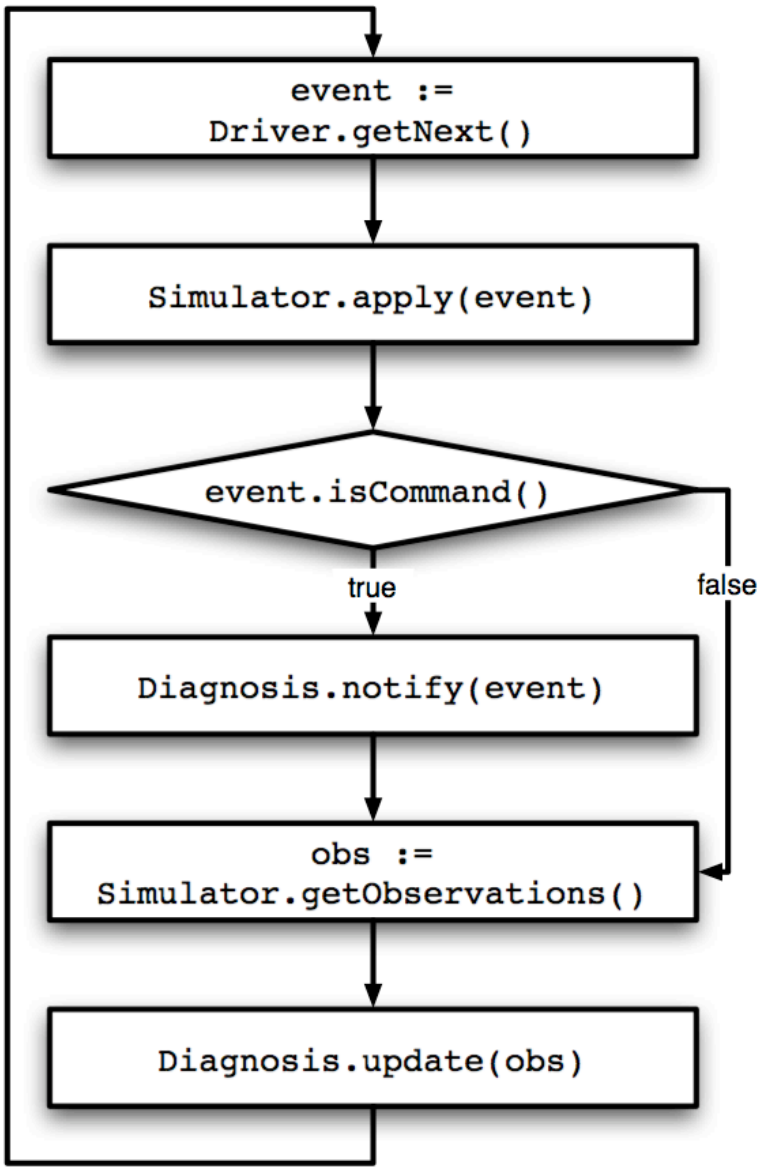
- Model-based verification
  - Analyze the model
  - That's the application "source code"
  - Symbolic model checking (NuSMV, SAT-based BMC)
- **Simulation-based verification**
  - Analyze the whole program (engine+model+testbed)
  - More comprehensive but less coverage
  - Controlled execution of the actual program

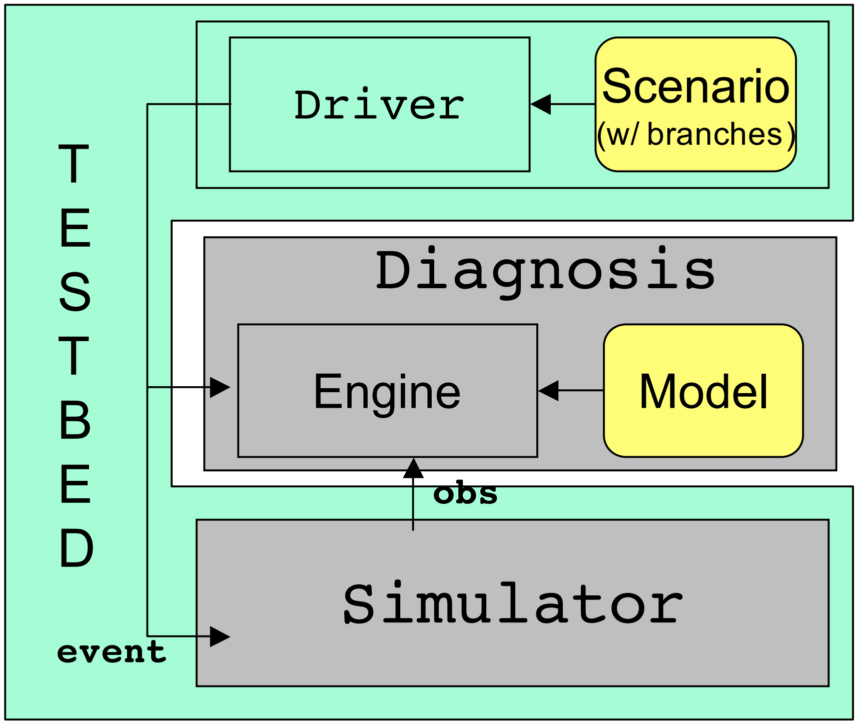
Livingstone PathFinder (LPF)



- Similar to VeriSoft[Godefroid 97]
- Uses checkpointing implemented in Livingstone
- In Java, accesses Livingstone (C++) through JNI

One Diagnosis Step





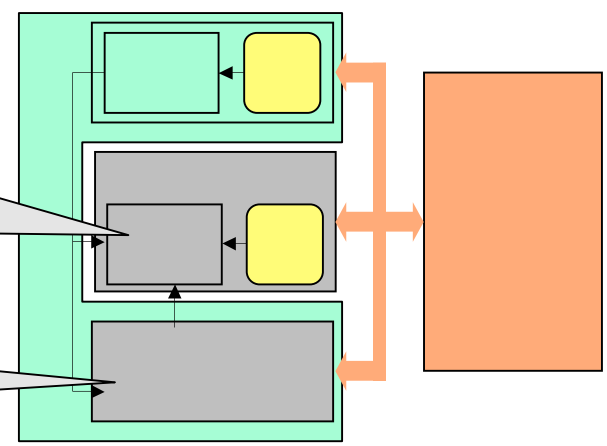
LPF Error Conditions

- Diagnosis candidates are "correct" w.r.t. Simulator modes
  - Mode Comparison (MC): first candidate is correct
  - Candidate Matching (CM): some candidate is correct
  - Candidate Subsumption (CS): some candidate's faults are included
- CS may miss errors but works best in practice

MC ✗
CM ✗
√ CS

breaker	bulb	meter
off^0	**ok^0**	**ok^0**
off^0	**ok^0**	**blown^1**
on^0	**dead^4**	**short^4**

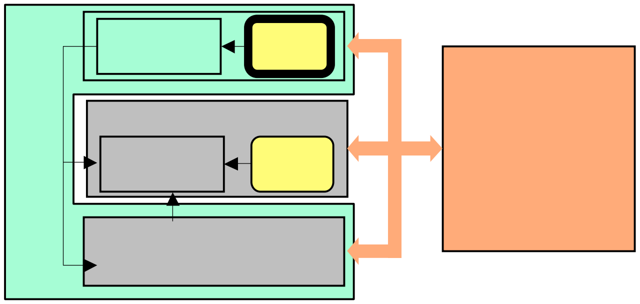
breaker	bulb	meter
off^0	**dead^4**	**blown^1**



LPF Simulation Scenarios

- Defines the tree of executions to be explored
- Described as a non-deterministic program using a simple scripting language

```
stmt  ::= " event "  ;                      single event
        |  { stmt* }                         sequence
        |  mix stmt (and stmt)*          interleaving
        |  choose stmt (or stmt)*           choice
```

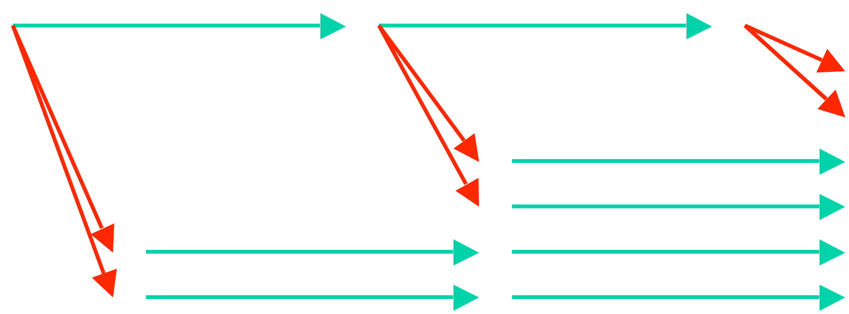


- Implemented as a hierarchy of automata objects matching the scenario script structure

UCL Université catholique de Louvain INGI

LPF Scenario Example

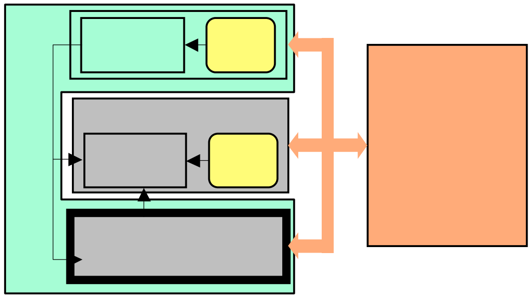
```
mix {
   "command test.sv02.valveCmdIn=close";
   "command test.sv02.valveCmdIn=open";
   ...
} and {
   choose
     "fault test.forwardLO2.mode=unknownFault"; or
     "fault test.mpre101p.mode=faulty"; or
     ...
}
```



- Sequence of commands || choice of faults
- "default" scenario, can be generated automatically

LPF Simulators

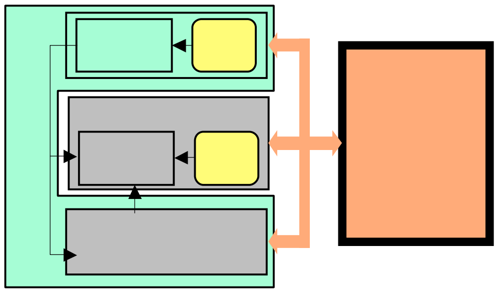
- Framework allows to use any (suitably instrumented) simulation software
  - Trade-off: higher-fidelity simulators may restrict instrumentation
- Current implementation uses **second Livingstone engine** as simulator
  - Same or different model
  - Different mode of operation:
    Diagnosis : cmds, obs –> modes
    Simulator : cmds, modes –> obs
  - Simulator comes "for free"
  - Rationale: verify diagnosis assuming the model is correct



- Also considered: CONFIG (hybrid, NASA JSC)

LPF Search

- The whole testbed is seen as a transition system
- API to enumerate transitions, backtrack, get/set state
  - Shared with Java PathFinder (v.2)[Visser et al. 00]
  - Principle inspired from OPEN/CAESAR[Garavel 98]
- Search engine fixes exploration strategy
  - Depth-First
  - Breadth-First
  - **Heuristic**
  - Others are possible (random, pattern-based, interactive)
- + Halting conditions (for any strategy)
  - Find first / all / shortest error trace(s)



LPF Heuristic Search

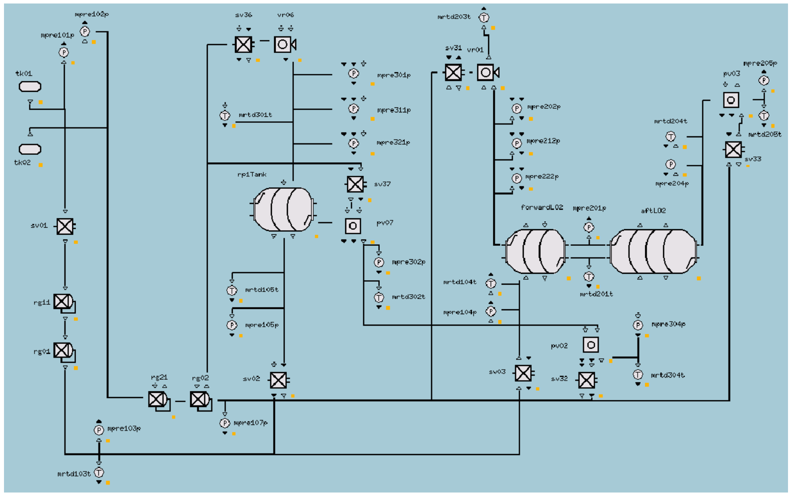
- Based on valuation function (heuristic) on states
- Greedy best-first search (priority queue)
- Current heuristic: **candidate-count**
  - number of diagnosis candidates
  - Less is better
  - Progress towards absence of valid candidate

Other Heuristics?

- Structural coverage
  - model states, faults, fault pairs, ...
  - or scenario events, ...
  - function on traces (or extended states) rather than states
- Ranks (probabilities)
  - of actual (simulator) states or estimated (candidates)
  - higher or lower probability best
    - ... except for nominal transitions
- To probe further...

Application: PITEX

- Propulsion feed system of space vehicle
- Livingstone model: 2300 lines, 823 vars, ≈10^{33} states (SMV)
- Two scenarios:
  - Random Scenario (10216 states): sequence of commands || choice of faults
  - PITEX Scenario (89 states): combines 29 test cases used by application team



LPF on PITEX: Results

scenario	strategy	search	condition	errors	non-trivial	states	states/min
baseline	DFS	all	CM	27	4	89	44
baseline	DFS	all	CS	0	0	89	67
random	DFS	all	CM	9621	137	10216	51
random	DFS	all	CS	5	5	10216	52

scenario	strategy	search	condition	max. depth	states	states/min
random	DFS	one	CS	16	8648	49
random	BFS	one	CS	3	154	38
random	CC	one	CS	5	154	38

DFS=depth-first, BFS=breadth-first, CC=candidate-count
all=all errors, one=first error, min=shortest trace
CM=candidate matching, CS=candidate subsumption

trivial error=no fault reported

Perspectives

- Extend search options
  - More heuristics (including application-specific)
  - New search strategies (randomized, coverage-based)
- Improve usability
  - GUI, post-process and display results
- Generalize to reactive control
  - From fault detection to fault recovery
  - Uncompleted: adapt LPF to Titan (MIT)
- Other approach: apply SMV (and BMC) to Livingstone models, verify diagnosability[Cimatti et al. 03]
  - using Livingstone-to-SMV translator[Pecheur et al. 00]

Extra Slides

Verification of Diagnosis systems

Verify what?

1. Model Correctness: the model is OK
   i.e. the model is a valid abstraction of the process
2. Engine Correctness: the software is OK
   i.e. all that can be diagnosed is correctly diagnosed
3. **Diagnosability: the design is OK**
   i.e. all that needs to be diagnosed can be diagnosed

In principle, 1+2+3 => diagnosis will be correct

Here we look at 3 only!

PITEX Scenarios

```
mix {
  "command test.sv02.valveCmdIn=close";
  "command test.sv02.valveCmdIn=open";
  ...
} and {
  choose
    "fault test.forwardLO2.mode=unknownFault"; or
    "fault test.mpre101p.mode=faulty"; or
    ...
}
```

```
{
  choose { "fault test.mpre202p.mode=biased"; }
  or { "fault test.mpre212p.mode=biased"; }
  or {
    "command test.sv31.valveCmdIn=open";
    choose {
      "fault test.sv31.sv.mode=stuckOpen";
      "command test.sv31.valveCmdIn=close";
    } or {
      "command test.sv31.valveCmdIn=close";
       ...
} } }
```