Verification of Intelligent Software

Charles Pecheur (RIACS / NASA Ames)

Contents

Model Checking for Intelligent Software

- Why?

  Intelligent software, how to verify it?

- What?

  A bird's-eye view of model checking

- How?

  Experiences in the ASE Group

Contents

Model Checking for Intelligent Software

- Why?

  Intelligent software, how to verify it?

- What?

  A bird's-eye view of model checking

- How?

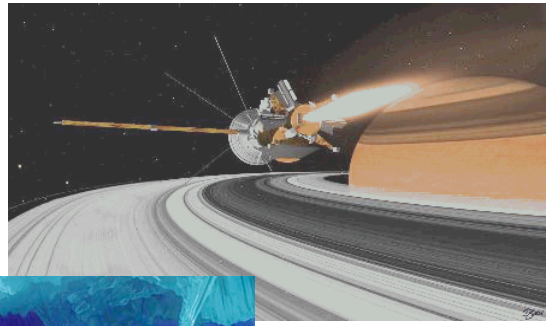
  Experiences in the ASE Group

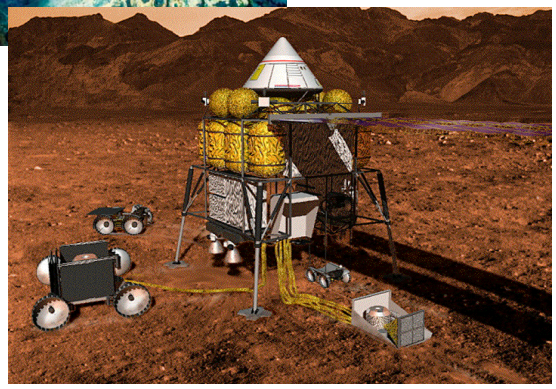
Autonomous Systems

Deep space mission spacecrafts

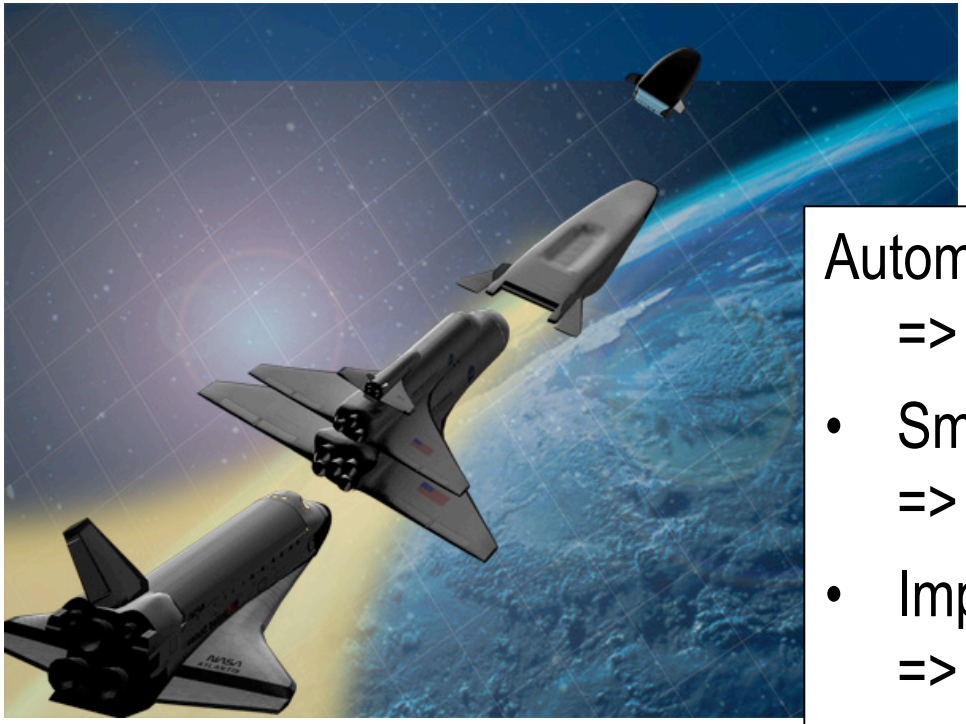
=> add on-board intelligence

- From self-diagnosis
  to on-board science.
- Smaller mission control crews
  => reduced cost
- Less reliance on control link
  => OK for deep space





Integrated Vehicle Health Maintenance

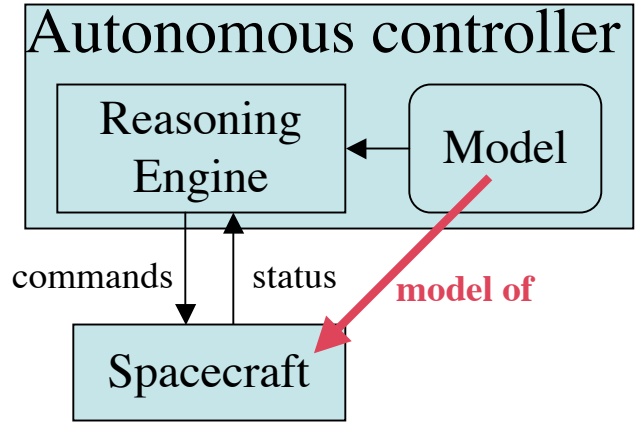


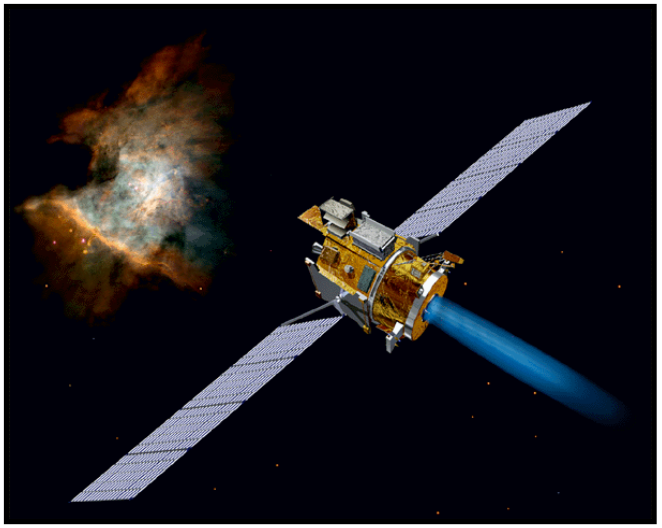
Automated analysis of vehicle data
=> improved diagnosis and prognosis

- Smaller mission control crews
  => reduced cost
- Improved health knowledge
  => optimize maintenance costs, reduce risk

Model-Based Autonomy

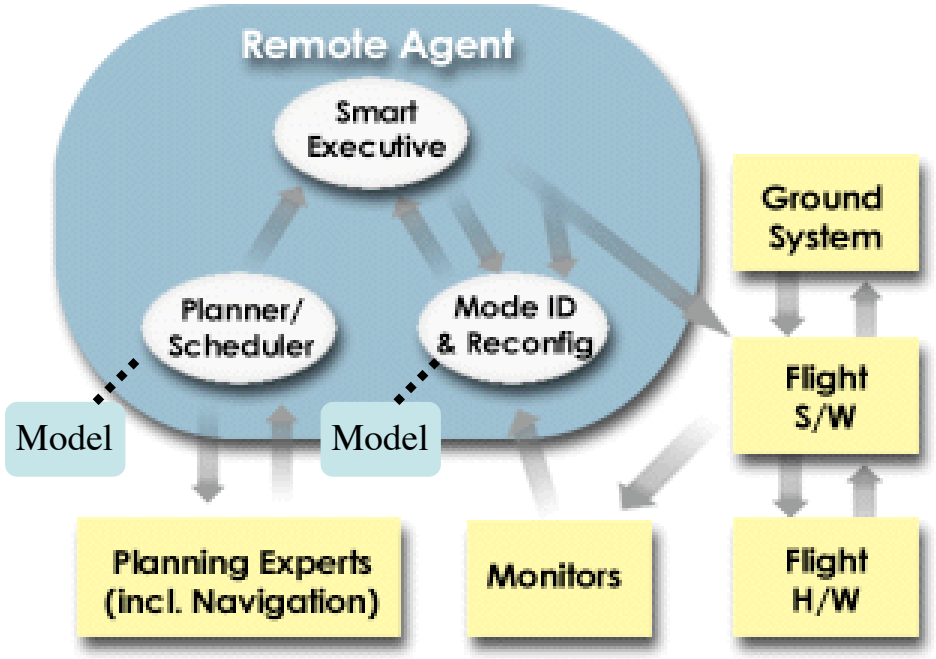
- Based on AI technology
- General reasoning engine + application-specific model
- Use model to respond to unanticipated situations





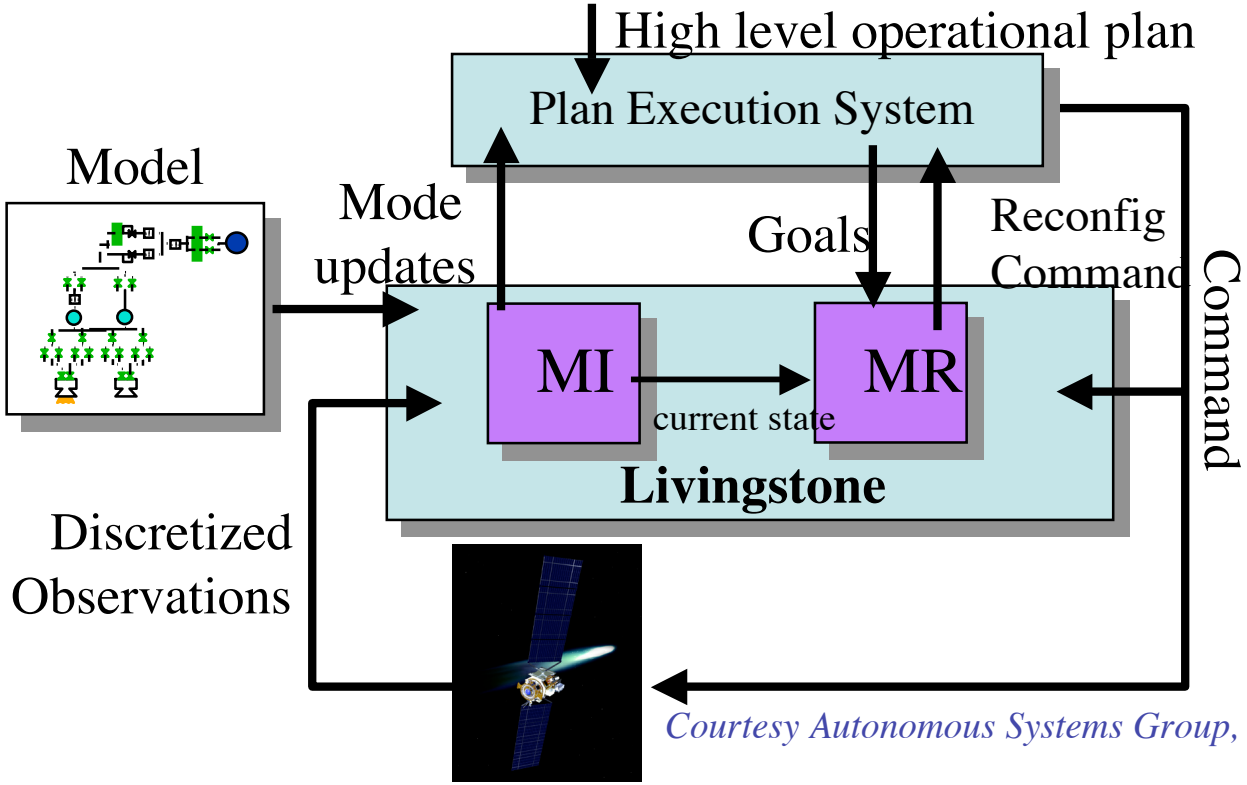
Example: Remote Agent

- From Ames and JPL
- On Deep Space One in May 1999 (1st AI in space!)



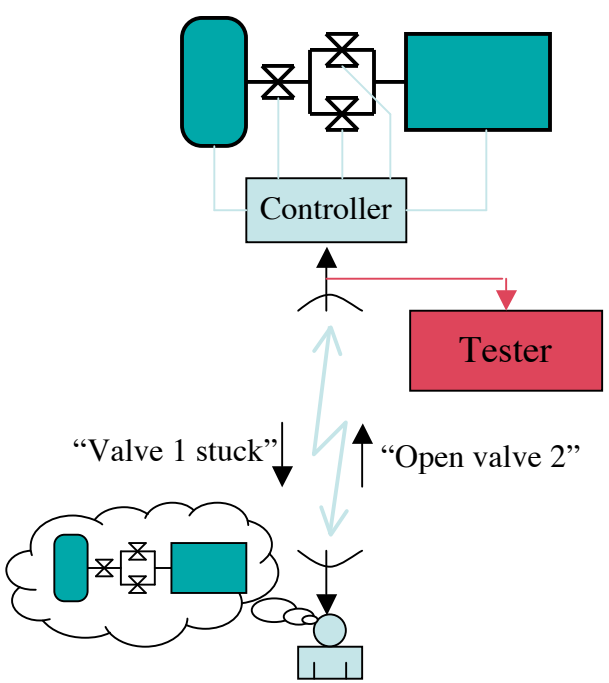
Livingstone
Model-Based Diagnosis

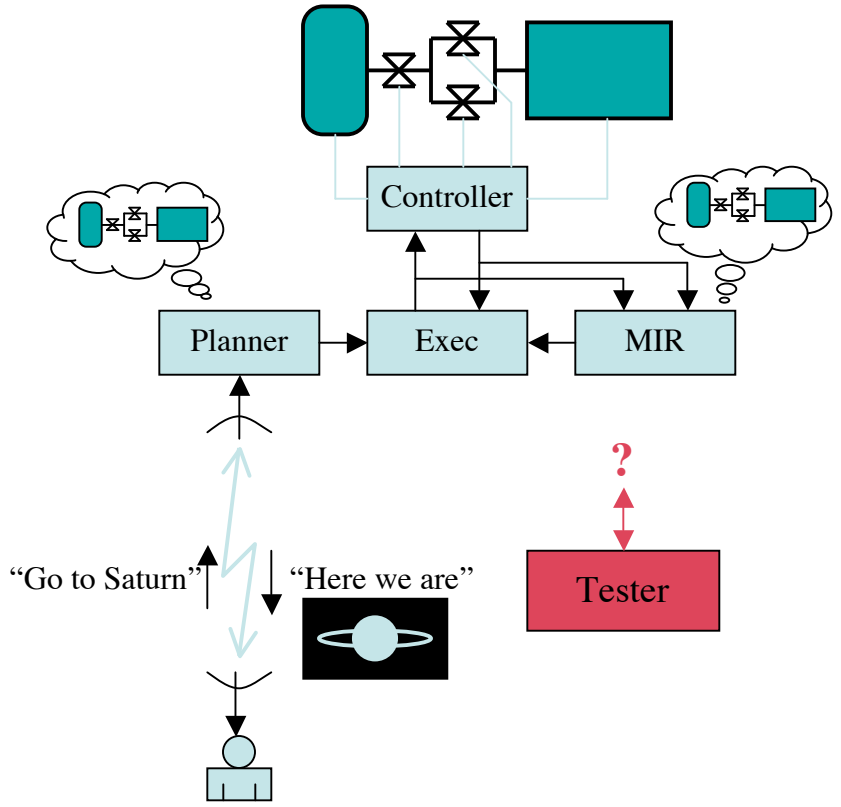
Remote Agent's model-based fault recovery sub-system



Courtesy Autonomous Systems Group, NASA Ames

Controlled vs. Autonomous





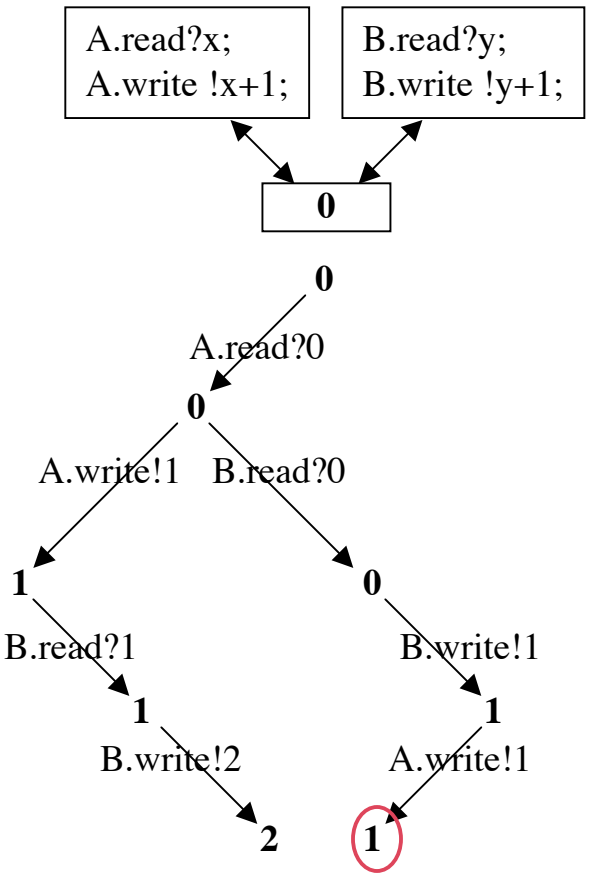
Testing intelligent software?

- Programs are much more complex
- Many more scenarios

  => testing gives low coverage
- Concurrency!

  Due to scheduling,
  the same inputs (test) can give
  different outputs (results)

  => test results are not reliable



Contents

Model Checking for Intelligent Software

- Why?

  Intelligent software, how to verify it?

- What?

  A bird's-eye view of model checking

- How?

  Experiences in the ASE Group

Model Checking

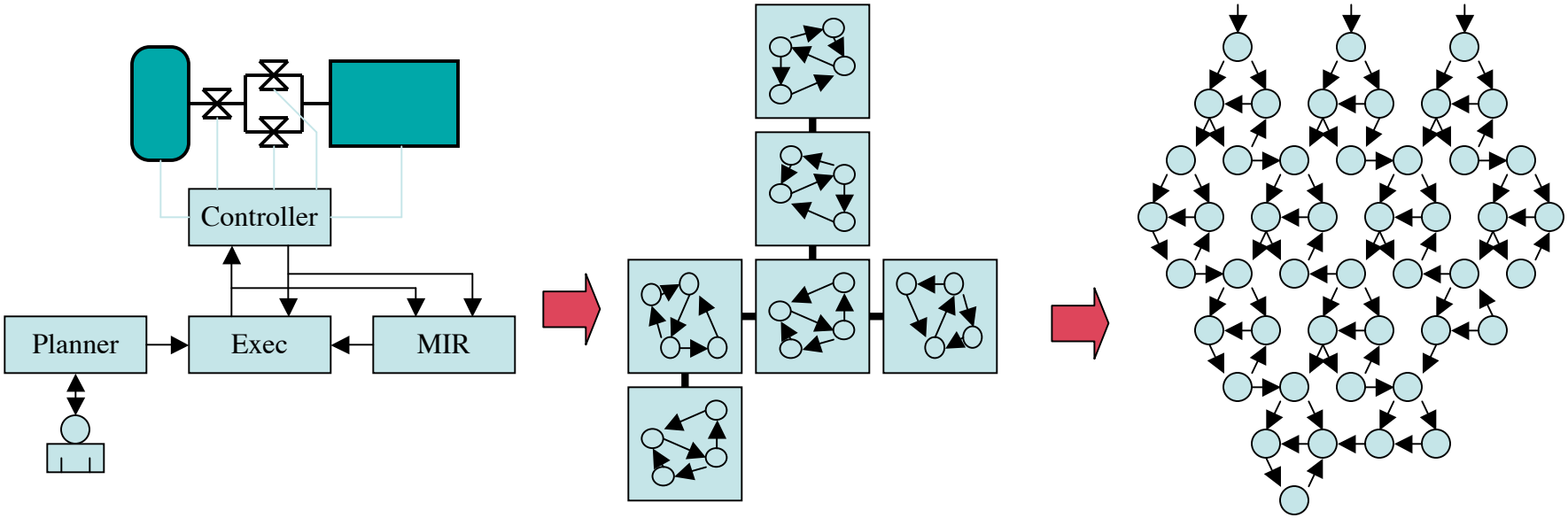
Check whether a system S satisfies a property P by exhaustive exploration of all executions of S

- Controls scheduling => better coverage
- Can be done at early stage => less costly
- Widely used in hardware, coming in software
- Examples: Spin (Bell Labs), Murphi (Stanford)

Model ...

Modeling
Abstraction

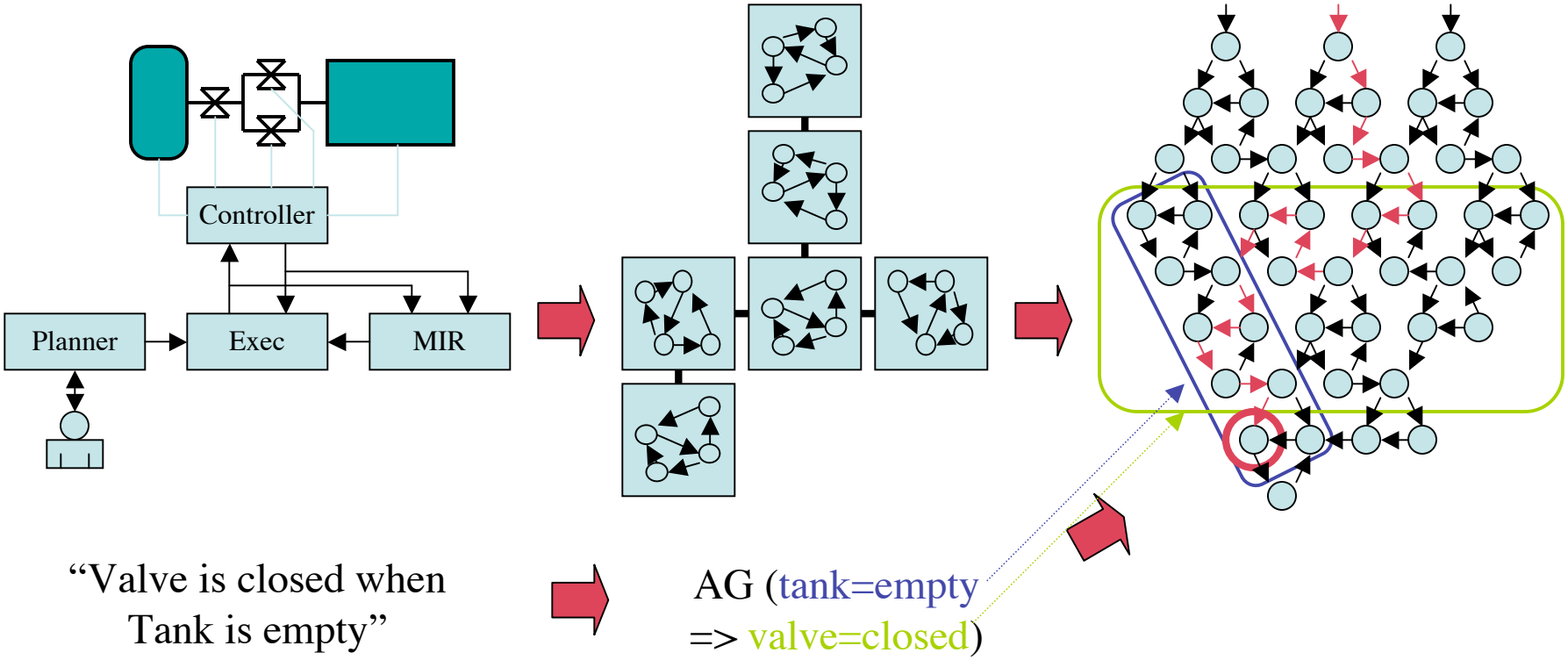
Verification



Model Checking

Modeling Abstraction

Verification



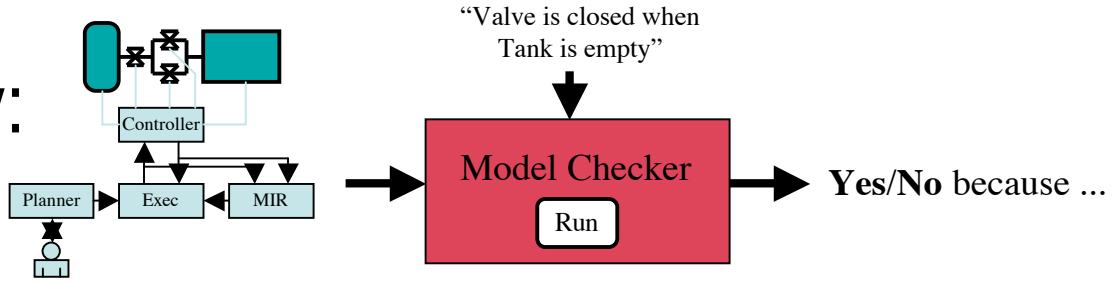
"Valve is closed when Tank is empty"

AG (tank=empty => valve=closed)

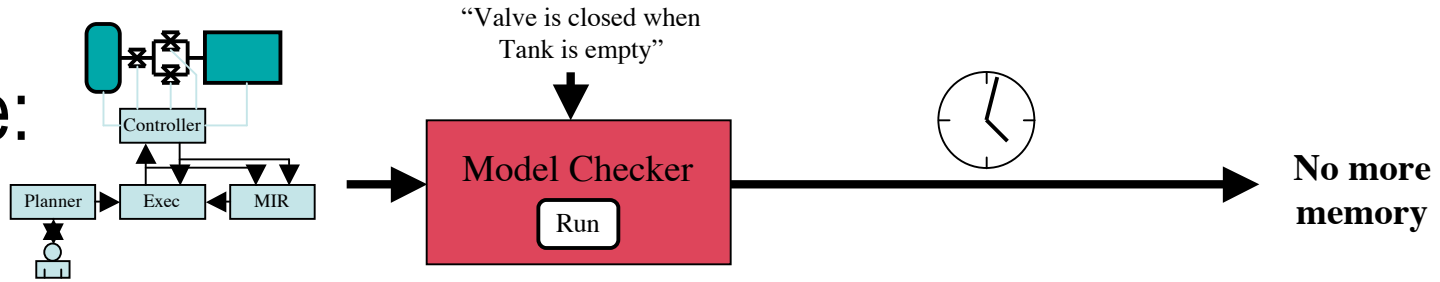
State Space Explosion

K processes with N local states ≤ N^K global states

Theory:

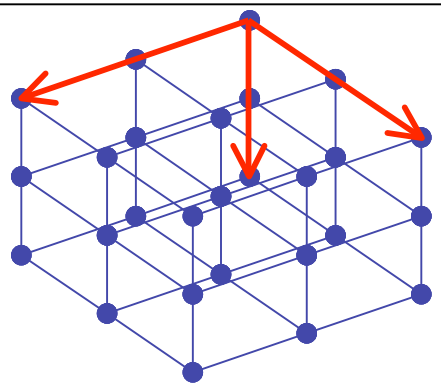


Practice:



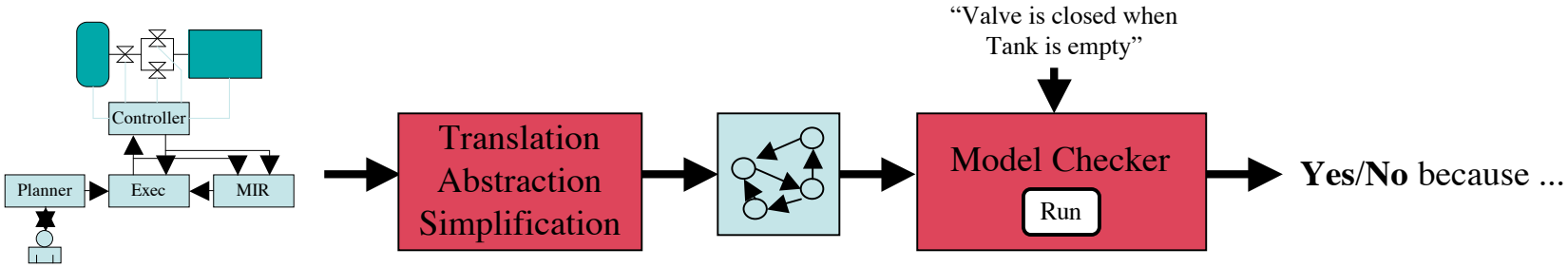
State Space Explosion
(by the numbers)

- *k* independent threads of *n* steps each =
  - $(n+1)^k$ states
  - $k(n+1)(n)^{k-1}$ transitions
  - $(kn)! / (n!)^k$ execution paths



threads	steps	states	trans	paths
k	n	(n+1)^k	k*n*(n+1)^(k-1)	(k*n)! / (n!)^k
3	2	27	54	90
1	10	11	10	1
2	10	121	220	184756
3	10	1331	3630	5.551E+12
4	10	14641	53240	4.7054E+21
5	10	161051	732050	4.8335E+31

Modeling



This is the tough job!

- **Translation**: to model checker's syntax
  e.g. C —> Promela (Spin)
- **Abstraction**: ignore irrelevant parts
  e.g. contents of messages
- **Simplification**: downsize relevant parts
  e.g. number of processes, size of buffers

Temporal Logic

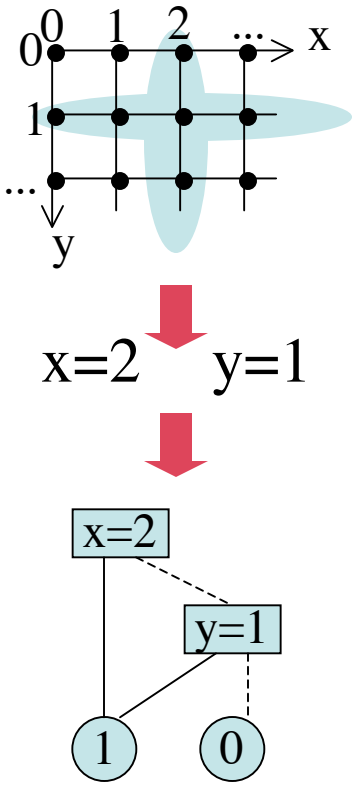
- Propositional logic + quantifiers over executions
- Example: "every request gets a response"

  AG (Req => **AF** Resp)

  Always Globally, if Req then **Always Finally** Resp
- Branching (CTL) vs. linear (LTL)
  - different verification techniques
  - neither is more general than the other
- Model checking without TL
  - Assertions, invariants
  - Compare systems, observers

Symbolic Model Checking

- Manipulates sets of states,
  Represented as boolean formulas,
  Encoded as binary decision diagrams.
- Can handle large state spaces (10^{50} and up).
- BDD computations:
  - Efficient algorithms for needed operations.
  - BDD size is still exponential in worst case.
  - Highly sensitive (e.g. to variable ordering) and hard to optimize.
- Example: SMV/NuSMV (Carnegie Mellon/IRST)



Bounded Model Checking

- Symbolic model checking variant.
- Uses SAT (propositional satisfiability) rather than BDDs.
  - Idea: unroll transition relation a finite number of times into a (big) constraint network.
- Bounded-depth only, not complete.
- Polynomial space!
- Exponential time in the worst-case **but**
  modern SAT solvers are very efficient in most practical cases.
- Example: NuSMV (using the Chaff solver from Princeton)

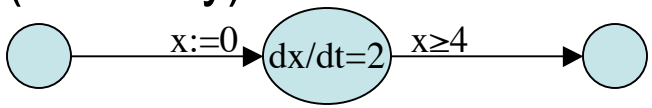
Real-Time and Hybrid

- "Classic" model checking: finite state, un-timed
- Real-time model checking: add clocks

  e.g. Khronos (Verimag), Uppaal (Uppsala/Aalborg)

- Hybrid model checking: add derivatives

  e.g. Hytech (Berkeley)



More complex problems & less mature tools

Contents

Model Checking for intelligent software

- Why?

  intelligent software, how to verify it?

- What?

  A bird's-eye view of model checking

- How?

  Experiences in the ASE Group at NASA Ames

Verification of Remote Agent Executive

(Lowry, Havelund and Penix)

- Smart executive system with AI features (Lisp)
- Modeled (1.5 month) and Model-checked with Spin (less than a week)
- 5 concurrency bugs found, that would have been hard to find through traditional testing

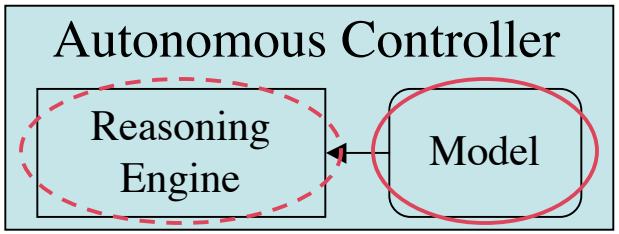
Hunting the RAX Bug

(Lowry, White, Havelund, Pecheur, ...)

- 18 May 1999: Remote Agent Experiment suspended following a deadlock in RA EXEC
  => Q: could V&V have found it?
- Over-the-week-end "clean room" experiment
- => A: V&V "found" it... two years ago!

  Similar to one of the 5 bugs found before (elsewhere)
  - Highly unlikely to occur
  - Never occurred during thorough testing
  - Occurred in flight!
- Morale: Testing not enough for concurrency bugs!

Verification of Model-Based Autonomy



Reasoning Engine

- Relatively small, generic algorithm => use prover
- Requires V&V expert level but once and for all
- At application level, assume correctness (cf. compiler)

Model

- Complex assembly of interacting components => model checking
- Avoid V&V experts => automated translation
  Not too hard because models are abstract

Reasoning Engine + Model ???

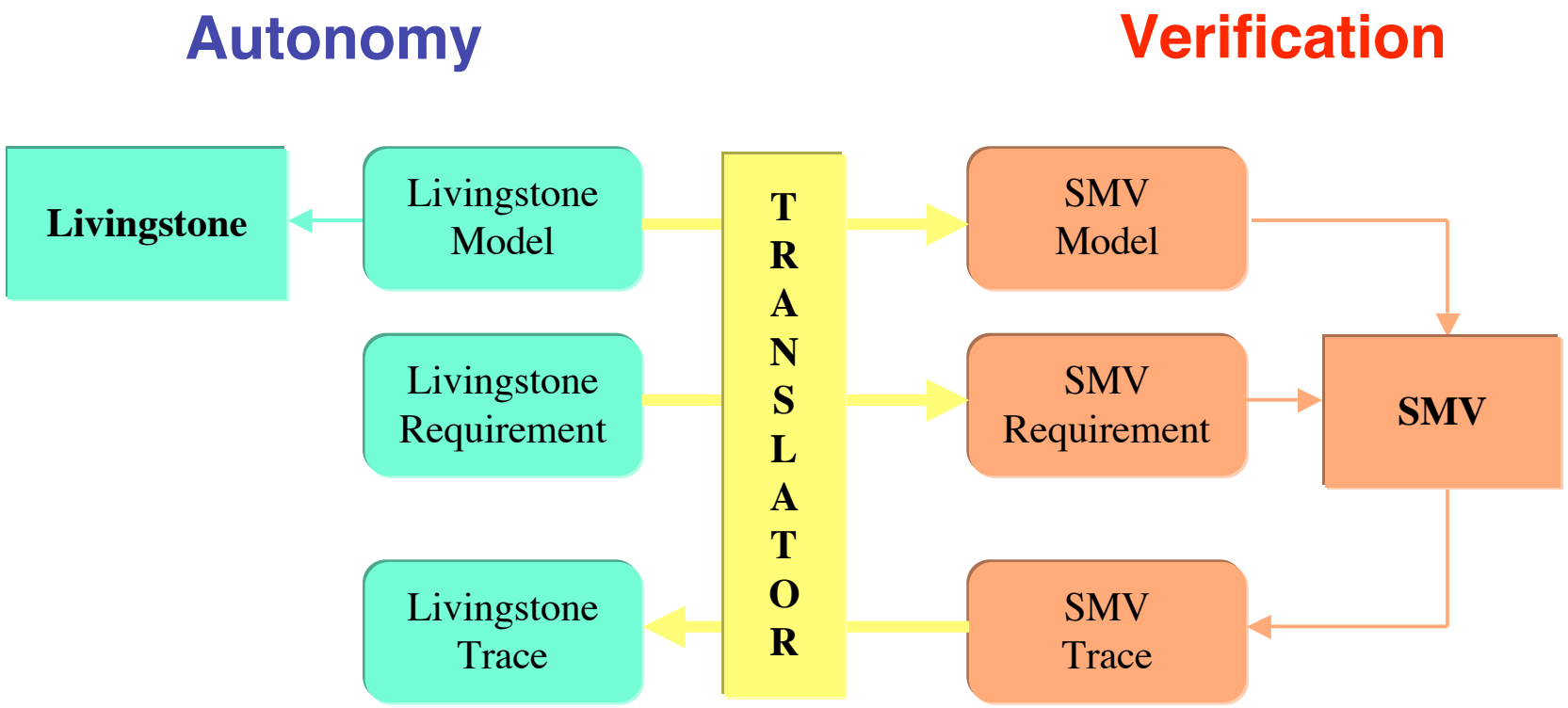
Verification of Planner/Scheduler Models

(Penix, Pecheur and Havelund)

- Model-based planner from Remote Agent
  - Models: constraint style, real-time
- Small sample model translated by hand
  - Subset of the full modeling language, untimed
- Compare 3 model checkers: Spin, Murphi, SMV
  => SMV much easier and faster (≈0.05s vs. ≈30s)
- Continuation *(Khatib)*: handle timed properties using real-time model checker (Uppaal)

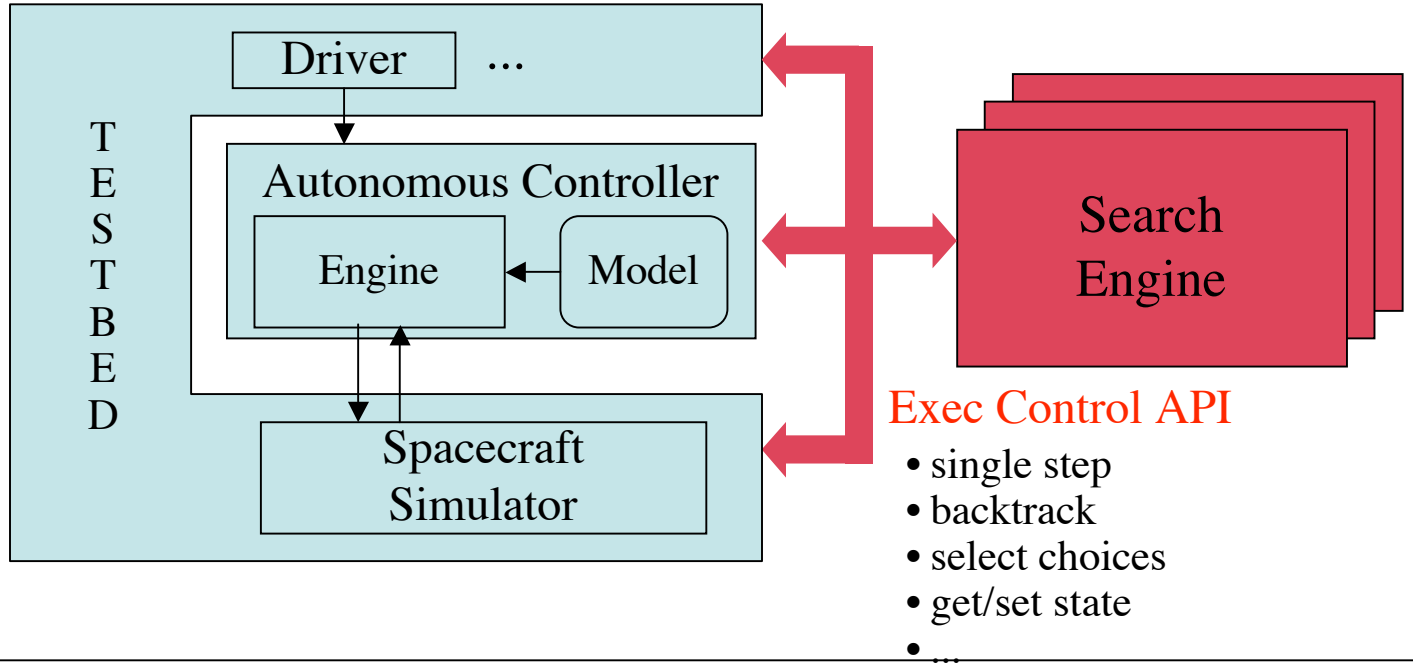
Verification of Livingstone Models

(Pecheur, Simmons)



Simulation-Based Verification Livingstone PathFinder (LPF)

(Pecheur, Lindsey)



- Real system => accuracy.
- More control => more coverage.
- For any discrete-event controller (not only model-based).

Model Checking Java
Java PathFinder (JPF)

(Havelund, Visser, ...)

- Java PathFinder 1: Translator to Promela (Spin)
- Java PathFinder 2: Based on custom Java VM.
  - Supports all Java bytecode.
  - Emphasis on efficient encoding of states (heap, GC).
  - Integrates static analysis for partial-order reduction, run-time analysis, abstraction, symbolic data, ...
  - Applied to DEOS avionics OS, planetary rover exec, MD-11 autopilot simulator...

Compositional verification

(Giannakopoulou, Pasareanu)

- Assume-guarantee reasoning on separate components of a system

=> prove properties of the whole from properties of the parts

- Automated extraction of assumptions

Runtime Analysis
Java Path Explorer (JPaX)

(Havelund, Rosu)

- An observer analyses an event stream from an instrumented program to detect anomalies.
- Analyze temporal logic properties, hazardous concurrency patterns (lock ordering, data races).
- On actual program runs => limited but highly scalable.

Conclusions

Model checking:

- Needed for automated/autonomous systems testing is not enough
- General pros&cons apply:
  - exhaustive... if model is small enough
  - automatic verification... but tough modeling
- Works nicely on autonomy models
- Solutions inbetween testing and model checking
- Not short of tough problems:
  - Real-time, hybrid, AI
  - Learning/adaptive systems: *after* training/*including* training

Pointers

- Pecheur's home page

  http://ase.arc.nasa.gov/pecheur
  http://ase.arc.nasa.gov/pecheur/publi.html
  http://ase.arc.nasa.gov/pecheur/talks.html

- JavaPathFinder

  http://ase.arc.nasa.gov/visser/jpf

- ASE group at NASA Ames

  http://ase.arc.nasa.gov