



SAPIENZA
UNIVERSITÀ DI ROMA

Simulation Based Formal Verification of Onboard Software — A Case Study —

SyLVer: System Level Verifier

Toni Mancini, Annalisa Massini, **Federico Mari**, Igor Melatti,
Ivano Salvo, Enrico Tronci

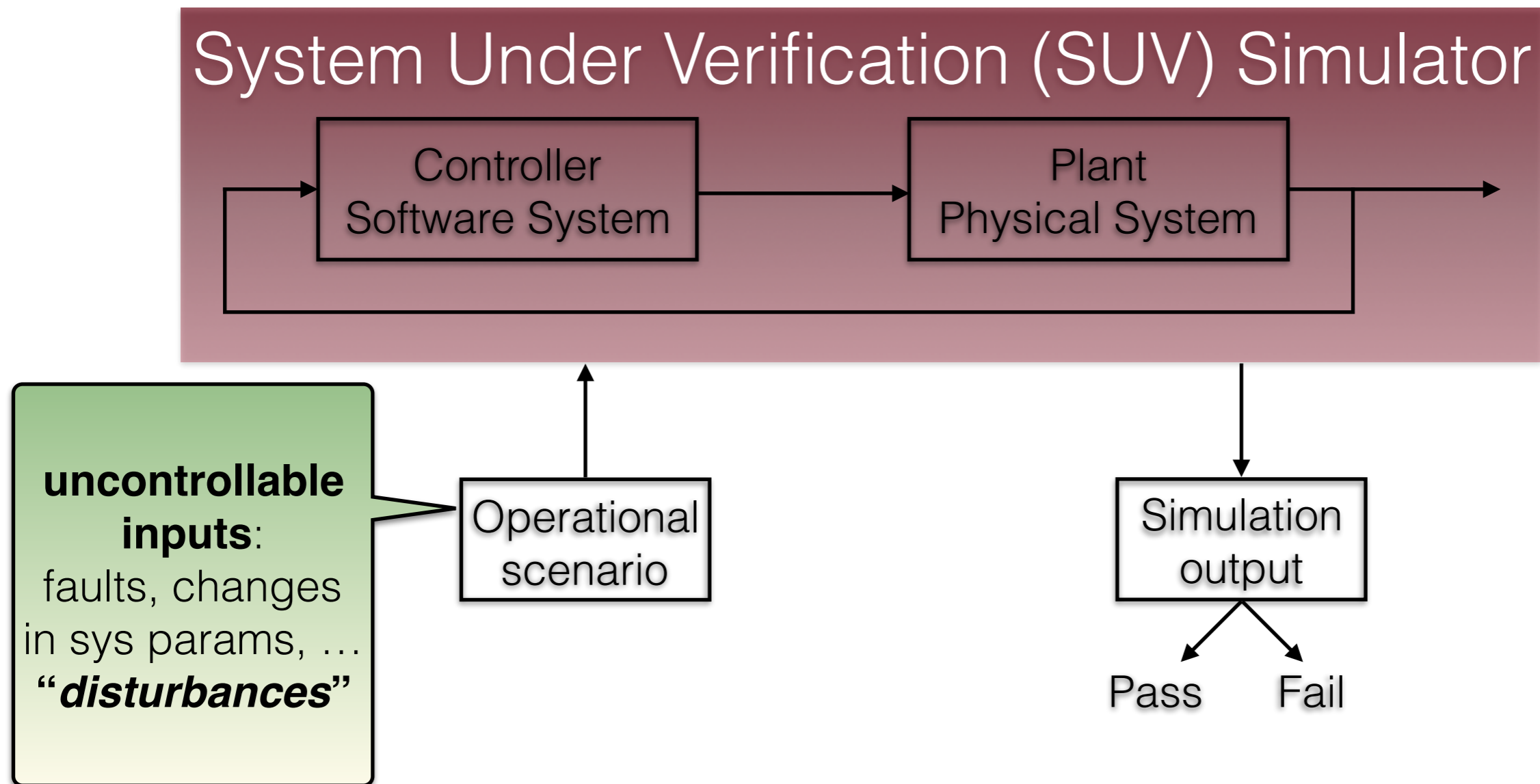
Computer Science Department
Sapienza University of Rome, Italy
<http://mclab.di.uniroma1.it>

System Level Verification of CPSs

- **Cyber Physical System (CPS):** hw + sw components
⇒ Can be modelled as **Hybrid System**
- **System Level Verification (SLV):** to verify that the **whole** system (hw+sw) satisfies given specifications
- CPSs of industrial relevance **too complex** for SLV to be performed by model checkers for Hybrid Systems
- **Main workhorse for SLV:** Hardware in The Loop Simulation (HILS)

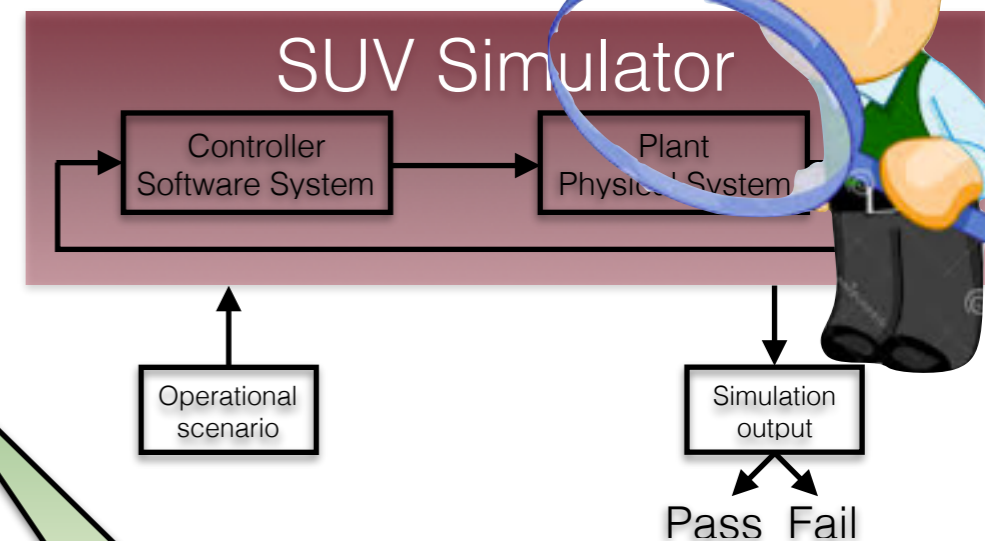
Hardware in The Loop Simulation

- Hardware in The Loop Simulation (**HILS**):
replace hardware with a software simulator
- Supported by Model Based Design Tools as Simulink, VisSim, ...



HILS Campaign: Main Obstacles

- **Effort** needed to define the **operational scenarios** defining disturbances to be injected into the system under verification.
- **Computation time** needed to carry out the simulation campaign itself.
- **Degree of assurance** achieved at the end of the HILS campaign: did we consider **all** relevant operational scenarios?
- **Graceful degradation**: what can we say about the error probability **during** the HILS campaign?



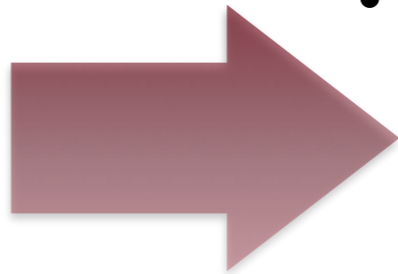
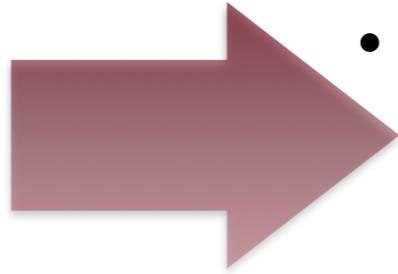
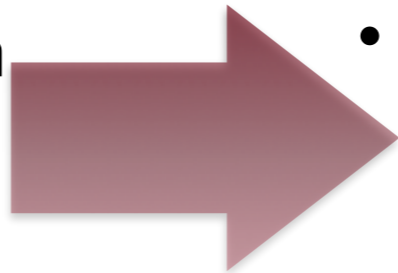
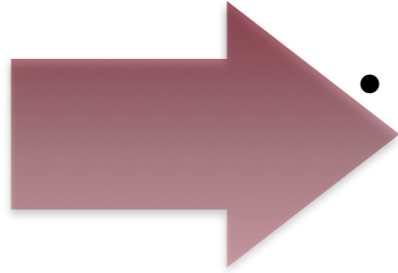
Hard to be done **manually**

Can take **weeks!**

“Did I overlook anything?”

“What can I say if I **abort** verification **now?**”

Our approach to System Level Formal Verification

- **Effort** needed to define the **operational scenarios** defining disturbances to be injected into the system under verification. 
- **Degree of assurance**: did we consider **all** relevant operational scenarios? 
- **Graceful degradation**: what can we say about the error probability **during** the HILS campaign? 
- **Computation time** needed to carry out the simulation campaign itself. 
- **Formal** model of operational scenarios (**disturbance model**) as a FSA described in a high-level language (CMurphi)
- **Exhaustive** system level verification wrt operational scenarios defined by the model
- **Anytime random algorithm**: at any time we compute an **upper bound** to **Omission Probability**
- **Embarrassing parallel multi-core** approach to speed up simulation + optimisation

[CAV13, PDP14, DSD14, PDP15, Microprocessors & Microsystems 2016, Fundamenta Informaticae 2016]

Model-Based System Verification @ MCLab

Disturbance Model (formal model of operational scenarios)

SyLVer System Level Formal Verifier

<https://bitbucket.org/mclab/sylver-simulink-driver>

LOAD - RUN - FREE -STORE

Optimised
Simulation
Campaign

Simulator
Driver

CPS
Model

Monitor

Omission Probability

Monitor output
fail
1
0
pass

Parallel (cluster)

Optimised
Simulation
Campaign

Simulator
Driver

CPS
Model

Monitor

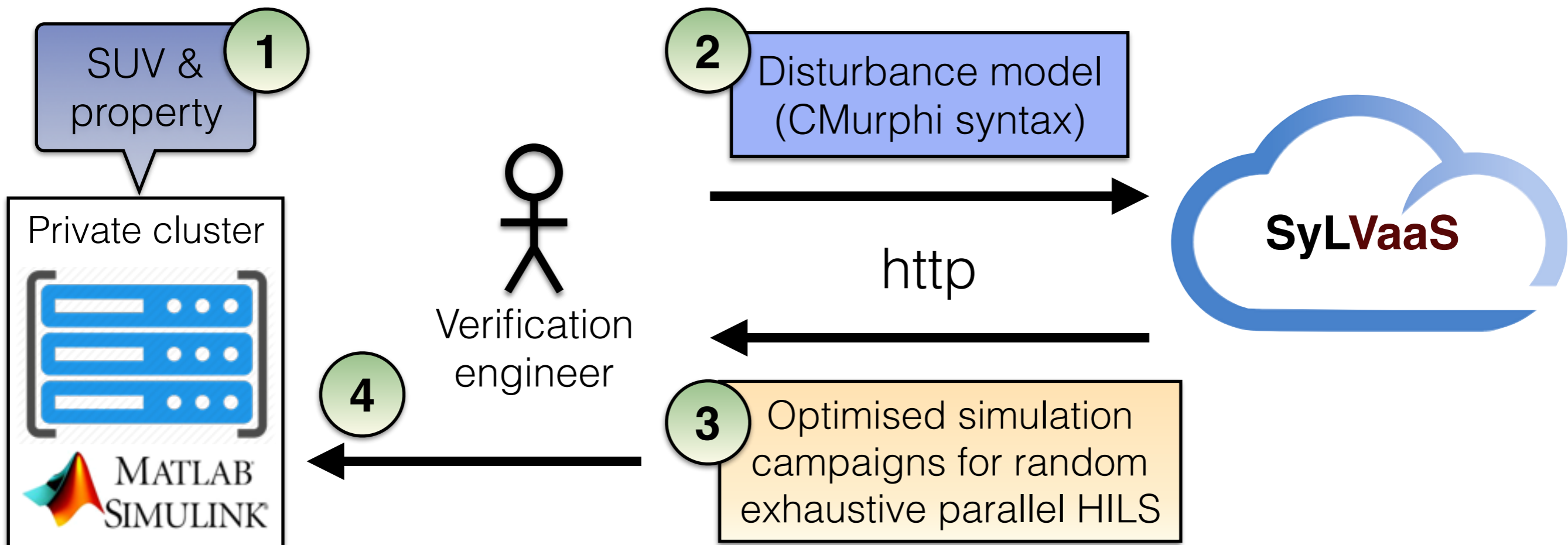
Omission Probability

Monitor output
fail
1
0
pass

Hardware-in-the-Loop Simulation (HILS)

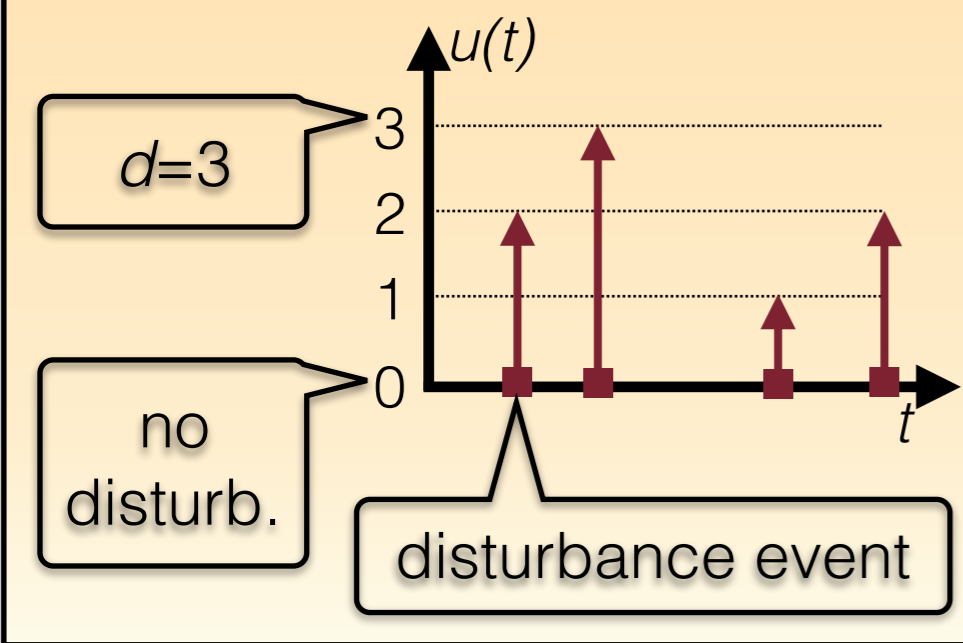
SyLVaaS

- Introduces **Verification as a Service paradigm**
- Supports companies in the CPS design business in their daily verification activities
- Allows keeping both the SUV model and the property to be verified secret (**Intellectual Property protection**)



Modelling the Operational Environment

Discrete event sequence $u(t)$



SUV input: discrete event seq.

- Associates to each (real) t a disturbance event within $[0, d]$
- Differs from 0 (no disturbance) in a finite number of time-points

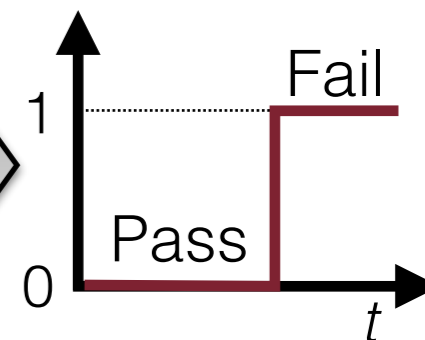
...no system can withstand an infinite number of disturbances within a finite time

Property to be verified:

embedded in a continuous-time SUV monitor

SUV: continuous-time input-state-output deterministic dynamical system

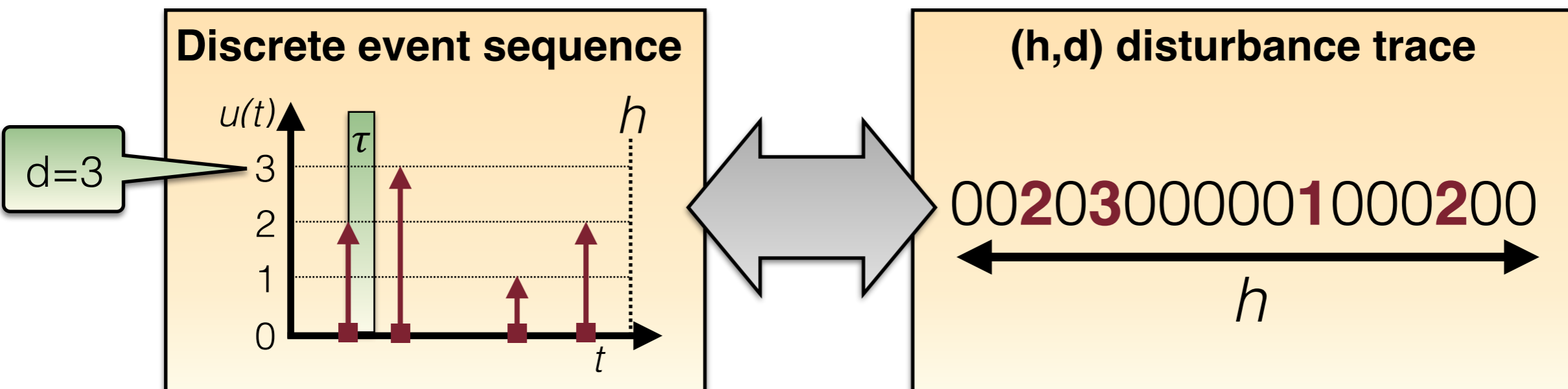
SUV output: 0 at start; goes to and stays 1 as soon as error is detected



Discrete Event Seq's & Disturbance Traces

We aim at **Bounded System Level Formal Verification**:

- Bounded **time horizon**: h
- Bounded **time quantum** between disturbances: τ



Disturbance Model

- Defining all disturbance sequences the SUV should withstand cannot be done manually for large CPSs
- Approach: use high-level modelling language to define disturbance model as a **Finite State Automaton**

A tiny example

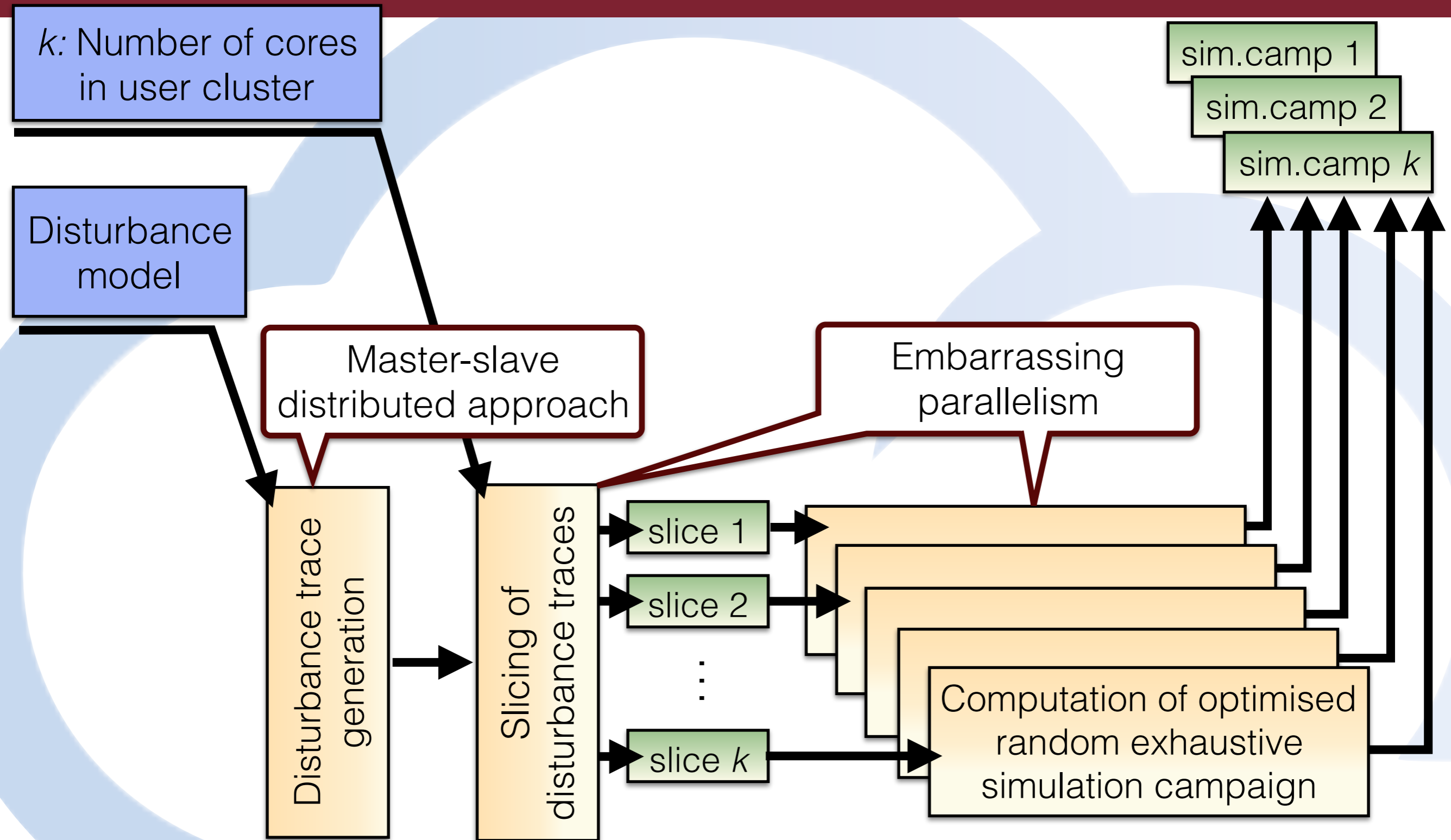
- Just one disturbance (fault), always recovered within 4 seconds
- At least 5 seconds between two consecutive disturbances
- Time quantum $\tau = 1$ second
- Time horizon $h = 6$ seconds

```
function disturbanceModel(h)
  c ← 0; /* counter */
  t ← 0; /* time */
  while t ≤ h do
    d ← read(); t ← t + 1;
    if c > 0 then c ← c - 1;
    if d = 1 then
      if c > 0 then return ⊗;
      else c ← 4;
  return ✓;
end
```

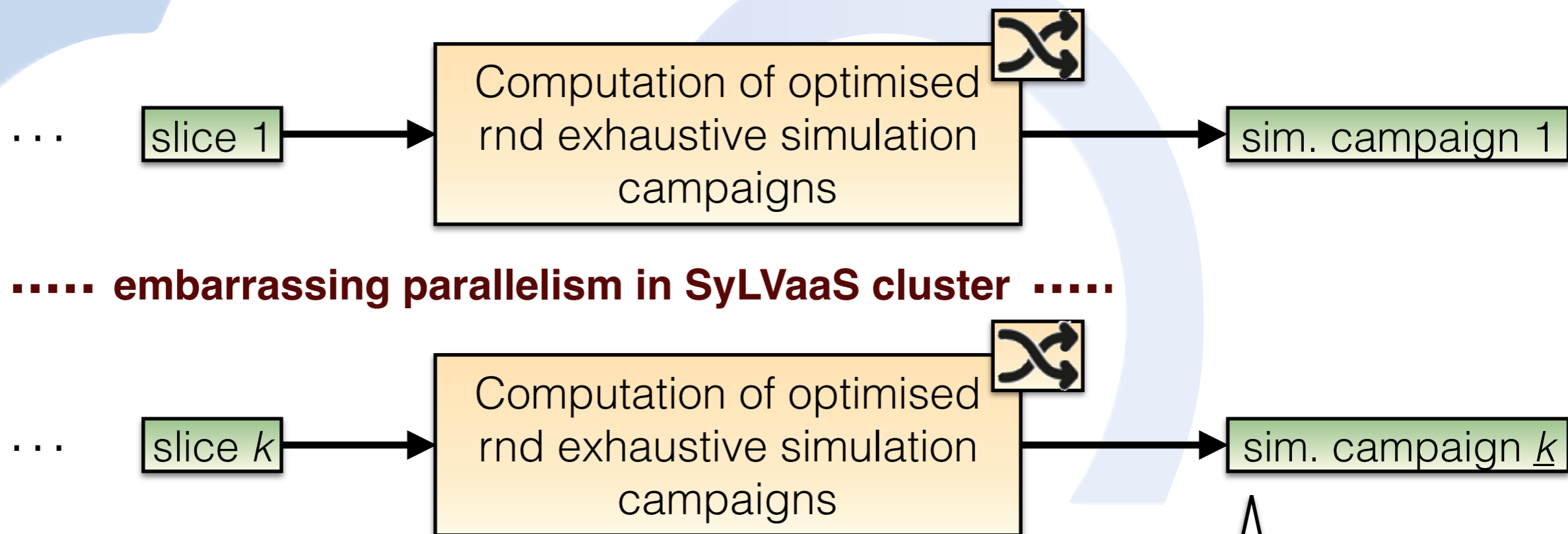
FSA recognising **admissible disturbance traces**
(we actually use the rich language of the
CMurphi model checker)

000000 ✓ 010000 ✓ ... overall 8 adm
000001 ✓ 010001 ⊗
000010 ✓ 01001 ⊗ disturbance traces

SyLVaaS Workflow



Optimised Rnd Exhaustive Sim. Campaigns

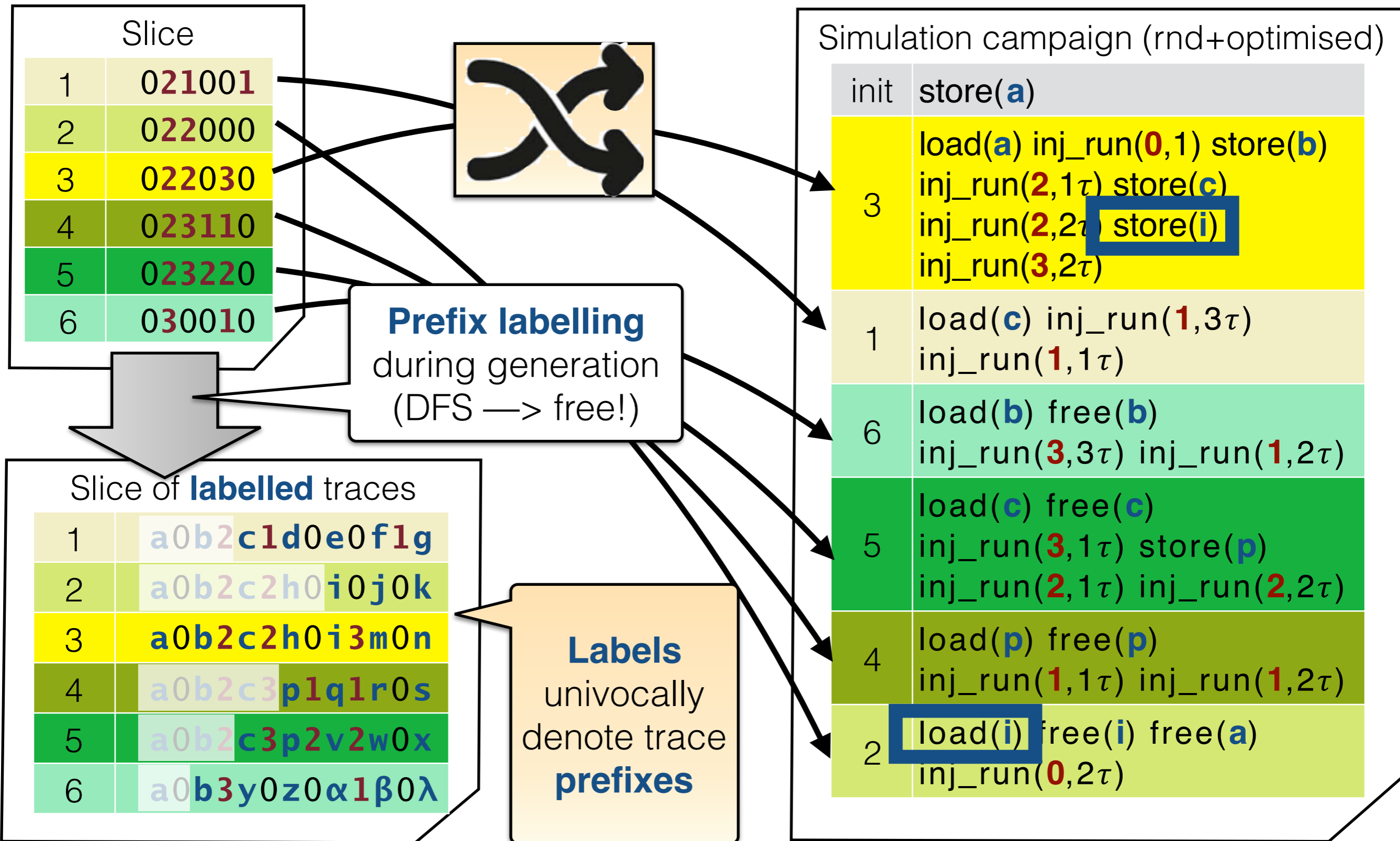


- **Optimisation:** use of load/store commands avoids revisiting previously visited simulation states as much as possible
- **Exhaustiveness:** all disturbance traces in input slice are verified
- **Randomness:** trace verification order is randomised

Sequence of simulator commands:

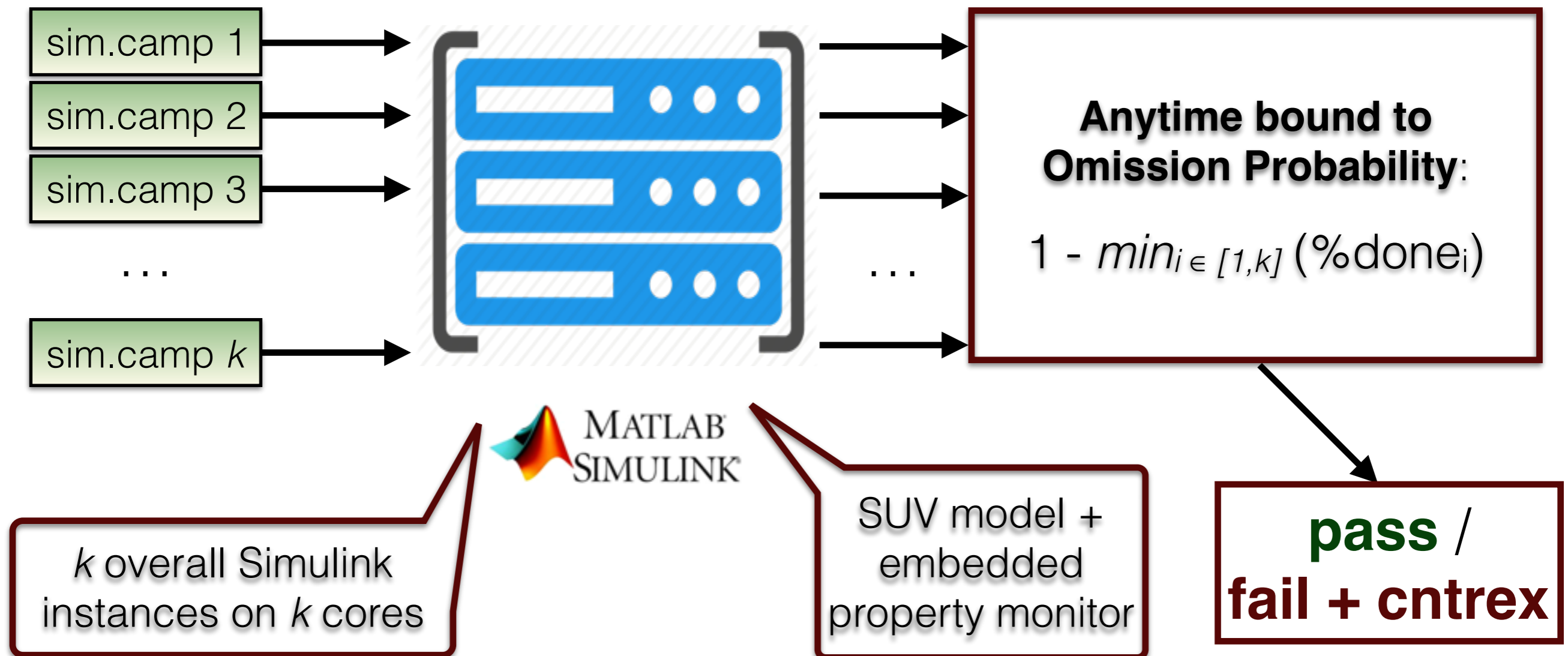
- **inj_run(e, t):** inject disturbance and advance simulation
- **store(l):** store current sim. state into mass memory
- **load(l):** set current sim. state from previously stored state
- **free(l):** free stored sim. state stored

Optimised Rnd Exhaustive Sim. Campaigns



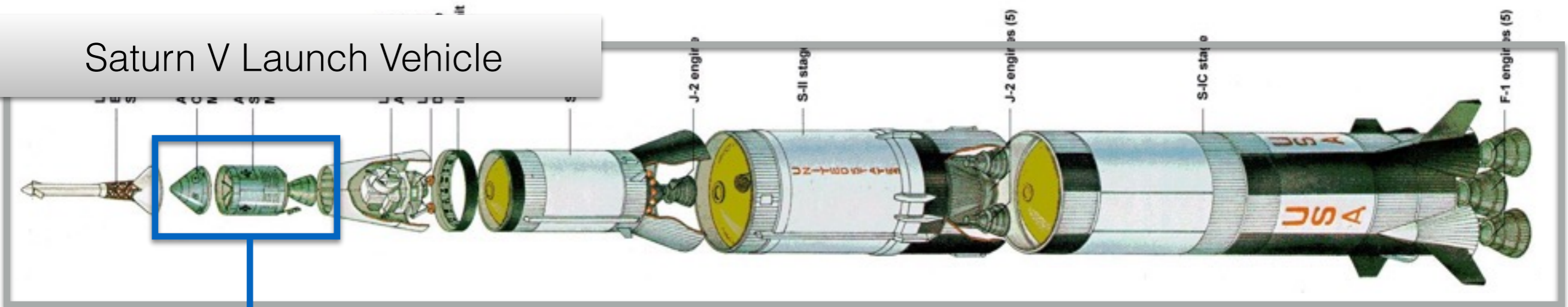
Embarrassingly Parallel Simulation

Simulation carried out on **user private cluster** (Intellectual Property protection)

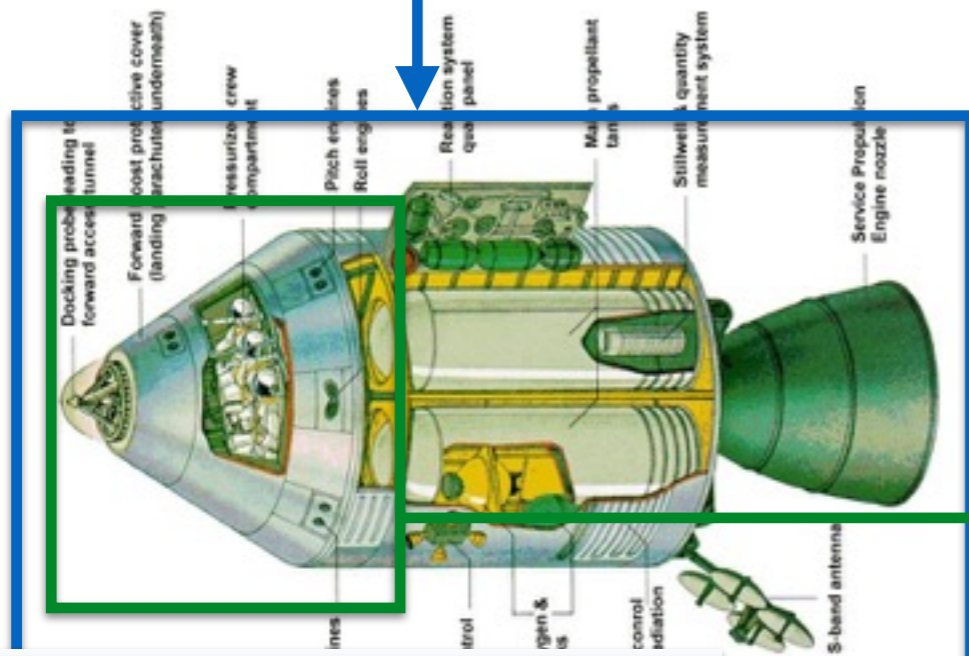


A Case Study: Apollo

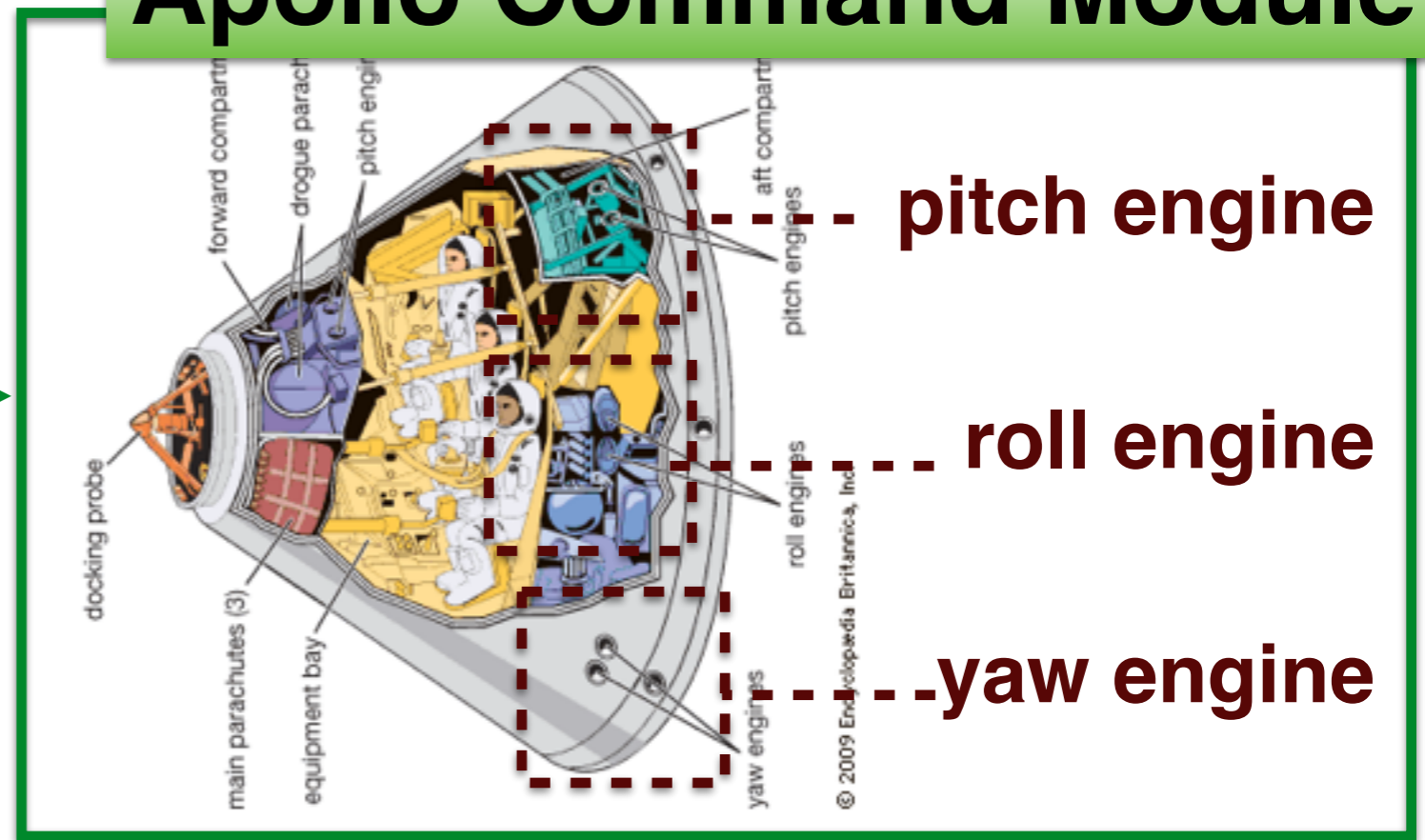
Saturn V Launch Vehicle



Apollo Command Module



Apollo Command and Service Modules



pitch engine

roll engine

yaw engine

© 2009 Encyclopædia Britannica, Inc.

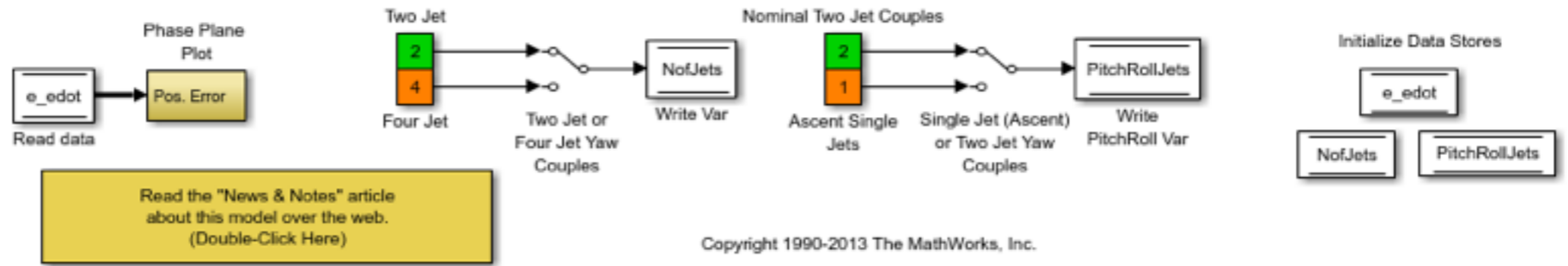
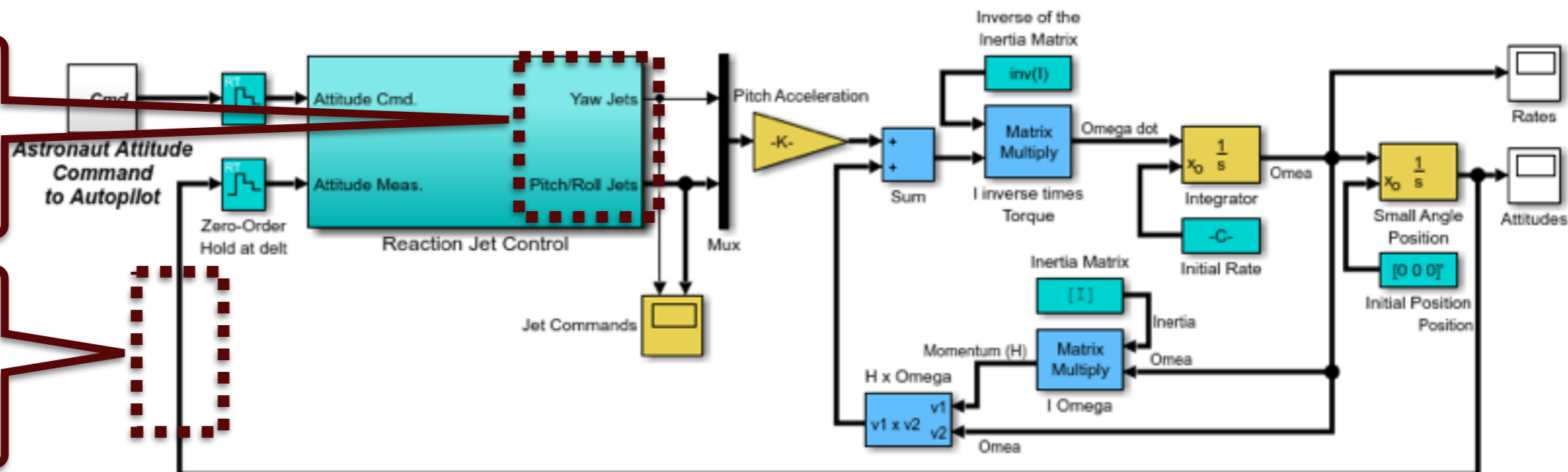
A Case Study: Apollo

The Lunar Module Digital Autopilot Design

How it Would be Done Today!

Yaw, Pitch and Roll Jets

Three signals: Yaw, Pitch and Roll sensors



Safety: Yaw, Pitch and Roll close to 0



Experiments: Disturbance Models

Disturbance model:

- sensor faults repaired after 1 second
- 5 different sensor faults possible
- at most 3 faults in each trace
- at most 1 fault active at any time
- $h = 10 \text{ sec}$, $\tau = 500\text{ms}$

—> **8.9M dist. traces**

Disturbance model CMurphi encoding

```
Ruleset d : FAULT_TYPE do
  Rule "Inject Fault"
    time_since_last_fault[d] = -1 &
    no_fault_needs_repair() &
    num_faults < MAX_NUMFAULTS &
    num_active_faults() < MAX_NUMACTIVE_FAULTS
  ==> begin
    time_since_last_fault[d] := 0;
    num_faults := num_faults+1;
    time_step();
  end;

  Rule "Repair Fault"
    time_since_last_fault[d] = FAULT_DURATION
  ==> begin — repair fault d
    time_since_last_fault[d] := -1;
    time_step();
  end;
End;
```

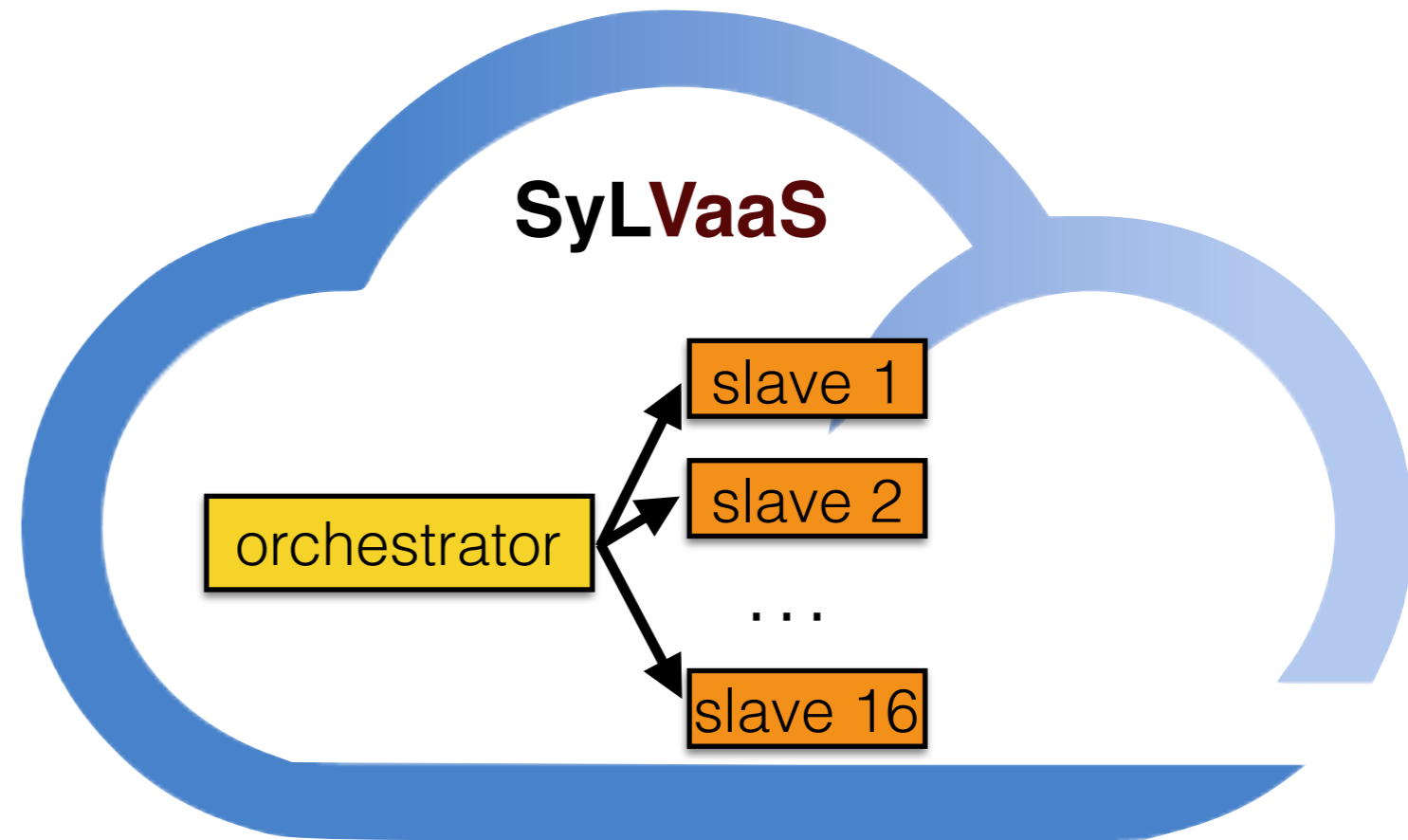
```
Ruleset d : INPUT_TYPE do
  Rule "Inject Input Variation"
    no_fault_needs_repair() &
    num_inputs < MAX_NUMINPUTS &
    is_input_variation_allowed()
  ==> begin
    num_inputs := num_inputs + 1;
    time_step();
  end;
End;

Rule "No Disturbance"
  no_fault_needs_repair() ==>
  begin time_step(); end;
...
Finalstate "Correct Length"
  no_faults() & num_faults <= MAX_NUMFAULTS &
  num_inputs <= MAX_NUMINPUTS;
```

Experiments: Infrastructure

SyLVaaS infrastructure:

- 1 orchestrator
- 16 slaves



User private cluster:

- 8 to 64 8-core machines
—> **up to 512 Simulink parallel instances**



Experiments (1)

SyLVaaS:

Parallel computation of **random exhaustive optimised simulation campaigns** (16 cores):

$k = \#cores$ in user cluster	Computation of simulation campaigns
128	0:22:18
256	0:22:18
512	0:24:30

h:m:s



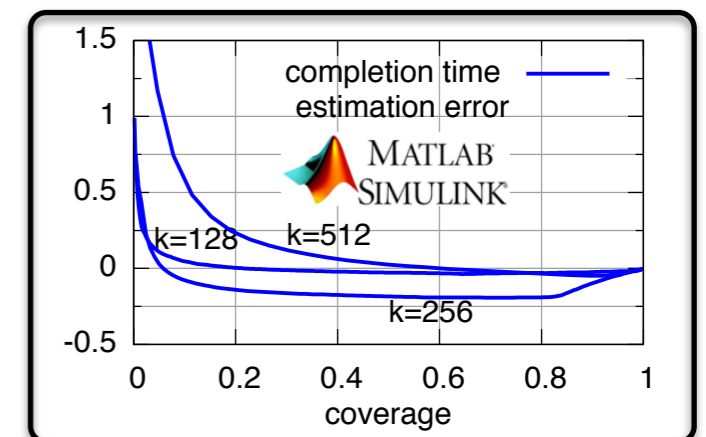
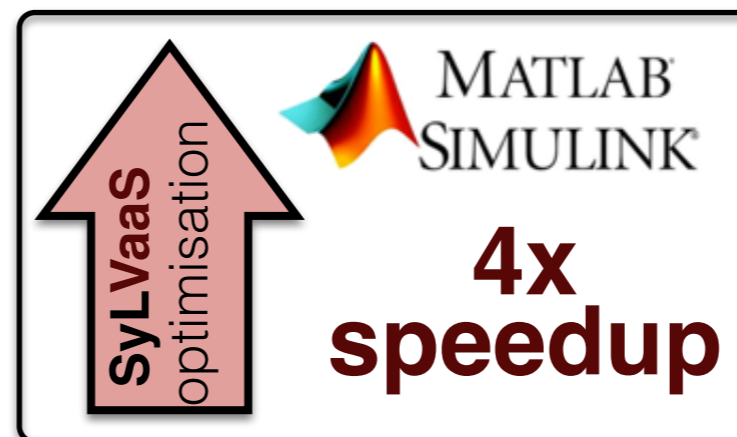
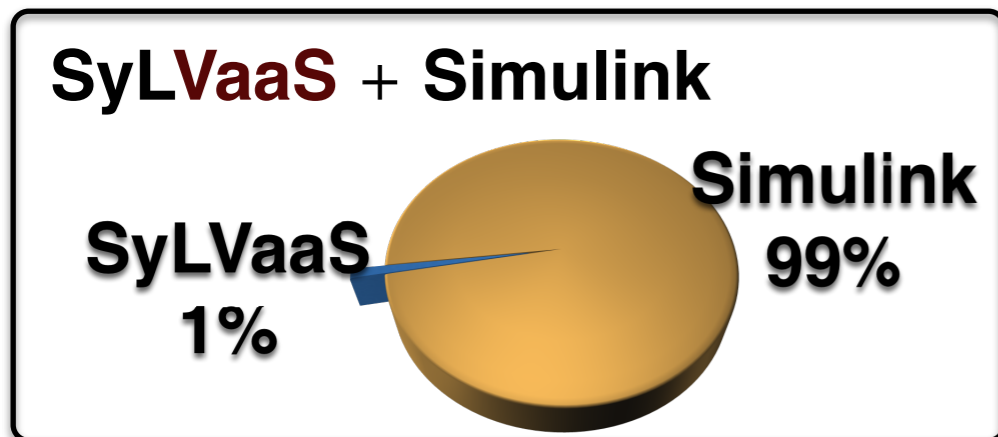
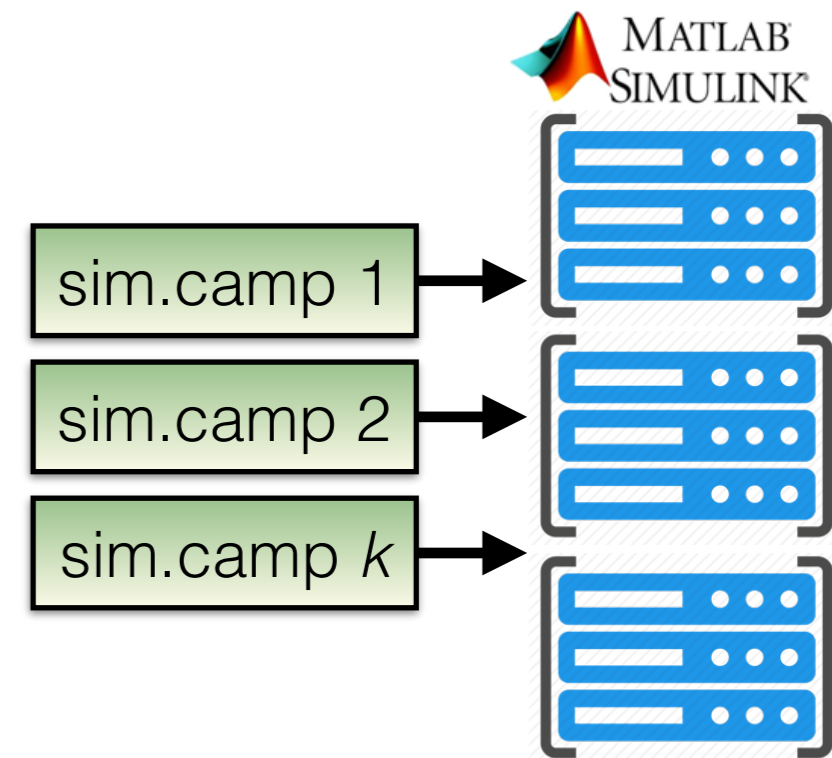
Experiments (2)

Private user cluster:

Formal verification via **embarrassing parallel execution of simulation campaigns** ($k=128, 256, 512$ parallel Simulink instances):

$k = \#cores$ in user cluster	Execution of simulation campaigns
128	726:53:25
256	121:06:28
512	44:26:37

h:m:s

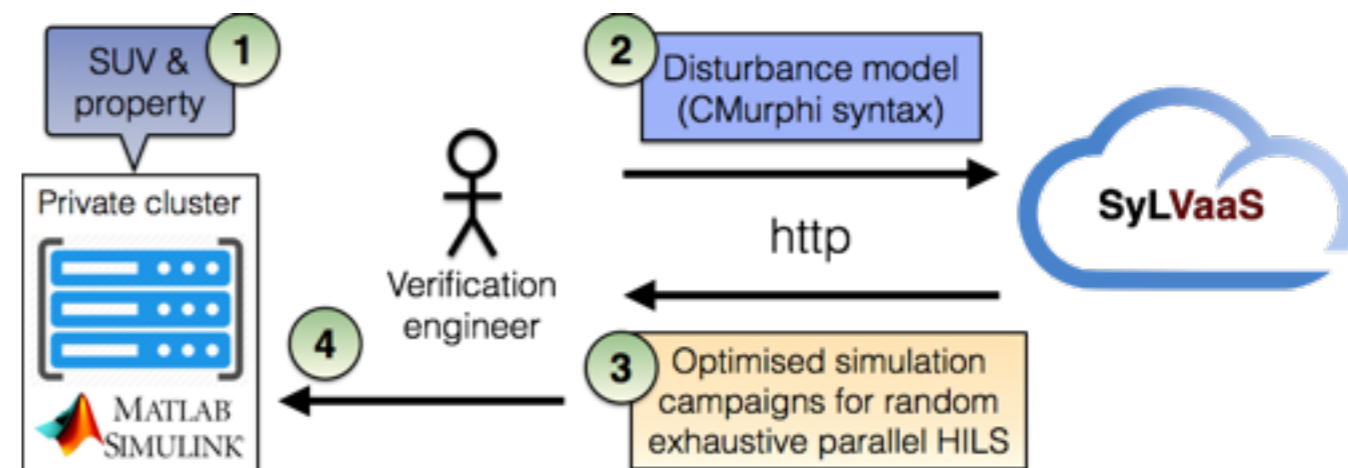


Conclusions

SyLVer: System Level Verifier

SyLVaaS: SyLVer as a Service

- Given formal model of **operational environment**
- Efficiently computes **random exhaustive simulation campaigns**
- **Approach scales well**: additional experiments with dist. models yielding **40M traces**
- Campaigns run **embarrassingly in parallel** on all Simulink instances available on private user cluster
- Campaigns **optimise** simulation activities (**4x speedups**) by storing/restoring intermediate simulation states as much as possible (depending on available mass memory space on user cluster)
- **Graceful degradation**: omission probability bound available anytime during verification
- **Completion time estimation** available anytime during verification
- Both SUV model and property to be verified kept secret (**Intellectual Property protection**)





Thank you!

Simulation Based Formal Verification of
Cyber-physical Systems

SyLVaaS: System Level Verification as a Service

Toni Mancini, Annalisa Massini, Federico Mari, Igor Melatti,
Ivano Salvo, **Enrico Tronci**

Computer Science Department
Sapienza University of Rome, Italy
<http://mclab.di.uniroma1.it>