SAPIENZA
UNIVERSITÀ DI ROMA

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

System Level Verification of CPSs

- **Cyber Physical System (CPS):** hw + sw components

  $\Longrightarrow$ Can be modelled as **Hybrid System**

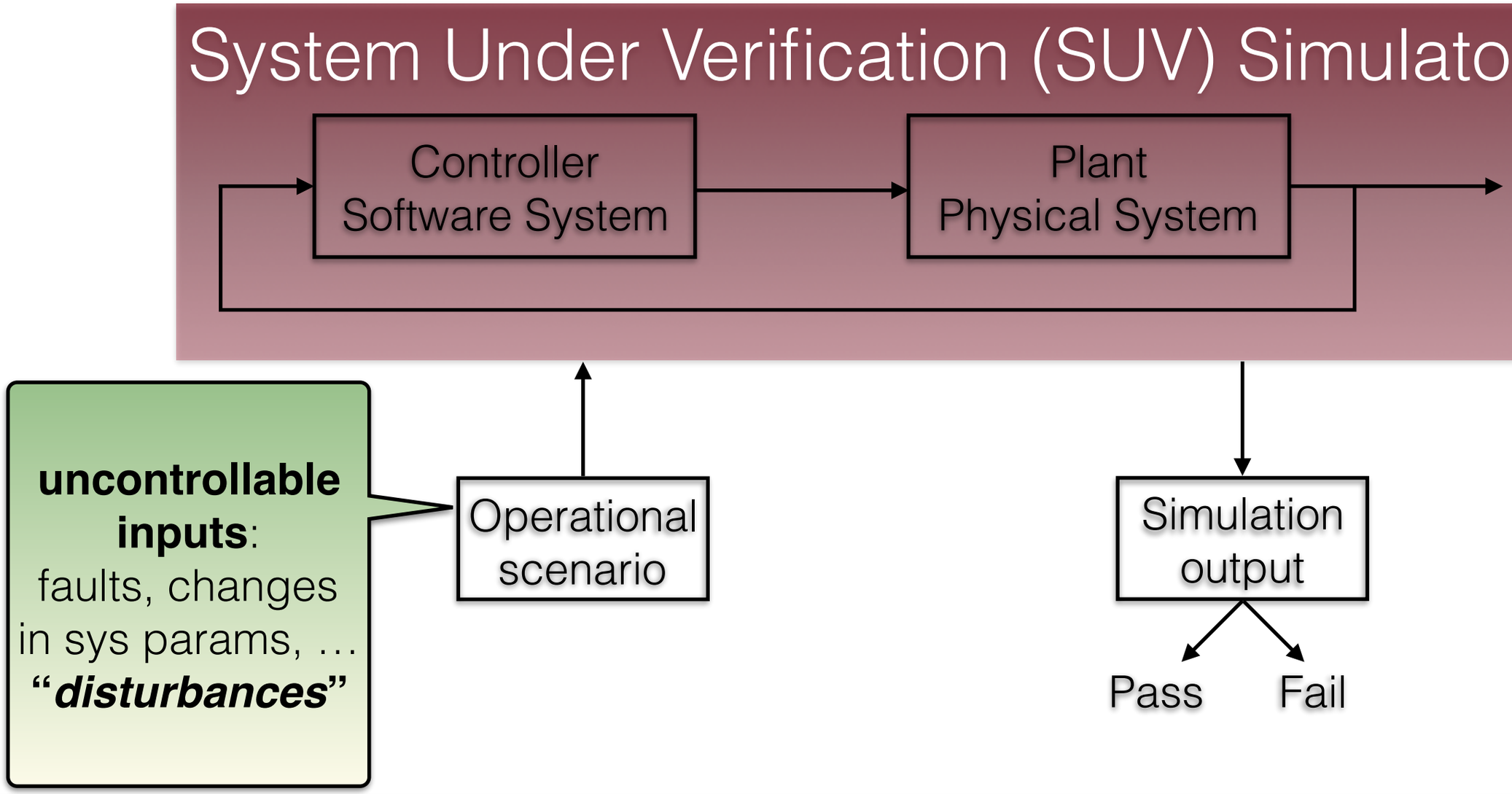
- **System Level Verification (SLV):** to verify that the **whole** system (hw+sw) satisfies given specifications

- 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

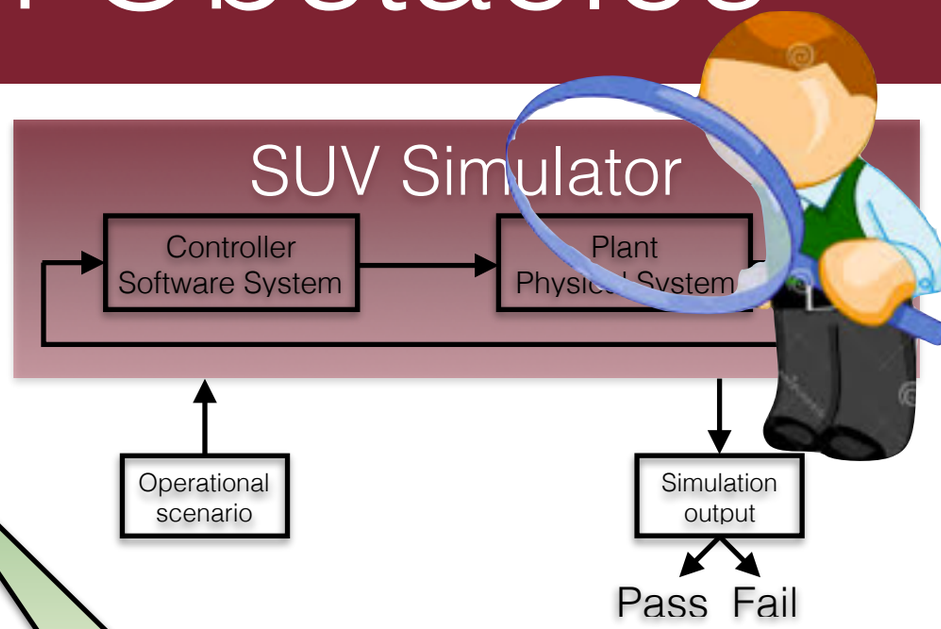
Hardware in The Loop Simulation

- 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

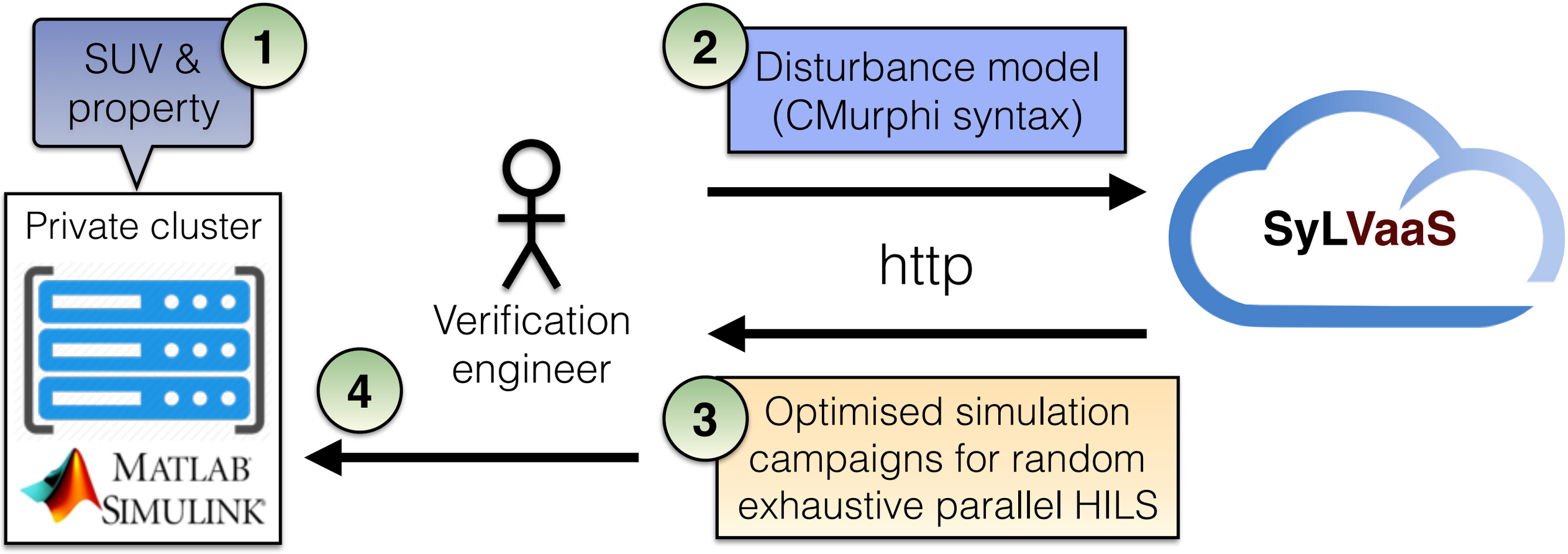
- **Effort** needed to define the **operational scenarios** defining disturbances to be injected into the system under verification.
- **Degree of assurance**: did we consider **all** relevant operational scenarios?
- **Graceful degradation**: what can we say about the error probability **during** the 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]

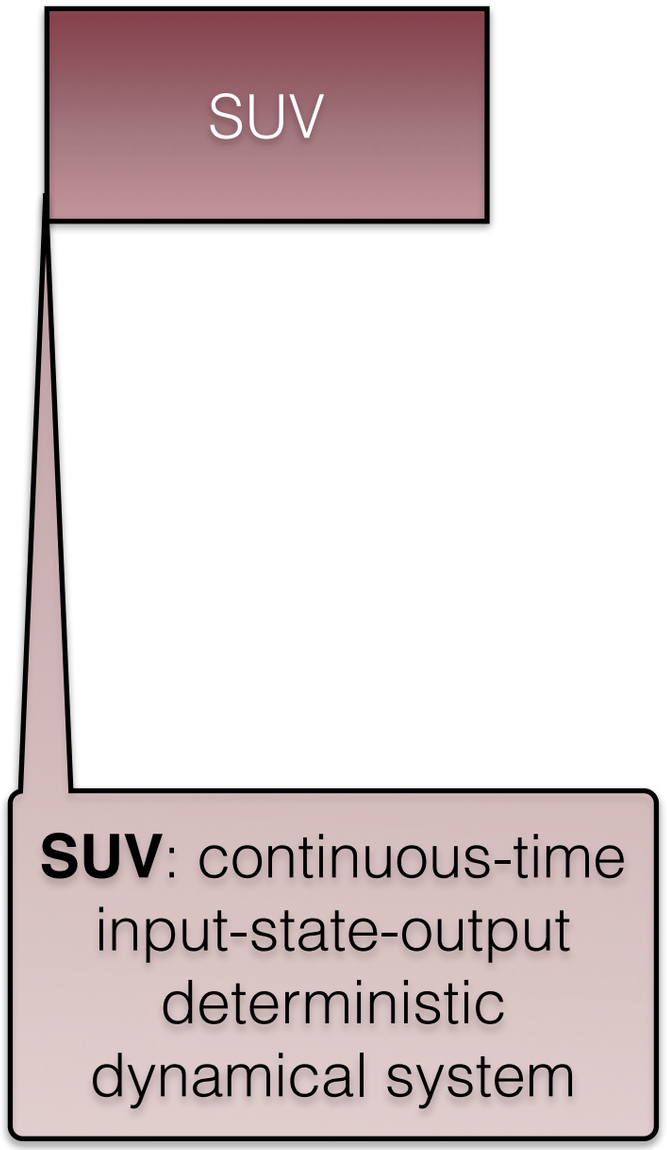
SyLVaaS

- 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**)

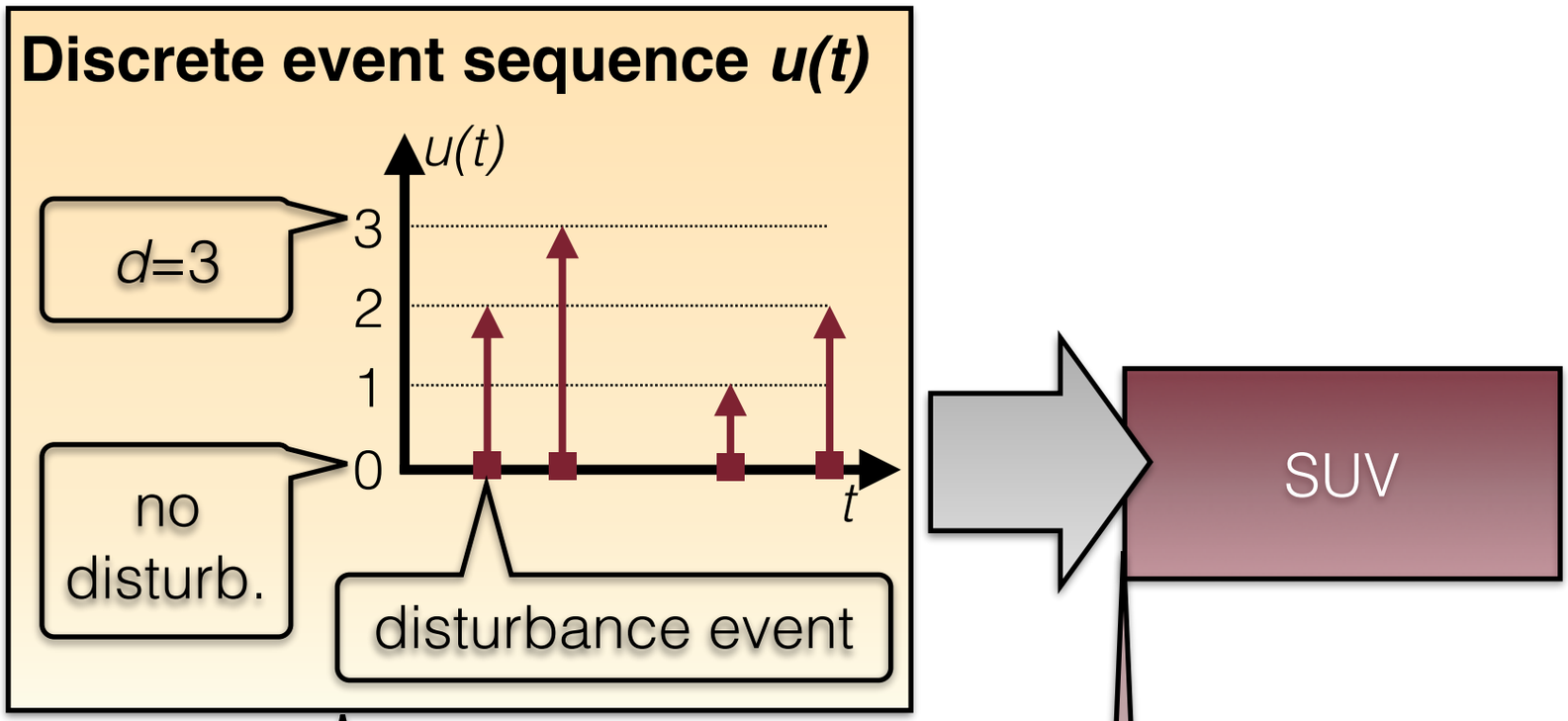


Modelling the Operational Environment

Modelling the Operational Environment



Modelling the Operational Environment



SUV input: discrete event seq.

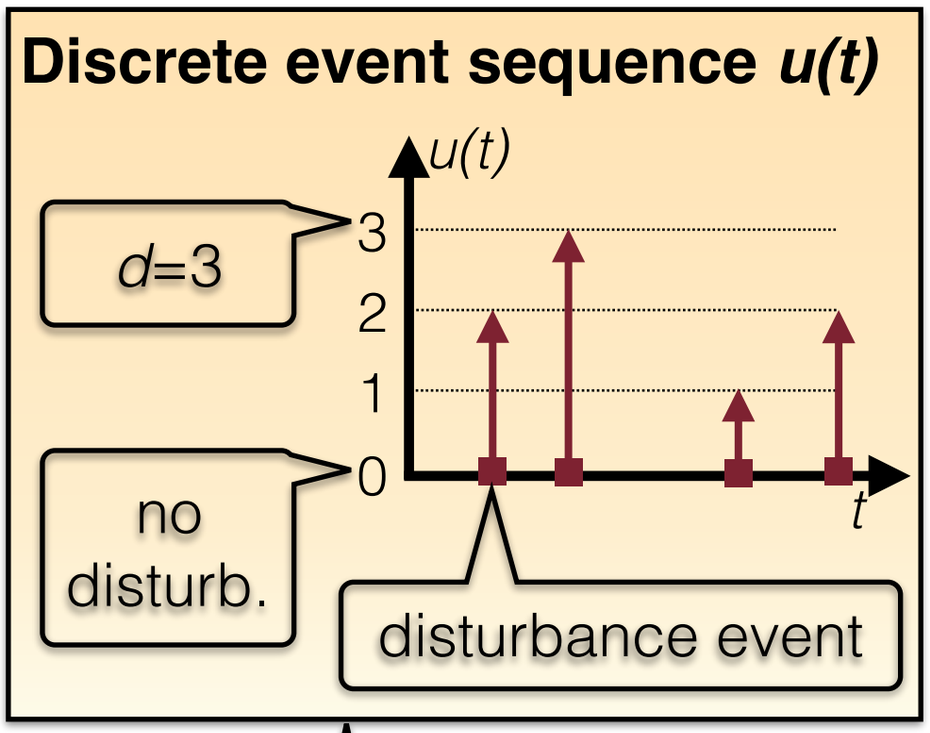
- *A*ssociates to each (real) *t* a disturbance event within [0,*d*]
- Differs from 0 (no disturbance) in a finite number of time-points

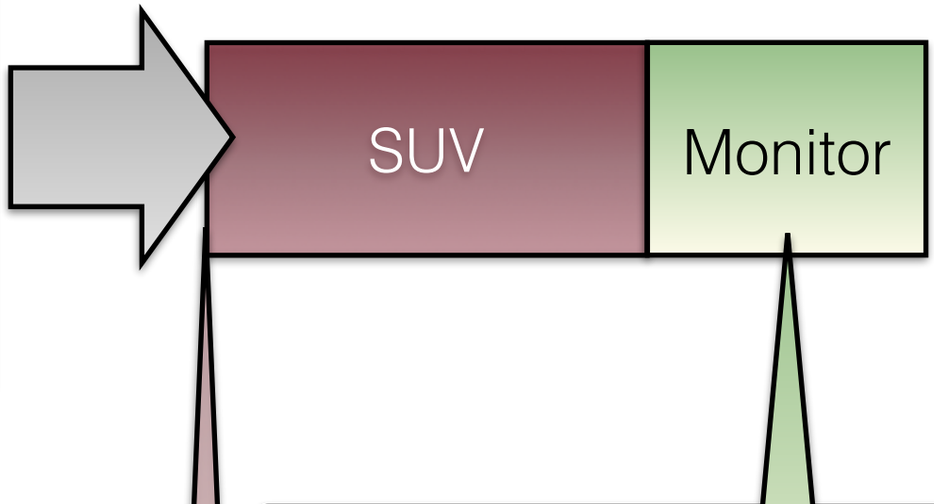
…no system can withstand an infinite number of disturbances within a finite time

SUV

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

Modelling the Operational Environment





SUV input: discrete event seq.

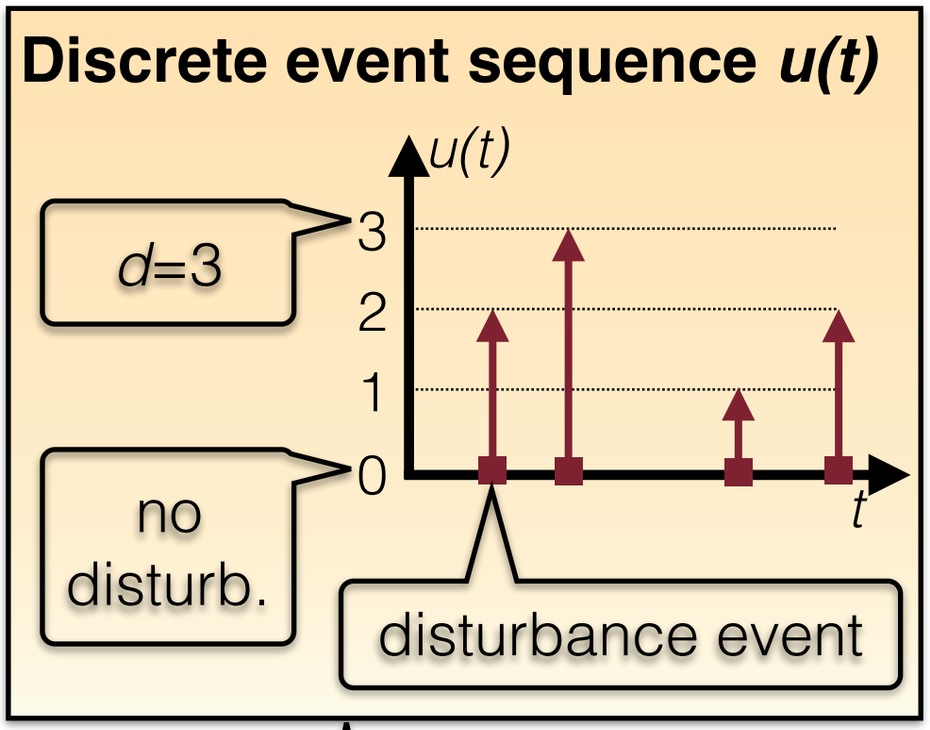
- *A*ssociates to each (real) *t* a disturbance event within [0,*d*]
- Differs from 0 (no disturbance) in a finite number of time-points

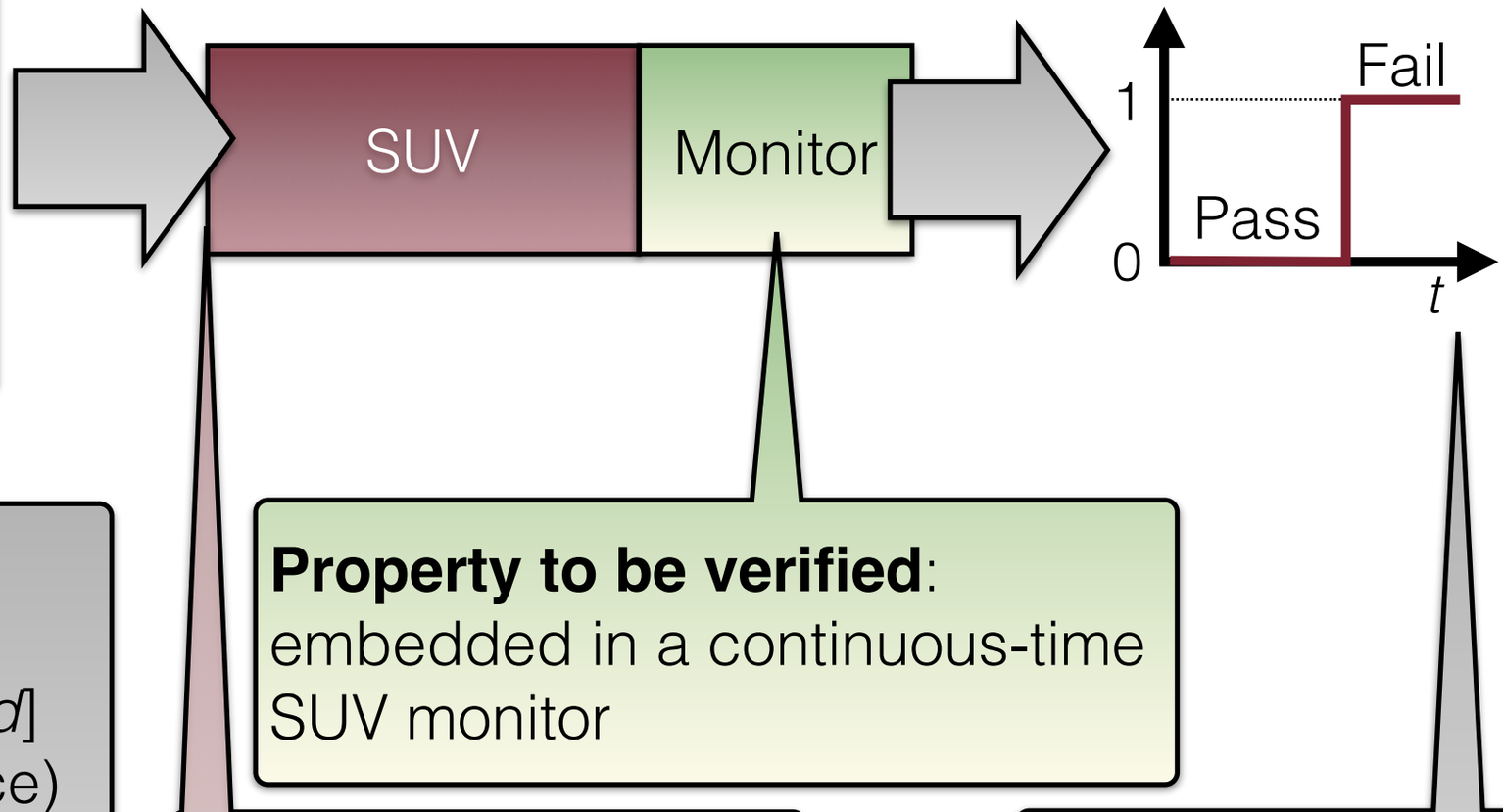
…no system can withstand an infinite number of disturbances within a finite time

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

Property to be verified: embedded in a continuous-time SUV monitor

Modelling the Operational Environment





SUV input: discrete event seq.

- *A*ssociates 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

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

Property to be verified: embedded in a continuous-time SUV monitor

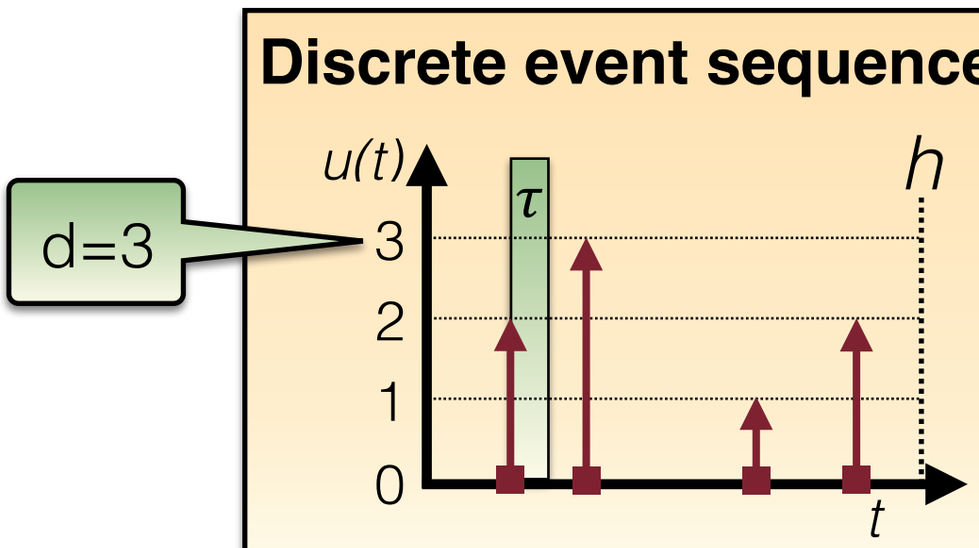
SUV output: 0 at start; goes to and stays 1 as soon as error is detected

Discrete Event Seq's & Disturbance Traces

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

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

Discrete event sequence



(h,d) disturbance trace

00**2**0**3**0000**1**000**2**00

h

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 τ = 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✓
000001✓ 010001⊗
000010✓ 01001⊗

… overall 8 adm disturbance traces

SyLVaaS Workflow

k: Number of cores in user cluster

Disturbance model

Disturbance trace generation

Slicing of disturbance traces

slice 1

slice 2

⋮

slice *k*

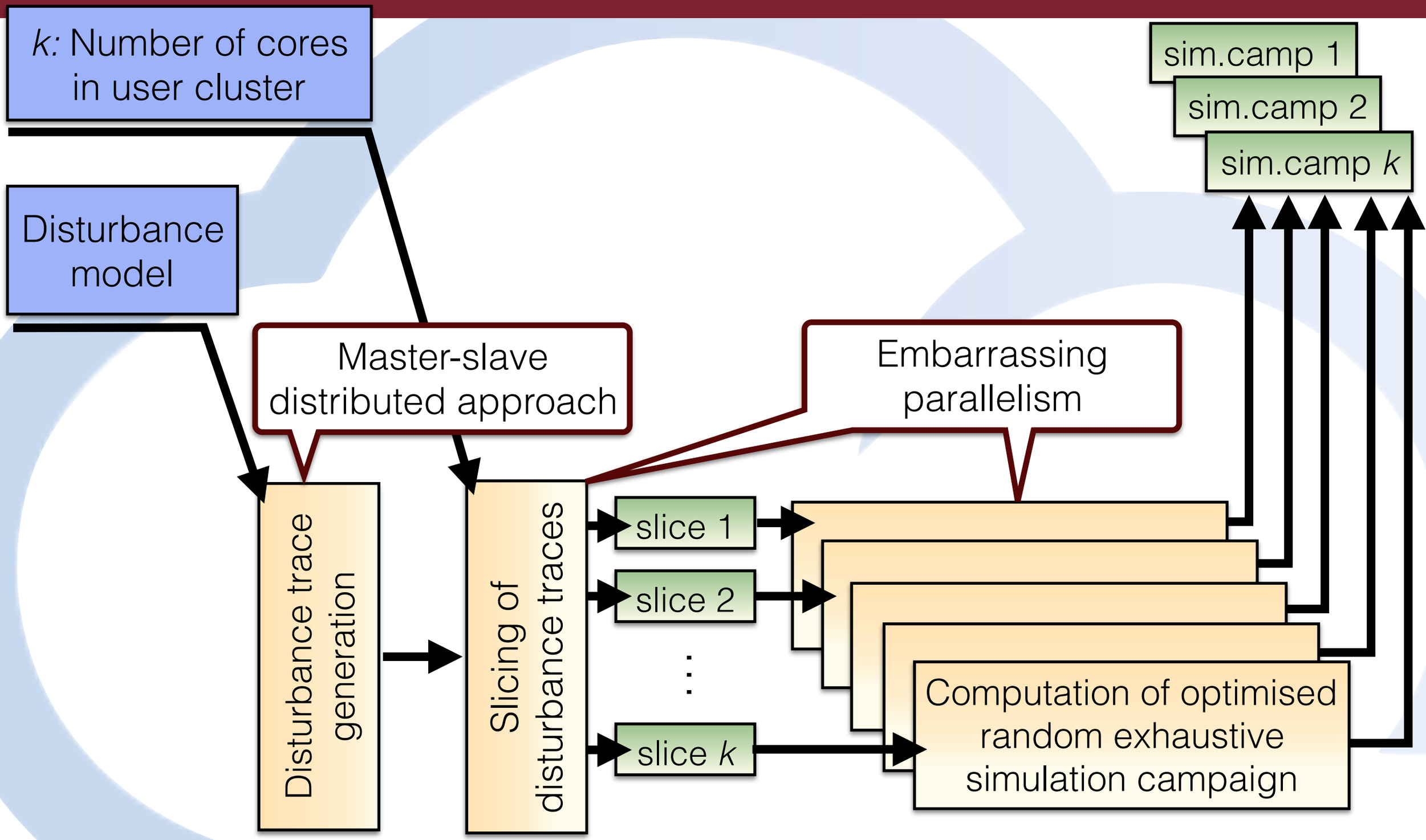
sim.camp 1

sim.camp 2

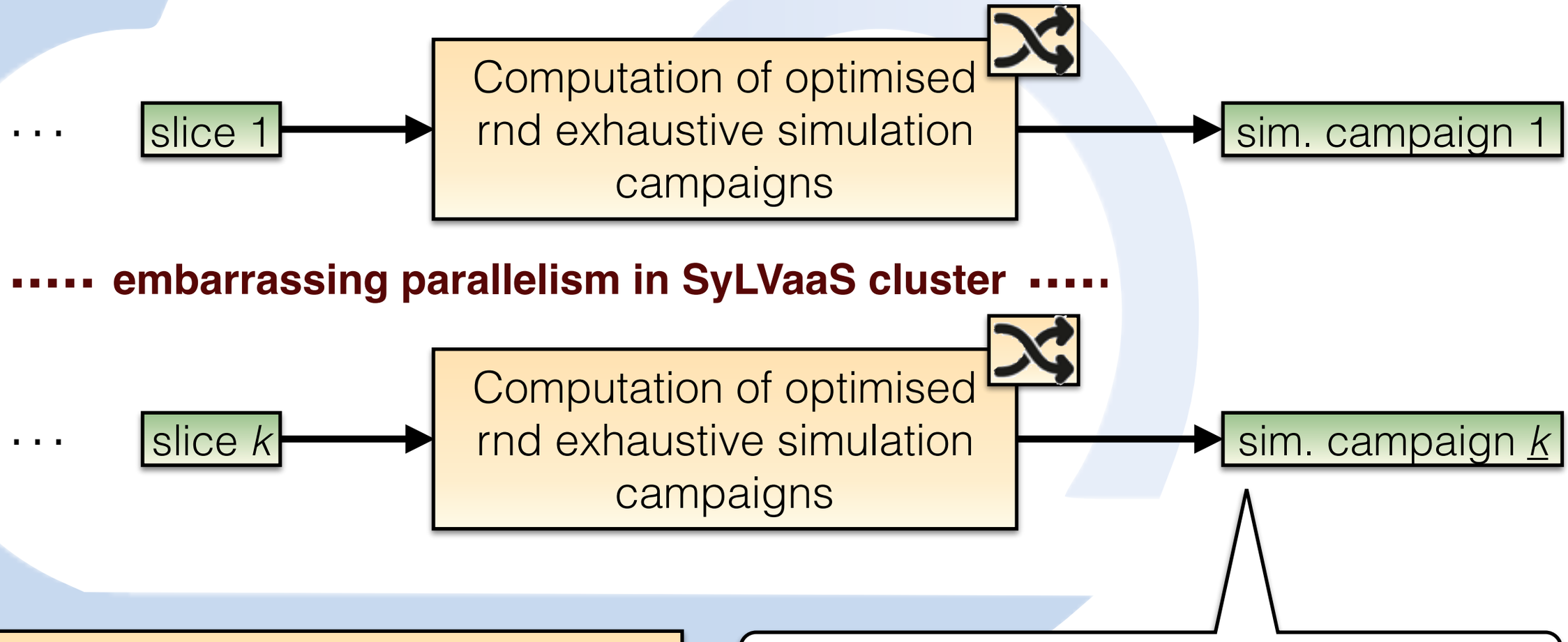
sim.camp *k*

Computation of optimised random exhaustive simulation campaign

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

Optimised Rnd Exhaustive Sim. Campaigns

Slice	
1	021001
2	022000
3	022030
4	023110
5	023220
6	030010

Optimised Rnd Exhaustive Sim. Campaigns

Slice	
1	021001
2	022000
3	022030
4	023110
5	023220
6	030010

Prefix labelling during generation (DFS —> free!)

Slice of **labelled** traces

1	a0b2c1d0e0f1g
2	a0b2c2h0i0j0k
3	a0b2c2h0i3m0n
4	a0b2c3p1q1r0s
5	a0b2c3p2v2w0x
6	a0b3y0z0α1β0λ

Labels univocally denote trace **prefixes**

Optimised Rnd Exhaustive Sim. Campaigns

Slice

1	021001
2	022000
3	022030
4	023110
5	023220
6	030010

Prefix labelling during generation (DFS —> free!)

Slice of **labelled** traces

1	a0b2c1d0e0f1g
2	a0b2c2h0i0j0k
3	a0b2c2h0i3m0n
4	a0b2c3p1q1r0s
5	a0b2c3p2v2w0x
6	a0b3y0z0α1β0λ

Labels univocally denote trace **prefixes**

Simulation campaign (rnd+optimised)

init	store(**a**)
3	load(**a**) inj_run(**0**,1) store(**b**) inj_run(**2**,1τ) store(**c**) inj_run(**2**,2τ) store(**i**) inj_run(**3**,2τ)
1	load(**c**) inj_run(**1**,3τ) inj_run(**1**,1τ)
6	load(**b**) free(**b**) inj_run(**3**,3τ) inj_run(**1**,2τ)
5	load(**c**) free(**c**) inj_run(**3**,1τ) store(**p**) inj_run(**2**,1τ) inj_run(**2**,2τ)
4	load(**p**) free(**p**) inj_run(**1**,1τ) inj_run(**1**,2τ)
2	load(**i**) free(**i**) free(**a**) inj_run(**0**,2τ)

Optimised Rnd Exhaustive Sim. Campaigns

Slice	
1	021001
2	022000
3	022030
4	023110
5	023220
6	030010

Prefix labelling during generation (DFS —> free!)

Slice of **labelled** traces

1	a0b2c1d0e0f1g
2	a0b2c2h0i0j0k
3	a0b2c2h0i3m0n
4	a0b2c3p1q1r0s
5	a0b2c3p2v2w0x
6	a0b3y0z0α1β0λ

Labels univocally denote trace **prefixes**

Simulation campaign (rnd+optimised)

init	store(**a**)
3	load(**a**) inj_run(**0**,1) store(**b**) inj_run(**2**,1τ) store(**c**) inj_run(**2**,2τ) store(**i**) inj_run(**3**,2τ)
1	load(**c**) inj_run(**1**,3τ) inj_run(**1**,1τ)
6	load(**b**) free(**b**) inj_run(**3**,3τ) inj_run(**1**,2τ)
5	load(**c**) free(**c**) inj_run(**3**,1τ) store(**p**) inj_run(**2**,1τ) inj_run(**2**,2τ)
4	load(**p**) free(**p**) inj_run(**1**,1τ) inj_run(**1**,2τ)
2	load(**i**) free(**i**) free(**a**) inj_run(**0**,2τ)

Optimised Rnd Exhaustive Sim. Campaigns

Slice	
1	021001
2	022000
3	022030
4	023110
5	023220
6	030010

Prefix labelling during generation (DFS —> free!)

Slice of **labelled** traces

1	a0b2c1d0e0f1g
2	a0b2c2h0i0j0k
3	a0b2c2h0i3m0n
4	a0b2c3p1q1r0s
5	a0b2c3p2v2w0x
6	a0b3y0z0α1β0λ

Labels univocally denote trace **prefixes**

Simulation campaign (rnd+optimised)

init	store(**a**)
3	load(**a**) inj_run(**0**,1) store(**b**) inj_run(**2**,1τ) store(**c**) inj_run(**2**,2τ) store(**i**) inj_run(**3**,2τ)
1	load(**c**) inj_run(**1**,3τ) inj_run(**1**,1τ)
6	load(**b**) free(**b**) inj_run(**3**,3τ) inj_run(**1**,2τ)
5	load(**c**) free(**c**) inj_run(**3**,1τ) store(**p**) inj_run(**2**,1τ) inj_run(**2**,2τ)
4	load(**p**) free(**p**) inj_run(**1**,1τ) inj_run(**1**,2τ)
2	load(**i**) free(**i**) free(**a**) inj_run(**0**,2τ)

Embarrassingly Parallel Simulation

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

Embarrassingly Parallel Simulation

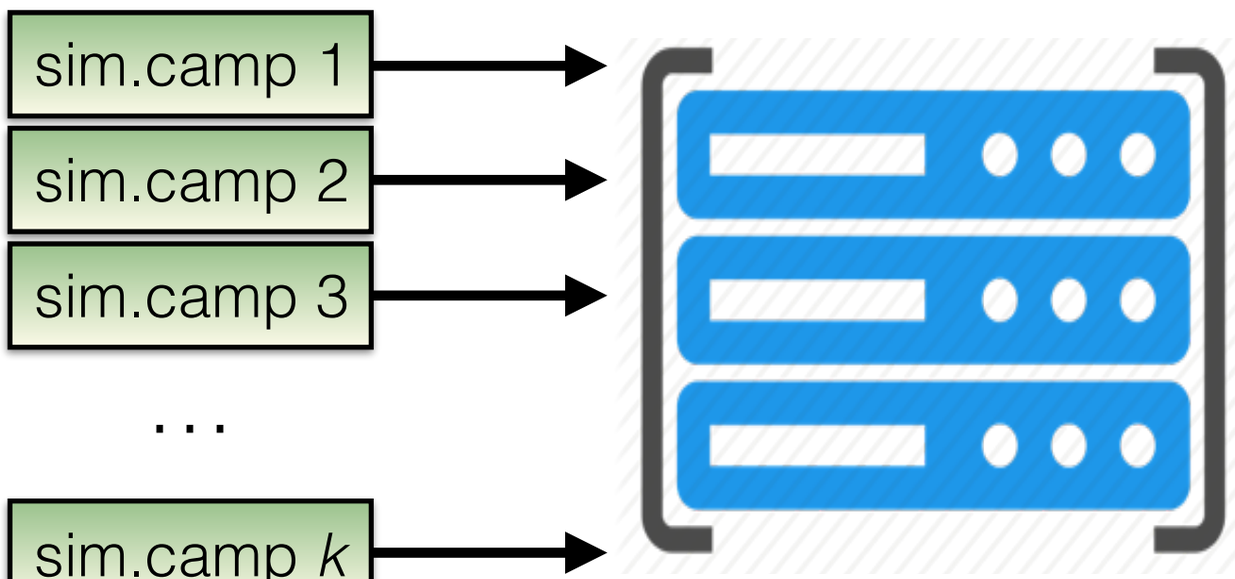
Simulation carried out on **user private cluster** (Intellectual Property protection)

k overall Simulink instances on *k* cores

SUV model + embedded property monitor

Embarrassingly Parallel Simulation

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



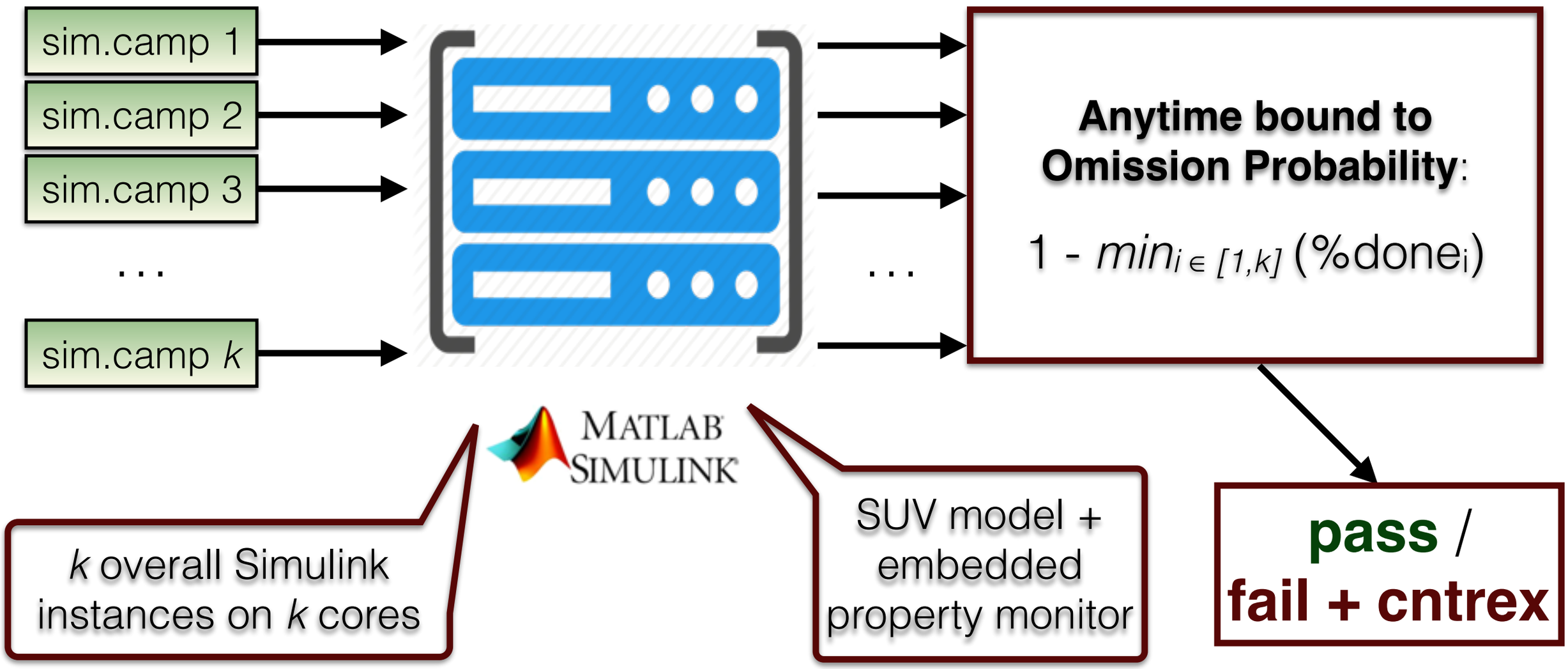
MATLAB SIMULINK

k overall Simulink instances on *k* cores

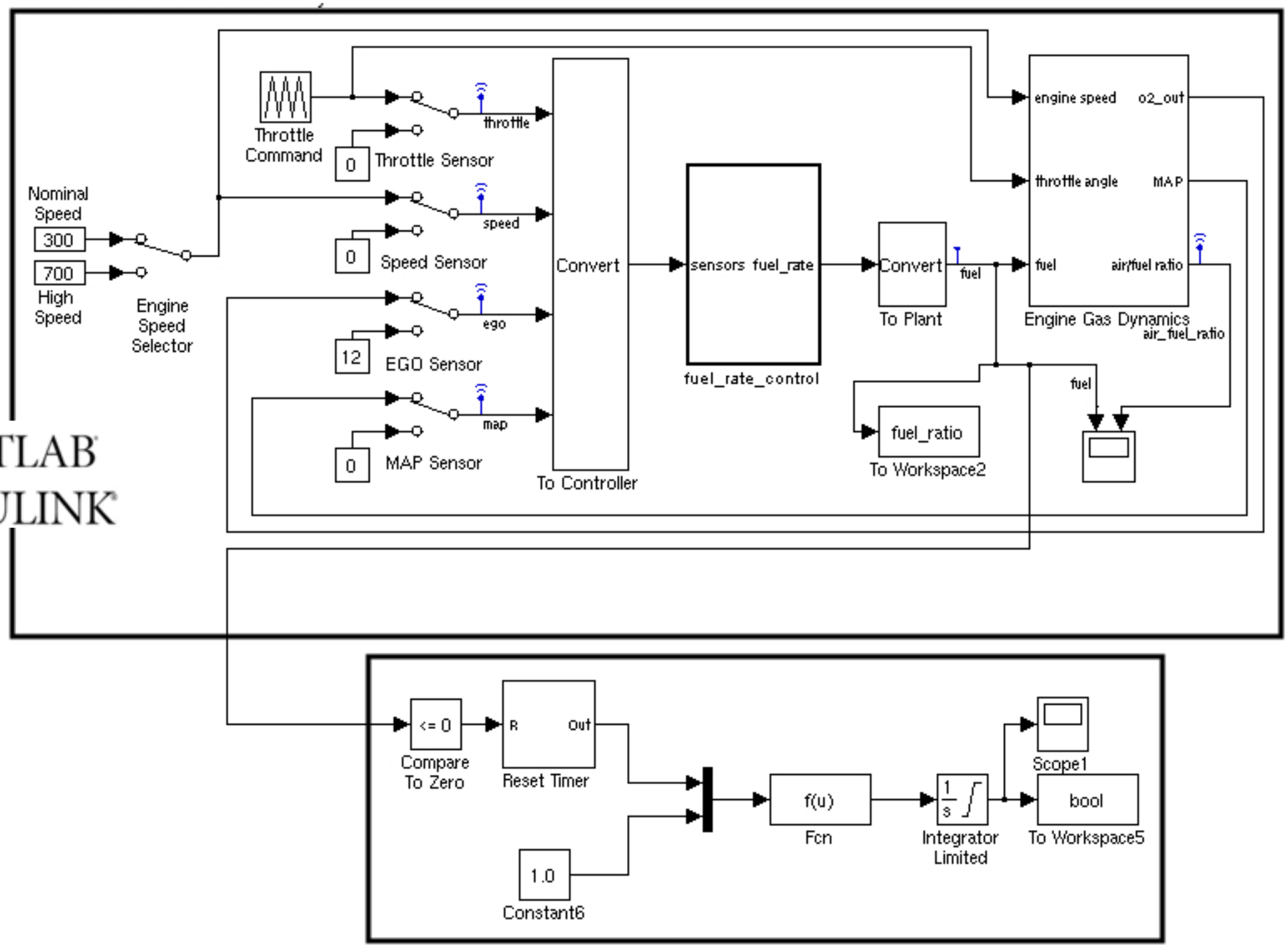
SUV model + embedded property monitor

Embarrassingly Parallel Simulation

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



Experiments: Fuel Control System

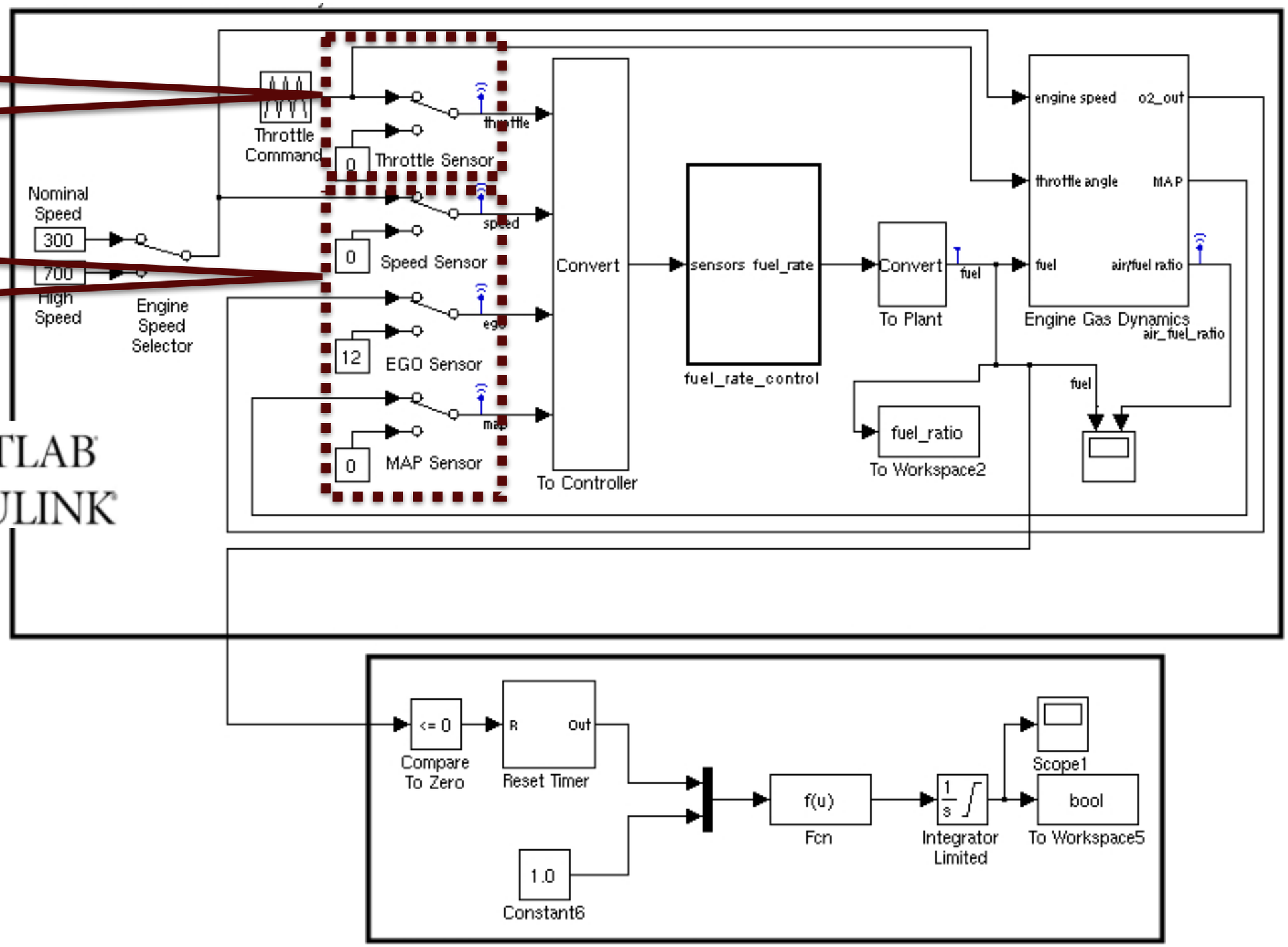


MATLAB SIMULINK

Experiments: Fuel Control System

Throttle degree as uncontrollable input

Three sensors subject to faults



MATLAB SIMULINK

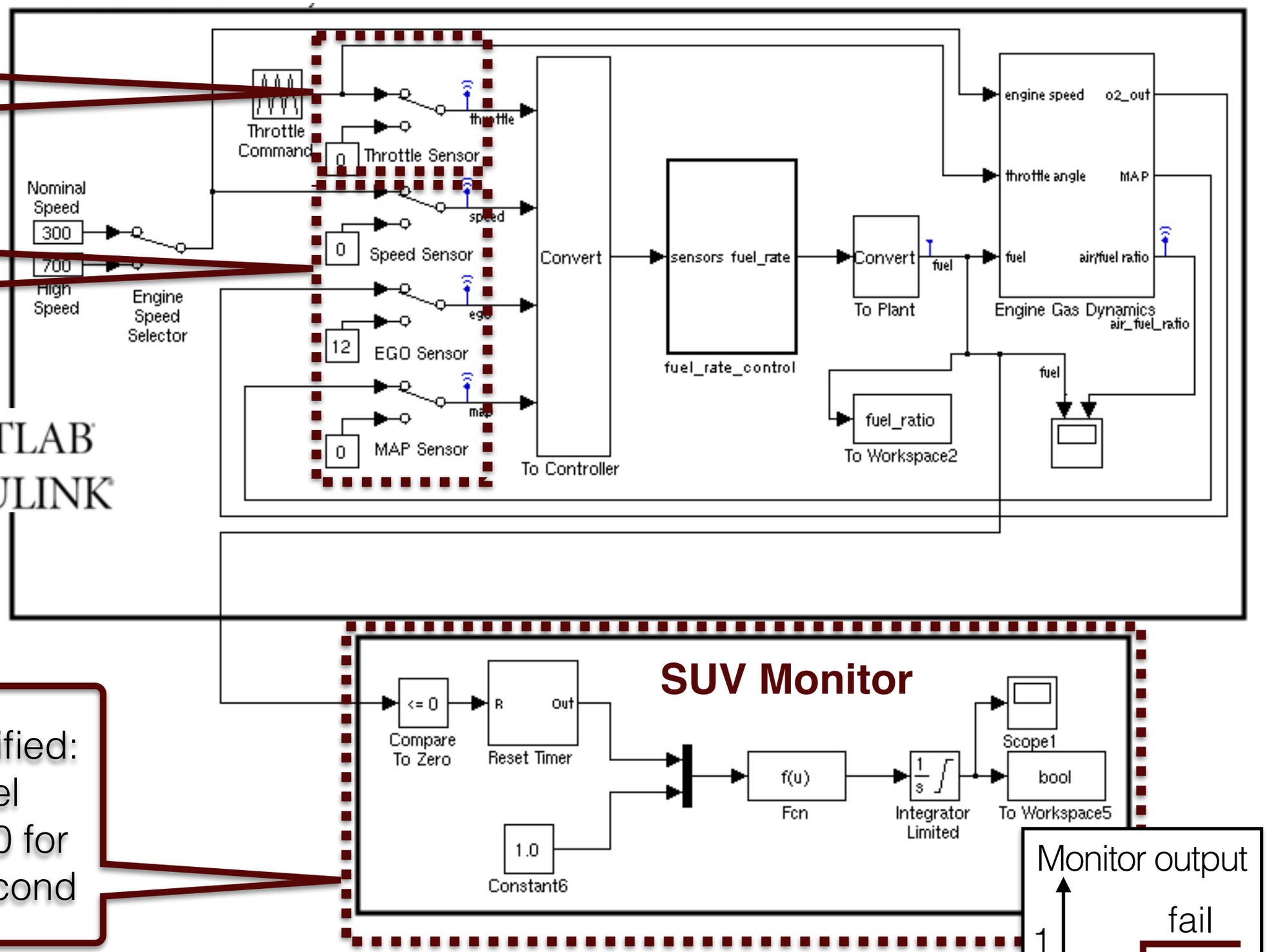
Experiments: Fuel Control System

Throttle degree as uncontrollable input

Three sensors subject to faults

MATLAB SIMULINK

Property to be verified: the “fuel” model variable is never 0 for more than one second



Experiments: Disturbance Models

Two disturbance models:

- sensor faults repaired after 1 second
- at most one fault active at any time
- **Model $\mathcal{D}_1$**: h = 100 sec, τ = 1s —> **4M dist. traces**
- **Model $\mathcal{D}_2$**: h = 200 sec, τ = 500ms —> **13M 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_NUM_FAULTS &
      num_active_faults() < MAX_NUM_ACTIVE_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;
```

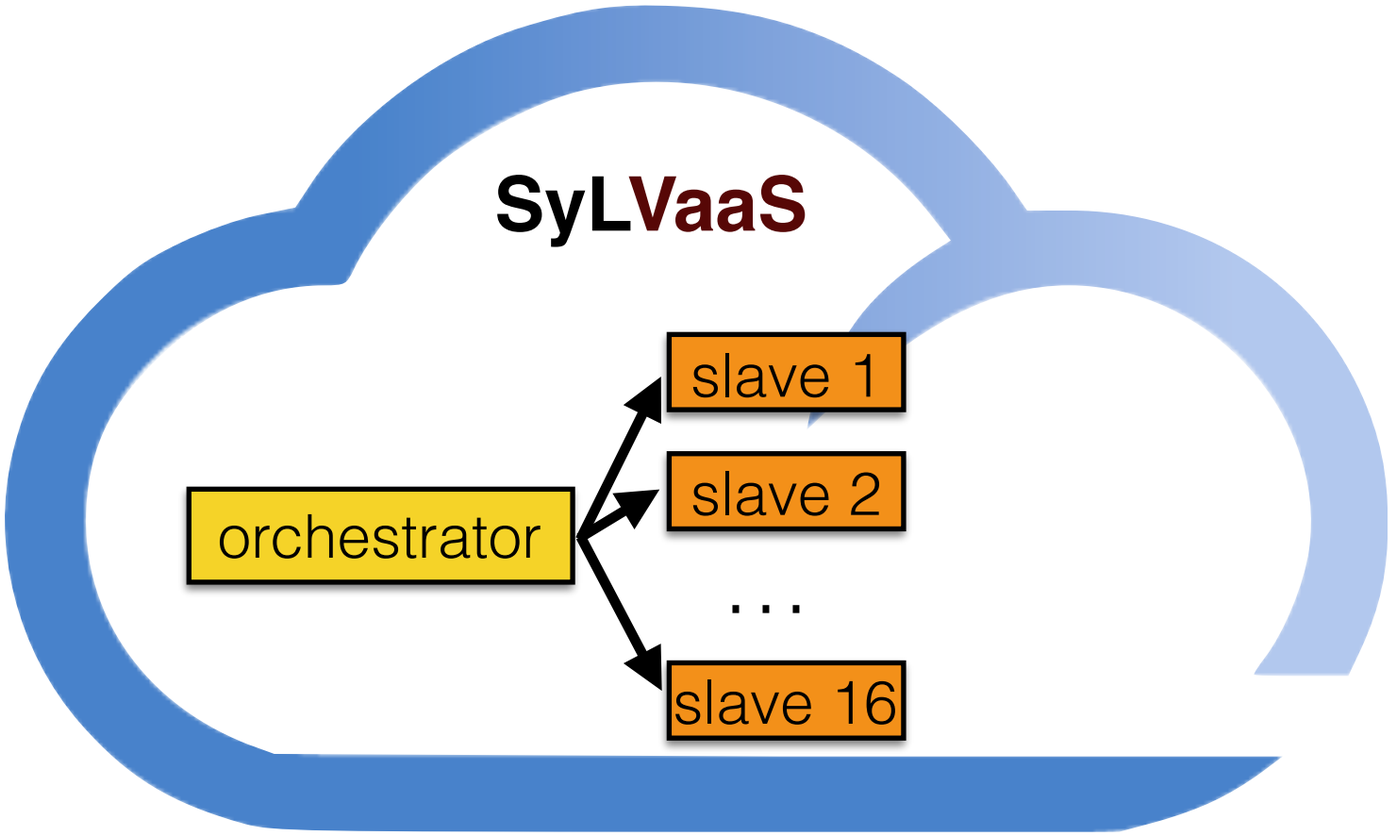
```
Ruleset d : INPUT_TYPE do
  Rule "Inject Input Variation"
      no_fault_needs_repair() &
      num_inputs < MAX_NUM_INPUTS &
      is_input_variation_allowed()
  ==> begin
          num_inputs := num_inputs + 1;
          time_step();
      end;
End;

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

Experiments: Infrastructure

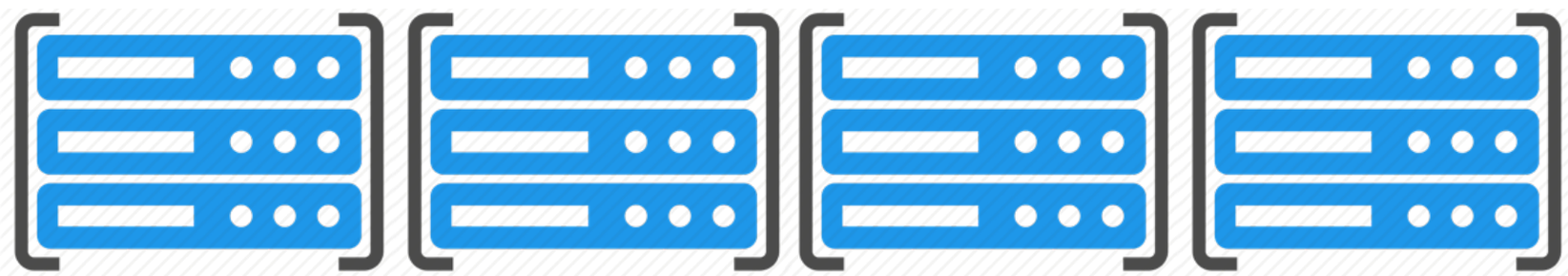
SyLVaaS infrastructure:

- 1 orchestrator
- 1 to 16 slaves



User private cluster:

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



Experiments: Trace Generation & Slicing

Parallel generation of disturbance traces

4M traces

13M traces

#slaves (S)	**disturbance model** $\mathcal{D}_1$			**disturbance model** $\mathcal{D}_2$		
	time (h:m:s)	**speedup**	**efficiency**	**time (h:m:s)**	**speedup**	**efficiency**
1	0:32:32	1.00×	100.00%	4:45:47	1.00×	100.00%
8	0:5:32	5.88×	73.50%	0:43:2	6.64×	83.00%
16	0:3:11	10.22×	63.88%	0:26:16	10.88×	68.00%

Slicing of disturbance traces

#slices (k)	$\mathcal{D}_1$ **(h:m:s)**	$\mathcal{D}_2$ **(h:m:s)**
128	0:4:1	0:8:7
256	0:4:32	0:11:25
512	0:4:52	0:13:17

Experiments: Random Exhaustive Campaigns

Computation of random exhaustive optimised simulation campaigns:

4M traces

13M traces

	disturbance model $\mathcal{D}_1$		**disturbance model $\mathcal{D}_2$**	
#slices (k)	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**
128	0:1:44	0:8:56	0:4:1	0:38:24
256	0:0:42	0:8:25	0:2:27	0:40:8
512	0:0:13	0:8:16	0:0:24	0:39:57

Experiments: Random Exhaustive Campaigns

Computation of random exhaustive optimised simulation campaigns:

4M traces

13M traces

	disturbance model $\mathcal{D}_1$		**disturbance model $\mathcal{D}_2$**	
#slices (k)	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**
128	0:1:44	0:8:56	0:4:1	0:38:24
256	0:0:42	0:8:25	0:2:27	0:40:8
512	0:0:13	0:8:16	0:0:24	0:39:57

Computation of **optimised random exhaustive** simulation campaigns via **embarrassing parallelism** (16 cores)

Experiments: Random Exhaustive Campaigns

Computation of random exhaustive optimised simulation campaigns:

4M traces

13M traces

	disturbance model $\mathcal{D}_1$		disturbance model $\mathcal{D}_2$	
#slices (k)	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**
128	0:1:44	0:8:56	0:4:1	0:38:24
256	0:0:42	0:8:25	0:2:27	0:40:8
512	0:0:13	0:8:16	0:0:24	0:39:57

Computation of **optimised random exhaustive** simulation campaigns via **embarrassing parallelism** (16 cores)

Overall SyLVaaS **response time** (gen+slicing+sim.camp.comp.)

Experiments: Random Exhaustive Campaigns

Computation of random exhaustive optimised simulation campaigns:

4M traces

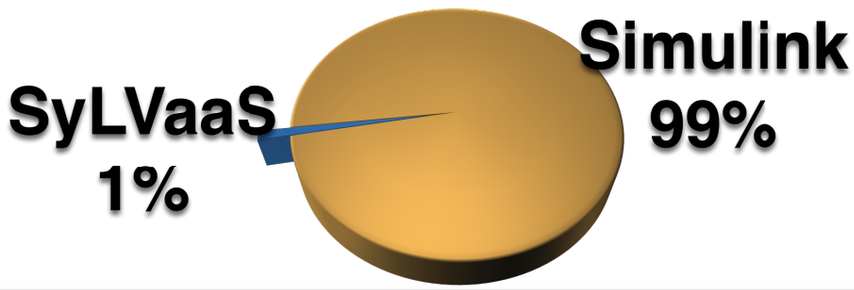
13M traces

	disturbance model $\mathcal{D}_1$		disturbance model $\mathcal{D}_2$	
#slices (k)	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**
128	0:1:44	0:8:56	0:4:1	0:38:24
256	0:0:42	0:8:25	0:2:27	0:40:8
512	0:0:13	0:8:16	0:0:24	0:39:57

Computation of **optimised random exhaustive** simulation campaigns via **embarrassing parallelism** (16 cores)

Overall SyLVaaS **response time** (gen+slicing+sim.camp.comp.)

SyLVaaS vs. **Simulink**



Experiments: Random Exhaustive Campaigns

Computation of random exhaustive optimised simulation campaigns:

4M traces

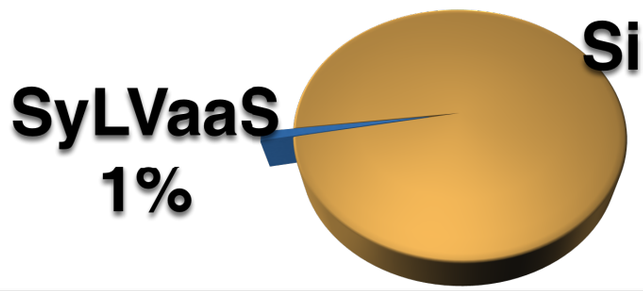
13M traces

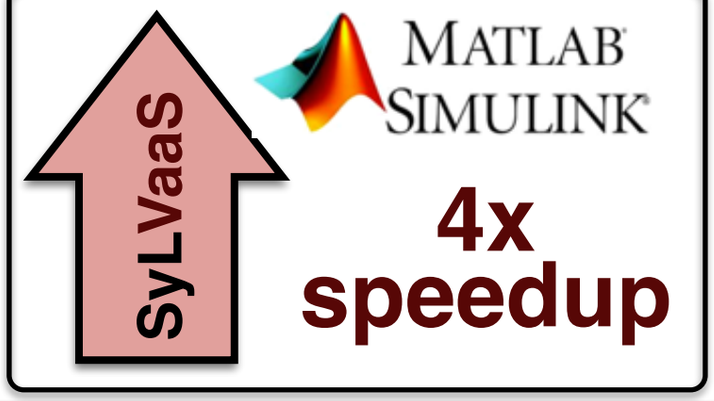
	disturbance model $\mathcal{D}_1$		disturbance model $\mathcal{D}_2$	
#slices (k)	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**
128	0:1:44	0:8:56	0:4:1	0:38:24
256	0:0:42	0:8:25	0:2:27	0:40:8
512	0:0:13	0:8:16	0:0:24	0:39:57

Computation of **optimised random exhaustive** simulation campaigns via **embarrassing parallelism** (16 cores)

Overall SyLVaaS **response time** (gen+slicing+sim.camp.comp.)

SyLVaaS vs. **Simulink**





Experiments: Random Exhaustive Campaigns

Computation of random exhaustive optimised simulation campaigns:

4M traces

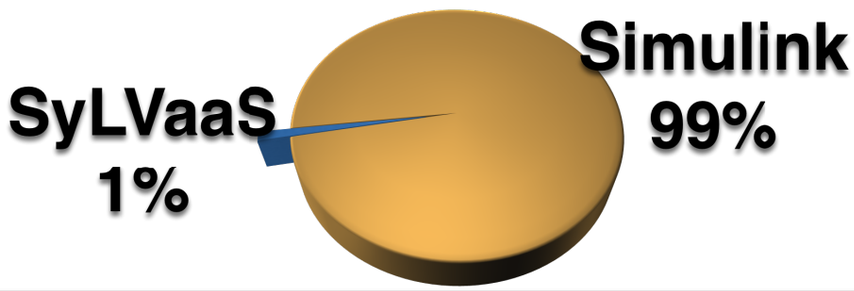
13M traces

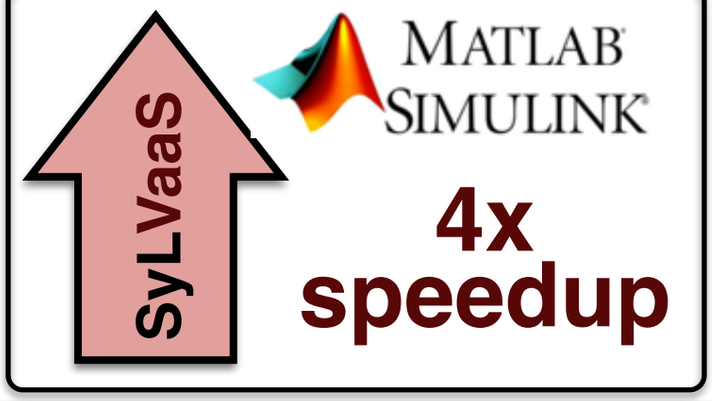
	disturbance model $\mathcal{D}_1$		disturbance model $\mathcal{D}_2$	
#slices (k)	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**	**sim. campaign comp. time (h:m:s)**	**overall time (h:m:s)**
128	0:1:44	0:8:56	0:4:1	0:38:24
256	0:0:42	0:8:25	0:2:27	0:40:8
512	0:0:13	0:8:16	0:0:24	0:39:57

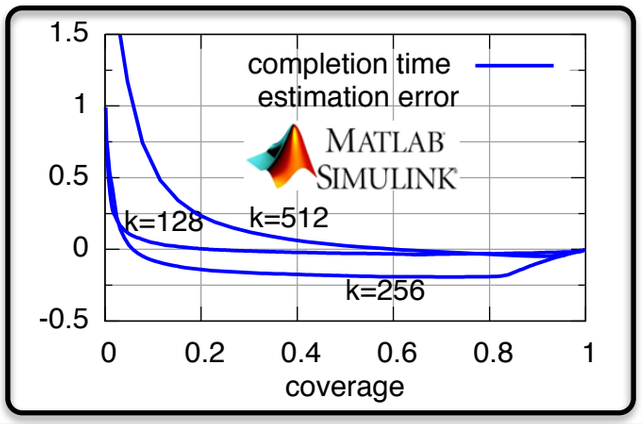
Computation of **optimised random exhaustive** simulation campaigns via **embarrassing parallelism** (16 cores)

Overall SyLVaaS **response time** (gen+slicing+sim.camp.comp.)

SyLVaaS vs. **Simulink**



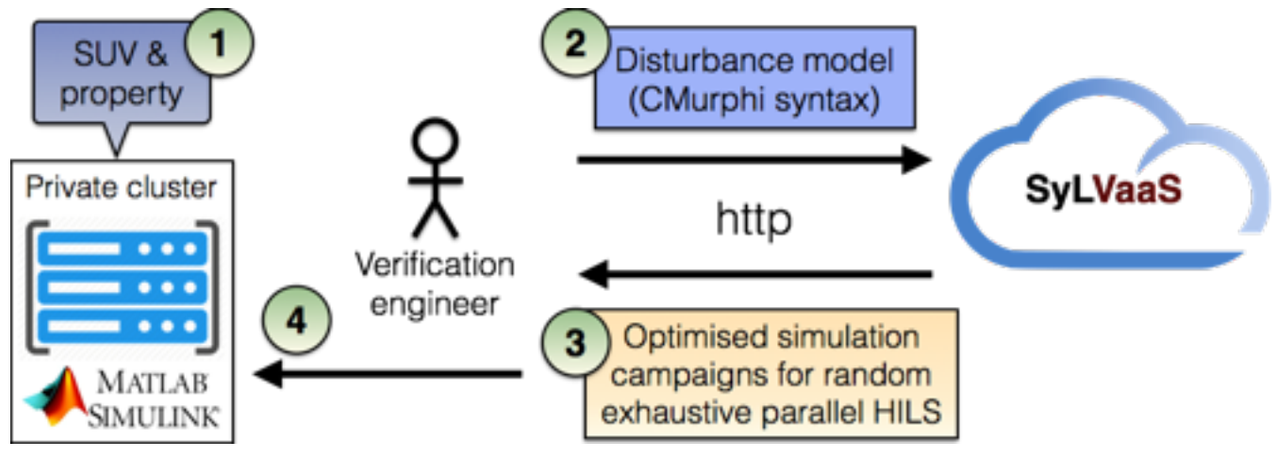




Conclusions

SyLVaaS: **Sy**stem **L**evel **V**erification **a**s **a** **S**ervice

- 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**)



SAPIENZA
UNIVERSITÀ DI ROMA

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