

# Unmasking Fault Tolerance

Quantifying deterministic recovery dynamics  
in probabilistic environments

Nils Müllner

Fakultät II, Department für Informatik  
Carl von Ossietzky Universität Oldenburg



February 26, 2014

# Agenda

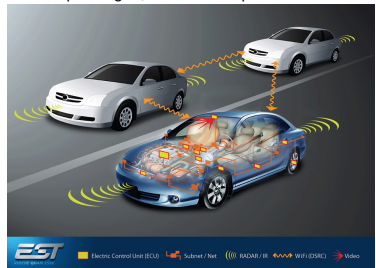
- 1 Introduction
- 2 Concept
- 3 Computation
- 4 Composition
- 5 Conclusion
- 6 Literature

# The world of fault tolerance

- ▶ Distributed systems are omnipresent,
  - ▶ like consumers of a power grid or
  - ▶ distributed sensors in a car or airplane.
- ▶ A distributed system comprises processes that can collaborate to provide a service, like providing energy or environmental data.



power grid, source: wordpress.com



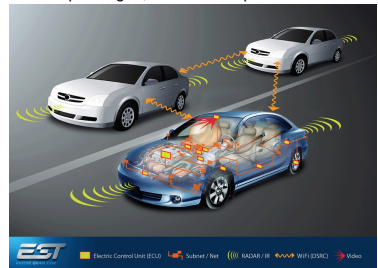
distributed sensors, source: mathworks.com

# The world of fault tolerance

- ▶ Distributed systems are omnipresent,
  - ▶ like consumers of a power grid or
  - ▶ distributed sensors in a car or airplane.
- ▶ A distributed system comprises processes that can collaborate to provide a service, like providing energy or environmental data.



power grid, source: wordpress.com



distributed sensors, source: mathworks.com



# The world of fault tolerance

- ▶ The service such a system provides
  - a) can be critical (*hard* safety requirements)  
⇒ must never fail!
  - b) or not (*soft* safety requirements)  
⇒ temporary downtimes are acceptable.
- ▶ Focus on category b).  
Less critical systems that can cope with temporary invalidation of safety.
- ▶ Systems that can recover from the effects of faults can run indefinitely.

# The world of fault tolerance

- ▶ The service such a system provides
  - a) can be critical (*hard* safety requirements)  
⇒ must never fail!
  - b) or not (*soft* safety requirements)  
⇒ temporary downtimes are acceptable.
- ▶ Focus on category b).
- Less critical systems that can cope with temporary invalidation of safety.
- ▶ Systems that can recover from the effects of faults can run indefinitely.

# The world of fault tolerance

- ▶ The service such a system provides
  - a) can be critical (*hard* safety requirements)  
⇒ must never fail!
  - b) or not (*soft* safety requirements)  
⇒ temporary downtimes are acceptable.
- ▶ Focus on category b).  
Less critical systems that can cope with temporary invalidation of safety.
- ▶ Systems that can recover from the effects of faults can run indefinitely.

# The world of fault tolerance

- ▶ The service such a system provides
  - a) can be critical (*hard* safety requirements)  
⇒ must never fail!
  - b) or not (*soft* safety requirements)  
⇒ temporary downtimes are acceptable.
- ▶ Focus on category b).
- Less critical systems that can cope with temporary invalidation of safety.
- ▶ Systems that can recover from the effects of faults can run indefinitely.

# The world of fault tolerance

- ▶ Distributed systems are prone to *sporadic transient faults*.
- ▶ Such faults occur *probabilistically* and can corrupt values stored in processes.
- ▶ Other *probabilistic* influence – like a central scheduler – might further influence the system.

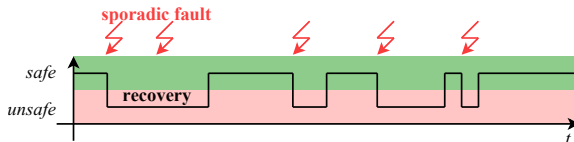
# The world of fault tolerance

- ▶ Distributed systems are prone to *sporadic transient faults*.
- ▶ Such faults occur *probabilistically* and can corrupt values stored in processes.
- ▶ Other *probabilistic* influence – like a central scheduler – might further influence the system.

# The world of fault tolerance

- ▶ Distributed systems are prone to *sporadic transient faults*.
- ▶ Such faults occur *probabilistically* and can corrupt values stored in processes.
- ▶ Other *probabilistic* influence – like a central scheduler – might further influence the system.

# Key questions



- ▶ How well does such a distributed system *provide its service over time*?
- ▶ How well does such a distributed system *recover over time*?



# Requirements

To answer these questions, we need:

1. a *measure* to quantify recovery and
2. a *method* to compute that measure.

# Limiting window availability

... is the *probability*

that a system works according to its specification

(i.e. it is in a *safe* state)

at least once *within a time frame*,

considering the stationary as initial distribution.

[SSS2006,ATC2009,WAINA2011]

# Limiting window availability

... is the *probability*  
that a system works according to its specification  
(i.e. it is in a *safe state*)  
at least once *within a time frame*,  
considering the stationary as initial distribution.  
[SSS2006,ATC2009,WAINA2011]

# Limiting window availability

... is the *probability*  
that a system works according to its specification  
(i.e. it is in a *safe state*)  
at least once *within a time frame*,  
considering the stationary as initial distribution.  
[SSS2006,ATC2009,WAINA2011]

# Limiting window availability

... is the *probability*  
that a system works according to its specification  
(i.e. it is in a *safe* state)  
at least once *within a time frame*,  
considering the stationary as initial distribution.  
[SSS2006,ATC2009,WAINA2011]

# Independent limit

## Why *limiting*?

---

Generally, any arbitrary distribution is applicable.

---

For *indefinitely running* systems,  
the stationary distribution is the interesting one.

# Independent limit

Why *limiting*?

---

Generally, any arbitrary distribution is applicable.

---

For *indefinitely running* systems,  
the stationary distribution is the interesting one.

# Independent limit

Why *limiting*?

---

Generally, any arbitrary distribution is applicable.

---

For *indefinitely running* systems,  
the stationary distribution is the interesting one.





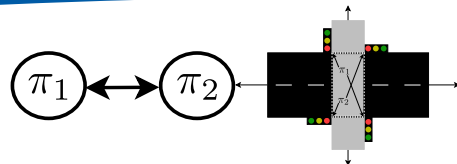
# System model and sporadic faults

A distributed system comprises *processes* taking serial execution steps.

Processes can *communicate to achieve common goal*, controlling a critical intersection.

An algorithm contains *both functional and recovery* instructions.

*Sporadic faults* let executing processes store arbitrary values.



Step	Process	Variable	Value	Step	Process	Variable	Value
1	π1	color	red	1	π2	color	red
2	π1	color	red	2	π2	color	red
3	π1	color	red	3	π2	color	red
4	π1	color	red	4	π2	color	red
5	π1	color	red	5	π2	color	red
6	π1	color	red	6	π2	color	red
7	π1	color	red	7	π2	color	red
8	π1	color	red	8	π2	color	red
9	π1	color	red	9	π2	color	red
10	π1	color	red	10	π2	color	red
11	π1	color	red	11	π2	color	red
12	π1	color	red	12	π2	color	red
13	π1	color	red	13	π2	color	red
14	π1	color	red	14	π2	color	red
15	π1	color	red	15	π2	color	red
16	π1	color	red	16	π2	color	red
17	π1	color	red	17	π2	color	red
18	π1	color	red	18	π2	color	red
19	π1	color	red	19	π2	color	red
20	π1	color	red	20	π2	color	red
21	π1	color	red	21	π2	color	red
22	π1	color	red	22	π2	color	red
23	π1	color	red	23	π2	color	red
24	π1	color	red	24	π2	color	red
25	π1	color	red	25	π2	color	red
26	π1	color	red	26	π2	color	red
27	π1	color	red	27	π2	color	red
28	π1	color	red	28	π2	color	red
29	π1	color	red	29	π2	color	red
30	π1	color	red	30	π2	color	red
31	π1	color	red	31	π2	color	red
32	π1	color	red	32	π2	color	red
33	π1	color	red	33	π2	color	red
34	π1	color	red	34	π2	color	red
35	π1	color	red	35	π2	color	red
36	π1	color	red	36	π2	color	red
37	π1	color	red	37	π2	color	red
38	π1	color	red	38	π2	color	red
39	π1	color	red	39	π2	color	red
40	π1	color	red	40	π2	color	red
41	π1	color	red	41	π2	color	red
42	π1	color	red	42	π2	color	red
43	π1	color	red	43	π2	color	red
44	π1	color	red	44	π2	color	red
45	π1	color	red	45	π2	color	red
46	π1	color	red	46	π2	color	red
47	π1	color	red	47	π2	color	red
48	π1	color	red	48	π2	color	red
49	π1	color	red	49	π2	color	red
50	π1	color	red	50	π2	color	red
51	π1	color	red	51	π2	color	red
52	π1	color	red	52	π2	color	red
53	π1	color	red	53	π2	color	red
54	π1	color	red	54	π2	color	red
55	π1	color	red	55	π2	color	red
56	π1	color	red	56	π2	color	red
57	π1	color	red	57	π2	color	red
58	π1	color	red	58	π2	color	red
59	π1	color	red	59	π2	color	red
60	π1	color	red	60	π2	color	red
61	π1	color	red	61	π2	color	red
62	π1	color	red	62	π2	color	red
63	π1	color	red	63	π2	color	red
64	π1	color	red	64	π2	color	red
65	π1	color	red	65	π2	color	red
66	π1	color	red	66	π2	color	red
67	π1	color	red	67	π2	color	red
68	π1	color	red	68	π2	color	red
69	π1	color	red	69	π2	color	red
70	π1	color	red	70	π2	color	red
71	π1	color	red	71	π2	color	red
72	π1	color	red	72	π2	color	red
73	π1	color	red	73	π2	color	red
74	π1	color	red	74	π2	color	red
75	π1	color	red	75	π2	color	red
76	π1	color	red	76	π2	color	red
77	π1	color	red	77	π2	color	red
78	π1	color	red	78	π2	color	red
79	π1	color	red	79	π2	color	red
80	π1	color	red	80	π2	color	red
81	π1	color	red	81	π2	color	red
82	π1	color	red	82	π2	color	red
83	π1	color	red	83	π2	color	red
84	π1	color	red	84	π2	color	red
85	π1	color	red	85	π2	color	red
86	π1	color	red	86	π2	color	red
87	π1	color	red	87	π2	color	red
88	π1	color	red	88	π2	color	red
89	π1	color	red	89	π2	color	red
90	π1	color	red	90	π2	color	red
91	π1	color	red	91	π2	color	red
92	π1	color	red	92	π2	color	red
93	π1	color	red	93	π2	color	red
94	π1	color	red	94	π2	color	red
95	π1	color	red	95	π2	color	red
96	π1	color	red	96	π2	color	red
97	π1	color	red	97	π2	color	red
98	π1	color	red	98	π2	color	red
99	π1	color	red	99	π2	color	red
100	π1	color	red	100	π2	color	red

Table: Traffic lights algorithm – guarded commands

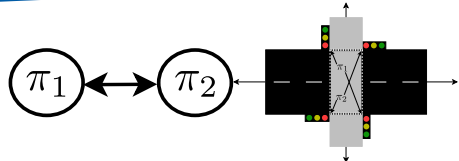
# System model and sporadic faults

A distributed system comprises *processes* taking serial execution steps.

Processes can *communicate* to achieve common goal, controlling a critical intersection.

An algorithm contains *both functional and recovery* instructions.

Sporadic *faults* let executing processes store arbitrary values.



Label	Guard enabled and $s = \pi_1$	Command	Label	Guard enabled and $s = \pi_2$	Command
a1	$R_1 = \text{green} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a26	$R_1 = \text{green} \wedge R_2 = \text{green}$	$R_2 := \text{red}_1$
a2	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a27	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a3	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a28	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a4	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a29	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_2 := \text{red}_1$
a5	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a30	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_2 := \text{red}_1$
a6	$R_1 = \text{green} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a31	$R_1 = \text{green} \wedge R_2 = \text{red}$	$R_2 := \text{red}_1$
a7	$R_1 = \text{green} \wedge R_2 = \text{red}_1$	$R_1 := \text{yellow}_1$	a32	$R_1 = \text{green} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}_1$
a8	$R_1 = \text{red} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a33	$R_1 = \text{red} \wedge R_2 = \text{green}$	$R_2 := \text{red}_1$
a9	$R_1 = \text{red}_1 \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a34	$R_1 = \text{red}_1 \wedge R_2 = \text{green}$	$R_2 := \text{yellow}_1$
a10	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a35	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a11	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a36	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a12	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a37	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a13	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a38	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a14	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a39	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_2 := \text{red}_1$
a15	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a40	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_2 := \text{red}_1$
a16	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_1 := \text{green}$	a41	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}_1$
a17	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_1 := \text{red}_1$	a42	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}_1$
a18	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a43	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a19	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a44	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a20	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a45	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_2 := \text{green}$
a21	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a46	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a22	$R_1 = \text{red} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a47	$R_1 = \text{red} \wedge R_2 = \text{red}$	$R_2 := \text{red}_1$
a23	$R_1 = \text{red}_1 \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a48	$R_1 = \text{red}_1 \wedge R_2 = \text{red}$	$R_2 := \text{yellow}$
a24	$R_1 = \text{red} \wedge R_2 = \text{red}_1$	$R_1 := \text{green}$	a49	$R_1 = \text{red} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}_1$
a25	$R_1 = \text{red}_1 \wedge R_2 = \text{red}_1$	$R_1 := \text{yellow}_1$	a50	$R_1 = \text{red}_1 \wedge R_2 = \text{red}_1$	$R_2 := \text{green}$

Table: Traffic lights algorithm – guarded commands

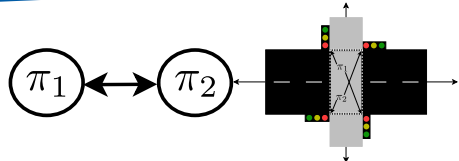
# System model and sporadic faults

A distributed system comprises *processes* taking serial execution steps.

Processes can *communicate* to achieve common goal, controlling a critical intersection.

An algorithm contains *both functional and recovery* instructions.

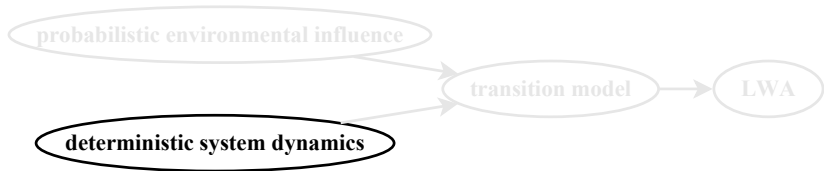
Sporadic *faults* let executing processes store arbitrary values.



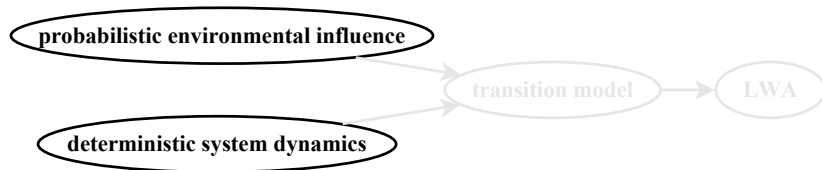
Label	Guard enabled and $s = \pi_1$	Command	Label	Guard enabled and $s = \pi_2$	Command
a <sub>1</sub>	$R_1 = \text{green} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a <sub>26</sub>	$R_1 = \text{green} \wedge R_2 = \text{green}$	$R_2 := \text{red}_1$
a <sub>2</sub>	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>27</sub>	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a <sub>3</sub>	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>28</sub>	$R_1 = \text{green} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a <sub>4</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a <sub>29</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_2 := \text{red}_1$
a <sub>5</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a <sub>30</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{green}$	$R_2 := \text{red}_1$
a <sub>6</sub>	$R_1 = \text{green} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a <sub>31</sub>	$R_1 = \text{green} \wedge R_2 = \text{red}$	$R_2 := \text{red}_1$
a <sub>7</sub>	$R_1 = \text{green} \wedge R_2 = \text{red}_1$	$R_1 := \text{yellow}_1$	a <sub>32</sub>	$R_1 = \text{green} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}_1$
a <sub>8</sub>	$R_1 = \text{red} \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a <sub>33</sub>	$R_1 = \text{red} \wedge R_2 = \text{green}$	$R_2 := \text{red}$
a <sub>9</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{green}$	$R_1 := \text{red}_1$	a <sub>34</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{green}$	$R_2 := \text{yellow}_1$
a <sub>10</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>35</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a <sub>11</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>36</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a <sub>12</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>37</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a <sub>13</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>38</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a <sub>14</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a <sub>39</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_2 := \text{red}_1$
a <sub>15</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a <sub>40</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}$	$R_2 := \text{red}_1$
a <sub>16</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_1 := \text{green}$	a <sub>41</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}_1$
a <sub>17</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_1 := \text{red}_1$	a <sub>42</sub>	$R_1 = \text{yellow} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}_1$
a <sub>18</sub>	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>43</sub>	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}$
a <sub>19</sub>	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>44</sub>	$R_1 = \text{red} \wedge R_2 = \text{yellow}$	$R_2 := \text{red}$
a <sub>20</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>45</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_2 := \text{green}$
a <sub>21</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_1 := \text{red}_1$	a <sub>46</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{yellow}$	$R_2 := \text{red}_1$
a <sub>22</sub>	$R_1 = \text{red} \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a <sub>47</sub>	$R_1 = \text{red} \wedge R_2 = \text{red}$	$R_2 := \text{red}$
a <sub>23</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{red}$	$R_1 := \text{red}_1$	a <sub>48</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{red}$	$R_2 := \text{yellow}$
a <sub>24</sub>	$R_1 = \text{red} \wedge R_2 = \text{red}_1$	$R_1 := \text{green}$	a <sub>49</sub>	$R_1 = \text{red} \wedge R_2 = \text{red}_1$	$R_2 := \text{red}$
a <sub>25</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{red}_1$	$R_1 := \text{yellow}$	a <sub>50</sub>	$R_1 = \text{red}_1 \wedge R_2 = \text{red}_1$	$R_2 := \text{green}$

Table: Traffic lights algorithm – guarded commands

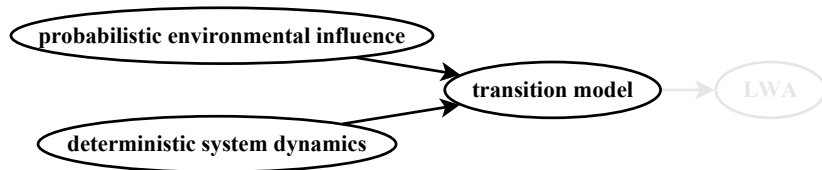
# From system and environment to computing LWA



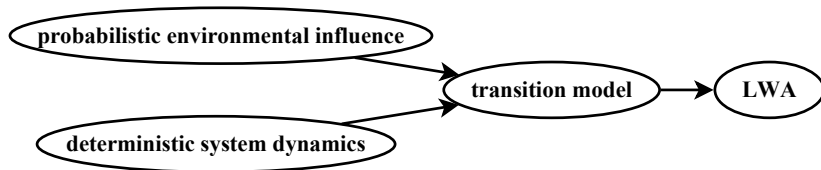
# From system and environment to computing LWA



# From system and environment to computing LWA



# From system and environment to computing LWA





# State space, transition model, safety

- ▶ *system state*  $s_t = \langle g, r_1 \rangle$
- 

- ▶ *state space*

$$\mathcal{S} = \{ \langle g, g \rangle, \langle g, y \rangle, \dots, \langle r_1, r_1 \rangle \}$$

---

- ▶ *transition probability*

$$\text{prob}(\overrightarrow{\langle g, g \rangle, \langle g, r_1 \rangle})$$

---

- ▶ (state based) *safety*:  
at least one light shows,  $r$  or  $r_1$
- 

- ▶ partitions state space into *legal*  
and *illegal* states



# State space, transition model, safety

- ▶ *system state*  $s_t = \langle g, r_1 \rangle$

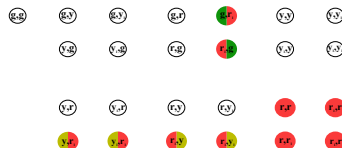
- ▶ *state space*

$$\mathcal{S} = \{ \langle g, g \rangle, \langle g, y \rangle, \dots, \langle r_1, r_1 \rangle \}$$

- ▶ *transition probability*  
 $\text{prob}(\langle g, g \rangle, \langle g, r_1 \rangle)$

- ▶ (state based) *safety*:  
at least one light shows,  $r$  or  $r_1$

- ▶ partitions state space into *legal* and *illegal* states



# State space, transition model, safety

- ▶ *system state*  $s_t = \langle g, r_1 \rangle$

- ▶ *state space*

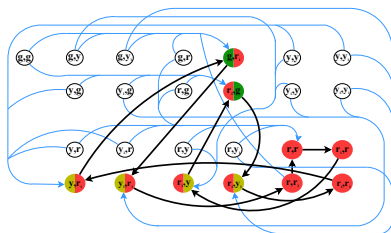
$$\mathcal{S} = \{ \langle g, g \rangle, \langle g, y \rangle, \dots, \langle r_1, r_1 \rangle \}$$

- ▶ *transition probability*

$$\text{prob}(\langle g, g \rangle, \langle g, r_1 \rangle)$$

- ▶ (state based) *safety*:  
at least one light shows,  $r$  or  $r_1$

- ▶ partitions state space into *legal*  
and *illegal* states



# State space, transition model, safety

- ▶ system state  $s_t = \langle g, r_1 \rangle$

- ▶ state space

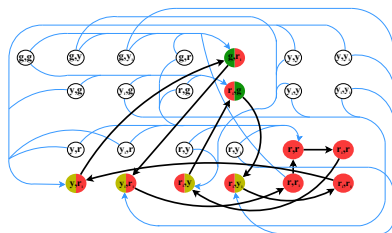
$$\mathcal{S} = \{ \langle g, g \rangle, \langle g, y \rangle, \dots, \langle r_1, r_1 \rangle \}$$

- ▶ transition probability

$$\text{prob}(\langle g, g \rangle, \langle g, r_1 \rangle)$$

- ▶ (state based) safety:  
at least one light shows,  $r$  or  $r_1$

- ▶ partitions state space into *legal*  
and *illegal* states



# State space, transition model, safety

- ▶ system state  $s_t = \langle g, r_1 \rangle$

- ▶ state space

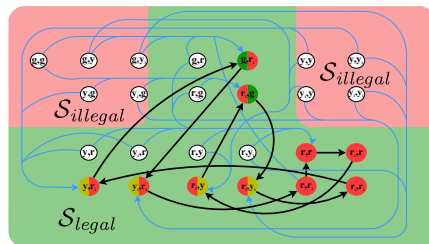
$$\mathcal{S} = \{ \langle g, g \rangle, \langle g, y \rangle, \dots, \langle r_1, r_1 \rangle \}$$

- ▶ transition probability

$$\text{prob}(\langle g, g \rangle, \langle g, r_1 \rangle)$$

- ▶ (state based) safety:  
at least one light shows,  $r$  or  $r_1$

- ▶ partitions state space into *legal* and *illegal* states



# Traffic light transition matrix

Table: Transition model  $\mathcal{D}$  of traffic light example

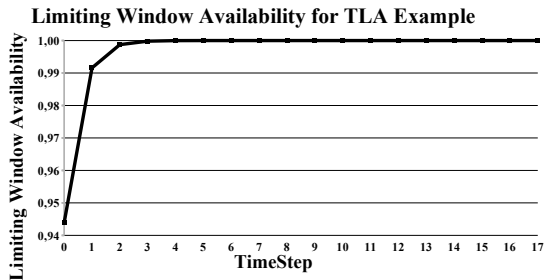
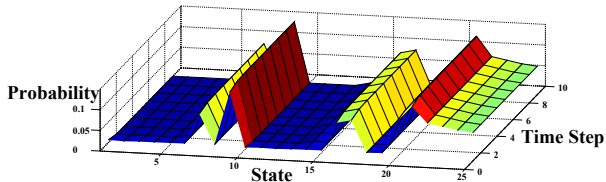
number of states =

number of *possible values* to the power of  
number of *registers* to the power of  
number of *processes*

here: 5 values in 1 register for each of 2 traffic lights processes:

$$|\mathcal{S}| = 5^{1^2} = 25$$

# Traffic lights LWA



# State space, transition model, safety

- ▶ small two process system for proof of concept: ✓
- ▶ next step: larger, more complex system, different algorithm

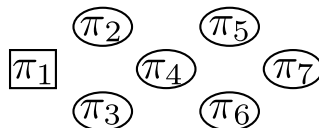


# State space, transition model, safety

- ▶ small two process system for proof of concept: ✓
- ▶ next step: larger, more complex system, different algorithm

# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.



Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

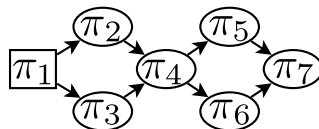


Sporadic faults let executing processes store wrong values (2).

Figure: Broadcast algorithm – self-stabilizing (BASS)

# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.



Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

```
const Id >= 0;
var R;
repeat {
  R := 0;
}
```

```
const neighbors := {n1, ..., nk};
const distance
  := min(distance(neighbors)) + 1;
const set := {R1, ..., Rk} (∀ n1 :
  {n1 ∈ neighbors} ∧
  (distance(n1) = distance + 1));
var R;
repeat {
  ¬((∃ R1 : n1 ∈ set ∧ R1 = 0) ∧
    ∃ R1 : n1 ∈ set ∧ R1 = 0)
  → R := 1;
  ∃ R1 : n1 ∈ set ∧ R1 = 0
  → R := 0;
  ∃ R1 : n1 ∈ set ∧ R1 = 2
  → R := 2;
}
```

Sporadic faults let executing processes store wrong values (2).

Figure: Broadcast algorithm – self-stabilizing (BASS)

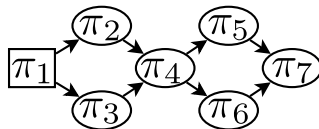
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



```
const id := 0,
var R,
repeat {
  R := 0
}.
```

```
const neighbors := ⟨πi, ...⟩,
const distance
:= min(distance(neighbors)) + 1,
const set := ⟨Rj, ...⟩ | ∀πj :
  (πj ∈ neighbors) ∧
  (distance(πj) = distance - 1),
var R,
repeat{
  ¬((∃Ri : πi ∈ set ∧ Ri = 2) xor
    ∃Ri : πi ∈ set ∧ Ri = 0))
  → R := 1;
  □∃Ri : πi ∈ set ∧ Ri = 0
  → R := 0;
  □∃Ri : πi ∈ set ∧ Ri = 2
  → R := 2
}.
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

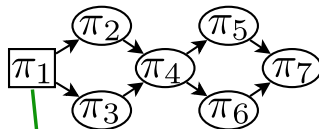
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



let's store  
the value 0

```
const id := 0,
var R,
repeat {
  R := 0
}.
```

```
const neighbors := ⟨πi, ...⟩,
const distance
:= min(distance(neighbors)) + 1,
const set := ⟨Rj, ...⟩ | ∀ πj :
  (πj ∈ neighbors) ∧
  (distance(πj) = distance - 1),
var R,
repeat {
  ¬((∃ Ri : πi ∈ set ∧ Ri = 2) xor
    ∃ Ri : πi ∈ set ∧ Ri = 0))
  → R := 1;
  □ ∃ Ri : πi ∈ set ∧ Ri = 0
  → R := 0;
  □ ∃ Ri : πi ∈ set ∧ Ri = 2
  → R := 2
}.
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

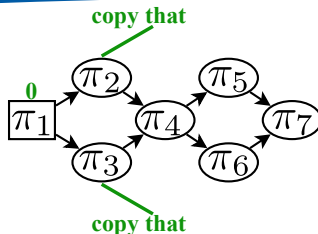
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



```
const id := 0,
var R,
repeat {
  R := 0
}
```

```
const neighbors := ⟨πi, ...⟩,
const distance
:= min(distance(neighbors)) + 1,
const set := ⟨Rj, ...⟩ | ∀ πj :
  (πj ∈ neighbors) ∧
  (distance(πj) = distance - 1),
var R,
repeat {
  ¬((∃ Ri : πi ∈ set ∧ Ri = 2) xor
    ∃ Ri : πi ∈ set ∧ Ri = 0)
  → R := 1;
  □ ∃ Ri : πi ∈ set ∧ Ri = 0
  → R := 0;
  □ ∃ Ri : πi ∈ set ∧ Ri = 2
  → R := 2
}
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

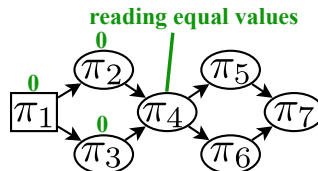
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



```
const id := 0,
var R,
repeat {
  R := 0
}
```

```
const neighbors := ⟨πi, ...⟩,
const distance
:= min(distance(neighbors)) + 1,
const set := ⟨Rj, ...⟩ | ∀πj :
  (πj ∈ neighbors) ∧
  (distance(πj) = distance - 1),
var R,
repeat {
  ¬((∃ Ri : πi ∈ set ∧ Ri = 2) xor
    ∃ Ri : πi ∈ set ∧ Ri = 0)
  → R := 1;
  □ ∃ Ri : πi ∈ set ∧ Ri = 0
  → R := 0;
  □ ∃ Ri : πi ∈ set ∧ Ri = 2
  → R := 2
}
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

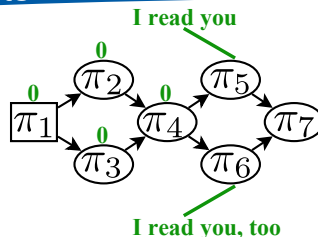
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



```
const id := 0,
var R,
repeat {
  R := 0
}
```

```
const neighbors := ⟨πi, ...⟩,
const distance
:= min(distance(neighbors)) + 1,
const set := ⟨Rj, ...⟩ | ∀πj :
  (πj ∈ neighbors) ∧
  (distance(πj) = distance - 1),
var R,
repeat {
  ¬((∃Ri : πi ∈ set ∧ Ri = 2) xor
    ∃Rj : πj ∈ set ∧ Rj = 0)
  → R := 1;
  □∃Ri : πi ∈ set ∧ Ri = 0
  → R := 0;
  □∃Rj : πj ∈ set ∧ Rj = 2
  → R := 2
}
```

Figure: Broadcast algorithm – self-stabilizing (BASS)



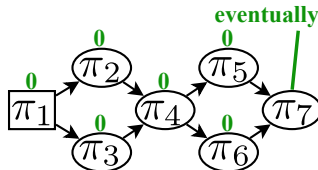
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



```
const id := 0,
var R,
repeat {
  R := 0
}
```

```
const neighbors := ⟨ $\pi_i, \dots$ ⟩,
const distance := min(distance(neighbors))+1,
const set := ⟨ $R_j, \dots$ ⟩ |  $\forall \pi_j :$ 
  ( $\pi_j \in \text{neighbors}$ )  $\wedge$ 
  (distance( $\pi_j$ ) = distance-1),
var R,
repeat{
   $\neg((\exists R_i : \pi_i \in \text{set} \wedge R_i = 2) \text{ xor } \exists R_i : \pi_i \in \text{set} \wedge R_i = 0))$ 
   $\rightarrow R := 1$ ;
   $\square \exists R_i : \pi_i \in \text{set} \wedge R_i = 0$ 
   $\rightarrow R := 0$ ;
   $\square \exists R_i : \pi_i \in \text{set} \wedge R_i = 2$ 
   $\rightarrow R := 2$ 
}
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

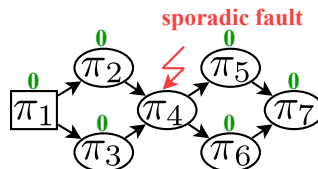
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



```
const id := 0,
var R,
repeat {
  R := 0
}
```

```
const neighbors := <pi_1, ...>,
const distance
:= min(distance(neighbors)) + 1,
const set := <R_j, ...> | pi_j :
  (pi_j in neighbors) ^
  (distance(pi_j) = distance - 1),
var R,
repeat {
  ¬((∃ R_i : pi_i in set ^ R_i = 2) xor
    ∃ R_i : pi_i in set ^ R_i = 0)
  → R := 1;
  □ ∃ R_i : pi_i in set ^ R_i = 0
  → R := 0;
  □ ∃ R_i : pi_i in set ^ R_i = 2
  → R := 2
}
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

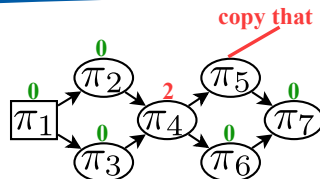
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



```
const id := 0,
var R,
repeat {
  R := 0
}
```

```
const neighbors := ⟨πi, ...⟩,
const distance
:= min(distance(neighbors)) + 1,
const set := ⟨Rj, ...⟩ | ∀ πj :
  (πj ∈ neighbors) ∧
  (distance(πj) = distance - 1),
var R,
repeat {
  ¬((∃ Ri : πi ∈ set ∧ Ri = 2) xor
    ∃ Ri : πi ∈ set ∧ Ri = 0)
  → R := 1;
  □ ∃ Ri : πi ∈ set ∧ Ri = 0
  → R := 0;
  □ ∃ Ri : πi ∈ set ∧ Ri = 2
  → R := 2
}
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

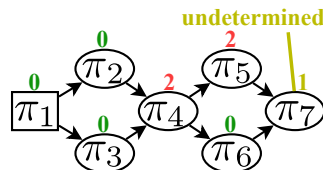
# System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).

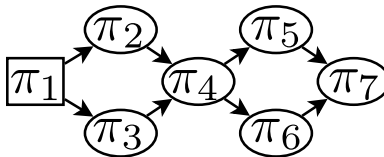


```
const id := 0,
var R,
repeat {
  R := 0
}
```

```
const neighbors := <pi_1, ...>,
const distance
:= min(distance(neighbors)) + 1,
const set := <R_j, ...> | forall pi_j :
  (pi_j in neighbors) ^
  (distance(pi_j) = distance - 1),
var R,
repeat {
  ~((exists R_i : pi_i in set ^ R_i = 2) xor
    exists R_i : pi_i in set ^ R_i = 0)
  -> R := 1;
  [] exists R_i : pi_i in set ^ R_i = 0
  -> R := 0;
  [] exists R_i : pi_i in set ^ R_i = 2
  -> R := 2
}
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

# Broadcast algorithm – self-stabilizing

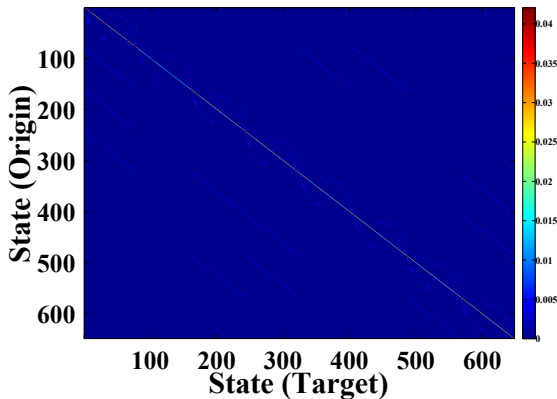


7 processes, 1 register each, 3 possible values<sup>1</sup>:  
here:  $|\mathcal{S}| = 2^3 \cdot 3^4 = 648$

---

<sup>1</sup>Processes  $\pi_1 - \pi_3$  cannot derive 1.

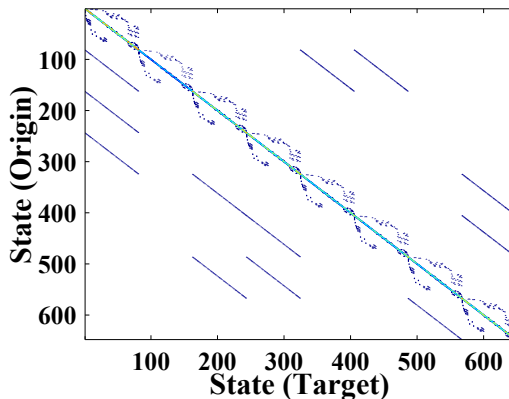
# Broadcast algorithm – self-stabilizing



7 processes, 1 register each, 3 possible values<sup>1</sup>:  
here:  $|\mathcal{S}| = 2^3 \cdot 3^4 = 648$

<sup>1</sup>Processes  $\pi_1 - \pi_3$  cannot derive 1.

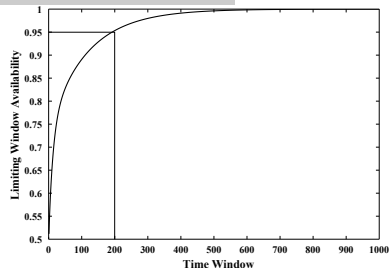
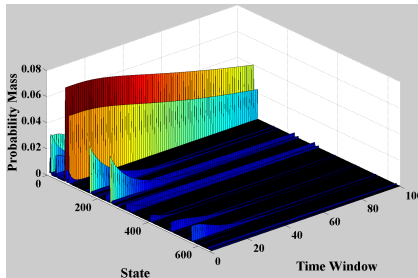
# Broadcast algorithm – self-stabilizing



7 processes, 1 register each, 3 possible values<sup>1</sup>:  
here:  $|\mathcal{S}| = 2^3 \cdot 3^4 = 648$

<sup>1</sup>Processes  $\pi_1 - \pi_3$  cannot derive 1.

# LWA Examples



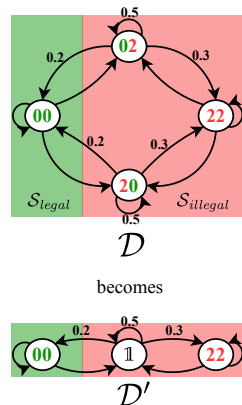


# Concluding LWA

- 😊 a measure to quantify recovery
- 😊 a method to compute that measure
- 😞 yet, inherently confined by state space explosion

# Lumping

- ▶ The first step in reducing the size of the state space is *lumping*.
- ▶ *Lumping* coalesces *bisimilar* states, i.e. states *that do the same* in a transition model



# Decomposition

- ▶ Lumping requires a transition model.
- ▶ But a transition model is likely too large to be constructed at one go.
- ▶ Idea: Successive construction of transition model.
  - ▶ Mutually independent processes  $\Rightarrow$  already discussed [Boudali et al., 2010, AINA2014]
  - ▶ Hierarchically structured  $\Rightarrow$  challenging, but feasible [WAINA2011, AINA2012, JCSS2013]

# Decomposition

- ▶ Lumping requires a transition model.
- ▶ But a transition model is likely too large to be constructed at one go.
- ▶ Idea: Successive construction of transition model.
  - ▶ Mutually independent processes  $\Rightarrow$  already discussed [Boudali et al., 2010, AINA2014]
  - ▶ Hierarchically structured  $\Rightarrow$  challenging, but feasible [WAINA2011, AINA2012, JCSS2013]

# Decomposition

- ▶ Lumping requires a transition model.
- ▶ But a transition model is likely too large to be constructed at one go.
- ▶ Idea: Successive construction of transition model.
  - ▶ Mutually independent processes  $\Rightarrow$  already discussed [Boudali et al., 2010, AINA2014]
  - ▶ Hierarchically structured  $\Rightarrow$  challenging, but feasible [WAINA2011, AINA2012, JCSS2013]

# Decomposition

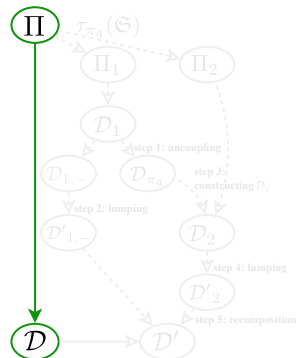
- ▶ Lumping requires a transition model.
- ▶ But a transition model is likely too large to be constructed at one go.
- ▶ Idea: Successive construction of transition model.
  - ▶ Mutually independent processes  $\Rightarrow$  already discussed [Boudali et al. , 2010, AINA2014]
  - ▶ Hierarchically structured  $\Rightarrow$  challenging, but feasible [WAINA2011, AINA2012, JCSS2013]

# Decomposition

- ▶ Lumping requires a transition model.
- ▶ But a transition model is likely too large to be constructed at one go.
- ▶ Idea: Successive construction of transition model.
  - ▶ Mutually independent processes  $\Rightarrow$  already discussed  
[Boudali et al. , 2010, AINA2014]
  - ▶ Hierarchically structured  $\Rightarrow$  challenging, but feasible  
[WAINA2011, AINA2012, JCSS2013]

# Tackling state space explosion

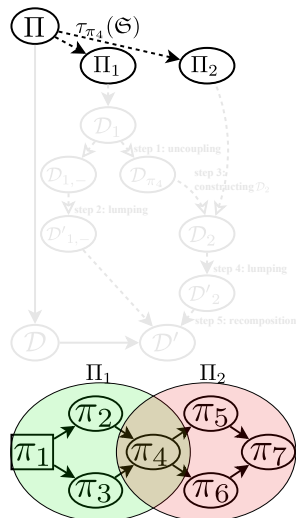
- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system  $\Pi_1$ .
- Uncouple gateway process, i.e.  $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$ .
- Lump  $\mathcal{D}_{1,-}$  to  $\mathcal{D}'_{1,-}$ .
- Build transition model of lower sub-system  $\Pi_2$ .
- Lump  $\mathcal{D}_2$  to  $\mathcal{D}'_2$ .
- Recompose  $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ .





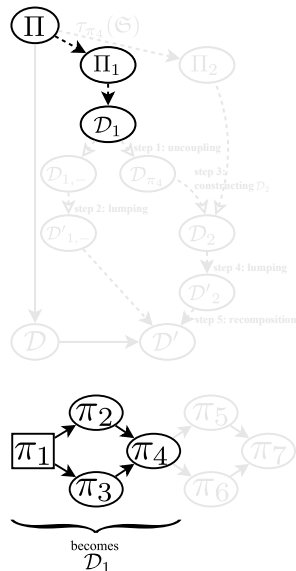
# Tackling state space explosion

- ▶ Ideally, tractable system size.
- ▶ If not, then slice system.
- ▶ Build transition model of upper sub-system  $\Pi_1$ .
- ▶ Uncouple gateway process, i.e.  $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$ .
- ▶ Lump  $\mathcal{D}_{1,-}$  to  $\mathcal{D}'_{1,-}$ .
- ▶ Build transition model of lower sub-system  $\Pi_2$ .
- ▶ Lump  $\mathcal{D}_2$  to  $\mathcal{D}'_2$ .
- ▶ Recompose  $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ .



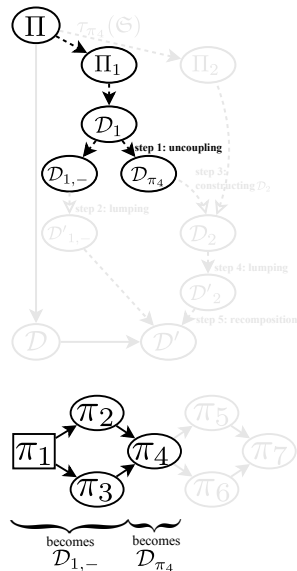
# Tackling state space explosion

- ▶ Ideally, tractable system size.
- ▶ If not, then slice system.
- ▶ **Build transition model of upper sub-system  $\Pi_1$ .**
- ▶ Uncouple gateway process, i.e.  $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$ .
- ▶ Lump  $\mathcal{D}_{1,-}$  to  $\mathcal{D}'_{1,-}$ .
- ▶ Build transition model of lower sub-system  $\Pi_2$ .
- ▶ Lump  $\mathcal{D}_2$  to  $\mathcal{D}'_2$ .
- ▶ Recompose  $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ .



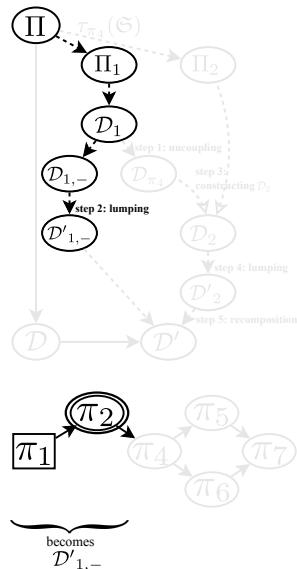
# Tackling state space explosion

- ▶ Ideally, tractable system size.
- ▶ If not, then slice system.
- ▶ Build transition model of upper sub-system  $\Pi_1$ .
- ▶ **Uncouple gateway process, i.e.  $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$ .**
- ▶ Lump  $\mathcal{D}_{1,-}$  to  $\mathcal{D}'_{1,-}$ .
- ▶ Build transition model of lower sub-system  $\Pi_2$ .
- ▶ Lump  $\mathcal{D}_2$  to  $\mathcal{D}'_2$ .
- ▶ Recompose  $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ .



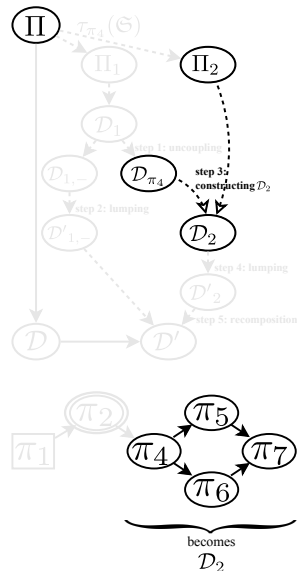
# Tackling state space explosion

- ▶ Ideally, tractable system size.
- ▶ If not, then slice system.
- ▶ Build transition model of upper sub-system  $\Pi_1$ .
- ▶ Uncouple gateway process, i.e.  $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$ .
- ▶ **Lump  $\mathcal{D}_{1,-}$  to  $\mathcal{D}'_{1,-}$ .**
- ▶ Build transition model of lower sub-system  $\Pi_2$ .
- ▶ Lump  $\mathcal{D}_2$  to  $\mathcal{D}'_2$ .
- ▶ Recompose  $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ .



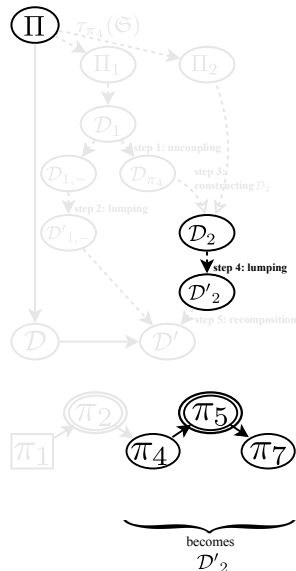
# Tackling state space explosion

- ▶ Ideally, tractable system size.
- ▶ If not, then slice system.
- ▶ Build transition model of upper sub-system  $\Pi_1$ .
- ▶ Uncouple gateway process, i.e.  $D_1 \rightarrow D_{1,-} \otimes D_{\pi_4}$ .
- ▶ Lump  $D_{1,-}$  to  $D'_{1,-}$ .
- ▶ Build transition model of lower sub-system  $\Pi_2$ .
- ▶ Lump  $D_2$  to  $D'_2$ .
- ▶ Recompose  $D' = D'_{1,-} \otimes D'_2$ .



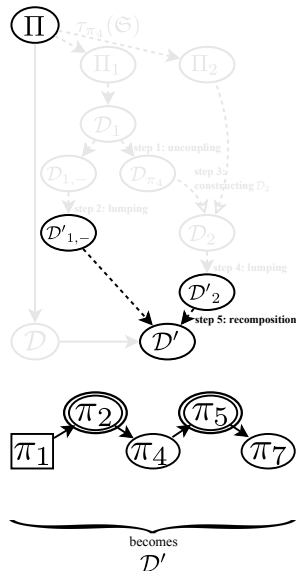
# Tackling state space explosion

- ▶ Ideally, tractable system size.
- ▶ If not, then slice system.
- ▶ Build transition model of upper sub-system  $\Pi_1$ .
- ▶ Uncouple gateway process, i.e.  $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$ .
- ▶ Lump  $\mathcal{D}_{1,-}$  to  $\mathcal{D}'_{1,-}$ .
- ▶ Build transition model of lower sub-system  $\Pi_2$ .
- ▶ Lump  $\mathcal{D}_2$  to  $\mathcal{D}'_2$ .
- ▶ Recompose  $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ .



# Tackling state space explosion

- ▶ Ideally, tractable system size.
- ▶ If not, then slice system.
- ▶ Build transition model of upper sub-system  $\Pi_1$ .
- ▶ Uncouple gateway process, i.e.  $D_1 \rightarrow D_{1,-} \otimes D_{\pi_4}$ .
- ▶ Lump  $D_{1,-}$  to  $D'_{1,-}$ .
- ▶ Build transition model of lower sub-system  $\Pi_2$ .
- ▶ Lump  $D_2$  to  $D'_2$ .
- ▶ Recompose  $D' = D'_{1,-} \otimes D'_2$ .



# Tackling state space explosion

- ▶ No state space larger than 81 states during computation:
- ▶  $|\mathcal{S}| = 648$  while  $|\mathcal{S}'| = 324$ , and
- ▶ only half the states and quarter the transitions!



# Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components  
⇒ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

# Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components  
⇒ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

# Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components  
⇒ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

# Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components  
⇒ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

# Conclusion

- ▶ Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- ▶ Goal: measure recovery
- ▶ Method: transition model analysis
- ▶ Challenge: state space explosion
- ▶ Solution: efficiently combining lumping and decomposition

# Conclusion

- ▶ Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- ▶ Goal: measure recovery
- ▶ Method: transition model analysis
- ▶ Challenge: state space explosion
- ▶ Solution: efficiently combining lumping and decomposition

# Conclusion

- ▶ Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- ▶ Goal: measure recovery
- ▶ Method: transition model analysis
- ▶ Challenge: state space explosion
- ▶ Solution: efficiently combining lumping and decomposition

# Conclusion

- ▶ Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- ▶ Goal: measure recovery
- ▶ Method: transition model analysis
- ▶ Challenge: state space explosion
- ▶ Solution: efficiently combining lumping and decomposition



# Conclusion

- ▶ Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- ▶ Goal: measure recovery
- ▶ Method: transition model analysis
- ▶ Challenge: state space explosion
- ▶ Solution: efficiently combining lumping and decomposition

# Summarizing achievements

- ▶ Recovery is an important attribute to quantify.
- ▶ Lumping and decomposition are helpful assets from model checking,
- ▶ but required to be adapted in that matter.
- ▶ Methods have been successfully demonstrated on numerous examples:
  - ▶ traffic lights,
  - ▶ broadcast algorithm [AINA2012, JCSS2013],
  - ▶ power grids [WAINA2014] and
  - ▶ wireless sensor network [AINA2014].

# Summarizing achievements

- ▶ Recovery is an important attribute to quantify.
- ▶ Lumping and decomposition are helpful assets from model checking,
- ▶ but required to be adapted in that matter.
- ▶ Methods have been successfully demonstrated on numerous examples:
  - ▶ traffic lights,
  - ▶ broadcast algorithm [AINA2012, JCSS2013],
  - ▶ power grids [WAINA2014] and
  - ▶ wireless sensor network [AINA2014].

# Summarizing achievements

- ▶ Recovery is an important attribute to quantify.
- ▶ Lumping and decomposition are helpful assets from model checking,
- ▶ but required to be adapted in that matter.
- ▶ Methods have been successfully demonstrated on numerous examples:
  - ▶ traffic lights,
  - ▶ broadcast algorithm [AINA2012, JCSS2013],
  - ▶ power grids [WAINA2014] and
  - ▶ wireless sensor network [AINA2014].

# Summarizing achievements

- ▶ Recovery is an important attribute to quantify.
- ▶ Lumping and decomposition are helpful assets from model checking,
- ▶ but required to be adapted in that matter.
- ▶ Methods have been successfully demonstrated on numerous examples:
  - ▶ traffic lights,
  - ▶ broadcast algorithm [AINA2012,JCSS2013],
  - ▶ power grids [WAINA2014] and
  - ▶ wireless sensor network [AINA2014].

# Own publications

AnSS2008

Nils Müllner, Abhishek Dhama, and Oliver Theel. *Derivation of Fault Tolerance Measures of Self-Stabilizing Algorithms by Simulation*. In Proceedings of the 41st Annual Symposium on Simulation (AnSS2008), pages 183 – 192, Ottawa, ON, Canada, April 2008. IEEE Computer Society Press.

WAINA2011

Nils Müllner and Oliver Theel. *The Degree of Masking Fault Tolerance vs. Temporal Redundancy*. In Proceedings of the 25th IEEE of the International Conference on Advanced Information Networking and Applications Workshops (WAINA2011), Track "The Seventh International Symposium on Frontiers of Information Systems and Network Applications (FINA2011)", pages 21 – 28, Biopolis, Singapore, 2011. IEEE Computer Society Press.

JCSS2013

Nils Müllner, Oliver Theel, and Martin Fränzle. *Combining Decomposition and Reduction for the State Space Analysis of Self-Stabilizing Systems*. In Journal of Computer and System Sciences (JCSS), volume 79, pages 1113 – 1125. Elsevier Science Publishers B. V., November 2013. The paper is an extended version of AINA2012.

AINA2014

Nils Müllner, Oliver Theel, and Martin Fränzle. *Combining Decomposition and Lumping to Evaluate Semi-hierarchical Systems*. In Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications (AINA2014), Victoria, BC, Canada, 2014. accepted for publication.

ATC2009

Nils Müllner, Abhishek Dhama, and Oliver Theel. *Deriving a Good Trade-off Between System Availability and Time Redundancy*. In Proceedings of the Symposia and Workshops on Ubiquitous, Automatic and Trusted Computing, number E3737 in Track "International Symposium on UbiCom Frontiers - Innovative Research, Systems and Technologies (Ufirst-09)", pages 61 – 67, Brisbane, QLD, Australia, July 2009. IEEE Computer Society Press.

AINA2012

Nils Müllner, Oliver Theel, and Martin Fränzle. *Combining Decomposition and Reduction for State Space Analysis of a Self-Stabilizing System*. In Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA2012), pages 936 – 943, Fukuoka-shi, Fukuoka, Japan, March 2012. IEEE Computer Society Press. **Best Paper Award**.

IREP2013

Maryam Kamgarpour, Christian Ellen, Sadegh Esmaeil Zadeh Soudjani, Sebastian Gerwinn, Johanna L. Mathieux, Nils Müllner, Alessandro Abate, Duncan S. Callaway, Martin Fränzle, and John Lygeros. *Modeling Options for Demand Side Participation of Thermostatically Controlled Loads*. In Proceedings of the IREP Symposium-Bulk Power System Dynamics and Control -IX (IREP), August 25-30, 2013, Rethymnon, Greece, 2013.

WAINA2014

Nils Müllner, Oliver Theel, and Martin Fränzle. *Composing Thermostatically Controlled Loads to Determine the Reliability against Blackouts*. In Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA 2014), Victoria, BC, Canada, 2014. accepted for publication.



Questions?

**SSS2006**

Abhishek Dhama, Oliver Theel, and Timo Warns. *Reliability and Availability Analysis of Self-Stabilizing Systems*. In Proceedings of the Eighth International Conference on Stabilization, Safety, and Security of Distributed Systems (SSS2006), pages 244 – 261, 2006.

**Boudali et al., 2010**

Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. *A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis*. IEEE Trans. Dependable Sec. Comput., 7(2):128 – 143, 2010.