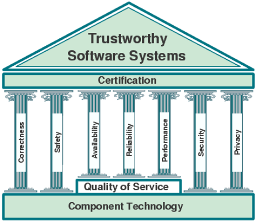
# Unmasking Fault Tolerance

## Masking vs. Non-masking Fault-tolerant Systems

Nils Müllner

University of Oldenburg - Graduiertenkolleg TrustSoft

7. Juli 2010



# Contents

# Redundancy
## Spatial and Temporal

- Fault tolerance demands redundancy

# Redundancy
## Spatial and Temporal

- Fault tolerance demands redundancy
  - either spatial redundancy (coding theory)

# Redundancy
## Spatial and Temporal

- Fault tolerance demands redundancy
  - either spatial redundancy (coding theory)
  - or temporal redundancy (re-requests)

# Redundancy
## Spatial and Temporal

- Fault tolerance demands redundancy
  - either spatial redundancy (coding theory)
  - or temporal redundancy (re-requests)
  - or a mix (re-requests with error detection)

# Redundancy

## Spatial and Temporal

- Fault tolerance demands redundancy
  - either spatial redundancy (coding theory)
  - or temporal redundancy (re-requests)
  - or a mix (re-requests with error detection)
- Coding theory already widely discussed

# Redundancy

## Spatial and Temporal

- Fault tolerance demands redundancy
  - either spatial redundancy (coding theory)
  - or temporal redundancy (re-requests)
  - or a mix (re-requests with error detection)
- Coding theory already widely discussed
- Temporal redundancy and combination in current focus

# Liveness and Safety

| | safe | $\neg$ safe |
|---|---|---|
| live | masking | non-masking |
| $\neg$ live | failsafe | |

# Liveness and Safety

| | safe | ¬ safe |
|---|---|---|
| live | masking | non-masking |
| ¬ live | failsafe | |

- Focus on live systems, so liveness is not an issue here

# Liveness and Safety

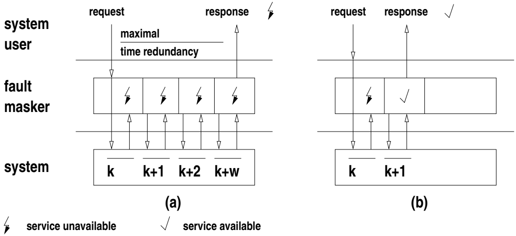
| | safe | $\neg$ safe |
|---|---|---|
| live | masking | non-masking |
| $\neg$ live | failsafe | |

- Focus on live systems, so liveness is not an issue here
- Safety is an issue

# Liveness and Safety

| | safe | ¬ safe |
|---|---|---|
| live | masking | non-masking |
| ¬ live | failsafe | |

- Focus on live systems, so liveness is not an issue here
- Safety is an issue
- Focus: systems that are always live, but not always safe!



# Unmasking Fault Tolerance

fault tolerance

# Unmasking Fault Tolerance

masking nonmasking fault tolerance

# Unmasking Fault Tolerance

Between masking nonmasking fault tolerance

# Unmasking Fault Tolerance

Between masking $\Rightarrow$ nonmasking fault tolerance

# Unmasking Fault Tolerance

Between masking $\Longleftarrow\!\!\Longrightarrow$ nonmasking fault tolerance

# Unmasking Fault Tolerance

Between masking $\Longleftarrow\!\!\Longrightarrow$ nonmasking fault tolerance

The degree of fault masking is a desired quantity that costs.

# Unmasking Fault Tolerance

Between masking $\Longleftarrow\!\!\Longrightarrow$ nonmasking fault tolerance

The degree of fault masking is a desired quantity that costs.

*Unmasking* here is to find out:

# Unmasking Fault Tolerance

Between masking $\Longleftrightarrow$ nonmasking fault tolerance

The degree of fault masking is a desired quantity that costs.

*Unmasking* here is to find out:

- What trade-off solutions are possible?

# Unmasking Fault Tolerance

Between masking $\Longleftarrow\!\!\Longrightarrow$ nonmasking fault tolerance

The degree of fault masking is a desired quantity that costs.

*Unmasking* here is to find out:

- What trade-off solutions are possible?
- We must calculate one to show how much we pay for which degree of masking fault tolerance

# Unmasking Fault Tolerance

Between masking $\Longleftarrow\!\!\Longrightarrow$ nonmasking fault tolerance

The degree of fault masking is a desired quantity that costs.

*Unmasking* here is to find out:

- What trade-off solutions are possible?
- We must calculate one to show how much we pay for which degree of masking fault tolerance
- Which of them are *favorable* (Pareto optimal)?

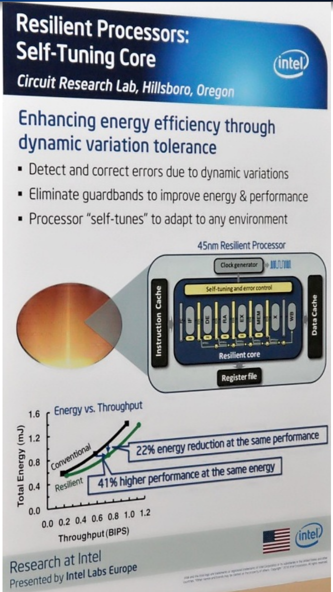
# Palisades as an Example for Cost-Benefit Ratio

# Palisades as an Example for Cost-Benefit Ratio

- Intel developing *Palisades*

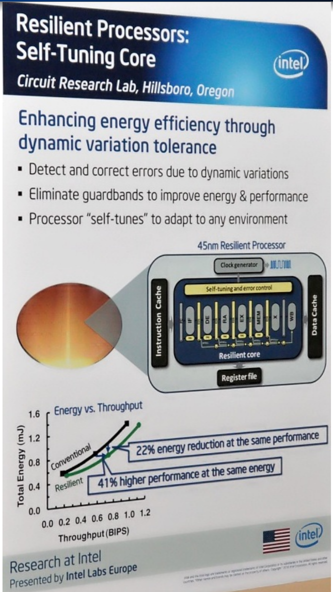
# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC



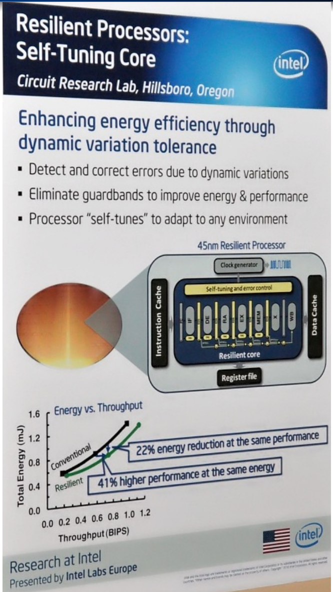
# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC
- Capable to handle up to 8 million additional faults per second



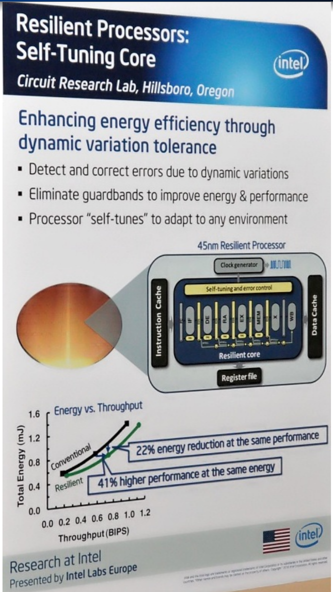
# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC
- Capable to handle up to 8 million additional faults per second
- Fault frequency increases by



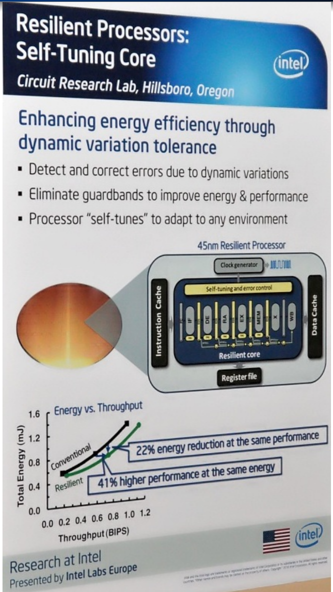
# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC
- Capable to handle up to 8 million additional faults per second
- Fault frequency increases by
  - undervolting



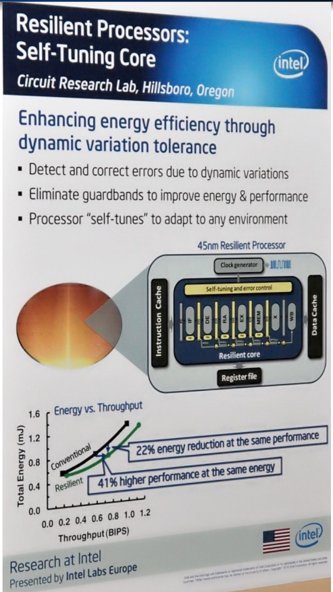
# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC
- Capable to handle up to 8 million additional faults per second
- Fault frequency increases by
  - undervolting
  - overvolting



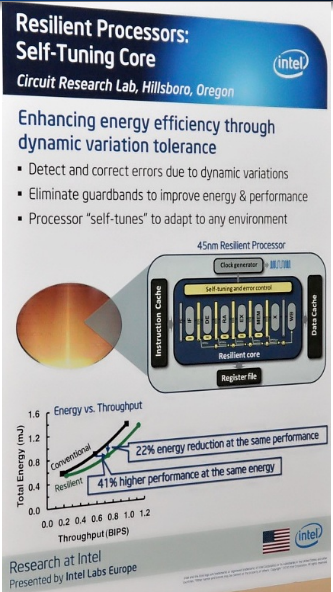
# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC
- Capable to handle up to 8 million additional faults per second
- Fault frequency increases by
  - undervolting
  - overvolting
- either save energy or increase



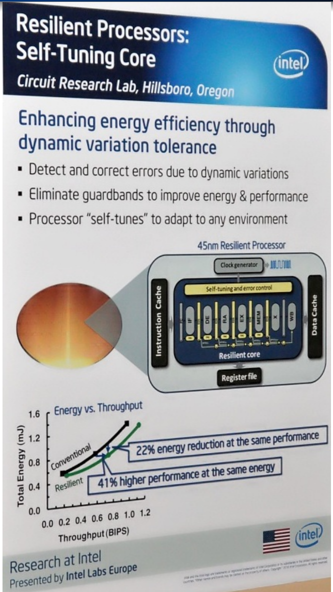
# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC
- Capable to handle up to 8 million additional faults per second
- Fault frequency increases by
  - undervolting
  - overvolting
- either save energy or increase
- cheap



# Palisades as an Example for Cost-Benefit Ratio

- 1.4GHz CPU gets additional EDC
- Capable to handle up to 8 million additional faults per second
- Fault frequency increases by
  - undervolting
  - overvolting
- either save energy or increase
- cheap
- early stage already implemented in Core i5 and Core i7



# Palisades as an Example for Cost-Benefit Ratio

- Intel developing *Palisades*
- Measure of cost: time, energy, …

# Palisades as an Example for Cost-Benefit Ratio

- Intel developing *Palisades*
- Measure of cost: time, energy, ...
- Measure of quality: availability

# Palisades as an Example for Cost-Benefit Ratio

- Intel developing *Palisades*
- Measure of cost: time, energy, ...
- Measure of quality: availability
- Basically, wherever certain classes of faults occur, liveness is guaranteed and masking of faults costs something

# System Paramters and the Degree of Masking

- We look at *parametric systems*

# System Paramters and the Degree of Masking

- We look at *parametric systems*
- What parameter values give the best trade-off?

# System Paramters and the Degree of Masking

- We look at *parametric systems*
- What parameter values give the best trade-off?
- Each possible system configuration has a certain degree of masking

# System Paramters and the Degree of Masking

- We look at *parametric systems*
- What parameter values give the best trade-off?
- Each possible system configuration has a certain degree of masking
- How to compute the degree of masking for a system configuration?

# Limiting (Window) Availability

## Definition

*Limiting Availability* (or Steady State Availability) is the probability, that the system satisfies its safety and liveness predicate as $t$ approaches infinity $A = \lim_{t \to \infty} A_t$.

# Limiting (Window) Availability

## Definition

*Limiting Availability* (or Steady State Availability) is the probability, that the system satisfies its safety and liveness predicate as $t$ approaches infinity $A = \lim_{t \to \infty} A_t$.

## Definition

*Limiting Window Availability* ($l_i$) is the limiting probability that a system will have satisfied its safety and liveness predicates at least once within $i + 1$ calculation steps.

# Limiting (Window) Availability

## Definition

*Limiting Availability* (or Steady State Availability) is the probability, that the system satisfies its safety and liveness predicate as $t$ approaches infinity $A = \lim_{t \to \infty} A_t$.

## Definition

*Limiting Window Availability* ($l_i$) is the limiting probability that a system will have satisfied its safety and liveness predicates at least once within $i + 1$ calculation steps.

$$l_i = \lim_{t \to \infty} \sum_{j=t}^{t+i} p(\forall k, 0 \leq k < j : c_k \not\models \mathscr{P} \land c_j \models \mathscr{P})$$

# Limiting (Window) Availability

## Definition

*Limiting Availability* (or Steady State Availability) is the probability, that the system satisfies its safety and liveness predicate as $t$ approaches infinity $A = \lim_{t \to \infty} A_t$.

## Definition

*Limiting Window Availability* ($l_i$) is the limiting probability that a system will have satisfied its safety and liveness predicates at least once within $i + 1$ calculation steps.

$$l_i = \lim_{t \to \infty} \sum_{j=t}^{t+i} p(\forall k, 0 \leq k < j : c_k \not\models \mathscr{P} \wedge c_j \models \mathscr{P})$$

## Definition

*Limiting Window Availability Sequence (LWAS)* is the infinite sequence of limiting window availabilities $LWAS = \langle l_0, l_1, \ldots \rangle$.

# System Definition (Ingredients)

- System structure (processes and communication channels)
- Communication model (shared memory or message passing)
- Variable domains
- Algorithm
- Scheduler
- Fault model

# Self Stabilization

## Definition

A system is self-stabilizing if and only if:

Dolev, Shlomi (2000), Self-Stabilization, MIT Press, ISBN 0-262-04178-2.

# Self Stabilization

## Definition

A system is self-stabilizing if and only if:

1. Starting from any state, it is guaranteed that the system will eventually reach a state that satisfies the safety predicate(*convergence*).

Dolev, Shlomi (2000), Self-Stabilization, MIT Press, ISBN 0-262-04178-2.

# Self Stabilization

## Definition

A system is self-stabilizing if and only if:

1. Starting from any state, it is guaranteed that the system will eventually reach a state that satisfies the safety predicate(*convergence*).
2. Given that the system satisfies the safety predicate, it is guaranteed to stay in a state that satisfies the safety predicate, provided that no fault happens (*closure*).

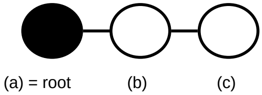
Dolev, Shlomi (2000), Self-Stabilization, MIT Press, ISBN 0-262-04178-2.

# Example 1/8

- Three process system

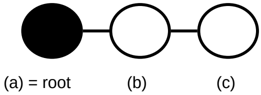
# Example 1/8

- Three process system



# Example 1/8

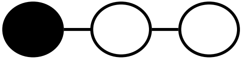
- Three process system




- Equi-probabilistic scheduler electing one process per cycle

# Example 1/8

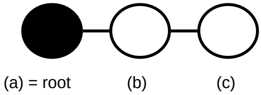
- Three process system




- Equi-probabilistic scheduler electing one process per cycle
- Three values (*true*, *false* and *dk* (don't know))

# Example 1/8

- Three process system




- Equi-probabilistic scheduler electing one process per cycle
- Three values (*true*, *false* and *dk* (don't know))
- Fault model: transient faults with probability $q = 0.01$
- Simple broadcast algorithm

# Example 2/8 : the Root Algorithm

Repeat

  $true \rightarrow reg := true$

end.

# Example 3/8 : the Non-Root Algorithm

define $vector := \{reg_i \mid proc_i \in neighbors\}$

Repeat

$\quad \neg((false \in vector) xor (true \in vector)) \rightarrow$

$\quad reg := dk$

$\quad \Box((false \in vector) \wedge \neg(true \in vector)) \rightarrow$

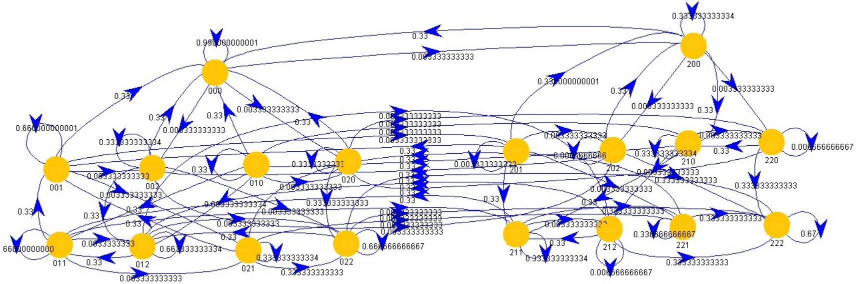
$\quad reg := false$

$\quad \Box((true \in vector) \wedge \neg(false \in vector)) \rightarrow$

$\quad reg := true$

end.

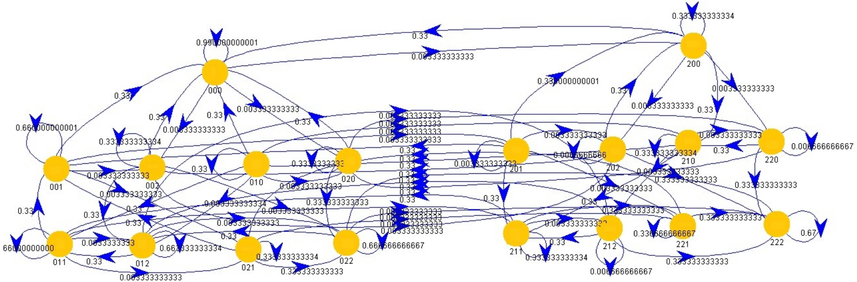
# Example 4/8 : State Space and Markov Chain




0 = *true*, 1 = *dk*, 2 = *false*

Markov Chain made with tool *jAndrej* by Fabian Grüning

# Example 4/8 : State Space and Markov Chain

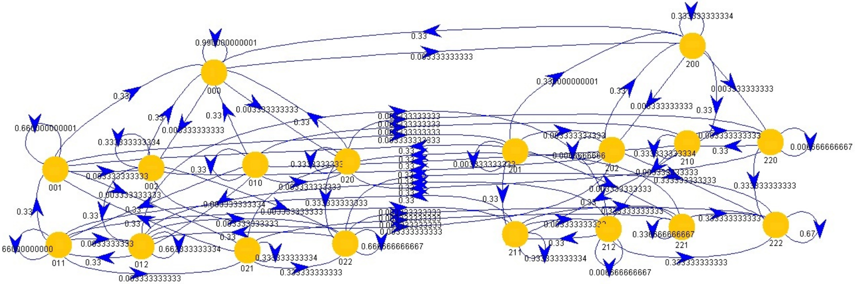



$0 = true$, $1 = dk$, $2 = false$

Markov Chain made with tool *jAndrej* by Fabian Grüning

Now we can calculate the steady state probability distribution

# Example 4/8 : State Space and Markov Chain

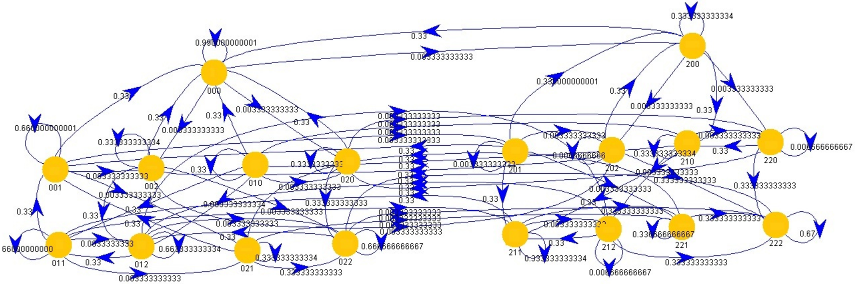



$0 = \textit{true}$, $1 = \textit{dk}$, $2 = \textit{false}$

Markov Chain made with tool *jAndrej* by Fabian Grüning

Now we can calculate the steady state probability distribution and the limiting availability

# Example 4/8 : State Space and Markov Chain




$0 = true$, $1 = dk$, $2 = false$

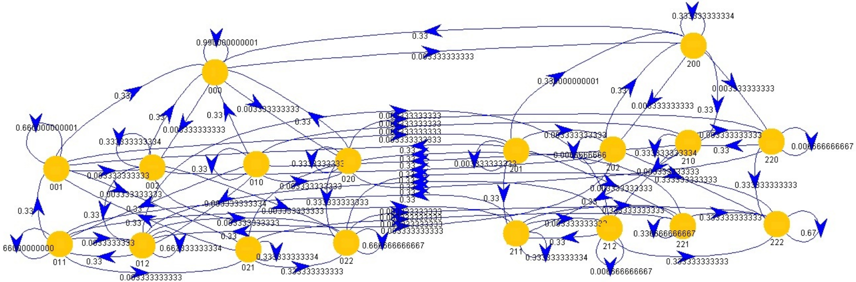
Markov Chain made with tool *jAndrej* by Fabian Grüning

Now we can calculate the steady state probability distribution

and the limiting availability

and with a small alteration the *Limiting Window Availability*.

# Example 4/8 : State Space and Markov Chain




## Definition

*Limiting Window Availability* ($l_i$) is the limiting probability that a system will have satisfied its safety and liveness predicates at least once within $i+1$ calculation steps.

$$l_i = \lim_{t\to\infty} \sum_{j=t}^{t+i} p(\forall k, 0 \leq k < j : c_k \not\models \mathscr{P} \wedge c_j \models \mathscr{P})$$

# Example 5/8 : Using the Markov Chain to Calculate the *LTR*

Just five small steps to go...

# Example 5/8 : Using the Markov Chain to Calculate the *LTR*

Just five small steps to go...

1. Calculate the steady state probability distribution

# Example 5/8 : Using the Markov Chain to Calculate the *LTR*

Just five small steps to go...

1. Calculate the steady state probability distribution
2. Erase all transitions originating from state $\langle 0,0,0 \rangle$ and
3. add transition $p((\langle 0,0,0 \rangle, \langle 0,0,0 \rangle)) = 1$
4. to take care of *„a system will have satisfied $\mathscr{P}$ within i calculation steps“*

# Example 5/8 : Using the Markov Chain to Calculate the *LTR*

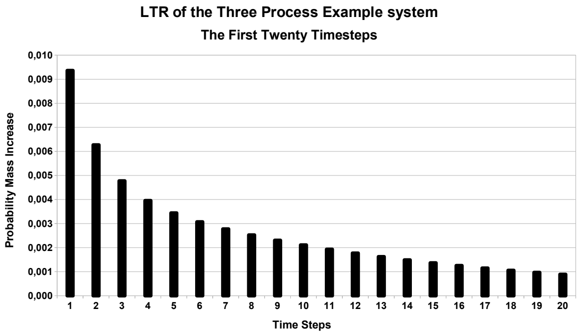
Just five small steps to go...

1. Calculate the steady state probability distribution
2. Erase all transitions originating from state $\langle 0,0,0\rangle$ and
3. add transition $p((\langle 0,0,0\rangle,\langle 0,0,0\rangle)) = 1$
4. to take care of *„a system will have satisfied $\mathscr{P}$ within i calculation steps“*
5. Calculate the probability distribution for each time step
   while the initial probability distribution is given by the former steady state distribution

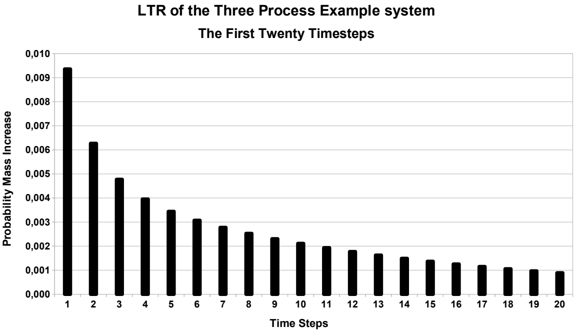
# Example 6/8 : *LWAS*

| $\downarrow$timestep/state$\rightarrow$ | $\langle 0,0,0 \rangle$ |
|---|---|
| 0 | 0.935981 |
| 1 | 0.945341 |
| 2 | 0.951612 |
| 3 | 0.956386 |
| 4 | 0.960342 |
| 5 | 0.963783 |
| 6 | 0.966854 |
| 7 | 0.969629 |
| 8 | 0.972154 |
| 9 | 0.974459 |
| 10 | 0.976569 |
| 11 | 0.978501 |
| ... | ... |

# Example 7/8 : The Relative Increase of Availability over Time

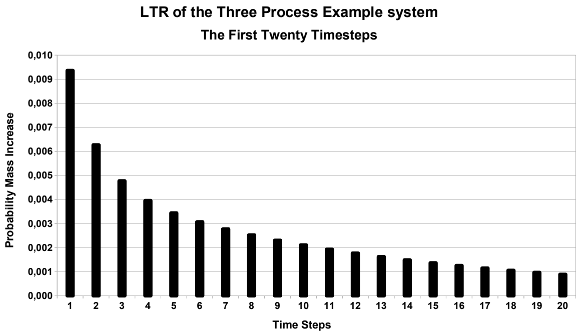


# Example 7/8 : The Relative Increase of Availability over Time



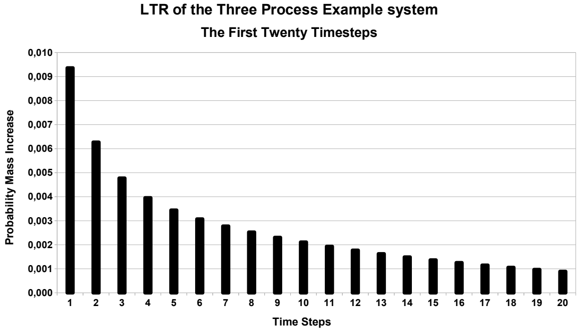

- The Increase of probability that the system has satisfied the safety predicate within *i* steps

# Example 7/8 : The Relative Increase of Availability over Time



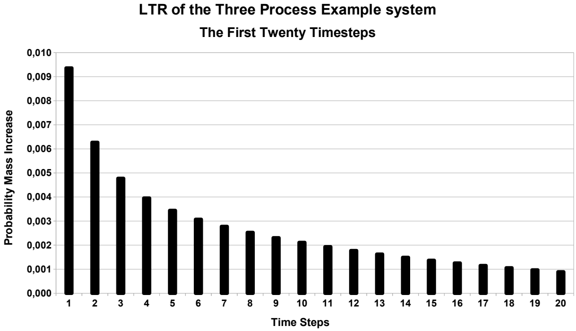

- The Increase of probability that the system has satisfied the safety predicate within *i* steps
- How long would you wait for a system to be *up* ($c \models \mathscr{P}$) again?

# Example 7/8 : The Relative Increase of Availability over Time



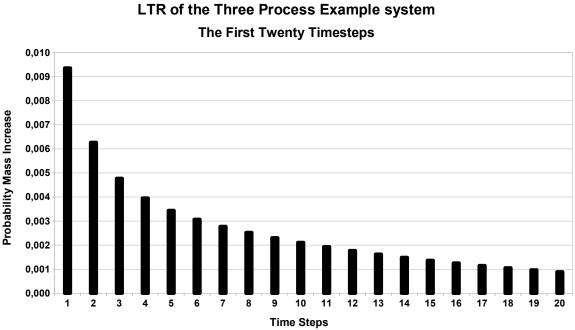

- The Increase of probability that the system has satisfied the safety predicate within *i* steps
- How long would you wait for a system to be *up* ($c \models \mathscr{P}$) again?
  - Minimum availability reached?

# Example 7/8 : The Relative Increase of Availability over Time



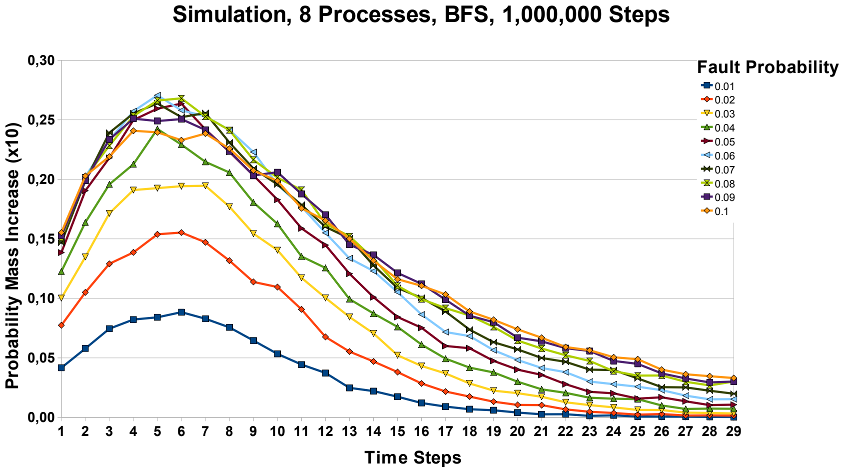

- The Increase of probability that the system has satisfied the safety predicate within *i* steps
- How long would you wait for a system to be *up* ($c \models \mathscr{P}$) again?
  - Minimum availability reached?
  - Until increase is too low?

# Example 7/8 : The Relative Increase of Availability over Time




- The Increase of probability that the system has satisfied the safety predicate within *i* steps
- How long would you wait for a system to be *up* ($c \models \mathscr{P}$) again?
  - Minimum availability reached?
  - Until increase is too low?
  - Can systems have significant spots?

# Example 8/8 : Example System with *Significant Spot*



# Markov Chain Abstraction and Decomposition

- We have the *LWAS*

# Markov Chain Abstraction and Decomposition

- We have the *LWAS*
- We can decrease the level of detail of the Markov Chain to better recognize contexts (abstraction, cf. [Kli10])

# Markov Chain Abstraction and Decomposition

- We have the *LWAS*
- We can decrease the level of detail of the Markov Chain to better recognize contexts (abstraction, cf. [Kli10])
- We can try to cope with large systems that suffer from state space explosion (decomposition, cf. [Mal93])

# Abstraction of Markov Chains 1/3

- System Definition
- State Space
- Build Markov Chain
- Abstract Markov Chain
- Analyze the *LWAS*

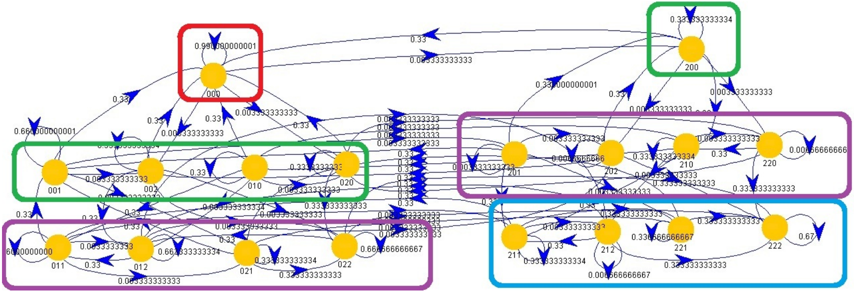
# Abstraction of Markov Chains 2/3

- Combine states that *have something in common* to subsets like. . .

# Abstraction of Markov Chains 2/3

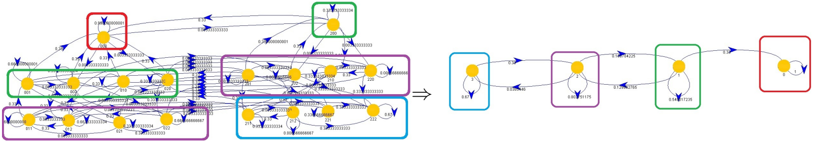
- Combine states that *have something in common* to subsets like…
- …the number of *correct* processes

# Abstraction of Markov Chains 3/3




$$p(v_i, w_i) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} p(v_i, w_j) \cdot p(v_i)}{\sum_{i=0}^{n} p(v_i)} \quad (1)$$

# Abstraction of Markov Chains 3/3



| ↓timestep/state→ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0.935981 | 0.028363 | 0.032848 | 0.002808 |
| 1 | 0.945341 | 0.019003 | 0.032848 | 0.002808 |
| 2 | 0.951612 | 0.014466 | 0.031115 | 0.002808 |
| 3 | 0.956386 | 0.011989 | 0.028867 | 0.002759 |
| 4 | 0.960342 | 0.010429 | 0.026567 | 0.002663 |
| 5 | 0.963783 | 0.009304 | 0.024379 | 0.002533 |
| 6 | 0.966854 | 0.008409 | 0.022352 | 0.002385 |

Table: Probability Mass Distribution over Time (0 Column Equals *LWAS*)

# Decomposition of Markov Chains

- If the system is too large to handle

# Decomposition of Markov Chains

- If the system is too large to handle
- split it into managable subsystems,

# Decomposition of Markov Chains

- If the system is too large to handle
- split it into managable subsystems,
- calculate each subsystems *LWAS* and

# Decomposition of Markov Chains

- If the system is too large to handle
- split it into managable subsystems,
- calculate each subsystems *LWAS* and
- take propagation between subsystems into account.
- But how?

# Decomposition of Markov Chains

- Each subsystem gets input (and sometimes faults) propagated from neighbors

# Decomposition of Markov Chains

- Each subsystem gets input (and sometimes faults) propagated from neighbors
- Each possible input from that neighbor has a probability to occur

# Decomposition of Markov Chains

- Each subsystem gets input (and sometimes faults) propagated from neighbors
- Each possible input from that neighbor has a probability to occur
- Case distinction: for each possible input, how would the subsystem behave?

# Decomposition of Markov Chains

- Each subsystem gets input (and sometimes faults) propagated from neighbors
- Each possible input from that neighbor has a probability to occur
- Case distinction: for each possible input, how would the subsystem behave?
- Build appropriate Markov chain

# Decomposition of Markov Chains

- Each subsystem gets input (and sometimes faults) propagated from neighbors
- Each possible input from that neighbor has a probability to occur
- Case distinction: for each possible input, how would the subsystem behave?
- Build appropriate Markov chain
- According to input probability link all these Markov chains

# Challenges

- Accuracy issues!

# Challenges

- Accuracy issues!
- Cyclic dependencies

# Progress So Far

- Redundancy in time/space

# Progress So Far

- Redundancy in time/space
- Unmasking fault tolerance

# Progress So Far

- Redundancy in time/space
- Unmasking fault tolerance
- Metric: *LWAS*

# Progress So Far

- Redundancy in time/space
- Unmasking fault tolerance
- Metric: *LWAS*
- Calculation of *LWAS*

# Progress So Far

- Redundancy in time/space
- Unmasking fault tolerance
- Metric: *LWAS*
- Calculation of *LWAS*
- Markov chain abstraction

# Progress So Far

- Redundancy in time/space
- Unmasking fault tolerance
- Metric: *LWAS*
- Calculation of *LWAS*
- Markov chain abstraction
- Markov chain decomposition

# ToDo List

- Complete Decomposition

# ToDo List

- Complete Decomposition
- Build framework that automatically finds set of favorable trade-offs

Questions?

[Mal93] Manish Malhotra.
*Specification and Solution of Dependability Models of Fault-tolerant Systems*.
PhD thesis, Durham, NC, USA, 1993.

[Kli10] Daniel Klink.
*Three-valued Abstraction for Stochastic Systems*.
PhD thesis, RWTH Aachen, Mar 2010.

[MDT09] 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 Ubiquitious, Automatic and Trusted Computing*, number E3737, pages 61–67.
IEEE Computer Society Press, July 2009.