A Compositional Approach to Coordinated Software Rejuvenation of Component-Based Systems

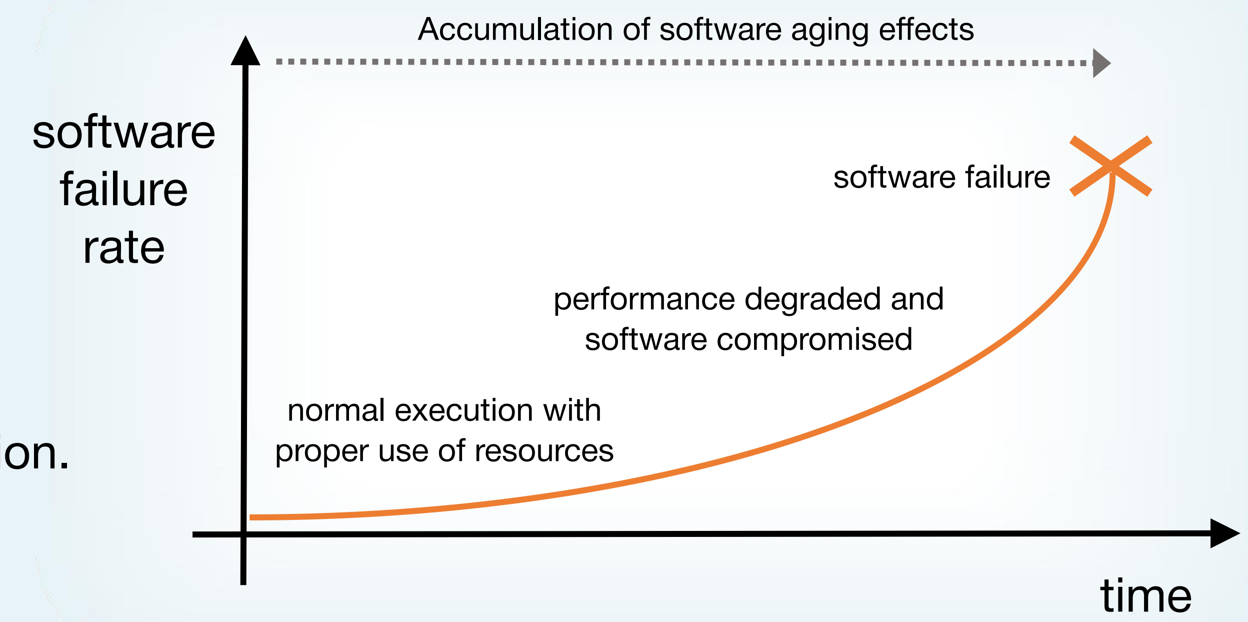
Leonardo Paroli Tommaso Botarelli
Laura Carnevali Enrico Vicario

Software Technologies Lab,
University of Florence, Italy

- Use of micro-rejuvenation strategies in component-based systems to mitigate software aging.
- Modeling the system as a static fault tree and components as MRGPs for compositional unavailability analysis.
- Minimizing unavailability caused by micro-rejuvenation through coordination of components in offsetting first rejuvenations.
- Definition of a complementary macro-rejuvenation policy to mititgate synchronization loss over time.

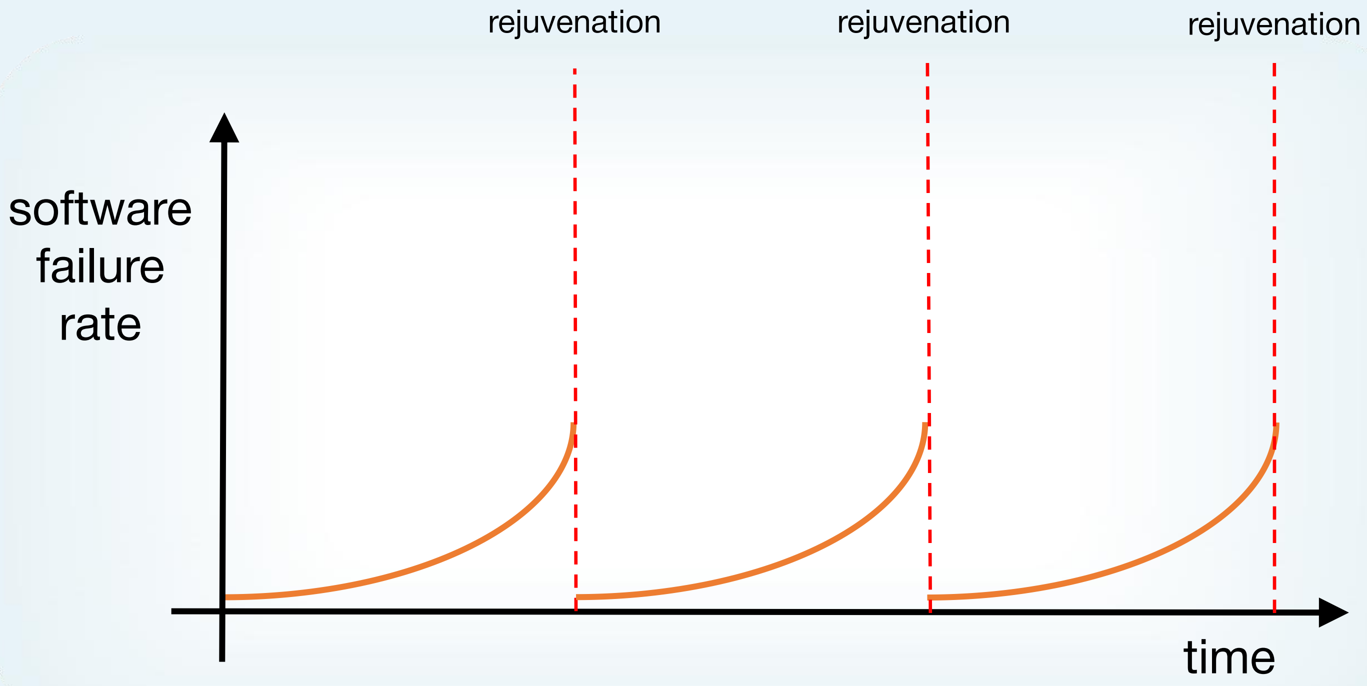
What is software aging?

- **Software aging**: accumulation of errors.
- Recovering a system after a failure is costly.
- Problem mitigation: redundancies and synchronization.



- Software failure rate.

What is software rejuvenation?

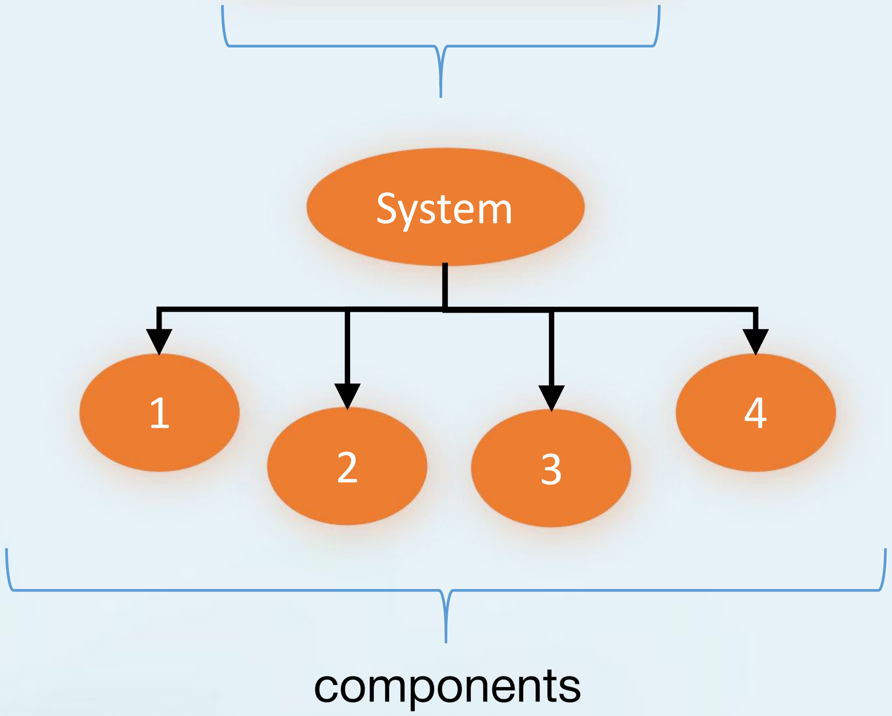


- Software failure rate, with rejuvenation.

- Proactive maintenance like **rejuvenation** (or regeneration) mitigates software aging.
- Rejuvenation is cheaper than recovery.
- It introduces more complexity.

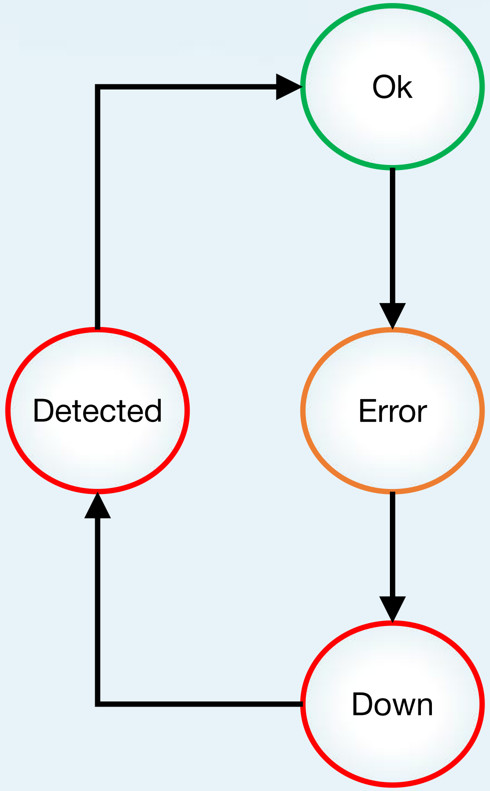
Towards the system model

- Predictive analysis of complex systems is often unfeasible due to the dimension of the resulting model.
- Number of interactions, functionalities and inner mechanisms are often the cause.
- We consider systems that can be naturally decomposed into independent sub-components.



Component Model

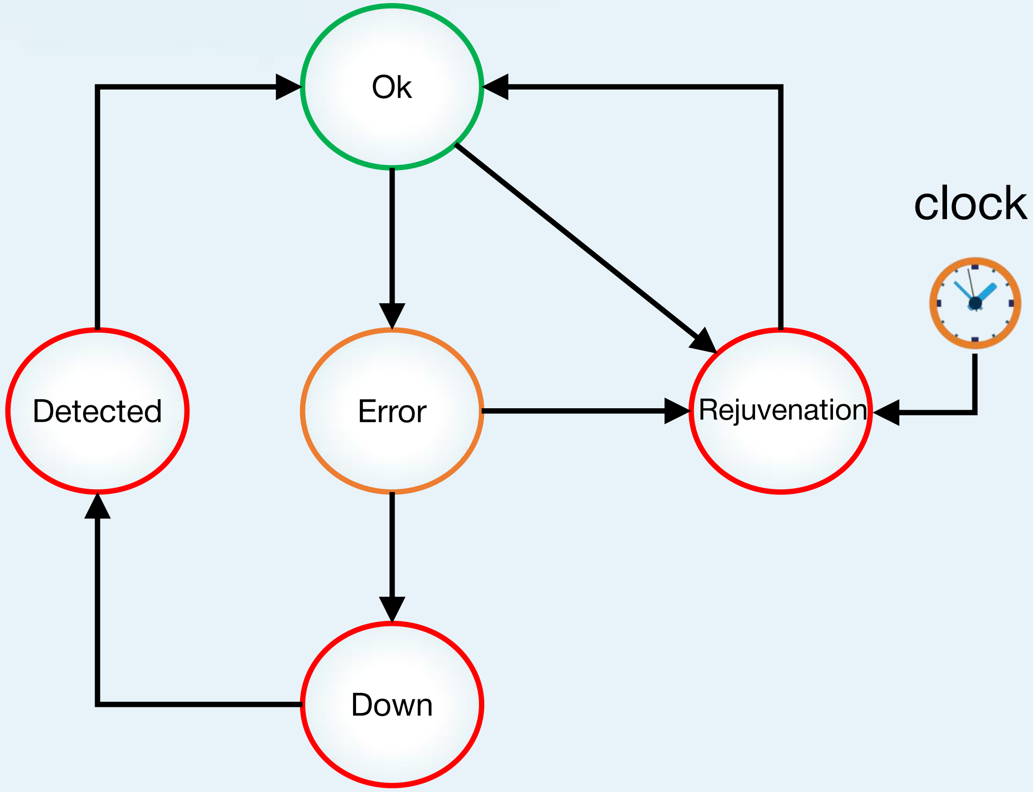
- Each component, at ant given time, can be in a single state:
  - **Ok state**: component is performing normally.
  - **Error state**: component has collected one or more errors.
  - **Down state**: errors manifested a fault, which stops the component from performing normally.
  - **Detected state**: fault has been detected and is being repaired.



- State automata of the component.

Component Model

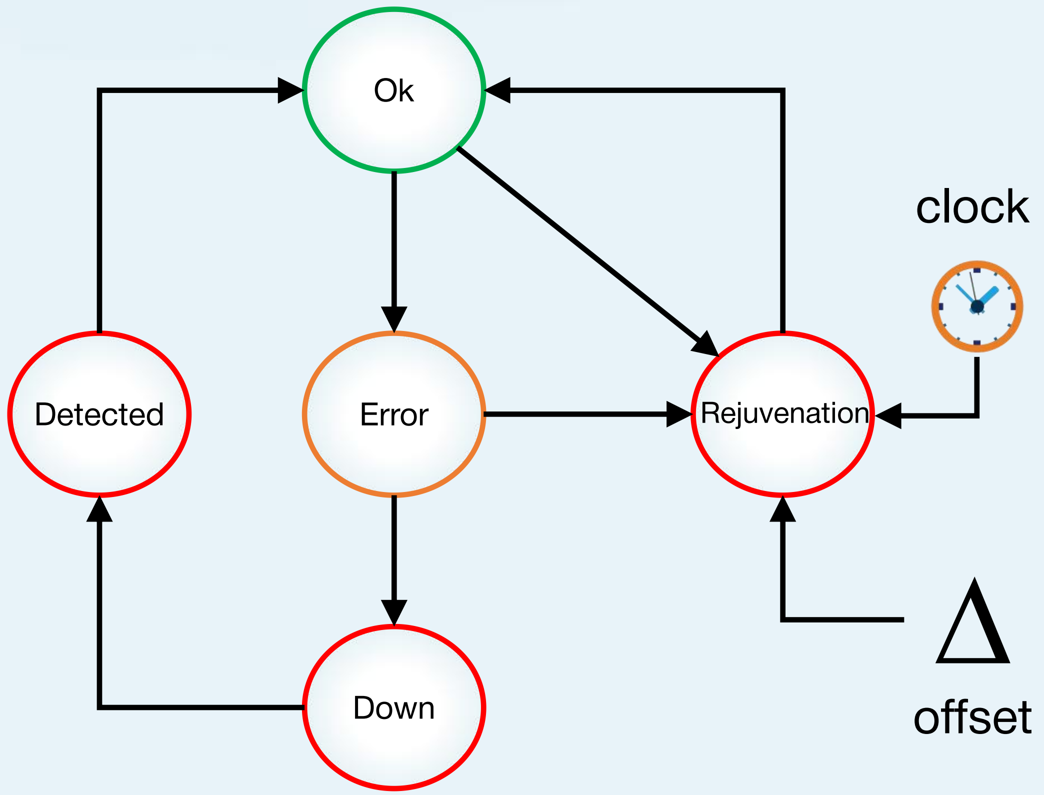
- Components are able to perform proactive maintenance: we call this **micro-rejuvenation.**
- **Rejuvenation state**: restart or maintenance to restore the component to a nominal state.
- Periodic rejuvenation (clock).
- Down & Detected states inhibit rejuvenation (and its clock).



- State automata of the component, with rejuvenation mechanism.

Component Model

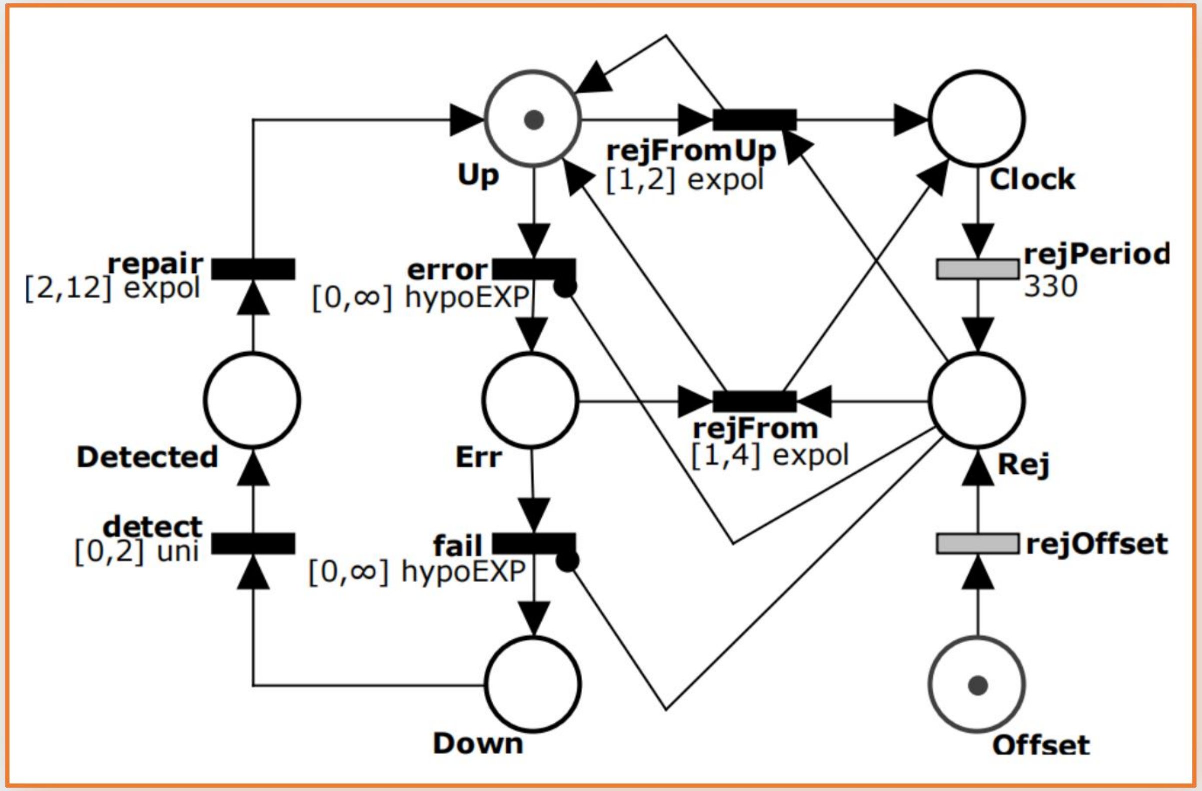
- Rejuvenating all components at the same time makes the entire system unavailable.
- Desynchronization of rejuvenations lowers unavailability.
- Same rejuvenation period for each component.
- Different offsets for each component.



- State automata of the component, with rejuvenation mechanism, and offset.

Component Model

- This model is compatible to be represented as a Markov Regenerative Process (MRGP) ...
- ... and again, this can be represented through a Stochastic Timed Petri Net (STPN).

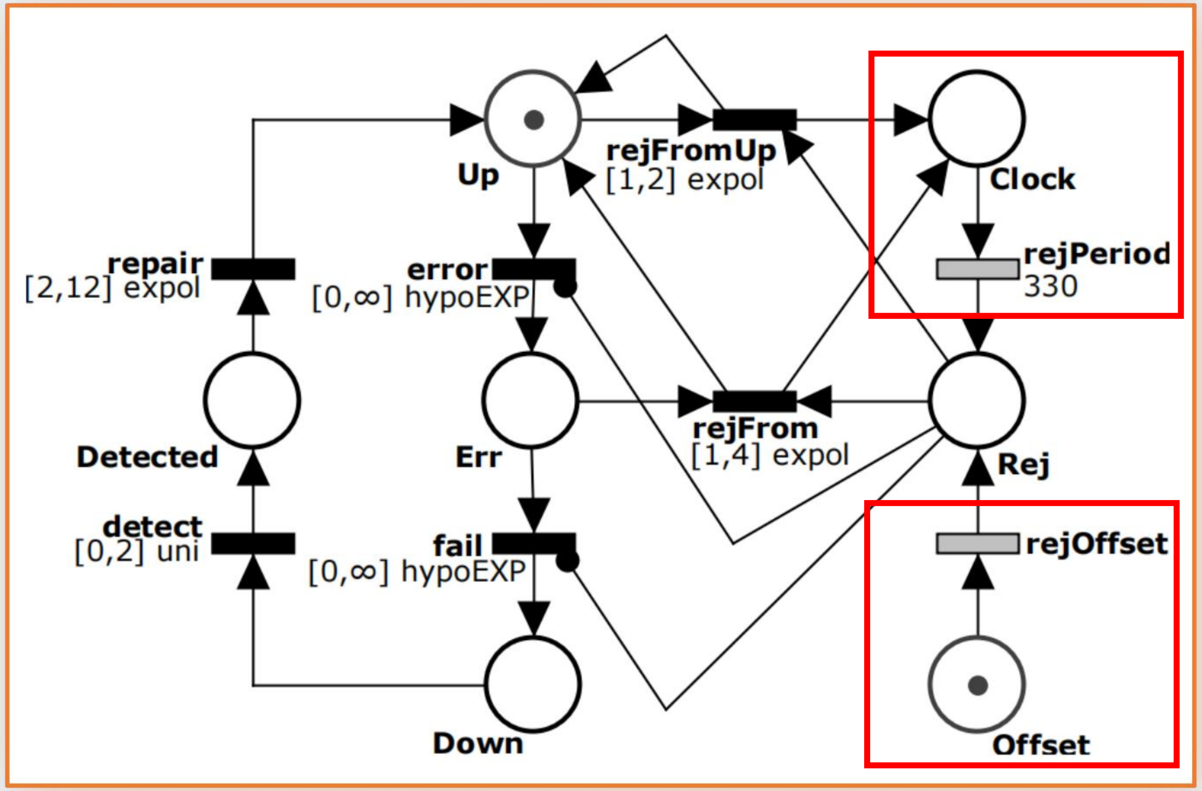


- STPN of the component MRGP model.

Component Model

- The Clock forces a regeneration periodically, if the system is not down or under repair.

- Clock is disabled until the Offset has fired.

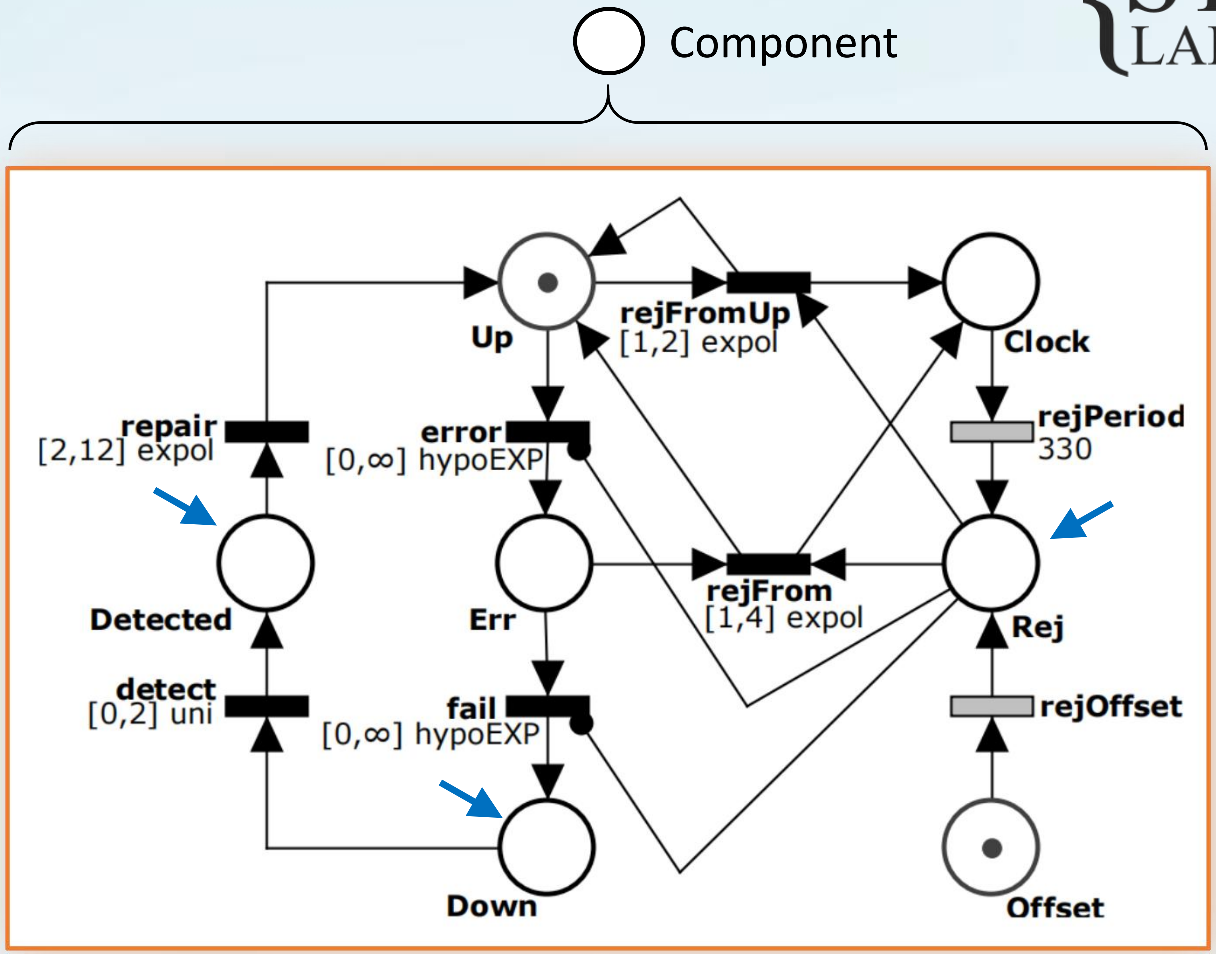


- STPN of the component MRGP model.

Component Analysis

- Each component is analysed in isolation.
- Unavailability is derived by the expected value of reward:

  Down + Detected > 0 || Rej > 0



- STPN of the component MRGP model.

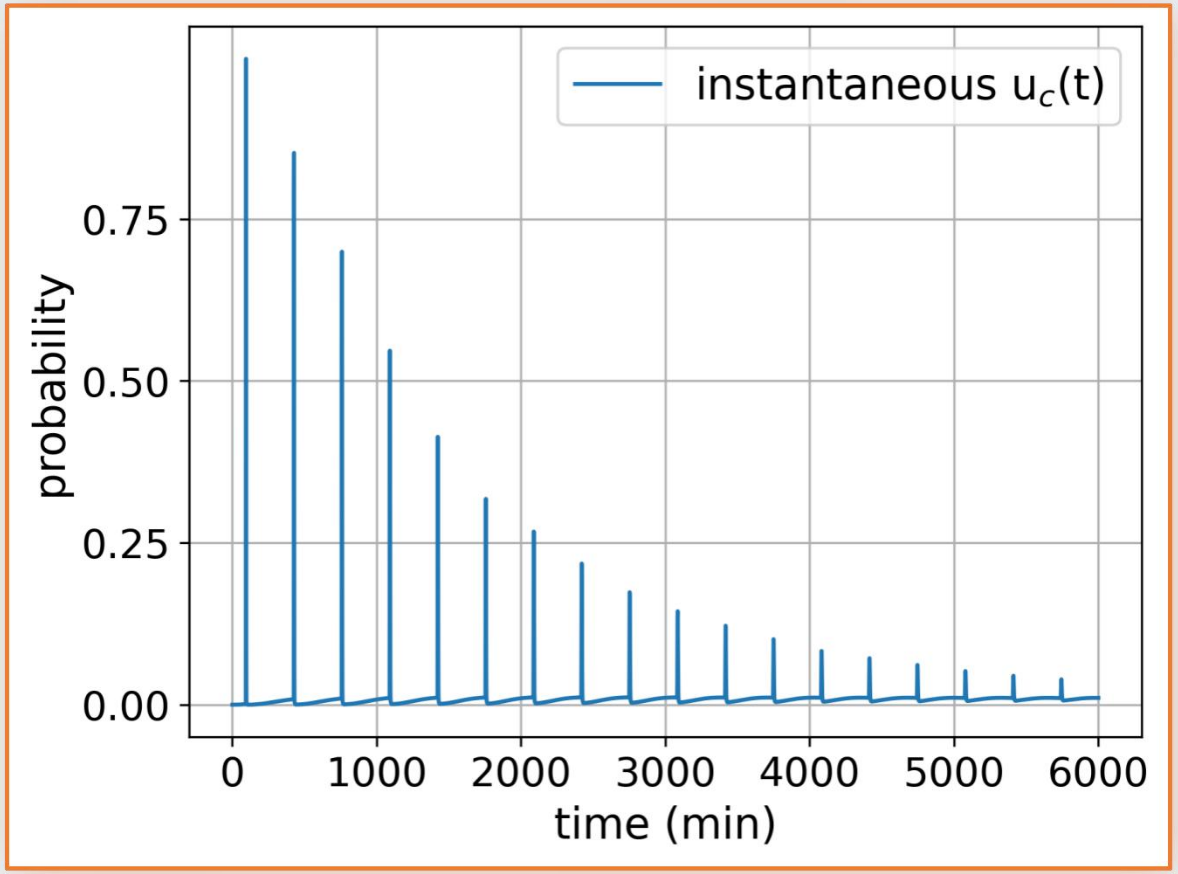
Component Analysis

- Instantaneous unavalability of the component is defined as:

$$u_c(t) := E[\mathbf{1}_{\mathcal{E}}(s(t))]$$

where ε = {failure} is a set made of the system failure state,
1 is the indicator function of set ε,
and $s(x) \in$ {failure, . . .} is the state of the system at time x.

- $u_c(t)$ is calculated using the Sirio library of the ORIS tool **, through a regenerative transient analysis on the component.



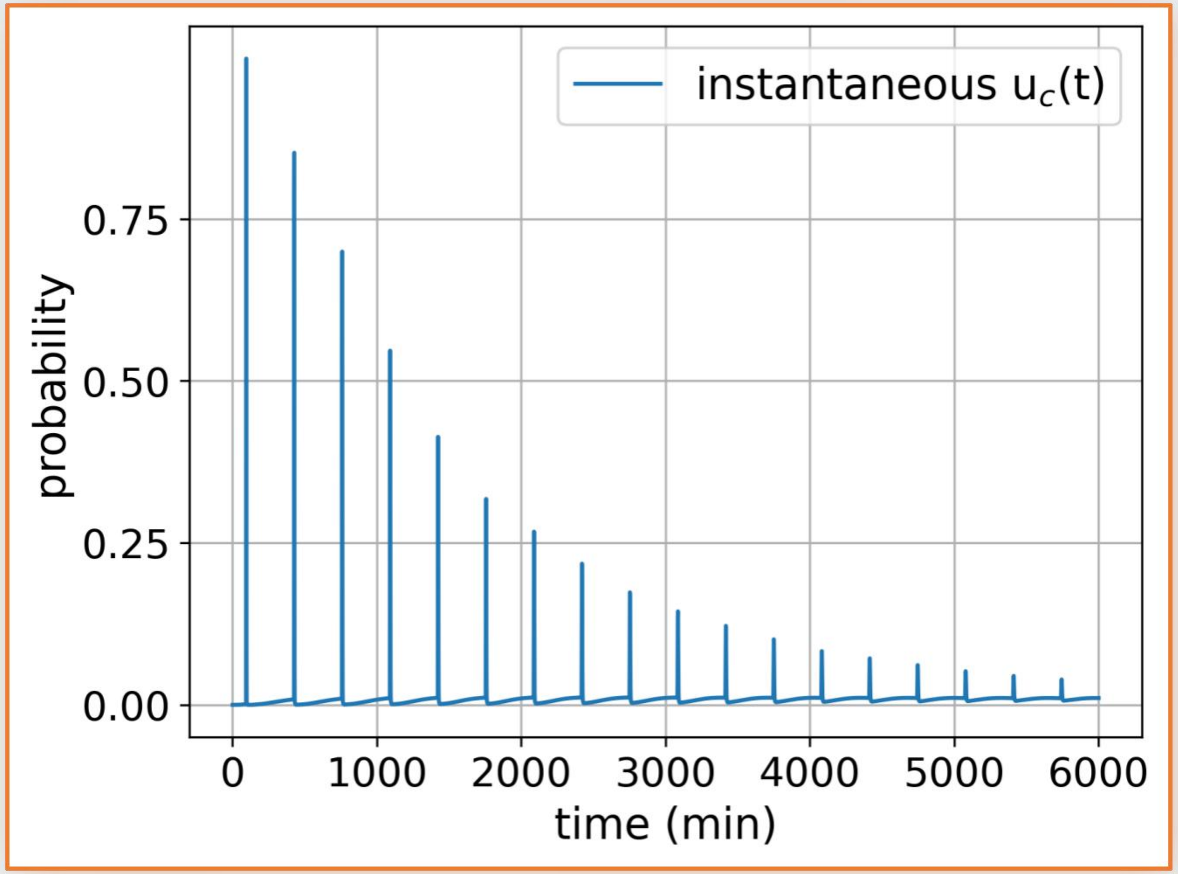
- Instantaneous unavailability of the component.

** https://www.oris-tool.org/, https://github.com/oris-tool/sirio
M. Paolieri, M. Biagi, L. Carnevali, and E. Vicario, The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems, IEEE Trans. Softw. Eng., May 2019

Component Analysis

- Peaks in the $u_c(t)$ are due to the periodic micro-rejuvenation of the component.

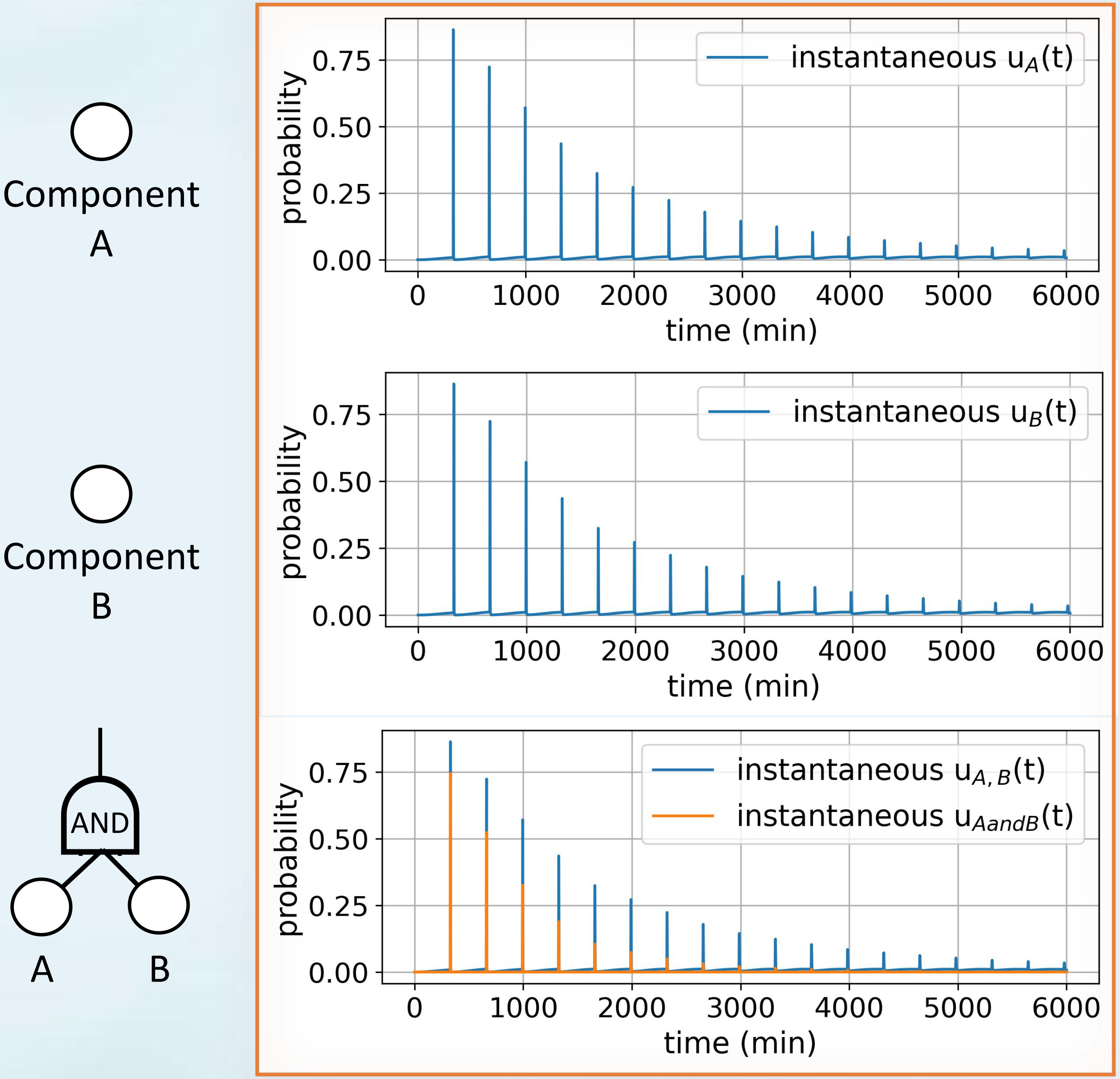
- On the long run, the component eventually reaches a **steady state** (i.e $u_c(t)$ converges to a single value).



- Instantaneous unavailability of the component.

Phasing

- Assigning an offset to the first rejuvenation greatly improves availability of the system.
- If components all rejuvenate at the same time, system can undergo a global rejuvenation period.

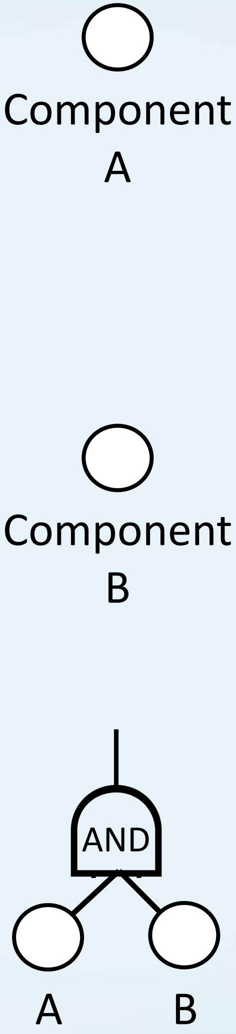


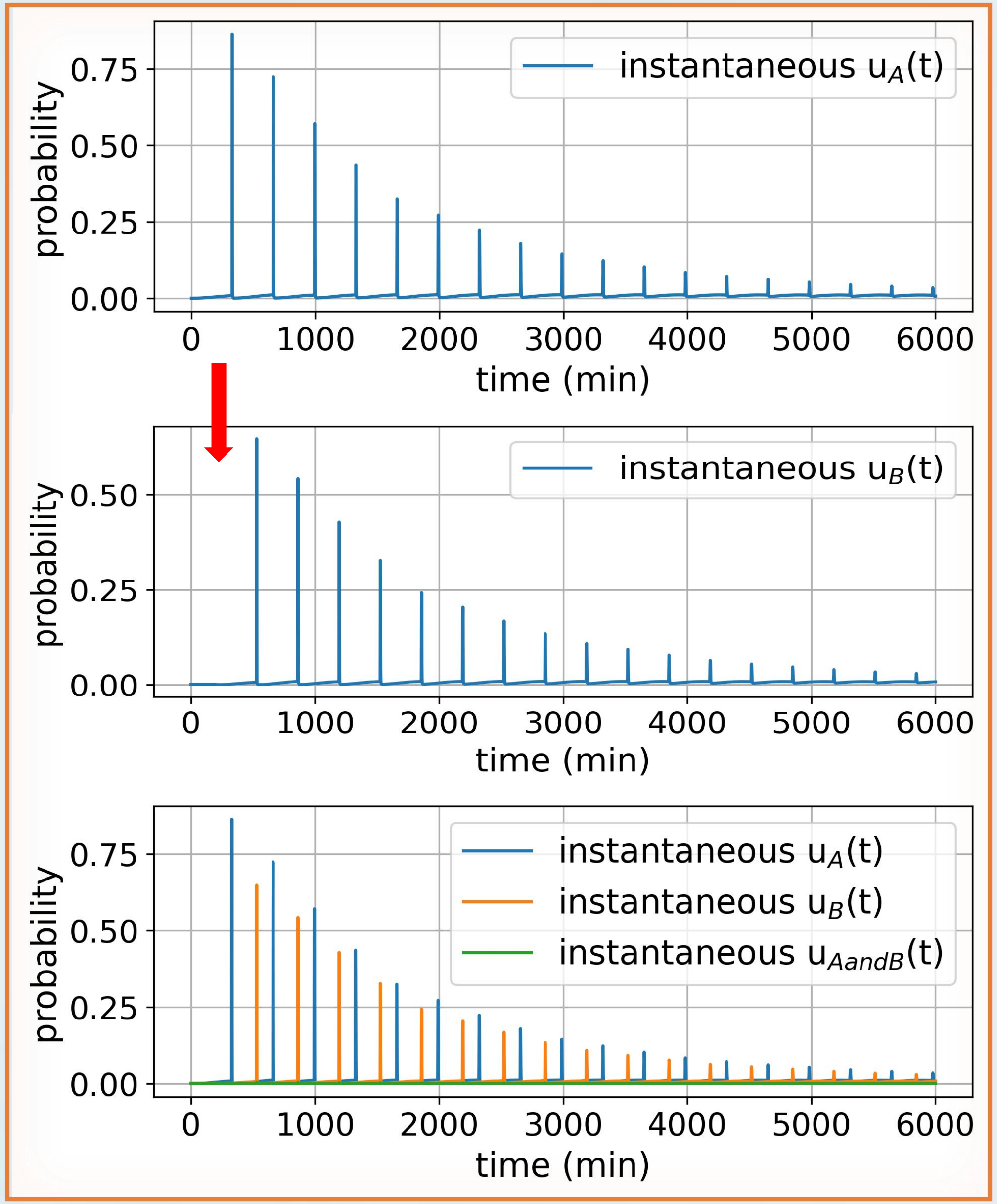
Phasing

- Assigning an offset to the first rejuvenation greatly improves availability of the system.

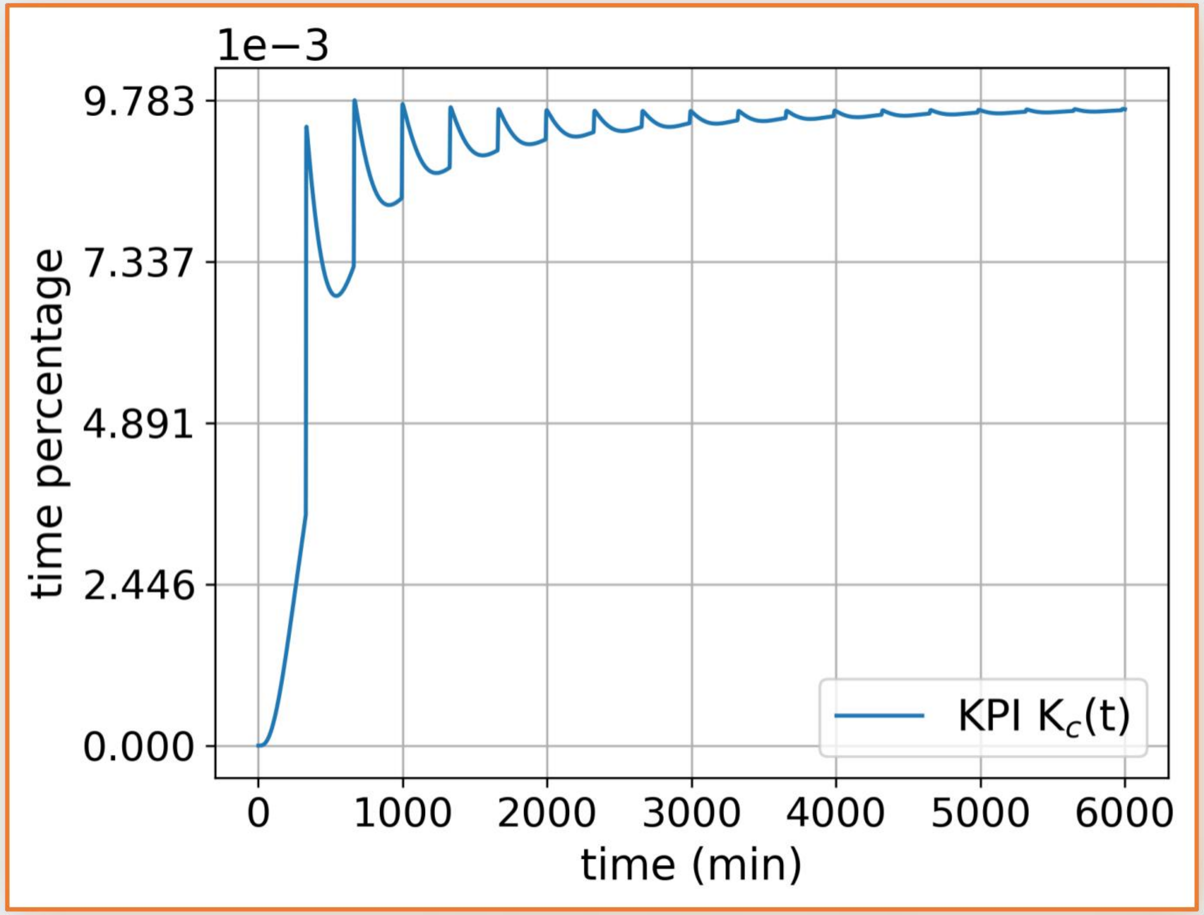
- If components all rejuvenate at the same time, system can undergo a global rejuvenation period.

- If components have their rejuvenation phased by an offset, unavailablity is greatly diminished.





Key Performance Indicator



- Unavailability KPI of the component.

- We consider $K_c(t)$ where:

$$K_c(t) := \frac{1}{t} \int_{x=0}^{t} E[\mathbf{1}_{\mathcal{E}}(s(x))]\, dx$$

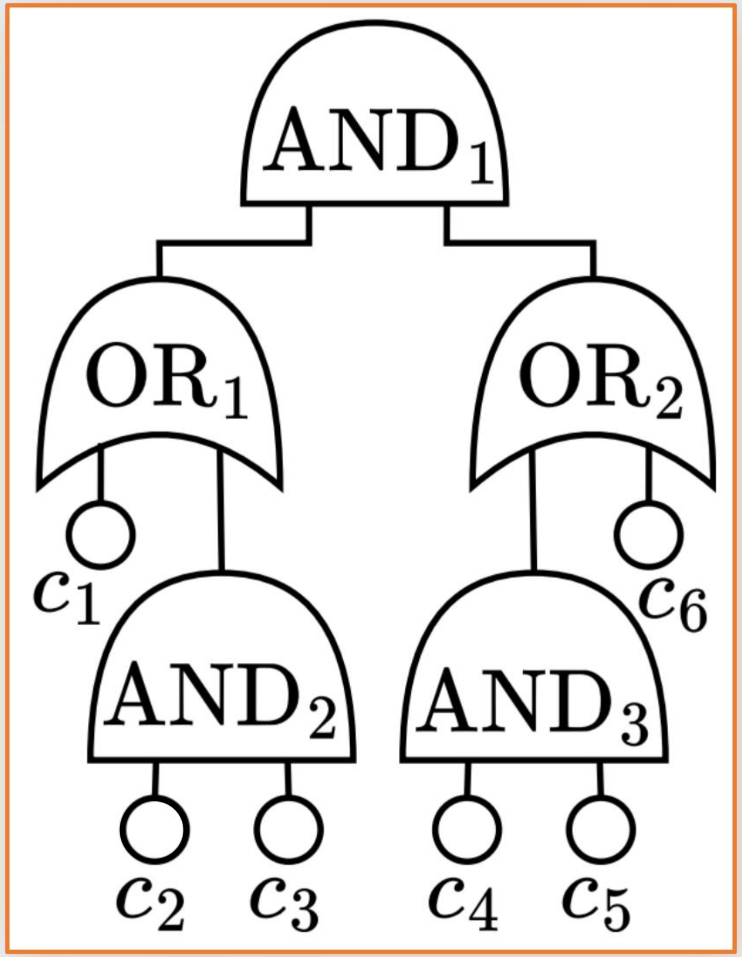
is the expected percentage of time during which the system has been unavailable in the interval [0, t].

- We aim at selecting values of the component offsets that minimize the system cumulative unavailability over time K(t).

System Model

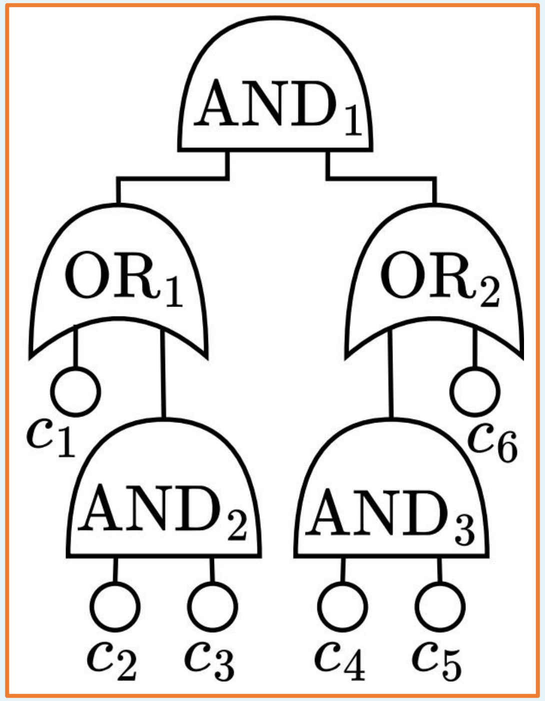
- Objective: evaluate the availability (**unavailablity**) of the system under observation.
- Consider a static fault tree with AND / OR logical gates, with components as leaves.
- System unavailability can be naturally decomposed and computed for each component in isolation, and then combined through the boolean expression of the fault tree.

$$u_{\mathrm{AND}}(t) = \prod_{i=1}^{n} u_i(t) \qquad u_{\mathrm{OR}}(t) = 1 - \prod_{i=1}^{n}(1 - u_i(t))$$



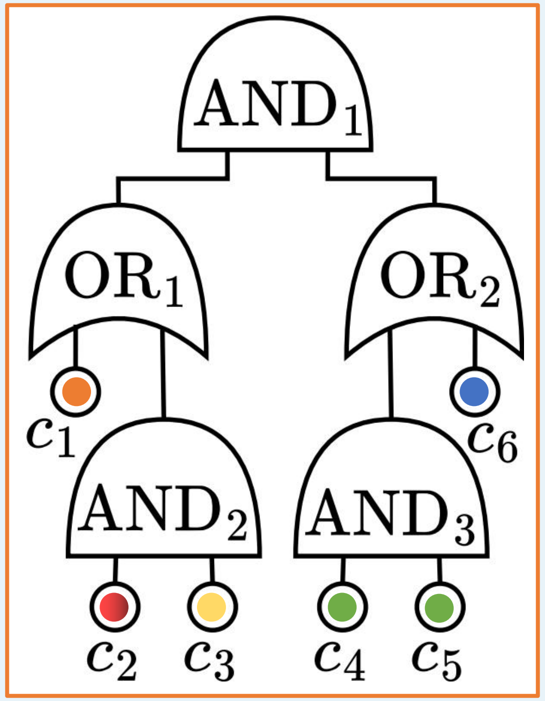
- Static fault tree of a system with 6 components.

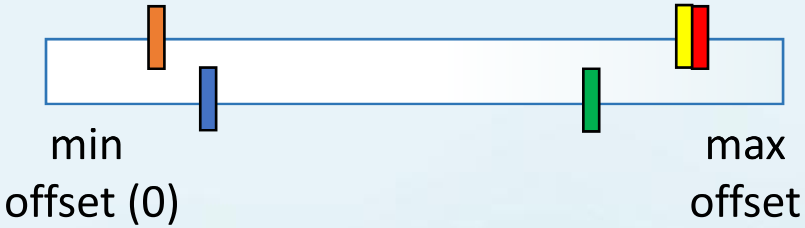
Optimization Algorithm



- Two possible policies for offset assignment: **uncoordinated** and **coordinated**.
- **Uncoordinated** approach : components will choose a random offset.
- **Coordinated** approach: a centralized controller visits the fault tree, assigning offset based on an algorithm or policy.
- Offset values between 0 and the duration of the rejuvenation period.

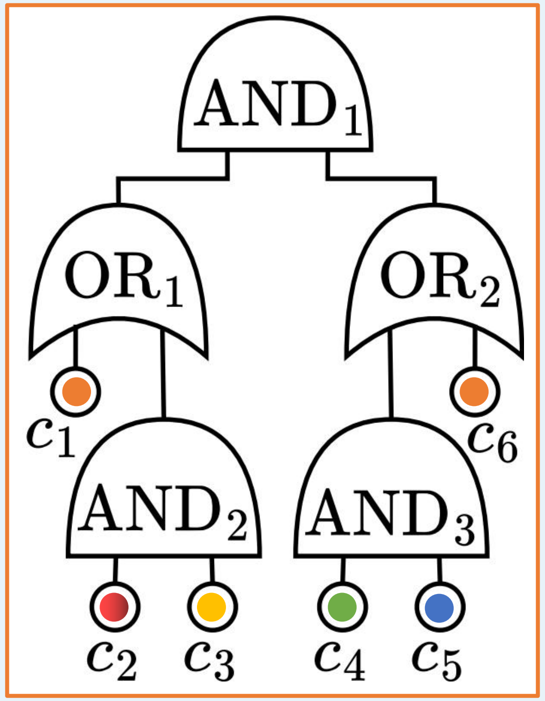
Optimization Algorithm

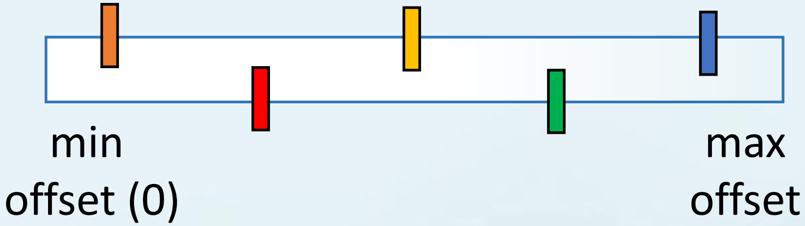




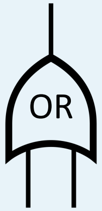
- **Uncoordinated approach** : components will choose a random offset.
- Each component chooses independently, without any coordination.
- Components rejuvenate either at the same time or different times.
- Doesn’t require a centralized controller.

Optimization Algorithm

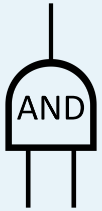




- **Coordinated approach** : a centralized controller visits the fault tree, assigning offset based on an algorithm.
- Our heuristic is based on a top-down visit to determine the required offsets to be assigned, and a bottom-up visit to assign them.

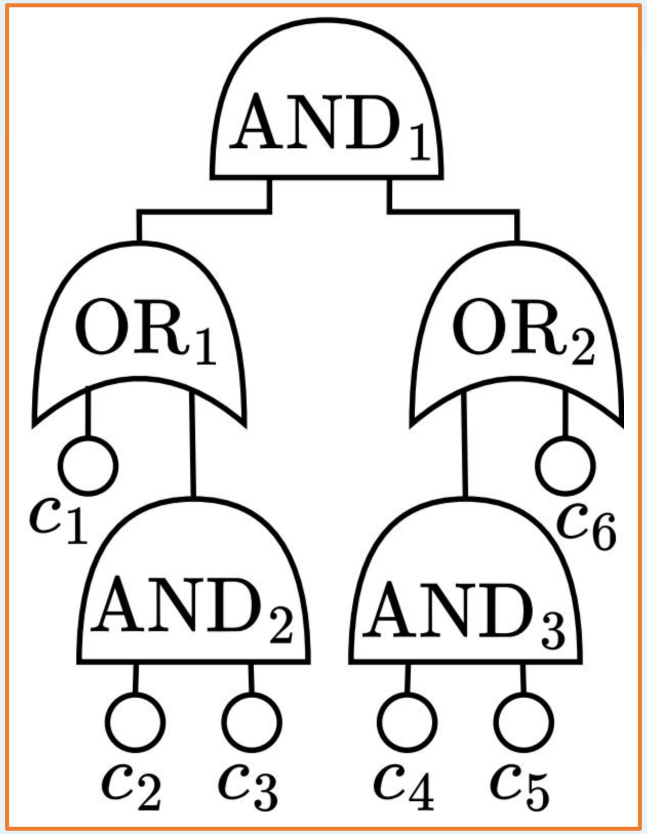


- Components under OR gates are phased with the same offset.
- Components under AND gates must be phased with different offsets.



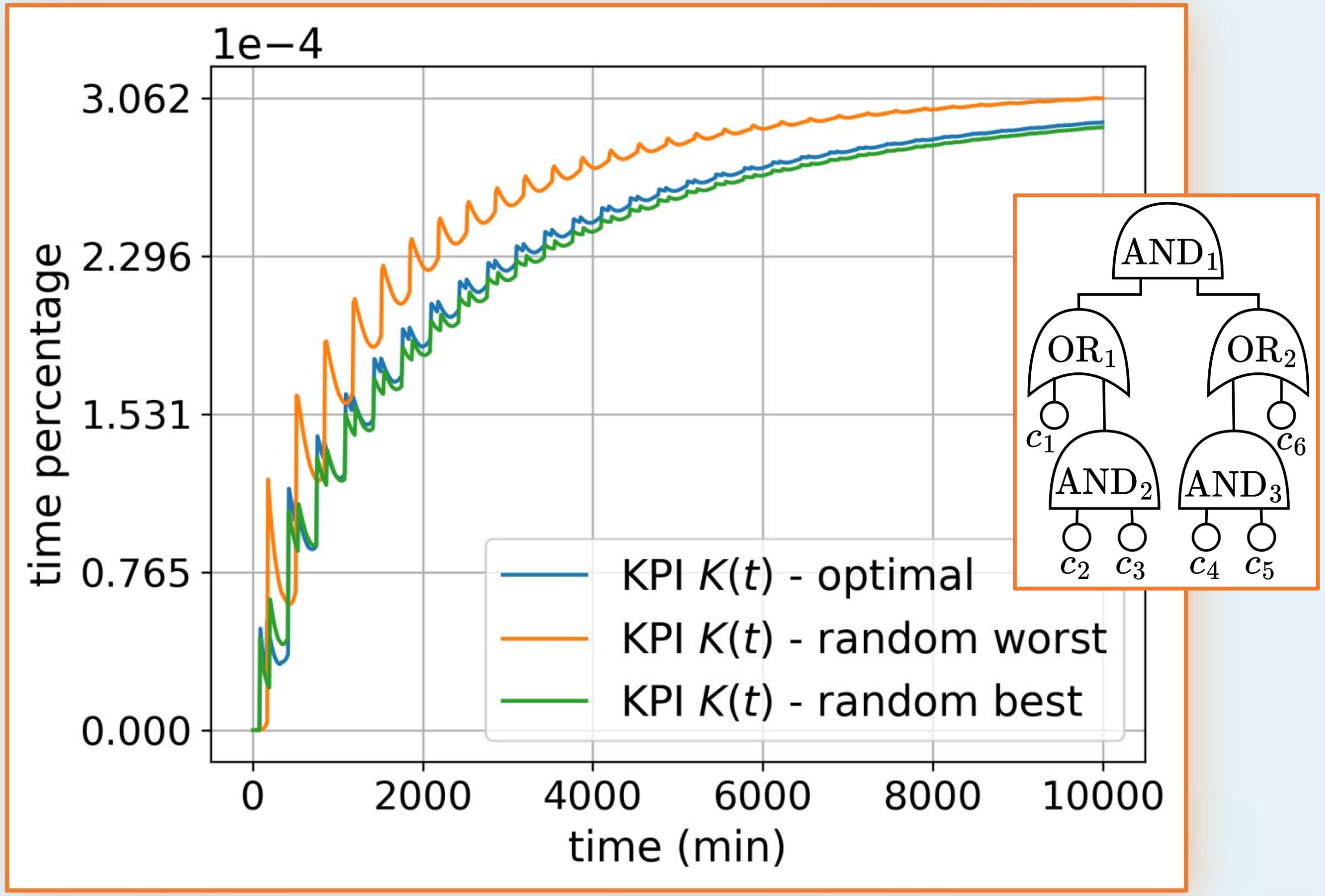
System Analysis

- First tests have been performed on an example fault tree.
- All components have the same model, with the same rejuvenation period, but must be assigned an offset.
- Offsets are assigned through both coordinated (once) and uncoordinated (n times) methods.
- System is then analyzed after each offset assignment.



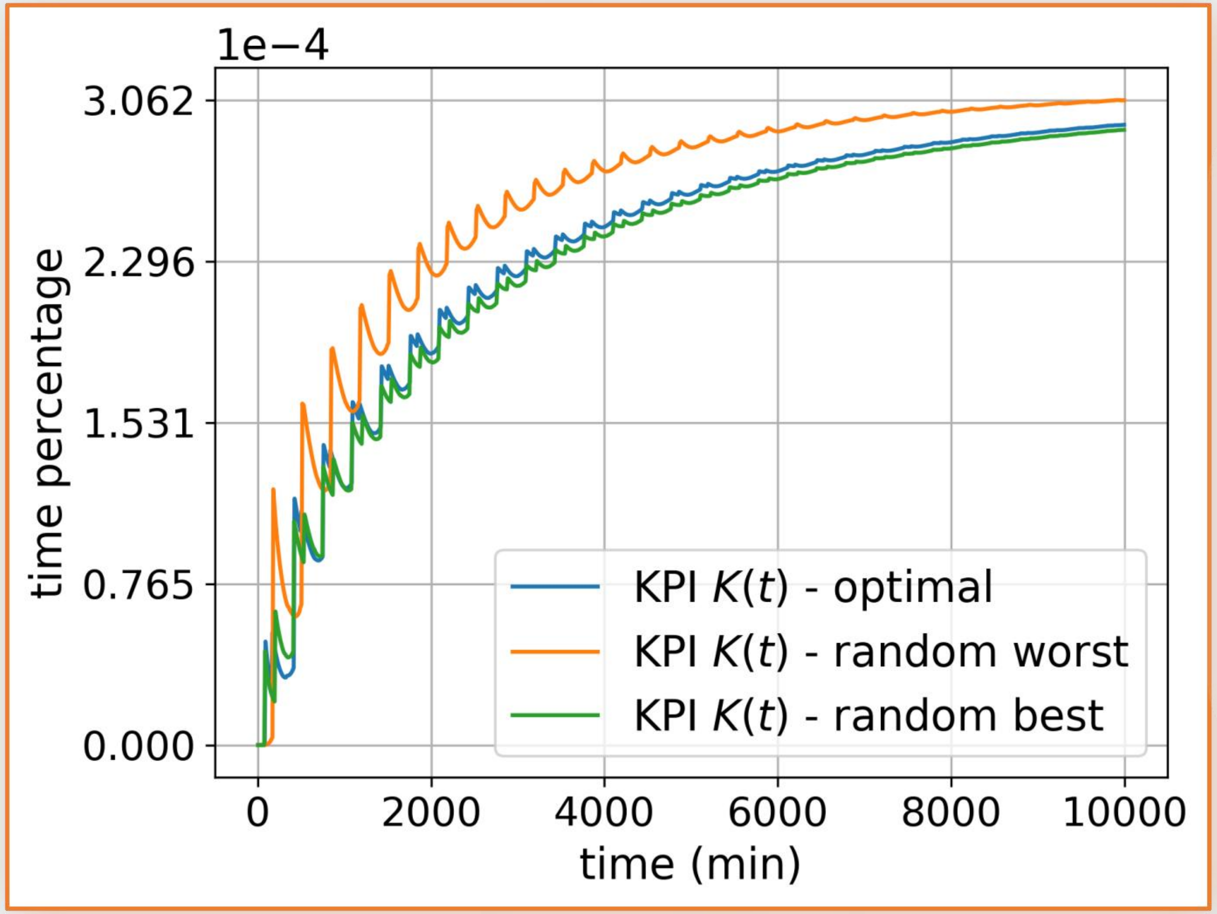
- Static fault tree of a system with 6 components.

System Analysis

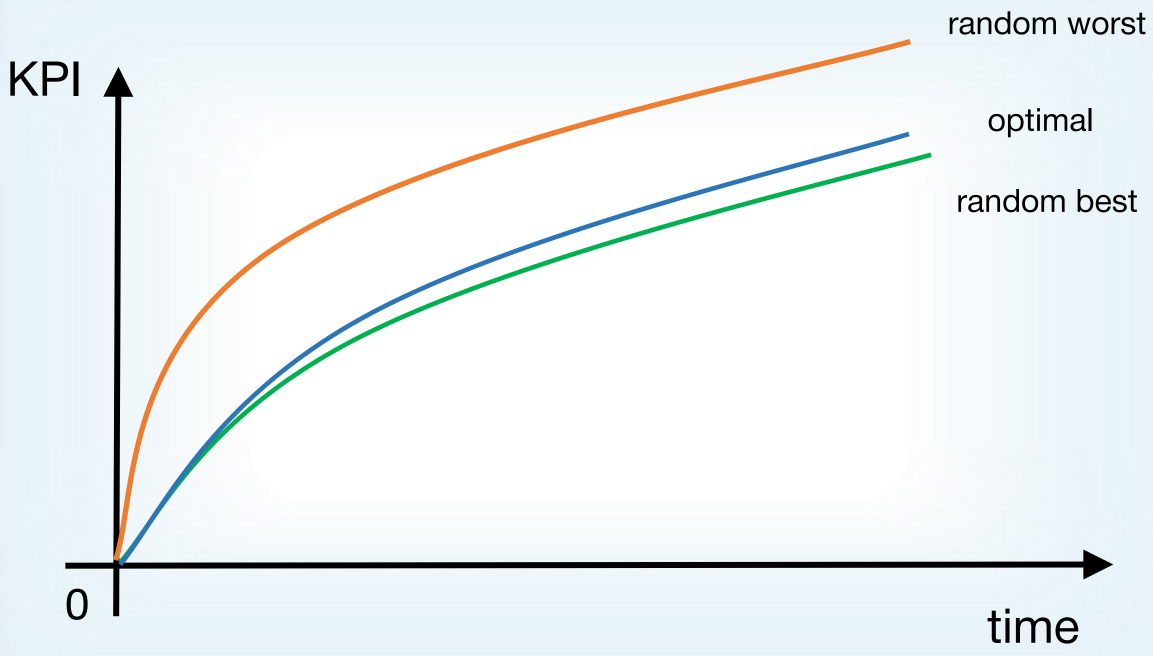


- Unavailability KPI under coordinated and uncoordinated offset assignments.

System Analysis



- Unavailability KPI under coordinated and uncoordinated offset assignments.

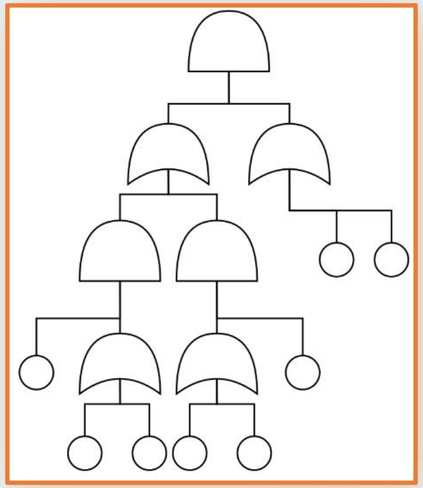


- KPI behaviour under coordinated and uncoordinated offset assignments.

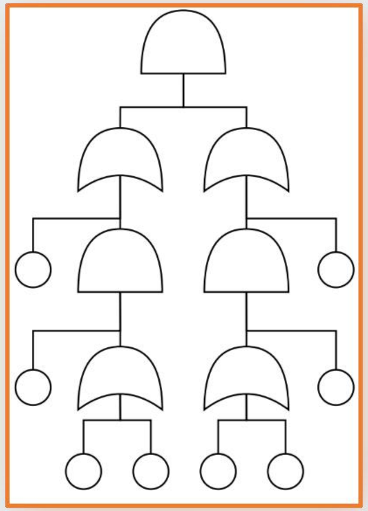
Results

- Approach has been tested on randomly generated fault trees with varying number of components, using an ad-hoc algorithm to generate properly distributed fault trees.

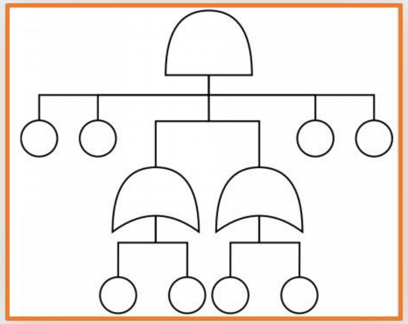
- Tests have been made on FTs with fixed shape:



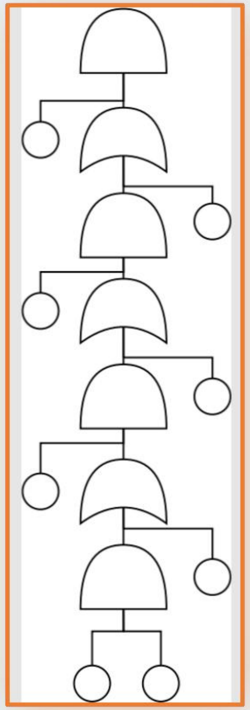
- Unbalanced FT



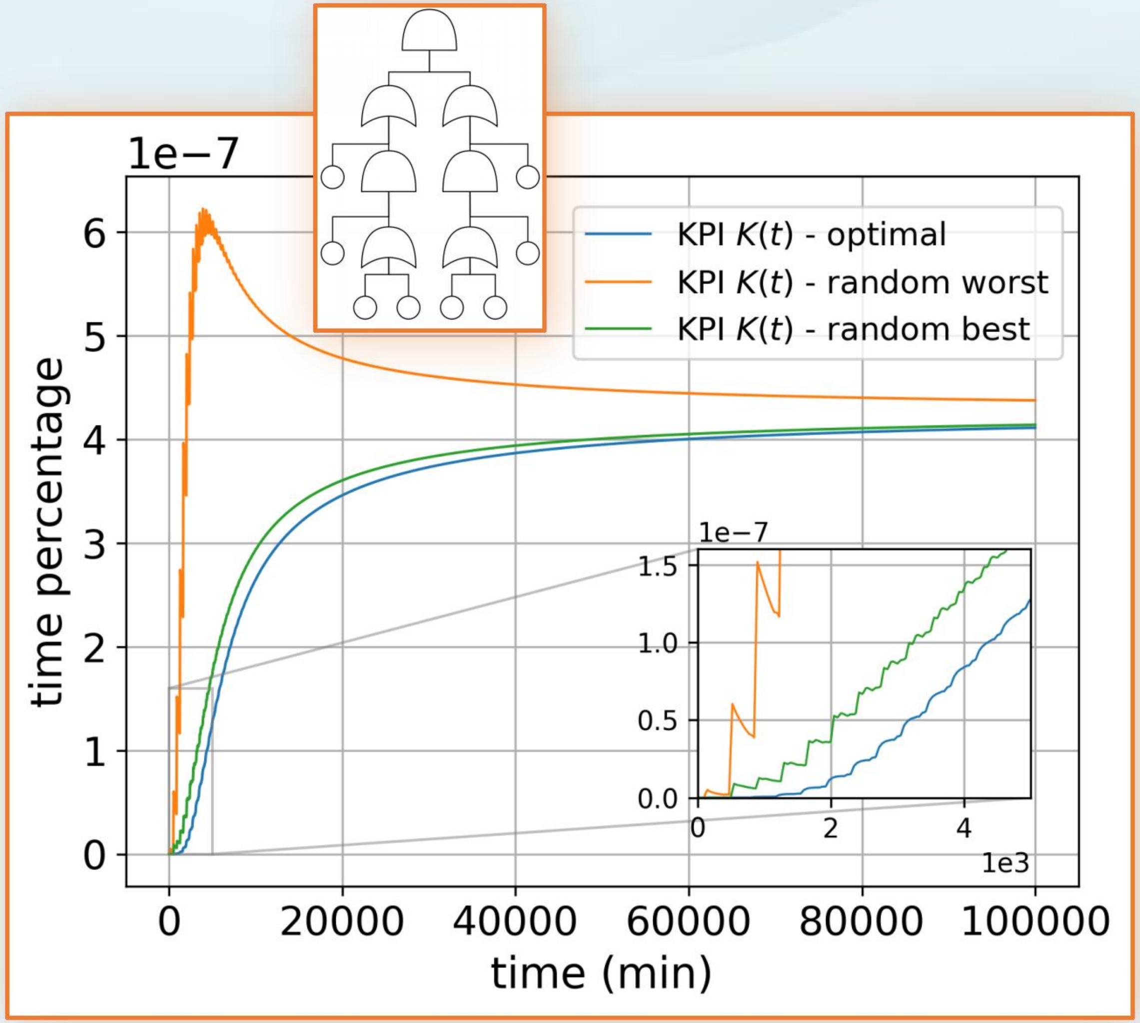
- Balanced FT



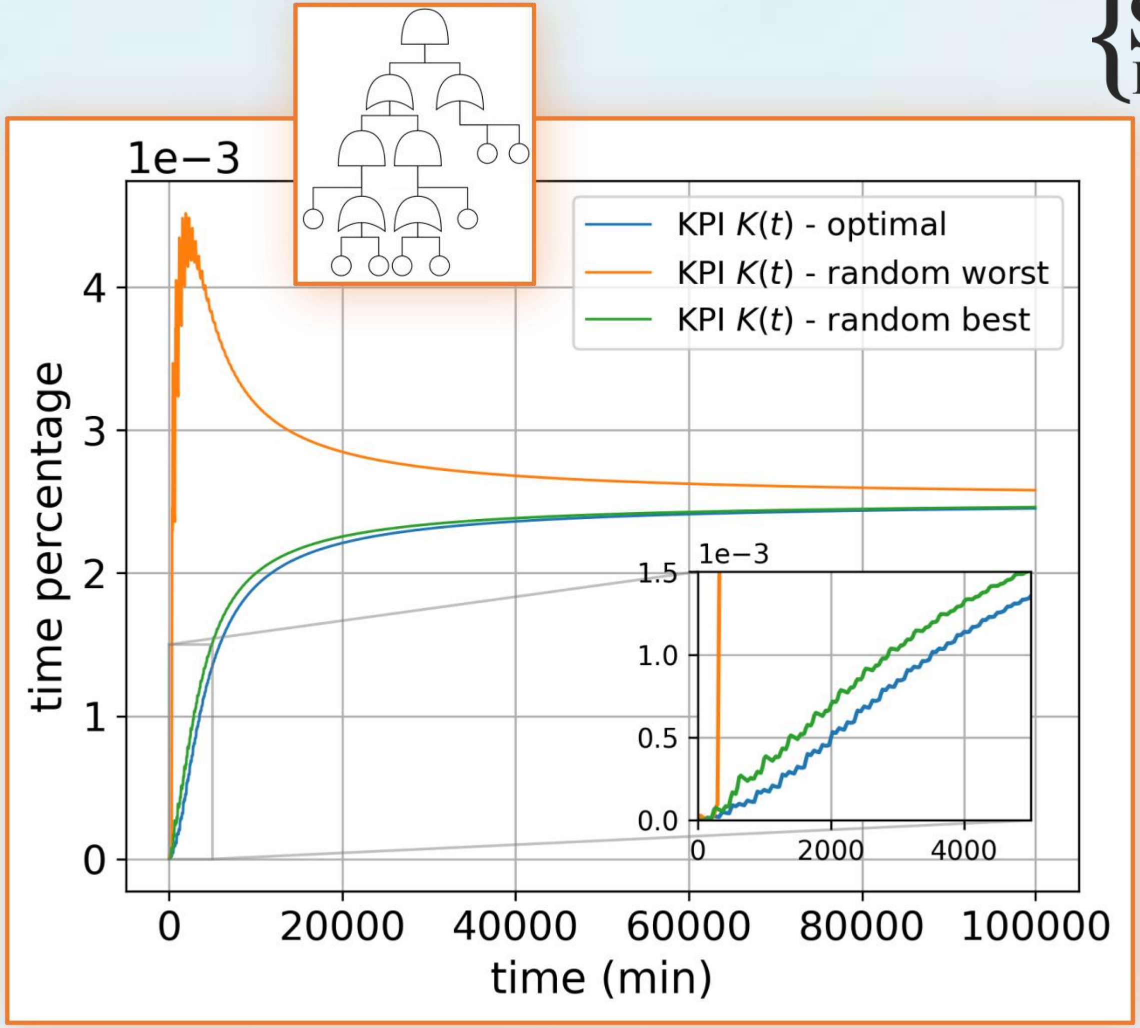
- Shallow FT



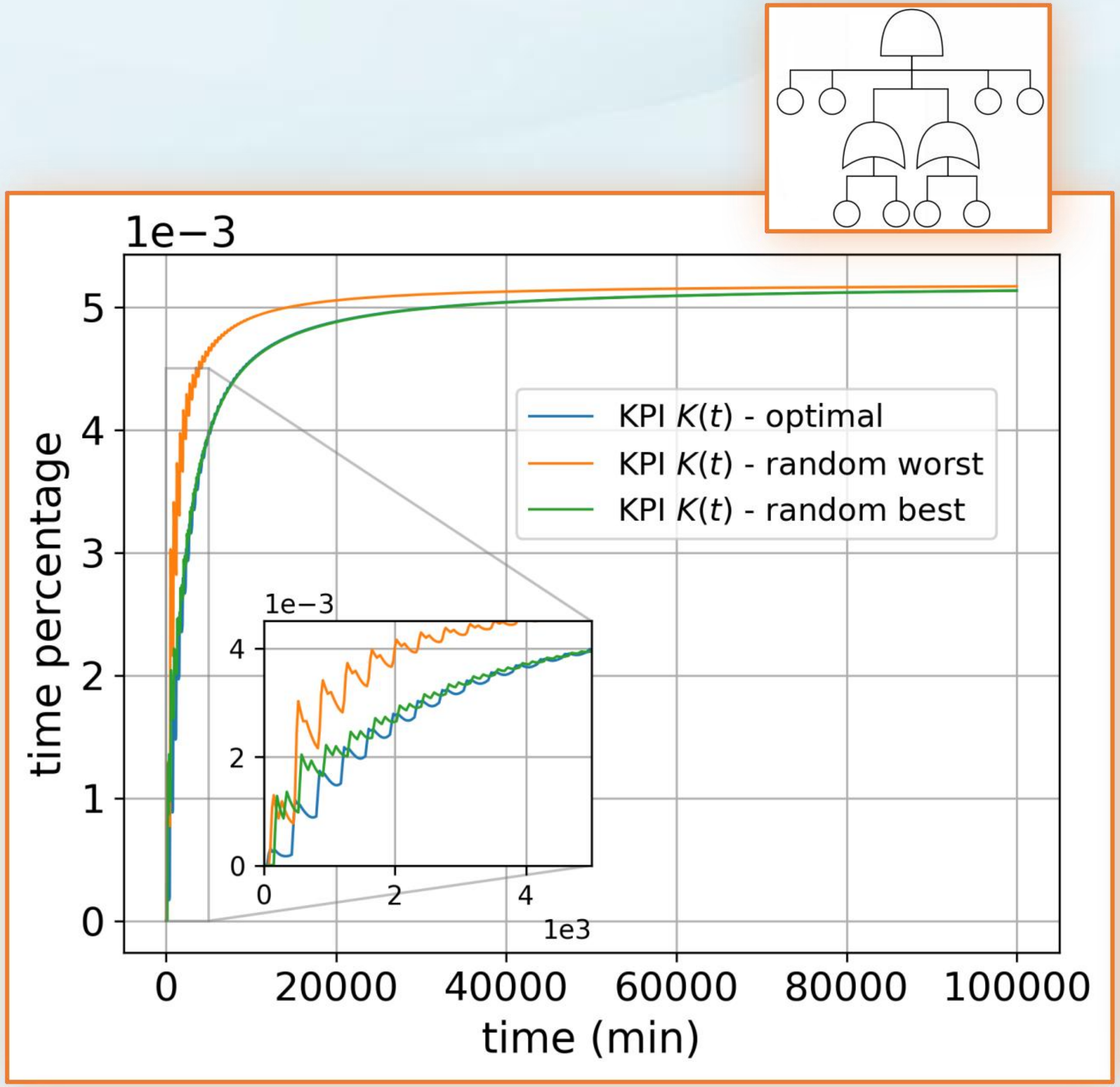
- Deep FT



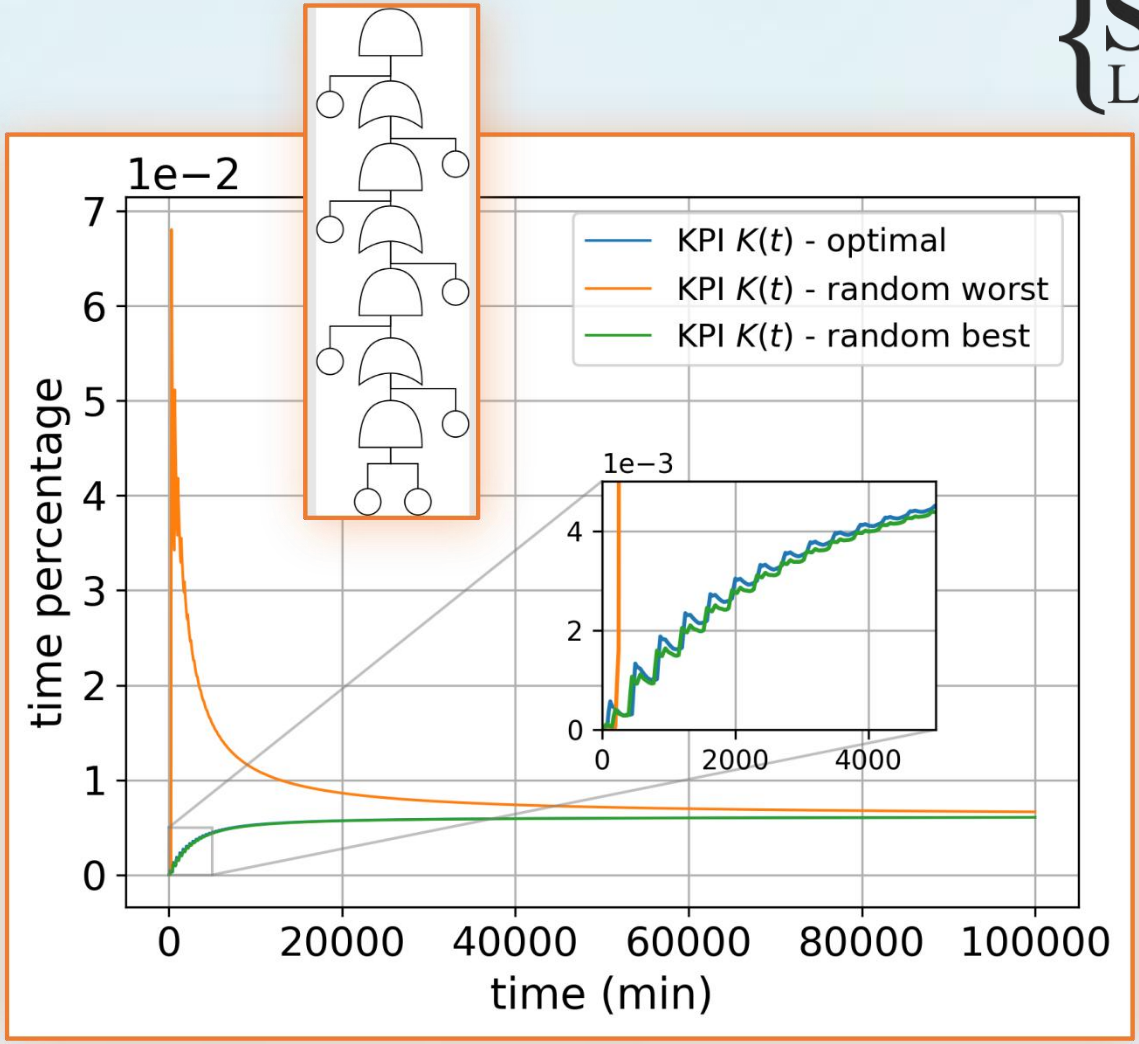
- Balanced FT: Unavailability KPI under coordinated and uncoordinated offset assignments.



- Unbalanced FT: Unavailability KPI under coordinated and uncoordinated offset assignments.



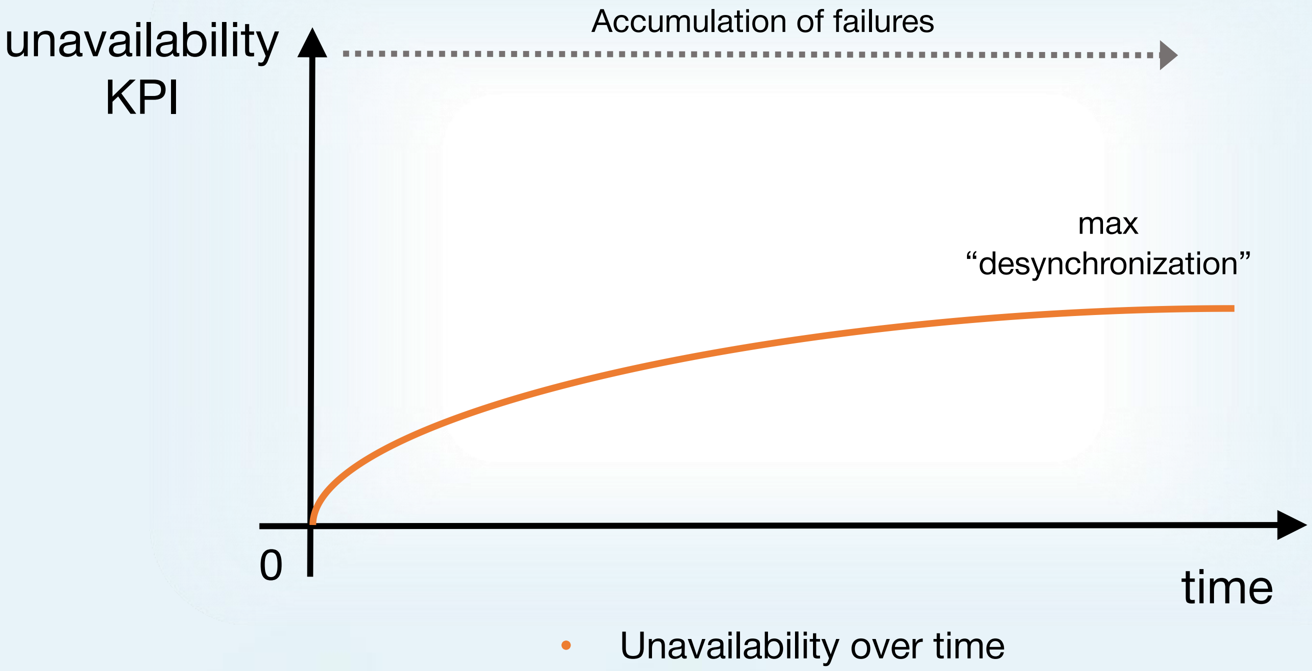
- Deep FT: Unavailability KPI under coordinated and uncoordinated offset assignments.



- Wide FT: Unavailability KPI under coordinated and uncoordinated offset assignments.

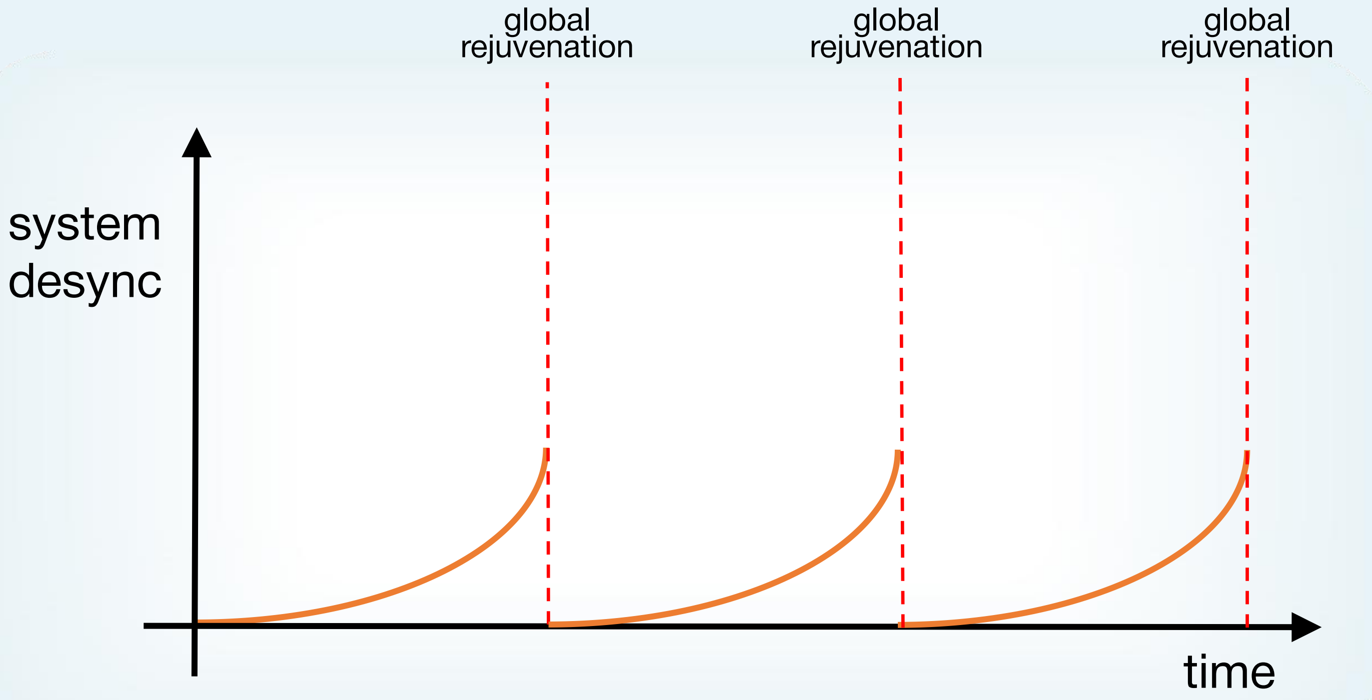
On Macro Rejuvenation

- System accumulates “desynchronization” effects from optimal phasing due to component failures.
- Drift from optimal synchronization reduces the effect of coordinated rejuvenation in the long term.

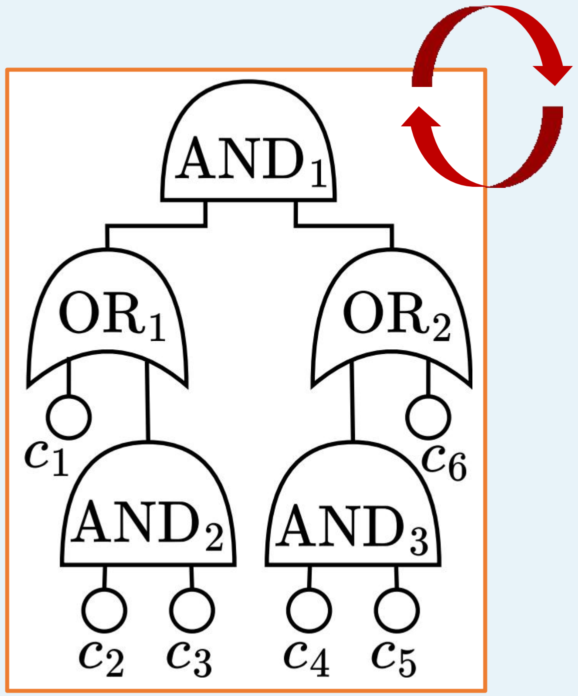


On Macro Rejuvenation

- Adding a global rejuvenation mechanism to the system mitigates the problem of desynchronization.

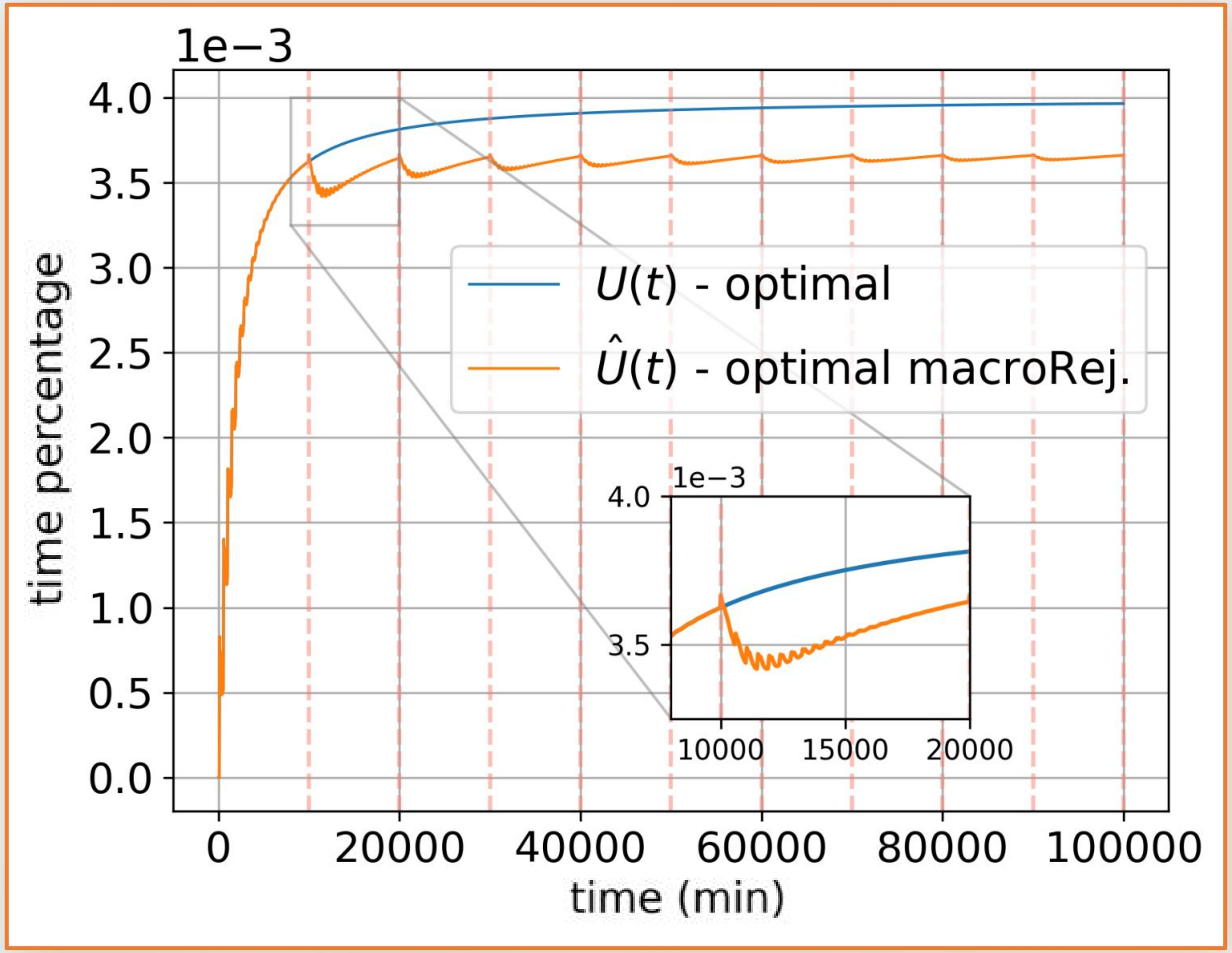


- Software desynchronization, with global rejuvenation.



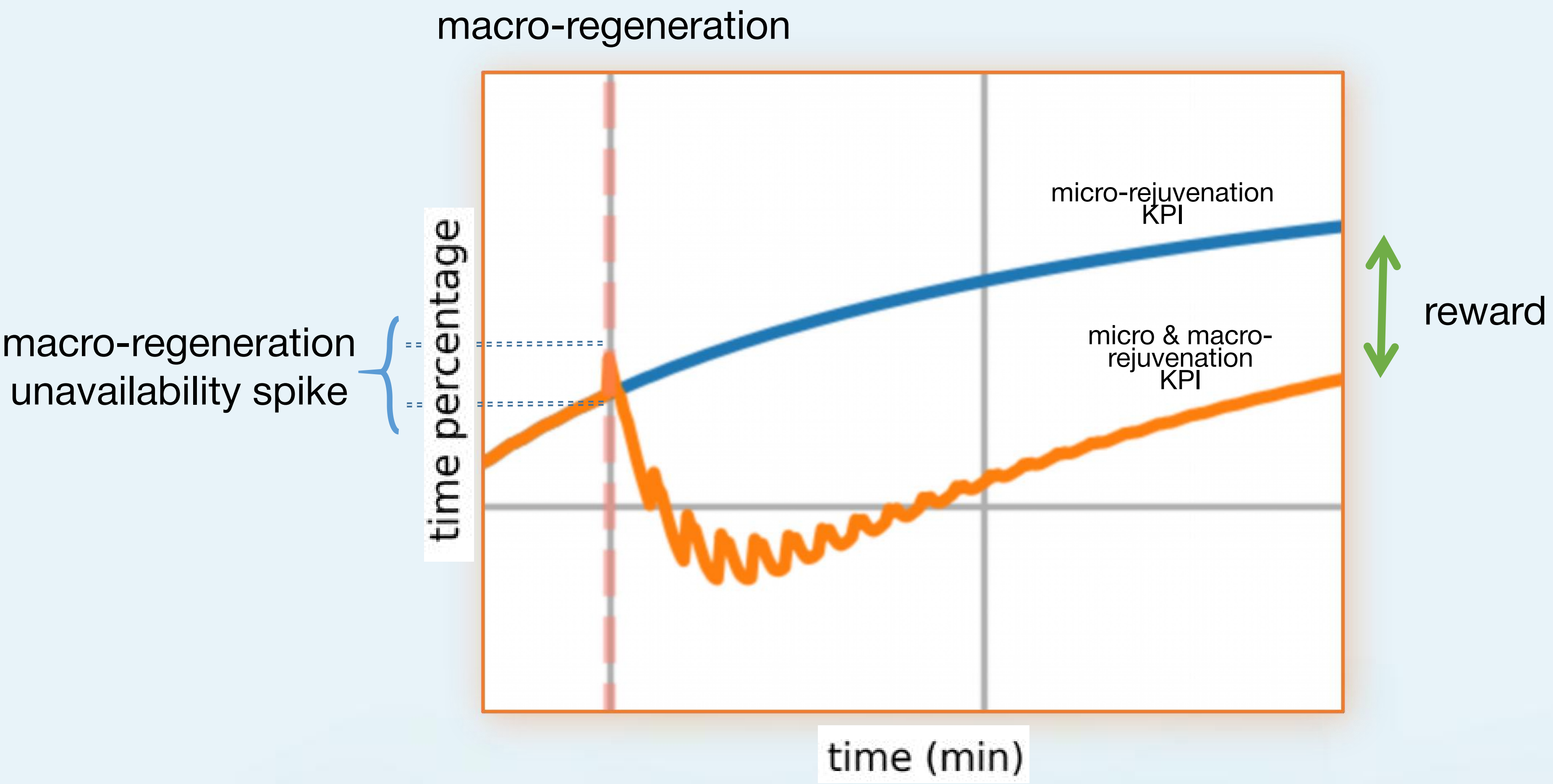
On Macro Rejuvenation

- Global or macro-rejuvenation rewards higher availability to the system.
- At the cost of higher complexity.
- Macro-rejuvenation period and duration must be well parameterized.



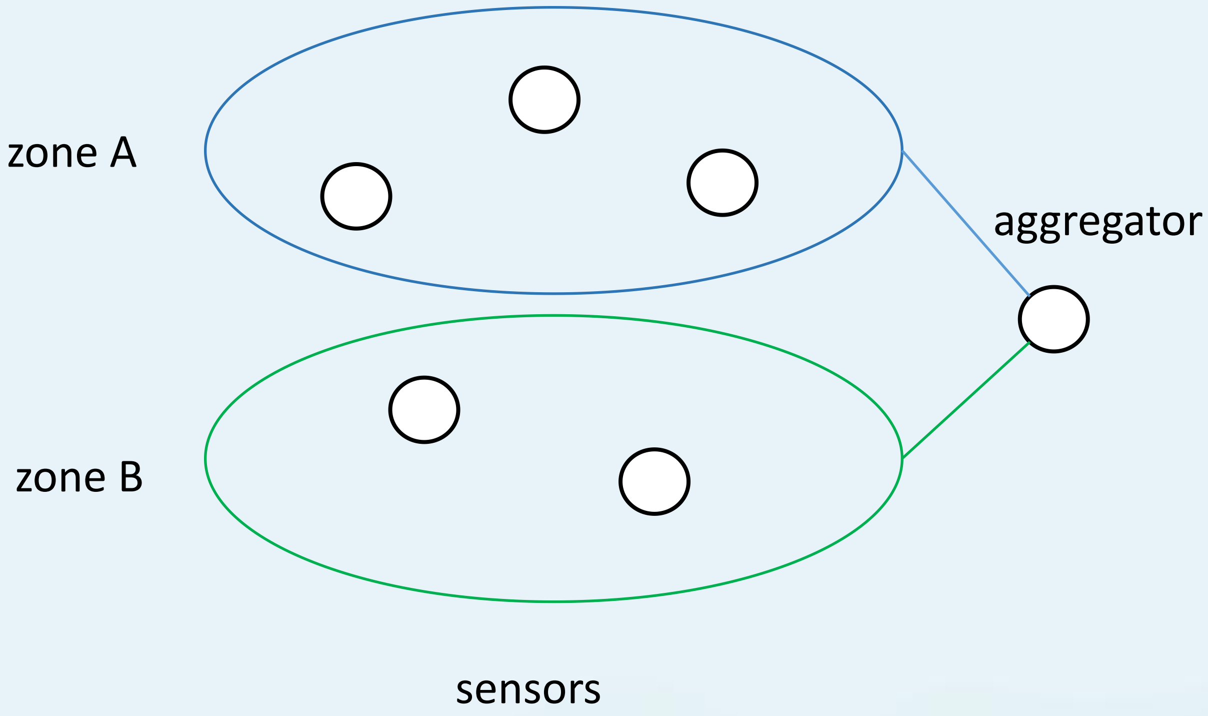
- Unavailability KPI of the system.

On Macro Rejuvenation



- Zoom-in section of unavailability KPI graph.

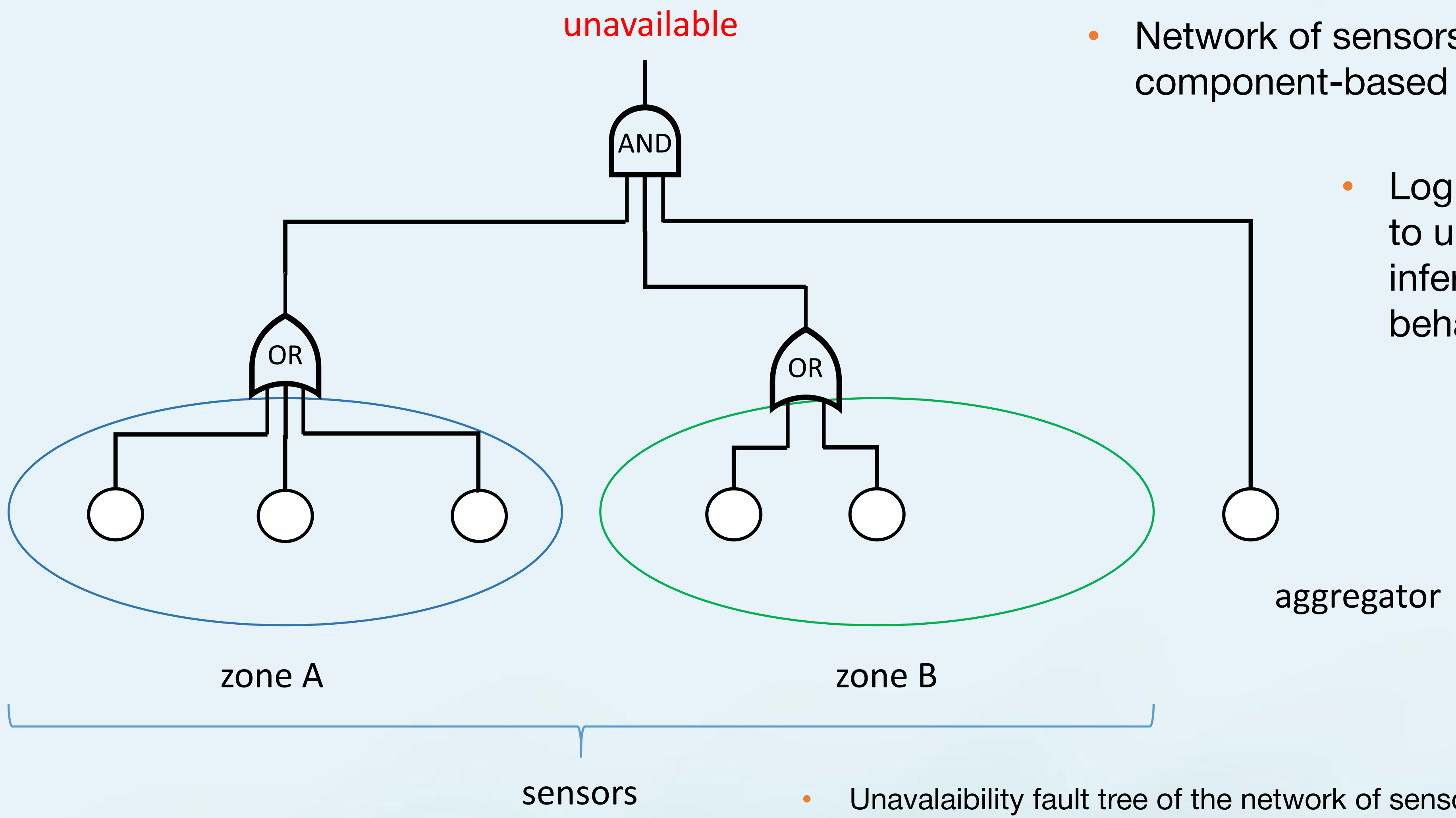
Practical Implications



- Representation of the network of sensors.

- Consider a group of sensors covering two zones, with an aggregator collecting their data.
- All sensors in a zone must be available at all times.
- System becomes unavailable if every zone and the aggregator aren't available.

Practical Implications / Consequences



- Network of sensors can be modeled as a component-based system.
- Logic relationships in regards to unavalaibility can be inferred from the system's behaviour.
- System can also apply a macro-rejuvenation policy.
- Unavalaibility fault tree of the network of sensors.

Conclusions

- First contribution: proposal of a coordinated offset assignment policy for component-based systems and compositional analysis to evaluate unavailability KPI by composition.
- Second contribution: study on the applications of macro-rejuvenation mechanisms to component-based systems.

- Evaluation of fault trees with more complex logical gates is ongoing.
- We are also considering the use of **dynamic** fault trees.
- We are researching on the concept of mixing time (time to steady state) for component-based systems.

Thank you for the attention.

Leonardo Paroli
leonardo.paroli@unifi.it

Software Technologies Lab, University of Florence