State-Based Dependability Modeling

Dr. Peter Tröger

Lena Herscheid

Sources:

Eusgeld, Irene et al.: Dependability Metrics. 4909. Springer Publishing, 2008

Menasce, Daniel A.; Almeida, Virgilio A.: Capacity Planning for Web Services: Metrics, Models, and Methods. Prentice Hall, 2002. , 0-13-065903-7

M. A. Marsan, “Stochastic petri nets: An elementary introduction,” in Advances in Petri Nets, pp. 1–29, Springer, 1989.

C. A. Petri, Kommunikation mit Automaten. PhD thesis, Darmstadt University of Technology, Germany, 1962

Malhotra, Manish, and Kishor S. Trivedi. "Power-hierarchy of dependability-model types." Reliability, IEEE Transactions on 43.3 (1994): 493-502.

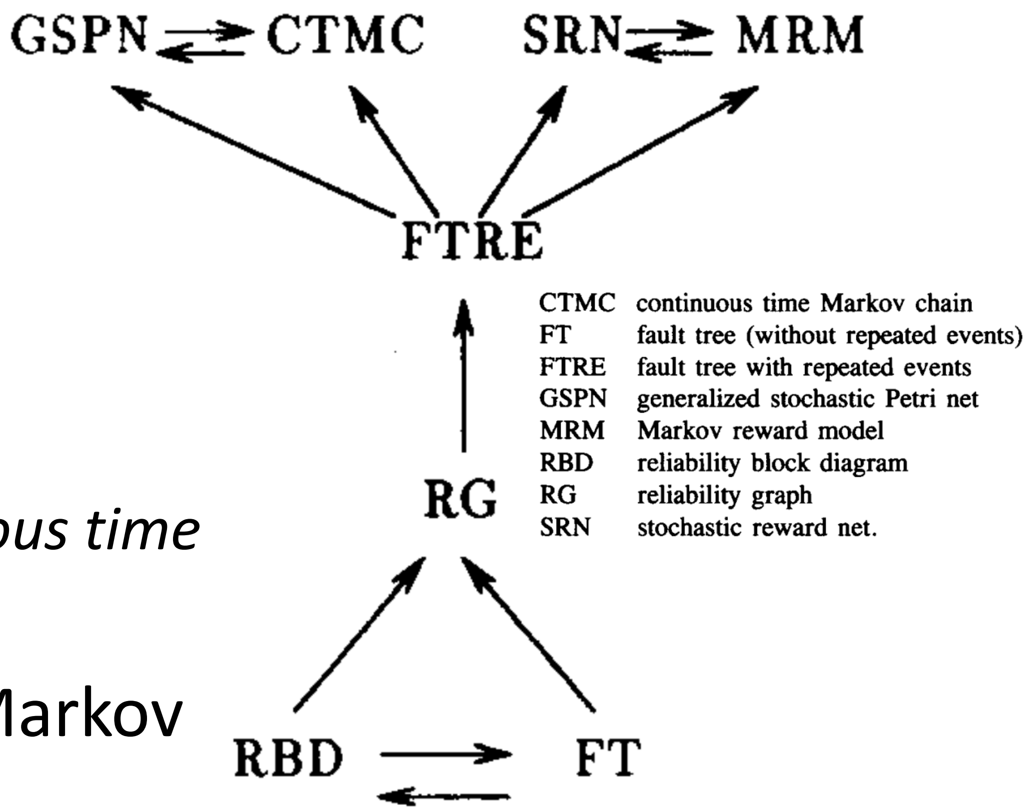
Dependability Modeling

RECAP

- Use a formalism to model system dependability
  - Quantify dependability attributes of components
    - Calculate system availability/reliability
    - Based on a set of data and assumptions - the *availability model*
  - Most models expose the same expressiveness
  - Each formalism allows to focus on certain aspects
  - **Component-based** models: Reliability block diagrams, fault trees
  - **State-based** models: Markov chains, petri nets
- System understanding evolved from hardware to software to IT infrastructures
  - Example: Organization management influence on business service reliability
    - Information Technology Infrastructure Library (ITIL)
    - CoBiT(Control Objectives for Information and related Technology)

Structural vs State-Based Dependability Models

- **Structural** / combinatorial models:
  - Focus on static system structure
  - High-level graphical modelling
  - Mapping components to model elements
- **State-based** / Markov models:
  - Focus on dynamic behaviour
  - Notion of *stochastic distributions in continuous time*
  - Can be solved analytically or simulated
- Structural models are often mapped to Markov models for quantitative analysis



Malhotra, Manish, and Kishor S. Trivedi. "Power-hierarchy of dependability-model types."

State-based models

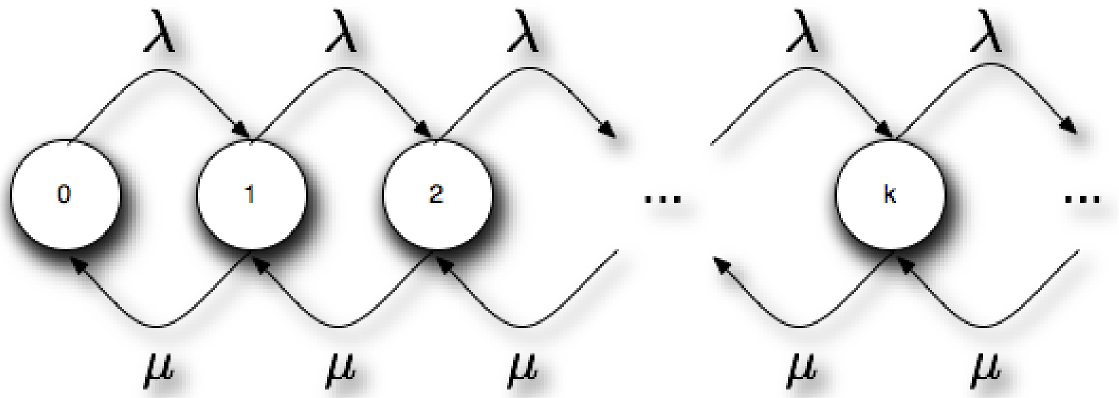
- Component-based models work well if failure events are stochastically independent
  - But: Catastrophic events destroy multiple components
- State-based models focus on failure states of the system
  - Can handle transitions between failure states
  - Independent of the system structure
- **Analytical** solution
  - Demands independent failures, constant failure rates, (exponential distribution)
- Solution through **simulation**
  - State model is simulated to estimate the resulting dependability metrics
  - Arbitrary failure event distributions, approximations, long simulation time

State Transition Diagrams

- Modelling approach typically used for queueing systems
- Assumptions
  - **Homogeneous workload assumption**
    All request are indistinguishable, so only their sum counts
  - **Operational equilibrium**
    Number of requests in the system is the same at the start and end of investigation
    - May vary in the interval, but average throughput is constant
    - Number of departures tends to approach the number of arrivals - „all forces on the object are balanced“
  - **Memoryless assumption**
    Server state is a single parameter - number of processed requests

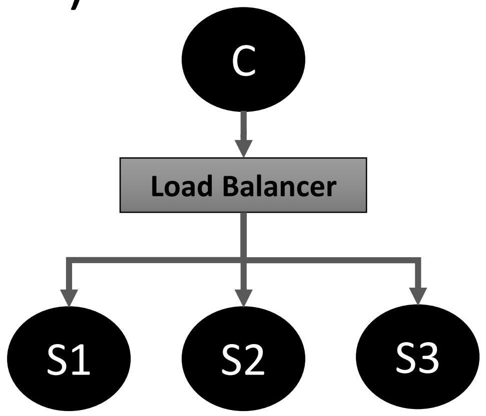
State Transition Diagrams

- Transitions between states happen at some *rate*
  - Arrival rate λ (transitions / sec), request completion rate μ (transitions / sec)
- **Flow-In Flow-Out** principle
  - Operational equilibrium ensures that transitions into the state are equal to transitions out of that state
  - Not relevant how this state was reached and how long it stays in it



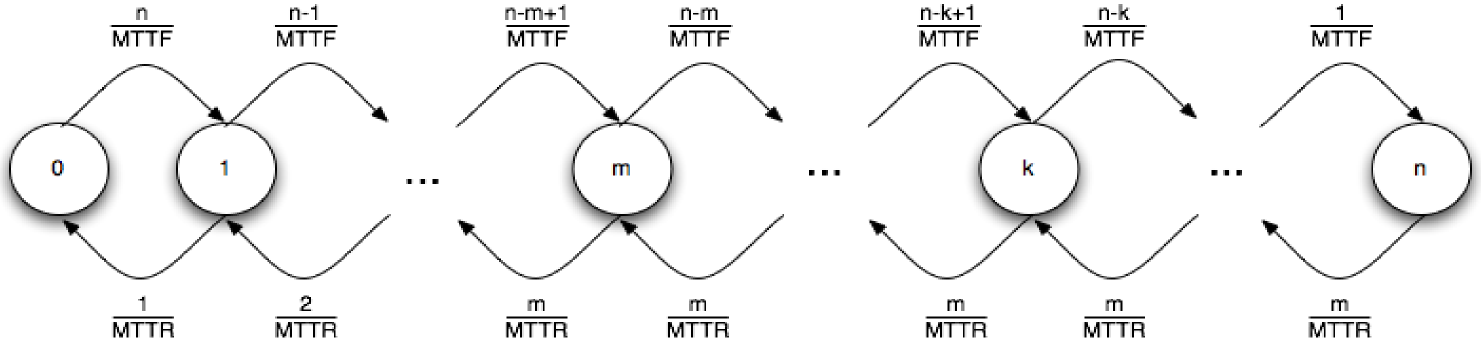
State Transition Diagrams: Application

- System of n parallel servers which ‚arrive‘ at repair situation (i.e., fail)
  - Maximum number *m* of parallel repair activities
  - Maximum *k-out-of-n* servers are allowed to be failed
  - Arrival rate == failure rate
  - Completion rate == repair rate
  - State: number of servers down
  - Transitions: a component failure or a component back in operation



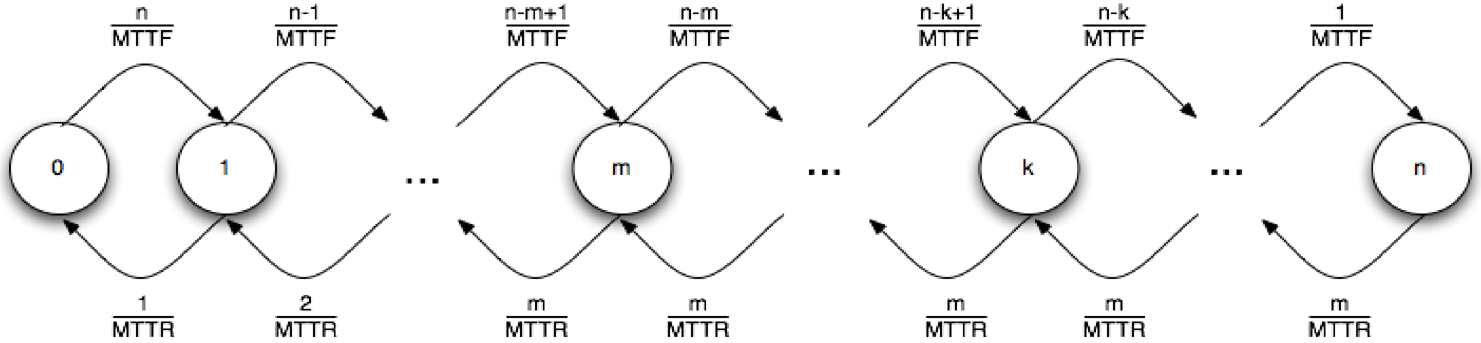
$$\lambda_k = \frac{n-k}{MTTF}$$

$$\mu_k = \begin{Bmatrix} k/MTTR & k = 1, ..., m \\ m/MTTR & k = m+1, ..., n \end{Bmatrix}$$



State Transition Diagrams: Analysis

Probability that k servers are down	p_k
Site availability	$A = 1 - p_n$
Average number of working servers	$N = \sum_{k=0}^{n-1} (n-k) \times p_k$
Probability of more than k servers down	$P = \sum_{j=k+1}^{n} p_j$



Markov Chains

- Discrete random process, usually drawn as state transition diagram
- **Markov property:** next step depends only on the current step

$$P(X_{n+1}|X_1, X_2, \dots, X_n) = P(X_{n+1}|X_n)$$

  - Impossible to predict future states, but useful for statistical properties
  - Finite state space (chain), transitions with probabilities, initial state probabilities
- **Transient state**: probability > 0 to not return to this state (finite number of visits)
- **Recurrent state**: probability of 1 to return to this state after unspecified time t
  - *Mean recurrence time* can be used as MTTF metric
- **Time-homogeneous Markov chains:** transition probabilities/rates do not change in time

$$P(X_{n+1} = x|X_n = y) = P(X_n = x|X_{n-1} = y)$$

Markov Chains: Time Model

Discrete-time Markov chain (DTMC)

- State changes after fixed time intervals
- Discrete parameter space, discrete state space
- System is in exactly one state
- Transition to next state depends on *transition probability* at t
- Probability transition matrix
  - Rows: flow out of that state
  - Columns: flow into the state
  - Rows and columns sum up to 1

Continuous-time Markov chain (CTMC)

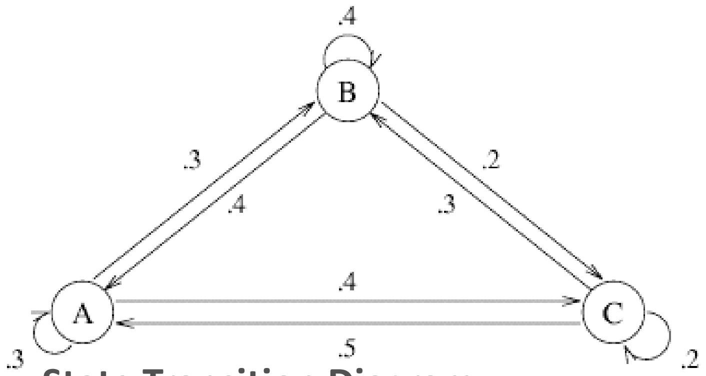
- State changes at any point in time
- Continuous parameter space, discrete state space
- Transition to next state after spending some time in a state (holding time)
- *Transition rates* instead of probabilities
- Transition rate / generator matrix Q
  - q_{ij} : rate departing from i and arriving in j
  - q_{ii} : -(total rate out of i) → no state change
  - Rows sum up to 0

Markov Chains: DTMC

- Each row sum of the transition matrix is 1
- For each step: apply transition matrix to state probability vector

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} .3 & .3 & .4 \\ .4 & .4 & .2 \\ .5 & .3 & .2 \end{bmatrix} \end{array} = S \qquad q_{i,j} = \lim_{\Delta t \to 0} \frac{P\{X_{t+\Delta t} = j \,|\, X_t = i\}}{\Delta t} \qquad i \neq j$$

Transition Matrix



State Transition Diagram

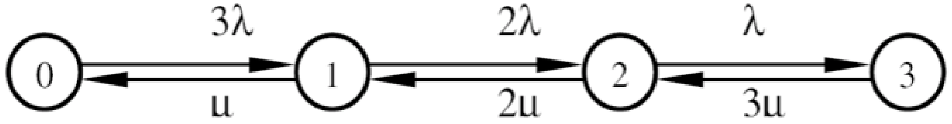
Initial state probabilities (distribution vector) $\begin{bmatrix} .\overline{3} & .\overline{3} & .\overline{3} \end{bmatrix} \begin{bmatrix} .3 & .3 & .4 \\ .4 & .4 & .2 \\ .5 & .3 & .2 \end{bmatrix} = \begin{bmatrix} .4 & .\overline{3} & .2\overline{6} \end{bmatrix}$

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} .41 & .33 & .26 \\ .38 & .34 & .28 \\ .37 & .33 & .30 \end{bmatrix} \end{array} = S^2$$

Probability Matrix after 2 steps

(C) Tamara Lynn Anthony

Dependability Modelling with CTMCs



- **State**: represents a particular error state
  - E.g., number of failed components at any given time
- **Transition**: assigned with component failure rate
  - *Time-homogeneous* process: failure / repair rates do not change over time
  - Failure / repair events are stochastically independent, process is memory-less
- Each row sum is 0
  - Probability mass flowing out of a state will go to some other state
- **Stationary Distribution**: the probability distribution to which the chain converges after a long time
  - E.g., the availability distribution

Example

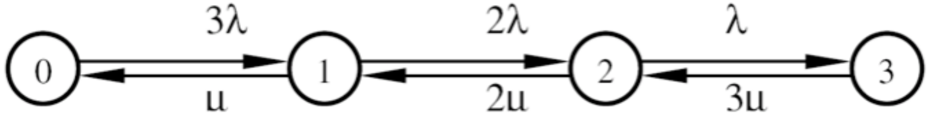
Consider a *k-out-of-n* system with *n* components.

- Their failure rates are distributed following the bathtub curve
- Their repair rates are exponentially distributed

- Would (and can) you model this system using
  - Time-homogenous DTMC?
  - Time-homogenous CTMC?
  - Time-inhomogenous CTMC?
- In a Markov chain modelling this system, which are recurring states?

Example: Availability Analysis

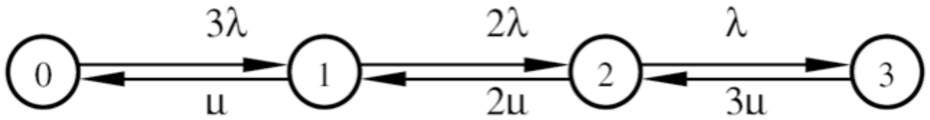
- Interested in **steady-state availability** of the system
  - Interpretation as steady-state probability for the system being operational at t
  - Derived from **probability vector**: steady-state probabilities for the system being in one of the failure states after a number of steps
- “Static” steady-state availability computable if probabilities are in **equilibrium**
  - Probability for leaving state is similar to probability for going into that state - probability mass is evenly distributed
  - Typically achieved after a high number of steps



$$Q = \begin{pmatrix} -3\lambda & 3\lambda & 0 & 0 \\ \mu & -\mu - 2\lambda & 2\lambda & 0 \\ 0 & 2\mu & -2\mu - \lambda & \lambda \\ 0 & 0 & 3\mu & -3\mu \end{pmatrix}$$

Example: 2-out-of-3 System

$$Q = \begin{pmatrix} -3\lambda & 3\lambda & 0 & 0 \\ \mu & -\mu - 2\lambda & 2\lambda & 0 \\ 0 & 2\mu & -2\mu - \lambda & \lambda \\ 0 & 0 & 3\mu & -3\mu \end{pmatrix}$$



1. Balance equations (steady-state equilibrium criterion):

Equilibrium: P(leaving s_0) = P(entering s_0) $3\lambda s_0 = \mu s_1$

$$3\lambda s_0 + 2\mu s_2 = \mu s_1 + 2\lambda s_1$$
$$2\lambda s_1 + 3\mu s_3 = 2\mu s_2 + \lambda s_2$$
$$\lambda s_2 = 3\mu s_3$$
$$s_0 + s_1 + s_2 + s_3 = 1$$

solve for s_i

$$s_0 = \frac{\mu}{3\lambda} s_1$$
$$s_2 = \frac{\lambda}{\mu} s_1$$
$$s_3 = \frac{\lambda^2}{3\mu^2} s_1$$

$$s_1 = \frac{3\mu^2\lambda}{(\mu+\lambda)^3}$$

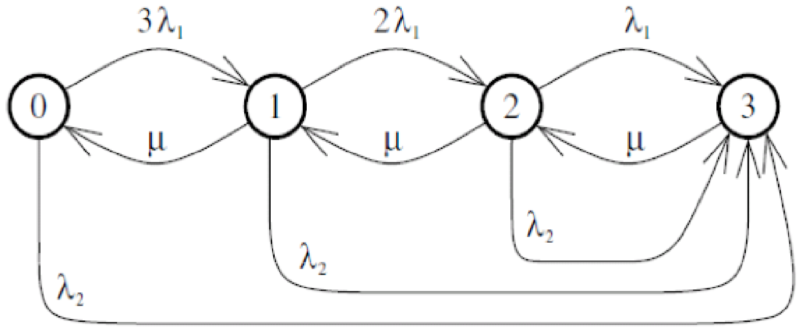
2. Compute per-state steady-state probabilities:

$$s_0 = \frac{\mu^3}{(\mu+\lambda)^3}; s_1 = \frac{3\mu^2\lambda}{(\mu+\lambda)^3}; s_2 = \frac{3\mu\lambda^2}{(\mu+\lambda)^3}; s_3 = \frac{\lambda^3}{(\mu+\lambda)^3}$$

3. 2-out-of-3 availability:

< 2 failed nodes: $A = s_0 + s_1 = \frac{\mu^2(\mu + 3\lambda)}{(\mu+\lambda)^3} = 3a^2 + 2a^3 \quad a = \frac{\mu}{(\mu+\lambda)}$

Markov Chains: Complexity



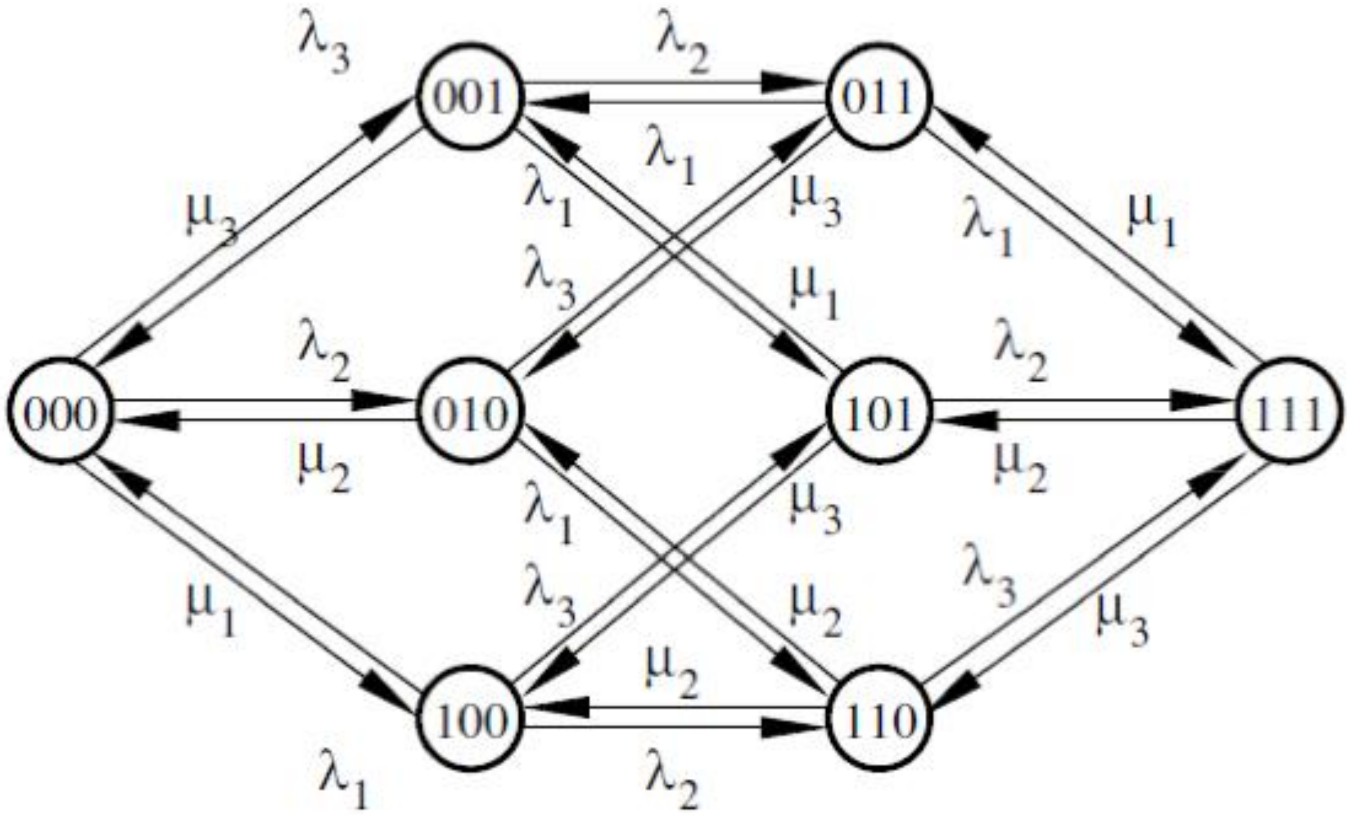
- Resulting formula equals result from Boolean investigation
- Markov chains also support non-independent events
  - Common cause failures
- Markov chains sizes grow *exponentially* with their number of components - which is bad

  → Divide-and-conquer: decompose and aggregate chain parts
  - **Structural decomposition:** consider a system as set of independent subsystems
  - **Behavioral decomposition:** assume time constants for some fault occurrences and handling processes based on criticality - e.g. fault in parked airplane

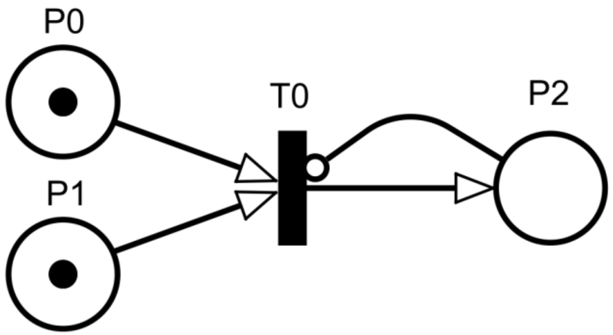
Markov Chains: Complexity

3-component model, where each component has its own failure and repair rate

→ 2^3 states



Petri Nets



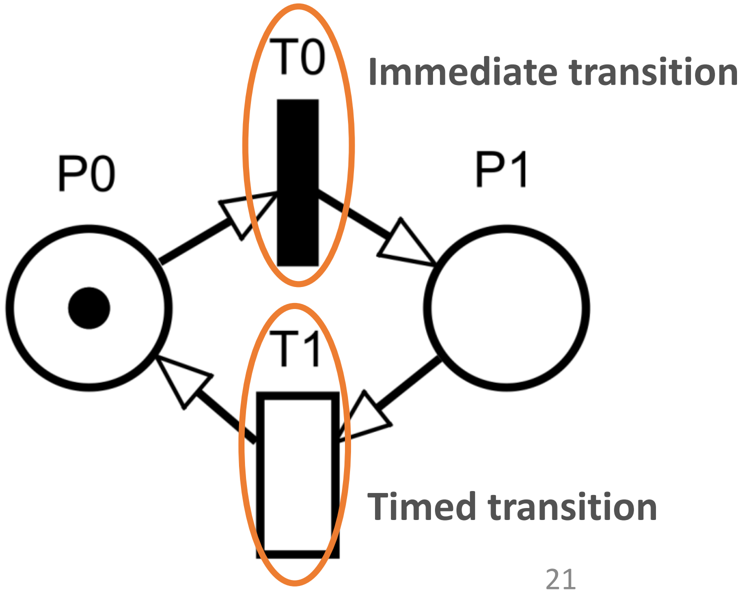
- Modelling language for concurrent, distributed systems
- Bipartite graph
  - **Places** contain **tokens** (marking) → *state*
  - **Transitions** consume & produce tokens → *behaviour*
- Simultaneous enabling of multiple transitions: *concurrent behaviour*
- Transition firing: consume tokens from input places, produce tokens in output places
  - Places: *pre/post-conditions* for state changes
  - If all input places contain tokens, a transition is *enabled*
    - Necessary number determined by arc cardinality
  - **Inhibitor arcs** disable transitions if tokens lie in their origin places
- **Conflict**: When two transitions need the same token, only one can fire
  - Resolved by priorities (absolute) or weights (randomized)
  - E.g., competing for resources

Petri Nets – Conceptual Mapping

Input Places	**Transitions**	**Output Places**
Required Resources	Task	Freed Resources
Input Data	Computations	Output Data
Input Signals	Signal Processing	Output Signals
Buffers / Registers	Processor	Buffers / Registers

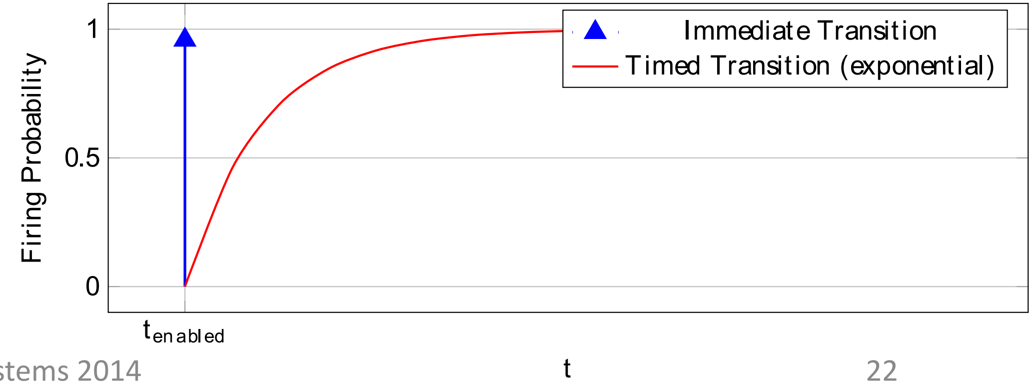
Stochastic Petri Nets

- Extend petri nets by stochastic temporal properties
  - Delayed transition firing
  - Temporal properties allow to study quantitative, time-dependent metrics
    - E.g., MTTF
  - Event propagation can be modelled in time, not just logically
    → Increased expressiveness
- **Generalized Stochastic Petri Nets** (GSPN)
  - Immediate transitions: fire immediately
  - Timed transitions: fire with stochastic delay
  - Model in *continuous time*
    - Marking corresponds to a continuous probability distribution
    - For simulation, *time discretization* becomes necessary

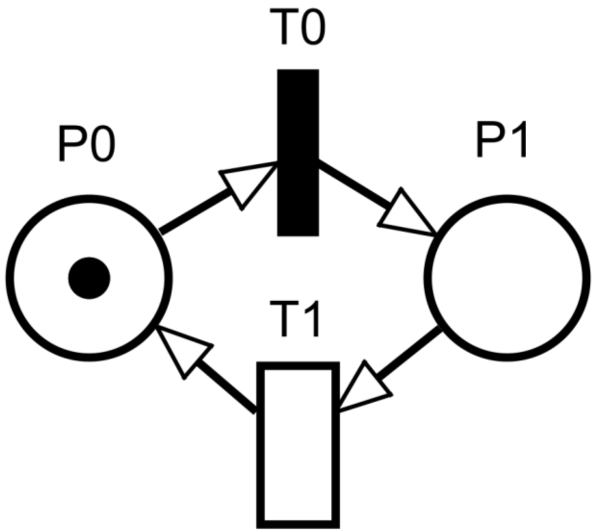


Stochastic Petri Nets: Transitions

- **Immediate transitions**
  - Model logical inter-dependencies, e.g. error propagation chains
  - Fire ‘in no time’, *0-Dirac distribution*
- **Timed transitions**
  - Model stochastic behaviour, e.g. random component failure
  - Delayed firing, defined by probability distribution in continuous time
  - For GSPN: *exponential distribution*
  - Firing policies: when to sample the delay?
    - Race with enabling memory: at enabling time
    - Race with age memory: at firing time

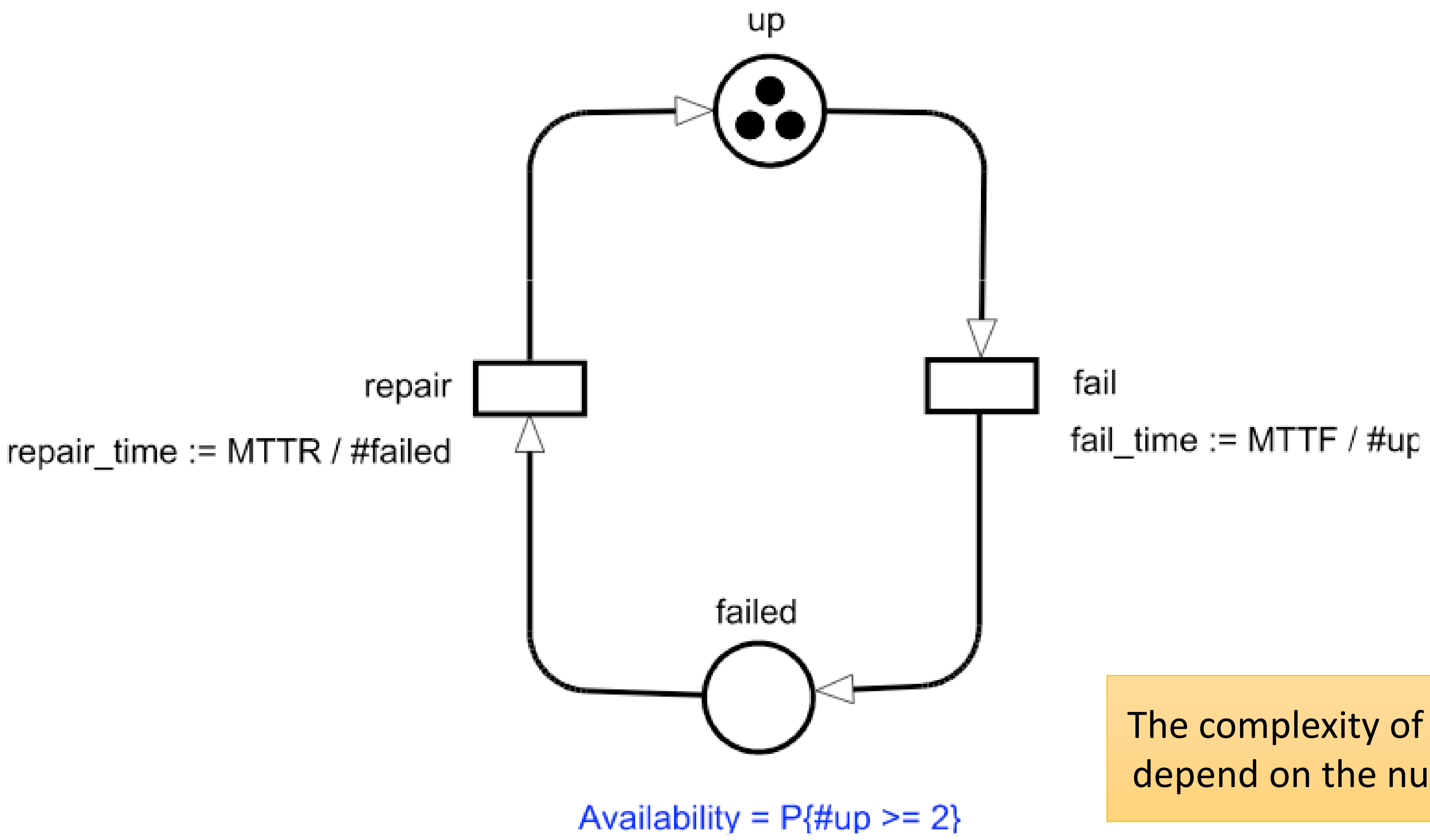


Stochastic Petri Nets – Properties



- **Reachability set**
  - Contains all possible markings reachable from the initial marking
  - Analysis questions:
    - Can some system state (e.g. an error state) be reached at all?
    - Does a firing sequence exist, that transforms M0 to M?
- **Vanishing marking (GSPN)**
  - A marking that is abandoned again at once, due to immediate transition firing
  - Probability of observing this marking: 0 (continuous time)
- **Tangible marking (GSPN)**
  - A marking that the net remains in for some time
- **Reduced Reachability graph**
  - Graph of *reachable tangible markings* from the initial marking
- **Boundedness**
  - A place is *k-bounded* if for every reachable marking, the number of tokens in it does not exceed k
  - A net is k-bounded if all places are k-bounded
  - Useful for modeling limited (bounded) resources

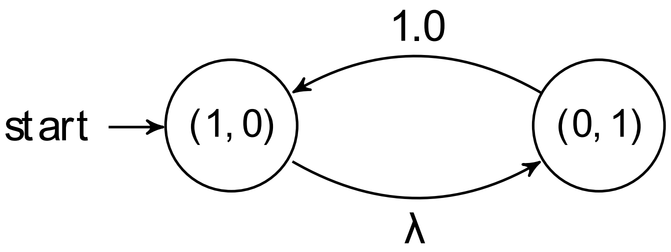
Example: 2-of-3 System

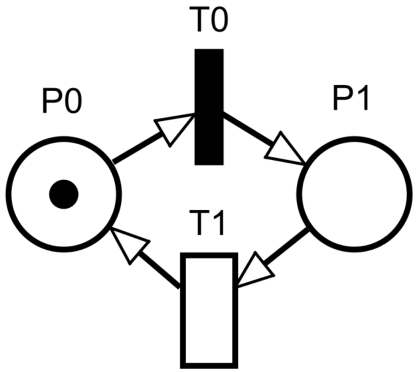


The complexity of the petri net does not depend on the number of components!

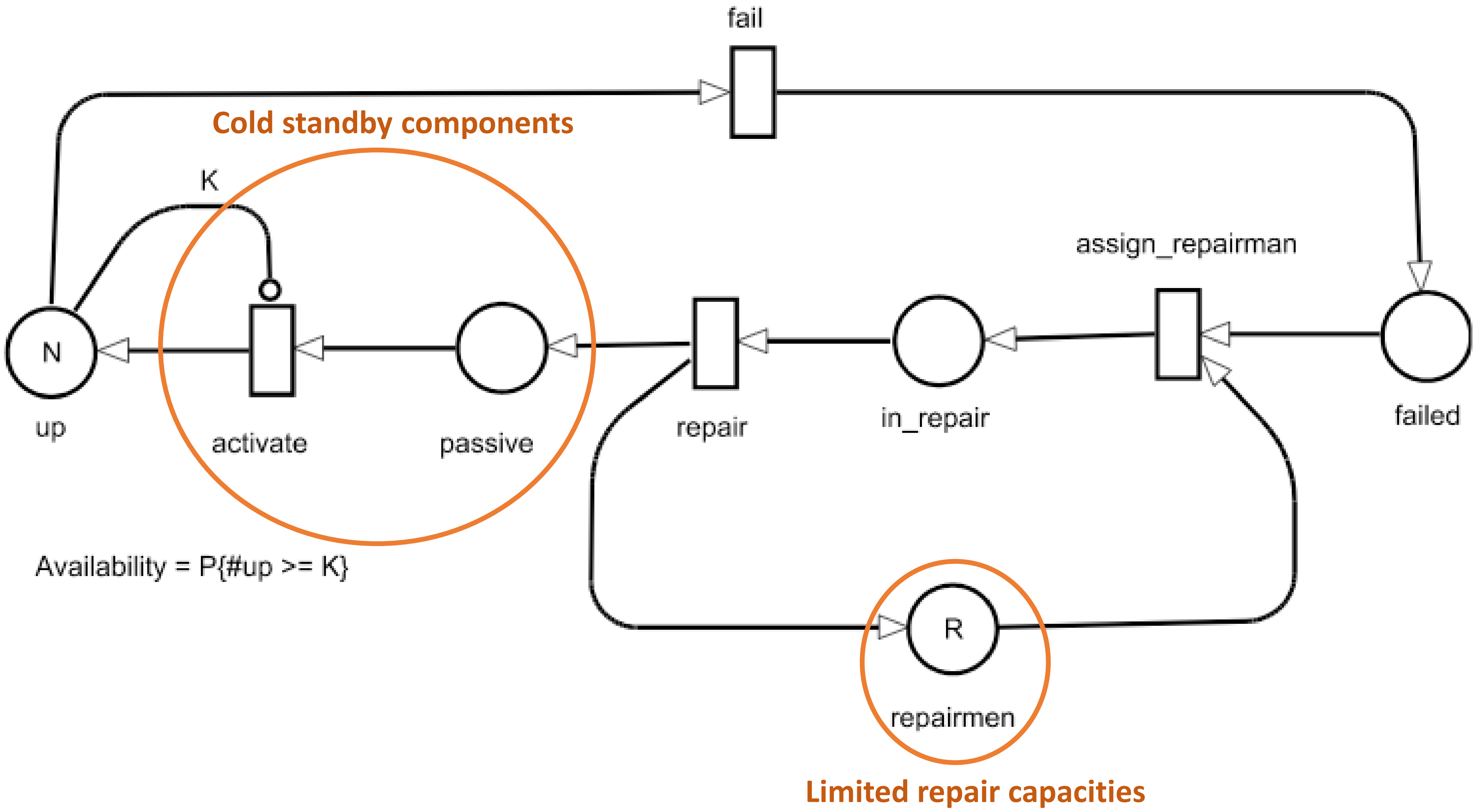
Stochastic Petri Nets vs Markov Chains

- **Reachability graphs** of GSPN are isomorphic to CTMC
- GSPN → "compact representation" of a CTMC
  - CTMC: one node per state (exponential growth with #components)
  - GSPN: one marking per state (linear growth with #components)
- GSPN simulation
  - Traverse underlying CTMC at random
  - No need to generate all states beforehand

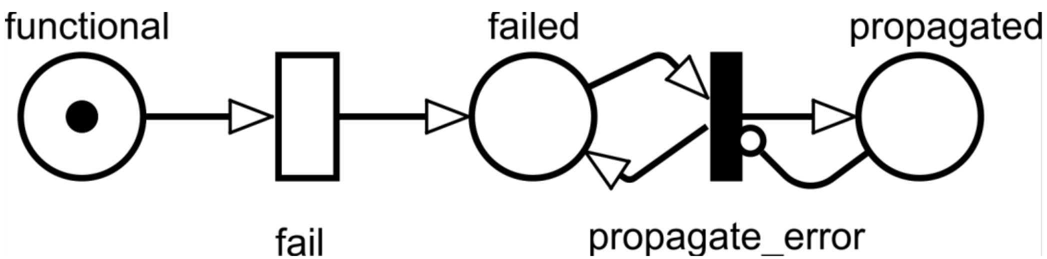




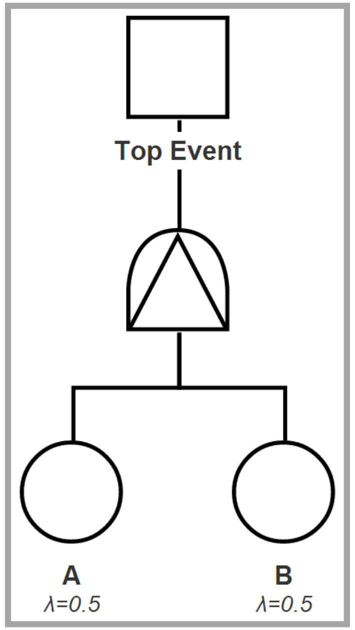
Example: K-of-N with Standby and Repairmen

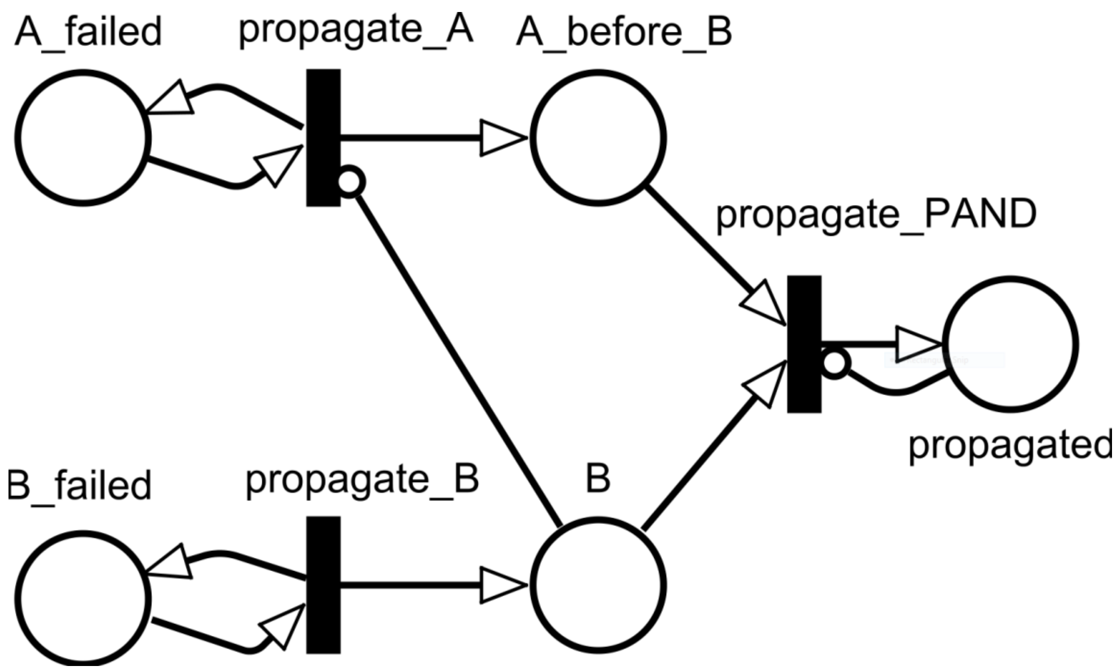


Example: Priority AND → Stochastic Petri Net



Basic Event

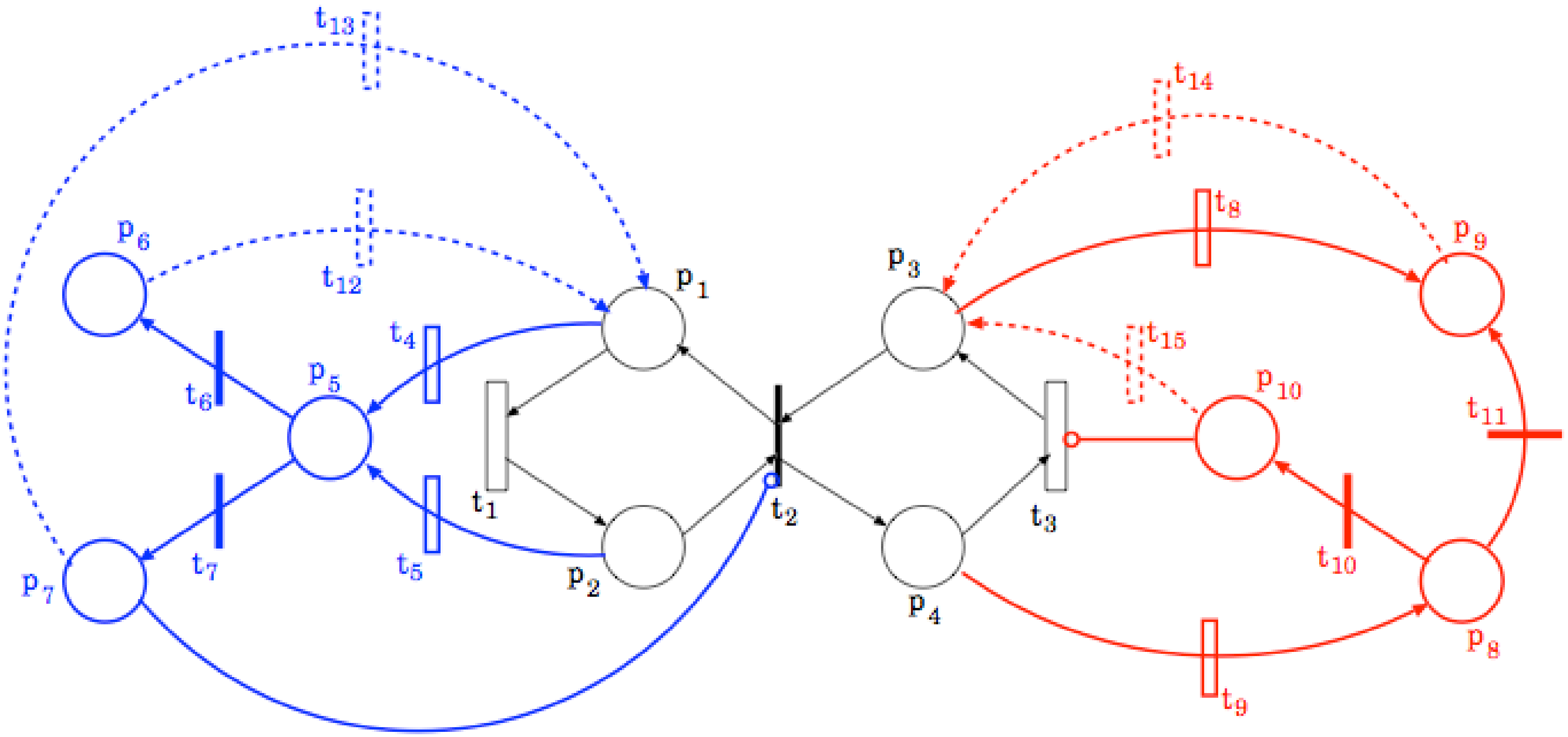




PAND Gate

Example: Parallel System with Input Buffer

p_1	Free buffer stage	
p_2	Occupied buffer stage	
p_3	Idle unit	
p_4	Active unit	
p_5	Failed buffer stage	
p_6	Recovered buffer stage failure	
p_7	Unrecovered buffer stage failure	
p_8	Failed active unit	
p_9	Recovered unit failure	
p_{10}	Unrecovered unit failure	
		firing rate
t_1	Buffer stage becomes occupied	λ
t_2	Transfer from buffer to unit	*immed.*
t_3	Unit ends a task	$m_4\,\mu$
t_4	Free buffer stage fails	$m_1\,\gamma_4$
t_5	Occupied buffer stage fails	$m_2\,\gamma_5$
t_6	Buffer stage failure is recovered	v_B
t_7	Buffer stage failure is not recovered	$(1 - v_B)$
t_8	Idle unit fails	$m_3\,\gamma_8$
t_9	Active unit fails	$m_4\,\gamma_9$
t_{10}	Unit failure is not recovered	$(1 - v_U)$
t_{11}	Unit failure is recovered	v_U
t_{12}	Repair of recovered buffer stage	ρ_{12}
t_{13}	Repair of unrecovered buffer stage	ρ_{13}
t_{14}	Repair of recovered unit	ρ_{14}
t_{15}	Repair of unrecovered unit	ρ_{15}



Light lines: Fault free operation
Heavy lines: Failures
Dotted lines: Repairs

Petri Net Simulation vs Analysis

Computational analysis

- *Compute* static properties of the net
- Probability of an event defined through place markings
  - Add up the probabilities of all markings in which the condition corresponding to the event definition holds true
- Requires construction of reachability graph → **state space explosion**
- Additional challenges
  - Transition guard functions
  - Non-exponential distributions

Simulation

- Execute the model to randomly *explore* the state space
- Play the "token game" many times (Monte Carlo approaches)
- Challenges
  - **Rare event simulation**: small failure rates → importance sampling
  - Random number generation
  - Verification of results (statistical tests)

Petri Net Simulation: Token Game

```
bool PetriNetSimulation::simulationStep(PetriNet* pn, int tick)
{
    bool immediateCanFire = true;
    while (immediateCanFire)
        tryImmediateTransitions(pn, tick, immediateCanFire);

    tryTimedTransitions(pn, tick);

    vector<TimedTransition*> toRemove;
    for (TimedTransition* tt : pn->m_inactiveTimedTransitions)
    {
        if (tt->tryUpdateStartupTime(tick))
        { // tt has become enabled -> start its timer
            toRemove.emplace_back(tt);
            pn->updateFiringTime(tt); // register in event queue
        }
    }
    for (TimedTransition* tt : toRemove)
        pn->m_inactiveTimedTransitions.erase(tt);

    return !pn->markingInvalid();
}
```

immediate transitions
(error propagation)

timed transitions
(basic events)

update event queue

Rare Event Simulation: Importance Sampling

- Problem: naïve simulation is inefficient for very rare events
  - Such as simulating components with low failure rates
  - Monte Carlo methods with moderately many rounds have *high variance*
  - Many rounds needed to achieve a desired confidence level

Importance Sampling: Compute $\mathrm{p} = E\big(\phi(X)\big)$, where $\phi(X)$ is a desired dependability metric, and X a rare random variable

- But sample from a different, not rare, distribution!
- Instead of sampling from $PDF(x)$, sample from $PDF^*(x)$
  - $PDF(x) > 0 \Rightarrow PDF^*(x) > 0$
  - Likelihood ratio: $w(x) = \frac{PDF(x)}{PDF^*(x)}$

→ $\mathrm{p} = E(\phi(X^*) * w(x))$

Importance Sampling

$$\mathcal{E}(\phi(x)) = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i) \quad , x_i \sim p$$

$$\approx \int \phi(x)\, p(x)\, dx = \int \phi(x) \underbrace{\frac{p(x)}{q(x)}}_{\omega(x)} q(x)\, dx$$

$$\approx \frac{1}{n}\sum_{i=1}^{n} \phi(x_i) \cdot \omega(x) \quad , x_i \sim q$$

$$\Rightarrow \underset{x \sim p}{E}(\phi(x)) = \underset{x \sim q}{E}(\phi(x') \cdot \omega(x))$$

Reliability Tools

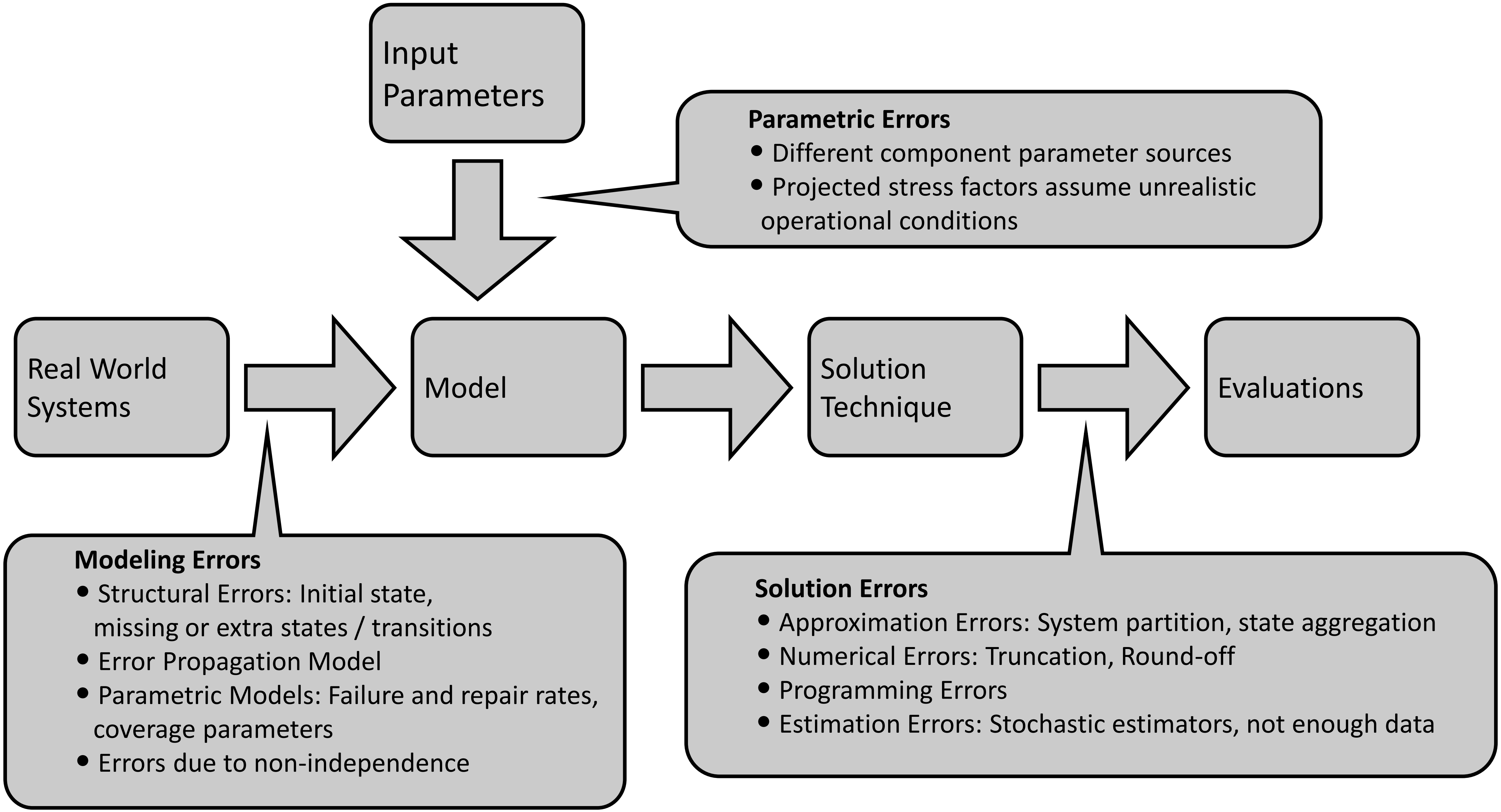
System structure
Fault classes
Failure rates
Fault handling procedures
Repair procedures
Success criteria

Model development

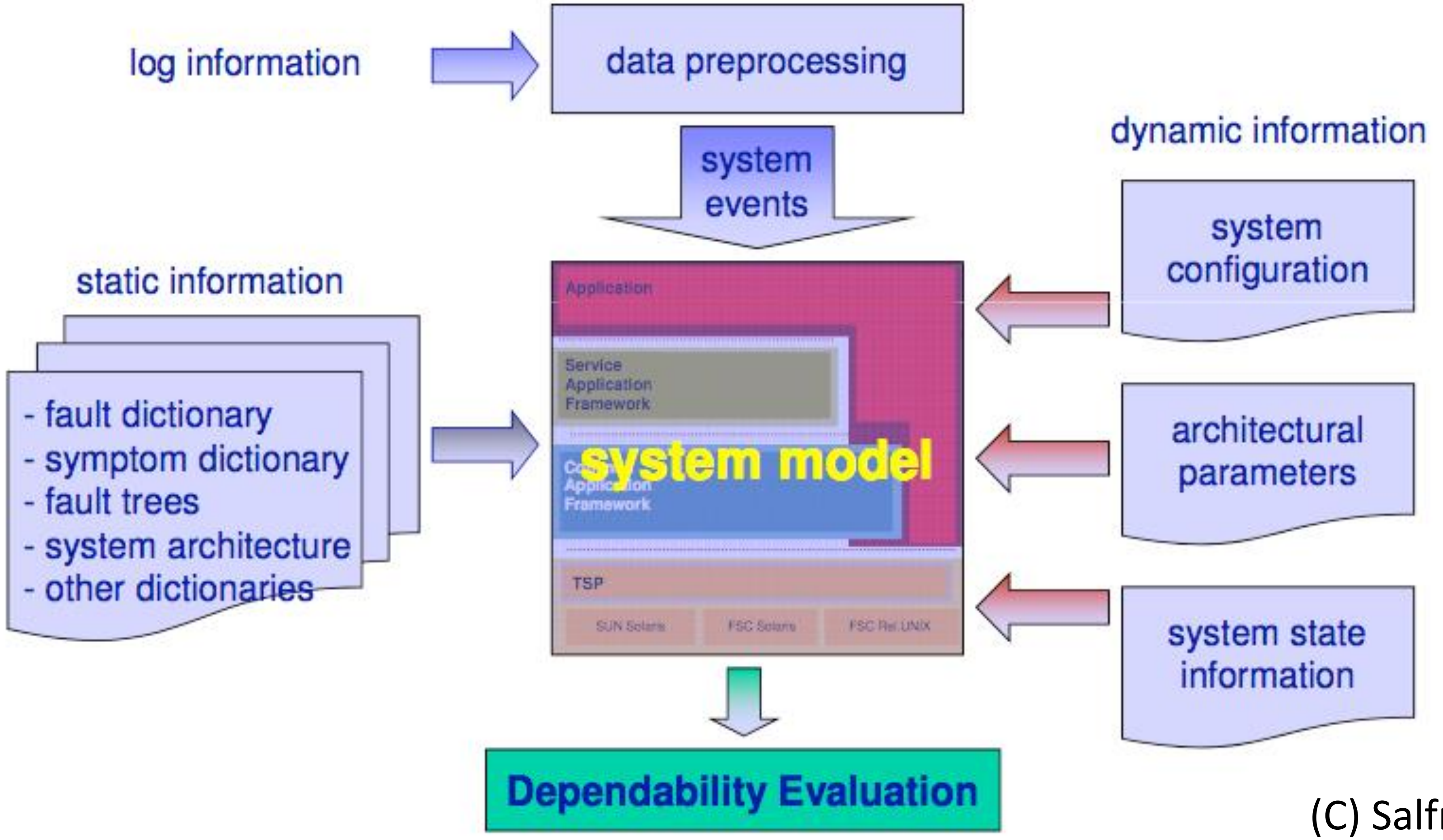
Model solution (combinatorial, simulation)

Dependability evaluation

Fault Coverage
Reliability
Availability
Life Cycle
MTTF / MTTR
Costs
…


Input
Parameters
Parametric Errors
• Different component parameter sources
• Projected stress factors assume unrealistic operational conditions
Real World Systems
Model
Solution Technique
Evaluations
Modeling Errors
• Structural Errors: Initial state, missing or extra states / transitions
• Error Propagation Model
• Parametric Models: Failure and repair rates, coverage parameters
• Errors due to non-independence
Solution Errors
• Approximation Errors: System partition, state aggregation
• Numerical Errors: Truncation, Round-off
• Programming Errors
• Estimation Errors: Stochastic estimators, not enough data

Runtime Dependability Evaluation

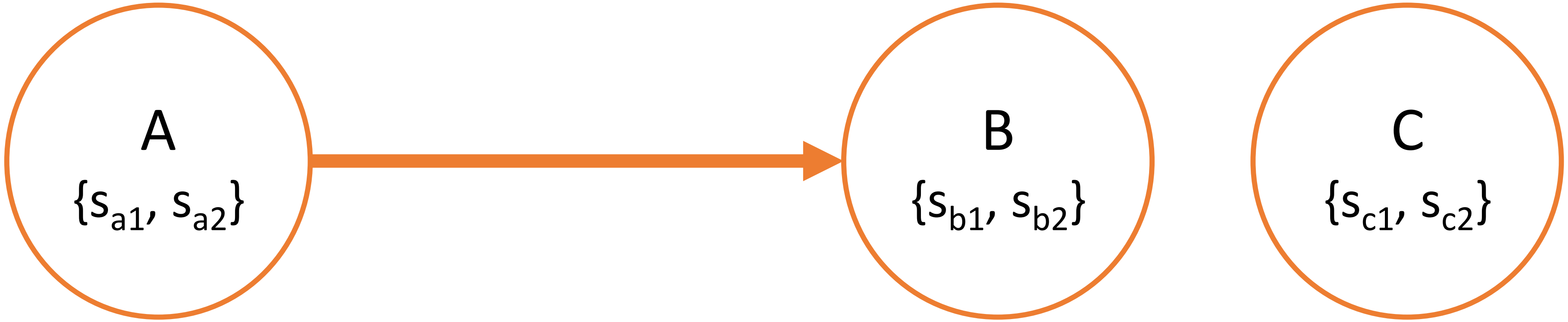


(C) Salfner

Bayesian Networks

- Encode **uncertain expert knowledge**
- Encode **causality relationships**
- Given a set of random variables, find **Joint Probability Distribution (JPD)**
- Bayesian approach:
  prior probability + likelihood → posterior probability
- Applications to dependability modeling
  - *Error propagation chains* as causality relationships:
    What is the probability of an overall error state, given prior per-component failure probabilities?
  - Online fault diagnosis
    What is the probability of observing the current system state, given that a processor is faulty/non-faulty?

Bayesian Networks



s_{a1}	s_{a2}
$P(A = s_{a1})$	$P(A = s_{a2})$

CPT (B)	s_{a1}	s_{a2}
s_{b1}	$P(B = s_{b1} \mid A = s_{a1})$	$P(B = s_{b1} \mid A = s_{a2})$
s_{b2}	$P(B = s_{b2} \mid A = s_{a1})$	$P(B = s_{b2} \mid A = s_{a2})$

s_{c1}	s_{c2}
$P(C = s_{c1})$	$P(A = s_{c2})$

$$\boldsymbol{JPD}: \quad P(x_1, \ldots, x_n) = \prod_{i}^{n} P(x_i \mid parents(X_i))$$

$$\text{e.g.}: \quad P(s_{a1} \wedge s_{b1} \wedge s_{c1}) = P(s_{a1}) P(s_{b1} | s_{a1}) P(s_{c1})$$

Example: Fault Tree → Bayesian Network