

Modeling using SANs

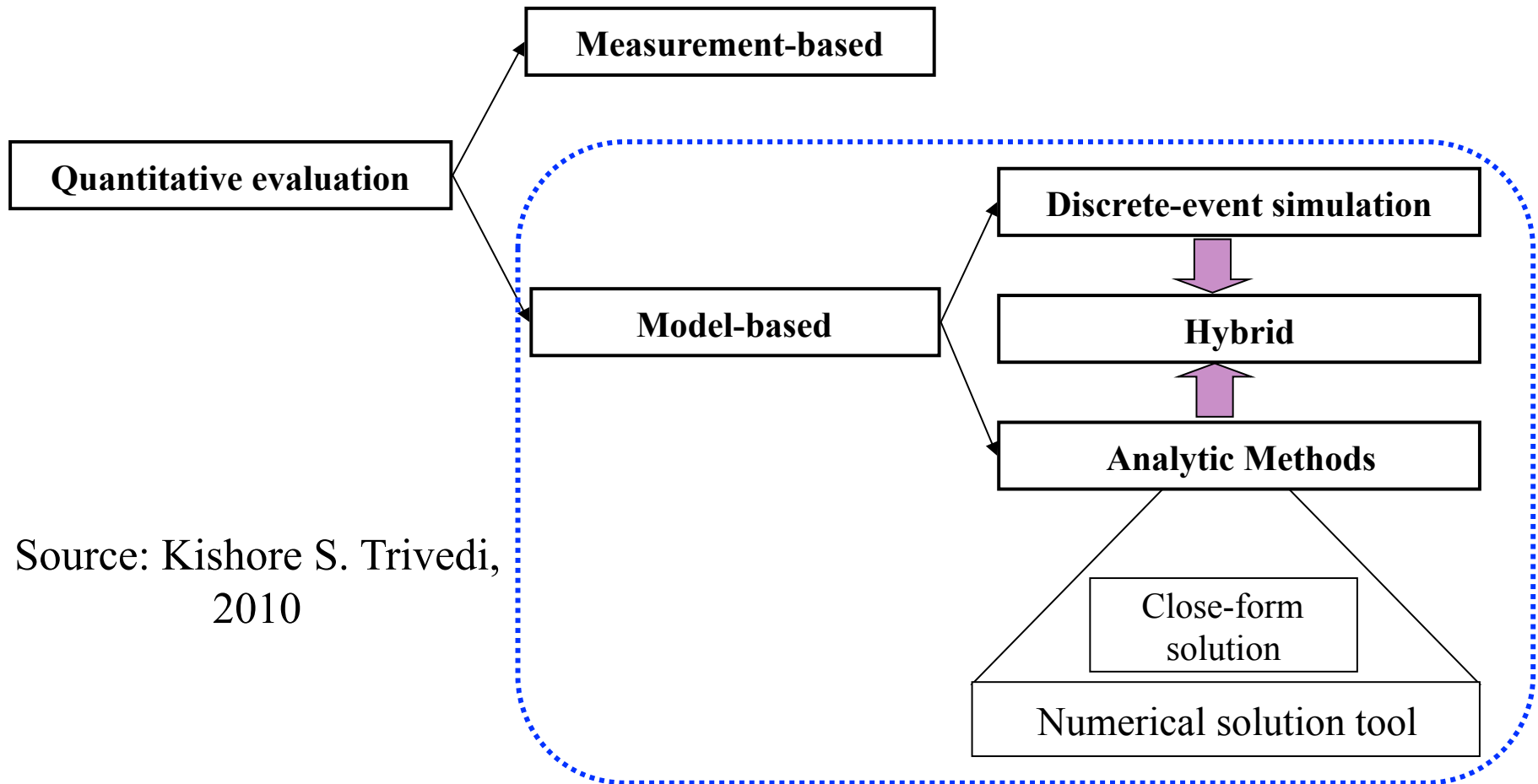
EECE 513: Design of Fault-tolerant
Digital Systems

(based on Bill Sander's slides at UIUC
and Saurabh Bagchi's slides at Purdue)

Learning Objectives

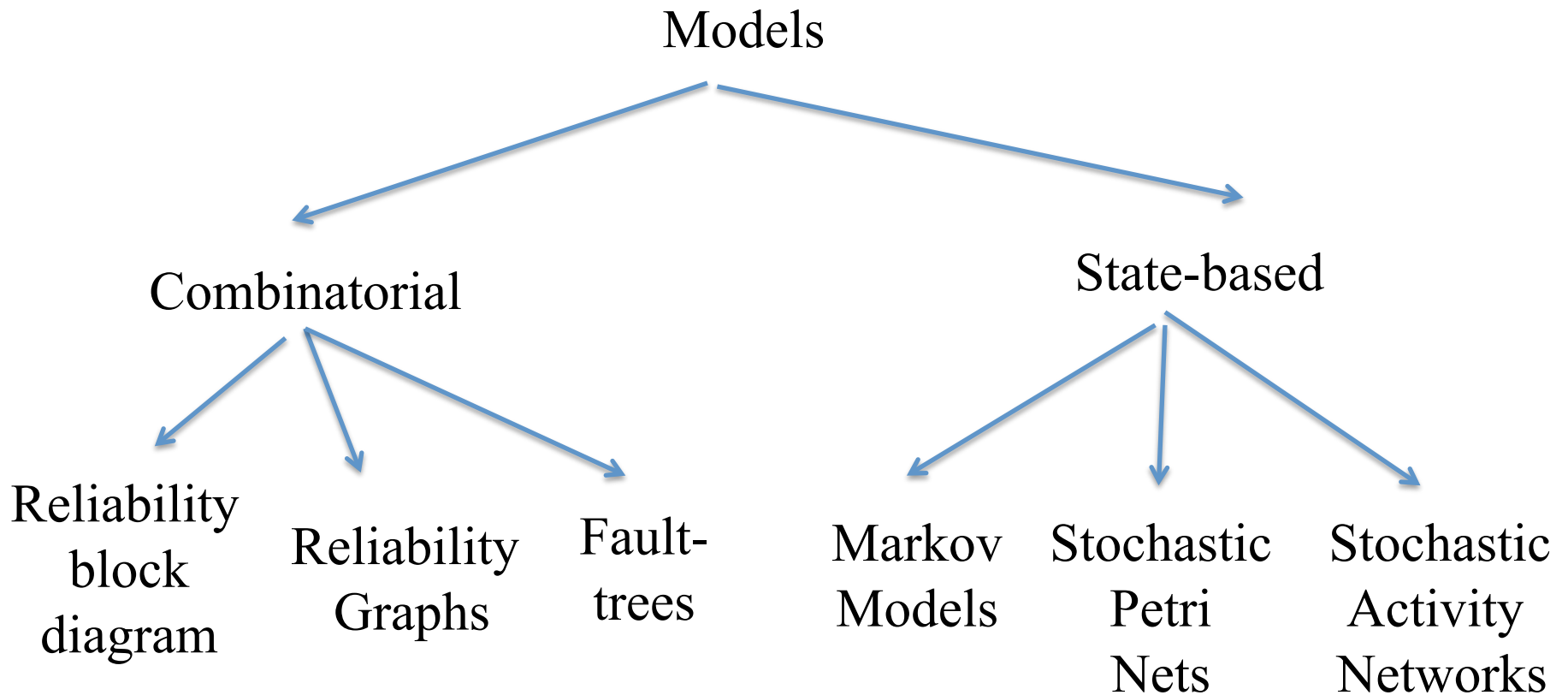
- Build reliability models of systems with discrete time Markov models and solve them
- Build reliability models of systems with continuous time Markov models
- Model systems using Stochastic Petri nets
- Model systems using Stochastic Activity nets

Reliability Evaluation Methods



Source: Kishore S. Trivedi,
2010

Types of Models



State-Space Models

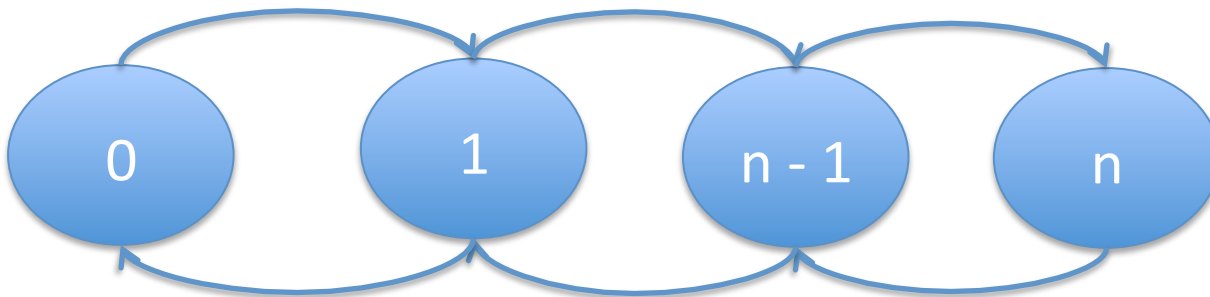
- States can keep track of:
 - Number of functioning resources of each type
 - States of recovery for each failed resource
 - Number of tasks of each type waiting at each resource
 - Allocation of resources to tasks
- A transition:
 - Can occur from any state to any other state
 - Can represent a simple or a compound event

Why State-based Models ?

- States can keep track of history
 - System's dependence on time is explicit
 - Many real systems need this
- Do not need to limit to independent faults
- Can be solved using analytical methods (if exponential) or discrete event simulation

Markov Chains - 1

- **Markov process**: Probability distribution for future state only depends on the present state, not on how the process arrived at the present state
- **Markov chain**: State space is discrete
- **Discrete time Markov chain**: Time is discrete



Markov Chains - 2

Let X_0, X_1, \dots, X_n : observed state at discrete times t_0, t_1, \dots, t_n

$X_n = j \Rightarrow$ system state at time step n is j .

$$\begin{aligned} P(X_n = i_n \mid X_0 = i_0, X_1 = i_1, \dots, X_{n-1} = i_{n-1}) \\ = P(X_n = i_n \mid X_{n-1} = i_{n-1}) \quad (\text{Markov Property}) \end{aligned}$$

$p_{jk}(m, n) \equiv P(X_n = k \mid X_m = j), 0 \leq m \leq n$ (conditional pmf)

$p_j(n) \equiv P(X_n = j)$ (unconditional pmf)

Homogeneous Markov Chain

- **Homogeneous Markov chain:** $p_{jk}(m,n)$ only depends on $n-m$
- **n-step transition probability:** $p_{jk}(n) = P(X_{m+n} = k \mid X_m = j)$
 - 1-step transition probability: $p_{jk} = p_{jk}(1) = P(X_n = k \mid X_{n-1} = j)$
- **Initial probability row vector:** $\mathbf{p}(0) = [p_0(0), p_1(0), \dots, p_k(0), \dots]$
- Transition probability matrix:

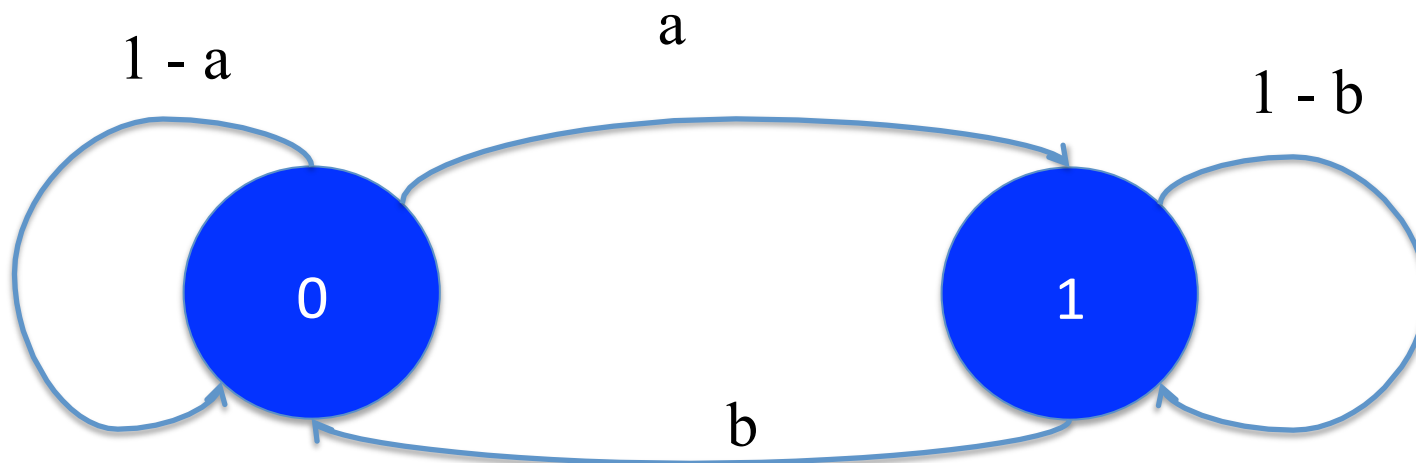
$$P = [p_{ij}] = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdot & \cdot \\ p_{10} & p_{11} & p_{12} & \cdot & \cdot \\ \vdots & \vdots & \vdots & \cdot & \cdot \end{bmatrix}$$

Markov Chain Example

System is in state 0 if it is operational, state 1 if it is undergoing repair. Transitions occur only at the end of a day. It's a homogeneous DTMC.

1. Prob of failing = a
2. Prob of repair = b

$$P = \begin{bmatrix} 1-a & a \\ b & 1-b \end{bmatrix}, 0 \leq a, b \leq 1$$



Computation of n-step Transition Probabilities

- For a DTMC, find $p_{ij}(n) = P(X_{m+n} = j | X_m = i)$
- Events: State reaches k (from i) & reaches j (from k) are independent due to the Markov property
- Invoking the theorem of total probability:

$$p_{ij}(m+n) = \sum_k p_{ik}(m) p_{kj}(n)$$

- i.e., the Markov chain has to be somewhere at $m+n$
- Let $P(n)$: n -step prob. transition matrix (i,j) th entry is $p_{ij}(n)$. Making $m=1, n=n-1$ in the above equation:

$$P(n) = P.P(n-1) = P^n$$

$$p_j(n) = P(X_n = j) = \sum_i P(X_0 = i) P(X_n = j | X_0 = i) = \sum_i p_i(0) p_{ij}(n)$$

Computation of n-step Transition Probabilities

- The pmf of the random variable X_n is

$$p_j(n) = P(X_n = j) = \sum_i P(X_0 = i)P(X_n = j|X_0 = i) = \sum_i p_i(0)p_{ij}(n)$$

$$\mathbf{p}(n) = [p_0(n), p_1(n), \dots, p_j(n), \dots]$$

- j , in general can assume countable values, 0,1,2, Defining,

- $p_j(n)$ for $j=0,1,2,\dots,k,\dots$ can be written in the vector form as, $[p_0 p_1 p_2 \dots p_j \dots](n) =$

$$\underbrace{[p_0 \ p_1 \ p_2 \ \dots \ p_j \ \dots]}_{\mathbf{p}(0)} \begin{bmatrix} p_{00}(n) & p_{01}(n) & \dots & p_{0j}(n) & \dots \\ p_{10}(n) & p_{11}(n) & \dots & p_{1j}(n) & \dots \\ p_{20}(n) & p_{21}(n) & \dots & p_{2j}(n) & \dots \\ \vdots & \vdots & & \vdots & \vdots \\ p_{i0}(n) & p_{i1}(n) & \dots & p_{ij}(n) & \dots \\ \vdots & \vdots & & \vdots & \vdots \end{bmatrix}$$

Two state Markov chain example

- For a two state DTMC, say the transition prob. matrix is

$$P = \begin{bmatrix} 1-a & b \\ b & 1-b \end{bmatrix}, \quad 0 \leq a, b \leq 1, \quad |1-a-b| < 1$$

- Then, the n-step transition probability matrix $P(n) = P^n$ is

$$P(n) = \begin{bmatrix} \frac{b+a(1-a-b)^n}{a+b} & \frac{a-a(1-a-b)^n}{a+b} \\ \frac{b-b(1-a-b)^n}{a+b} & \frac{a+b(1-a-b)^n}{a+b} \end{bmatrix}$$

Exercise

Consider a communication network consisting of a sequence of binary communication channels, with $a=1/4$, $b=1/2$. Here X_n denotes the bit leaving the n th stage and X_0 represents the bit entering the system.

1. If a bit enters as 1, what is the probability of being correctly transmitted over (a) two stages (b) three stages?
2. Assuming initial probabilities $P(X_0=0) = 1/3$ and $P(X_0=1) = 2/3$, what is $\mathbf{p}(n)$?
3. What happens as n goes to infinity ?

Steady state probability distribution: Theorems

- For any irreducible, aperiodic Markov chain, the limiting state probabilities $p(n)$ as $n \rightarrow \infty$,
 - exist, and
 - are independent of the initial prob. vector
- The limiting probability vector v is called the steady state probability vector:

$$v = v P$$

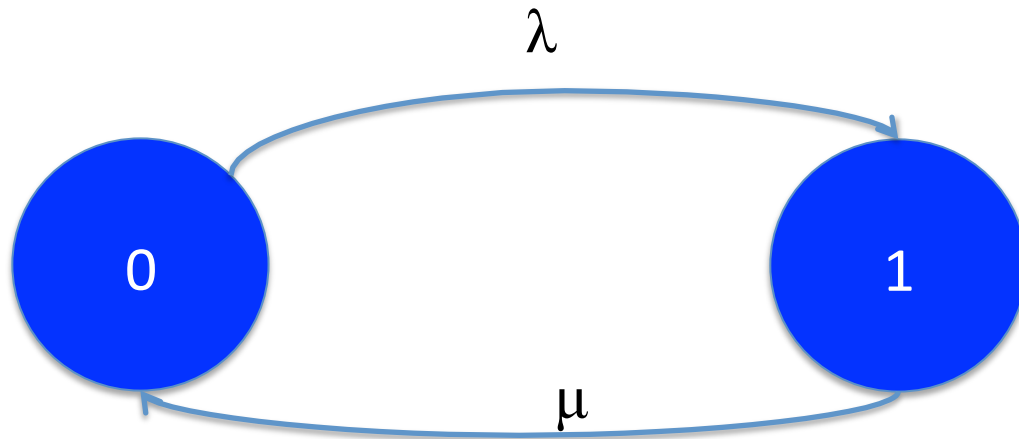
Learning Objectives

- Build reliability models of systems with discrete time Markov models and solve them
- Build reliability models of systems with continuous time Markov models
- Model systems using Stochastic Petri nets
- Model systems using Stochastic Activity nets

CTMCs

- An important limitation of DTMCs is that events can occur only at discrete times
 - Cannot model continuous events such as failures
 - Need to artificially subdivide time into discrete units
- **CTMCS:** Markov models in which events can occur continuously in time (at any instance)
 - But we assume that the transition probabilities are exponentially distributed for mathematical tractability

CTMC: Example



- Edge weights do not represent probabilities
- No self-loops in any of the states
- Sum of outflows not equal to 1

CTMC: Transfer matrix

$$q_{i,i} = - \sum_{j \neq i} q_{i,j}$$

$$\sum_i q_{x,i} = 0 \quad \forall x$$

$$Q = \begin{pmatrix} -(\sum_{j \neq 1} q_{1,j}) & q_{1,2} & \cdots & q_{1,n} \\ q_{2,1} & -(\sum_{j \neq 2} q_{2,j}) & \cdots & q_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n,1} & q_{n,2} & \cdots & -(\sum_{j \neq n} q_{n,j}) \end{pmatrix}$$

For the simple CTMC considered earlier,

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

CTMC: Exercise

- A computer is in one of two states, working or idle. When it is idle, jobs arrive at the rate α and they're completed at the rate β . When the computer is working, it fails with rate λ_w . When it is idle, it fails with rate λ_i . Also, the computer is repaired with rate μ .
- Draw the CTMC corresponding to the computer and specify its rate matrix.

Transfer Matrix: Interpretation

- In a CTMC, there is no notion of “one step”
- So one way to think of the matrix Q is as a set of competing alarm clocks and derive the equivalent DTMC’s transition matrix (next)
- This deviates from the standard view of CTMC as differential equations, but is more intuitive.

Converting CTMCs to DTMCs - 1

- Assume that each transition is an alarm clock that competes with other transitions' clocks
 - If the alarm clock fires, the transition is triggered
 - Each clock fires at random, exp. distributed times
- At each state 'i', the clocks associated with the transitions to the next state 'j' get activated. The first to fire is the successful transition.

Converting CTMCs to DTMCs - 2

- Because times are exponentially distributed,
 - No two timers go off at the same time $P(T_i = t) = 0$,
 - Minimum of exponentially distributed r.v.'s is an exponentially distributed r.v with rate $\lambda_{\min} = \sum \lambda_i$

$$P(T_i \leq t) = 1 - e^{-\nu_i t}, \quad t \geq 0,$$

where,

$$\nu_i \equiv -Q_{i,i} = \sum_{j \neq i} Q_{i,j},$$

Prob. that the i^{th} random variable is the min. one is

$$P(N_i = j) = \frac{Q_{i,j}}{\sum_{k, k \neq i} Q_{i,k}} = \frac{Q_{i,j}}{\nu_i} \quad \text{for } j \neq i.$$

Converting CTMCs to DTMCs - 3

- Based on the previous results, we can convert the CTMC to a DTMC whose one-step transition matrix P is specified as follows:

$$P_{i,j} = P(N_i = j) = \frac{Q_{i,j}}{\sum_{k,k \neq i} Q_{i,k}} = \frac{Q_{i,j}}{\nu_i} \quad \text{for } j \neq i,$$

Based on the above equation, $P_{i,i} = ?$

This process can be used only for steady state.

Converting DTMC to CTMC

$$Q_{i,j} = \nu_i P_{i,j} \quad \text{for } j \neq i \quad \text{and} \quad Q_{i,i} = - \sum_{j:j \neq i} Q_{i,j} = \nu_i \quad \text{for all } i .$$

- The previous formulation can also help us go the other way. To get a DTMC from a CTMC.
- However, the conversion runs into trouble if you have a self loop in the DTMC. You can get rid of the self loop by adjusting the other probabilities in the DTMC accordingly.

$$\hat{P}_{i,j} = \frac{P_{i,j}}{1 - P_{i,i}} \quad \text{for all } i \quad \text{and } j . \quad \hat{\nu}_i = \nu_i (1 - P_{i,i}) .$$

Learning Objectives

- Build reliability models of systems with discrete time Markov models and solve them
- Build reliability models of systems with continuous time Markov models
- Model systems using Stochastic Petri nets
- Model systems using Stochastic Activity nets

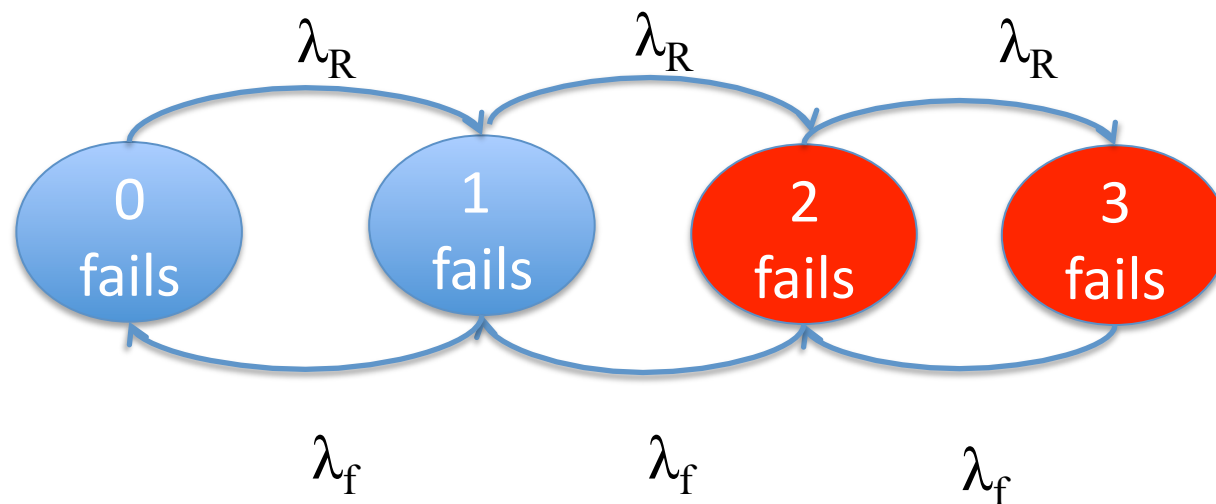
Stochastic Petri Nets

- **Main disadvantage of Markov chains**
 - Number of states grows exponentially with number of paths in the system
 - Unnecessarily repetitive because of global chains
- **Petri-Nets: Higher-level formalism for representing processes' behaviors**
 - Represent local state rather than global state
 - Can be converted to Markov chains automatically

SPNs Versus Markov Models - 1

Consider a system with three components, each of which can fail.

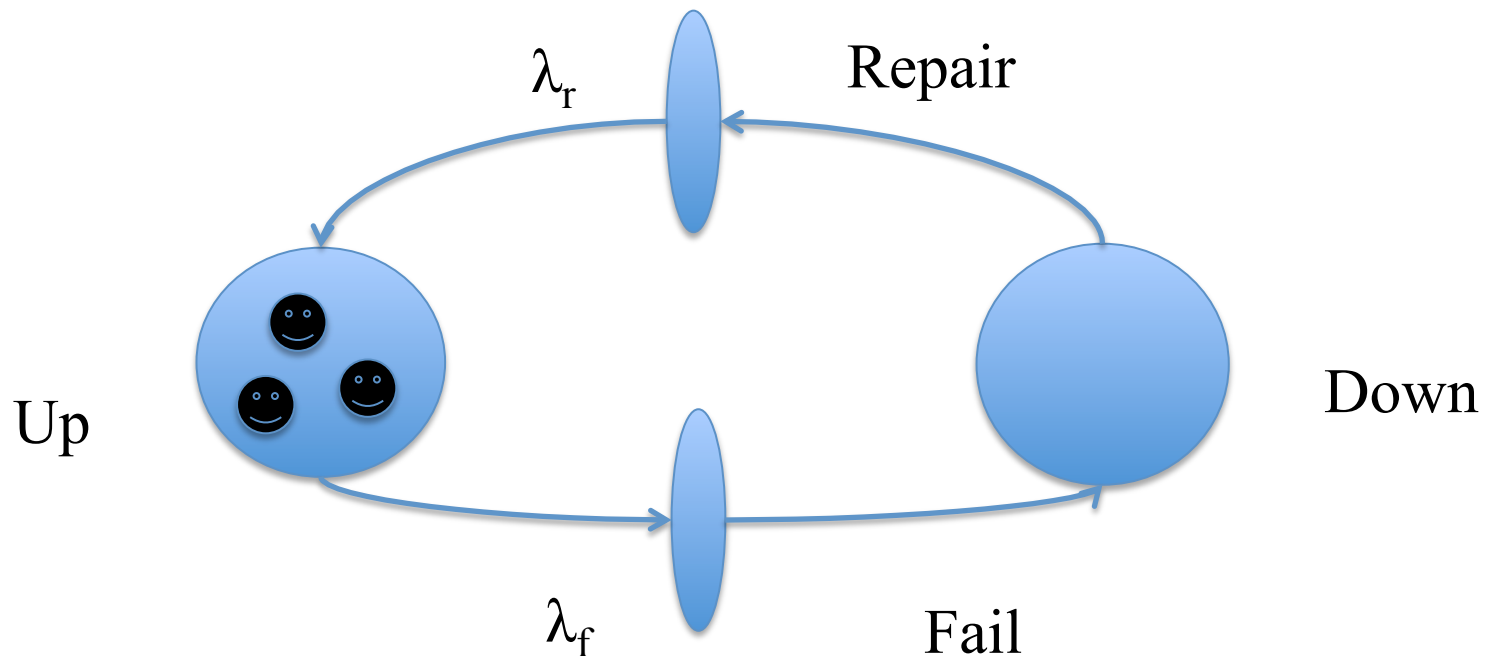
The system needs at least 2 components in order to work. The Markov chain representation would need to consider all possible combinations of up/down components and assign a state for each.



No. of states with n components = $n + 1$

SPNs Versus Markov Models - 2

The equivalent Petri-Net Formulation for the above system will be as follows. Note that there are only 2 states in the system (i.e., places), and the components are represented as tokens. This is so for any number of components.



Stochastic Petri Nets

One of the simplest high-level modeling formalisms is called *stochastic Petri nets*. A stochastic Petri net is composed of the following components:

- Places: which contain tokens, and are like variables



- tokens: which are the “value” or “state” of a place



- transitions:  which change the number of tokens in places

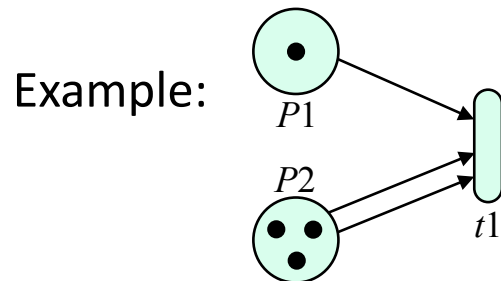
- input arcs:  which connect places to transitions

- output arcs:  which connect transitions to places

Firing Rules for SPNs

A stochastic Petri net (SPN) executes according to the following rules:

- A transition is said to be *enabled* if for each place connected by input arcs, the number of tokens in the place is \geq the number of input arcs connecting the place and the transition.

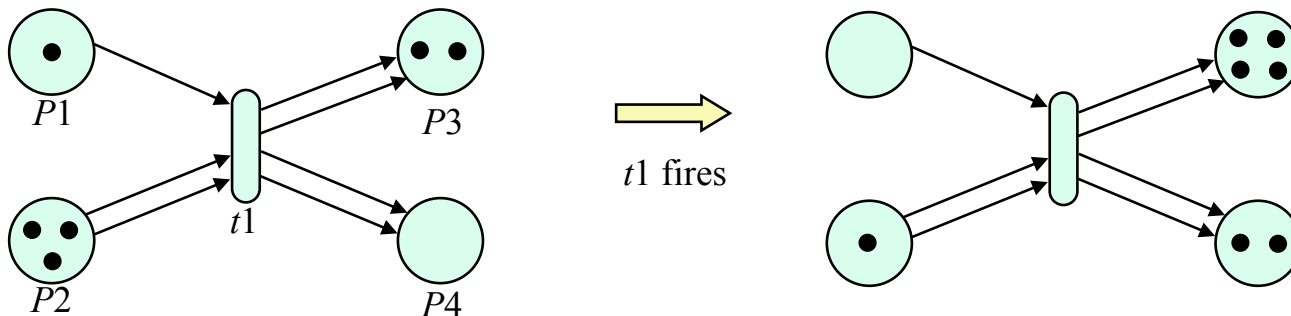


Transition $t1$ is enabled.

Firing Rules, cont.

- A transition may *fire* if it is enabled. (More about this later.)
- If a transition fires, for each input arc, a token is removed from the corresponding place, and for each output arc, a token is added to the corresponding place.

Example:



Specification of Stochastic Behavior of an SPN

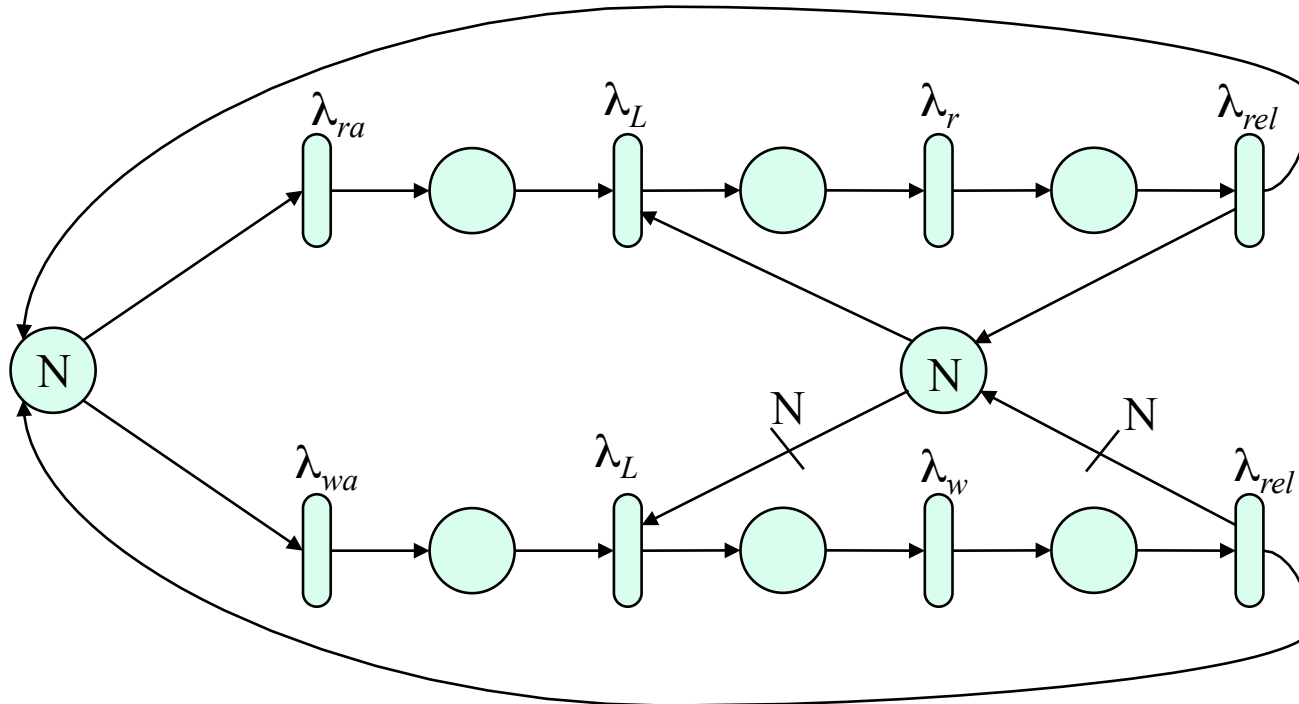
- A stochastic Petri net is made from a Petri net by
 - Assigning an exponentially distributed time to all transitions.
 - Time represents the “delay” between enabling and firing of a transition.
 - Transitions “execute” in parallel with independent delay distributions.
- Since the minimum of multiple independent exponentials is itself exponential, time between transition firings is exponential.
- If a transition t becomes enabled, and before t fires, some other transition fires and changes the state of the SPN such that t is no longer enabled, then t *aborts*, that is, t will not fire.
- Since the exponential distribution is memoryless, one can say that transitions that remain enabled continue or restart, as is convenient, without changing the behavior of the network.

SPN Example: Readers/Writers Problem

- There are at most N requests in the system at a time. Read requests arrive at rate λ_{ra} , and write requests at rate λ_{wa} . Any number of readers may read from a file at a time, but only one writer may write at a time. The writer and the readers may not access the file simultaneously.
- Locks are obtained with rate λ_L (for both read and write locks); reads and writes are performed at rates λ_r and λ_w respectively. Locks are released at rate λ_{rel} .



SPN Representation of Reader/Writers Problem



In the example, the top portion represents the readers while the bottom portion represents the writers. Each reader is represented by a single token.

Notes on SPNs

- SPNs can be converted into Markov chains automatically provided they use exponential distribution for the transition probabilities.
 - We skip the algorithm, but it's fairly straightforward.
- However, SPNs are limited in their expressive power: may only perform $+$, $-$, $>$, and test-for-zero operations. This may not be sufficient.
 - What if we wanted to express the condition that a writer may write in conjunction with certain readers but not others (say, due to conflicts) ?
 - Usually restricted to exponential arrival rates

Learning Objectives

- Build reliability models of systems with discrete time Markov models and solve them
- Build reliability models of systems with continuous time Markov models
- Model systems using Stochastic Petri nets
- Model systems using Stochastic Activity nets

Stochastic Activity Networks

The need for more expressive modeling languages has led to several extensions to stochastic Petri nets. One extension that we will examine is called *stochastic activity networks*. Because there are a number of subtle distinctions relative to SPNs, stochastic activity networks use different words to describe ideas similar to those of SPNs.

Stochastic activity networks have the following properties:

- A general way to specify that an activity (transition) is enabled
- A general way to specify a completion (firing) rule
- A way to represent zero-timed events
- A way to represent probabilistic choices upon activity completion
- State-dependent parameter values
- General delay distributions on activities

SAN Symbols

Stochastic activity networks (hereafter SANs) have four new symbols in addition to those of SPNs:

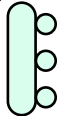
- Input gate: to define complex enabling predicates



- Output gate: to define complex completion functions



- Cases: (small circles on activities) to specify probabilistic choices



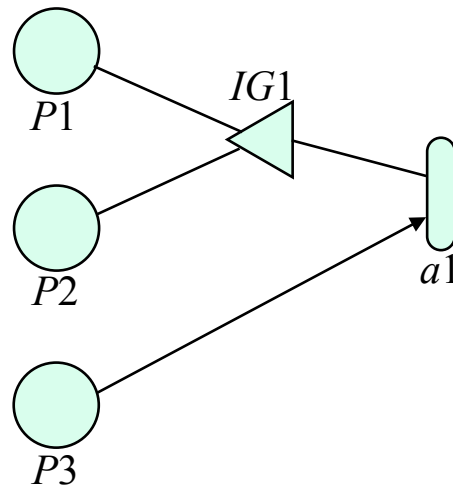
- Instantaneous activities: to specify zero-timed events



SAN Enabling Rules

- An input gate has two components:
 1. `enabling_function (state) → boolean`; also called the enabling predicate and is written in C-like language
 2. `input_function(state) → state`; rule for changing the state of the model. Also written in C-like language
- An activity is enabled if for every connected input gate, the enabling predicate is true, and for each input arc, the number of tokens in the connected place \geq number of arcs.
- We use the notation `MARK(P)` to denote the number of tokens in place `P`

Example SAN Enabling Rule



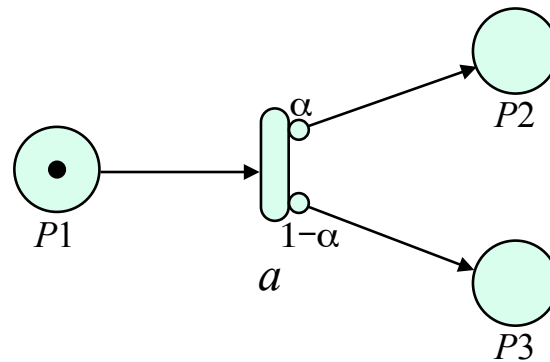
IG1 Predicate:

```
if ( (MARK(P1) > 0 && MARK(P2) == 0) ||  
      (MARK(P1) == 0 && MARK(P2) > 0) )  
    return 1;  
else return 0;
```

Activity $a1$ is enabled if $IG1$ predicate is true (1) and $MARK(P3) > 0$.
(Note that "1" is used to denote true.)

Cases

Cases represent a probabilistic choice of an action to take when an activity completes.

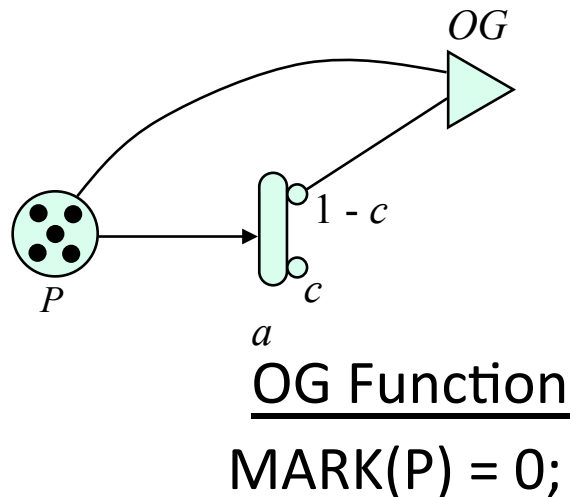


When activity a completes, a token is removed from place $P1$, and with probability α , a token is put into place $P2$, and with probability $1 - \alpha$, a token is put into place $P3$.

Note: cases are numbered, starting with 1, from top to bottom.

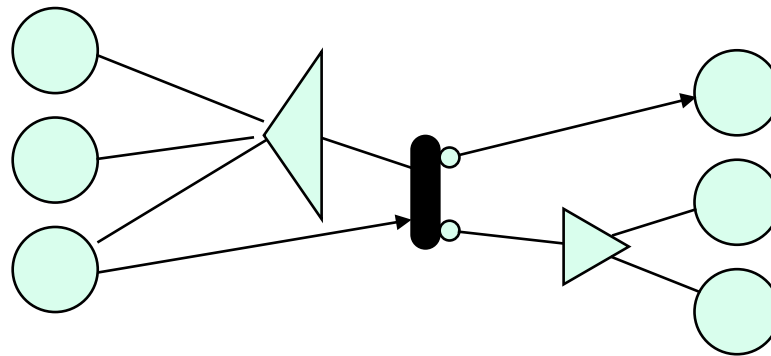
Output Gates

When an activity completes, an output gate allows for a more general change in the state of the system. This output gate function is usually expressed using pseudo-C code.



Instantaneous Activities

Another important feature of SANs is the instantaneous activity. An *instantaneous activity* is like a normal activity except that it completes in zero time after it becomes enabled. Instantaneous activities can be used with input gates, output gates, and cases.



Instantaneous activities are useful when modeling events that have an effect on the state of the system, but happen in negligible time, with respect to other activities in the system, and the performance/dependability measures.

SAN Terms

1. *activation* - time at which an activity begins
2. *completion* - time at which activity completes
3. *abort* - time, after activation but before completion, when activity is no longer enabled
4. *active* - the time after an activity has been activated but before it completes or aborts.

Completion Rules

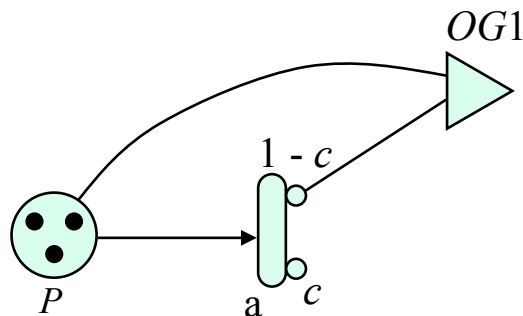
- When an activity completes, the following events take place (in the order listed), possibly changing the marking of the network:
 - 1. If the activity has cases, a case is (probabilistically) chosen.
 - 2. The functions of all the connected input gates are executed (in an unspecified order).
 - 3. Tokens are removed from places connected by input arcs.
 - 4. The functions of all the output gates connected to the chosen case are executed (in an unspecified order).
 - 5. Tokens are added to places connected by output arcs connected to the chosen case.
- Ordering is important, since effect of actions can be marking-dependent.

Marking Dependent Behavior

Virtually every parameter may be any function of the state of the model.
Examples of these are

- rates of exponential activities
- parameters of other activity distributions
- case probabilities

An example of this usefulness is a model of three redundant computers where the probability that a single computer crashing crashes the whole system (due to loss of coverage) increases after a failure.



	a
case 1	$0.1 + 0.02 * \text{MARK}(P)$
case 2	$0.9 - 0.02 * \text{MARK}(P)$

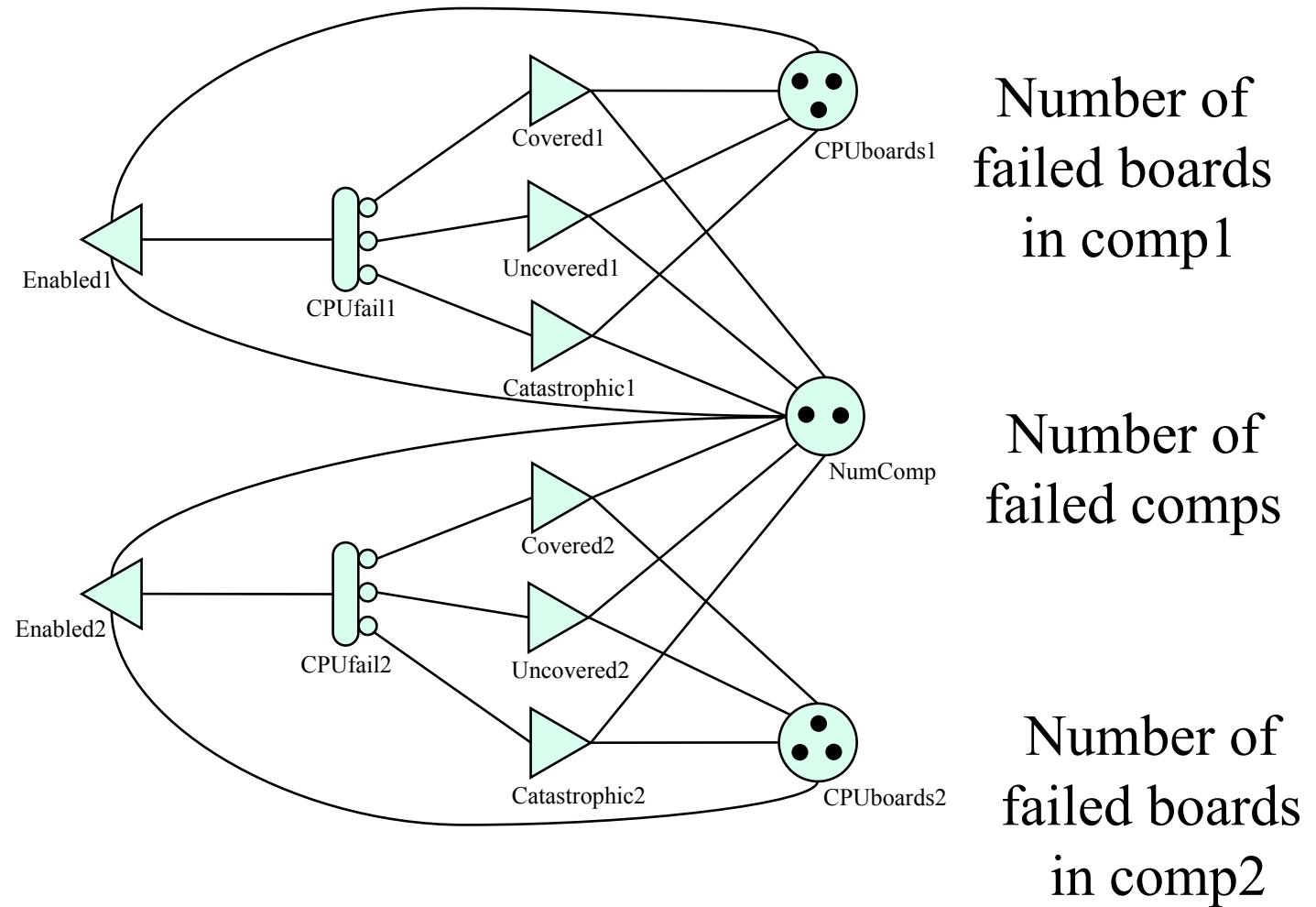
Fault-Tolerant Computer Failure Model

Example

A fault-tolerant computer system is made up of two redundant computers. Each computer is composed of three redundant CPU boards. A computer is operational if at least 1 CPU board is operational, and the system is operational if at least 1 computer is operational.

CPU boards fail at a rate of $1/10^6$ hours, and there is a 0.5% chance that a board failure will cause a computer failure, and a 0.8% chance that a board will fail in a way that causes a catastrophic system failure.

SAN for Computer Failure Model



Activity Case Probabilities and Input Gate Definition

<i>Activity</i>	<i>Case</i>	<i>Probability</i>
<i>CPUfail</i>	<i>1</i>	<i>0.987</i>
	<i>2</i>	<i>0.005</i>
	<i>3</i>	<i>0.008</i>

<i>Gate</i>	<i>Definition</i>
<i>Enabled1</i>	<u>Predicate</u> $MARK(CPUboard1 > 0) \ \&\& \ MARK(NumComp) > 0$
	<u>Function</u> $MARK(CPUboard1) \ \text{--};$

Output Gate Definitions

<i>Gate</i>	<i>Definition</i>
<i>Covered1</i>	<u>Function</u> $if (MARK(CPUboards1) == 0)$ $MARK(NumComp)--;$
<i>Uncovered1</i>	<u>Function</u> $MARK(CPUboards1) = 0$ $MARK(NumComp)--;$
<i>Catastrophic1</i>	<u>Function</u> $MARK(CPUboards1) = 0;$ $MARK(NumComp) = 0;$

Note that the output gate does not encode the probabilities

Reward Models

- Way of measuring performance or dependability features of a model
- Examples
 - Expected time until service
 - System availability
 - Number of misrouted packets in a time interval
 - Processor utilization
 - Length of downtime

Reward Models : Types

- Rate Rewards
 - Earned when the model is in a certain state or set of states for a period of time
 - Example: Amount of time a system is in the “up” state(s) is a measure of its availability
- Impulse Rewards
 - Earned when certain activities in the model complete
 - Example: Time until the occurrence of the first failure (activity) is a measure of its MTTF

$$\text{Total Reward} = \text{Rate Reward} + \text{Impulse Reward}$$

Reward Models: Computer System Example

Rate Reward

- Reliability $R(t)$

Amount of time until both systems have failed

MARK(NumComp) ≥ 0

Impulse Reward

- Number of board failures

How many board failures can occur in a certain time ?

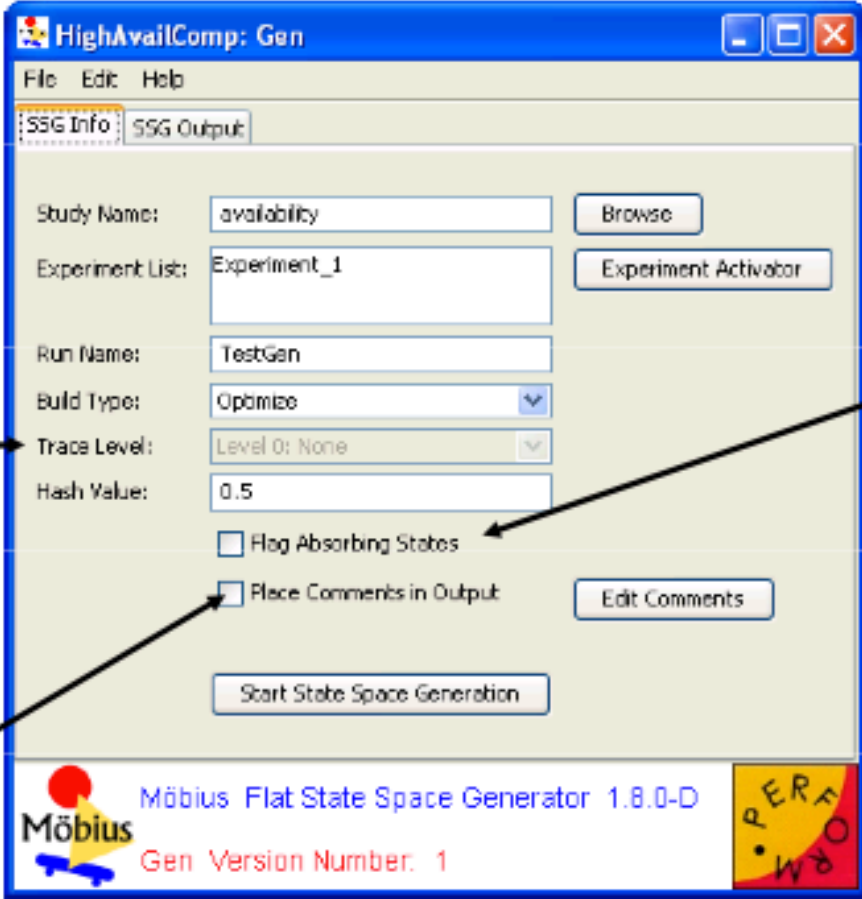
activity: CPUFail1, value = 1

activity: CPUFail2, value = 1

Rate Distributions

- Can specify activity time distributions
 - Distribution parameters \rightarrow marking dependent
- Exponential and Deterministic solutions
 - Analytical solution possible
- Other distributions: Normal, Binomial etc.
 - No analytical solution –discrete event simulation

Möbius: State Space Generation



The screenshot shows the HighAvailComp: Gen software interface. The window title is "HighAvailComp: Gen". The menu bar includes "File", "Edit", and "Help". There are two tabs: "SSG Info" and "SSG Output". The main area contains several input fields and buttons:

- Study Name: availability (with a Browse button)
- Experiment List: Experiment_1 (with an Experiment Activator button)
- Run Name: TestGen
- Build Type: Optimize (dropdown menu)
- Trace Level: Level 0: None (dropdown menu)
- Hash Value: 0.5
- Flag Absorbing States
- Place Comments in Output (with an Edit Comments button)
- Start State Space Generation button

Annotations with arrows point to specific features:

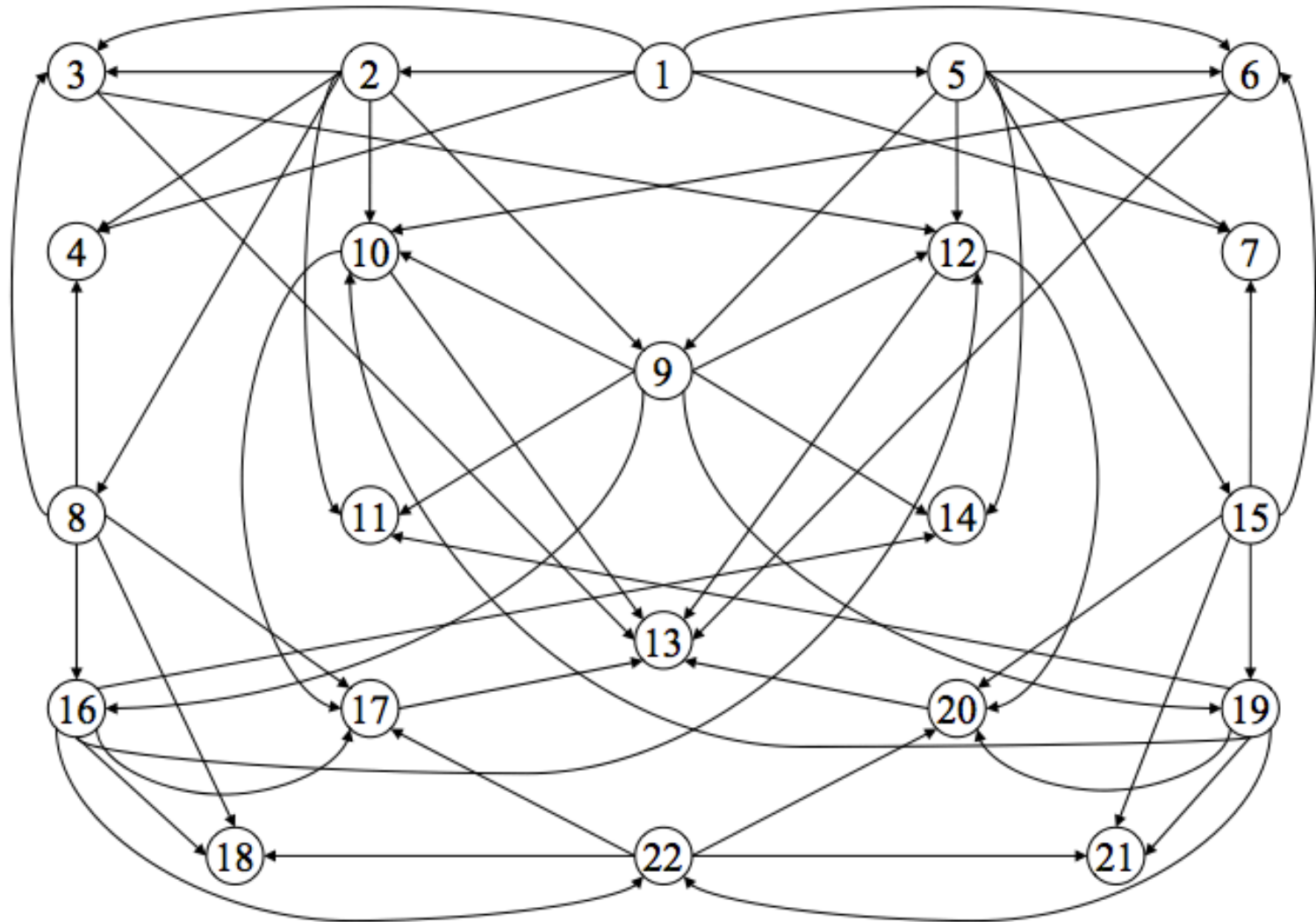
- "Print out states and reward variables" points to the Trace Level dropdown.
- "Print out absorbing states. Useful to detect problems when attempting a steady-state solution." points to the "Flag Absorbing States" checkbox.
- "Place comments, as specified by edit comments, in file." points to the "Place Comments in Output" checkbox.

The bottom of the window features a logo for "Möbius" and text: "Möbius Flat State Space Generator 1.8.0-D" and "Gen Version Number: 1". There is also a small graphic on the right side of the bottom bar.

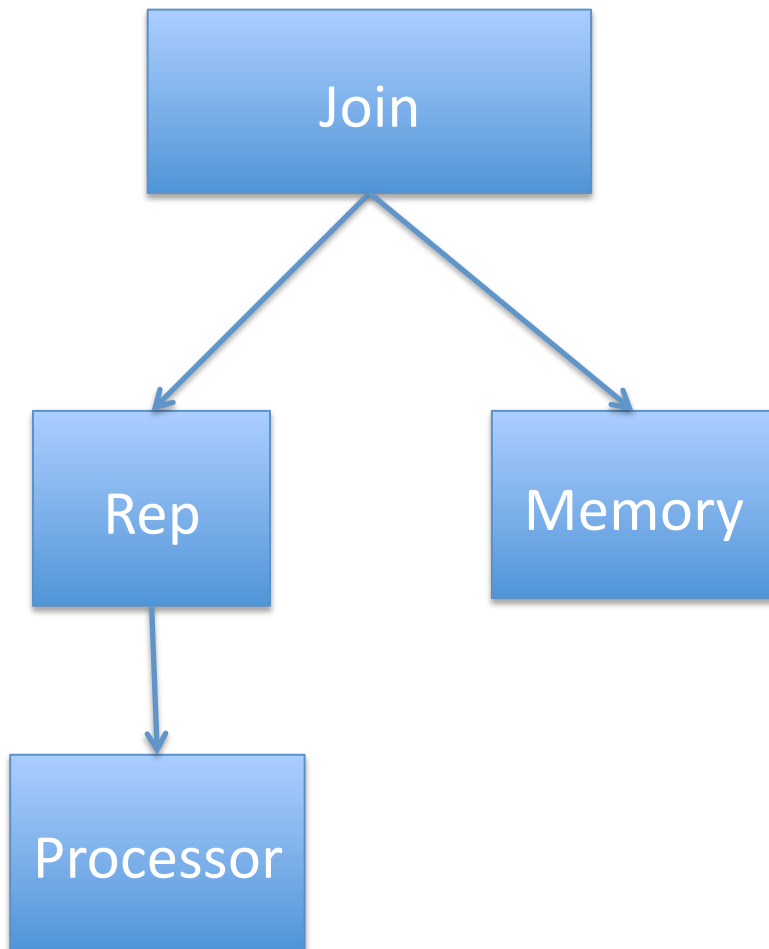
State space generated by Mobius

State No.	CPUboards1	CPUboards2	NumComp	(Next State, Rate)
1	3	3	2	(2, $p1\lambda$), (3, $p2\lambda$), (4, $p3\lambda$), (5, $p1\lambda$), (6, $p2\lambda$), (7, $p3\lambda$)
2	2	3	2	(8, $p1\lambda$), (3, $p2\lambda$), (4, $p3\lambda$), (9, $p1\lambda$), (10, $p2\lambda$), (11, $p3\lambda$)
3	0	3	1	(12, $p1\lambda$), (13, $(p2+p3)\lambda$)
4	0	3	0	
5	3	2	2	(9, $p1\lambda$), (12, $p2\lambda$), (14, $p3\lambda$), (15, $p1\lambda$), (6, $p2\lambda$), (7, $p3\lambda$)
6	3	0	1	(10, $p1\lambda$), (13, $(p2+p3)\lambda$)
7	3	0	0	
8	1	3	2	(3, $(p1+p2)\lambda$), (4, $p3\lambda$), (16, $p1\lambda$), (17, $p2\lambda$), (18, $p3\lambda$)
9	2	2	2	(16, $p1\lambda$), (12, $p2\lambda$), (14, $p3\lambda$), (19, $p1\lambda$), (10, $p2\lambda$), (11, $p3\lambda$)
10	2	0	1	(17, $p1\lambda$), (13, $(p2+p3)\lambda$)
11	2	0	0	
12	0	2	1	(20, $p1\lambda$), (13, $(p2+p3)\lambda$)
13	0	0	0	
14	0	2	0	
15	3	1	2	(19, $p1\lambda$), (20, $p2\lambda$), (21, $p3\lambda$), (6, $(p1+p2)\lambda$), (7, $p3\lambda$)
16	1	2	2	(12, $(p1+p2)\lambda$), (14, $p3\lambda$), (22, $p1\lambda$), (17, $p2\lambda$), (18, $p3\lambda$)
17	1	0	1	(13, λ)
18	1	0	0	
19	2	1	2	(22, $p1\lambda$), (20, $p2\lambda$), (21, $p3\lambda$), (10, $(p1+p2)\lambda$), (11, $p3\lambda$)
20	0	1	1	(13, λ)
21	0	1	0	
22	1	1	2	(20, $(p1+p2)\lambda$), (21, $p3\lambda$), (17, $(p1+p2)\lambda$), (18, $p3\lambda$)

Example: Generated Markov Model



Model Combination

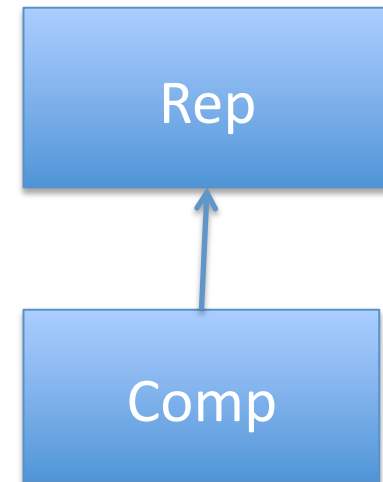
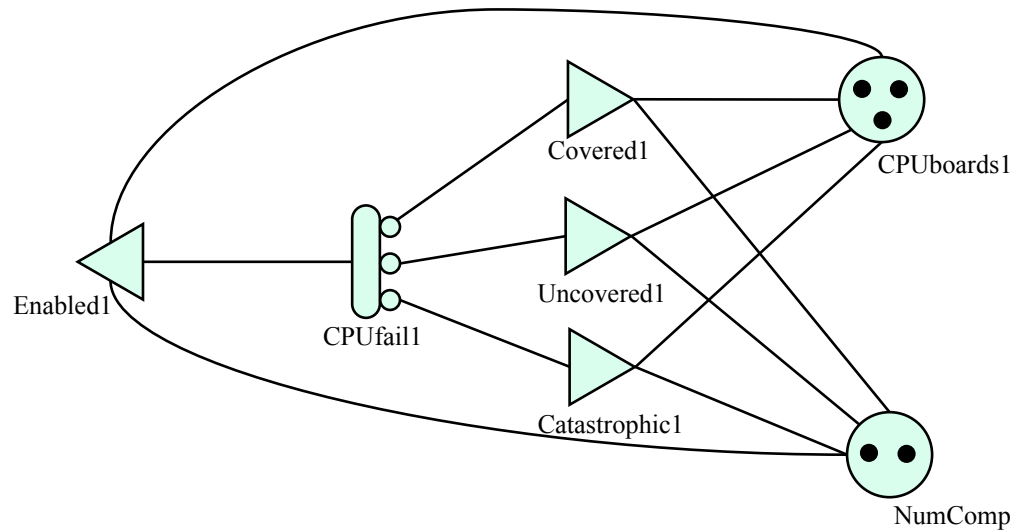


- A composed model is a way to combine two or more SANs together so that they can share places and rewards
- Two kinds of Joins
 - Join
 - Replicate

Why combine models ?

- Fault-tolerant systems have redundancy. Rep is a natural way of adding redundancy to system
- Most systems are too complex to have 1 single model. So we need a hierarchical structure
- State-space generation for composed models more efficient (Strong Lumping Theorem)

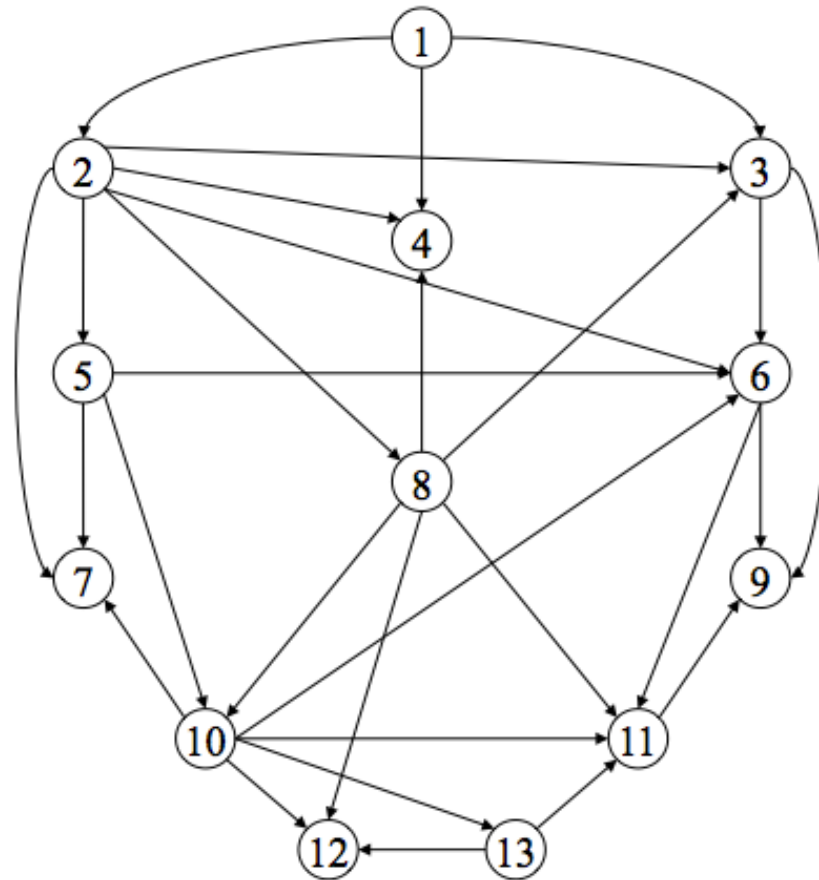
Model Combination: Example



Note: Initial marking of NumComp is still two for two computers

NumComp is a shared place

Reduced Markov Model due to Model Combination



Tricks to reduce state space

- Lump all failures into the same state if you don't care which component failed
- Use replication if you don't care which component or which module is in which state
- Use marking dependent rates to model failures leveraging the fact that minimum of exponential distributions is exponential

Mobius: Analytical Solvers

- Can be done **only after** state space generation

<i>Analytic Solvers (for reward variables only)</i>				
<i>Model Class</i>	<i>Steady-state or Transient</i>	<i>Instant-of-time or Interval-of-time</i>	<i>Mean, Variance, or Distribution</i>	<i>Applicable Analytic Solver</i>
All activities exponential	Steady-state	Instant-of-time ^a	Mean, Variance, and Distribution	<i>dss</i> and <i>iss</i>
	Transient	Instant-of-time	Mean, Variance, and Distribution	<i>trs</i> and <i>atrs</i>
		Interval-of-time	Mean	<i>ars</i>
Exponential and Deterministic activities	Steady-state	Instant-of-time ^b	Mean, Variance, and Distribution	<i>diss</i> and <i>adiss</i>

Summary

- State-based models for reliability evaluation
 - Markov Chains
 - Stochastic Petri Nets
 - Stochastic Activity Networks
- Stochastic Activity Networks are a powerful mechanism to specify states and transitions
 - can be automatically converted to Markov chains
 - High-level formalism avoids gratuitous states