Reliability and Availability Modeling in Practice

IFIP 7.3 Performance Conference -- Tutorial

Nov. 2, 2020

Kishor Trivedi
Department of Electrical and Computer Engineering
Duke University, Durham, NC

Andrea Bobbio
DiSit - Computer Science Section
Università del Piemonte Orientale
Alessandria - Italy

IFIP 7.3 Performance Conference 2020

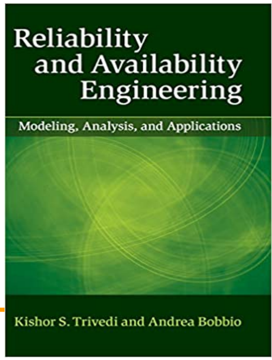
- Students may get a certificate for participation in this Tutorial

- You need to fill in a google form:
- https://docs.google.com/forms/d/e/1FAIpQLSe9Fgh5rdxcP9RLPMktHUHrGXZlMGKuxKC_7JICedzQWjrVyw/viewform

Tutorial Objective

- To provide an overview and state of the art of analytic methods for reliability and availability assessment
- To provide real-life examples that show the use of analytic methods in practice
- To provide current challenges faced in such modeling projects
- Ref: Trivedi & Bobbio, Reliability and Availability:

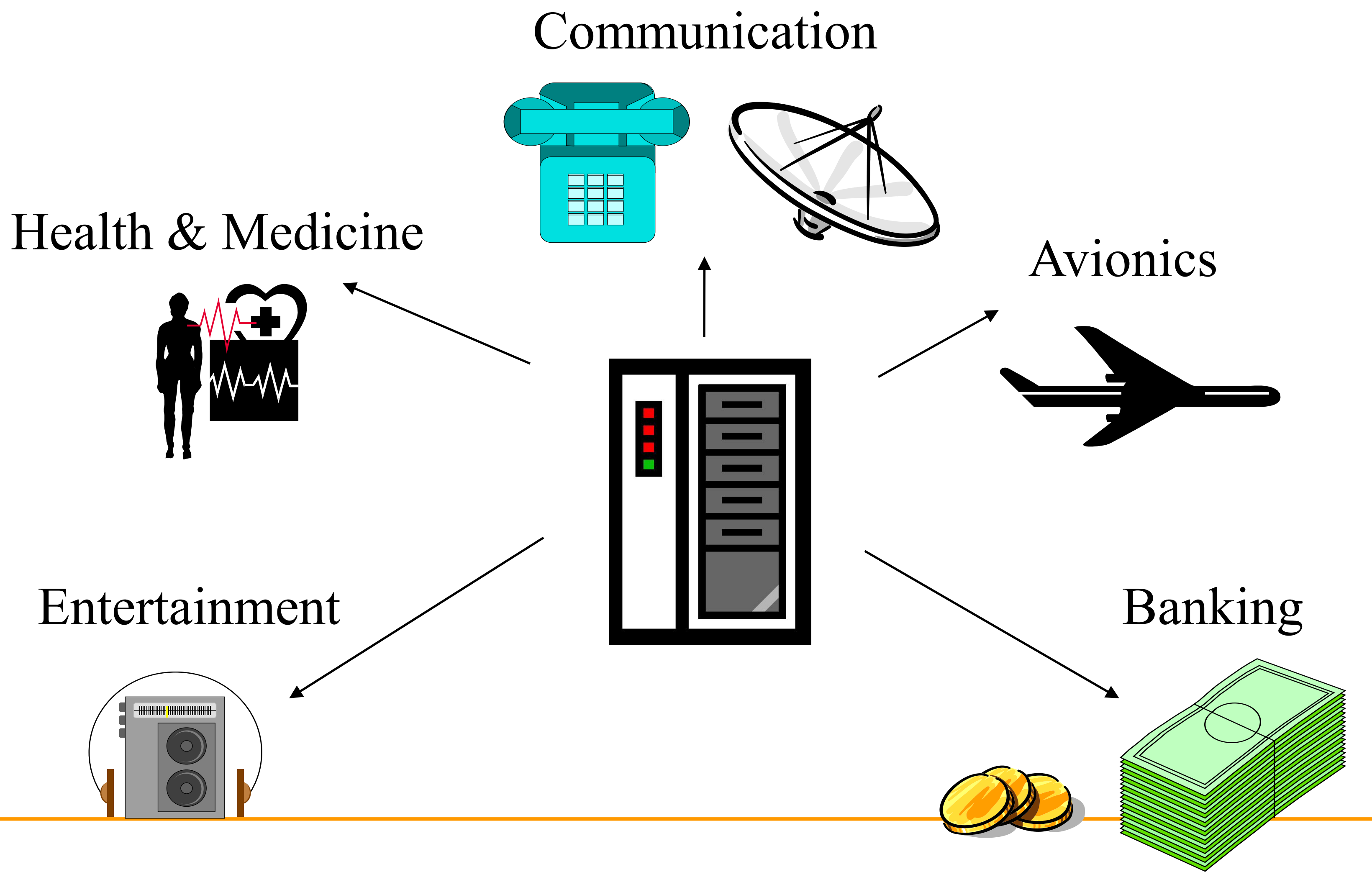
  Modeling, Analysis, Applications, Cambridge University Press, 2017



Tutorial Outline

- Introduction
- Reliability and Availability Models
  - Model Types in Use
  - Illustrated through several real examples
- Conclusions
- References

Motivation:
Dependence on Technical Systems



Example Failures from High Tech companies

Mar. 2015 , Gmail was down for 4 hours and 40 min.

Mar. 2015, Down for 3 hours affecting Europe and US

Dec. 2015, Microsoft Office 365 and Azure down for 2 hours

Sept. 2015, AWS DynamoDB down for 4 hours impacting among others Netflix, AirBnB, Tinder

Mar. 2015, Apple ITunes, App Stores long outage: 12 hours

More examples of real failures

amazon.com Feb. 2017 Amazon S3 service outage (almost 6 hours)

Google Jul. 2017 - Google Cloud Storage service outage (3 hours and 14 min.) - API low-level software defect

Microsoft Jul. 2017 - Microsoft Azure service outage (4 hours) – Load Balancer Software bug

Very Recent Examples

- In Commercial aircrafts (Boeing 737 Max software problem)
  - Ethiopian Airlines Flight, March 2019,

    149 people died
  - Lion Air Flight crash, Oct. 2018,

    189 people died

Need Methods

- That reduce the occurrence of system failures and reduce downtime due to these failures (contributed by hardware, software and humans)

- For System Reliability/Availability assessment and bottleneck detection during:
  - Design phase
  - Certification phase
  - Operational phase

Introduction

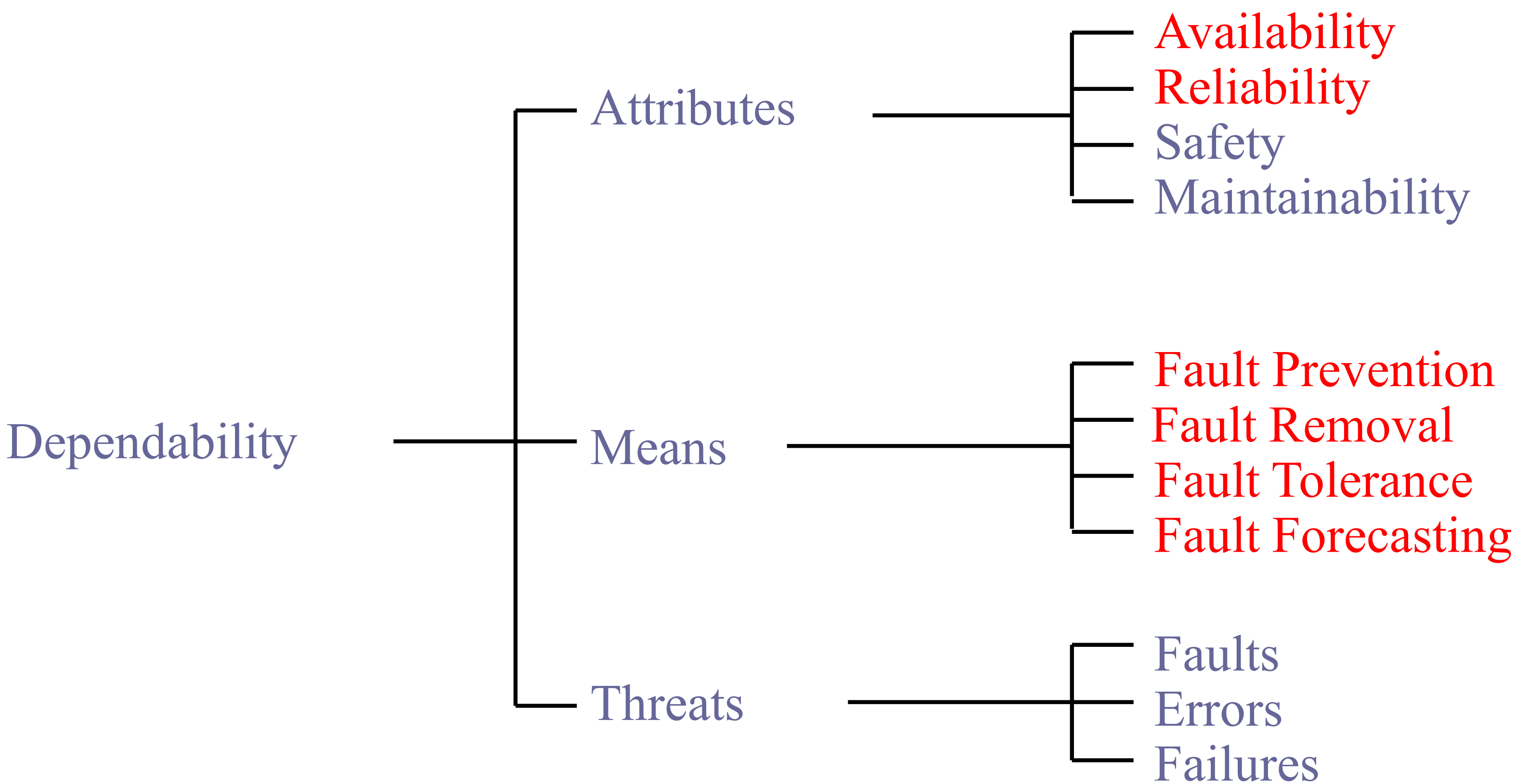
Basic Definitions

Need for a new term

- *Reliability* is often used in a generic sense as an umbrella term.
- *Reliability* is also used as a precisely defined mathematical function.
- To remove the confusion, IFIP WG 10.4 proposed *Dependability* as an umbrella term and *Reliability* is to be used as a well-defined mathematical function.

Dependability– An umbrella term

- Trustworthiness of a system such that reliance can justifiably be placed on the service it delivers



Difference between reliability and availability

- reliability refers to failure-free operation during an entire interval,
- availability refers to failure-free operation at a given instant of time.

Definitions from IFIP WG10.4

- **Failure** occurs when the delivered service no longer complies with the desired service
- **Error** is that part of the system state which is liable to lead to subsequent failure
- **Fault** is adjudged or hypothesized cause of an error

Faults are the cause of errors that may lead to failures

Fault → Error → Failure

A Classification of Faults

- Hardware vs. Software vs. Human
- Hardware: Permanent, Intermittent, Transient
- Network: Node vs. Link
- Software: Bohrbugs, Mandelbugs, Heisenbugs, Aging-related bugs

Failure Classification

- Omission failures (Send/receive failures)
  - Crash failures
  - Infinite loop
- Response or Value failures
- Timing failures
  - Early
  - Late (aka performance failure or dynamic failures)
- Safe vs. Unsafe failure
- Breach of confidentiality or breach of integrity or loss of use

Basic Definitions

- One shot Reliability *R:*

  When is this applicable?

- Reliability $R(t)$:

  X: Time to failure of a system (TTF), or lifetime random variable

  $F(t)$: distribution function of system lifetime

$$R(t) = P(X > t) = 1 - F(t)$$

Reliability is the complementary distribution function of TTF

Basic Definitions

- Mean Time To system Failure:

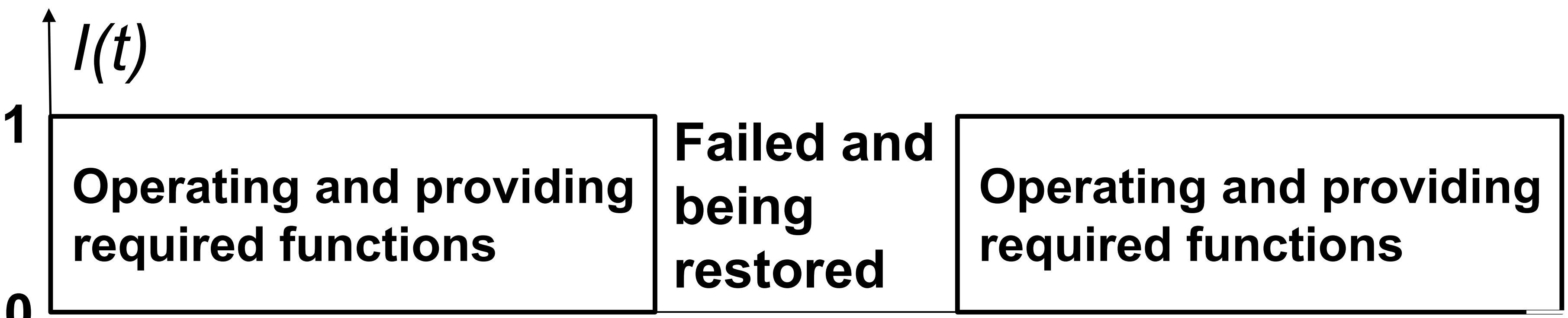
$$MTTF = E[X] = \int_0^{\infty} t f(t) dt = \int_0^{\infty} R(t) dt$$

where $f(t)$: density function of system lifetime

Make a clear distinction between *TTF*, *R(t)* and *MTTF*

Basic Definitions

- Availability



System Failure and Restoration Process

I(t) is the indicator function

Basic Definitions

- Instantaneous Availability $A(t)$:

$$A(t) = P\,(\text{system working at } t)$$

  - From the figure in the last slide, the availability at time t becomes:

$$A(t)=P(I(t)=1)$$

  - This is sometimes called point-wise availability, instantaneous availability, or transient availability. $A(t)$ can be asked for at any point t in time

Basic Definitions

- **Interval reliability** measure introduced by Barlow and Hunter in 1961, combines availability $A(t)$ and reliability $R(\tau)$:
  - Available when needed (at time t) & as long as needed (for τ time units)
- Interval reliability further developed in:
  - Trivedi & Bobbio, **Reliability and Availability: Modeling, Analysis, Applications,** Cambridge University Press, 2017
  - Wang & Trivedi, **Modeling User-Perceived Service Reliability based User-Behavior Graphs**, *IJRQS, 2011*
  - Trivedi, Wang & Hunt, **Computing the number of calls dropped due to failures**, *ISSRE2010*
  - Mondal, Yin, Muppala, Alonso, Trivedi, **Defects per Million Computation in Service-Oriented Environments**, *IEEE Trans. on Services Comp.,* 2015

Basic Definitions

Steady-state availability (A_{ss}) or just availability

Long-term probability that the system is available when requested:

- $$A_{ss} = \frac{MTTF}{MTTF + MTTR}$$
- MTTF is the system mean time to failure
- MTTR is the system mean time to recovery
  - may consist of many phases

For a non-fault-tolerant system no distributional assumptions needed

Basic Definitions

Steady-state availability (A_{ss}) or just availability

Long-term probability that the system is available when requested (also applies to a fault-tolerant system):

$$A_{ss} = \frac{MTTF}{MTTF + MTTR}$$

- MTTF is the "equivalent" system mean time to failure, a complex combination of component MTTFs
- MTTR is the "equivalent" system mean time to recovery

Basic Definitions

- Downtime in minutes per year
  - In industry, steady-state (un)availability is usually presented in terms of annual (steady-state) downtime.
  - Downtime = $8760 \times 60 \times (1 - A_{ss})$ minutes.
  - It is also common to define the availability in terms of number of nines

5 NINES ($A_{ss} = 0.99999$) → 5.26 minutes annual downtime

4 NINES ($A_{ss} = 0.9999$) → 52.56 minutes annual downtime

Number of Nines– Reality Check

- 49% of Fortune 500 companies experience at least 1.6 hours of downtime per week
  - Approx. 80 hours/year=4800 minutes/year
  - $A_{ss}=(8760-80)/8760=0.9908$
  - **That is, between 2 NINES and 3 NINES!**
- This study combines planned and unplanned downtime

Failures & Downtime Lead to

- A Loss of Reputation
- A Loss of Revenue
- Possible Loss of Life

Need Methods

- That reduce system failures and reduce downtime due to these failures (contributed by hardware, software and humans)

- System Reliability/Availability assessment and bottleneck detection methods can be used:
  - To compare alternative designs/architectures
  - Find bottlenecks, answer what if questions, design optimization and conduct trade-off studies
  - At certification time
  - At design verification/testing time
  - Configuration selection phase
  - Operational phase for system tuning/on-line control

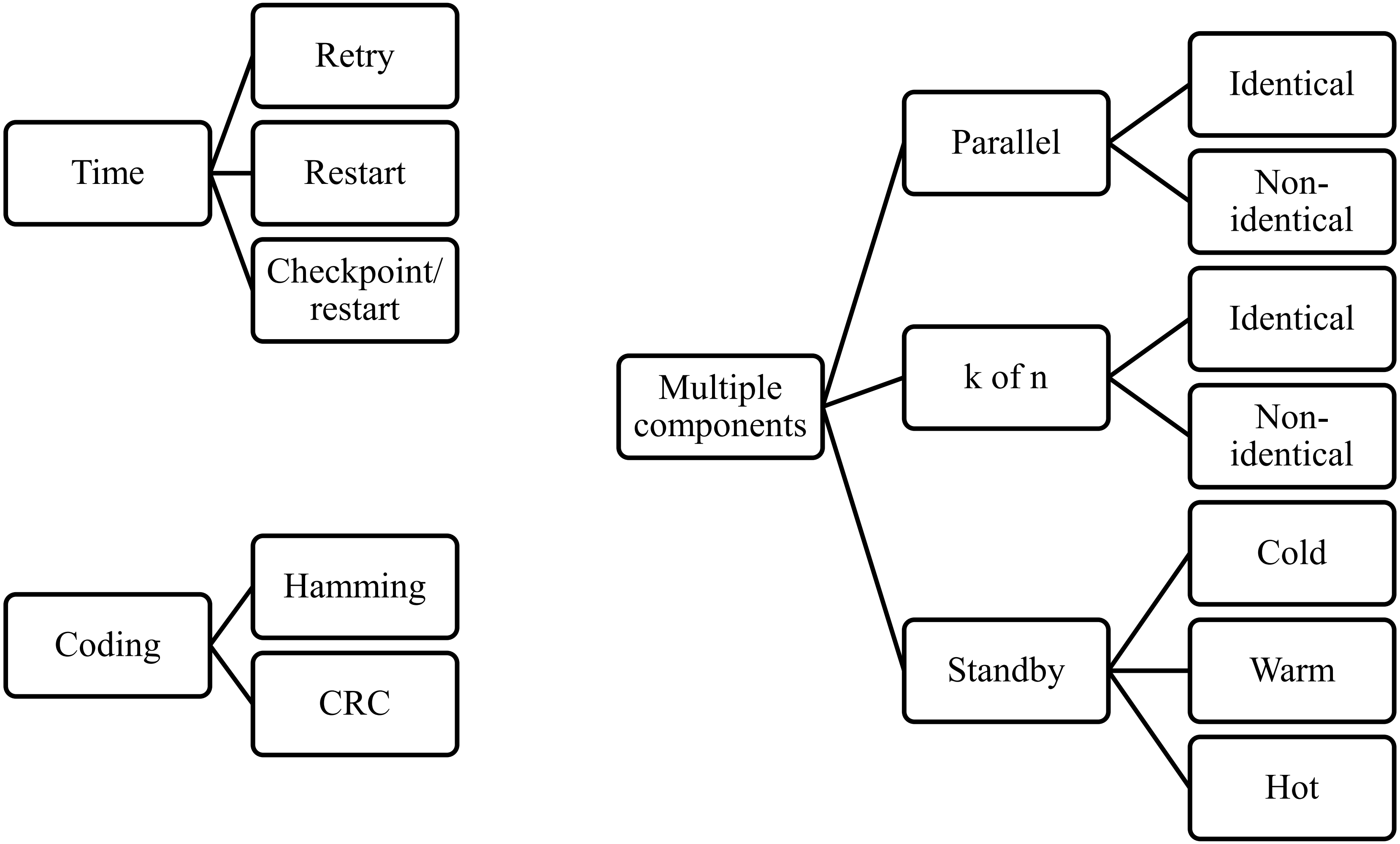
Methods to Improve Dependability

- Fault Avoidance
  - Employ high reliability components
- Fault Removal
  - Careful Testing to remove faults
- Fault Tolerance
  - Utilize Redundancy
- Fault Forecasting
  - Predict failures and use for preventive maintenance

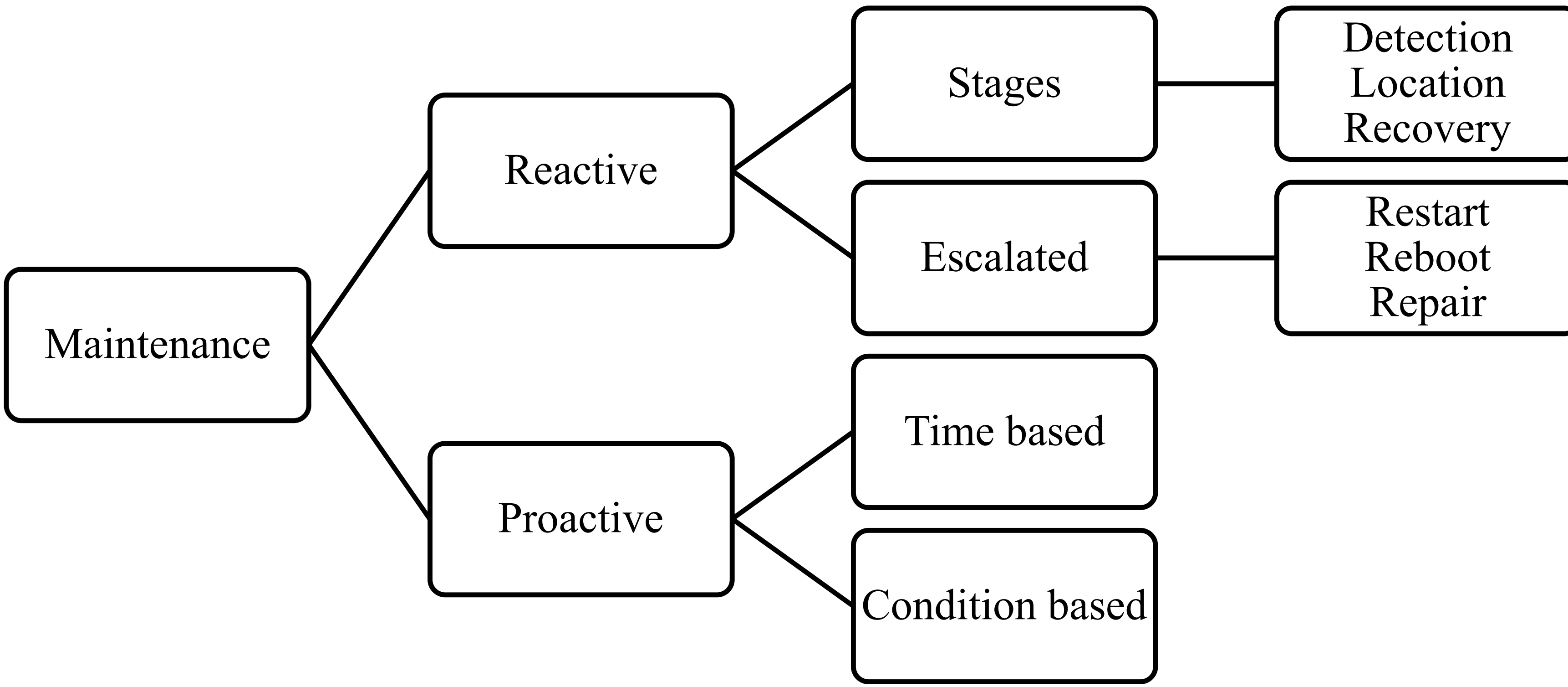
Methods Overview (Redundancy)

- Redundancy
  - Coding
  - Time
  - Use of Multiple Redundant Components, that is, more than required for the performance needs

Methods Overview (Redundancy)



Methods Overview (Maintenance)



Need Methods

- That reduce system failures and reduce downtime due to these failures (contributed by hardware, software and humans)

- System Reliability/Availability assessment and bottleneck detection methods can be used:
  - To compare alternative designs/architectures
  - Find bottlenecks, answer what if questions, design optimization and conduct trade-off studies
  - At certification time
  - At design verification/testing time
  - Configuration selection phase
  - Operational phase for system tuning/on-line control

Quantitative Assessment approaches

- Black-box or Data-driven

  (measurement data + statistical inference):
  - The system is treated as a monolithic whole, without explicitly taking its internal structure into account
  - Very expensive especially for ultra-reliable systems
    - ALT can help reduce the cost
  - Generally applicable to small systems that are not very highly reliable
  - Not feasible for system under design/development

Quantitative Assessment approaches

- White-box (or Model-driven):
  - When no data is available for the system as a whole
  - Probability Model (e.g., RBD, Ftree, Markov chain) constructed based on the known internal structure of system – its components, their characteristics and interactions between components
  - Derive the behavior of ensembles (combinations of components to form a system or combinations of multiple systems to form a system of systems) from first principles
  - Used to analyze a system with many interacting and interdependent components
  - Need input parameters for components and subsystems

Quantitative Assessment approaches

- Combined approach
  - Use black-box approach at subsystem/component level
  - Use white-box approach at the system level
  - Thus a combined Data + Model driven approach

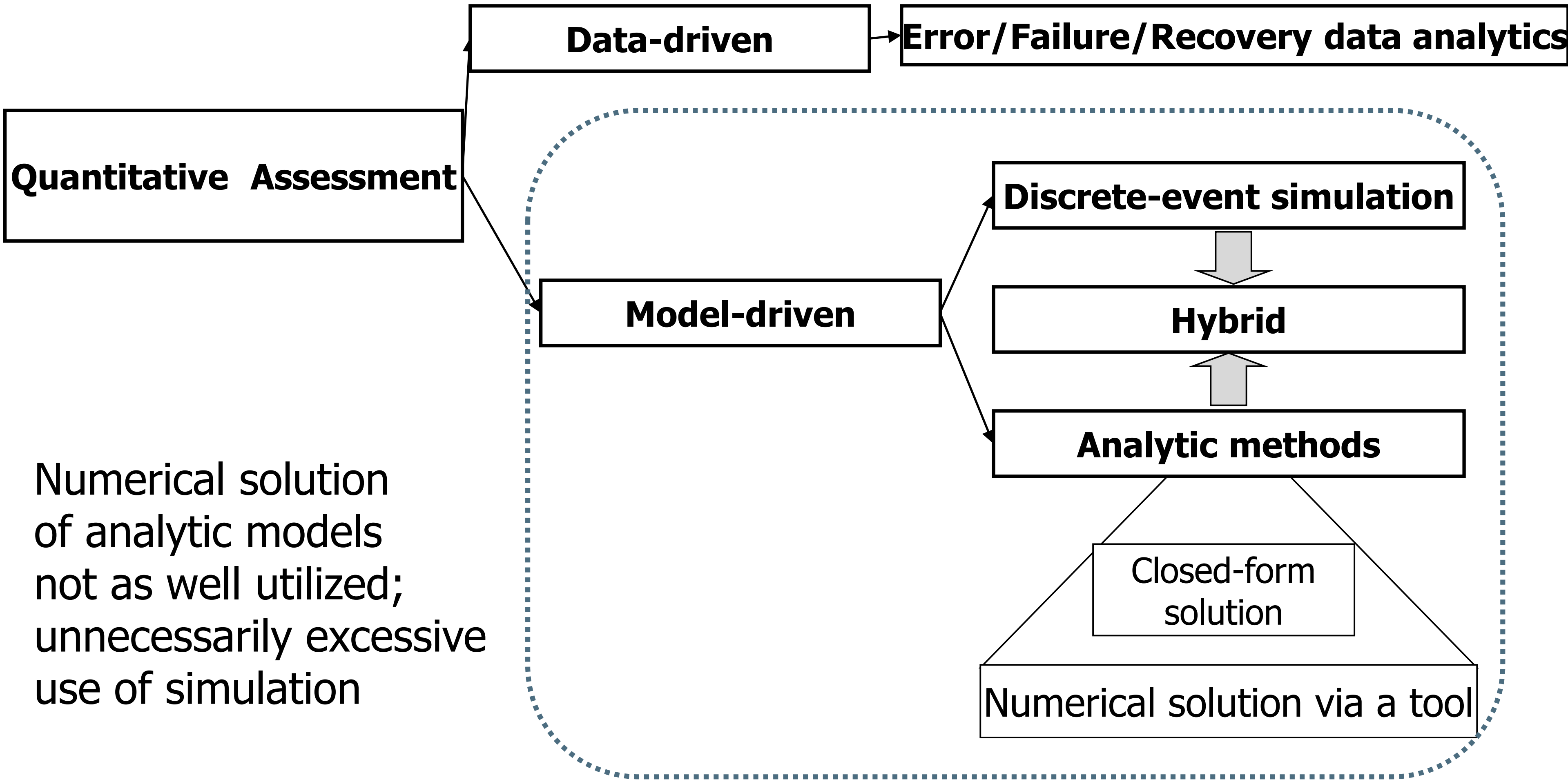
Two Types of Uncertainty

- Aleatory (irreducible)
  - Randomness of event occurrences in the real system captured by various distributions in the Probability Model (e.g., RBD, Fault tree, Markov chain)
- Epistemic (reducible)
  - Introduced due to finite sample size in estimating parameters to be input to the Probability Model
- Propagating epistemic uncertainty through a Probability Model is a topic that will not be covered in this tutorial – can be a subject of another tutorial!

Outline

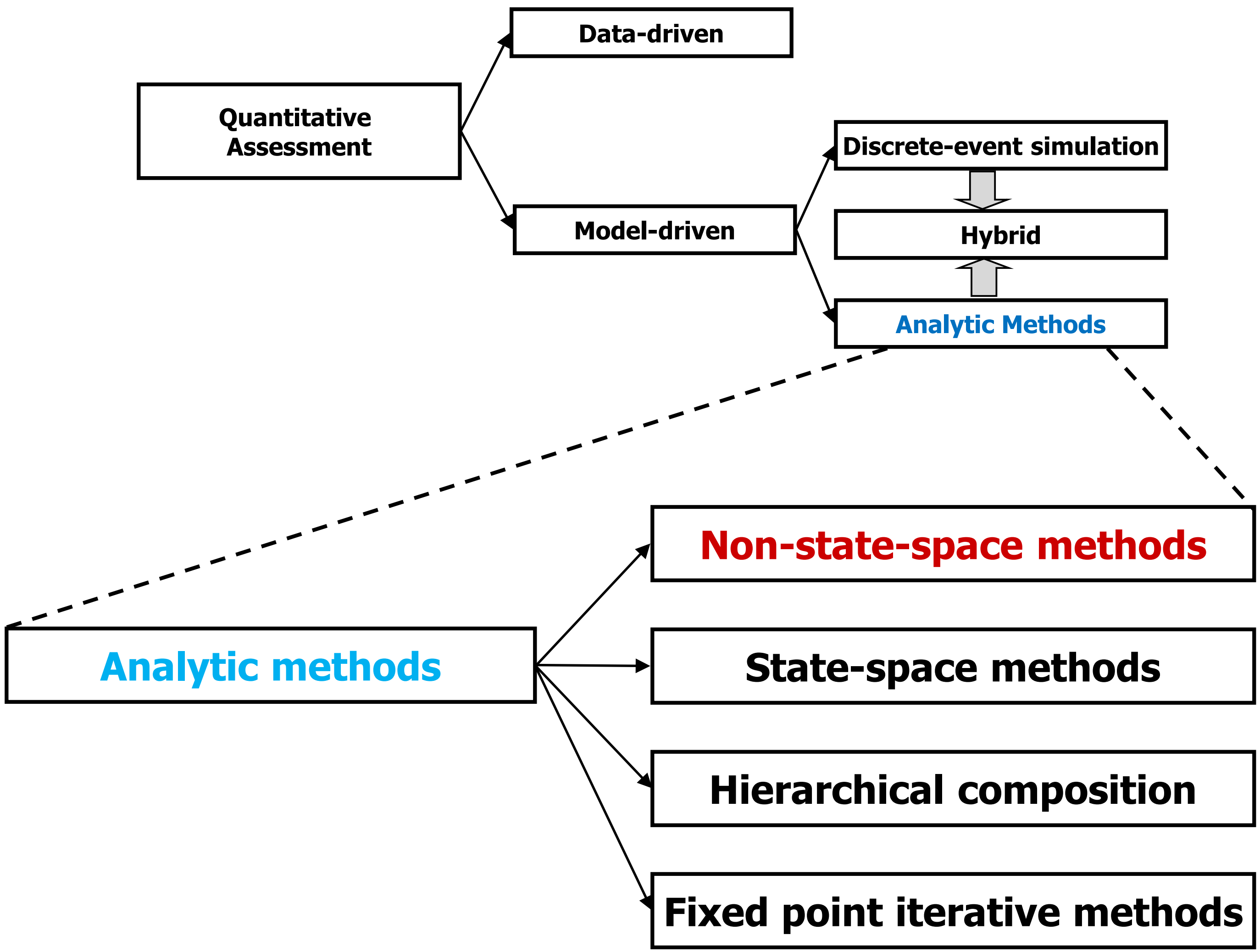
- Introduction and Motivation
- Reliability and Availability Models
- Conclusions
- References

Overview of Assessment Methods

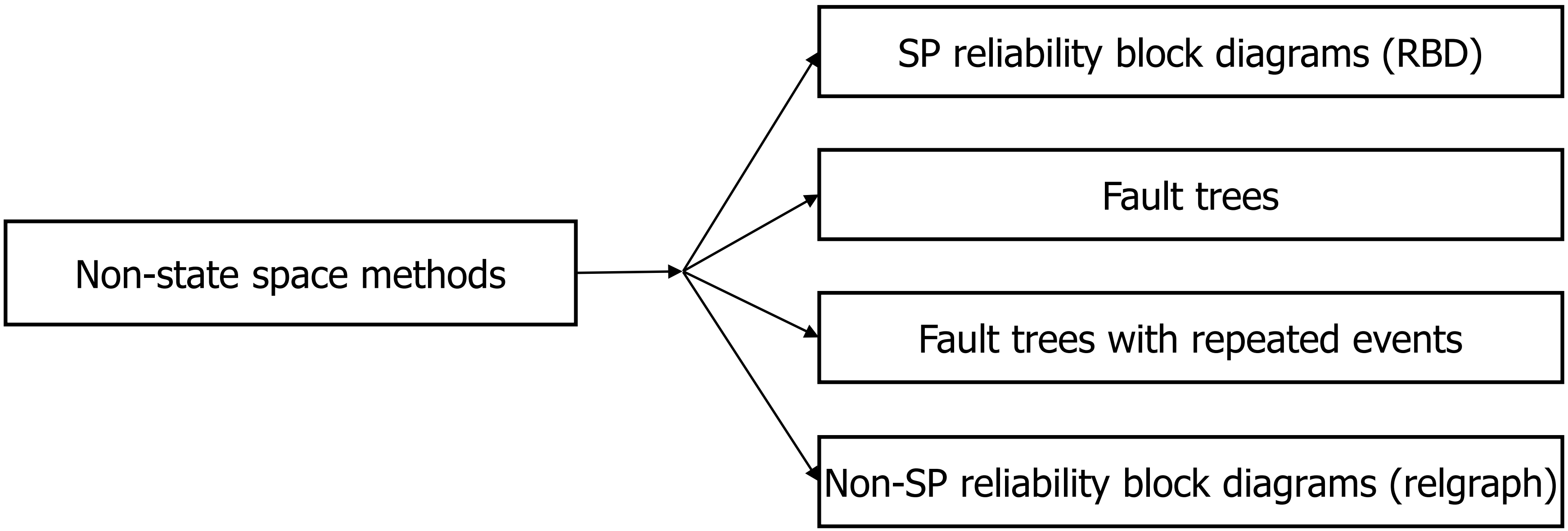


Numerical solution of analytic models not as well utilized; unnecessarily excessive use of simulation

Analytic Methods Taxonomy

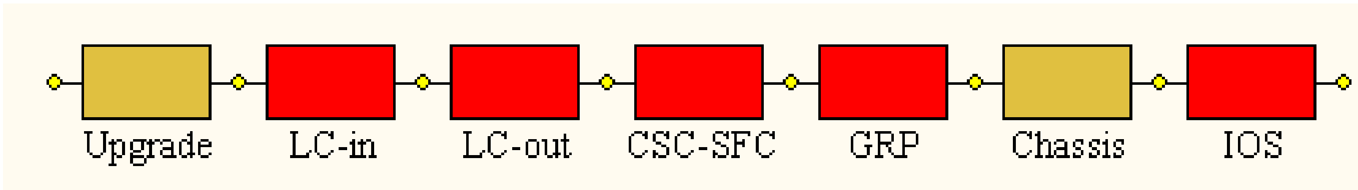


Non-State-Space Methods : taxonomy

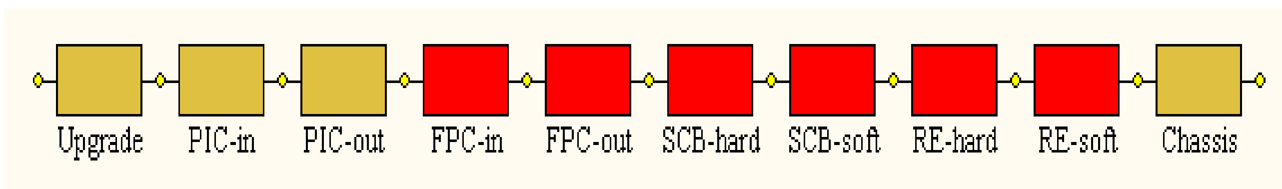


Extensions such as multi-state components/systems, phased-mission systems etc.

Cisco & Juniper Routers



RBD of Cisco 12000 GSR



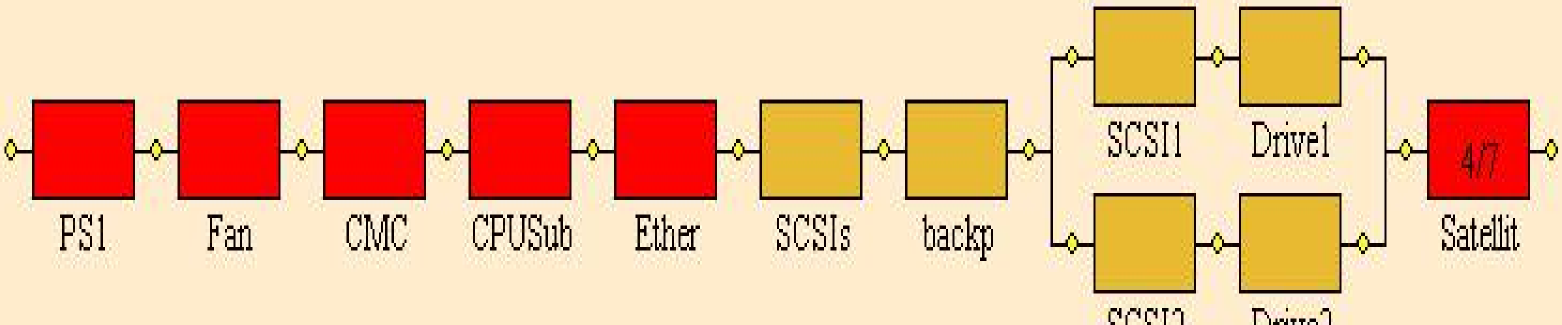
RBD of Juniper M20

K. Trivedi, "Availability Analysis of Cisco GSR 12000 and Juniper M20/M40" Cisco Internal report, 2000.

Red colored block means a sub-model.

Modeling High Availability Systems: Sun Microsystems

Trivedi et al., Modeling High Availability Systems, PRDC'06 Conference, Dec. 2006, Riverside, CA



Sun Microsystems

Top level RBD consists of all the subsystems joined by series, parallel and *k*/*n* blocks.
Red color means a sub-model.

Series-Parallel RBDs

- System reliability (availability) formulas :
  - Assuming statistical Independence of failures (and repairs)
  - Reliabilities (availabilities) multiply for blocks in series
    $$R_s = \prod_{i=1}^{n} R_i$$
  - Un-reliabilities (un-availabilities) multiply for blocks in parallel
    $$R_p = 1 - \prod_{i=1}^{n} (1 - R_i)$$
  - Blocks in k-out-of-n have a simple formula
    - Identical case $R_{k/n} = \sum_{j=k}^{n} \binom{n}{j} R^j (1 - R^{n-j}$
    - Non-identical case

$$\begin{cases} R_{k|n} = (1 - R_n) \cdot R_{k|n-1} + R_n \cdot R_{k-1|n-1} \\ R_{0|n} = 1 \\ R_{j|i} = 0, \text{ when } j > i \end{cases}$$

Fault Trees

- Fault Tree is a pessimist's model as opposed to RBD that can be considered optimists' models
- Components are represented as leaves or terminal nodes
- Internal nodes are logic gates and Root node indicates system failure
- Components or subsystems in series are connected with OR gates
- Components or subsystems in parallel are connected with AND gates
- Failure of a component or subsystem causes the corresponding input to the gate to become TRUE
- Whenever the output of the topmost gate (root node) is TRUE, the system is considered failed

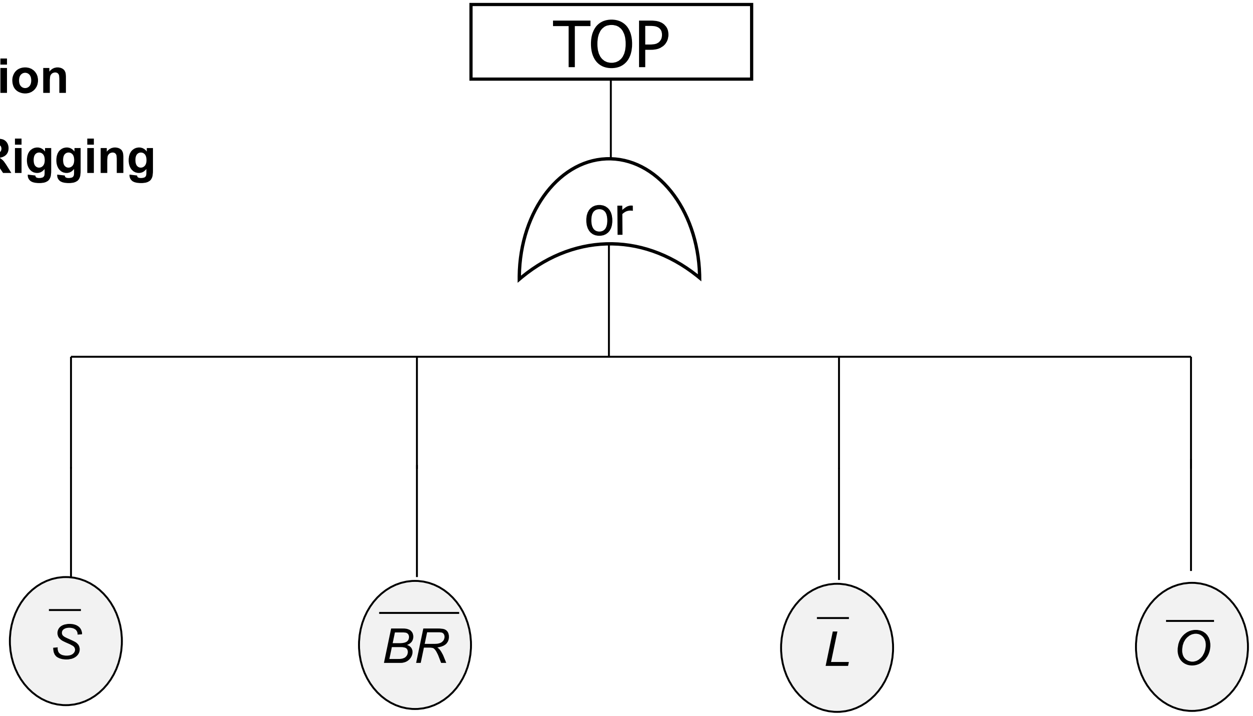
Fault Tree Model of GE Truck- AC6000

S **= Suspension**

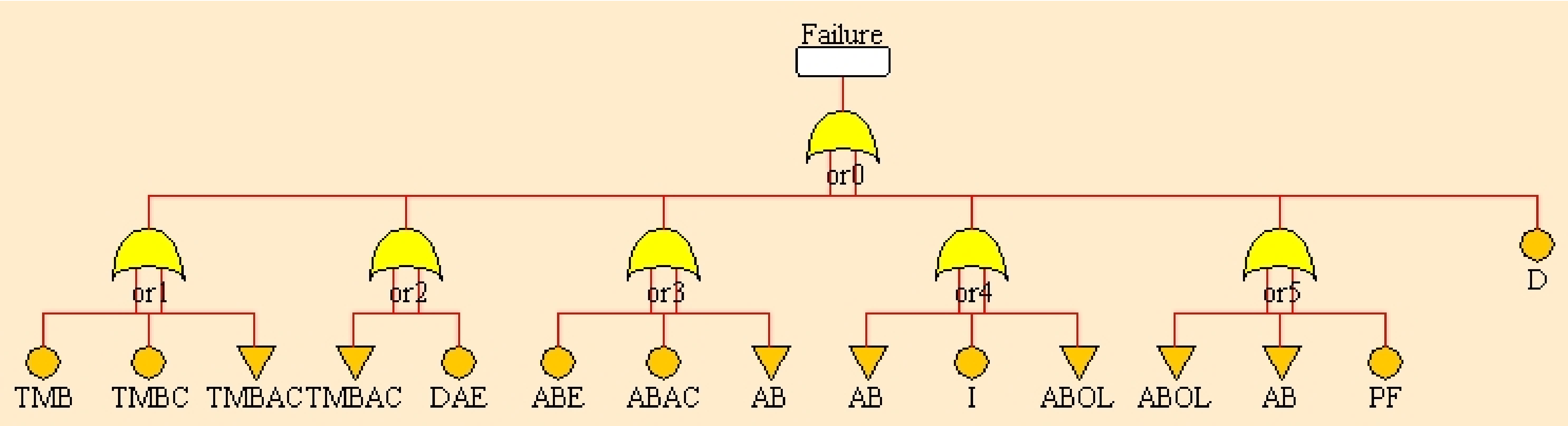
BR **= Brake Rigging**

L **= Liner**

O **= Others**

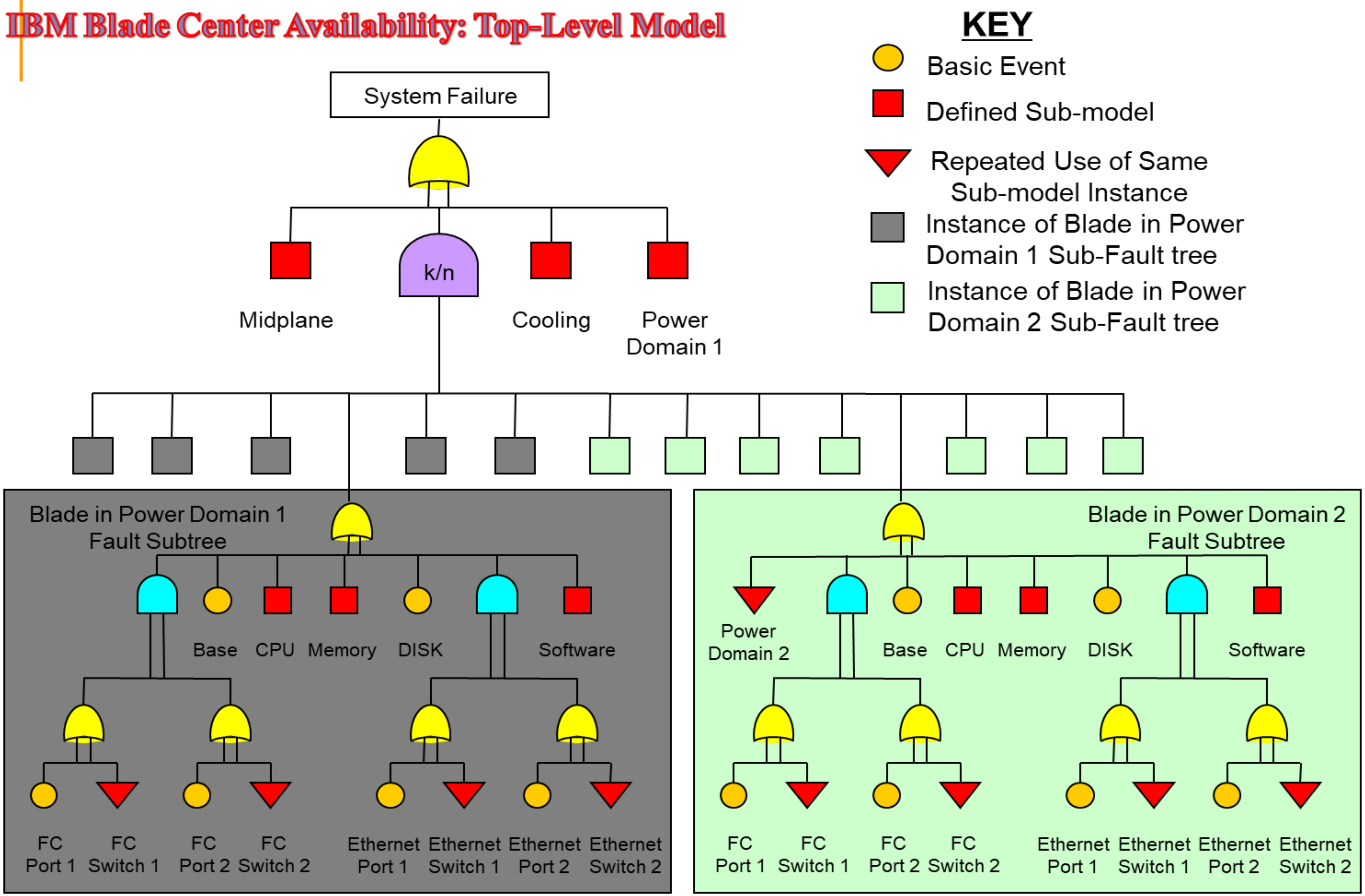


Fault Tree Model of GE Equipment Ventilation System



Fault Tree with Repeated events; inverted triangle indicates such events

IBM Blade Center Availability: Top-Level Model



Smith, Trivedi et al., IBM Systems J., 2008

Software Package SHARPE

- SHARPE: Symbolic-Hierarchical Automated Reliability and Performance Evaluator
- Stochastic Modeling tool installed at over 1000 Sites; companies and universities
- Ported to most architectures and operating systems
- Used for Education, Research, Engineering Practice
- Users: Boeing, 3Com, EMC, AT & T, Alcatel-Lucent, IBM, NEC, Motorola, Siemens, GE, HP, Raytheon, Honda,…
- http://sharpe.pratt.duke.edu/
- It is the core of Boeing's internal tool called IRAP

A Fool with a Tool is still a fool

Fault trees

- Major characteristics:
  - Fault trees without repeated events can be solved in polynomial time
  - Fault trees with repeated events -Theoretical complexity: exponential in number of components

- Use **Factoring** (conditioning) [In SHARPE use **factor on** and **bdd off**]

- Find all minimal cut-sets & then use Sum of Disjoint products (**SDP**) to compute reliability [In SHARPE use **factor off** and **bdd off**]

- Use **BDD** (Binary Decision Diagram) approach [In SHARPE use **bdd on**]

- In practice, can solve fault trees with thousands of components

Solution time for Very Large Fault trees

Number of total leaves	Computation Time (second)		
	factoring	BDD	SDP
10000	0.98	2.06	11.12
20000	2.63	7.11	23.07
30000	5.58	14.89	37.97
40000	10.23	26.69	52.46
50000	13.94	42.04	69.37
60000	19.53	59.33	85.90
70000	26.48	—	102.09
80000	34.49	—	122.04
90000	41.25	—	141.35
100000	52.41	—	162.84

Such large models can be solved because of independence assumption – non-states-space models

Fault Trees (Continued)

- Extensions to Fault-trees include a variety of different gate types: NOT, EXOR, Priority AND, cold spare gate, functional dependency gate, sequence enforcing gate, etc. Some of these are “static” while others are “dynamic” gates

Reliability Graph (relgraph)

- Consists of a set of nodes and edges
- Edges represent components that can fail
- Source and target (sink) nodes
- System fails when no path from source to sink
- A non-series-parallel RBD
- S-t connectedness or network reliability problem

Relgraphs

- Solution methods for Relgraph
  - Find all minpaths followed by SDP (Sum of Disjoint Products)
  - BDD (Binary Decision Diagrams)-based method
  - Factoring or conditioning
  - Monte Carlo method
- The first two methods have been implemented in our SHARPE software package

Avionics

- Reliability analysis of each major subsystem of a commercial airplane needs to be carried out and presented to Federal Aviation Administration (FAA) for certification

Real world example from Boeing Commercial Airplane Company

Reliability Analysis of Boeing 787

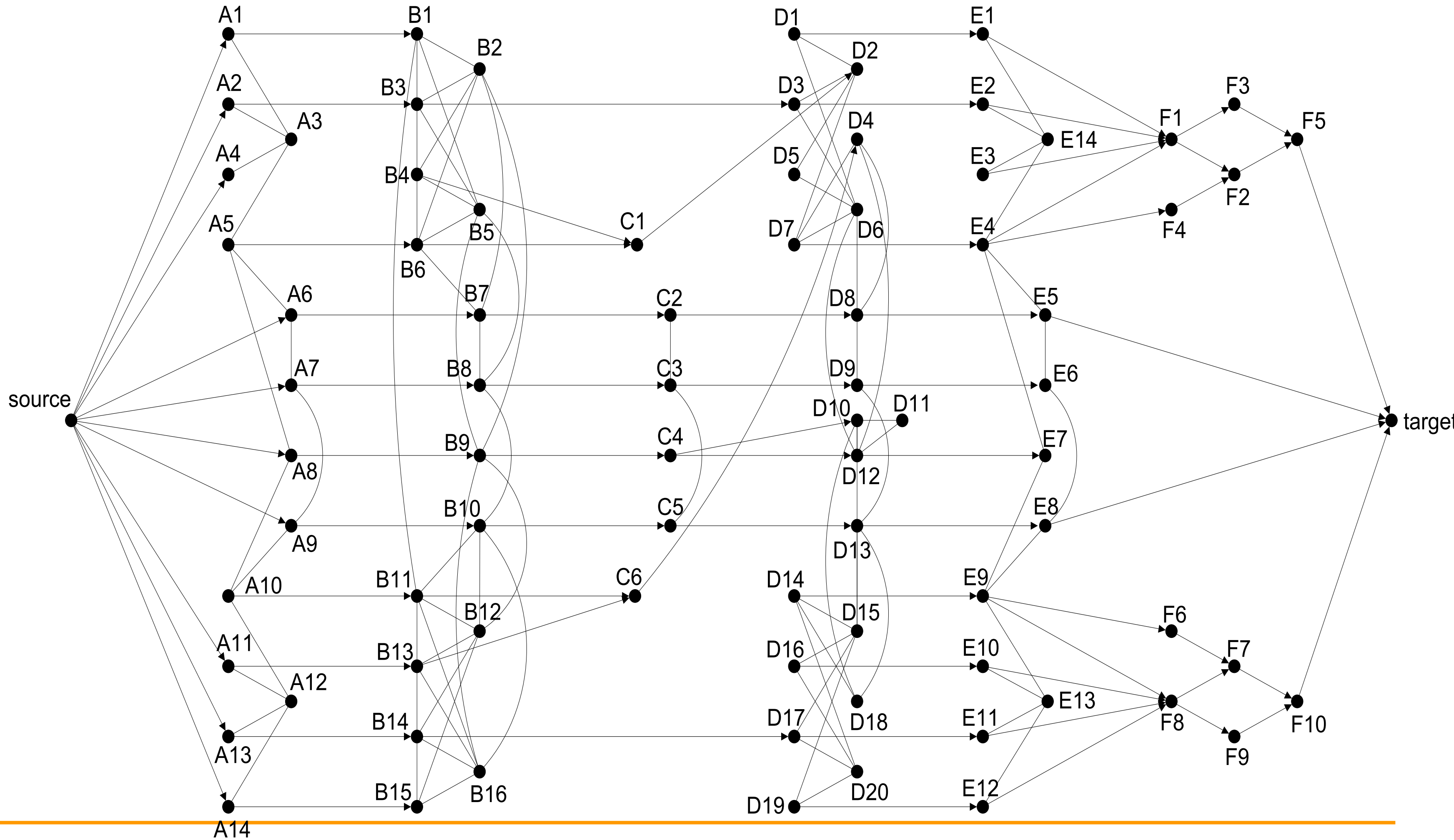
- Most of the subsystems are improved or modified versions of subsystems used in earlier planes
  - Models are also modified version of the earlier models
- Occasionally there is an entirely new subsystem
  - Model needs to be done from scratch
- Current Return Network in Boeing 787 is one such example
- Several of my former students are in the Boeing Reliability Engineering group

Reliability Analysis of Boeing 787

- Current Return Network Subsystem
- Modeled as a Reliability Graph
  - Consists of a set of nodes and edges
  - Edges represent components that can fail
  - Source and target nodes
  - System fails when no path from source to target
  - Compute probability of a path from source to target

Reliability Analysis of Boeing 787

- Current Return Network Modeled as a Reliability Graph

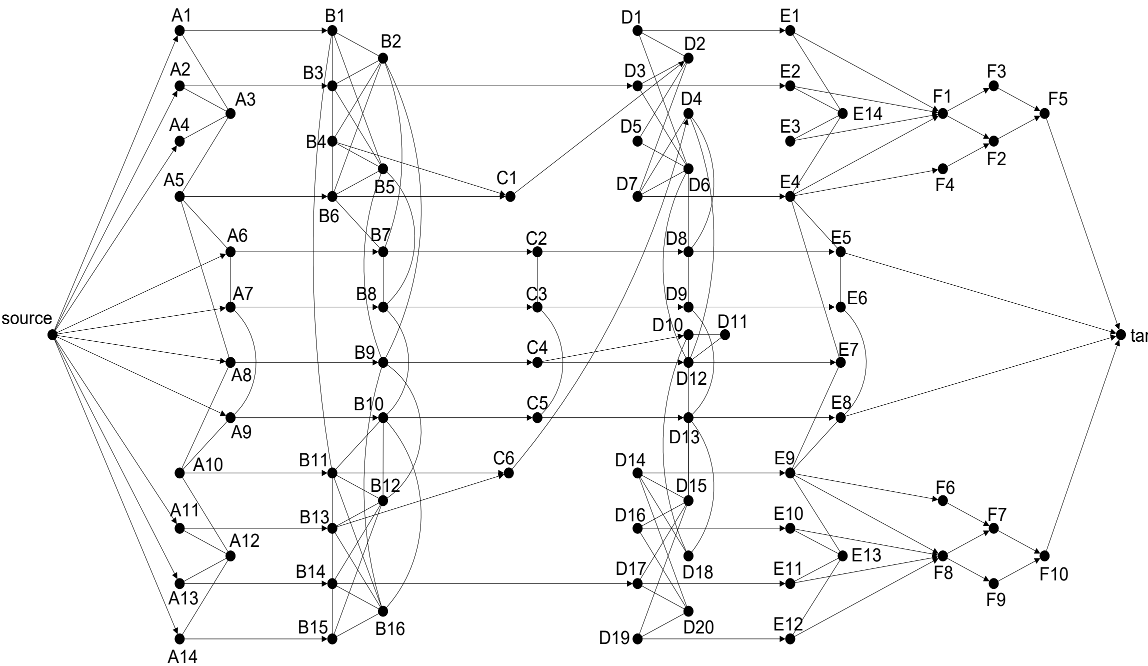


Reliability Analysis of Boeing 787 (cont'd)

- Solution methods implemented in our SHARPE software package for relgraph
  - Find all minpaths followed by SDP (Sum of Disjoint Products)
  - BDD (Binary Decision Diagrams)-based method
- Boeing tried to use SHARPE for this problem but …

Reliability Analysis of Boeing 787 (cont'd)

- Too many minpaths



node	#paths
E_7 →target	40
D_{12} →target	143140
C_4 →target	308055
B_9 →target	21054950355
A_8 →target	461604232201
source → target	4248274506778 $\approx 4 \times 10^{12}$

- Idea: Compute bounds instead of exact reliability
- Lower bound by taking a subset of minpaths
- Upper bound by taking a subset of mincuts

Reliability Analysis of Boeing 787 (cont'd)

- **Our Approach** : Developed a new efficient algorithm for (un)reliability bounds computation and incorporated in SHARPE

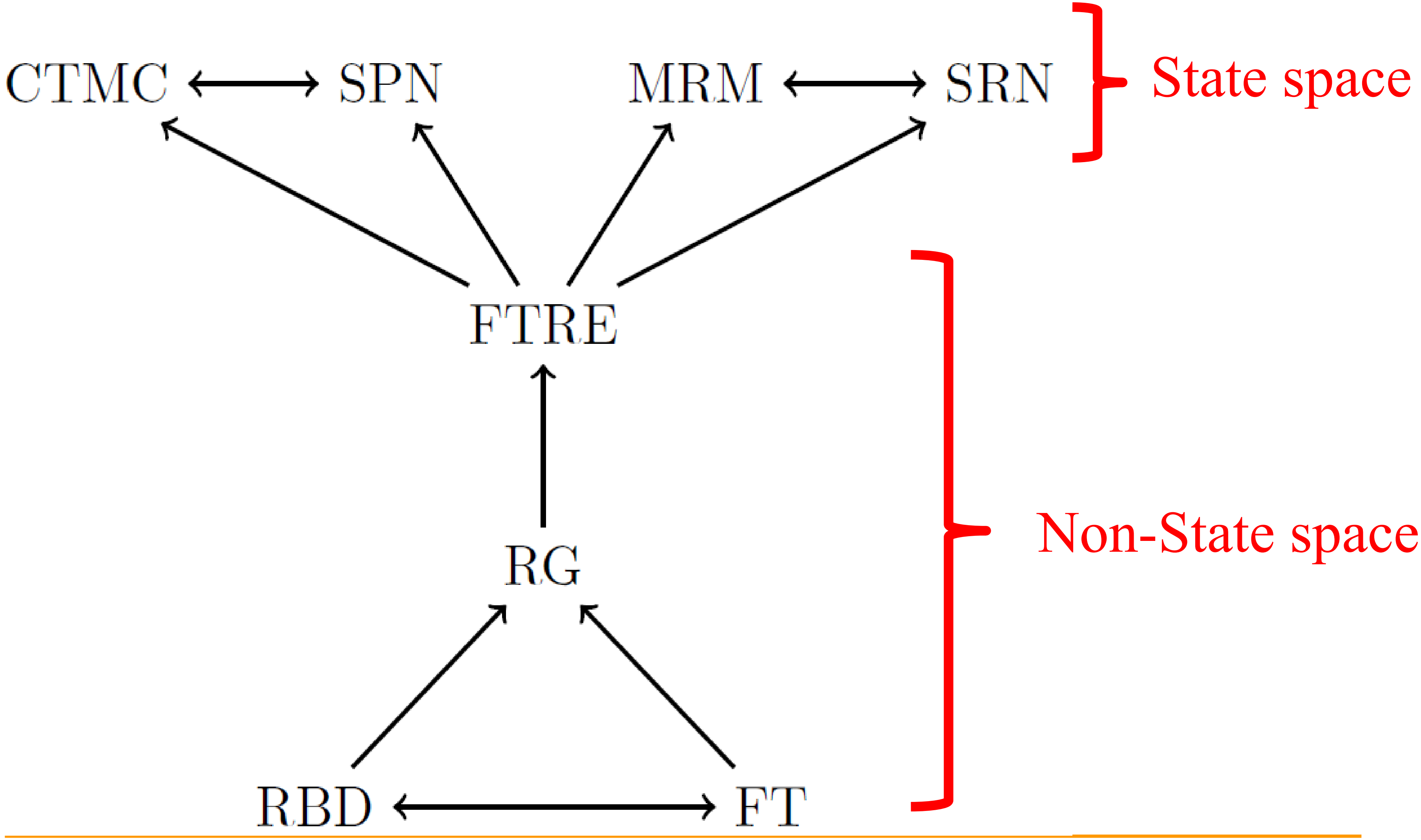
runtime	20 seconds	120 seconds	900 seconds
upper bound	1.1460365721e-008	1.0814324701e-008	1.0255197263e-008
lower bound	1.0199959877e-008	1.0199959877e-008	1.0199959877e-008

- 2011 patent for the algorithm jointly with Boeing/Duke
- "Fast computation of bounds for two-terminal network reliability", EJOR 2014
- Satisfying FAA that SHARPE development used DO-178 B software standard was the hardest part
- As per A.V. Ramesh (Boeing), this algorithm (and SHARPE) are always used for modeling CRN subsystem in other Boeing commercial aircraft

RBD->Relgraph->ftree

- Series-parallel RBD and Fault trees without repeated event are equivalent
- Relgraph is more powerful than RBD since non-series-parallel behavior can be accommodated
- Fault trees with repeated event are more powerful than relgraphs
- Most scalable method is the bounding algorithm for relgraphs; this needs to be extended to fault trees

Power-hierarchy of modeling formalisms



Non-state-space Methods (cont'd)

- Non-state-space methods are easy to use and have relatively fast algorithms for system reliability, system availability, system MTTF & to find bottlenecks assuming stochastic independence between system components
  - Series-parallel composition algorithm
  - Factoring (conditioning) algorithms
  - All minpaths followed by Sum of Disjoint Products (SDP) algorithm
  - Binary Decision Diagrams (BDD) based algorithms
  - Bounding algorithm for relgraphs
- All of the above implemented in SHARPE
- Failure/Repair Dependencies are often present; RBDs, relgraphs, FTREEs cannot easily handle these (e.g., shared repair, warm/cold spares, imperfect coverage, non-zero switching time, travel time of repair person, reliability with repair).

Statistical Dependence

The independence assumption is often unrealistic

Dependencies in the failure process are

- load dependencies,
- functional dependencies,
- cascading failures
- common cause failures
- Coincident (or near-coincident) faults

Dependencies in the repair process

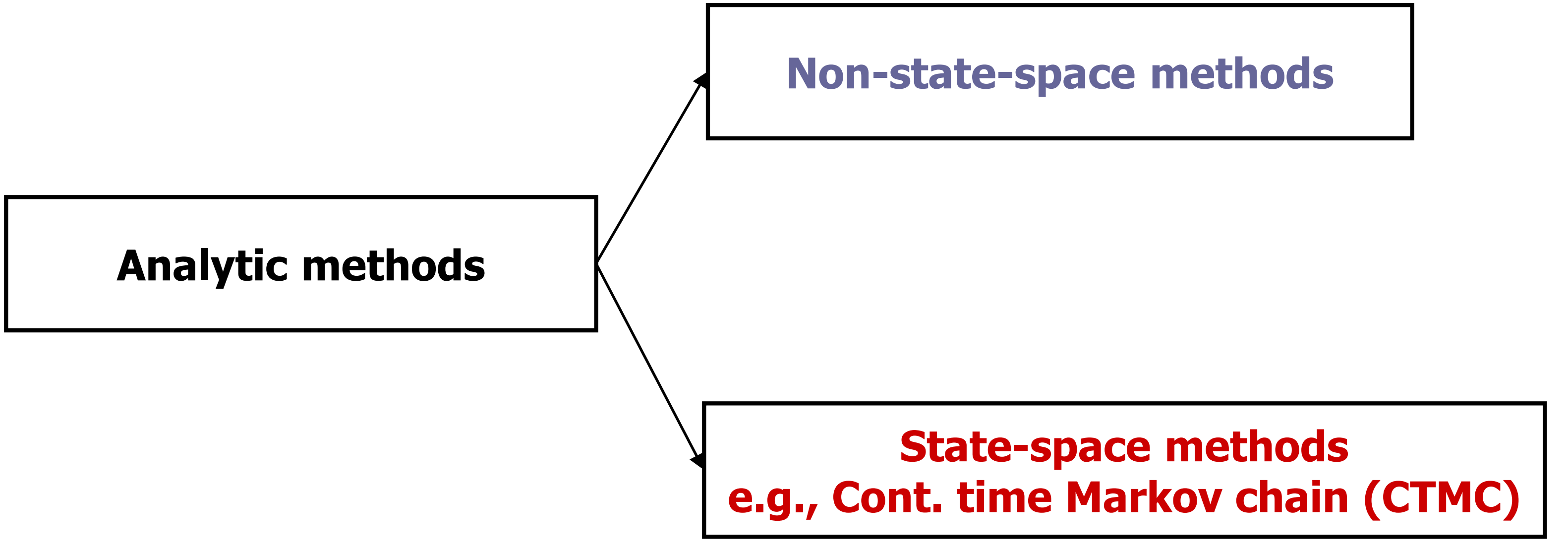
- deferred maintenance
- shared repair facilities.

R. Fricks and K. Trivedi, "Modeling failure dependencies in reliability analysis using stochastic Petri nets," in *Proc. European Simulation Multi-Conference (ESM '97)*, 1997.

State-space methods : Markov chains

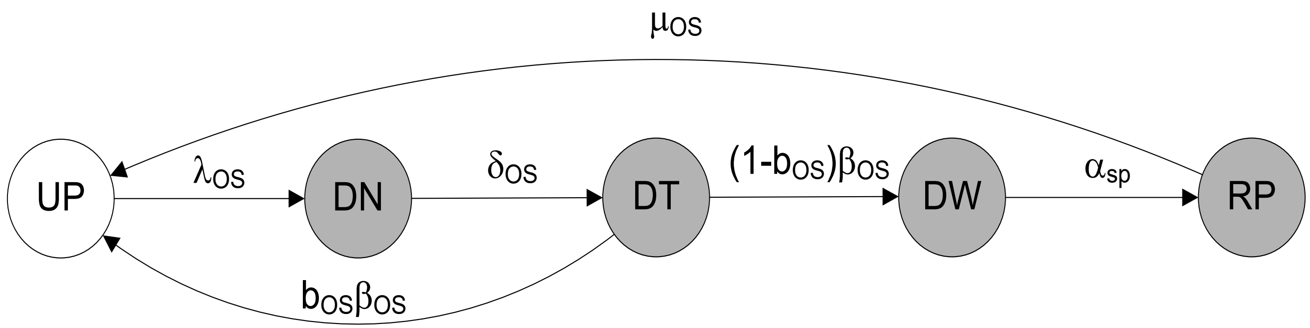
- To model complex interactions between components, need to use paradigms like Markov chains or more generally state space models.
- Many examples of dependencies among system components have been observed in practice and captured by continuous-time Markov chains (CTMCs).
- Extension to Markov reward models makes computation of measures of interest relatively easy.

Analytic Methods Taxonomy



Markov model of SIP on IBM WebSphere

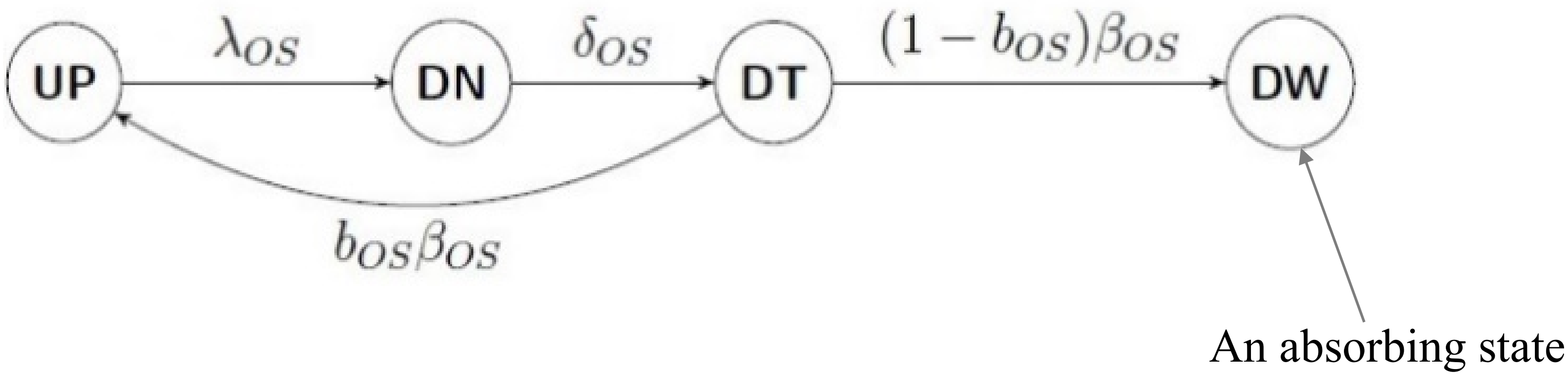
- A CTMC availability model of the Linux OS



Detection delay, imperfect coverage, two-levels of recovery modeled

How can you turn this into a reliability model?

A CTMC Reliability model of the Linux OS



A reliability model will have one or more absorbing states

An availability model will have no absorbing states

SHARPE Input file for the Linux Model

```
echo Linux OS Availability Model
markov LinuxOS
1 2 los
2 3 dos
3 1 bos*beta
3 4 (1-bos)*beta
4 5 asp
5 1 mos
end
```

```
* Parameter values in per  hr
bind
los 1/4000
dos 1
beta 6
bos 0.9
asp 1/2
mos 1
end

echo  Steady-state availability
equal to probability of state UP

expr prob (LinuxOS,1)

end
```

SHARPE Output file for the Linux Model

Linux OS Availability Model

Steady-state availability equal to probability of state UP

prob (LinuxOS,1): 9.99633468e-001

SHARPE Input file for the Linux Reliability Model (state 4 absorbing)

```
echo Linux OS Reliability Model
markov LinuxOS
1 2 los
2 3 dos
3 1 bos*beta
3 4 (1-bos)*beta
end
* initial state probabilities
1 1.0
2 0.0
3 0.0
4 0.0
end
```

```
* Parameter values
bind
los 1/4000
dos 1
beta 6
bos 0.9
asp 1/2
mos 1
end

echo  Reliability at times 0 thru 10000 in
steps of 2000 equal to probability of state 1
func rel(t) tvalue(t;LinuxOS,1)
loop t,0,  10000, 2000
expr rel(t)
end
end
```

SHARPE Input file for the Linux Reliability Model (state 4 absorbing)

Linux OS Reliability Model
Reliability vs time equal to probability of state 1 at time t
System reliability at times 0 thru 10000 in steps of 2000

t=0.000000
rel(t): 1.00000000e+000

t=2000.000000
rel(t): 9.50987909e-001

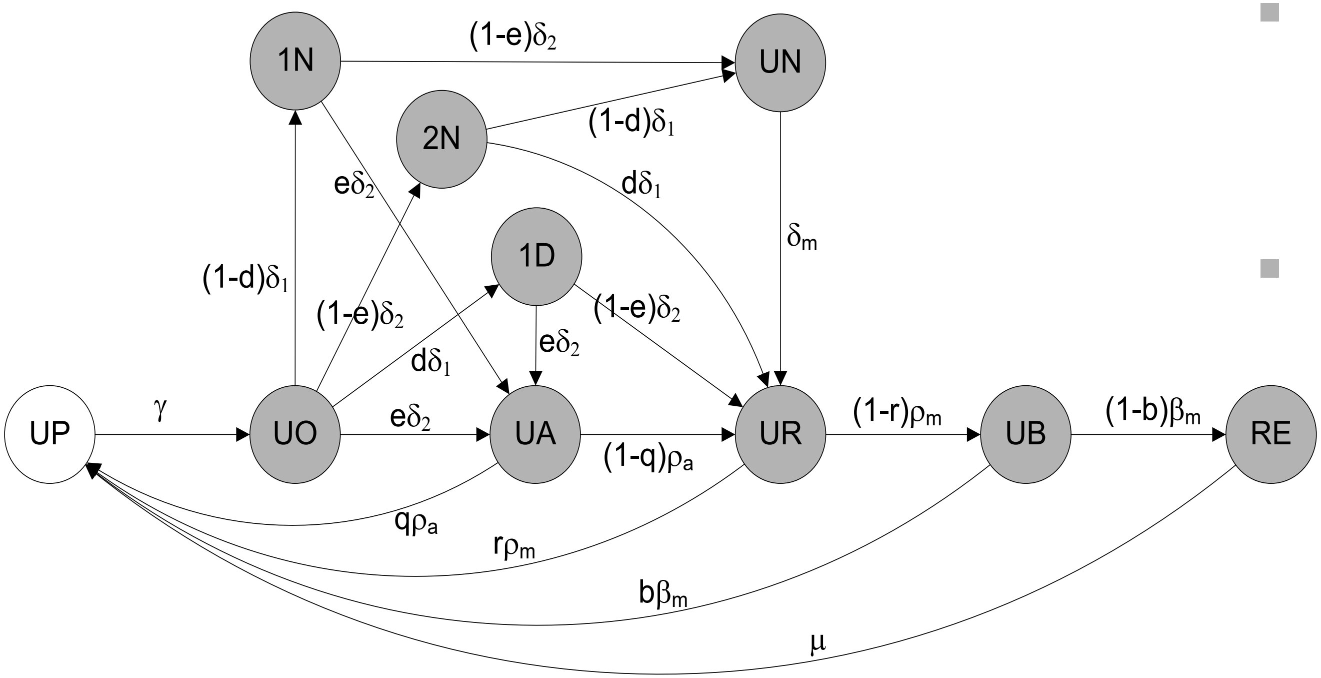
t=4000.000000
rel(t): 9.04617746e-001

t=6000.000000
rel(t): 8.60508971e-001

t=8000.000000
rel(t): 8.18551046e-001

t=10000.000000
rel(t): 7.78639141e-001

Markov (CTMC) Availability model of App Server



- Failure detection
  - By WLM
  - By Node Agent
  - Manual detection
- Recovery
  - Node Agent
    - Auto process restart
  - Manual recovery
    - Process restart
    - Node reboot
    - Repair

Application server and proxy server (with escalated levels of recovery)

Delay and imperfect coverage in each step of detection/recovery modeled

CTMC with Infinitesimal Generator matrix Q

- Efficient/Scalable algorithms are known & are implemented in software packages SHAREPE, SPNP for:
- Steady-state behavior:

$$\pi Q = 0, \qquad \sum_i \pi_i = 1$$

- Transient behavior:

$$\frac{d\pi(t)}{dt} = \pi(t)Q, \qquad given \ \ \pi(0)$$

- Cumulative Transient behavior:

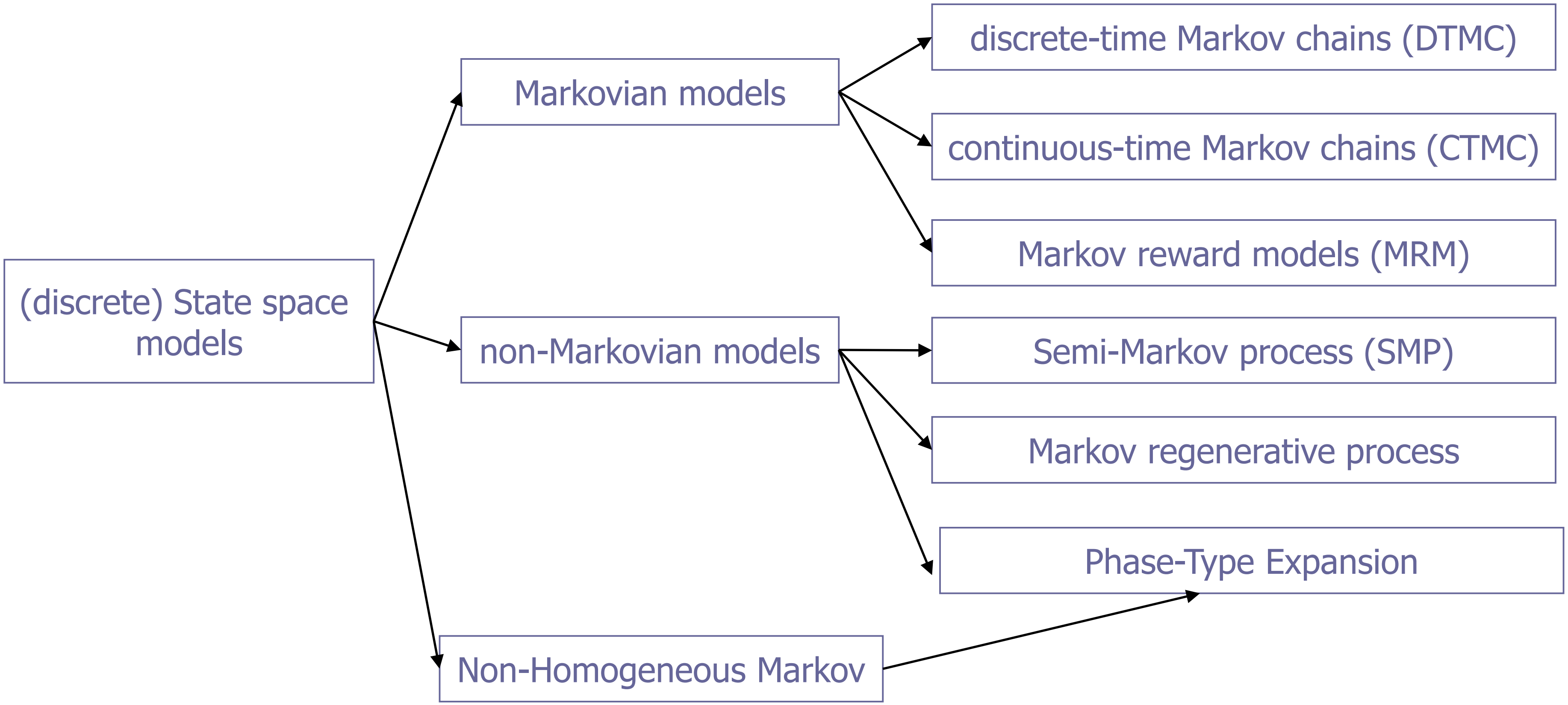
$$dL(t)/dt = L(t)\,\mathbf{Q} + \pi(0)$$

***L(t)*: integrals of state probability vector**

- Also derivatives of the probabilities with respect to parameters – parametric sensitivity functions are computed

State-Space methods taxonomy

Can relax the assumption of exponential distributions



Should I Use Markov (CTMC) Models?

+ Model Fault-Tolerance and Recovery/Repair

+ Model Dependencies

+ Model Contention for Resources and concurrency (performance)

+ Generalize to Markov Reward Models for Degradable systems

+ Can relax exponential assumption – SMP, MRGP, NHCTMC, PH

+ Performance, Availability, Performability, Survivability, Resilience Modeling Possible

- Large State Space

Markov Models

- Modeling inter-dependence among components
  - Simple model types such as RBD, Ftree, etc. do not suffice – need to use Markov and other state space model types
- State space explosion problem

Problems with Markov (or State Space) Models and their solutions

- State space explosion or the model largeness problem or scalability problem
- Stochastic Petri nets and related formalisms (stochastic process algebras) for ease of specification and automated generation/solution of underlying Markov model ---
- This is called Largeness Tolerance

Scalable Model for IaaS Cloud Availability and Downtime

Ref: Ghosh, Longo, Frattini, Russo, Trivedi, "Scalable Analytics for IaaS Cloud Availability," *IEEE Trans. Cloud Comput.*, 2014

Three Pools of Physical Machines (PMs)

- To reduce power usage costs, physical machines are divided into three pools [IBM Research Cloud]
  - Hot pool (high performance & high power usage)
  - Warm pool (medium performance & power usage)
  - Cold pool (lowest performance & power usage)

Similar grouping of PMs is recommended by Intel*

*Source: http://www.intel.com/content/dam/www/public/us/en/documents/guides/lenovo-think-server-smart-grid-technology-cloud-builders-guide.pdf

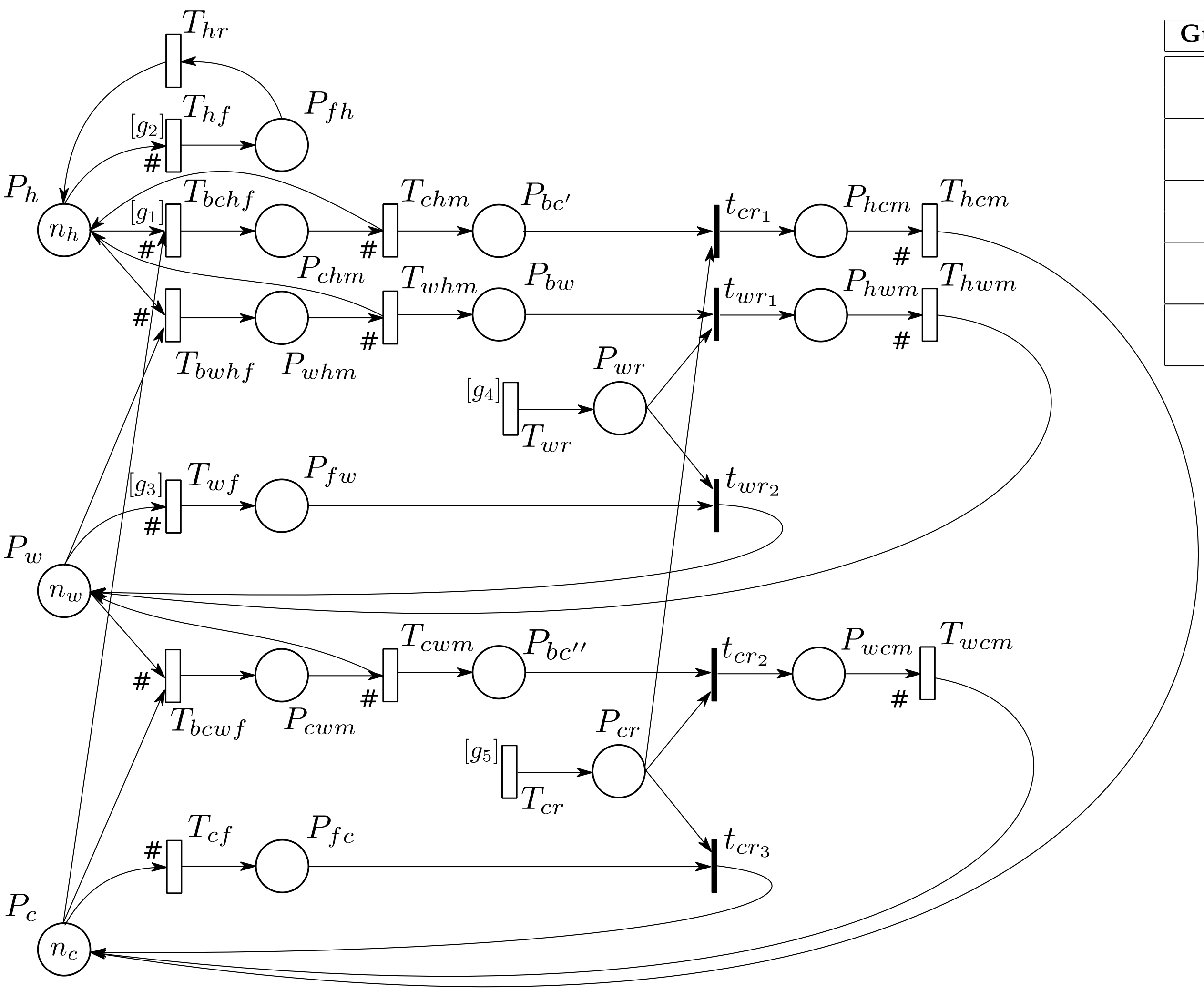
System Operation Details

- Failure/Repair (Availability):
  - PMs may fail and get repaired.
  - A minimum number of operational hot PMs are required for the system to function.
  - PMs in other pools may be temporarily assigned to the hot pool to maintain system operation (migration).
  - Upon repair, PMs migrate back to their original pool
  - Migration creates dependence among pools

Analytic model

- Markov model (CTMC) is too large to construct by hand.
- We use a high-level formalism of stochastic Petri net (the flavor known as stochastic reward net (SRN)).
- SRN models can be automatically converted into underlying Markov (reward) model and solved for the measures of interest such as DT (downtime), steady-state (instantaneous, interval) availability, reliability, derivatives of these measures --- all numerically by forming and solving underlying equations
- Analytic-numeric solution as opposed to discrete-event simulation
- Ref: Ciardo, Blakemore, Chimento, Muppala, Trivedi, "Automated generation and analysis of Markov reward models using stochastic reward nets," *Linear Algebra, Markov Chains, and Queueing Models*, Springer, 1993

Monolithic Stochastic Reward Net Model



Guard functions	Values
$[g_1]$	1 if $\#P_w = 0$ 0 otherwise
$[g_2]$	1 if $\#P_w = 0$ and $\#P_c = 0$ 0 otherwise
$[g_3]$	1 if $\#P_c = 0$ 0 otherwise
$[g_4]$	1 if $\#P_{fw} + \#P_{bw} > 0$ 0 otherwise
$[g_5]$	1 if $\#P_{fc} + \#P_{bc'} + \#P_{bc''} > 0$ 0 otherwise

Other High-Level Formalisms

- Many other High-level formalism (like SRN) are available and corresponding software packages exist (SAN, SPA, ….)
- Can generate/store/solve moderate size Markov models
- Have been extended to non-Markov and fluid (continuous state) models [MRSPN, FSPN]
- Ref: Choi, Kulkarni, Trivedi, "Markov Regenerative Stochastic Petri Nets," *Perform. Evaluation,* 1994
- Ref: Horton, Kulkarni, Nicol, Trivedi, "Fluid stochastic Petri nets: Theory, applications, and solution techniques," *Eur. J. Oper. Res.,* 1998

Monolithic Model

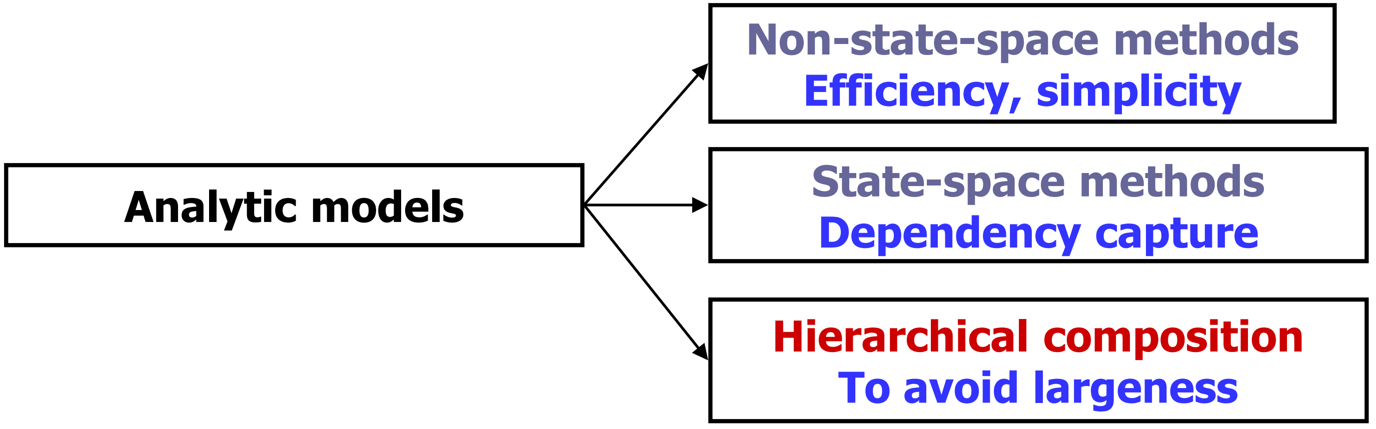
- Monolithic SRN model is automatically translated into CTMC or Markov Reward Model
- However the model not scalable as state-space size of this model is extremely large

#PMs per pool	#states	#non-zero matrix entries
3	10, 272	59, 560
4	67,075	453, 970
5	334,948	2, 526, 920
6	1,371,436	11, 220, 964
7	4,816,252	41, 980, 324
8	Memory overflow	Memory overflow
10	-	-

Problems with Markov (or State Space) Models and their solutions

- State space explosion or the largeness problem
- Stochastic Petri nets and related formalisms for easy specification and automated generation/solution of underlying Markov model --- Largeness Tolerance
- Use hierarchical (Multilevel) model composition
  - Largeness Avoidance
  - e.g. Upper level : FT or RBD, lower level: Markov chains
  - Many practical examples of the use of hierarchical models exist
  - Can also use state truncation

Analytic Modeling Taxonomy



State Space Explosion

- Number of components in systems can be hundreds, nay thousands!
- Number of states in a Markov model will be a gazillion!
- State space explosion can be avoided by decomposing system into subsystems, modeling each subsystem separately and then composing sub-model results together – SHARPE facilitates this
- Use state-space methods for those subsystems that require them, and use simple non-state-space methods (RBD, Ftree) for the more “well-behaved” parts of the system

Largeness Avoidance

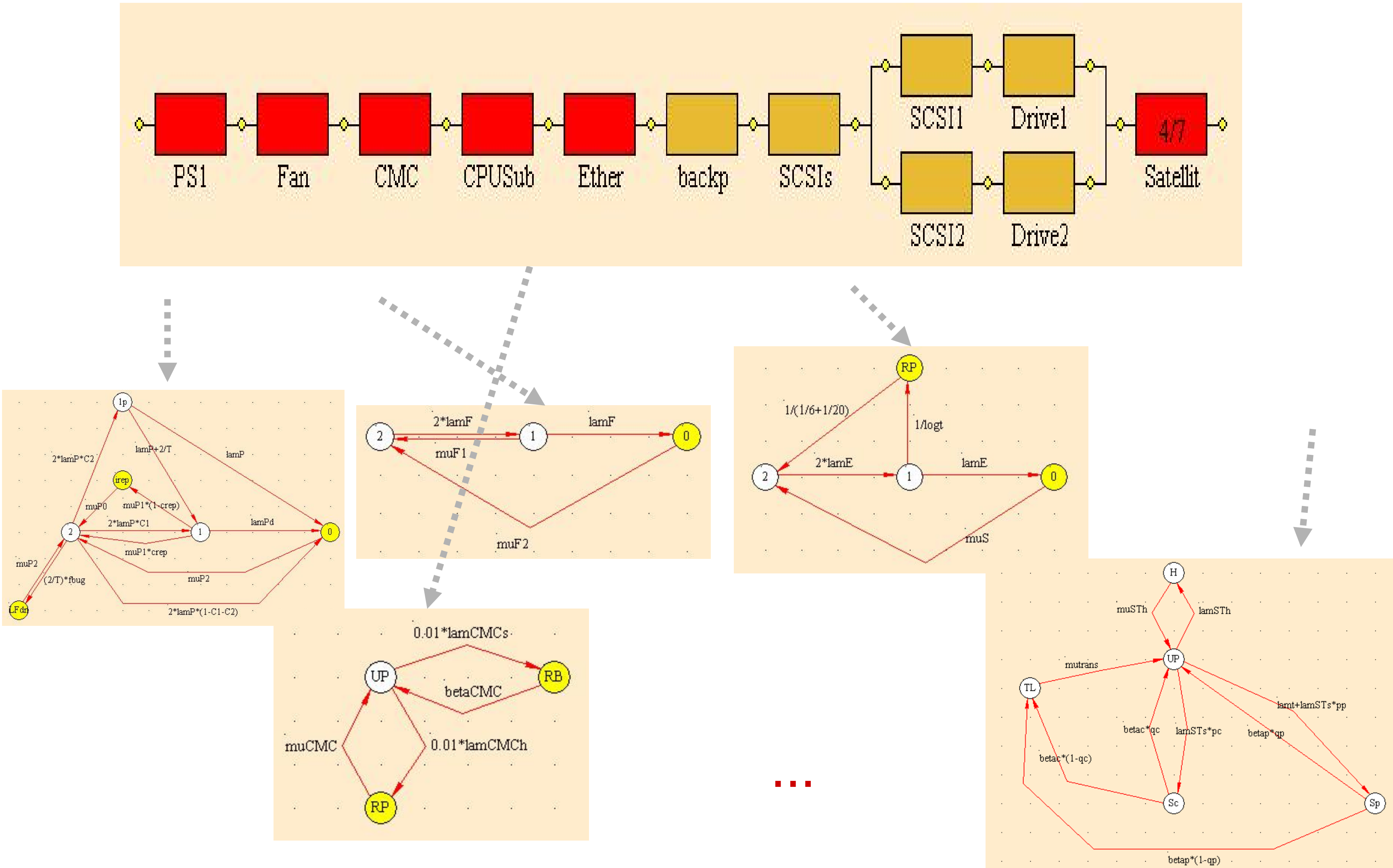
- Main reason for hierarchical (or multilevel) models: avoid generating/solving large monolithic models; that is for tractability
- In SHARPE we can mix and match different paradigms and to arbitrary levels
- Can choose the “right” paradigm for each subsystem
- Note that some tools/approaches use hierarchy merely for specification and a monolithic model is constructed by the tool
- We are advocating hierarchy not only for specification but also for solution
- Hierarchy does not always mean an approximation
- Most practical problems I have solved have 2 or more levels with the top level being RBD/ftree and Markov models at the lowest level

Availability Analysis: SUN Microsystems

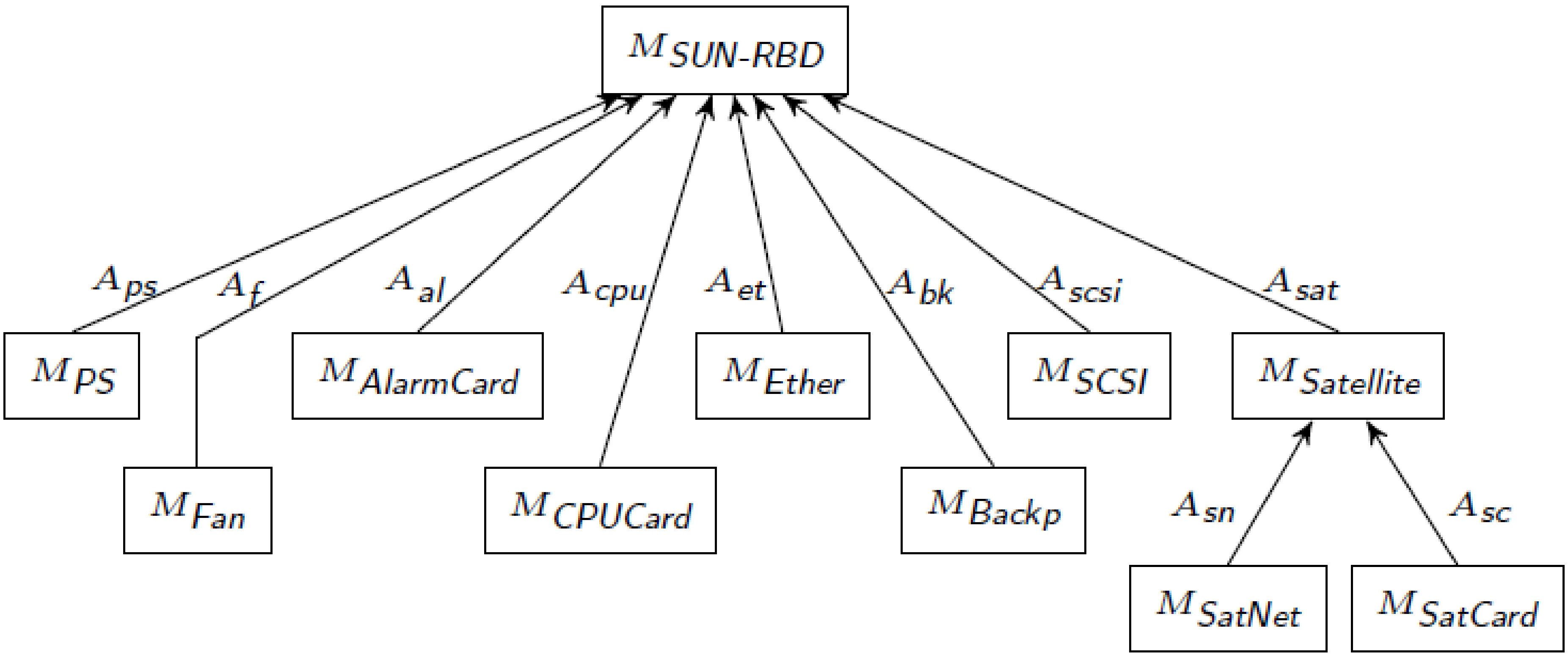
- Carrier-Grade High Availability Software Platform
- Model taking into account hardware component failures, software component failures and various types of recovery
- Hierarchical model composition – Markov chains at the lower-level, RBD at the top level

- Ref: Trivedi, Vasireddy, Trindade, Nathan, Castro, “Modeling High Availability Systems,” Proc. PRDC 2006.

Sun Microsystems – overall model hierarchy



Import Graph – SUN Model



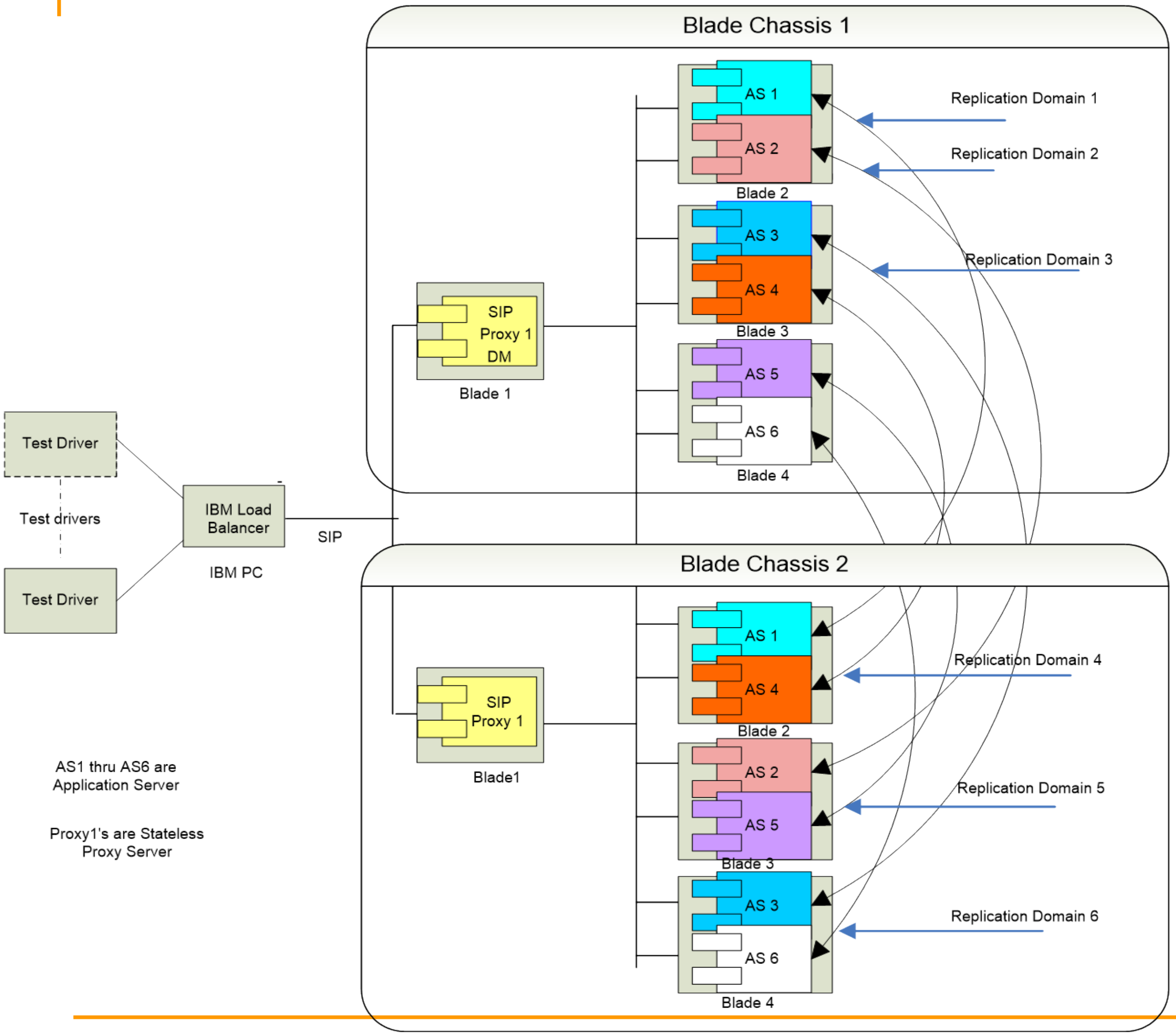
In the Import graph, Nodes are submodels

Arc indicates output of a submodel as an input parameter to another submodel

High-Availability SIP System

- Real problem from IBM
- SIP: Session Initiation Protocol
- Hardware platform: IBM Blade Center
- Software platform: IBM WebSphere
- A Telco (potential) customer asked IBM for models to quantify this product
- IBM asked me to lead the modeling project
  - To quantify system (steady-state) availability
    Ref: Trivedi, Wang, Hunt, Rindos, Smith, Vashaw, "Availability Modeling of SIP Protocol on IBM WebSphere," *PRDC 2008*
  - To quantify a user-oriented metric called DPM
    Ref: Trivedi, Wang & Hunt. "Computing the number of calls dropped due to failures," *ISSRE2010*

Architecture of SIP on IBM WebSphere



AS: WebSphere Appl. Server (WAS)

Replication domain	Nodes
1	A, D
2	A, E
3	B, F
4	B, D
5	C, E
6	C, F

Architecture of SIP on IBM WebSphere

- Hardware configuration:
  - Two BladeCenter chassis; 4 blades (nodes) on each chassis (**1 chassis would have been sufficient from the performance perspective**)

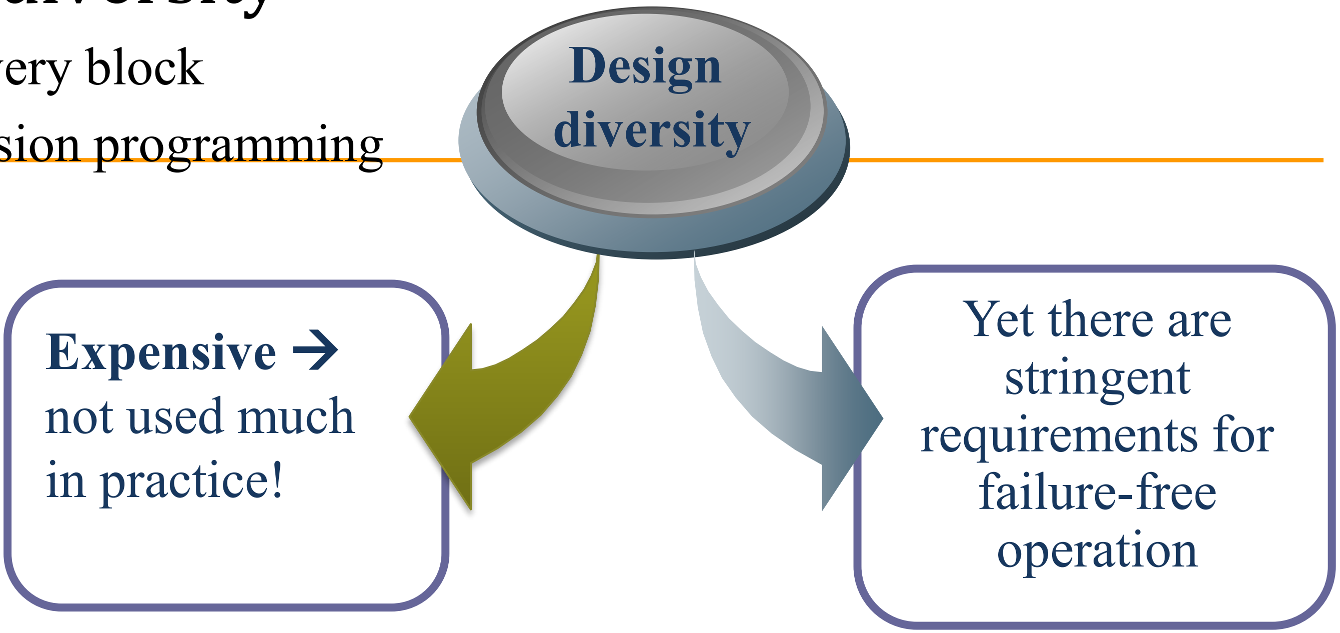
- Software configuration:
  - 2 copies of SIP/Proxy servers (**1 sufficient for performance**)
  - 12 copies of WAS (**6 sufficient for performance**)
  - Each WAS instance forms a redundancy pair (**replication domain**) with WAS installed on another node on a different chassis

The system has both, hardware redundancy and software redundancy

Software Fault Tolerance:

Classical Techniques

- Design diversity
  - Recovery block
  - N-version programming
  - ……



Challenge: *Affordable* Software Fault Tolerance

A possible answer: Environmental Diversity

SIP Application Server on IBM WebSphere

Software Redundancy

- Identical copies of SIP proxy used as backups (**hot spares**)
- Identical copies of WebSphere Applications Server (WAS) used as backups (**hot spares**)
- **Type of software redundancy** – (not design diversity) but replication of identical software copies
- **Normal recovery after a software failure – uses time redundancy**
  - ✓ Restart software, reboot node or fail-over to a software replica; only when all else fails, a "software repair" is invoked

Software Fault Tolerance: New Thinking

1

Have been known to help in dealing with hardware transients

Retry

Restart

Reboot!

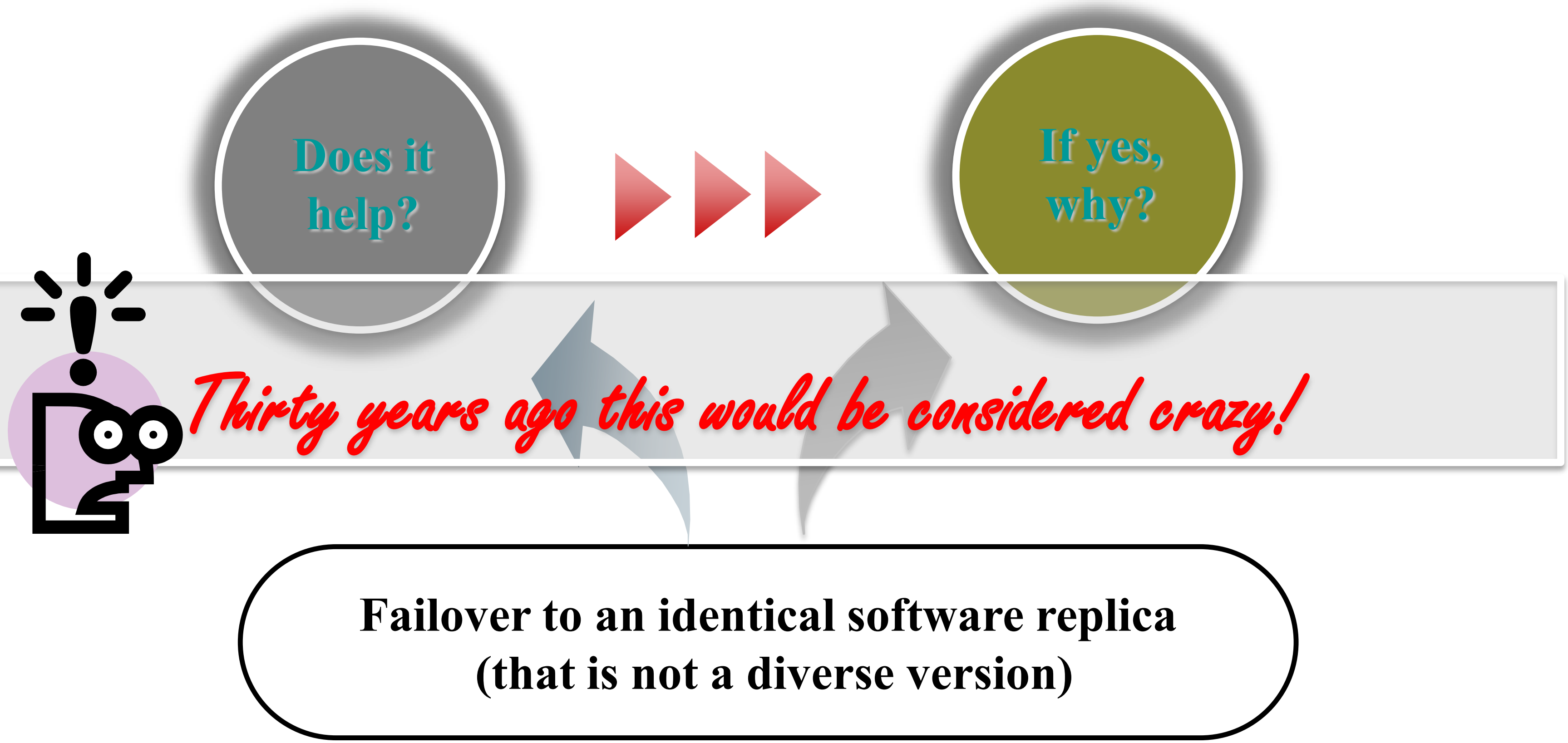
2

RQ: Do they help in dealing with failures caused by software bugs?

3

If yes, why?

Software Fault Tolerance: New Thinking



Software fault classification

Bohrbug (BOH) := A fault that is easily isolated and that manifests consistently under a well-defined set of conditions, because its activation and error propagation lack complexity.

Non-Aging related Mandelbug (NAM) := A fault whose activation depends on the environment besides the workload. Environment refers to other applications concurrently running, interactions with OS and hardware

Aging related bug (ARB) := A fault that leads to the accumulation of errors either inside the running application or in its system-context environment, resulting in an increased failure rate and/or degraded performance.

Ref:. Grottke, Trivedi, "Fighting Bugs: Remove, Retry, Replicate and Rejuvenate," *IEEE Computer*, 2007

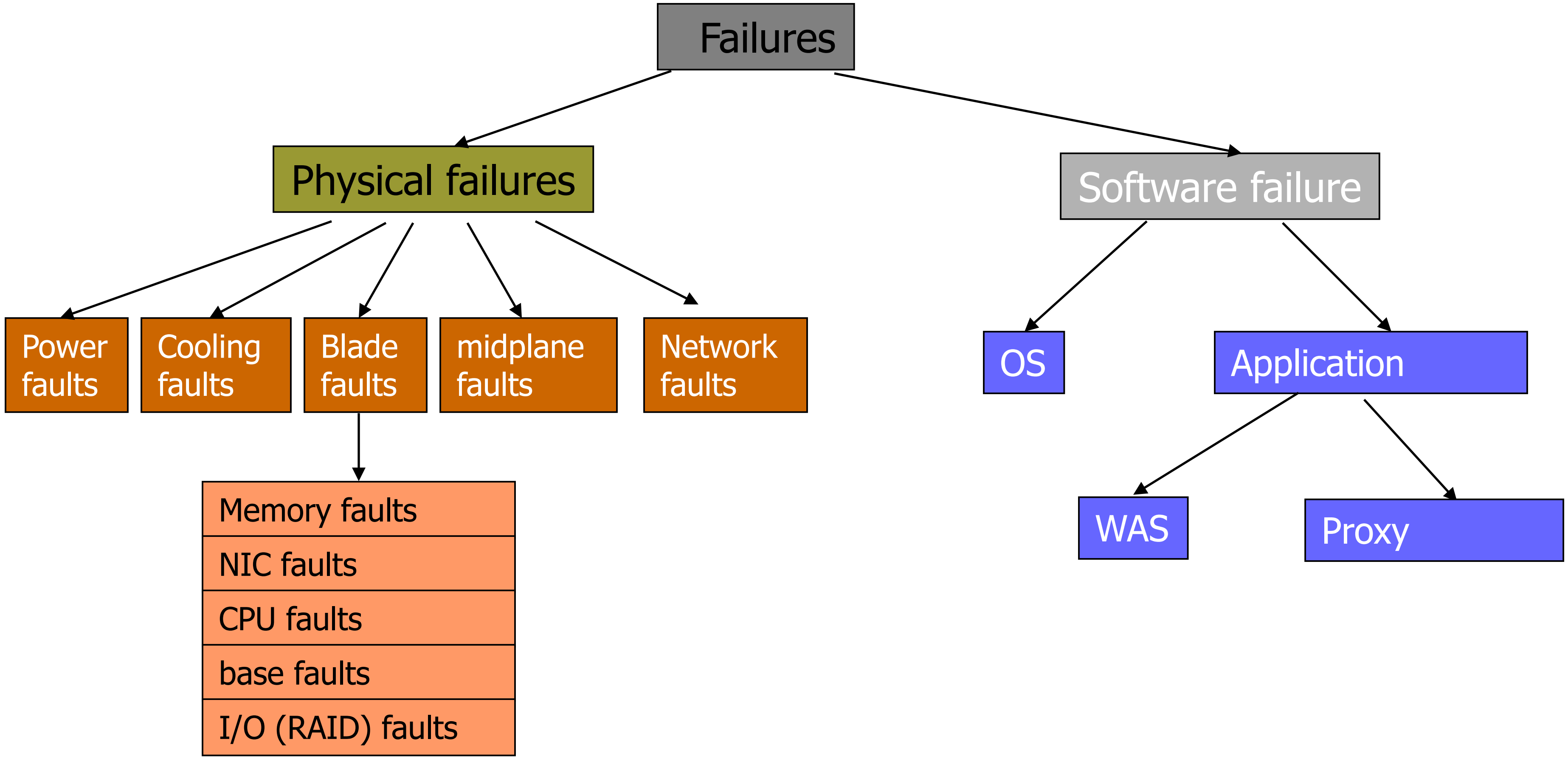
Software Fault Tolerance: New Thinking

- **Environmental Diversity** as opposed to **Design Diversity**

- Our claim is that this (**retry**, **restart**, **reboot**, **failover to identical software copy**) may well work since failures due to **Mandelbugs** are not negligible. We thus have an affordable software fault tolerance technique that we call **Environmental Diversity**

Back to the Availability Model

112 components (hardware and software)



Availability model of SIP on IBM WebSphere

- Single monolithic Markov model will have extraordinarily large number of states – we use a multi-level approach
- Subsystems modeled using Markov chains to capture dependence within
- Fault tree used at higher levels as independence across subsystems can be reasonably assumed
- This is an example of hierarchical composition
    - A single monolithic model is not constructed/stored/solved
    - Each submodel is built and solved separately and results are propagated up to the higher-level model
    - Our software package SHARPE facilitates such hierarchical model composition

Availability model of SIP on IBM WebSphere

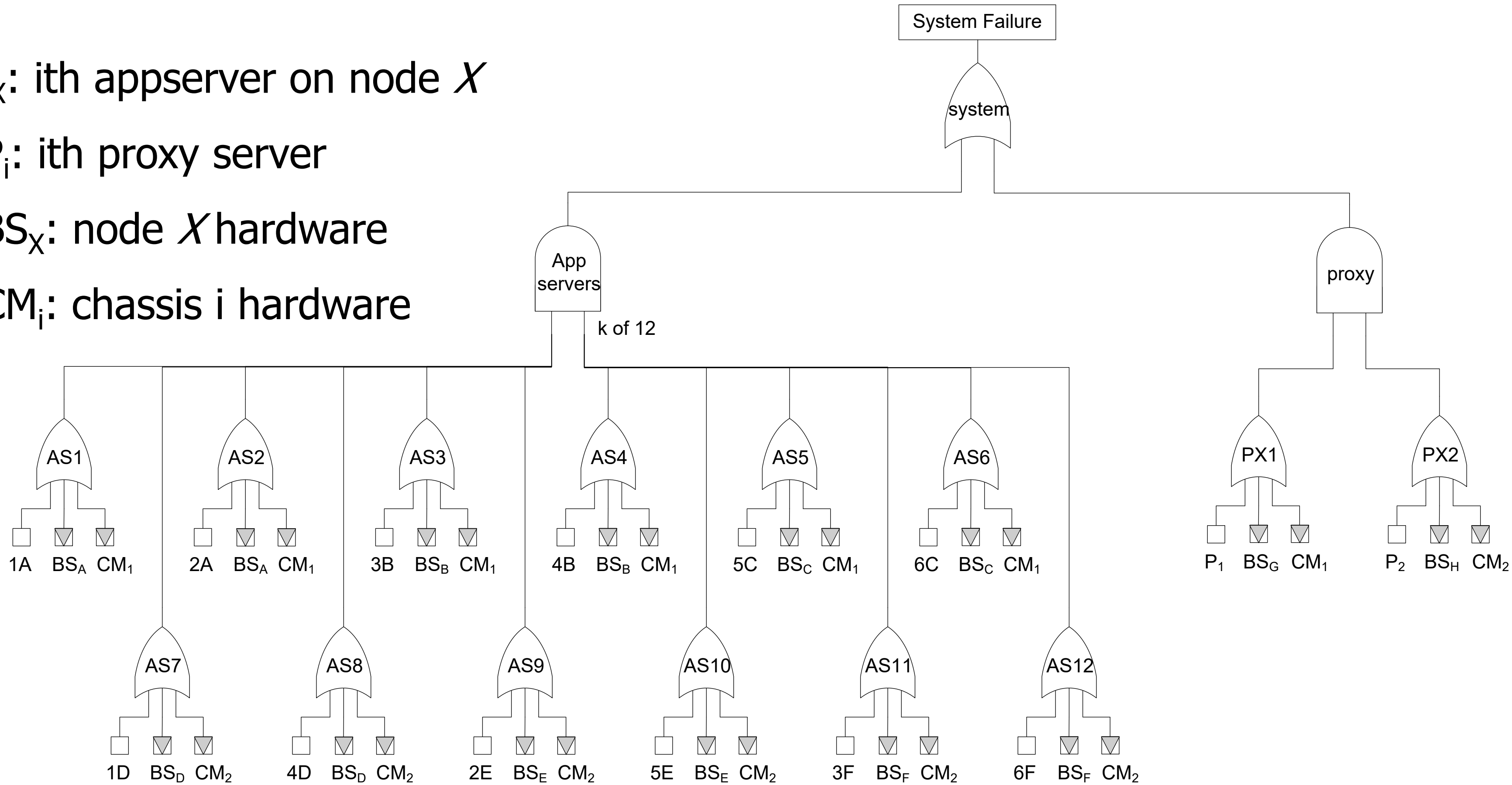
- SIP top level of the availability model

i_X: ith appserver on node *X*

P_i: ith proxy server

BS_X: node *X* hardware

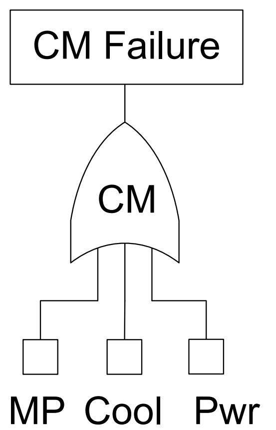
CM_i: chassis i hardware



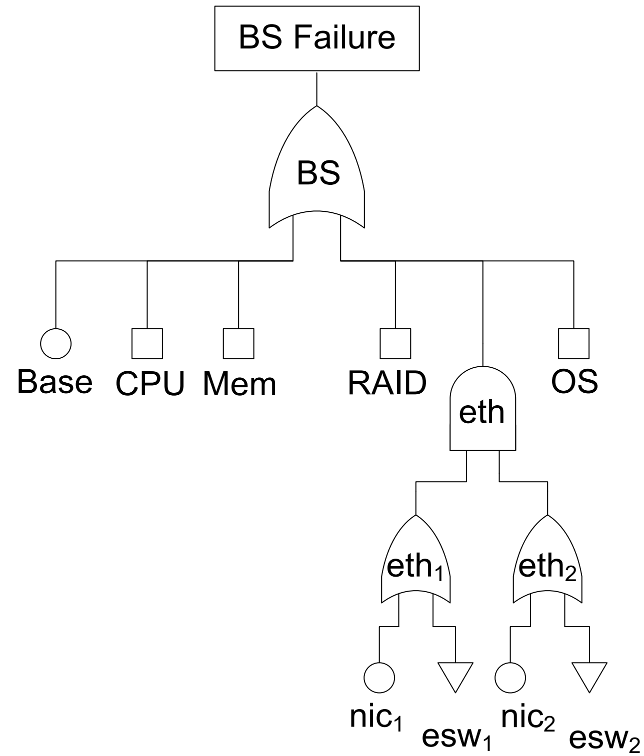
Availability model of SIP on IBM WebSphere

- Availability models of a Blade Server and Common Blade Center Hardware

A circle as a leaf node is a basic event
An inverted triangle is a shared event
A square indicates a submodel



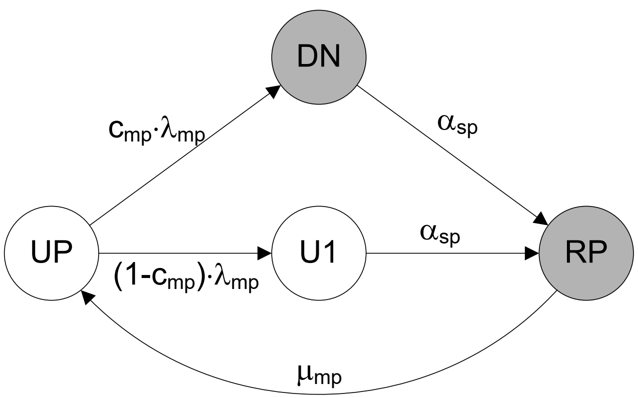
Chassis failure



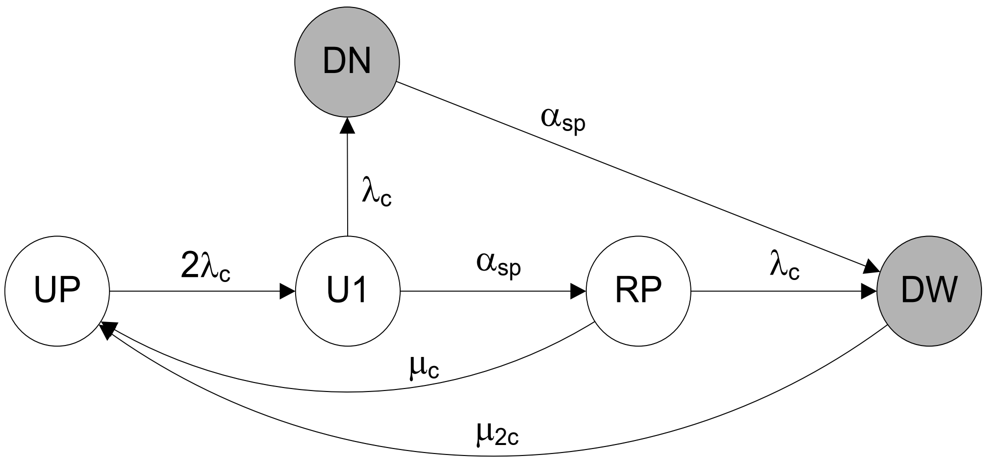
Blade server failure

Availability model of SIP on IBM WebSphere

- Markov Availability models of subsystems



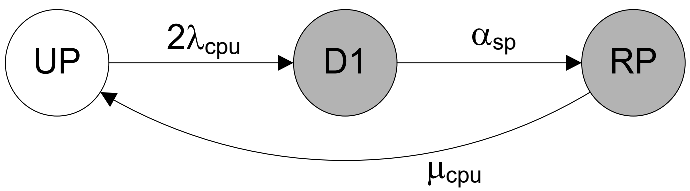
midplane model



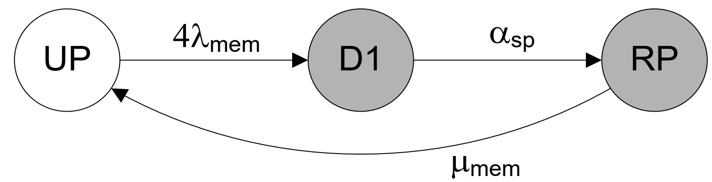
Cooling subsystem model

Availability model of SIP on IBM WebSphere

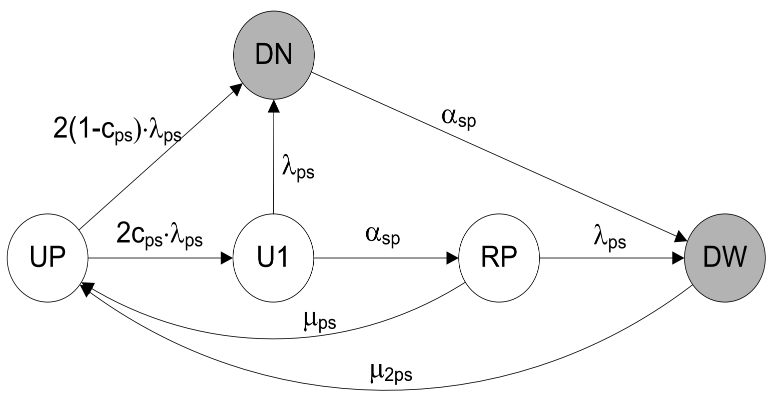
- Availability models of subsystems



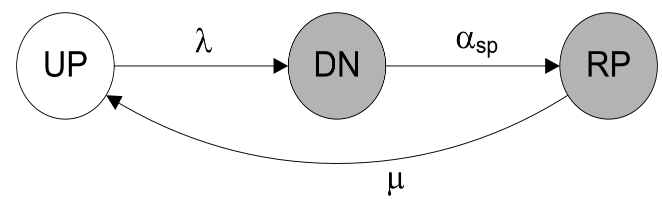
CPU model



memory model



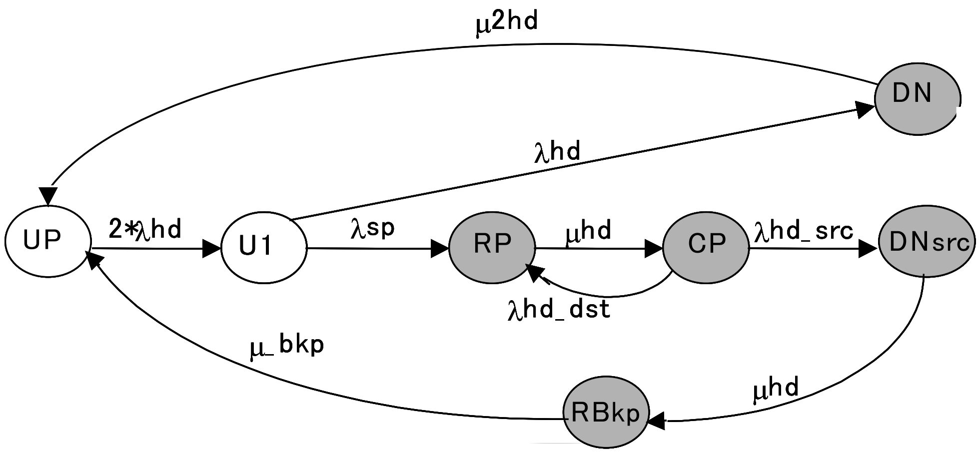
Power domain model



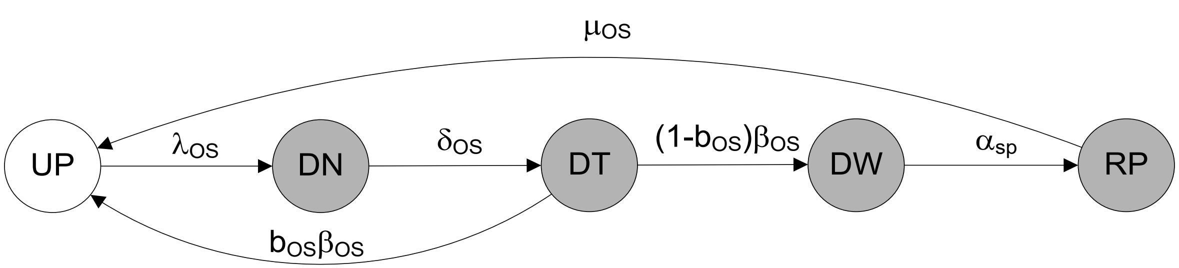
Base, Switch and NIC

Availability model of SIP on IBM WebSphere

- Availability models for subsystems (cont.)

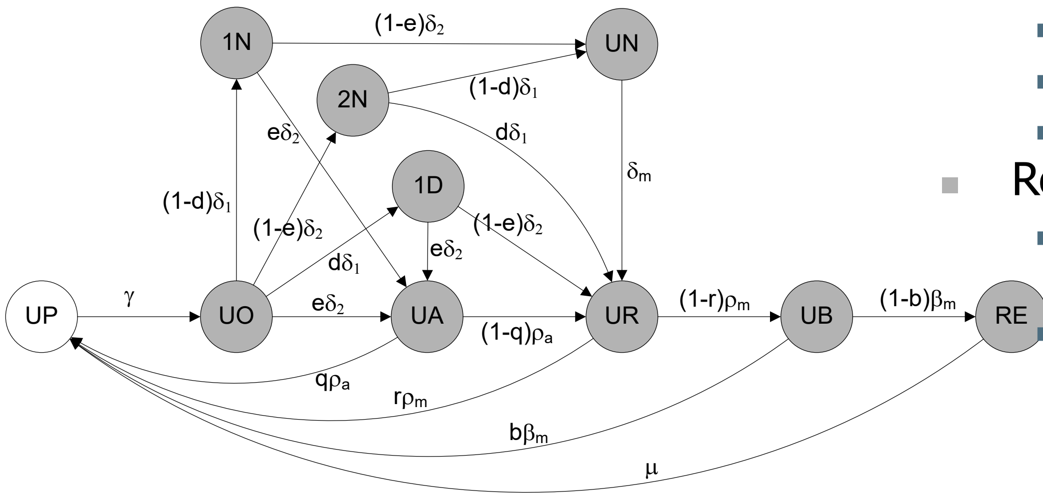


RAID model



OS model

Markov Availability model WebSphere AP Server

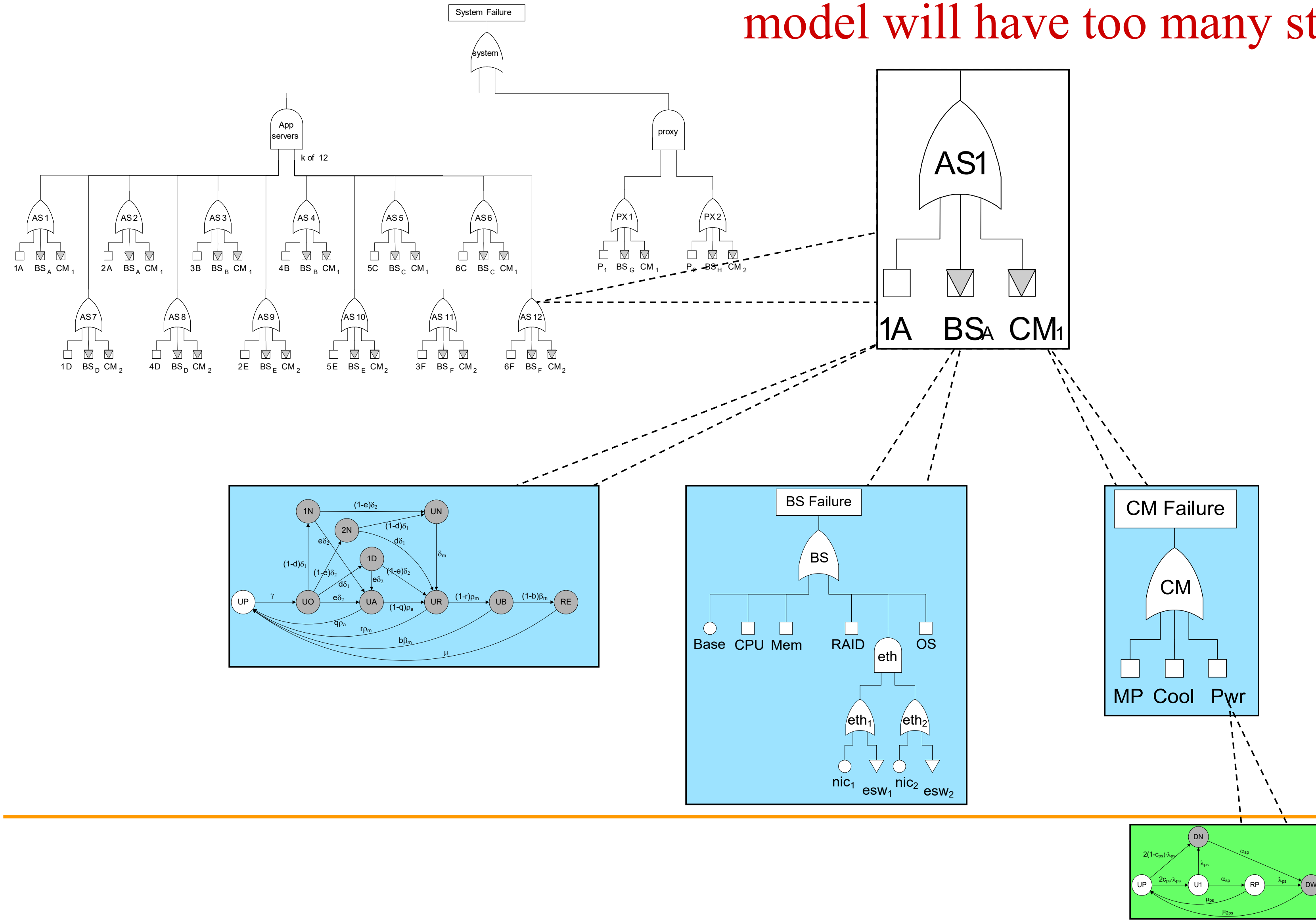


- Failure detection
  - By WLM
  - By Node Agent
  - Manual detection
- Recovery
  - Node Agent
    - Auto process restart
  - Manual recovery
    - Process restart
    - Node reboot
    - Repair

- Application server and proxy server (with escalated levels of recovery)
- Delay and imperfect coverage in each step of recovery modeled
- Use of restart, failover to an identical replica or reboot as a method of recovery after a software failure

Hierarchical Composition

A single monolithic Markov model will have too many states



Model Parameterization

- Types of parameters
  - Hardware component failure rates
  - Software component failure rates
  - Detection, restart, reboot, repair delays
  - Imperfect coverages for each of the above recovery phases
- The parameter values obtained from
  - Field data for hardware component failure rates
  - High availability testing for detection/restart/reboot delays
  - Agreed upon assumptions for other parameters
- Uncertainty in parameter values (assumed value or based on limited test data)
  - Sensitivity analysis w.r.t. that parameter performed

Parameters for the Hardware Components

Params	Description	Values
$1/\lambda_{mp}$	mean time for mid-plane failure	10^6 hours
$1/\lambda_c$	mean time for blower failure	10^6 hours
$1/\lambda_{ps}$	mean time for power module failure	10^6 hours
$1/\lambda_{cpu}$	mean time for processor failure	10^6 hours
$1/\lambda_{base}$	mean time for Base failure	10^6 hours
$1/\lambda_{OS}$	mean time for OS failure	4000 hours
$1/\lambda_{swh}$	mean time for ethernet switch failure	10^6 hours
$1/\lambda_{nic}$	mean time for NIC failure	10^6 hours
$1/\lambda_{mem}$	mean time for memory DIMM failure	10^6 hours
$1/\lambda_{hd}$	mean time for hard disk failure	10^6 hours
$1/\lambda_{sp}$	mean time for failure detection plus repair person arrival	2 hours
c_{mp}	prob. of mid-plane common mode failure	0.001
c_{ps}	coverage factor for power module failure	0.99
$1/\mu_{mp}$	mean time to repair mid-plane	1 hour
$1/\mu_c$	mean time to repair blower	1 hour
$1/\mu_{2c}$	mean time to repair two blowers	1.5 hours
$1/\mu_{ps}$	mean time to repair power module	1 hour
$1/\mu_{2ps}$	mean time to repair two power modules	1.5 hours
$1/\mu_{cpu}$	mean time to repair processor	1 hour
$1/\mu_{base}$	mean time to repair Base	1 hour
$1/\delta_{OS}$	mean time to detect the OS failure	1 hour
b_{OS}	coverage factor for node reboot to recover OS	0.9
$1/\beta_{OS}$	mean time for node reboot	10 minutes
$1/\mu_{OS}$	mean time to repair OS	1 hour
$1/\mu_{swh}$	mean time to repair the ethernet switch	1 hour
$1/\mu_{nic}$	mean time to repair NIC	1 hour
$1/\mu_{mem}$	mean time to repair memory bank	1 hour
$1/\mu_{hd}$	mean time to repair hard disk	1 hour
$1/\mu_{2hd}$	mean time to repair two hard disks	1.5 hours
$1/\chi_{hd}$	mean time to copy disk data	10 minutes
k	min number of failed app. servers for system unavail.	6

Parameters for the software components

Parameters	Description	Values
$1/\gamma$	mean time to server failure	1000 hours
$1/\delta_1$	mean time for WLM failure detection	2 seconds
$1/\delta_2$	mean time for node agent failure detection	2 seconds
$1/\delta_m$	mean time for manual failure detection	10 minutes
$1/\phi$	mean time for failover	1 second
$1/\rho_a$	mean time for automatic process restart	10 seconds
$1/\rho_m$	mean time for manual process restart	60 seconds
$1/\beta_m$	mean time for manual node reboot	10 minutes
$1/\mu$	mean time for manual repair	8 hours
c	coverage factor for failover	0.9
d	coverage factor for WLM detection	0.9
e	coverage factor for node agent detection	0.9
q	coverage factor for auto process restart	0.9
r	coverage factor for manual process restart	0.9
b	coverage factor for manual node restart	0.9

System and subsystem downtime (min/year)

- Downtime at different levels of AS redundancy (*k-1*)
  - Downtime of individual components

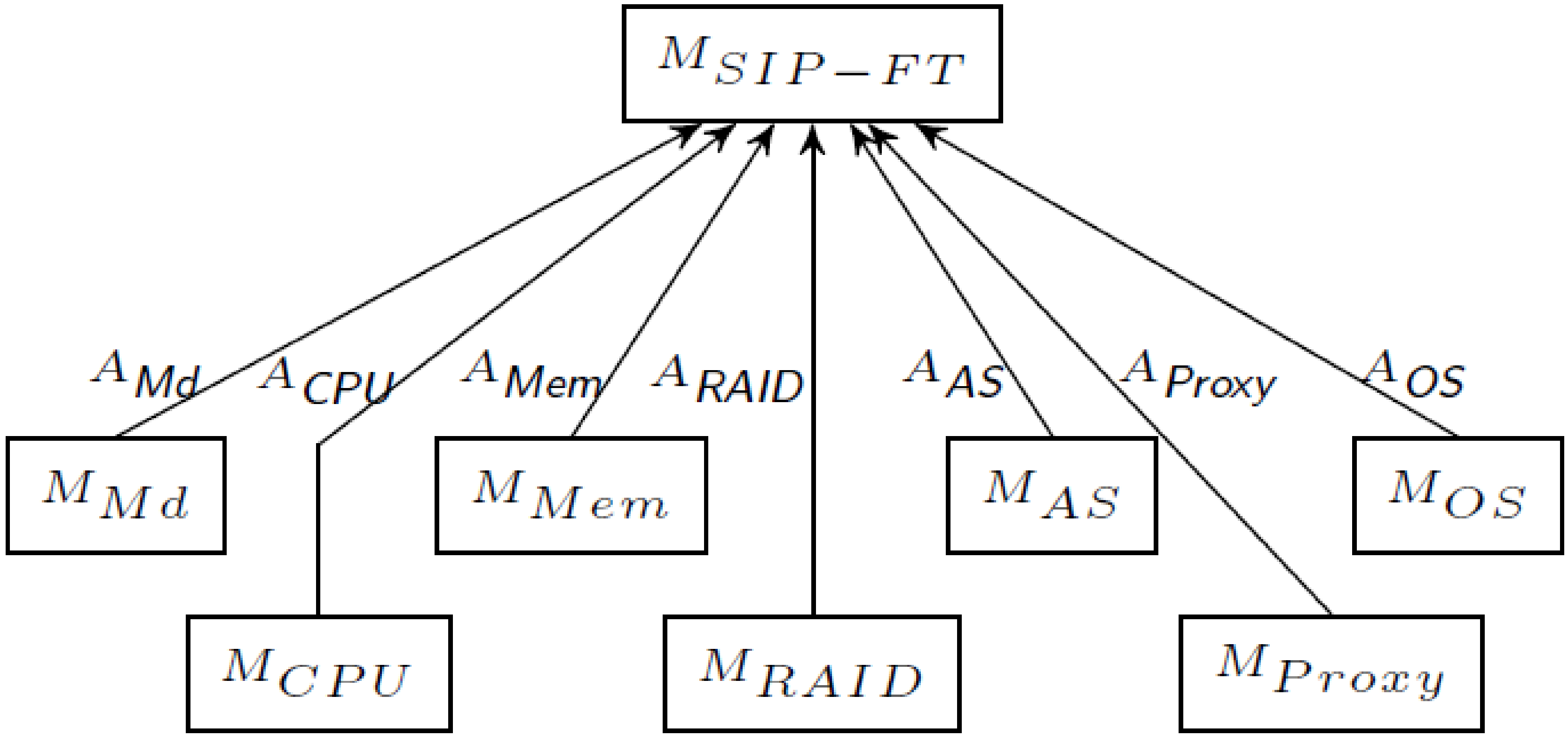
Downtime by various causes

k	OS	hardware	proxy	app server	total downtime
1	1,155	73.65	0.00036	165.7	1,394
2	1,155	73.65	0.00036	0.024	1,228
3	1.13	1.13	0.00036	0.0000005	2.7350
4	1.13	1.13	0.00036	0	2.413
5	0.07	1.13	0.00036	0	1.219
6	0.07	1.13	0.00036	0	1.218
7	0.07	0.00038	0.00036	0	0.093

Availability model of SIP on IBM WebSphere (contributions)

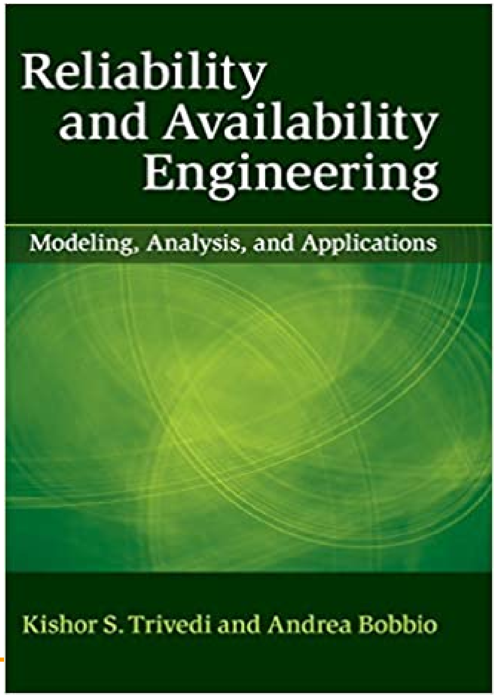
- Developed a very comprehensive availability model
  - Hardware and software failures
  - Hardware and Software failure-detection delays
  - Software Failover delay
  - Escalated levels of recovery
    - Automated and manual restart, reboot, repair
  - Imperfect coverage (detection, failover, restart, reboot)
- Many of the parameters collected from experiments, some obtained from tables; few of them assumed
- Detailed sensitivity analysis to find bottlenecks and give feedback to designers
- Developed a new method for calculating DPM (defects per million)
  - Taking into account interaction between call flow and failure/recovery
  - Retry of messages (this model will be published in the future)
- This model was responsible for the sale of the system by IBM

Import graph for SIP Availability Model



Hierarchical Composition

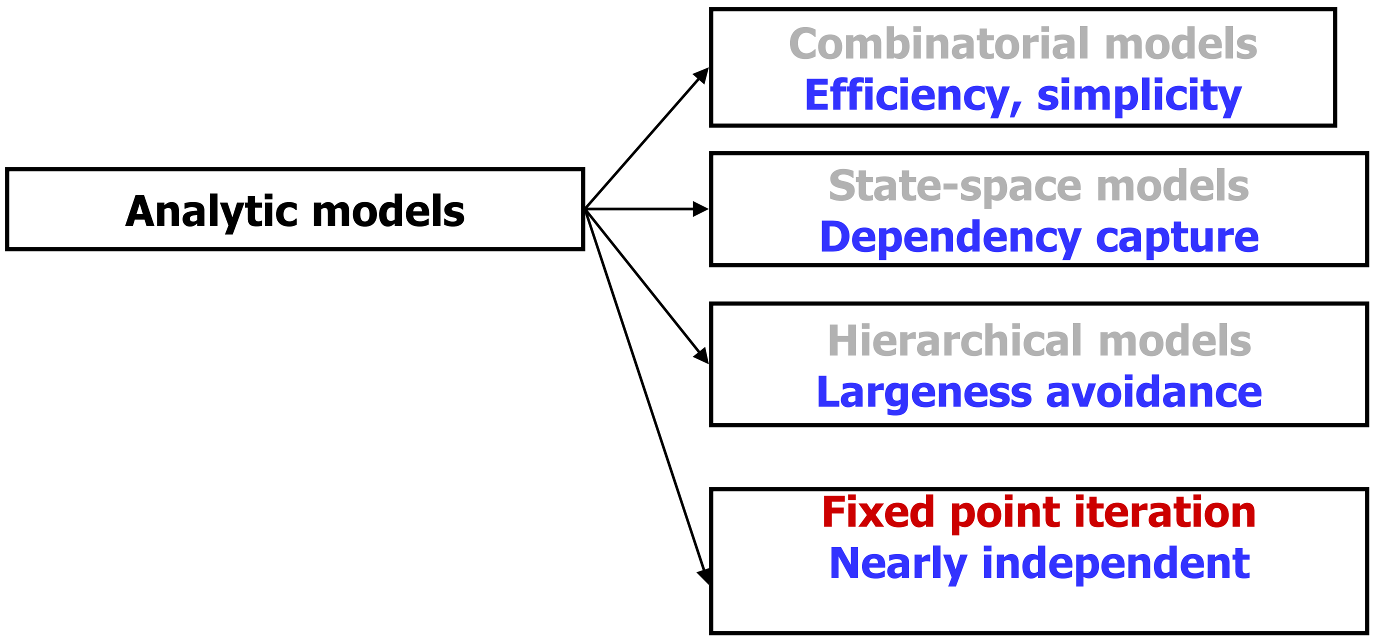
- Many more examples of such models can be found in the book (Trivedi & Bobbio, **Reliability and Availability: Modeling, Analysis, Applications**, Cambridge University Press, 2017) and other papers
  - Availability Models
  - Reliability Models
  - Performance Models
  - Performability Models
  - Survivability Models
  - Dynamic Fault Tree Models



Hierarchical Composition

- Matrix-Level vs. Model-Level vs. System-Level Decomposition
- Multi-level modeling formalism -- meta-modeling language?
- What kinds of quantities to pass between sub-models?
- Exact vs. approximate solution
- If approximate, bounding/estimating errors of approximation?
- Import graph
  - Acyclic
  - Cyclic → Fixed-point iteration

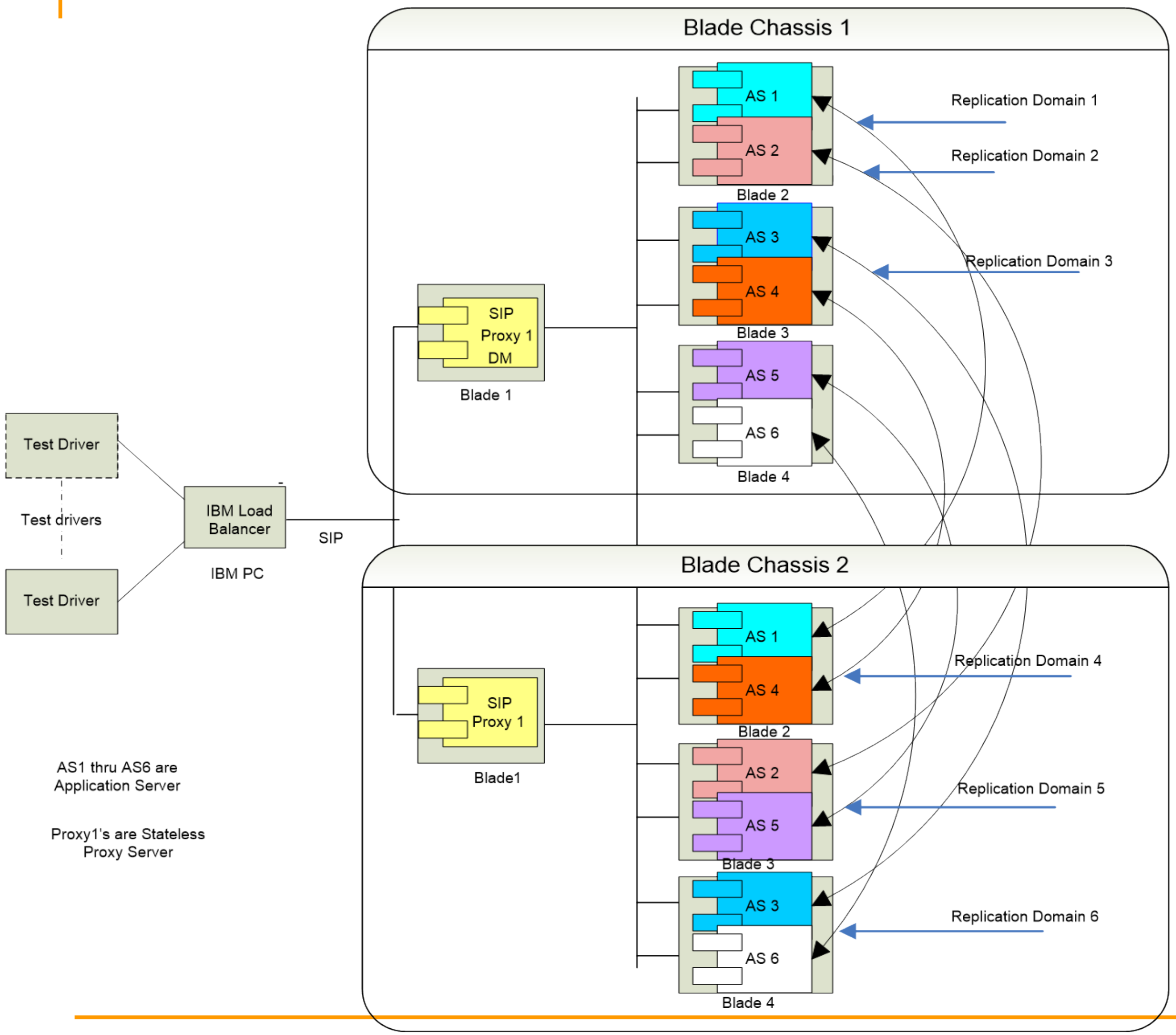
Analytic Modeling Taxonomy



Return to the SIP Availability Model

- We ignored one dependence assuming its effect will be negligible
- Two App servers share a blade server node

Architecture of SIP on IBM WebSphere



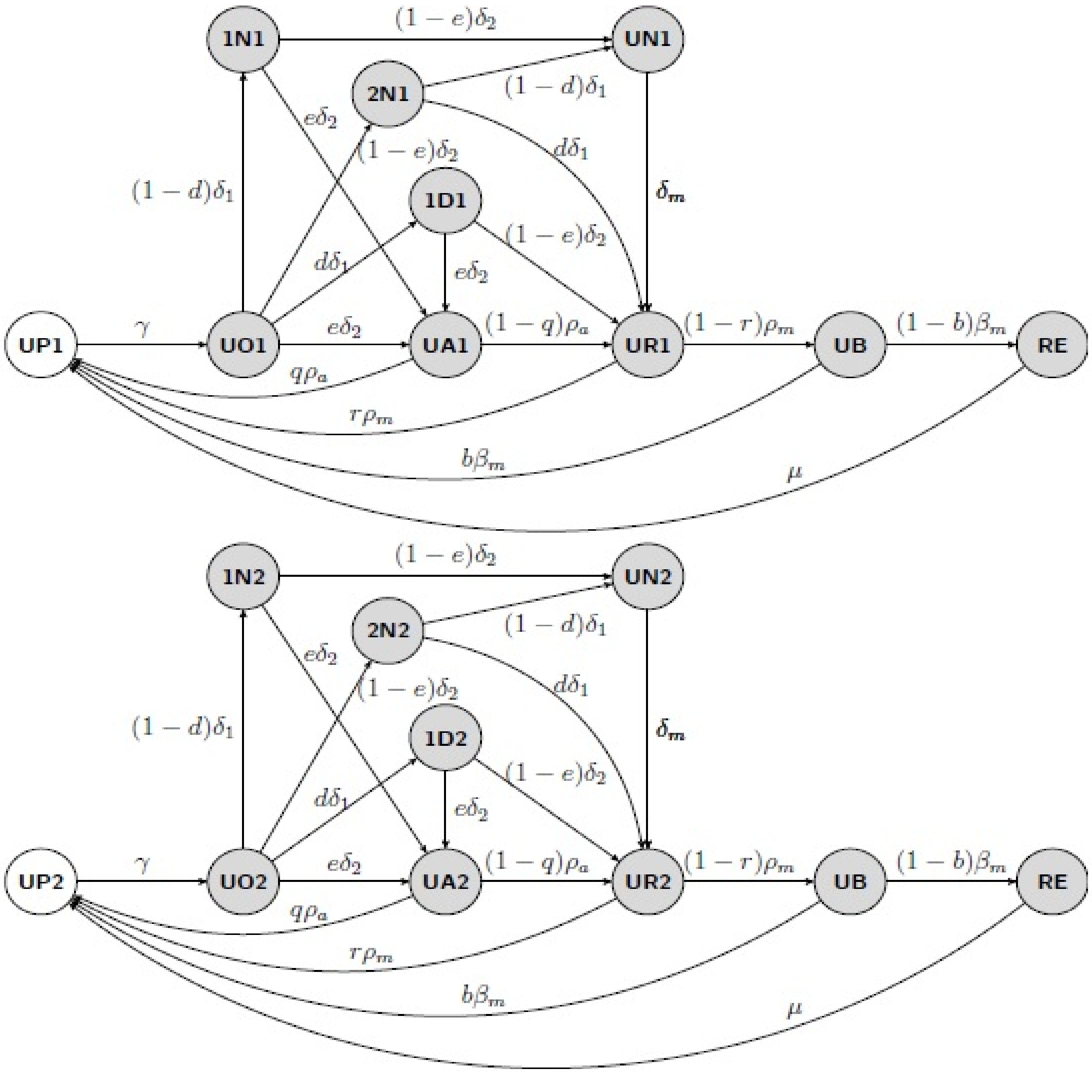
AS: WebSphere Appl. Server (WAS)

Replication domain	Nodes
1	A, D
2	A, E
3	B, F
4	B, D
5	C, E
6	C, F

Return to the SIP Availability Model

- Two App servers share a blade server node
- If one App server needs reboot its OS or repair of the blade is needed, it affects the other app server on the same blade – a forced dependence
- We ignored this dependence earlier
- We now account for this dependence

Two app server CTMCs run nearly independently



Two app server CTMCs run *nearly* independently

But need to be synchronized at state UB
since when one app server needs to be rebooted, the other is forced to be rebooted

Similarly the two need to be synchronized at states RE since when one app server blade needed to be repaired, the other is to repair

Need to combine the two CTMCs

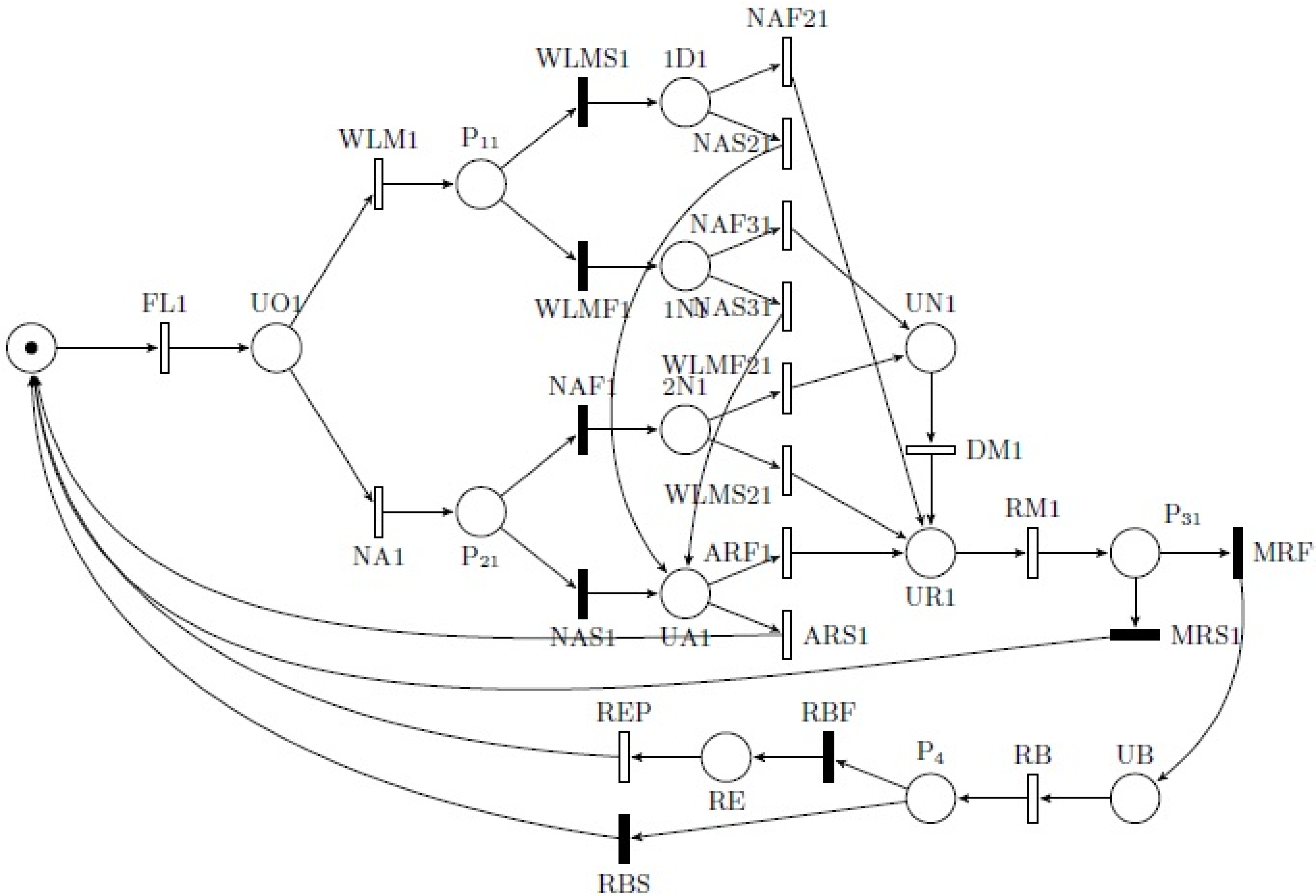
Two Synchronized app servers

- Combined Markov model (CTMC) is too large to construct by hand.
- We use a high-level formalism of stochastic Petri net (the flavor known as stochastic reward net (SRN)).
- SRNs extend other SPN formalisms by adding variable cardinality arcs, transition priorities, guard functions and the ability to specify reward rates at the net level
- SRN models can be automatically converted into underlying Markov (reward) model and solved for the measures of interest such as DT (downtime) and many more

In order to model the synchronization

- We use an SRN model to show two synchronized CTMCs and solve the SRN model using SHARPE software package
- We start by first converting single app server CTMC to an SRN

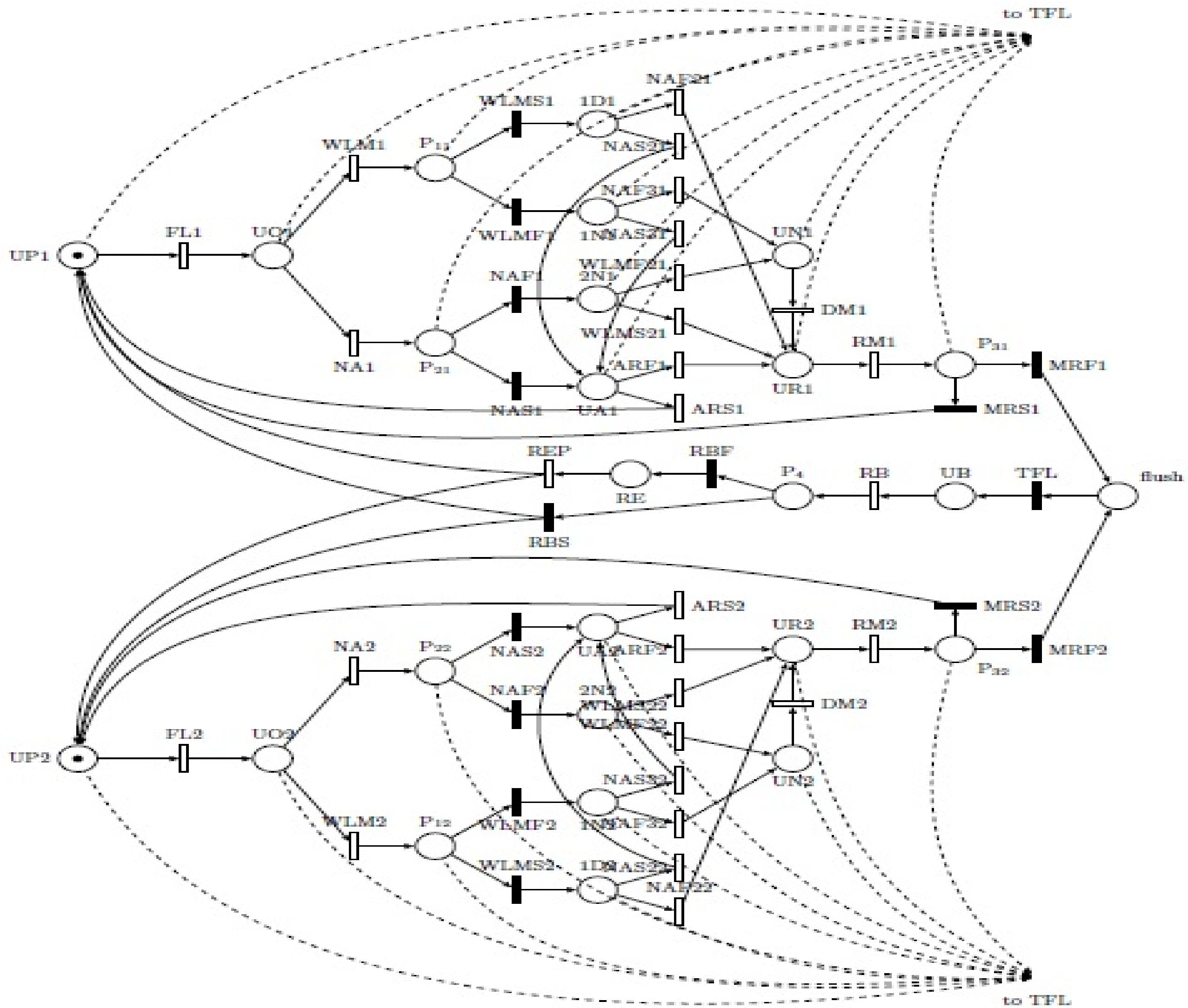
SRN Availability Model of a single app server



In order to model the synchronization

- We use an SRN model to show two synchronized CTMCs and solve the SRN model using SHARPE
- Dotted arcs are variable cardinality arcs that flush the places of any token they may have

SRN Availability Model of two synchronized app servers



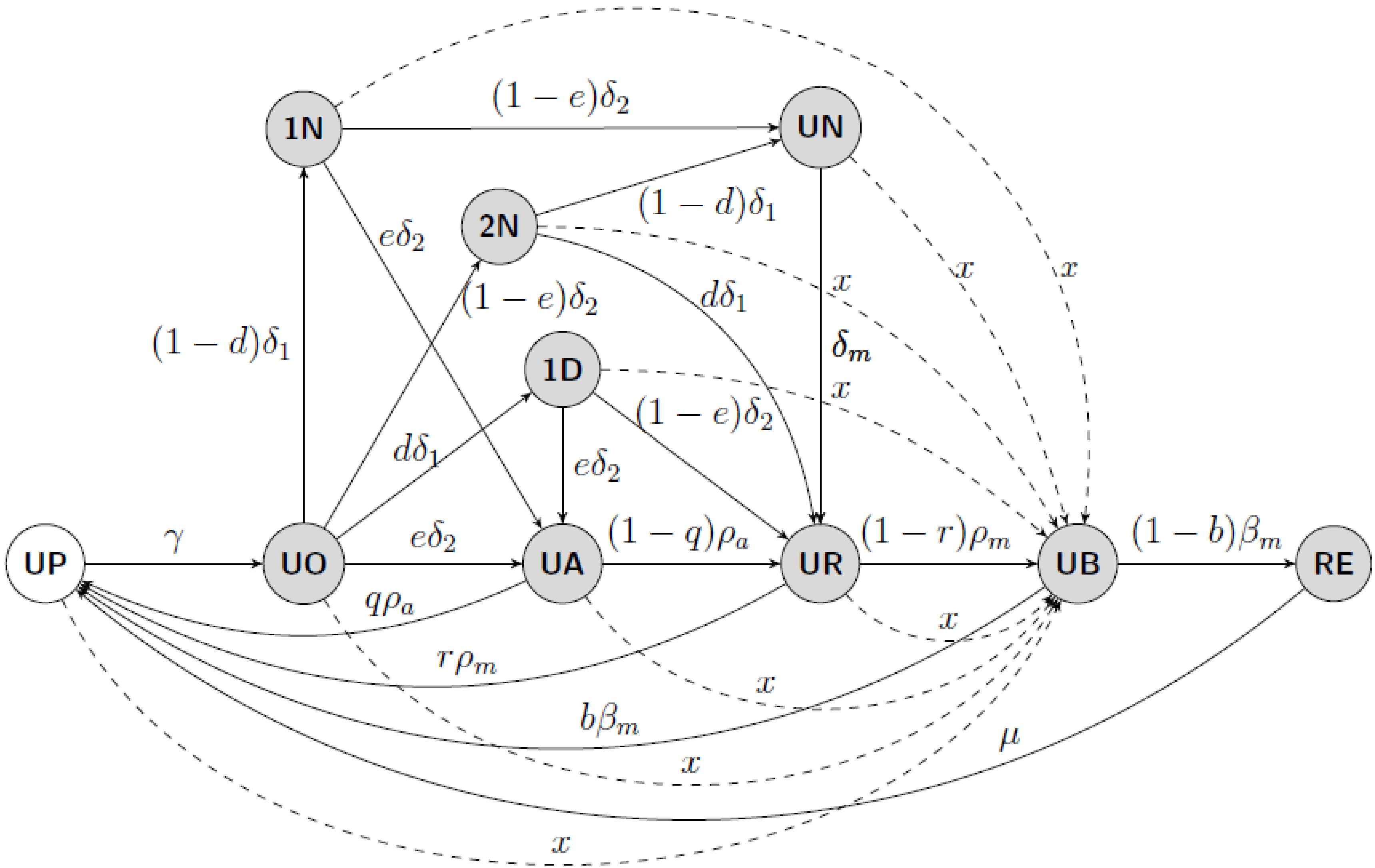
Two synchronized app servers model

- We used an SRN model to capture two synchronized CTMCs and solve the SRN model using SHARPE
- Underlying reachability graph has 65 vanishing markings and 66 tangible markings – so the CTMC generated from this SRN has 66 states
- Note that if two CTMCs were independent then the combined composed CTMC will have the cross-product state space with 10*10=100 states
- But 8 out of 10 states of each CTMC are independent while 2 states are common or shared states. Hence the resulting number of states is 8*8+2=66

CTMC approximation

- Then we develop a simple approximate CTMC in which we have added a transition (shown as dotted arcs) from each of the states, UP, UO, 1N, 2N, 1D, UA, UN and UR to state UB at rate x as shown in the next slide
- Now each app server CTMC model can be considered independent for the overall SIP availability model

Modified CTMC to account for forced reboot



Fixed-Point Iteration

- Since x is an input parameter for the CTMC, π_{UR} is a function of x, thus, we have a fixed-point problem:
- Rate $x = \pi_{UR}(x)*(1-r)*\varrho_m$
- We initialize x so that $x_0 = 0.0001*(1-r)*\varrho_m$
- We solve iteratively using successive substitution:
- $x_{i+1} = \pi_{UR}(x_i)*(1-r)*\varrho_m$

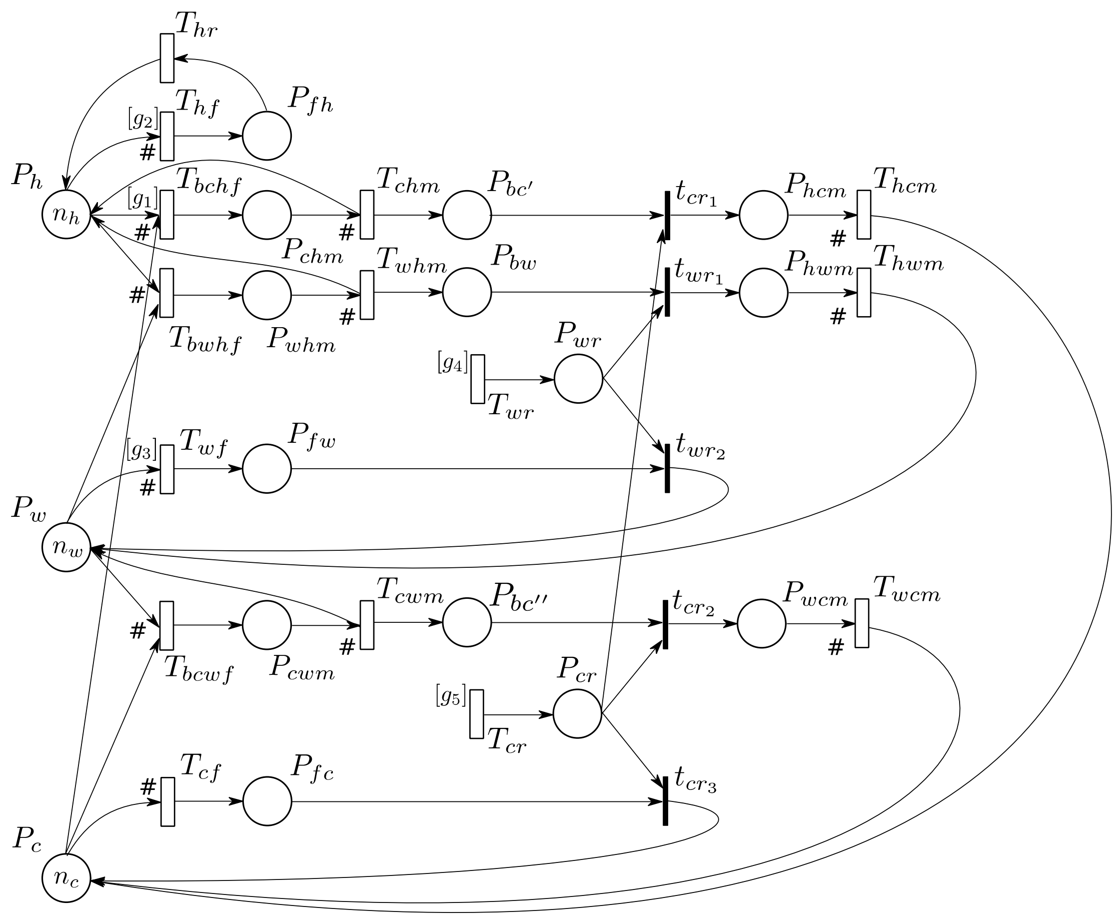
Fixed-Point Iteration

- It took only 3 iterations to converge to a fixed point
- App server steady state availability computed with the exact composed CTMC and with the fixed-point iteration approximation are both 0.999844145
- The effect of this dependency is negligible as the steady state availability of the app server without the dependence is 0.999845429 while with the dependence it is 0.999844145

Scalable Model for IaaS Cloud Availability and Downtime

Ref: Ghosh, Longo, Frattini, Russo, Trivedi, "Scalable Analytics for IaaS Cloud Availability," IEEE Trans. Cloud Comput., 2014

Monolithic SRN Model



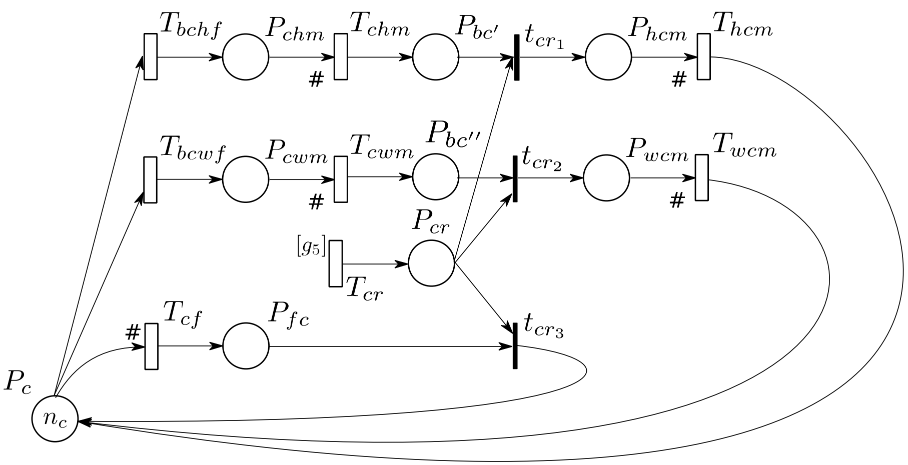
Guard functions	**Values**
$[g_1]$	1 if $\#P_w = 0$ 0 otherwise
$[g_2]$	1 if $\#P_w = 0$ and $\#P_c = 0$ 0 otherwise
$[g_3]$	1 if $\#P_c = 0$ 0 otherwise
$[g_4]$	1 if $\#P_{fw} + \#P_{bw} > 0$ 0 otherwise
$[g_5]$	1 if $\#P_{fc} + \#P_{bc'} + \#P_{bc''} > 0$ 0 otherwise

Monolithic Model

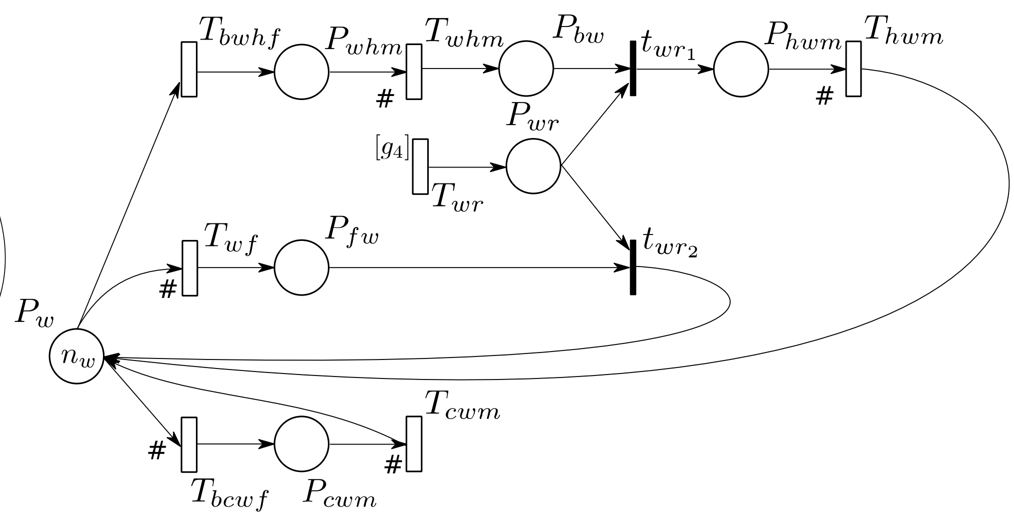
- Monolithic SRN model is automatically translated into CTMC or Markov Reward Model
- However the model not scalable as state-space size of this model is extremely large

#PMs per pool	#states	#non-zero matrix entries
3	10, 272	59, 560
4	67,075	453, 970
5	334,948	2, 526, 920
6	1,371,436	11, 220, 964
7	4,816,252	41, 980, 324
8	Memory overflow	Memory overflow
10	-	-

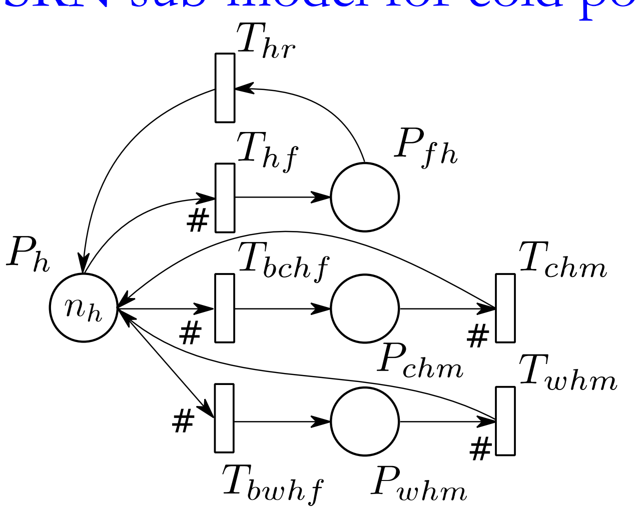
Decompose into Interacting Sub-models



SRN sub-model for cold pool

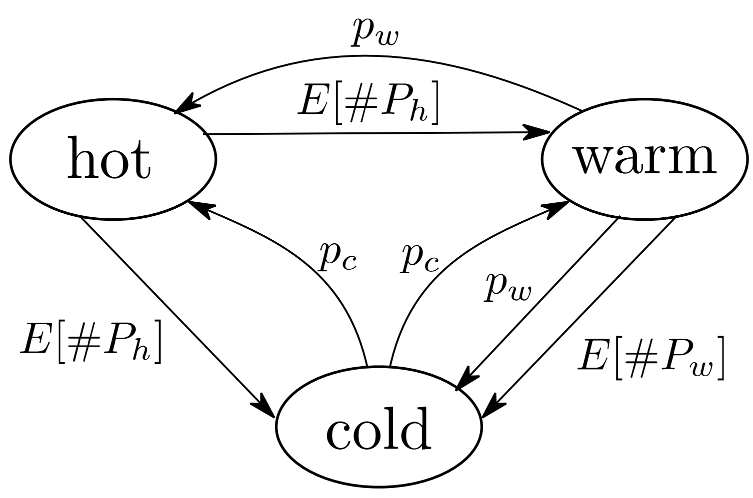


SRN sub-model for warm pool



SRN sub-model for hot pool

Import graph and model outputs



- **Model outputs:**
  - mean number of PMs in each pool ($E[\#P_h]$, $E[\#P_w]$, and $E[\#P_c]$)
  - Downtime in minutes per year

Many questions

- Existence of Fixed Point (easy): IEEE TCC 2014 (In a more general setting: Mainkar & Trivedi paper in IEEE-TSE, 1996)
- Uniqueness (some cases)
- Rate of convergence
- Accuracy
- Scalability

Monolithic vs. interacting sub-models

- #states, #non-zero entries

n	Monolithic model		Interacting sub-models	
	#states	#non-zero entries	#states	#non-zero entries
3	$10,272$	$59,560$	196	588
4	$67,075$	$453,970$	491	$1,768$
5	$334,948$	$2,526,920$	$1,100$	$4,518$
6	$1,371,436$	$11,220,964$	$2,262$	$10,272$
7	$4,816,252$	$41,980,324$	$3,770$	$18,434$
8	Memory overflow	Memory overflow	$6,939$	$36,316$
10	-	-	$20,460$	$118,710$
20	-	-	$21,273$	$106,300$
40	-	-	$271,543$	$1,481,000$
60	-	-	$1,270,813$	$7,148,100$
80	-	-	$3,859,083$	$22,051,600$
100	-	-	$9,196,353$	$53,055,500$

Steps for system availability modeling

- List all possible component level failures (hardware, software)
- List of all failure detectors & match with failure types
- List all recovery mechanisms & match with failure types
- Allocation of software modules to hardware units
- Formulate the model
- Face validation and verification of the model
- Parameterization of the model (tables, websites, experiments)
- Solve the model (using SHARPE, SPNP or similar software packages) to detect bottlenecks, sensitivity analysis, suggest parameters to be monitored more accurately
- What-if analysis to suggest improvements
- Validate the model

Outline

- Introduction and Motivation
- Reliability and Availability Models
- Conclusions
- References

System Reliability/Availability Models

- Techniques & software packages are available for the construction & solution of reliability and availability models of real systems
- System decomposition followed by hierarchical model composition is the typical approach
- Modeling has been used
  - To compare alternative designs/architectures (Cisco)
  - Find bottlenecks, answer what if questions, design optimization and conduct trade-off studies
  - At certification time (Boeing)
  - At design verification/testing time (IBM)
  - Configuration selection phase (DEC)
  - Operational phase for system tuning/on-line control

System Reliability/Availability Models

- Model Types in Use
  - Non-state-Space: Reliability Block Diagram, Fault tree, Reliability graph
  - State-space: Markov models & stochastic Petri nets, Semi-Markov, Markov regenerative and non-homogeneous Markov models
  - Hierarchical composition
    - Top level is usually an RBD or a fault tree
    - Bottom level models are usually Markov chains
  - Fixed-point iterative
- Solution types
  - Analytic closed-form
  - Analytic numerical (using a software package)
  - Simulative
- Software packages
  - SHARPE or similar tools are used to construct and solve such models
- Structural as well as parametric assumptions means that numbers produced should be taken with a grain of salt

Challenges in Reliability/Availability Models

- Model Largeness (in spite of: hierarchy, fixed-point iteration, approximations) – Smartgrid models
- Dealing with non-exponential distributions (in spite of SMP, MRGP, NHCTMC, PH)
- Service (or user)-oriented measures as opposed to system-oriented measures
- Combining performance, power and failure/repair
  - Performability, two-level models, use of Markov-reward models
- Model Parameterization
- Model Validation and Verification
- Parametric uncertainty propagation

Challenges in Reliability/Availability Models

- Model Verification and Validation
  - Verification
    - checked by someone else,
    - check logical flows,
    - cross-check using alternative solutions (e.g. alternative analytic/simulation)
  - Validation
    - Face validation,
    - Input-Output validation,
    - Validation of model assumptions

Model Parameterization

- Hardware/Software Configuration parameters
- Hardware component MTTFs
- Software component MTTFs
  - OS, IBM Application, customer software, third party
- Hardware/Software Failover times
- Restart/Reboot times
- Coverage (Success) probabilities
  - Detection, location, restart, reconfiguration, repair
- Repair time
  - Hot swap, multiple component at once, DOA (dead on arrival), shared/not shared, field service travel time, preventive vs. corrective
- Uncertainty propagation: Dealing with not only Aleatory (built into the system models) but also epistemic (parametric) uncertainty

Message to Young Researchers

- Pick a real problem rather than one from literature whenever possible
  - There should be plenty of real problems in Industry
  - Keep an open mind
  - Ask questions and Listen carefully
- It is possible to write scholarly articles based on work done on real problems
- Use software packages [e.g., SHARPE, SPNP] whenever applicable [as opposed writing your own code to generate and solve models]

Outline of the book: *Reliability and Availability Engineering*

Part I – Introduction (Chapters 1:3)

Part II - Non-state-space models (Chapters 4:8)

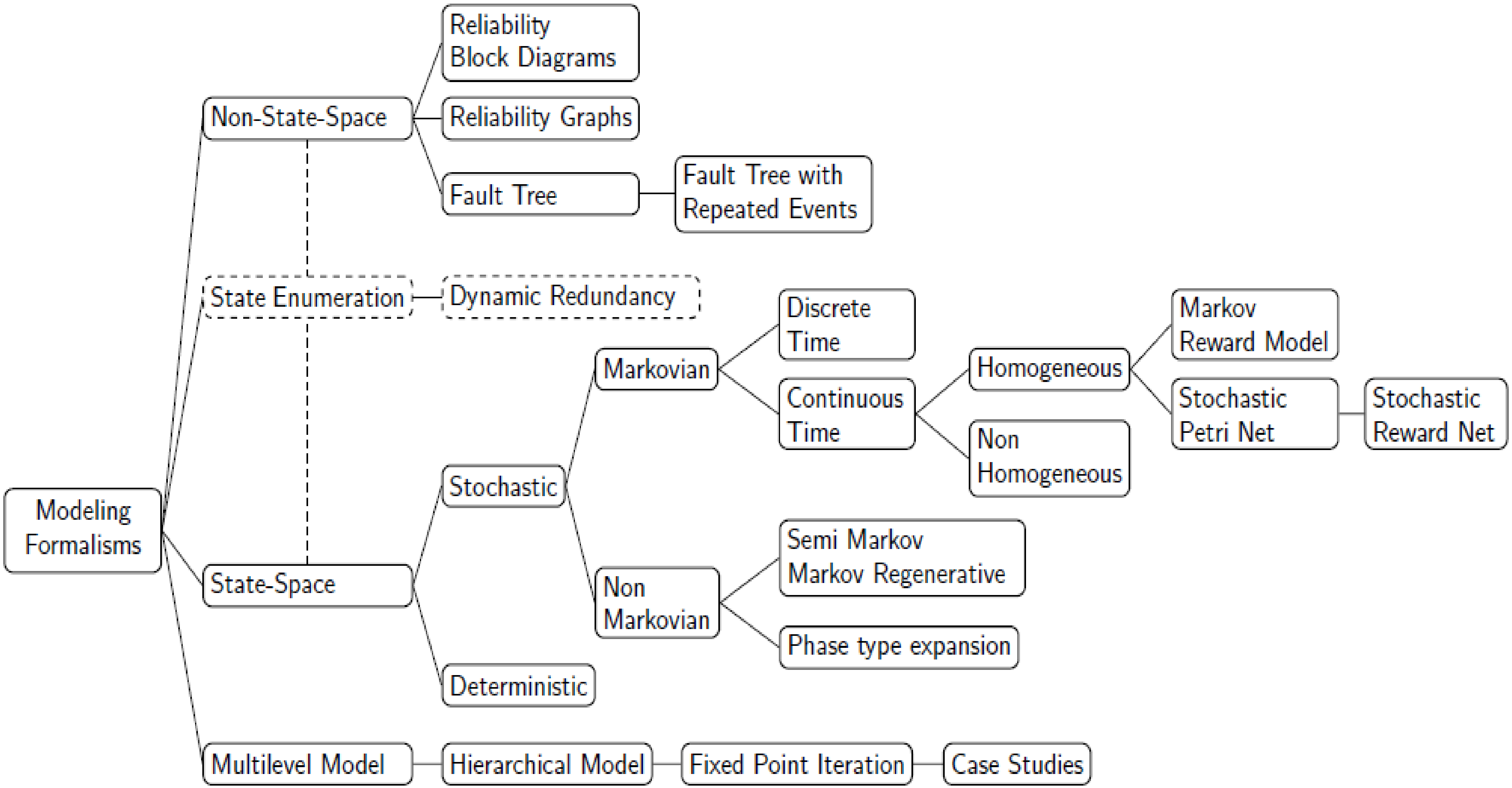
Part III - State-space Models with Exponential Distributions (Chapters 9:12)

Part IV - State-space Models with Non-Exponential Distributions (Chapters 13:15)

Part V - Multi-Level Models (Chapters 16:17)

Part VI - Case Studies (Chapter 18)

Outline of the book: *Reliability and Availability Engineering*



Outline

- Introduction and Motivation
- Reliability and Availability Models
- Conclusions
- References

Selected References

- Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications. John Wiley, 2nd edition, 2001; revised paperback, 2016
- Tomek & Trivedi, Fixed-Point Iteration in Availability Modeling, Informatik-Fachberichte, Dal Cin (ed.), Springer-Verlag, Berlin, 1991
- Mainkar & Trivedi, Sufficient Conditions for Existence of a Fixed Point in Stochastic Reward Net-Based Iterative Models, IEEE TSE, 1996
- Trivedi, Vasireddy, Trindade, Nathan, Castro, “Modeling High Availability Systems,” *PRDC* 2006
- Trivedi & Bobbio, Reliability and Availability: Modeling, Analysis, Applications, Cambridge University Press, 2017
- Trivedi, Wang, Hunt, Rindos, Smith, Vashaw, “Availability Modeling of SIP Protocol on IBM WebSphere,” *PRDC* 2008
- Smith, Trivedi, Tomek, Ackaret, Availability analysis of blade server systems, *IBM Sys. J.*, 2008.
- Trivedi & Sahner, SHARPE at the Age of Twenty two, *ACM SIGMETRICS, Performance Evaluation Review*, 2008
- Trivedi, Wang & Hunt. “Computing the number of calls dropped due to failures,” *ISSRE2010*
- Mishra, Trivedi & Some. "Uncertainty Analysis of the Remote Exploration and Experimentation System*", AIAA Journal of Spacecraft and Rockets*, 2012
- Ghosh, Longo, Frattini, Russo & Trivedi, “Scalable Analytics for IaaS Cloud Availability”, *IEEE Trans. on Cloud Computing*, 2014
- Malhotra, Trivedi, “Power-Hierarchy of Dependability -Model Types,” *IEEE-TR*, 1994

Thank you!

Contact Information and more sources

Kishor Trivedi: ktrivedi@duke.edu

www.researchgate.net/profile/Kishor_Trivedi2

Andrea Bobbio: andrea.bobbio@uniupo.it