Politecnico
di Milano

Analyzing architectures: a quantitative approach

Availability analysis slides credits by Danilo Ardagna

Analysis of architectures

- A number of QoS dimensions of the resulting system are directly influenced by the architectural style of choice
  - Scalability
  - Reliability
  - Availability
  - Usability
  - …
- Specific methodologies to analyze these aspects exist

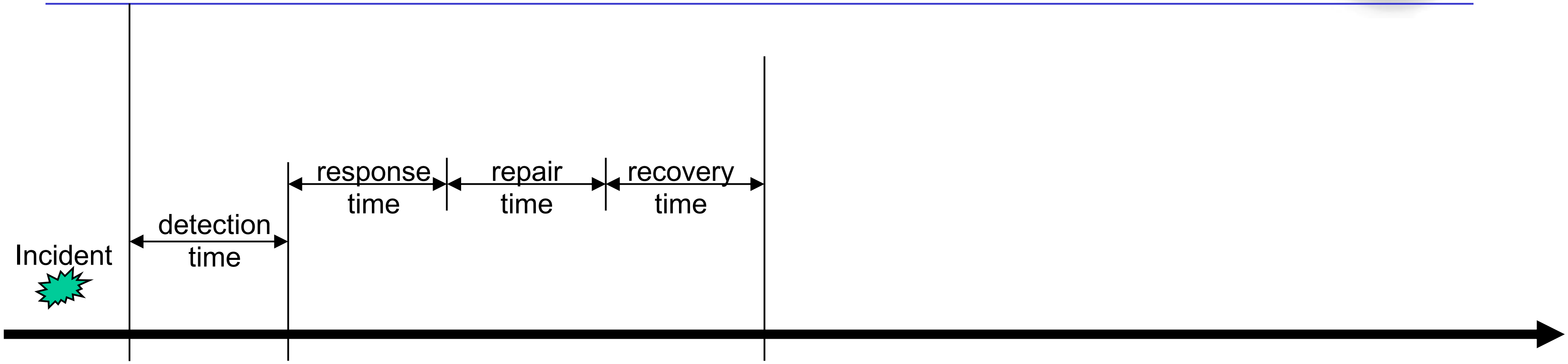
Availability

- A service shall be **continuously available** to the user
  - Little downtime and rapid service recovery
- The availability of a service depends on:
  - Complexity of the IT infrastructure architecture
  - Reliability of the individual components
  - Ability to respond quickly and effectively to faults
  - Quality of the maintenance by support organizations and suppliers
  - Quality and scope of the operational management processes

Reliability

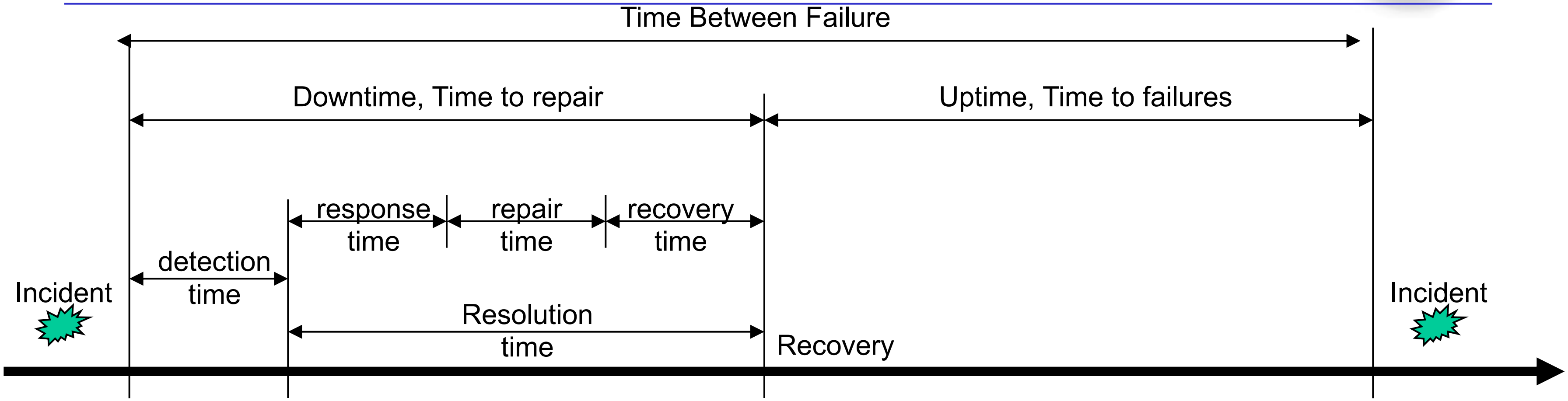
- **Adequate reliability** means that the service is available for an agreed period without interruptions
- The reliability of a service increases if downtime can be prevented
- Reliability is determined by:
  - Reliability of the components used to provide the service
  - Ability of a service or component to operate effectively despite failure of one or more subsystems
  - Preventive maintenance to prevent downtime

System life-cycle



- **Time of occurrence:** Time at which the user becomes aware of the fault
- **Detection time:** The service provider is informed of the fault
- **Response time:** Time required by the service provider (diagnosis) to respond to the user
- **Repair time:** Time required to restore the service or the components that caused the fault
- **Recovery time:** Time required to restore the system (re-configuration, re-initialization,…)

System life-cycle

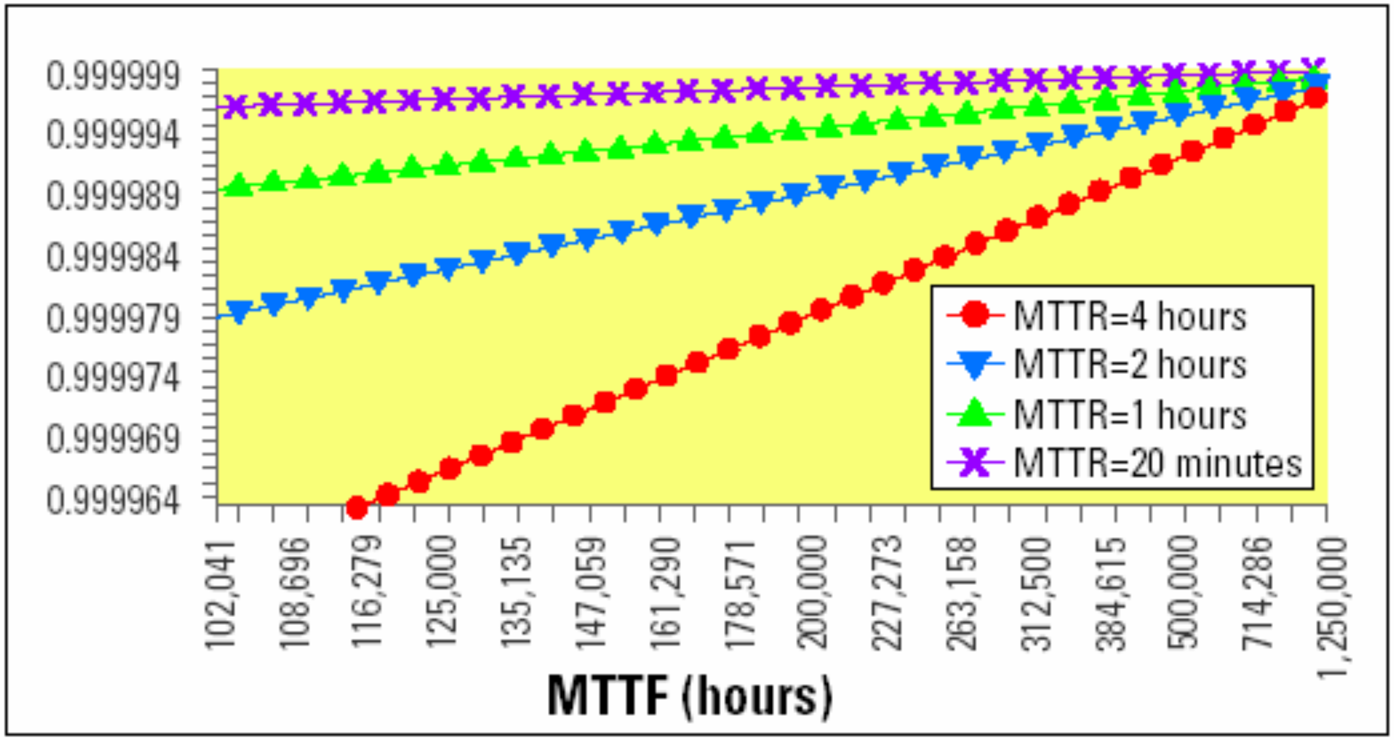


- **Mean Time to Repair (MTTR):** Average time between the occurrence of a fault and service recovery, also known as the downtime
- **Mean Time To Failures (MTTF):** Mean time between the recovery from one incident and the occurrence of the next incident, also known as uptime
- **Mean Time Between Failures (MTBF):** Mean time between the occurrences of two consecutive incidents

Availability vs. Reliablity

- **Availability**: The probability that a component is working properly at time *t*
  - $A = MTTF / (MTTF+MTTR)$
- **Reliability**: The probability that a component has always been working properly during a time interval (0,*t*)
  - $R = e^{-\lambda t}$ $\lambda = 1/MTTF$
- Reliability requires that the component never fails in the interval (0,*t*)
- From the availability perspective a given component could have failed in the interval (0,*t*), but it could have been repaired before t

Availability, MTTF, MTTR



MTBF = MTTF+MTTR (if MTTR small, MTBF ≅ MTTF)

Nines notation

- Availability is typically specified in nines notation
- For example 3-nines availability corresponds to 99.9%, 5-nines availability corresponds to 99.999% availability

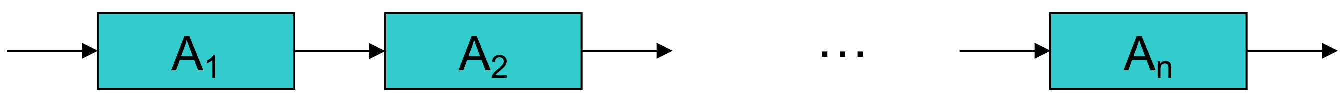
Availability	Downtime
90% (1-nine)	36.5 days/year
99% (2-nines)	3.65 days/year
99.9% (3-nines)	8.76 hours/year
99.99% (4-nines)	52 minutes/year
99.999% (5-nines)	5 minutes/year

Availability

- Calculated by modeling the system as an interconnection of parts in series and parallel
- If failure of a part leads to the combination becoming inoperable, the two parts are considered to be operating in **series**
- If failure of a part leads to the other part taking over the operations of the failed part, the two parts are considered to be operating in **parallel**

Availability in series

- The combined system is operational only if every part is available
- The combined availability is the **product** of the availability of the component parts



$$A = \prod_{i=1}^{n} A_i$$

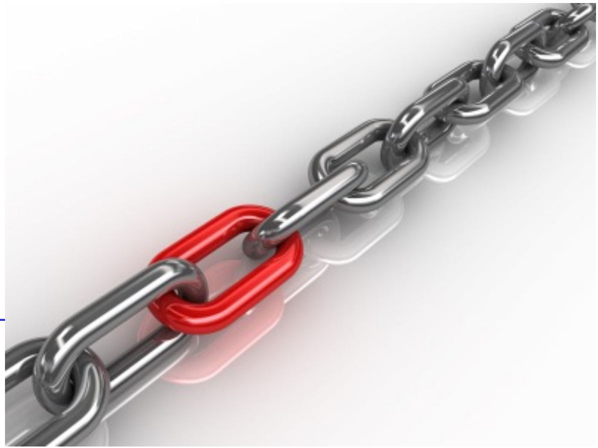
Availability in series – A numerical example

	Availability	Downtime
Component 1	99% (2-nines)	3.65 days/year
Component 2	99.999% (5-nines)	5 minutes/year
Combined	98.999%	3.65 days/year

Downtime=(1-A)*365 days/year

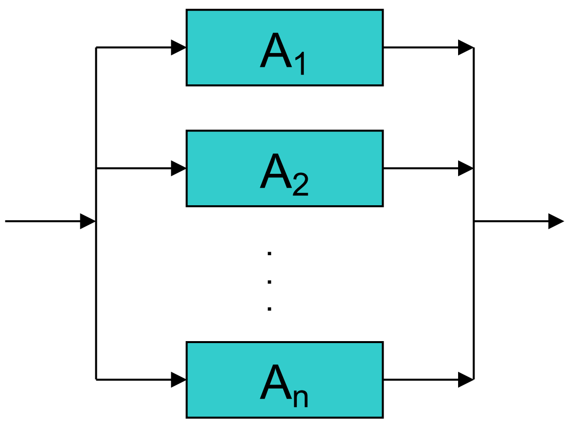
The availability of the entire system is negatively affected by the low availability of Component 1

A chain is as strong as the weakest link!



Availability in parallel

- The combined system is operational if at least one part is available
- The combined availability is **1 - (all parts are unavailable)**



$$A = 1 - \prod_{i=1}^{n} (1 - A_i)$$

Availability in parallel – A numerical example

	Availability	Downtime
Component 1	99% (2-nines)	3.65 days/year
Component 2	99% (2-nines)	3.65 days/year
Combined	99.99% (4-nines)	52 minutes/year

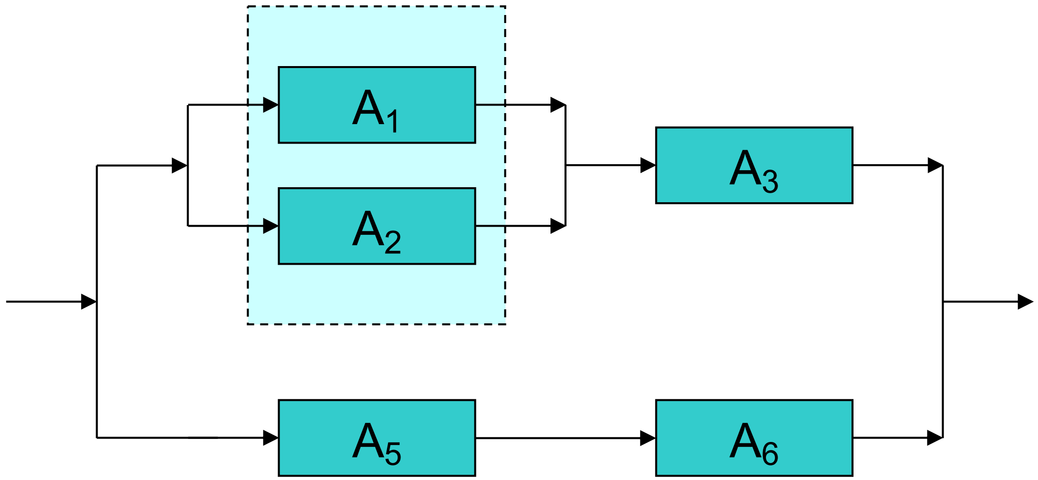
Downtime=(1-A)*365 days/year

Even though components with very low availability are used, the overall availability of the system is much higher

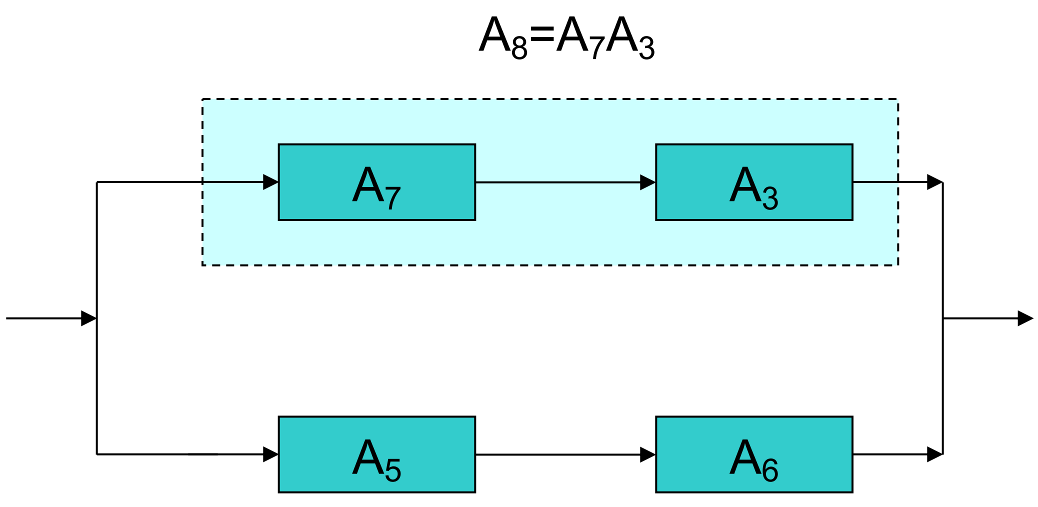
Mission critical systems are designed with redundant components!

Availability of complex systems

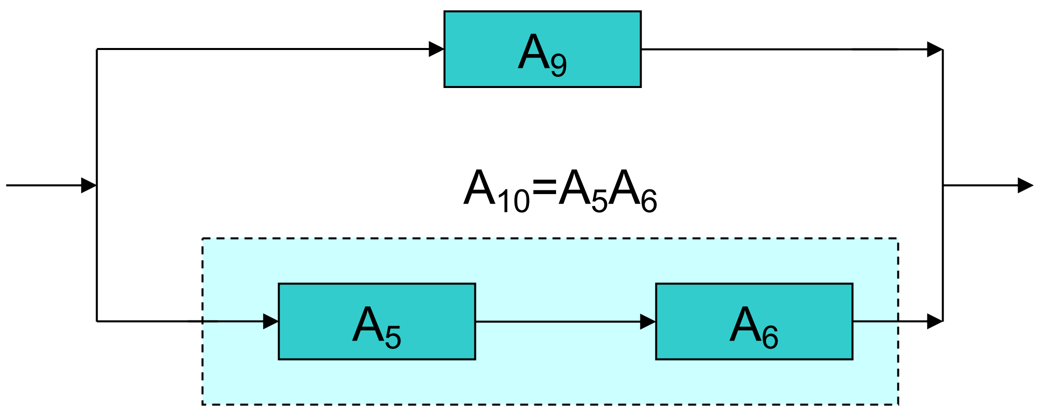
$A_7 = 1-(1-A_1)(1-A_2)$



Availability of complex systems

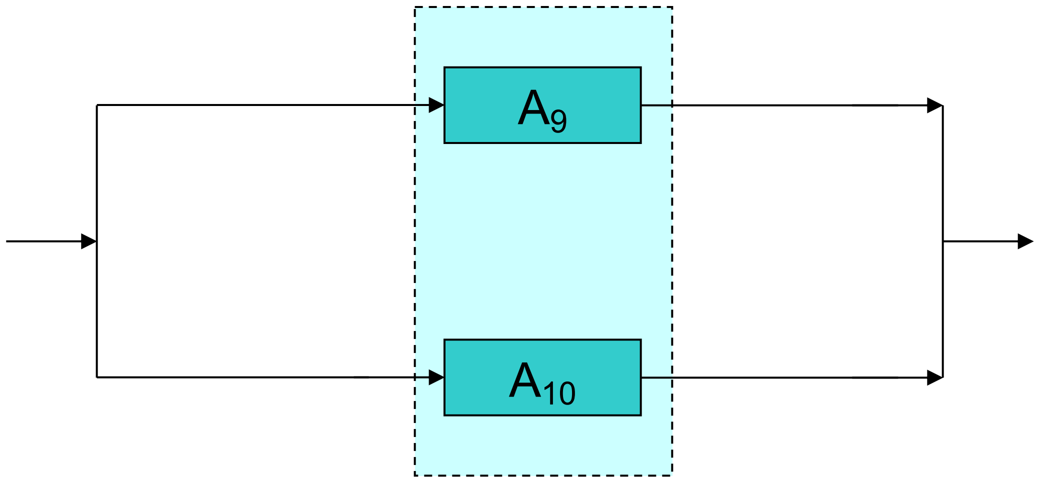


Availability of complex systems



Availability of complex systems

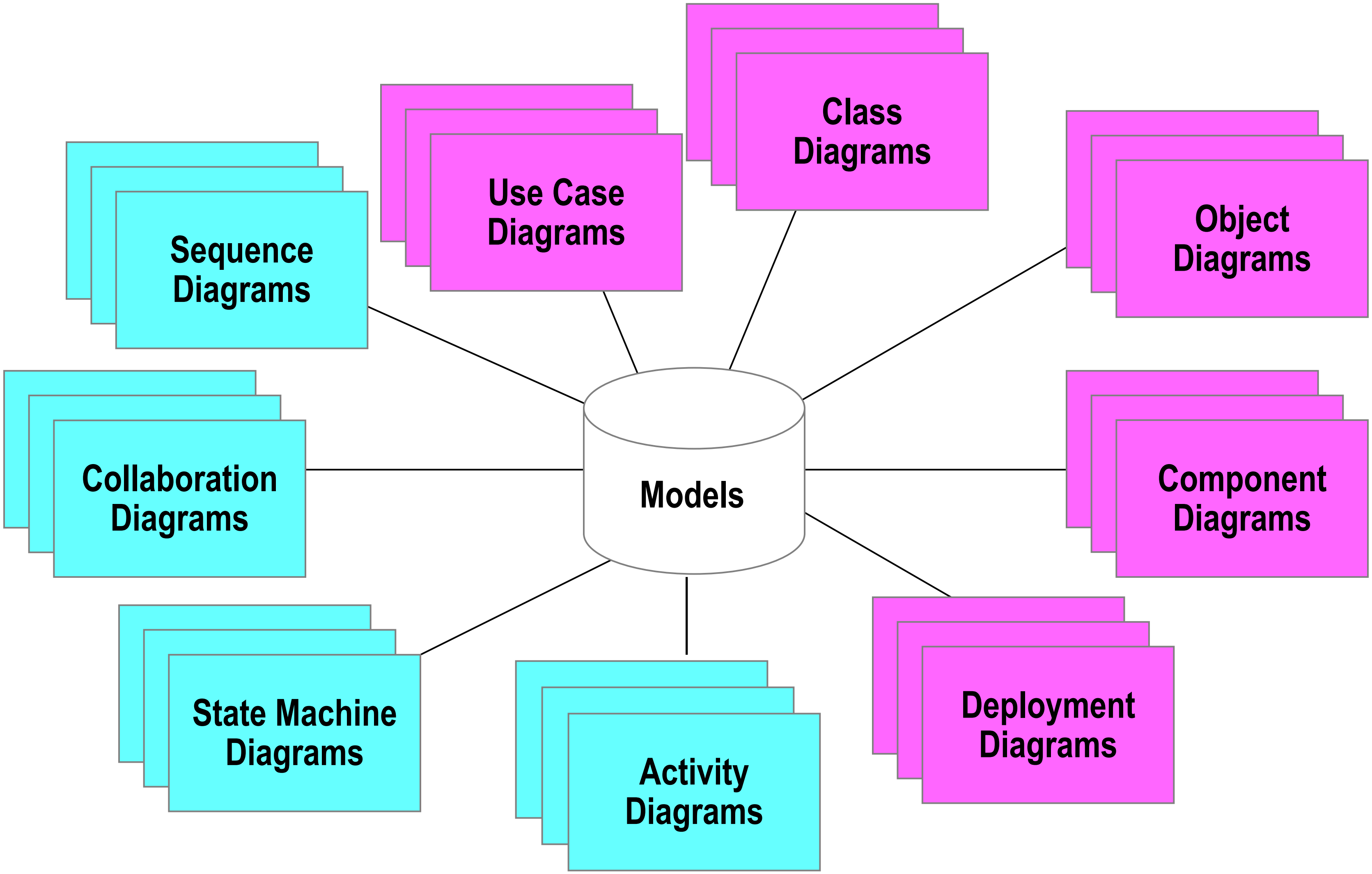
$$A=1-(1-A_9)(1-A_{10})$$



Politecnico
di Milano

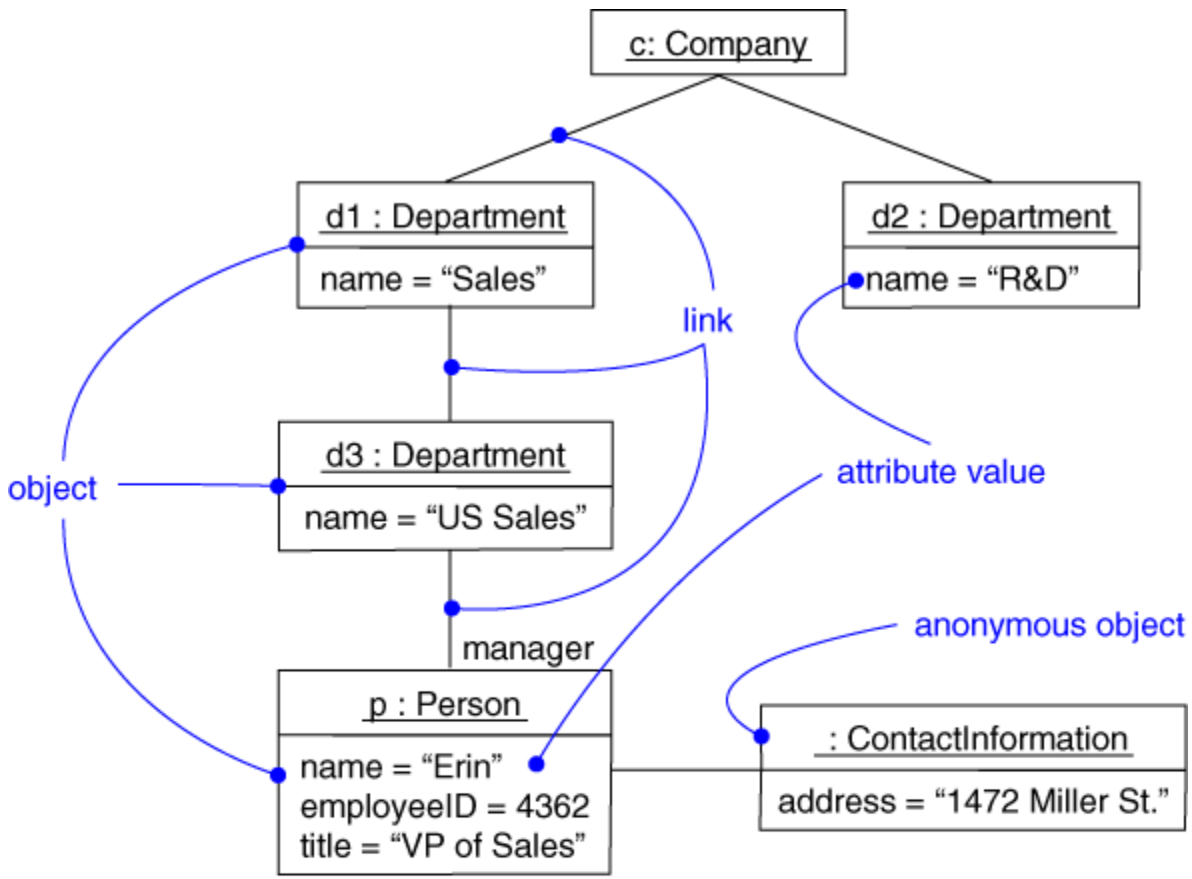
Software design descriptions and UML

UML Models, Views, and Diagrams



“Object” Diagram
(in UML 2.5 they are really Class Diagrams with only instances)

- Captures instances and links

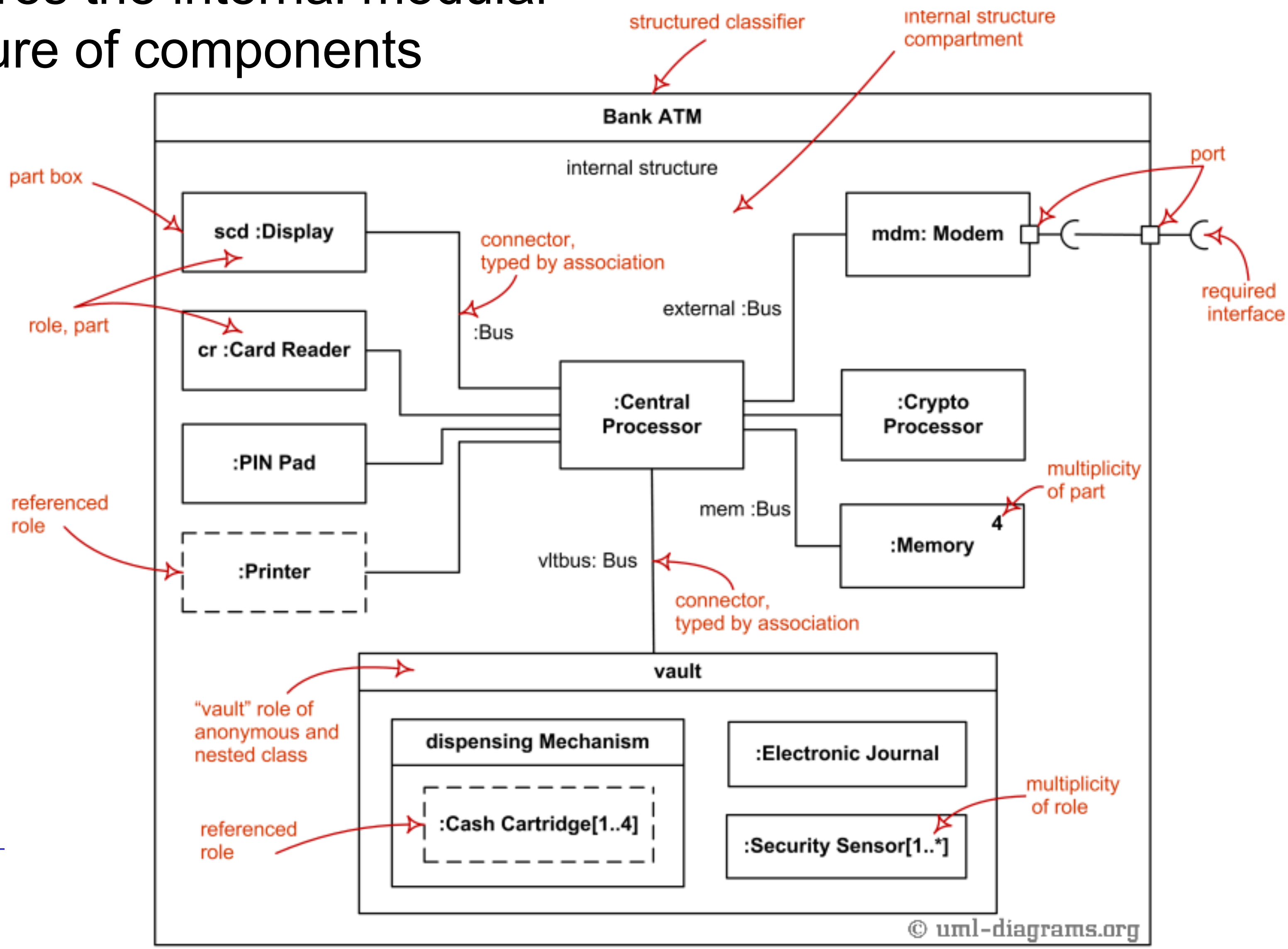


Object Diagram

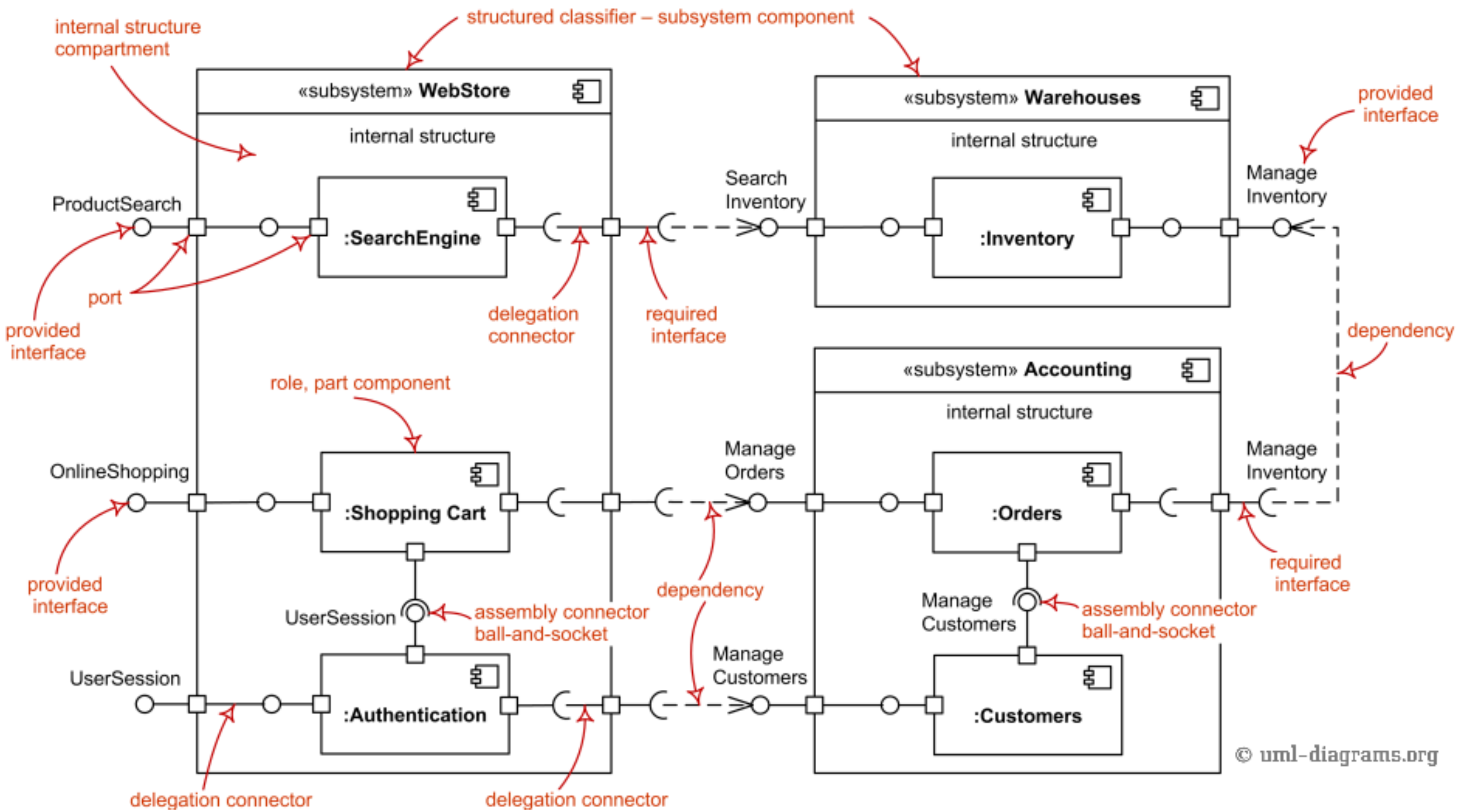
- Shows instances and links that architecture elements maintain across functions
- Built during analysis and design
- Purpose
  - Illustrate data/object structures
  - Specify architecture runtime snapshots
- Developed by analysts, designers, and implementers

Composite Structure Diagram

- Captures the internal modular structure of components



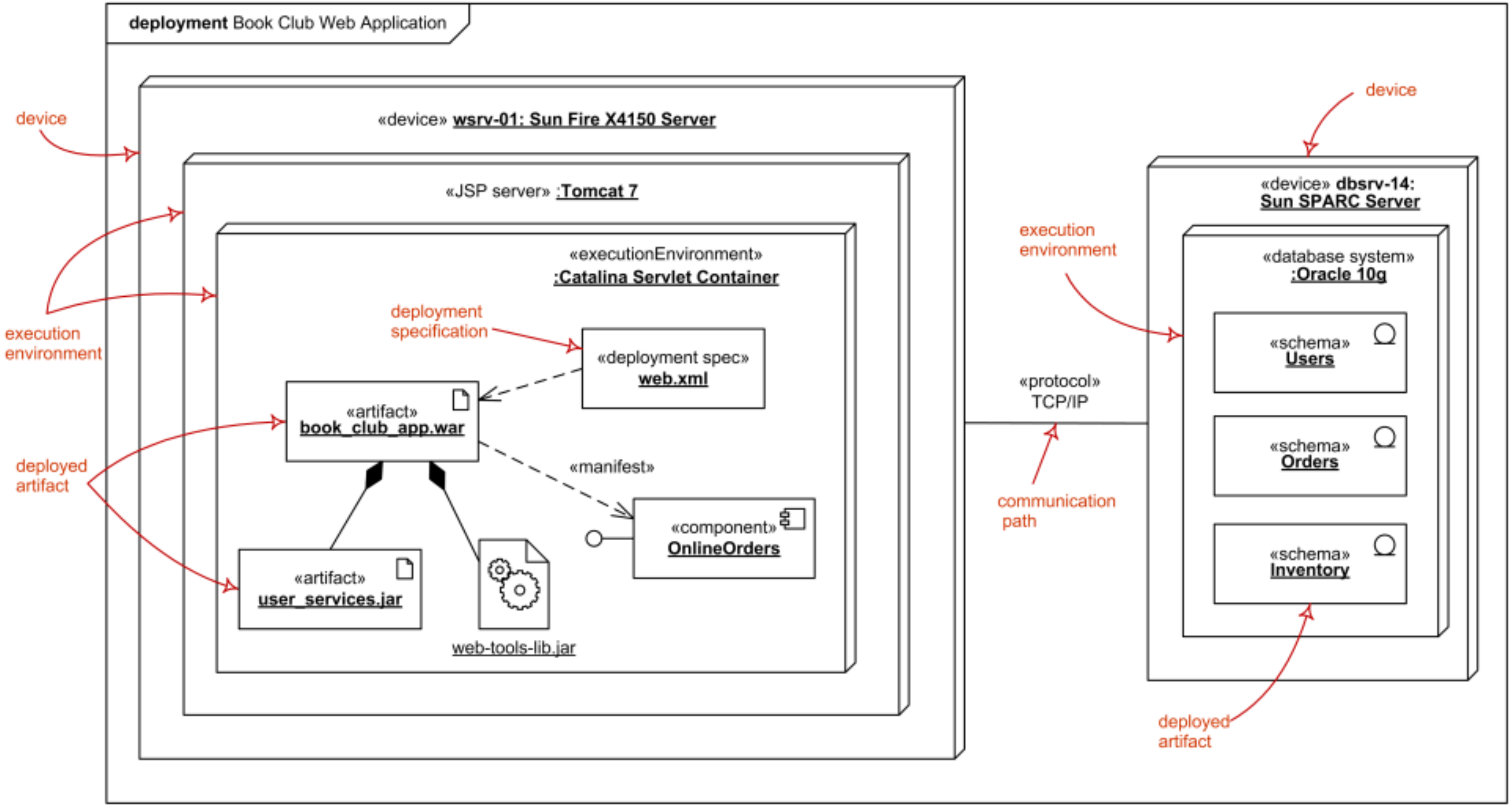
Component Diagram



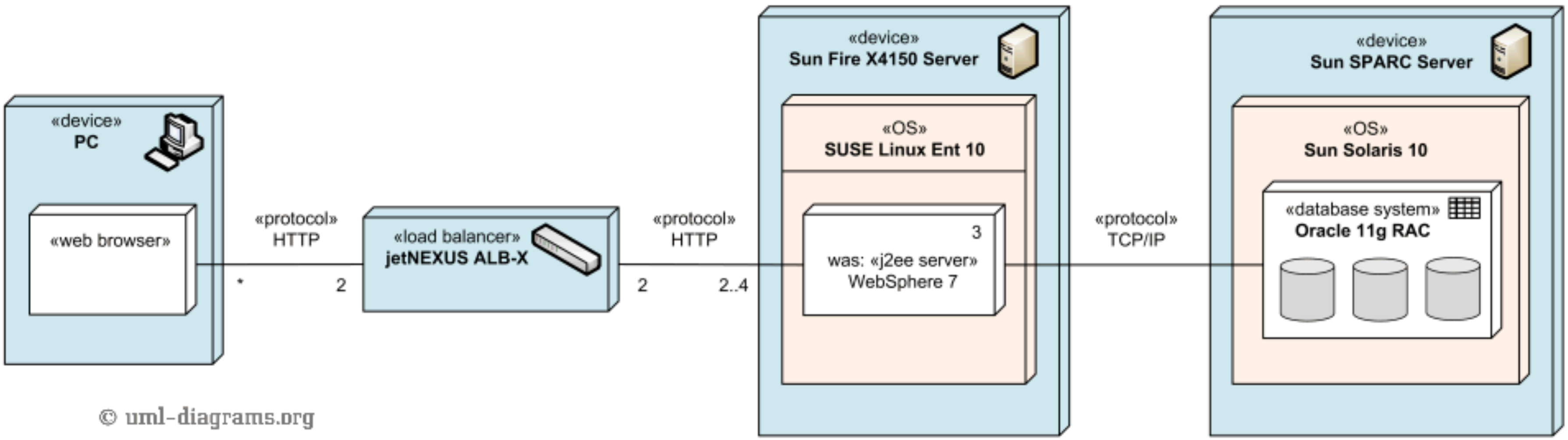
Component Diagram

- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database
- Developed by architects and programmers

Deployment diagram



Deployment diagram

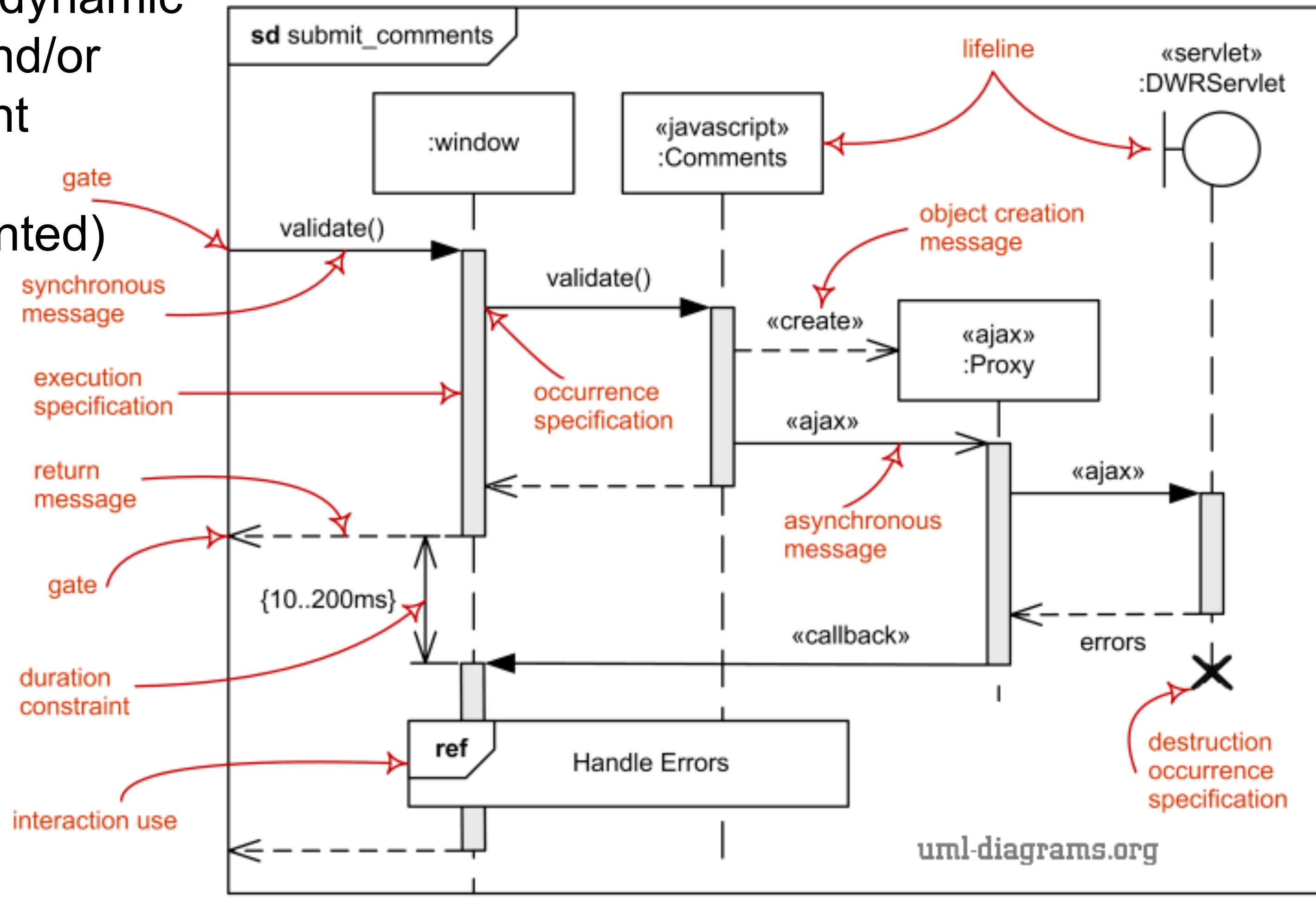


Deployment Diagram

- Captures the topology of a system's hardware
- Built as part of architectural specification
- Purpose
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers

"Architectural" Sequence Diagram
(i.e., showing interactions among components)

- Captures dynamic module and/or component behavior (time-oriented)



“Architectural” Sequence Diagram

- Captures dynamic behavior (time-oriented)
- Purpose
  - Model flow of control
  - Illustrate typical scenarios
  - Analyse architecture -ilities

References (Architecture)

- Len Bass, Paul Clements & Rick Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
- Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stahl, *Pattern-Oriented Software Architecture - A System of Patterns*, Wiley and Sons, 1996.
- Christine Hofmeister, Robert Nord, Dilip Soni, *Applied Software Architecture*, Addison-Wesley 1999.
- Eric Gamma, John Vlissides, Richard Helm, Ralph Johnson, *Design Patterns*, Addison-Wesley 1995.
- Philippe Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, 12 (6), November 1995, IEEE.
  - http://www.rational.com/support/techpapers/ieee/
- Eberhardt Rechtin, *Systems Architecting: Creating and Building Complex Systems,* Englewood Cliffs NJ, Prentice-Hall, 1991.

References (Architecture)

- Eberhardt Rechtin & Mark Maier, *The Art of System Architecting*, CRC Press, 1997.
- *Recommended Practice for Architectural Description*, Draft 2.0 of IEEE P1471, May 1998
  - http://www.pithecanthropus.com/~awg/
- Mary Shaw, and David Garlan, *Software Architecture—Perspectives on an Emerging Discipline,* Upper Saddle River, NJ, Prentice-Hall, 1996.
- Bernard I. Witt, F. Terry Baker, and Everett W. Merritt, *Software Architecture and Design—Principles, Models, and Methods*, New York NY, Van Nostrand Reinhold, 1995.
- The World-wide Institute of Software Architects
  - http://www.wwisa.org

References (UML4Arch)

- Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
- Kruchten, P.; Selic, B.; Kozaczynski, W.; Larsen, G. & Brown, A. W. (2001), Describing Software Architecture with UML., *in* Hausi A. Müller; Mary Jean Harrold & Wilhelm Schäfer, ed., 'ICSE' , IEEE Computer Society, pp. 777.