Abstracting Views of System Architecture
Exploring Perspectives of Software Intensive System Design

Raymond Jorgensen
Rockwell Collins

Objectives

- Define system architecture
- Describe traditional approach to develop systems architecture
- Discuss potential issues with traditional approach
- Discuss alternative approaches to develop system architecture
- Define architectural views necessary to capture the design of complex systems
- Discuss layering of architectural views – software views
- Demonstrate how different layers of the architecture are mapped to one another

What is System Architecture?

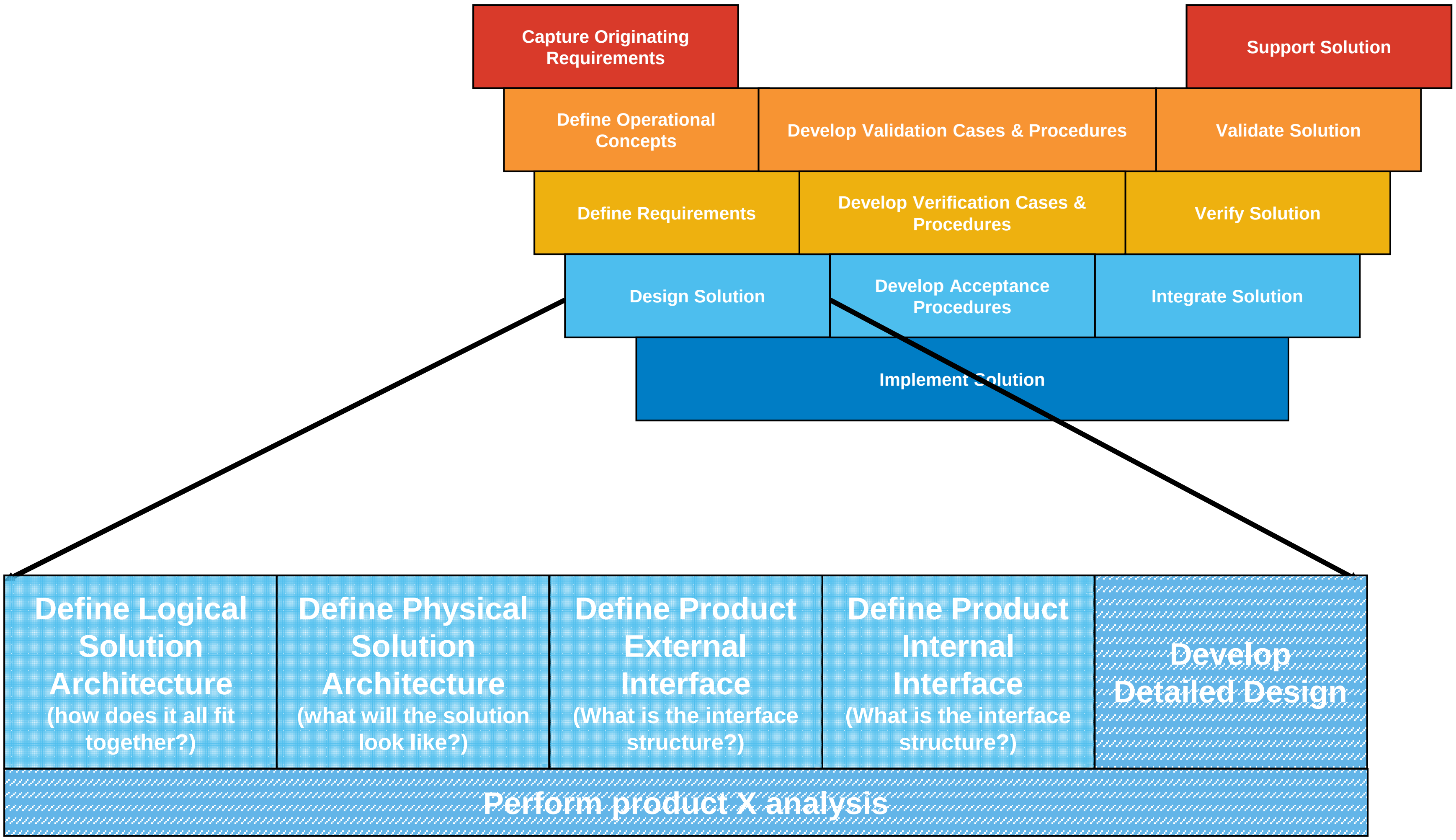
- Wikipedia:
  - A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system.
- IEEE:
  - The composite of the physical architectures for consumer products and their life-cycle processes. (P1220)
  - The organizational structure of a system or component. (STD 610.12)
  - A logical or physical representation of a product which depicts its structure, but, provides few or no implementation details. (P1220)
- DERA
  - The structure of levels and/or branches that partition a system into its constituent parts or components.
- NASA
  - How functions are grouped together and interact with each other. (MDP92)

Definition of System Architecture

- The definition adopted by Rockwell Collins Architecture Standard (RC-ENG-S-101)
  - The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

    ISO/IEC/IEEE 42010

Architecture in the Process

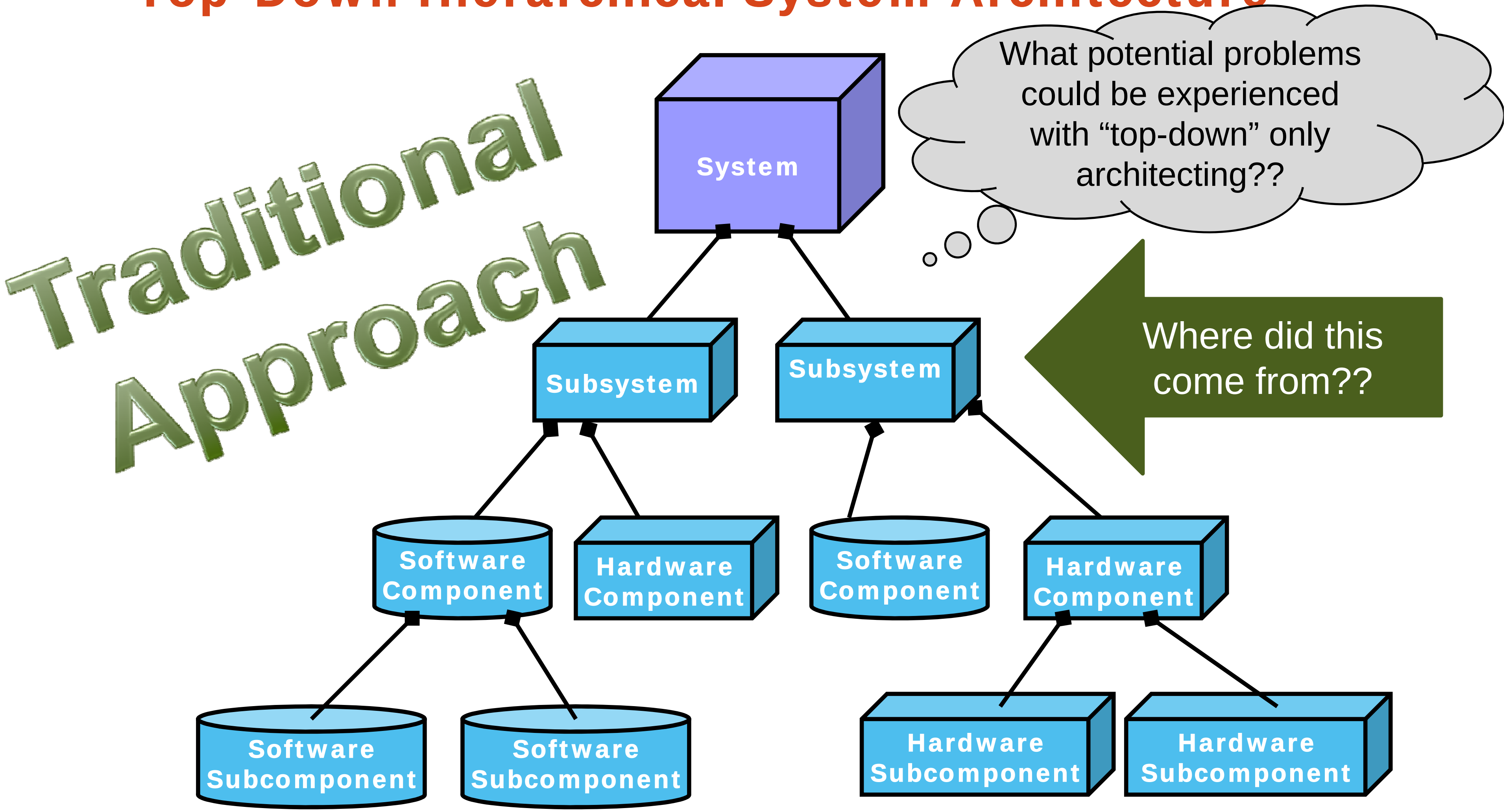


Why is Architecture important?

- Clearly defines the components of the system and how they relate to one another
  - Both logical and physical components
- Identifies dependencies between system components
- Allows for clearly allocating system non-functional requirements flow down to components
- Exposes interface points early and drives interface definition to improve independent development of components
  - As long as components conform to their interfaces, they can be developed independently
- Helps provide a map for integration of system components
- Aids in clearly communicating with all stakeholders
- Facilitates reuse of system components by clearly defining their boundaries, functions, and interactions

Rockwell Collins

Top-Down Hierarchical System Architecture



Rockwell Collins

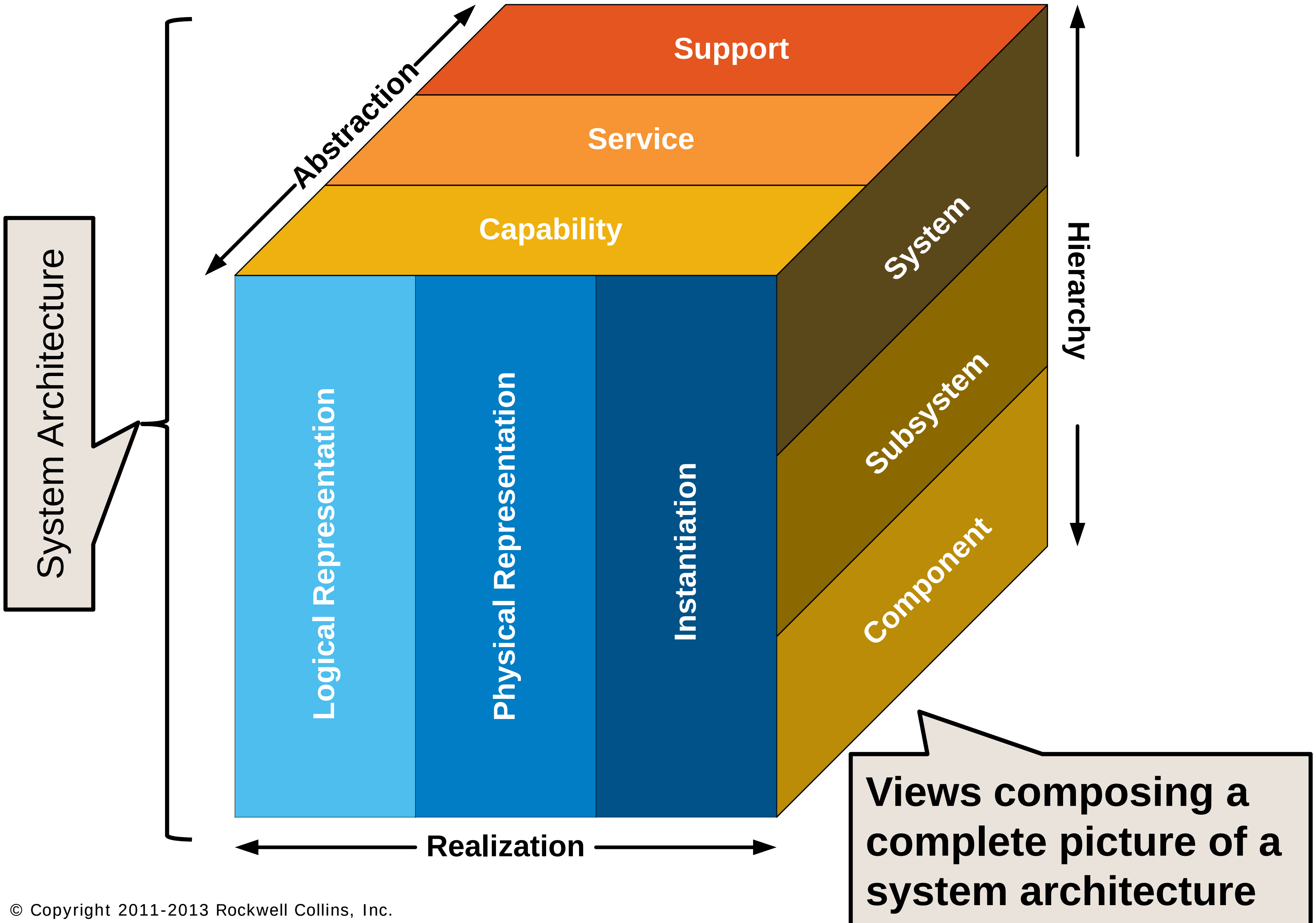
Traditional Top-Down System Approach

- What is a “subsystem”?
- How does a “system” differ from a “subsystem”?
- Can a system share components with other systems within the same hierarchy?
- What happens when the “subsystem” does not have unique components?
- How has “plug and play” and “modularity” changed the way we engineer or manage our products?

Is there a better way?

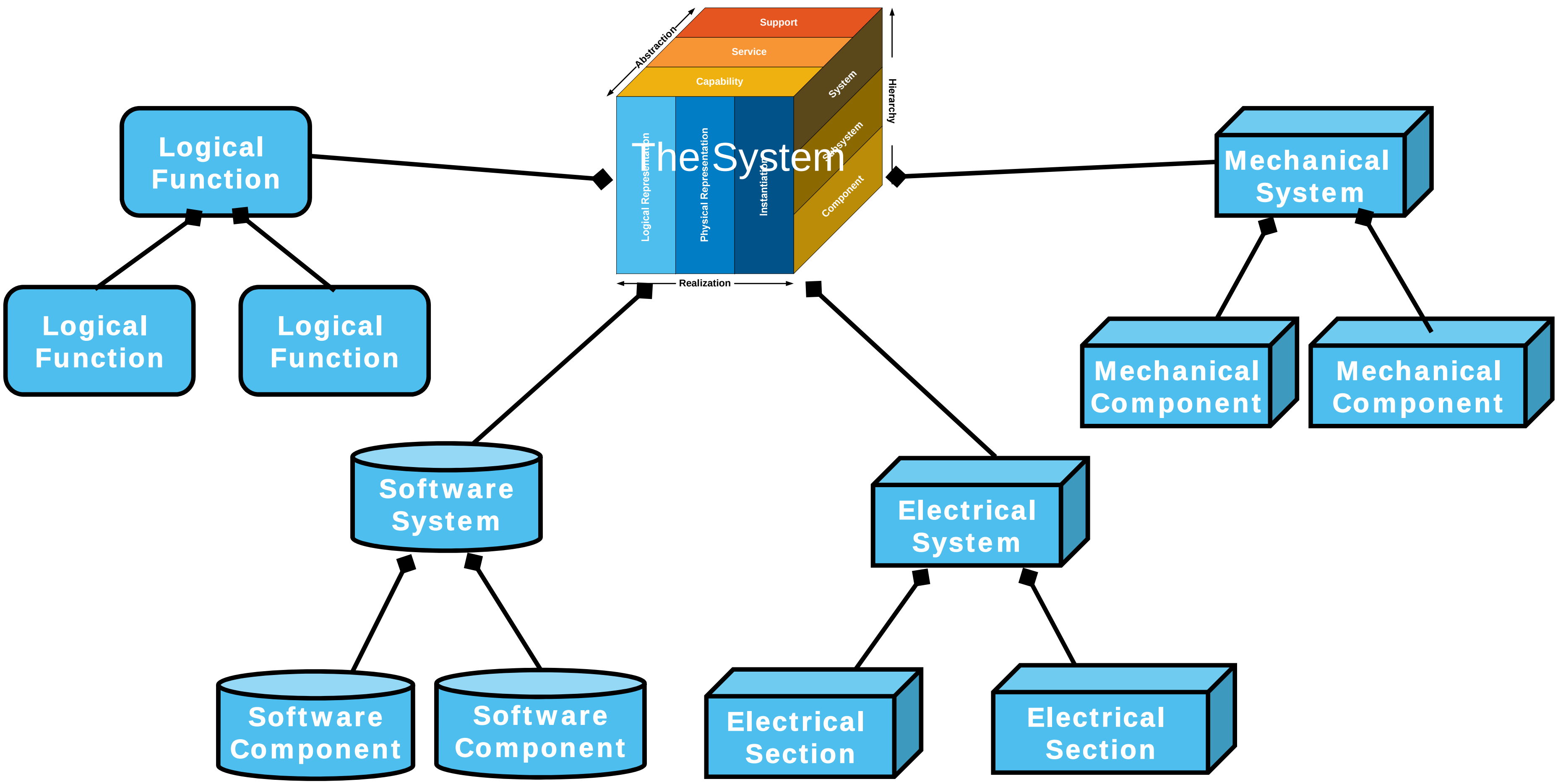

1978
1985
1990
1993
1995
2003
2010

Integrated Modular Solution Architecture (IMSA)



Rockwell Collins

Integrated Modular Solution Architecture (IMSA)



Characteristics of IMSA

- Independent software architectures
  - Software architecture can **exist** independently from underlying physical hardware architecture!
    - Two projects with identical software architecture could have very different hardware configurations.
  - Software applications/ programs can **reside** anywhere in overall physical hardware architecture
    - Application software design abstracts out hardware interface so that it is not tied to a single box
  - Software architecture **defined** independently from the hardware
    - Allocation of application software to specific processors requires analysis to assess resource usage
    - Concerns – latency, throughput, and processor loading – can the underlying architecture handle resource usage?

Solution Architecture Views

- No one "picture" shows the full architecture of the solution
  - How is the architecture captured and managed?
  - Is a hierarchy view enough?

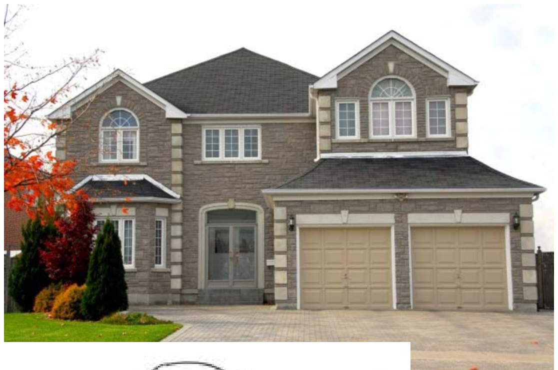
- Architecture contains
  - **Components** with structure and composition
  - **Behavior** showing inputs, processing, and outputs
  - **Relationships** showing interconnectivity of the solution

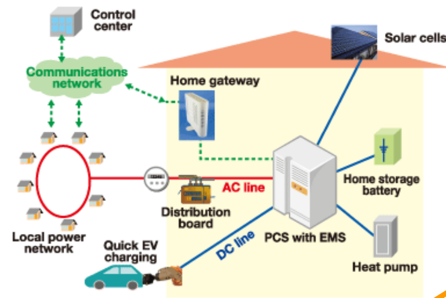
- Multiple perspectives result in multiple views of the architecture
  - Software, Hardware, Behavioral, etc.

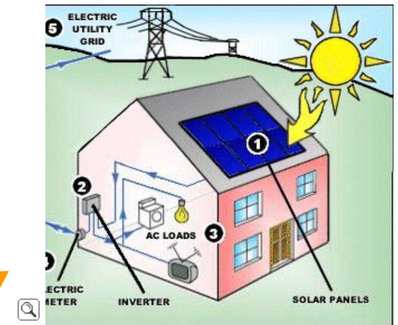
- Interdependencies between views must also be captured
  - Interfaces and relationships

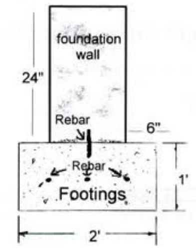
Rockwell Collins

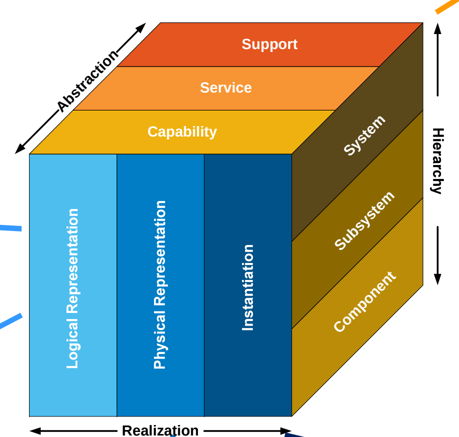
"Your House for Example"

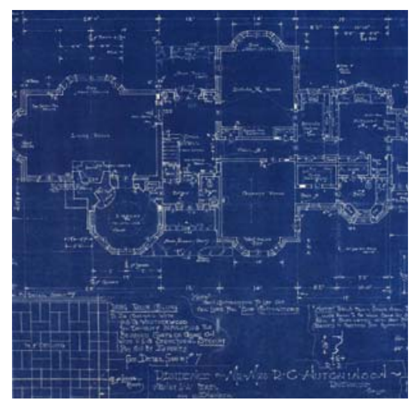






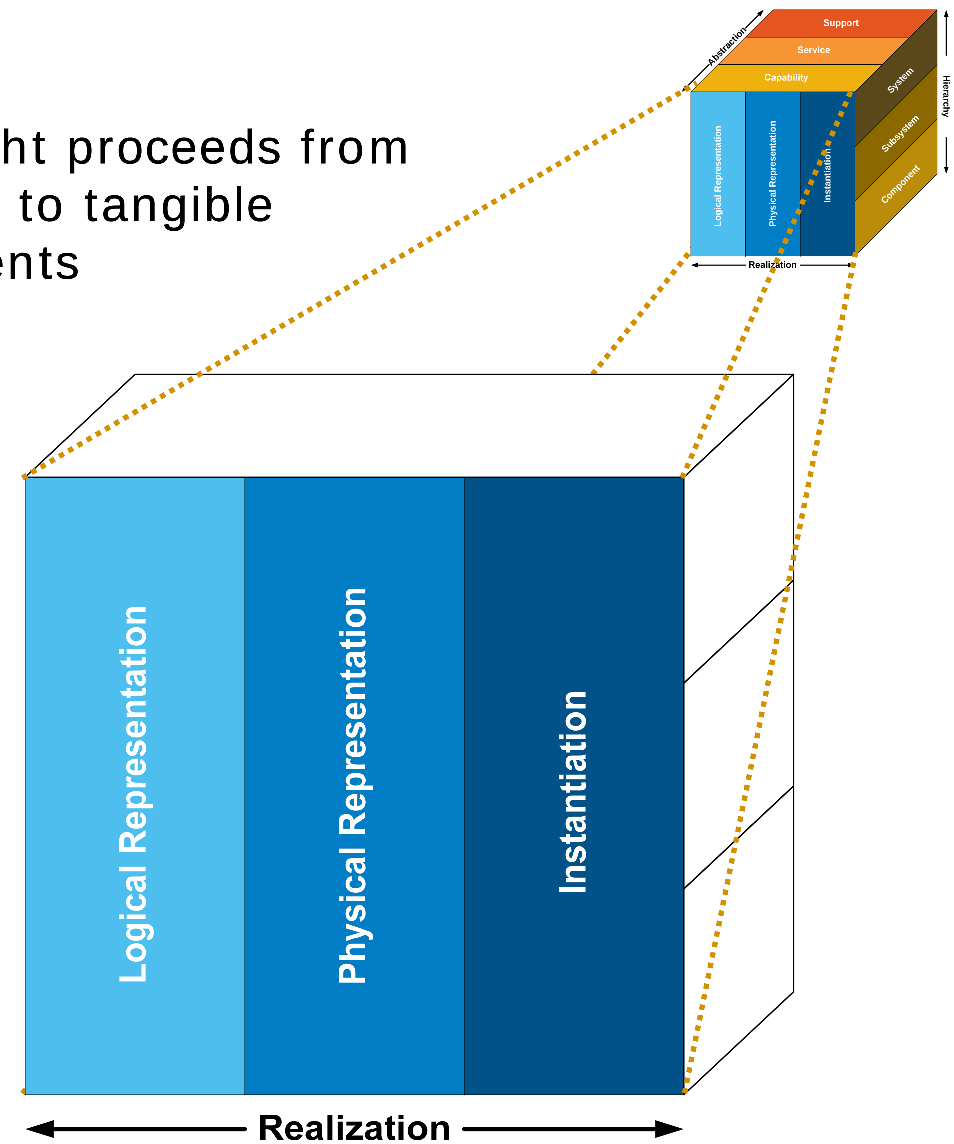


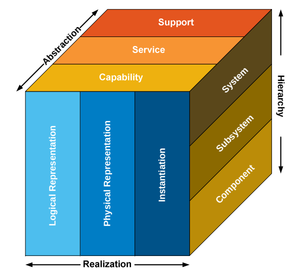




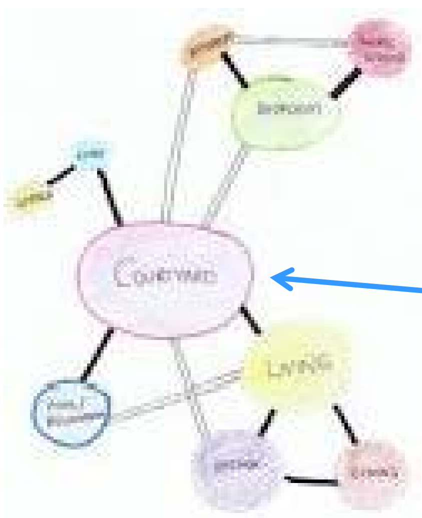
Realization Views

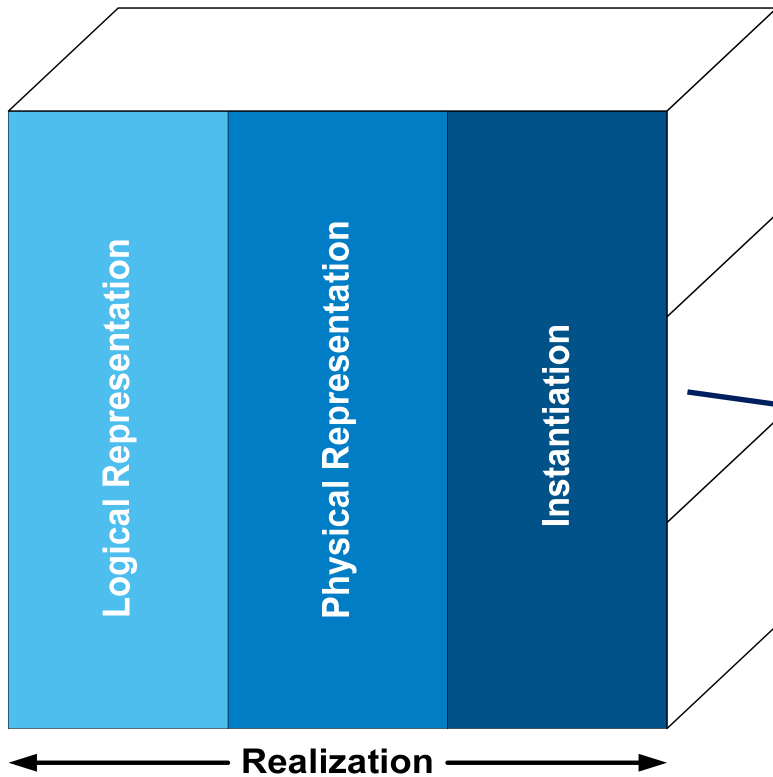
- Progression from left to right proceeds from abstract, notional concepts to tangible assets to realized components

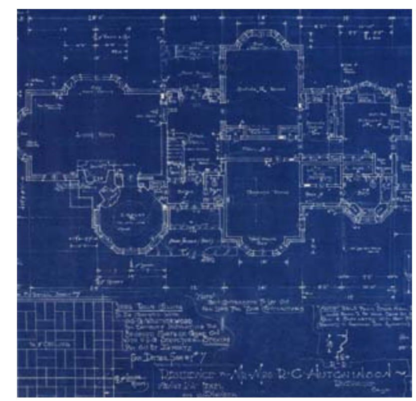




Realization Views

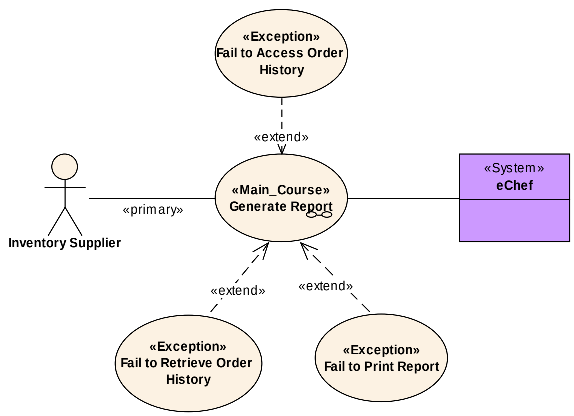


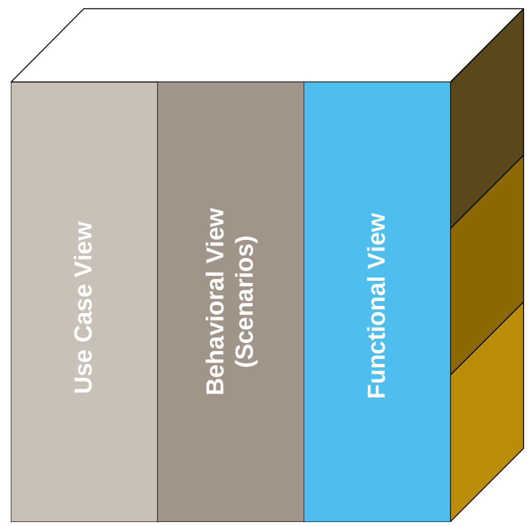


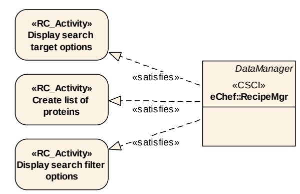


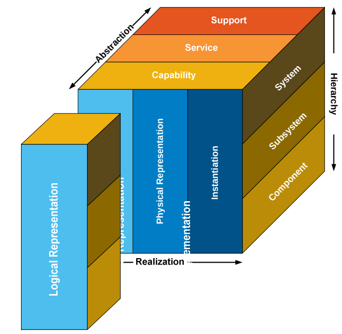
Rockwell Collins

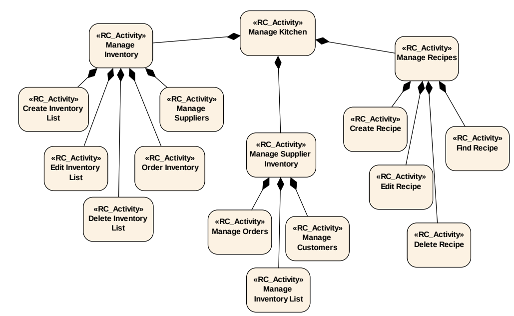
Logical Representations Summary

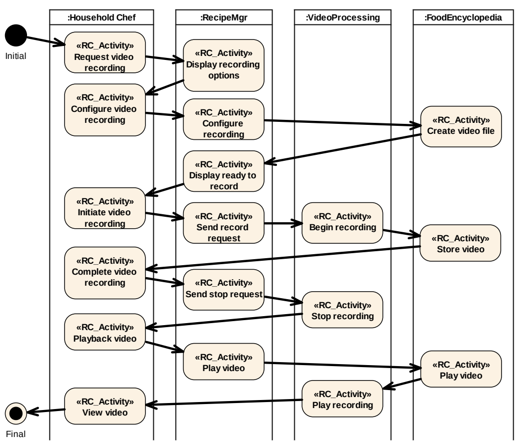










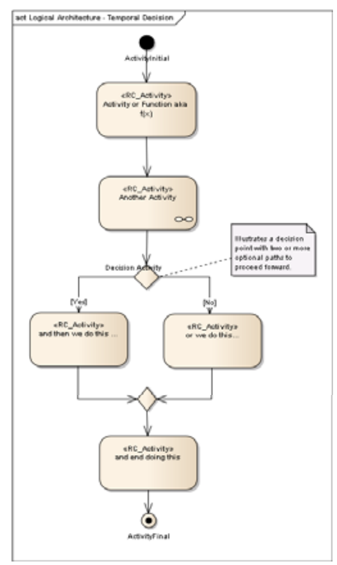


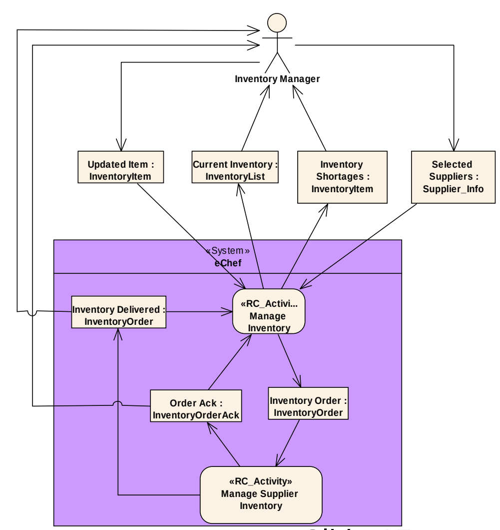
Hierarchical Perspective

Sequential Perspective

Transactional Perspective

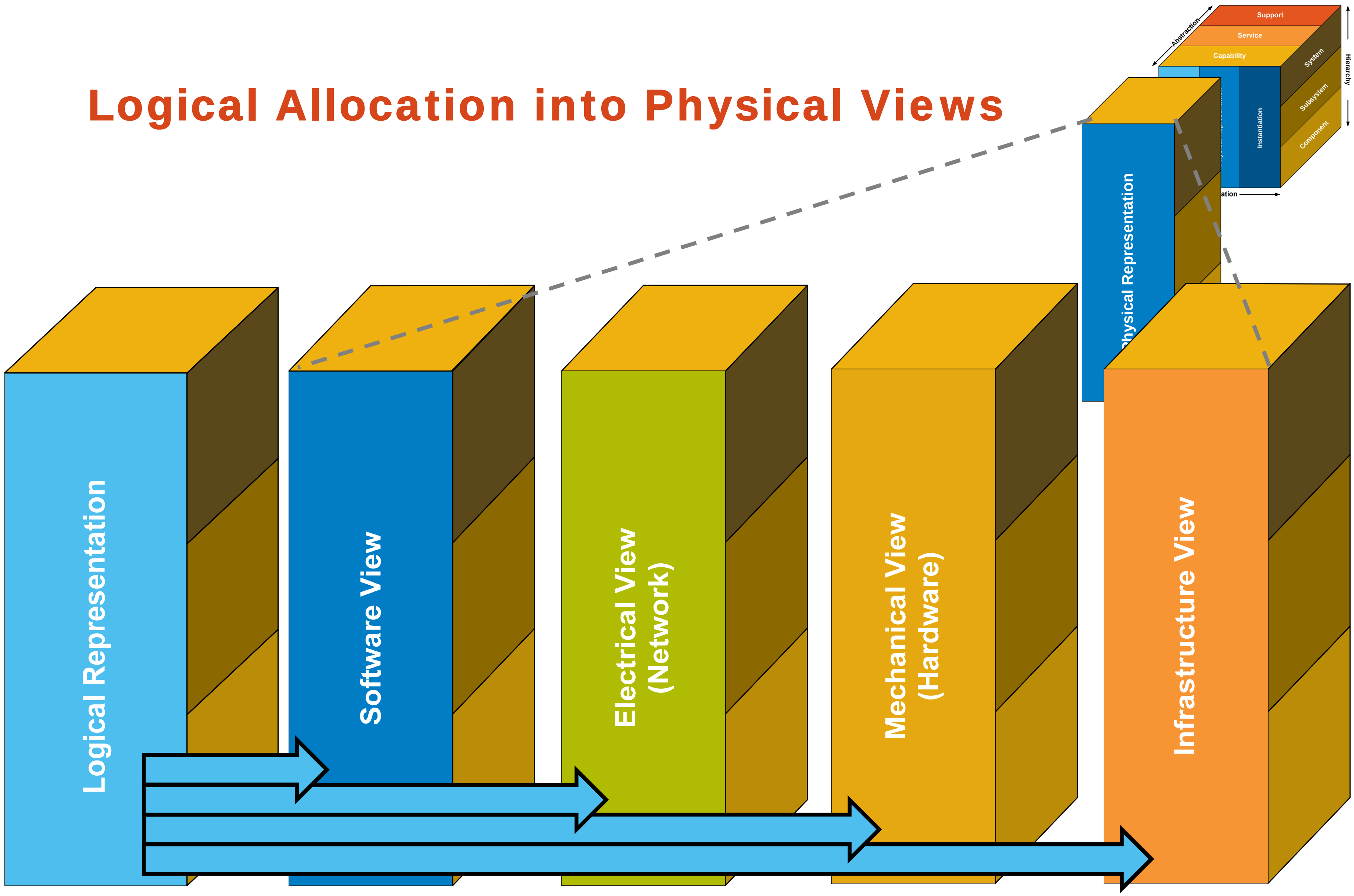
Allocation Perspective





Rockwell Collins

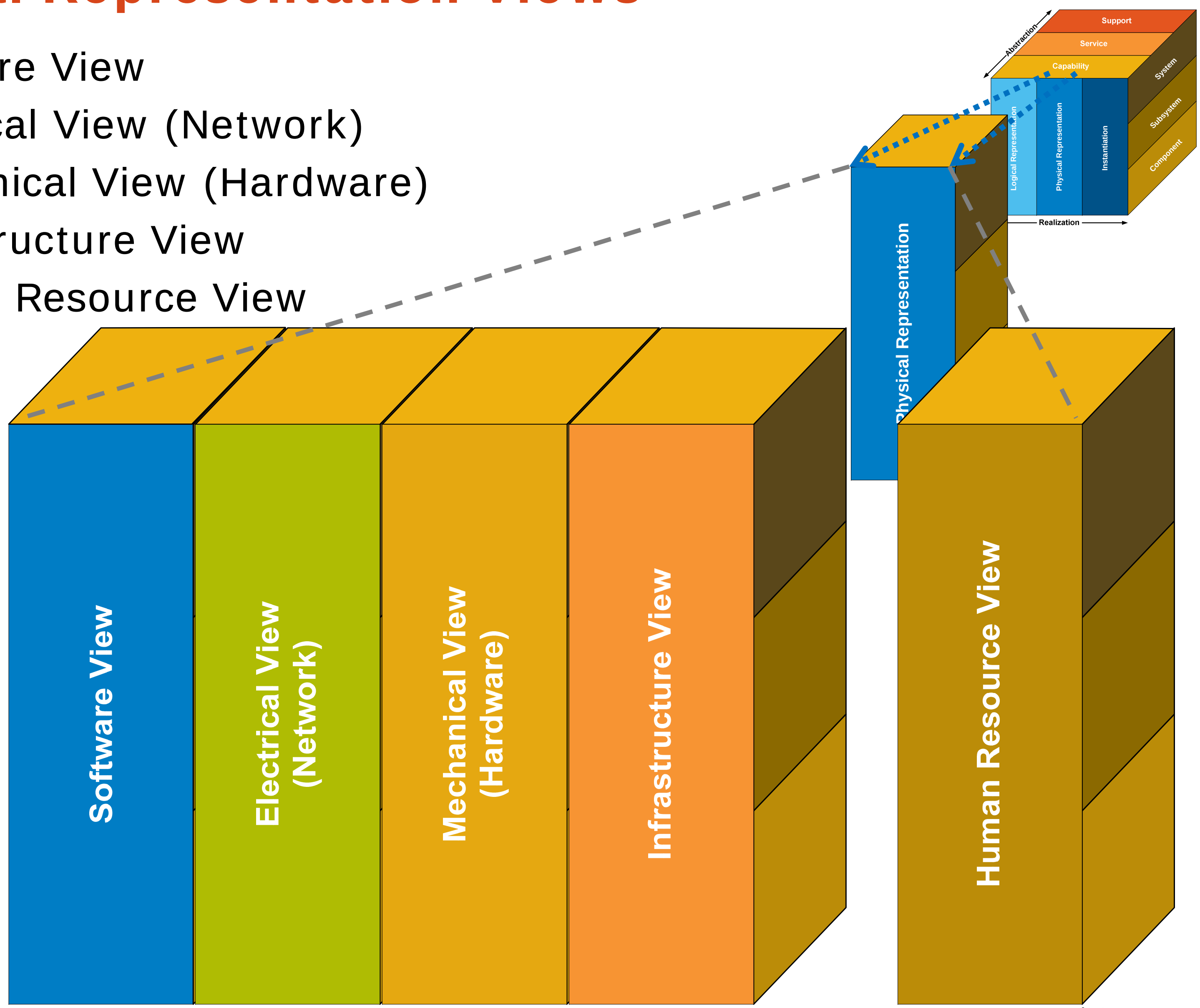
Logical Allocation into Physical Views



Rockwell Collins

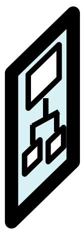
Physical Representation Views

- Software View
- Electrical View (Network)
- Mechanical View (Hardware)
- Infrastructure View
- Human Resource View



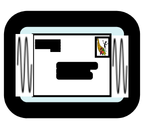
Physical Representation – Software View

- Software Views

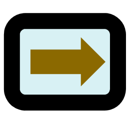


Hierarchical Perspective

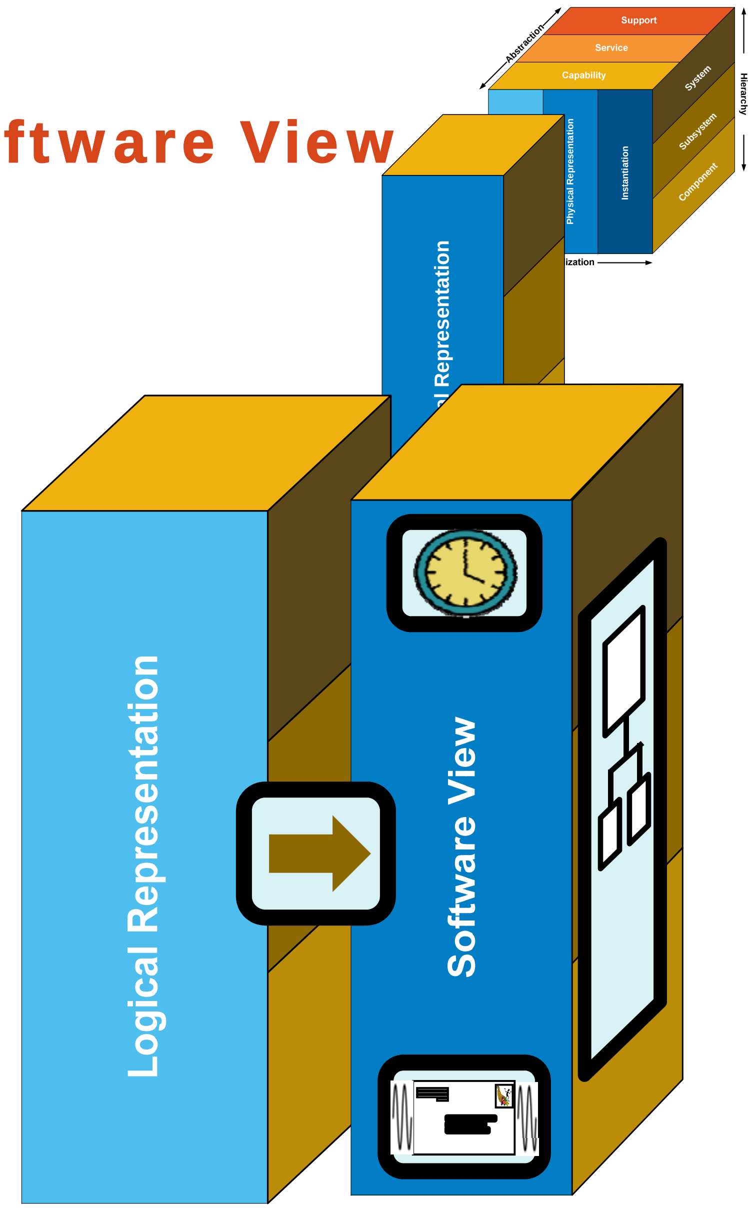
Sequential Perspective



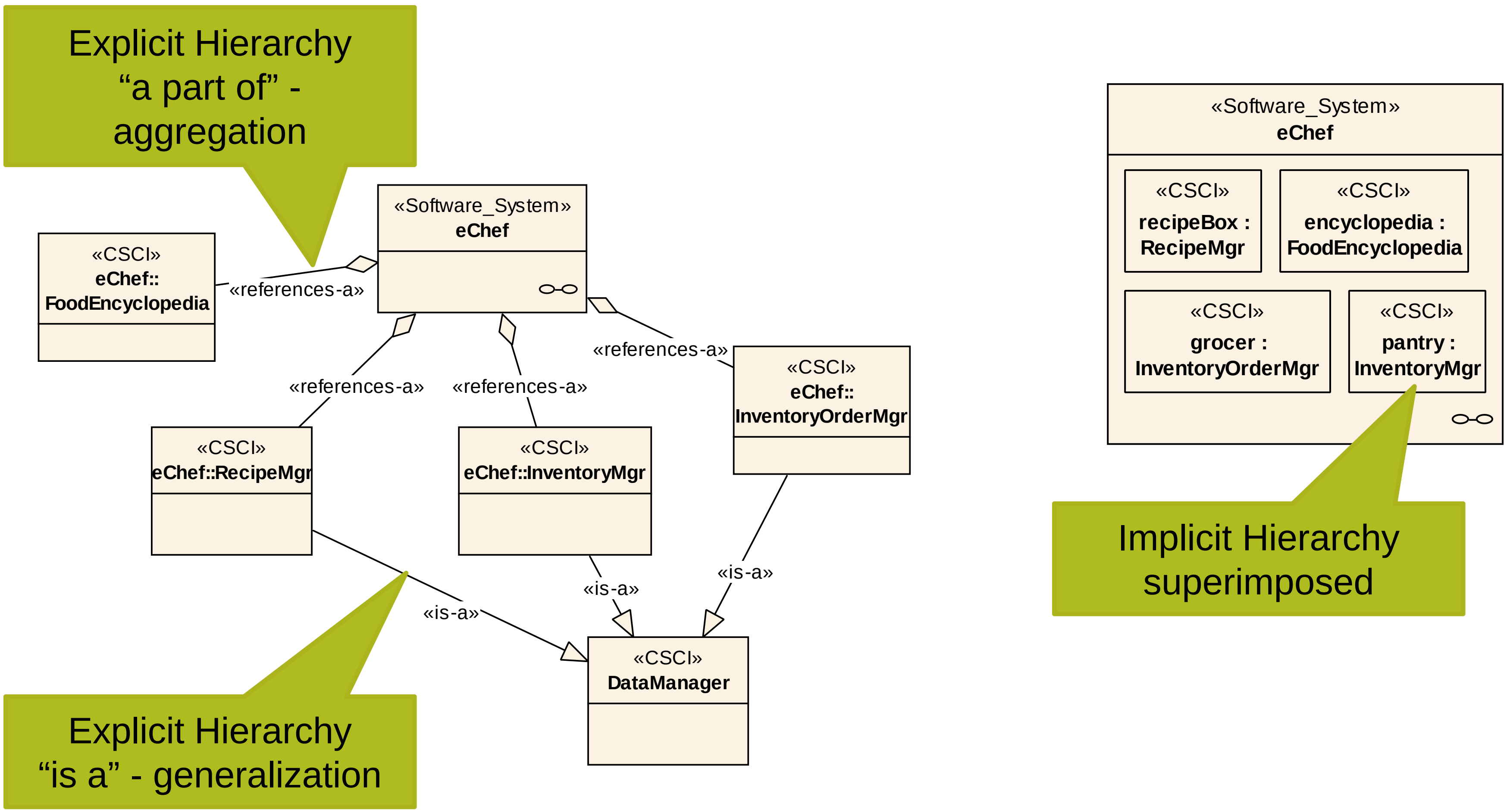
Transactional Perspective (Messaging)



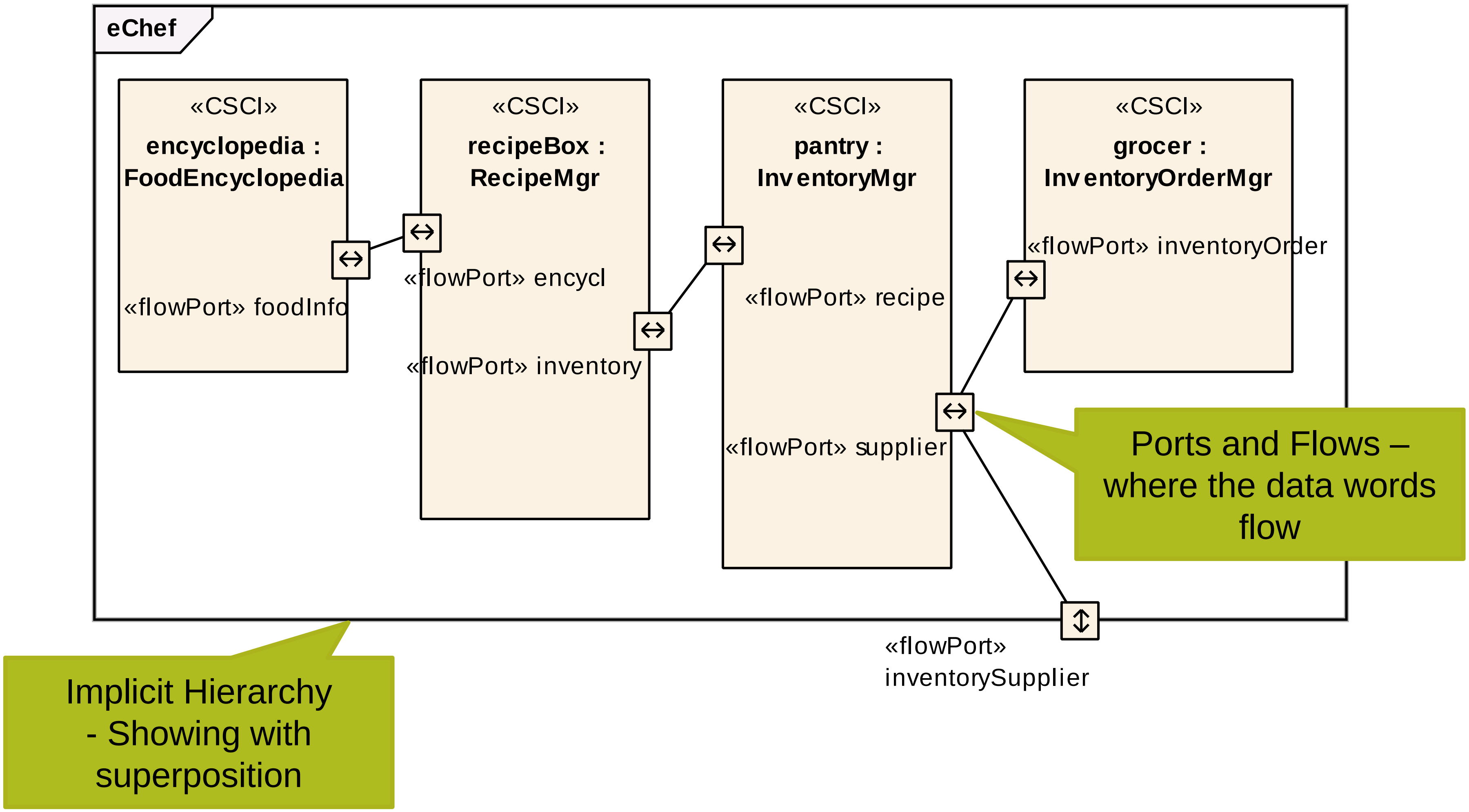
Allocation Perspective (from logical)



Software View – Hierarchical Perspective

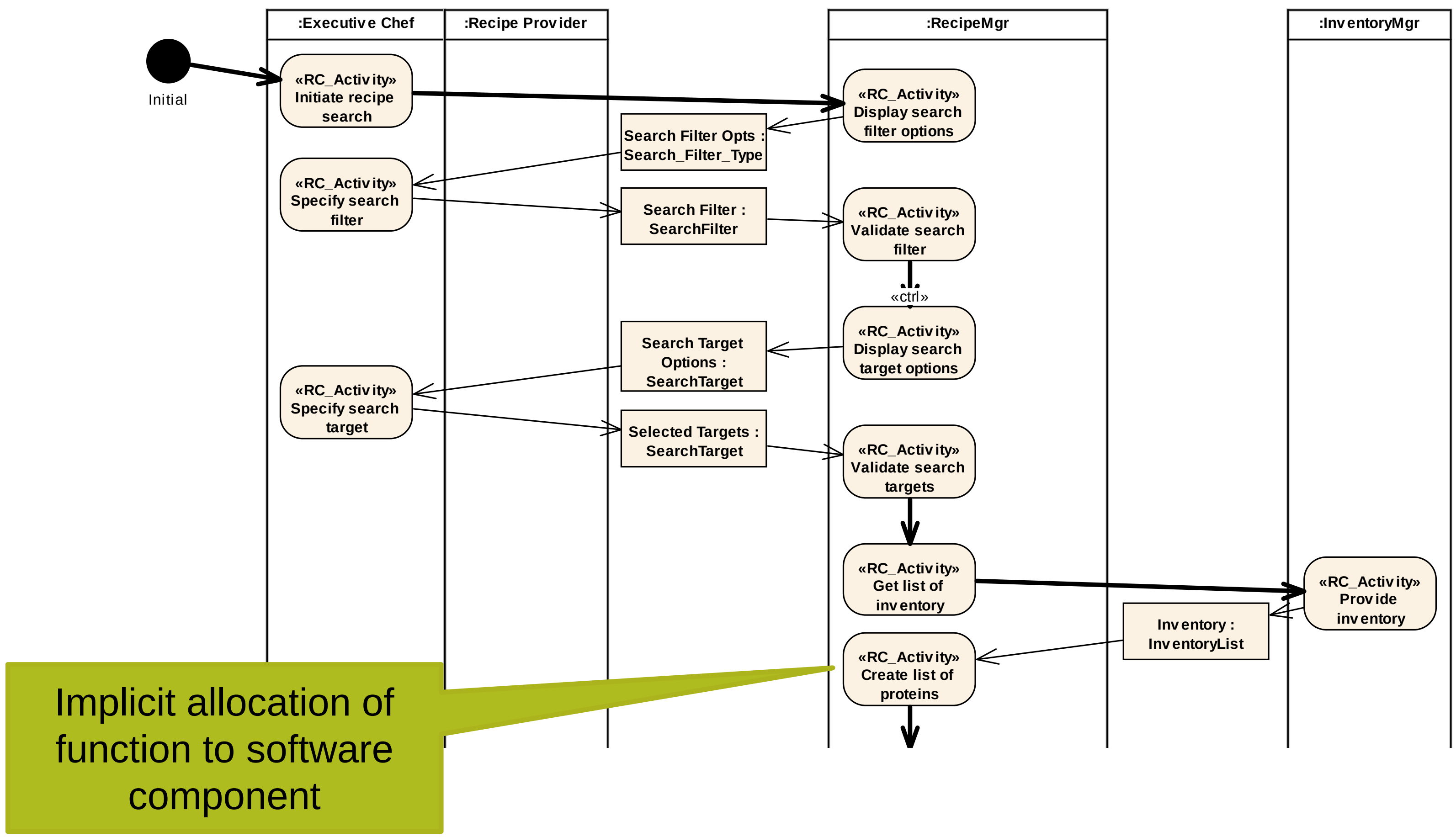


Software View – Transactional Perspective



Software View – Allocation Perspective

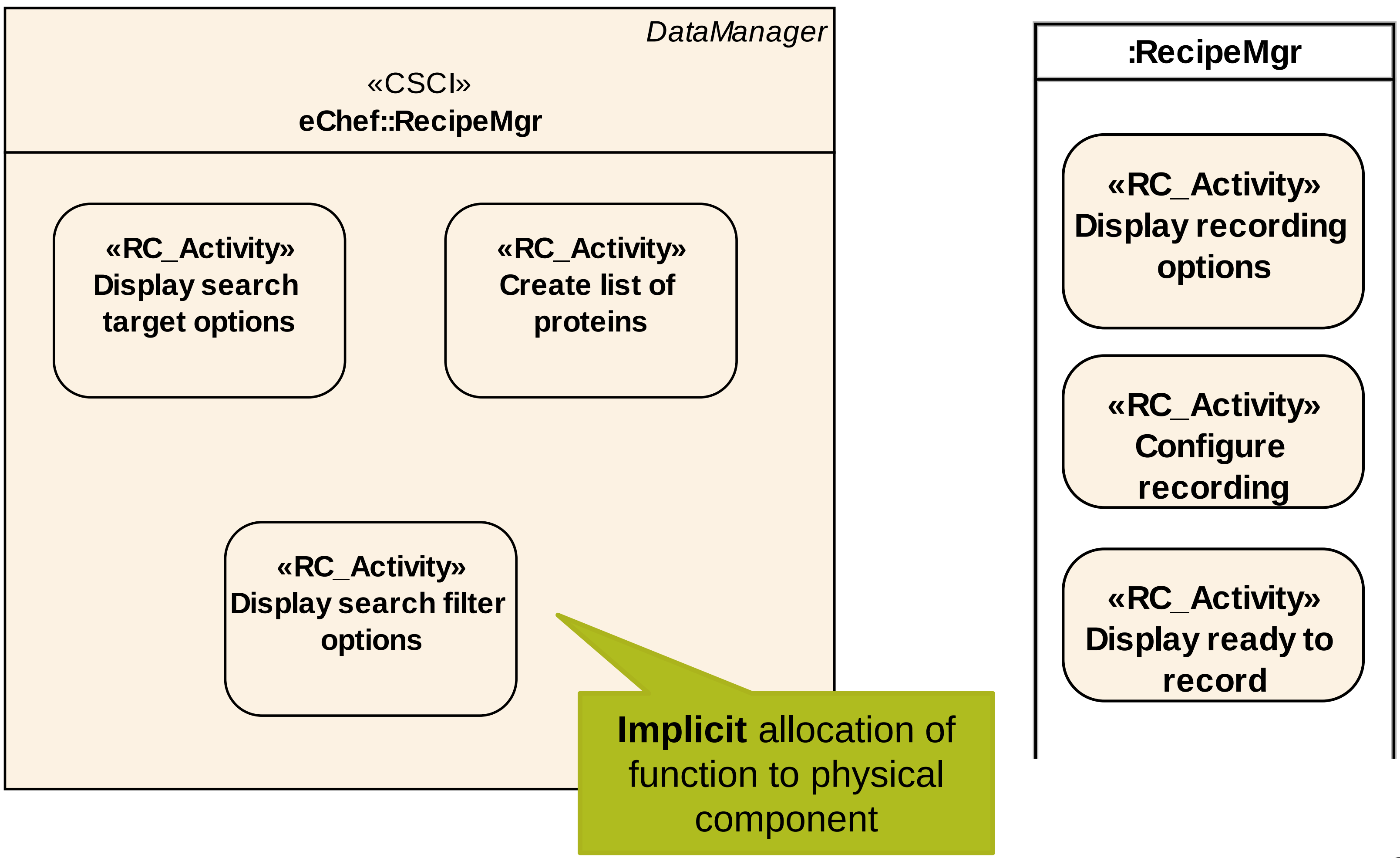
Logical Elements to Software Partitions



Functional View - Allocation Perspectives

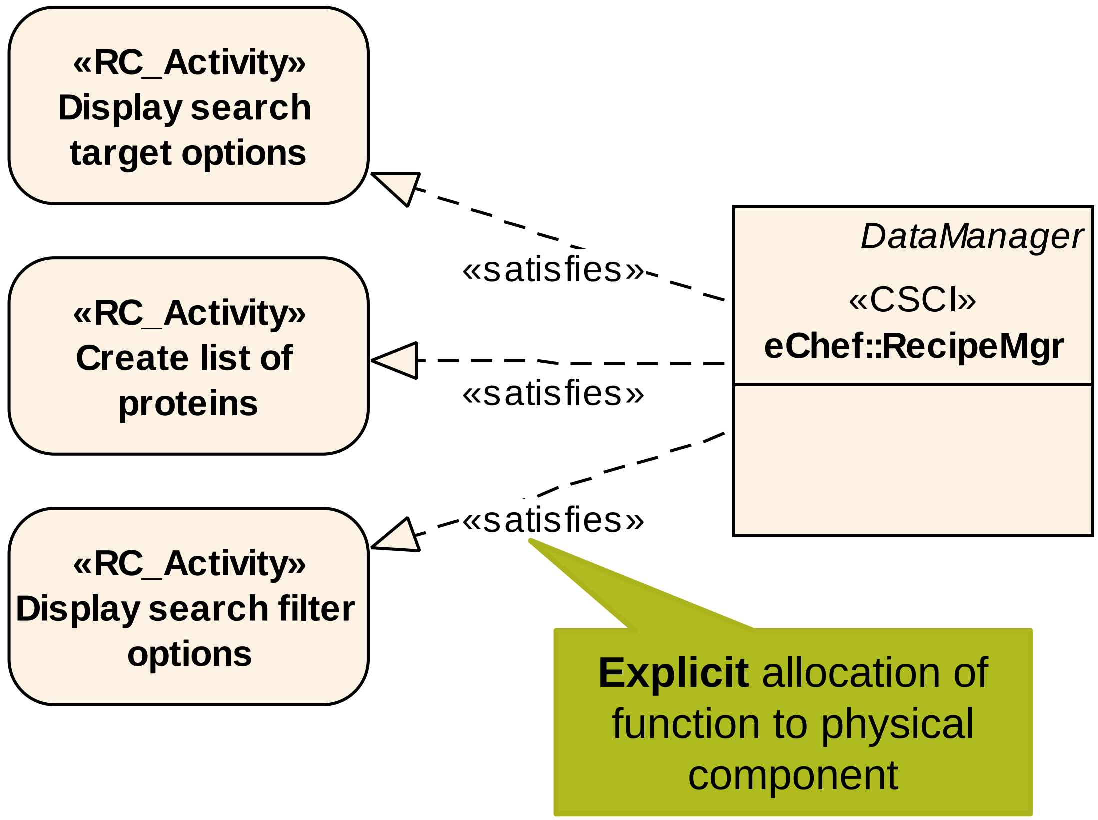
Activity Overlay on Physical Element

Activity in Partition/Swimlane

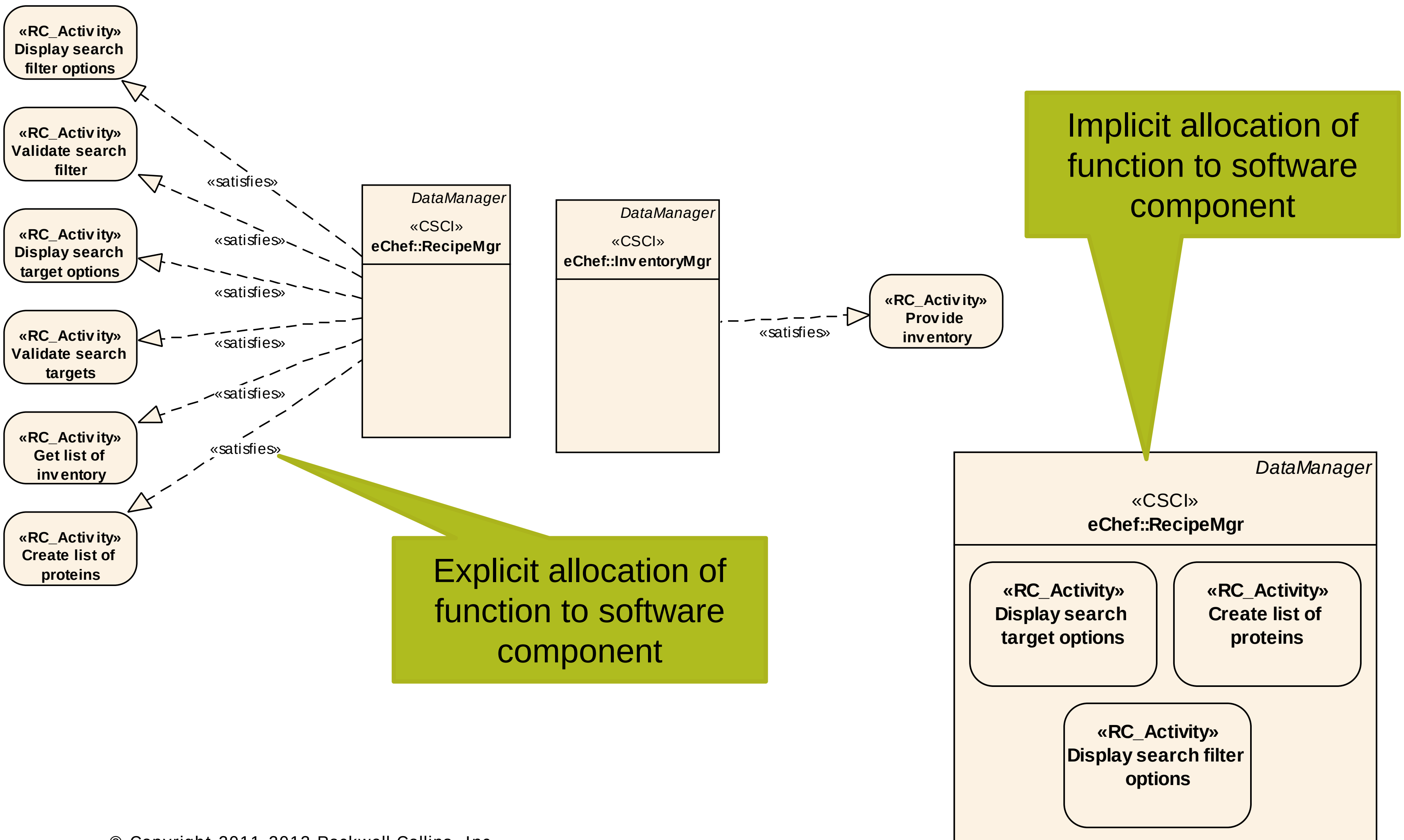


Functional View - Allocation Perspectives

Realization relationship between logical and physical elements

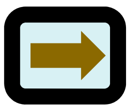


Software View – Allocation Perspective
Logical Elements to Software Elements

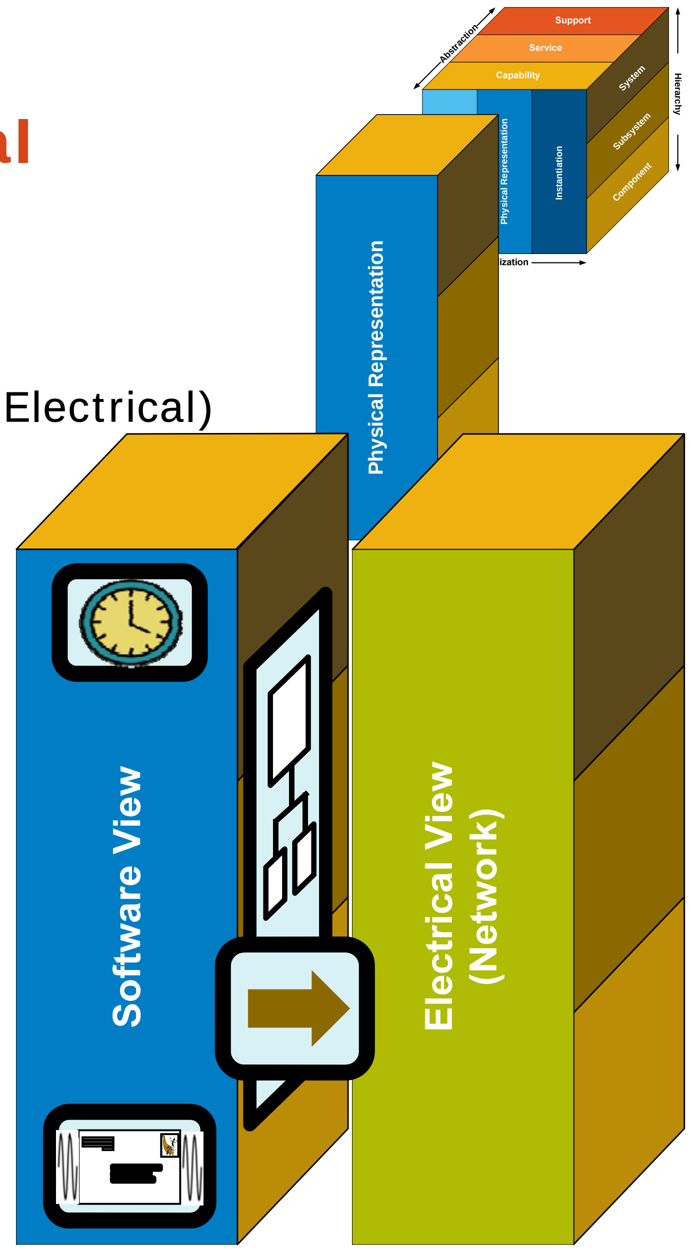


Software Allocation to Electrical

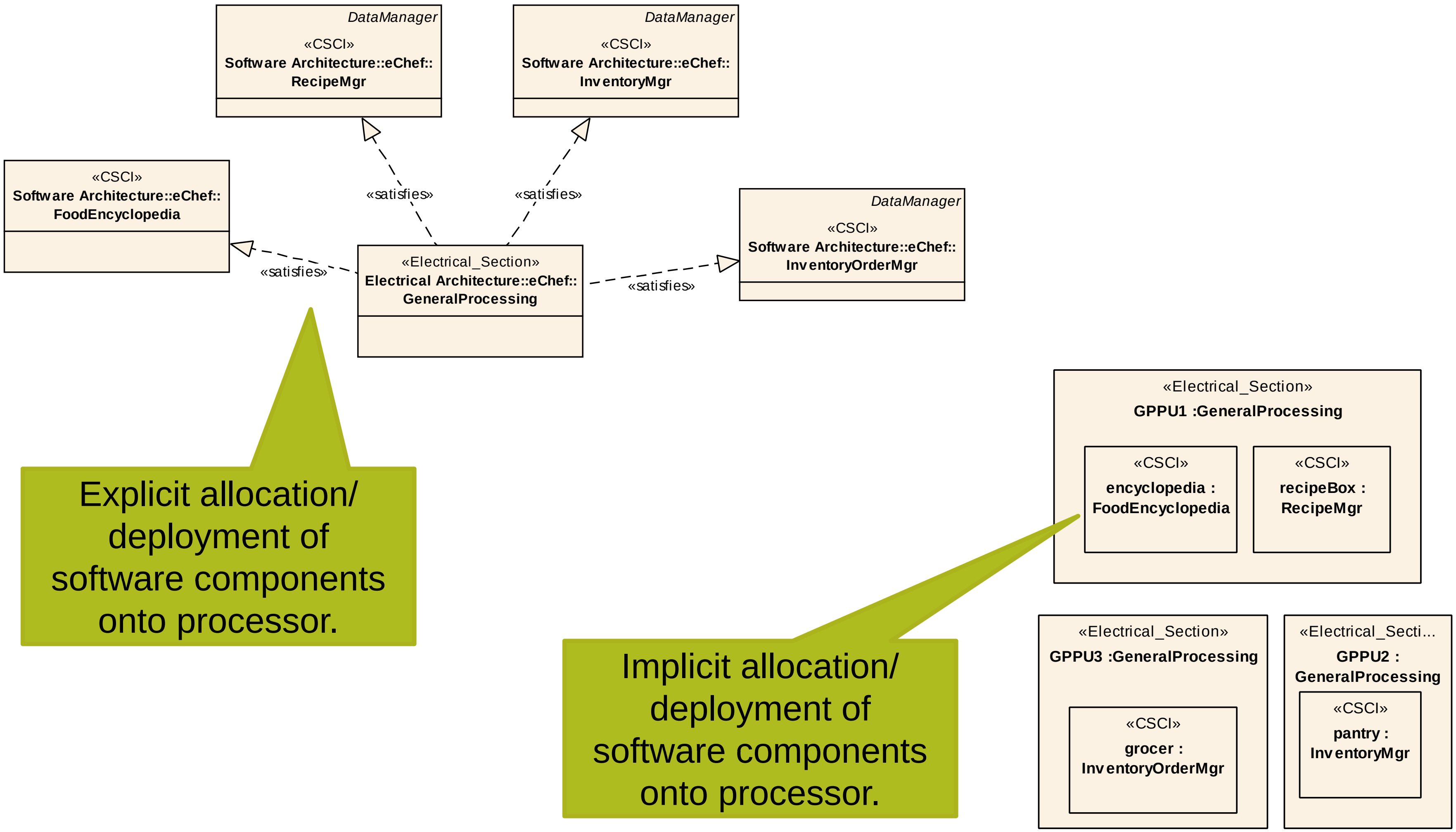
- Software View



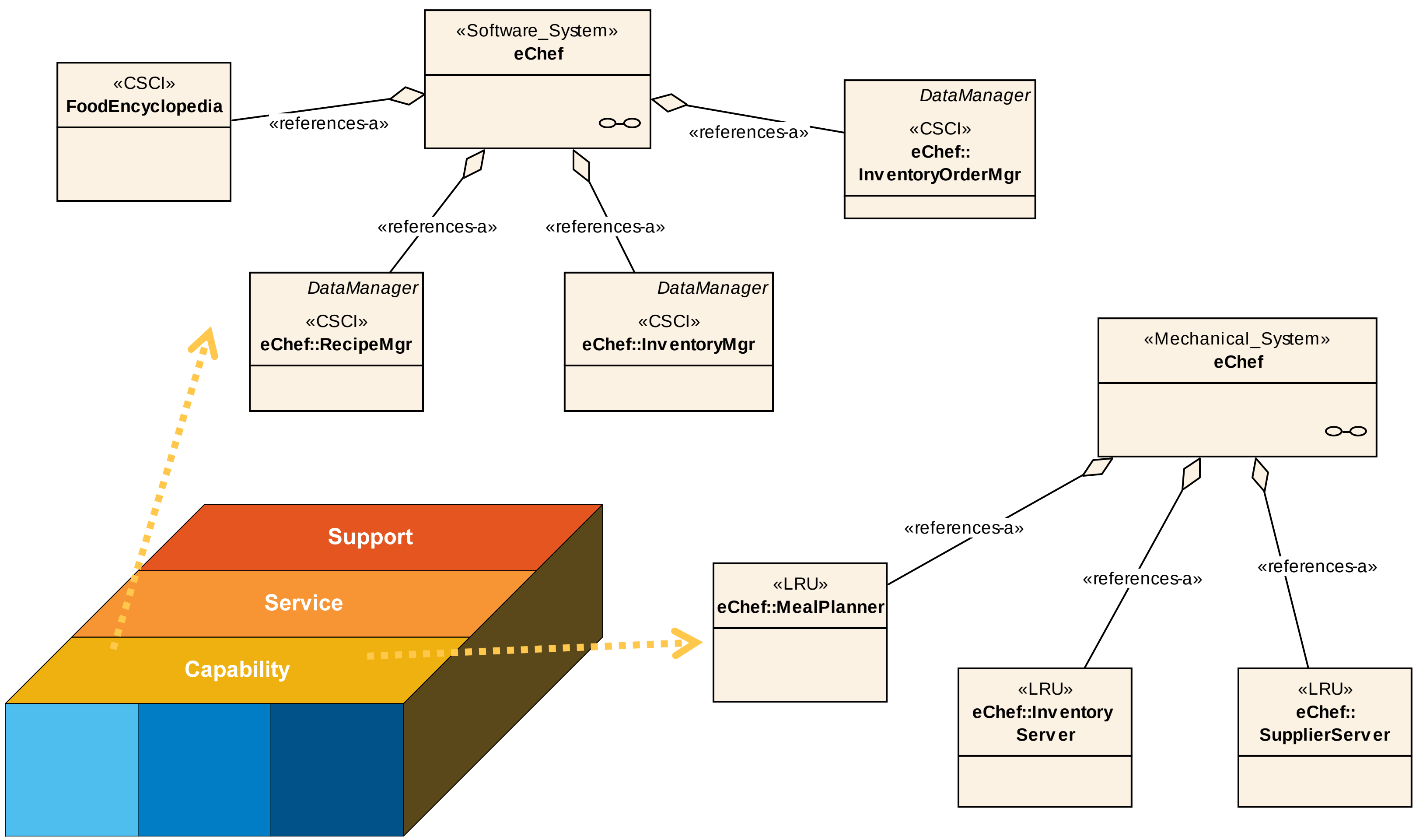
Allocation (or Deployment) Perspective (to Electrical)



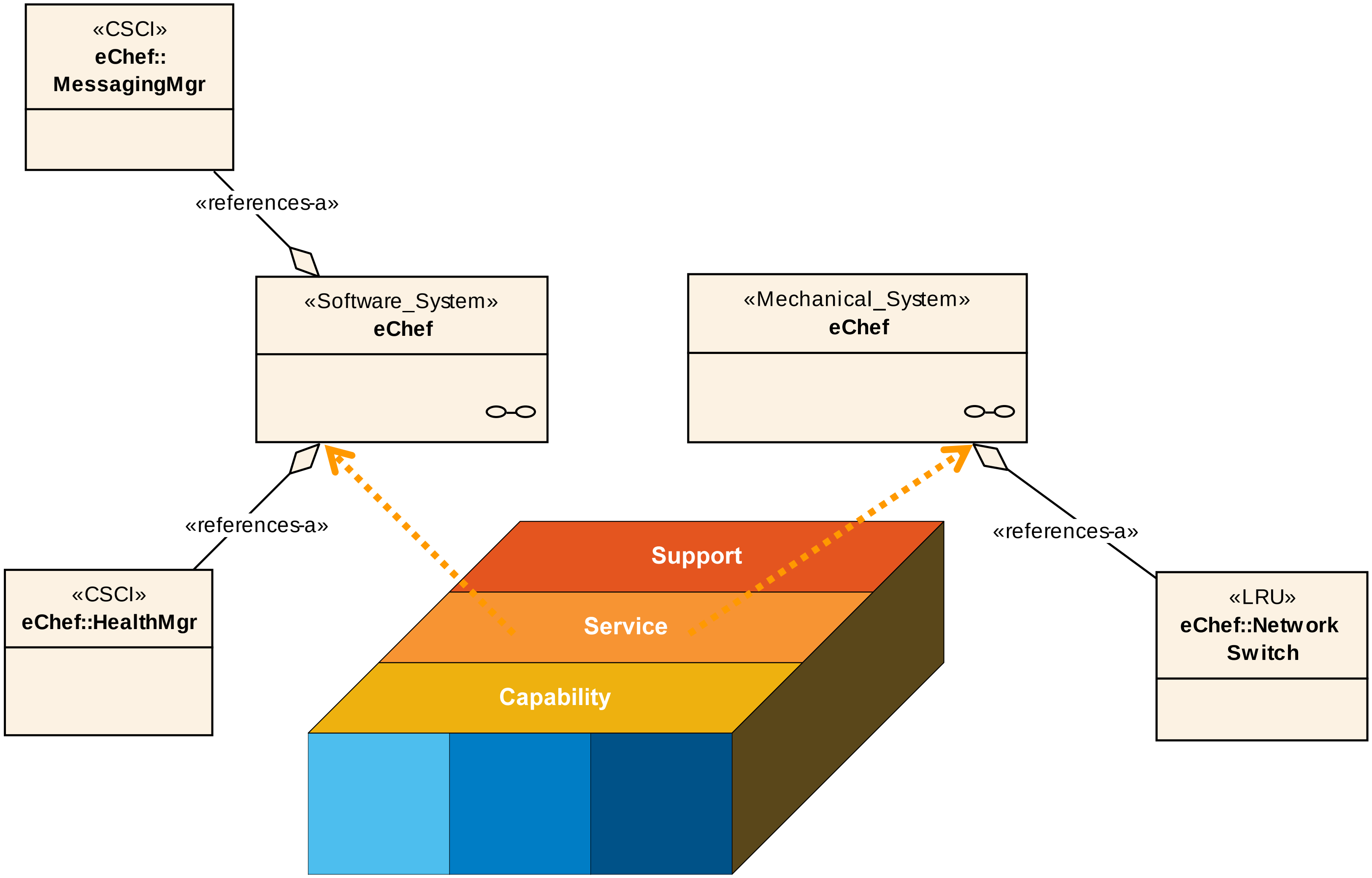
Software View –
Allocation (or Deployment) Perspective



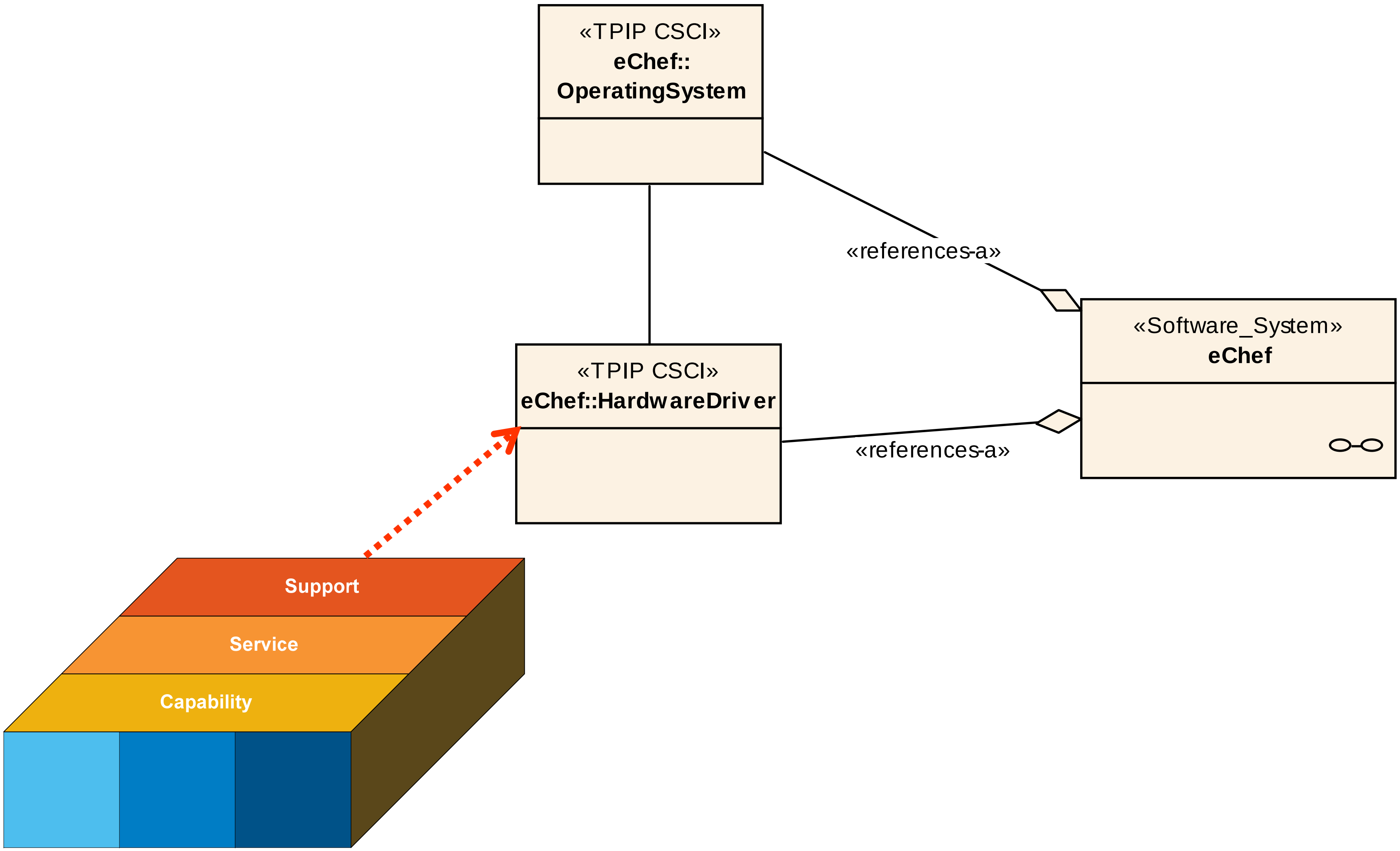
Abstraction Views - Capability



Abstraction Views – Service



Abstraction Views - Support



Summary

- Define system architecture
- Differentiate architectural design from detailed design
- Describe traditional approach to develop systems architecture
- Discuss potential issues with traditional approach
- Discuss alternative approaches to develop system architecture
- Define architectural views necessary to capture the design of complex systems
- Discuss layering of architectural views – software views
- Demonstrate how different layers of the architecture are mapped to one another

Conclusions

- Today's integrated modular solution architectures preclude a "top-down" only, traditional decomposition view
- The "software system" is a unique view that our complex systems must address
  – The interactions between software components has become one of the most complex facets of today's systems
- Four primary relationships to explore between objects
  – Hierarchical – composition, "kind of"
  – Transactional – information and object/material flows
  – Chronological – sequential/parallel timing
  – Allocation – aka deployment

Presenter Information

Raymond Jorgensen
Rockwell Collins, Inc
400 Collins Rd NE, MS 188-400
Cedar Rapids, IA 52498
rwjorgen@rockwellcollins.com
319-295-2615