Modeling SW-Architectures using UML-RT/UML 2.0

Ingolf H. Krueger
ikrueger@ucsd.edu

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA

California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA

Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

Is the UML Good Enough?

- The UML offers a plethora of description techniques for many aspects of software architectures
- The UML has, however, also significant deficits especially when it comes to modeling complex, service-oriented systems!
- In particular, we miss:
  - An adequate notation for services
  - A non-technical component notion
  - Clear concepts for hierarchy
  - Strong concepts and description techniques for
    - logical component distribution
    - non-technical interfaces
  - Formal means for behavior descriptions with respect to interfaces

Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

What is UML-RT?

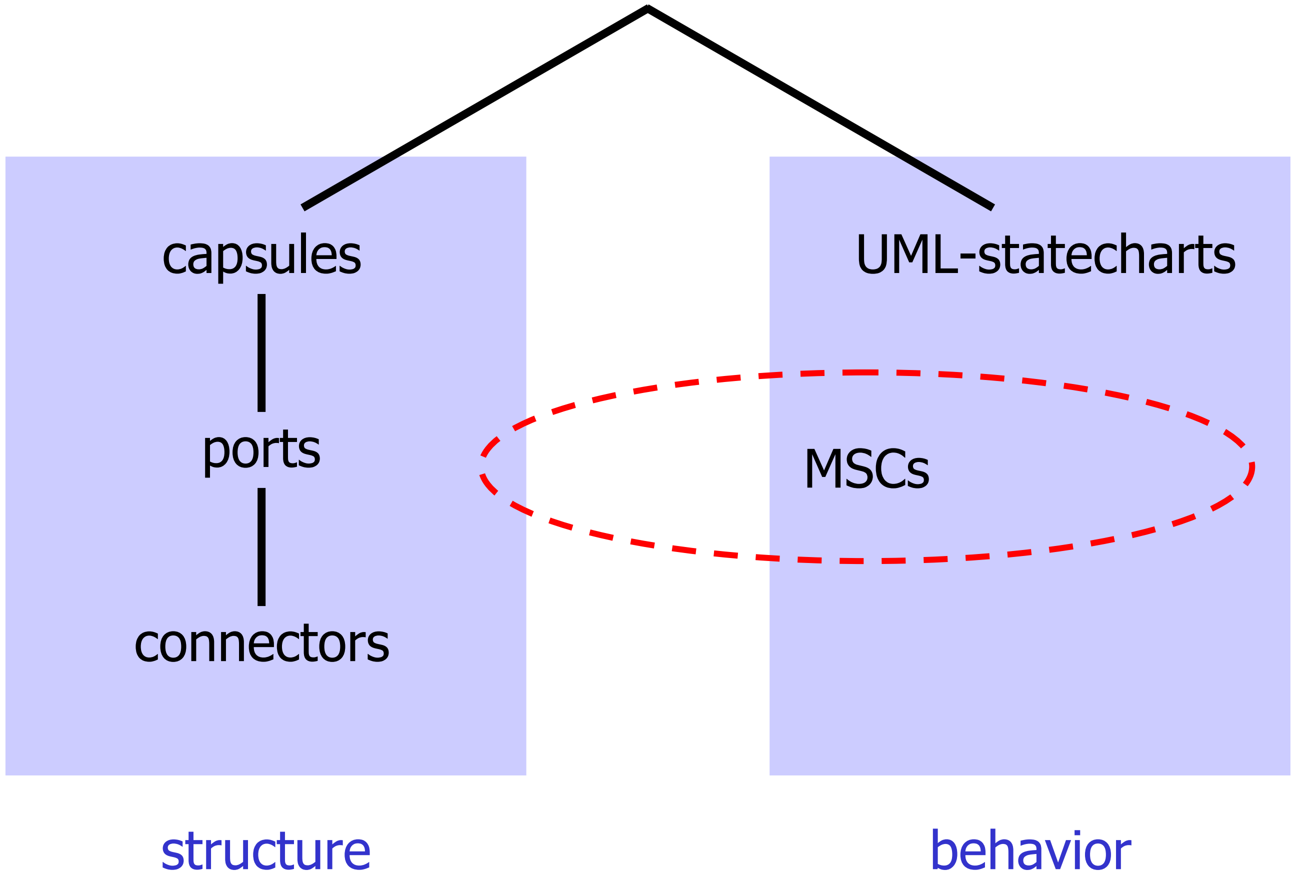
- Examples of “profiles” of the UML:
  - embedded real-time systems (“UML-RT”)
  - automotive
  - web applications
  - ...

Read: UML with component notion

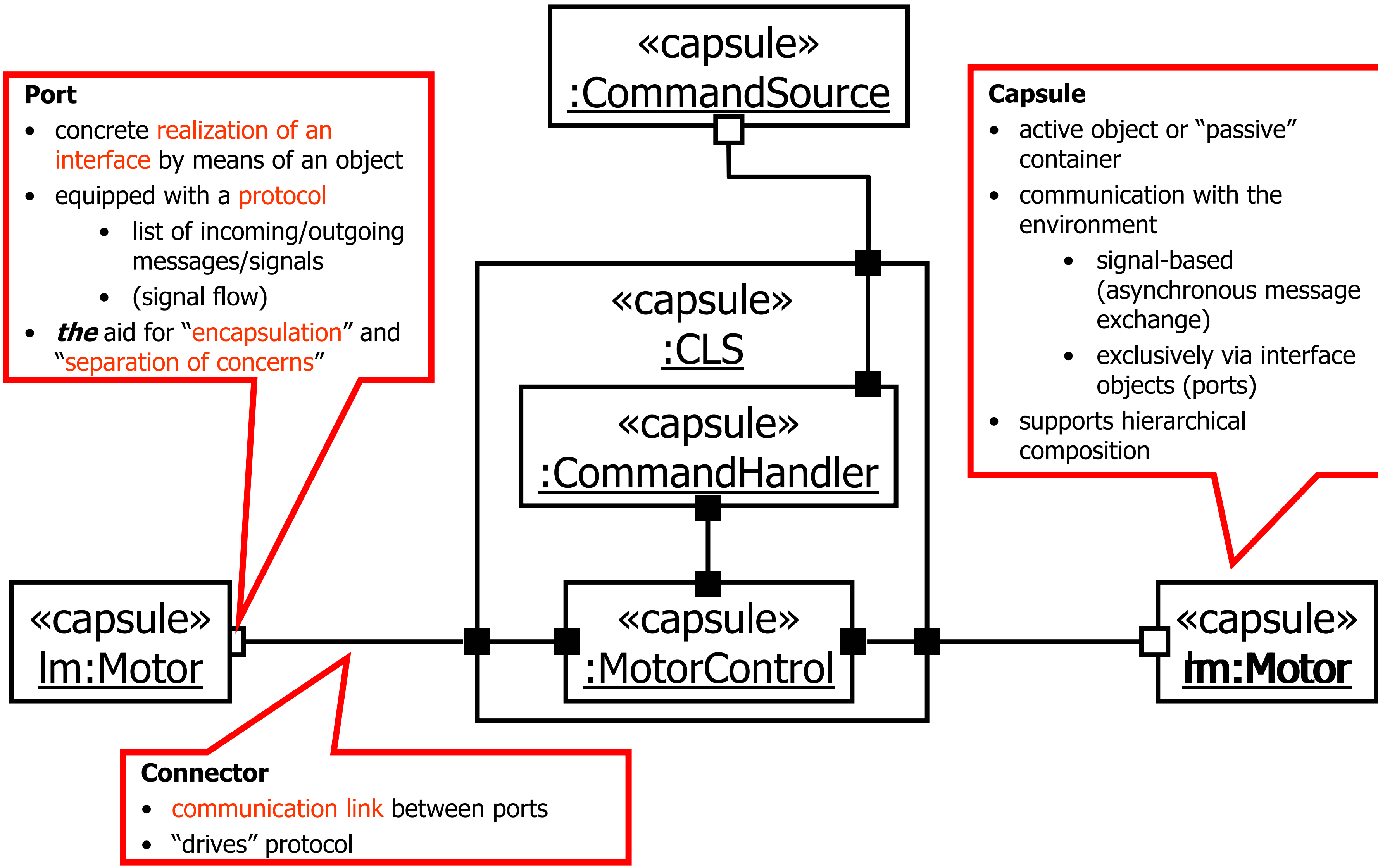
- Origin: ROOM [SGW94] + UML

- Focus of UML-RT/ROOM:
  - component-oriented development
  - all components are potentially active units
  - signal-/message-oriented communication
  - time concept
  - quality of service (in preparation)

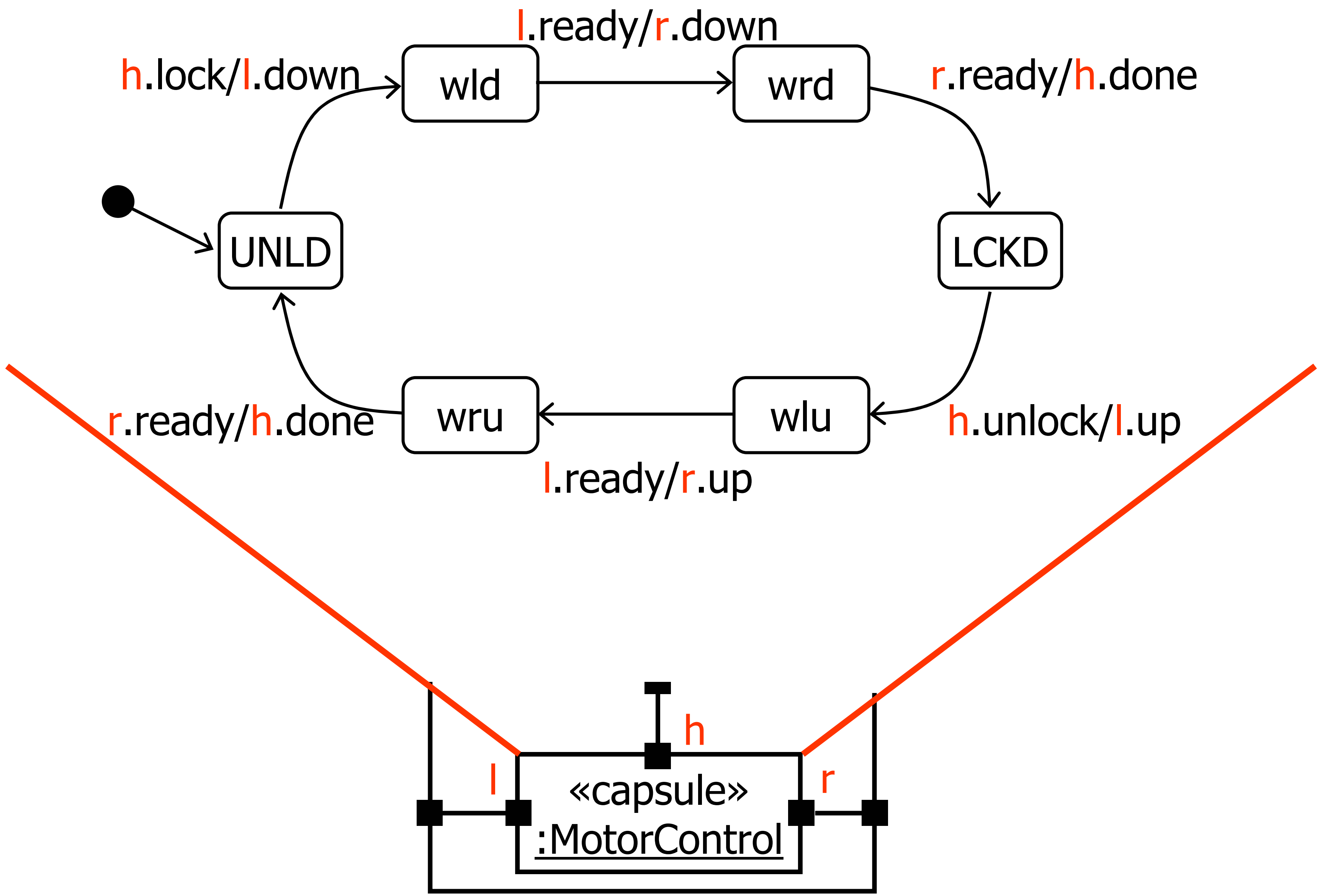
Description Techniques of UML-RT for Structure and Behavior



Hierarchical Composition in UML-RT

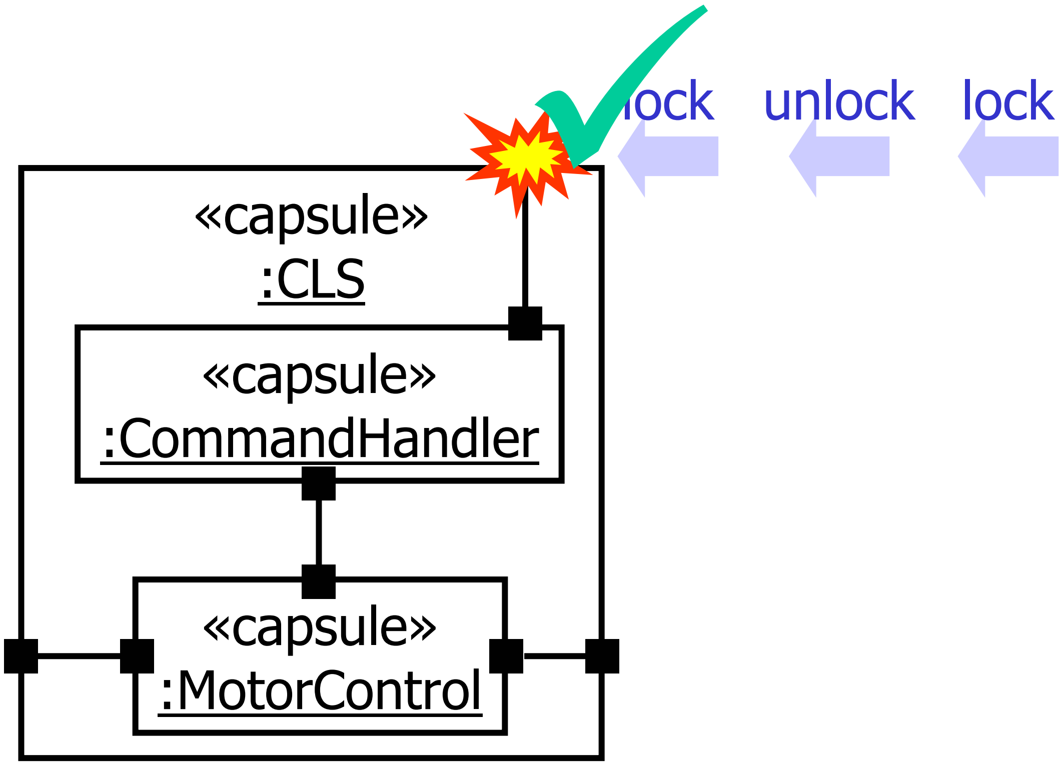


Example: UML-statecharts



Signal-Based Communication

- Capsules receive and send signals via their ports
- Signals, which cannot be processed immediately, are stored in a queue



Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

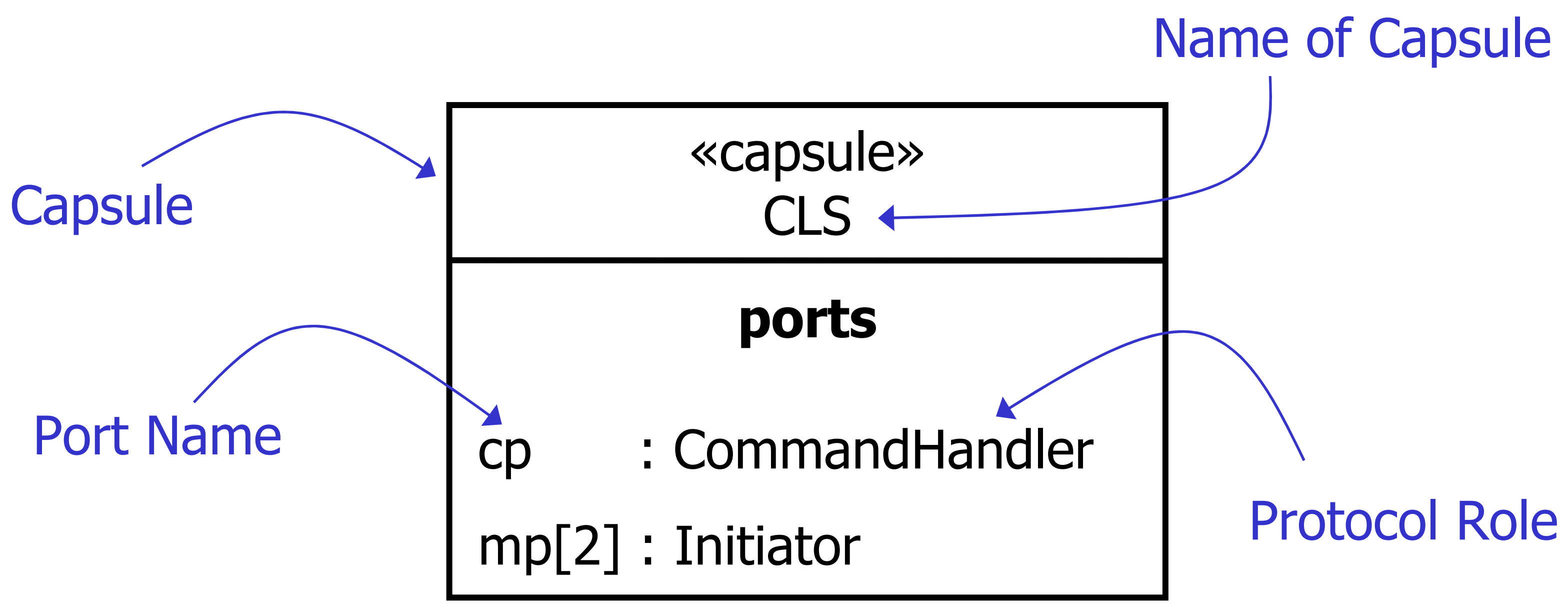
Capsules

- Every capsule represents a potentially active object
- Communication between capsule and environment: exclusively via ports
  - no public data
  - no public methods
- Hierarchical decomposition into sub-capsules
- Every capsule has (at most) one state automaton describing the capsule's behavior
  - $\Rightarrow$ capsule is "controller" for its sub-capsules
  - $\Rightarrow$ see architectural pattern "recursive control"

Capsules

- Upon its instantiation a capsule builds its internal structure (sub-capsules)

- The capsule can change its internal structure over time
  ⇒ Architectural integrity



Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

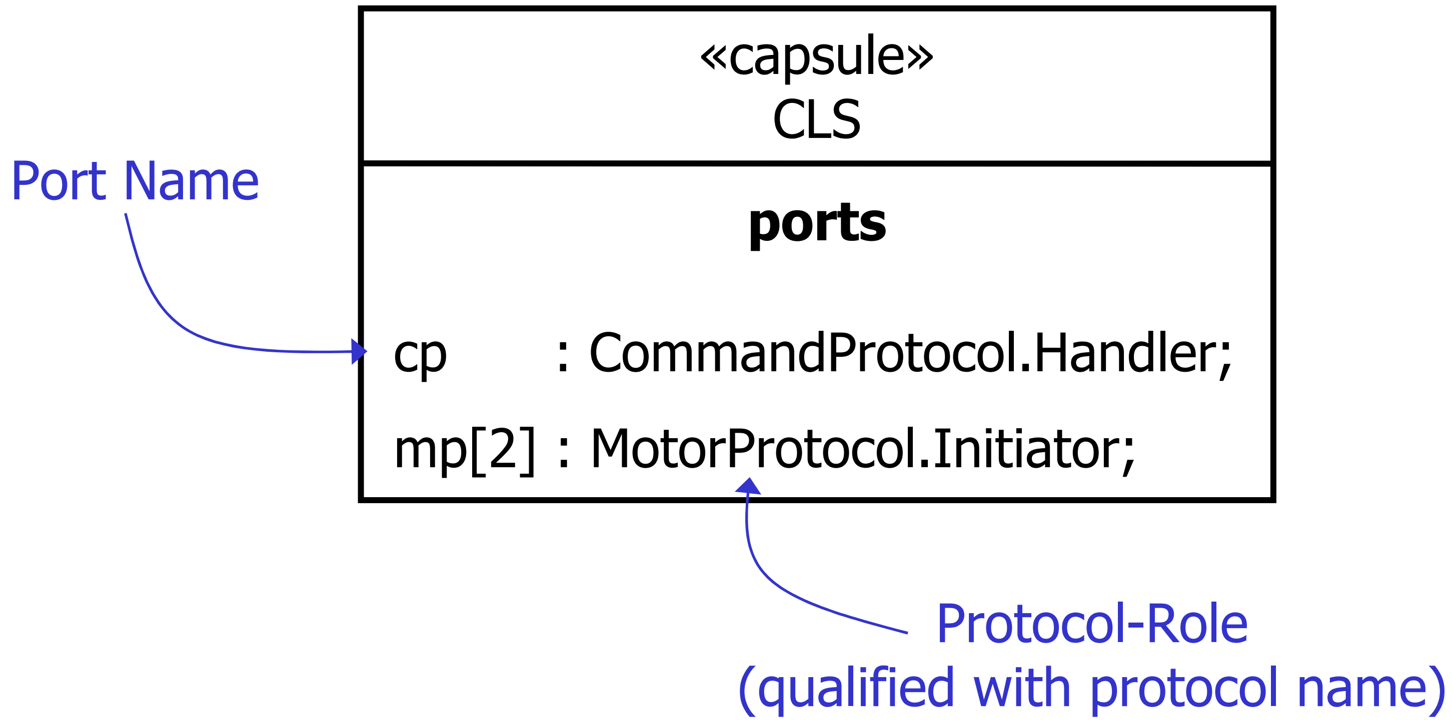
Ports and Connectors

- Port
  - belongs to precisely one capsule
    (the capsule creates and destroys its ports)
  - has identity and state
  - has behavior
  - implements the role of its capsule in a protocol

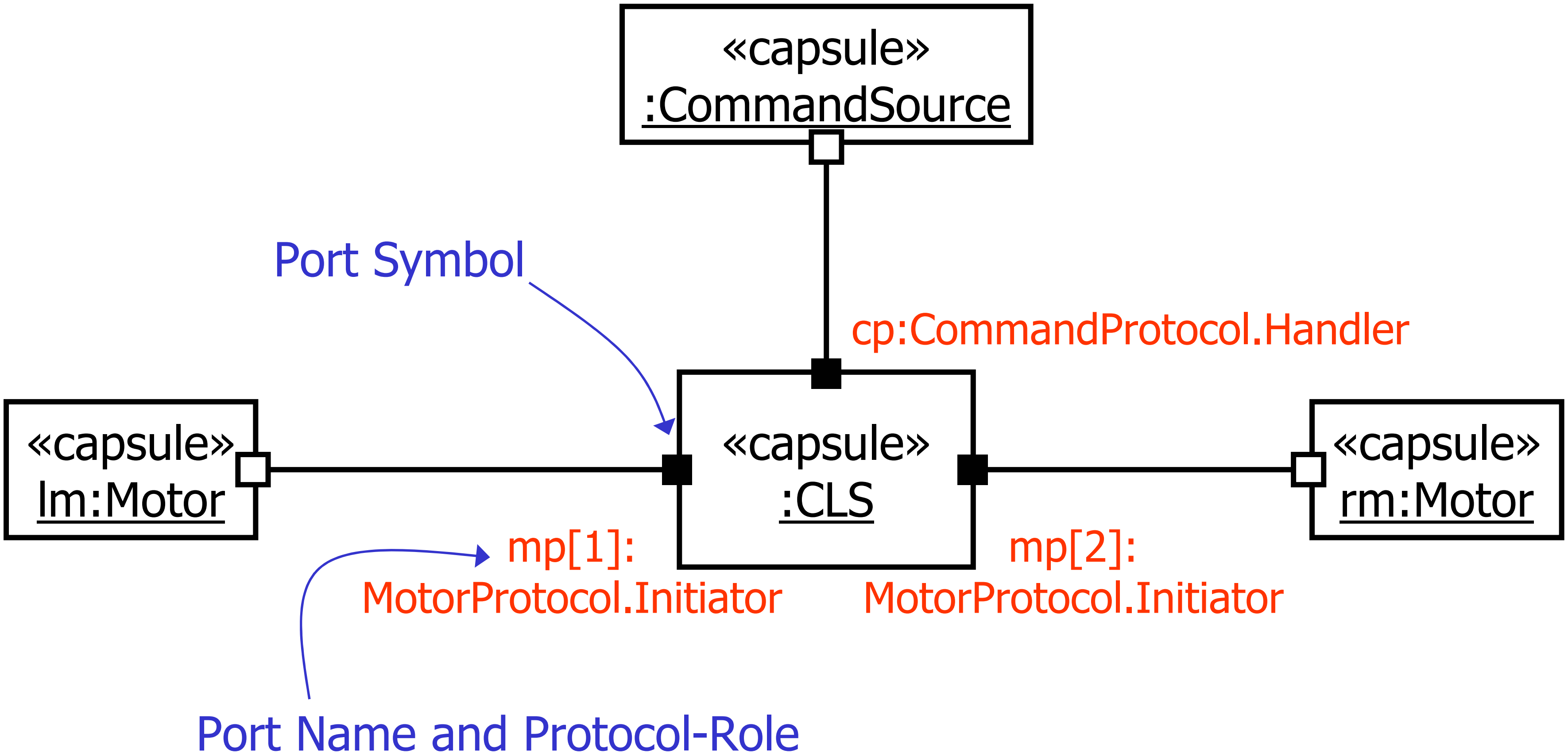
- Kinds of ports
  - Relay-Ports
    - relay signals between capsules and their sub-capsules
    - controlled interface export
  - End-Ports
    - relay signals between capsules and their state automata
    - have queues for signals already received, but not yet processed

Ports and Connectors

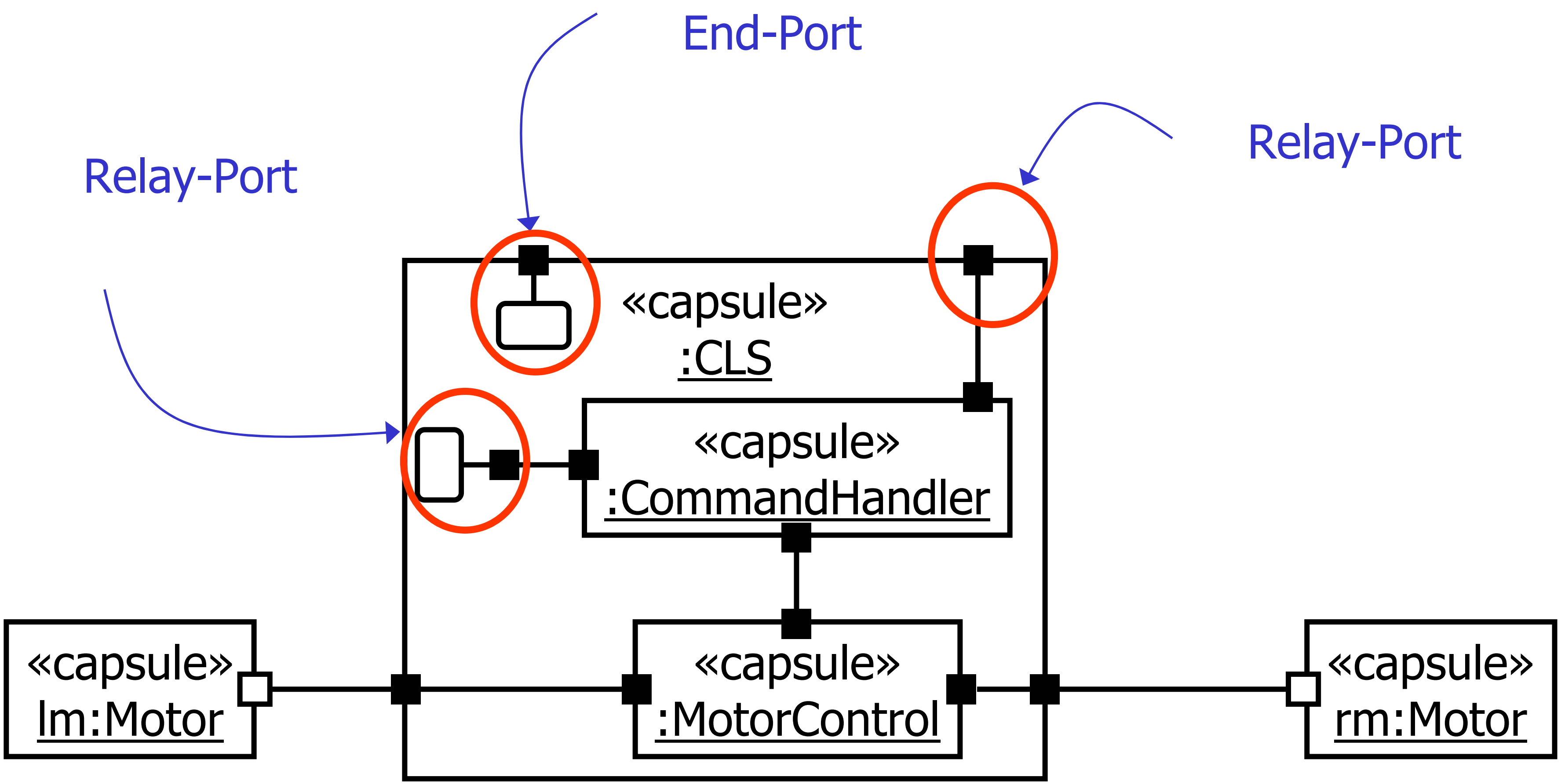
Simplified Representation



Simplified Representation in Collaboration Diagrams



Ports and Connectors



Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

Example: Simple Communication Protocol

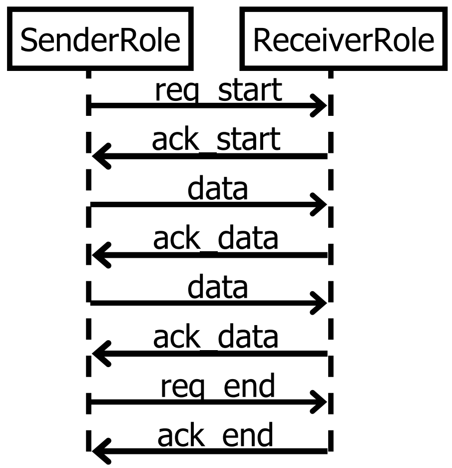
«protocolRole»
SenderRole

incoming

ack_start
ack_data
ack_end

outgoing

req_start
data
req_end



«protocolRole»
ReceiverRole

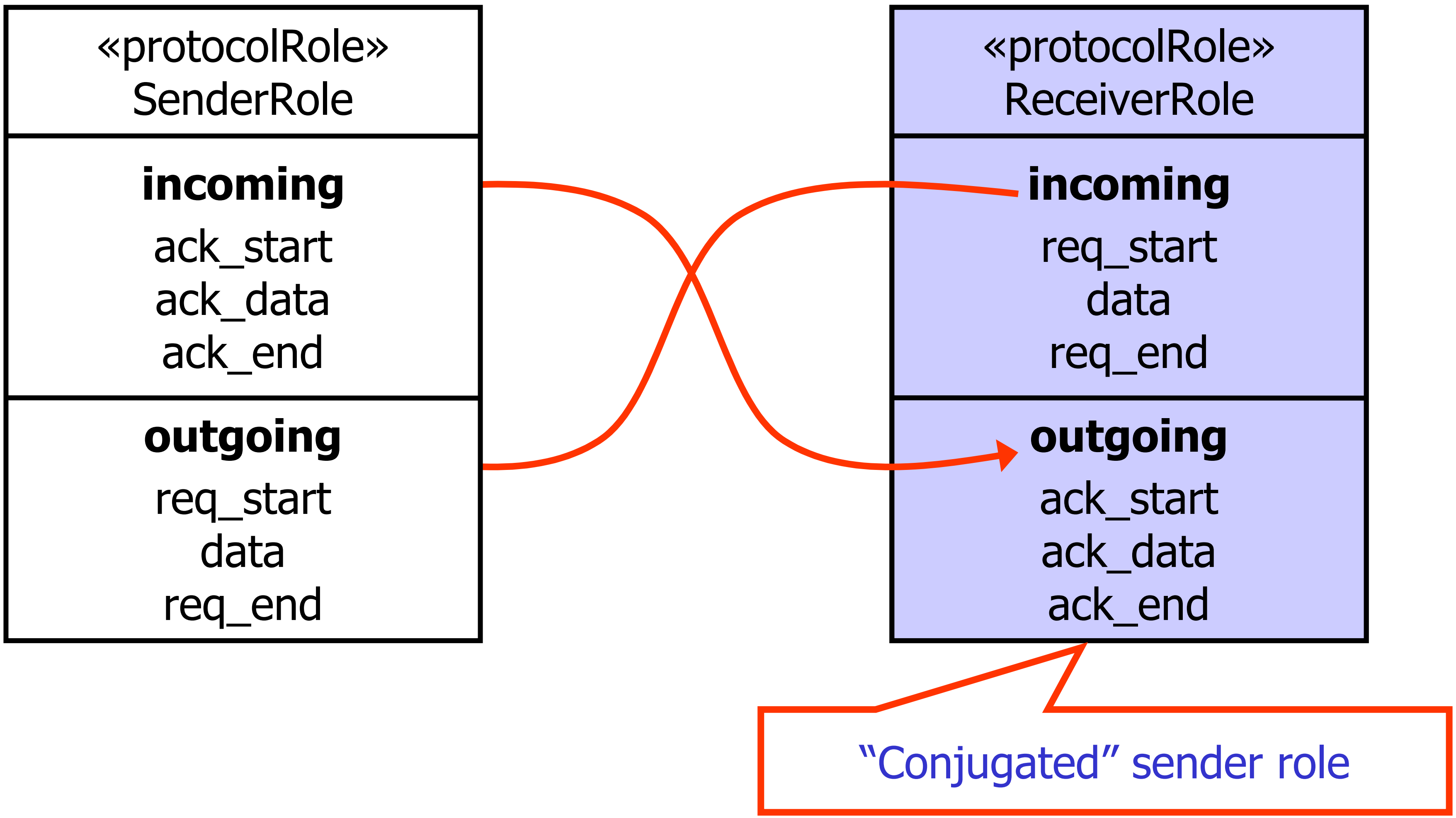
incoming

req_start
data
req_end

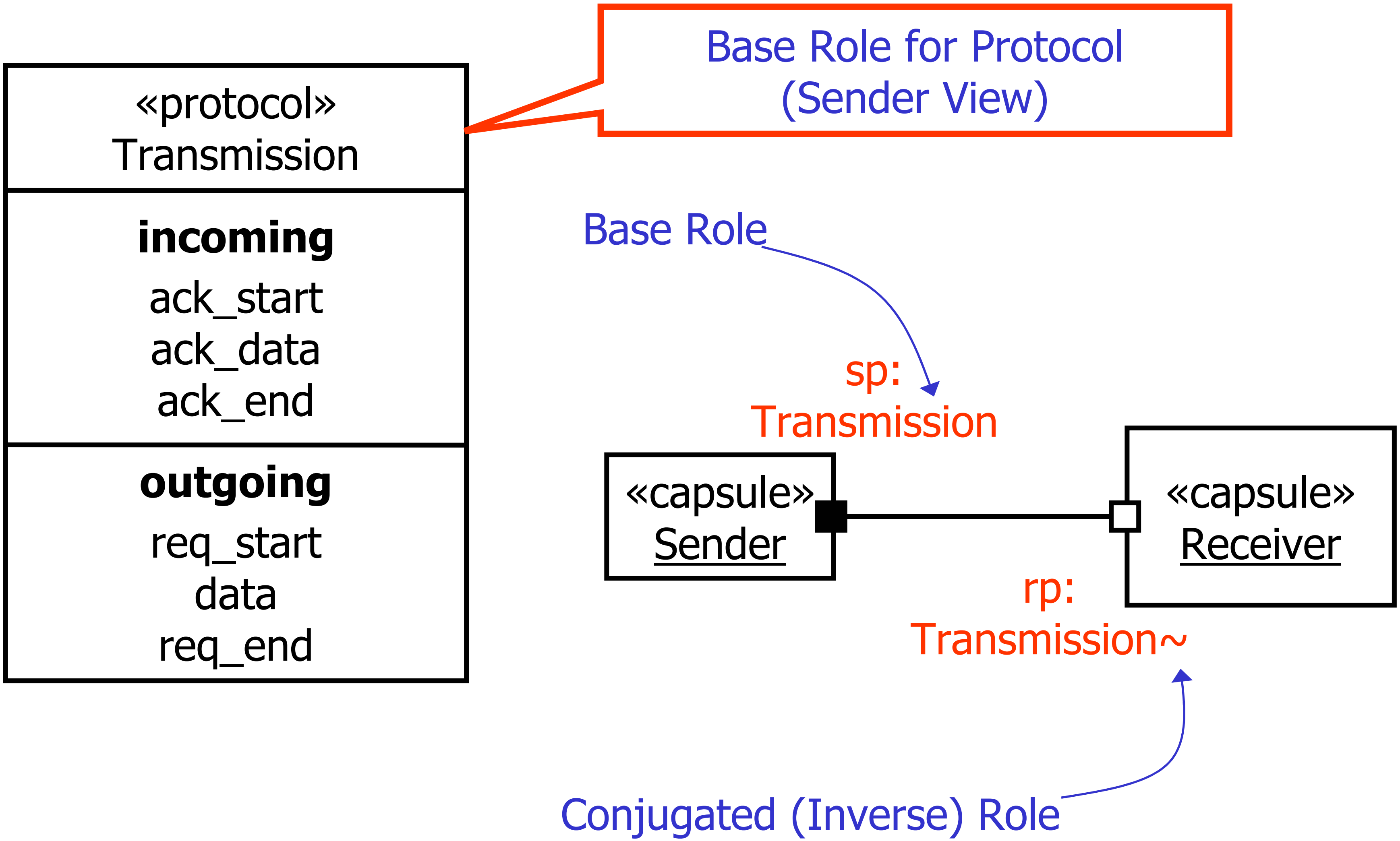
outgoing

ack_start
ack_data
ack_end

Example: Simple Communication Protocol



Simplification for Point-to-Point Protocols



Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

Behavior Description in UML-RT

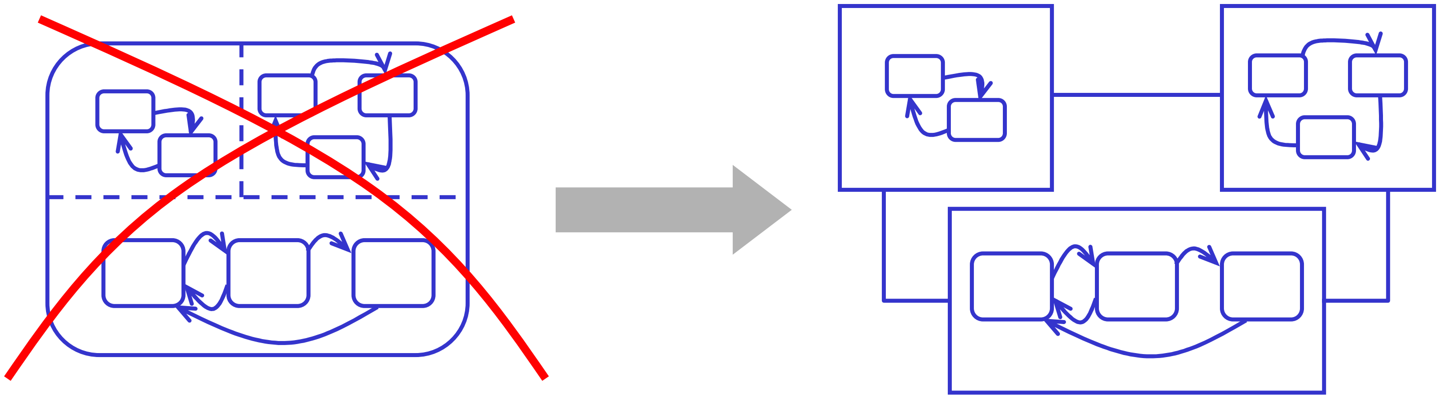
- Every capsule that has its own behavior is associated with a UML-statechart

- Max one statechart per capsule

- Hierarchical composition:
  - every sub-capsule can have its own statechart

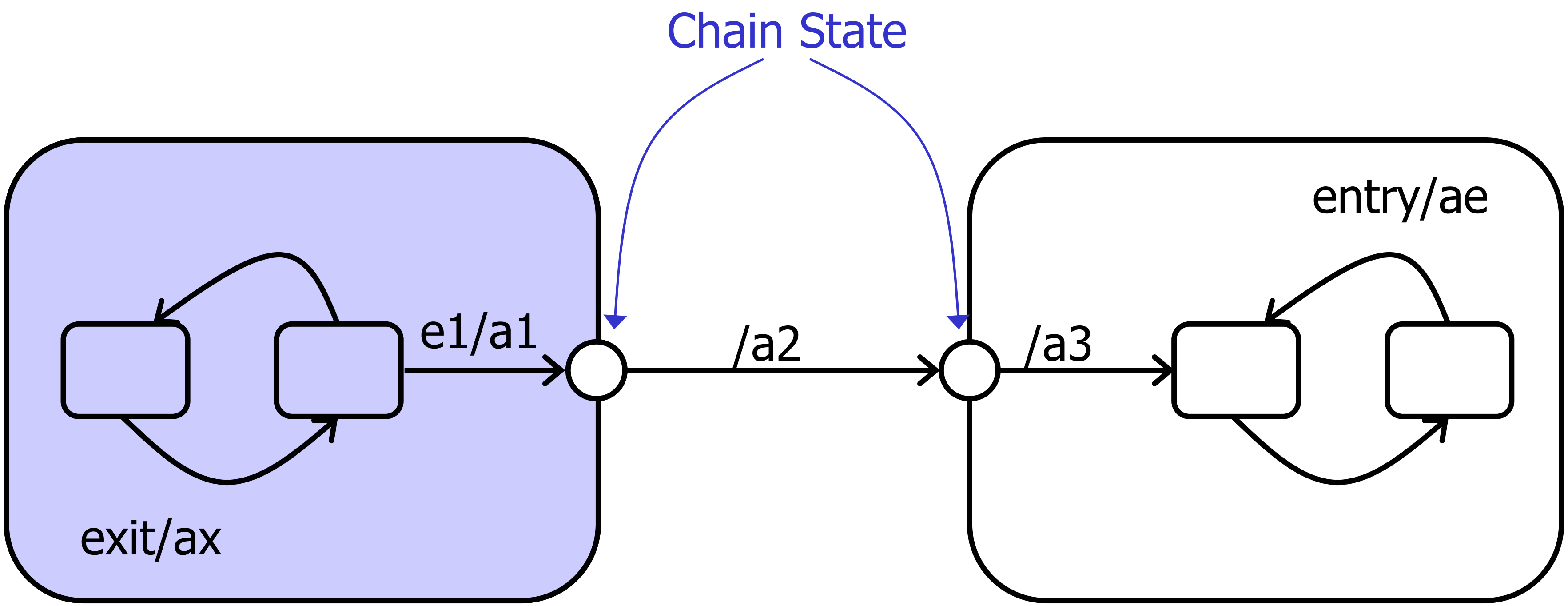
Doing without AND-states

- Concurrency via separate capsules
- Synchronization via explicit communication
- Result: stronger decoupling



Behavior Description in UML-RT

Encapsulation on the Level of States



- States become exchangeable entities
- Helps avoid “stub states”

Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

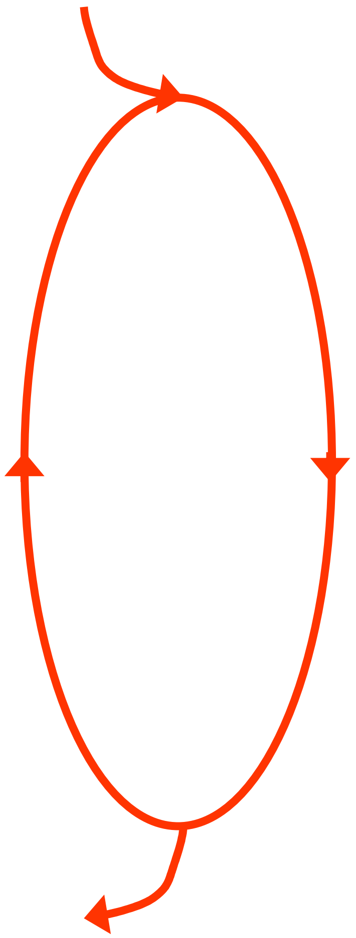
Evaluation

- UML-RT is much better suited for the specification of software architectures and services than "pure" UML:
  - hierarchic component model, precise behavior descriptions
  - interface concept
  - protocols and connectors
- Potentials for improvement (among others):
  - m2m communication instead of p2p
  - association of interaction patterns with ports/connectors
  - methodological guidelines for iterative service development
- Future:
  - (methodological!) treatment of Quality-of-Service aspects

Overview

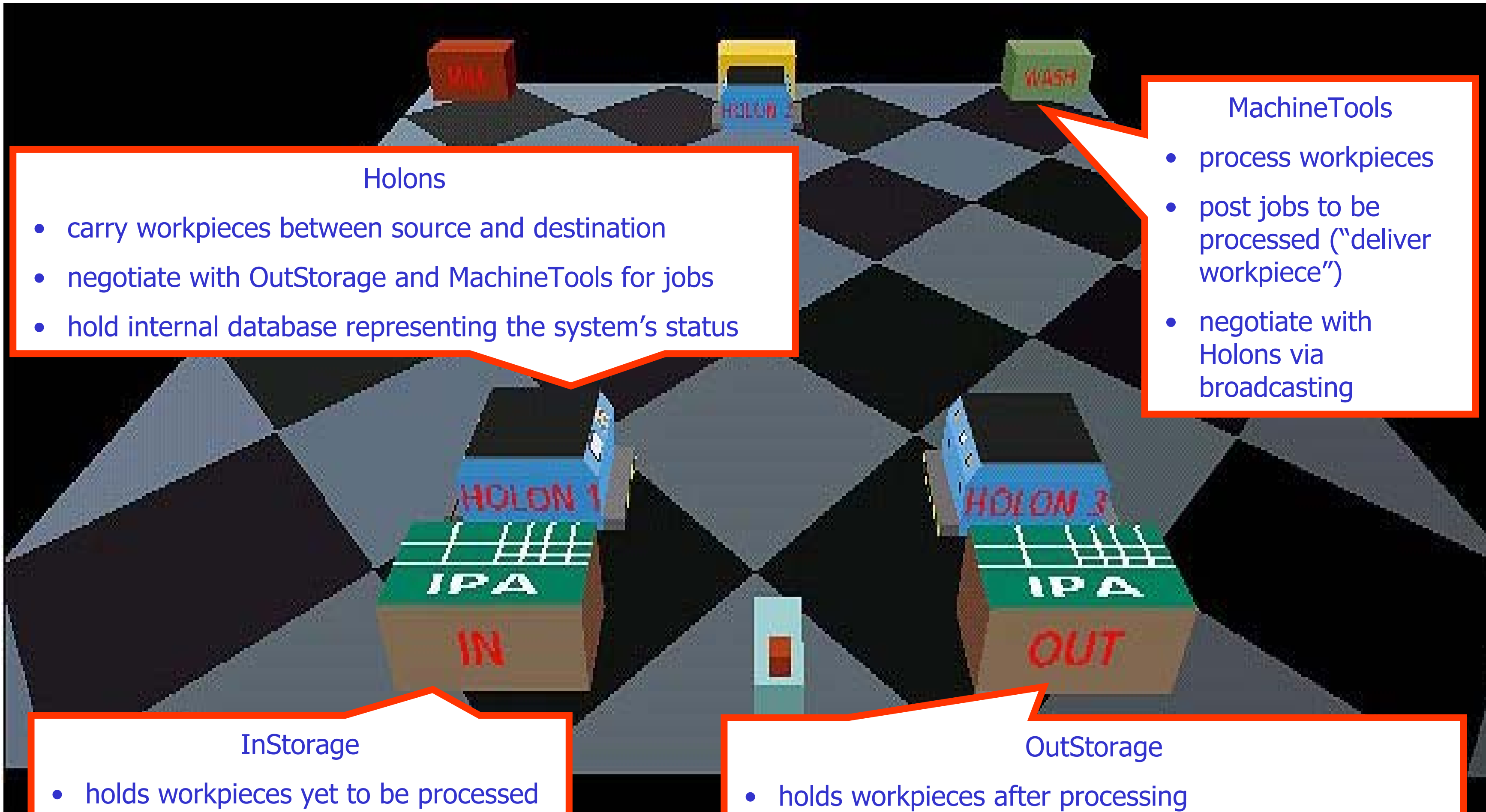
- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

Systematic Construction of Reliable SW-Systems



1. Develop/Refine domain model
2. Capture interaction patterns
3. Derive interface specification
   - messages/signals, types
   - behavior
4. Decompose components hierarchically

Example Application: Autonomous Transport System



Holons

- carry workpieces between source and destination
- negotiate with OutStorage and MachineTools for jobs
- hold internal database representing the system's status

MachineTools

- process workpieces
- post jobs to be processed ("deliver workpiece")
- negotiate with Holons via broadcasting

InStorage

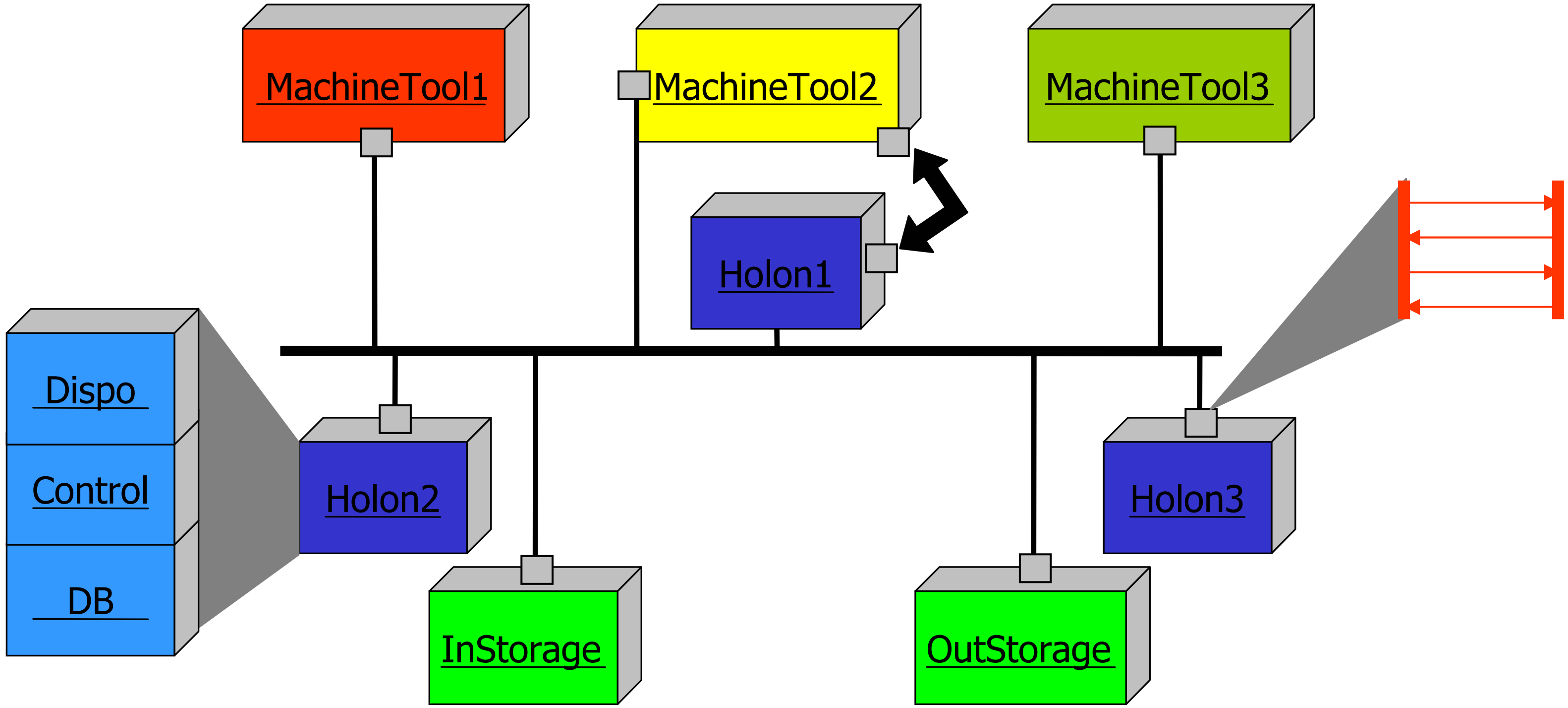
- holds workpieces yet to be processed

OutStorage

- holds workpieces after processing
- holds "production plan/program" (number/kind of workpieces to be processed per day)

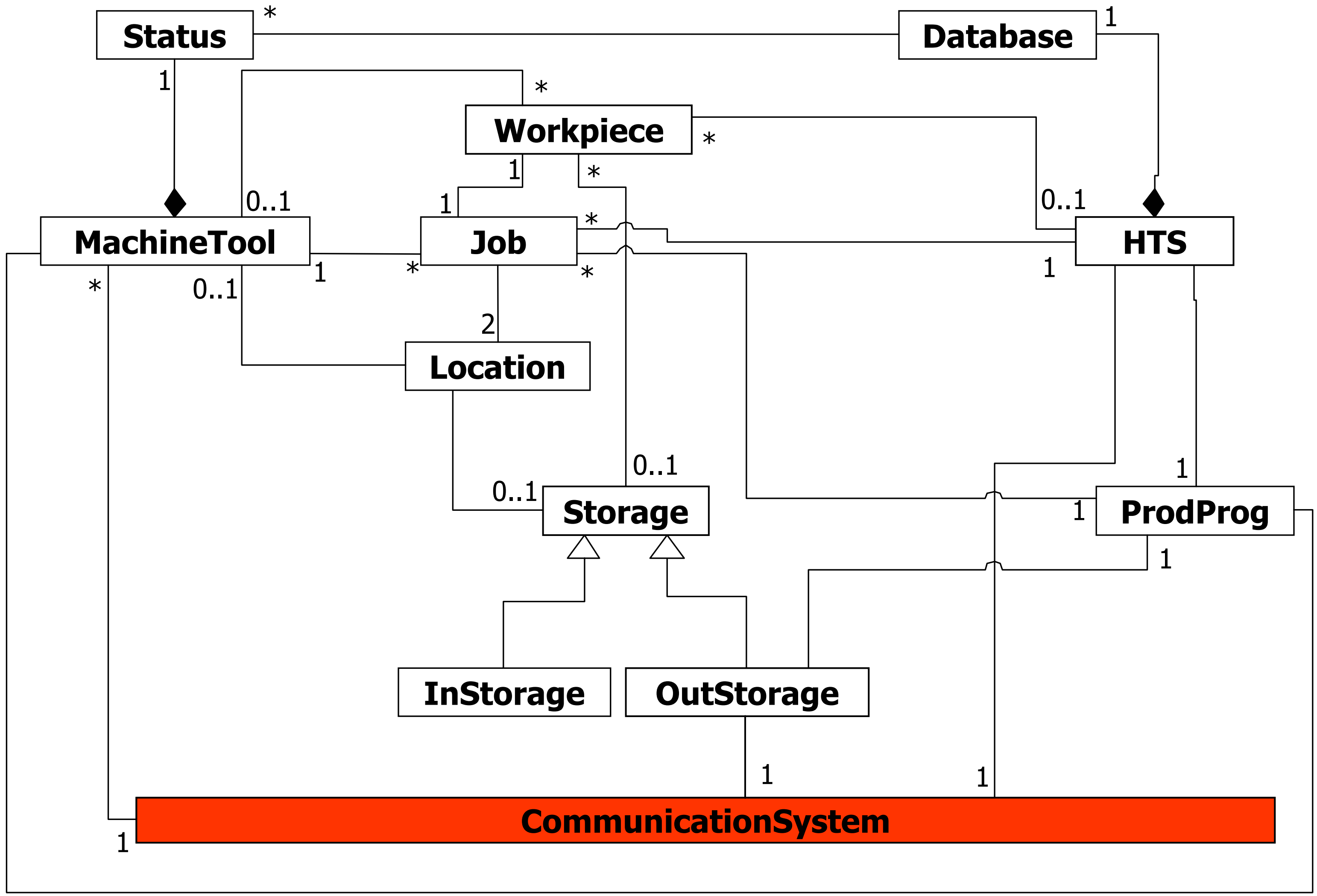
Example Application: Autonomous Transport System

Architectural Aspects:

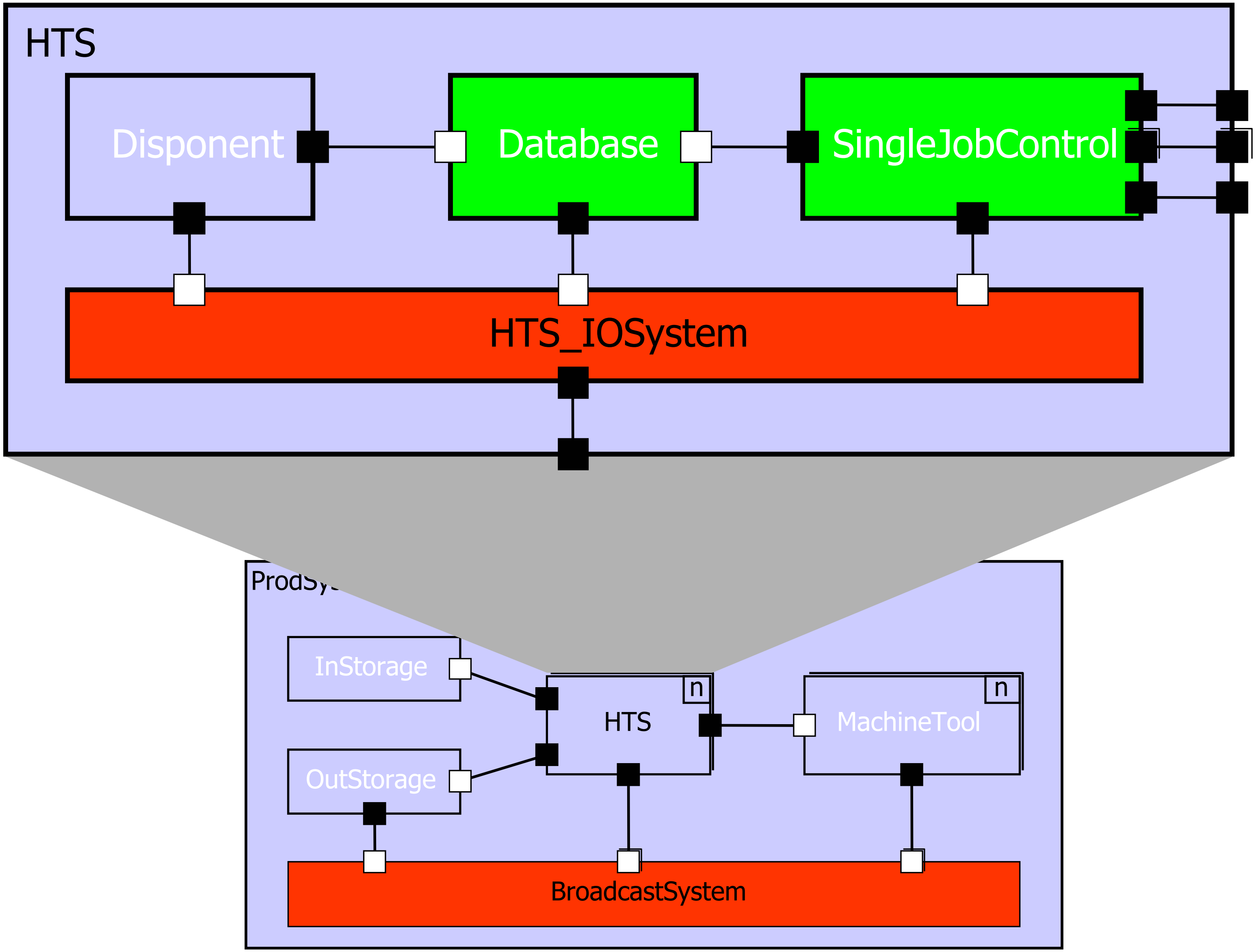


- components
- interfaces/behavior
- hierarchy/decomposition
- p2p communication
- broadcasting

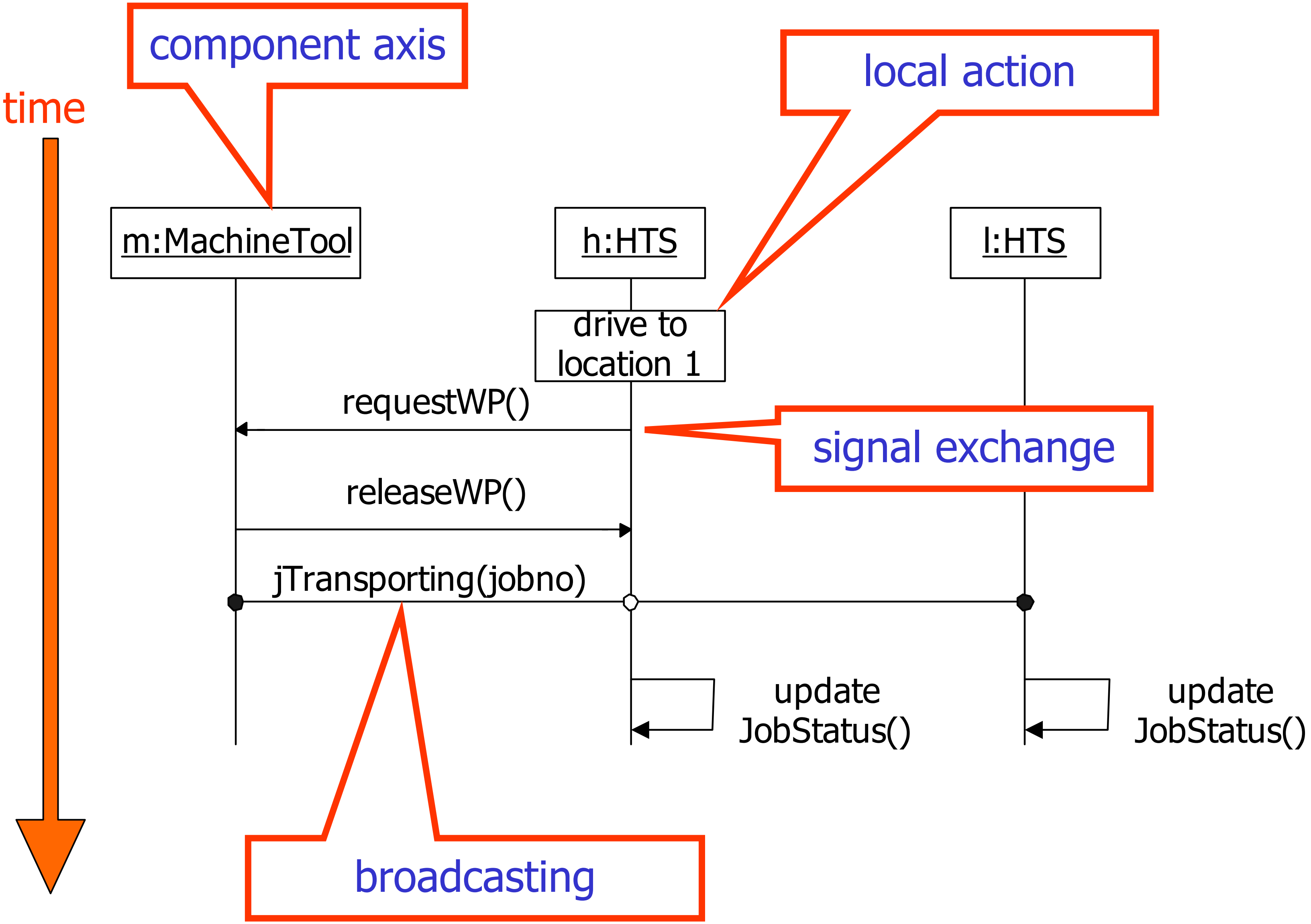
Domain Model



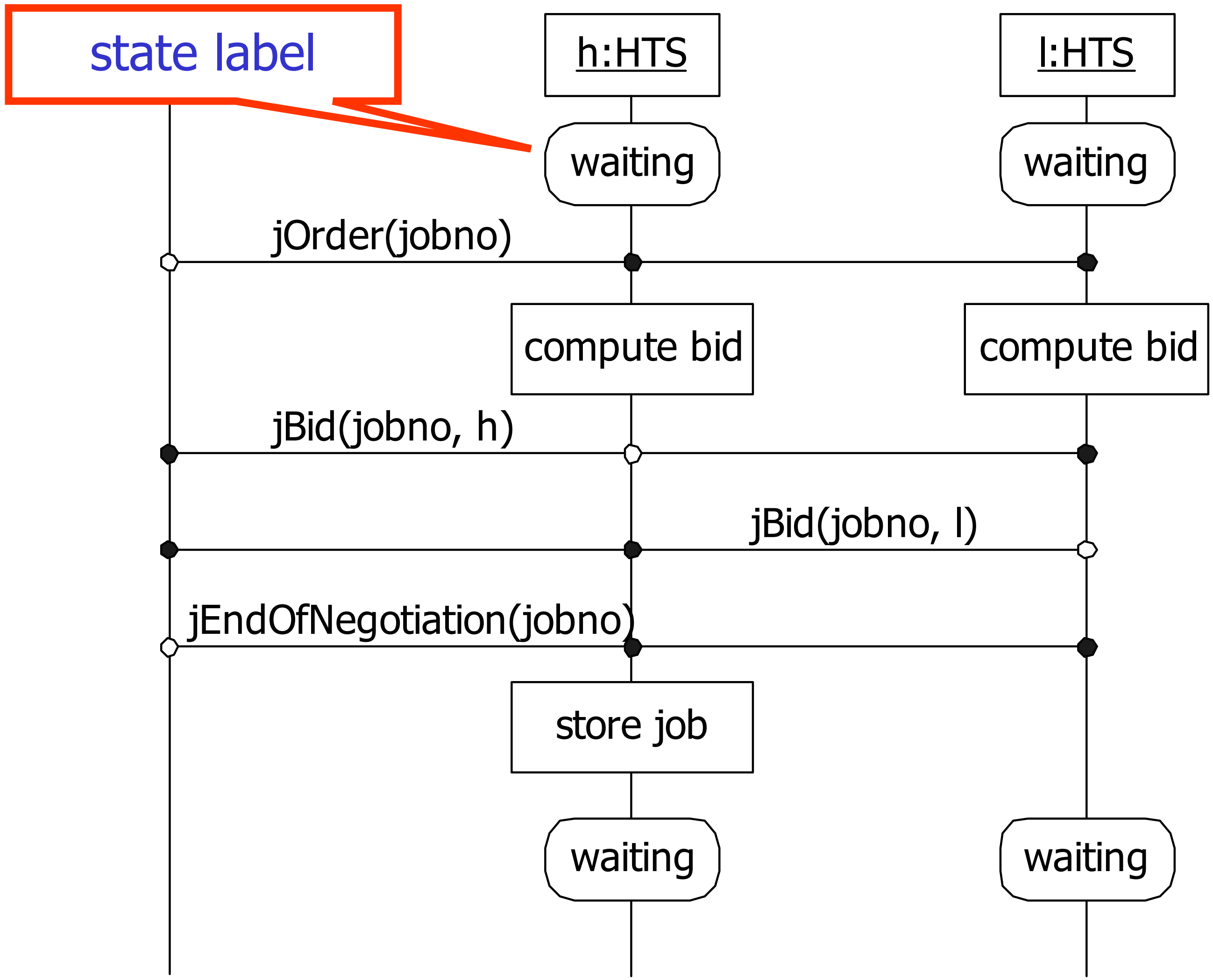
Architectural Pattern for Broadcasting



Sequence Charts for Broadcasting



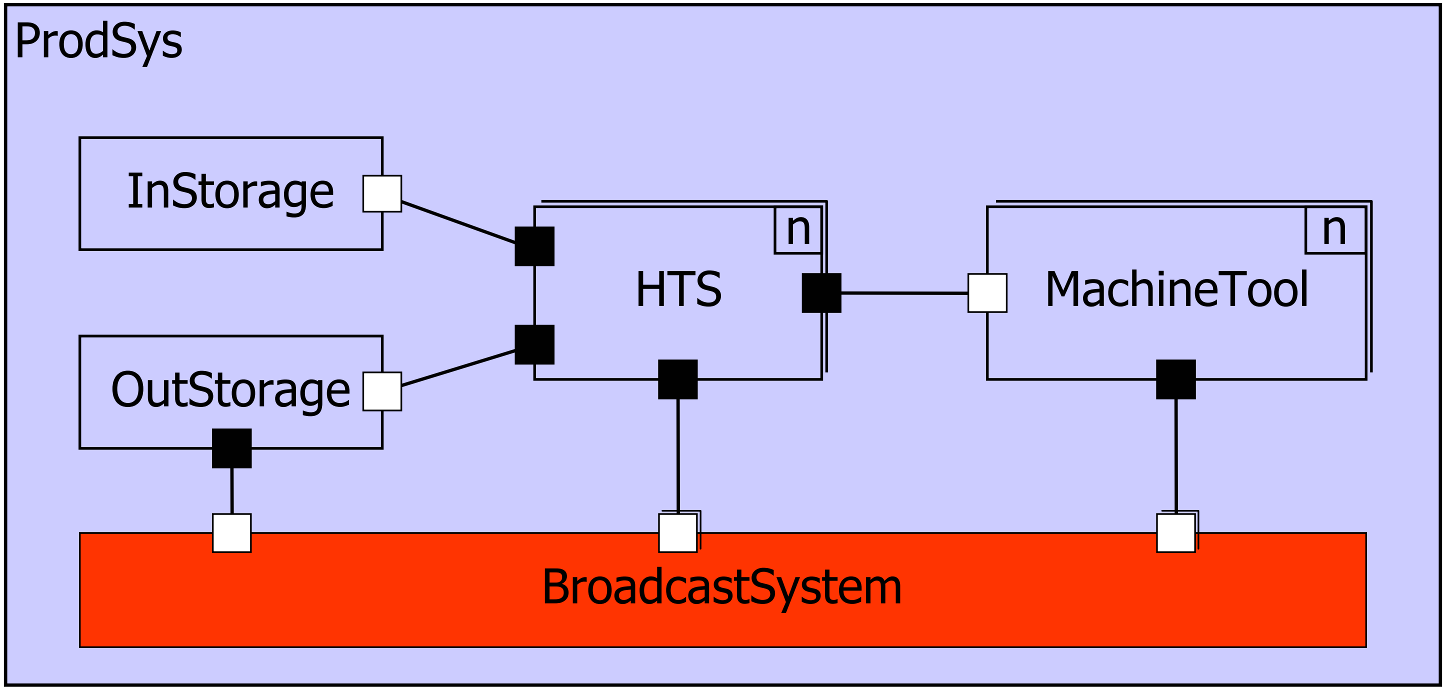
Sequence Charts for Broadcasting



Derivation of Component Structure

Captured scenarios & domain model indicate:

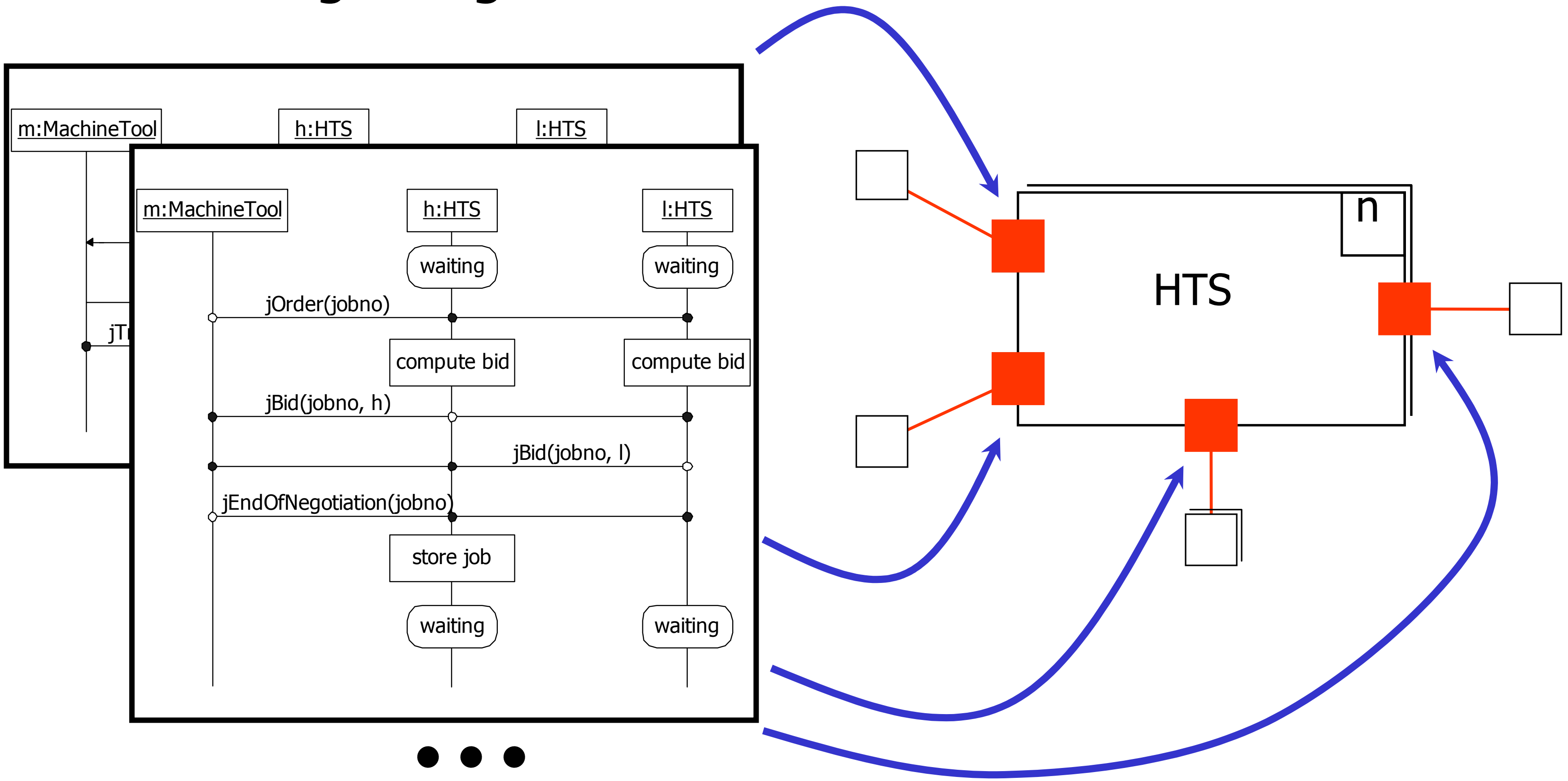
- active vs. passive components
- Point-to-point communication requirements
- broadcasting requirements



Derivation of Interface Behavior

Captured scenarios indicate also:

- names and types of signals
- ordering of signal flow



Overview

- UML: Good Enough for Specifying Architectures?
- UML-RT/UML 2.0
  - Overview
  - Capsules
  - Ports and Connectors
  - Protocols
  - Behavior Description
  - Evaluation
- Example: Autononomous Transport System
- Summary and Outlook

Summary and Outlook

- Modeling “in the real world”:

  often – if at all – done using UML/UML-RT/UML 2.0

- UML-RT/UML 2.0 better equipped for modeling software architectures than UML versions < 2.0

- Starting point for component- and service-oriented development: domain model, interaction scenarios

- How to avoid over-modeling and over-engineering?