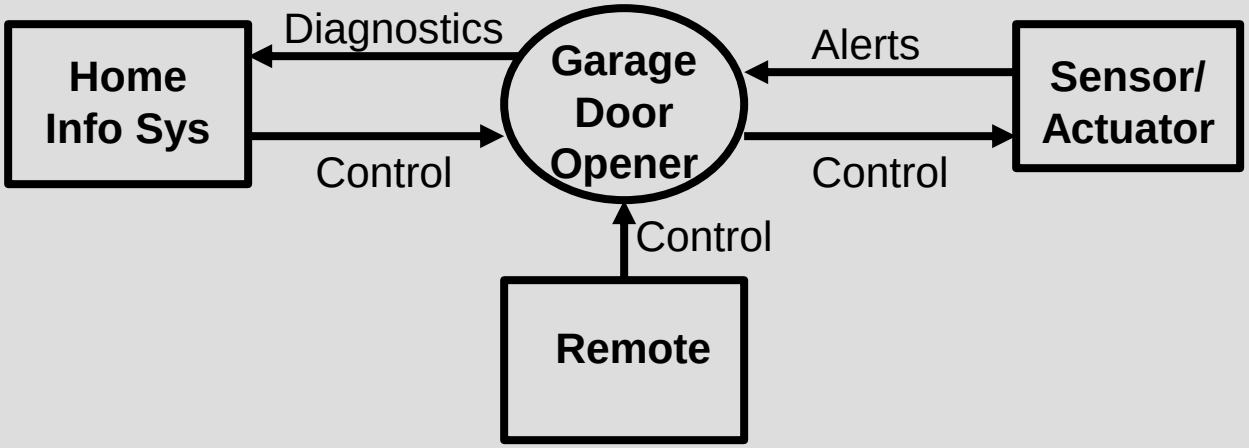
Software Architecture Design Example

Using Attribute Driven Design

Garage Door Example

- Design a product line architecture for a garage door opener integrated with a home information system
  - Raise/lower door via switch, remote, home info system
  - Problem diagnosis from home information system



Step 1: Choose a System Element to Design

- For **new** (green field) systems it is the **whole system**
- For **legacy**, what is being **added**
- After the first iteration what comes next, **element breath or depth?**
  - Depth if technology risk or resourcing concerns
- Garage door opener is the system

Step 2: Identify the ASRs
(Architecturally Significant Requirements)

- Start with quality **scenarios**
  - Device and controls differ for various products in product line
  - Product processors differ
  - Garage door descent must stop within 0.1 second after obstacle detection
  - Access to opener from home info system for control and diagnostics with proprietary protocol

Step 2: Identify the ASRs (cont)

- ASRs are a combination of **functional requirements, constraints and quality attributes**
- **Prioritize ASRs** and select those that will "drive" the architecture design

Garage door system:

- Real-time performance
- Modifiability to support the product line
- Interoperability for on-line control and diagnostics

Step 3: Generate a Design Solution For the Chosen Element

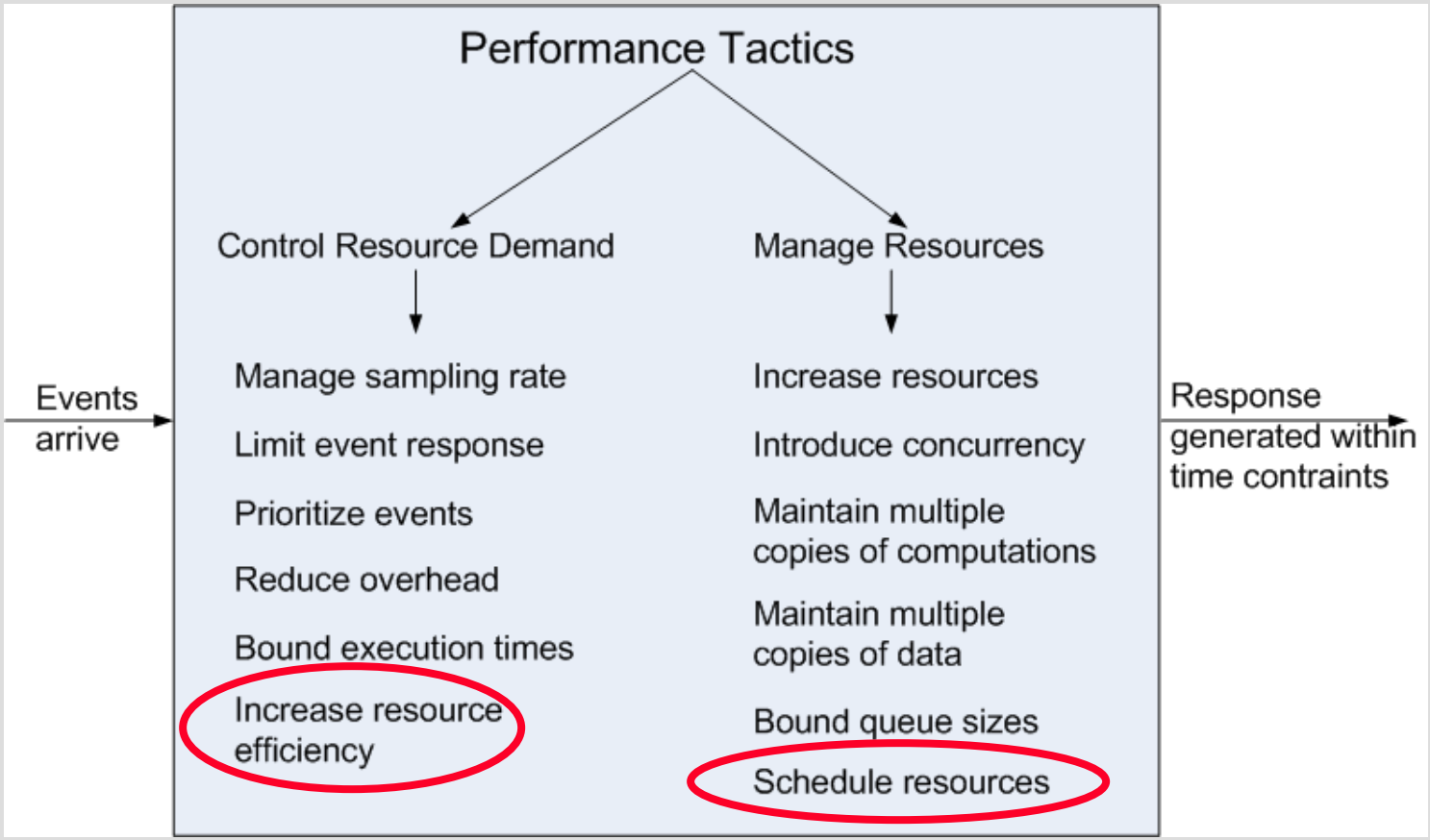
- Goal: establish an **overall architecture design** that satisfies architectural drivers
- **For each ASR** for this element choose a design solution …
- The patterns, tactics, design principles to achieve quality attributes
- Watch for QA design tradeoffs between tactics

It's possible the domain problem may call for a "custom" architecture pattern

Step 3: Generate a Design Solution (cont)

- Performance
  - Concerned with critical computational performance scheduling and efficiency
  - Need tactics to deal with the **control of resource demand and resource management**
  - Choose "**increase resource efficiency**" and "**schedule resources**"
  - Solution - separate critical and non-critical performance computation

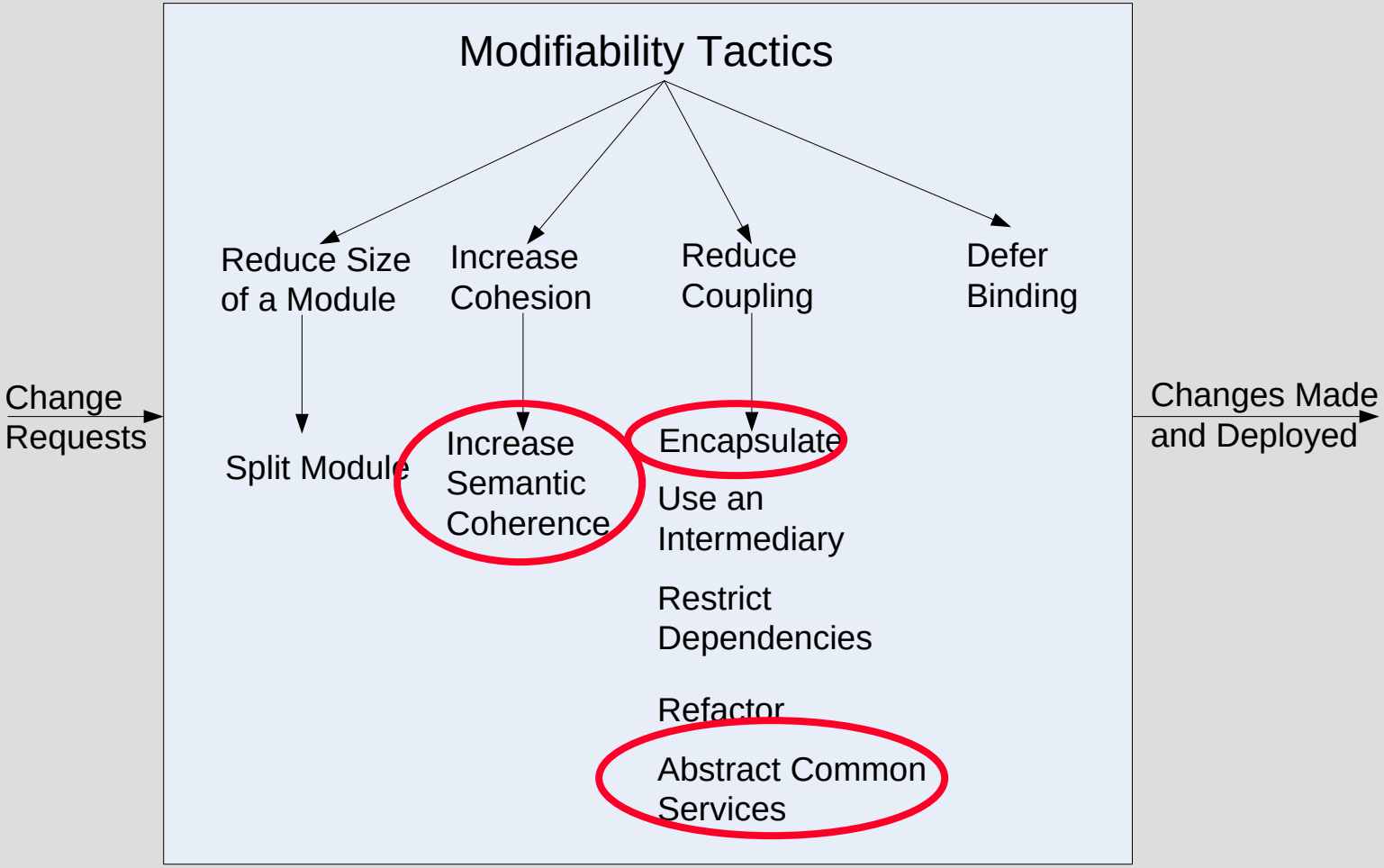
Performance Tactics



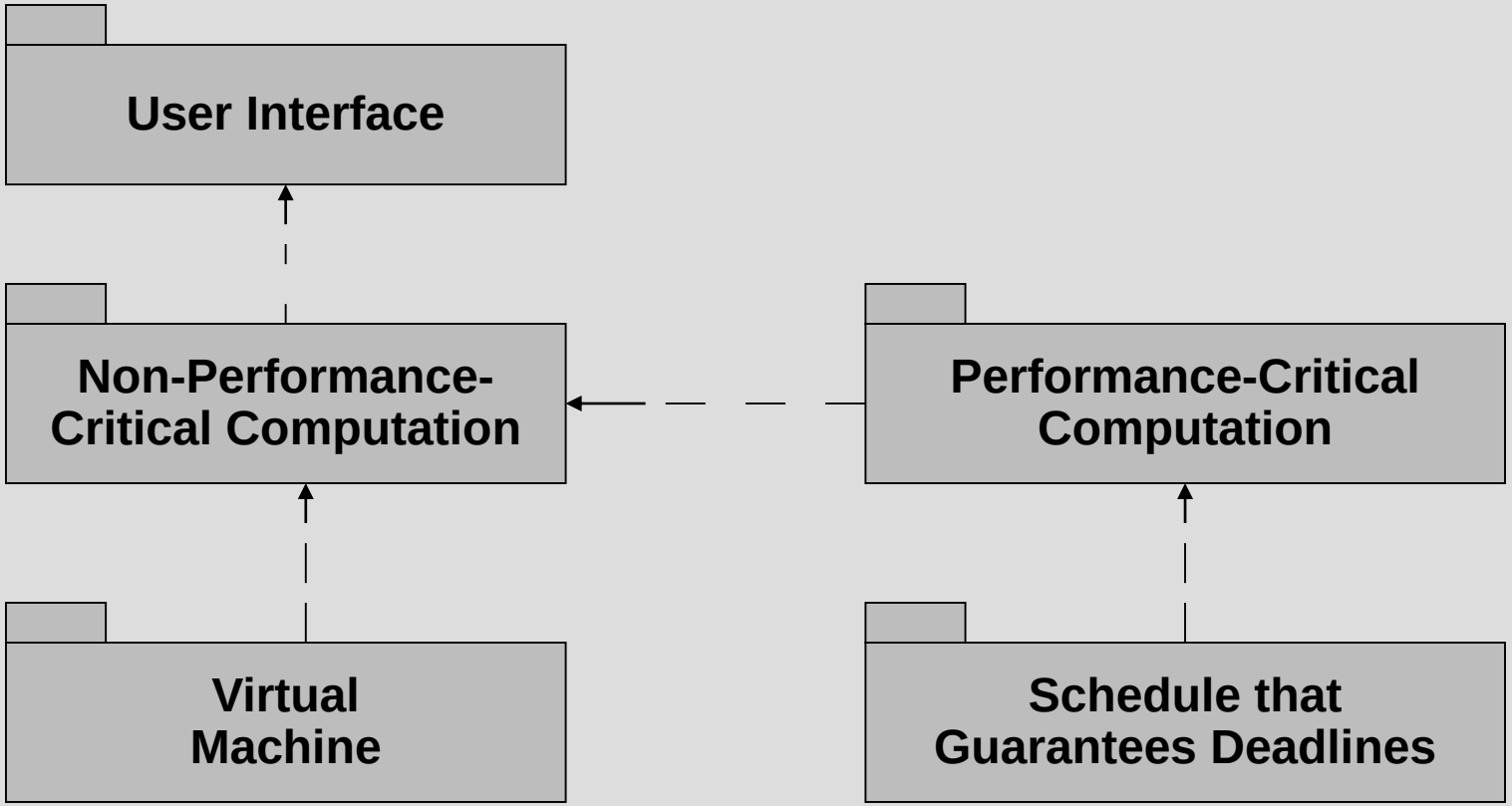
Step 3: Generate a Design Solution (cont)

- Modifiability
  - Primarily concerned with **changes at build time**, not runtime
  - Need tactics to support **separation of responsibilities to localize changes**
    - Increase cohesion, reduce coupling
  - Choose **"increase semantic coherence", "encapsulation", and "abstract common services"** as our tactics
  - Solution - separate responsibilities dealing with the user interface, communication, and sensors into their own modules

Modifiability Tactics



Pattern for Garage Door Opener



Step 4: Validate Design and Refine Requirements

Test the element design for requirements satisfaction

Requirements satisfied	Done, no more refinement
Requirements not fully satisfied	•Defer to the next iteration •Delegate or distribute requirement satisfaction to sub-module elements
Requirements cannot be satisfied with this design	•Revisit the design - backtrack •Refine or push back on the requirement

Step 4: Validate Design and Refine Requirements (cont)

ASRs Not Met	Action
Quality attribute	•Apply tactics to address tradeoff or downside
Functional responsibility	•Add responsibilities to existing module •Create new module
Constraint	•Modify the design •Relax the constraint

Note: Previous designs become a constraint

Step 5: Repeat Until all ASRs Have Been Satisfied

- If all ASR's satisfied, done – **a workable architecture**
  - Or elaborated sufficiently for construction
  - (or you run out of time and money)
- Otherwise …
- Repeat step 1 - choose the next (sub)element(s) to design
- Repeat steps 2-4
- As necessary refine use cases and QA scenarios as ASRs for the next design iteration

Are the ASR's Satisfied?
Or is the Design Sufficient?

- Device and controls differ for various products in product line
- Product processors differ
- Garage door descent must stop within 0.1 second after obstacle detection
- Access to opener from home info system for control and diagnostics with proprietary protocol

Next Iteration Decomposition

- Define sub-modules, assign functionality
- Two types of virtual machine – sensors/actuators and communications modules
- Non-performance critical functional modules – diagnostics and normal raising/lowering the door modules
- Obstacle detection and halting the door functions assigned to performance critical module
- Connections

Next Iteration Design Decomposition