



CLARAty: Improving Software Reliability for Robotic Space Applications

Coupled Layer Architecture for Robotic Autonomy

Issa A.D. Nesnas

Group Supervisor, Robotic Software Systems
Mobility and Robotics Section

Jet Propulsion Laboratory, California Institute of Technology

In Collaboration with

Ames Research Center
Carnegie Mellon University
University of Minnesota

49th Meeting of the IFIP 10.4 Working Group on Dependable and Fault Tolerant Computing
Tucson, AZ, USA, February 15-19, 2006



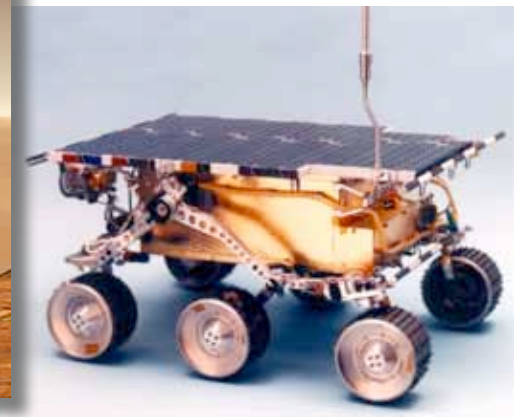
Presentation Overview



- Historical Antecedents
- Application Domains
- Techniques used to Improve Reliability
 - Rigorous Process
 - Software Reuse
 - Formal Validation
 - Continuous Automated Testing
- Challenges for Software Reuse and Interoperability
- Challenges in Technology Infusion

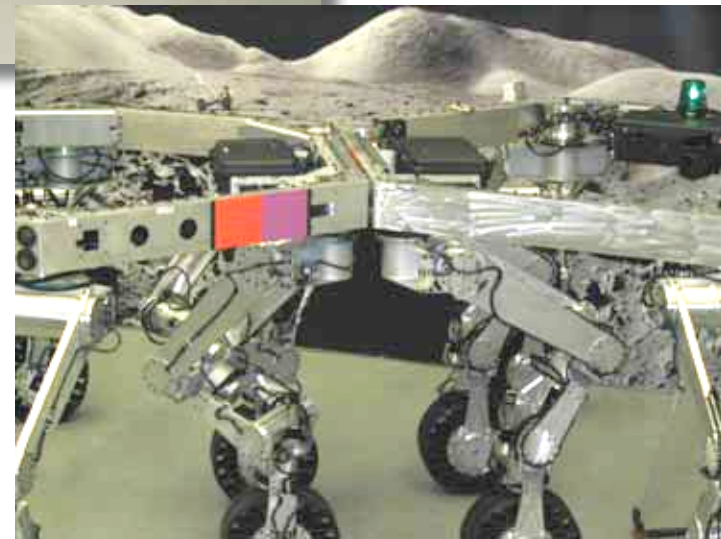


Some JPL Robots / Rovers





More Robots





Historical Antecedents



- **Late 80's - Early 90's:** parallel robotic developments
 - RSI, MOTES, Satellite Servicing, Robby, Mircover
 - No shared *hardware* or *software*
- **Mid 90's:** Mars rover research centralized with Rocky 7
 - First flight rover
- **Late 90's:** Expansion and diversification of rover work
 - No software interoperability (Rocky 7, FIDO, Athena, DARPA)
 - Autonomy demonstration of Remote Agent Experiment (ARC and JPL)
 - MDS investigates reusable software for spacecraft control.
- **'99-Early 00:** Exploration Technology Program develops concept for a unifying autonomy architecture
 - Unifying autonomy and robotic control
 - Started the CLARAty task
- **Today:**
 - Unification of several robotic developments at JPL, ARC, and CMU
 - Two flight rovers with several new robotic capabilities



Application Domains

- Navigation
- Target Tracking
- Single-cycle Instrument Placement



Navigation with Different Rovers

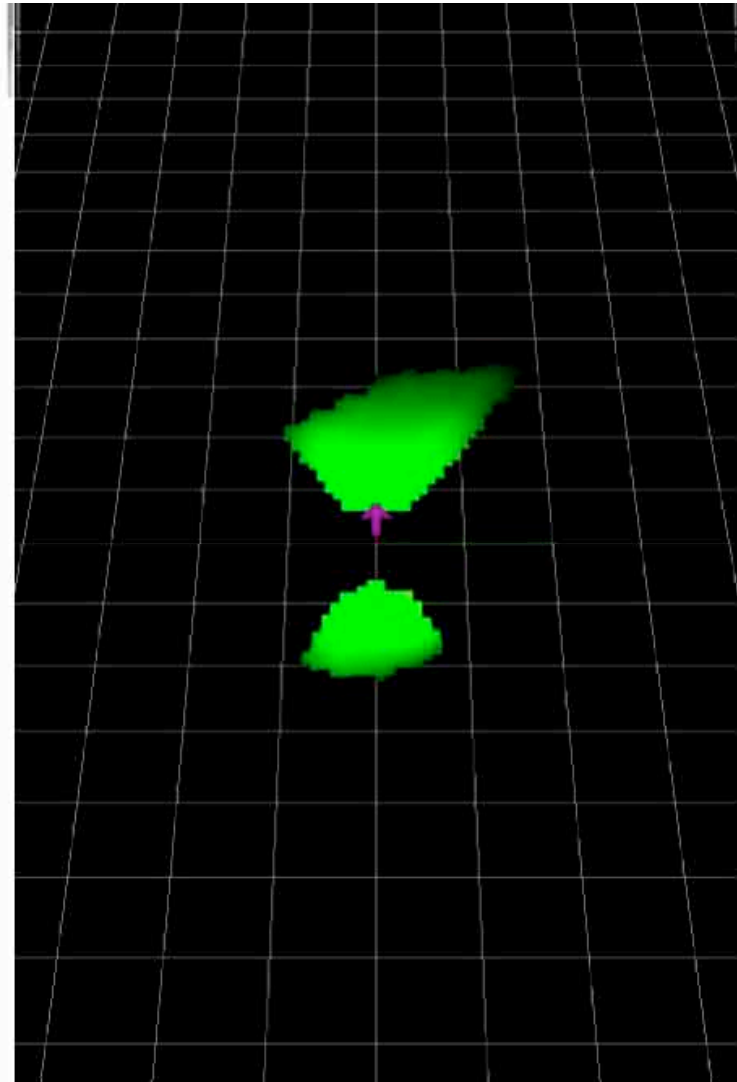


Complex Algorithms on different Platforms

- I/O, motion control
- Trajectory Generation
- Rough Terrain Locomotion
- Odometry Pose Estimation
- Stereo Processing
- Visual Odometry
- Navigation (Morphin)
 - Obstacle avoidance
 - Path Planning



Navigation with Dynamic Simulator



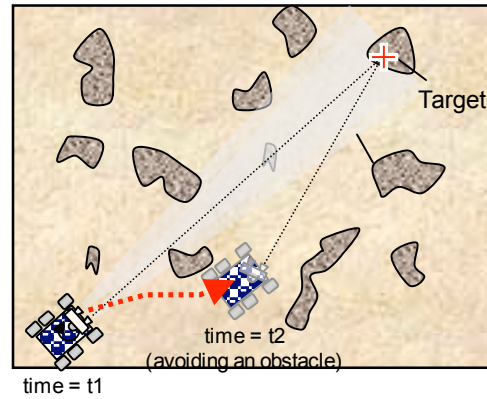


Application Domains

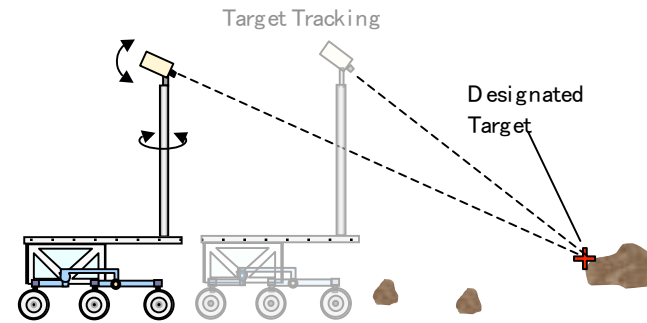
- Navigation
- Target Tracking
- Single-cycle Instrument Placement



Visual Target Tracking



(a)



(b)



1st Frame

Changes in FOV



37th Frame after 10 m



FALCON Visual Target Tracker on Rocky 8





Application Domains

- Navigation
- Target Tracking
- Single-cycle Instrument Placement



Integrated Single-Cycle Instrument Placement



February 16-17, 2006

Dependability Workshop - I.A.N.

13



Techniques We Use



- Some of the techniques that we have explored to improving software reliability are:
 - Improved processes and procedures for software development
 - Static code analysis and validation tools
 - Increased software reliability through reuse
 - Formal technology validation
 - Nightly regression testing
 - Fault-tolerant software and redundant computing

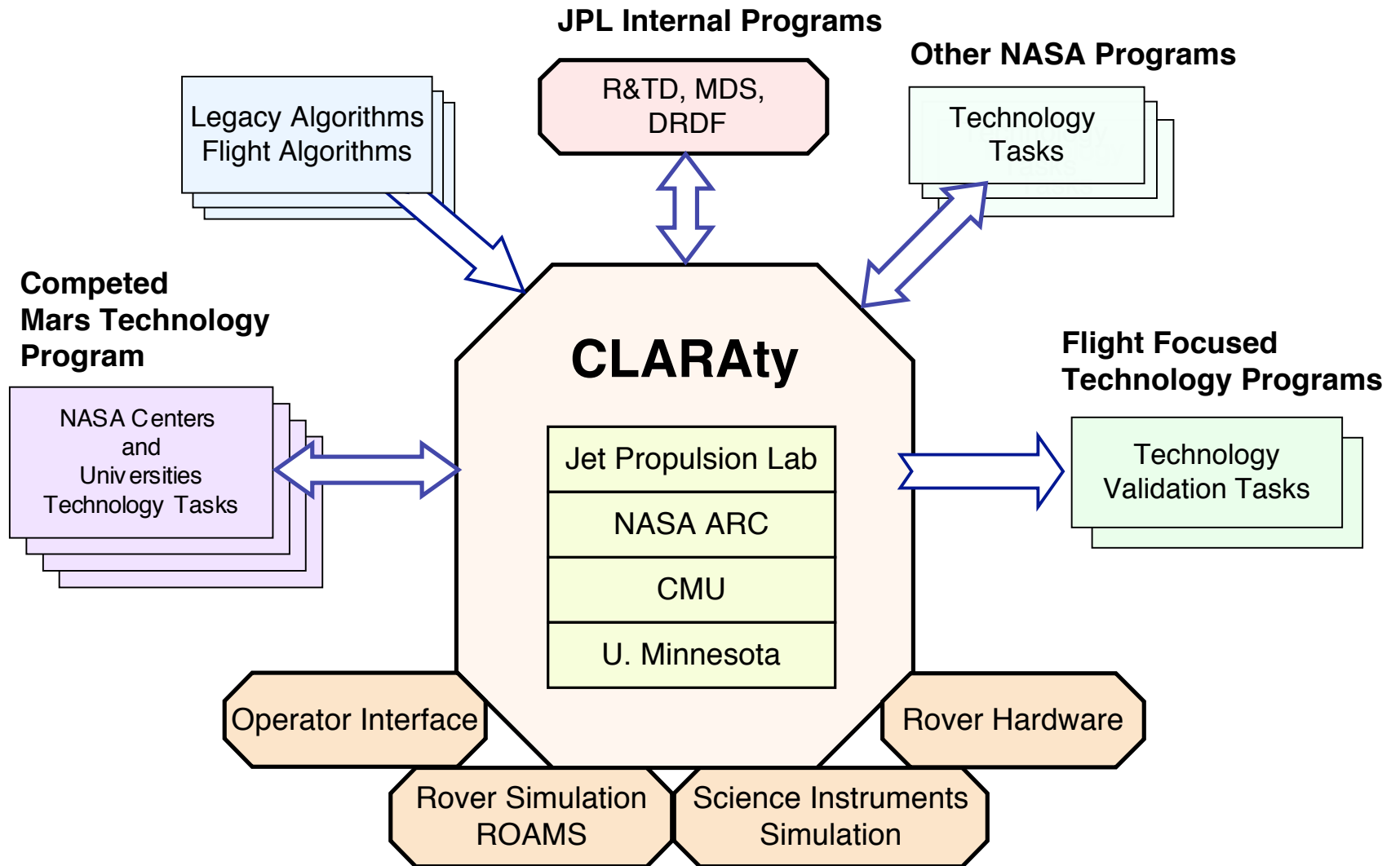


Process



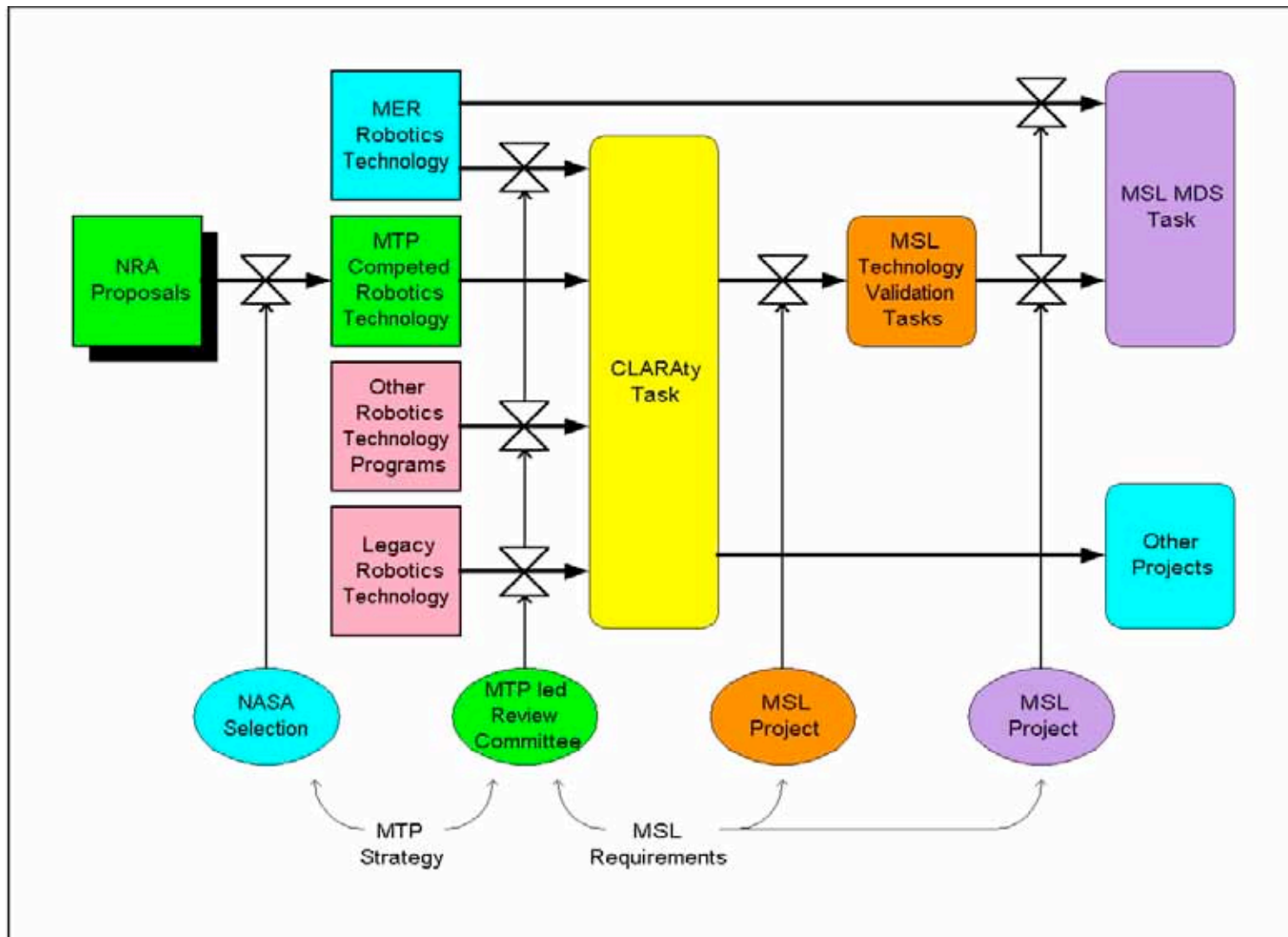


Technology Development, Integration and Validation





Technology Component Flow





Flight Software Processes and Tools



- List coding conventions, rules, and guidelines.
- Use only mission-proven or thoroughly tested technologies
- Hold formal design reviews
 - Review designs before and after implementation
 - Review interfaces, implementations, test plans, commands, and telemetry for each software component
- Use code buddy reviews
 - Have someone other than the developer statically review the code and look for potential problems or violations of the coding conventions.
 - Use automated tools for source code analysis to highlight suspicious code segments. For example, MER used Code Wizard and Cleanscape on early versions of the flight software and Coverity on most recent versions.
 - Review by internal and external teams. Use validation and verification group expertise.



Rigorous Flight Software Testing



- Unit Testing:
 - Extensive testing of each module in isolation by the developer
- Regression Testing:
 - Integrated module testing by a dedicated test team after new modules are integrated.
- System Testing:
 - Project wide rehearsals of expected mission scenarios
 - Can last several days where several different activities would be tested in the manner they would be used in the mission.
 - All communication is done during communication passes.



Flight Software Architecture and Implementation



- Assignment of one "owner" developer per software module
- Object-oriented style design, with emphasis placed on interfaces, encapsulation, and modularity
- Objects implemented as hierarchical state machines
- Asynchronous message passing as the principle means of communication between objects
- Severe limitations on use of dynamic memory allocation to avoid heap fragmentation
- Extensive use of diagnostics embedded throughout the software, including many design-by-contract assertions
- Reference:
 - Glenn E. Reeves & Joseph F. Snyder "A Overview of the Mars Exploration Rovers' Flight Software" 2005 IEEE International Conference on Systems, Man and Cybernetics Waikoloa, Hawaii, October 10-12, 2005



Historical Antecedents and Motivation





Problem and Approach



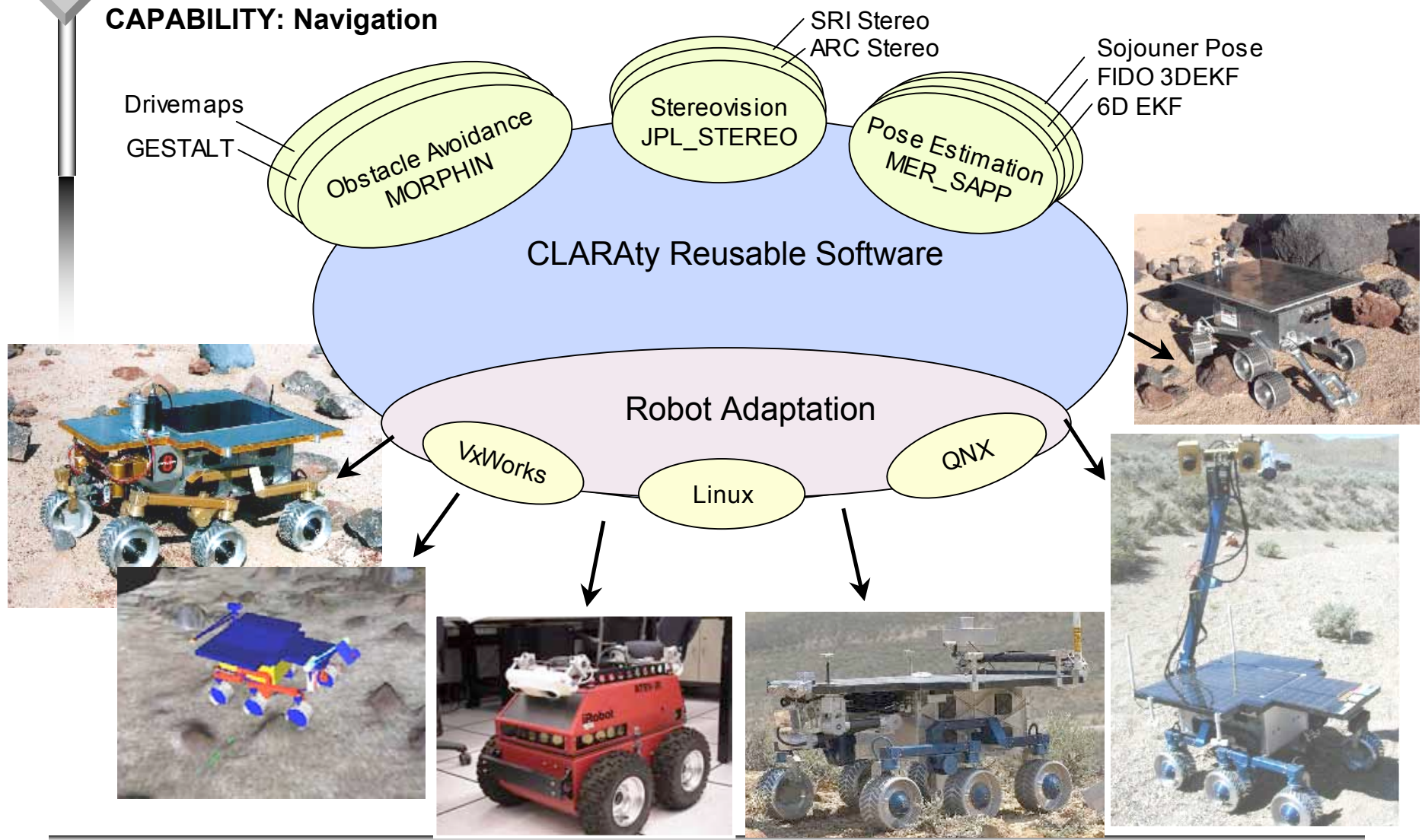
- Problem:
 - Difficult to share software/algorithms across systems
 - Different hardware/software infrastructure
 - No standard protocols and APIs
 - No flexible code base of robotic capabilities
- Objectives
 - Unify robotic infrastructure and framework
 - Capture and integrate legacy algorithms
 - Simplify integration of new technology
 - Operate heterogeneous robots



Interoperability: Software & Hardware



CAPABILITY: Navigation



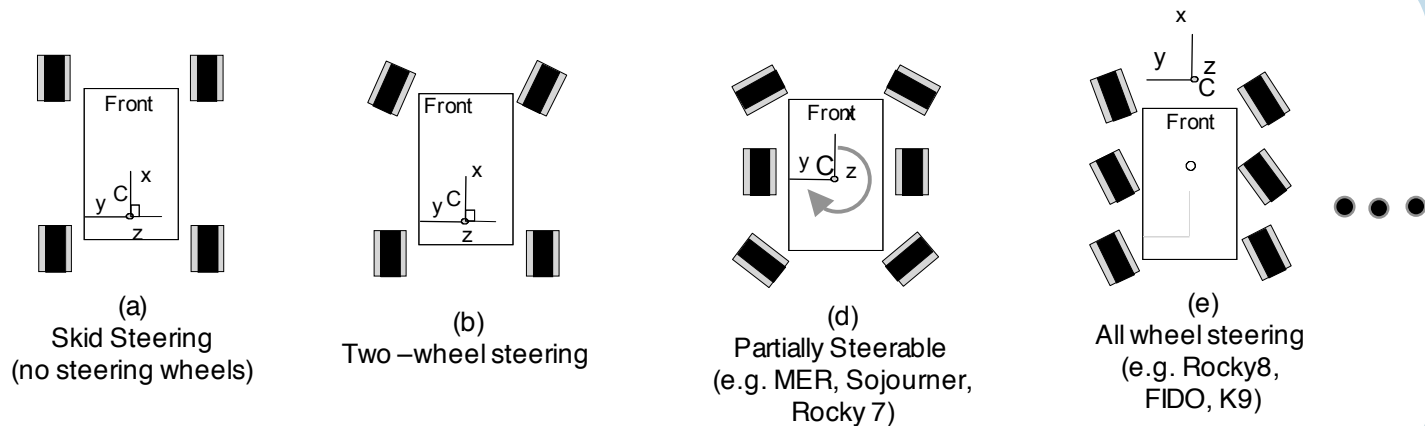


Challenges in Reuse

- Mechanisms and Sensors
- Hardware Architecture
- Software Algorithms

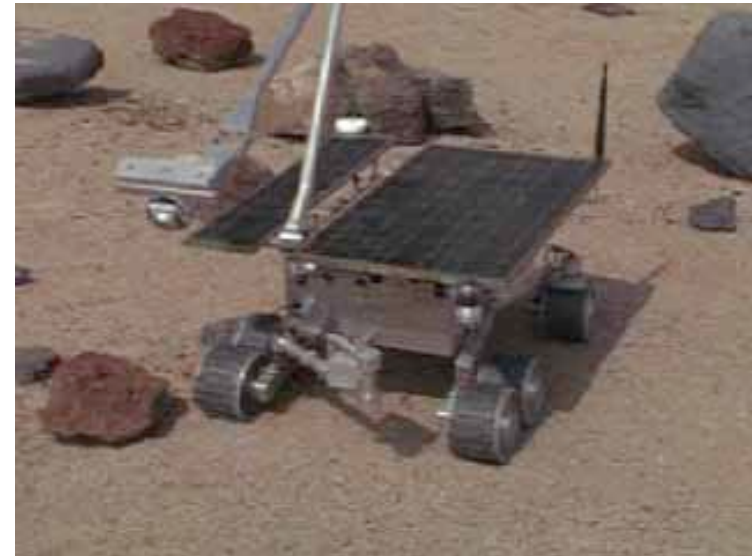
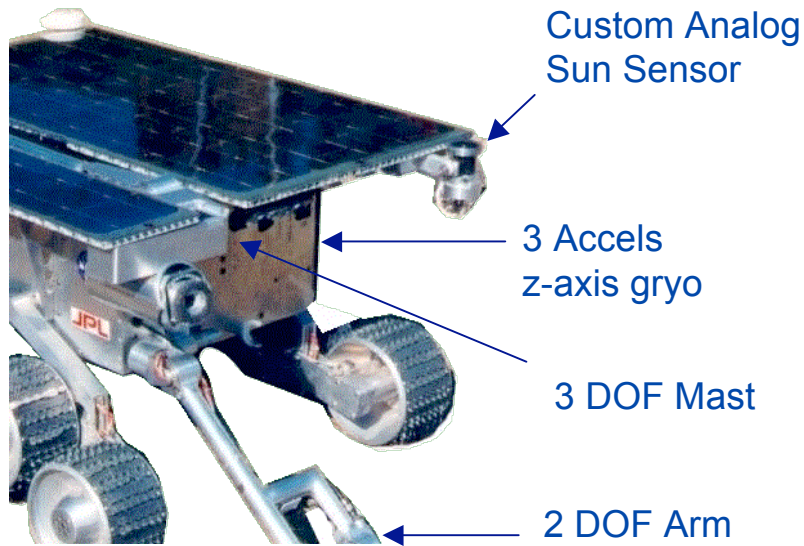


Different Mobility Mechanisms

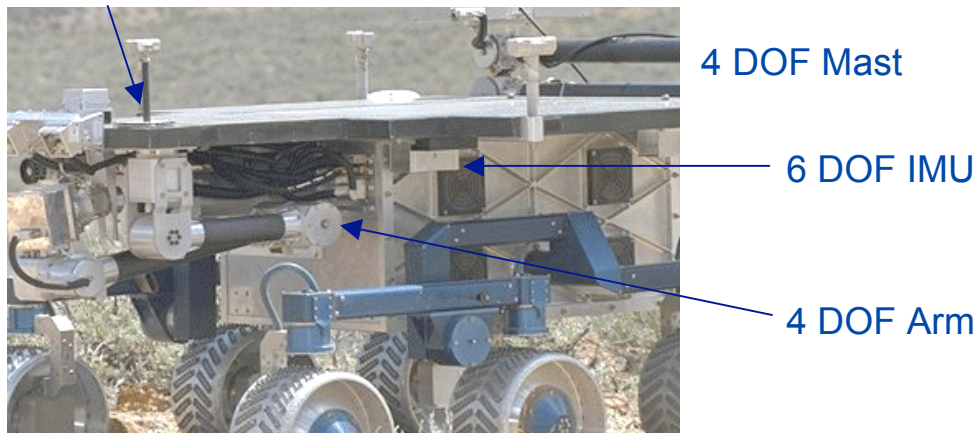




Different Sensors and Appendages



Camera Sun Sensor





Challenges in Reuse

- Mechanisms and Sensors
- **Hardware Architecture**
- Software Algorithms

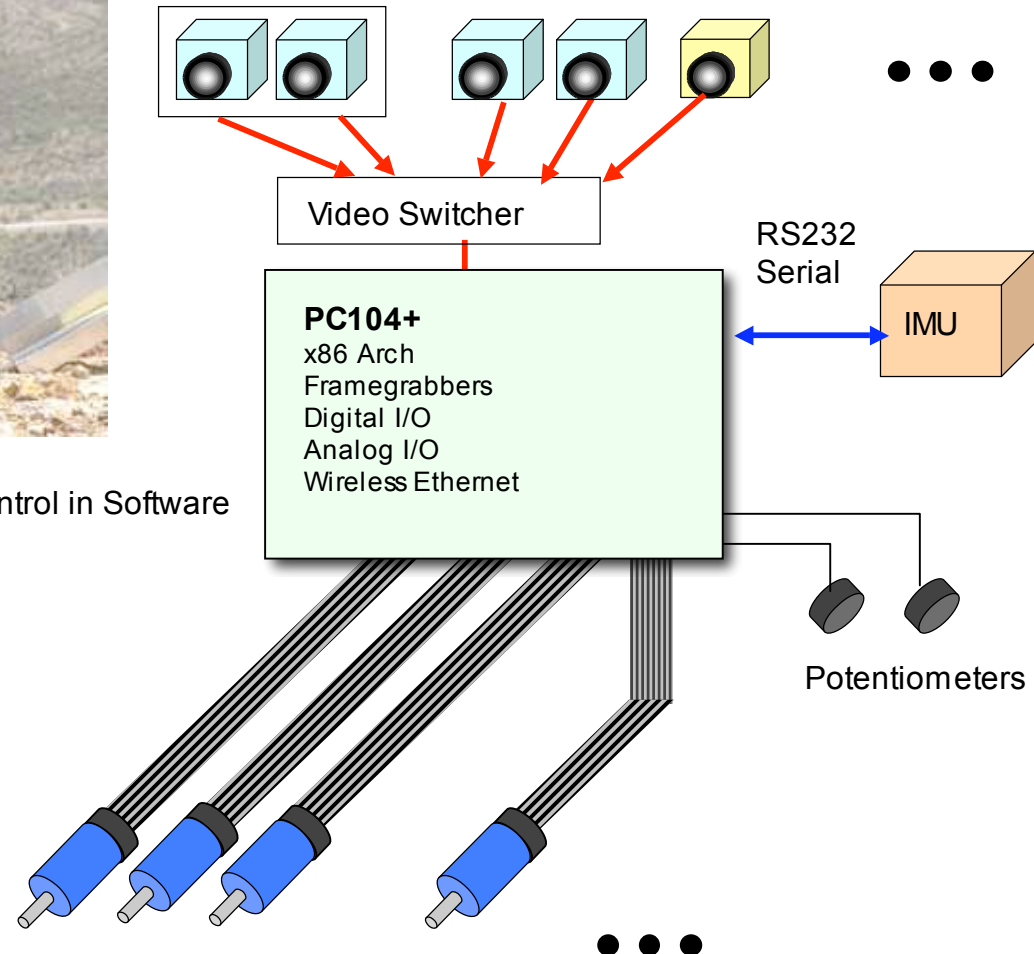


Centralized Hardware Architecture



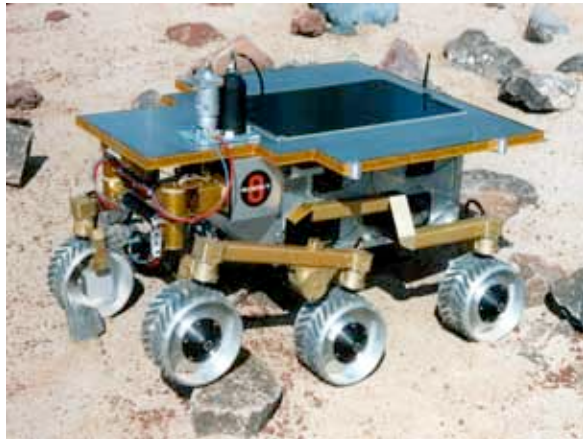
FIDO

PID Control in Software



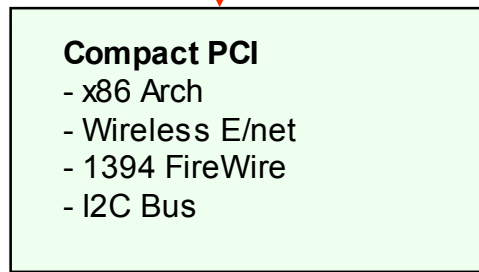
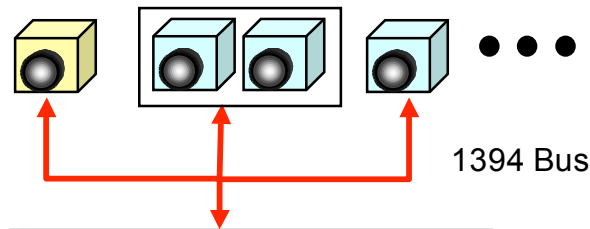


Distributed Hardware Architecture

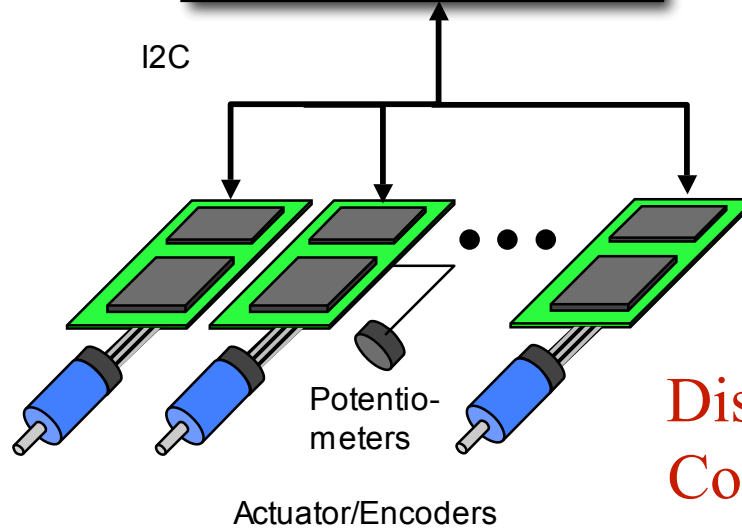


Rocky 8

Sun Sensor



I2C



Rocky Widgets

- Single-axis controllers
- Current sensing
- Digital I/O
- Analog I/O

Distributed Motion Control and Vision



Challenges in Interoperability

- Mechanisms and Sensors
- Hardware Architecture
- Software Algorithms



Software Challenges for Algorithm Infusion



The *new* algorithms to be integrated may:

- Have architectural mismatches with the framework
- Include multiple orthogonal functionalities
- Make implicit assumptions about the platform
- Duplicate functionality in the framework
- Use incompatible data structures
- Are complex and hard to tune
- Require highly specialized domain expertise
- Are poorly implemented

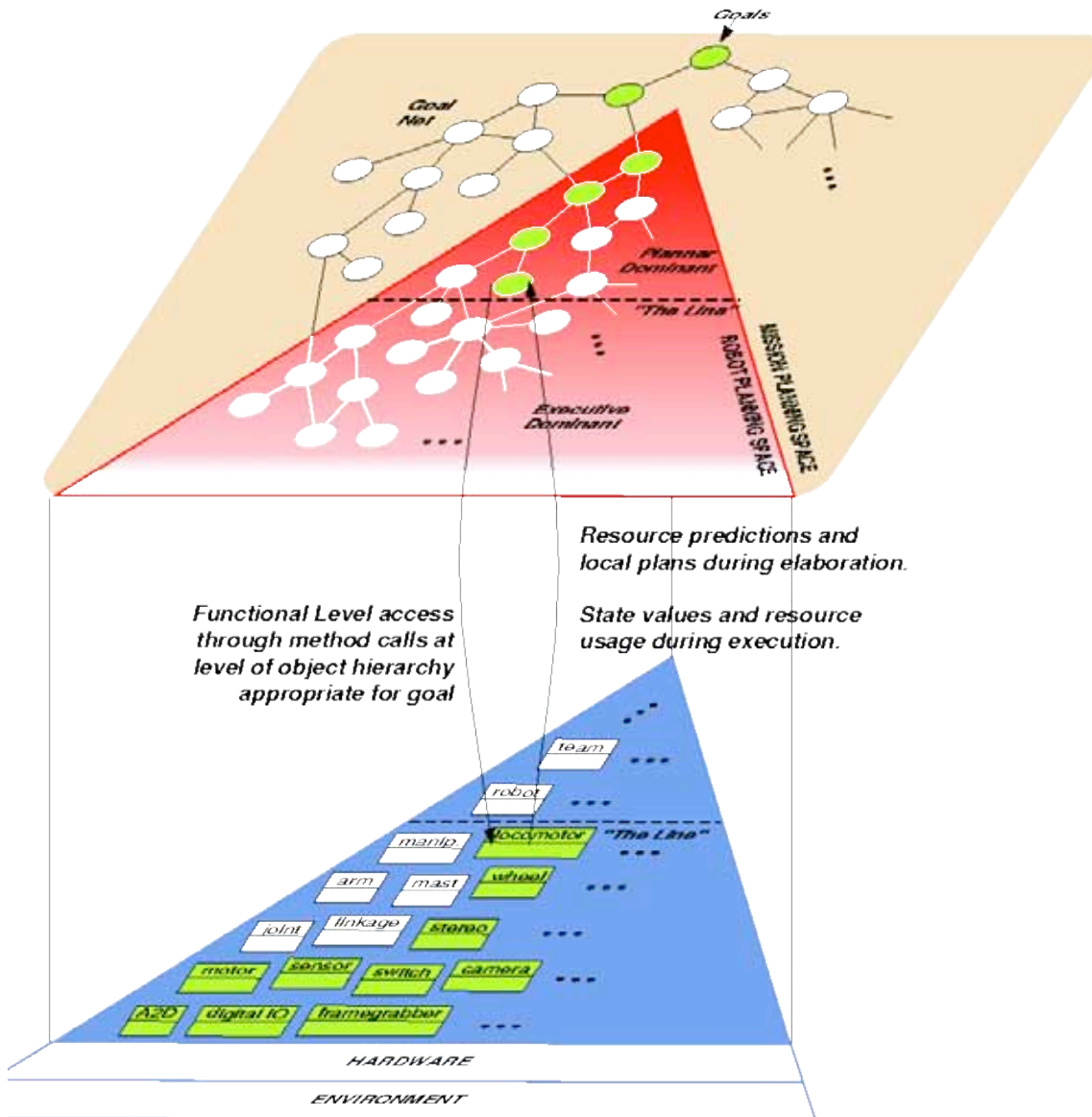


Architecture and Process



A Two-Layered Architecture

CLARAty = Coupled Layer Architecture for Robotic Autonomy



THE DECISION LAYER:

Declarative model-based
Global planning

INTERFACE:

Access to various levels
Commanding and updates

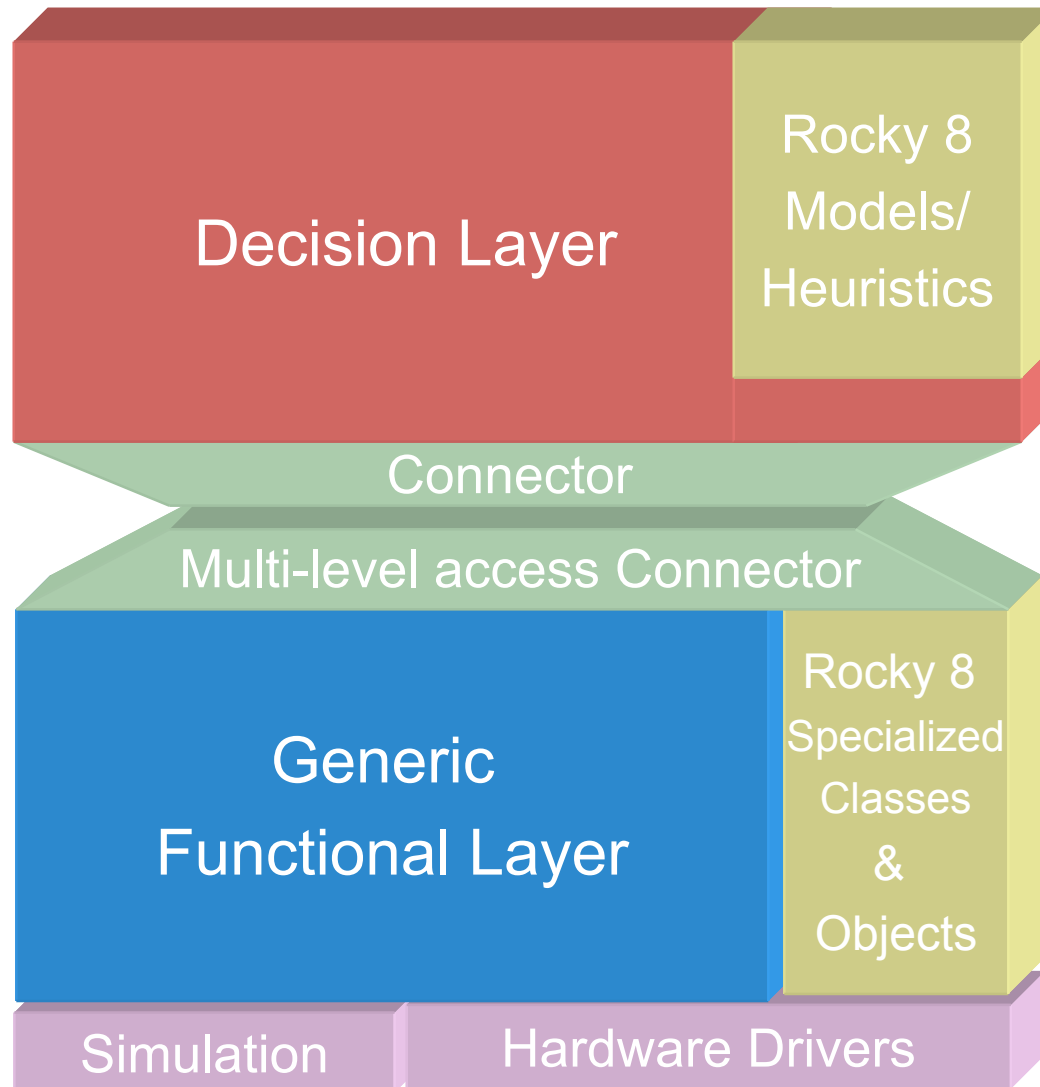
THE FUNCTIONAL LAYER:

Object-oriented abstractions
Autonomous behavior
Basic system functionality

Adaptation to a system

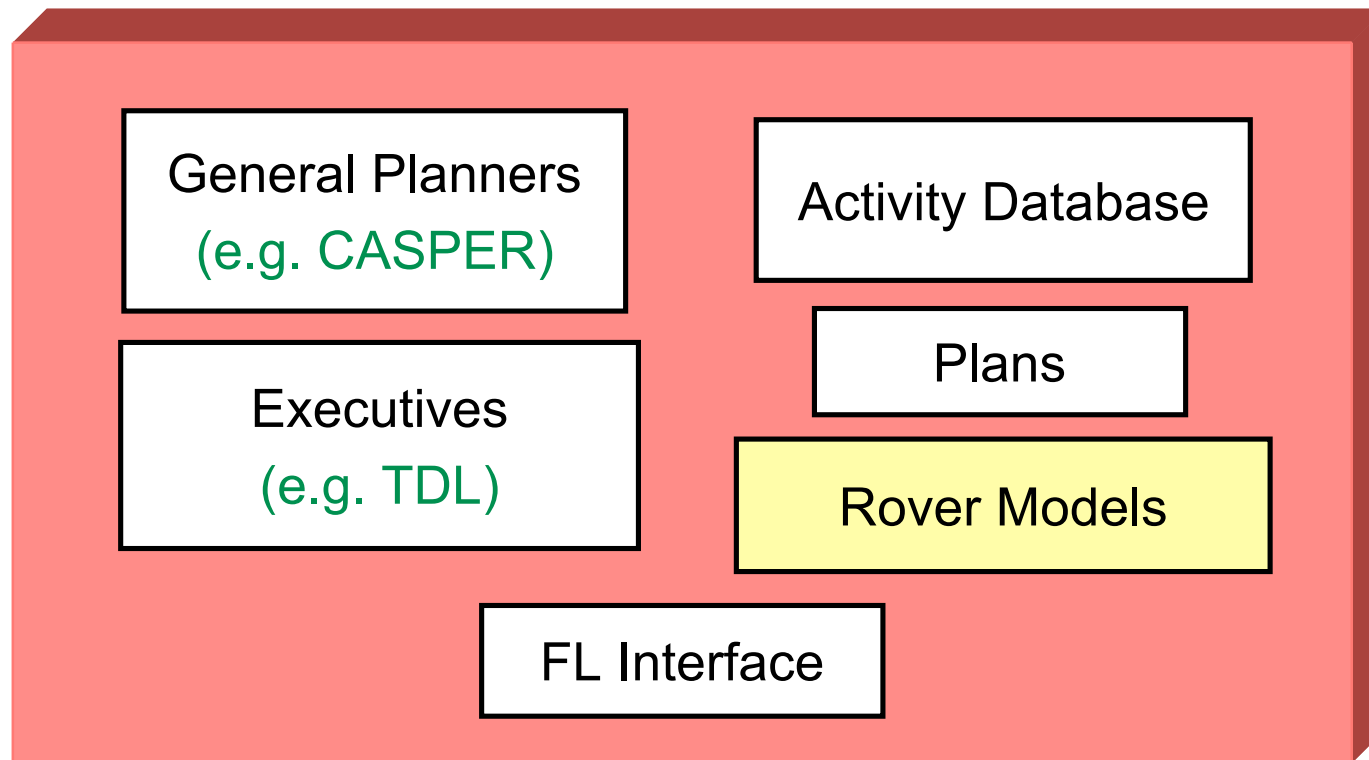


Adapting to a Rover



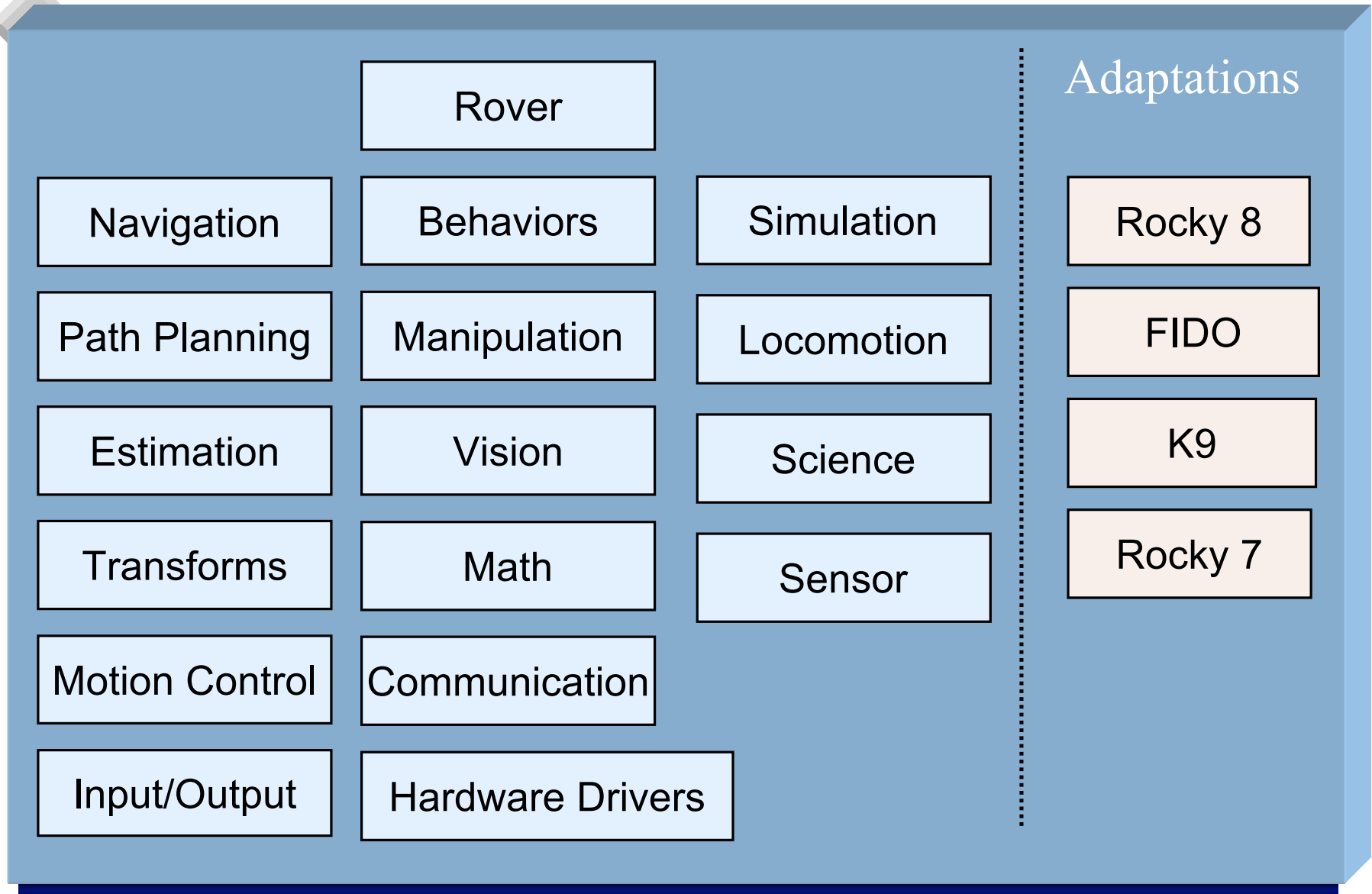


The Decision Layer





The Functional Layer



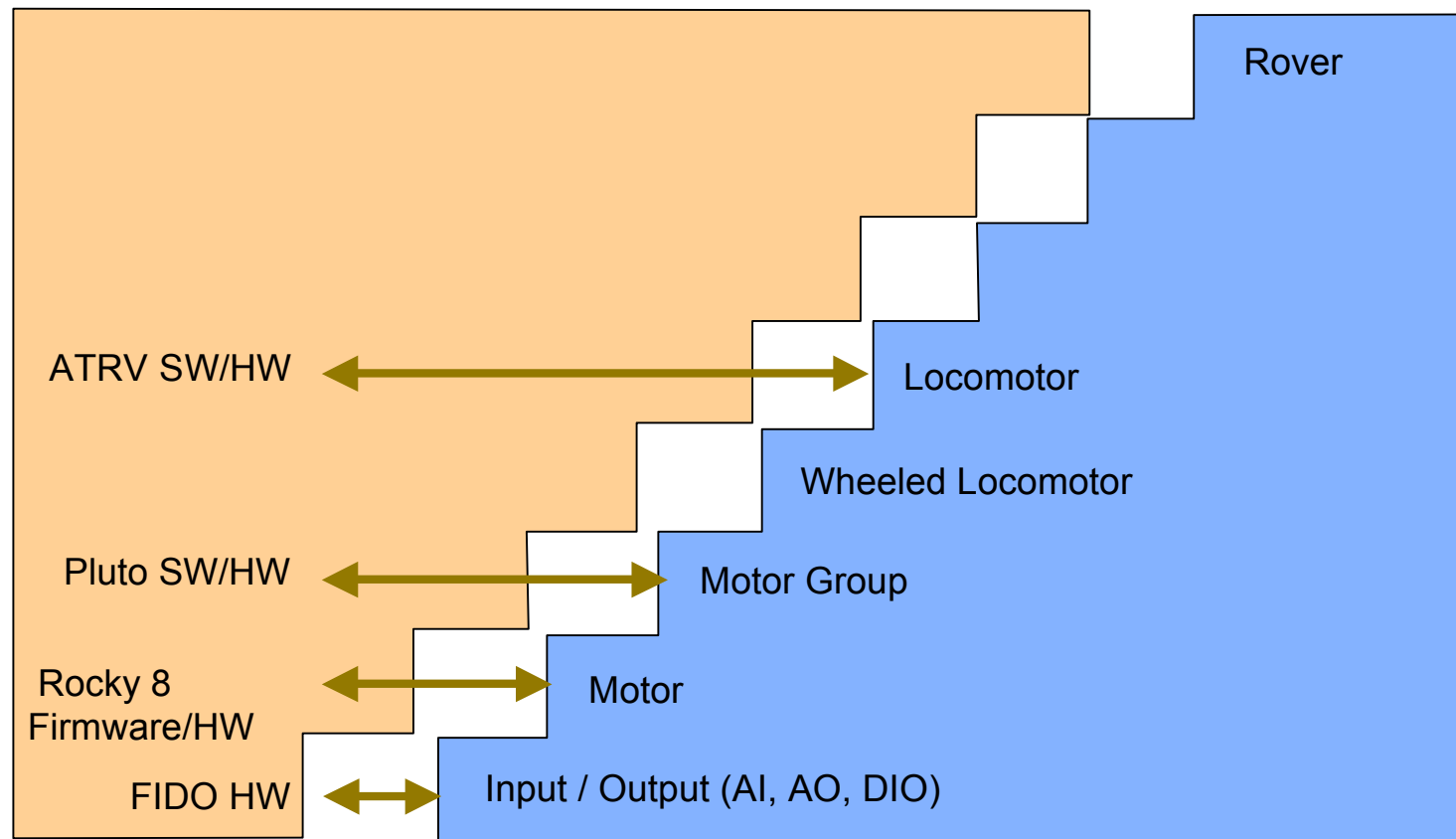


Multi-level Abstraction Model



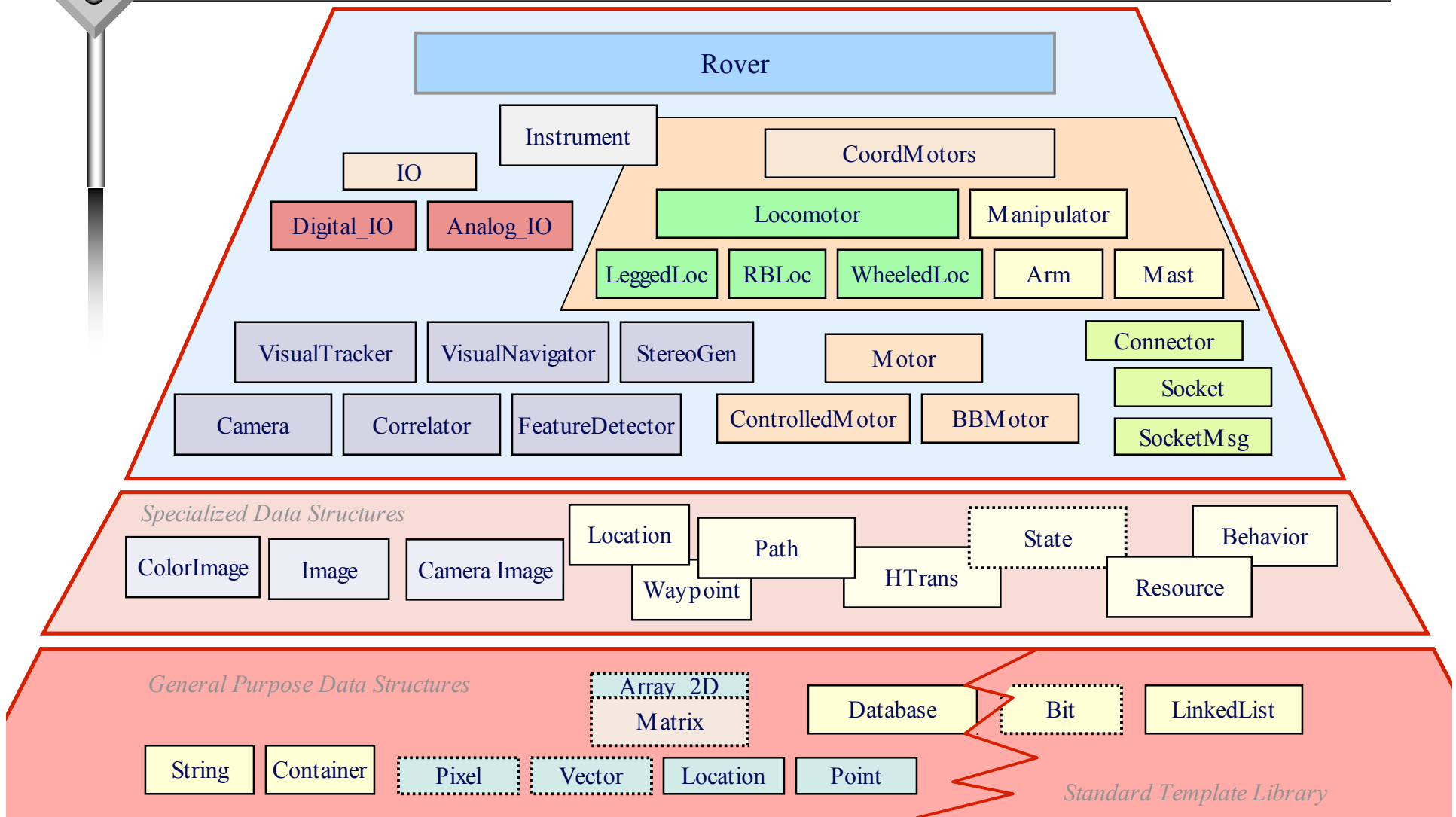
Use abstractions
Interface at different levels

Example of
Levels of Abstraction
for locomotors





Functional Layer Components

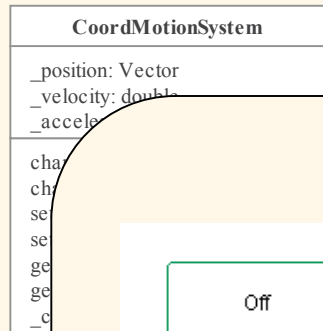




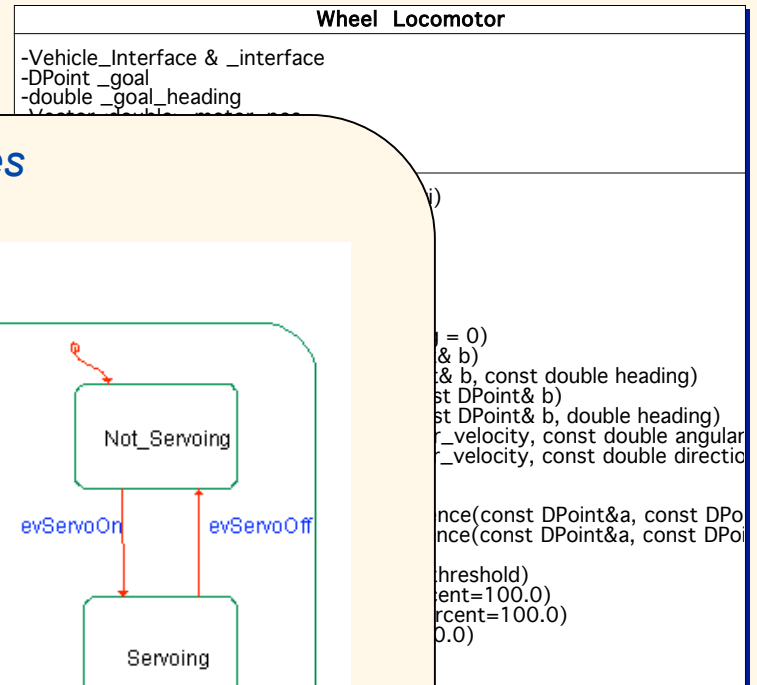
Standardizing Base Abstractions



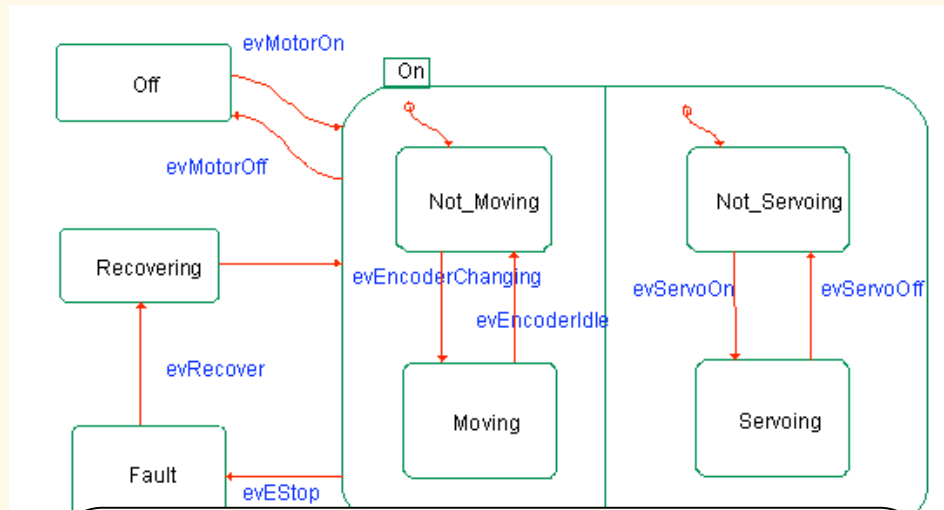
Abstractions



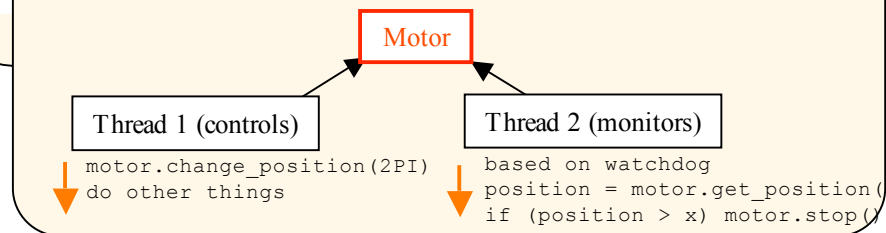
APIs and Behaviors



State Machines



Runtime Models

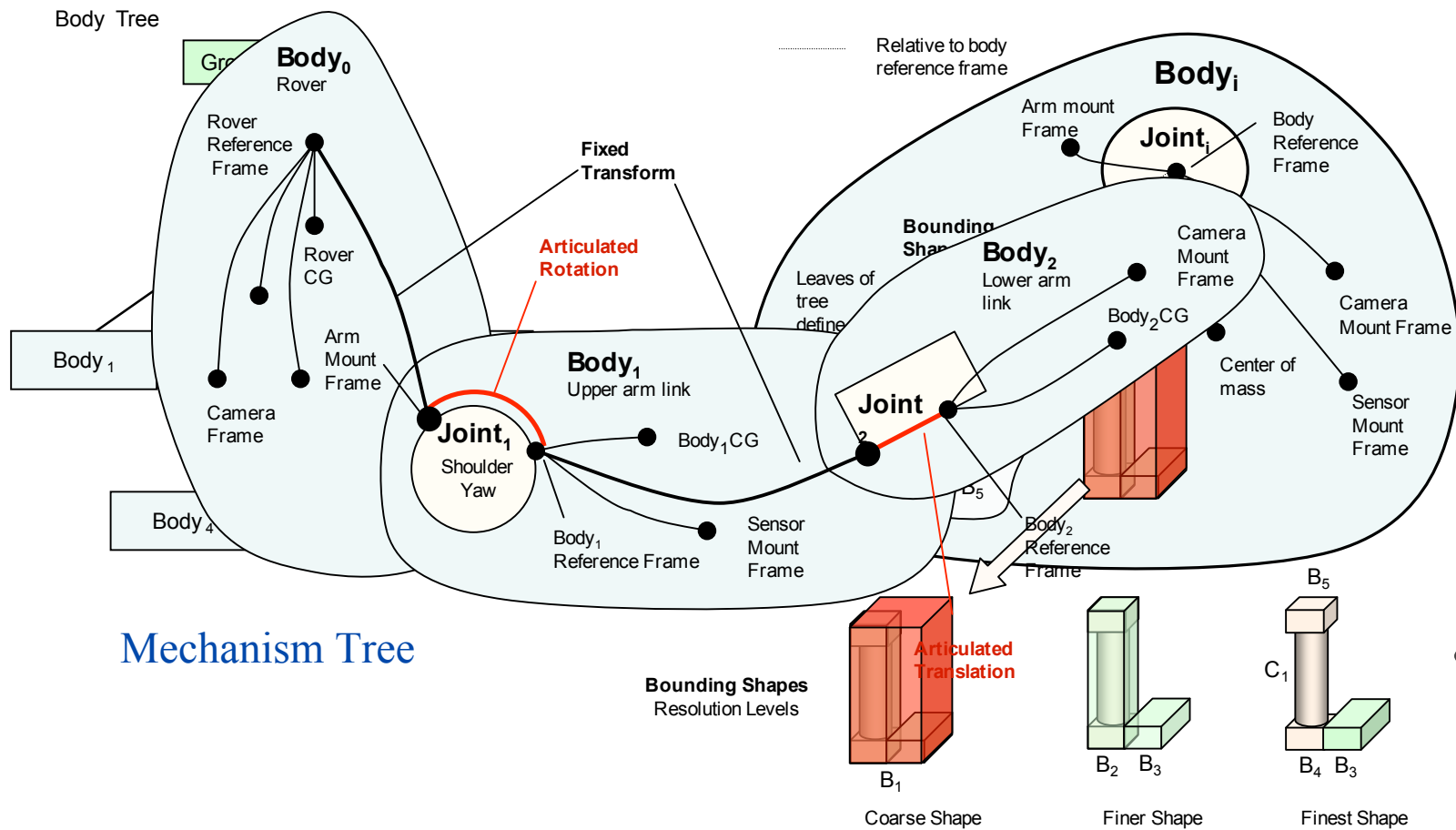




Unify Mechanism Model



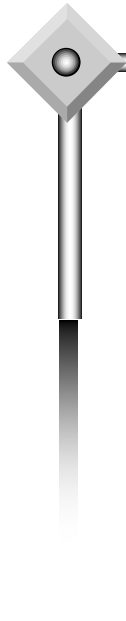
Bodies and Joints



Mechanism Tree

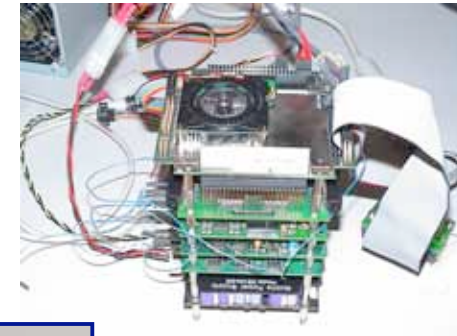


Unit and Regression Testing





CLARAty Test Bed for Regression Testing



FIDO2 Stack



ATRV Jr.



Dexter Manipulator
Bench top



Rocky 8 PPC Bench top



Summary



- Software complexity and size continues to rise
- Increasing software reliability remains a challenge because algorithms are sensitive to environmental uncertainties
- Formal validation of algorithms helps assess performance and risk of technology
- Reuse of framework helps improve reliability of infrastructure
- Deploying at multiple institutions and on heterogeneous robots improves reliability through diversity
- Assessing performance and risk is critical for infusion into flight



Current CLARAty Core Team

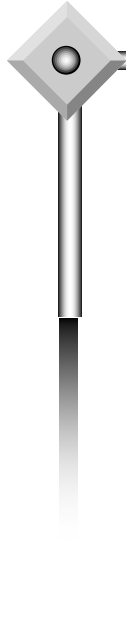


- **NASA Ames Research Center**
 - Clay Kunz
 - Eric Park
 - Susan Lee
- **Carnegie Mellon University**
 - David Apelfaum
 - Nick Melchior
 - Reid Simmons
- **University of Minnesota**
 - Stergios Roumeliotis
- **Jet Propulsion Laboratory**
 - Antonio Diaz Calderon
 - Tara Estlin
 - John Guineau
 - Won Soo Kim
 - Richard Madison
 - Michael McHenry
 - Mihail Pivtoraiko
 - Issa A.D. Nesnas
 - Babak Sapir
 - I-hsiang Shu
- **OphirTech**
 - Hari Das Nayar

Full Credits for all Developers and Contributors at:
<http://keuka.jpl.nasa.gov/main/project/team/index.shtml>

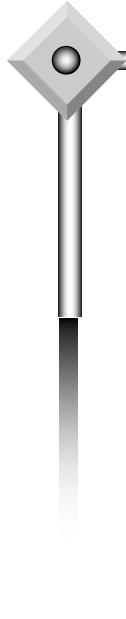


Thank you





Back-up Slides

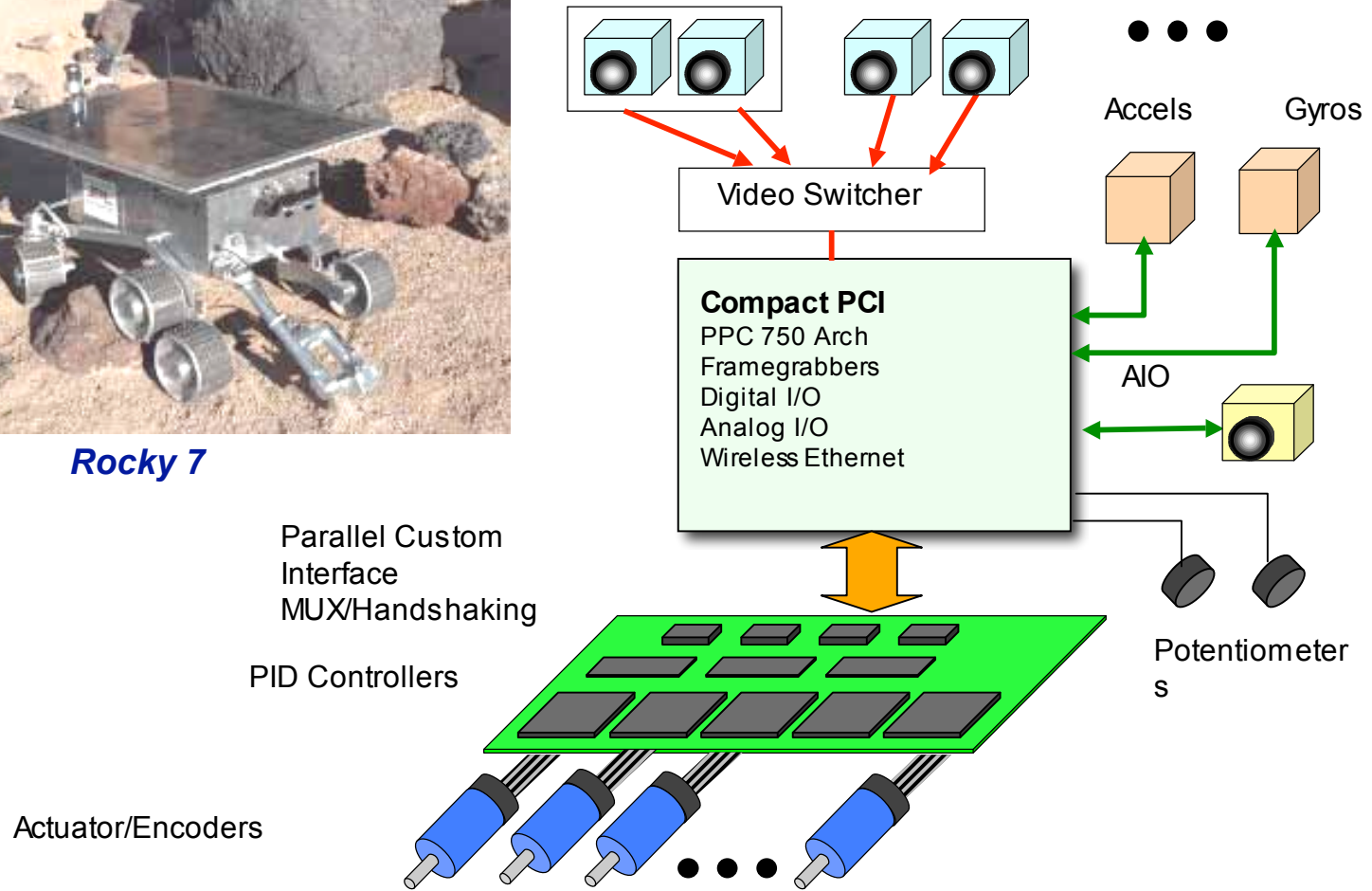




Semi-centralized Hardware Architecture



Rocky 7





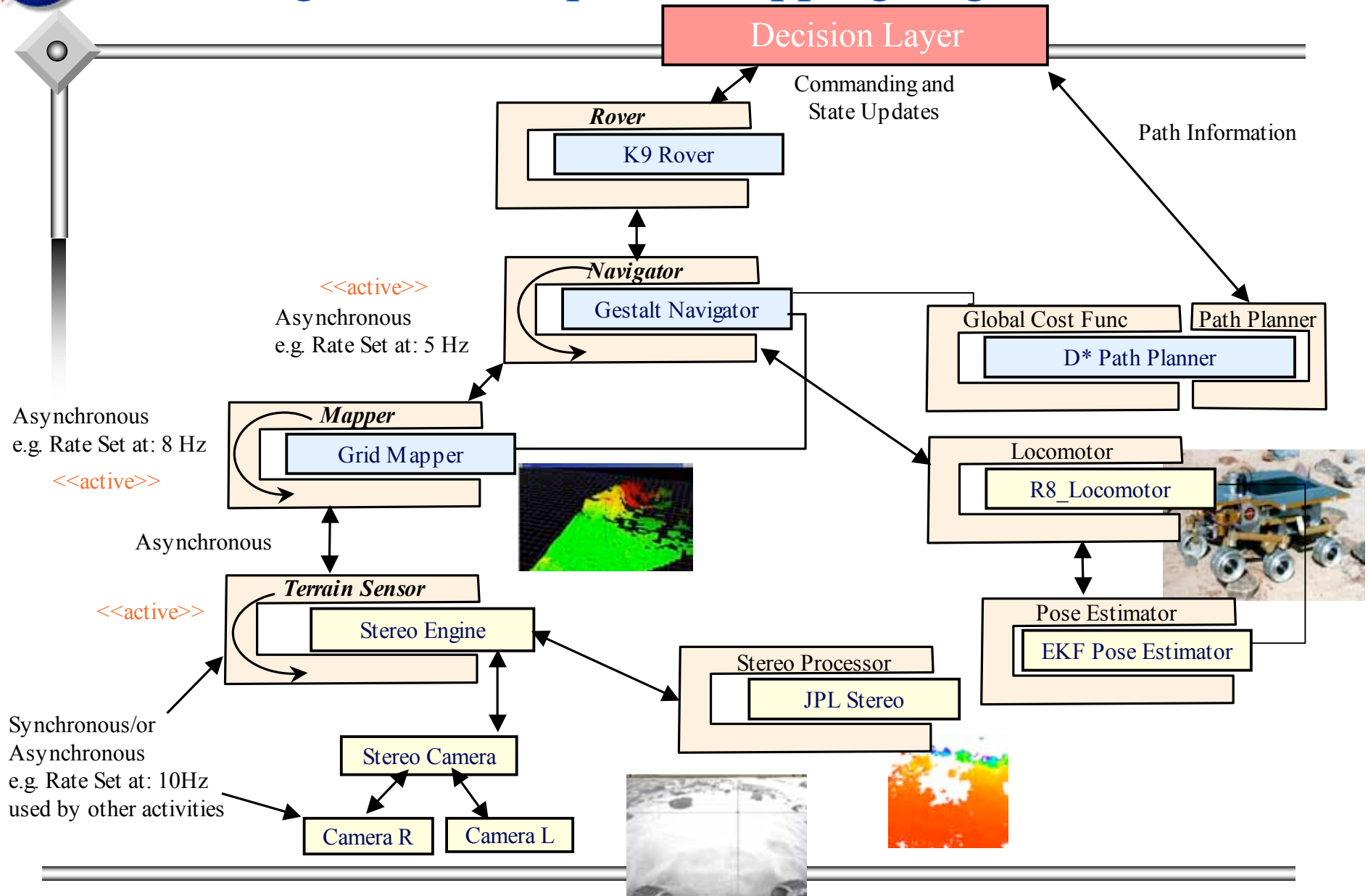
One Approach



- Develop
 - Common data structures
 - Physical & Functional Abstractions
 - E.g. motor, camera, locomotor. Stereo processor, visual tracker
 - Unified models for the mechanism
- Putting it together
 - Start with top level goals
 - Elaborate to fine sub-goals
 - Choose the appropriate level to stop elaboration
 - Interface with abstractions
 - Abstractions translate goals to action
 - Specialize abstractions to talk to hardware
 - Hardware controls the systems and provide feedback

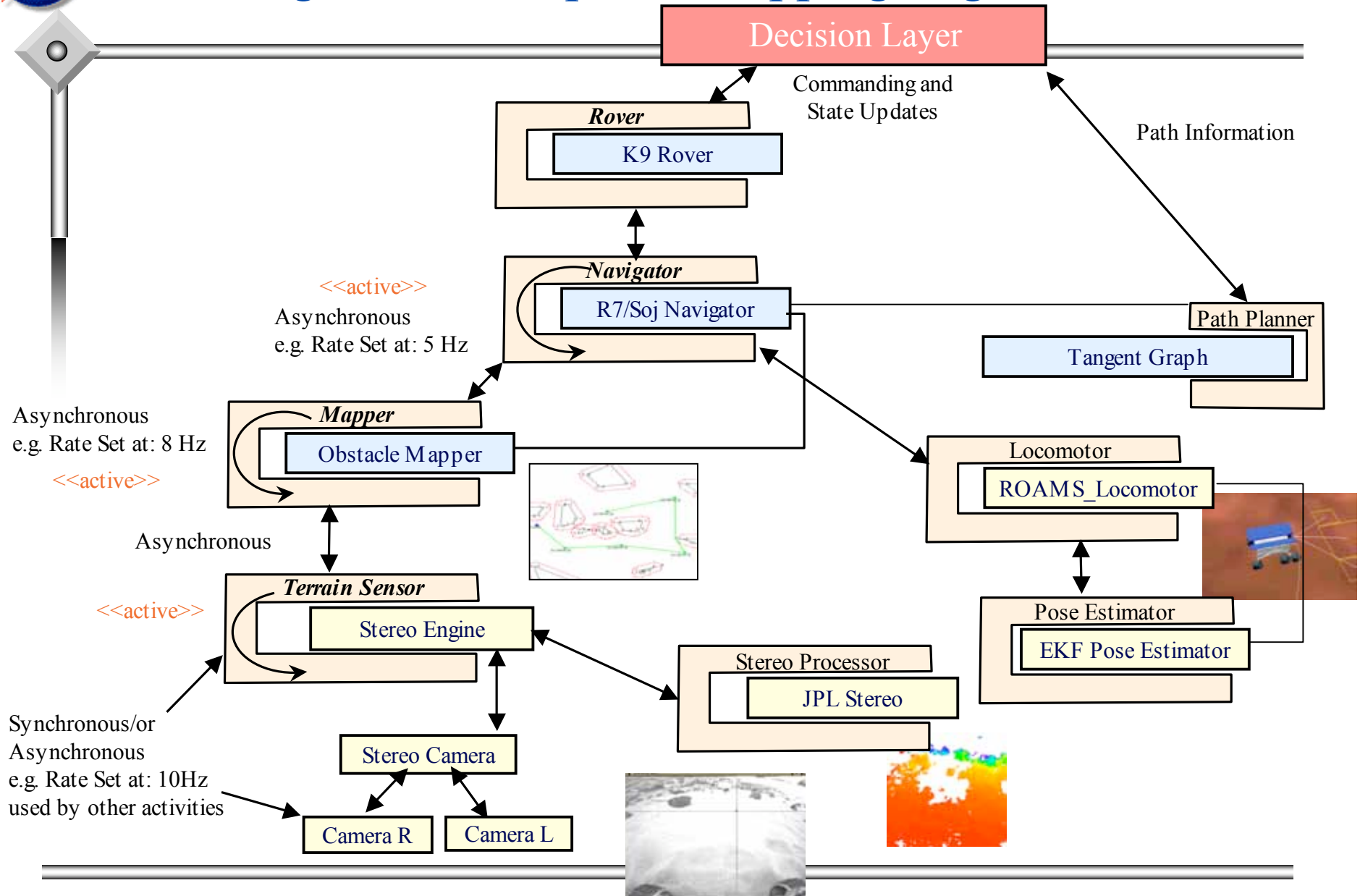


Navigation Example - Swapping Algorithms



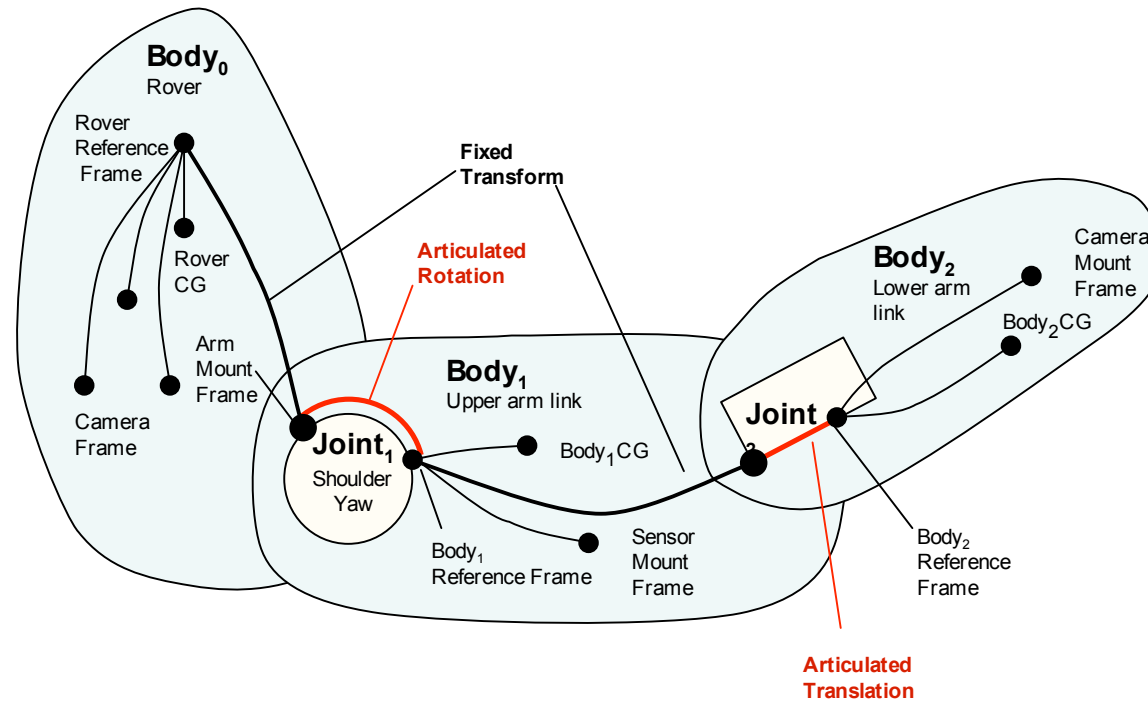


Navigation Example - Swapping Algorithms





Connecting Bodies and Joints





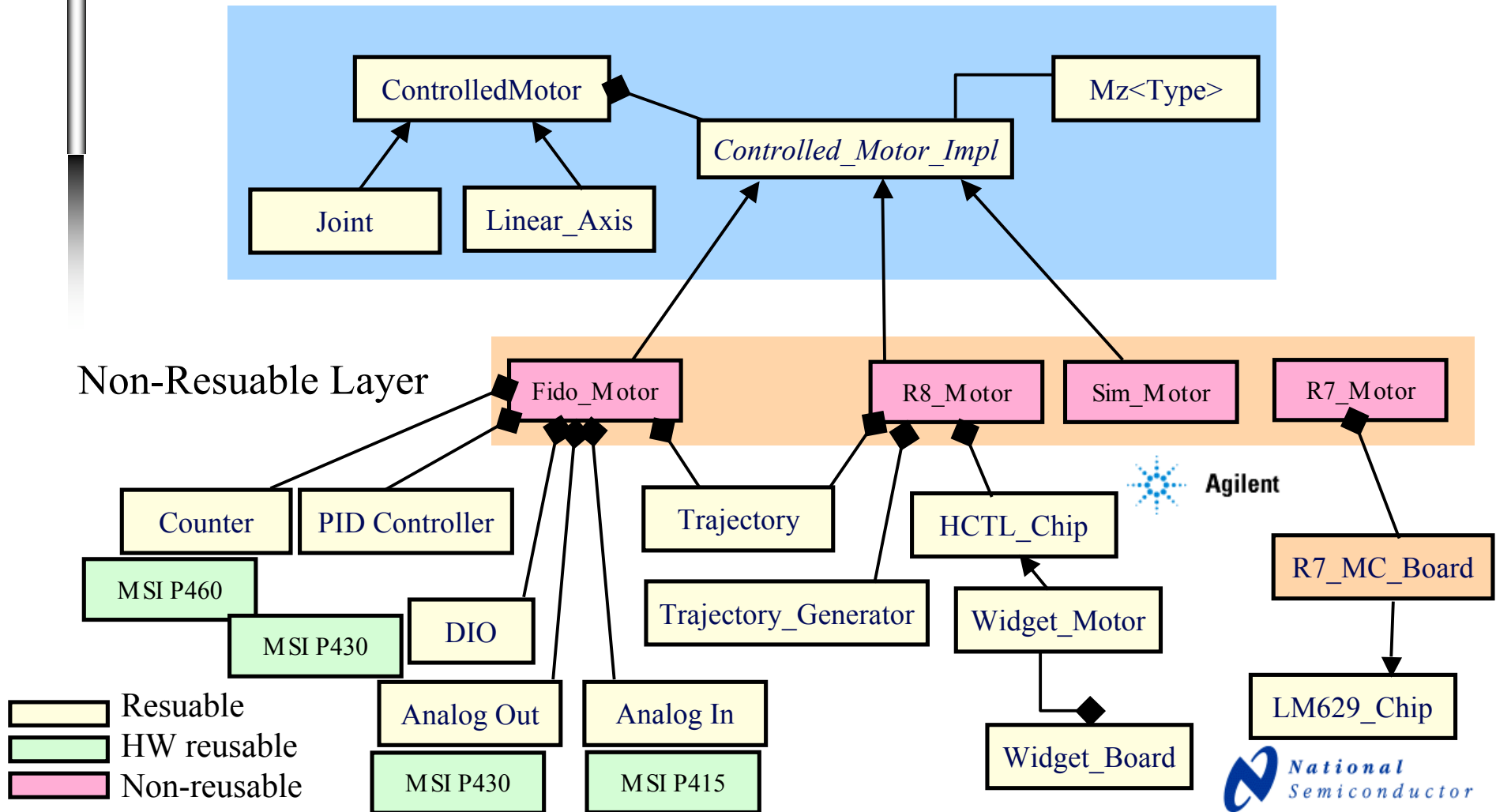
Conclusions



- Use abstraction to master complexity
- Encapsulate and abstract hardware variations
- Provide multi-level access through Decision Layer for fault diagnosis and recovery
- Use domain expertise to guide design
- Make all assumptions explicit
- Stabilize external interfaces rapidly
- Document processes and products well
- Avoid over-generalization - define scope
- Encapsulate system specific runtime models
- Do not compromise performance - least common denominator solutions are unacceptable in hw/sw interactions
- Standardize Hardware



Examples of CLARAty Reusability





Supported Platforms



Rocky 8

VxWorks x86

JPL



K9

Linux

x86

Ames



Rocky 7

VxWorks ppc

JPL



FIDO

VxWorks x86

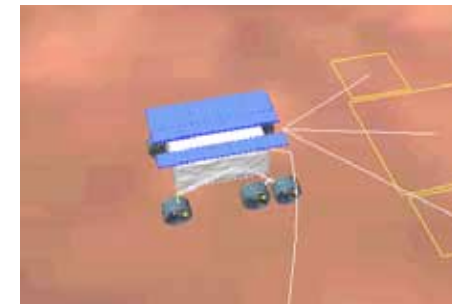
JPL



ATRV

Linux x86

CMU



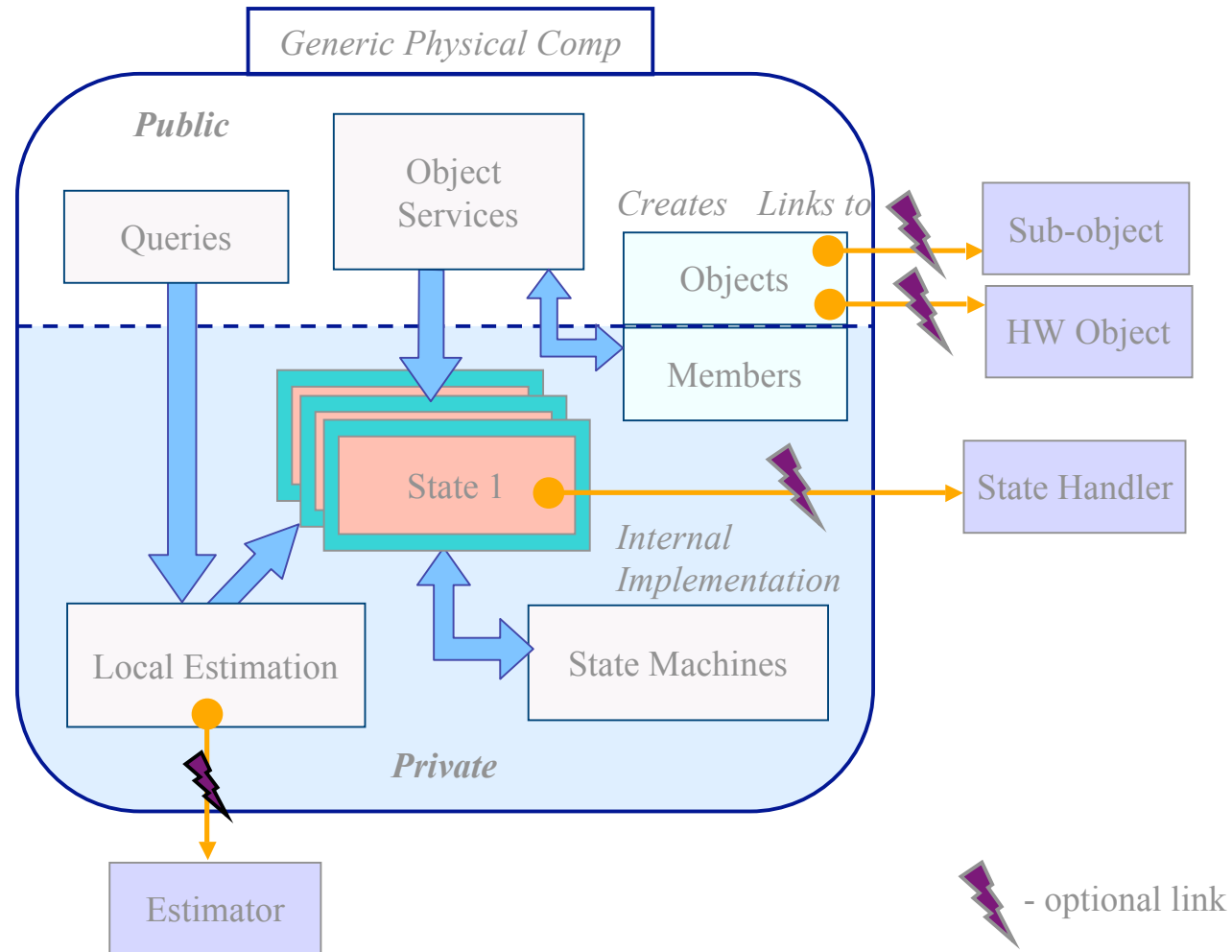
ROAMS

Solaris Linux

JPL



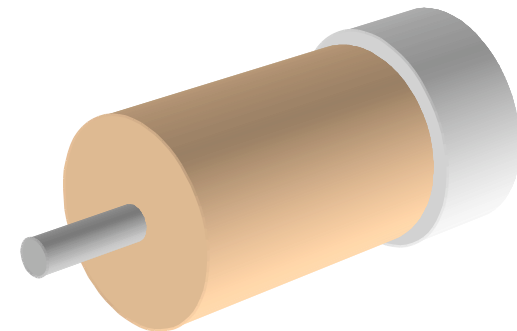
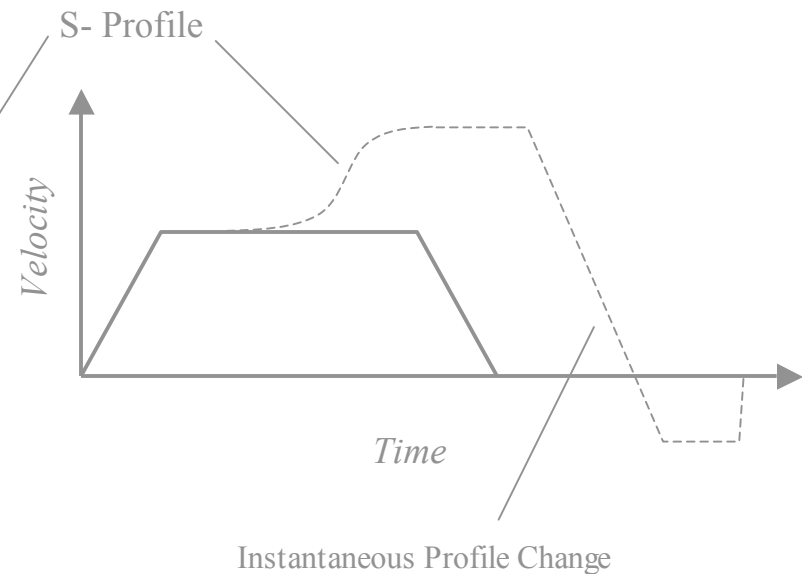
Component Analysis





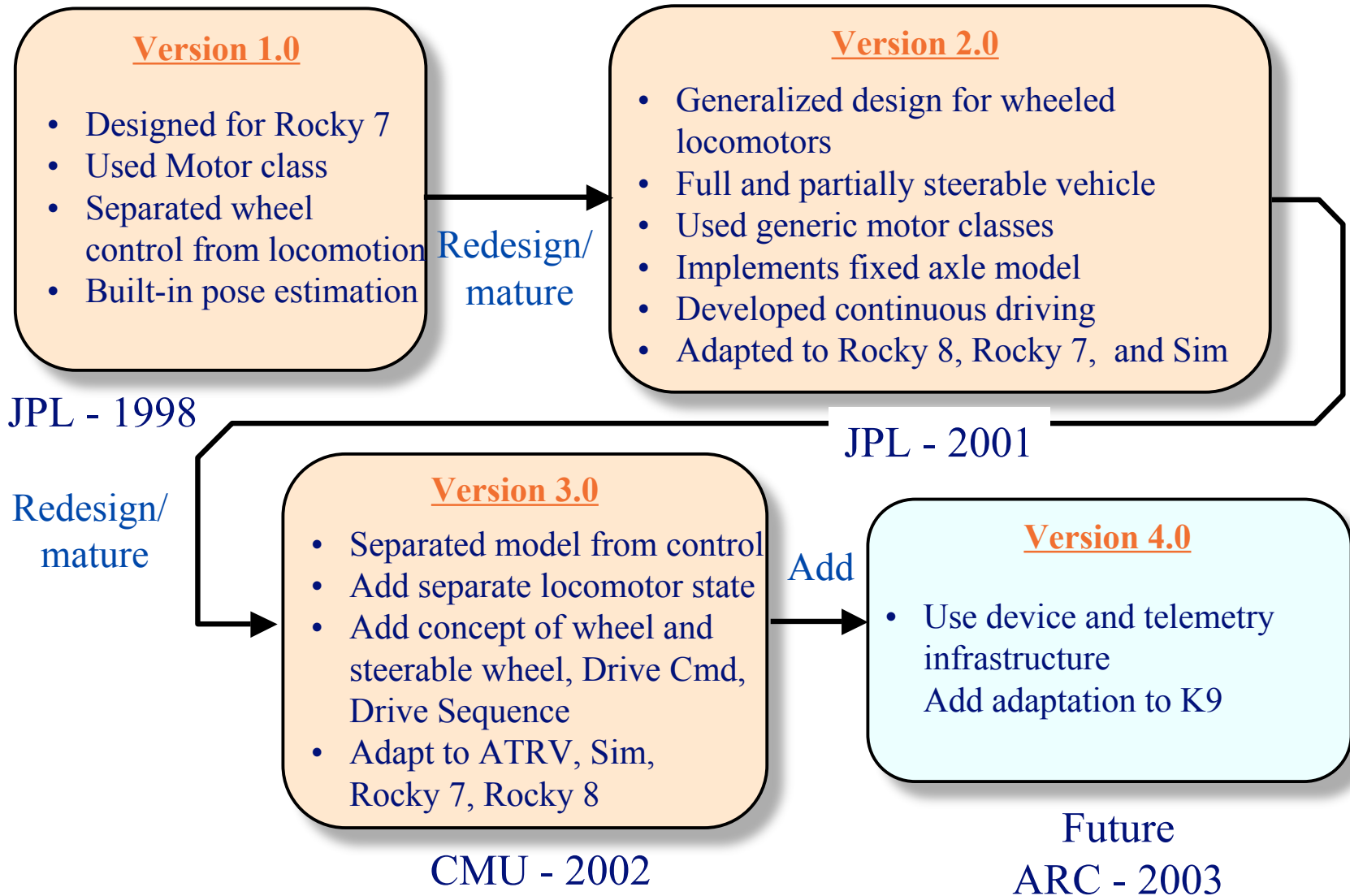
Example: Generic Controlled Motor

- Define generic capabilities independent of hardware
- Provide implementation for generic interfaces to the best capabilities of hardware
- Provide software simulation where hardware support is lacking
- Adapt functionality and interface to particular hardware by specialization inheritance
- Motor Example: public interface command groups:
 - Initialization and Setup
 - Motion and Trajectory
 - Queries
 - Monitors & Diagnostics





Example: collaborative development for locomotor

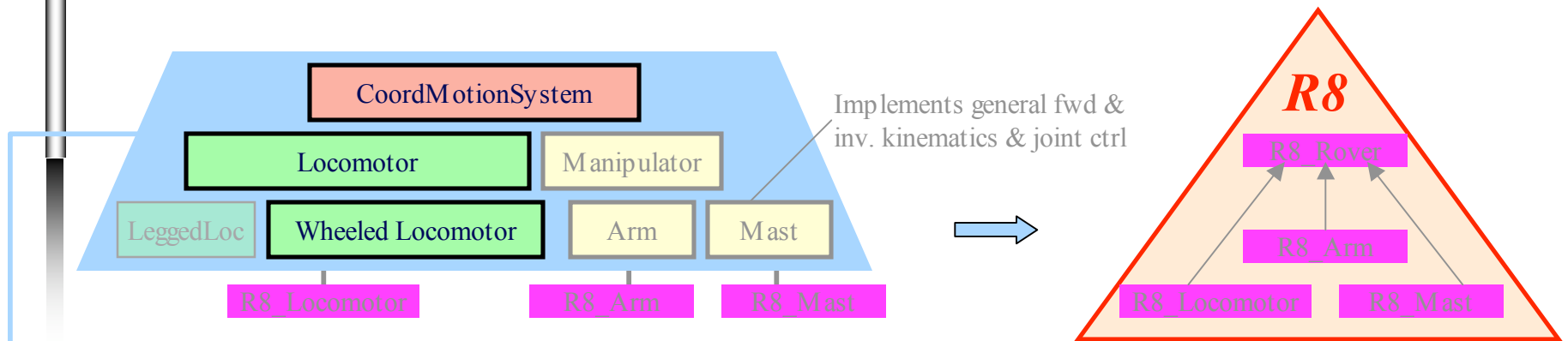




R8 Specific Rover Implementation

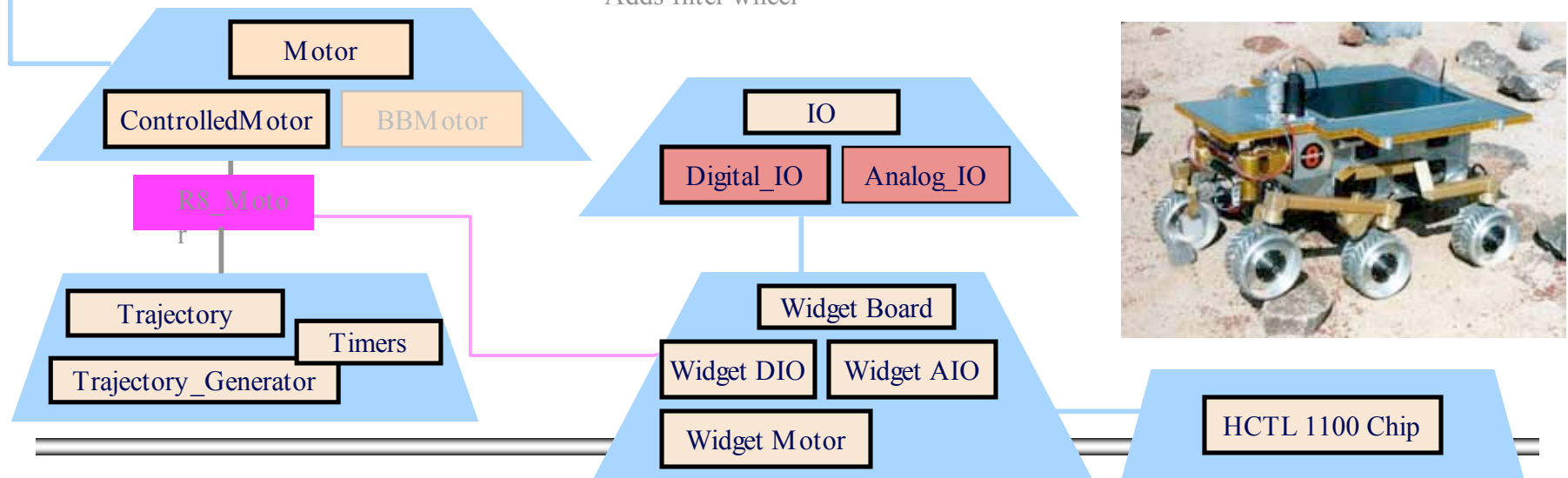


Non reusable Code Reusable Code



- Attaches proper motors
- Restricts Steering to 2 wheels

- Specialized inv. Kinematics (overrides default)
- Attaches proper motors
- Attaches proper cameras for mast
- Adds filter wheel





Capabilities of Wheel Locomotor

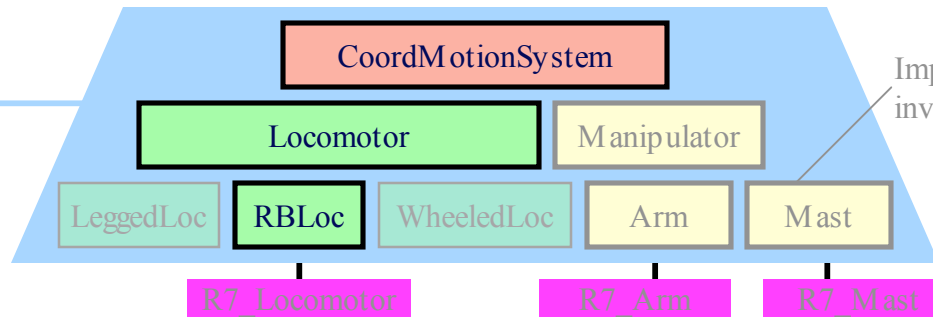


- Type of maneuvers:
 - Straight line motions (fwd / bkwd)
 - Crab maneuvers
 - Arc maneuvers
 - Arc crab maneuvers
 - Rotate-in-place maneuvers (arc turn $r=0$)
- Driving Operation
 - Non-blocking drive commands
 - Multi-threaded access to the Wheel_Locomotor class – e.g. one task can use Wheel_Locomotor for driving while the other for position queries
 - Querying capabilities during all modes of operation. Examples include position updates and state queries
 - Built-in rudimentary pose estimation that assumes vehicle follows commanded motion

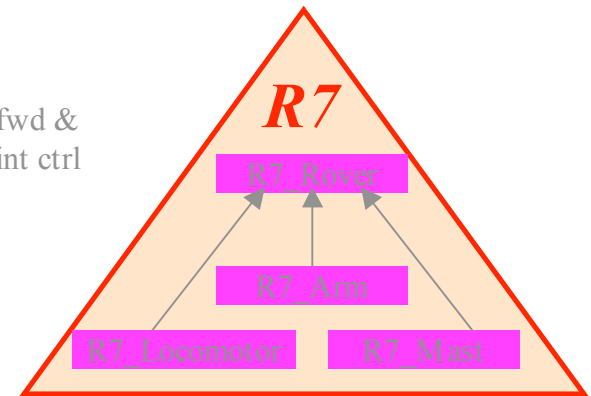


R7 Specific Rover Implementation

Non reusable Code Reusable Code

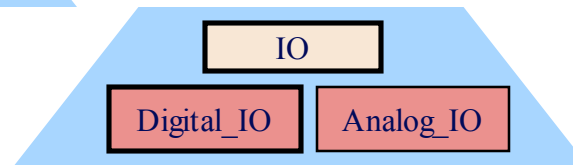
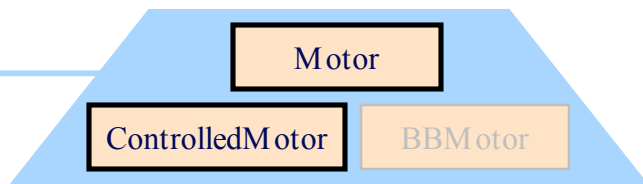


Implements general fwd & inv. kinematics & joint ctrl



- Attaches proper motors
- Restricts Steering to 2 wheels

- Specialized inv. Kinematics (overrides default)
- Attaches proper motors
- Attaches proper cameras for mast
- Adds filter wheel



VPAR10Board

Device Drivers





Why is robotic software “hard”?



- **Software:**
 - Software is large and complex
 - Has lots of diverse functionality
 - Integrates many disciplines
 - Requires real-time runtime performance
 - Talks to hardware
- **Hardware:**
 - Physical and mechanics are different
 - Electrical hardware architecture changes
 - Hardware component capabilities vary



What is CLARAty?



CLARAty is a *unified* and *reusable* **software** that provides robotic functionality and simplifies the integration of new technologies on robotic platforms

A research tool for technology development and maturation