Robotics 1

Programming
Supervision and control architectures

Prof. Alessandro De Luca

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

Robot programming

- real-time operating system
- sensory data reading
- motion control execution
- world modeling
- physical/cognitive interaction with the robot
- fault detection
- error recovery to correct operative conditions
- programming language (data structure + instruction set)

programming environments will depend also on the level at which an operator has access to the functional architecture of the robot

Programming by teaching

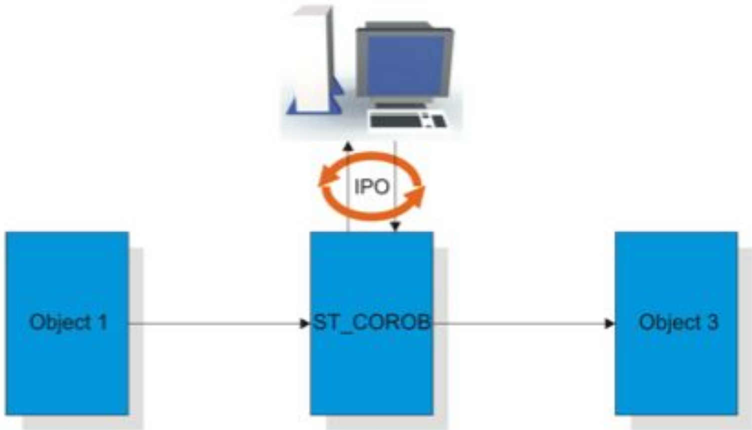
- "first generation" languages
- programming by directly executing (*teaching-by-showing*)
  - the operator guides (manually or via a teach-box) the robot along the desired path (off-line mode)
  - robot joint positions are sampled, stored, and interpolated for later repetition in on-line mode (access to the primitives level)
  - automatic generation of code skeleton (later modifications of parameters is possible): no need of special programming skills
- access to the primitive level
- early applications: spot welding, spray painting, palletizing
- examples of languages: T3 (Milacron), FUNKY (IBM)

Robot-oriented programming

- "second generation" languages: structured programming with characteristics of an interpreted language (interactive programming environment)
- typical instructions of high-level languages are present (e.g., logical branching and while loops)
  - ad-hoc structured robot programming languages (more common)
  - development of robotic libraries in standard languages (preferred)
- access to the action level
- handle more complex applications where the robot needs to cooperate/synchronize with other machines in a work cell
- examples of languages: VAL II (Unimation), AML (IBM), PDL 2 (Comau), KRL (KUKA)

KUKA user interfaces

- Teach pendant
- KRL programming
- Ethernet RSI XML



- Fast Research Interface

Fig. 4-1: Front view of KCP

1	Mode selector switch	10	Numeric keypad
2	Drives ON	11	Softkeys
3	Drives OFF / SSB GUI	12	Start backwards key
4	EMERGENCY STOP button	13	Start key
5	Space Mouse	14	STOP key
6	Right-hand status keys	15	Window selection key
7	Enter key	16	ESC key
8	Arrow keys	17	Left-hand status keys
9	Keypad	18	Menu keys

KRL language

- basic **instruction** set:

Variables and declarations	
DECL	(>>> 10.4.1 "DECL" page 138)
ENUM	(>>> 10.4.2 "ENUM" page 140)
IMPORT ... IS	(>>> 10.4.3 "IMPORT ... IS" page 141)
STRUC	(>>> 10.4.4 "STRUC" page 141)

Motion programming	
CIRC	(>>> 10.5.1 "CIRC" page 143)
CIRC_REL	(>>> 10.5.2 "CIRC_REL" page 144)
LIN	(>>> 10.5.3 "LIN" page 146)
LIN_REL	(>>> 10.5.4 "LIN_REL" page 146)
PTP	(>>> 10.5.5 "PTP" page 148)
PTP_REL	(>>> 10.5.6 "PTP_REL" page 148)

Program execution control	
CONTINUE	(>>> 10.6.1 "CONTINUE" page 150)
EXIT	(>>> 10.6.2 "EXIT" page 150)
FOR ... TO ... ENDFOR	(>>> 10.6.3 "FOR ... TO ... ENDFOR" page 150)
GOTO	(>>> 10.6.4 "GOTO" page 151)
HALT	(>>> 10.6.5 "HALT" page 152)
IF ... THEN ... ENDIF	(>>> 10.6.6 "IF ... THEN ... ENDIF" page 152)
LOOP ... ENDLOOP	(>>> 10.6.7 "LOOP ... ENDLOOP" page 153)
REPEAT ... UNTIL	(>>> 10.6.8 "REPEAT ... UNTIL" page 153)
SWITCH ... CASE ... ENDSWITCH	(>>> 10.6.9 "SWITCH ... CASE ... ENDSWITCH" page 154)
WAIT ... FOR	(>>> 10.6.10 "WAIT FOR" page 155)
WAIT ... SEC	(>>> 10.6.11 "WAIT SEC" page 156)
WHILE ... ENDWHILE	(>>> 10.6.12 "WHILE ... ENDWHILE" page 156)

Inputs/outputs	
ANIN	(>>> 10.7.1 "ANIN" page 157)
ANOUT	(>>> 10.7.2 "ANOUT" page 158)
DIGIN	(>>> 10.7.3 "DIGIN" page 159)
PULSE	(>>> 10.7.4 "PULSE" page 160)
SIGNAL	(>>> 10.7.5 "SIGNAL" page 164)

Subprograms and functions	
RETURN	(>>> 10.8.1 "RETURN" page 165)

Interrupt programming	
BRAKE	(>>> 10.9.1 "BRAKE" page 166)
INTERRUPT	(>>> 10.9.2 "INTERRUPT" page 166)
INTERRUPT ... DECL ... WHEN ... DO	(>>> 10.9.3 "INTERRUPT ... DECL ... WHEN ... DO" page 167)
RESUME	(>>> 10.9.4 "RESUME" page 169)

Path-related switching actions (=Trigger)	
TRIGGER WHEN DISTANCE	(>>> 10.10.1 "TRIGGER WHEN DISTANCE" page 170)
TRIGGER WHEN PATH	(>>> 10.10.2 "TRIGGER WHEN PATH" page 173)

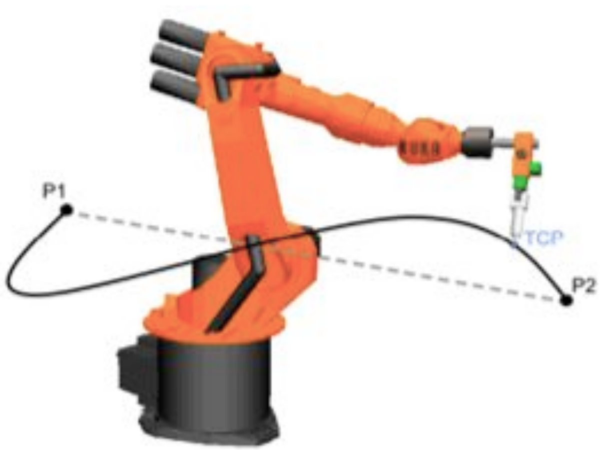
Communication
(>>> 10.11 "Communication" page 176)

System functions	
VARSTATE()	(>>> 10.12.1 "VARSTATE()" page 176)

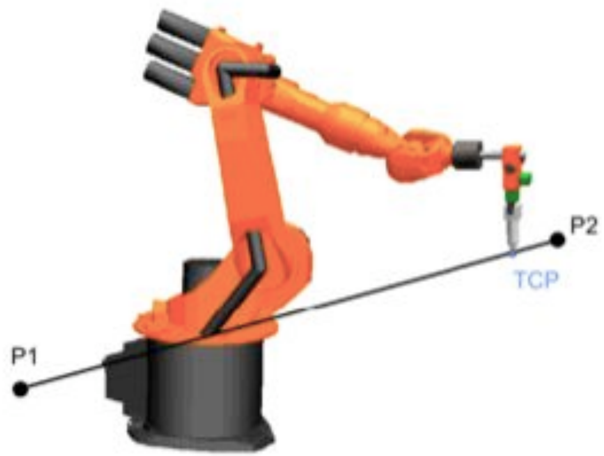
- basic **data** set: frames, vectors + DECLaration

KRL language

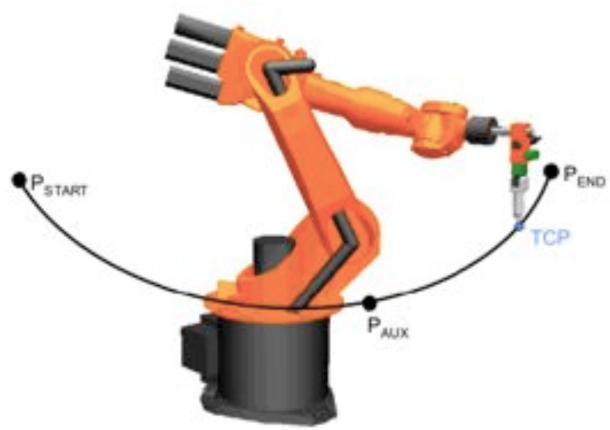
- typical motion primitives



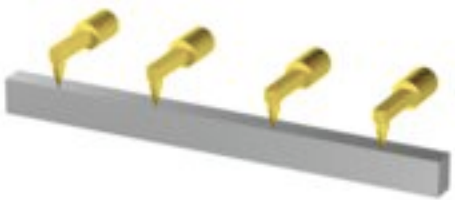
PTP motion
(point-to-point, linear
in joint space)



LIN motion
(linear in
Cartesian space)

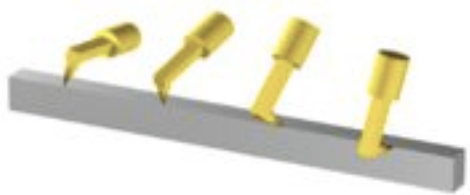


CIRC motion
(circular in
Cartesian space)



CONST orientation

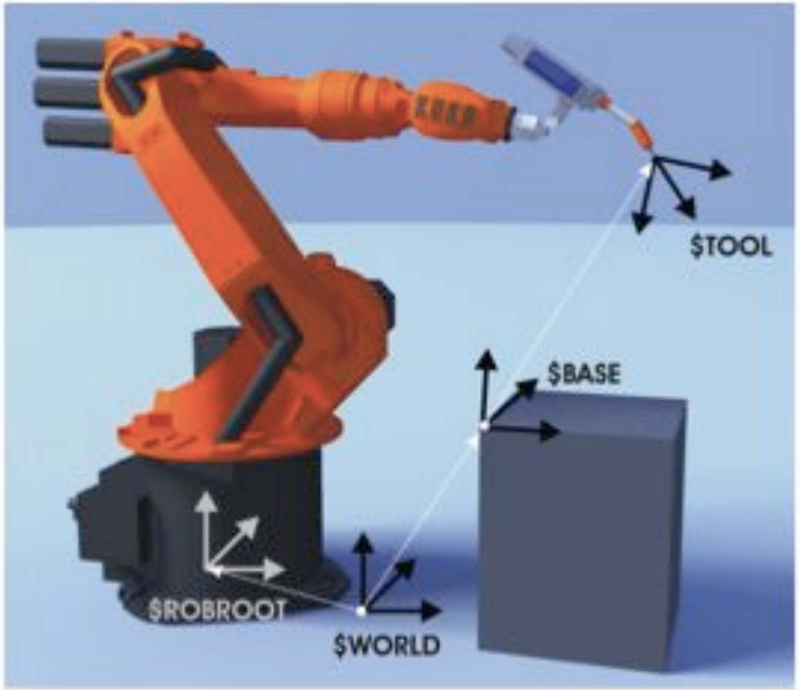
end-effector
orientation



PTP motion
(linear in RPY angles)

KRL language

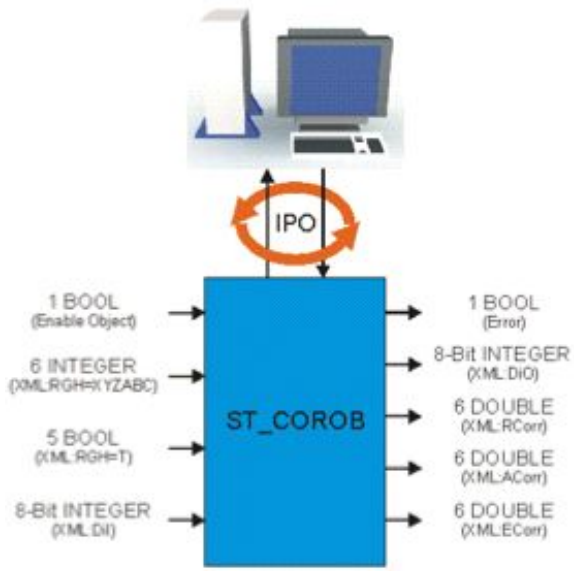
- multiple coordinate frames (in Cartesian space) and jogging of robot joints



KUKA Ethernet RSI

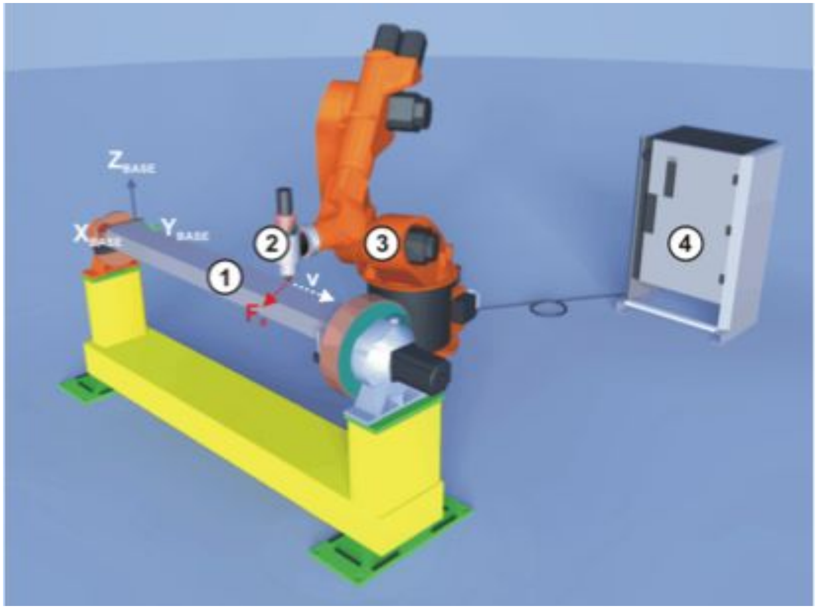
Robot Sensor Interface

- cyclical data transmission from the robot controller to an external system (e.g., position data, axis angles, operating mode, etc.) and vice versa (e.g., sensor data) in the interpolation cycle of 12 ms
- influencing the robot in the interpolation cycle by means of an external program
- direct intervention in the path planning of the robot
- recording/diagnosis of internal signals
- communication module with access to standard Ethernet via TCP/IP protocol as XML strings (real-time capable link)
- freely definable inputs and outputs of the communication object
- data exchange timeout monitoring



Example of RSI use - 1

- deburring task with robot motion controlled by a force sensor



① work piece to be deburred along the edge under force control

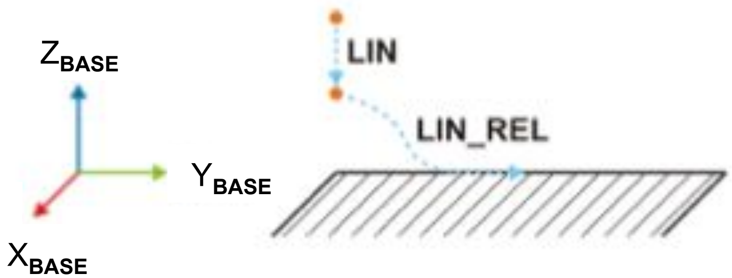
② tool with force sensor

③ robot

④ robot controller

F_X measured force in the X direction of the BASE coordinate system (perpendicular to the programmed path)

V direction of motion



LIN_REL = linear Cartesian path relative to an initial position (specified here by the force sensor signal)

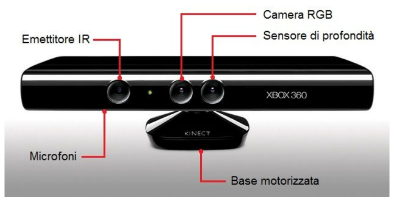
Example of RSI use - 2

- redundancy resolution on cyclic Cartesian paths
  - task involves position only (m=3, n=6 for the KUKA KR5 Sixx)
- without joint range limits or including virtual limits

video

Example of RSI use - 3

- human-robot interaction through vocal and gesture commands
- voice and human gestures acquired through a Kinect sensor



Kinect RGB-D sensor
(with microphone)

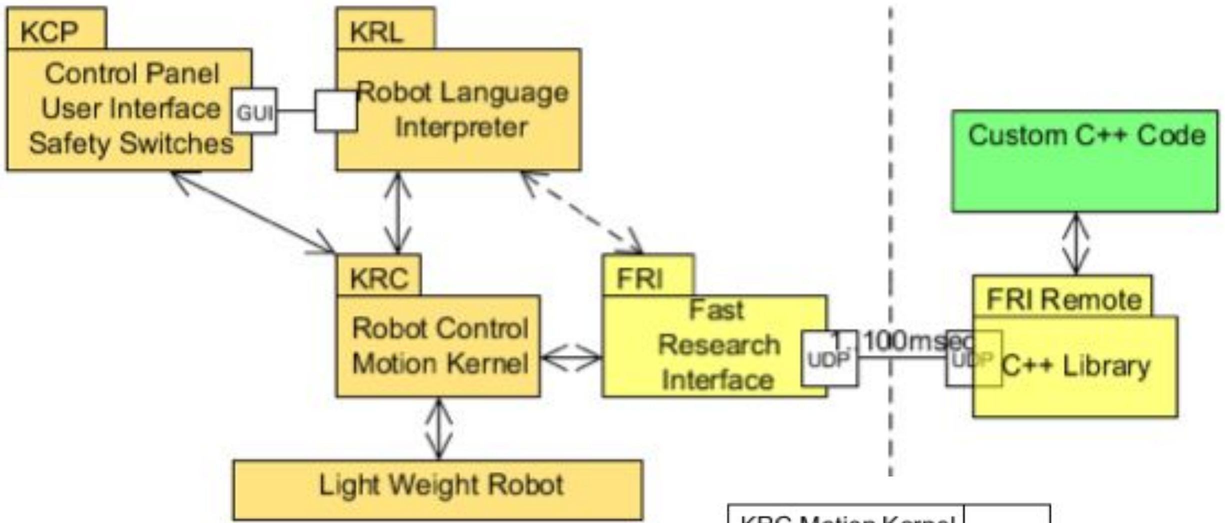
simple vocabulary, e.g.:
- listen to me
- give me
- follow
  - right/left hand
  - the nearest hand
- thank you
- stop collaboration

video

Fast Research Interface (FRI)

for KUKA Light Weight Robot (LWR-IV)

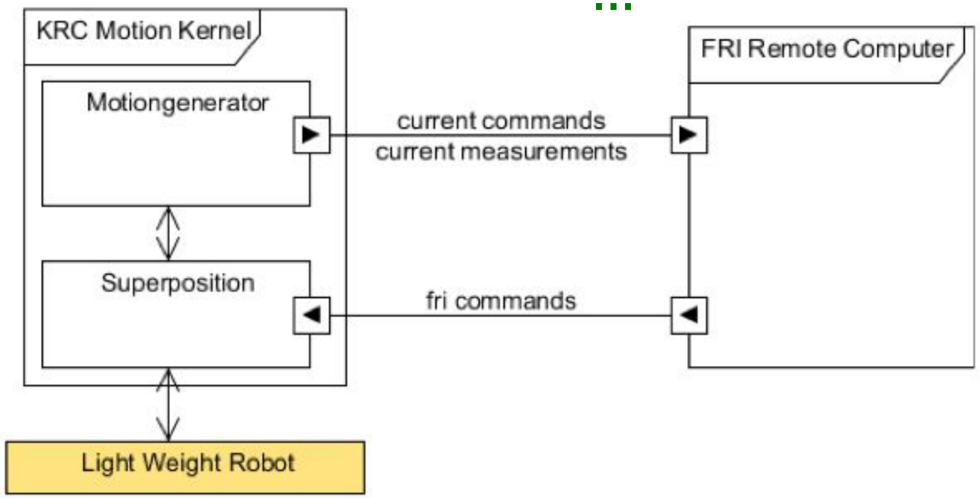
- UDP socket communication up to 1 KHz (1÷100 ms cycle time)



here, we develop
our C++/ROS code for:

- trajectory planning
- kinematic control
- redundancy resolution
- torque/dynamic control
- physical HRI
- ...

available
at DIAG
Robotics Lab
since Sep 2012



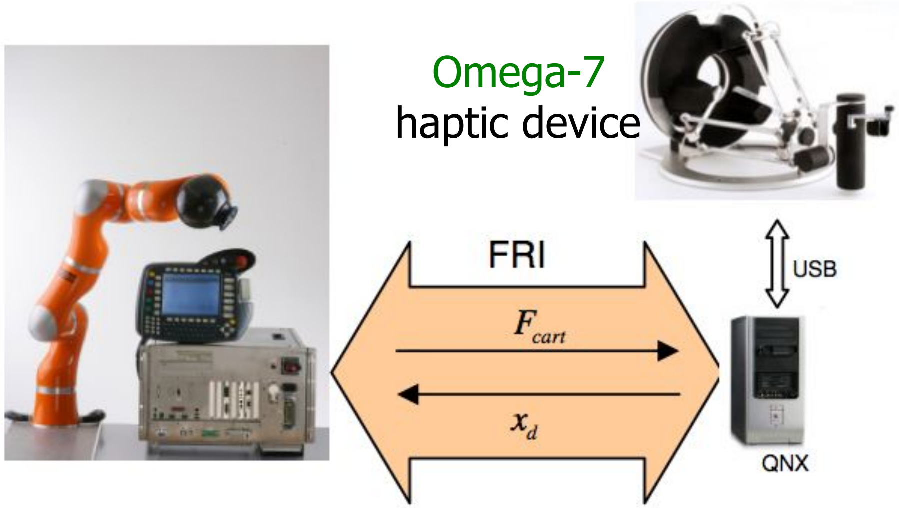
Kinematic control using the FRI

KUKA Light Weight Robot (LWR-IV)

- joint velocity commands that mimic second-order control laws (defined in terms of acceleration or torques), exploiting task redundancy of the robot
- discrete-time implementation is simpler and still very accurate

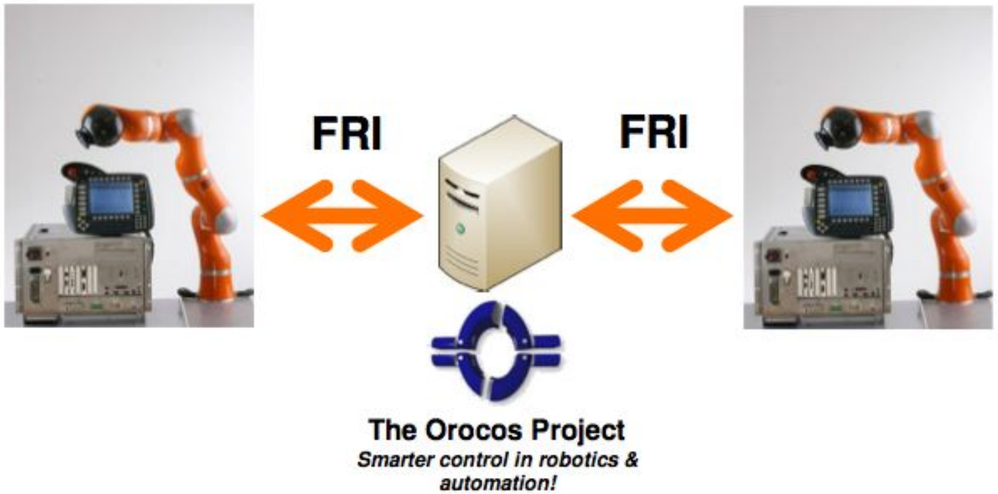
video

Other uses of the FRI



- haptic feedback to the user

- coordinated dual-arm motion



Robot research software

- a (partial) list of open source robot software
  - for simulation and/or real-time control
  - for interfacing with devices and sensors
  - research oriented

Player/Stage playerstage.sourceforge.net

  - networked robotics server (running on Linux, Mac OS X) as an abstraction layer supporting a variety of hardware + 2D robot simulation environment
  - Gazebo: 3D robot simulator (with ODE physics engine and OpenGL rendering), now an independent project

VREP (edu version) www.coppeliarobotics.com

  - each object/model controlled via an embedded script, a plugin, a ROS node, a remote API client, or a custom solution
  - controllers written in C/C++, Python, Java, Matlab, ...

Robot research software (cont'd)

Robotics Toolbox (free addition to Matlab) www.petercorke.com

- study and simulation of kinematics, dynamics, and trajectory generation for serial-link manipulators

OpenRDK openrdk.sourceforge.net

- "agents": modular processes dynamically activated, with blackboard-type communication (repository)

ROS (Robot Operating System) www.ros.org/wiki

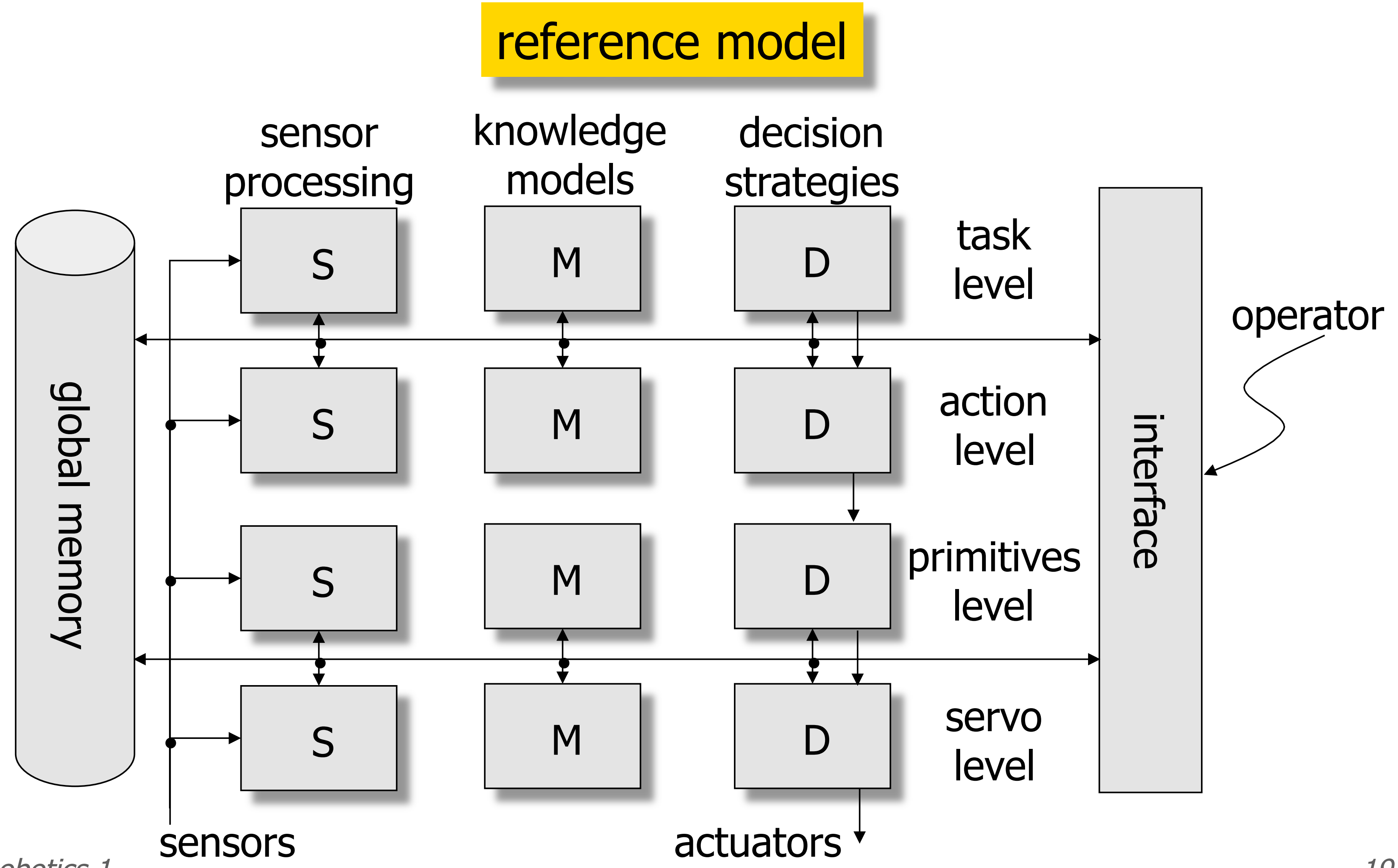
- middleware with: hardware abstraction, device drivers, libraries, visualizers, message-passing, package management
- "nodes": executable code (in Python, C++) running with a publish/subscribe communication style

Pyro (Python Robotics) pyrorobotics.org

Task-oriented programming

- “third generation” languages (for research, not yet available on the market)
- similar to object-oriented programming
- task specified by high-level instructions performing actions on the parts present in the scene (artificial intelligence)
- understanding and reasoning about a dynamic environment around the robot
- access to the task level

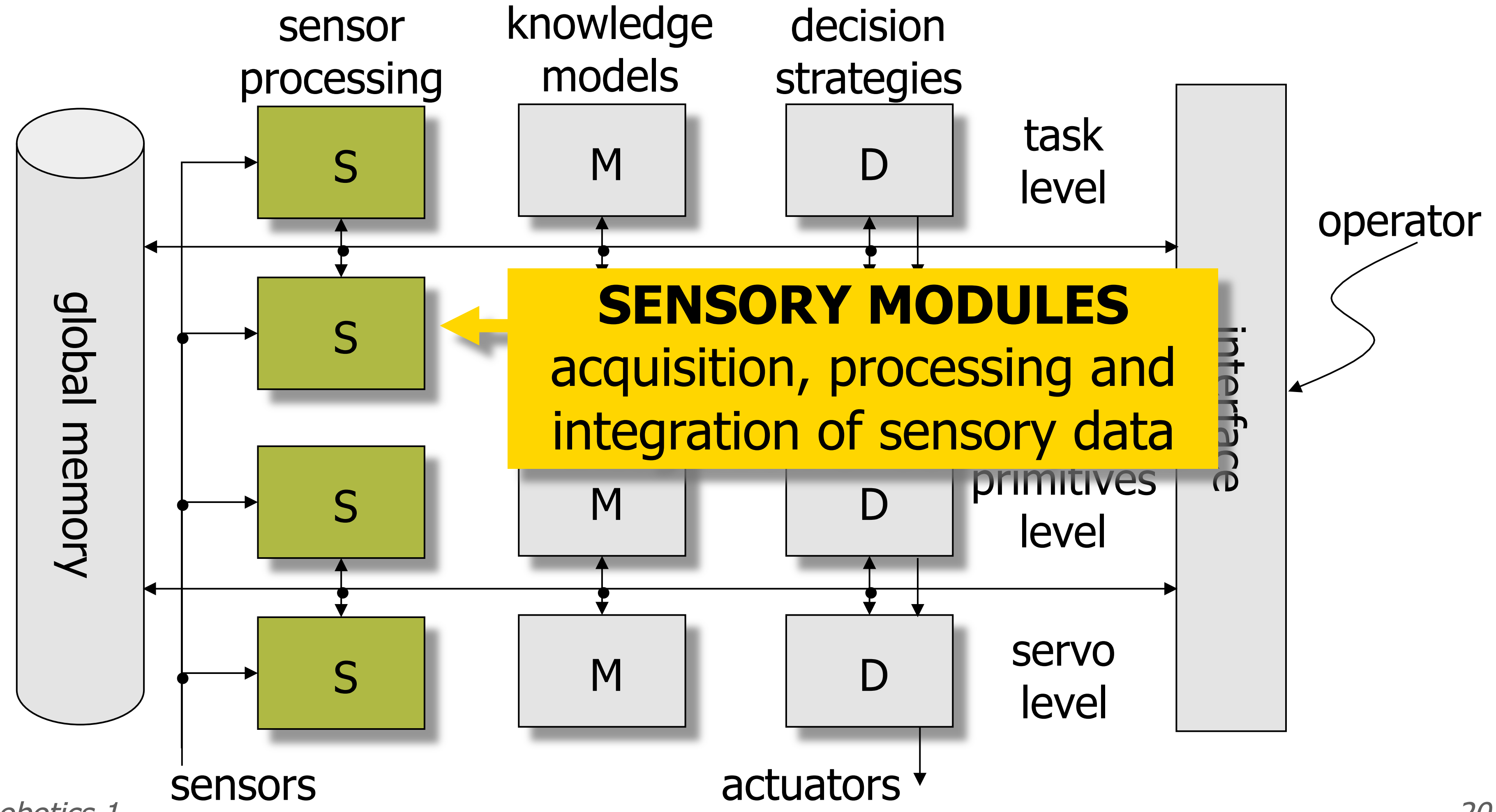
Functional control architecture



Functional architecture: Modules

horizontal decomposition

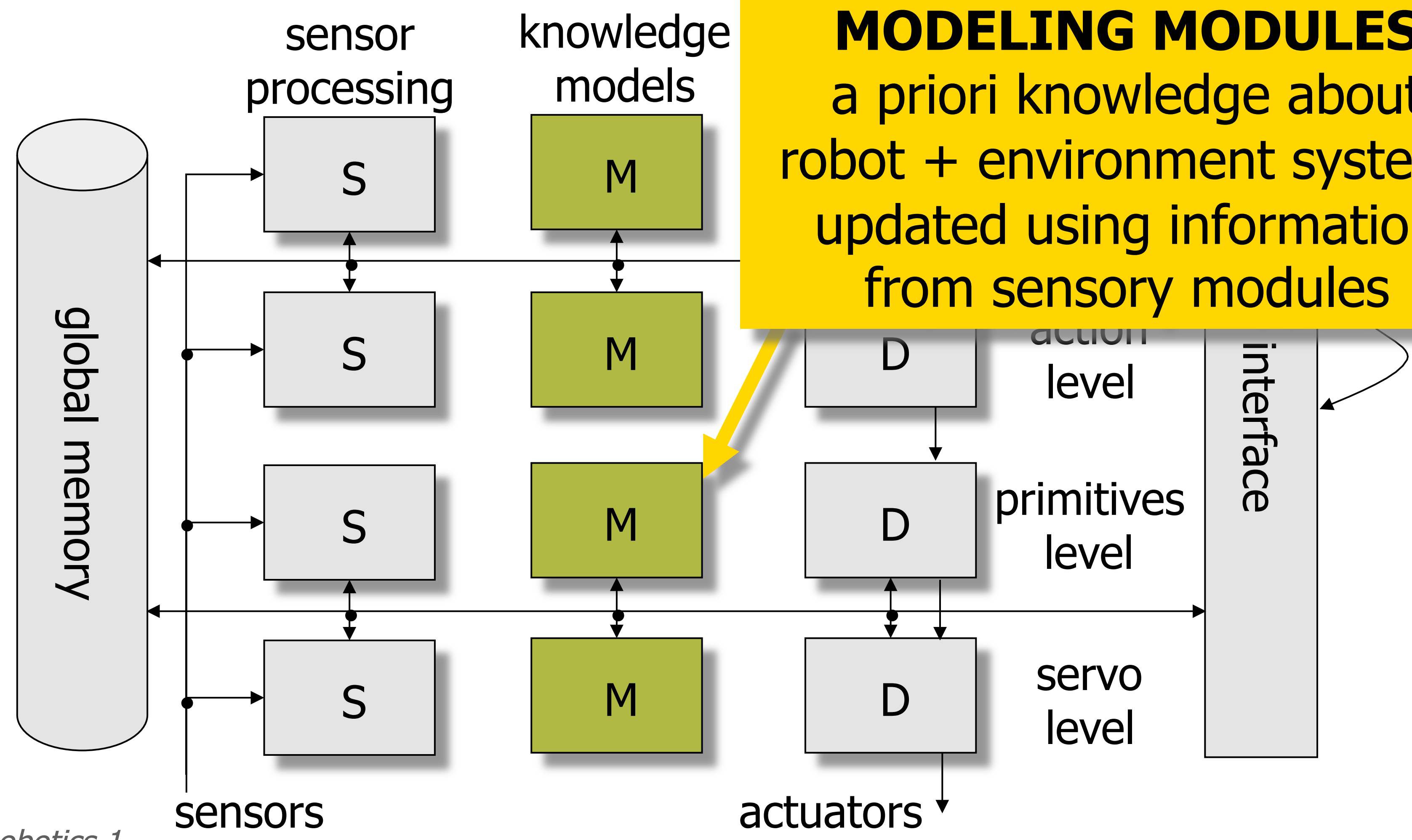
reference model



Functional architecture: Modules

horizontal decomposition

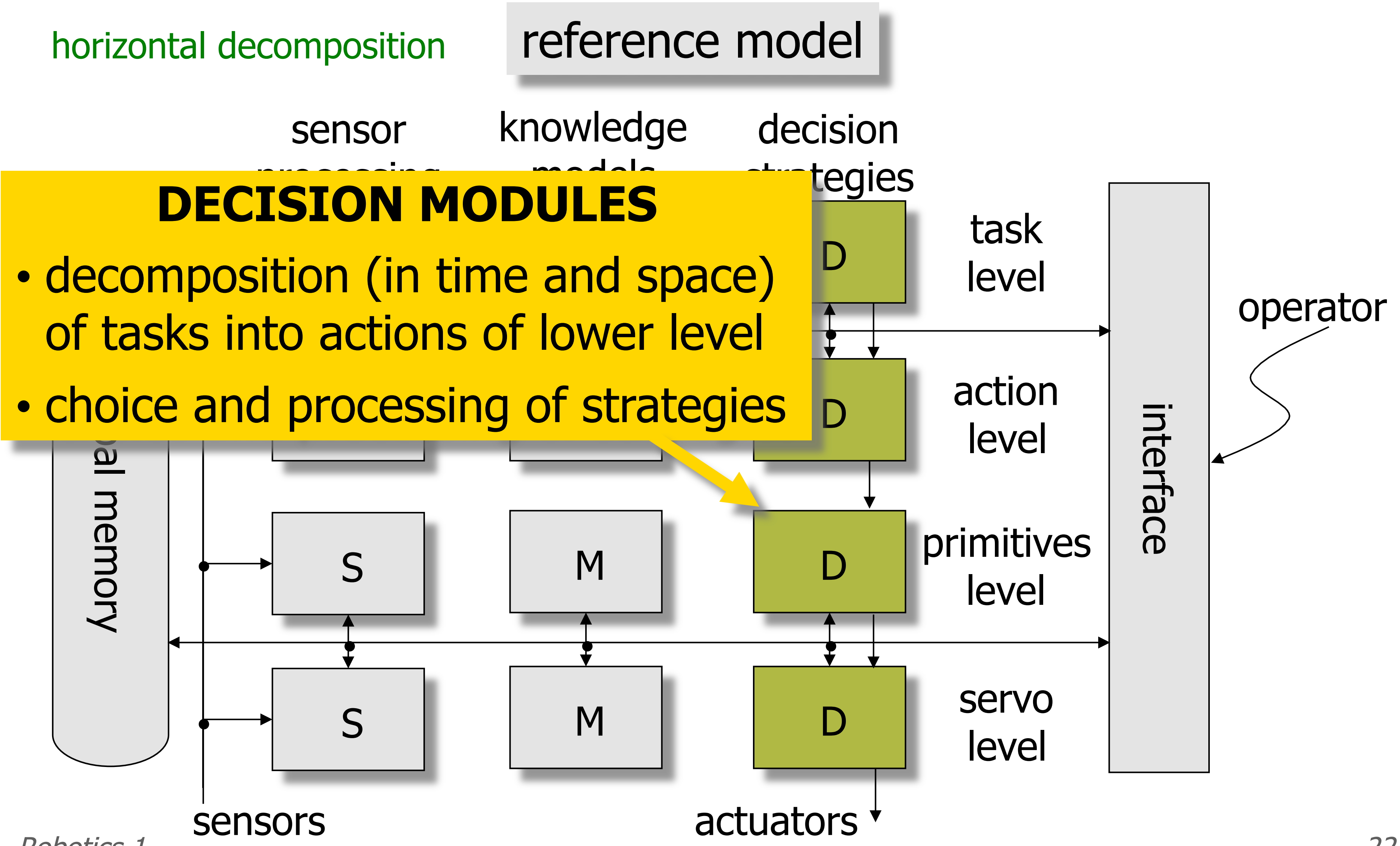
reference model



MODELING MODULES
a priori knowledge about robot + environment system, updated using information from sensory modules

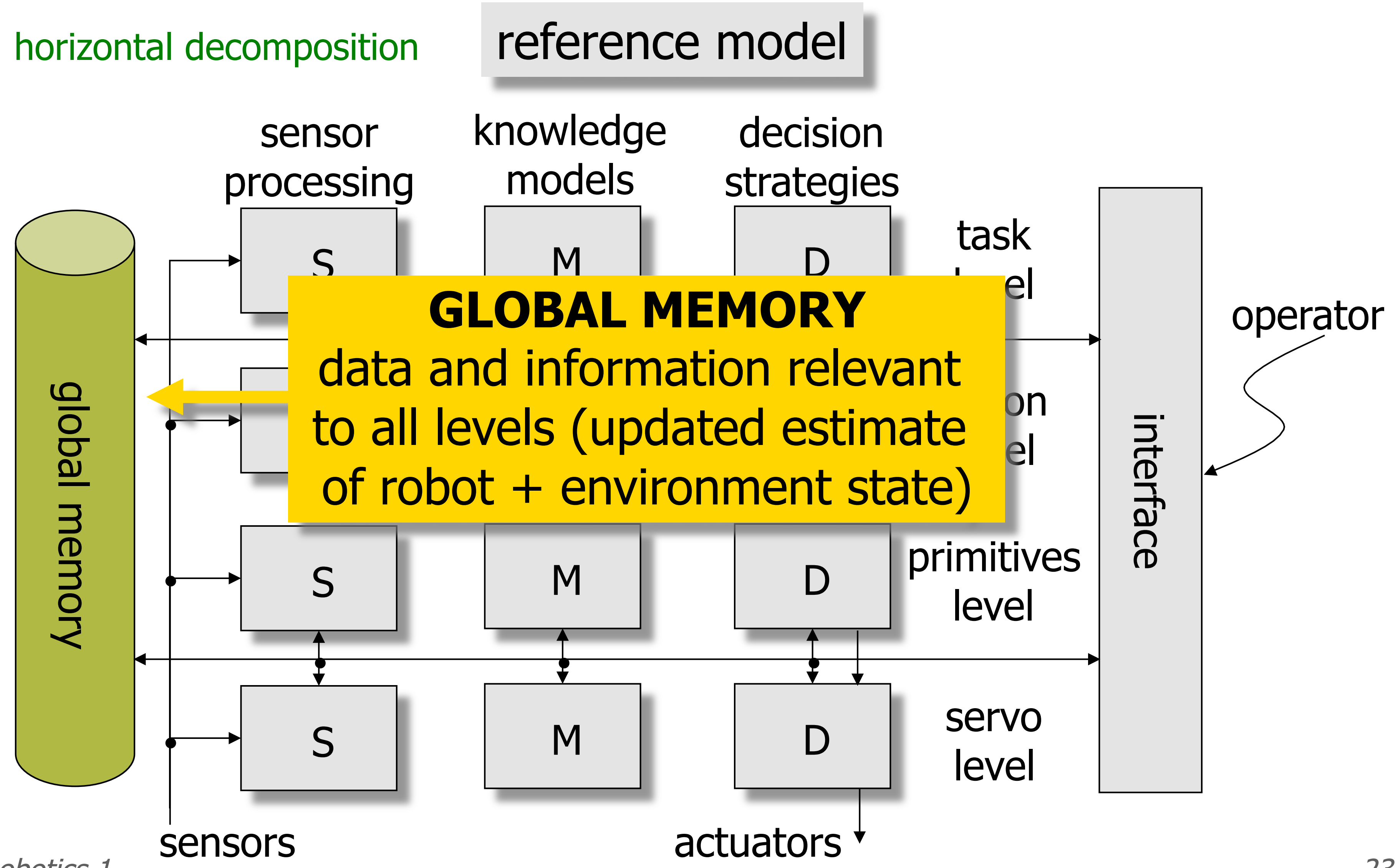
Functional architecture: Modules

horizontal decomposition



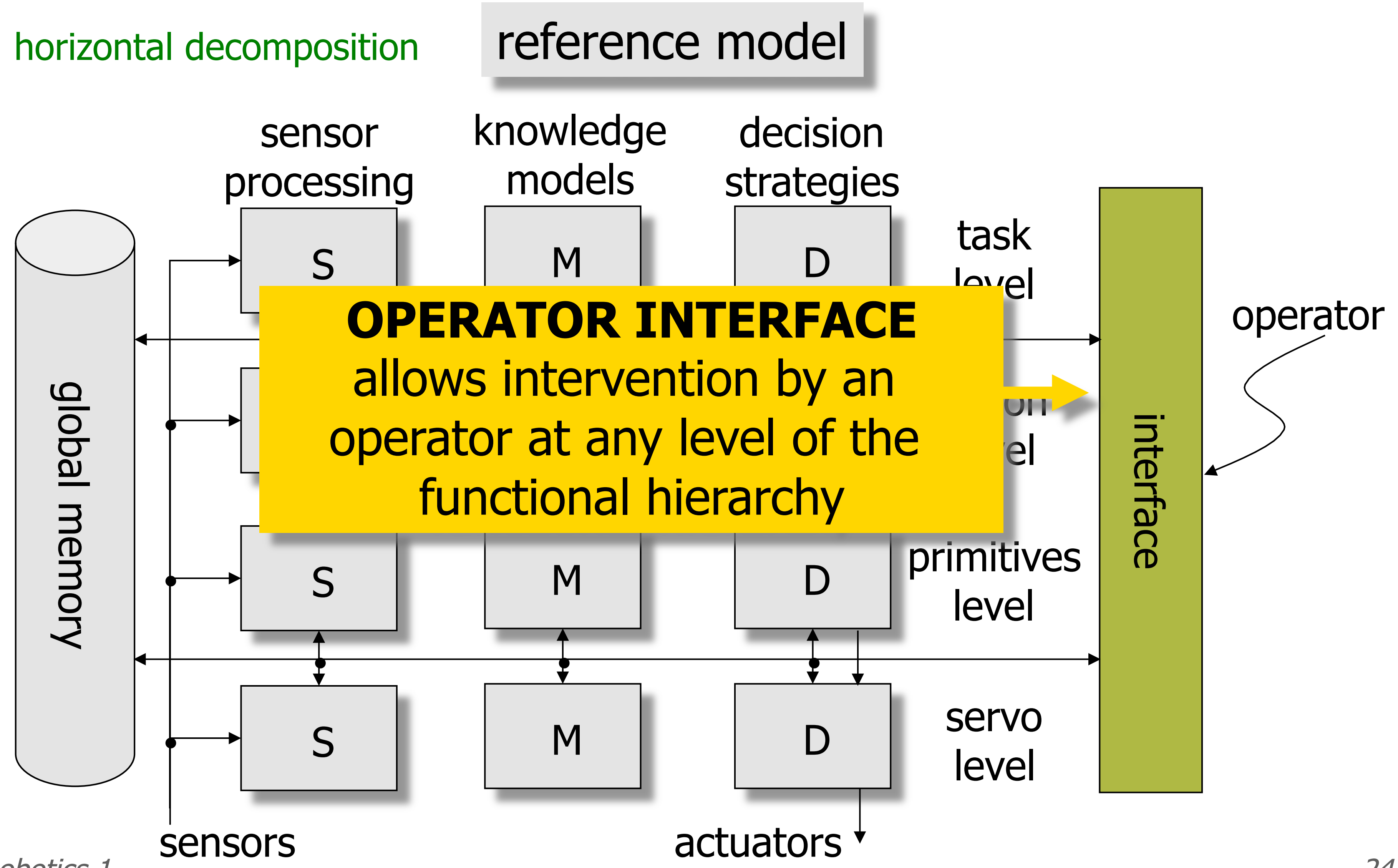
Functional architecture: Modules

horizontal decomposition

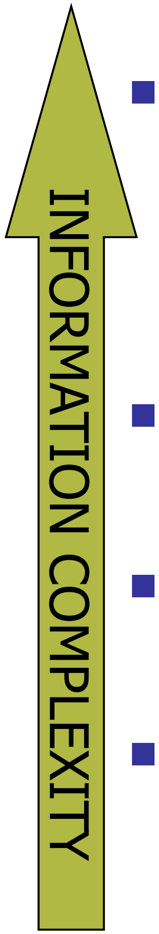


Functional architecture: Modules

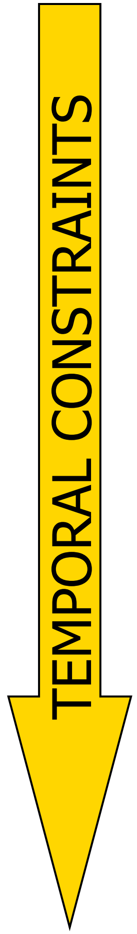
horizontal decomposition



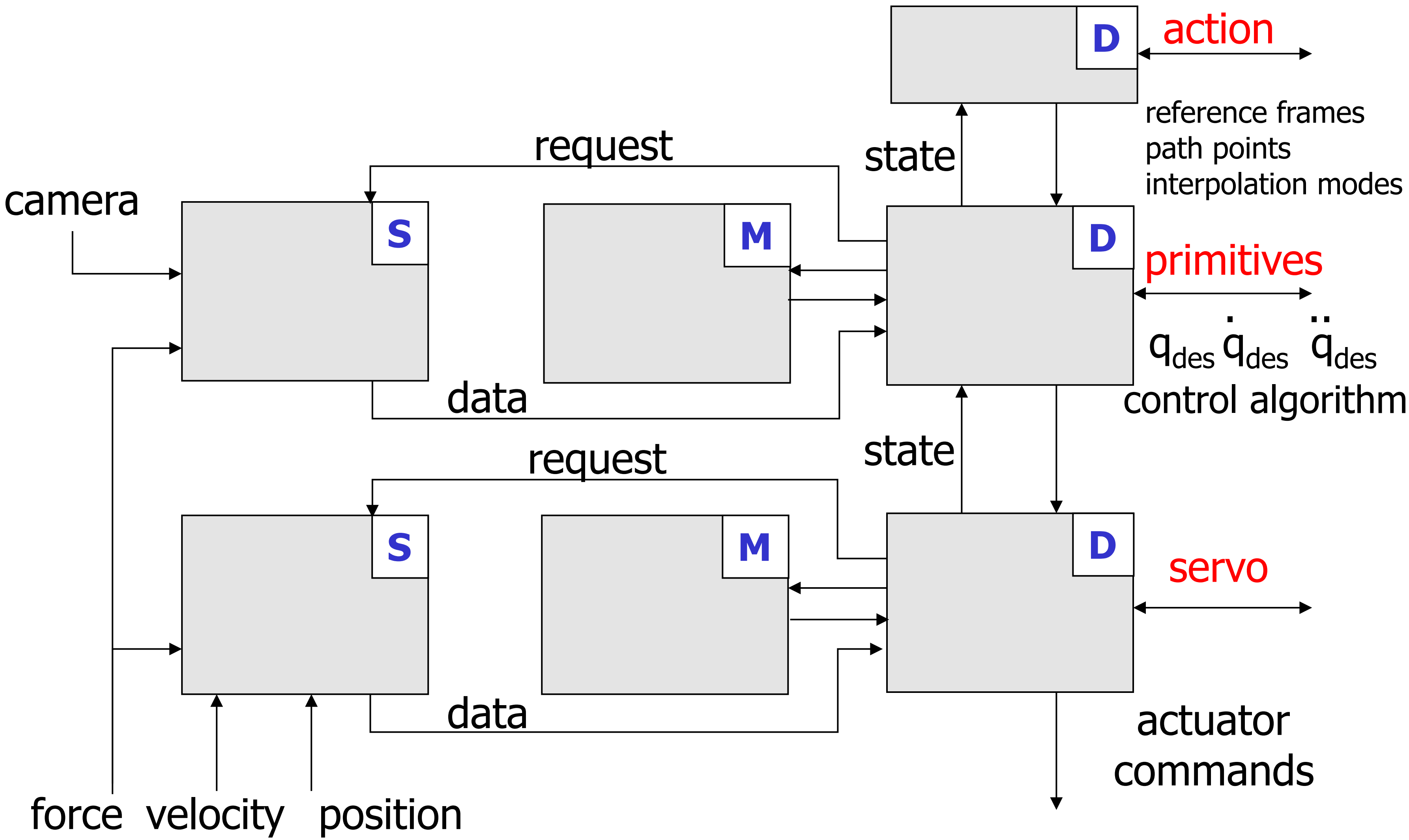
Reference model: Levels



- task level: objective of the task (as specified by the user) analyzed and decomposed into actions (based on knowledge models about the robot and the environment systems)
- action level: symbolic commands converted into sequences of intermediate configurations
- primitives level: reference trajectories generation for the servo level, choice of a control strategy
- servo level: implementation of control algorithms, real-time computation of driving commands for the actuating servomotors

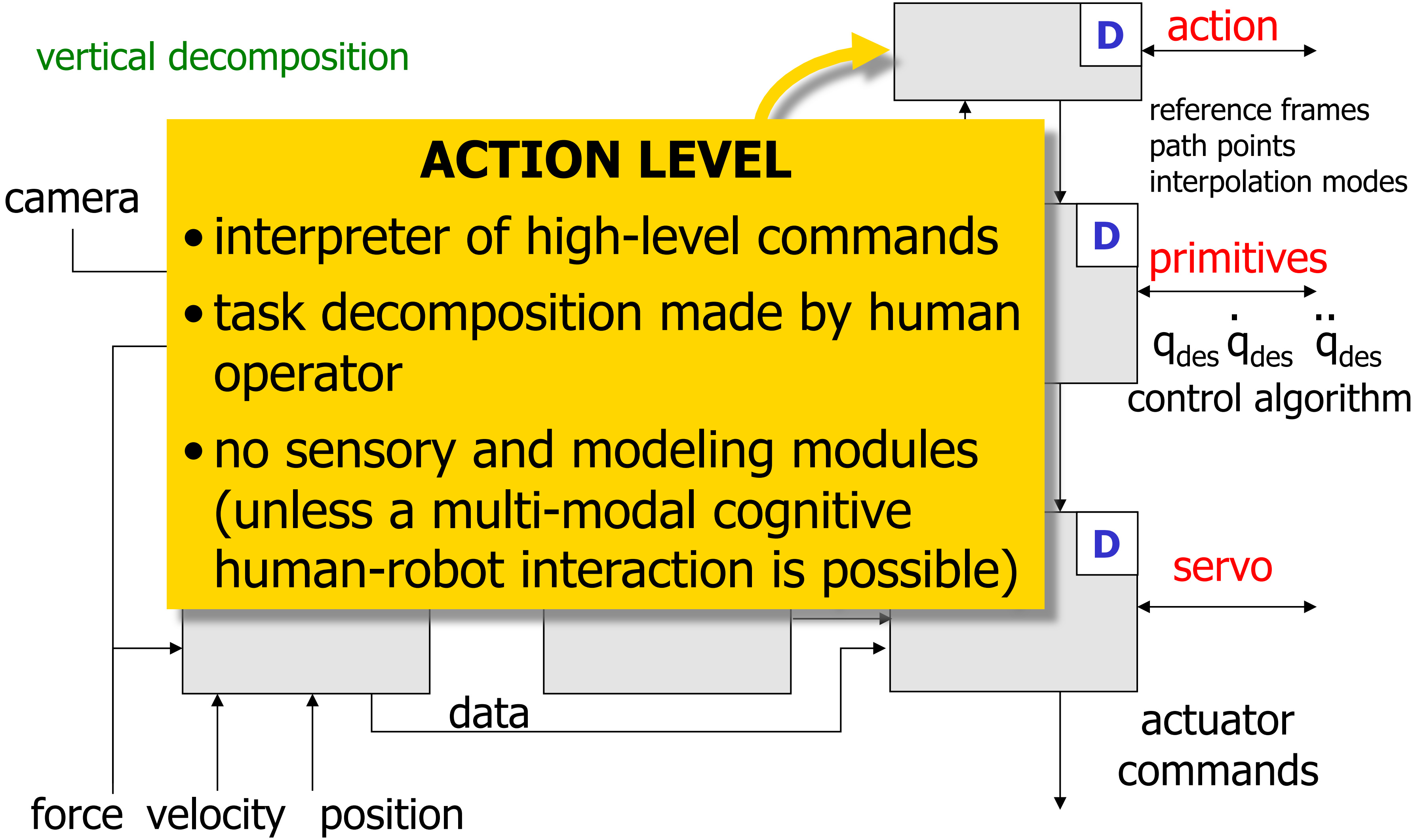


A functional architecture for industrial robots



A functional architecture for industrial robots

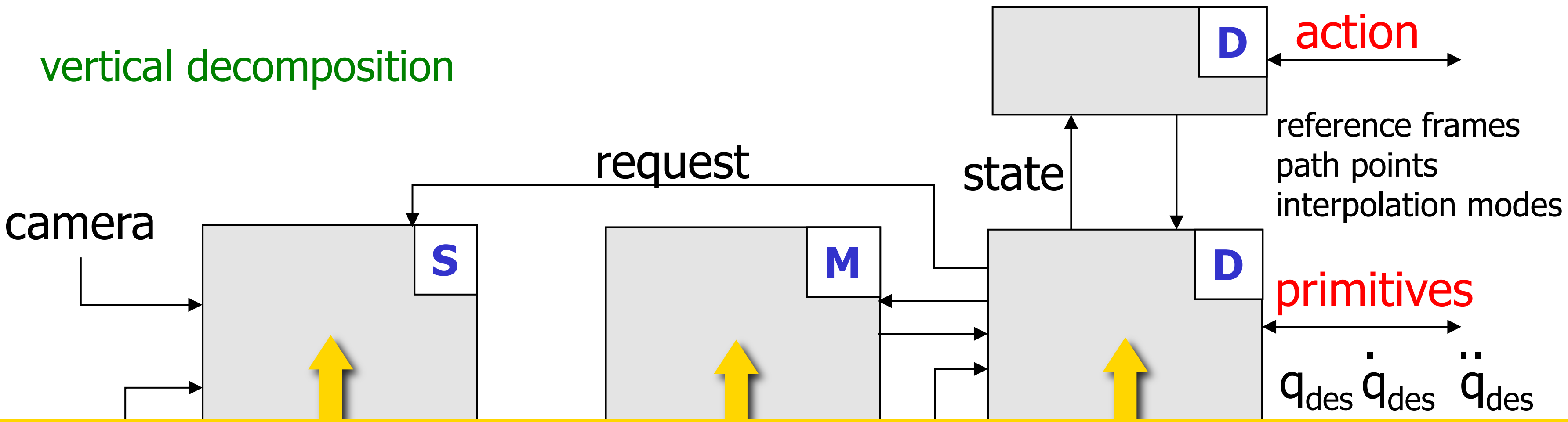
vertical decomposition



ACTION LEVEL

- interpreter of high-level commands
- task decomposition made by human operator
- no sensory and modeling modules (unless a multi-modal cognitive human-robot interaction is possible)

A functional architecture for industrial robots

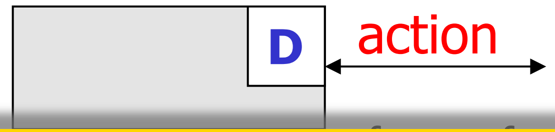


PRIMITIVES LEVEL

- **S:** (only for an active interaction with the environment) world geometry, interaction state
- **M:** direct and inverse kinematics, dynamic models
- **D:** command encoding, path generation, trajectory interpolation, kinematic inversion, analysis of servo state, emergency handling

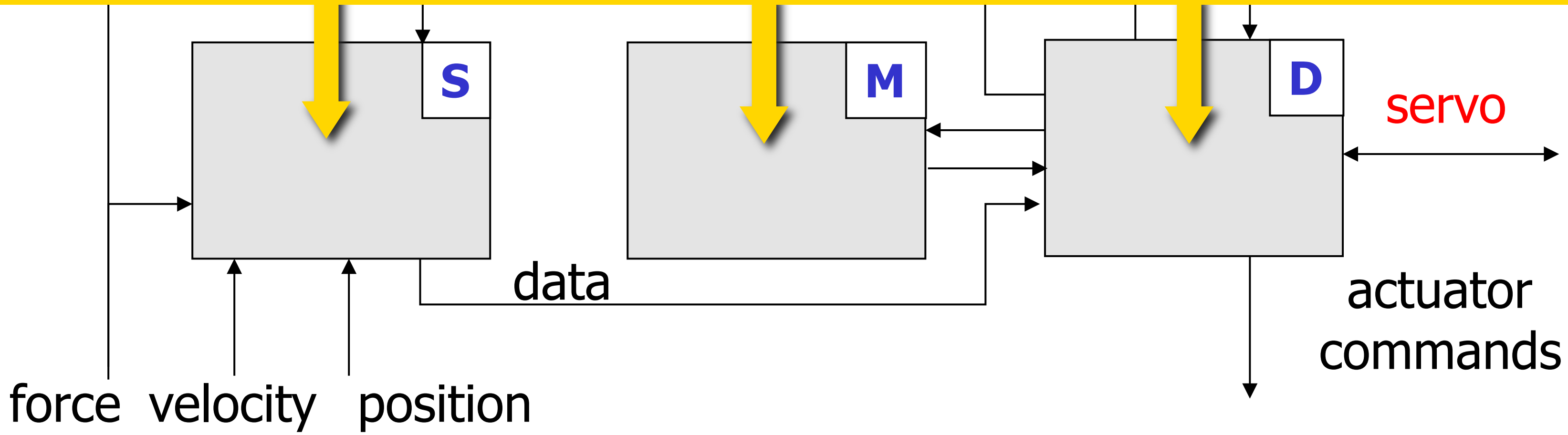
A functional architecture for industrial robots

vertical decomposition

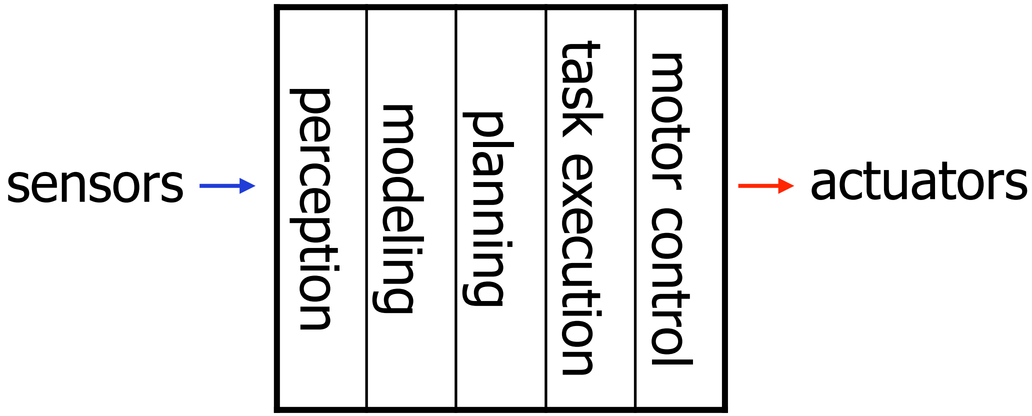


SERVO LEVEL

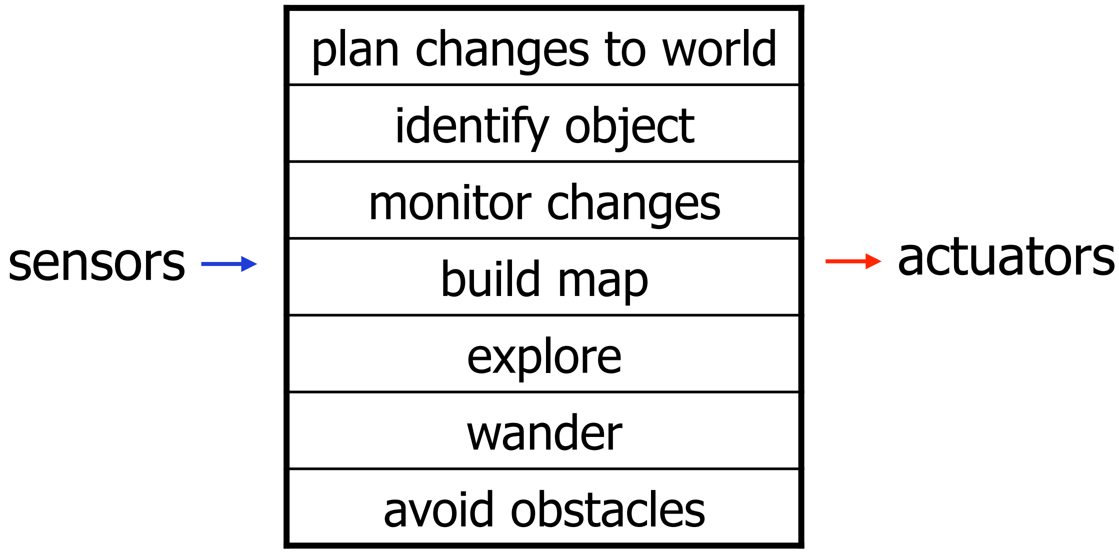
- **S:** signal conditioning, internal state of manipulator, state of interaction with environment
- **M:** direct kinematics, Jacobian, inverse dynamics
- **D:** command encoding, micro-interpolation, error handling, digital control laws, servo interface



Interaction among modules

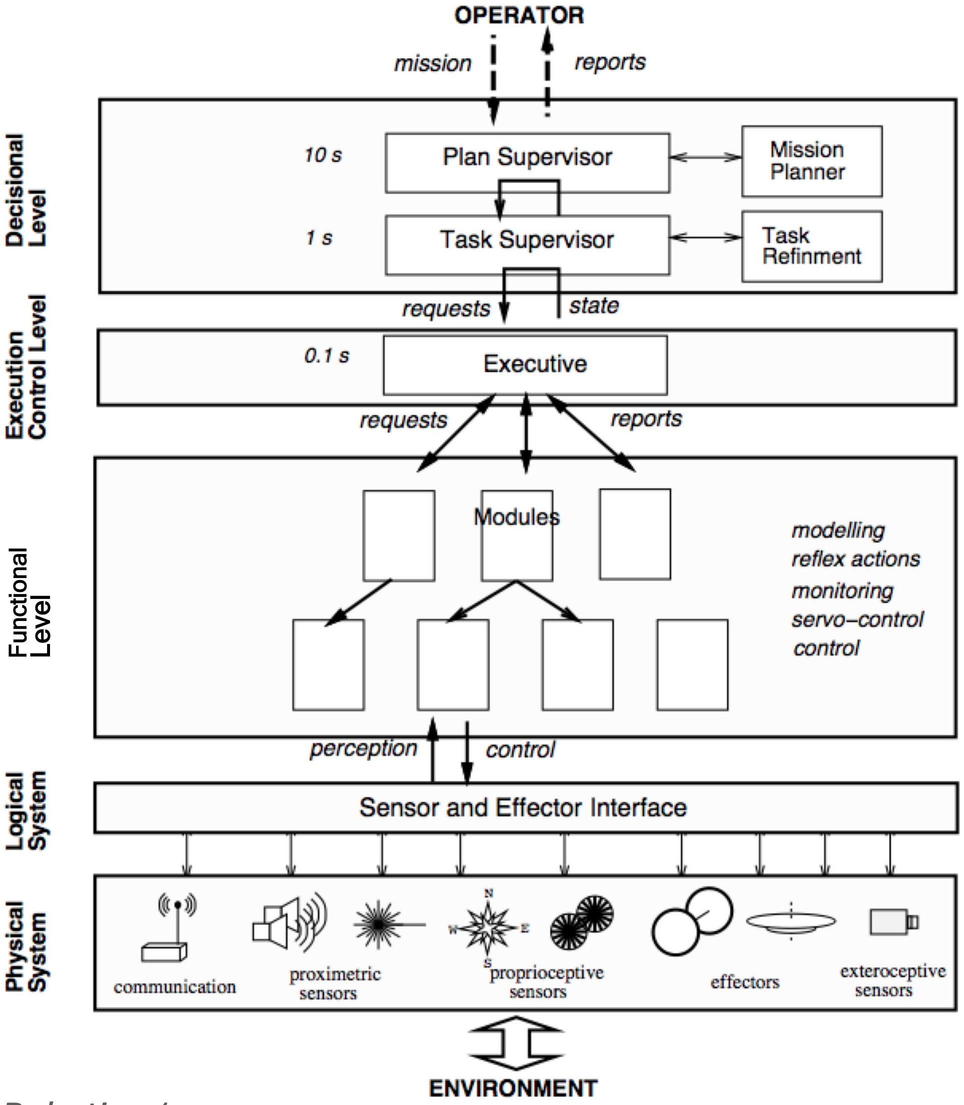


horizontal
activation
(sequential)



vertical
activation on demand
(subsumption)

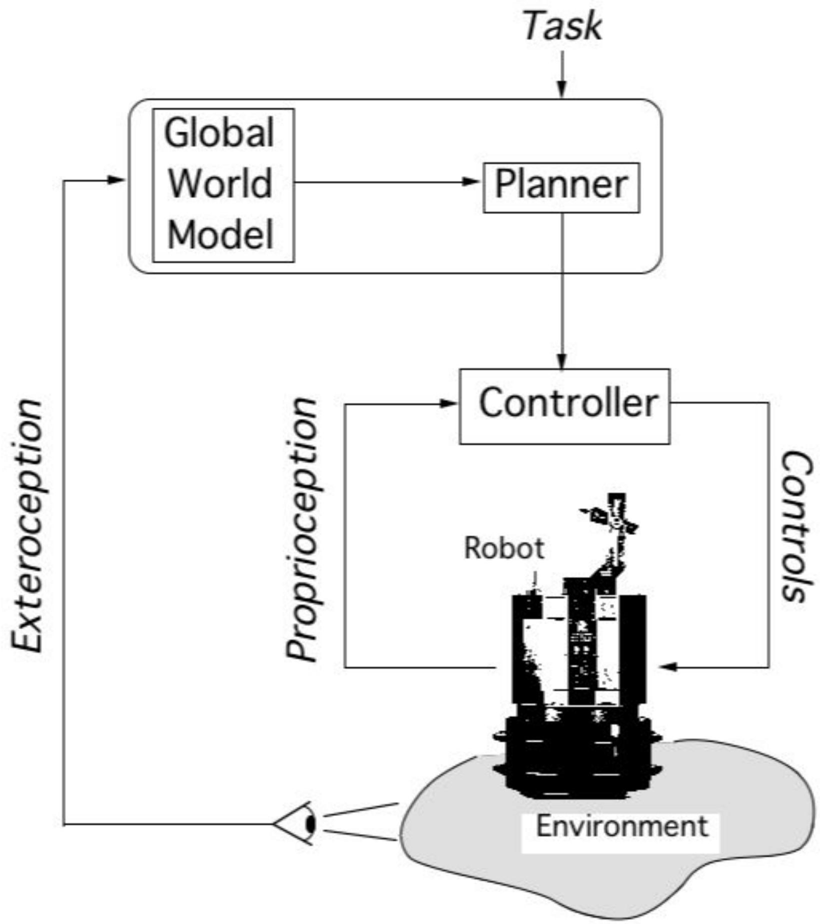
LAAS architecture



- alternative example by LAAS/CNRS in Toulouse
- five levels
  - decision
  - execution (synchronization)
  - functional (modules)
  - logical for interface
  - physical devices

R. Alami *et al.*
"An Architecture for Autonomy,"
Int. J. of Robotics Research, 1998

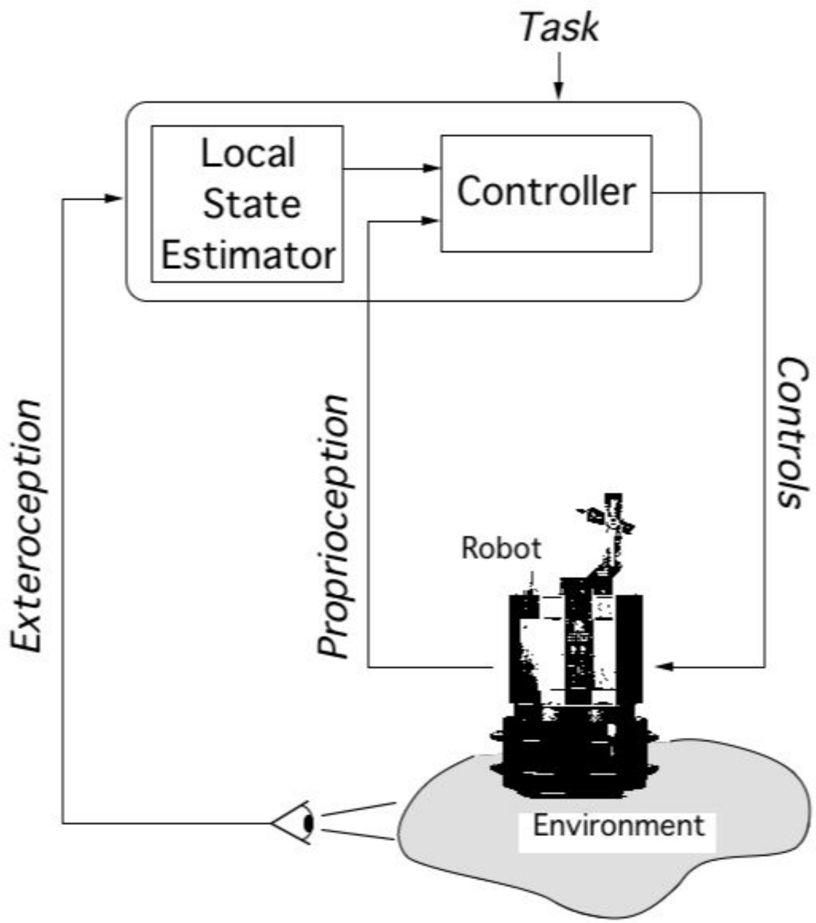
Development of architectures - 1



example: a navigation task for a wheeled mobile robot

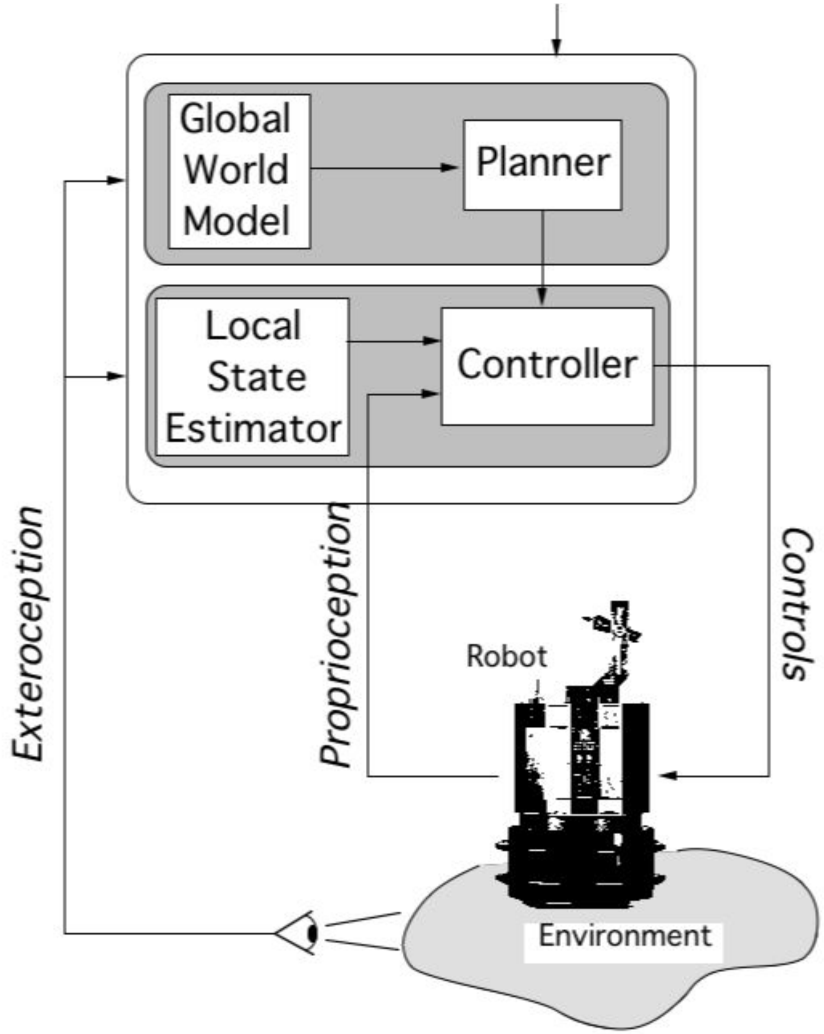
- hierarchical system
  - initial localization
  - off-line planning
  - on-line motion control
  - possible acquisition/update of a model of the environment = map (at a slow time scale)

Development of architectures - 2



- pure reactive system
  - global positioning task (goal)
  - on-line estimate of the local environment (unknown)
  - local reaction strategy for obstacle avoidance and guidance toward the goal

Development of architectures - 3



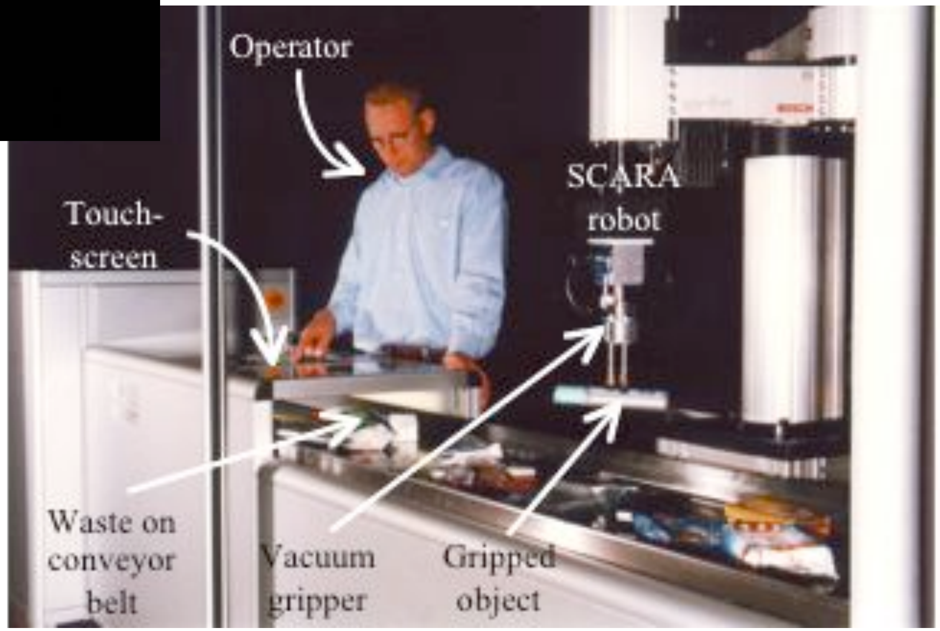
- hybrid system
  - SLAM = simultaneous localization and mapping
  - navigation/exploration on the current model (map)
  - sensory data fusion
  - on-line motion control

IPA robotic cell for garbage collection
and separation for recycling

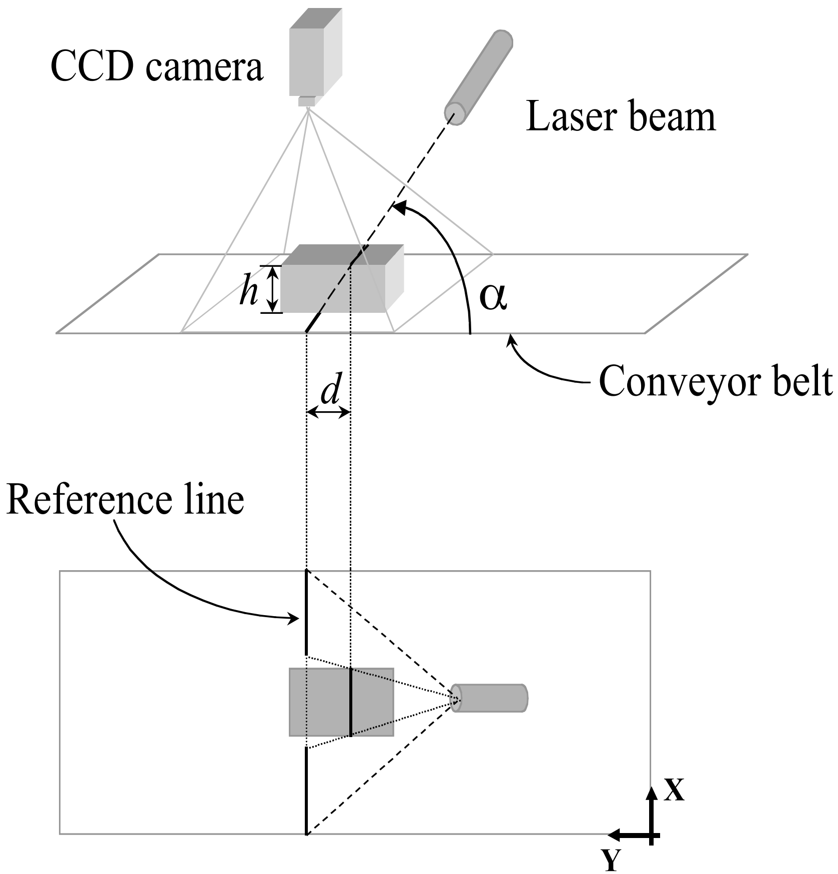
video

semi-automatic version
at Fraunhofer IPA
Stuttgart, 1997

objective: replace operator



Sensory module
in fully automatic version



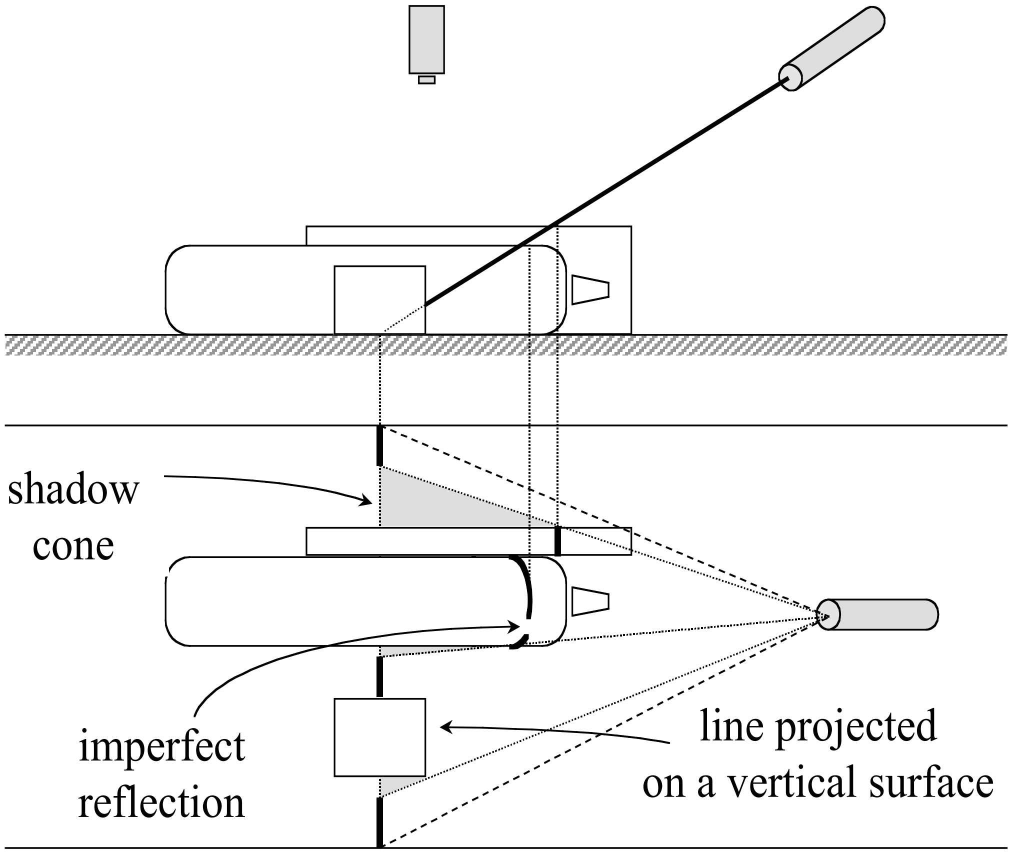
operator
+
touch-screen

replaced by

structured light vision
+
neuro-fuzzy system
for object localization
and classification

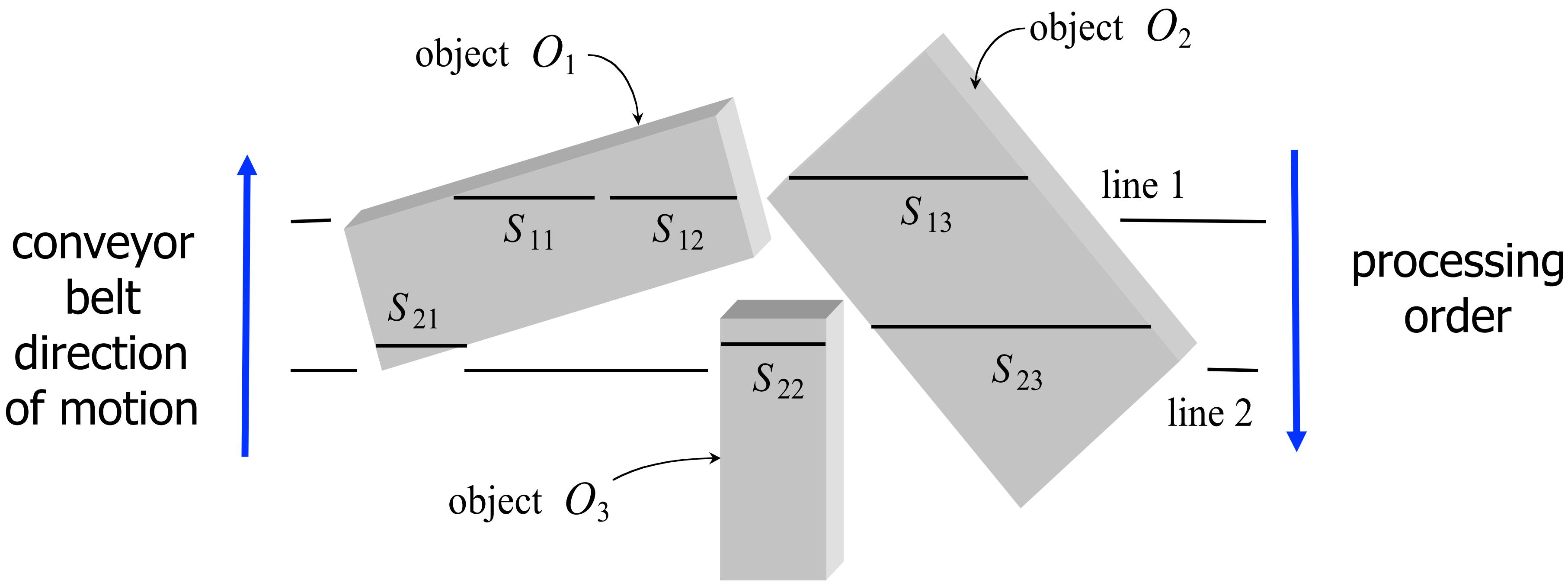
operation principle
of the structured light sensor

Sensory data interpretation



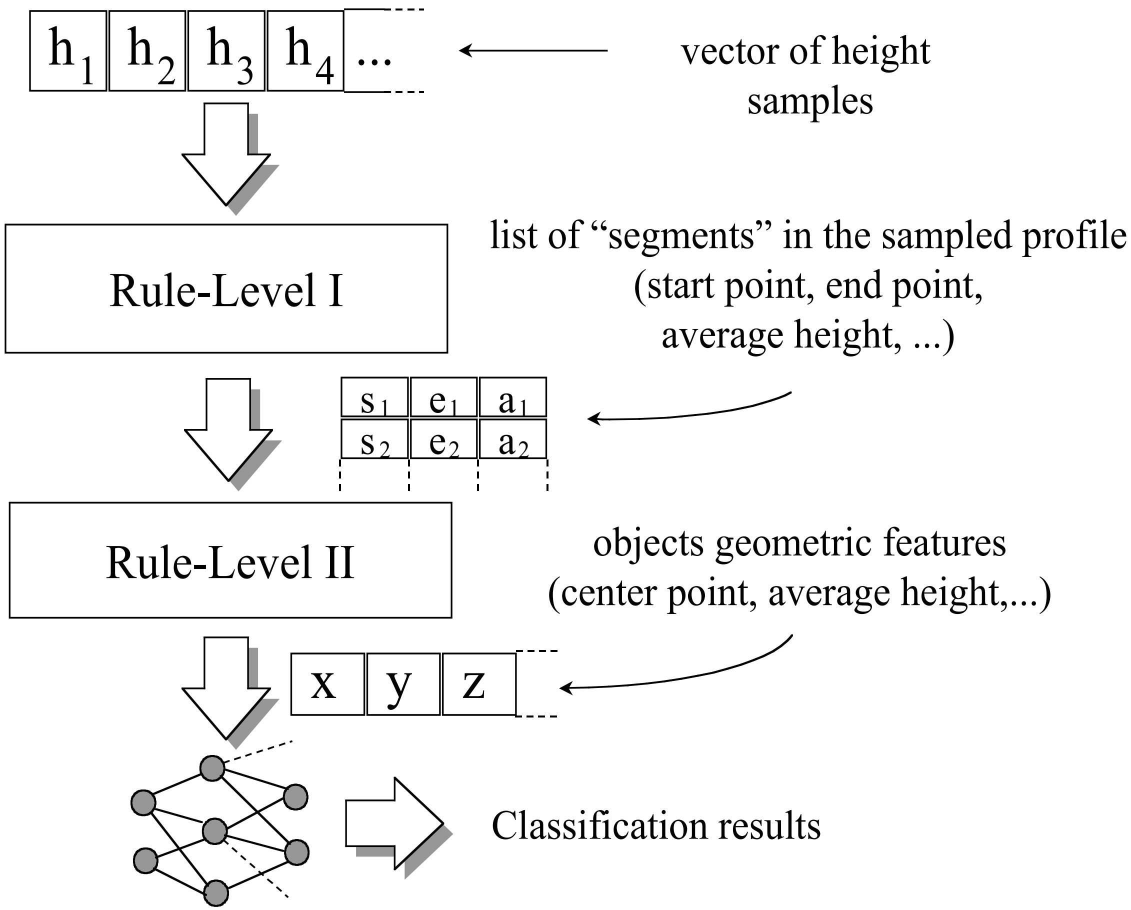
possible sources of lack of information on a single line scan

Sensory data interpretation



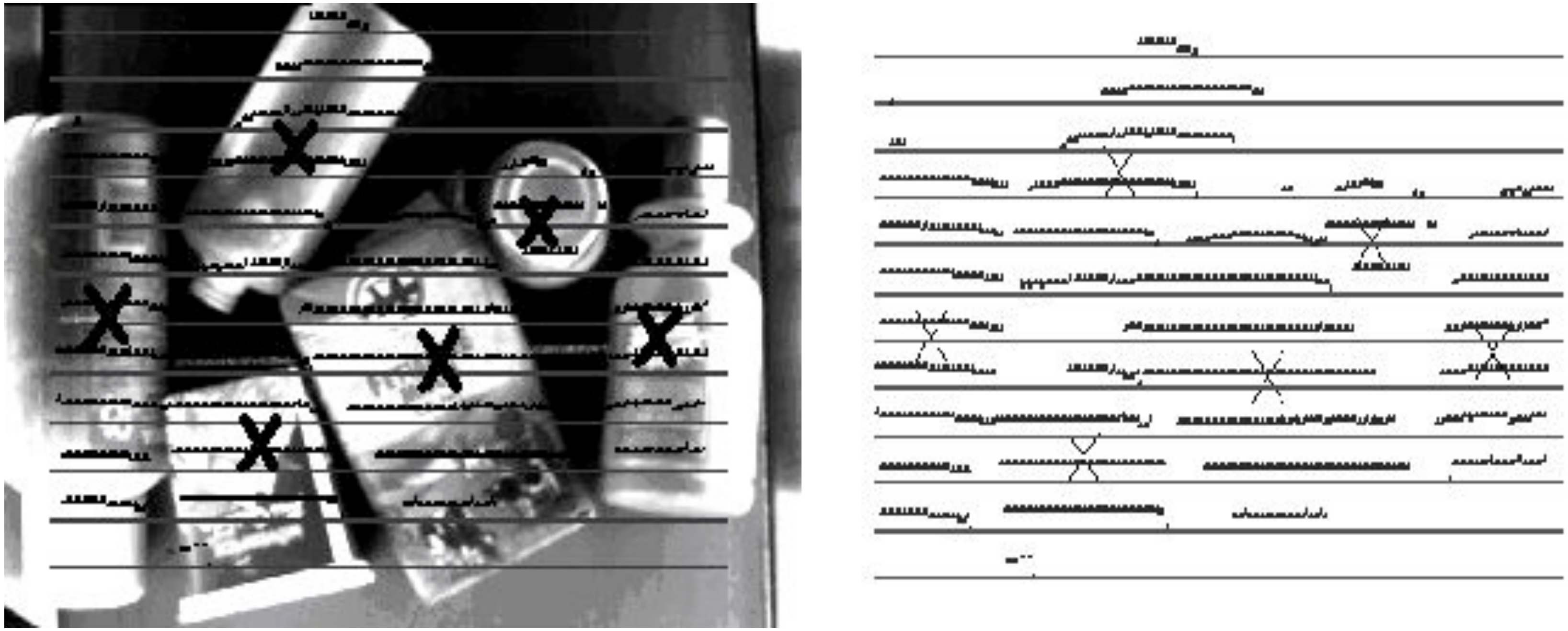
integration of data collected in successive sampling instants

Decision module



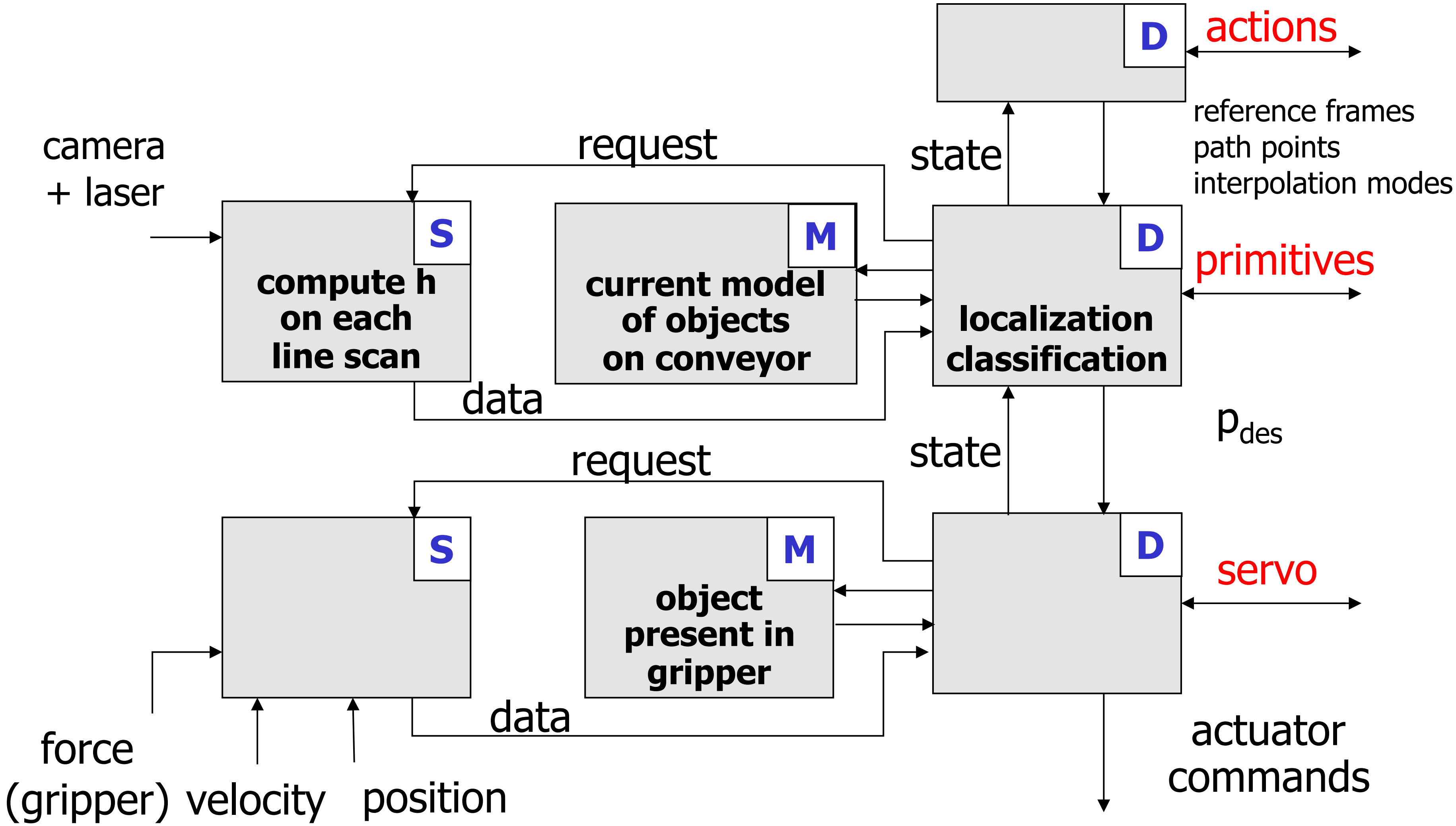
structure of the object localization and classification module

Modeling module



example of models for objects on the conveyor belt

Functional architecture of the IPA cell



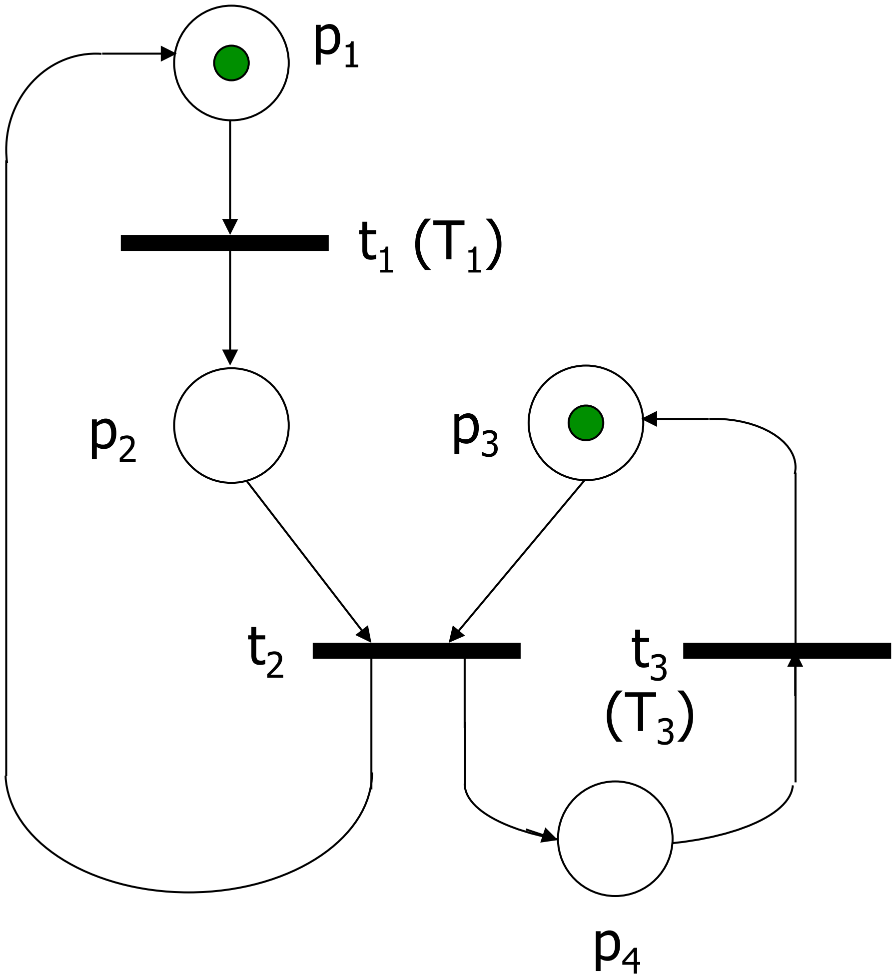
Test results

video

includes optimal scheduling of pick & place operations
to maximize throughput (minimize loss of pieces)

work by Dr. Raffaella Mattone (PhD @ DIS)

Flow diagrams of operation

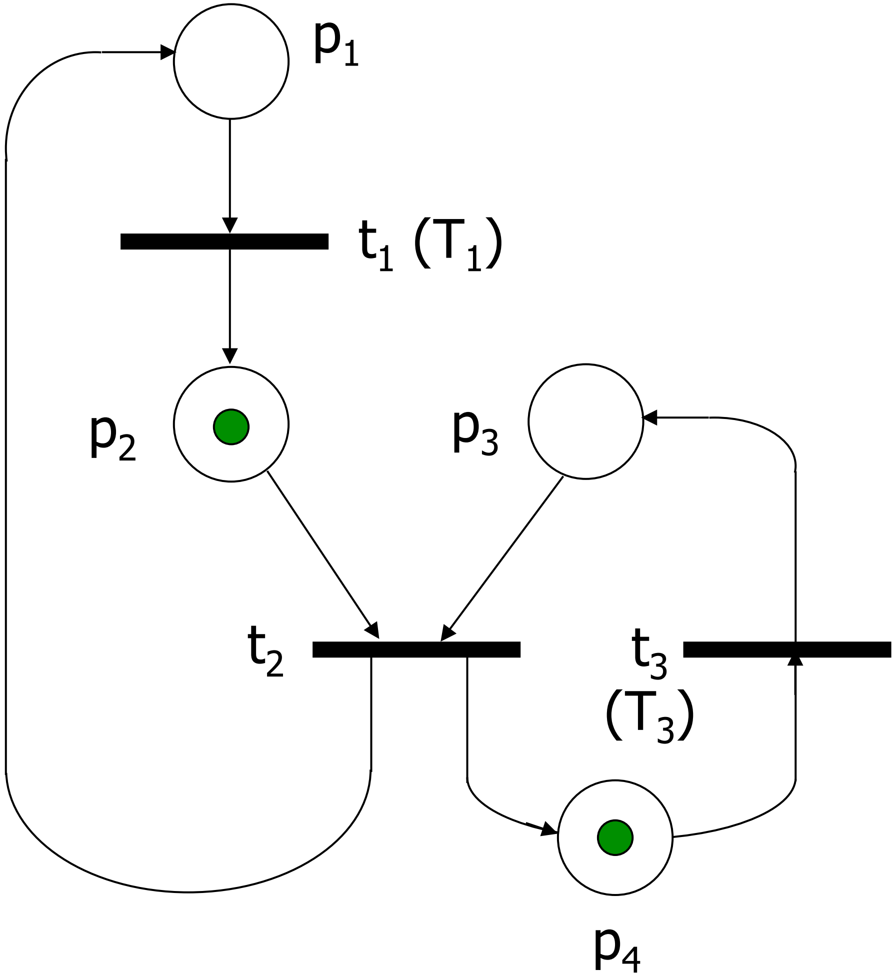


PETRI NETS

oriented graphs with two types of nodes

- places $(p_1,\ldots,p_4)$
  states or functional blocks: active if a "token" is present (e.g., p_1 and p_3)
- transitions $(t_1,\ldots,t_3)$
  changes from a state to another state, fired by events: if enough (at least one) tokens are present in all input places of a transition, tokens are moved to the output places; transitions may be timed (e.g., t_1 and t_3)

Petri net model of the IPA cell

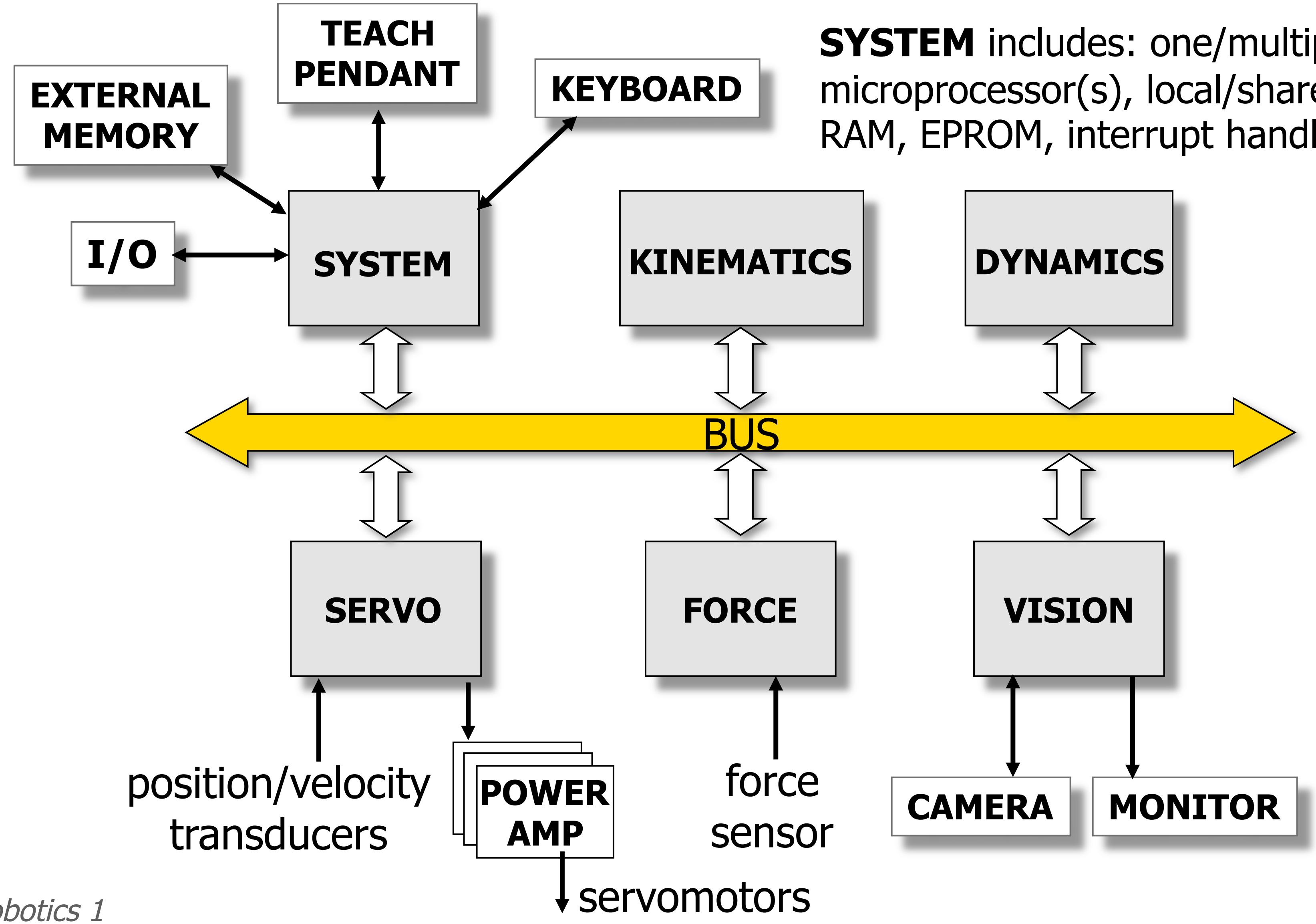


- p_1: robot picking & placing
  - T_1: pick & place time
- p_2: robot ready
- p_3: new part on conveyor
- p_4: waiting for a part
  - T_3 (random variable): time interval between two successive parts

initial marking/state:
robot ready, waiting for a part

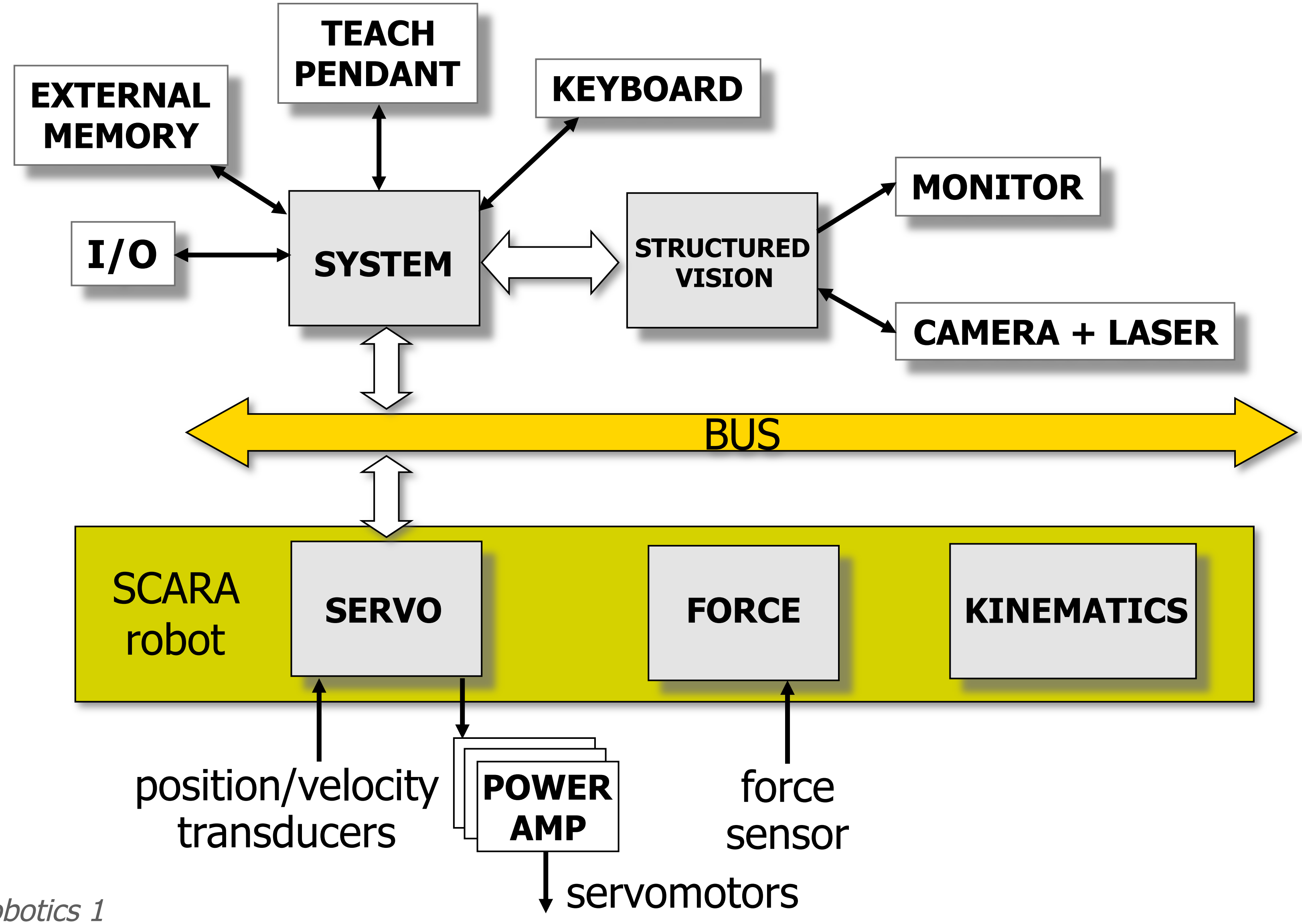
Hardware architecture

SYSTEM includes: one/multiple microprocessor(s), local/shared RAM, EPROM, interrupt handler, …



Hardware architecture

Example of the IPA cell



Hardware architecture

Example including vision in an open controller