Embedded System

Bo Chen, Zhe Yang

zheyang@zjut.edu.cn

https://mooc1.chaoxing.com/mooc-ans/course/236383867.html

Computers as Components

Principles of Embedded Computing System Design

A volume in The Morgan Kaufmann Series in Computer Architecture and Design

Book • Fifth Edition • 2022

Authors:

Marilyn Wolf

About the book

Browse this book

By table of contents

Book description

Computers as Components: Principles of Embedded Computing System Design, Fifth Edition continues to focus on foundational content in embedded systems technology and design while up … read full description

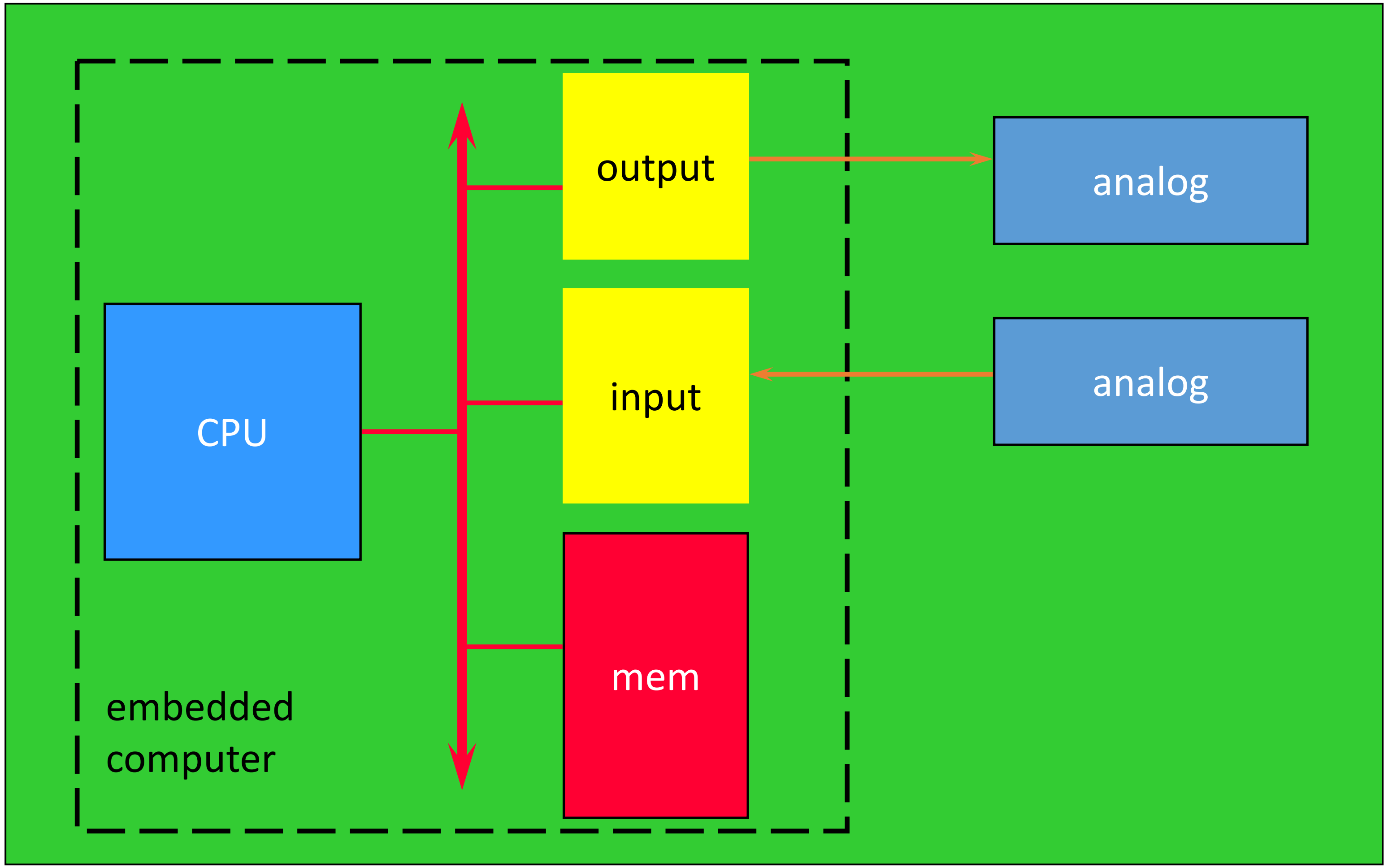
Introduction

- What are embedded computing systems?
- Challenges in embedded computing system design.
- Design methodologies.

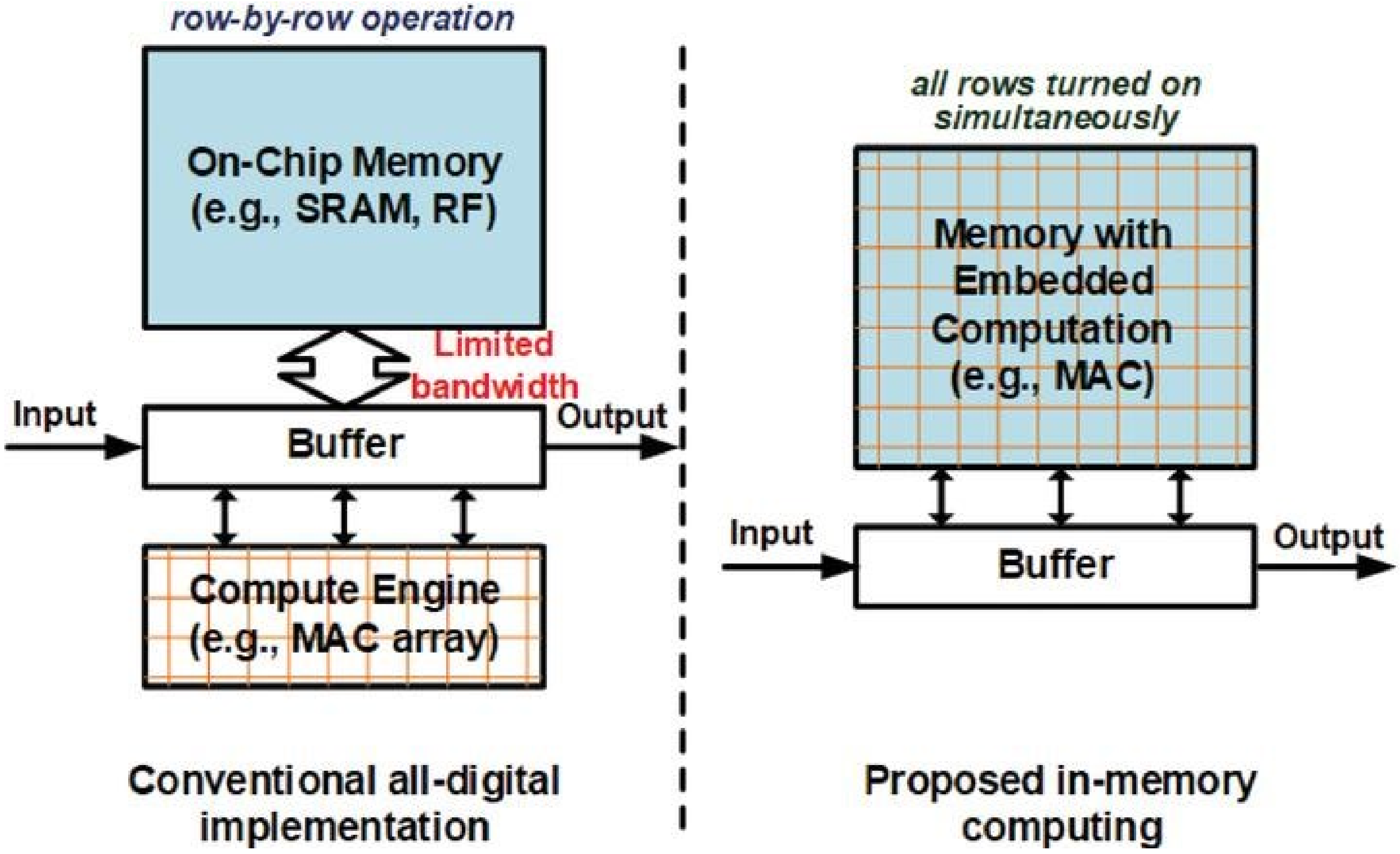
Definition

- Embedded computing system: any device that includes a programmable computer but is not itself a general-purpose computer.
- Take advantage of application characteristics to optimize the design:
  - don't need all the general-purpose bells and whistles.

Embedding a computer



Computing in Memory



Examples

- Cell phone.
- Printer.
- Automobile: engine, brakes, dash, etc.
- Airplane: engine, flight controls, nav/comm.
- Digital television.
- Household appliances.

Early history

- Late 1940's: MIT Whirlwind computer was designed for real-time operations.
  - Originally designed to control an aircraft simulator.
- First microprocessor was Intel 4004 in early 1970's.
- HP-35 calculator used several chips to implement a microprocessor in 1972.

Early history, cont'd.

- Automobiles used microprocessor-based engine controllers starting in 1980.
  - Control fuel/air mixture, engine timing, etc.
  - Multiple modes of operation: warm-up, cruise, hill climbing, etc.
  - Provides lower emissions, better fuel efficiency.

Microprocessor varieties

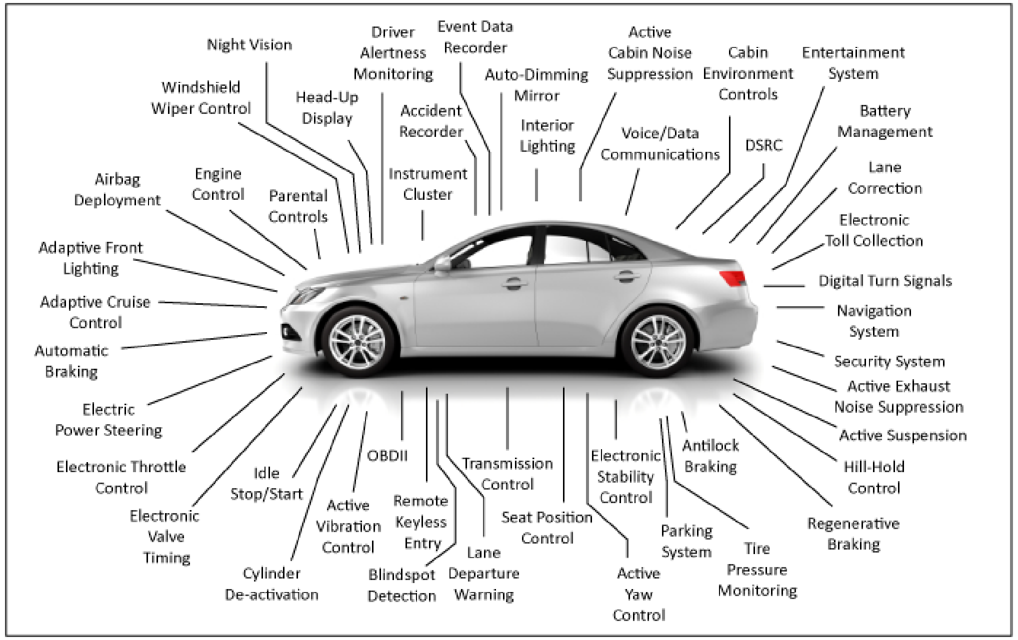
- Microcontroller: includes I/O devices, on-board memory.
- Digital signal processor (DSP): microprocessor optimized for digital signal processing.
- Typical embedded word sizes: 8-bit, 16-bit, 32-bit.

Application examples

- Simple control: front panel of microwave oven, etc.
- Canon EOS 3/R3 has three microprocessors.
  - 32-bit RISC CPU runs autofocus and eye control systems.
- Digital TV: programmable CPUs + hardwired logic.

Automotive embedded systems

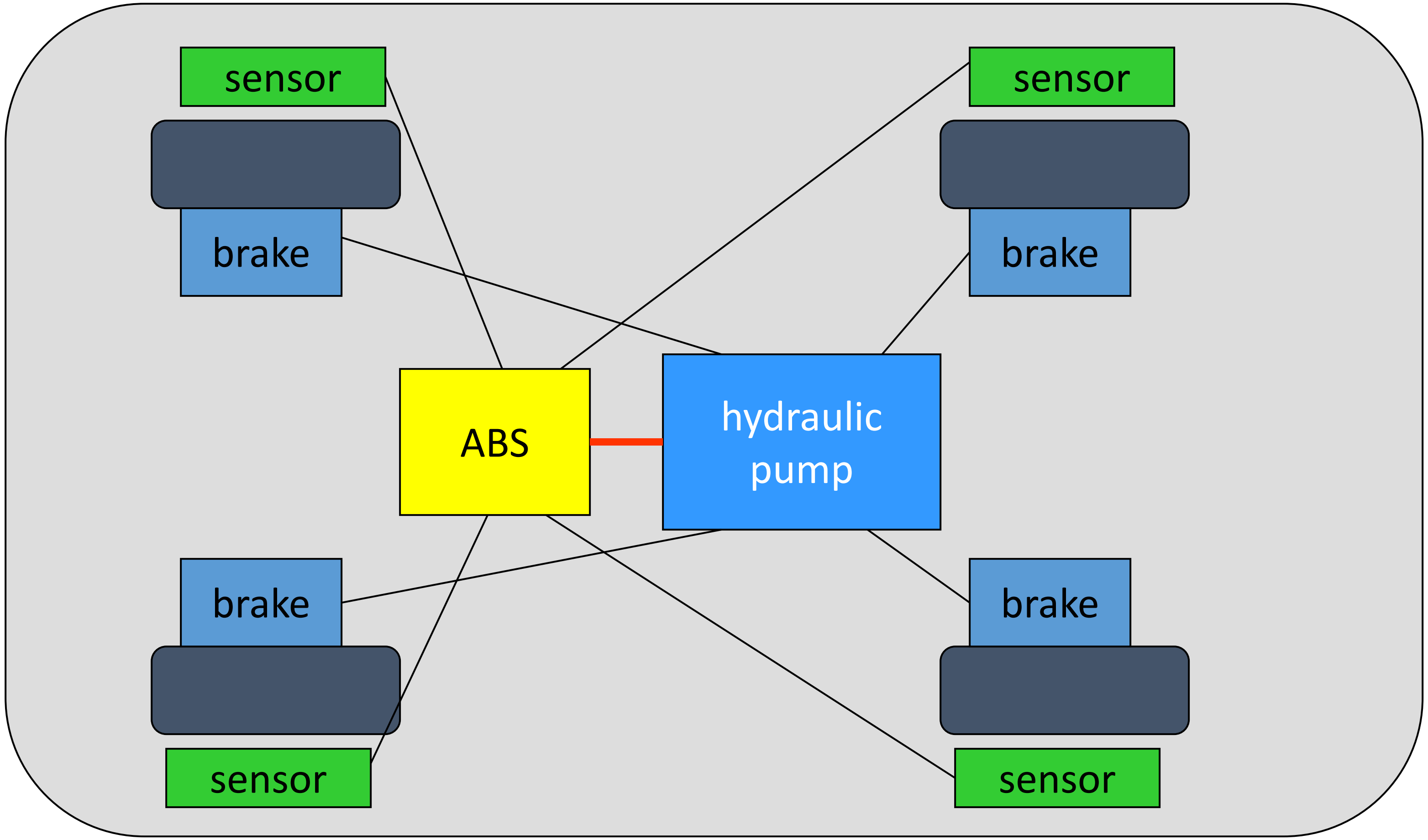
- Today's high-end automobile may have 100 microprocessors:
  - 4-bit microcontroller checks seat belt;
  - microcontrollers run dashboard devices;
  - 16/32-bit microprocessor controls engine.
- Low-end cars use 20+ microprocessors.



BMW 850i brake and stability control system

- Anti-lock brake system (ABS): pumps brakes to reduce skidding.
- Automatic stability control (ASC+T): controls engine to improve stability.
- ABS and ASC+T communicate.
  - ABS was introduced first---needed to interface to existing ABS module.

BMW 850i, cont'd.



Characteristics of embedded systems

- Sophisticated functionality.
- Real-time operation.
- Low manufacturing cost.
- Low power.
- Designed to tight deadlines by small teams.

Functional complexity

- Often have to run sophisticated algorithms or multiple algorithms.
  - Cell phone, laser printer.
- Often provide sophisticated user interfaces.

Real-time operation

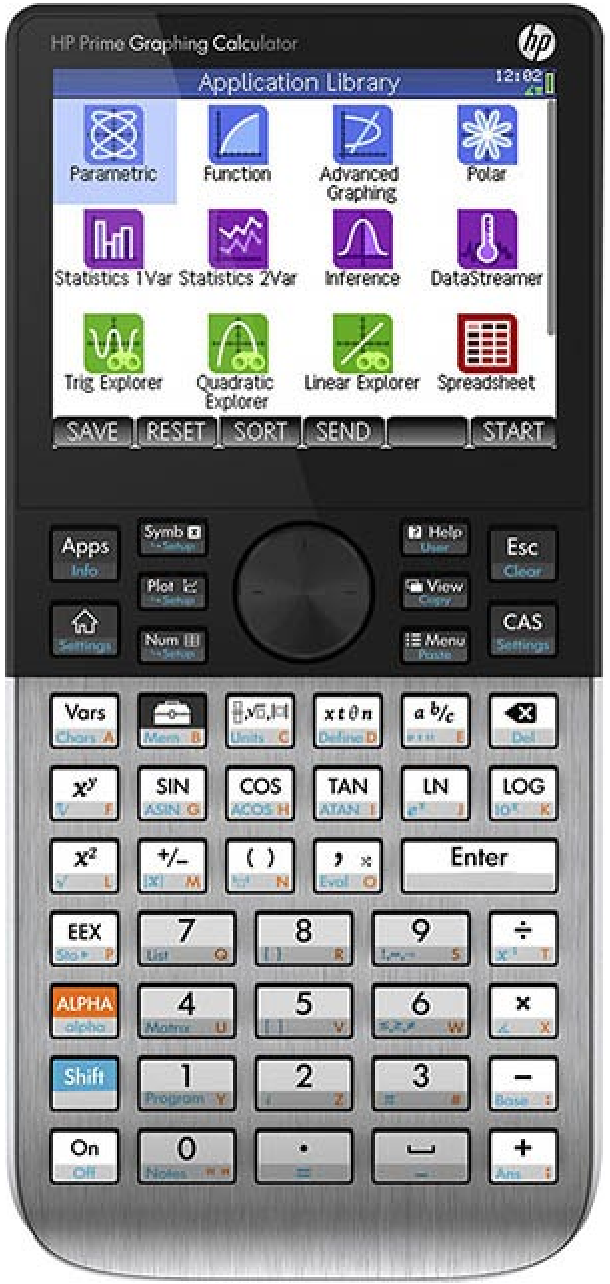
- Must finish operations by deadlines.
  - Hard real time: missing deadline causes failure.
  - Soft real time: missing deadline results in degraded performance.
- Many systems are multi-rate: must handle operations at widely varying rates.

Non-functional requirements

- Many embedded systems are mass-market items that must have low manufacturing costs.
  - Limited memory, microprocessor power, etc.
- Power consumption is critical in battery-powered devices.
  - Excessive power consumption increases system cost even in wall-powered devices.

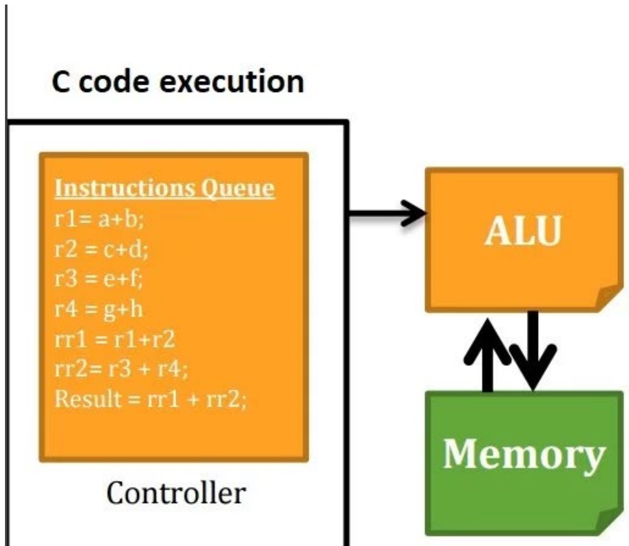
Design teams

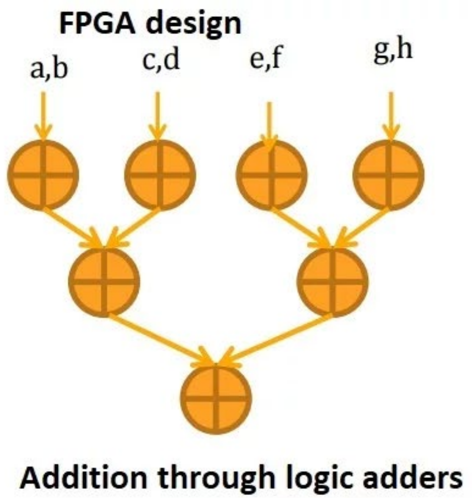
- Often designed by a small team of designers.
- Often must meet tight deadlines.
    - 6 month market window is common.
    - Can’t miss back-to-school window for calculator.



Why use microprocessors?

- Alternatives: field-programmable gate arrays (FPGAs), custom logic, etc.
- Microprocessors are often very efficient: can use same logic to perform many different functions.
- Microprocessors simplify the design of families of products.



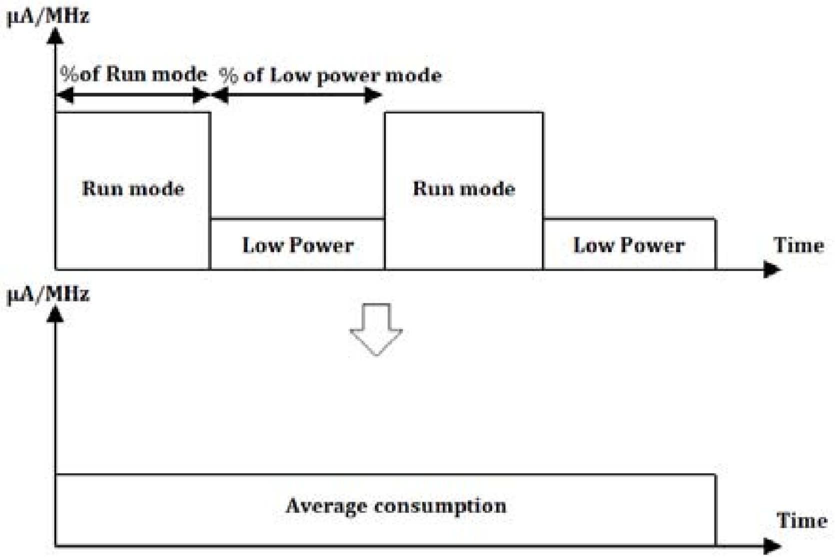


The performance paradox

- Microprocessors use much more logic to implement a function than does custom logic.
- But microprocessors are often at least as fast:
  - heavily pipelined;
  - large design teams;
  - aggressive VLSI technology.

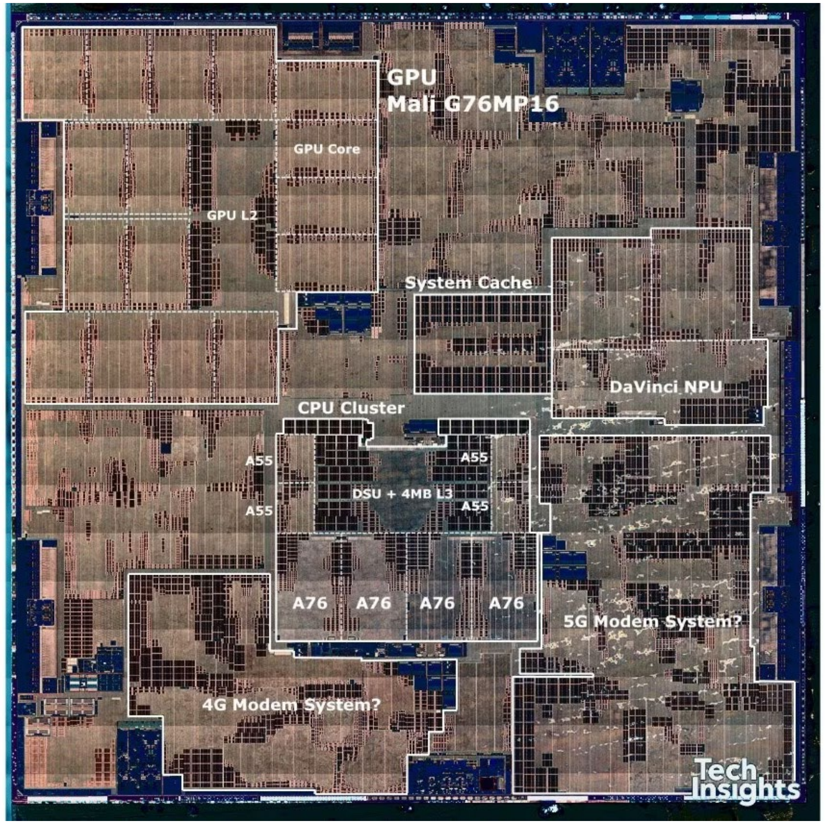
Power

- Custom logic uses less power, but CPUs have advantages:
  - Modern microprocessors offer features to help control power consumption.
  - Software design techniques can help reduce power consumption.
- Heterogeneous systems: some custom logic for well-defined functions, CPUs+software for everything else.



Platforms

- Embedded computing platform: hardware architecture + associated software.
- Many platforms are multiprocessors.
- Examples:
  - Single-chip multiprocessors for cell phone baseband.
  - Automotive network + processors.

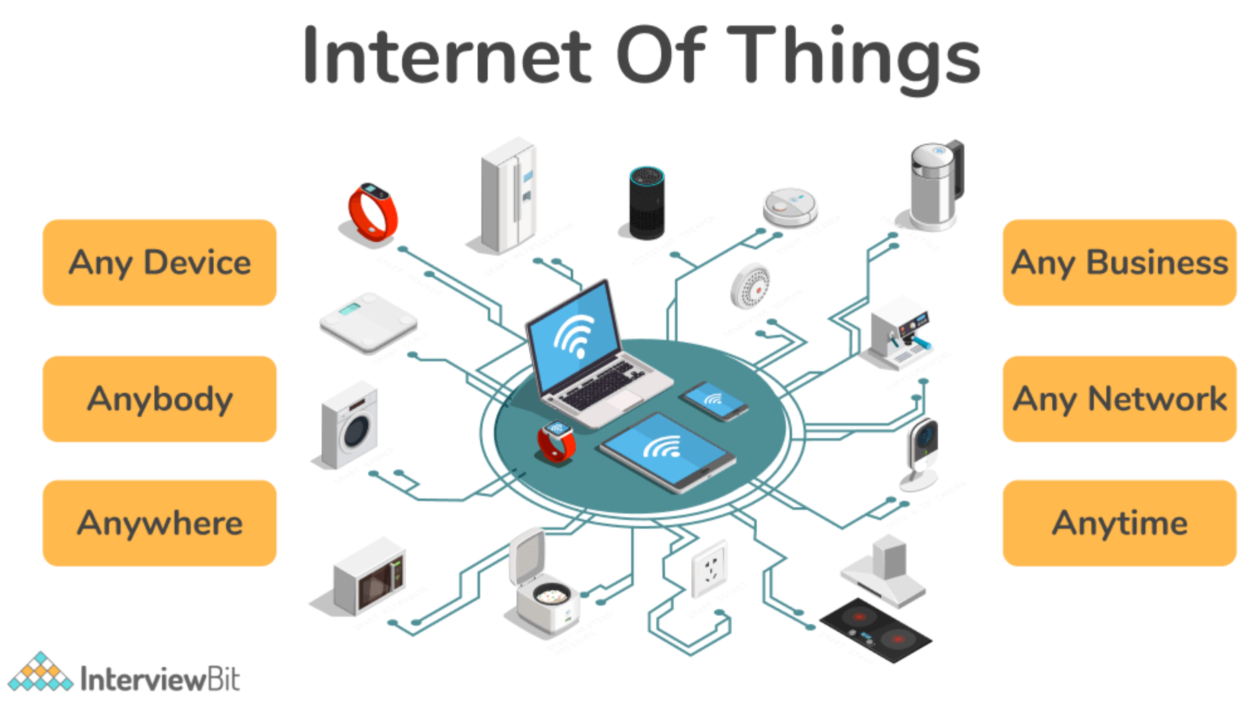


The physics of software

- Computing is a physical act.
  - Software doesn't do anything without hardware.
- Executing software consumes energy, requires time.
- To understand the dynamics of software (time, energy), we need to characterize the platform on which the software runs.

Internet-of-Things (IoT) system

- Combines sensing, actuating, computing, communication.
  - Some links in the network are often wireless.
- Example: manufacturing plant.



Cyber-physical systems

- A physical system that tightly interacts with a computer system.
- Computers replace mechanical controllers:
  - More accurate.
  - More sophisticated control.
- Engine controllers replace distributor, carburetor, etc.
  - Complex algorithms allow both greater fuel efficiency and lower emissions.

IoT and CPS

- IoT often lower sample rate, larger physical plant.
- CPS often more tightly coupled, higher sample rate.

Edge computing

- Systems that must respond to the physical world often can’t wait for answers from a remote data center.
- Edge computing:
  - Responsive.
  - Energy efficient.
  - Connected to other devices.

What does “performance” mean?

- In general-purpose computing, performance often means average-case, may not be well-defined.
- In real-time systems, performance means meeting deadlines.
  - Missing the deadline by even a little is bad.
  - Finishing ahead of the deadline may not help.

Characterizing performance

- We need to analyze the system at several levels of abstraction to understand performance:
  - CPU.
  - Platform.
  - Program.
  - Task.
  - Multiprocessor.

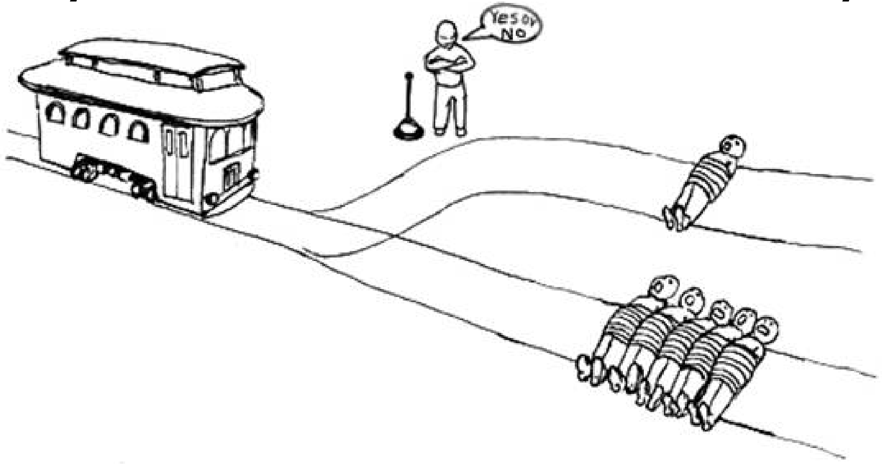
Security, safety

- Security: system's ability to prevent malicious attacks.
- Integrity: maintenance of proper data values.
- Privacy: no unauthorized releases of data.

- Safety: no harmful releases of energy.
  - No crashes, accidents, etc.

Safe, secure systems

- Traditional security is oriented to IT and data security.
- But insecure embedded computers can create unsafe cyber-physical systems.
- We need to combine safety and security:
  - Identify security breaches that compromise safety.
- Safety and security can't be bolted on---they must be baked in.



Safe and secure systems technologies

- Cryptography enables encryption and its follow-ons such as digital signatures.
- Security protocols use cryptography to authenticate, check integrity, etc.
- Safe and secure hardware architectures limit ability of adversaries to interfere with cryptographic operations.

Challenges in embedded system design

- How much hardware do we need?
  - How big is the CPU? Memory?
- How do we meet our deadlines?
  - Faster hardware or cleverer software?
- How do we minimize power?
  - Turn off unnecessary logic? Reduce memory accesses?

Challenges, etc.

- Does it really work?
  - Is the specification correct?
  - Does the implementation meet the spec?
  - How do we test for real-time characteristics?
  - How do we test on real data?
- How do we work on the system?
  - Observability, controllability?
  - What is our development platform?

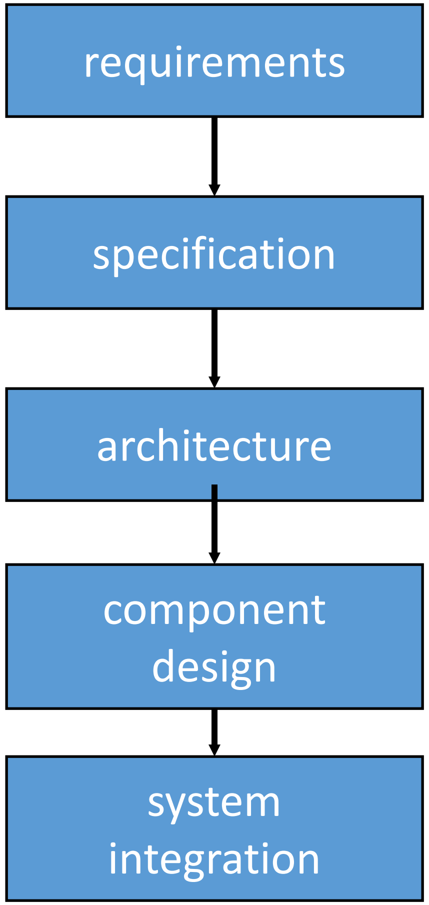
Design methodologies

- A procedure for designing a system.
- Understanding your methodology helps you ensure you didn’t skip anything.
- Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:
  - help automate methodology steps;
  - keep track of the methodology itself.

Design goals

- Performance.
    - Overall speed, deadlines.
- Functionality and user interface.
- Manufacturing cost.
- Power consumption.
- Other requirements (physical size, etc.)

Levels of abstraction



Top-down vs. bottom-up

- Top-down design:
  - start from most abstract description;
  - work to most detailed.
- Bottom-up design:
  - work from small components to big system.
- Real design uses both techniques.

Stepwise refinement

- At each level of abstraction, we must:
  - analyze the design to determine characteristics of the current state of the design;
  - refine the design to add detail.

Requirements

- Plain language description of what the user wants and expects to get.
- May be developed in several ways:
  - talking directly to customers;
  - talking to marketing representatives;
  - providing prototypes to users for comment.

Functional vs. non-functional requirements

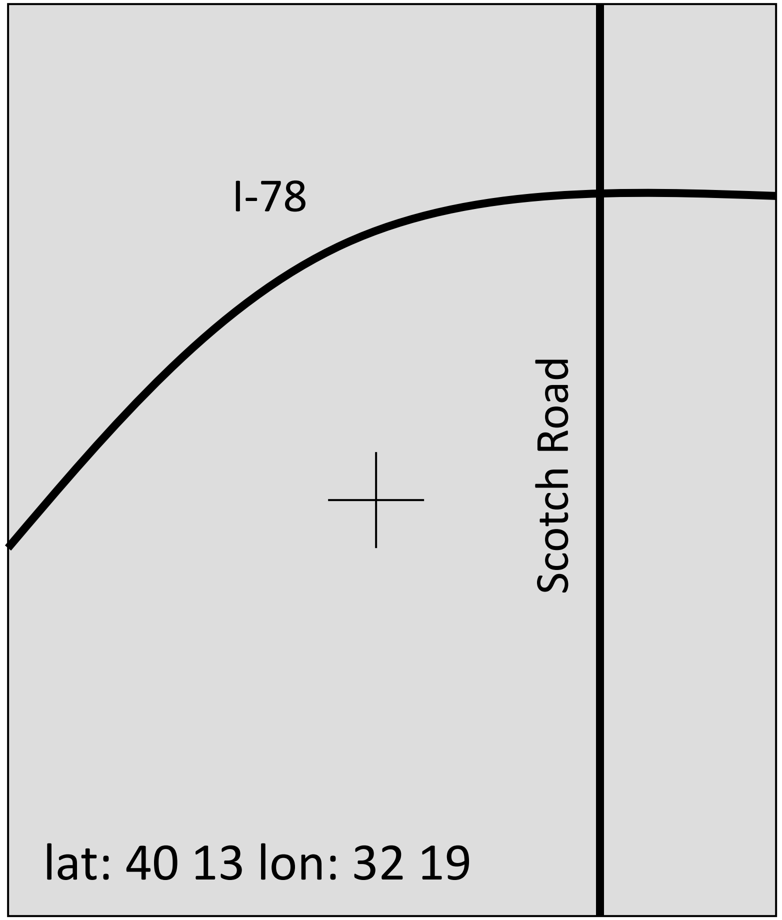
- Functional requirements:
  - output as a function of input.
- Non-functional requirements:
  - time required to compute output;
  - size, weight, etc.;
  - power consumption;
  - reliability;
  - etc.

Our requirements form

name
purpose
inputs
outputs
functions
performance
manufacturing cost
power
physical size/weight

Example: GPS moving map requirements

- Moving map obtains position from GPS, paints map from local database.



GPS moving map needs

- Functionality: For automotive use. Show major roads and landmarks.
- User interface: At least 400 x 600 pixel screen. Three buttons max. Pop-up menu.
- Performance: Map should scroll smoothly. No more than 1 sec power-up. Lock onto GPS within 15 seconds.
- Cost: $120 street price = approx. $30 cost of goods sold.

GPS moving map needs, cont'd.

- Physical size/weight: Should fit in hand.
- Power consumption: Should run for 8 hours on four AA batteries.

GPS moving map requirements form

name	GPS moving map
purpose	consumer-grade moving map for driving
inputs	power button, two control buttons
outputs	back-lit LCD 400 X 600
functions	5-receiver GPS; three resolutions; displays current lat/lon
performance	updates screen within 0.25 sec of movement
manufacturing cost	$100 cost-of-goods-sold
power	100 mW
physical size/weight	no more than 2: X 6:, 12 oz.

Specification

- A more precise description of the system:
  - should not imply a particular architecture;
  - provides input to the architecture design process.
- May include functional and non-functional elements.
- May be executable or may be in mathematical form for proofs.

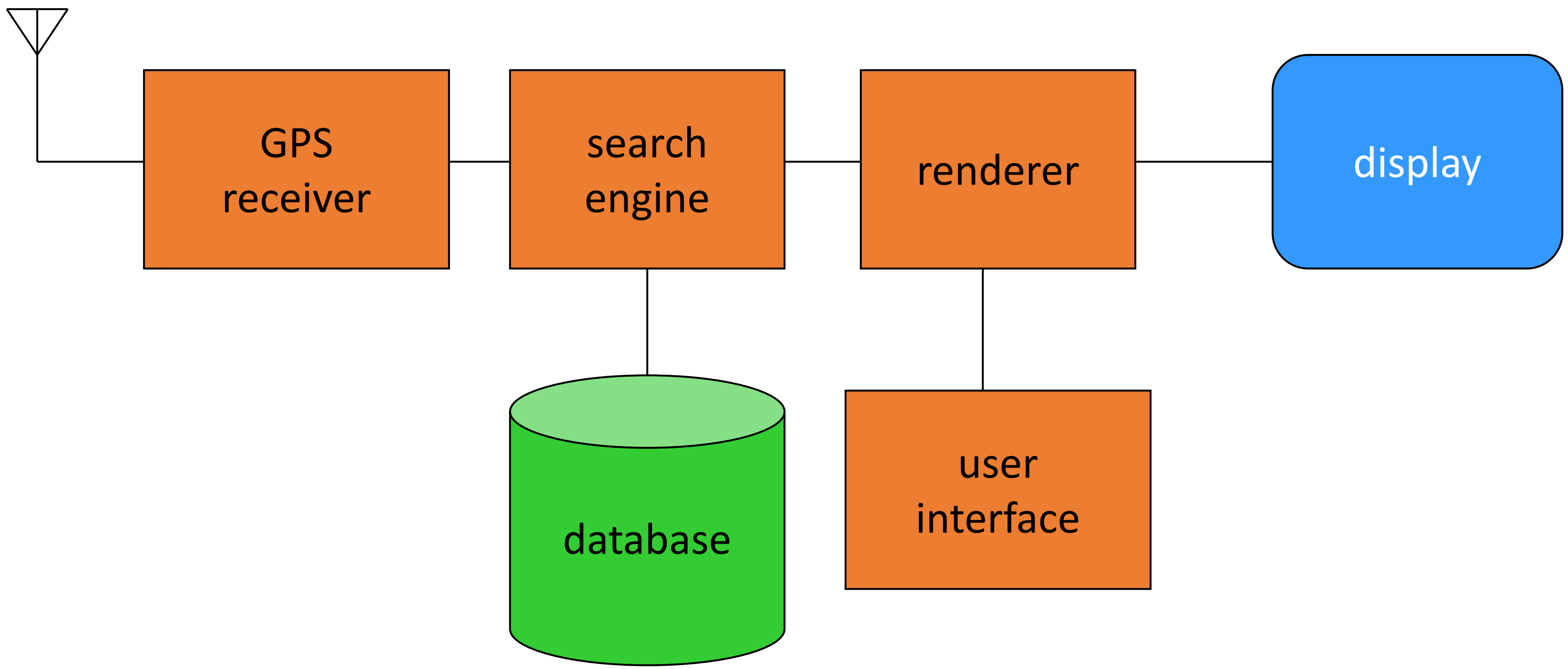
GPS specification

- Should include:
  - What is received from GPS;
  - map data;
  - user interface;
  - operations required to satisfy user requests;
  - background operations needed to keep the system running.

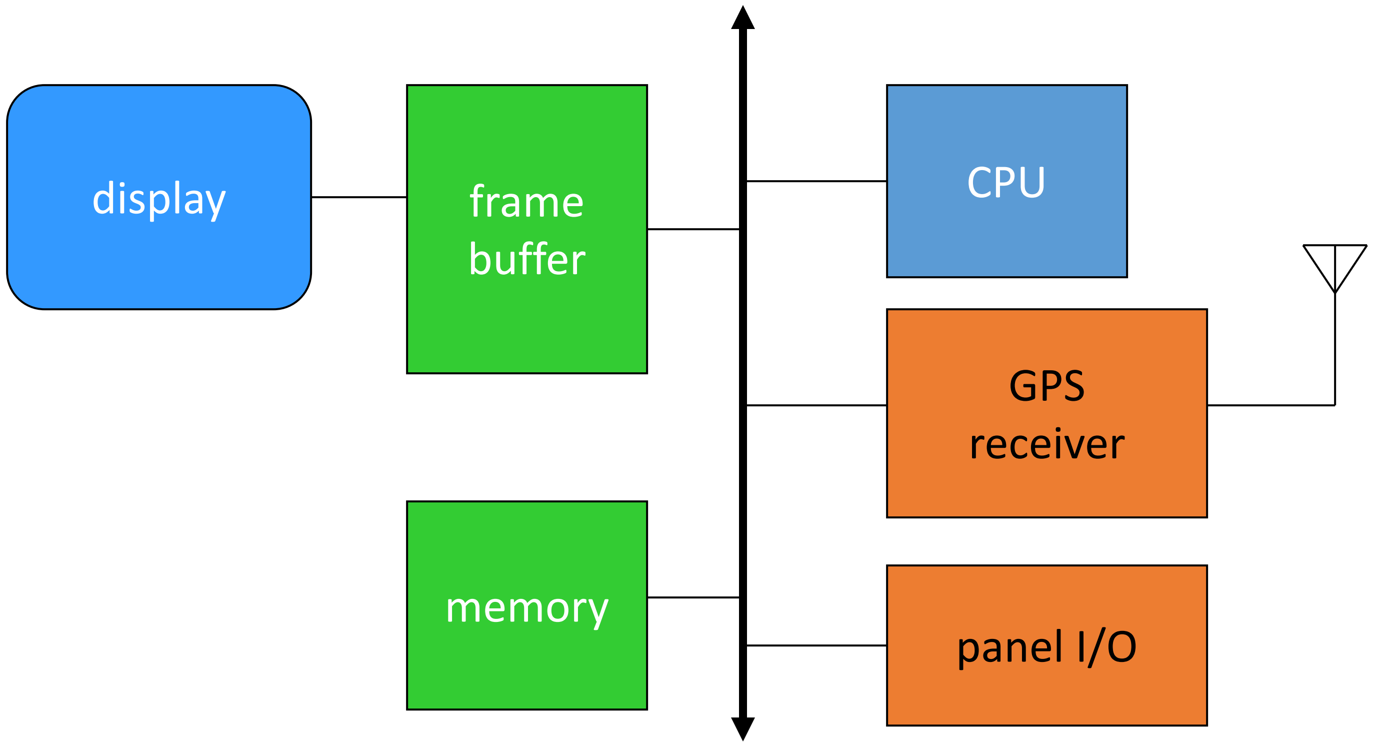
Architecture design

- What major components go satisfying the specification?
- Hardware components:
  - CPUs, peripherals, etc.
- Software components:
  - major programs and their operations.
- Must take into account functional and non-functional specifications.

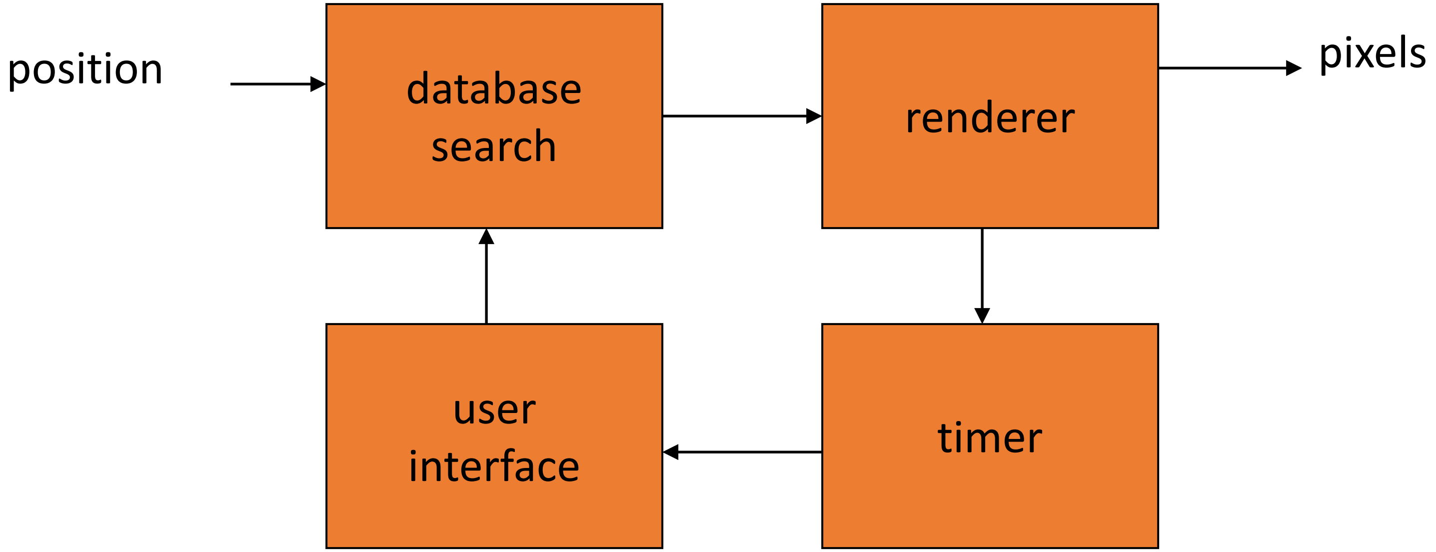
GPS moving map block diagram



GPS moving map hardware architecture



GPS moving map software architecture



Designing hardware and software components

- Must spend time architecting the system before you start coding.
- Some components are ready-made, some can be modified from existing designs, others must be designed from scratch.

System integration

- Put together the components.
  - Many bugs appear only at this stage.
- Have a plan for integrating components to uncover bugs quickly, test as much functionality as early as possible.

Summary

- Embedded computers are all around us.
  - Many systems have complex embedded hardware and software.
- Embedded systems pose many design challenges: design time, deadlines, power, etc.
- Design methodologies help us manage the design process.

Introduction

- Object-oriented design.
- Unified Modeling Language (UML).

System modeling

- Need languages to describe systems:
  - useful across several levels of abstraction;
  - understandable within and between organizations.
- Block diagrams are a start, but don’t cover everything.

Object-oriented design

- Object-oriented (OO) design: A generalization of object-oriented programming.
- Object = state + methods.
  - State provides each object with its own identity.
  - Methods provide an abstract interface to the object.

Objects and classes

- Class: object type.
- Class defines the object's state elements but state values may change over time.
- Class defines the methods used to interact with all objects of that type.
  - Each object has its own state.

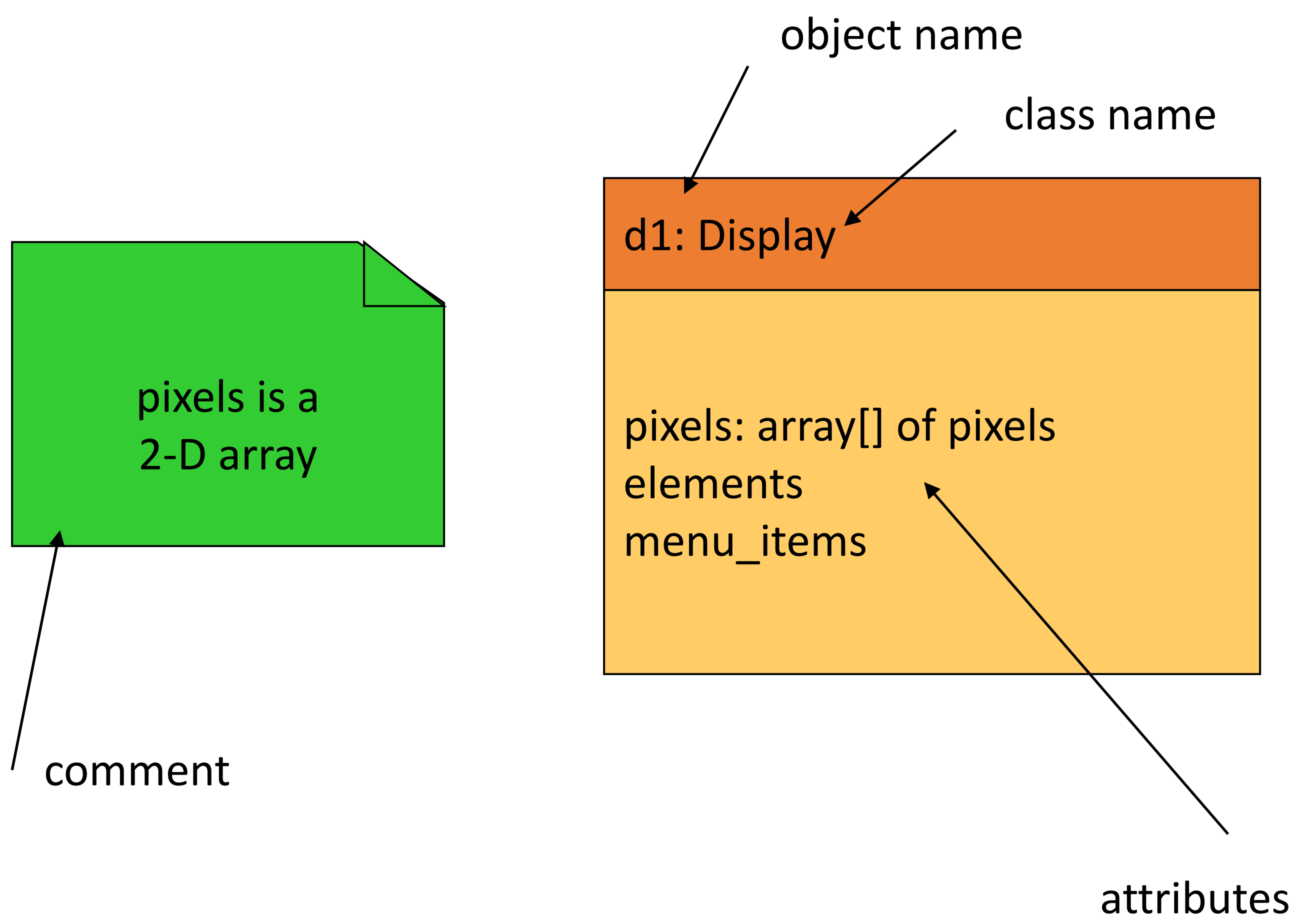
OO design principles

- Some objects will closely correspond to real-world objects.
  - Some objects may be useful only for description or implementation.
- Objects provide interfaces to read/write state, hiding the object’s implementation from the rest of the system.

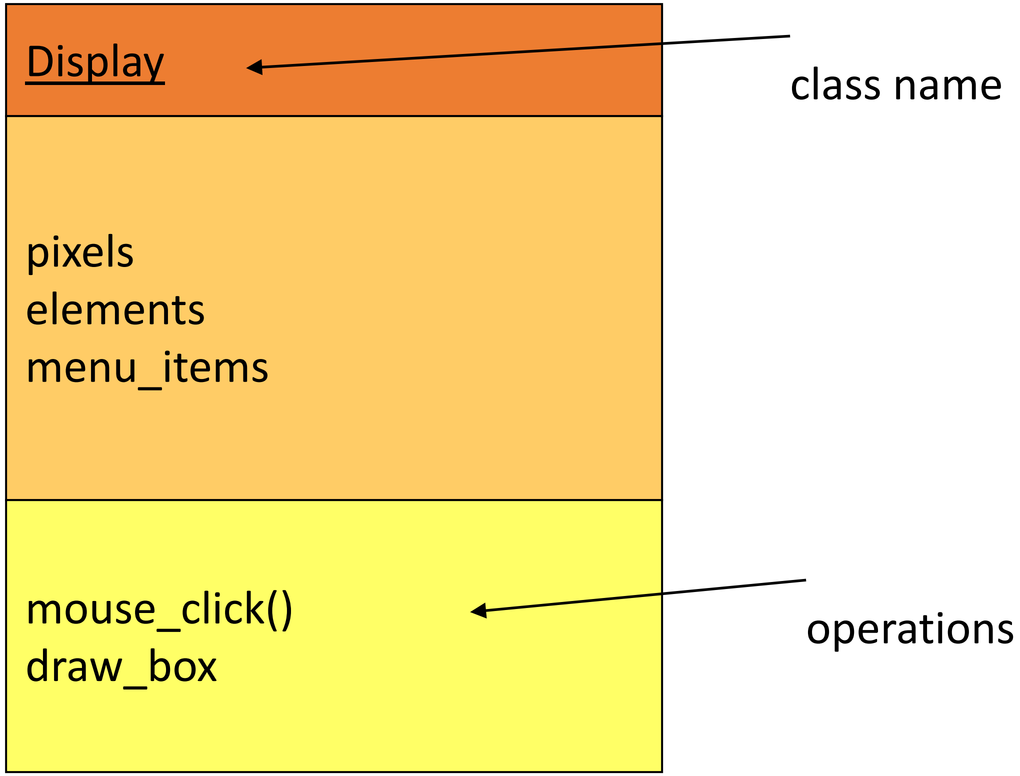
UML

- Developed by Booch et al.
- Goals:
  - object-oriented;
  - visual;
  - useful at many levels of abstraction;
  - usable for all aspects of design.

UML object


object name
class name
pixels is a
2-D array
d1: Display
pixels: array[] of pixels
elements
menu_items
comment
attributes

UML class


Display
pixels
elements
menu_items
mouse_click()
draw_box
class name
operations

The class interface

- The operations provide the abstract interface between the class's implementation and other classes.
- Operations may have arguments, return values.
- An operation can examine and/or modify the object's state.

Choose your interface properly

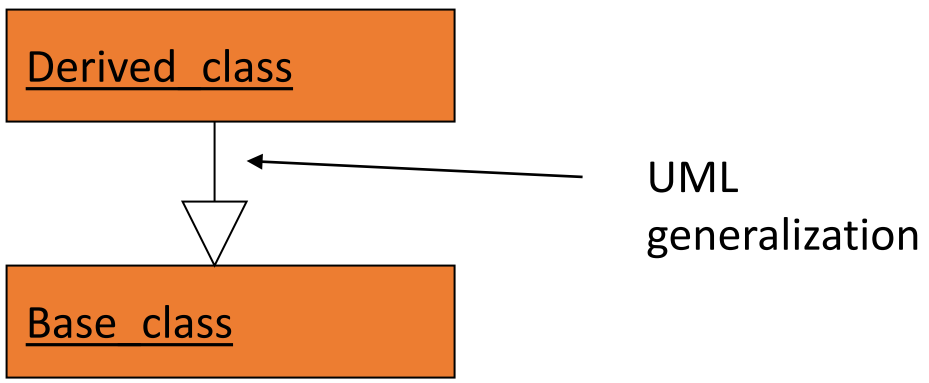
- If the interface is too small/specialized:
  - object is hard to use for even one application;
  - even harder to reuse.
- If the interface is too large:
  - class becomes too cumbersome for designers to understand;
  - implementation may be too slow;
  - spec and implementation are probably buggy.

Relationships between objects and classes

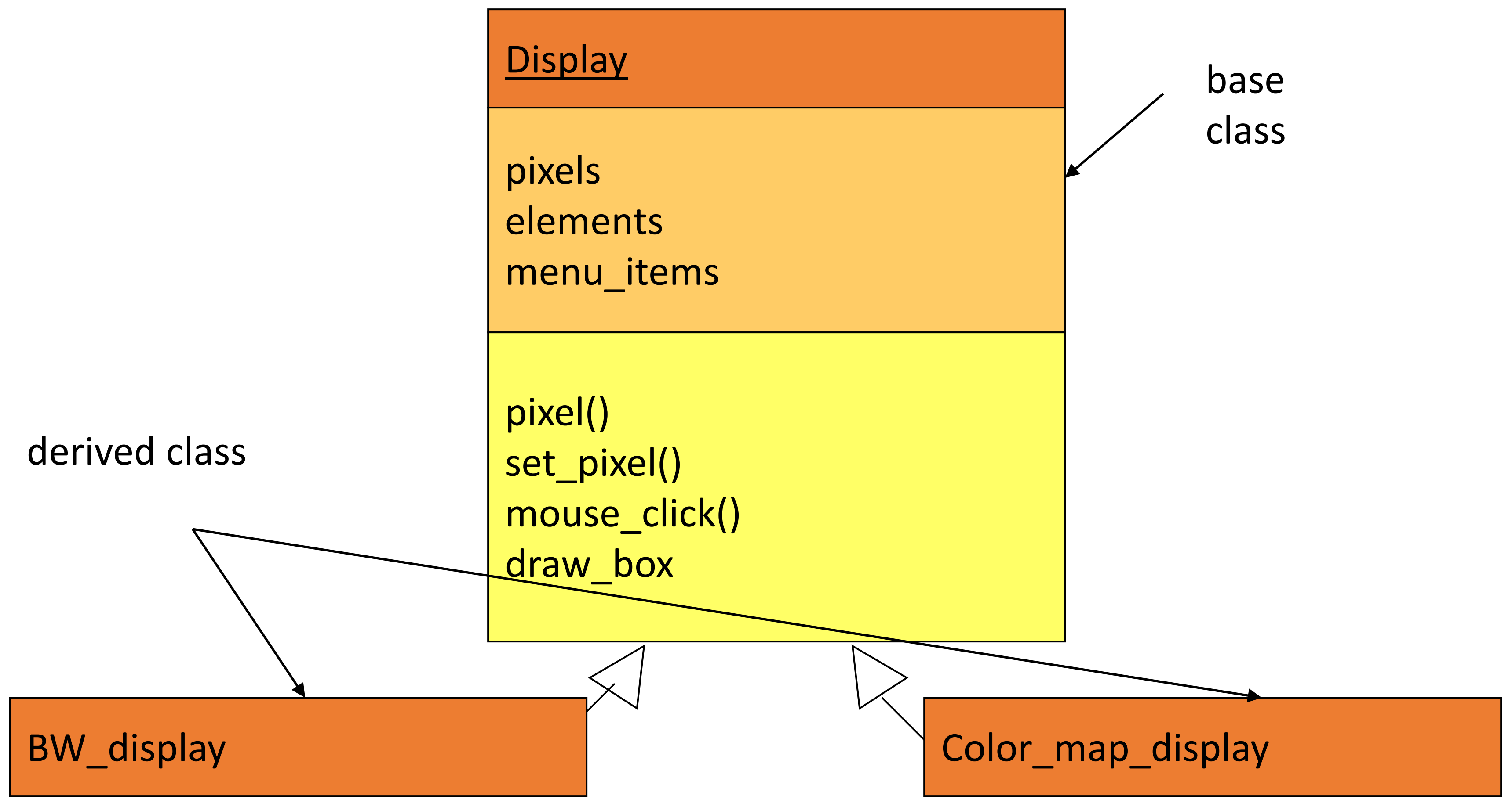
- Association: objects communicate but one does not own the other.
- Aggregation: a complex object is made of several smaller objects.
- Composition: aggregation in which owner does not allow access to its components.
- Generalization: define one class in terms of another.

Class derivation

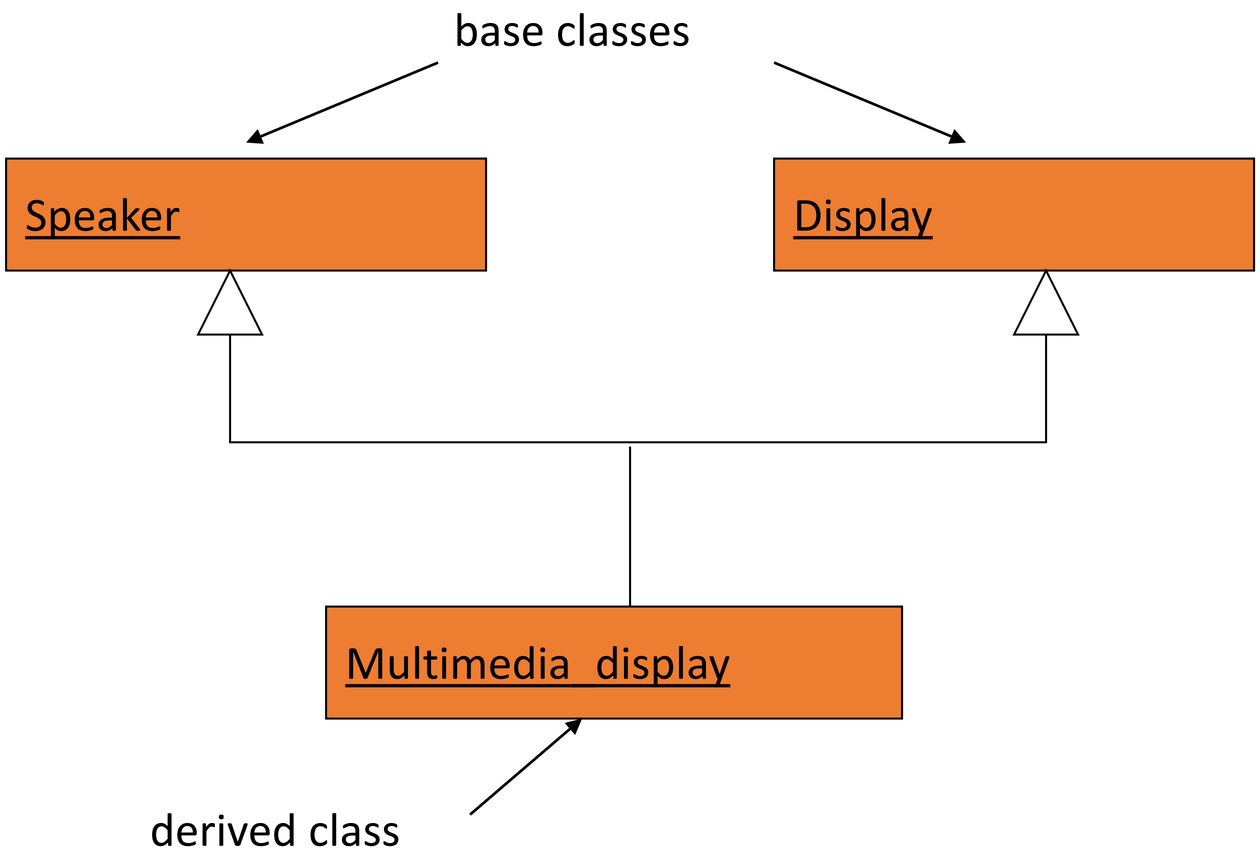
- May want to define one class in terms of another.
  - Derived class inherits attributes, operations of base class.



Class derivation example



Multiple inheritance

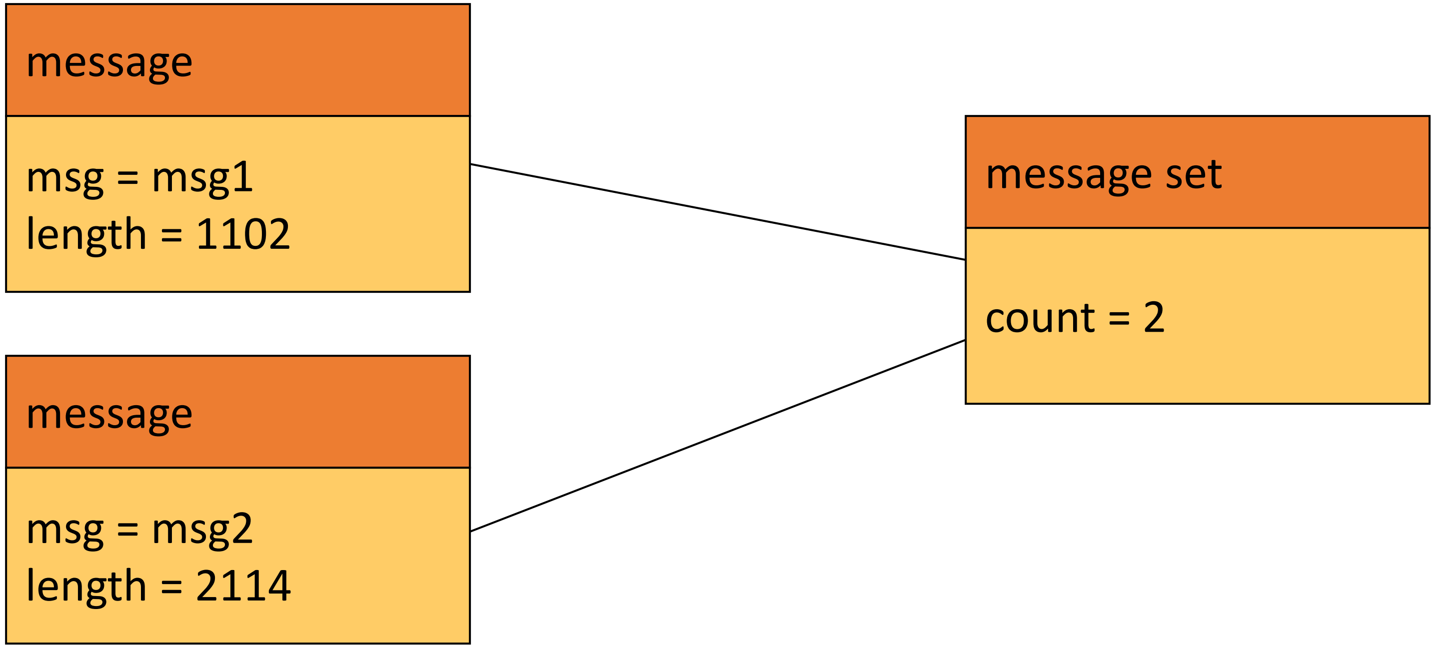


Links and associations

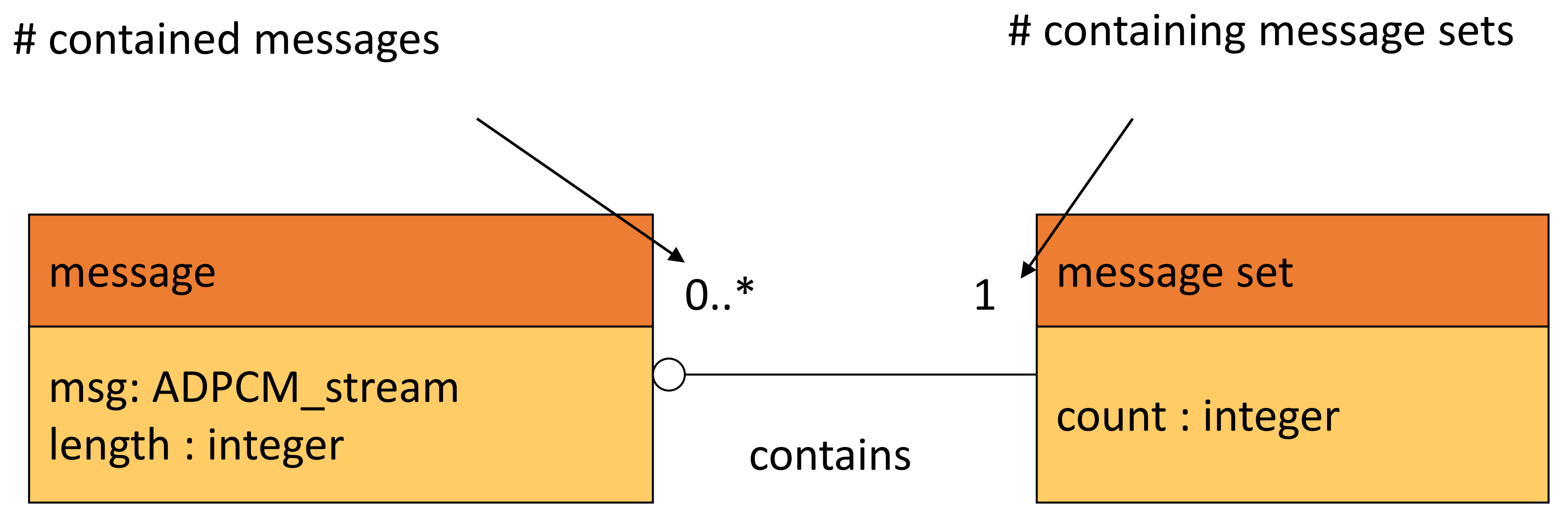
- Link: describes relationships between objects.
- Association: describes relationship between classes.

Link example

- Link defines the contains relationship:



Association example



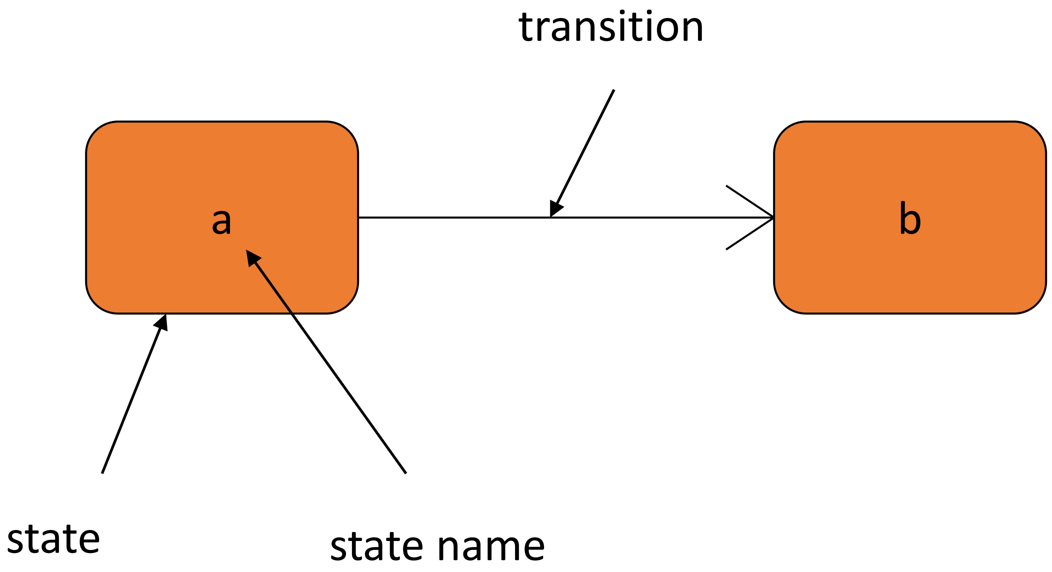
Stereotypes

- Stereotype: recurring combination of elements in an object or class.
- Example:
  - <<foo>>

Behavioral description

- Several ways to describe behavior:
  - internal view;
  - external view.

State machines



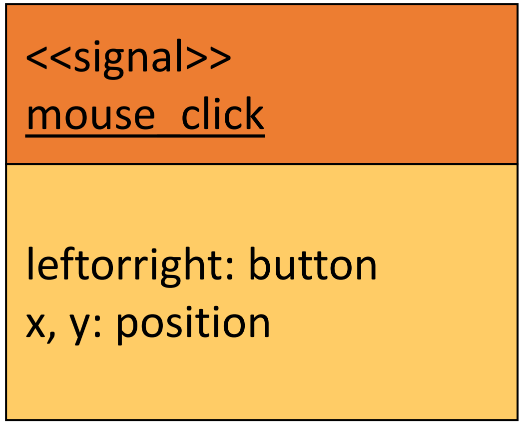
Event-driven state machines

- Behavioral descriptions are written as event-driven state machines.
  - Machine changes state when receiving an input.
- An event may come from inside or outside of the system.

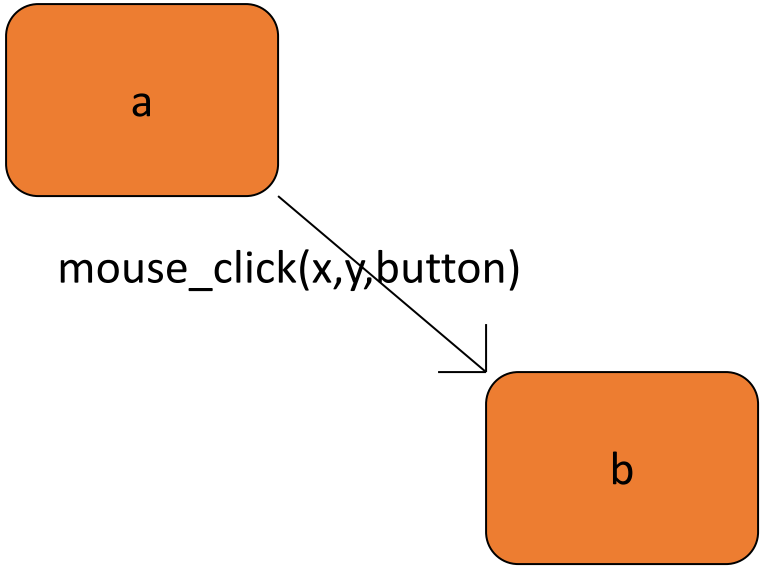
Types of events

- Signal: asynchronous event.
- Call: synchronized communication.
- Timer: activated by time.

Signal event

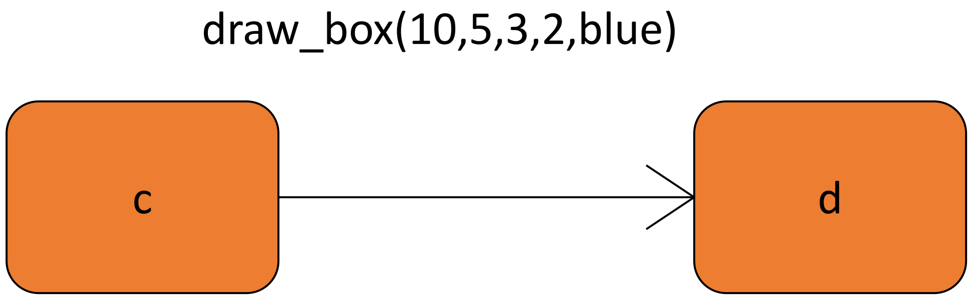


declaration

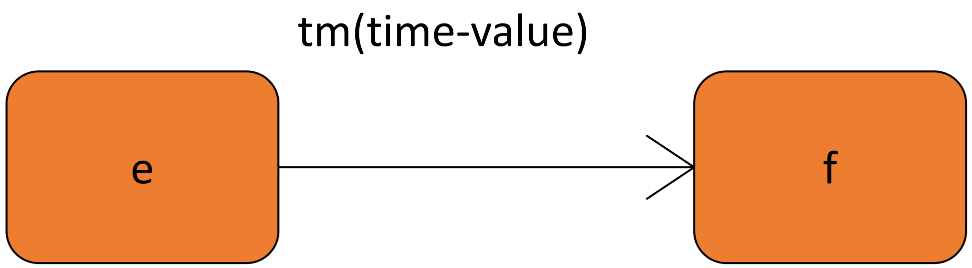


event description

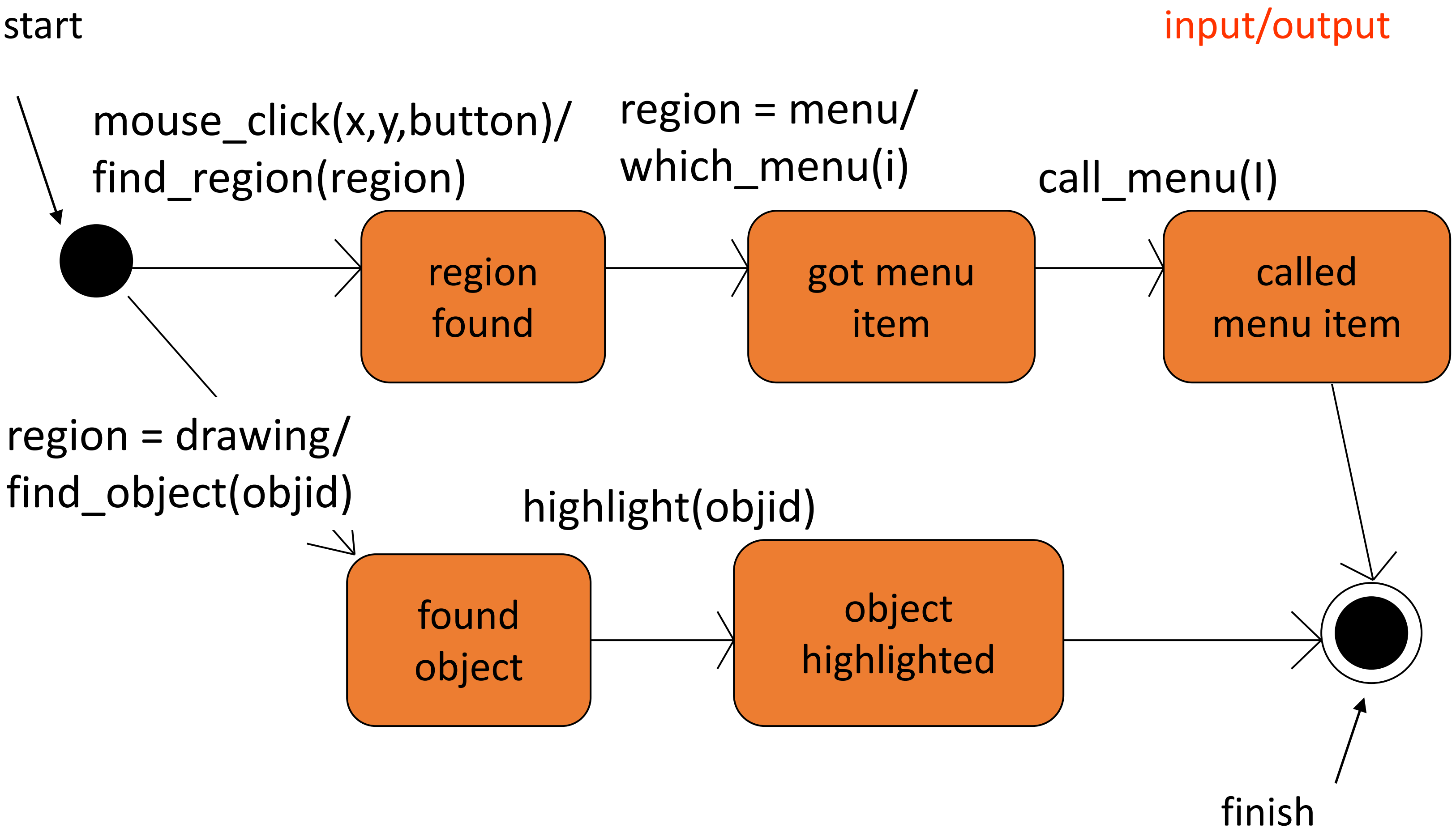
Call event



Timer event



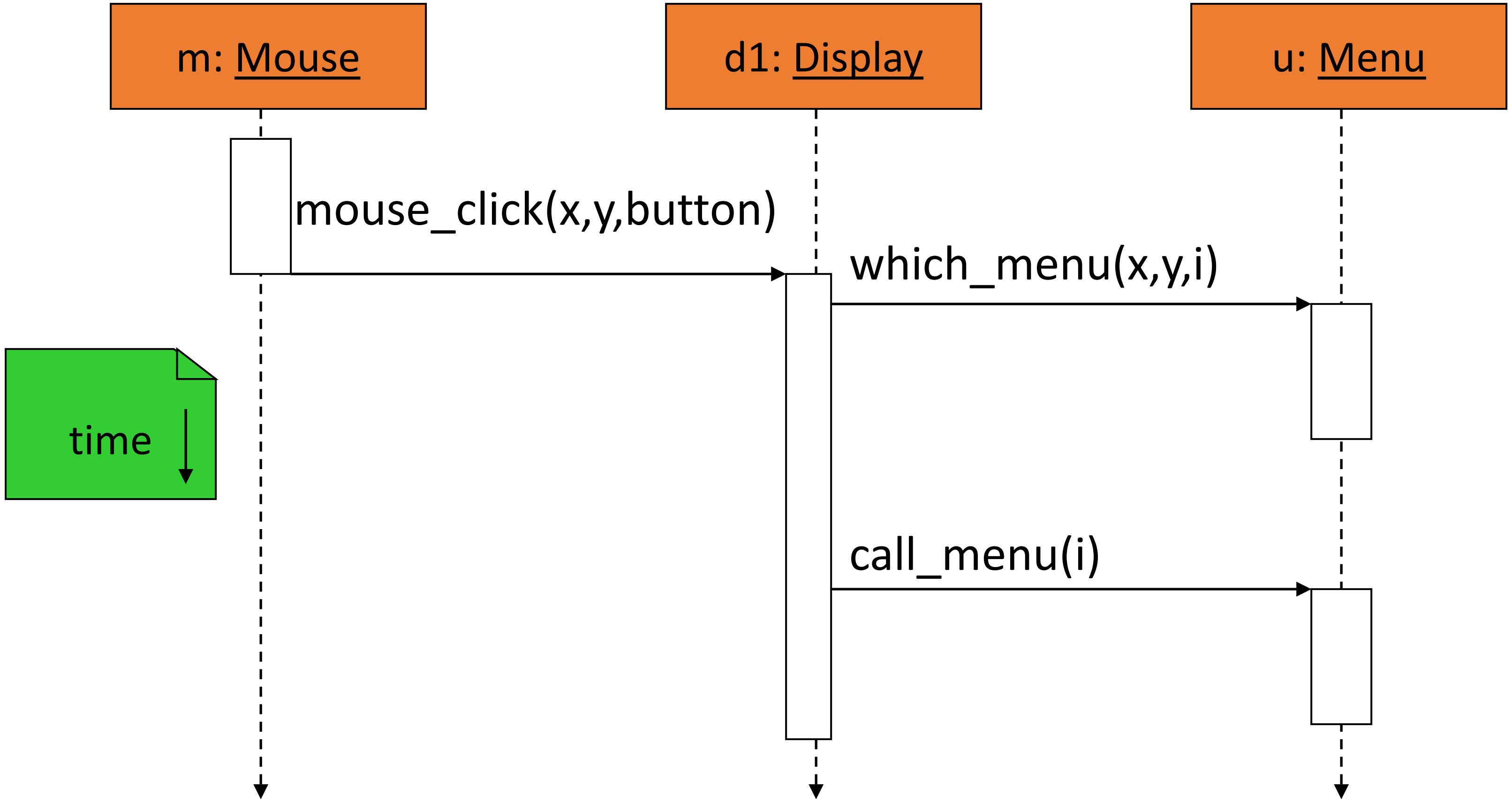
Example state machine



Sequence diagram

- Shows sequence of operations over time.
- Relates behaviors of multiple objects.

Sequence diagram example



Summary

- Object-oriented design helps us organize a design.
- UML is a transportable system design language.
  - Provides structural and behavioral description primitives.

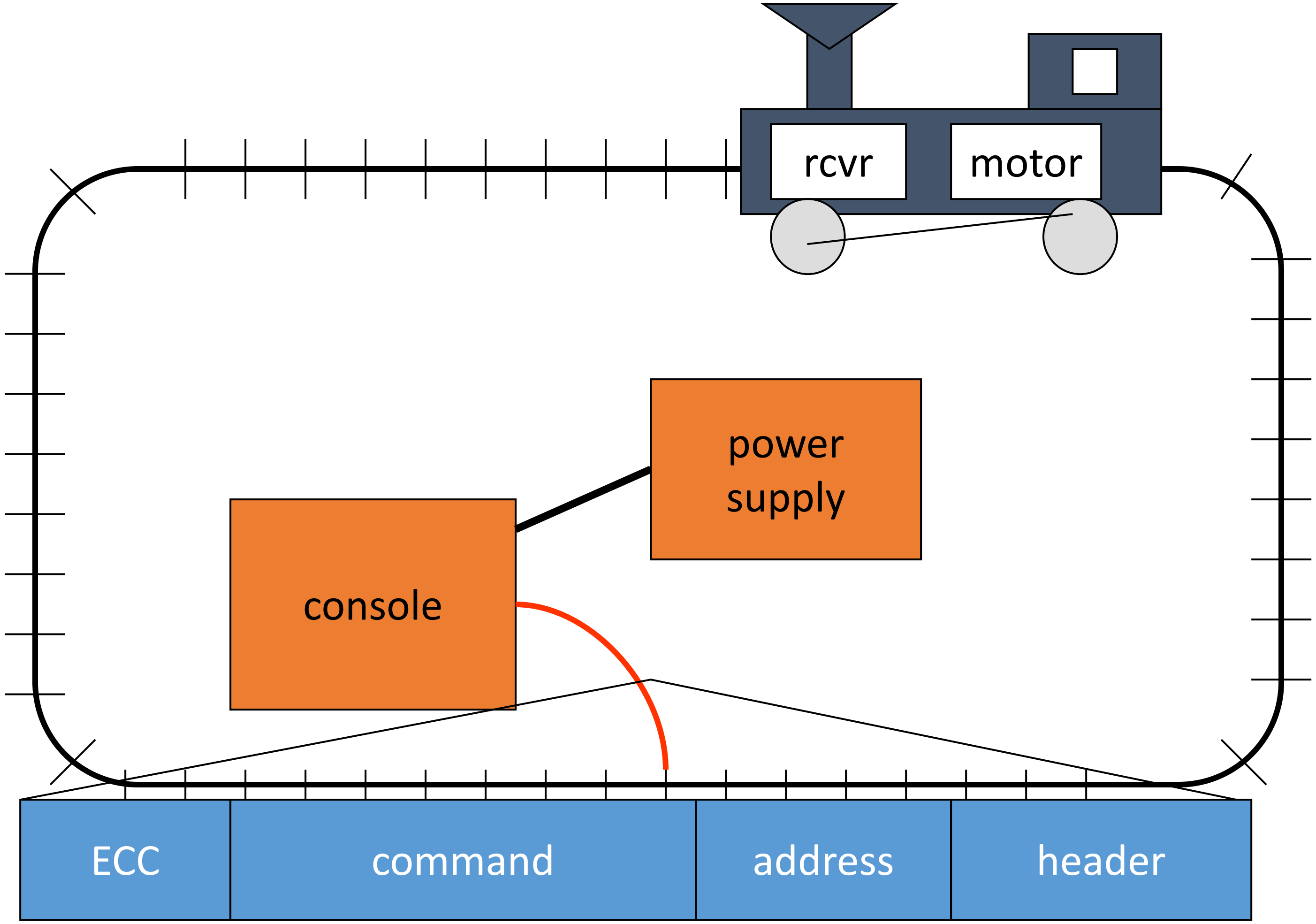
Introduction

- Example: model train controller.

Purposes of example

- Follow a design through several levels of abstraction.
- Gain experience with UML.

Model train setup



Digital Command Control

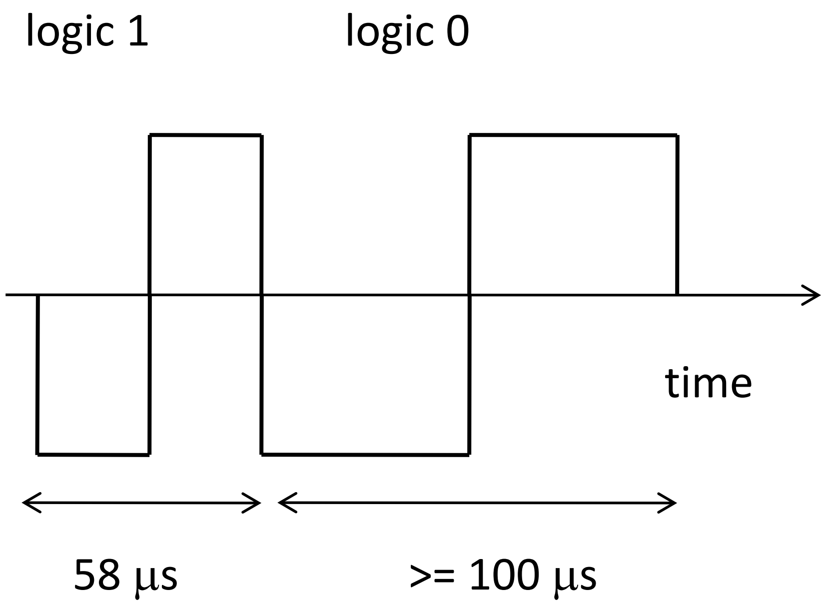
- DCC created by model railroad hobbyists, picked up by industry.
- Defines way in which model trains, controllers communicate.
  - Leaves many system design aspects open, allowing competition.
- This is a simple example of a big trend:
  - Cell phones, digital TV rely on standards.

DCC documents

- Standard S-9.1, DCC Electrical Standard.
  - Defines how bits are encoded on the rails.
- Standard S-9.2, DCC Communication Standard.
  - Defines packet format and semantics.

DCC electrical standard

- Voltage moves around the power supply voltage; adds no DC component.
- 1 is 58 μs, 0 is at least 100 μs.



DCC communication standard

- Basic packet format: PSA(sD)+E.
- P: preamble = 1111111111.
- S: packet start bit = 0.
- A: address data byte.
- s: data byte start bit.
- D: data byte (data payload).
- E: packet end bit = 1.

DCC packet types

- Baseline packet: minimum packet that must be accepted by all DCC implementations.
  - Address data byte gives receiver address.
  - Instruction data byte gives basic instruction.
  - Error correction data byte gives ECC.

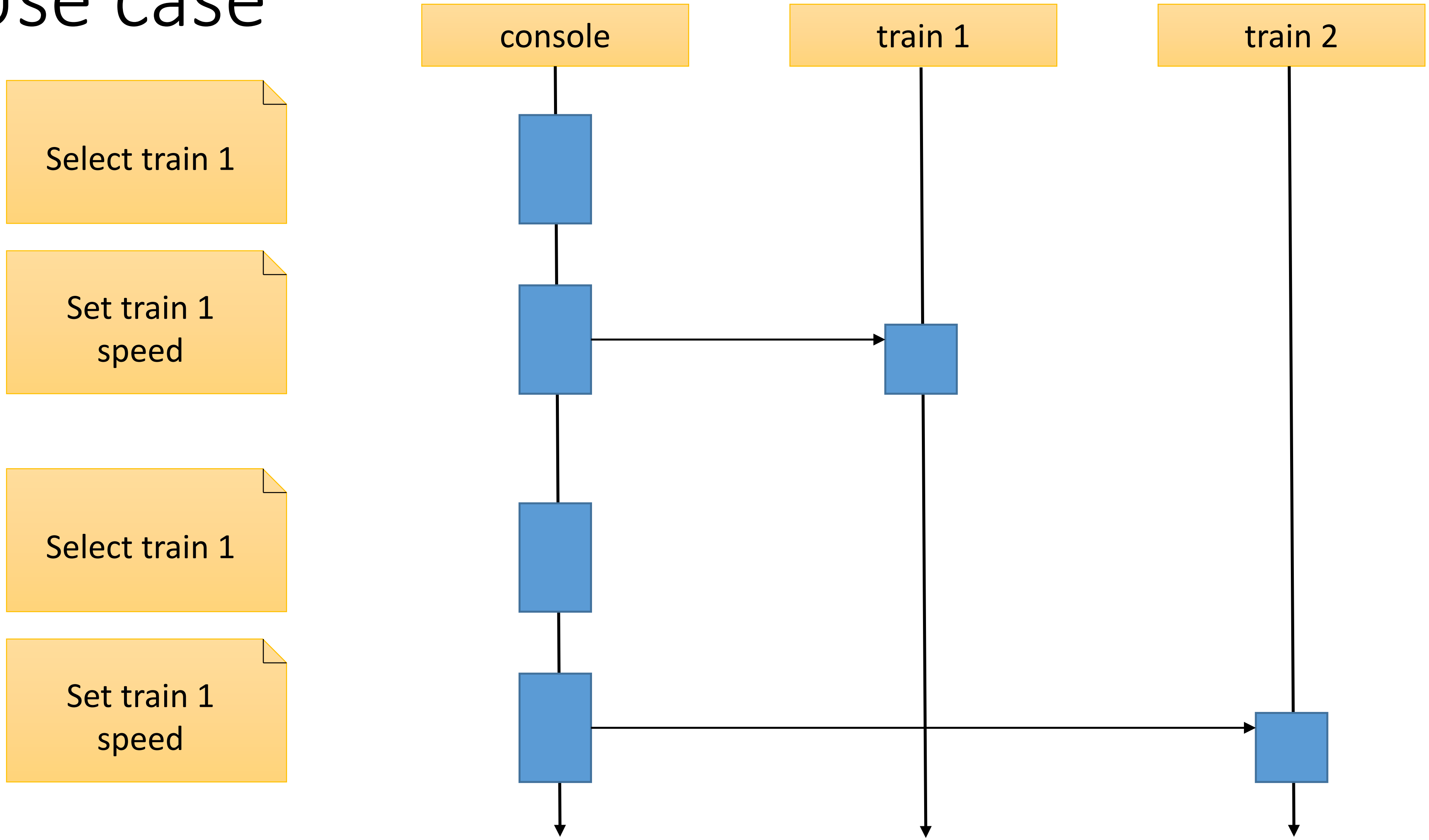
Requirements

- Console can control 8 trains on 1 track.
- Throttle has at least 63 levels.
- Inertia control adjusts responsiveness with at least 8 levels.
- Emergency stop button.
- Error detection scheme on messages.

Requirements form

name	model train controller
purpose	control speed of <= 8 model trains
inputs	throttle, inertia, emergency stop, train #
outputs	train control signals
functions	set engine speed w. inertia; emergency stop
performance	can update train speed at least 10 times/sec
manufacturing cost	$50
power	wall powered
physical size/weight	console comfortable for 2 hands; < 2 lbs.

Use case


Select train 1
Set train 1 speed
Select train 1
Set train 1 speed
console
train 1
train 2

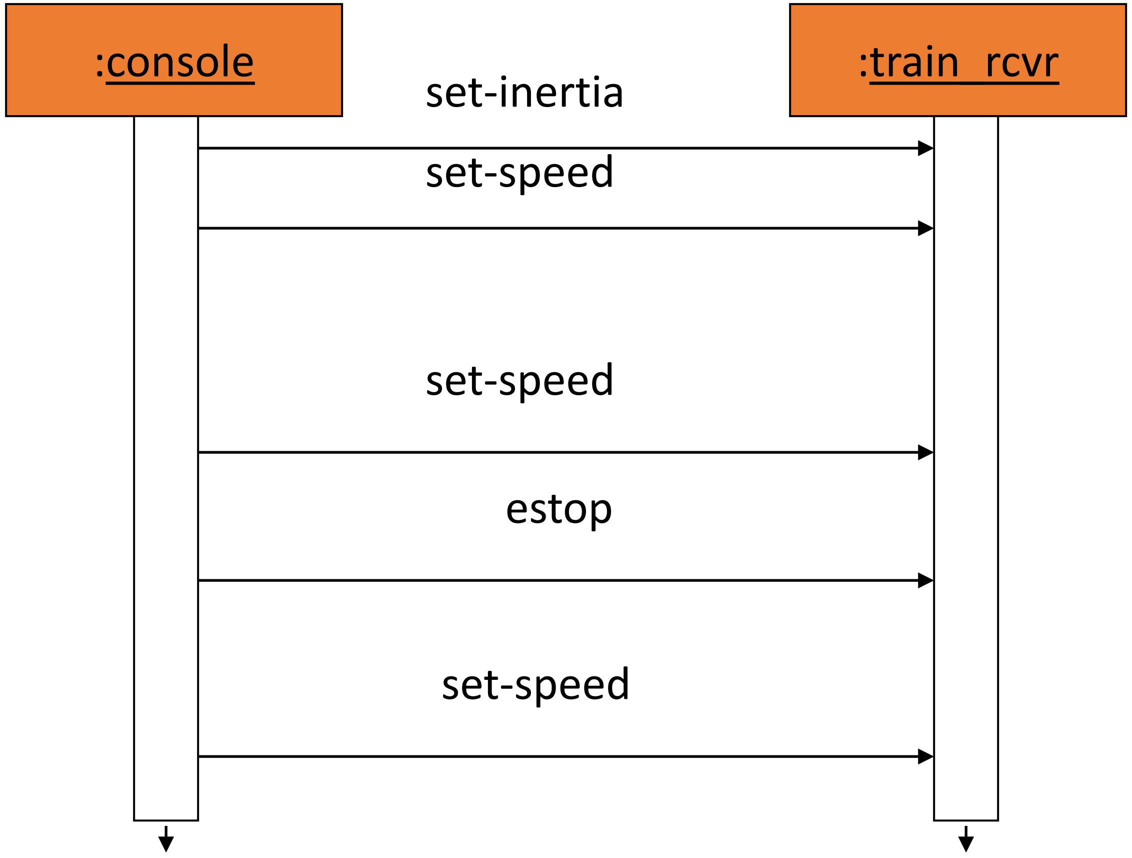
Conceptual specification

- Before we create a detailed specification, we will make an initial, simplified specification.
  - Gives us practice in specification and UML.
  - Good idea in general to identify potential problems before investing too much effort in detail.

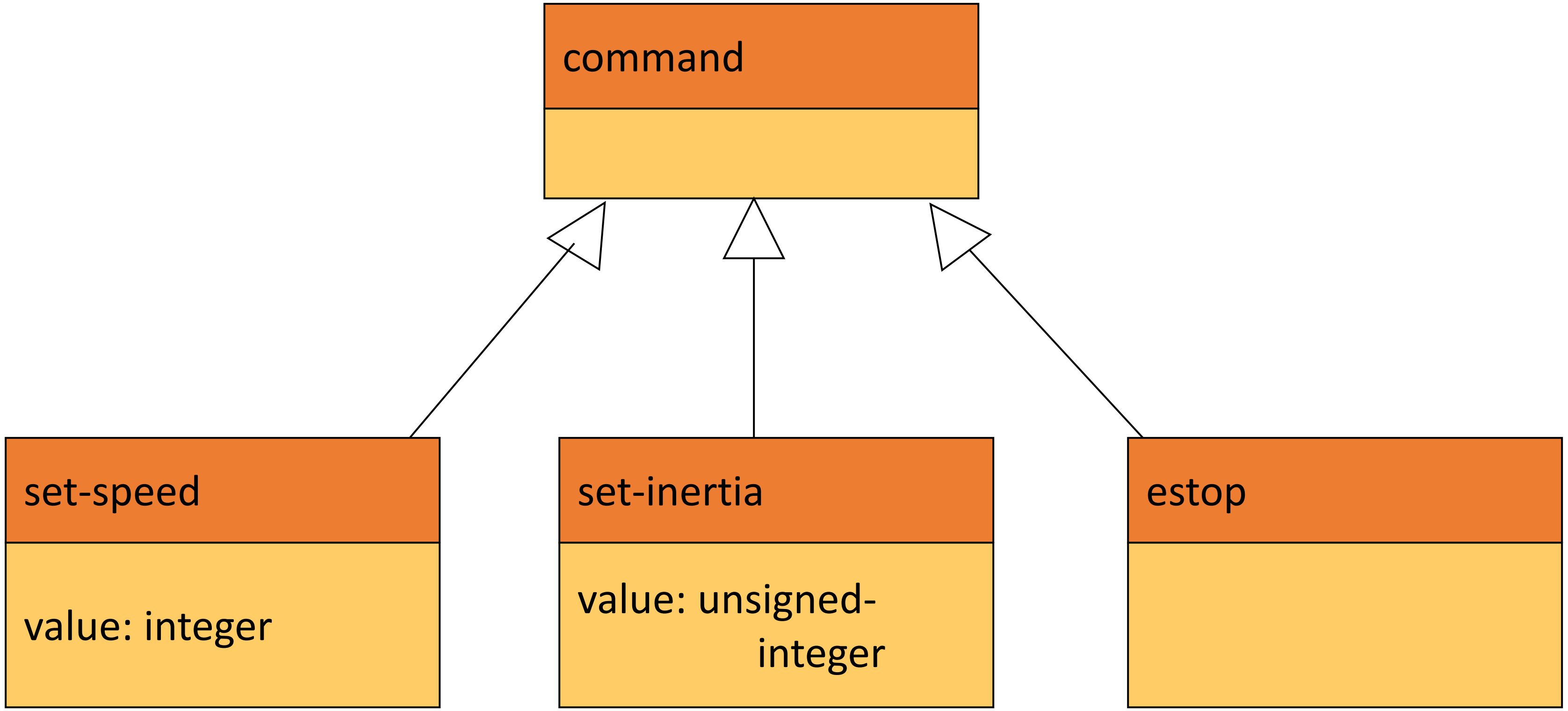
Basic system commands

command name	parameters
set-speed	speed (positive/negative)
set-inertia	inertia-value (non-negative)
estop	none

Typical control sequence



Message classes

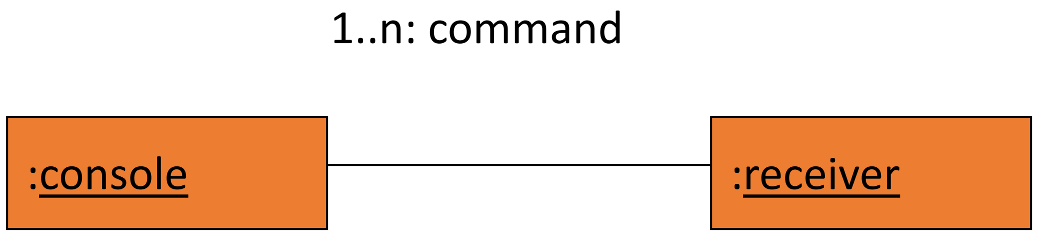


Roles of message classes

- Implemented message classes derived from message class.
  - Attributes and operations will be filled in for detailed specification.
- Implemented message classes specify message type by their class.
  - May have to add type as parameter to data structure in implementation.

Subsystem collaboration diagram

Shows relationship between console and receiver (ignores role of track):



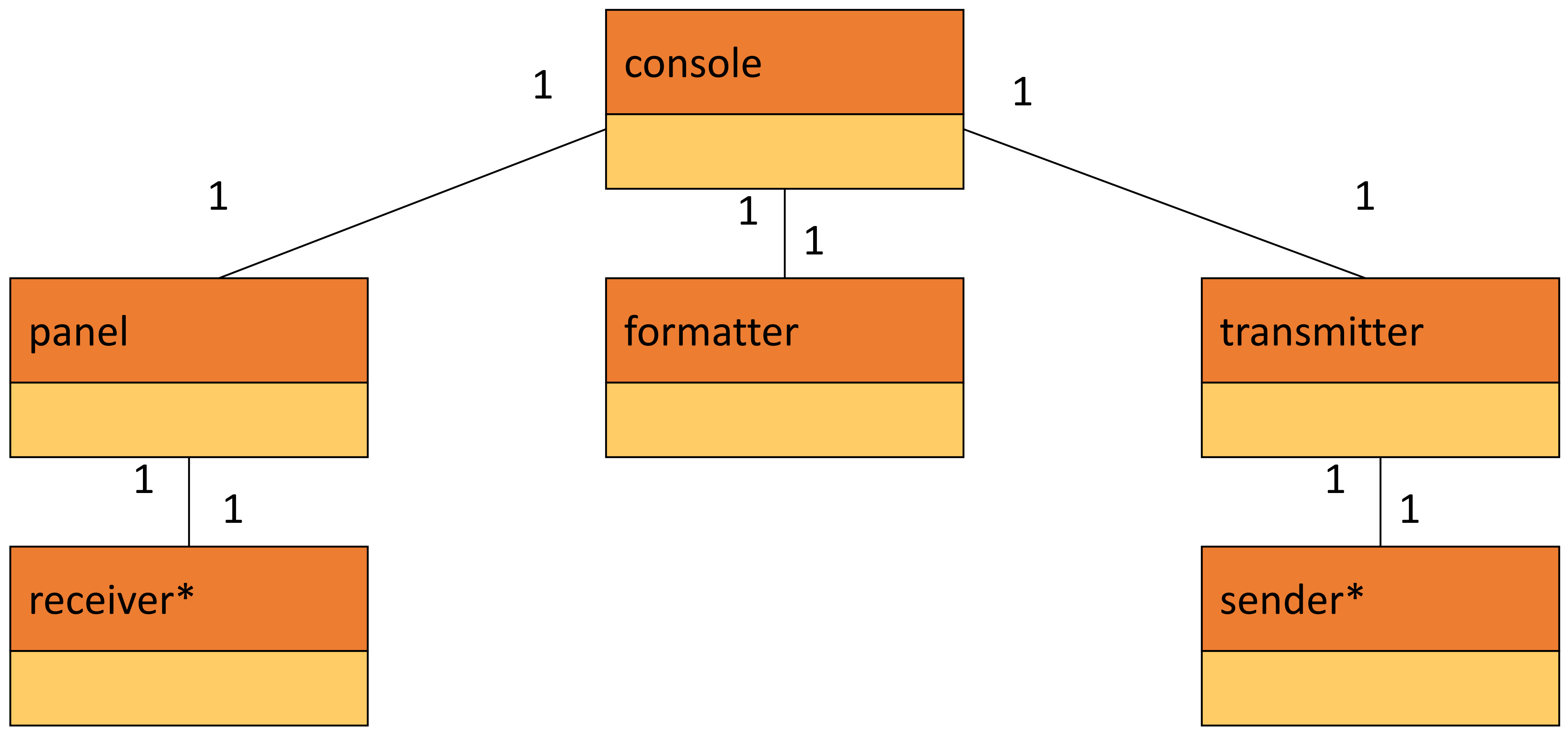
System structure modeling

- Some classes define non-computer components.
  - Denote by *name.
- Choose important systems at this point to show basic relationships.

Major subsystem roles

- Console:
  - read state of front panel;
  - format messages;
  - transmit messages.
- Train:
  - receive message;
  - interpret message;
  - control the train.

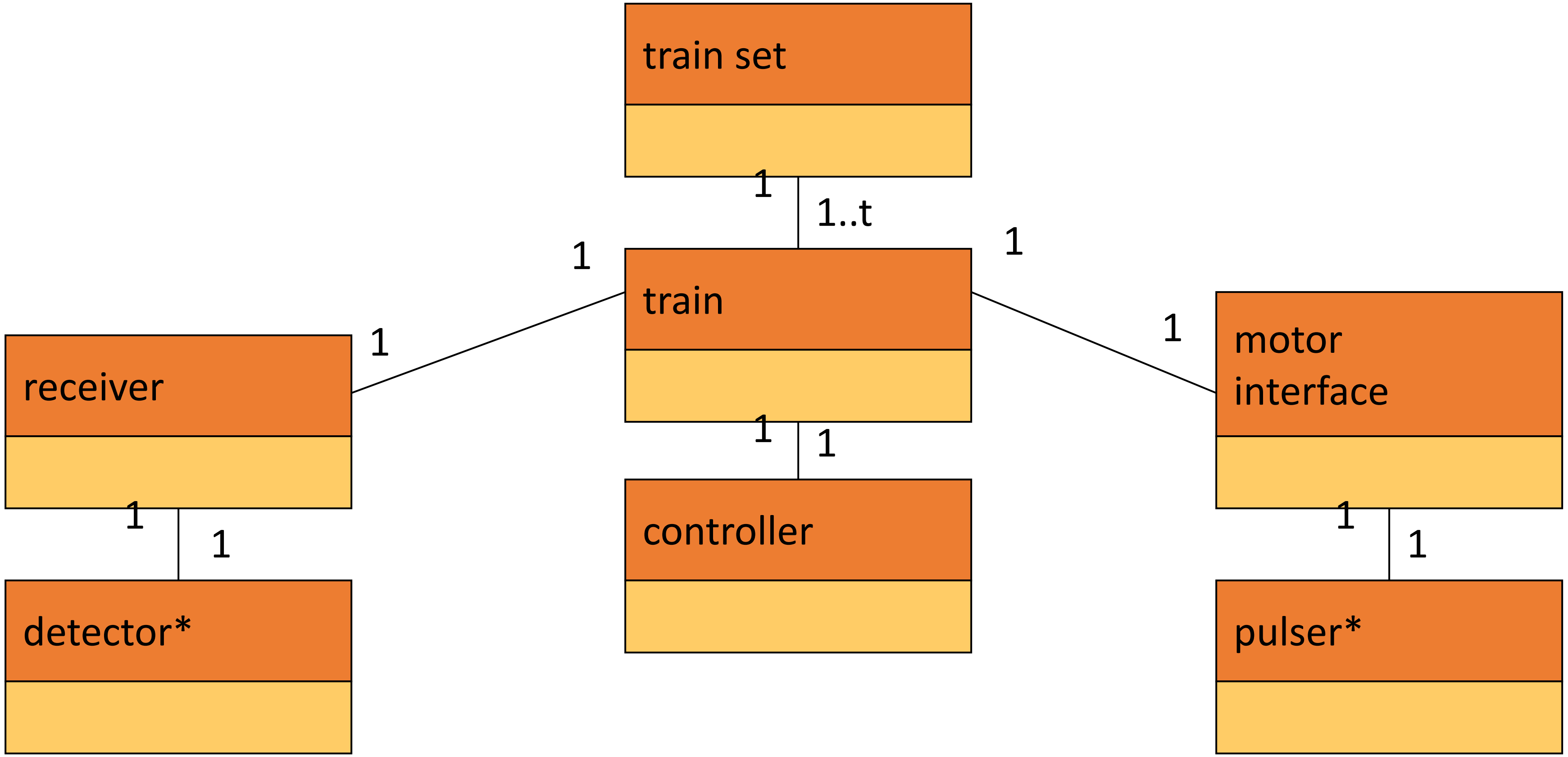
Console system classes



Console class roles

- panel: describes analog knobs and interface hardware.
- formatter: turns knob settings into bit streams.
- transmitter: sends data on track.

Train system classes



Train class roles

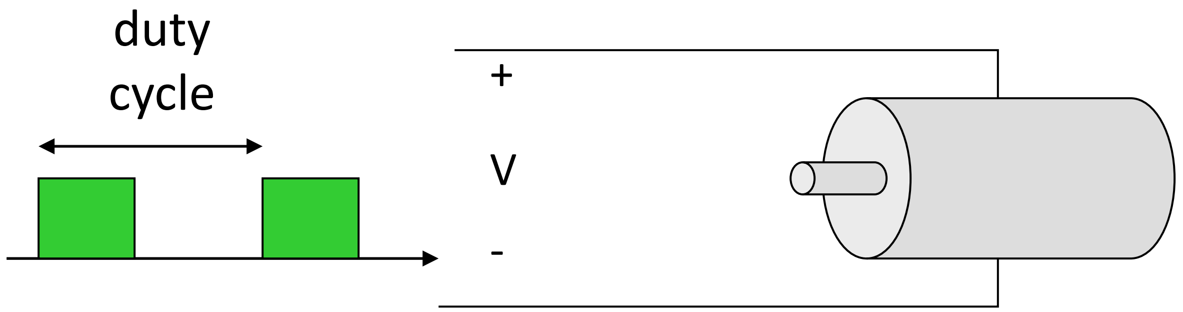
- receiver: digitizes signal from track.
- controller: interprets received commands and makes control decisions.
- motor interface: generates signals required by motor.

Detailed specification

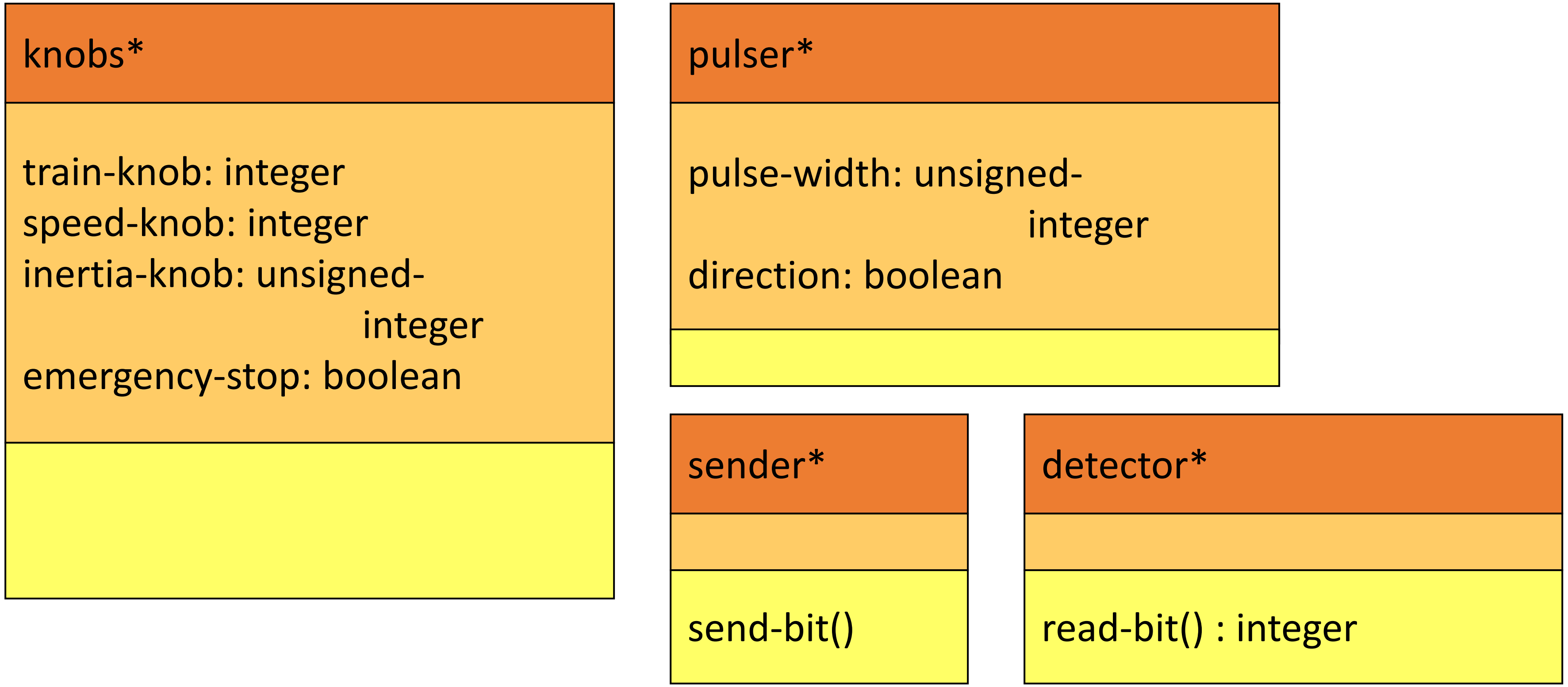
- We can now fill in the details of the conceptual specification:
  - more classes;
  - behaviors.
- Sketching out the spec first helps us understand the basic relationships in the system.

Train speed control

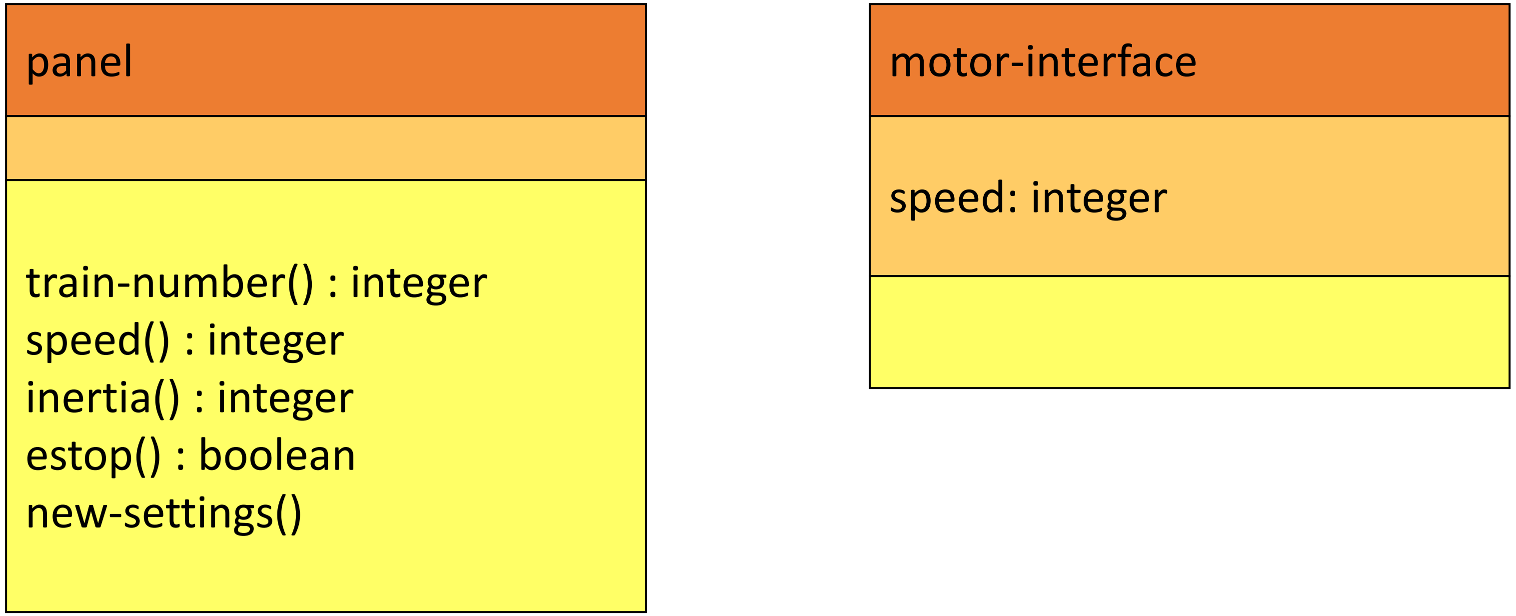
- Motor controlled by pulse width modulation:



Console physical object classes



Panel and motor interface classes



Class descriptions

- panel class defines the controls.
  - new-settings() behavior reads the controls.
- motor-interface class defines the motor speed held as state.

Transmitter and receiver classes

transmitter
send-speed(adrs: integer, speed: integer) send-inertia(adrs: integer, val: integer) set-estop(adrs: integer)

receiver
current: command new: boolean
read-cmd() new-cmd() : boolean rcv-type(msg-type: command) rcv-speed(val: integer) rcv-inertia(val:integer)

Class descriptions

- transmitter class has one behavior for each type of message sent.
- receiver function provides methods to:
  - detect a new message;
  - determine its type;
  - read its parameters (estop has no parameters).

Formatter class

formatter
current-train: integer current-speed[ntrains]: integer current-inertia[ntrains]: unsigned-integer current-estop[ntrains]: boolean
send-command() panel-active() : boolean operate()

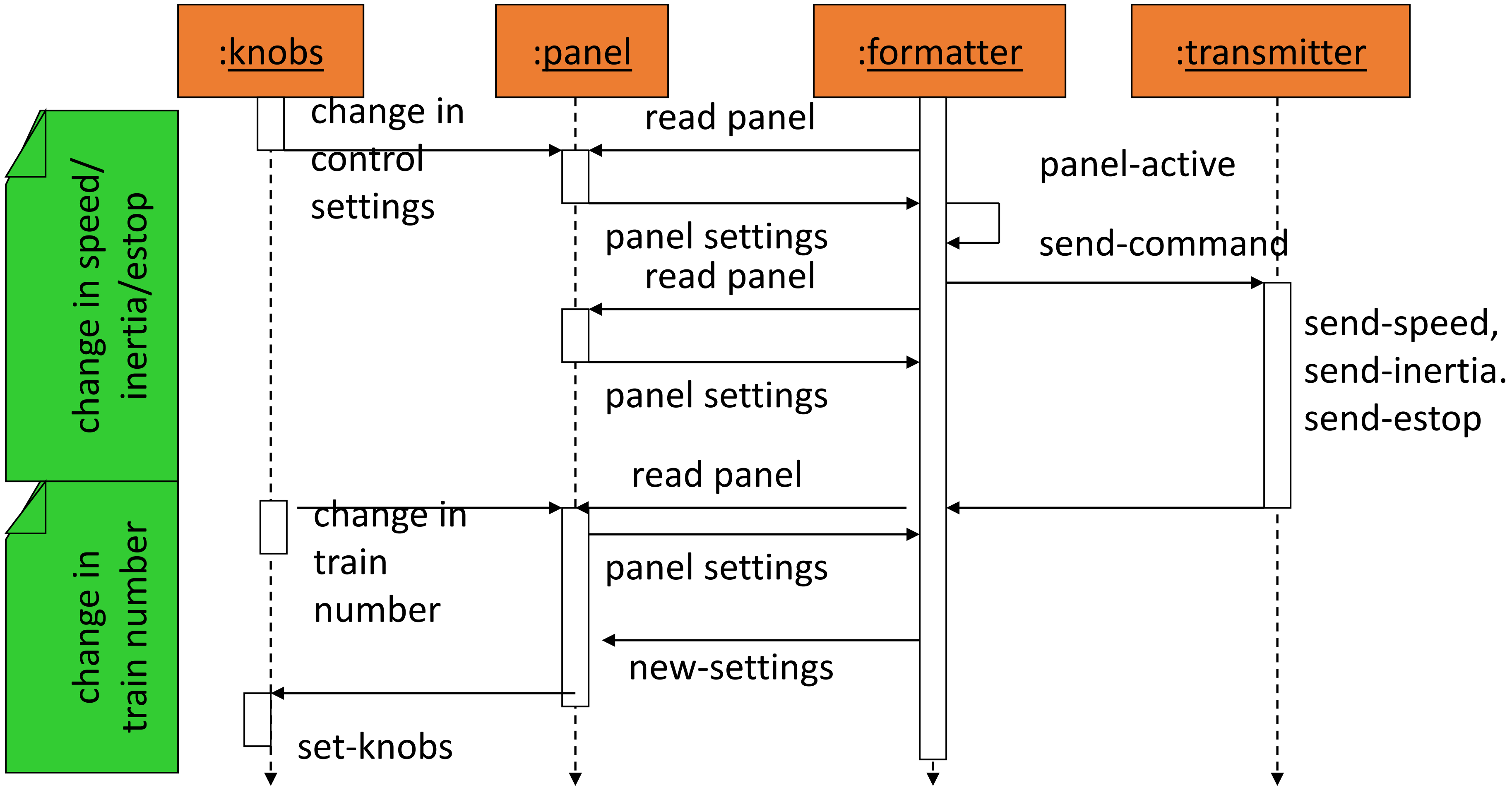
Formatter class description

- Formatter class holds state for each train, setting for current train.
- The operate() operation performs the basic formatting task.

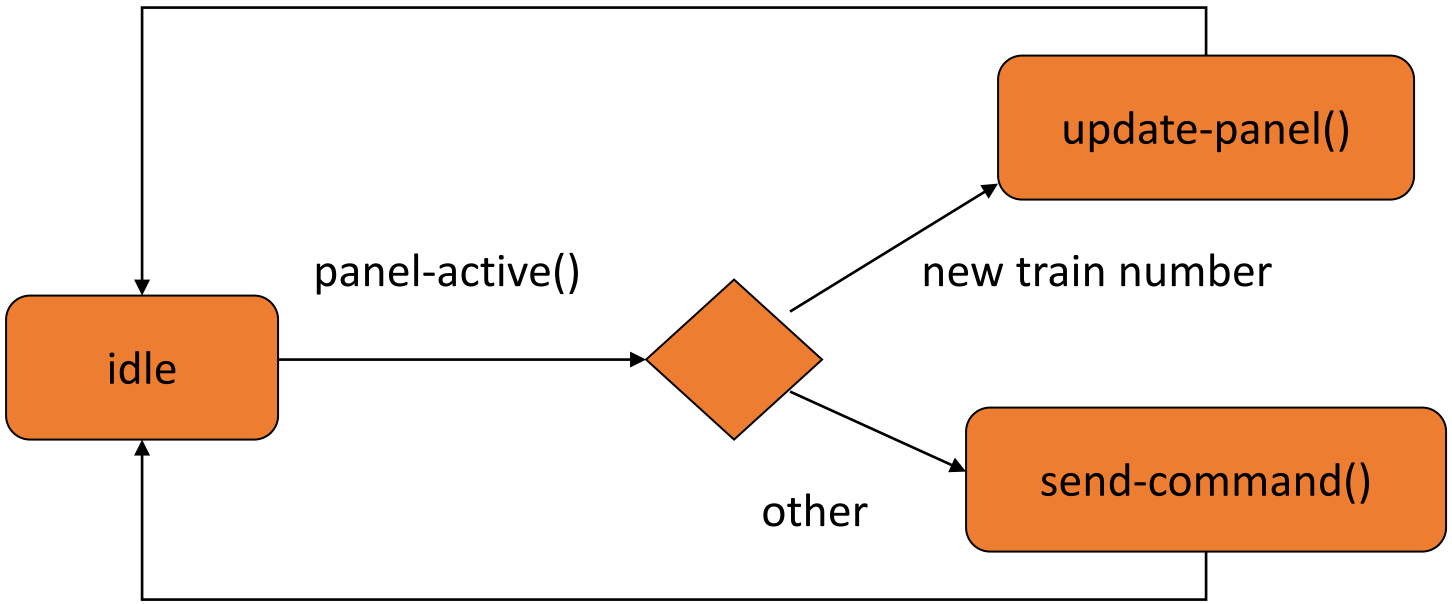
Control input cases

- Use a soft panel to show current panel settings for each train.
- Changing train number:
  - must change soft panel settings to reflect current train’s speed, etc.
- Controlling throttle/inertia/estop:
  - read panel, check for changes, perform command.

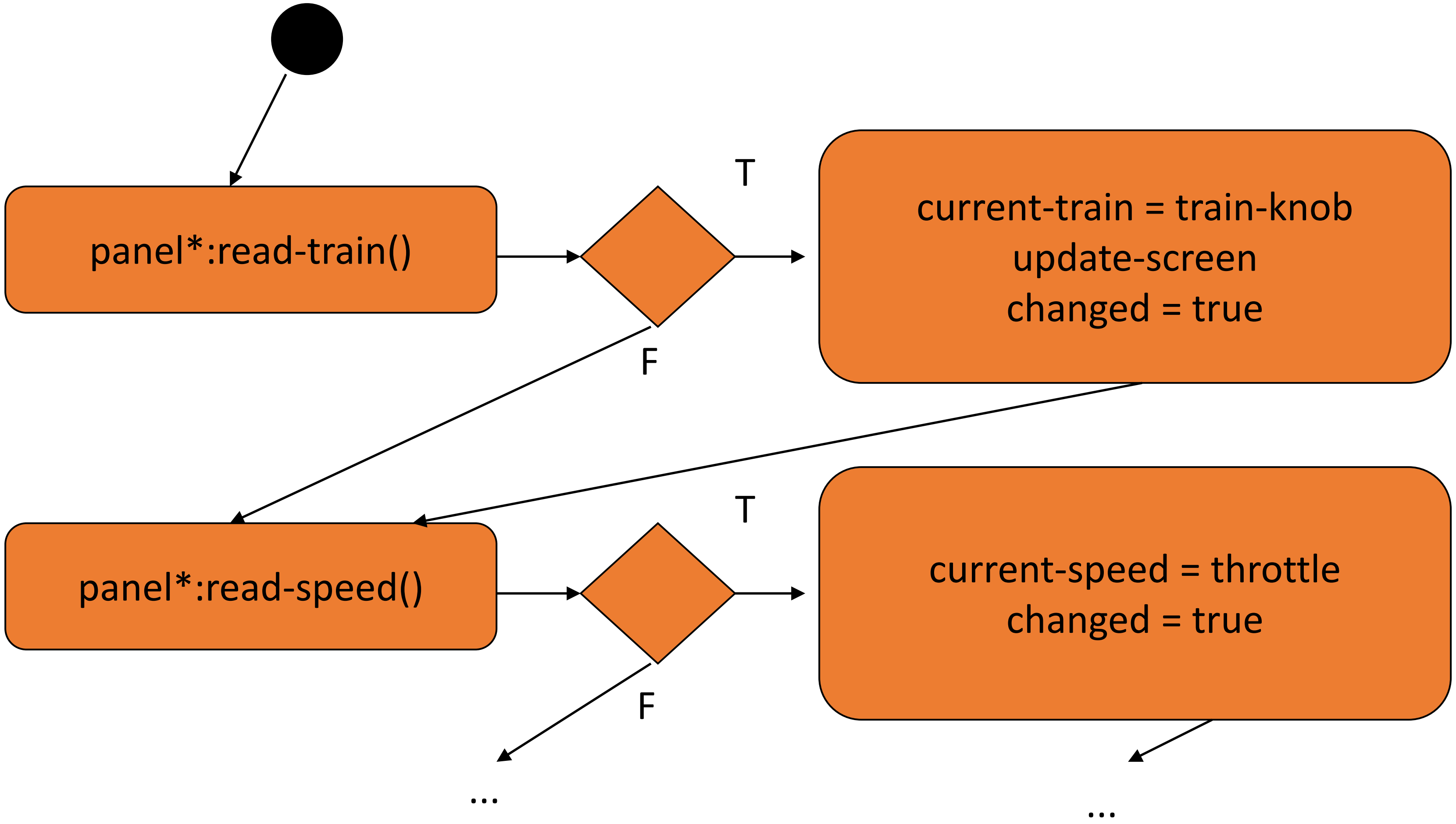
Control input sequence diagram



Formatter operate behavior



Panel-active behavior



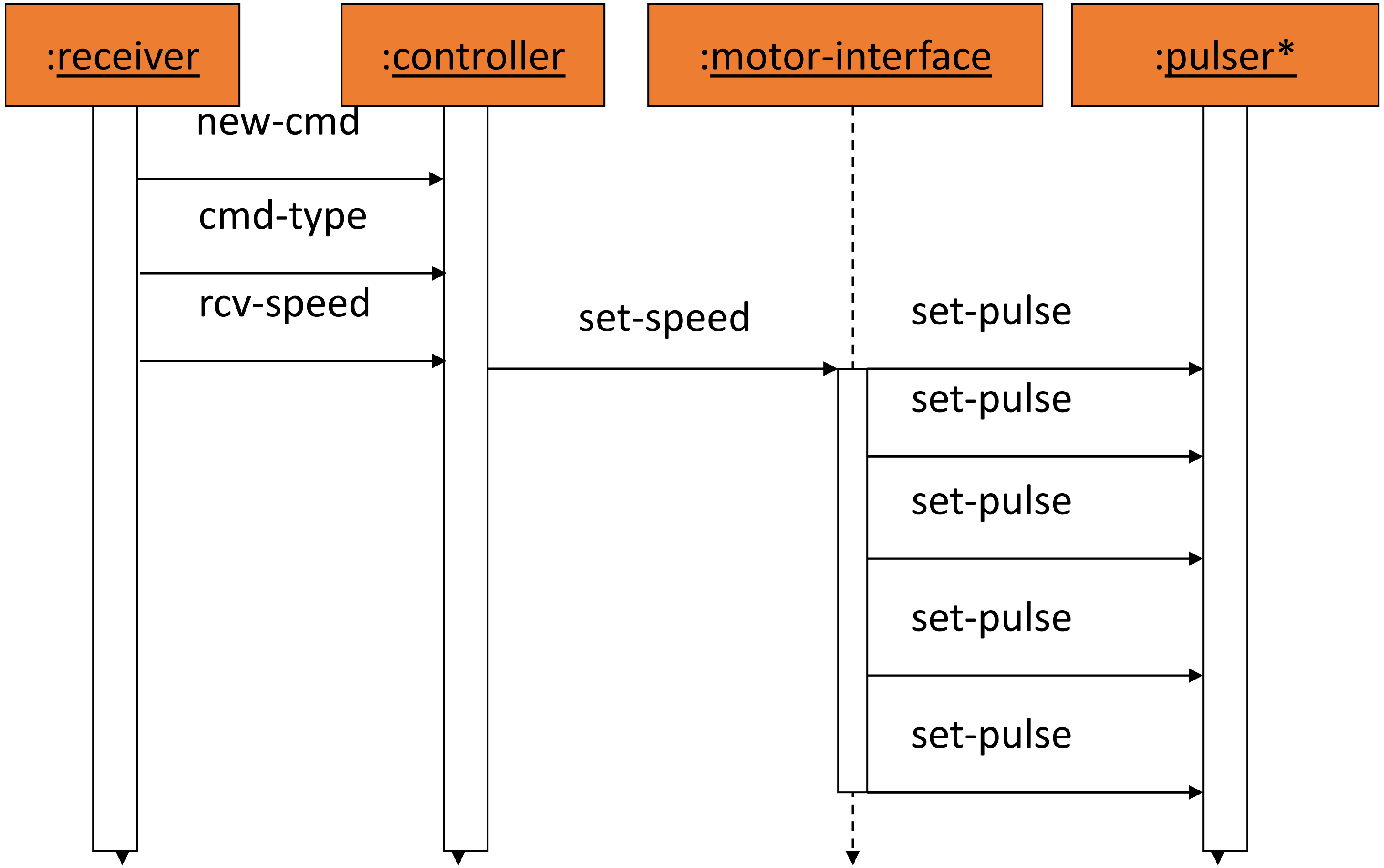
Controller class

controller
current-train: integer current-speed: integer current-direction: boolean current-inertia: unsigned-integer
operate() issue-command()

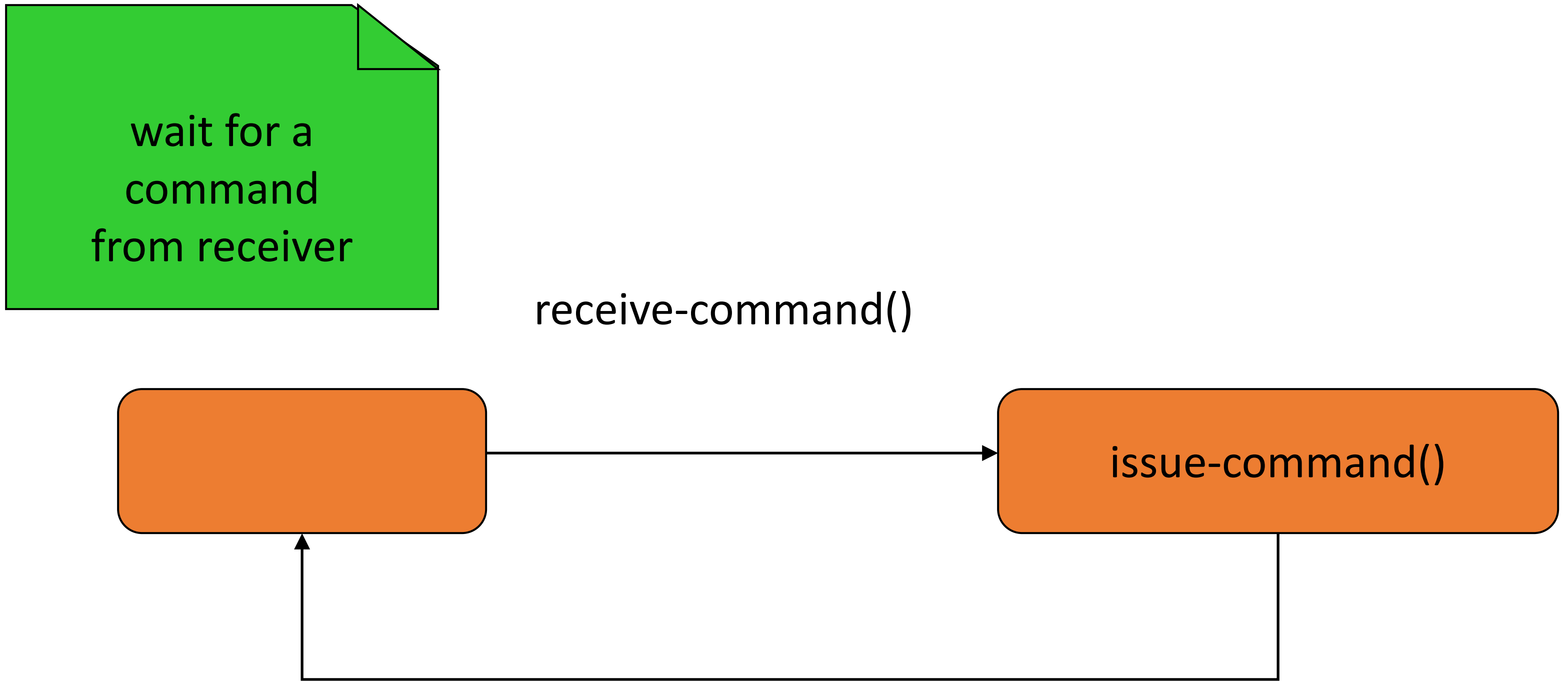
Setting the speed

- Don’t want to change speed instantaneously.
- Controller should change speed gradually by sending several commands.

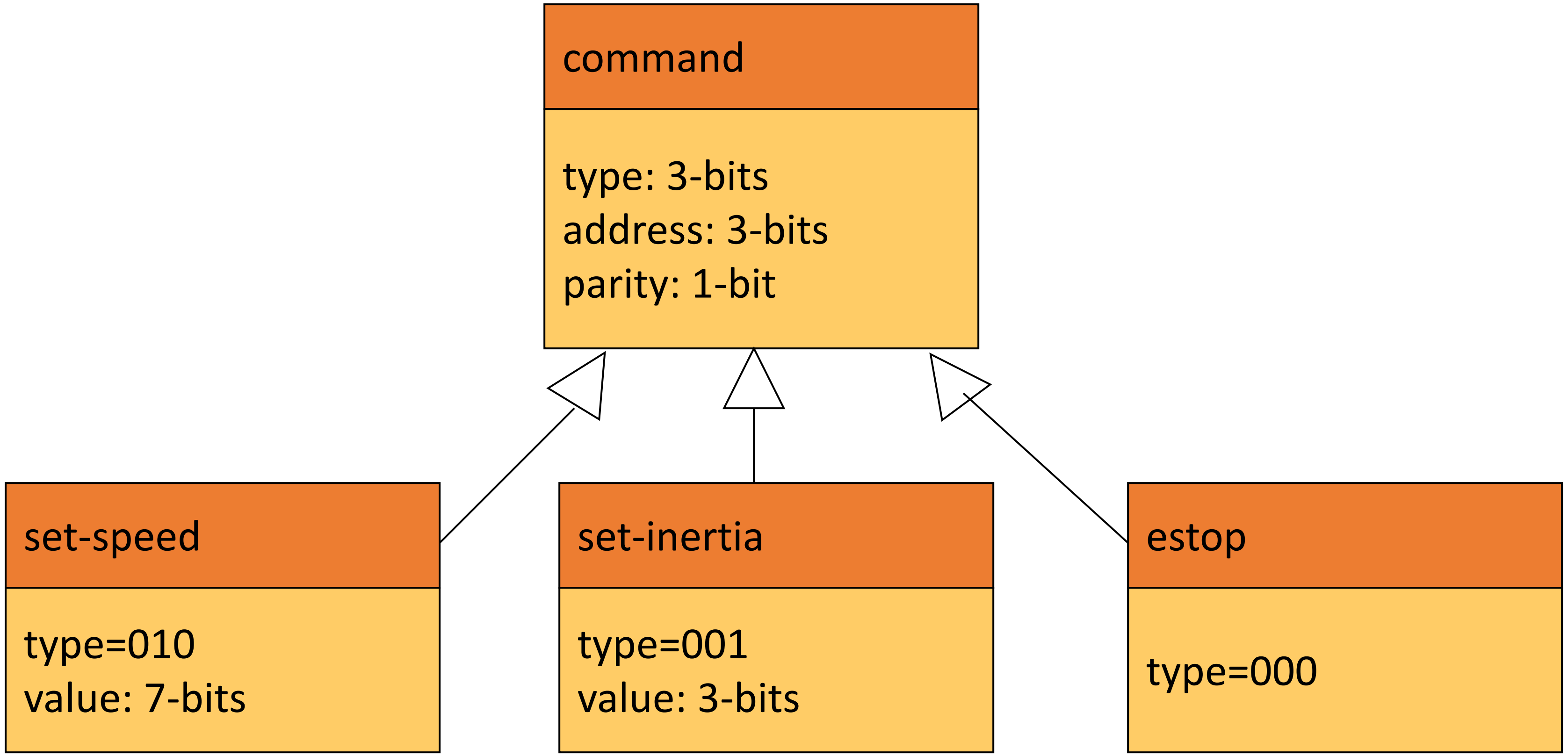
Sequence diagram for set-speed command



Controller operate behavior



Refined command classes



Summary

- Separate specification and programming.
  - Small mistakes are easier to fix in the spec.
  - Big mistakes in programming cost a lot of time.
- You can’t completely separate specification and architecture.
  - Make a few tasteful assumptions.