System design techniques

- Design methodologies.
- Requirements and specification.

Design methodologies

- Process for creating a system.
- Many systems are complex:
  - large specifications;
  - multiple designers;
  - interface to manufacturing.
- Proper processes improve:
  - quality;
  - cost of design and manufacture.

Product metrics

- Time-to-market:
  - beat competitors to market;
  - meet marketing window (back-to-school).
- Design cost.
- Manufacturing cost.
- Quality.

Mars Climate Observer

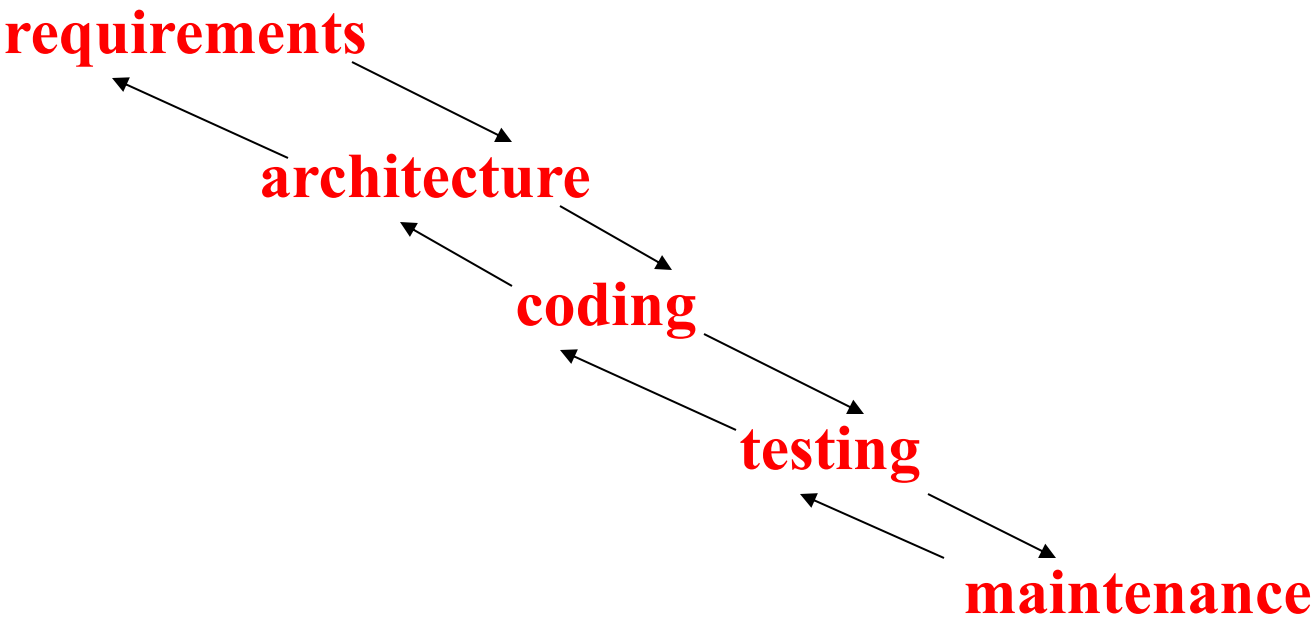
- Lost on Mars in September 1999.
- Requirements problem:
  - Requirements did not specify units.
  - Lockheed Martin used English; JPL wanted metric.
- Not caught by manual inspections.

Design flow

- Design flow: sequence of steps in a design methodology.
- May be partially or fully automated.
  - Use tools to transform, verify design.
- Design flow is one component of methodology. Methodology also includes management organization, etc.

Waterfall model

- Early model for software development:



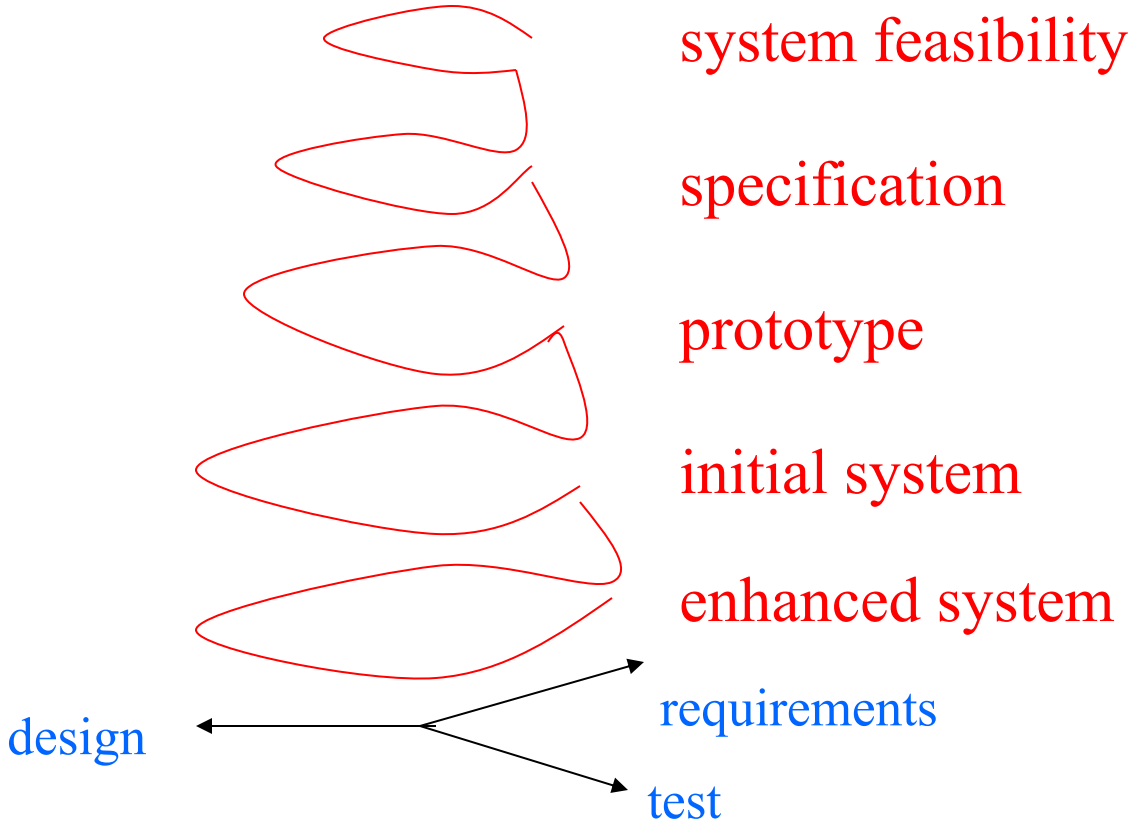
Waterfall model steps

- Requirements: determine basic characteristics.
- Architecture: decompose into basic modules.
- Coding: implement and integrate.
- Testing: exercise and uncover bugs.
- Maintenance: deploy, fix bugs, upgrade.

Waterfall model critique

- Only local feedback---may need iterations between coding and requirements, for example.
- Doesn’t integrate top-down and bottom-up design.
- Assumes hardware is given.

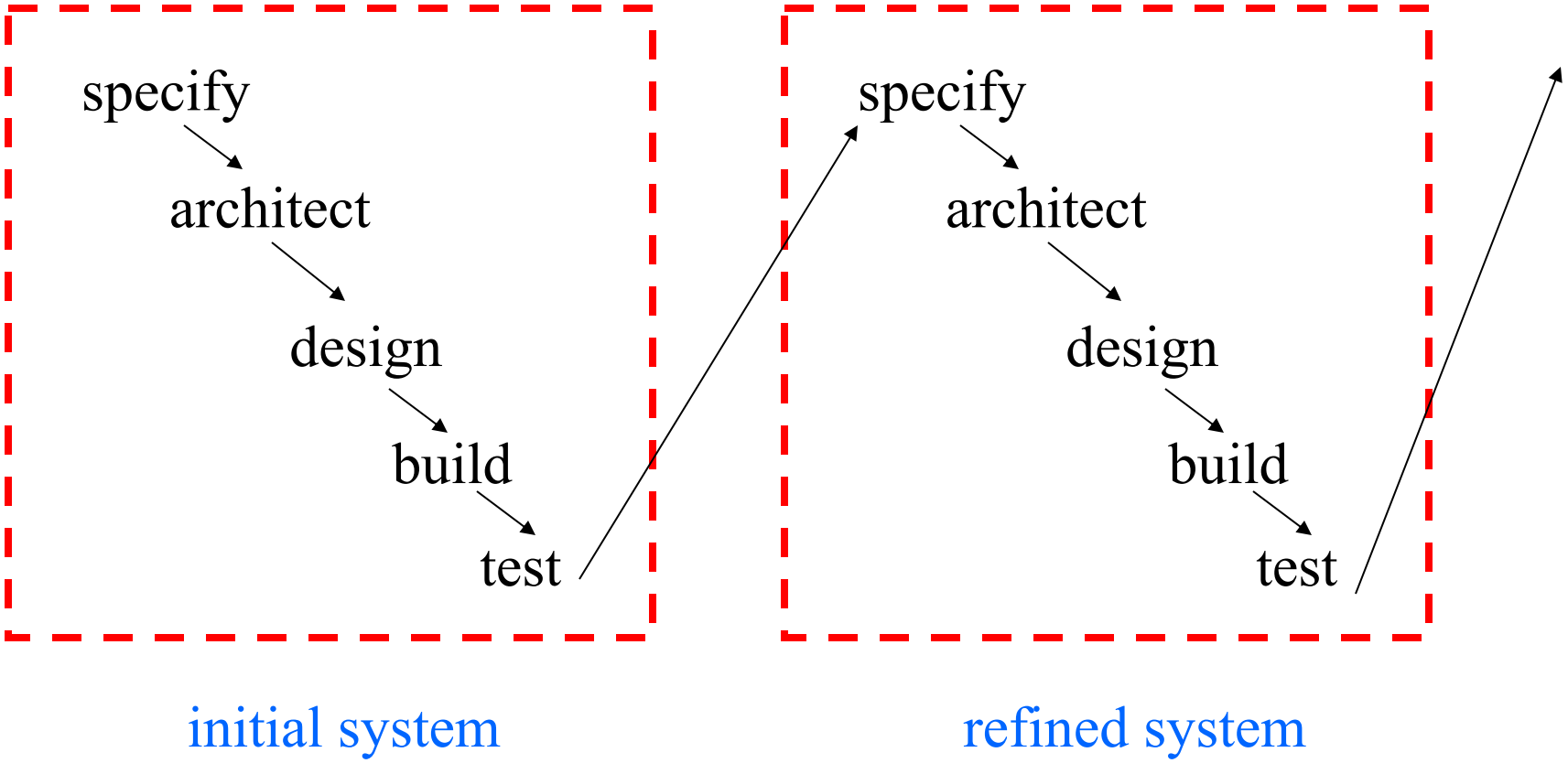
Spiral model



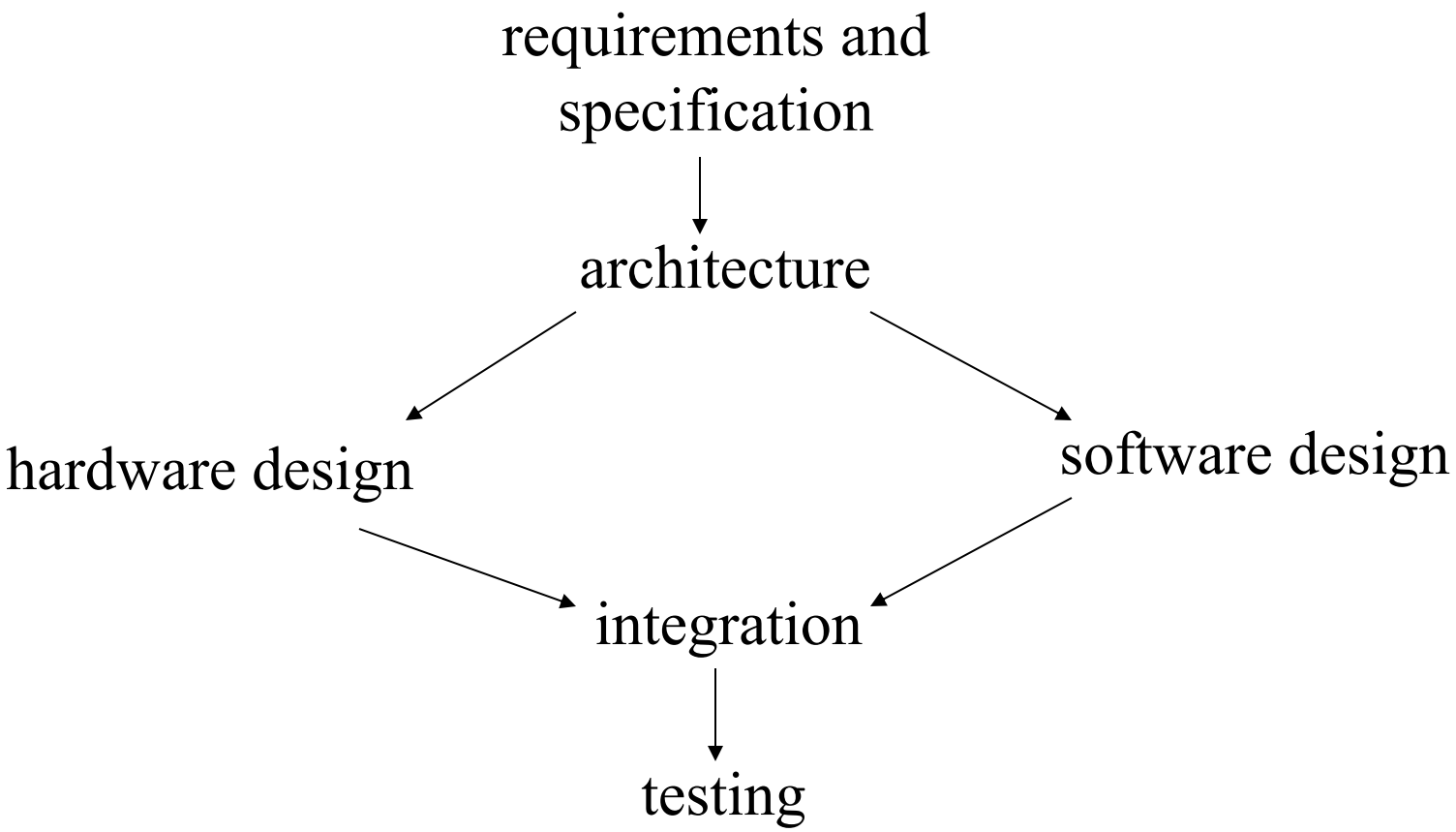
Spiral model critique

- Successive refinement of system.
  - Start with mock-ups, move through simple systems to full-scale systems.
- Provides bottom-up feedback from previous stages.
- Working through stages may take too much time.

Successive refinement model



Hardware/software design flow



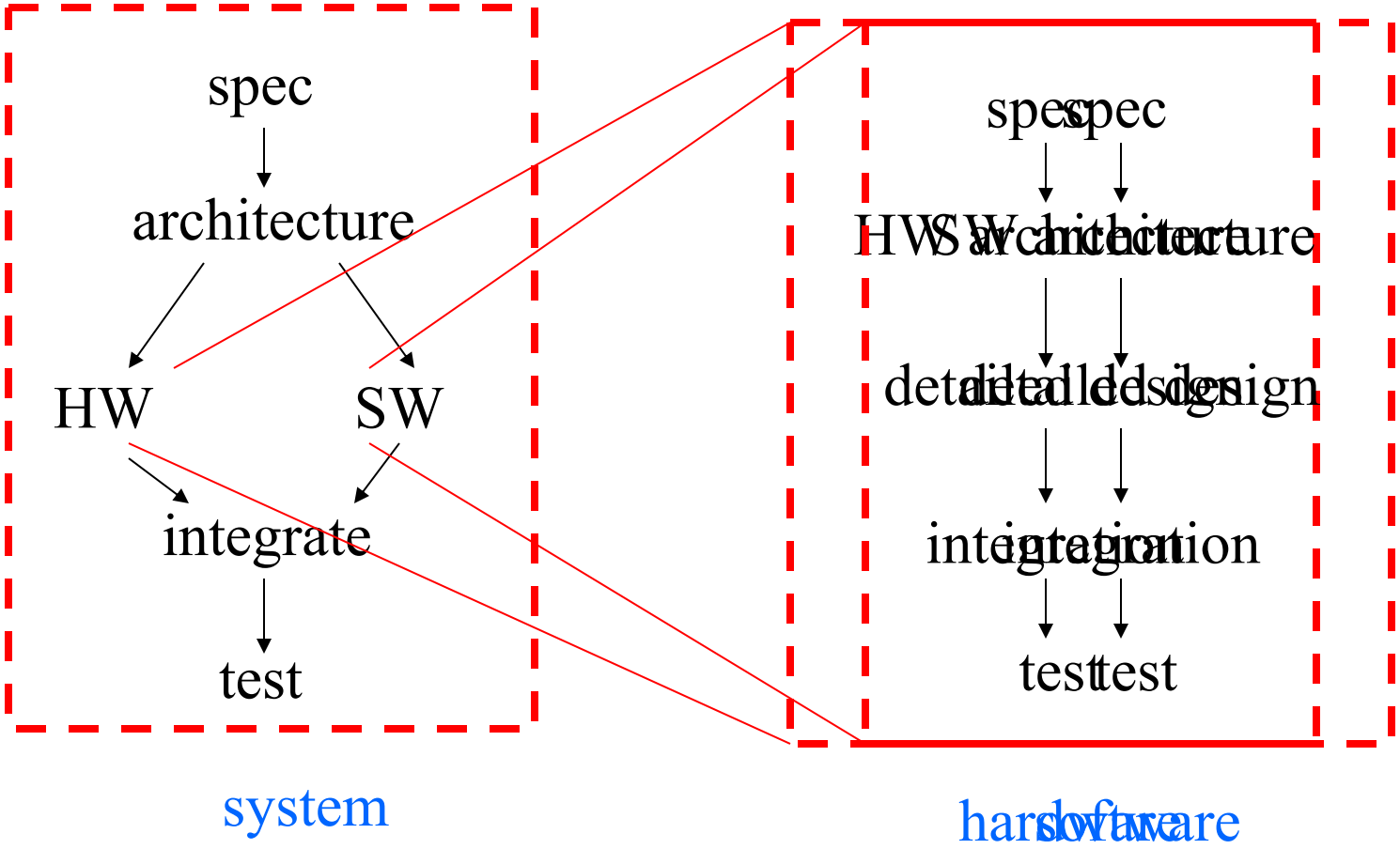
Co-design methodology

- Must architect hardware and software together:
  - provide sufficient resources;
  - avoid software bottlenecks.
- Can build pieces somewhat independently, but integration is major step.
- Also requires bottom-up feedback.

Hierarchical design flow

- Embedded systems must be designed across multiple levels of abstraction:
  - system architecture;
  - hardware and software systems;
  - hardware and software components.
- Often need design flows within design flows.

Hierarchical HW/SW flow



Concurrent engineering

- Large projects use many people from multiple disciplines.
- Work on several tasks at once to reduce design time.
- Feedback between tasks helps improve quality, reduce number of later design problems.

Concurrent engineering techniques

- Cross-functional teams.
- Concurrent product realization.
- Incremental information sharing.
- Integrated product management.
- Supplier involvement.
- Customer focus.

AT&T PBX concurrent engineering

- Benchmark against competitors.
- Identify breakthrough improvements.
- Characterize current process.
- Create new process.
- Verify new process.
- Implement.
- Measure and improve.

Requirements analysis

- Requirements: informal description of what customer wants.
- Specification: precise description of what design team should deliver.
- Requirements phase links customers with designers.

Types of requirements

- Functional: input/output relationships.
- Non-functional:
  - timing;
  - power consumption;
  - manufacturing cost;
  - physical size;
  - time-to-market;
  - reliability.

Good requirements

- Correct.
- Unambiguous.
- Complete.
- Verifiable: is each requirement satisfied in the final system?
- Consistent: requirements do not contradict each other.

Good requirements, cont'd.

- Modifiable: can update requirements easily.
- Traceable:
  - know why each requirement exists;
  - go from source documents to requirements;
  - go from requirement to implementation;
  - back from implementation to requirement.

Setting requirements

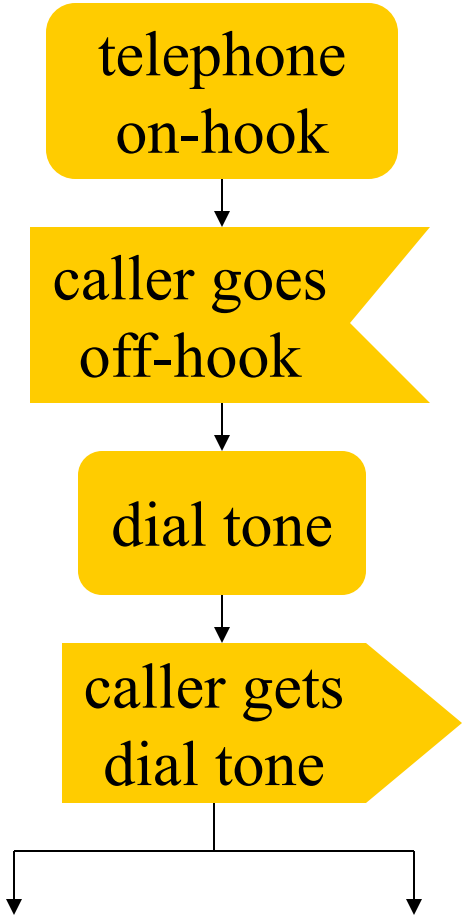
- Customer interviews.
- Comparison with competitors.
- Sales feedback.
- Mock-ups, prototypes.
- Next-bench syndrome (HP): design a product for someone like you.

Specifications

- Capture functional and non-functional properties:
  - verify correctness of spec;
  - compare spec to implementation.
- Many specification styles:
  - control-oriented vs. data-oriented;
  - textual vs. graphical.
- UML is one specification/design language.

SDL

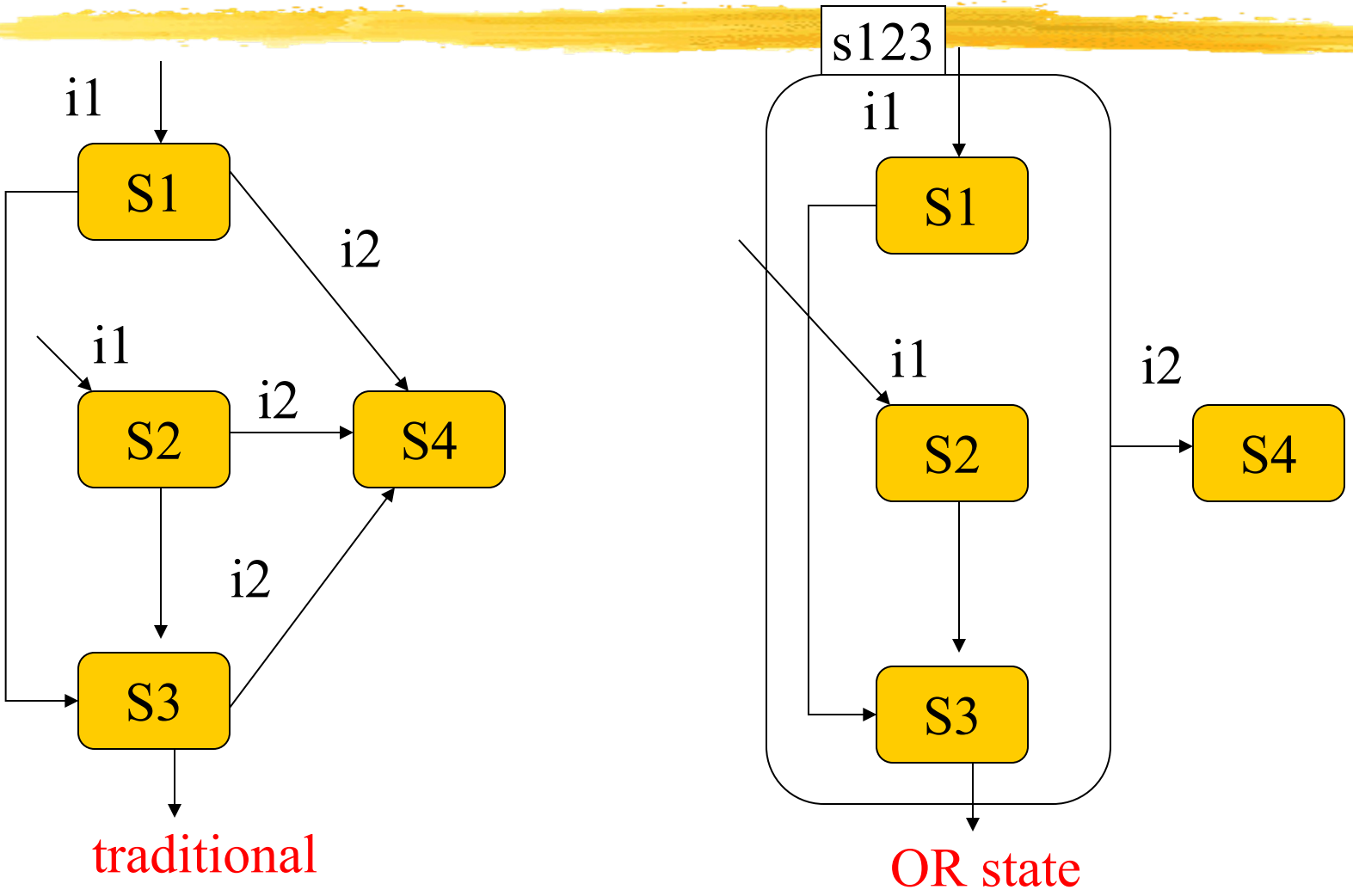
- Used in telecommunications protocol design.
- Event-oriented state machine model.



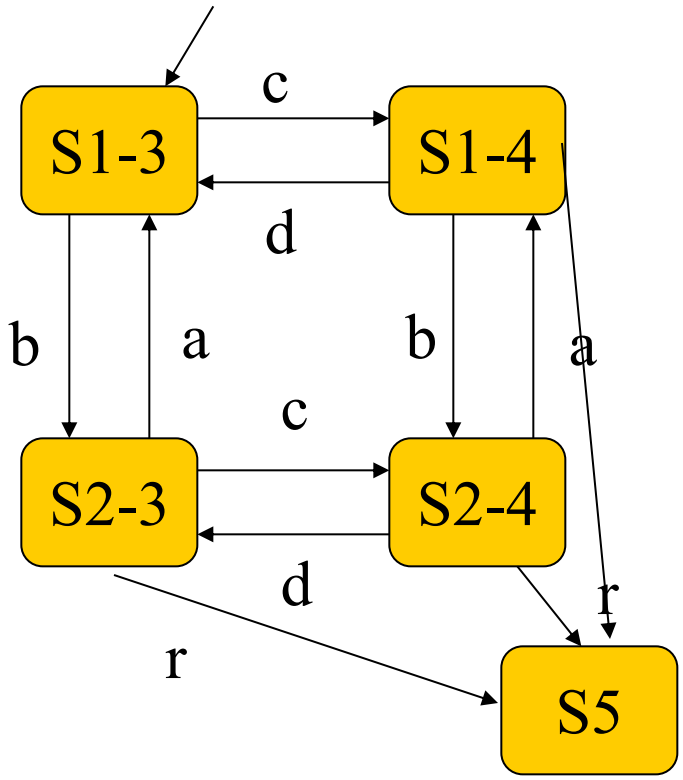
Statecharts

- Ancestor of UML state diagrams.
- Provided composite states:
  - OR states;
  - AND states.
- Composite states reduce the size of the state transition graph.

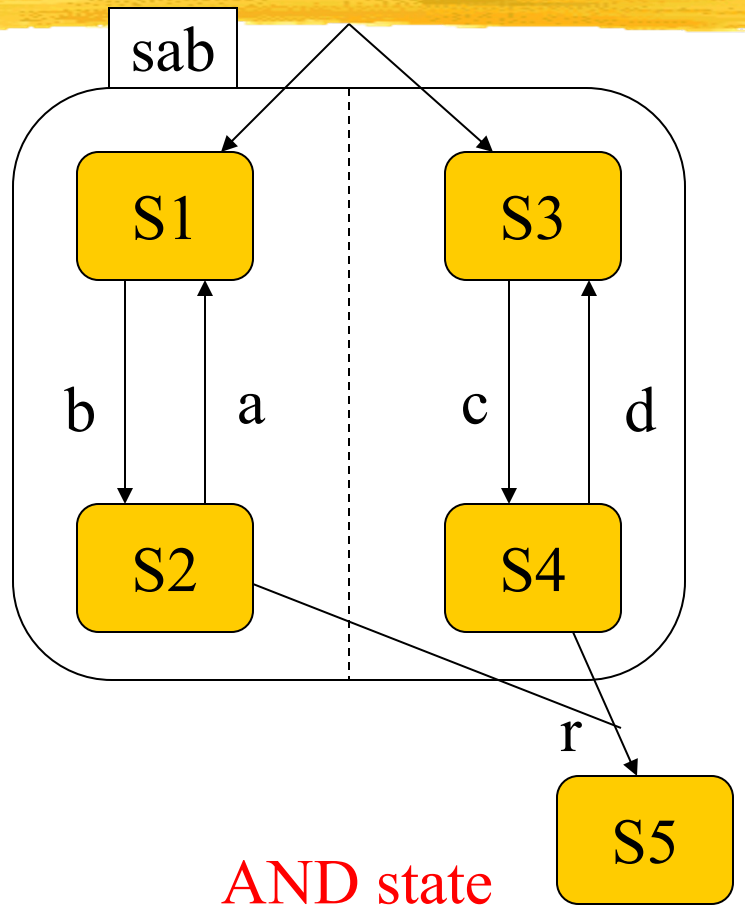
Statechart OR state



Statechart AND state



traditional



AND state

AND-OR tables

- Alternate way of specifying complex conditions:

  cond1 or (cond2 and !cond3)

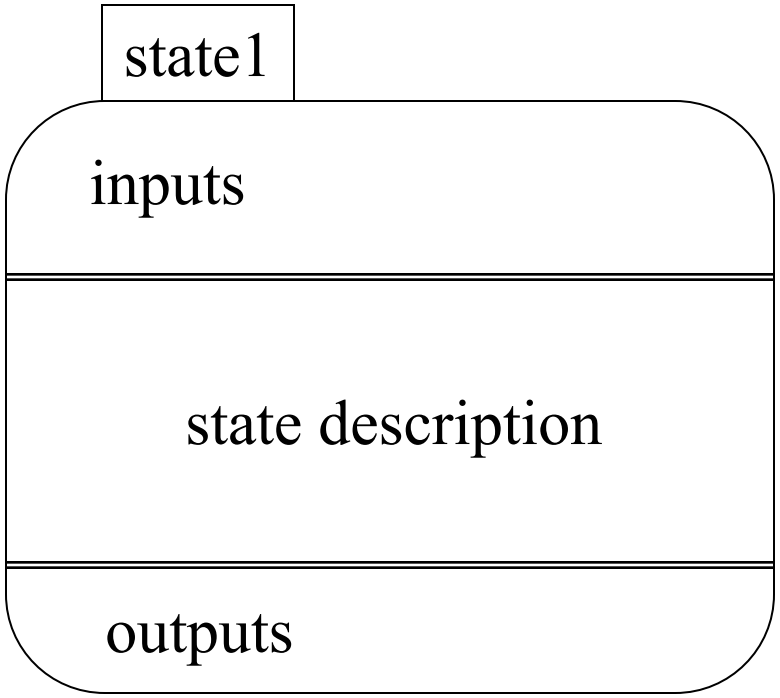
		OR	
	cond1	T	-
AND	cond2	-	T
	cond3	-	F

TCAS II specification

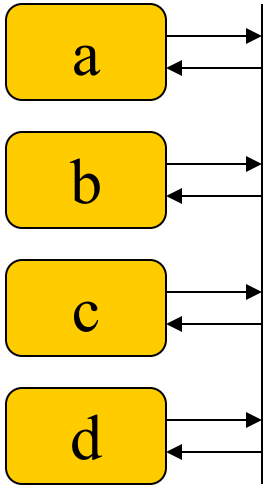
- TCAS II: aircraft collision avoidance system.
- Monitors aircraft and air traffic info.
- Provides audio warnings and directives to avoid collisions.
- Leveson et al used RMSL language to capture the TCAS specification.

RMSL

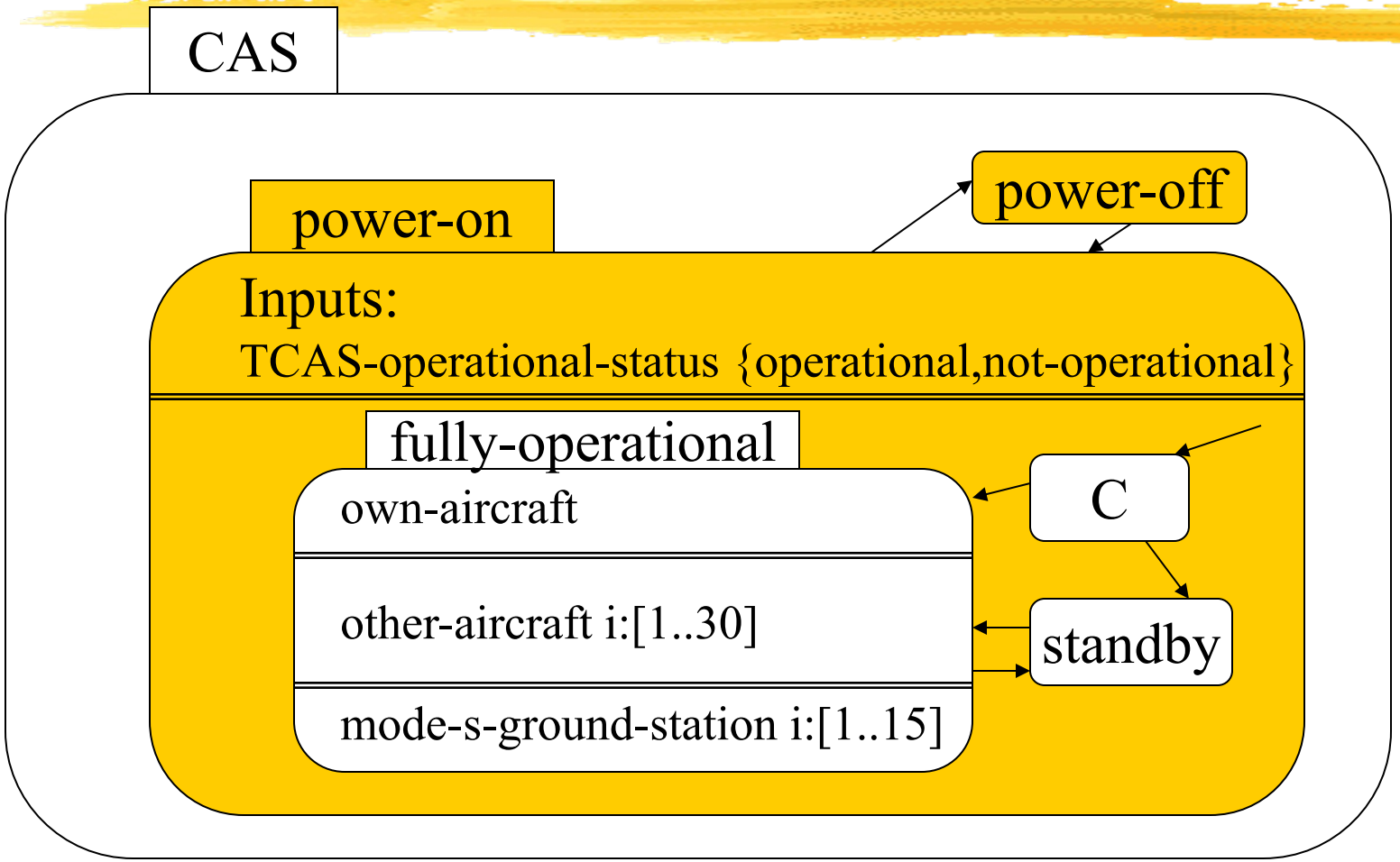
- State description:



- Transition bus for transitions between many states:



TCAS top-level description



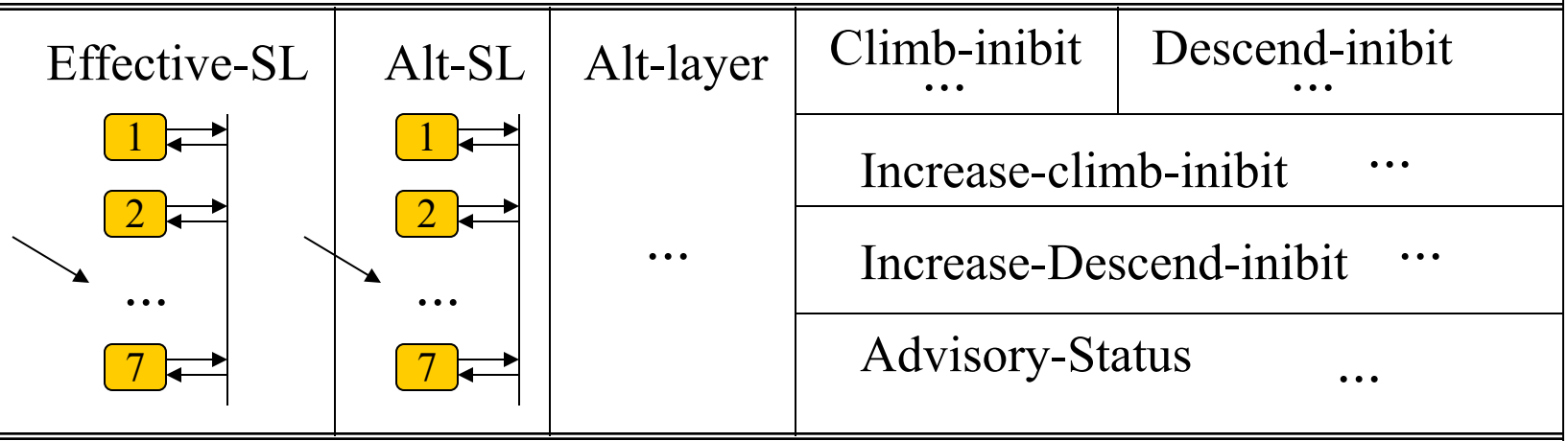
Own-Aircraft AND state

CAS

Inputs:

own-alt-radio: integer standby-discrete-input: {true,false}

own-alt-barometric:integer, etc.



Outputs:

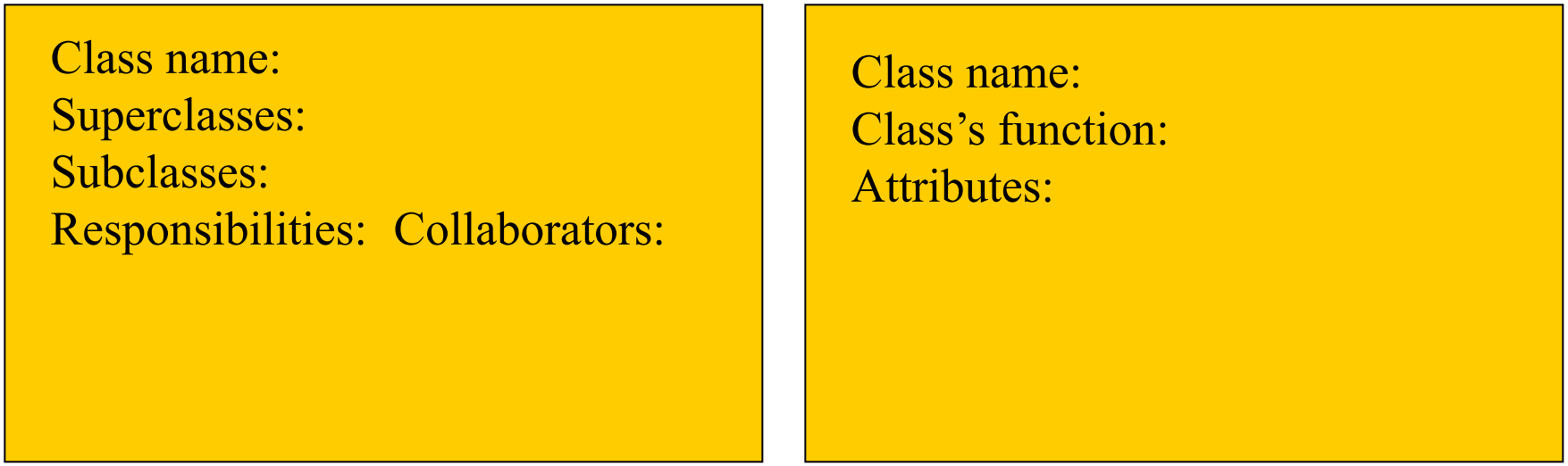
sound-aural-alarm: {true,false} aural-alarm-inhibit: {true, false}

combined-control-out: enumerated, etc.

CRC cards

- Well-known method for analyzing a system and developing an architecture.
- CRC:
  - classes;
  - responsibilities of each class;
  - collaborators are other classes that work with a class.
- Team-oriented methodology.

CRC card format



front back

CRC methodology

- Develop an initial list of classes.
  - Simple description is OK.
  - Team members should discuss their choices.
- Write initial responsibilities/collaborators.
  - Helps to define the classes.
- Create some usage scenarios.
  - Major uses of system and classes.

CRC methodology, cont'd.

- Walk through scenarios.
  - See what works and doesn't work.
- Refine the classes, responsibilities, and collaborators.
- Add class relatoinships:
  - superclass, subclass.

CRC cards for elevator

- Real-world classes:
  - elevator car, passenger, floor control, car control, car sensor.
- Architectural classes: car state, floor control reader, car control reader, car control sender, scheduler.

Elevator responsibilities and collaborators

class	responsibilities	collaborators
Elevator car*	Move up and down	Car control, car sensor, car control sender
Car control*	Transmits car requests	Passenger, floor control reader
Car state	Reads current position of car	Scheduler, car sensor