Development Process, Specification, SysML Diagrams Structure and Behavior

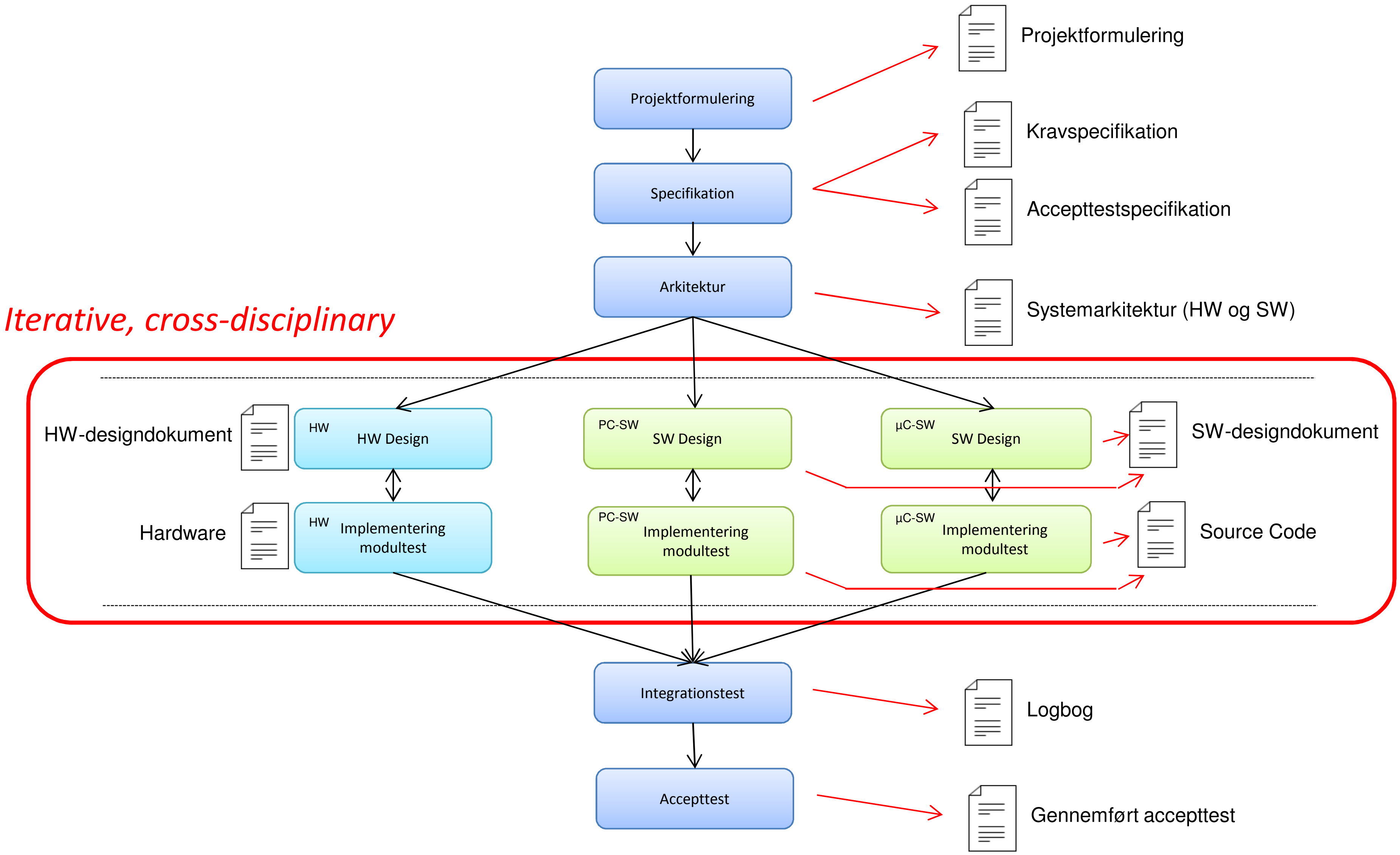
Speedway Course
for
Introduction to Systems Engineering

ISE - Lessons and topics

- ***System Specification, Quality and Process***
  - Development Processes (1)
  - Specification, Use Cases (2)
  - System Test
  - Quality Assurance
- ***SysML Diagrams***
  - SysML structure diagrams (3)
  - SysML behavior diagrams (4)
- ***System Analysis and Design, Architecture and Interfaces***
  - System Domain Analysis (5)
  - System Design and Architecture (6)
  - HW/SW Design
  - Interfaces
- ***Project Management***
  - Project Management
  - Scrum

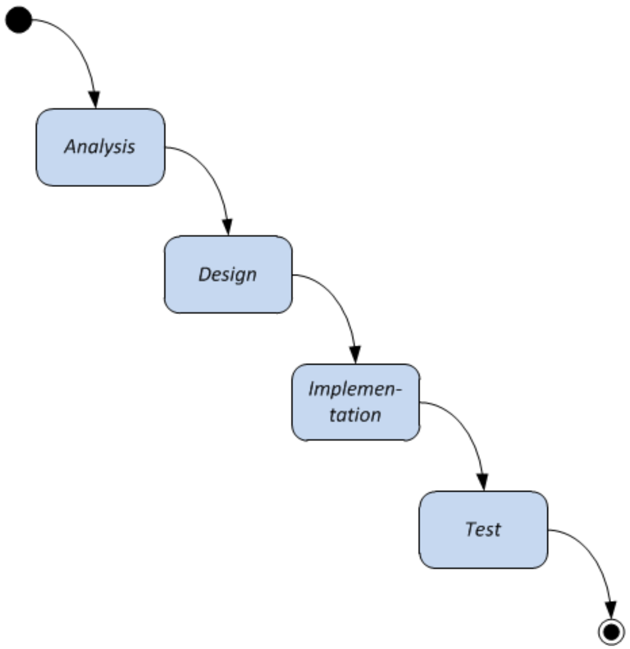
Development Process

The ASE Process

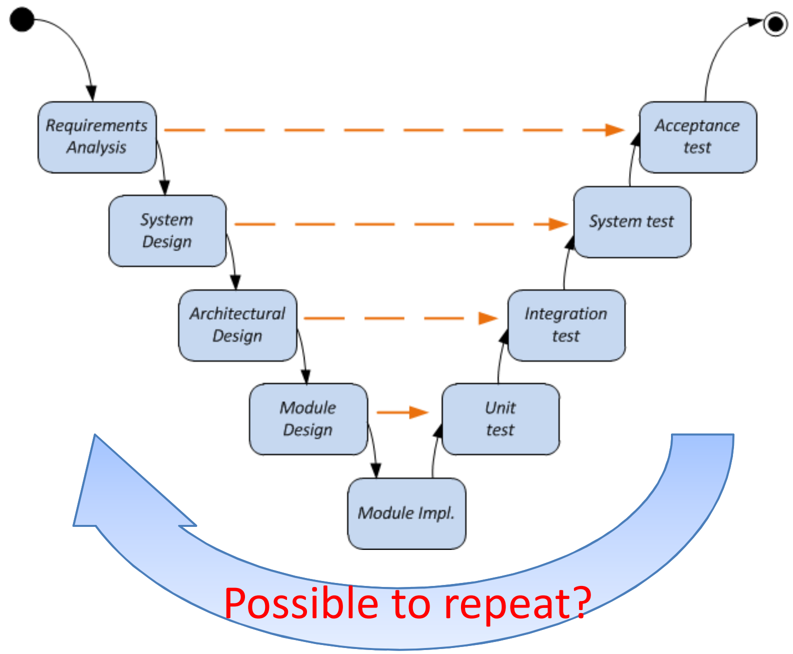


Discussion

- What is the difference?



Improves quality!



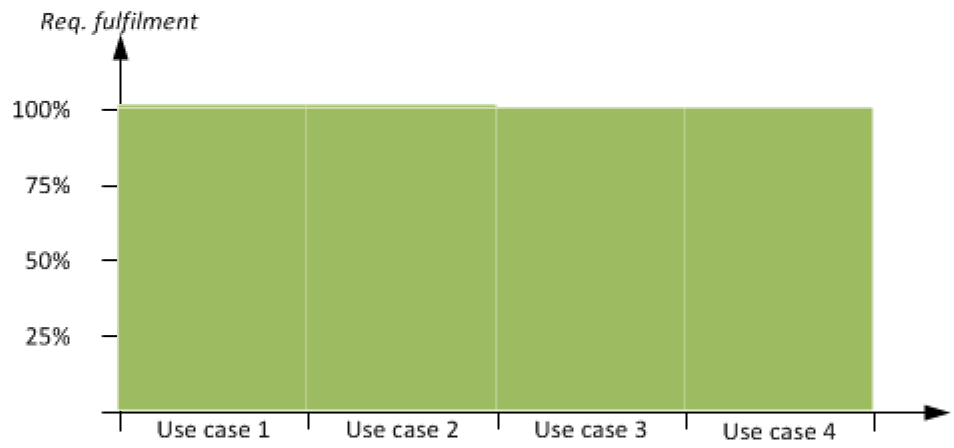
Iterative and incremental development processes

- *Iterative* refers to the repetitive nature of the process
  - An *iteration* is a single repetition of the same sub-process.
  - The sub-process result is a partial working system of *production-quality*

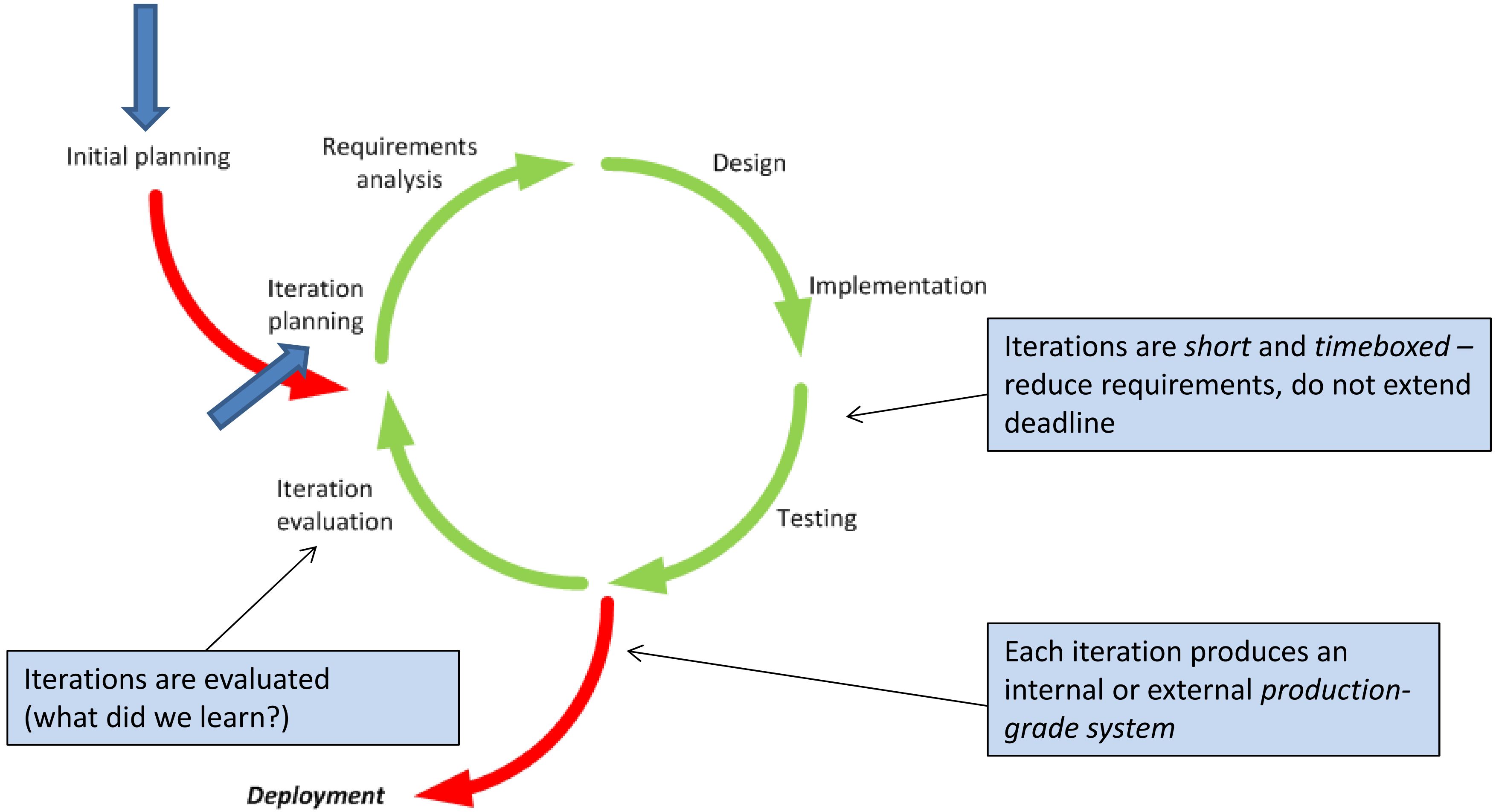
- *Incremental* refers to the *continued expansion* of system capabilities.

Iterative vs. *incremental*

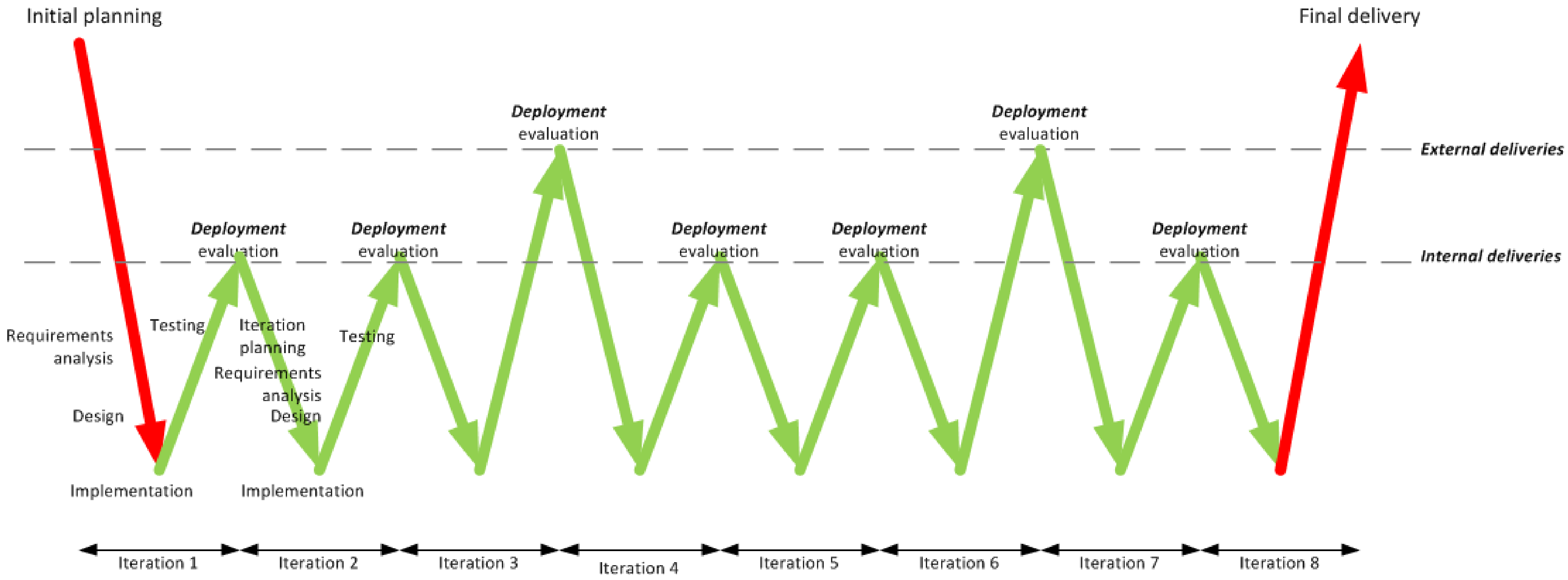
- Iterative *and* incremental



Iterations

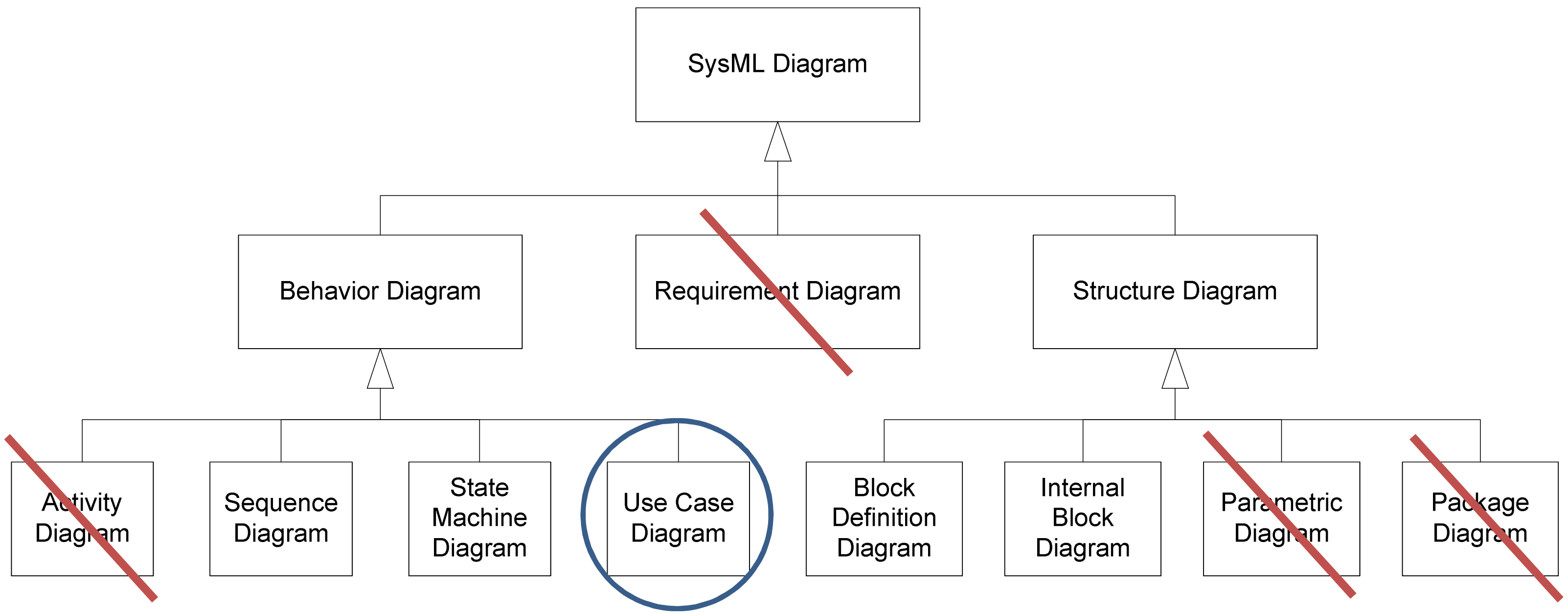

Initial planning
Requirements analysis
Design
Implementation
Iteration planning
Iterations are *short* and *timeboxed* – reduce requirements, do not extend deadline
Iteration evaluation
Testing
Iterations are evaluated (what did we learn?)
Each iteration produces an internal or external *production-grade system*
Deployment

Iterations – another view

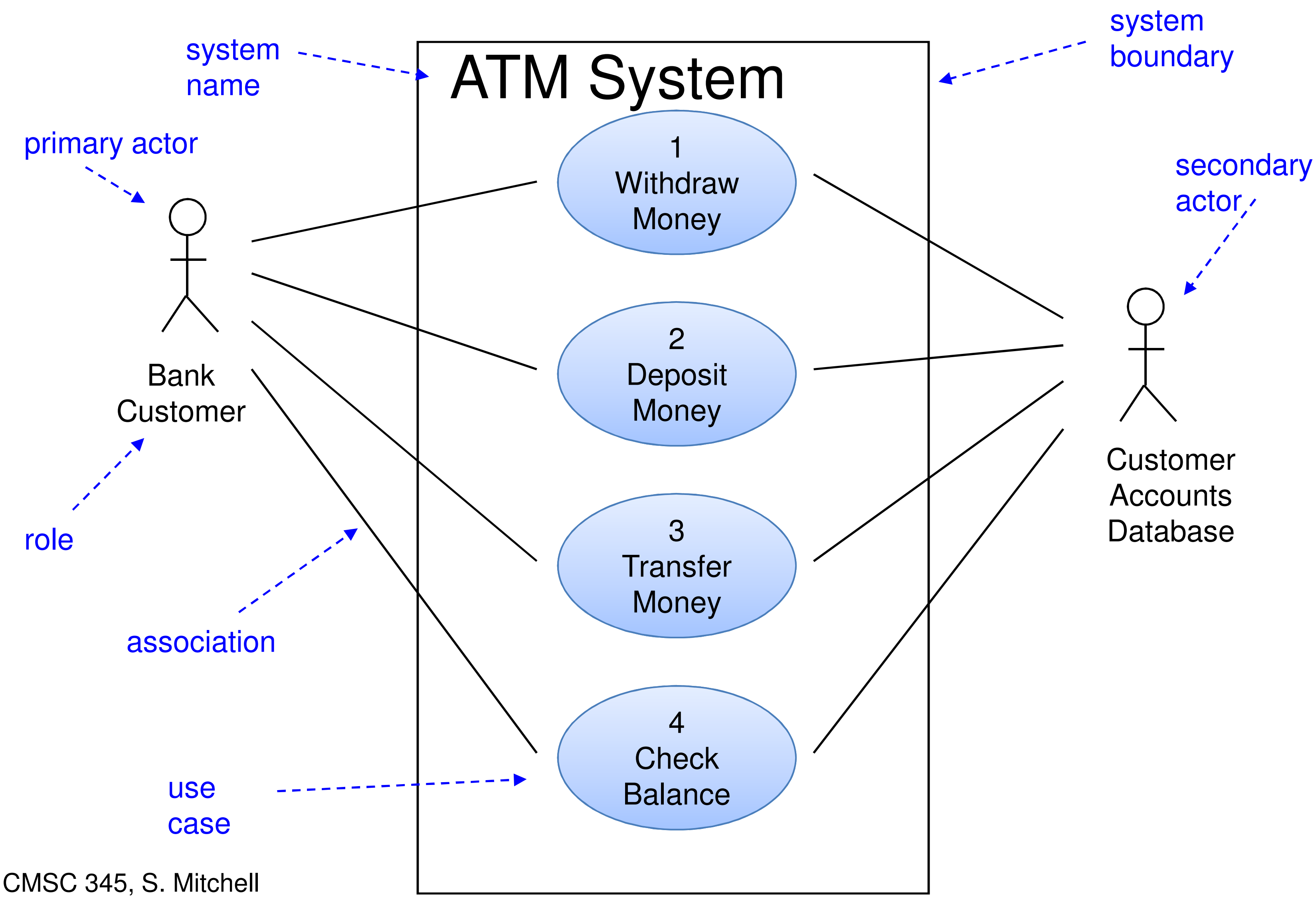


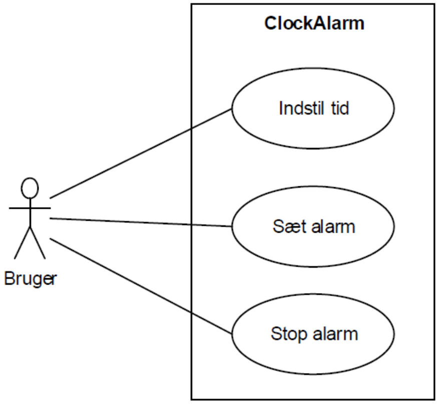
Specification

SysML: Diagram types



Use Case diagrams





Fully-dressed example (Alarm Clock)

Navn:	Sæt alarm
Mål:	Bruger ønsker at sætte alarmtiden.
Initiering:	Bruger trykker på ALARM knappen
Aktører:	Bruger - primær
Samtidige forekomster:	1
Prækondition:	Uret er tændt og operationel
Postkondition:	Alarmen er sat til den ønskede tid

Hovedscenarie:

1. Bruger trykker på ALARM
2. Urets display viser tidligere alarm
   [Extension 1a: Ingen tidligere alarm]
3. Bruger trykker på henholdsvis HOUR og MIN
4. Uret optæller time og minut visningen for alarm
5. Bruger trykker på ALARM for at afslutte indstillingen
6. Uret skifter tilbage til at vise klokken

Udvidelser/undtagelser:

[Extension 1a: Ingen tidligere alarm]

Alarm indstillingen starter ved 00:00.

Quality Demands/Non-functional Requirements

- Qualities or Constraints on the services or functions offered by the system

> Qualities are properties or characteristics of the system that its stakeholders care about and hence will affect their degree of satisfaction with the system.
> *[Defining Non-Functional Requirements,* Malan01]

- Quality demands/NFRs should satisfy two attributes
  - **Must be verifiable (measurable metrics)**
  - **Should be objective**

Types of requirements

- FURPS+ (Robert Grady, Hewlett-Packard)

  Functionality

  Usability

  Reliability

  Performance

  Supportability

  \+ (Design and Physical constraints, Interfaces, Legal, Test, Reuse, Economic constraints, Aesthetics, Comprehensibility, Technology tradeoffs)

- Others are:
  McCall, Boehm, Dromey

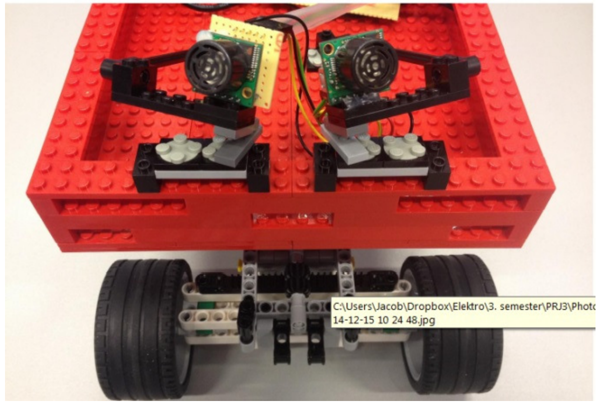
Textual Requirement Specifications

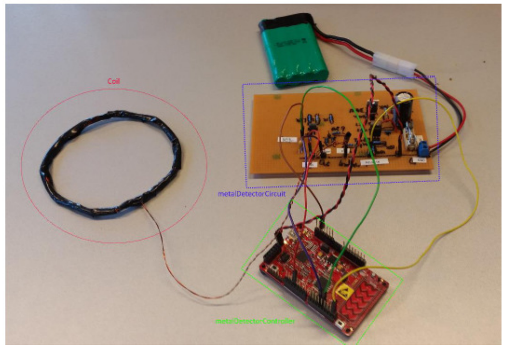
- Defined in word processor
  - Word, OpenOffice, NeoOffice, IWork, etc.
- Textual + diagrams and illustrations
  - Use Cases (Mainly For Functional Requirements)
- MoSCoW Method (prioritisation technique)

  M - MUST (skal) have this.
  S - SHOULD (bør) have this if possible,
  C - COULD (kunne) have this if it does not affect anything.
  W - WON'T have this time, but WOULD like in the future

Example - Treasure Robot (3. Semester project)

- Driving robot
- Obstacle sensor
- Metal detector
- GPS





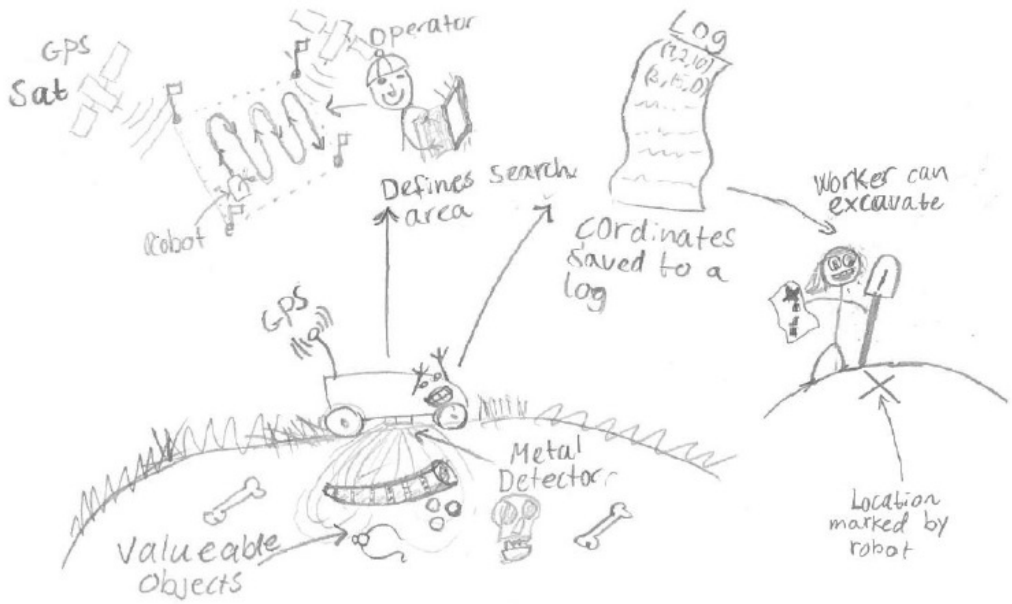


Figure 1 TreasureBot rich picture

Treasure Robot - Non-functional Requirements

3.1 Battery

3.1.1. Battery life **should** be minimum of 20 min when the car is in running mode
3.1.2. Battery life **should** be minimum of 1 hours when the car is idle

3.2 Robot

3.2.1. The robot **must** not exceed the dimension of 40 cm long, 25 cm wide and 15 cm tall
3.2.2. The robot **must** have enough storage capacity to be able to drive and record data for 20 min.
3.2.3. Should be able to save GPS-location every 5 sec. +/- ½ sec.

3.3 Metal detection

3.3.1. Must be able to detect metal to a depth of min. 5 cm from sensor, when driving on dirt or grass
3.3.2. Must be able to detect metal to a depth of min. 3 cm from sensor, when driving on gravel

3.4 Obstacle sensor

3.4.1. Must be able to detect a black box with dimensions 10x10x10 cm, from a distance of 1 m

3.5 GPS

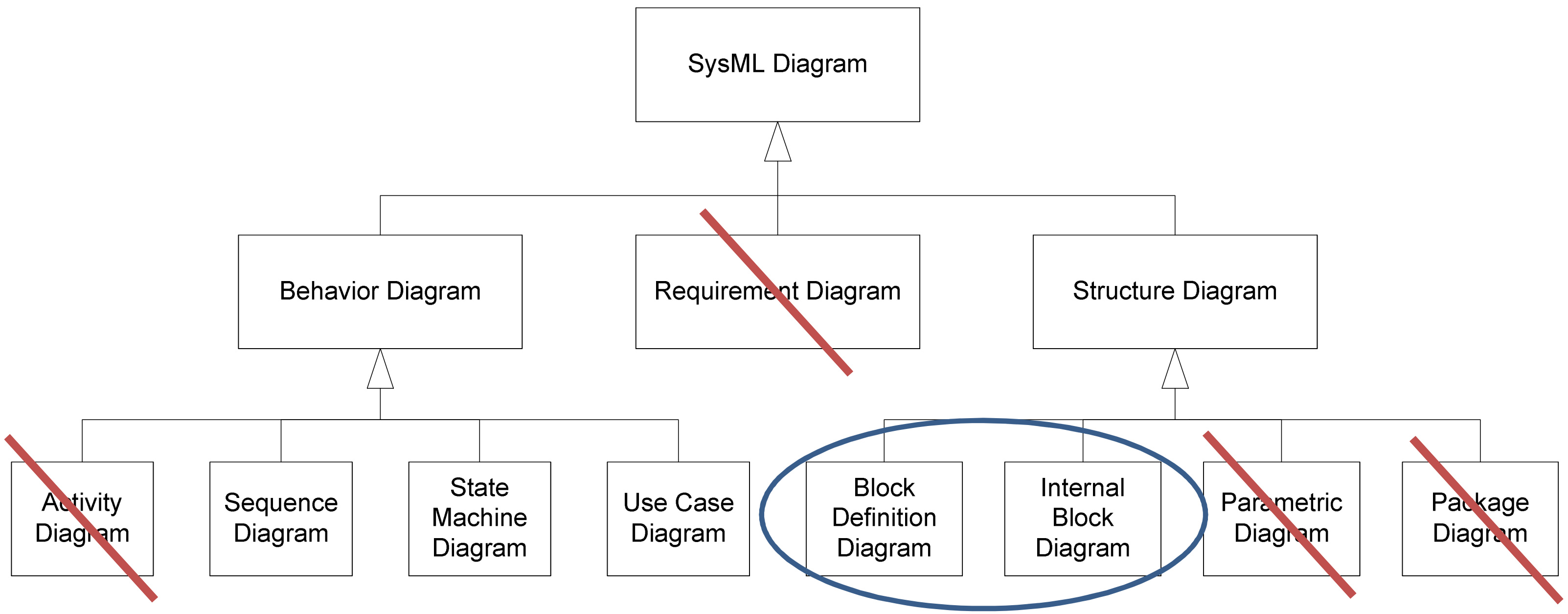
3.3.1. Should have an accuracy of minimum 3 m radius on a clear day
3.3.2. Should have an accuracy of minimum 5 m radius on a cloudy day

SysML Diagrams
Structure and Behavior

System design principles

- The system design principles include:
  - *Decomposition*
  - *Low coupling (kobling/binding)*
  - High *cohesion (samhørighed)*
  - Use *abstractions*
  - *Re-use* existing design solutions
  - Ensure *testability*

SysML: Diagram types

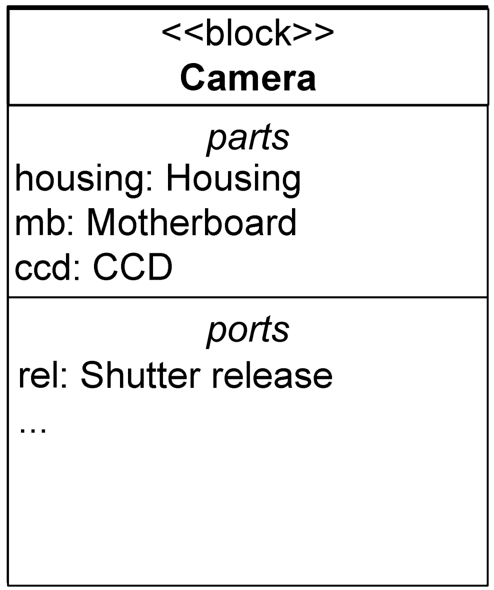


Introduction Structure

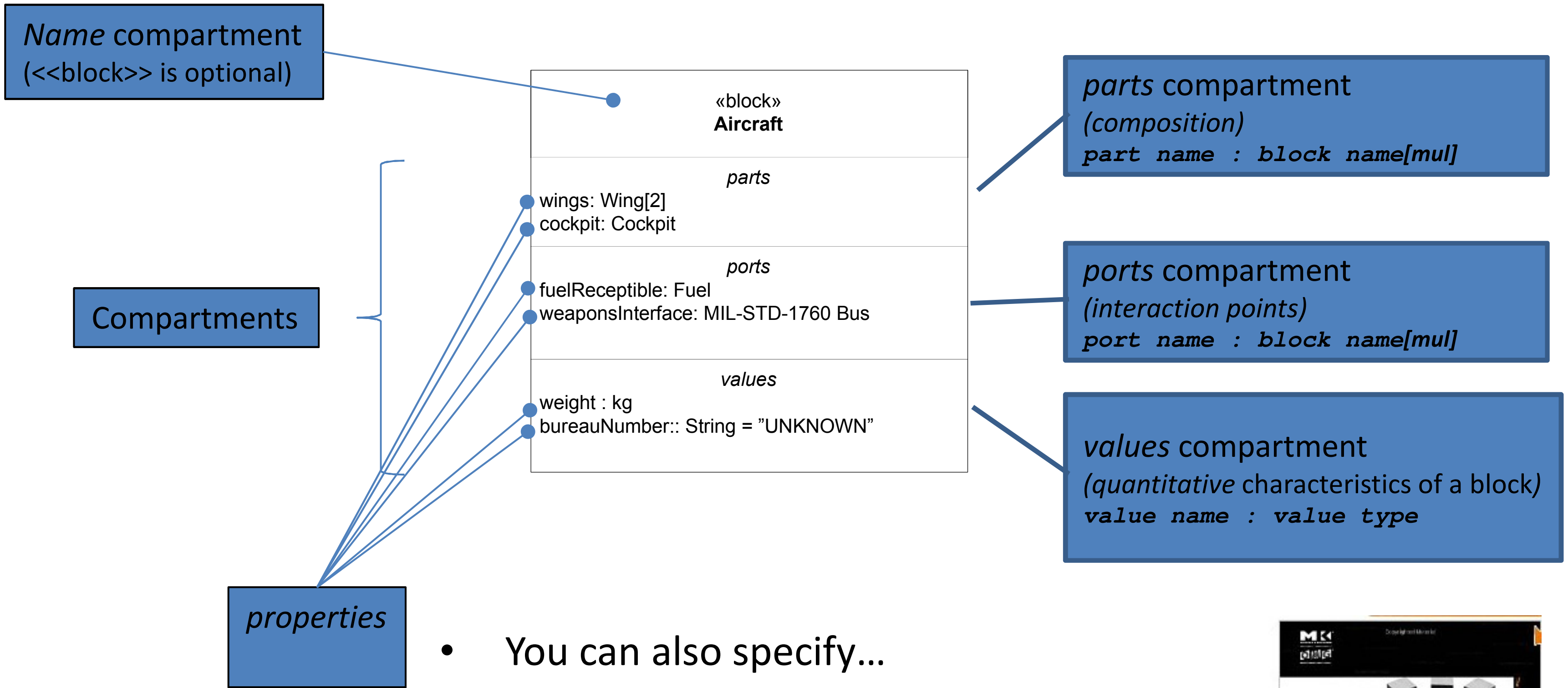
- There are 4 different types of structural diagams:
  - Block Definition Diagram (bdd) – Structural system elements called *blocks* and their composition
  - Internal Block Diagram (ibd) – Interconnection and interfaces between the *parts* of a block
  - Parametric diagram (par) – Constraints on property values
  - Package diagram (pkg) – The organization of a model into *packages* that contain model elements

Blocks

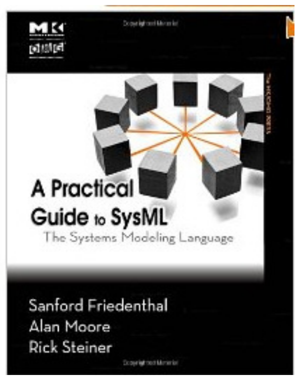
- The block is drawn as a *rectangle* on a diagram canvas
- The block may be divided into *compartments*
- The top compartment always contains the block's *name*
  - *Name* is mandatory
  - `<<block>>` is optional
- Other compartments may be used to represent other block features
  - Parts, operations, ports, …
- Each compartment contains *properties*



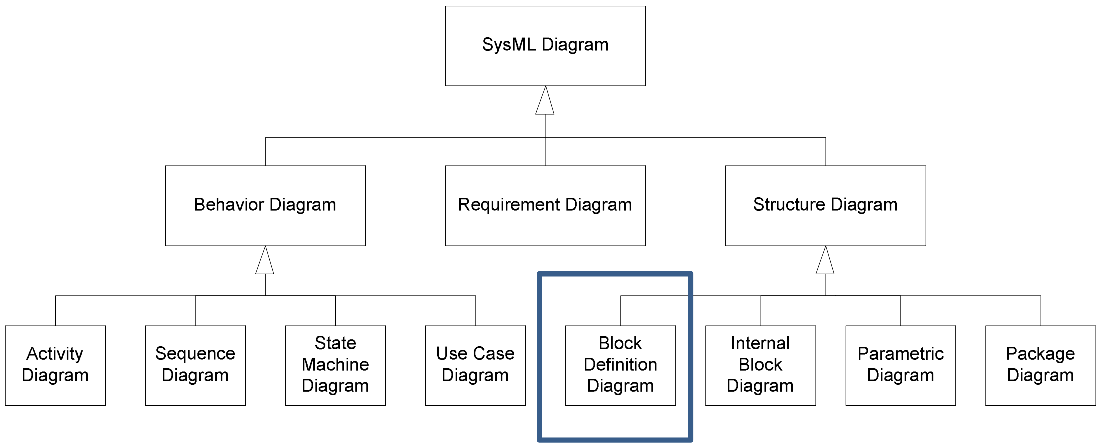
Blocks – the works



- You can also specify…
  - *references* (weaker connections)
  - *value types* and their *units* and *dimensions*
  - *read-only properties*
  - *initial* property values, their *distribution*
  - …



SysML
Block Definition Diagrams



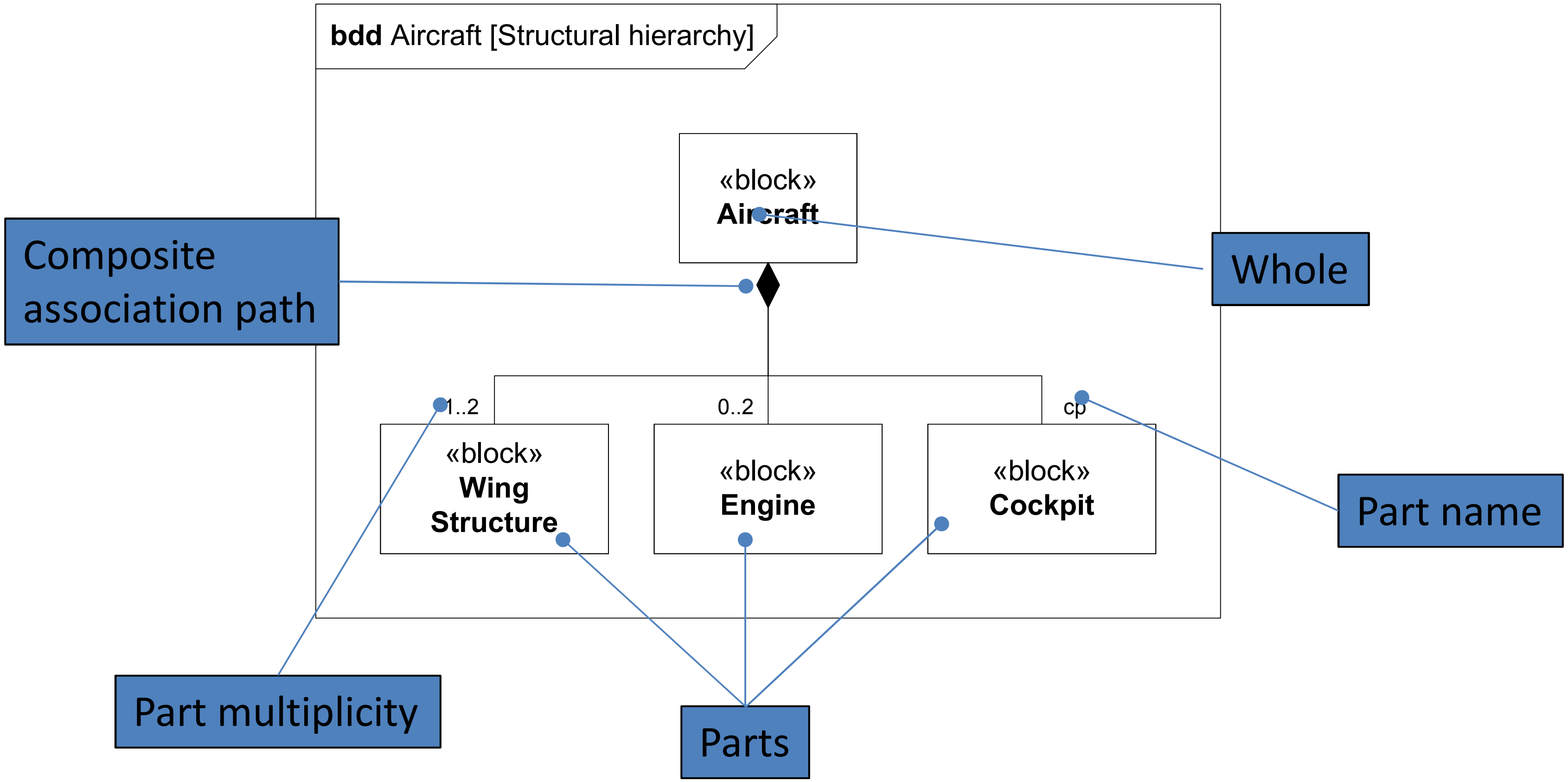
SysML: Block definition diagram

- A *Block Definition Diagram (BDD)* is used to define *blocks* and their relationship other blocks (their *composition*)
- A BDD may be used to define any kind of structure
  - Logical, physical, etc.
- BDDs are also used to define other relationships between blocks, e.g. allocation of functions to physical entities

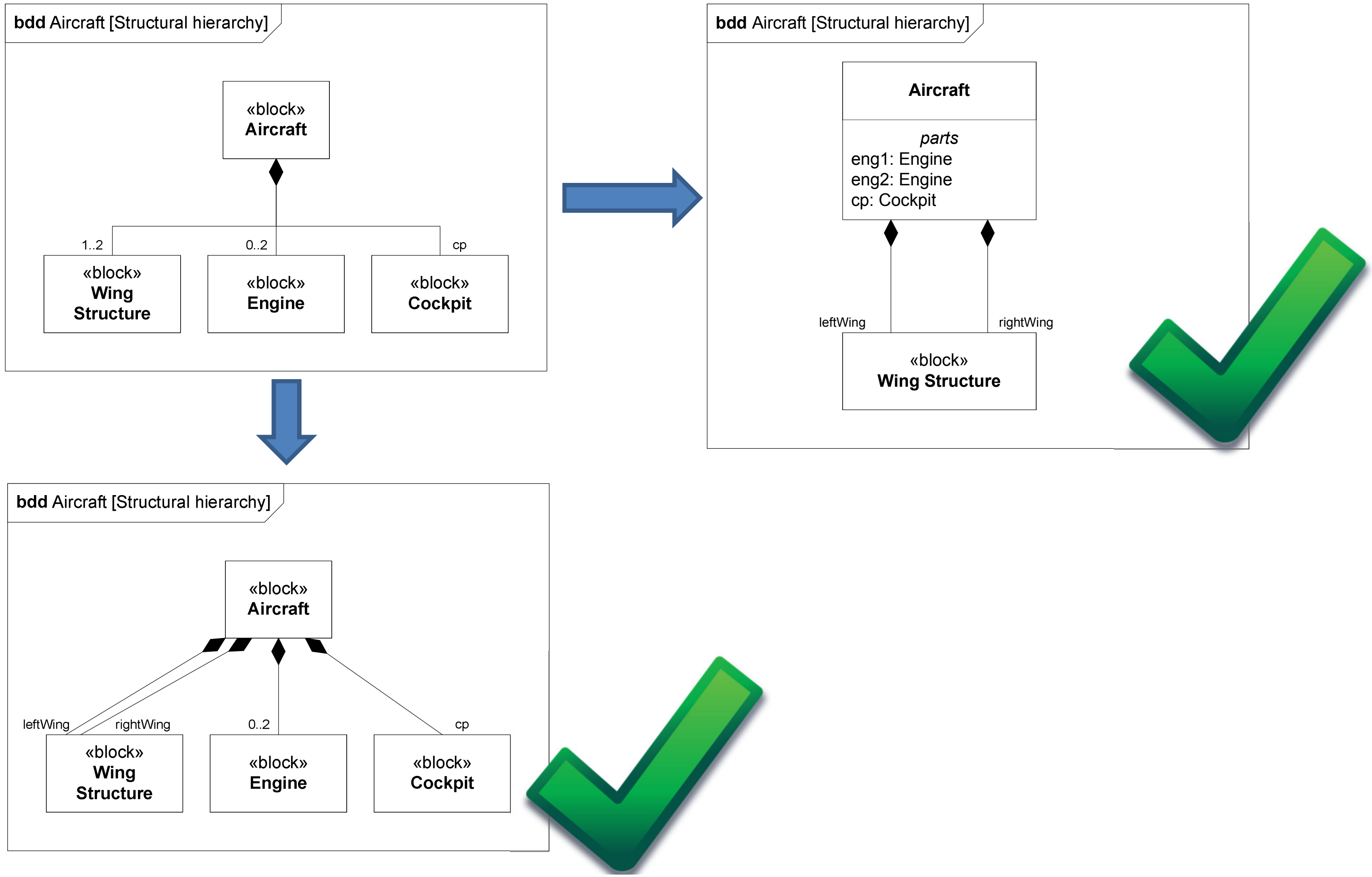
bdd: Composition relationships

The most common kind of relationship is *composition*:

- *"Consists-of"* or *"whole-part"* relationship, e.g. "an `Aircraft` consists-of 1-2 `wings`, 0-2 `engines` and 1 `cockpit`"

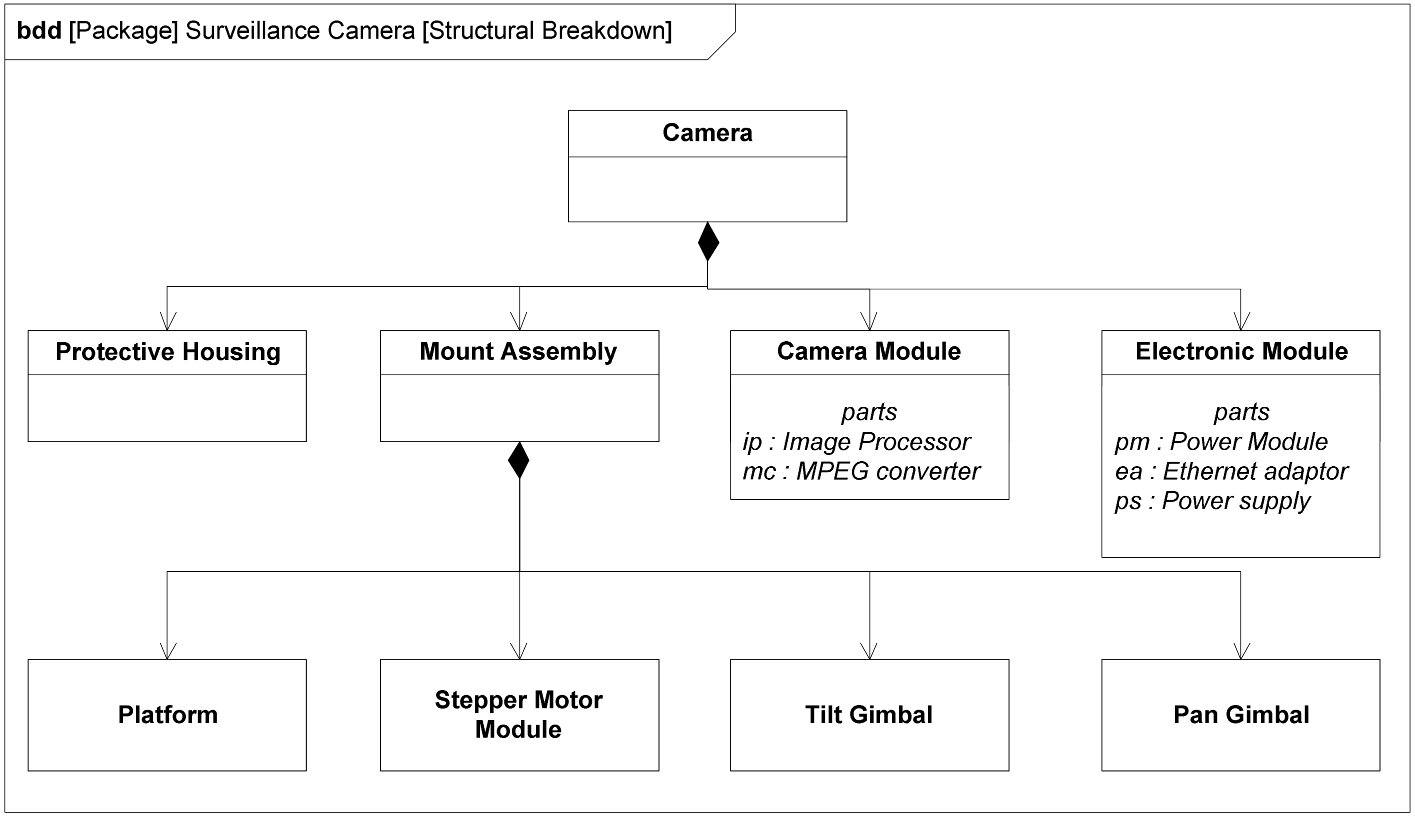


bdd: Variants



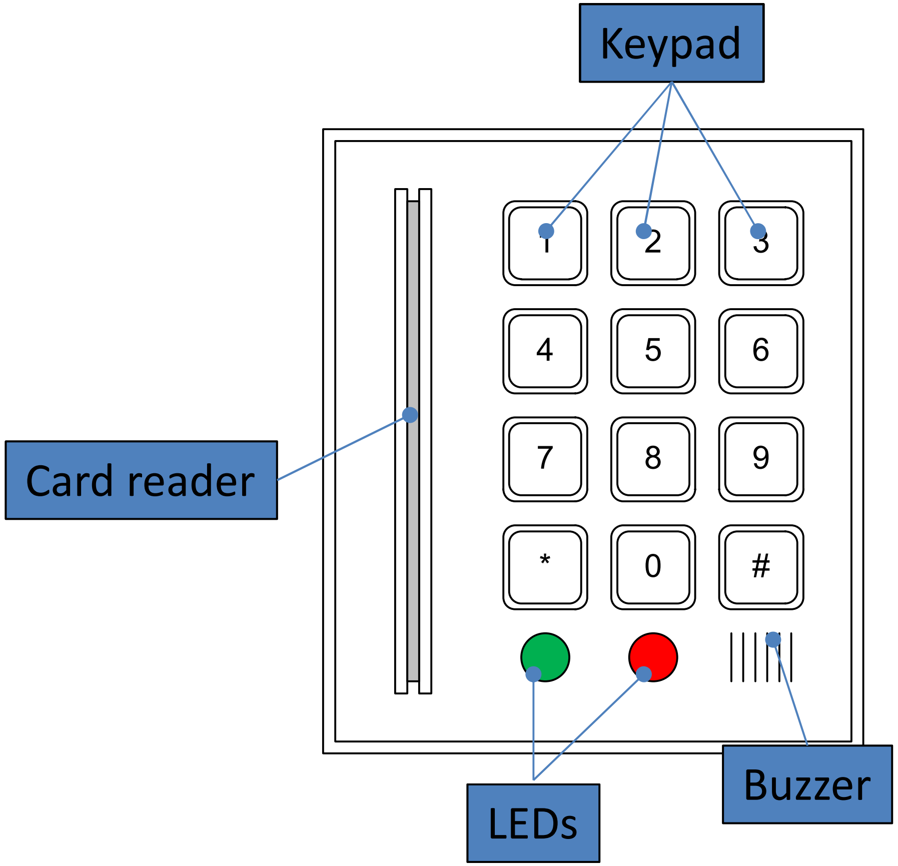
bdd: Deeper hierarchy

How would you read this diagram? *"A camera consists of…"*

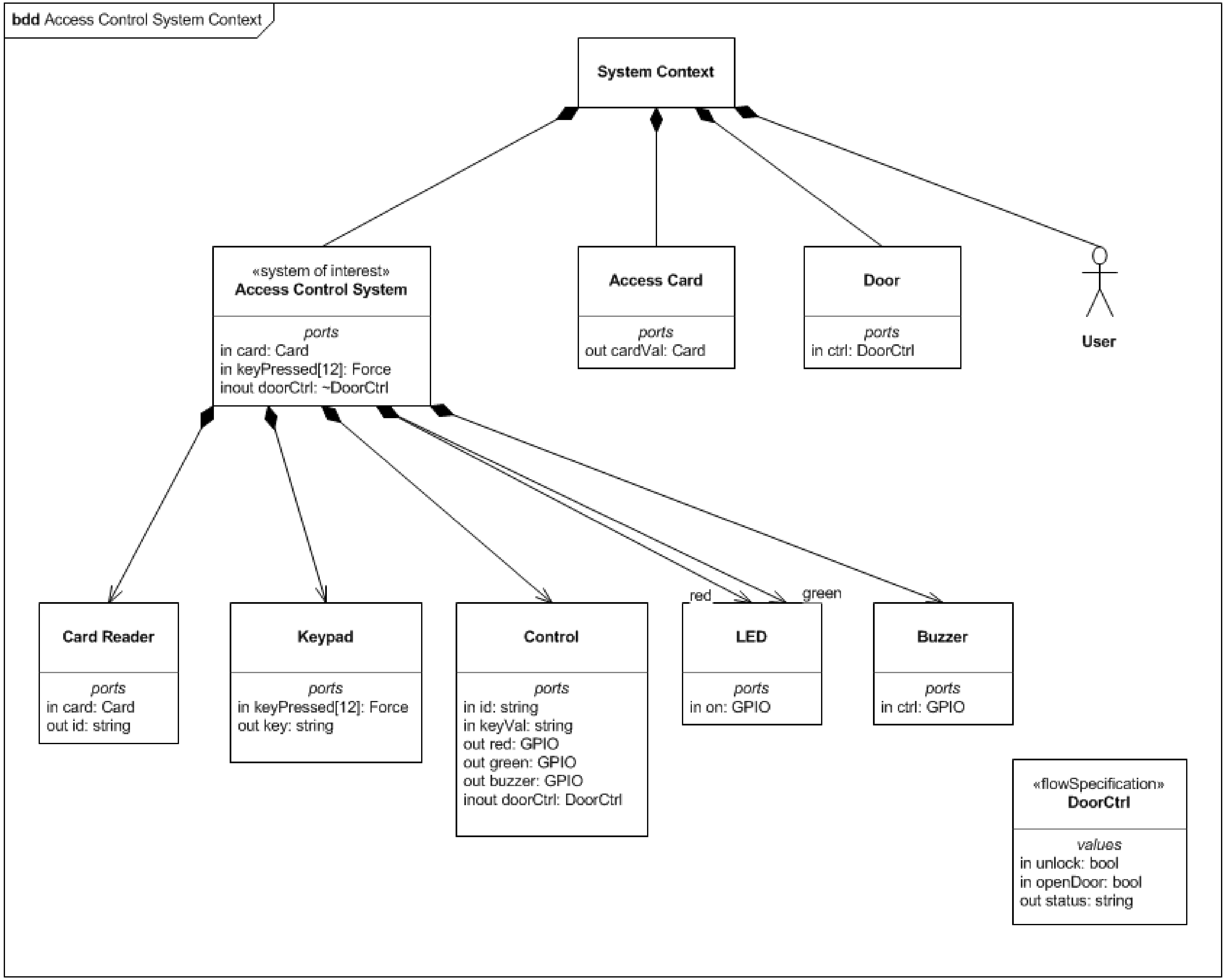


Example

- Create a bdd for an access control system



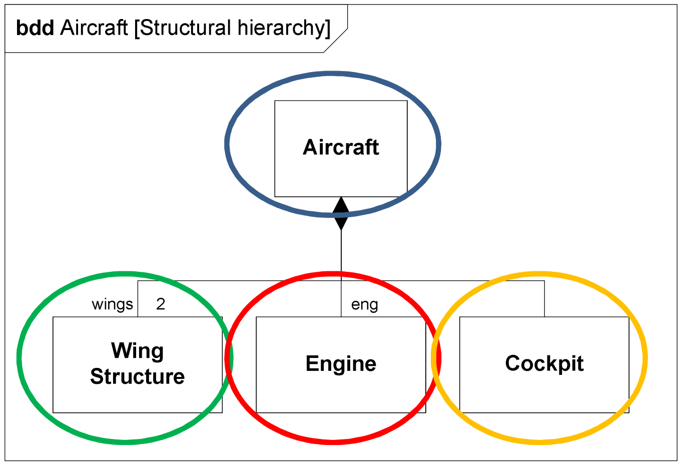
Example solution

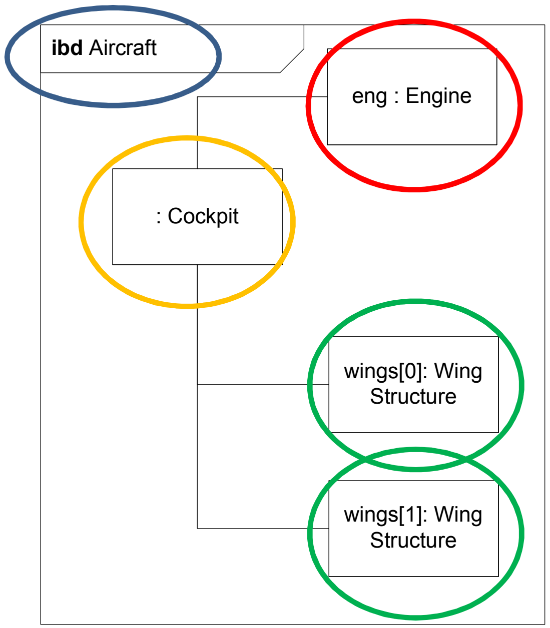


SysML: Internal Block Diagram

- An *Internal Block Diagram* (*ibd*) is used to define
  - the *interconnection* and *interfaces* of the parts of a block, and
  - the *information flow* between parts

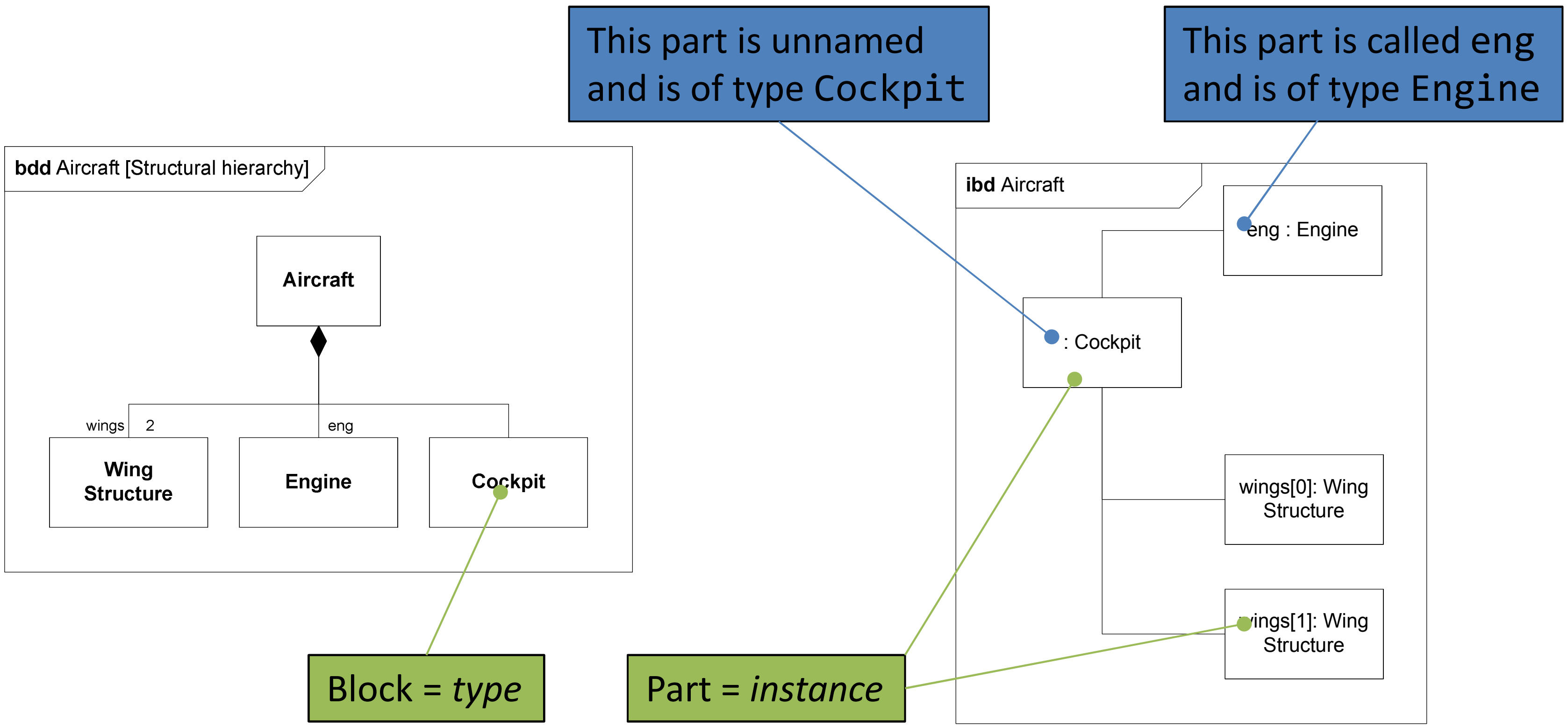
- An ibd always relates to a block on a bdd. It shows the internal connections of the block's constituents





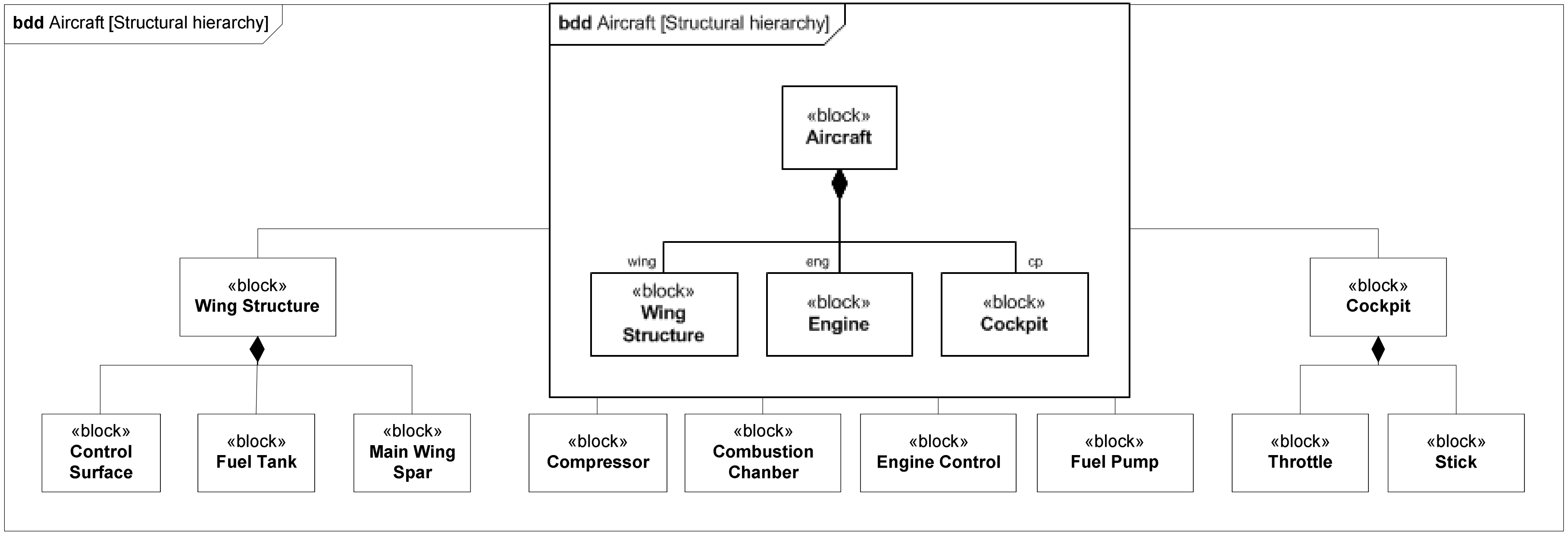
SysML: Blocks and parts

- A block is a *type definition* – there can be only one block with a given name
- A part is an *instance* of a block – there can be many instances of the same block

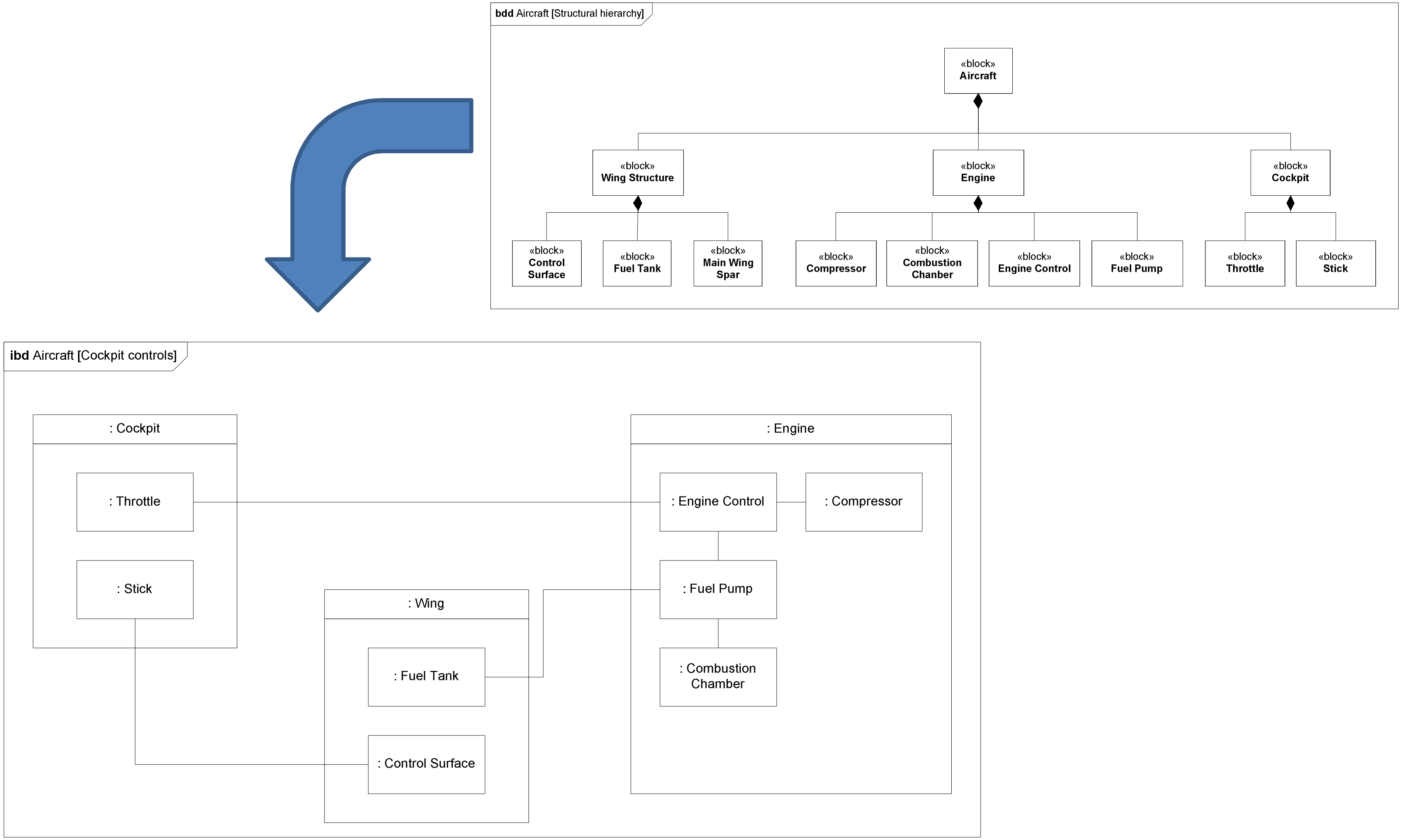


ibd: Aircraft - deep structure

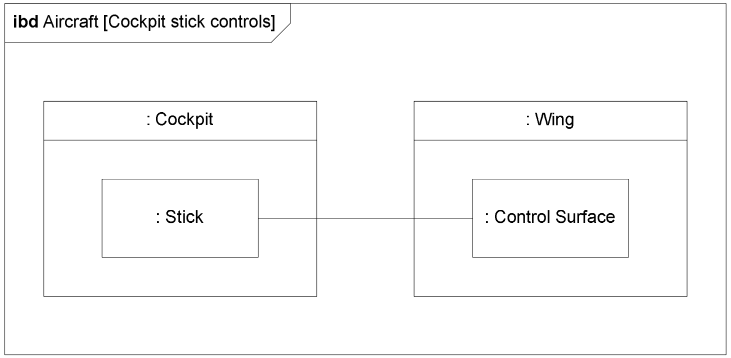
- Deep structure on a bdd can be shown in an ibd:

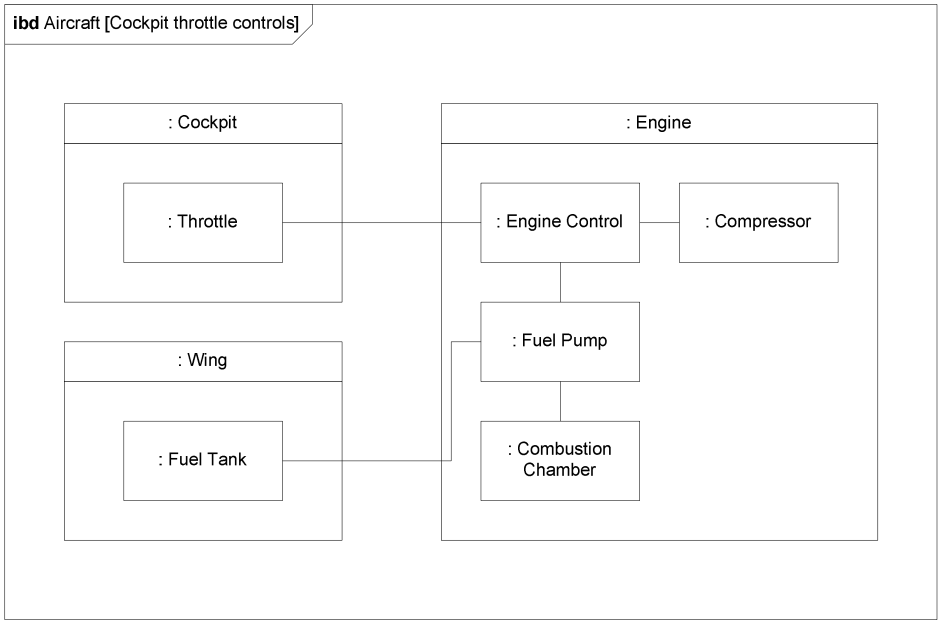


ibd: Aircraft - deep structure



ibd: Aircraft – better deep structure





Modeling interfaces

- We would like to express more about the *connection* between parts on the ibd
  - This would help us to define the *interface* of the parts

- To do this, we must define *items*, *item flows* and *ports*!

Items and item flows

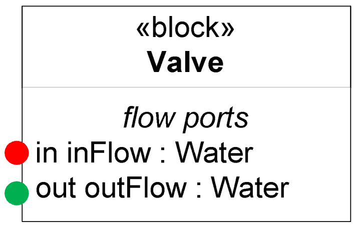
- An *item* describes an entity that flows through a system (blocks, value types or signals*)*
  - Physical flow, information flow, energy, …
  - Simple or complex

- An *item flow* is used to describe a flow of items (!) on a connector between two blocks on an ibd
  - Item flow = item *type* + flow *direction*

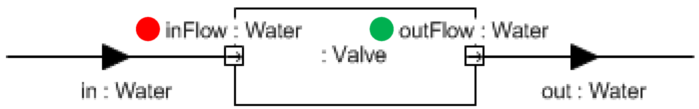
Ports

- A *port* is an interaction point on the boundary of a block
  – Ports are where the items flow into / out of
  – One block can have many ports

- Ports are *defined* on the blocks on a bdd and used to connect *parts* on ibds



bdd

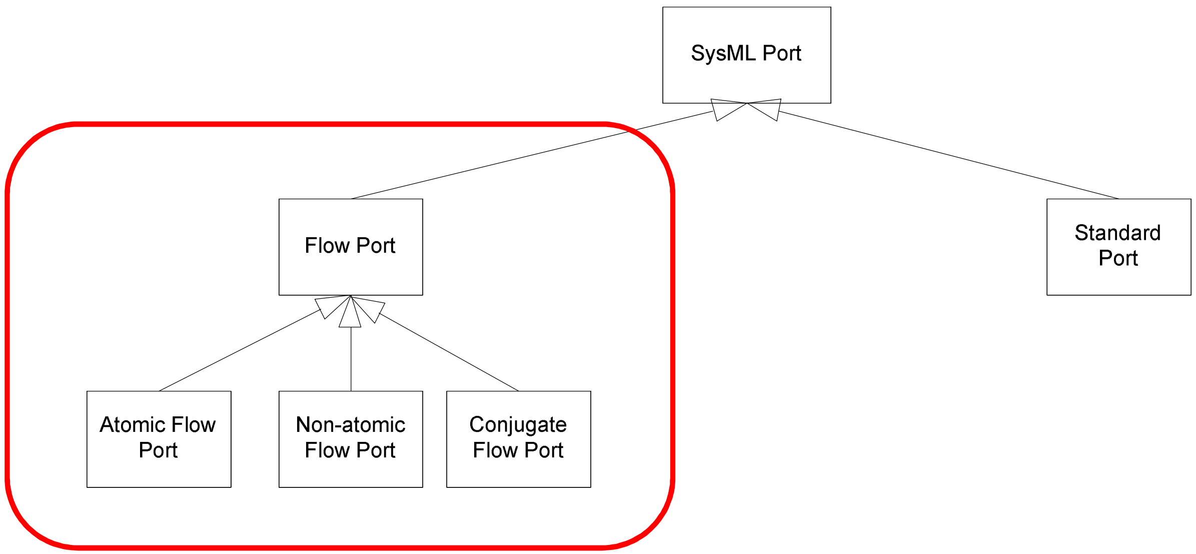


ibd

Ports

- Ports come in different flavours, each with different meaning and use

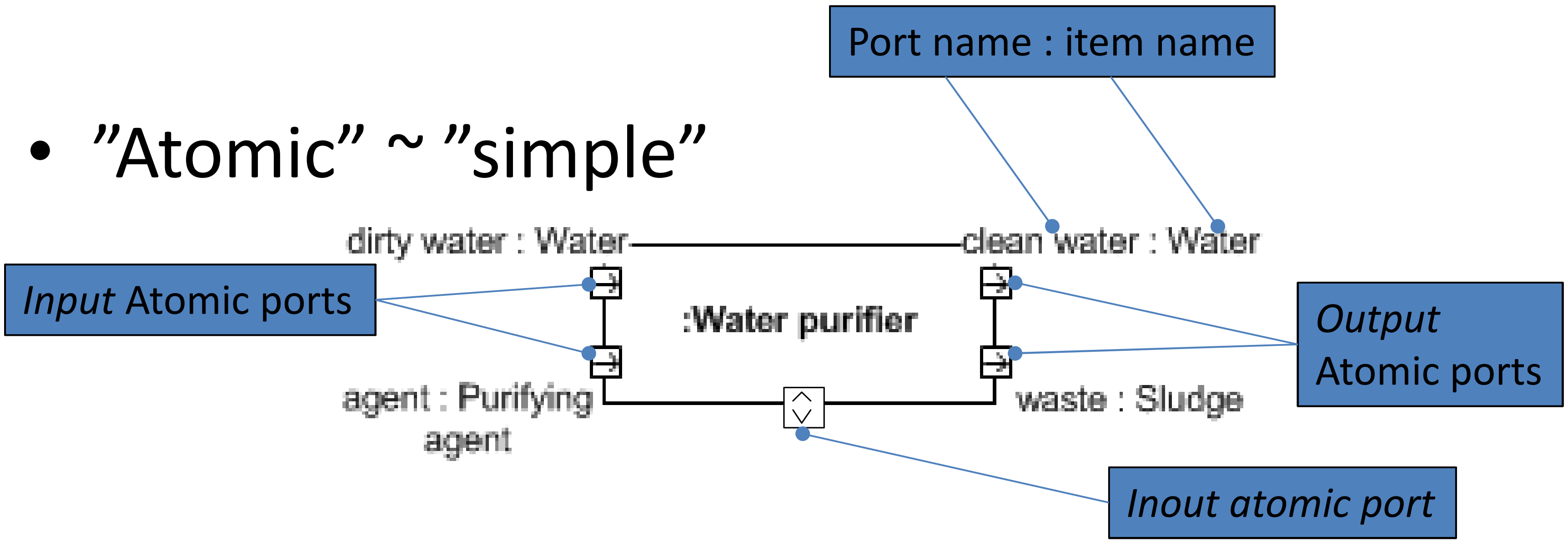
- We will concentrate on *flow* ports



Atomic flow ports

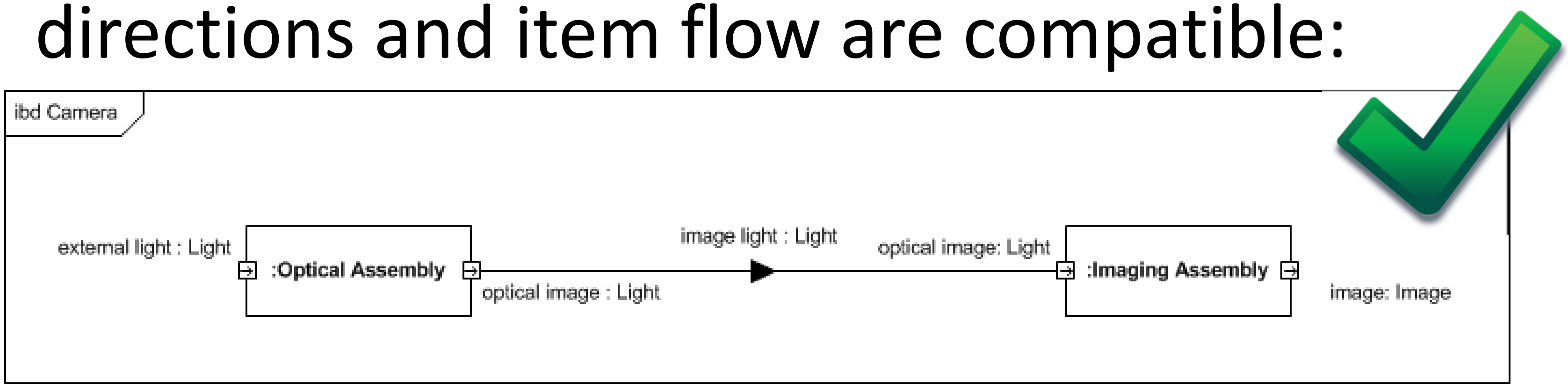
- *Atomic flow ports* are used to describe flows of a single, simple type of item flow to/from a block
  - – Directions: In, out or inout

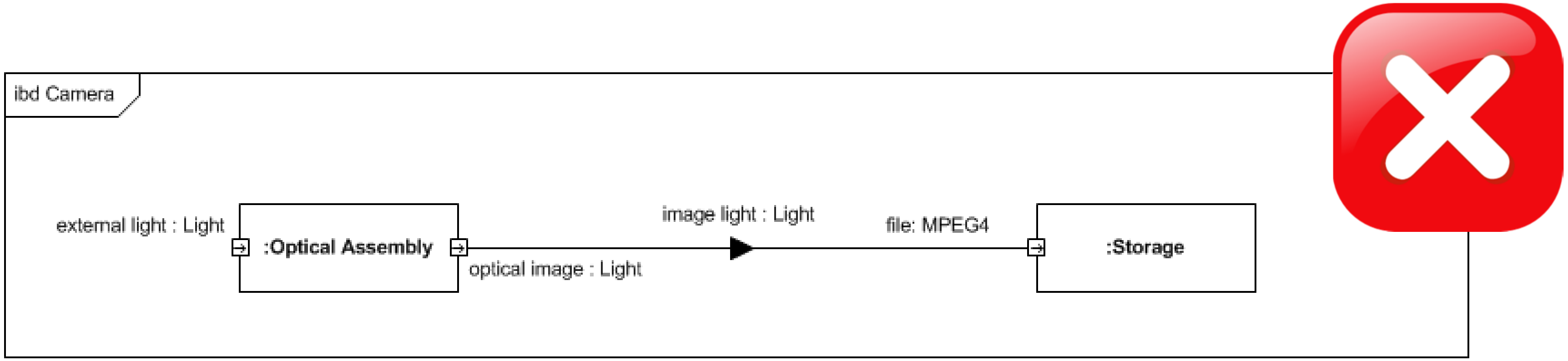
- ”Atomic” ~ ”simple”



Atomic flow ports

- Atomic flow ports can be connected only if directions and item flow are compatible:

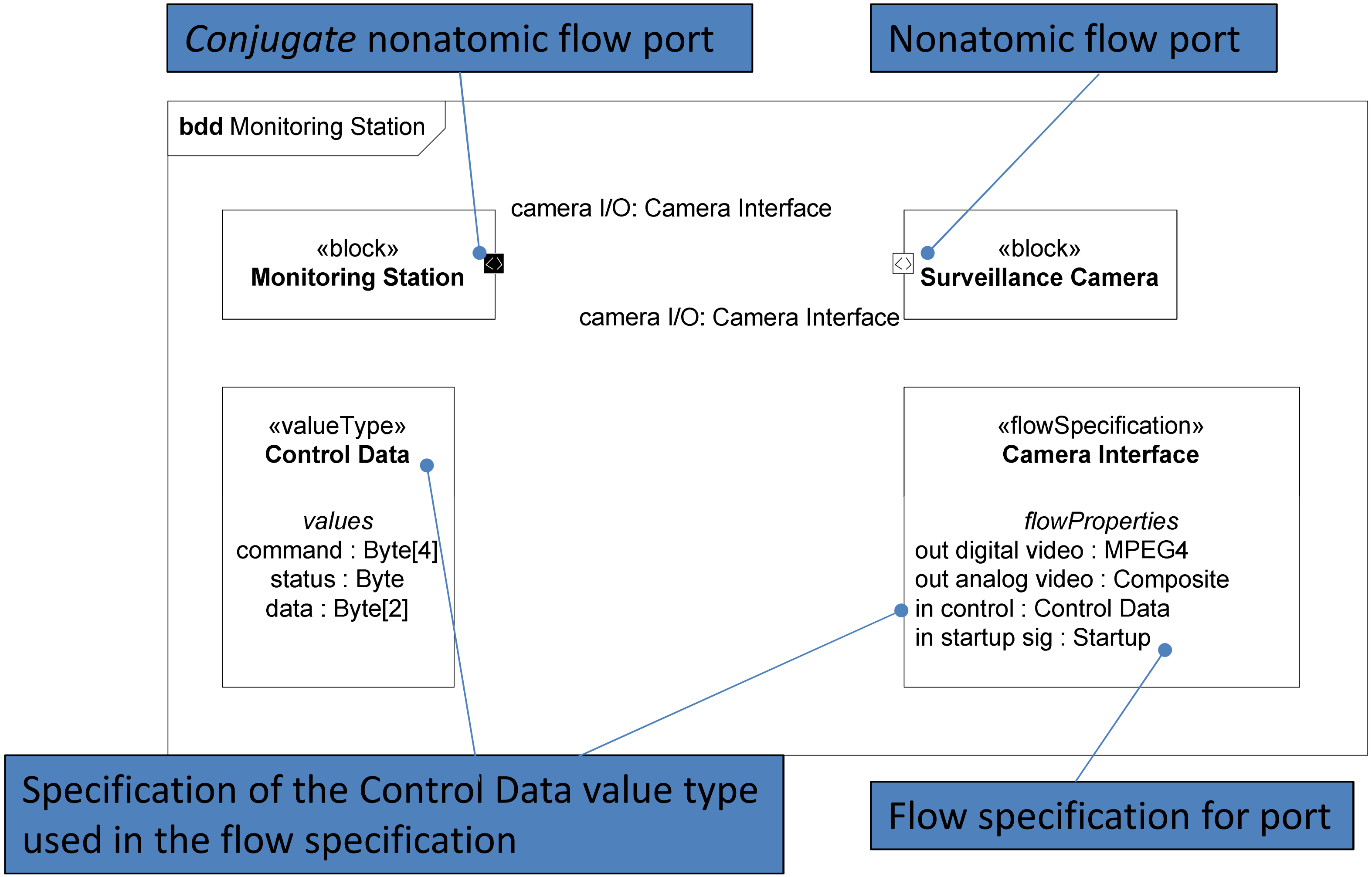




Nonatomic flow ports

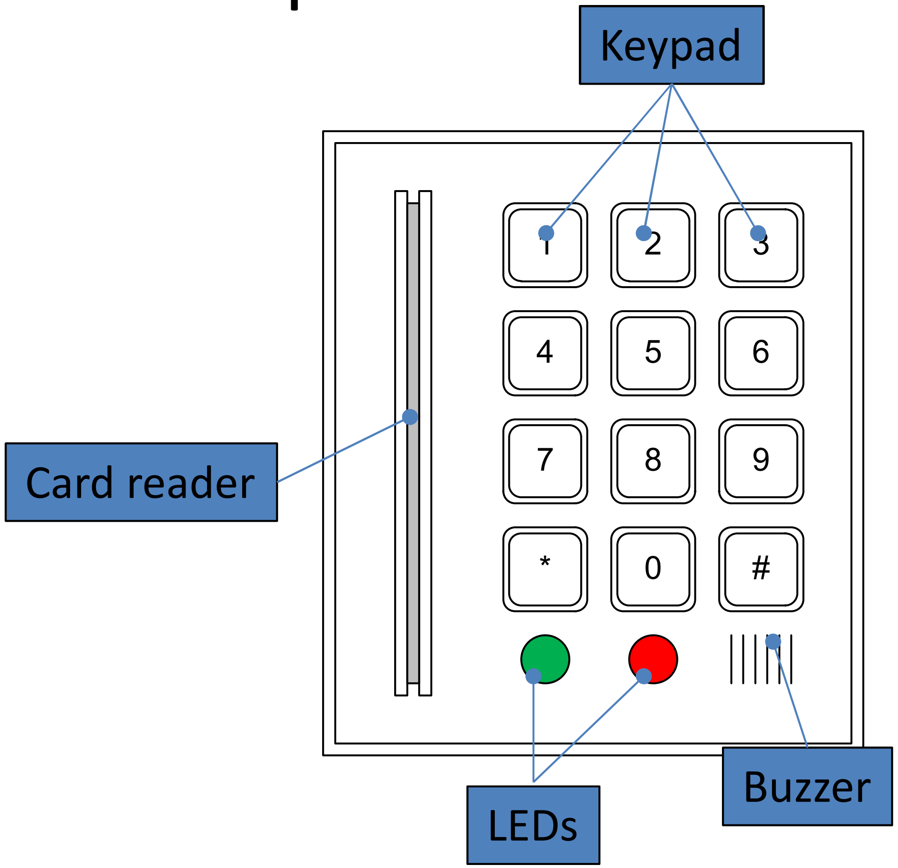
- Nonatomic flow ports are used for composite interfaces
  - ”Nonatomic” ~ ”composed of several things”
- A nonatomic flow port must be matched by a *flow specification* on a bdd
  - Each component given as a flow property (type and direction)
- You may also use a *conjugate flow port* (see next slide)

Nonatomic flow ports

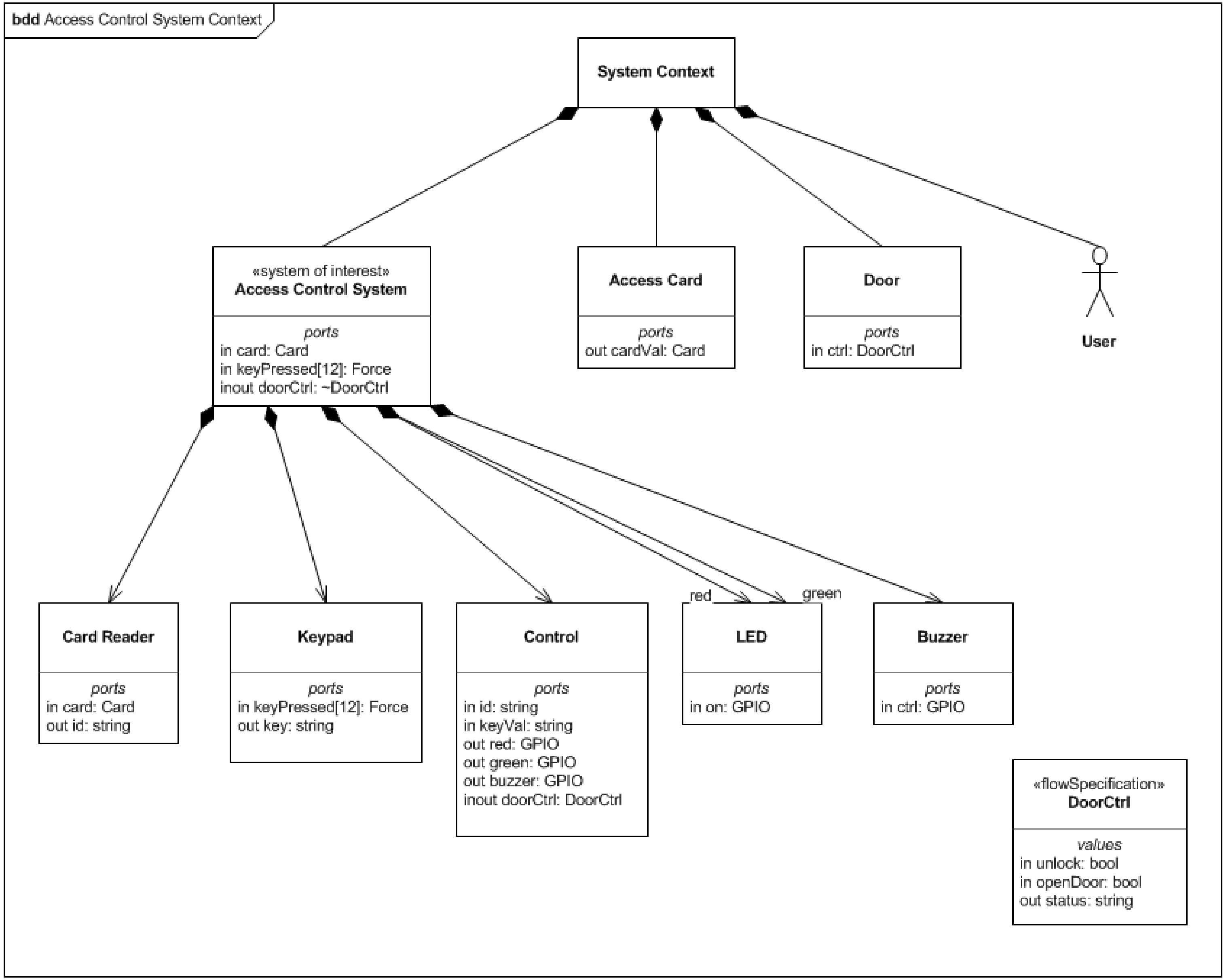


Your turn!

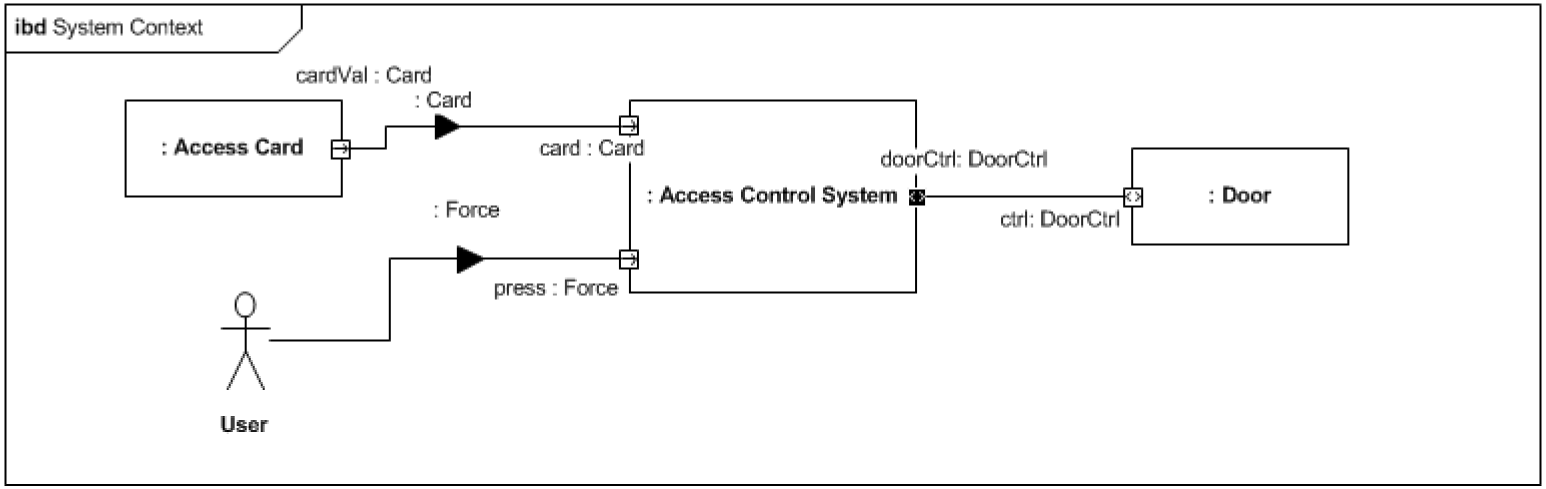
- Given a bdd for an access control system, create ibd incl. ports and item flows

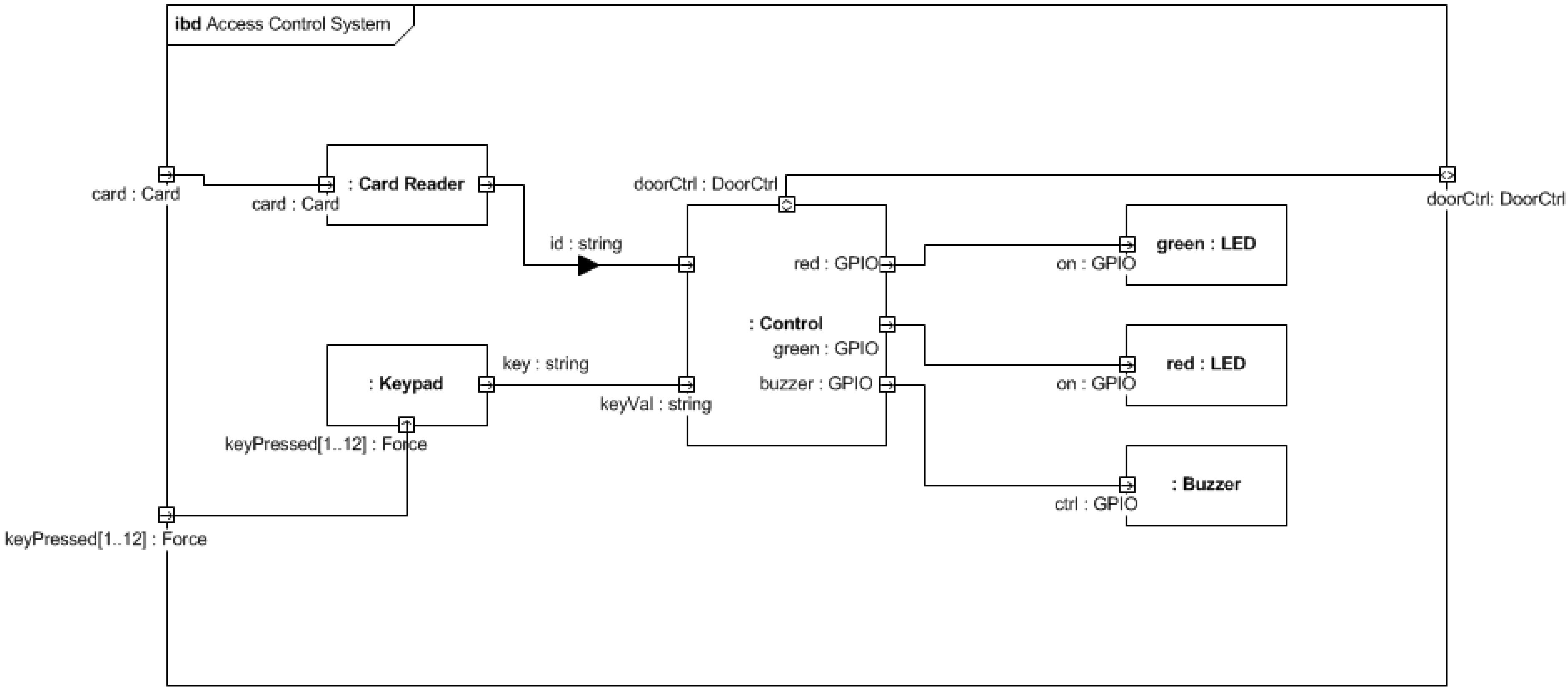


Your turn!

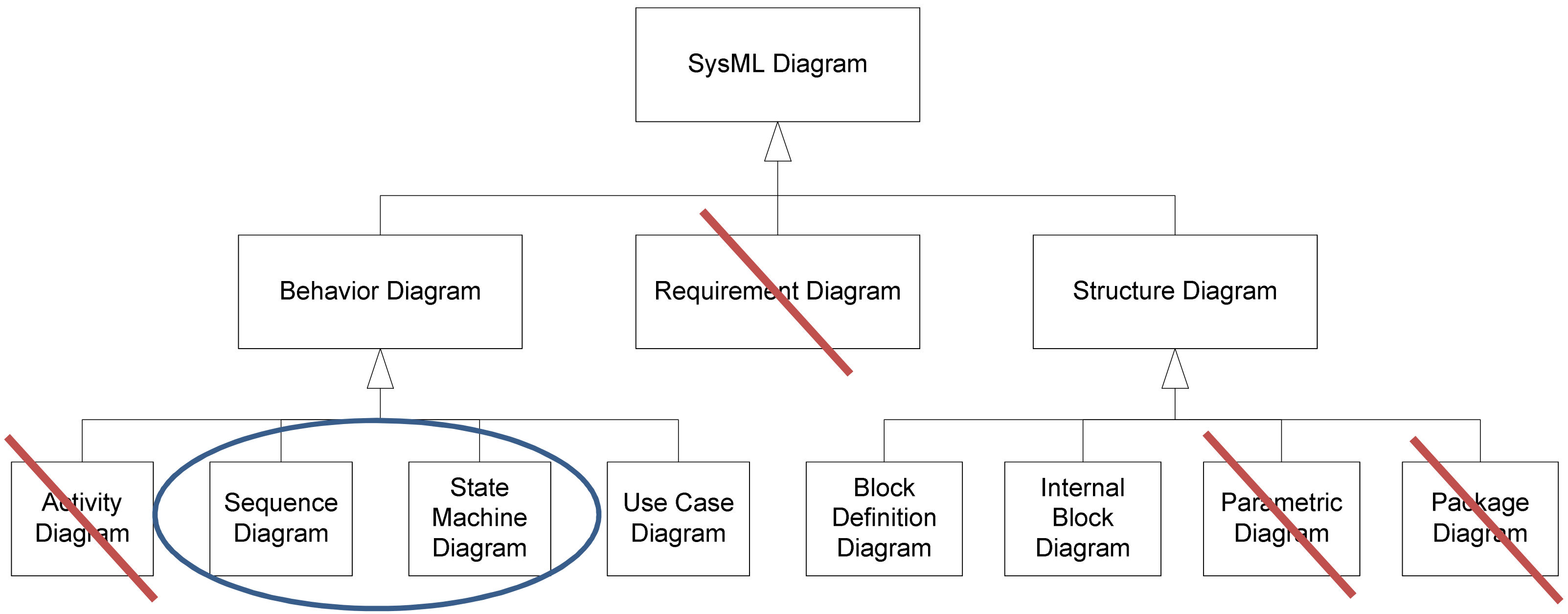


bdd Access Control System Context
System Context
«system of interest»
Access Control System
ports
in card: Card
in keyPressed[12]: Force
inout doorCtrl: ~DoorCtrl
Access Card
ports
out cardVal: Card
Door
ports
in ctrl: DoorCtrl
User
Card Reader
ports
in card: Card
out id: string
Keypad
ports
in keyPressed[12]: Force
out key: string
Control
ports
in id: string
in keyVal: string
out red: GPIO
out green: GPIO
out buzzer: GPIO
inout doorCtrl: DoorCtrl
red
green
LED
ports
in on: GPIO
Buzzer
ports
in ctrl: GPIO
«flowSpecification»
DoorCtrl
values
in unlock: bool
in openDoor: bool
out status: string


ibd System Context
cardVal : Card
: Card
: Access Card
card : Card
: Access Control System
doorCtrl: DoorCtrl
: Door
ctrl: DoorCtrl
: Force
press : Force
User

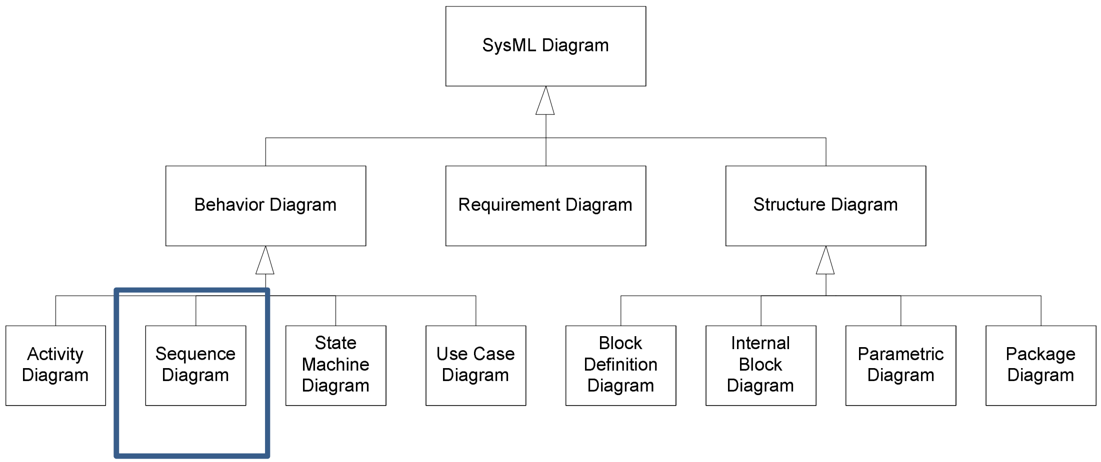

ibd Access Control System
card : Card
card : Card
: Card Reader
doorCtrl : DoorCtrl
doorCtrl: DoorCtrl
id : string
red : GPIO
green : LED
on : GPIO
: Control
green : GPIO
red : LED
on : GPIO
: Keypad
key : string
keyVal : string
buzzer : GPIO
keyPressed[1..12] : Force
: Buzzer
ctrl : GPIO
keyPressed[1..12] : Force

SysML: Diagram types



Sequence diagrams

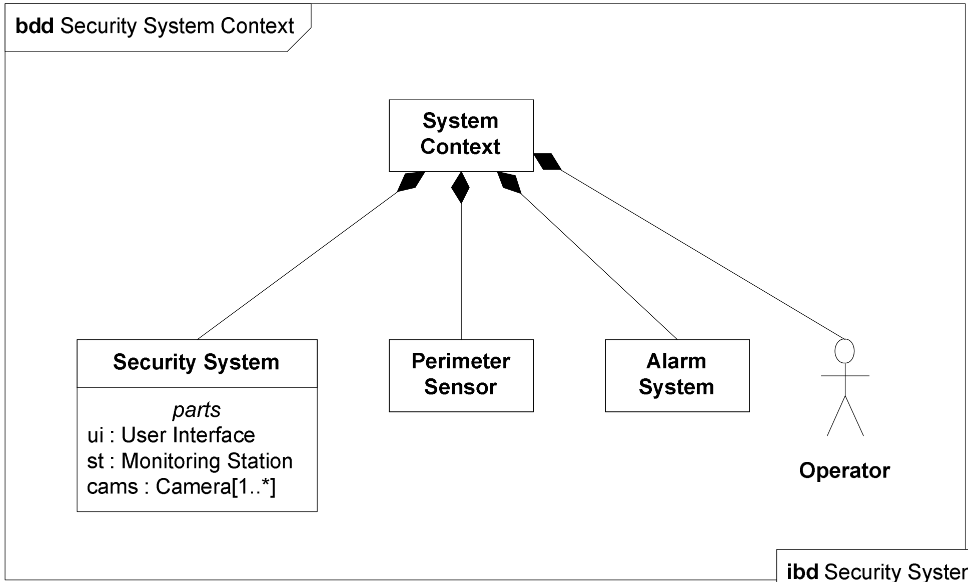
- Sequence diagrams (diagram type *sd*) model interactions between *parts* of a block

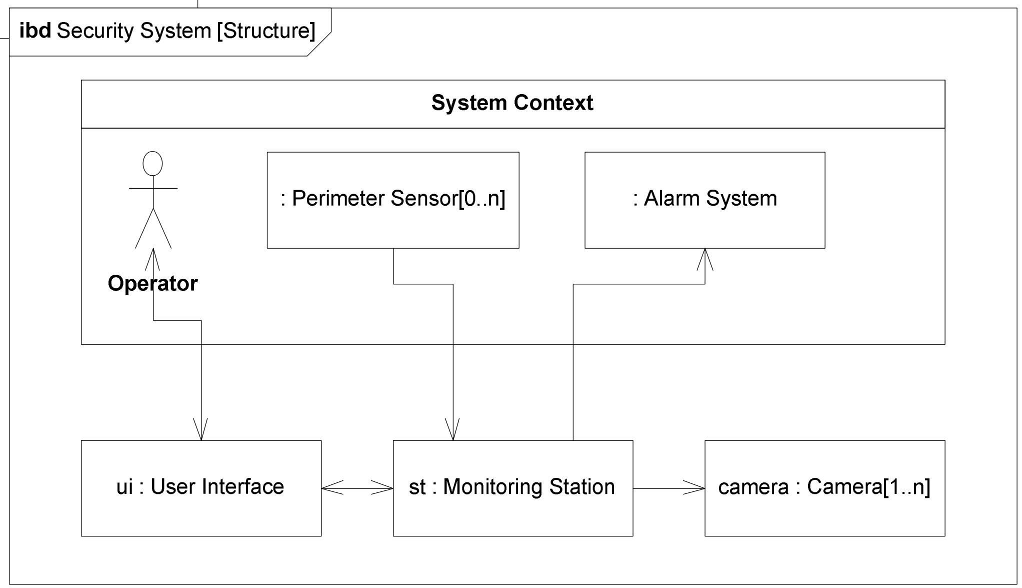


Sequence diagrams

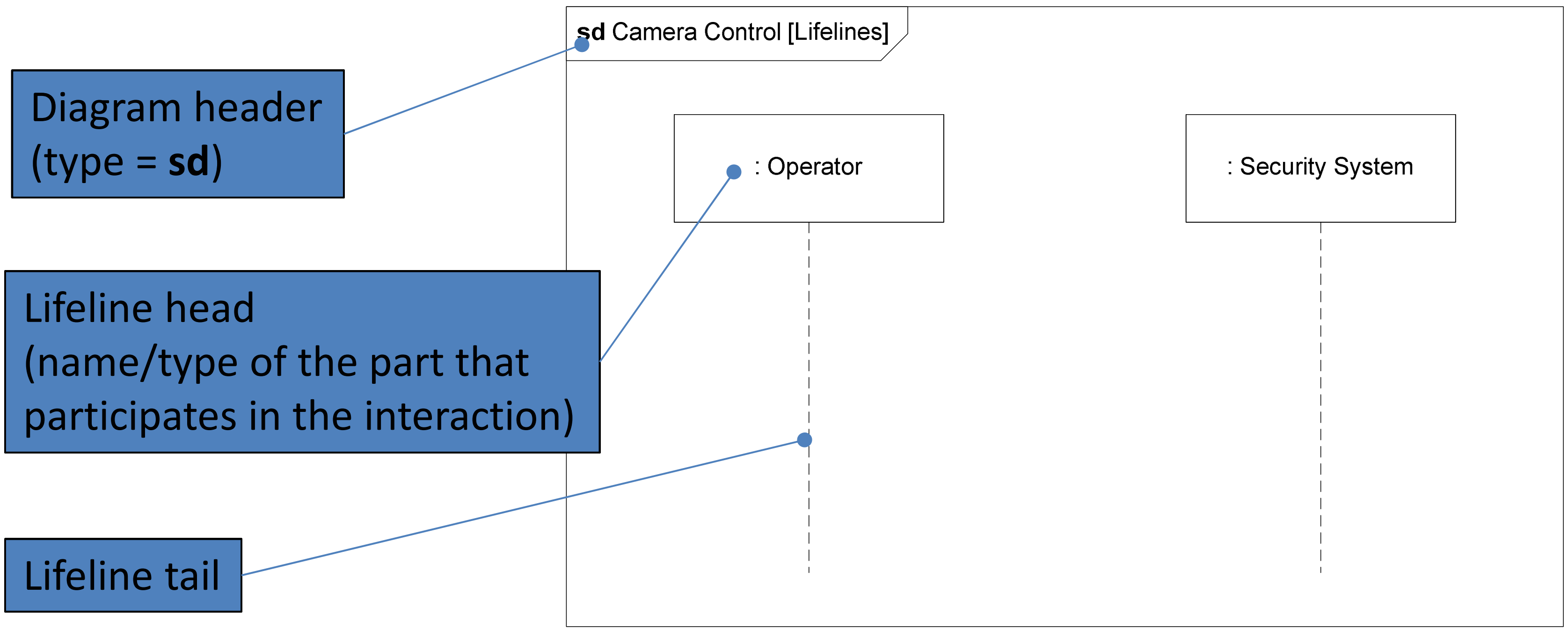
- Sequence diagrams are used to model *message*-based behaviour
- The interactions take place within a block between its elements of internal structure (parts)
- The basic diagram consists of *lifelines* with *messages* between them.

SD's – example system (structure)

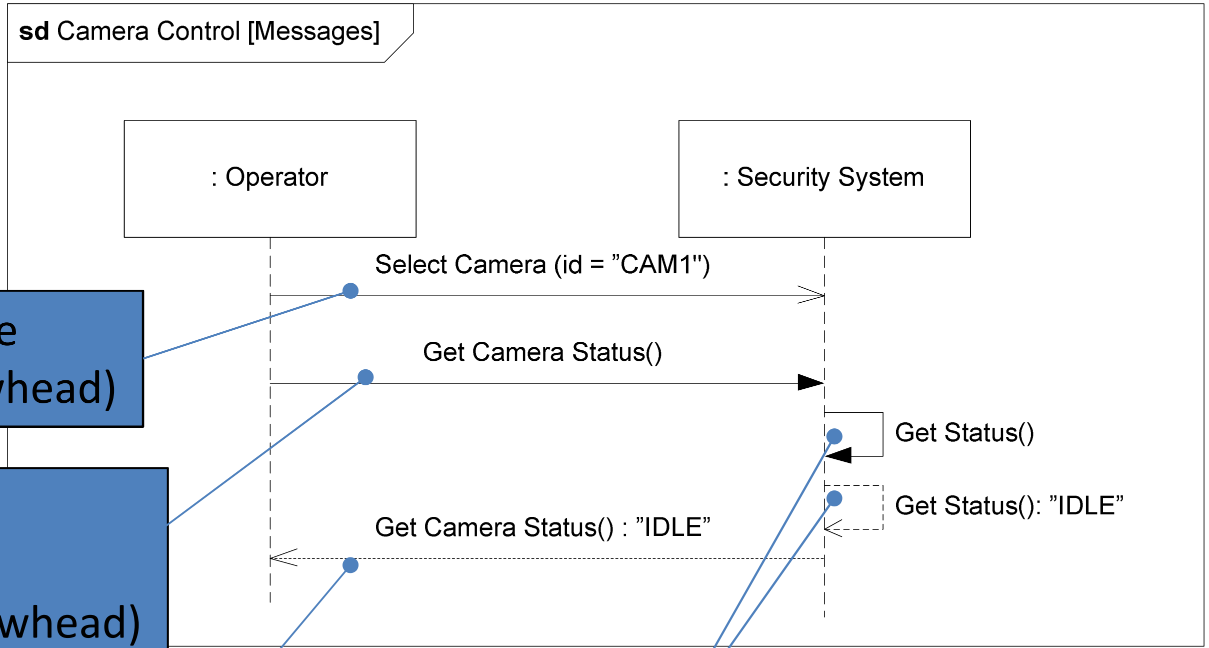




SD's – lifelines



SD's – messages



Asynchronous message
(solid line, open arrowhead)

Synchronous message
(method call)
(solid line, closed arrowhead)

Return message
(method call)
(dashed line, open arrowhead)

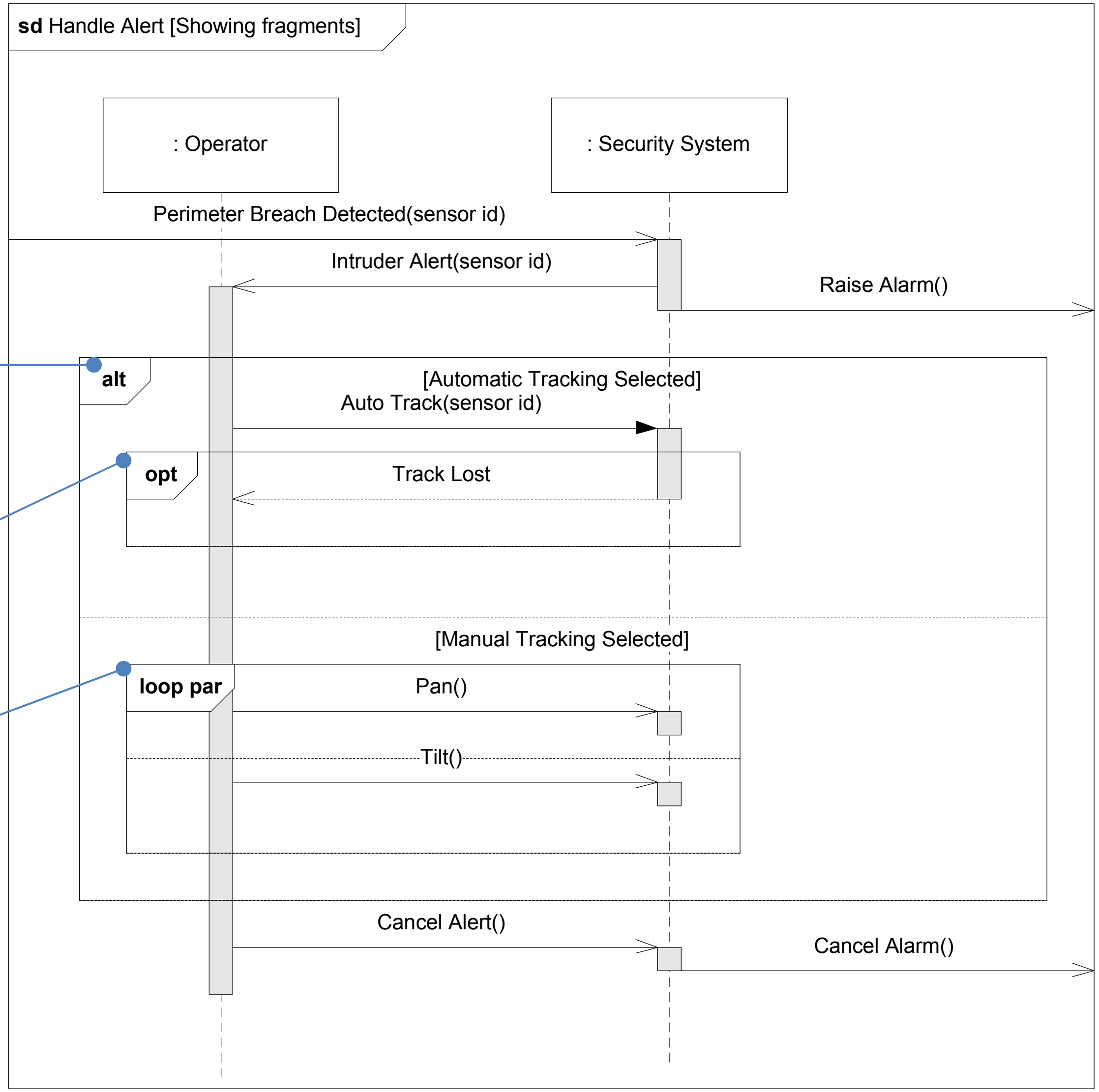
Message-to-self

SD’s – fragments

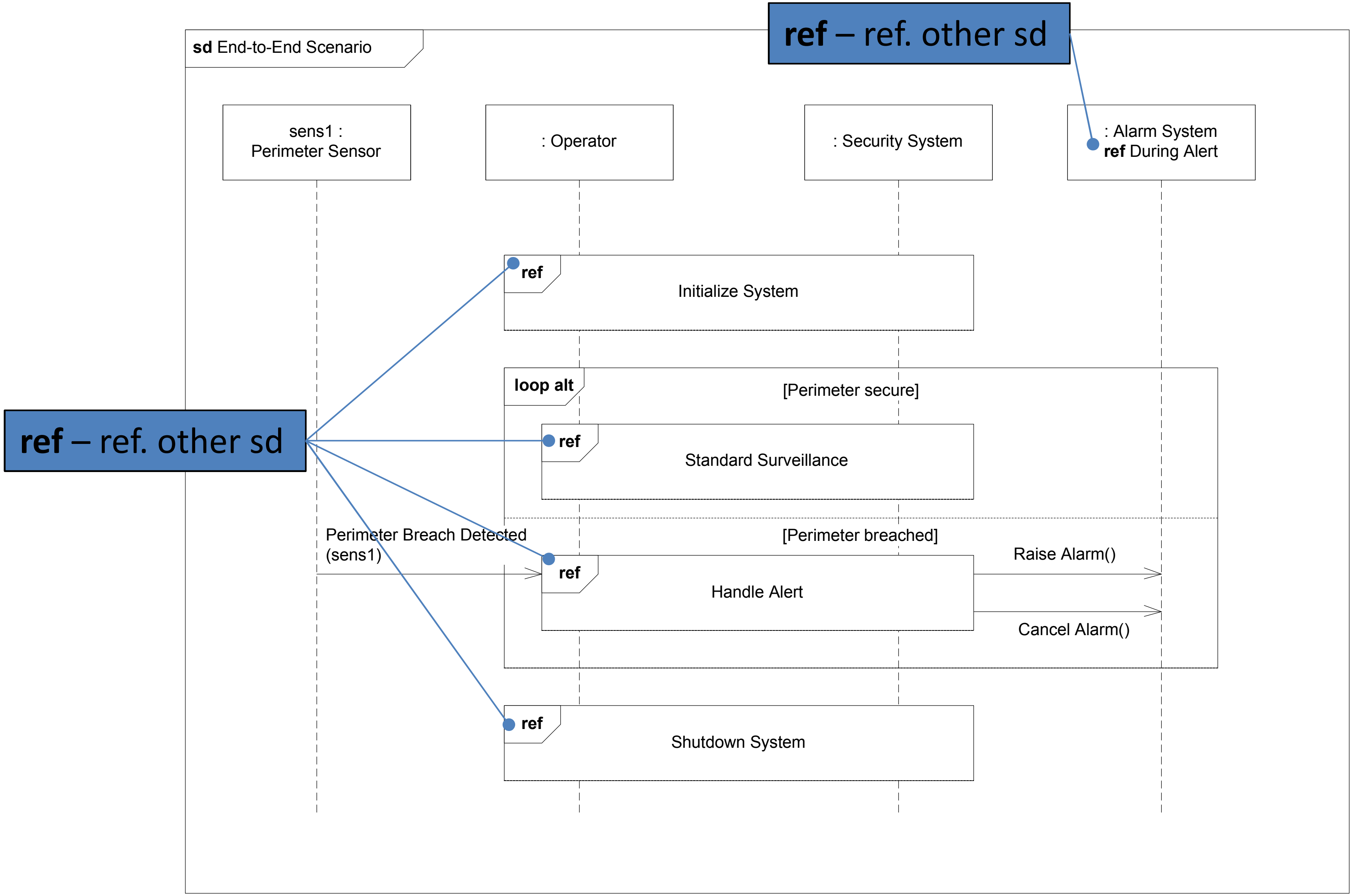
alt - Alternative activities

opt - Optional activity

loop – loop activity
par – parallel activities



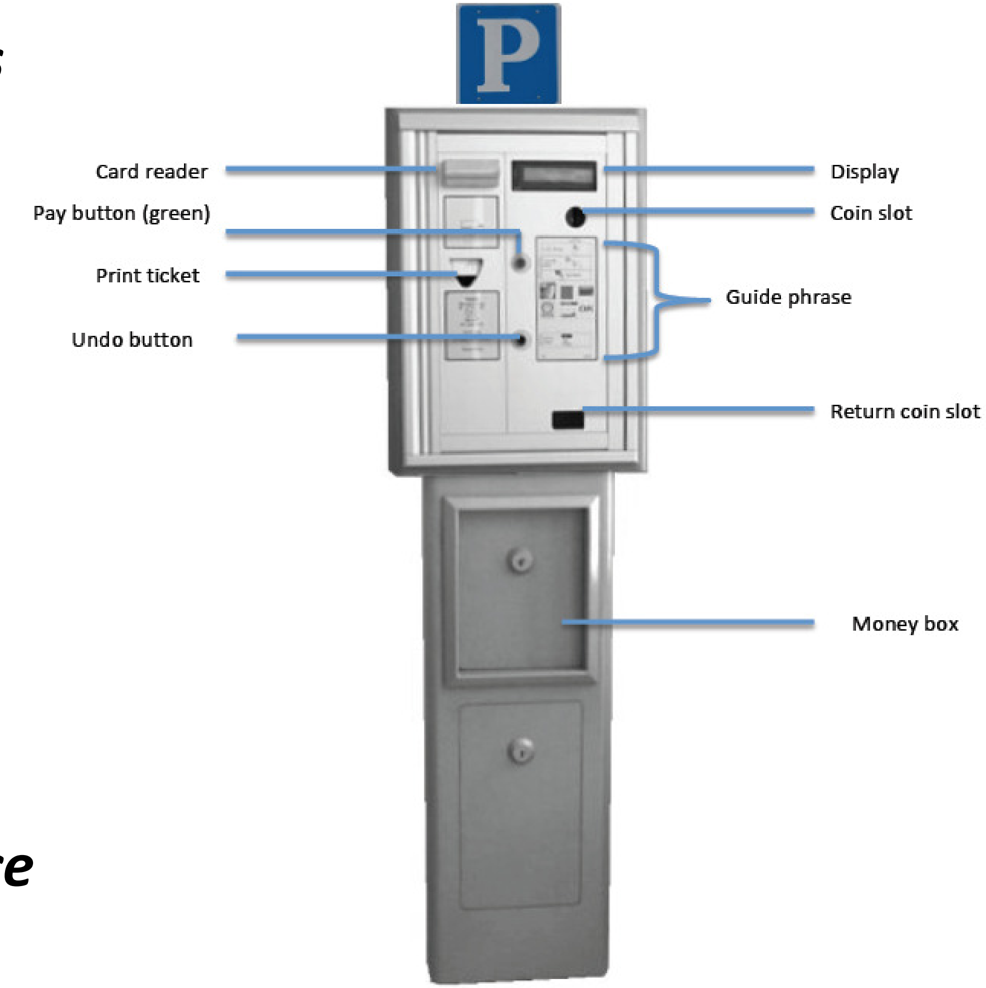
SD’s – reference blocks



Parkeringsautomat

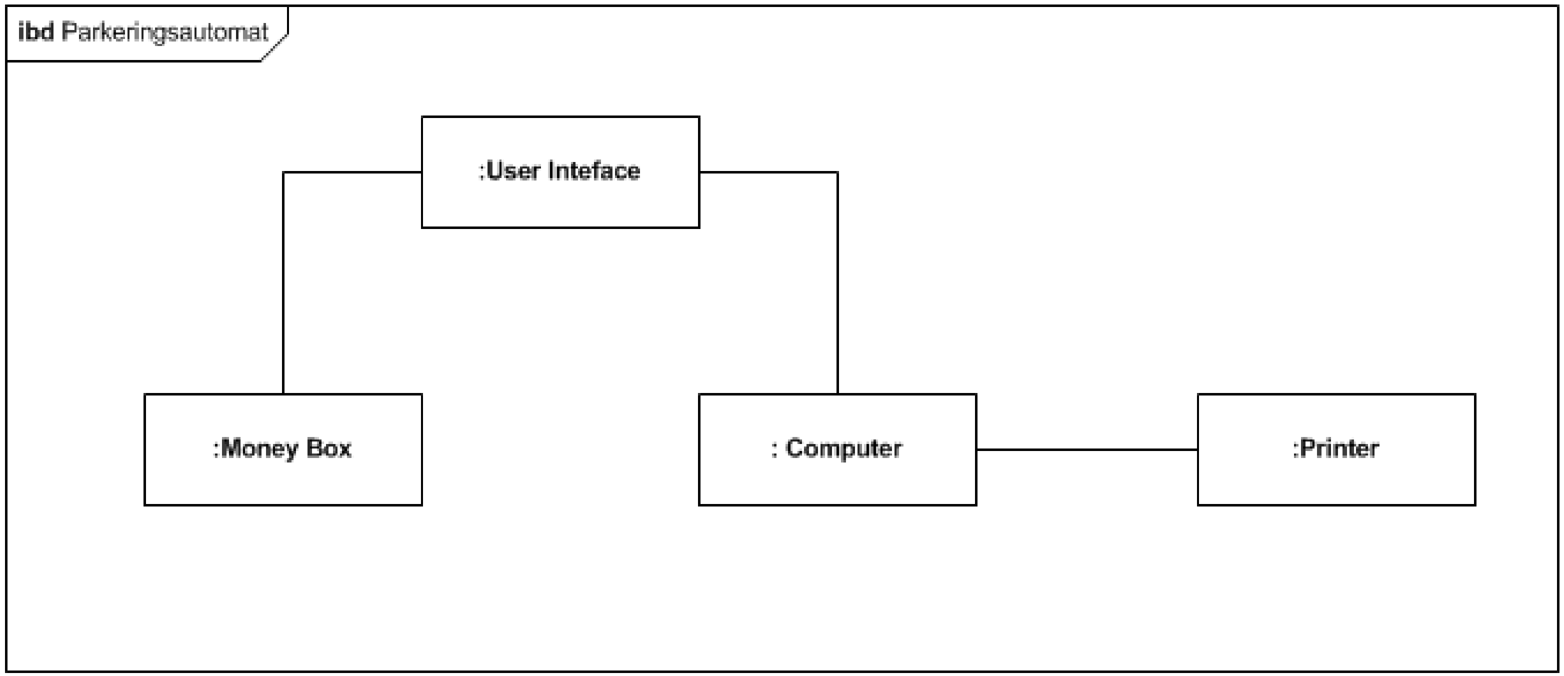
1. bdd diagram of components

- Parkeringsautomat
  - Printer
  - Money Box
  - Computer
  - User Interface
    - Card Reader
    - Controller
    - Display
    - Green Button
    - Red Button

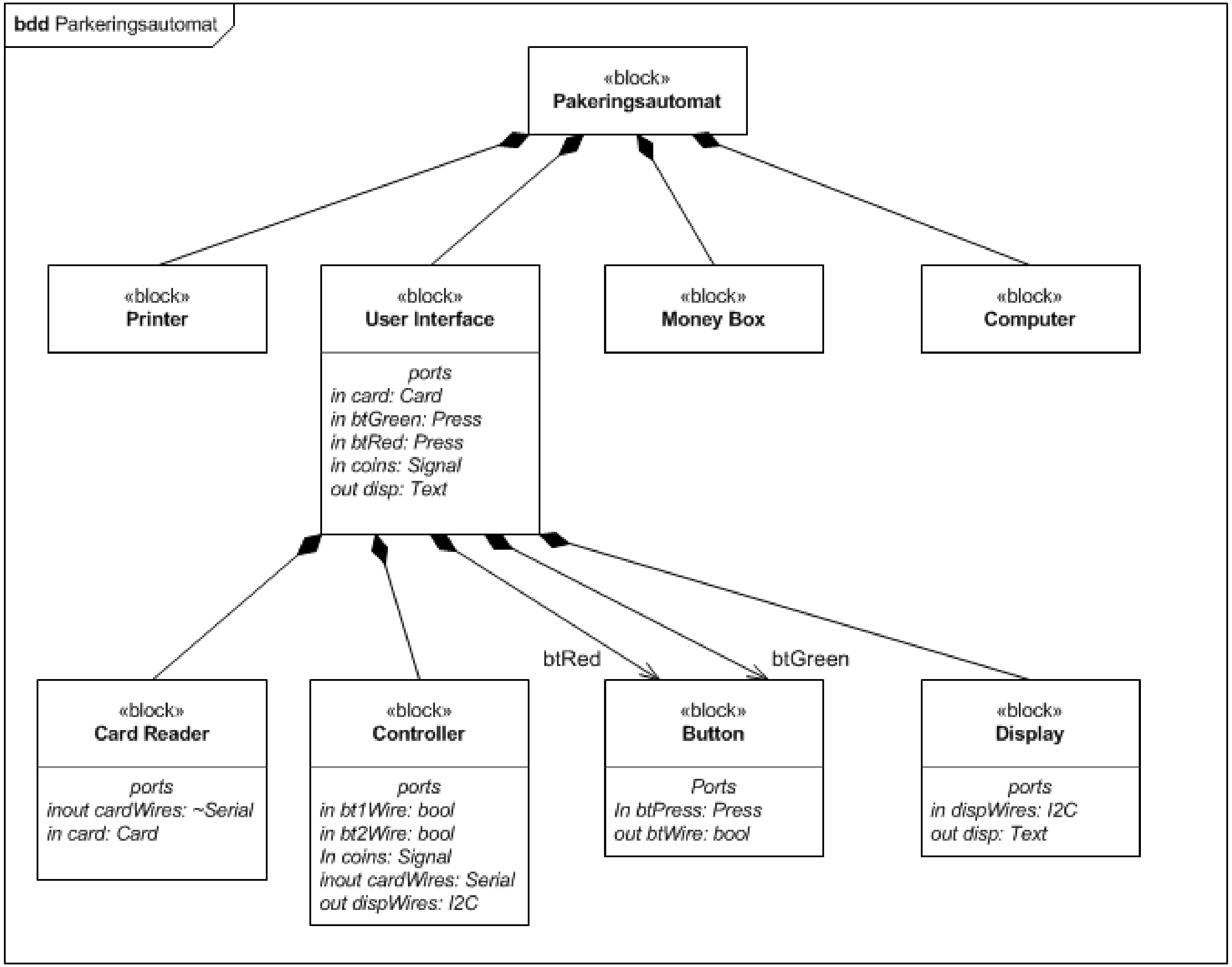


2. ibd diagram of User Interface

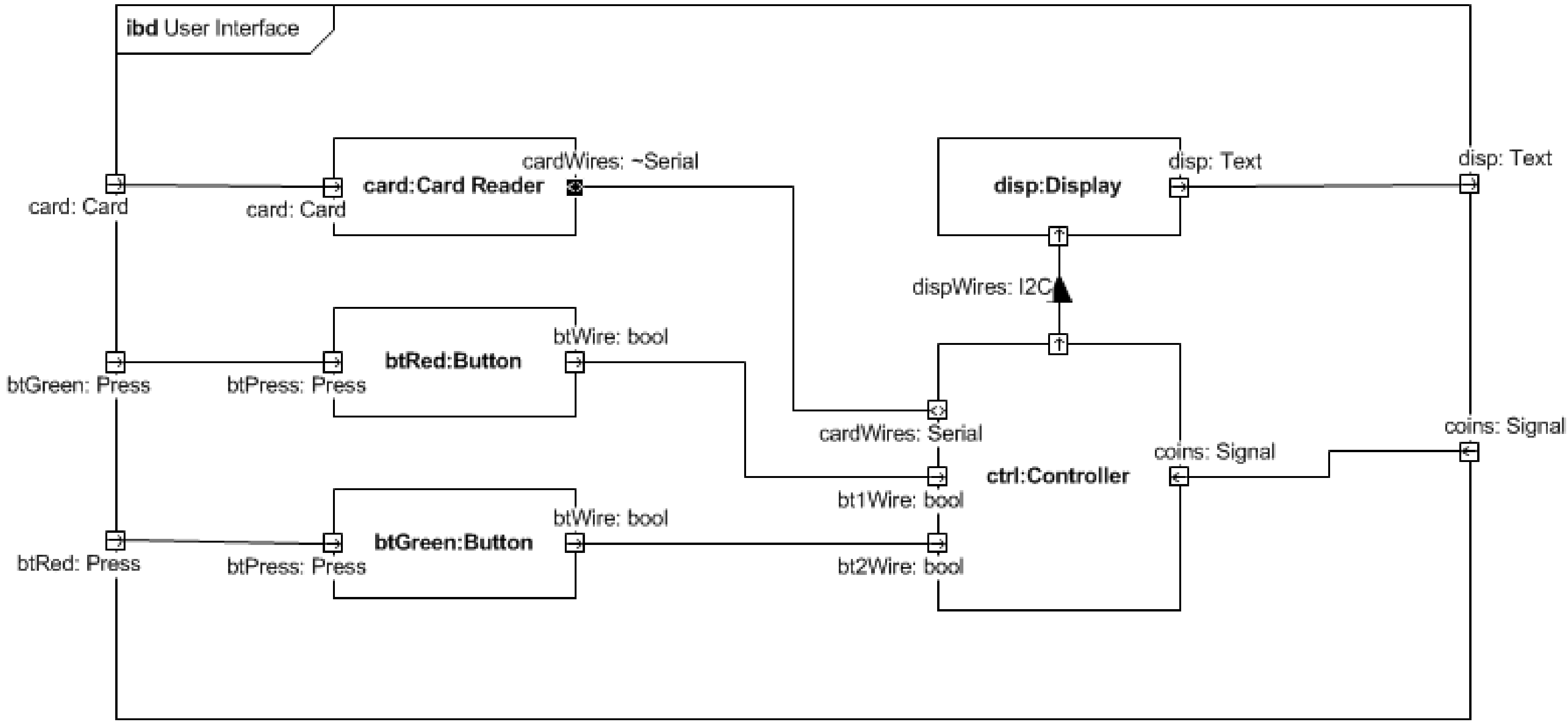
ibd Parkeringsautomat



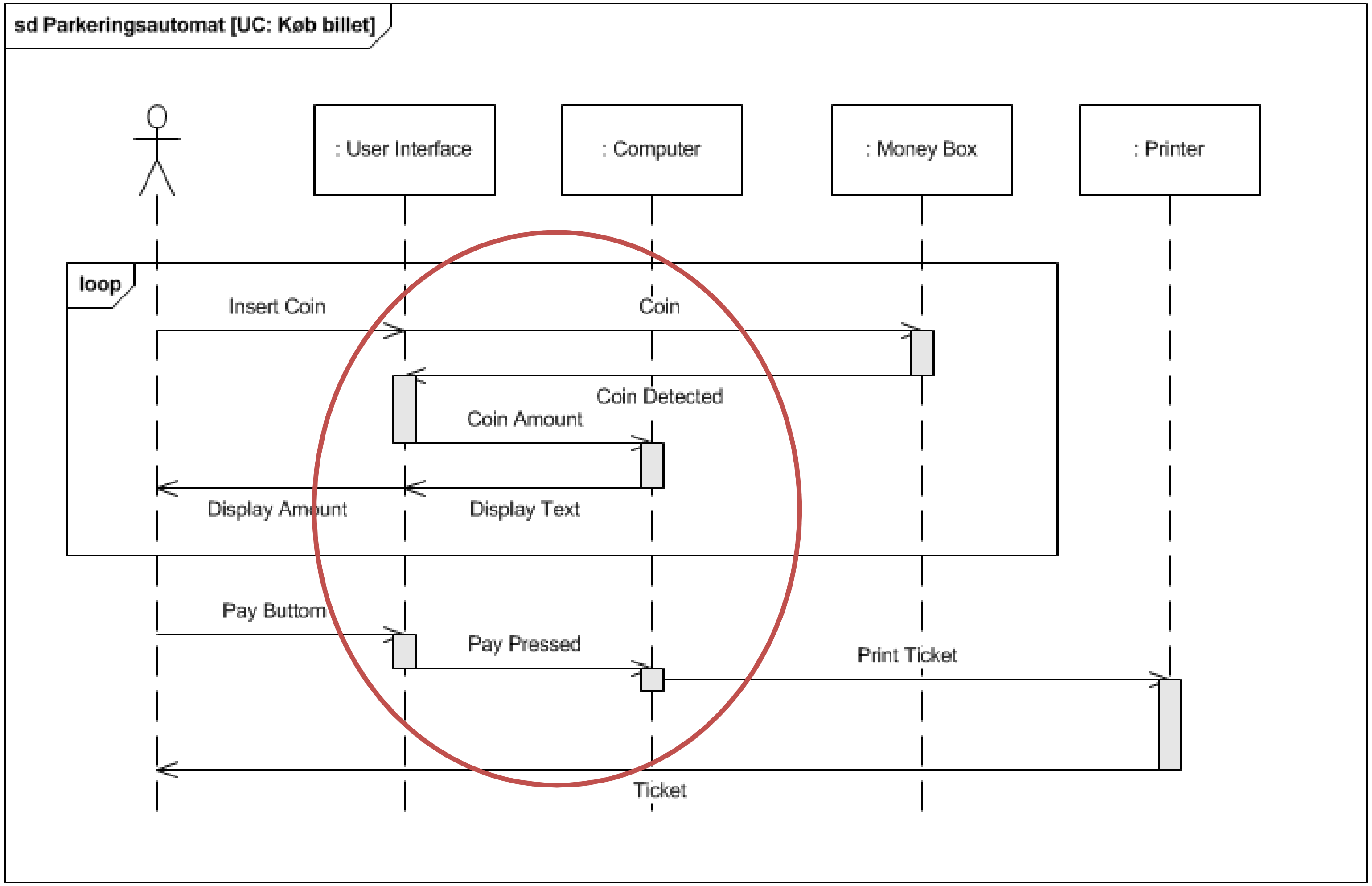
bdd Parkeringsautomat



ibd User Interface

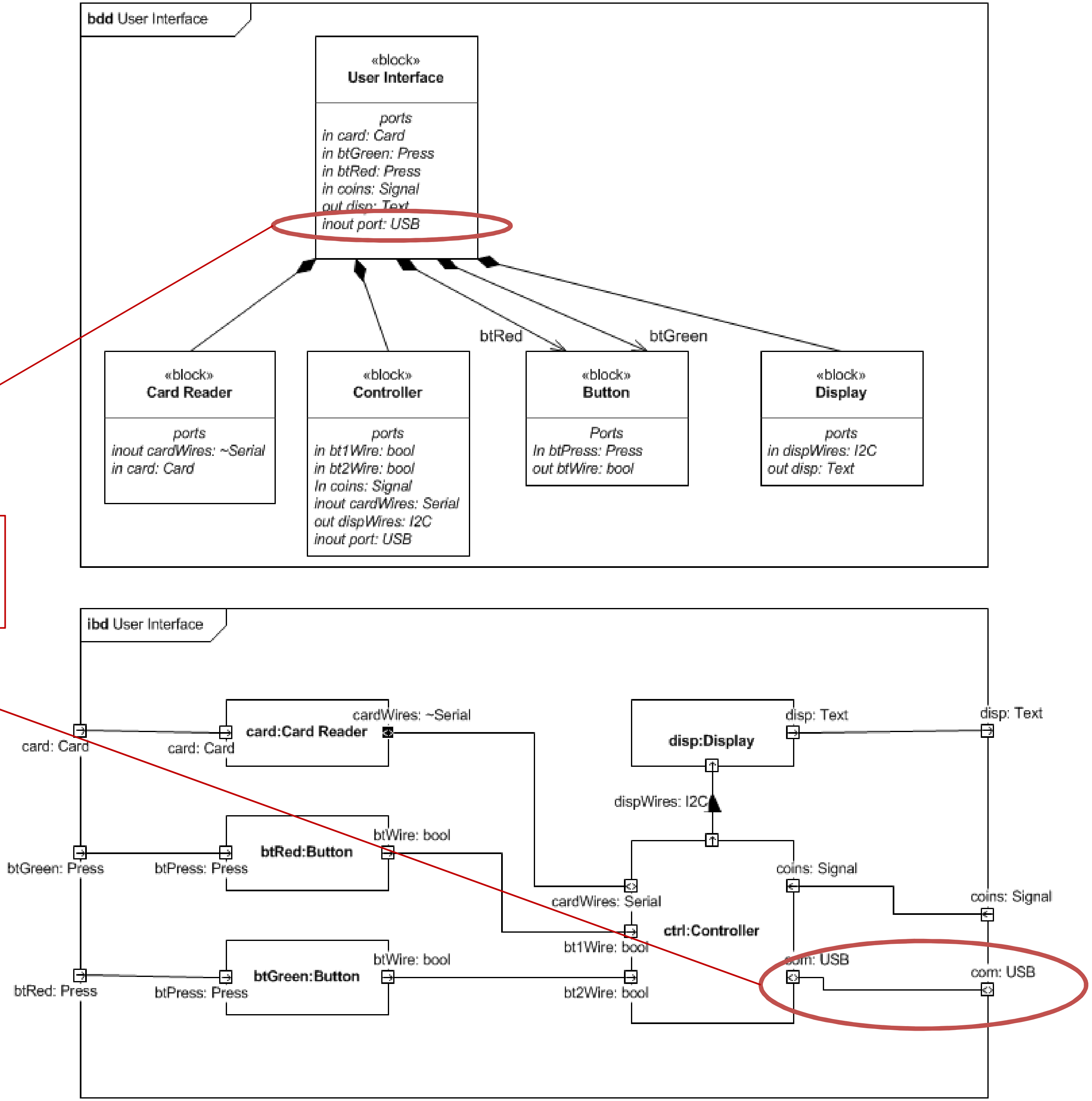


Sequence [UC:Køb billet]

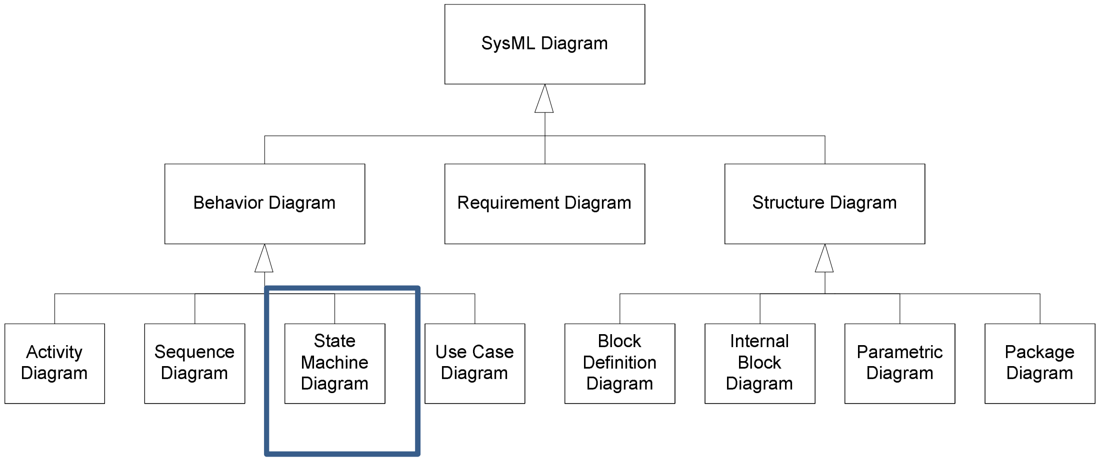


Revised bdd/ibd

- Connection between controller and computer!



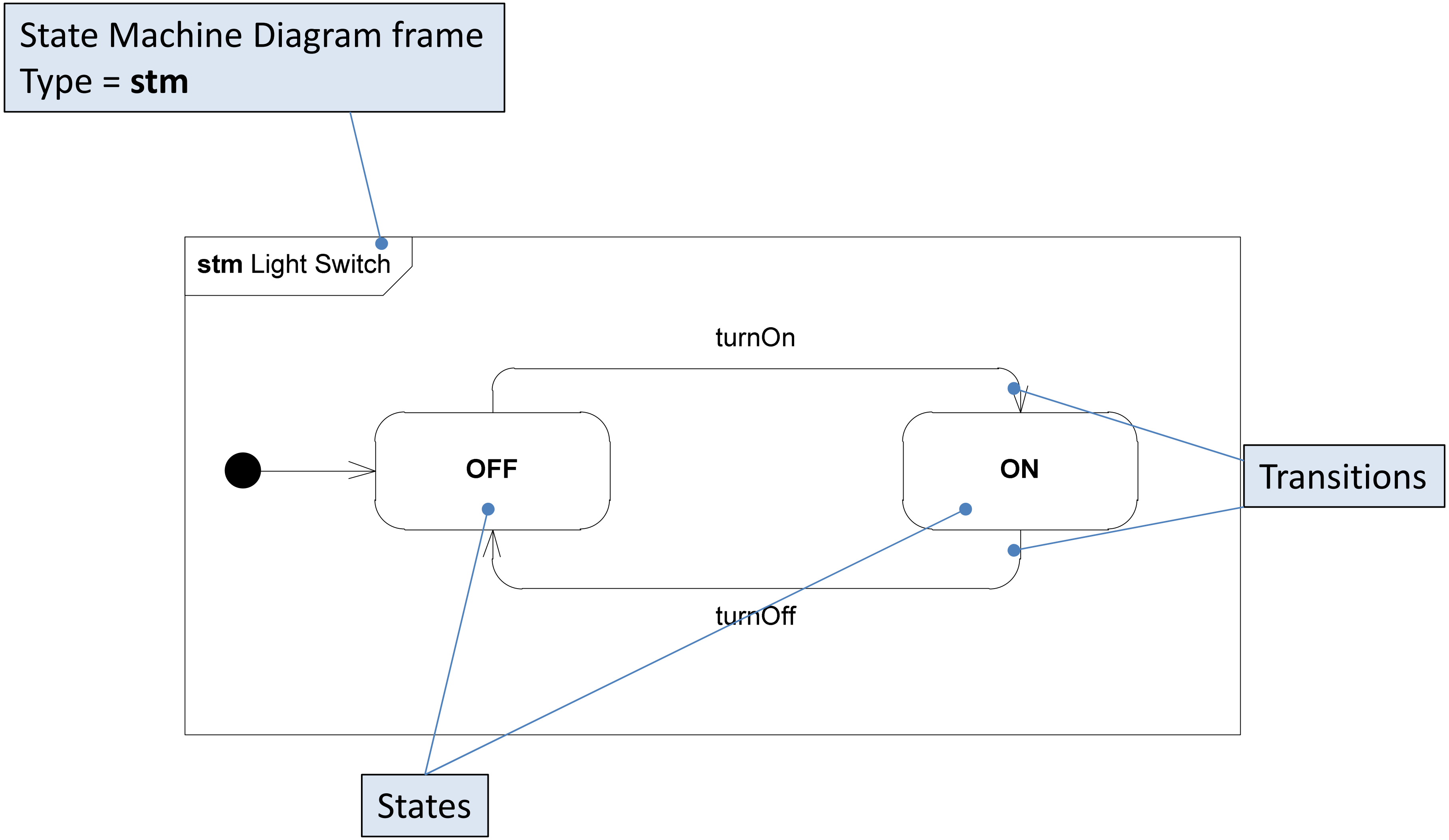
State Machine Diagrams



State Machines

- State Machines Diagrams (**stm**), aka *state charts*, are used to model *state-dependen*t behaviour of a block throughout its lifecycle
- A *state* is some significant condition in the life of a block
  - Typically, different states respond differently to same events
- A *state machine* is always in a certain *state* and will remain there until some *event* causes it to *transition* to another state.

States and transitions - basics

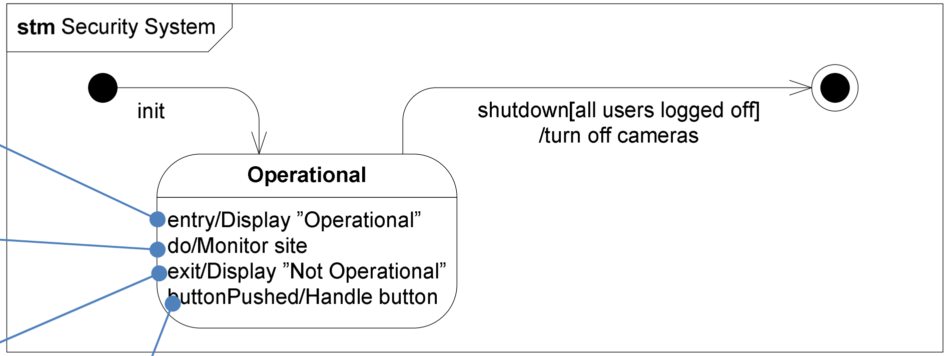


States in detail

entry behaviour is executed on entry into the state

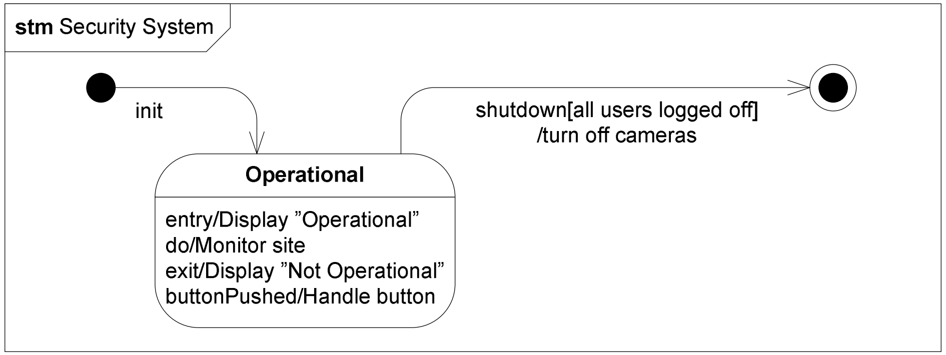
do behaviour is continuously executed after entry until exit

exit behaviour is executed just prior to exit of the state

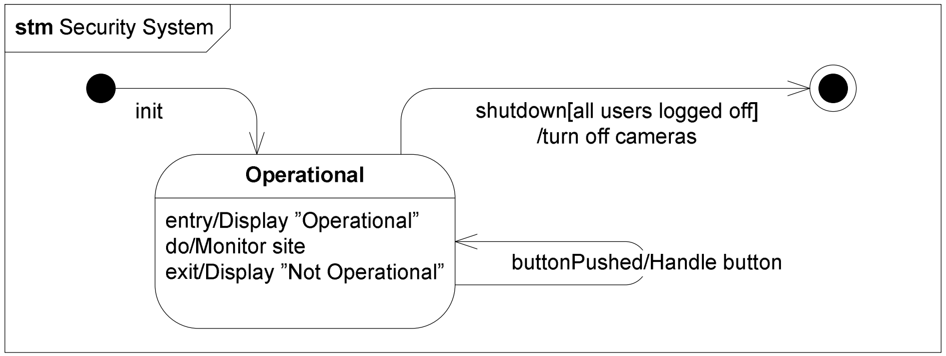


When *buttonPushed* occurs, do behaviour is interrupted and Handle button is executed. Then, do is resumed

States in detail
– what's the difference?



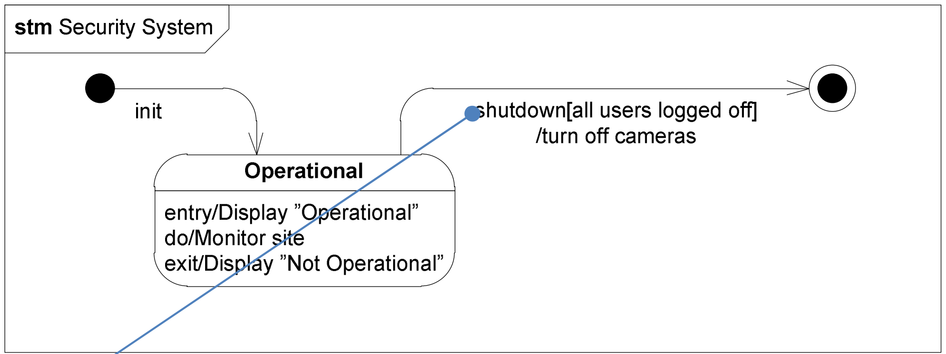
buttonPushed→
1. *Handle button*



buttonPushed →
1. *Display "Not operational"*
2. *Handle button*
3. *Display "Operational"*

Transitions in detail

- Transitions consist of *trigger*, *guard* and *effect*: `trigger[guard]/effect`
- When `trigger` occurs, `guard` is evaluated.
  - If `guard` is true, `effect` occurs.
  - If not, `trigger` is consumed without effect



Trigger = shutdown
Guard = all users logged off
Effect = turn off cameras

What happens if some user is still logged on?

States with multiple regions

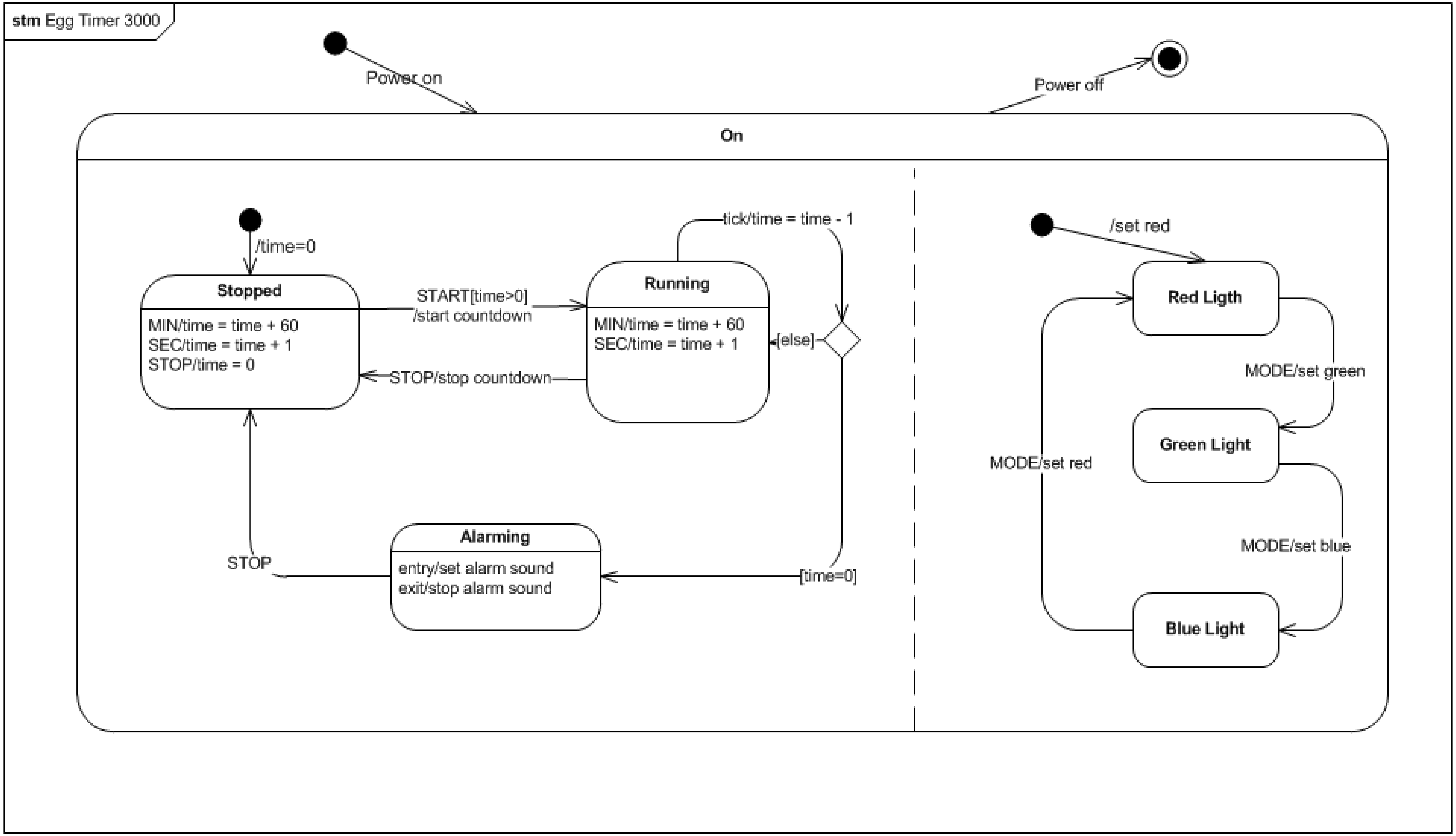
- A state may have multiple regions (aka. *orthogonal* or *independent substates*)
- If the enclosing state is active, each region will have exactly 1 active state
- State transitions in one region does not affect states in another region.
- State transitions can never transition the boundary between regions

Exercise 2 : Pimped Egg Timer 3000

- PET3000 is like ET2000, but the display can be backlit with either red, green or blue light. This is controlled with the **`MODE`** button which toggles the light.

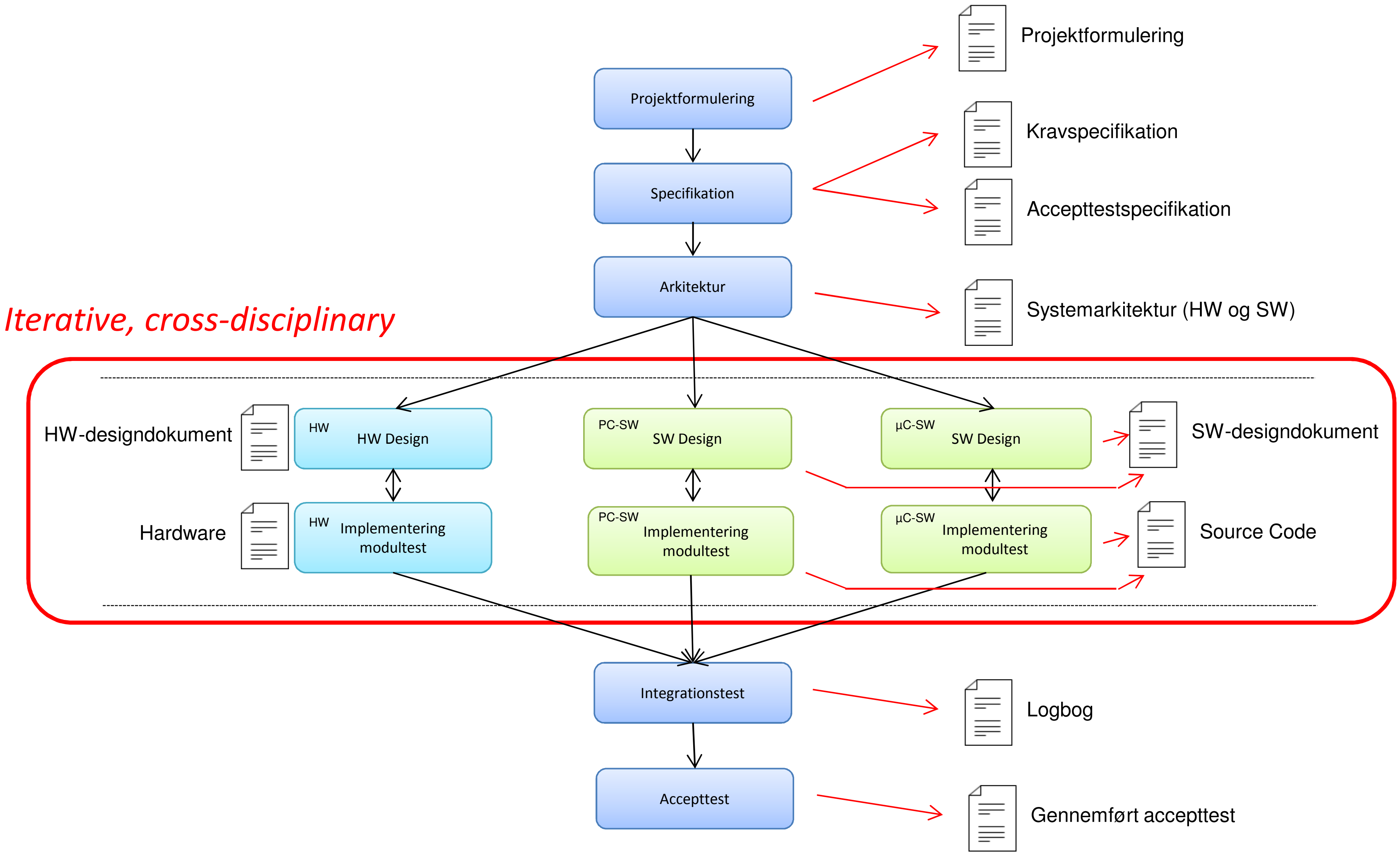
- Draw it's state machine diagram

Pimped Egg Timer 3000 state machine



Projekt dokumenter (Ekstra)

The ASE Process



Fælles for alle dokumenter

- **Forside**
  - **Title, version, dato, gruppe, navne**
- **Versionshistorik**
  - **Resumé af ændringer med dato, version og initialer**
- **Indholdsfortegnelse**
- **Indledning**
  - **Formål**
- **Referencer**

Det skal være muligt at beskrive hvilke versioner, der passer sammen for hver baseline i jeres projekt.

Baseline: Angiver hvilke versioner af jeres dokumentation, der hører sammen på et bestemt tidspunkt i jeres projekt f.eks. ved start på en ny udviklingsfase.

Projektformulering

- **Hvad er problemet**?
  - Omfang og relation til omverden
- **Beskriv hvad vil jeres løsning bidrage med**?
  - For hvem bidrager jeres løsning en funktionalitet og værdi?
  - Hvad er visionen for jeres projekt?
- Beskriv projektets overordnede fokus og funktionalitet
- **Afgrænsning – hvad er med og ikke med**?
  - Overordnet system illustration
- Fylder mellem 1-2 sider

Kravspecifikation

- **Systembeskrivelse**
- Funktionelle krav med **Use Case** (UC)
  - Beskrivelse af systemets aktører
  - Use Case diagram
- **Kvalitetskrav** (Ikke-funktionelle) med brug af kategorierne: (F)URPS+
  - Formulering af krav med brug af MoSCoW prioritering
- Andre krav:
  - Udviklingsproces
  - Tekniske krav og grænseflader
  - Udviklingsværktøjer
- Prototype af brugergrænseflade
  - Skærmbilleder

Accepttest

- Beskriv **testsystemet**
  - Testopstilling og udstyr krævet for at gennemføre testen
  - Testkomponenter
- Udarbejdes på baggrund af specifikation
- **Test cases og scenarier**
  - Udgangspunkt i Use Cases
    - Hovedscenarier og Undtagelser
    - Forventet resultat
  - Ikke-Funktionelle krav
- Underskrevet og godkendt gennemført test med kunden

Systemarkitektur

- **Systemets struktur** (HW)
  - Hardware strukturen beskrives med SysML (Bdd og Ibd)
  - Diagrammerne suppleres med tekst
    - Blokke beskrives med formål og funktion
    - Grænseflade af forbindelser mellem blokke
    - Specifikation af elektriske signaler og protokoller
    - Beskriv gerne signaler og forbindelser i tabeller

- **Software arkitekturen** (SW)
  - Domænemodel
  - Applikationsmodeller (UML)
    - En applikationsmodel for hver delsystem (CPU)
      - UML klasse- og sekvensdiagrammer

- **Eksterne grænseflader**
  - Kommunikations protokoller mellem systemet og eksterne enheder (aktører)

Design og test

- Detaljeret design af hver **HW block**
  – Elektriske diagrammer (Relation til arkitekturen)
  – Formler og beregninger
- Detaljeret design af hver **SW klasse**
  – Metoder med parameter og retur værdier
  – Sekvens- og state-diagrammer
- **Enhedstest** og resultat
- **Integrationstest** og resultater

ISE - Lessons and topics

- ***System Specification, Quality and Process***
  - Development Processes (1)
  - Specification, Use Cases (2)
  - System Test
  - Quality Assurance
- ***SysML Diagrams***
  - SysML structure diagrams (3)
  - SysML behavior diagrams (4)
- ***System Analysis and Design, Architecture and Interfaces***
  - System Domain Analysis (5)
  - System Design and Architecture (6)
  - HW/SW Design
  - Interfaces
- ***Project Management***
  - Project Management
  - Scrum

Læringsmål ISE

- **Redegøre** for de grundlæggende principper for en udviklingsproces til udvikling af kombinerede hardware/software-systemer
- **Udarbejde** en kravspecifikation bla. a. vha. use cases
- **Redegøre** for basale elementer i SysML til brug for udvikling af hardware/software systemer
- **Foretage** et overordnet systemdesign
- **Udarbejde, anvende og dokumentere** et overordnet design af hardware og software
- **Anvende basal konfigurationsstyring** af dokumenter
- **Udarbejde systemtest** for hardware/software komponenter
- **Anvende reviewteknikken** som kvalitetsforbedrende aktivitet
- **Redegøre for basal projektstyring**