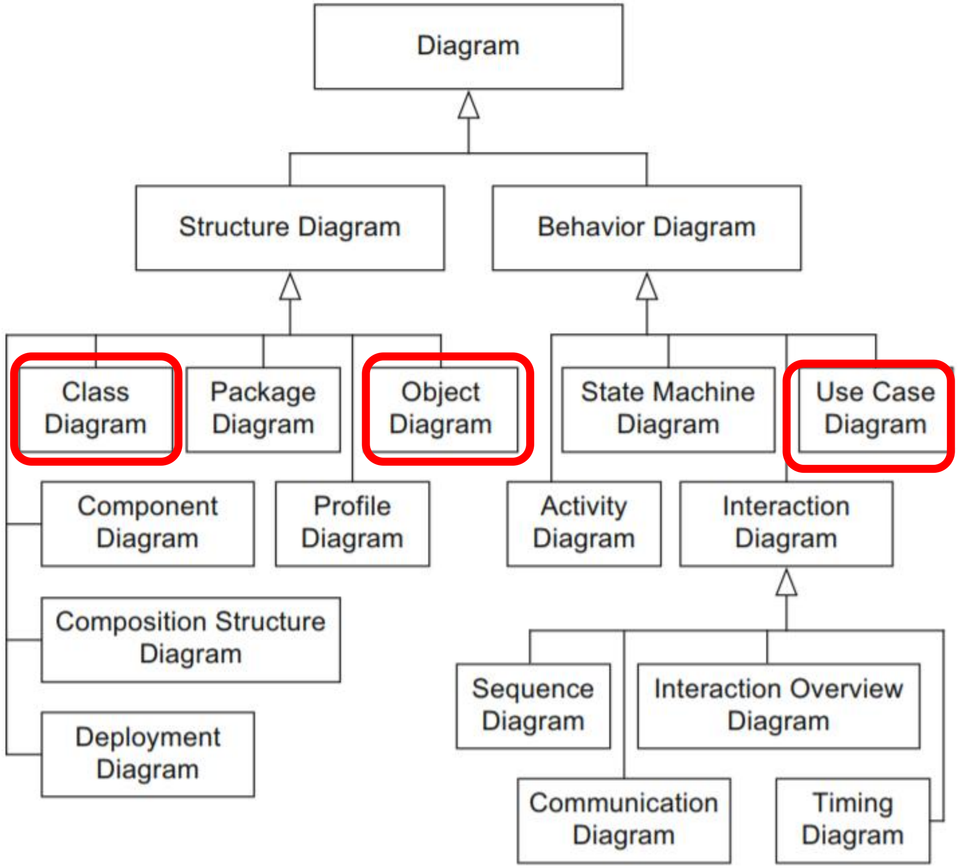
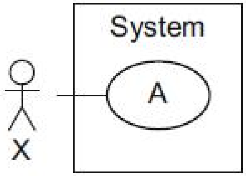
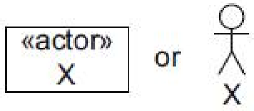
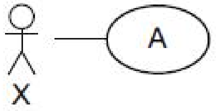
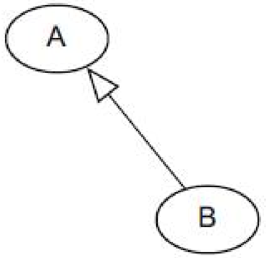
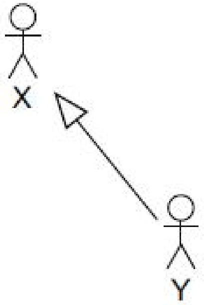
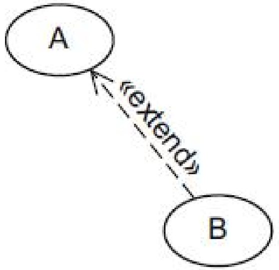
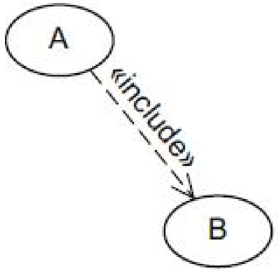
Lecture 5: UML Part 2

UNIFIED MODELING LANGUAGE™

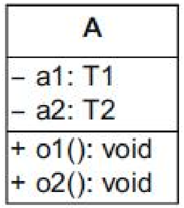
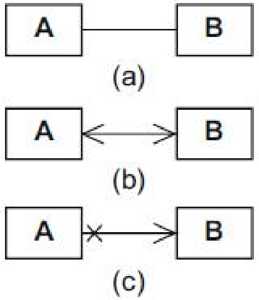
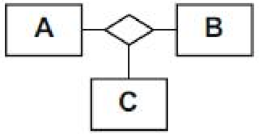
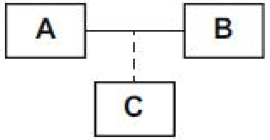
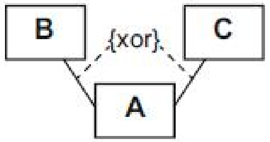
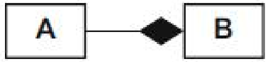
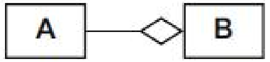
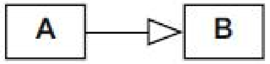
UML Diagrams



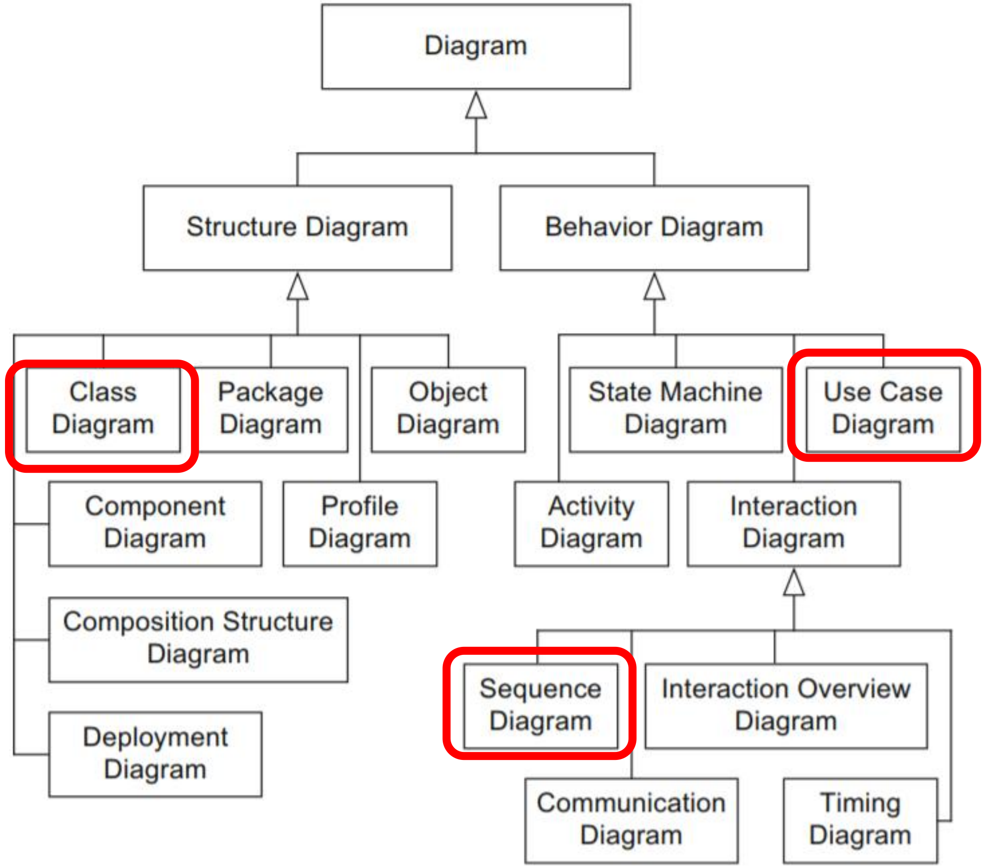
Summary: Use Case Diagram

Name	*Notation*	*Description*
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system
Association		X participates in the execution of A
Generalization (use case)		B inherits all properties and the entire behavior of A
Generalization (actor)		Y inherits from X; Y participates in all use cases in which X participates
Extend relationship		B extends A: optional incorporation of use case B into use case A
Include relationship		A includes B: required incorporation of use case B into use case A

Summary: Class Diagram

Name	*Notation*	*Description*
Class		Description of the structure and behavior of a set of objects
Abstract class		Class that cannot be instantiated
Association		Relationship between classes: navigability unspecified (a), navigable in both directions (b), not navigable in one direction (c)
N-ary association		Relationship between *N* (in this case 3) classes
Association class		More detailed description of an association
xor relationship		An object of A is in a relationship with an object of B or with an object of C but not with both
Strong aggregation = composition		Existence-dependent parts-whole relationship (A is part of B; if B is deleted, related instances of A are also deleted)
Shared aggregation		Parts-whole relationship (A is part of B; if B is deleted, related instances of A need not be deleted)
Generalization		Inheritance relationship (A inherits from B)
Object		Instance of a class
Link		Relationship between objects

UML Diagrams

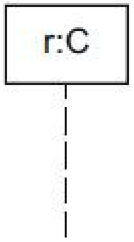


Sequence Diagram

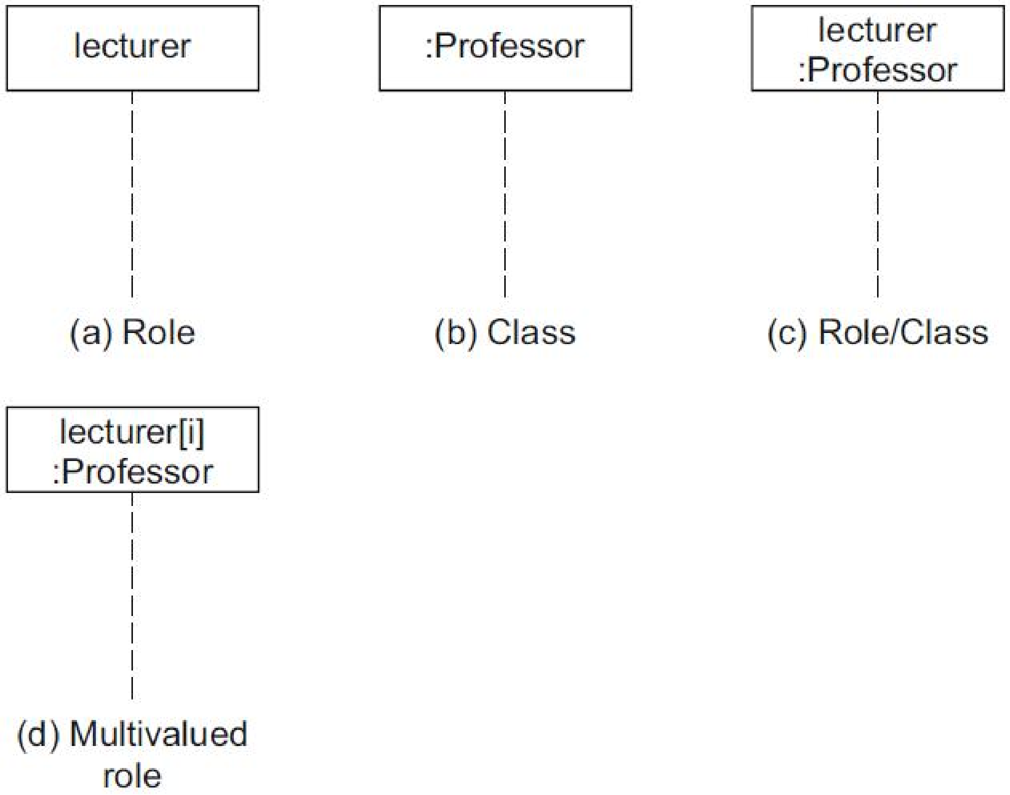
- Message among interaction partners
- Can be constructed with different granularity at different design stages
  - Interaction between the system and its environment
  - Interaction among system parts
  - Interaction between design objects

Interaction Partners

- Lifeline
  - r: role
  - C: class



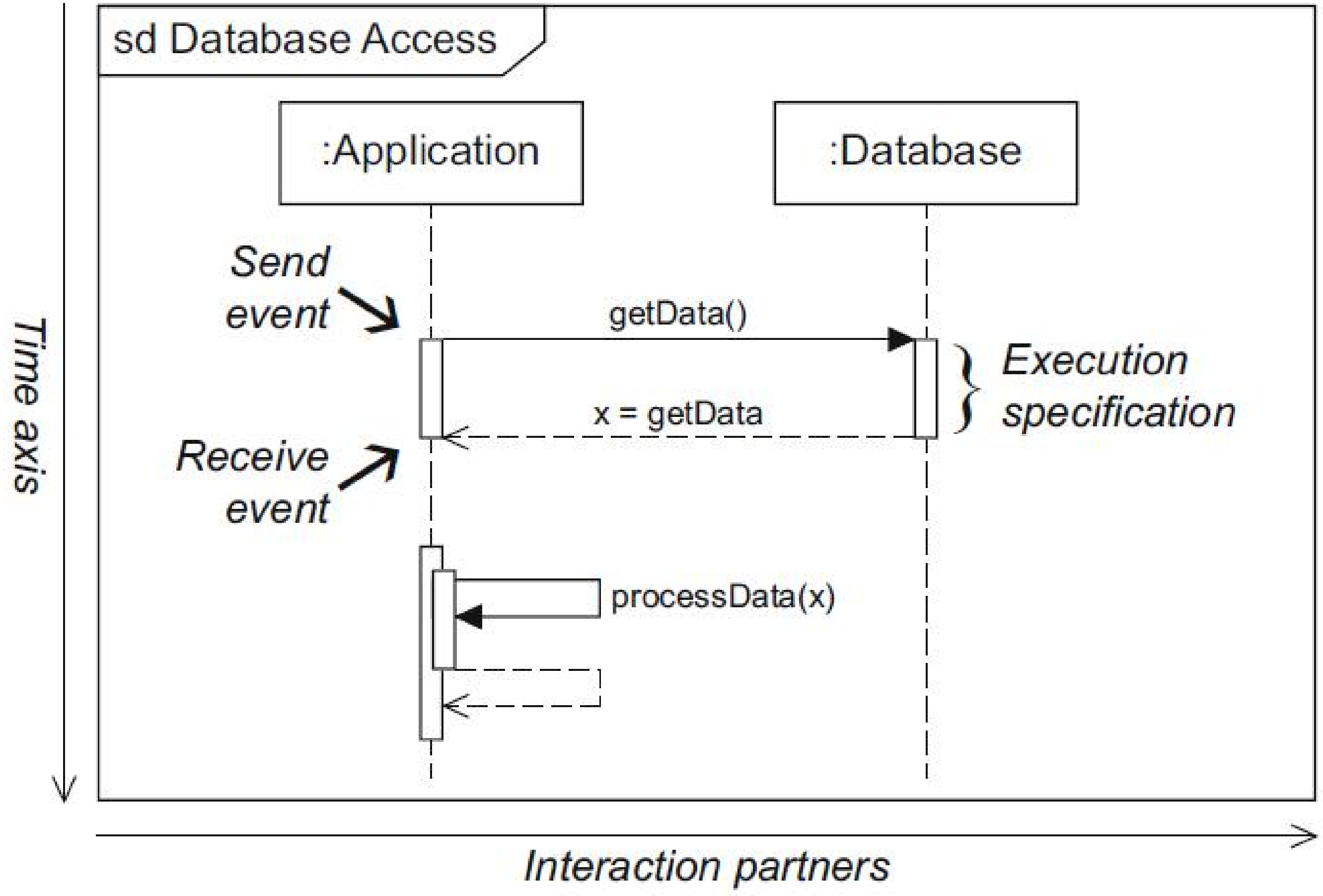
- Use roles instead of objects
  - Each object can play different roles



Message Exchange

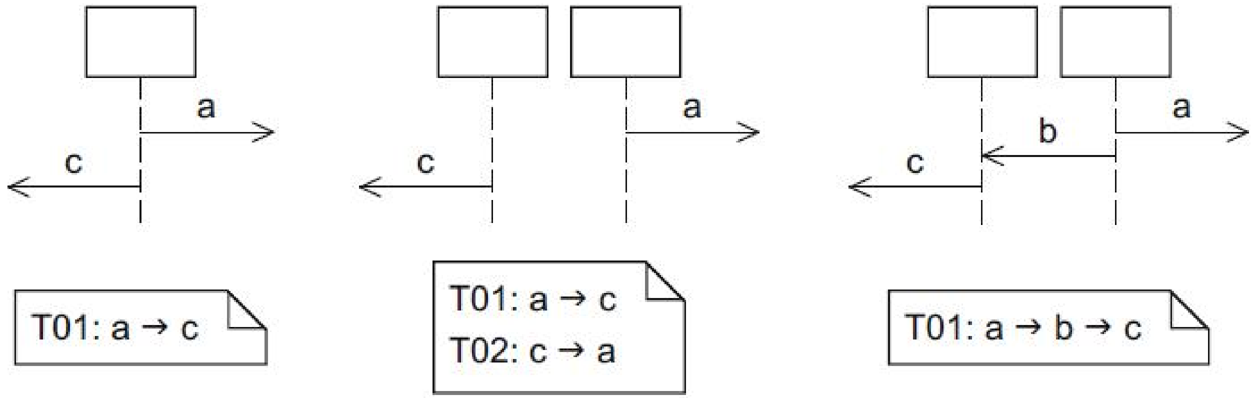
- Two dimensions
  - Time
  - Interaction partners

- Execution specification
  - Self message



Message Exchange: Order

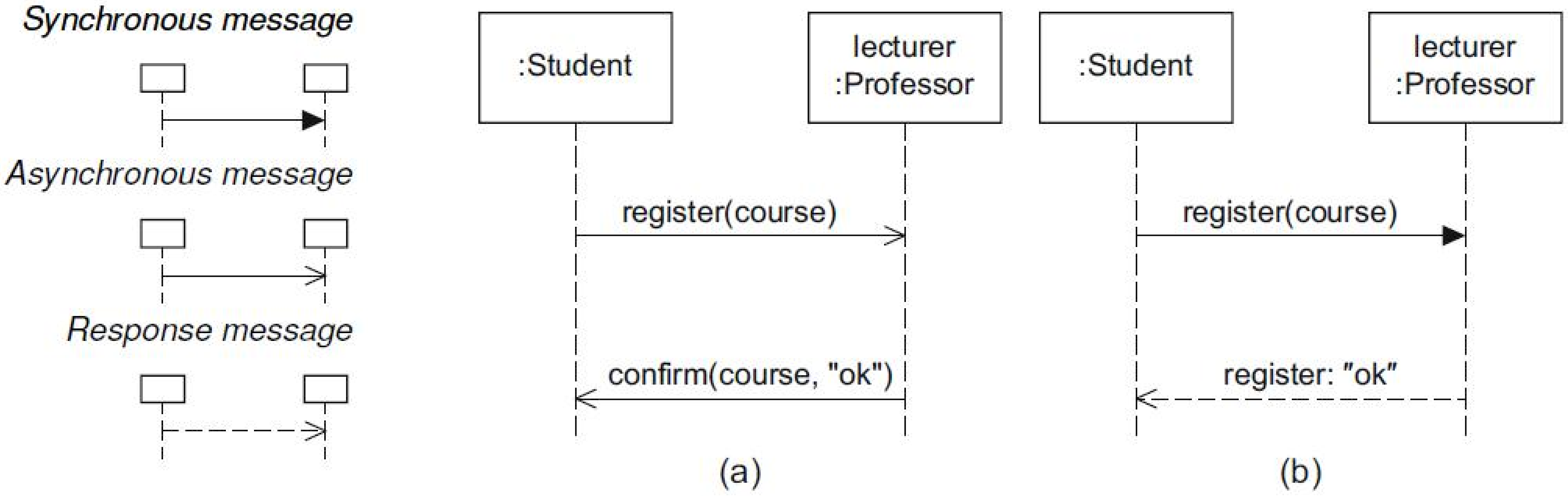
- Message order is chronical if messages on the same lifeline
  - It's a transitive relationship



(a) (b) (c)

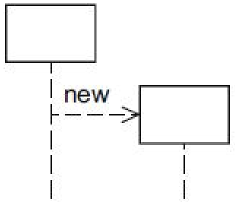
Message Exchange: Types

- (a) Register a course via email
- (b) Register a course in person

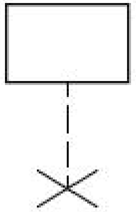


Special Messages

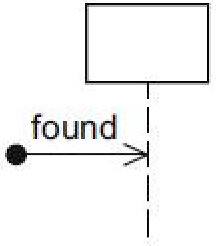
- Create message
  - Creating new object



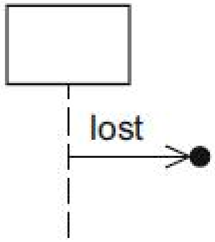
- Destruction event
  - Destruction of an object



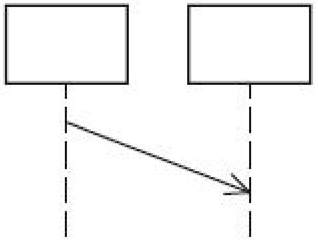
- Found message
  - Unknown/irrelevant sender



- Lost message
  - Unknown/irrelevant receiver

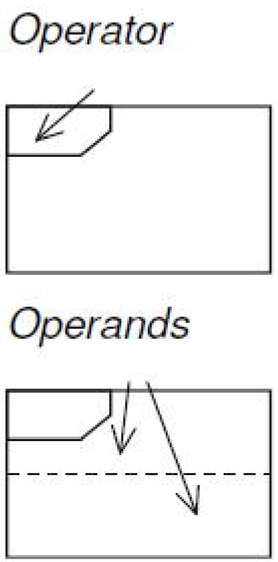


- Time-consuming message



Combined Fragments

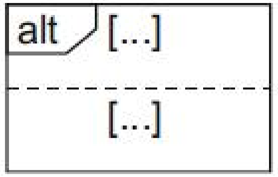
- Each operand has a guard



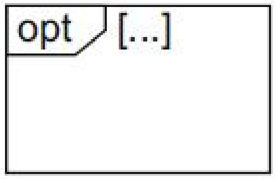
	Operator	Purpose
Branches and loops	alt	Alternative interaction
	opt	Optional interaction
	loop	Iterative interaction
	break	Exception interaction
Concurrency and order	seq	Weak order
	strict	Strict order
	par	Concurrent interaction
	critical	Atomic interaction

Branches and loops

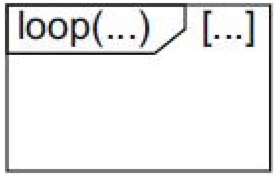
- Alternative interactions
  - If-else



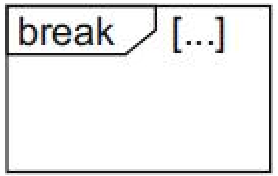
- Optional interactions
  - "if" without an "else"

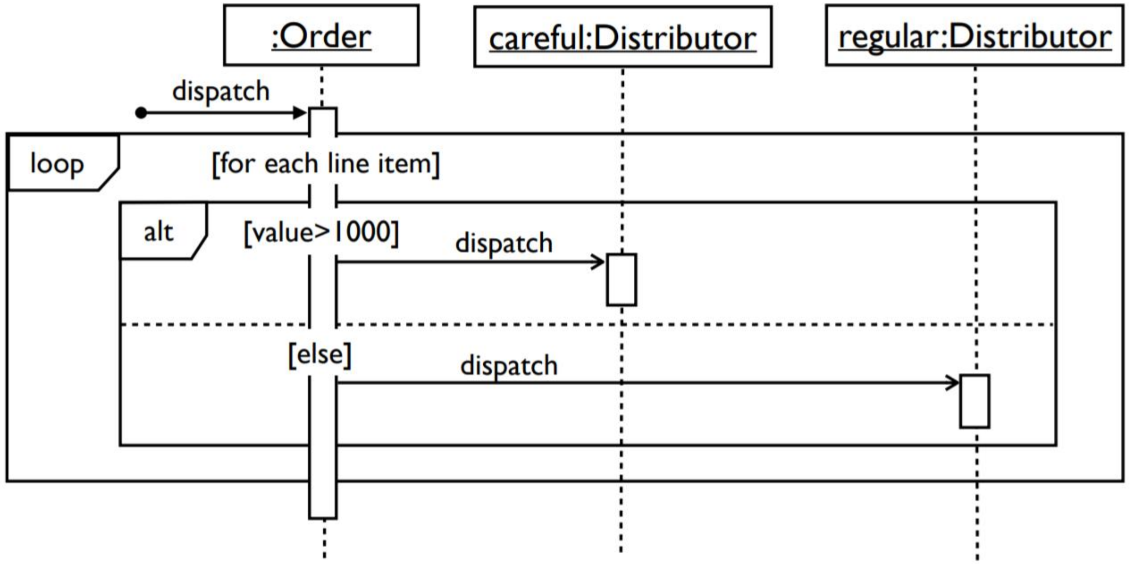


- Iterative interactions
  - For loop



- Exception interactions
  - Omit the remaining

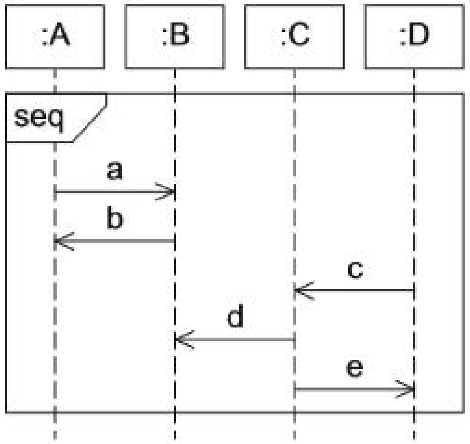




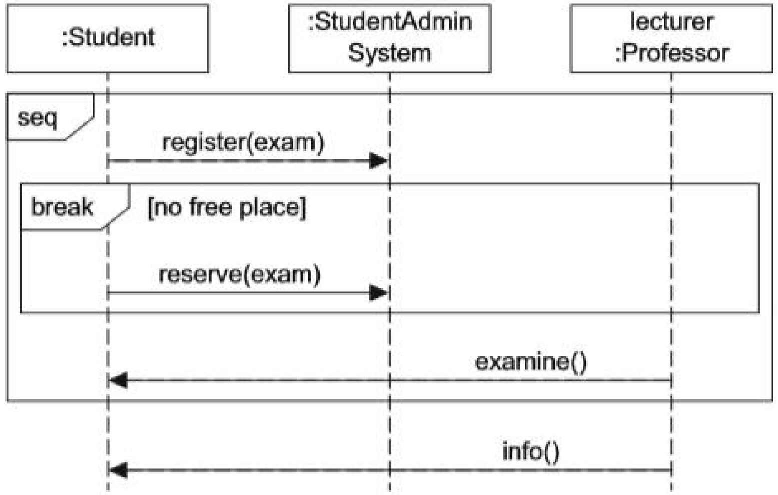
Concurrency and Order

- Seq fragment
  – Weak order
  – The ordering of events within each of the operands is maintained in the result.
  – Events on different lifelines from different operands may come in any order.
  – Events on the same life line from different operands are ordered such that an event of the first operand comes before that of the second operand.

a->b->d
c->d->e



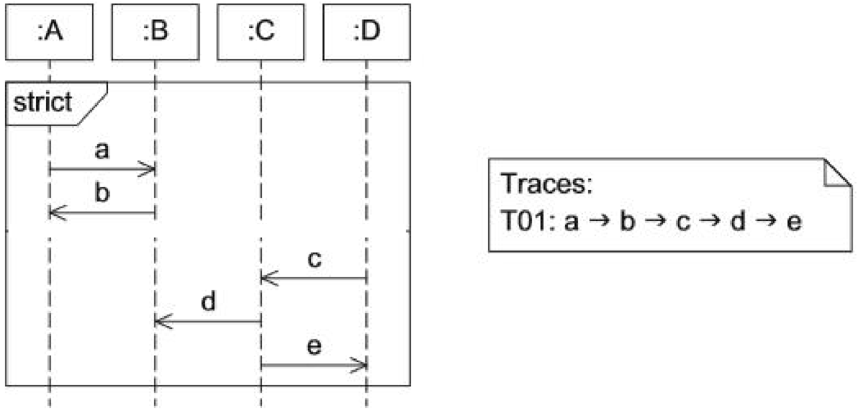
Traces:
T01: a → b → c → d → e
T02: a → c → b → d → e
T03: c → a → b → d → e



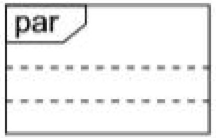
Concurrency and Order (cont.)

- Strict fragment

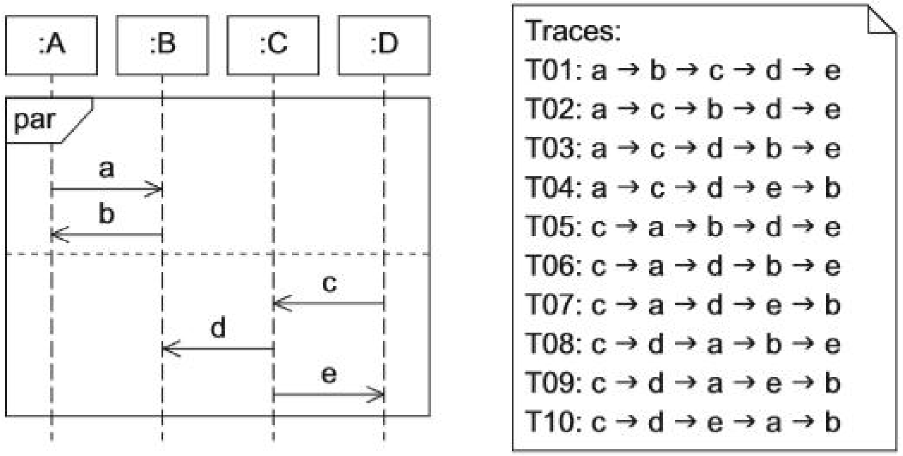
  - Strong & strict order



- Par fragment

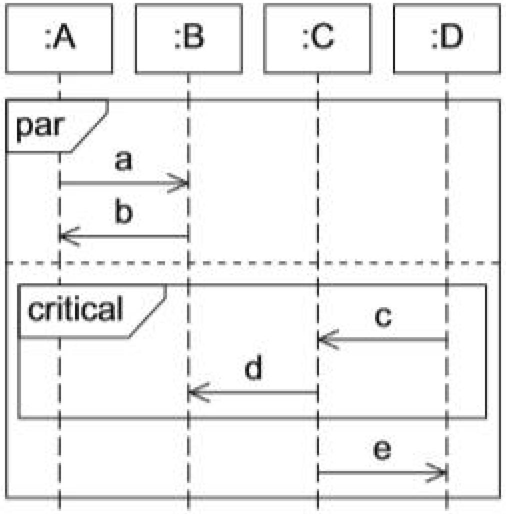


  - Order within the operands are respected
  - The order of operands does not matter



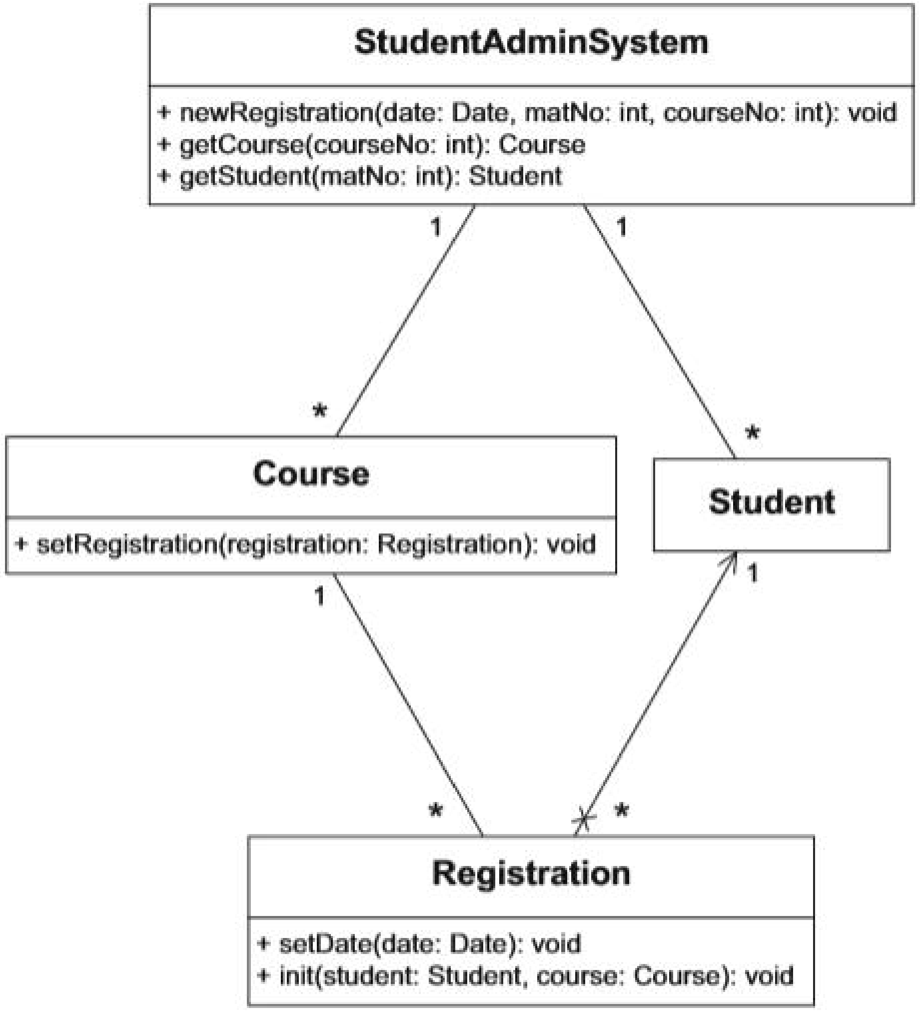
Concurrency and Order (cont.)

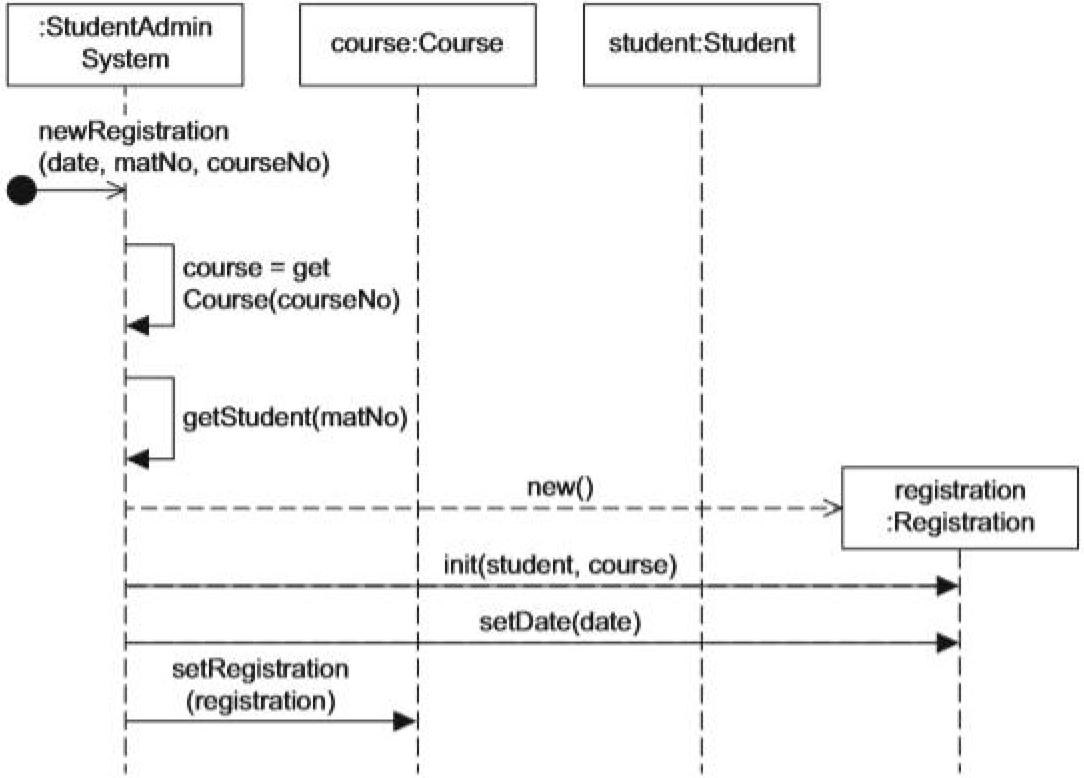
- Critical fragment
  - Atomic interaction
  - No other messages can happen during the execution

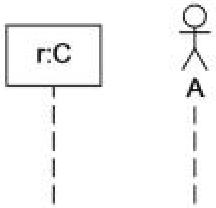
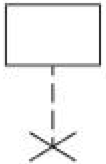
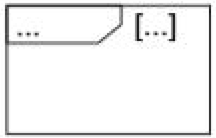
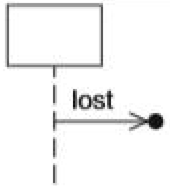
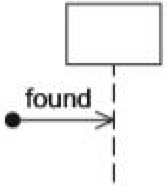


Traces:
T01: a → b → c → d → e
T02: a → c → d → b → e
T03: a → c → d → e → b
T04: c → d → a → b → e
T05: c → d → a → e → b
T06: c → d → e → a → b

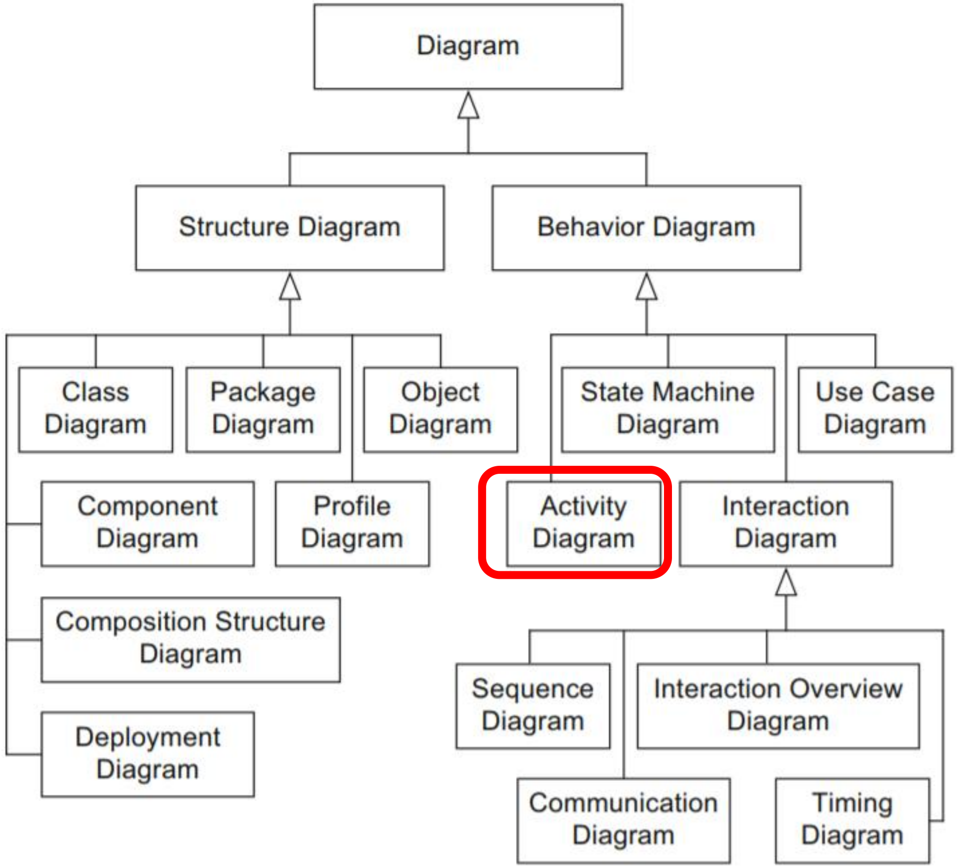
Example


StudentAdminSystem
+ newRegistration(date: Date, matNo: int, courseNo: int): void
+ getCourse(courseNo: int): Course
+ getStudent(matNo: int): Student
1
1
*
*
Course
+ setRegistration(registration: Registration): void
Student
1
1
*
*
Registration
+ setDate(date: Date): void
+ init(student: Student, course: Course): void


:StudentAdmin System
course:Course
student:Student
newRegistration (date, matNo, courseNo)
course = get Course(courseNo)
getStudent(matNo)
new()
registration :Registration
init(student, course)
setDate(date)
setRegistration (registration)

Name	*Notation*	*Description*
Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message
Asynchronous message		Sender continues its own work after sending the asynchronous message
Lost message		Message to an unknown receiver
Found message		Message from an unknown sender

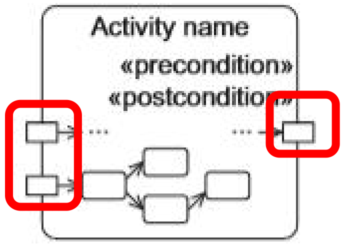
UML Diagrams



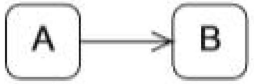
Activity Diagram

- What are the procedures of a system?
- At conceptual level: How to implement use case?
- At implementation level: How to implement an operation
- A flow-oriented language

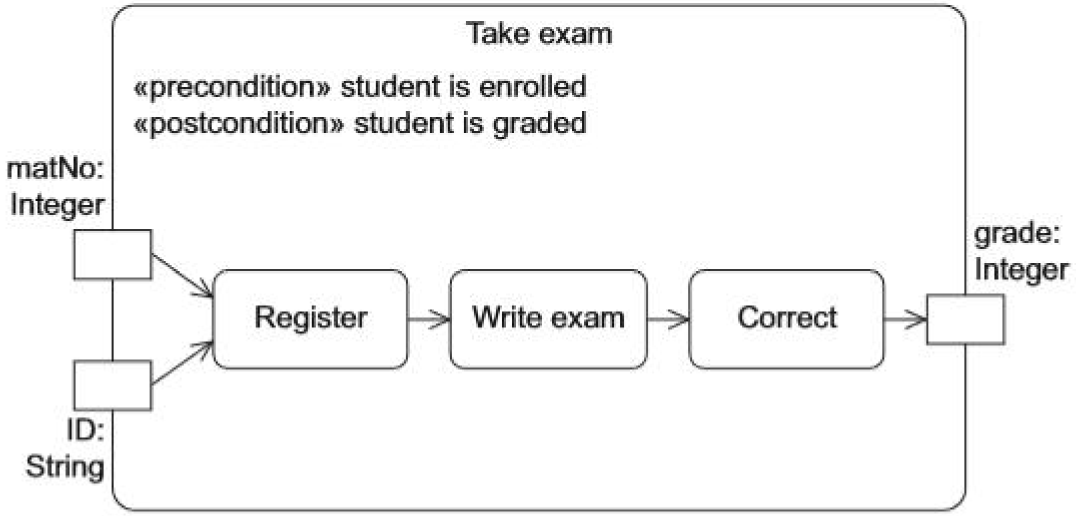
Activity Diagram: Syntax



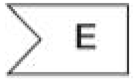
- Activity
  - Parameters
  - Precondition
  - Postcondition
  - Actions 

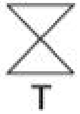
    - No language restrictions
    - Atomic: may be further broken down in other contexts
  - Edges 

    - Control flow edge: order between actions
    - Object flow edge: can exchange data

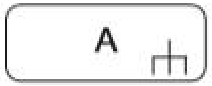


Predefined Actions

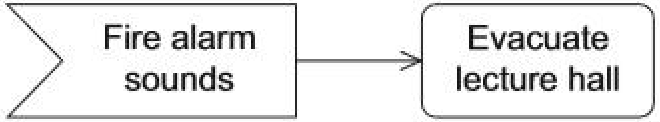
- Event-based actions
  - Accept event action 

    - Wait for a specific event E
  - Accept time event action 

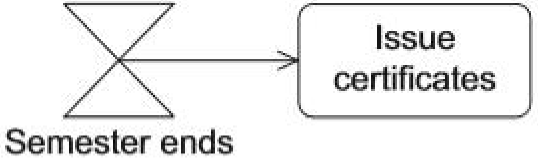
    - For time-based events
  - Send signal action 

- Call behavior actions 

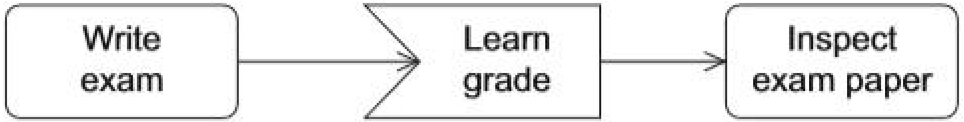
  - Calling other activities



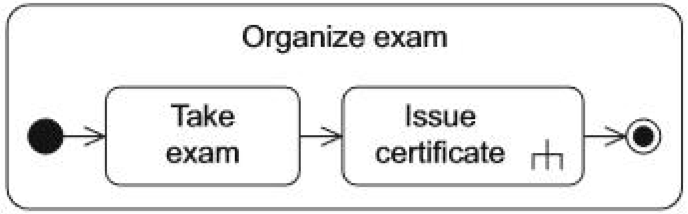
(a)

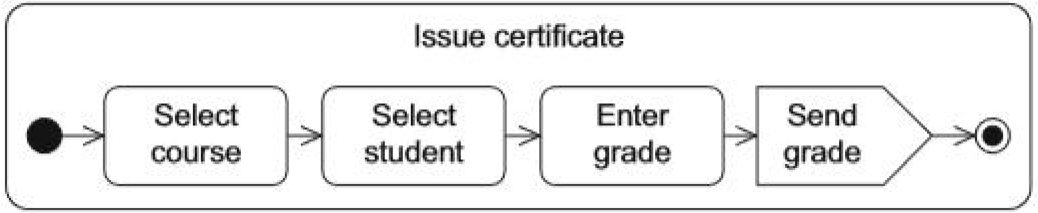


(b)



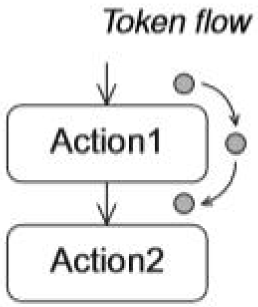
(c)





Control Flow

- Semantics for executing activities
- Token
  - Multiple incoming edges: token must be present for all incoming edges
  - Multiple outgoing edges: tokens are given to all edges



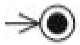
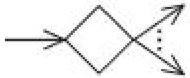
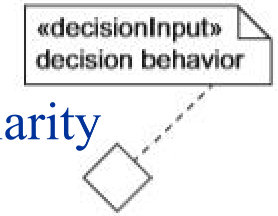
- Edge 

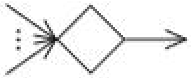
  - Guard
    - Similar to the guard definition in the State Machine diagram
  - Weight of an edge 

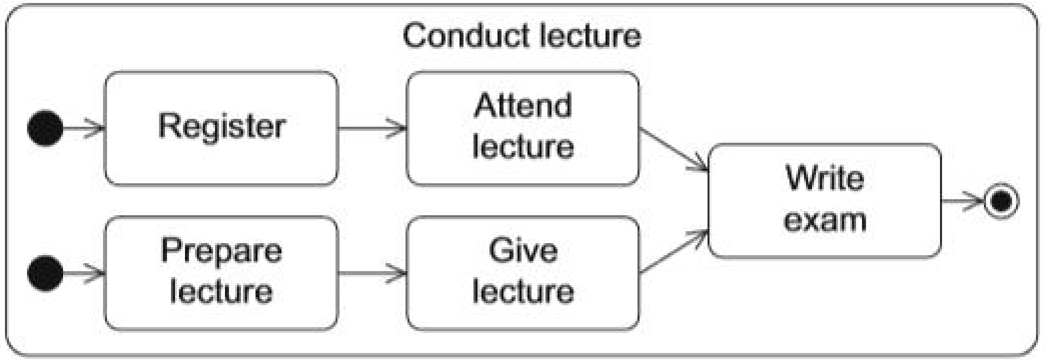
    - Tokens consumed on the edge

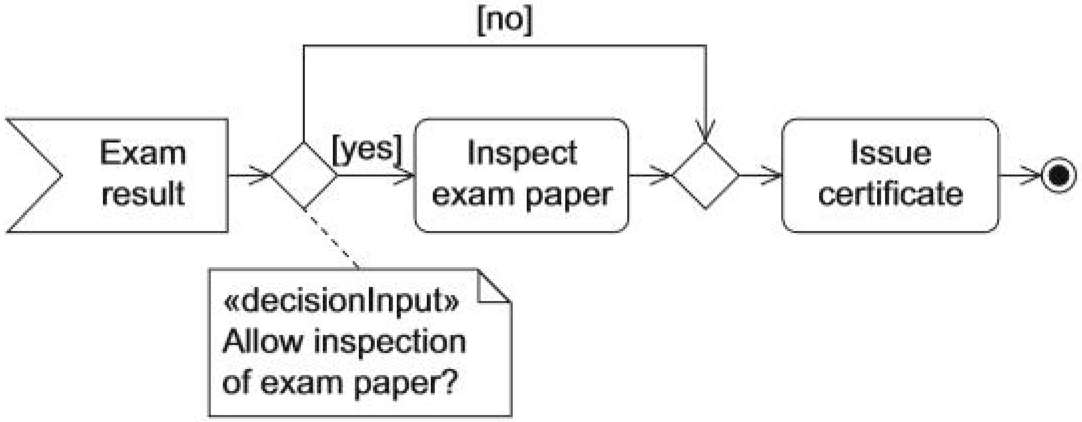
Control Flow (Cont.)

- Connector 

  - Just to make the diagram clearer
- Control nodes
  - Initial node 
  - Activity final node 
  - Decision node 
  - Decision behavior 

    - Save space & provide clarity
  - Merge node 

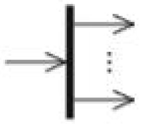




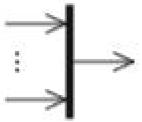
Control flow (cont.)

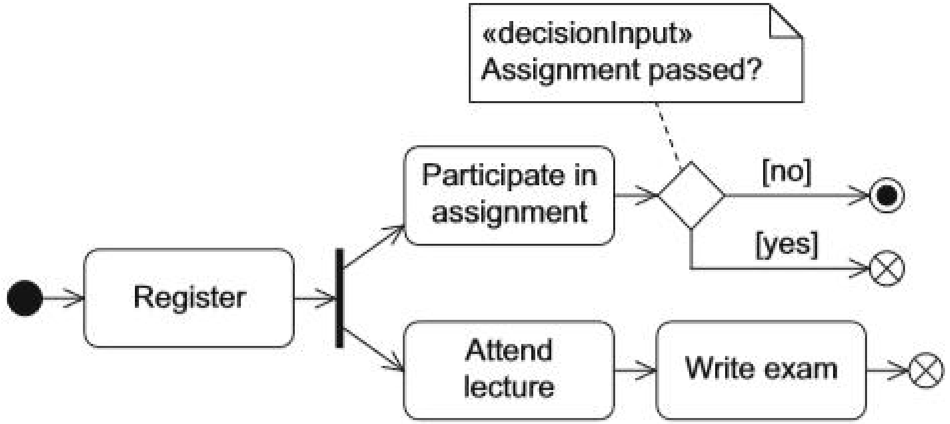
- Parallelization & Synchronization node
  - Decision node can take only one edge
- Activity final node 
  - Multiple final node: first reached final node terminates the activity
- Flow final node 
  - For concurrent activities only
  - Only terminate one concurrent path

Parallelization node



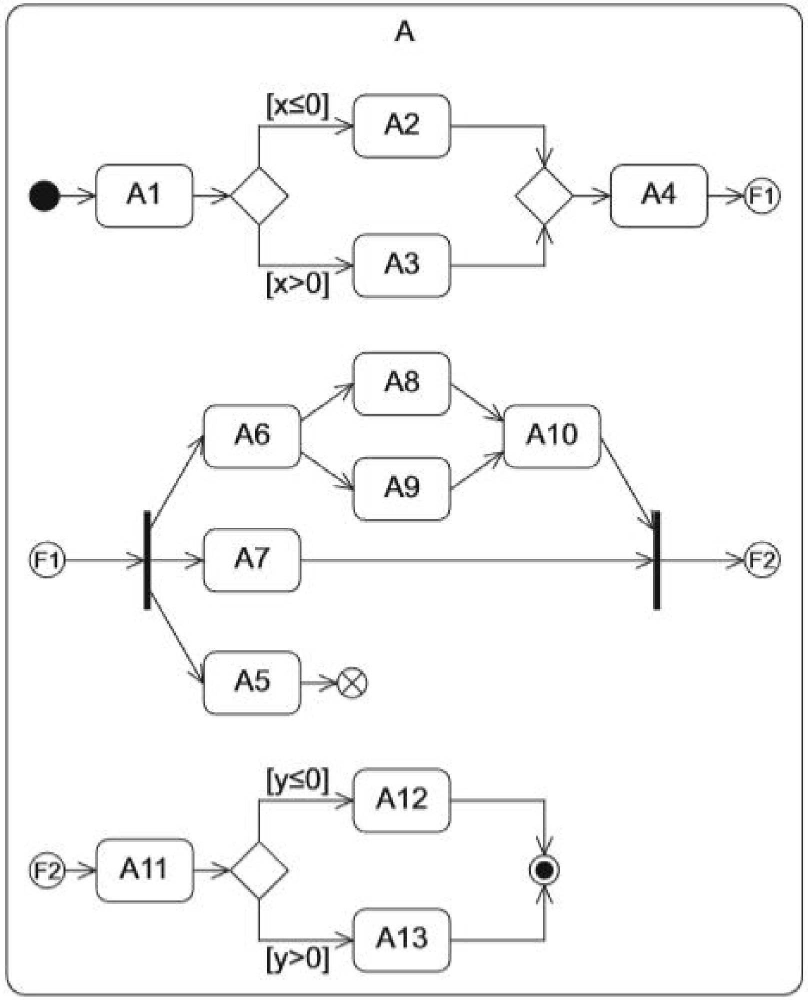
Synchronization node





Example: Control flow

- If A5 is still executing when the activity final node is reached, A5 is interrupted

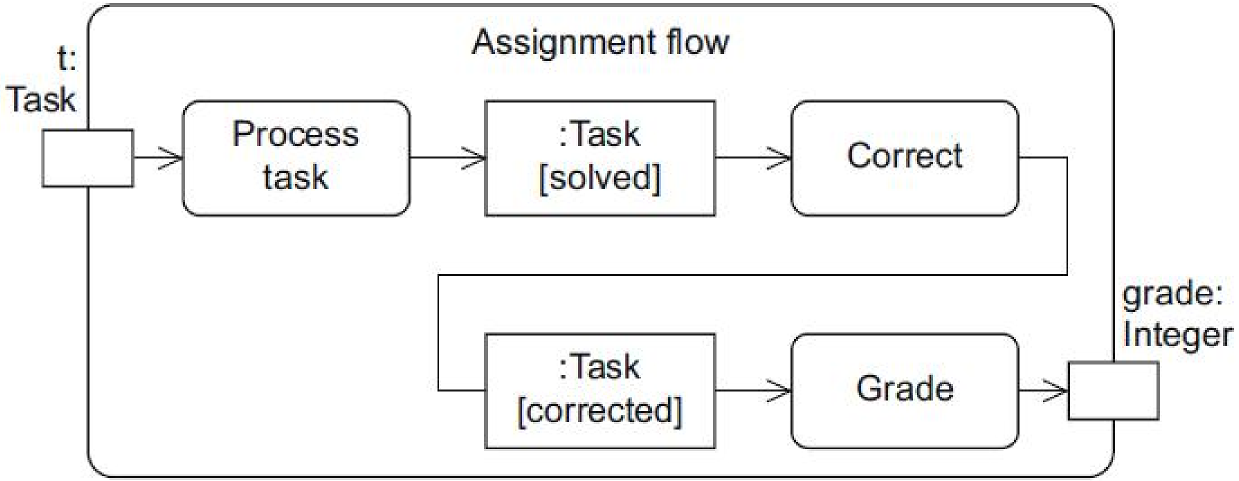


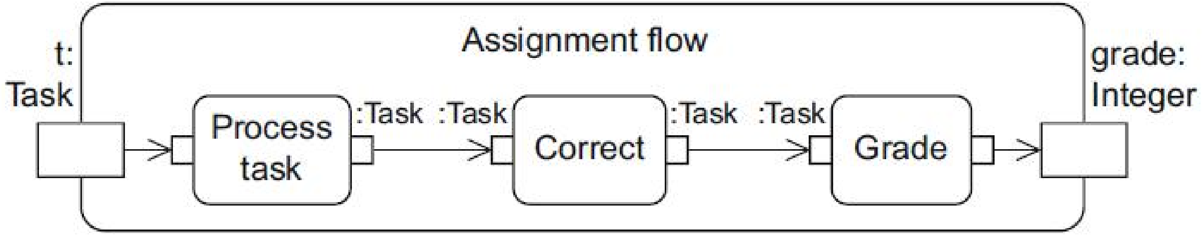
Object Flow: Syntax

- Exchange of data among actions
- Object 

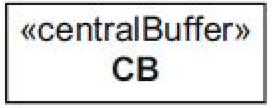
- Pin notation 

  - Parameters are also objects



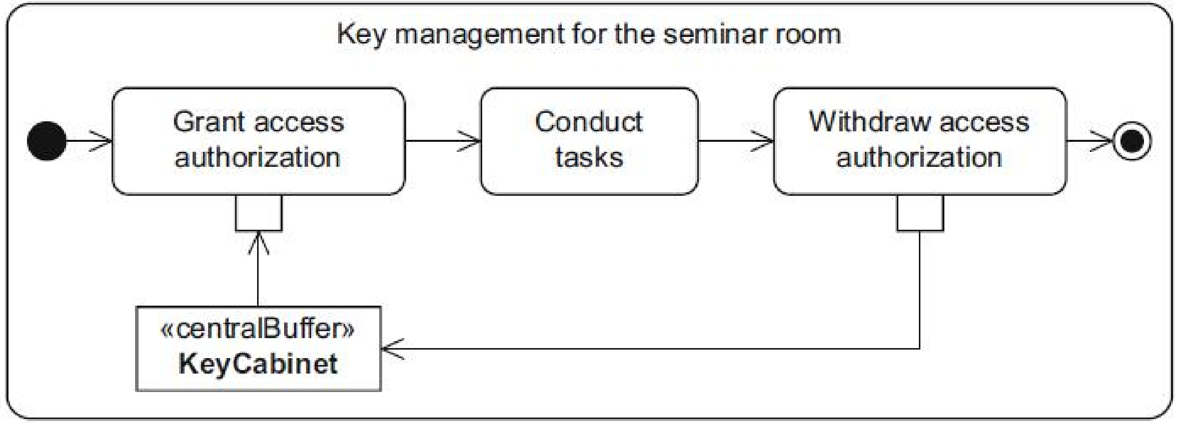


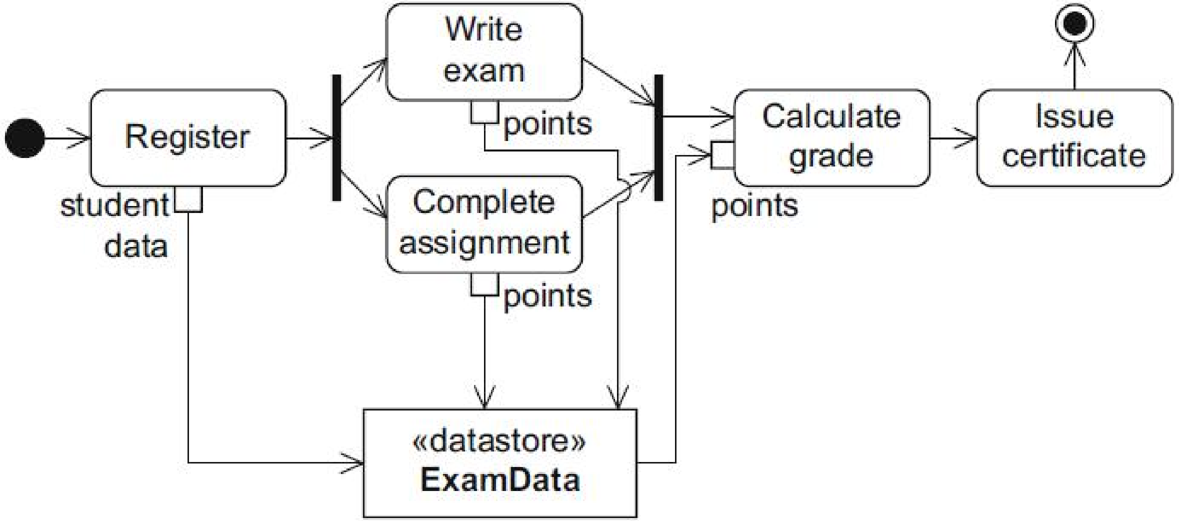
Object Flow: Syntax (cont.)

- Central buffer 

  - Data exited the central buffer is no longer in there
- Data store 

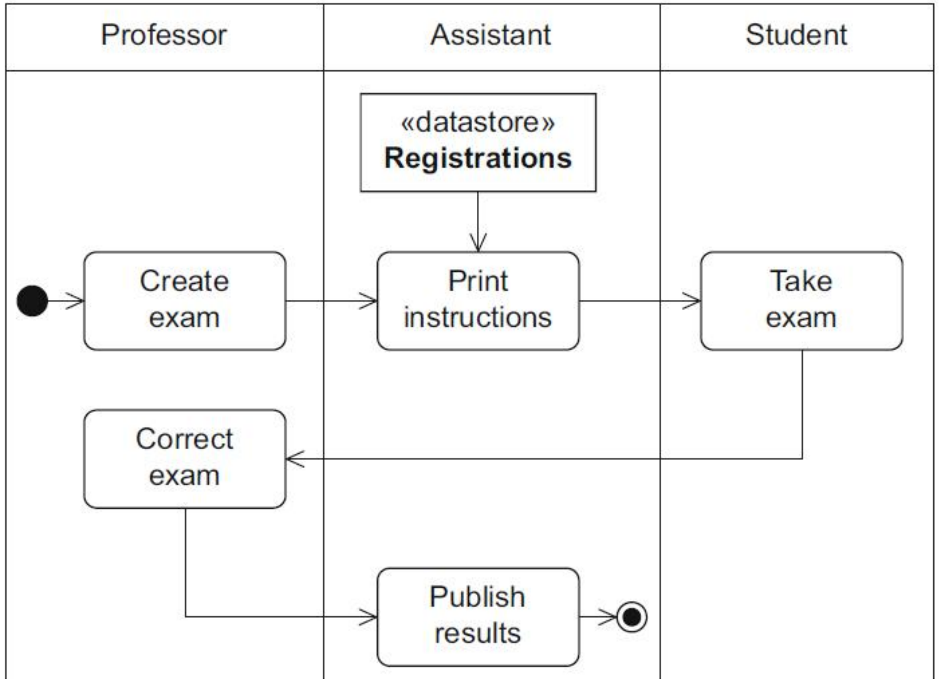
  - Data exited the data store still has a copy in there





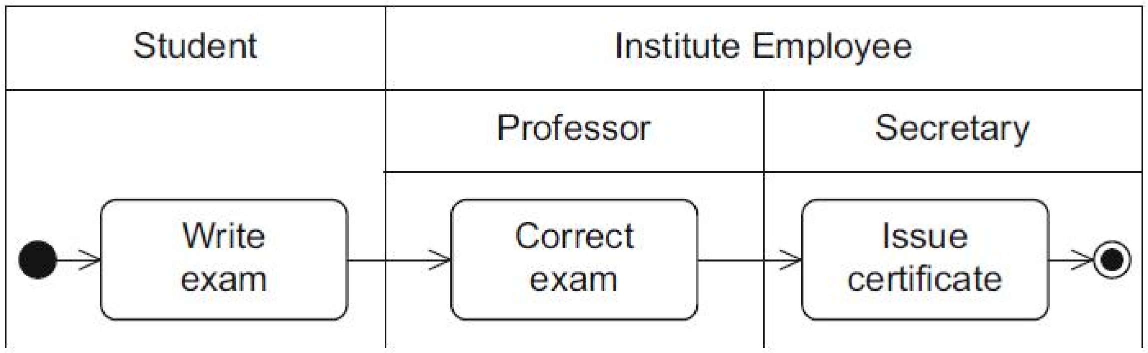
Swimlane/Partition

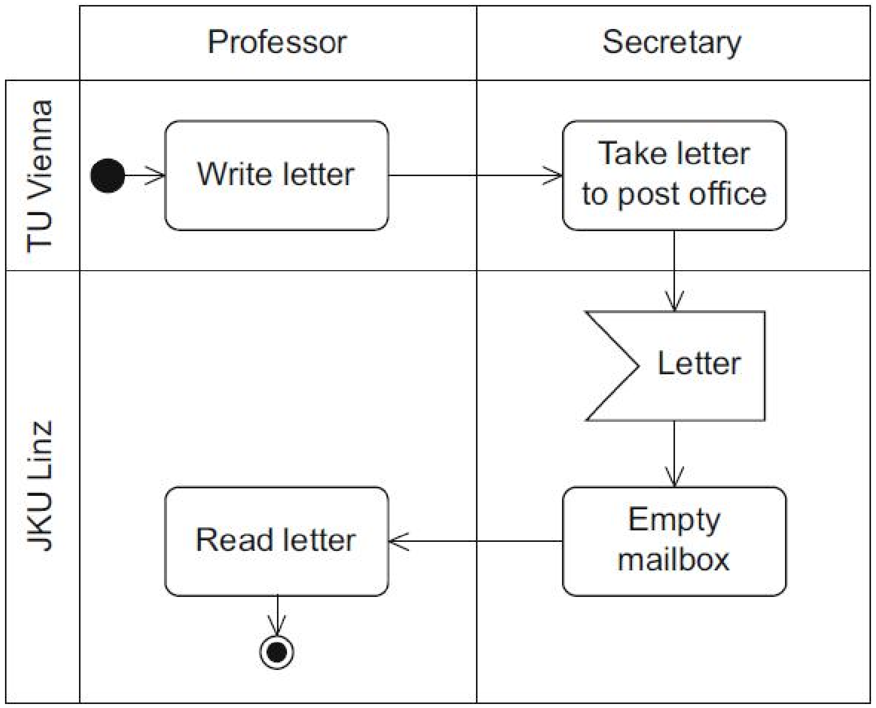
- Group actions in terms of who’s performing them
- A much clearer view of the activity diagram



Swimlane (cont.)

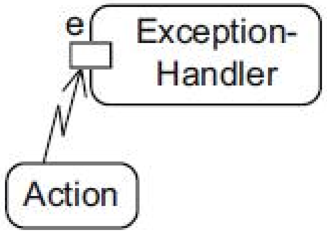
- Swim lanes can have sub-partitions
- Swim lanes can also have multiple dimensions

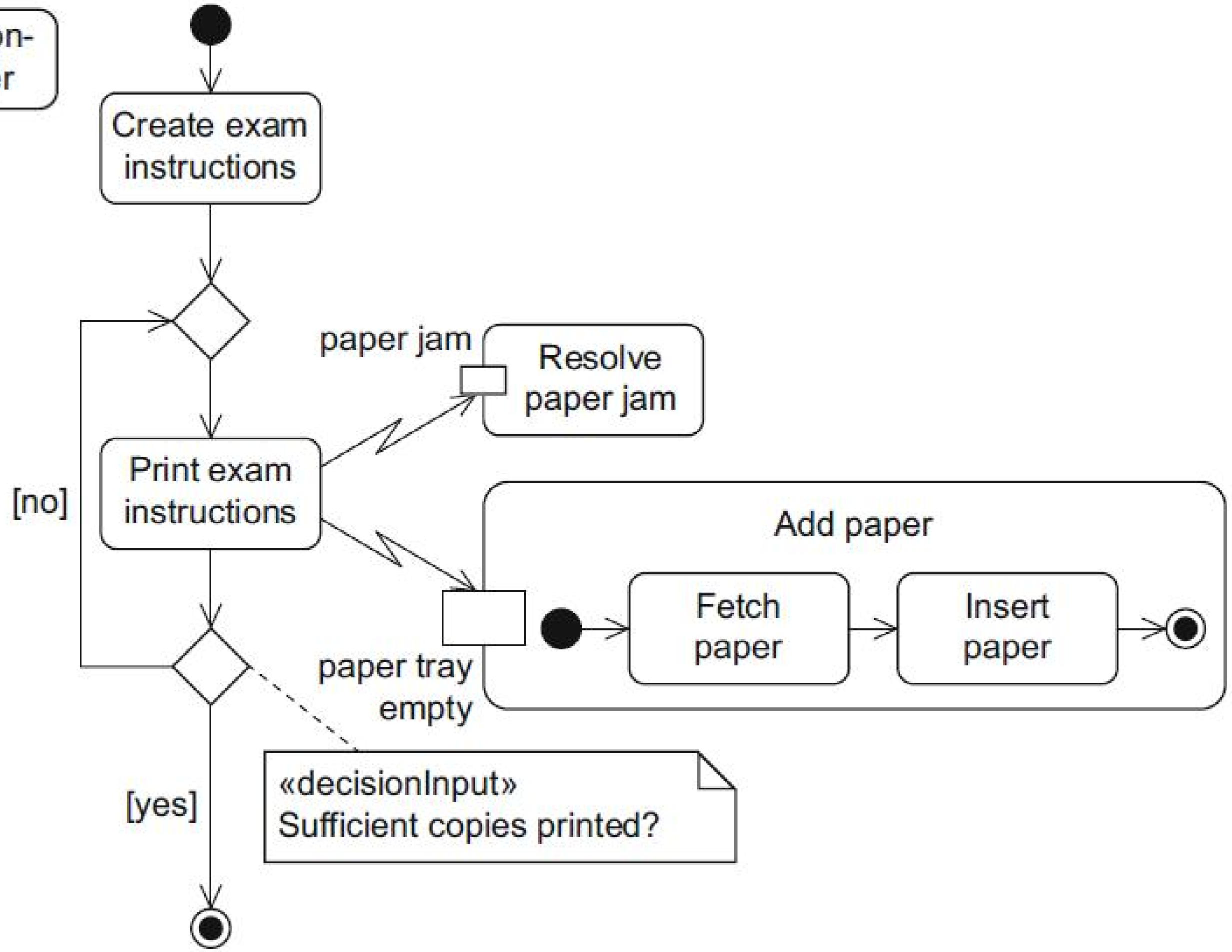




Exception Handling

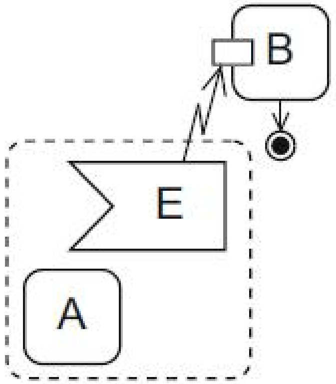
- Exception handler node
  - Error situation e
  - Execute exception handler when e happens
  - Then the sequence continues as if the action ended normally
  - One action can have multiple exception handlers

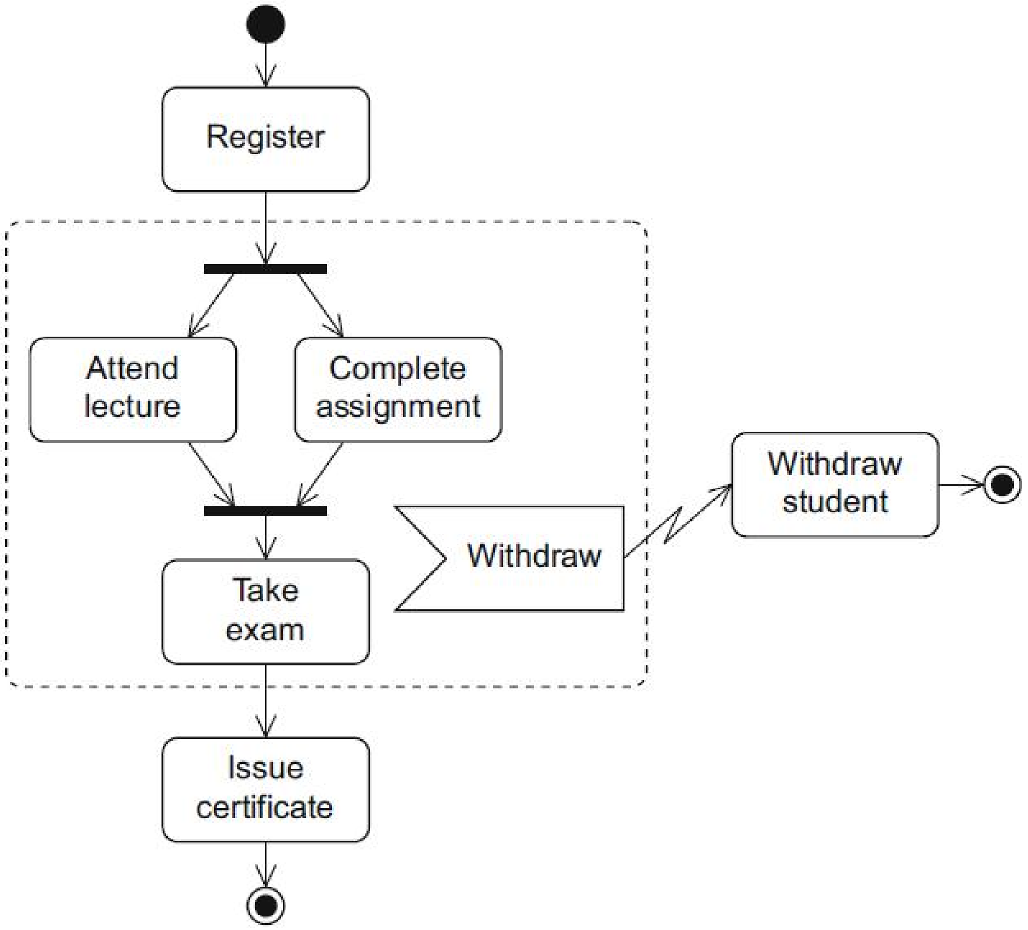




Interruptible activity region

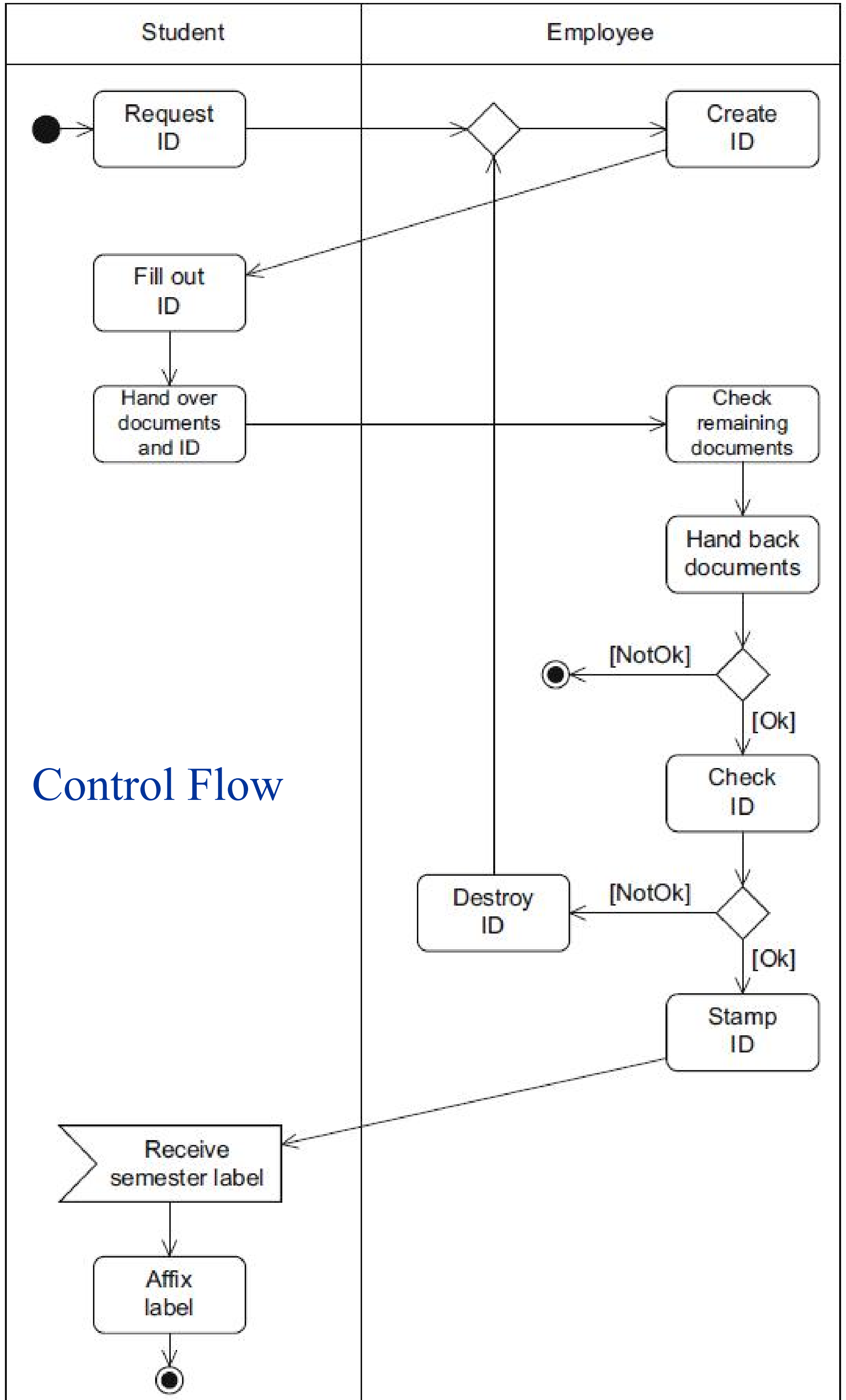
- Interruptible activity region
- Activities within the dashed region terminate immediately when E occur



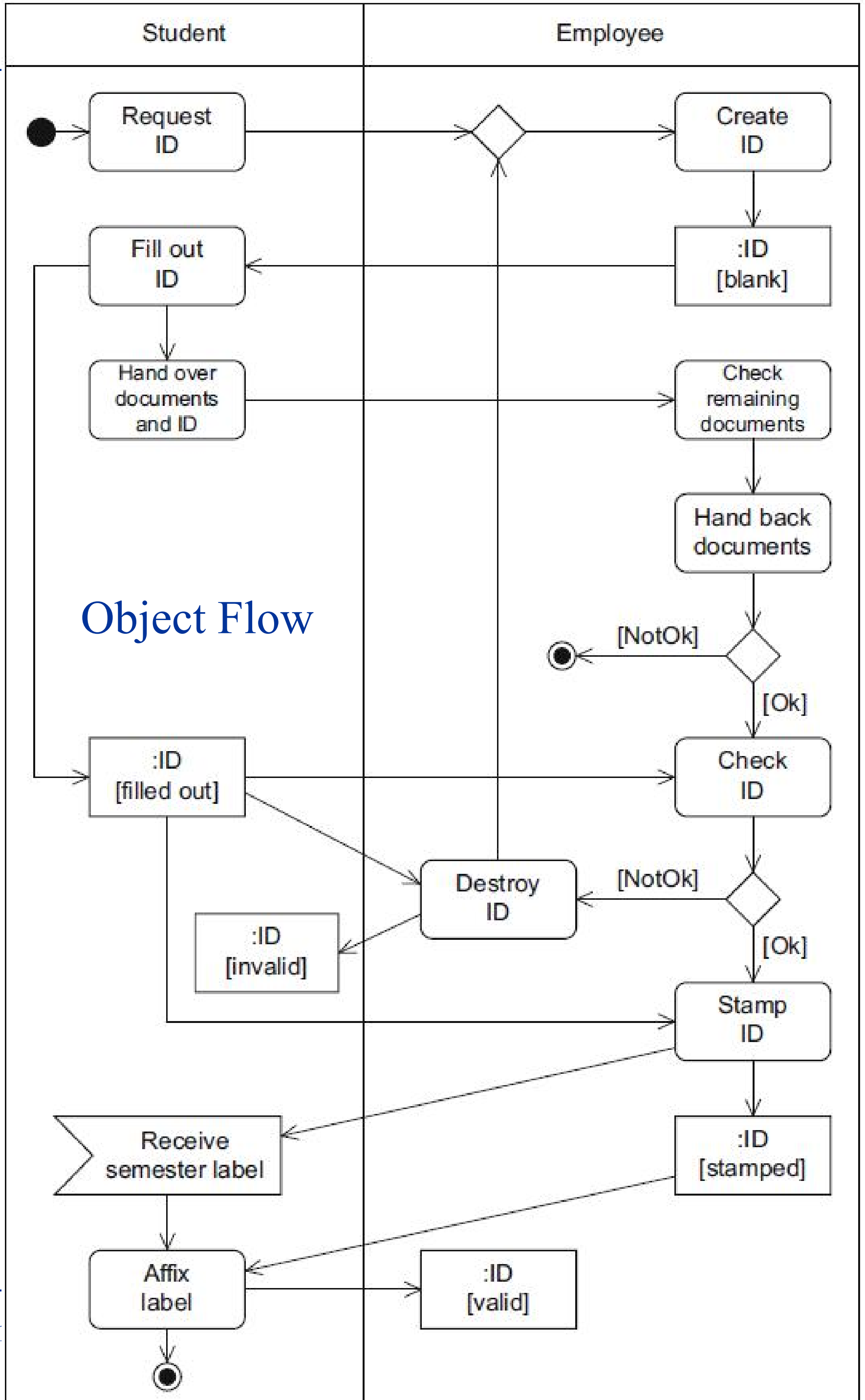


Example: Student ID

- To obtain a student ID, the student must request this ID from an employee of the student office.
- The employee hands the student the forms that the student has to fill out to register at the university.
- These forms include the student ID itself, which is a small, old-style cardboard card.
- The student has to enter personal data on this card and the employee confirms it with a stamp after checking it against certain documents.
- Once the student has filled out the forms, the student returns them to the employee in the student office and hands over documents such as photo identification, school-leaving certificate, and birth certificate.
- The employee checks the documents. If the documents are incomplete or the student is not authorized to receive a student ID for the university, the process is terminated immediately.
- If the documents are all in order, the employee checks whether the student has filled out the student ID correctly.
- If there are any errors, this ID is destroyed and the student has to fill out another one. Otherwise the ID is stamped.
- However, the student ID is not valid until it bears the semester label sent to the student by post.



Control Flow



Object Flow

Name	*Notation*	*Description*
Action node	Action	Actions are atomic, i.e., they cannot be broken down further
Activity node	Activity	Activities can be broken down further
Initial node	●	Start of the execution of an activity
Activity final node	◉	End of ALL execution paths of an activity
Flow final node	⊗	End of ONE execution path of an activity
Decision node		Splitting of one execution path into two or more alternative execution paths
Merge node		Merging of two or more alternative execution paths into one execution path
Parallelization node		Splitting of one execution path into two or more concurrent execution paths
Synchronization node		Merging of two or more concurrent execution paths into one execution path
Edge	A → B	Connection between the nodes of an activity
Call behavior action	A	Action A refers to an activity of the same name
Object node	Object	Contains data and objects that are created, changed, and read
Parameters for activities	Activity	Contain data and objects as input and output parameters
Parameters for actions (pins)	Action	Contain data and objects as input and output parameters

Name	*Notation*	*Description*
Partition	A B / A B	Grouping of nodes and edges within an activity
Send signal action	S	Transmission of a signal to a receiver
Asynchronous accept (time) event action	E or T	Wait for an event E or a time event T
Exception handler	e Exception-Handler, Action	Exception handler is executed instead of the action in the event of an error e
Interruptible activity region	B, E, A	Flow continues on a different path if event E is detected

Why do we need models?

- Prediction
  - We know the low-level mechanisms but we want to understand how they affect higher-level behaviors
  - Use simulation instead of testing on the real system
- Explain the data
  - Make assumptions and use our knowledge to explain mechanisms that we don’t understand
- Classification
  - i.e. definitions, machine learning algorithms

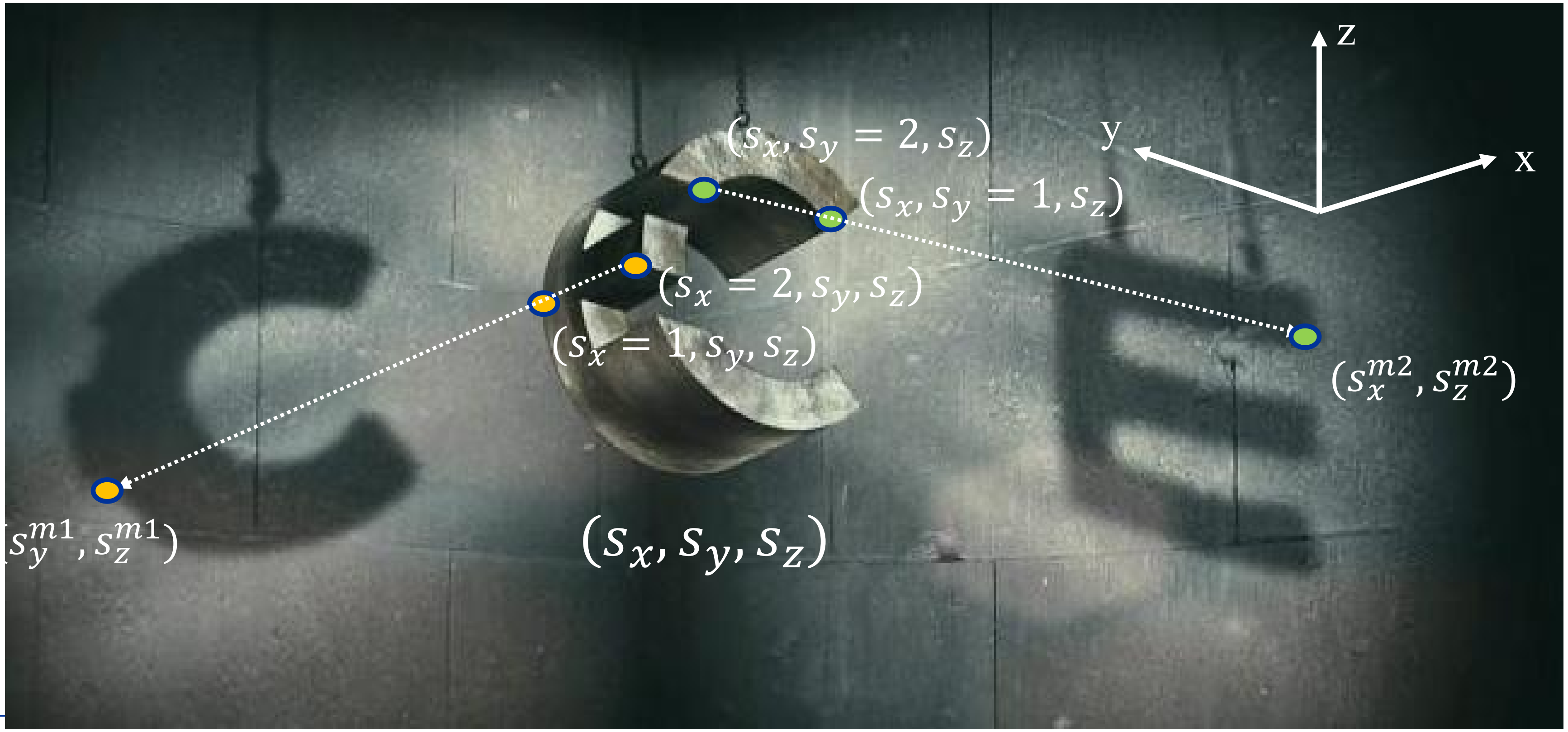
What are models?

- A system: $(S,\ I, T,\ O)$
  - S: States $s_1, s_2 \ldots s_n$
  - I: Inputs (could be $\emptyset$)
  - T: Transitions $S \times I \times S$
  - O: Observations $f(S_o),\ S_o \subseteq S$

- Model of the system $(S^m,\ I^m, T^m)$
  - S^m: Abstraction/interpolation of S
    - Much fewer state variables
  - I^{m}: abstraction of I (could be $\emptyset$)
  - T^m: Transitions $S^m \times I^m \times S^m$

Abstraction – removal of state variables

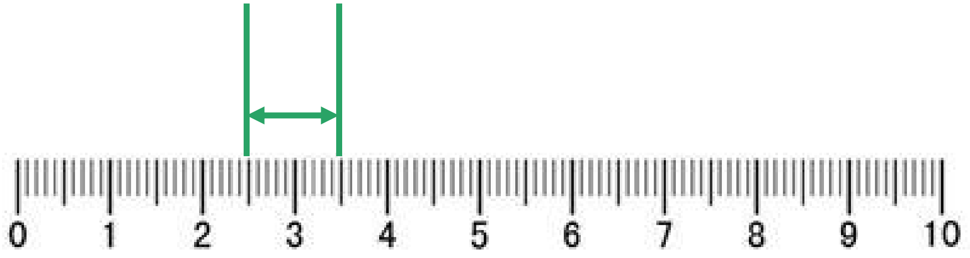
- States (s_x, s_y, s_z) *are abstracted to* (s_y^{m1}, s_z^{m1})
  - $(s_x, s_y, s_z) \rightarrow (s_y^{m1}, s_z^{m1})$
- Loss of information



Abstraction: Approximation of state variable values

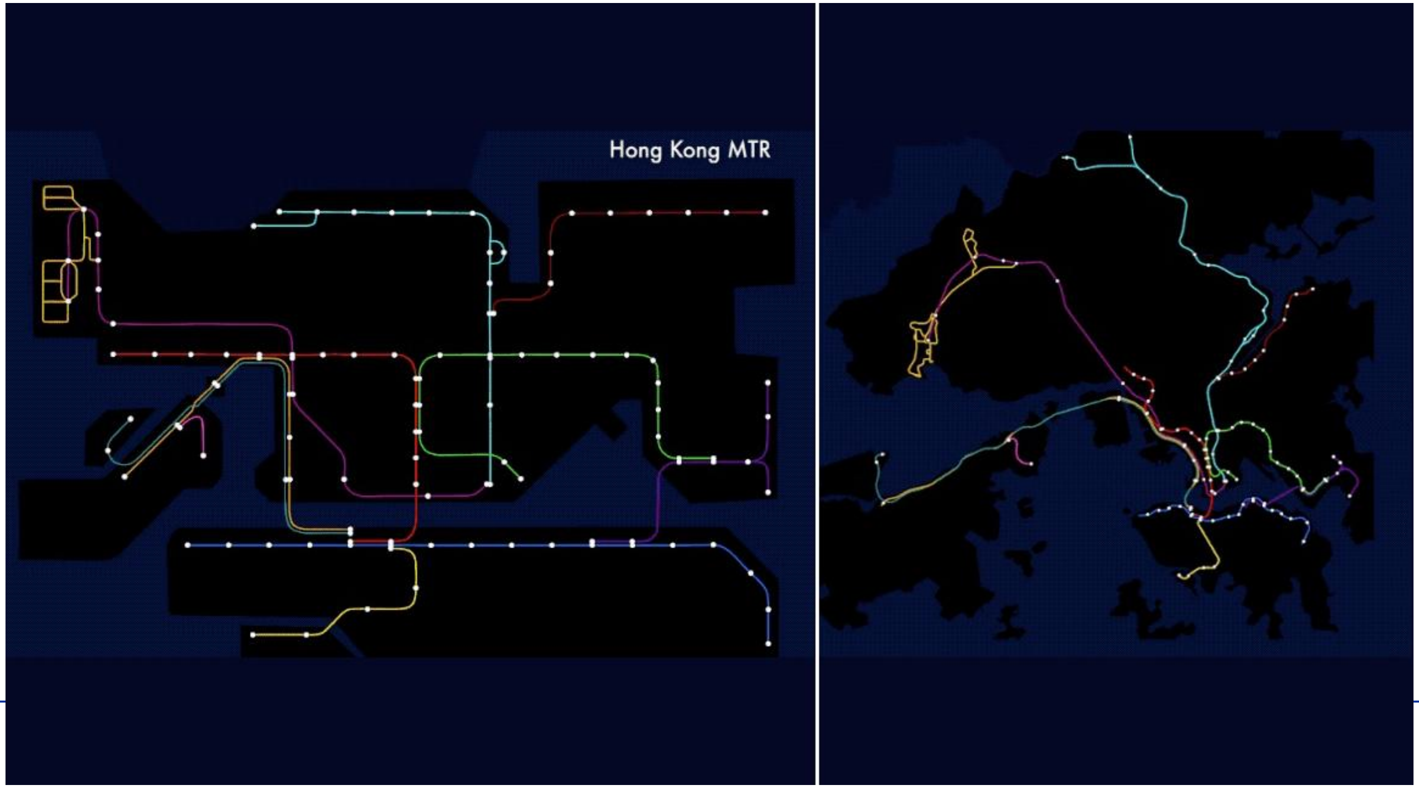
- Irrational numbers
  - $\pi \approx 3.1415$
  - $\sqrt{2} \approx 1.414$

- Approximation is another way of abstraction

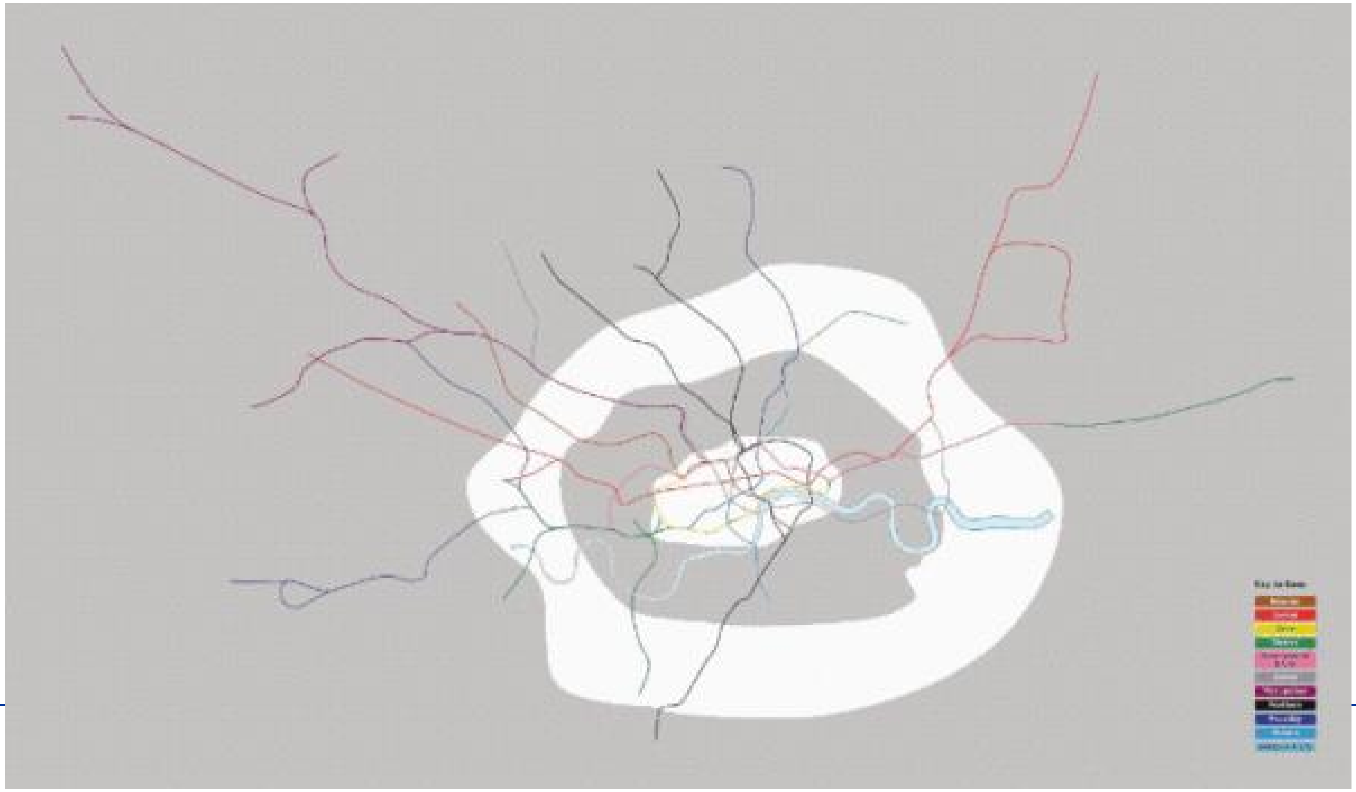


Interpolation: extracting interpretable information

- Locational information -> topological information
- $S^m = f(S_p),\ S_p \subseteq S$



More Interpolation: London MTR



What is considered as a “good” model?

- Accuracy
  - All models are wrong!
  - Error accumulates over time
  - Initial condition of the model cannot be determined due to limited observability
- Generality
  - The capability to explain not only training data, but also testing data
- Identifiability
  - Model parameters can be identified from data
- Interpretability
  - S^m are meaningful and interpretable by human

Newton vs. Einstein

- Newtonian physics is suitable for macro level objects at low speed
- $L = L_0\sqrt{1 - \frac{v^2}{c^2}}$
- A model can only be "good" within the context of its designated application
- The definition of "goodness" is changing over time

Modeling methodologies

- Bottom-up modeling
  - "White-box" model
  - Using first principles
  - Pros:
    - Interpretable
    - Convincing
  - Cons:
    - State space explosion
    - Difficult to be general
    - Low identifiability

- Data driven models (i.e. Neural networks)
  - "Black-box" model
  - From observable data
  - Pros:
    - No need to know domain knowledge
  - Cons:
    - Large and uninterpretable S^m
    - Depends highly on the quality and quantity of data