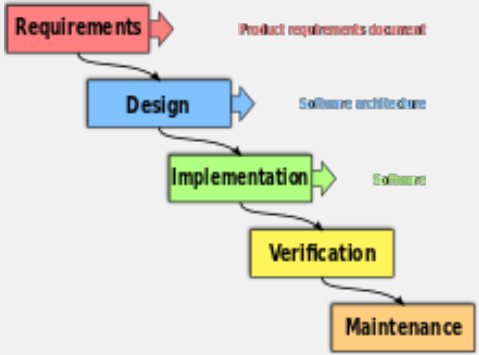
MODELING NOTATIONS

CS3213 FSE

Prof. Abhik Roychoudhury

National University of Singapore

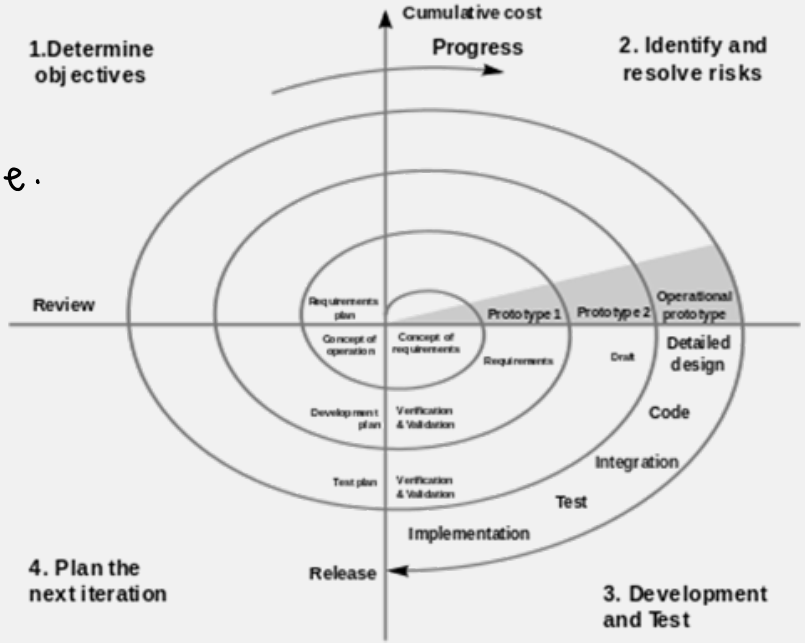
SOFTWARE ENGINEERING



Waterfall Model

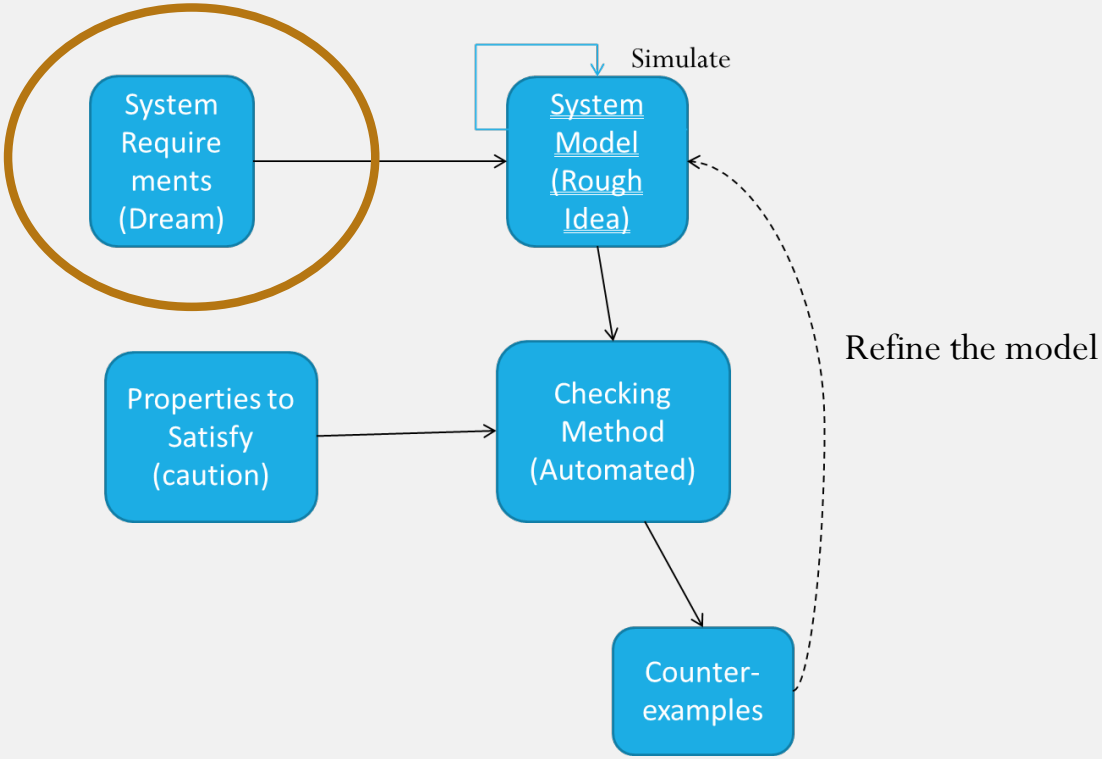
SOFTWARE ENGINEERING

Iterative.



Spiral Model

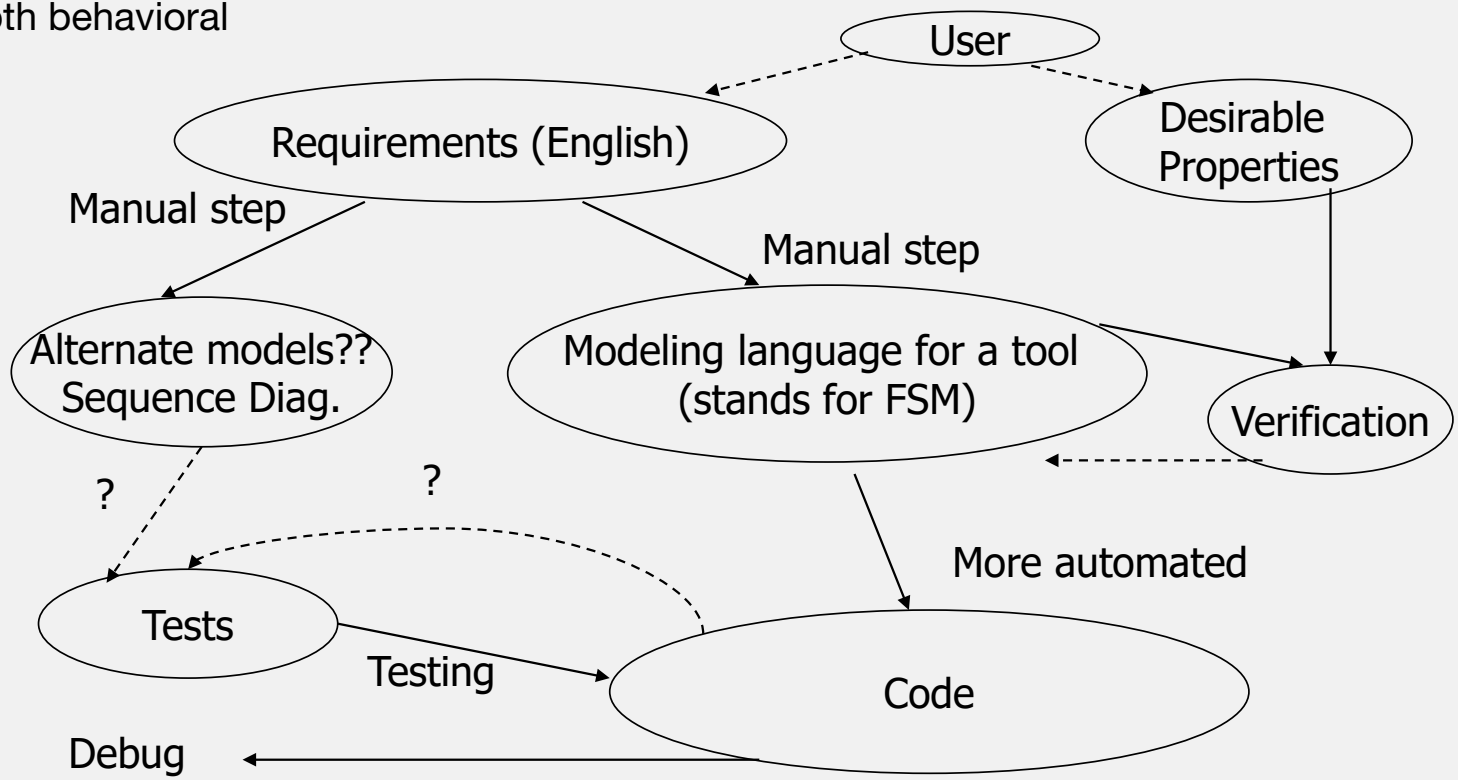
GETTING STARTED: LOOKING INSIDE



objects of different classes can interact

Seq diagram vs. FSM:
- intra-object / inter-object behavior
- both behavioral

MODEL DRIVEN SE



SYSTEM DESIGN MODEL

- We first clarify the following terms
  - System Architecture: Inter-connection among the system components.
  - System behavior: How the components change state, by communicating among themselves.

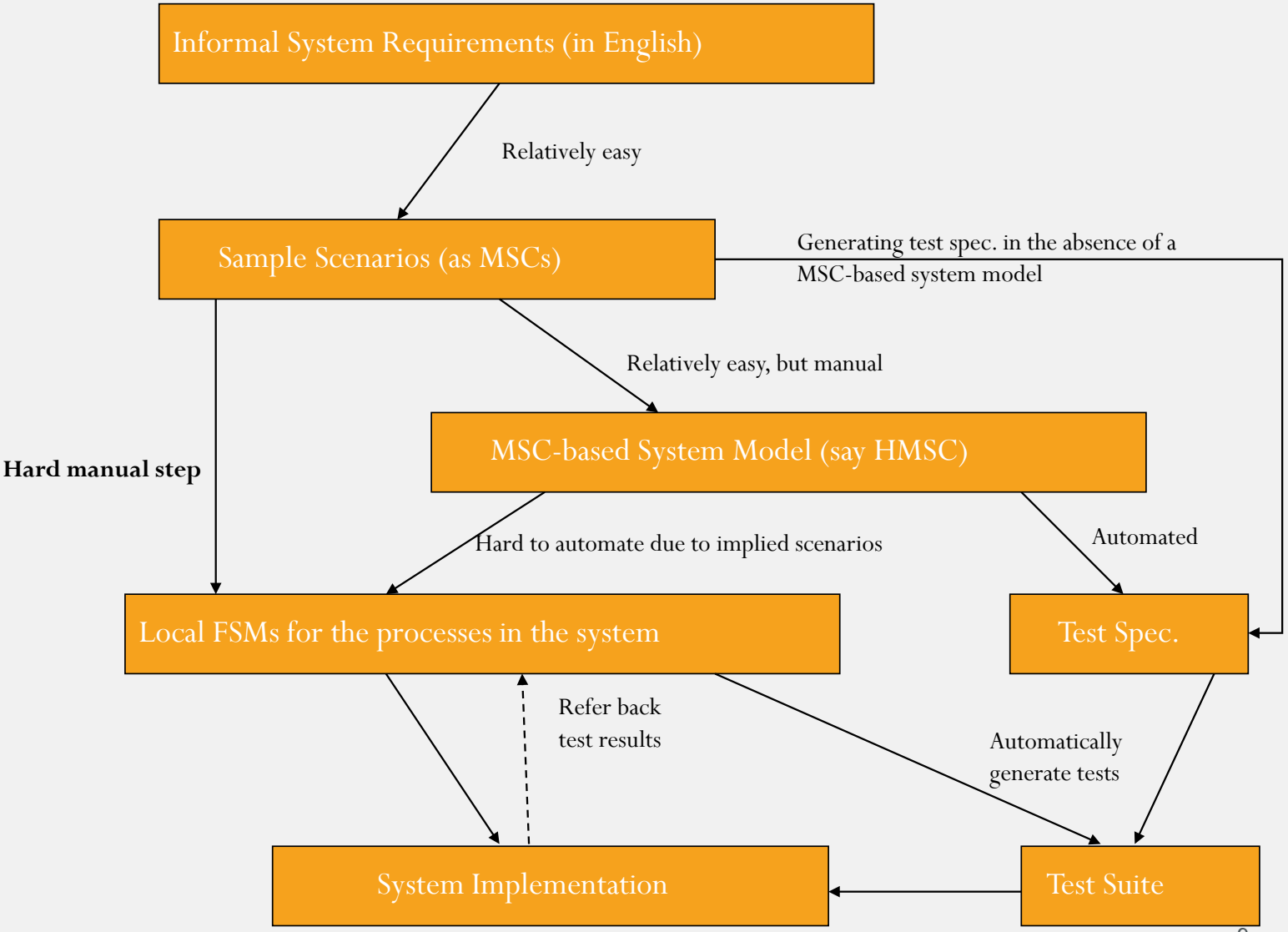
- System Design Model = Architecture + Behavior
  - We focus more on behavior, and checking of behavior.

CRITERIA FOR DESIGN MODEL

- Provides structure as well as behavior for the system components.
- Complete
  - Complete description of system behavior.
- Based on well-established modeling notations.
  - We use UML.
- Preferably executable
  - Can simulate the model, and get a feel for how the constructed system will behave!

ORGANIZATION OF SLIDES

- So Far
  - What is a Model?
  - ATC – Running Example
    - Informal Req. at a lab scale.
- Now, how to model/validate such requirements
  - Modeling Notations
    - Finite State Machines

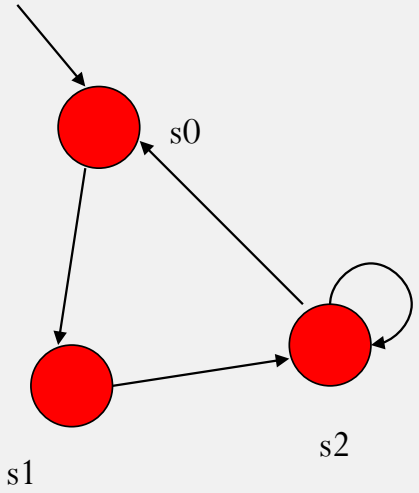

Informal System Requirements (in English)
Relatively easy
Sample Scenarios (as MSCs)
Generating test spec. in the absence of a MSC-based system model
Relatively easy, but manual
Hard manual step
MSC-based System Model (say HMSC)
Hard to automate due to implied scenarios
Automated
Local FSMs for the processes in the system
Test Spec.
Refer back test results
Automatically generate tests
System Implementation
Test Suite

FINITE STATE MACHINES

- $M = (S, I, \rightarrow)$
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\rightarrow \subseteq S \times S$ is the transition relation.

FSM: can be used to describe a class if all the objects of a class behave similarly

finite set of edges too



S = {s0, s1, s2}

I = {s0}

$\rightarrow$ = {(s0,s1), (s1,s2), (s2,s2), (s2,s0)}

ISSUES IN SYSTEM MODELING

- … using FSMs
  - Unit step: How much computation does a single transition denote?
  - Hierarchy: How to visualize a FSM model at different levels of details?
  - Concurrency: How to compose the behaviors of concurrently running subsystems (of a large sys.)
    - Each subsystem is modeled as an FSM!

WHAT IS IN A STEP?

- For hardware systems
  - A single clock cycle 1 edge = 1 clock cycle
- For software systems
  - Atomic execution of a "minimal" block of code
    - A statement or an instruction?
    - Depends on the level at which the software system is being modeled as an FSM !

EXAMPLE

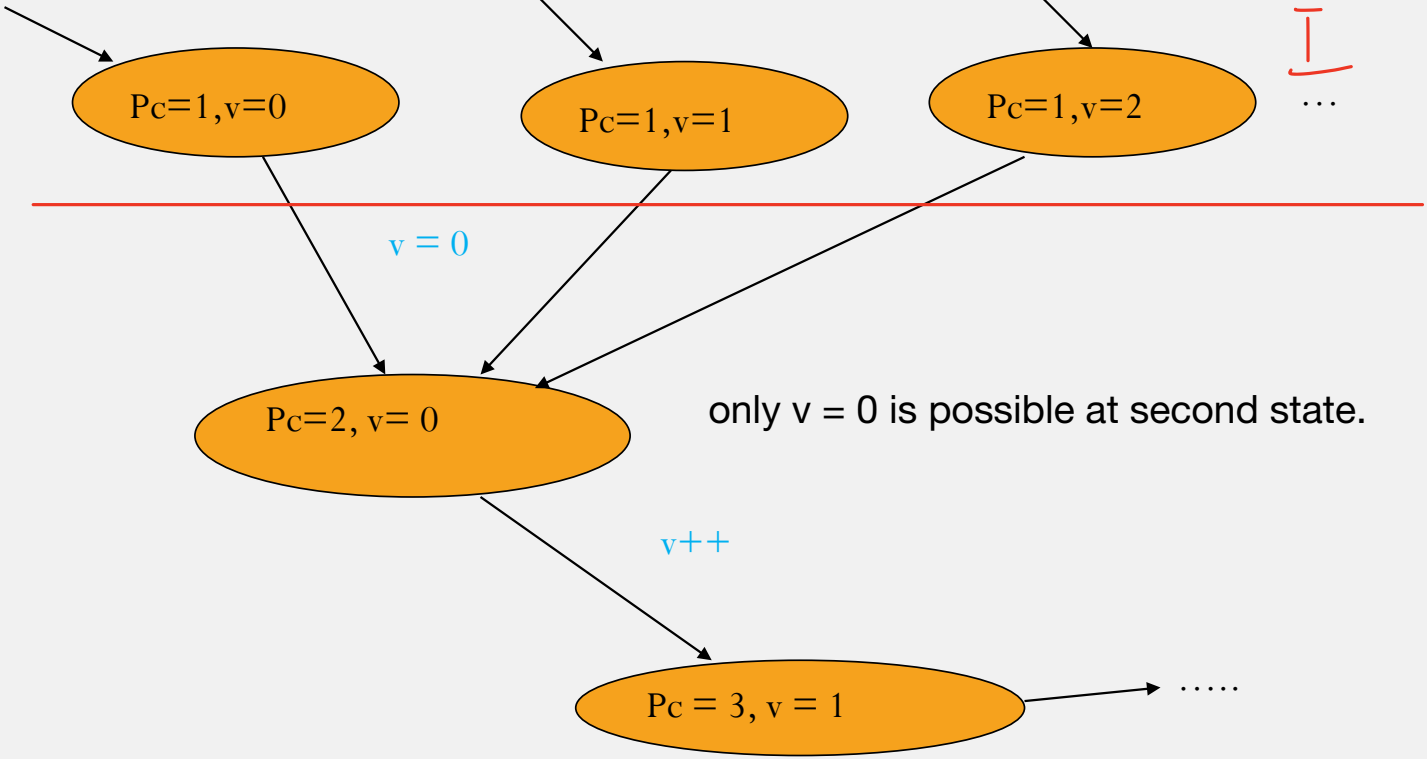
- 1 v = 0;
- 2 v++;
- 3 …
  - What are the states ?
    - (value of pc, value of v)
  - How many initial states are there ?
    - No info, depends on the type of v

- Draw the states and transitions corresponding to this program.

EXAMPLE

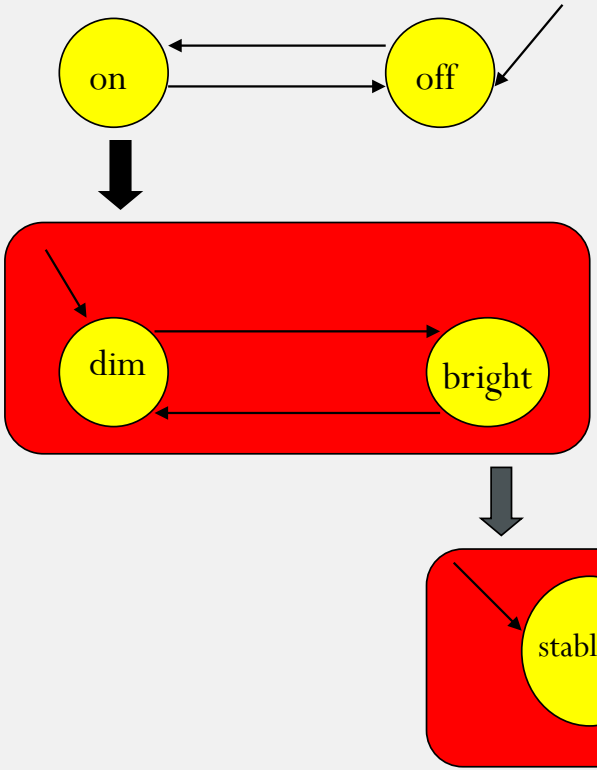
FSM

any value of v is possible at initial states (depend on the type of v)



HIERARCHY

Choice of steps at different levels of details also promotes hierarchical modeling.



look inside a state, gives another state diagram that describes behaviour in that state:

- e.g, "on state" --> "dim/bright"
- infinite hierarchy of state machines

CONCURRENT COMPOSITION

- $M1 = (S1, I1, \rightarrow_1)$ $M2 = (S2, I2, \rightarrow_2)$
- Define
  - $M1 \times M2 = (S1 \times S2, I1 \times I2, \rightarrow)$
  - Where $(s1,s2) \rightarrow (t1, t2)$ provided
    - $s1 \in S1, t1 \in S1,$
    - $s2 \in S2, t2 \in S2,$
    - $(s1 \rightarrow_1 t1)$ OR $(s2 \rightarrow_2 t2)$

2 objects running concurrently (at the same time) communicating at the same time = concurrent composition.

define as state machines S1*S2
set of initial states = I1 * I2

AND there is either an edge from s1 to t1, or an edge from s2 to t2

- Defines control flow of the composed FSM as an arbitrary interleaving of flows from components.
  - *Interleaving of independent flows, what about comm.?*

COMMUNICATING FSM

Basic FSM

- $M = (S, I, \rightarrow)$
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\rightarrow \subseteq S \times S$ is the transition relation.

Communicating FSM

- $M = (S, I, \sum, \rightarrow)$
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\sum$ is the set of action names that it takes part in
  - $\rightarrow \subseteq S \times \sum \times S$ is the transition relation.

Communication across FSMs via action names.

COMPOSITION OF COMMUNICATING FSM

- $M1 = (S1, I1, \Sigma_1, \rightarrow_1)$ $M2 = (S2, I2, \Sigma_2, \rightarrow_2)$
- Define
  - $M1 \times M2 = (S1 \times S2, I1 \times I2, \Sigma_1 \cup \Sigma_2, \rightarrow)$
  - And $(s1,s2) \xrightarrow{a} (t1,t2)$ provided
    - $s1 \in S1, t1 \in S1$, and
    - $s2 \in S2, t2 \in S2$, and
    - If $a \in \Sigma_1 \cap \Sigma_2$ we have $(s1 \xrightarrow{a} t1)$ and $(s2 \xrightarrow{a} t2)$
    - If $a \in \Sigma_1 - \Sigma_2$ we have $(s1 \xrightarrow{a} t1)$
    - If $a \in \Sigma_2 - \Sigma_1$ we have $(s2 \xrightarrow{a} t2)$

$a \in \Sigma_1 \cup \Sigma_2$.

Σ_1: 所有 $s_1 \rightarrow t_1$ 的情况.

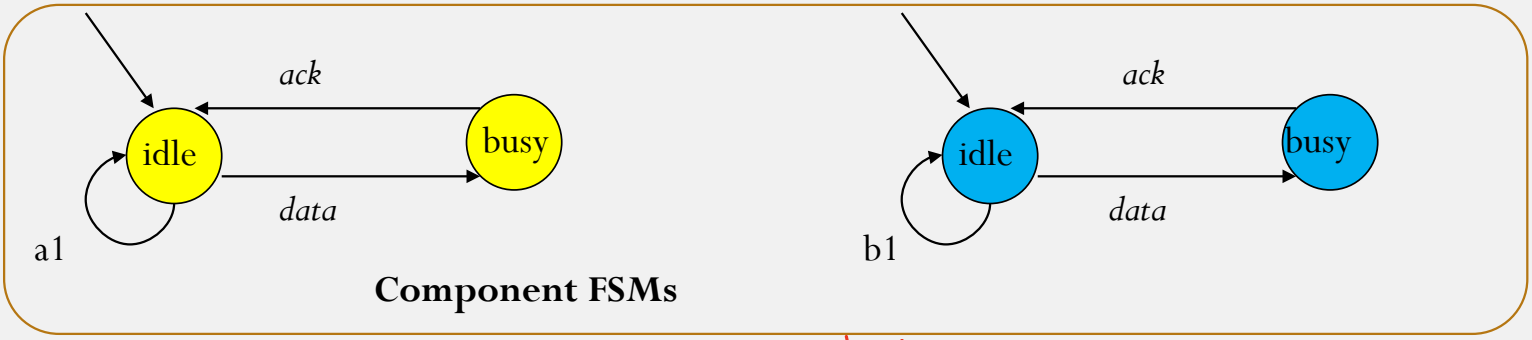
Σ_2: 所有 $s_2 \rightarrow t_2$ 的情况.

$\Sigma_1 \cap \Sigma_2$: $s_1 \rightarrow t_1$, $s_2 \rightarrow t_2$.

$\Sigma_1 - \Sigma_2$: 只有 $s_1 \rightarrow t_1$.

$\Sigma_2 - \Sigma_1$: 只有 $s_2 \rightarrow t_2$.

EXAMPLE



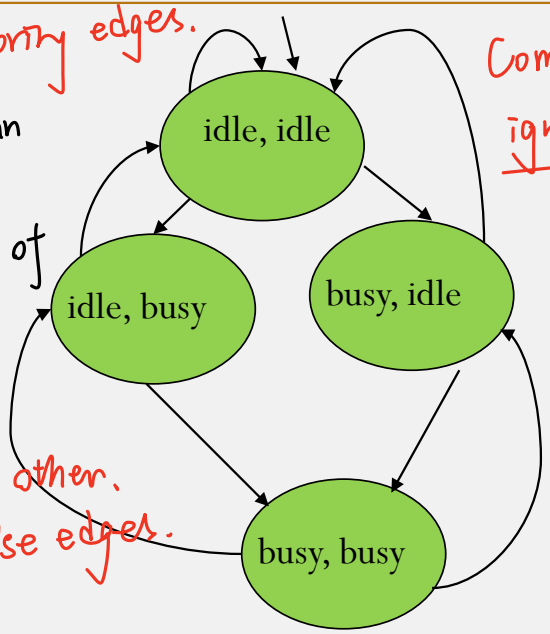
Component FSMs

All possible states. ignoring edges.

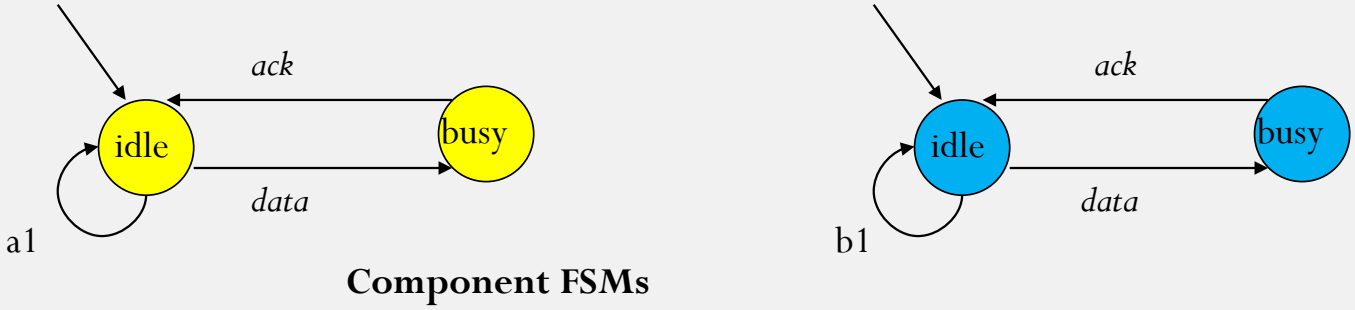
① Each object is making an independent move.

eg. Our lecture hall, each of us is a FSM;

② However, we want communication among each other. → Use edges.



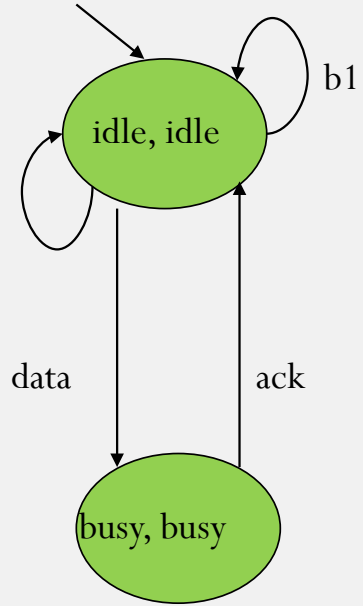
Composition of both. ignore edges.



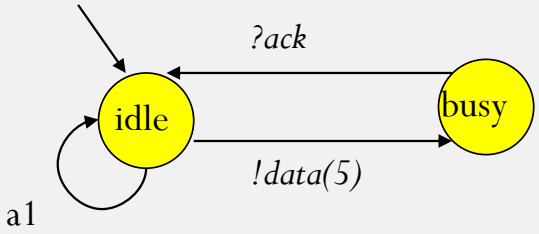
Component FSMs

Only 2 reachable states.

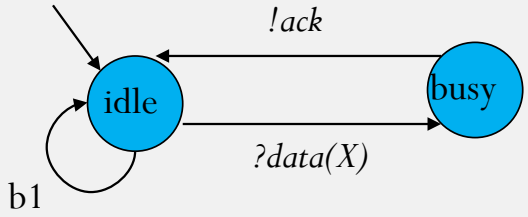
→ Composition considers action name now.



DATA COMMUNICATION



Sender Process

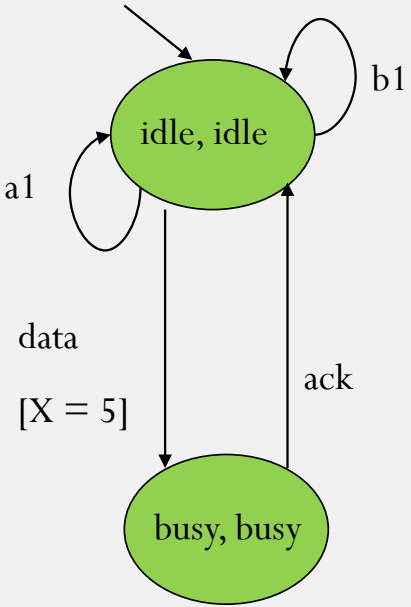


Receiver Process

① Send and receive can be done asynch (another model).

② State Charts : synch.

Usually common actions happen in a single step.



FSM – WRAP UP

- FSMs denote an intra-component style of modeling
  - Given a large system – identify its components
  - Model each component as FSM – M1, M2, M3
  - Overall system modeled as concurrent composition
    - M1 || M2 || M3

- Alternate style of modeling
  - Inter-component style
  - Emphasize communication over computation.
  - Sequence Diagrams are basic snippets for describing communication.

SEQUENCE DIAGRAMS

Prof. Abhik Roychoudhury

National University of Singapore

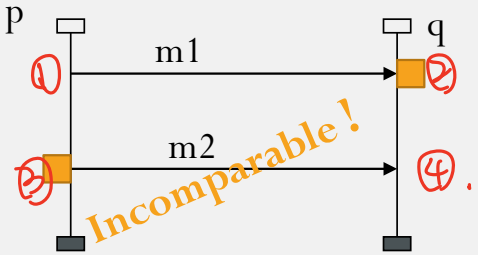
MSC BASED MODELS

2 process : p, q

4 events : send and receive of m1.
send and receive of m2.

MSC = Message Sequence Chart

- Labeled partial order of events
  - Highlights inter-process communications
  - While, FSMs highlight intra-process control flow.

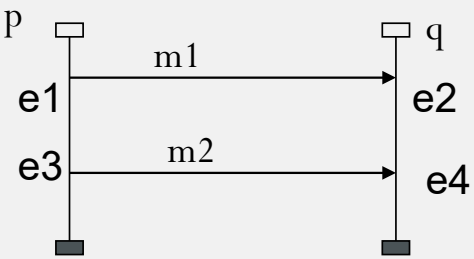


Incomparable : (receive of m1 and send of m2).

Next slides.

MSC PARTIAL ORDER

- How is the partial order constructed
  - *Time flows from top to bottom along each vertical line.*
    - *e1 < e3 and e2 < e4*
  - *Each message receive must occur after the corresponding send.*
    - *e1 < e2 and e3 < e4*
  - *Apply these rules over and over again to find out which event takes place before which other event.*
    - *e1 < e2, e2 < e4, e1 < e2, e3 < e4, e1 < e4*



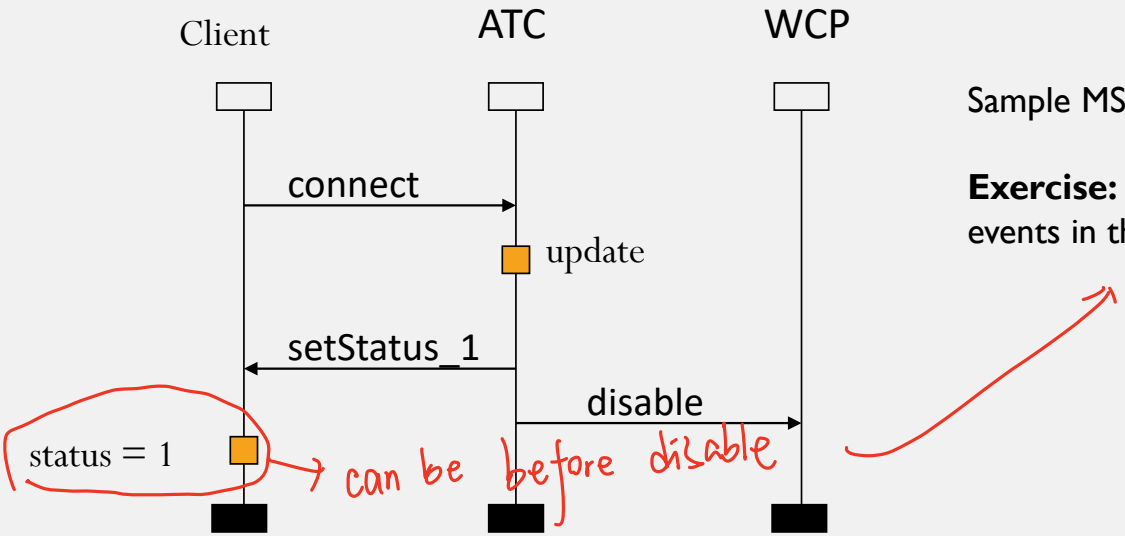
Cannot find total order.

Cannot deduce e2 < e3 or e3 < e2

Incomparable events

TYPICAL USE OF MSC

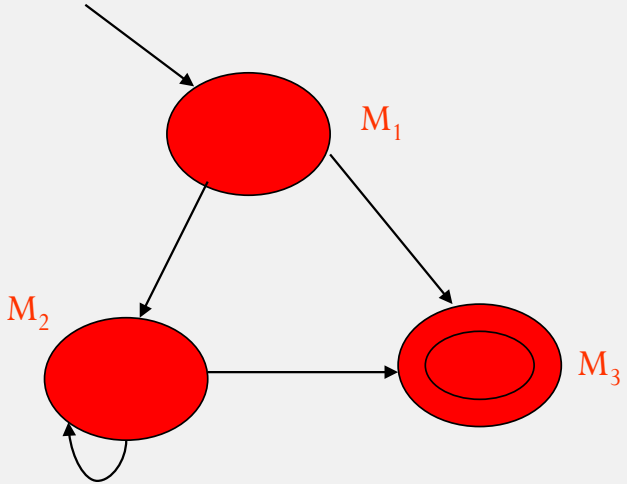
- Describe sample scenarios of system interaction
  - Appears in requirement documents
  - Do not describe "complete" system behavior



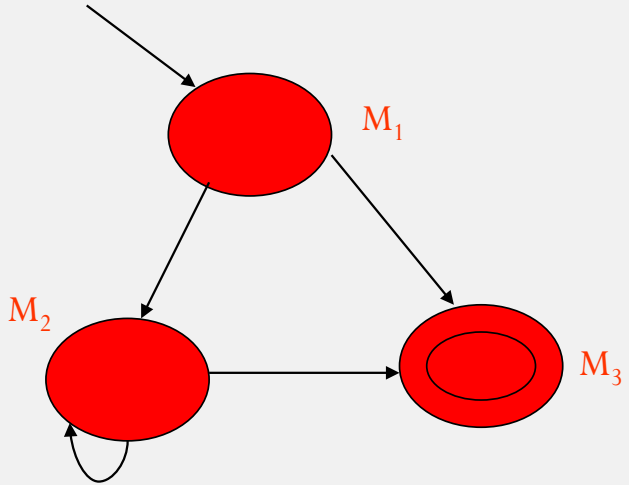
Sample MSC from ATC example

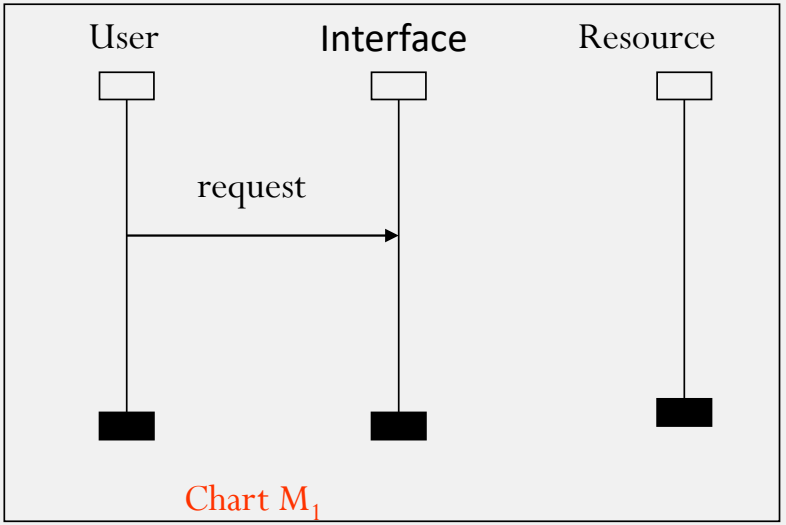
Exercise: Find two incomparable events in this MSC

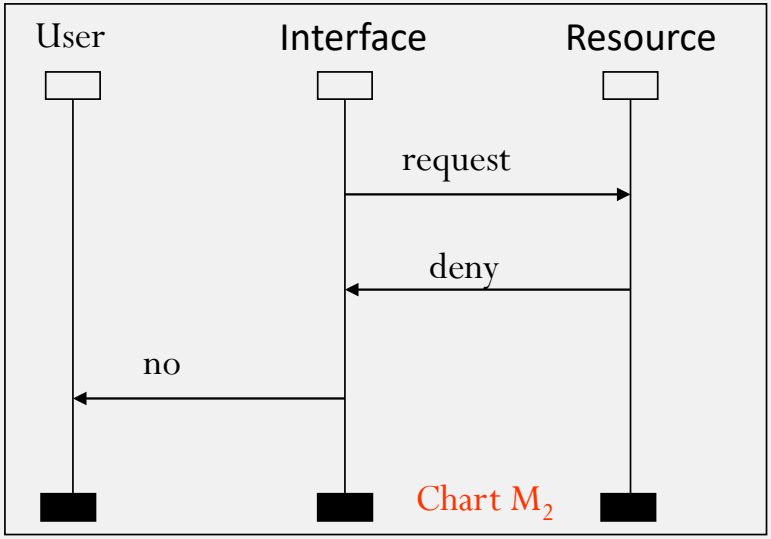
MSC BASED SYSTEM MODEL

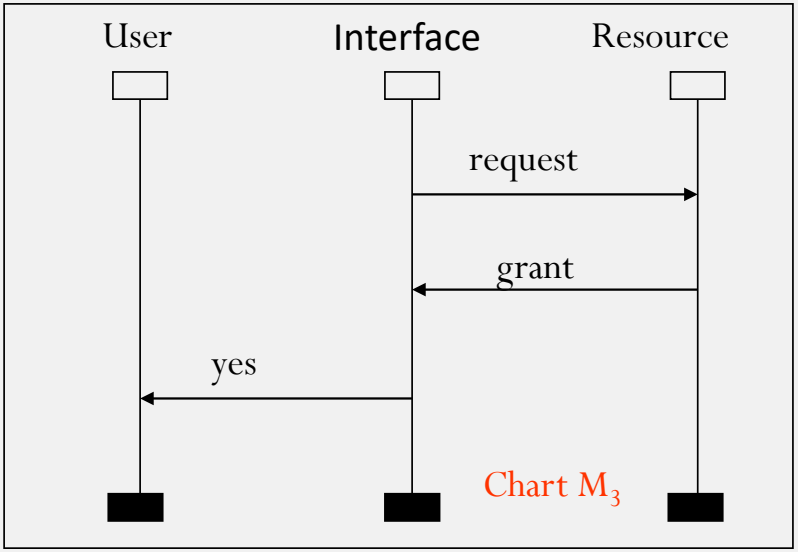


Connect MSCs into a graph – Message Sequence Graph (MSG)
Each node of the graph is a MSC.
Need to define the meaning of concatenation of MSCs

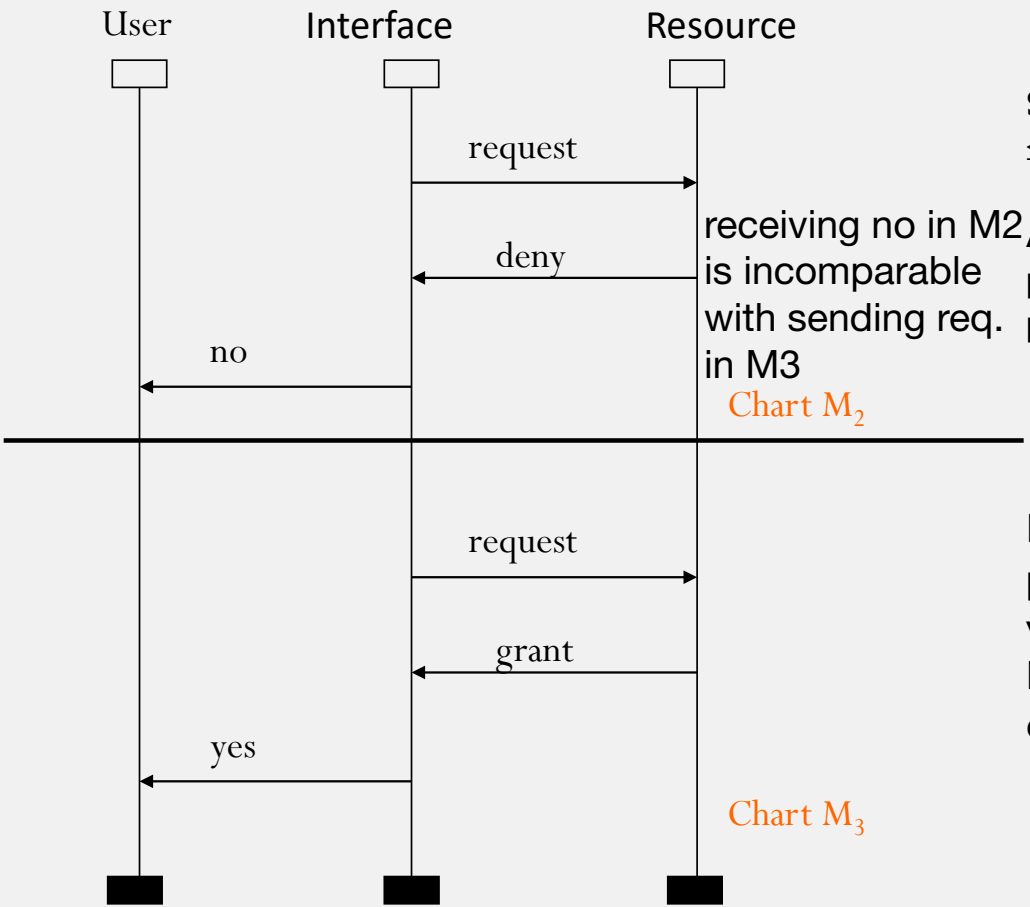

M_1
M_2
M_3


User
Interface
Resource
request
Chart M_1


User
Interface
Resource
request
deny
no
Chart M_2


User
Interface
Resource
request
grant
yes
Chart M_3

MSC CONCATENATION



Synchronous: All events in M2 ≤ All events in M3

Asynchronous: All events in process p of M2 ≤ All events in process p of M3

eg. receive no < send req. in M_2. in M_3

Interface and Resource processes can finish M3 while User process is still in M2 – provided asynchronous concatenation is considered.

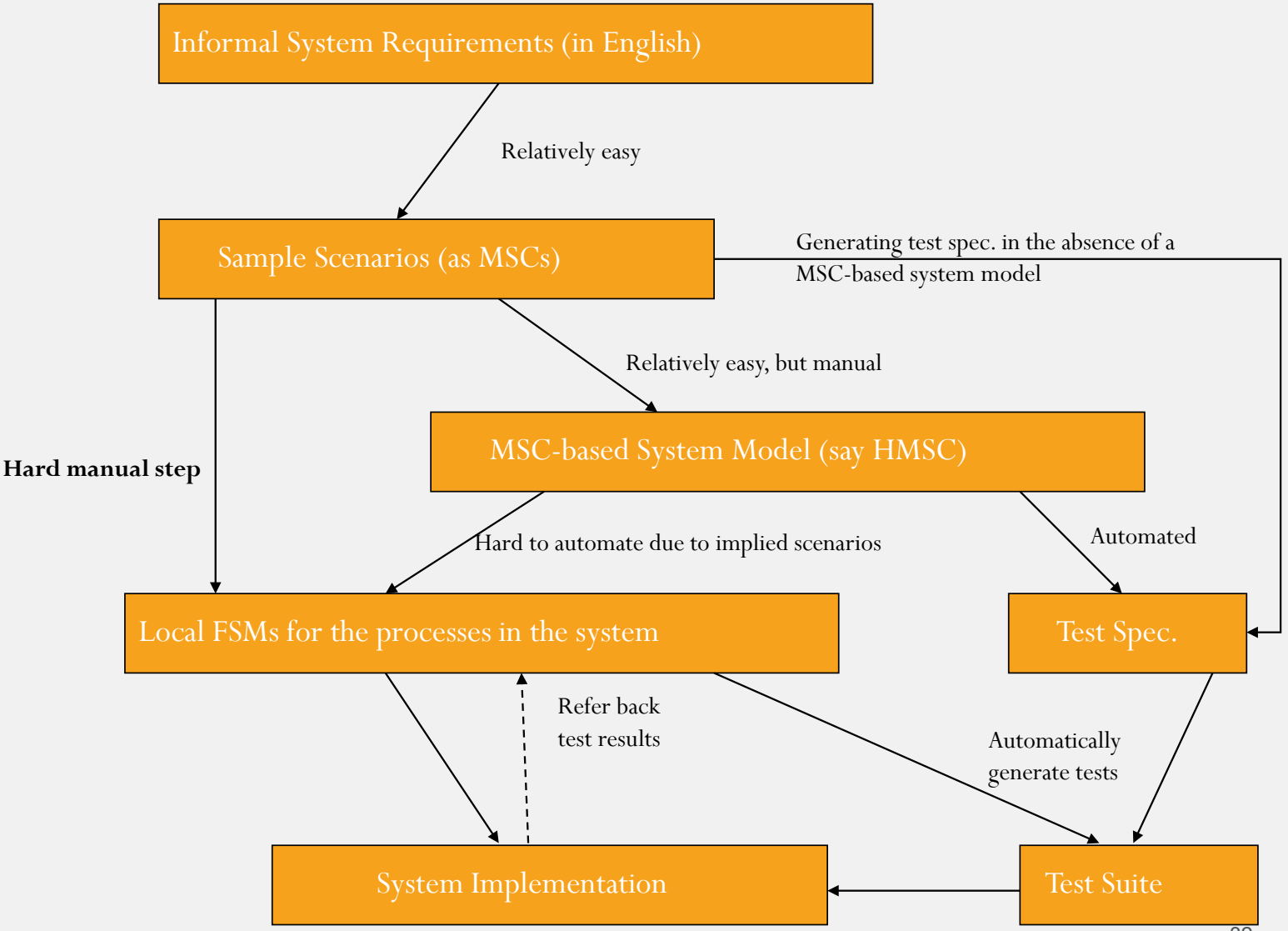
MSC-BASED DESIGN MODEL

- Complete
  - Complete description of system behavior.
  - MSG achieves this criterion.
- Based on well-established modeling notations.
  - We use UML Sequence Diagrams, which is OK.
- Preferably executable
  - Can simulate the model, and get a feel for how the constructed system will behave!
  - Global simulation of MSG is possible.
  - But not per-process execution !!

PUTTING NOTATIONS TOGETHER

- So, far we have studied 2 notational styles
  - Intra-process style FSM modeling notations
  - Inter-process style MSC-based modeling notation.

- In actual system modeling from English requirements
  - How do they fit together?
  - What roles do they play?
  - Are they both used in parallel?


Informal System Requirements (in English)
Relatively easy
Sample Scenarios (as MSCs)
Generating test spec. in the absence of a MSC-based system model
Relatively easy, but manual
Hard manual step
MSC-based System Model (say HMSC)
Hard to automate due to implied scenarios
Automated
Local FSMs for the processes in the system
Test Spec.
Refer back test results
Automatically generate tests
System Implementation
Test Suite

MODEL SIMULATIONS
CS3213 FSE

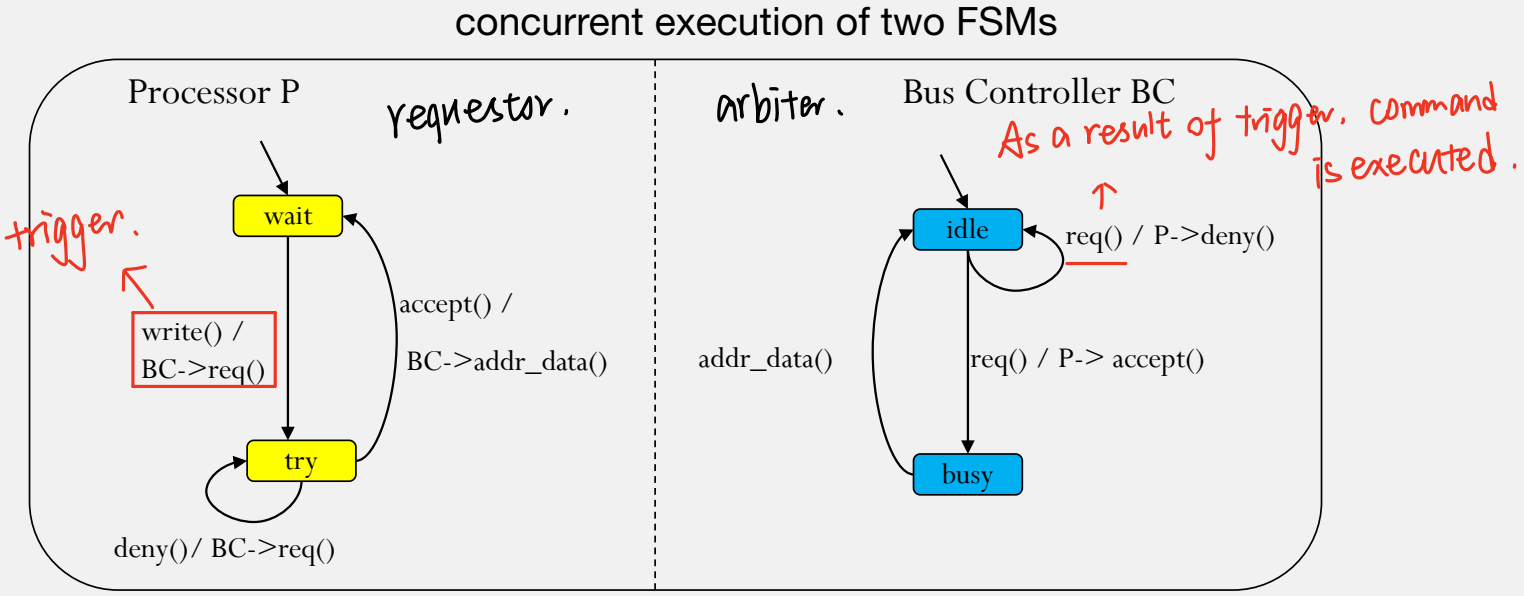
Prof. Abhik Roychoudhury

National University of Singapore

ORGANIZATION OF SLIDES

- So Far
  - What is a Model?
  - ATC – Running Example
    - Informal Req. at a lab scale.
    - Has subtle deadlock error (see textbook chap 2.3)
  - How to model such requirements
    - Modeling Notations
      - Finite State Machines
      - MSC based models
- Now, how to validate the models
  - Simulations of FSM models, MSC-based models.

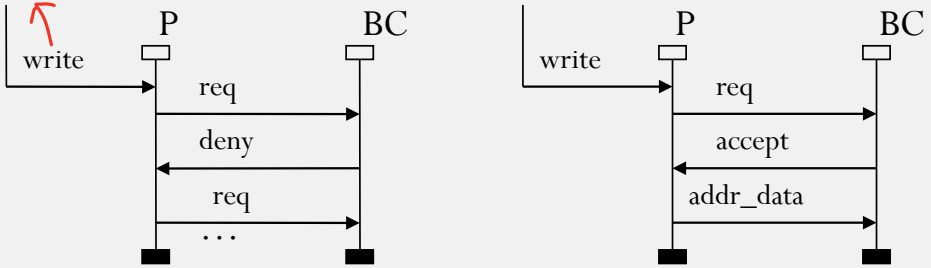
EXAMPLE: STATE DIAGRAM



Processor and Bus Controller – what does the example do?

POSSIBLE BEHAVIOR

req. from environment.



Sample scenarios of the State Diagram shown in the previous slide.

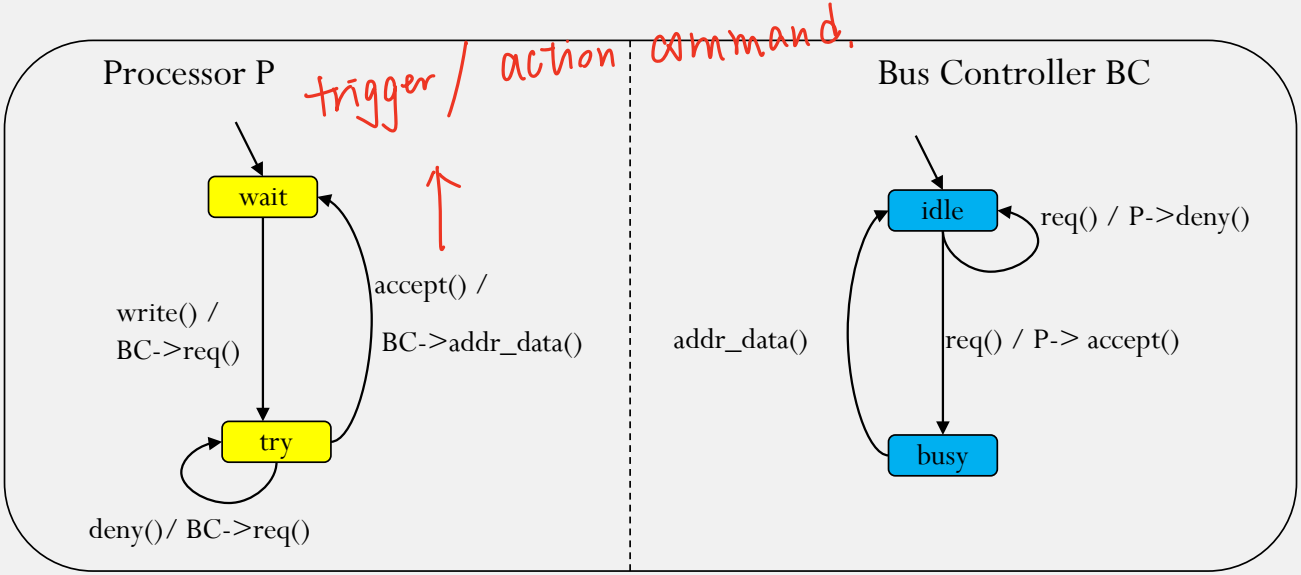
Super-step:

On encountering a write, the sequence of method calls executed is

write, req, (deny, req)*, accept, addr_data

How?

SIMULATION: STATE DIAGRAM



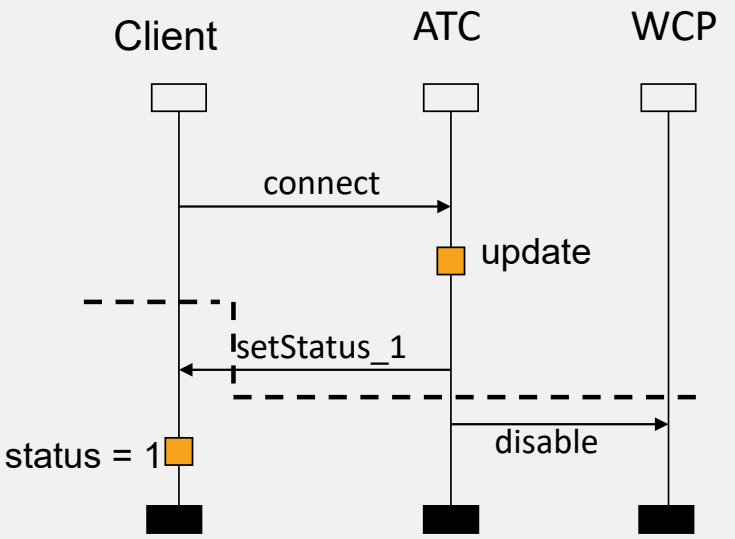
req(), deny(), req(), accept(), addr_data()

MODEL SIMULATION

- So far
  - FSMs and State Diagrams – Intra component style modeling
  - MSCs and MSGs - Inter component style modeling
  - Simulation of FSMs and State Diagrams

- How to simulate MSCs?
  - Generate a trace of events which satisfies the partial order denoted by a given MSC.
  - Always maintain a ``cut'' to denote the progress in each process – while simulating a given MSC.
    - The whole question now is how to advance a cut.
    - Let us look at this matter visually!

SIMULATING MSC



Cut 上的是一层.

下的是一层.

send connect done.

but receive setStatus, disable 还没.

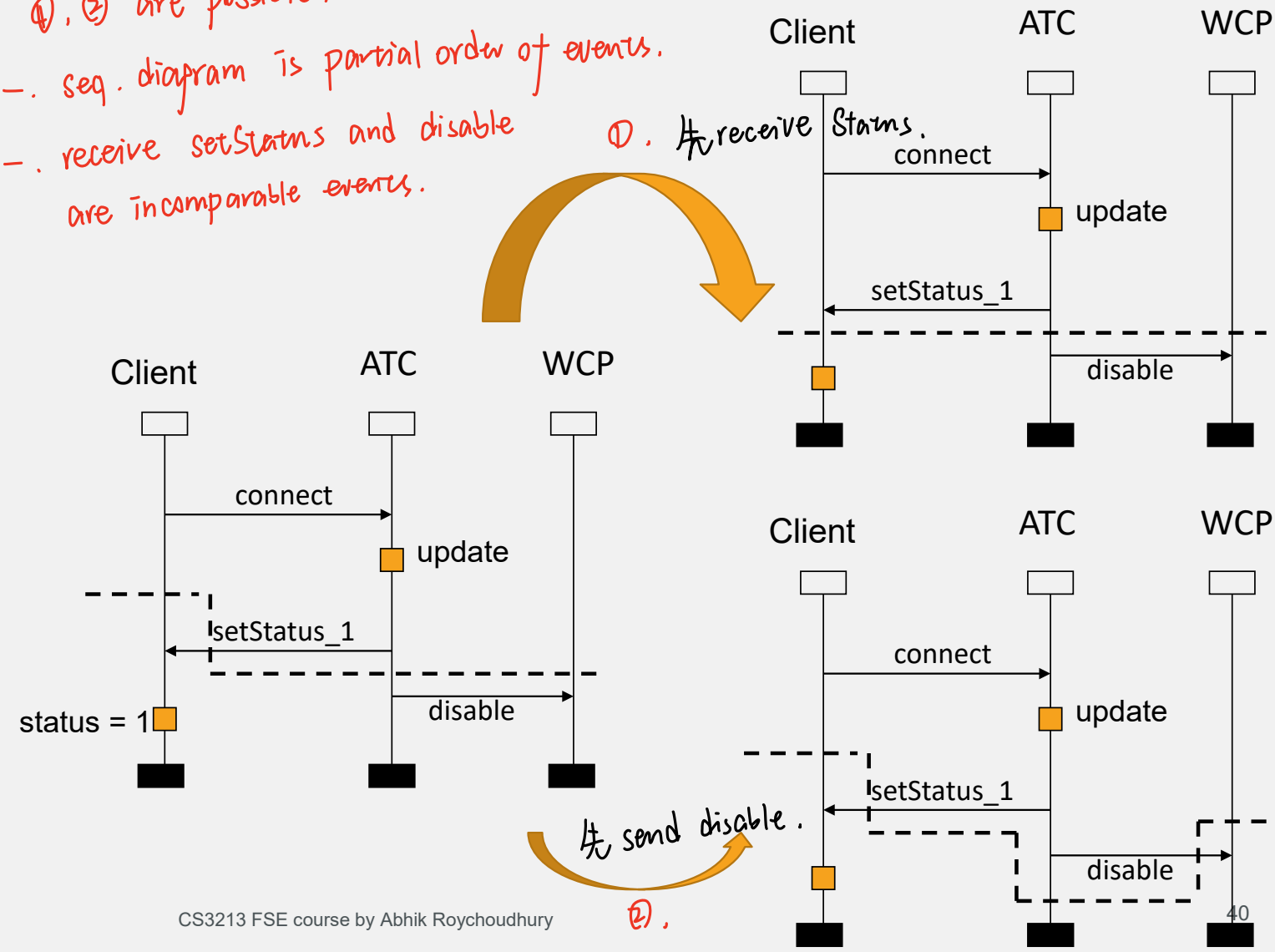
Cut

Shows progress of the individual processes / components: ATC, WCP, Client

①, ② are possible:

– seq. diagram is partial order of events.

– receive setStatus and disable are incomparable events.



RECAP: MSC SEMANTICS

- For a sequence of MSCs --- M1, M2
  - **Synchronous concatenation:** All events in M1 $\leq$ All events in M2
  - **Asynchronous concatenation**: All events in process p of M1 $\leq$ All events in process p of M2
- For any msg. m sent from process p to process q
  - **Synchronous message passing**: Send and receive happens in the form of a hand-shake.
  - **Asynchronous message passing**: Sender sends message which is stored in a queue, picked up by receiver later.
- Simulating a sequence of MSCs will need to follow the concatenation & message passing semantics.

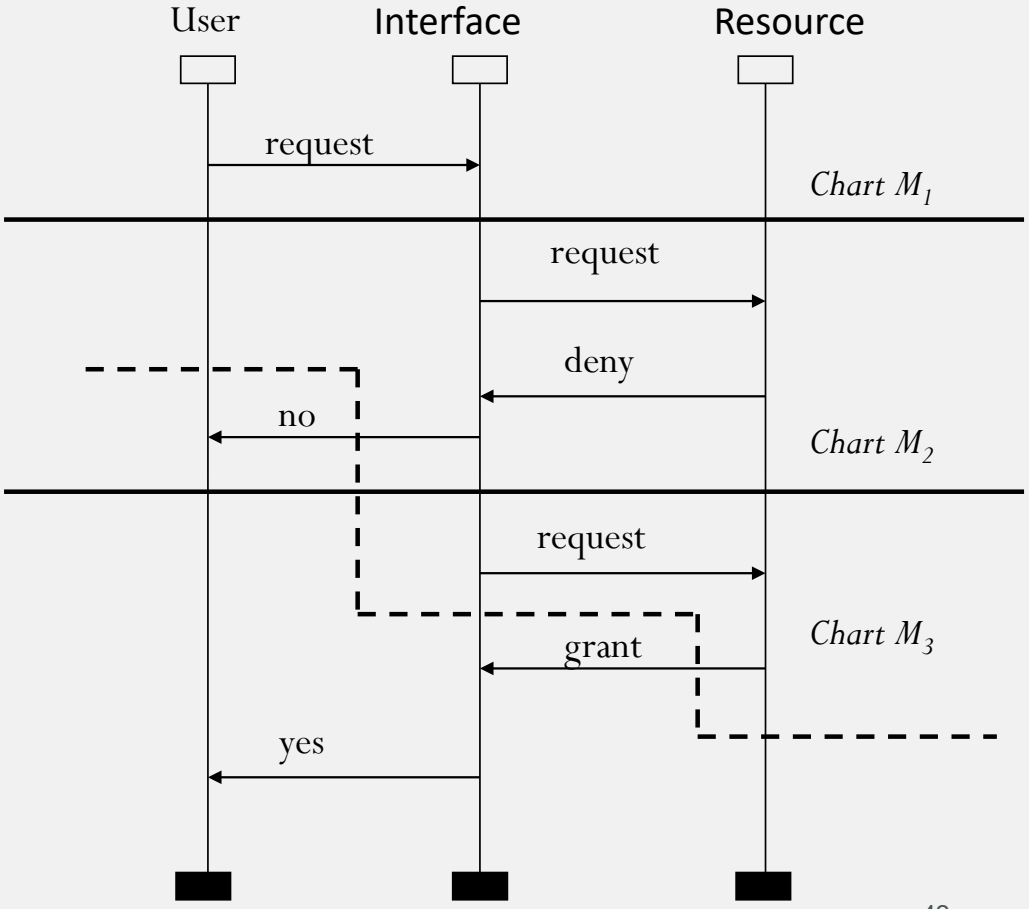
SIMULATING MSC SEQUENCE

Allowed for asynchronous concatenation.

Not allowed for synchronous concatenation.

User 在 M_2 时.

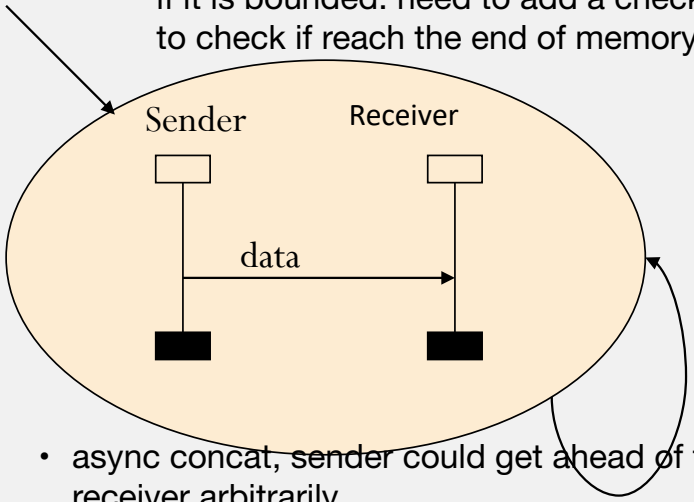
Interface, Resource 在 M_3 了



UNBOUNDED MEMORY?

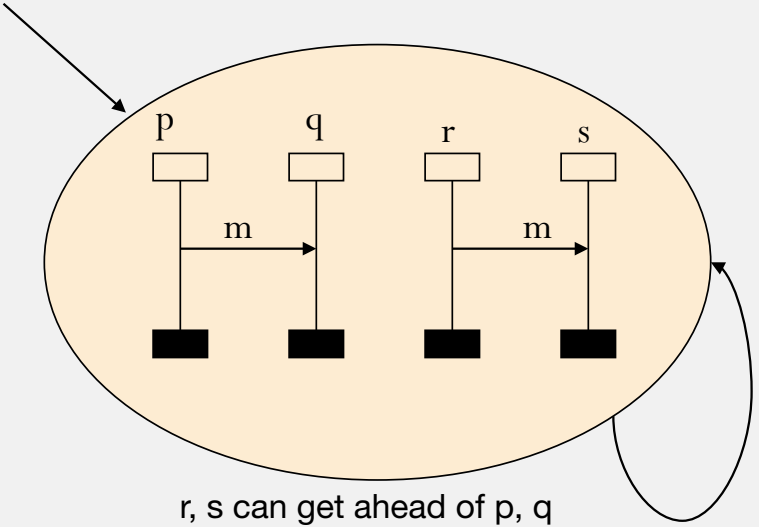
assume no need to complete current seq diagram before go on to execute next one.

if it is bounded: need to add a checker to check if reach the end of memory



- async concat, sender could get ahead of the receiver arbitrarily
- asyn message passing: possible that sender keep sending but nothing ever be received.

Simulation requires unbounded memory under asynchronous concatenation and asynchronous message passing



r, s can get ahead of p, q arbitrarily

Simulation requires unbounded memory under asynchronous concatenation and synchronous / asynchronous message passing.

CS3213 FSE COURSE AT A GLANCE

- Software Engineering
  - Requirements -> Modeling -> Coding -> Testing
  - Requirements: Elicitation and Representation, Checking without a single LoC
  - Modeling: UML Notations, and variants, Architecture modeling
  - Coding: Major part of the module and the project, Repair.
  - Testing: Validation concepts, Debugging.