

---

# ***SYSC 3303 Real-Time Concurrent Systems***

## **Introduction to Use Case Maps (UCMs)**

- Copyright © 2001-2003 D.L. Bailey, 2003 L.S. Marshall Systems and Computer Engineering, Carleton University
- revised March 10, 2003

---

## ***References***

- *Use Case Maps as Architectural Entities for Complex Systems*, R.J.A. Buhr, Carleton University, August 31, 1998 (available from the course Web site)
  - this was the last paper that Prof. Buhr wrote about UCMs (for publication in an IEEE Transactions) before his retirement
  - for this course, we will use selected elements of the core UCM notation described in sections 2 and 3 (but AND-forks and AND-joins will not be covered in our first pass through the notation)
  - dynamic components (3.2, 5.1, 6.7), dynamic plug-ins (3.4, 5.2, 6.7), layering (4.4), constrained and unconstrained path styles (6.5), shared responsibilities (6.6), and exceptions (6.8), are beyond the scope of this course

---

## References

- *An Introduction to Real-Time Systems: From Design to Multitasking with C/C++*, R.J.A. Buhr & D.L. Bailey, Prentice Hall, 1999
  - the UCM notation was evolving when this book was written; as such, there are some minor notational differences between UCMs as presented in the book and in the paper cited on the previous slide. Aside from these few differences, the description of UCMs in the book is consistent with the notation as presented in the paper. As well, many of the more complex elements of the UCM notation described in the paper are not presented in this book.
  - Section 5.3 introduces UCMs
  - additional elements of the UCM notation are presented in the context of the MTU case study in Chapters 6 and 7
  - Chapter 9 is an introduction to structurally dynamic systems (extensions to the UCM notation to support structural dynamics are in Sections 9.2.5, 9.2.6, and 9.3)

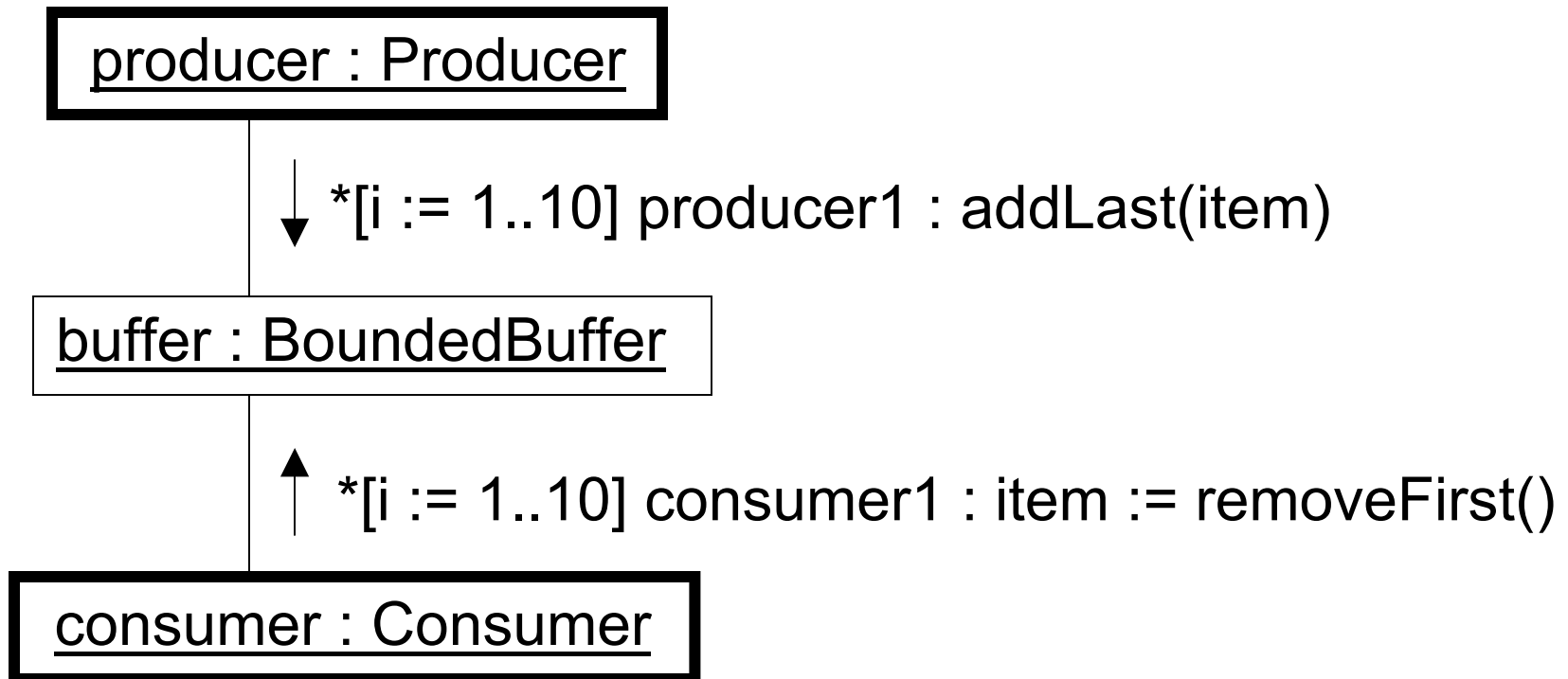
---

## ***Limitations of UML Collaboration Diagrams***

- Collaboration diagrams make commitments to objects, and links and messages between objects
  - discourages exploration of design alternatives
  - difficult to go from a blank sheet of paper to a good detailed design in one step
  - stimulus-response (causal) sequences that propagate through the system are obscured
    - e.g., what are the stimulus-response sequences in the collaboration diagram for the producer/consumer/bounded buffer system?

---

## *Limitations of UML Collaboration Diagrams*



- Imagine trying to explain to a colleague the key causal sequences in the system modelled by this collaboration diagram...

---

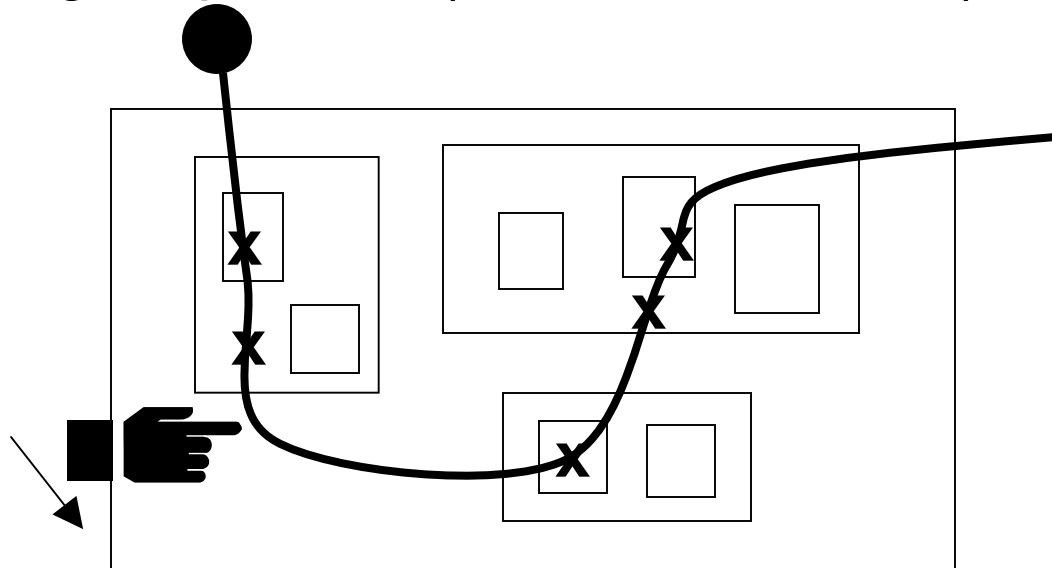
## ***Limitations of UML Collaboration Diagrams***

- Stimulus-response (i.e., causal) sequences are properties of the *problem*, but collaboration diagrams (and other UML diagrams) depict *solutions*
- We need a lightweight, high-level design notation that lets us reason about these stimulus-response sequences, without necessarily committing to specific components or intercomponent communication mechanisms
- This is especially useful for systems whose behaviour emerges at run-time, from collaborations of components that operate concurrently (no central controlling algorithm)

---

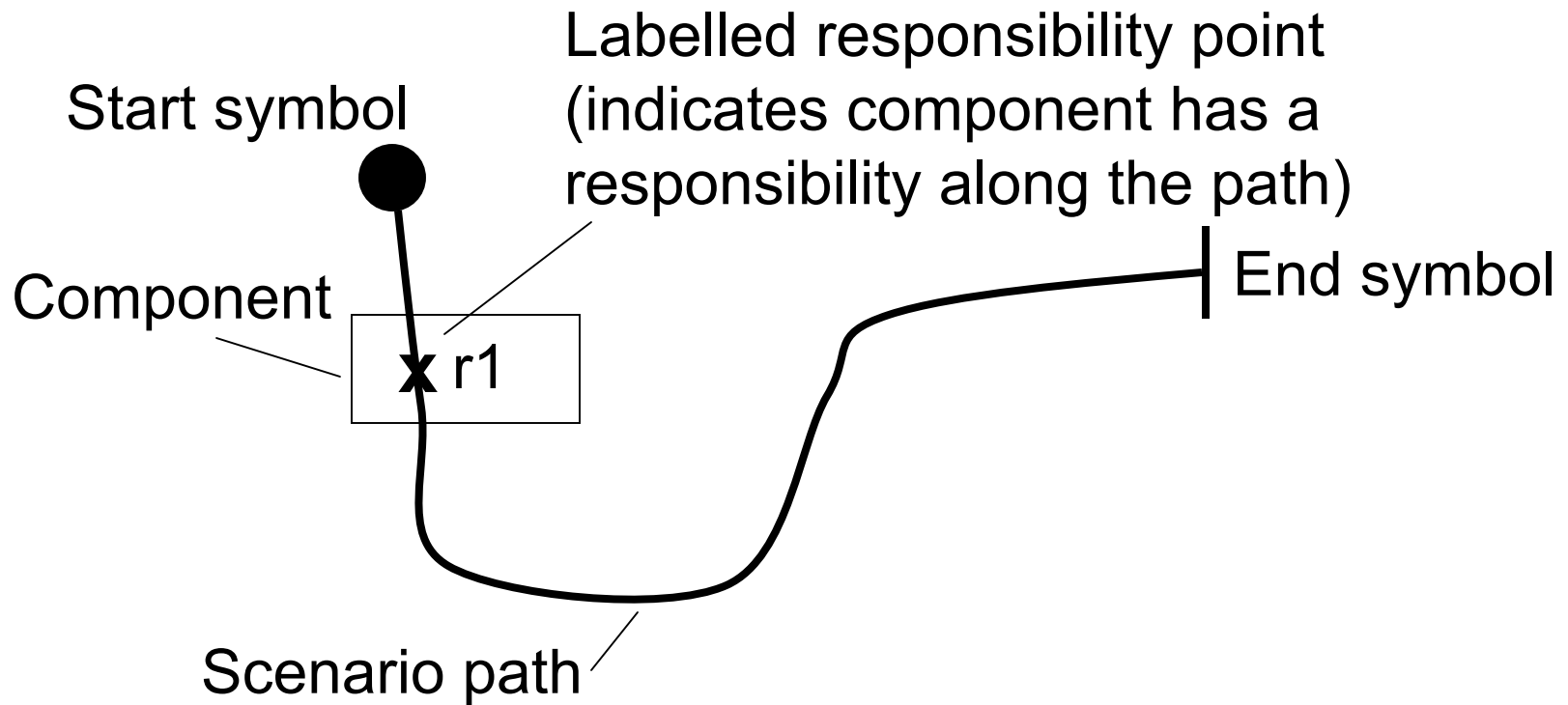
## ***Essence of Use Case Maps***

- Imagine tracing a path with your finger on top of a diagram of the structure of a system, to explain various causal sequences that might occur. A UCM is a visual record of paths traced by one or more finger-pointing sequences (called “scenarios”)



---

## ***Core UCM Notation***

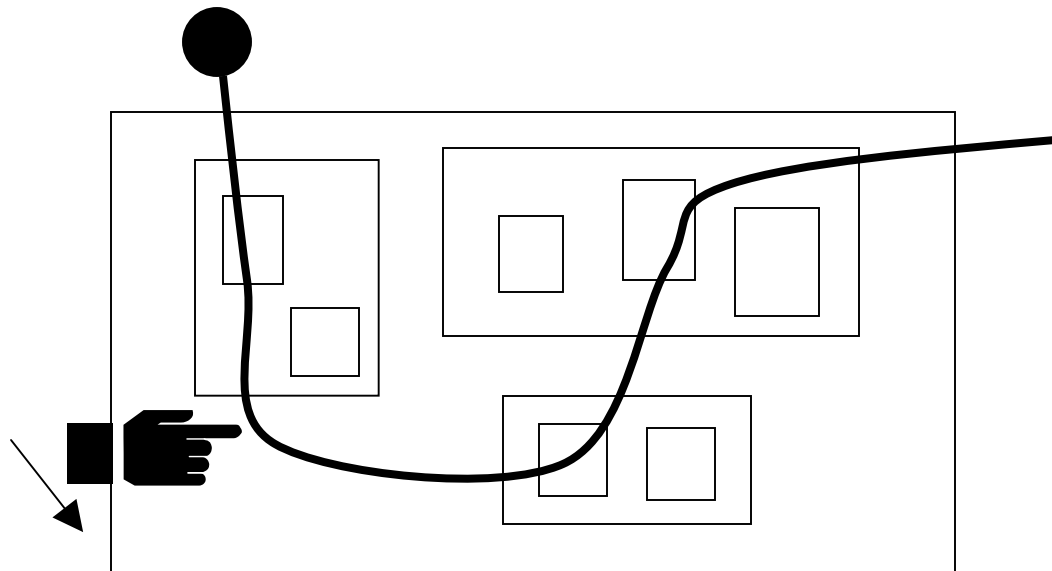




---

## ***Essence of Use Case Maps***

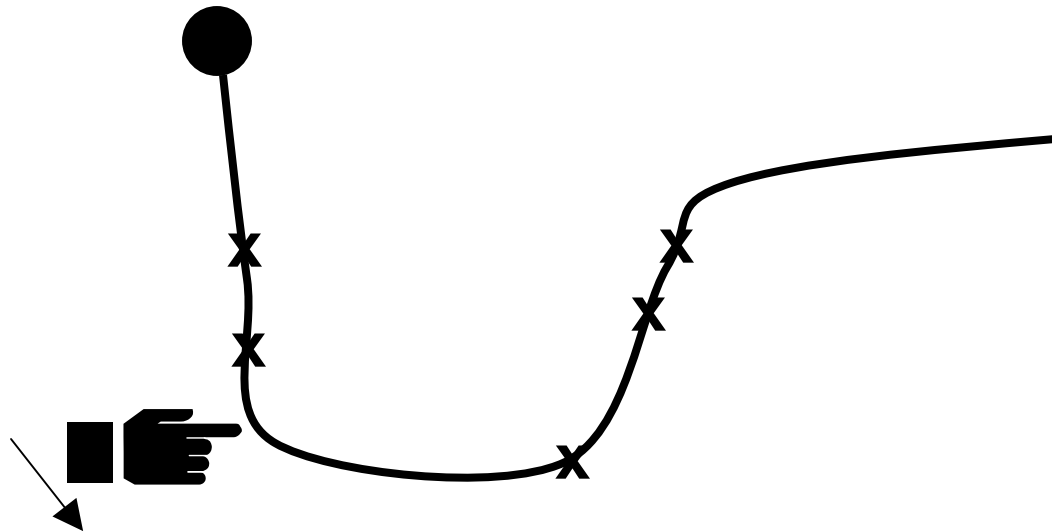
- Paths touch components at points called responsibilities, shown as x's in the previous slides, but they can also be shrunk to invisible dots, as in this slide



---

## ***Essence of Use Case Maps***

- Unbound maps (without components shown explicitly) are useful as requirements entities that provide a transition from prose use cases to UCMs, or as standalone "behaviour structures"



---

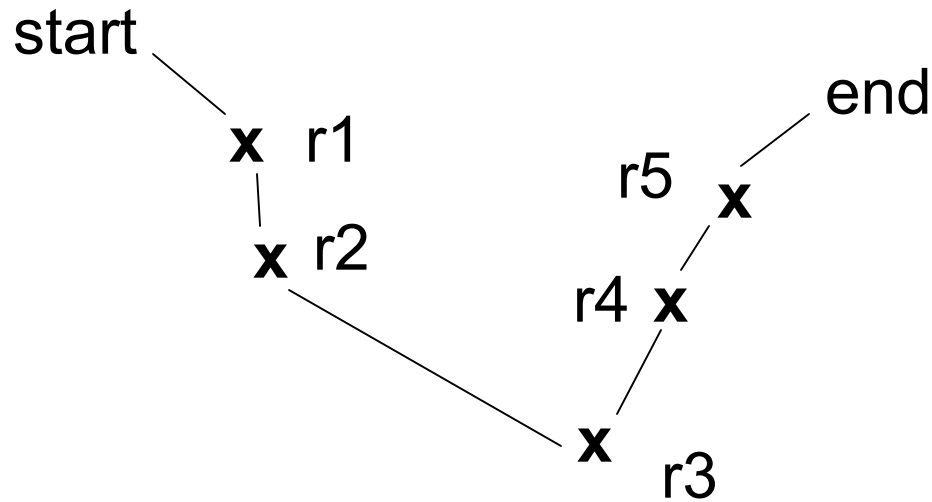
## ***Forward Engineering with UCMs***

**x** r1  
r5 **x**  
**x** r2  
r4 **x**  
**x** r3

- Labelled x's represent system responsibilities (actions to be performed by the system in response to a stimulus)
  - prose use cases are often a starting point for identifying responsibilities

---

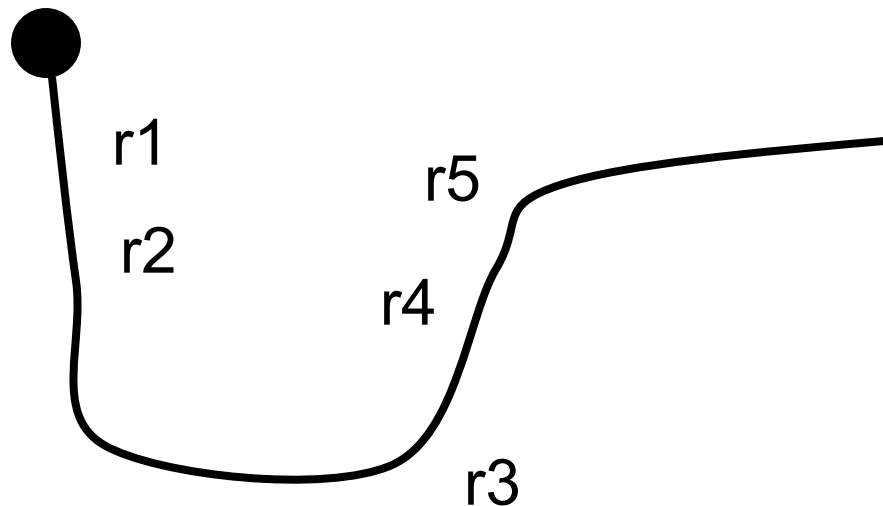
## ***Forward Engineering with UCMs***



- Link x's to show causal order of execution of responsibilities

---

## *Forward Engineering with UCMs*

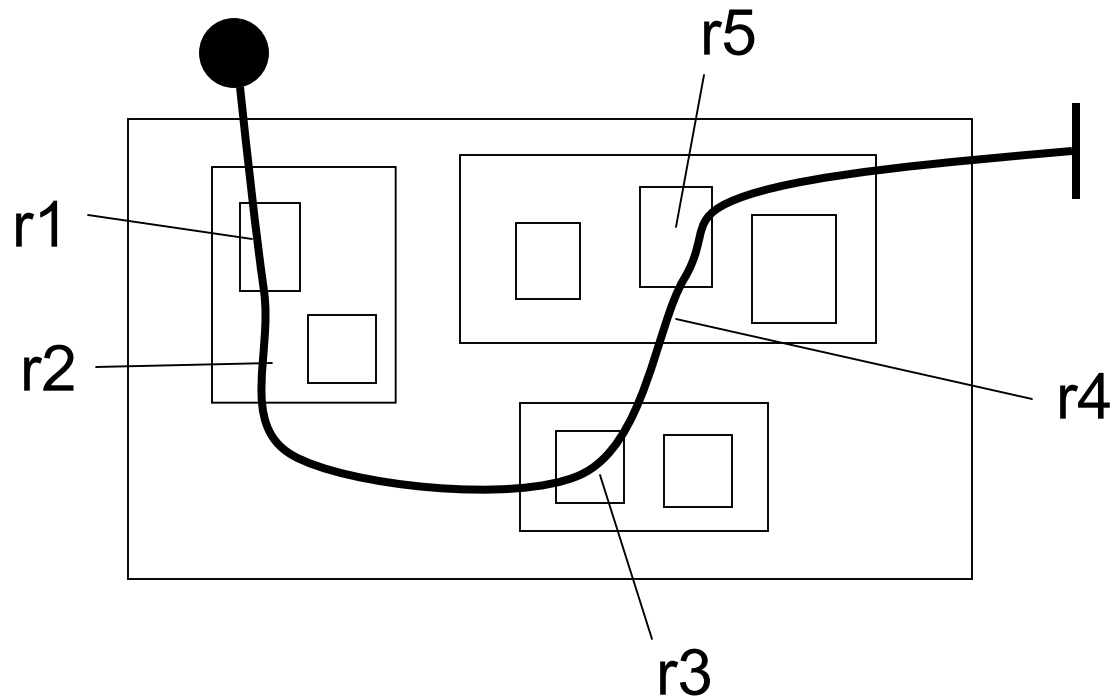


- Smooth line segments into a continuous curve, shrink responsibility points (x's) to invisible dots, add start and end symbols

---

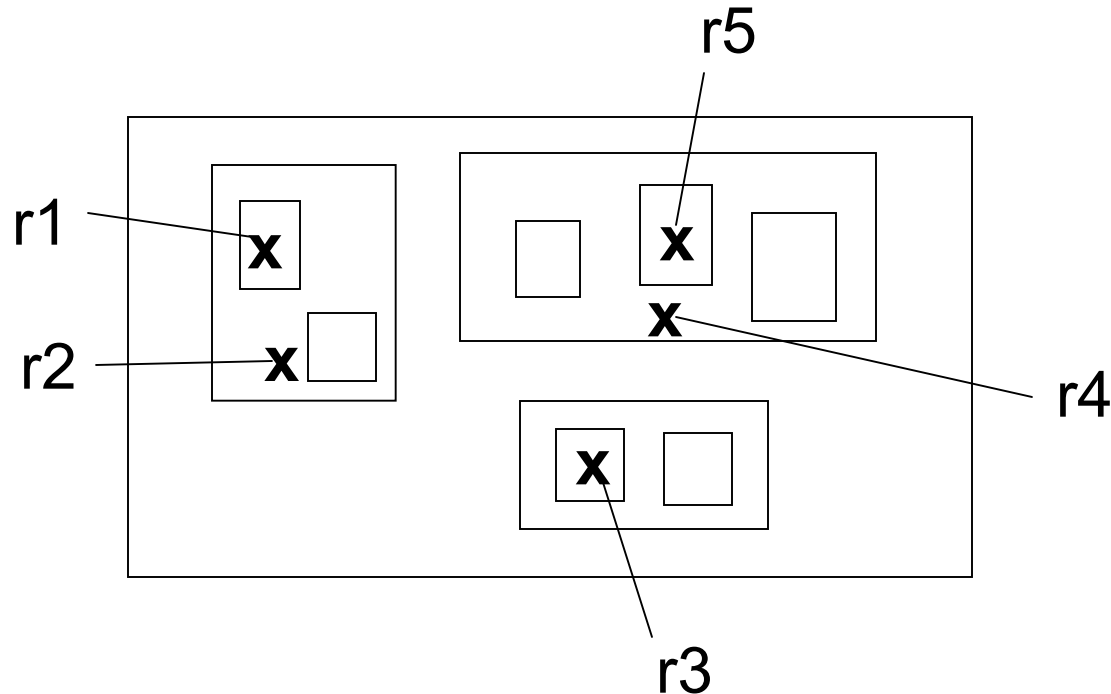
# ***Forward Engineering with UCMs***

- Bind components to responsibilities



---

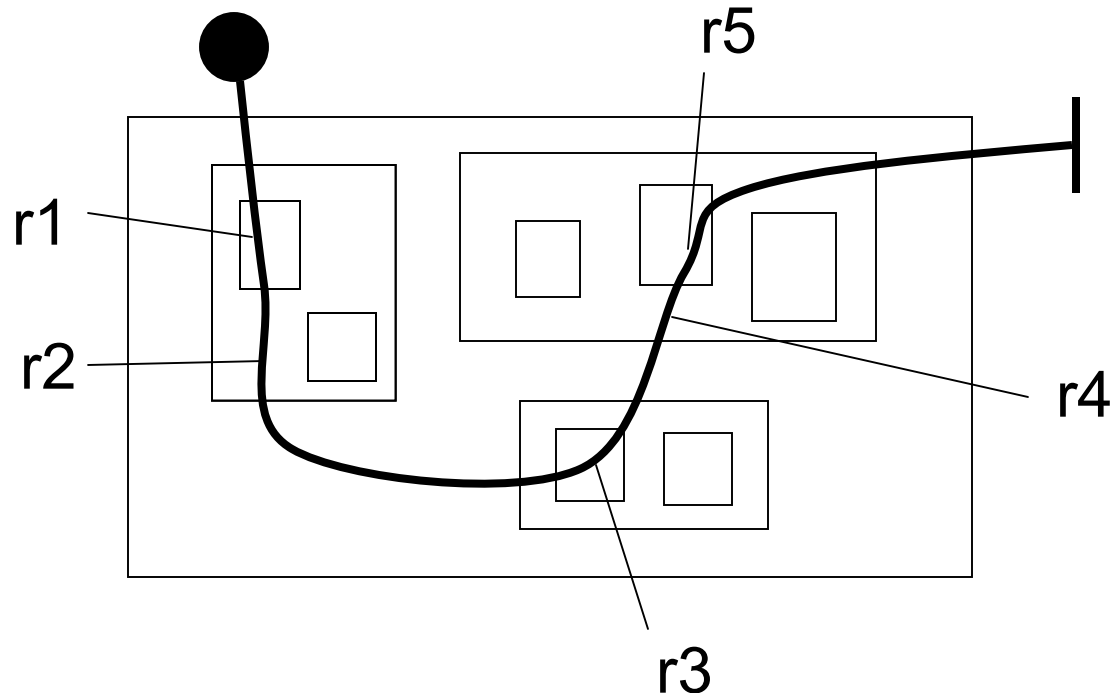
## ***Another Approach to Forward Engineering***



- Sometimes we start by identifying components and their responsibilities

---

## ***Another Approach to Forward Engineering***

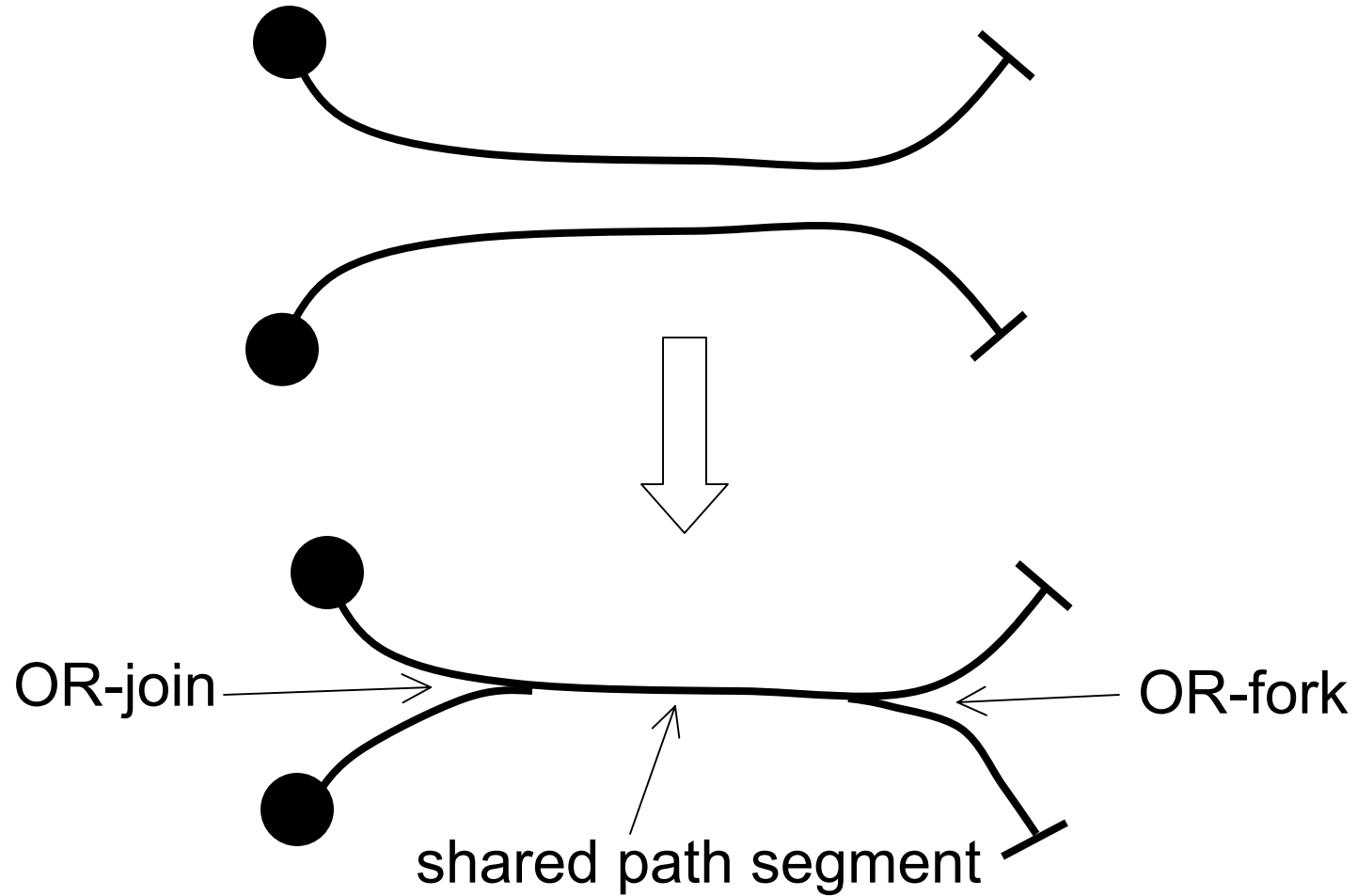


- Connect the x's into one path



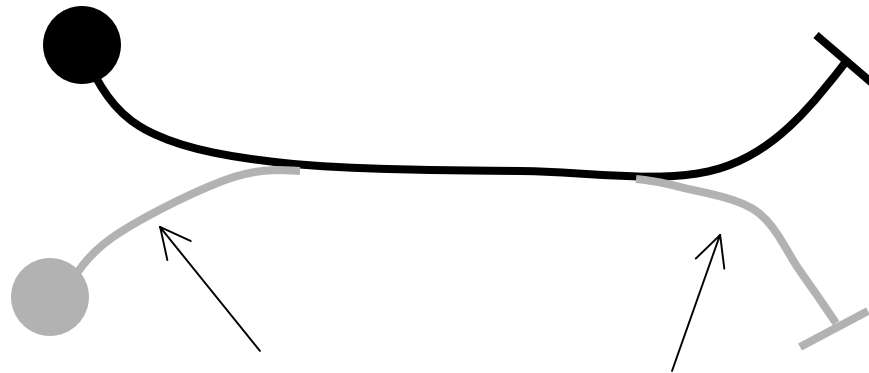
---

## ***Composite UCMs***



---

## ***Removing Visual Ambiguity***

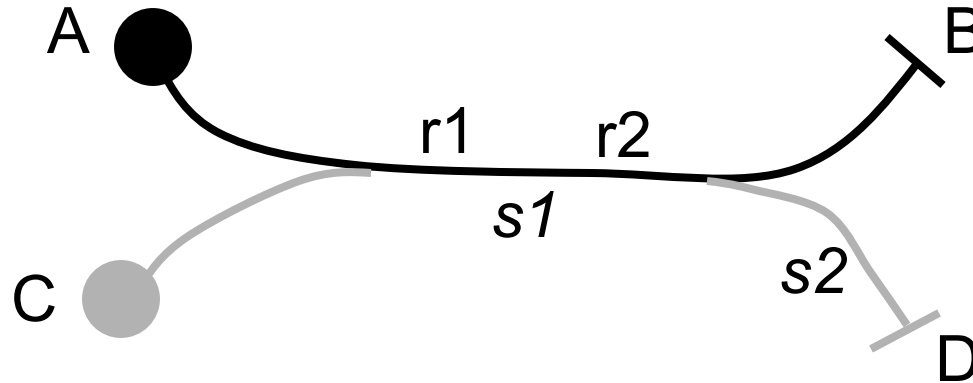


Highlight different paths using:

- different line shadings
- different line thicknesses
- different line colours

---

## ***Labelling Conventions***



- Uppercase, unitalicized labels indicate start & end points (e.g., A, B, C, D)
- Lowercase, unitalicized labels indicate responsibility points (e.g., r1, r2)
- Lowercase, italicized labels indicate path segments (e.g., s1, s2)

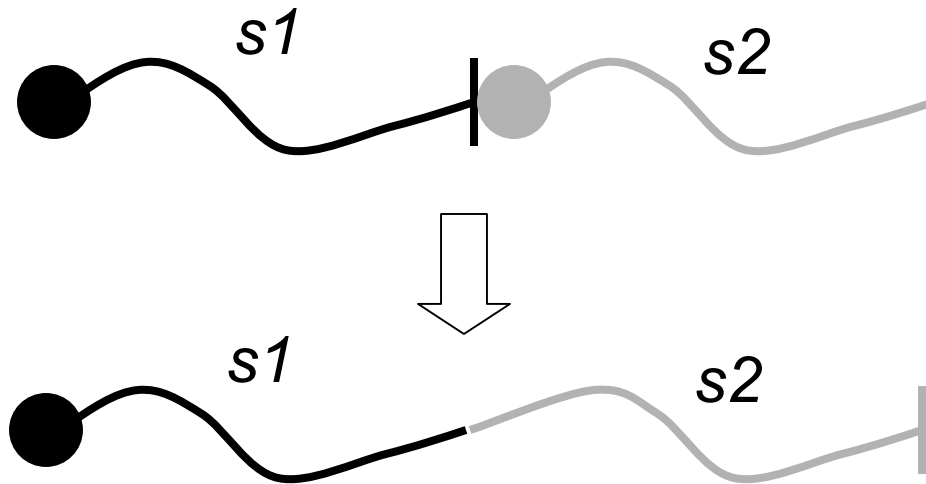
---

## ***Labelling Conventions***

- Informal information can be associated with labels;  
e.g.,
  - “a precondition of CD is that AB has been traversed at least once”
  - “r1 changes the system state, r2 reads the system state”

---

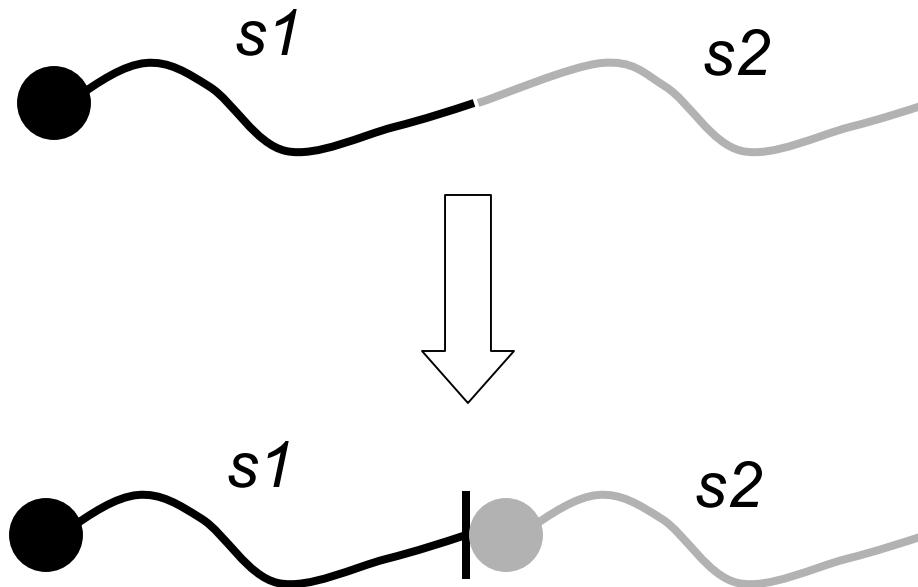
## ***Concatenating Paths***



- Effect is of one larger path with constituent segments joined end to end

---

## *Splitting Paths*



- Normally only used when we want to *factor* a UCM into multiple peer UCMs

---

## ***Producer/Consumer UCM***

- Consider the producer/consumer/bounded buffer system
- The major causal sequence is: an item produced by the producer is consumed by the consumer
- There are four major responsibilities:
  - produce item
  - add item to buffer
  - remove item from buffer
  - consume item

---

## ***Producer/Consumer UCM***

producer : Producer

x p

x a

buffer : BoundedBuffer

x r

consumer : Consumer

x c

Responsibilities:

p - produce item

a - add item to buffer

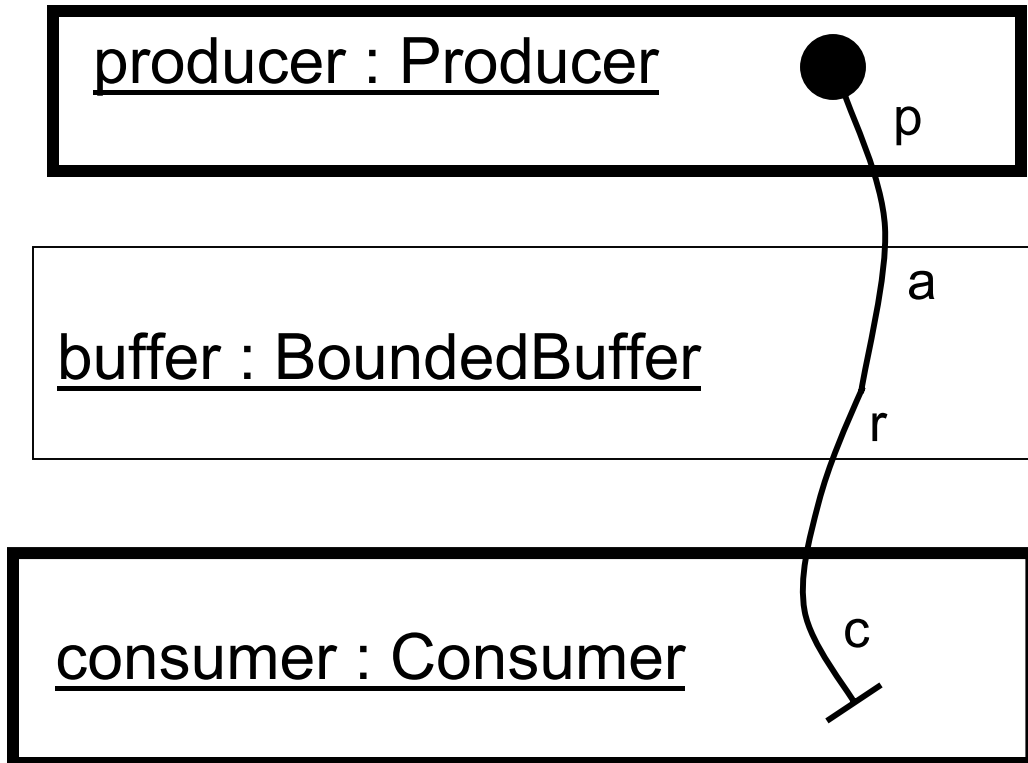
r - remove item from buffer

c - consume item



---

## ***Producer/Consumer UCM***



### Responsibilities:

p - produce item

a - add item to buffer

r - remove item from buffer

c - consume item

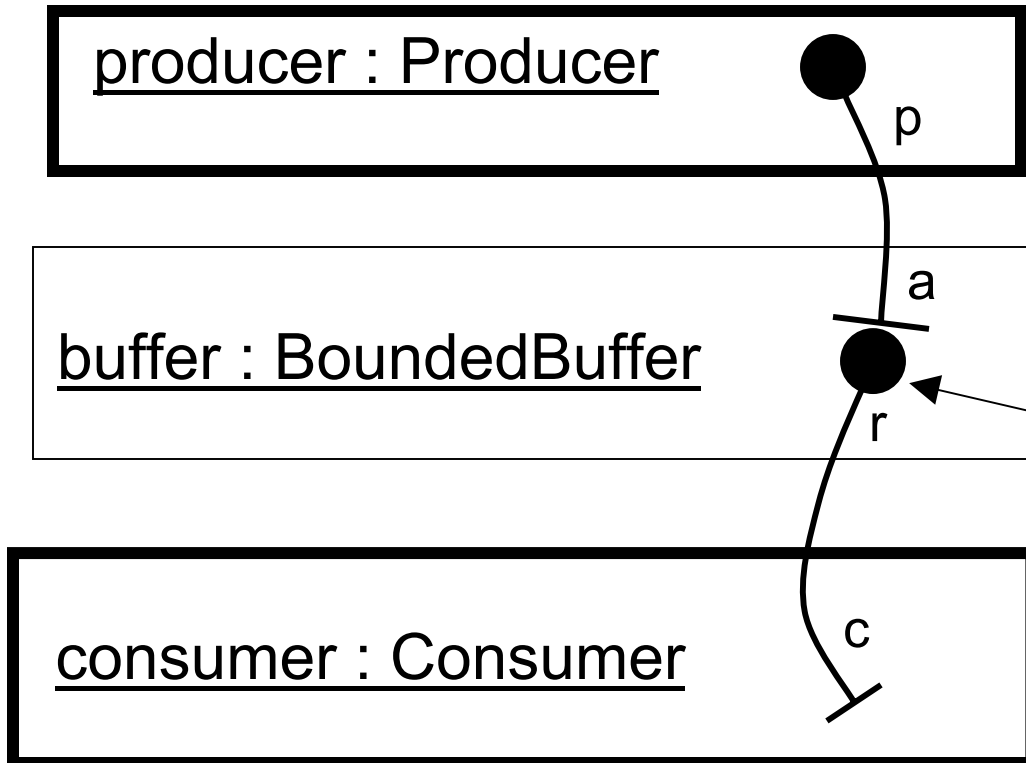
---

## ***Producer/Consumer UCM***

- From the problem definition, we know that the producer and consumer are concurrent objects, so we can factor the UCM
  - we add a precondition to the consumer's path, to document the condition under which a new scenario will start along that path

---

# ***Producer/Consumer UCM***



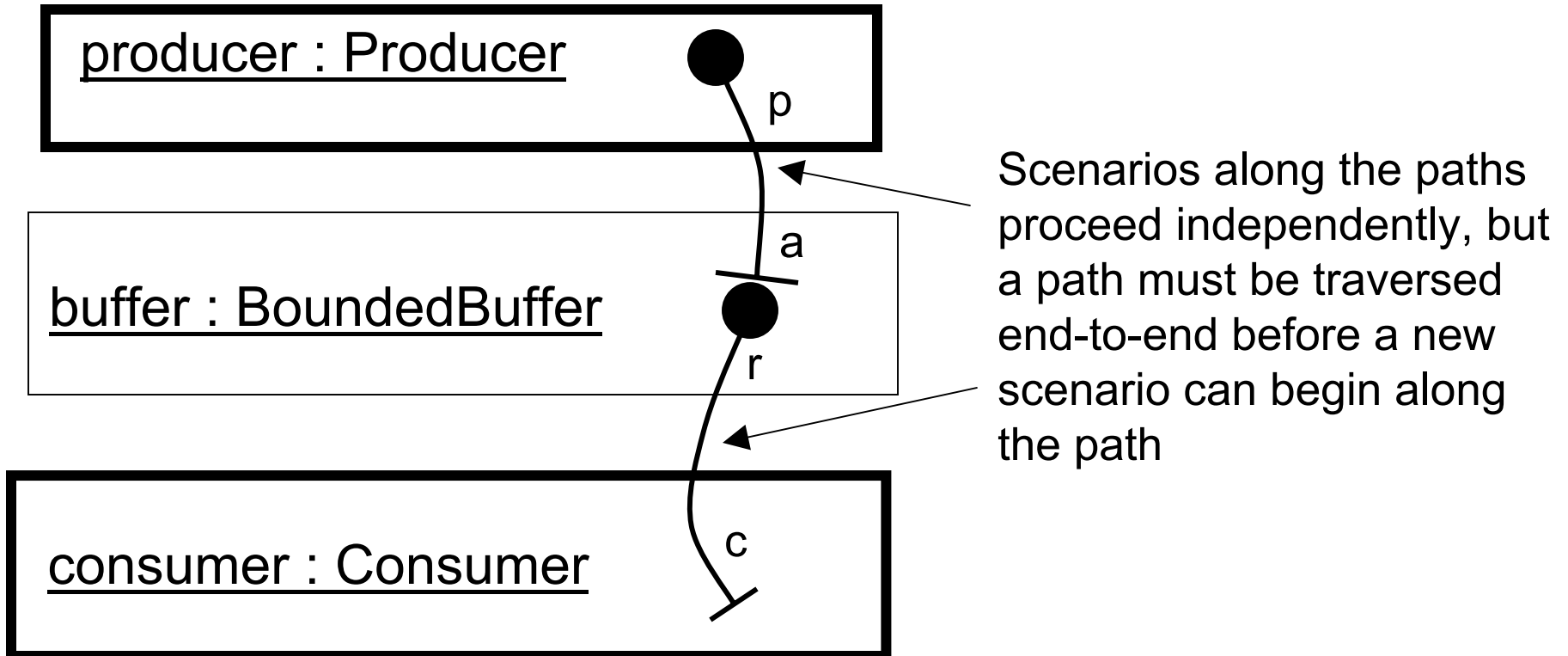
precondition for this path:  
there is at least one item  
in buffer

## Responsibilities:

p - produce item  
a - add item to buffer  
r - remove item from buffer  
c - consume item

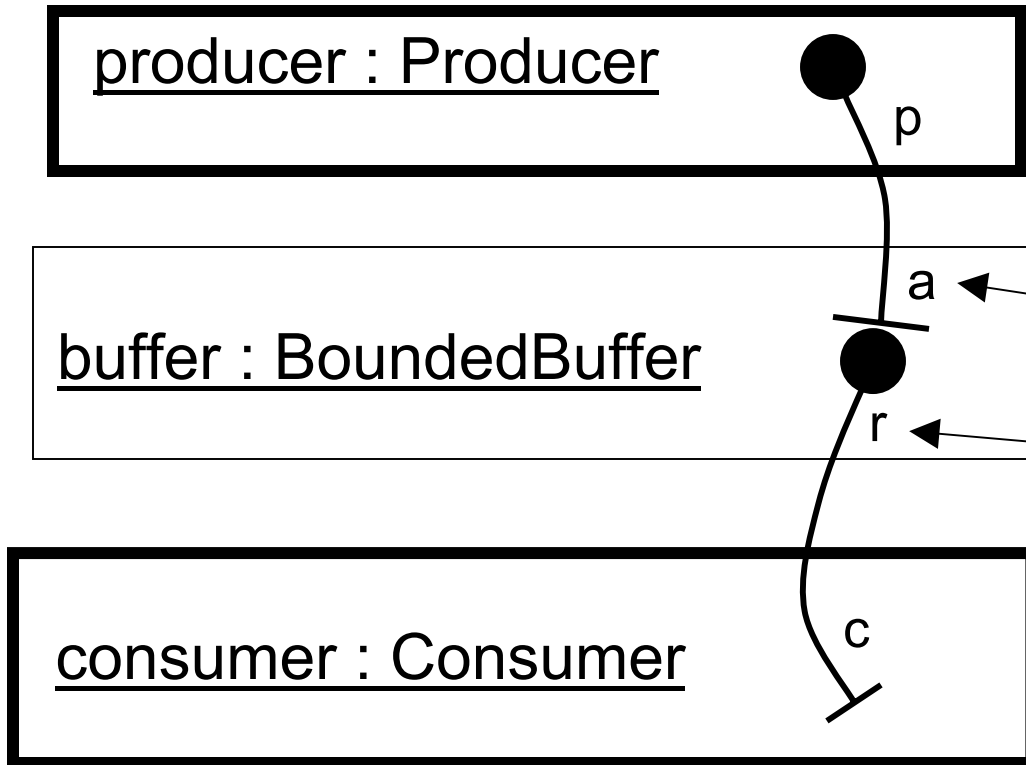
---

## ***Producer/Consumer UCM***



---

## ***Producer/Consumer UCM***



Two paths traversing the same passive component suggests the possibility of interference

- this issue must be dealt with during the design phase

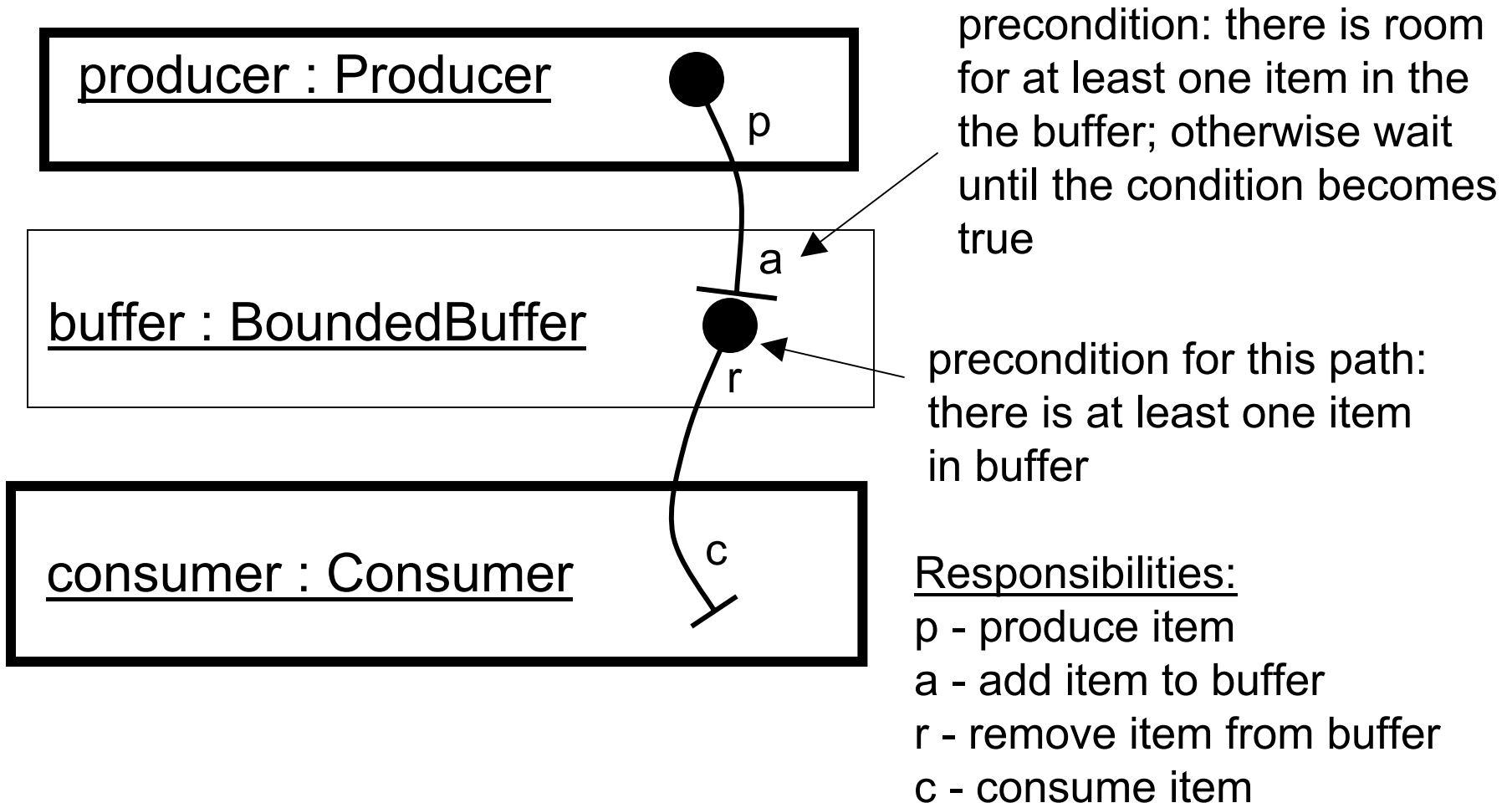
---

## ***Producer/Consumer UCM***

- Is there a way of showing that the producer should wait until there's room in the buffer before performing the "add item" responsibility?
- This could be depicted using elements of the UCM notation that we haven't seen, but it's probably better to deal with this by adding a prose precondition to the responsibility (less visual clutter)

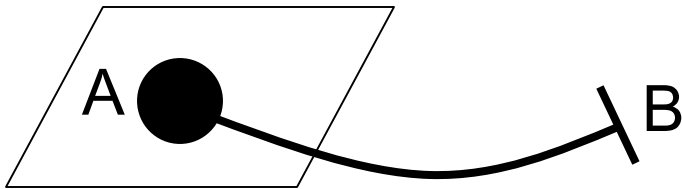
---

## ***Producer/Consumer UCM***



---

## *Paths and Active Objects*

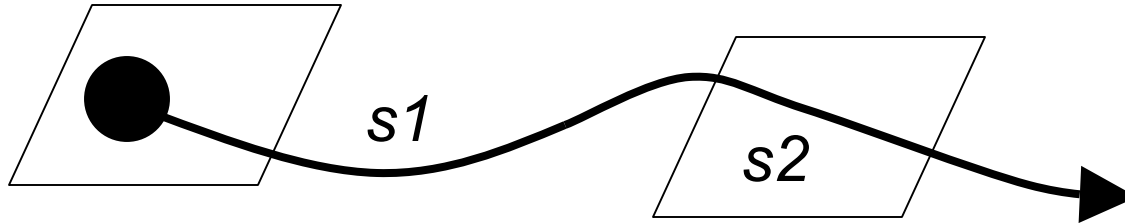


- An active object is an autonomous, self-directed component
- No concurrent scenarios along AB
  - active object must traverse AB end-to-end before the scenario can be repeated

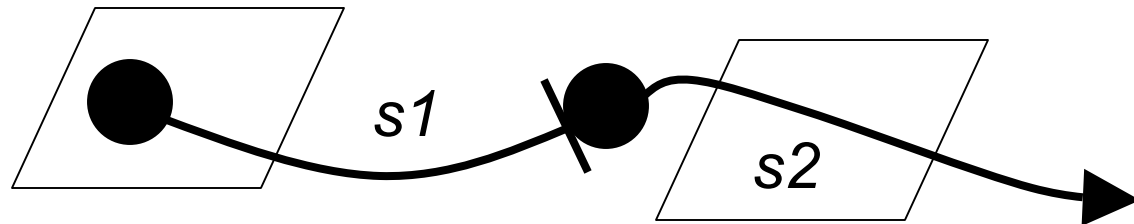


---

## *Paths and Active Objects*



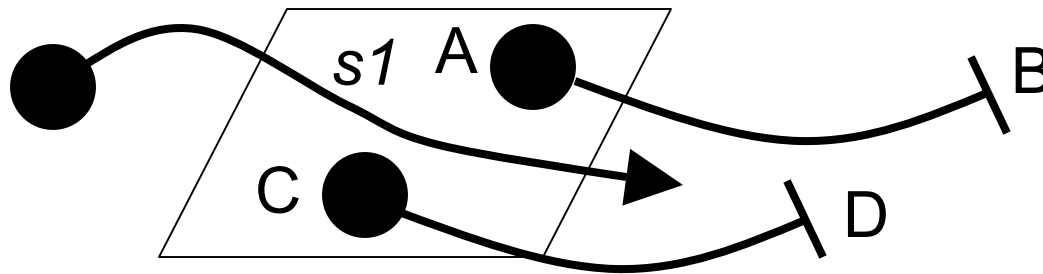
implies:



- Completion of *s1* triggers start of *s2*; however, *s1* can proceed concurrently with *s2*

---

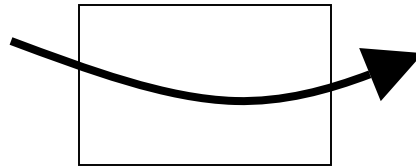
## *Multiple Paths Traversing Active Objects*



- Scenarios cannot proceed concurrently along paths AB, CD, and segment *s1*, because there is no concurrency within an active object

---

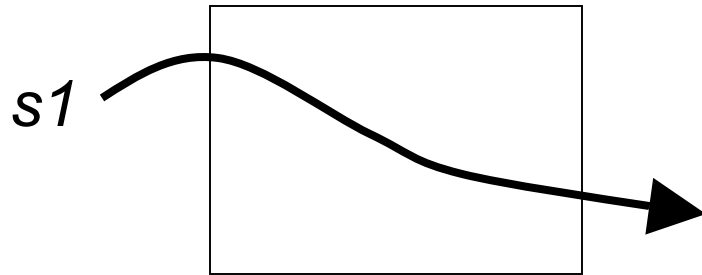
## ***Paths and Passive Objects***



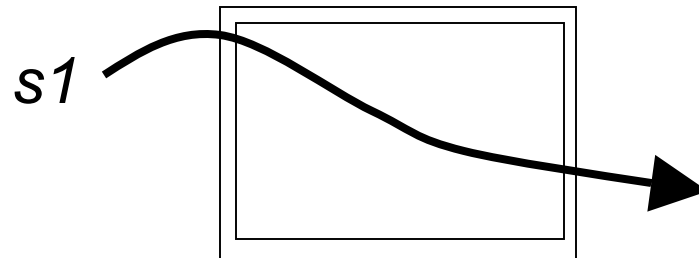
- A passive object has no independent thread of control
  - responsibilities are performed by interface operations in the context of an autonomous component; e.g., an active object
- We say that the object controls the sequence of responsibilities within its edges; however, this does not imply autonomous behaviour

---

## ***Concurrent Scenarios***



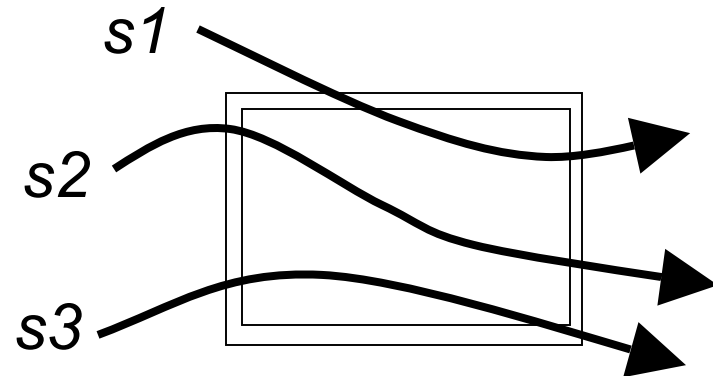
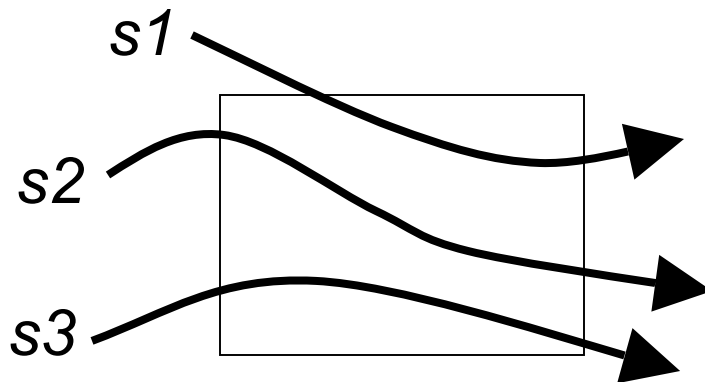
- Concurrent scenarios along segment *s1* may cause interference
- If mutual exclusion required, use a monitor to indicate this



---

## ***Concurrent Scenarios***

- Concurrent scenarios along segments *s1*, *s2*, *s3* may cause interference
- If mutual exclusion required, we can use a monitor to indicate this



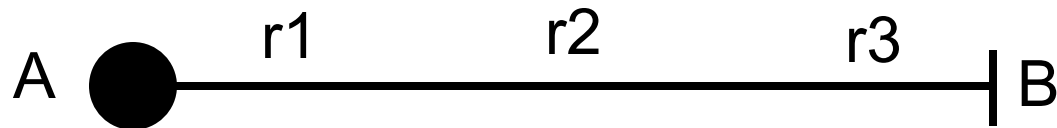
---

## ***UCMs and Concurrent Systems Design***

- UCMs help us reason from requirements (expressed as stimulus-response scenarios) to solutions (high-level designs)
- *"Let the paths tell you about the processes"*
  - identify concurrent scenarios
  - select active objects to achieve the required concurrency
  - passive objects tend to coalesce around active objects

---

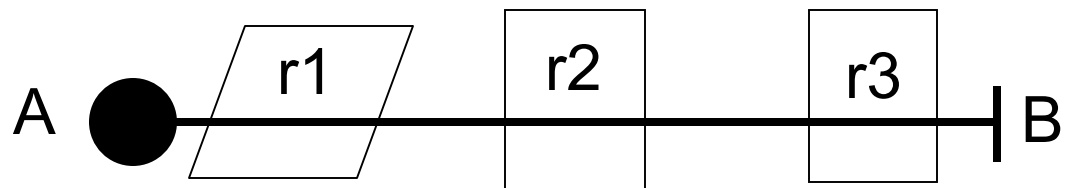
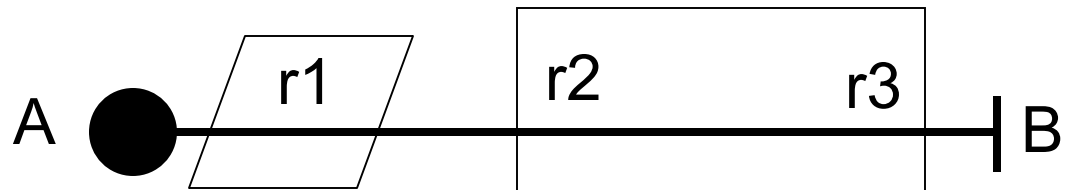
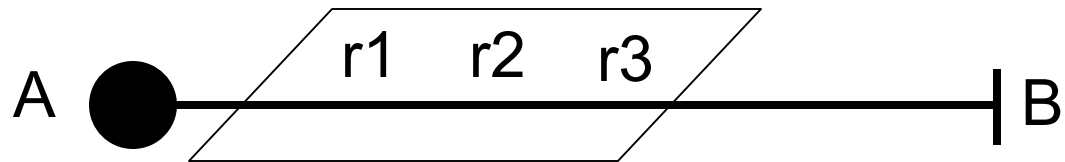
## ***Example 1***



- Requirement: the scenario along path AB must be completed before another scenario can begin along the same path
- Concurrency: one active object is sufficient

---

## ***Example 1: Possible Component Bindings***

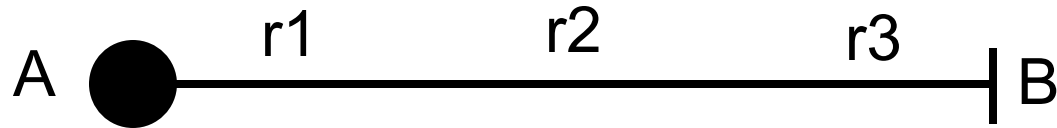


and others...

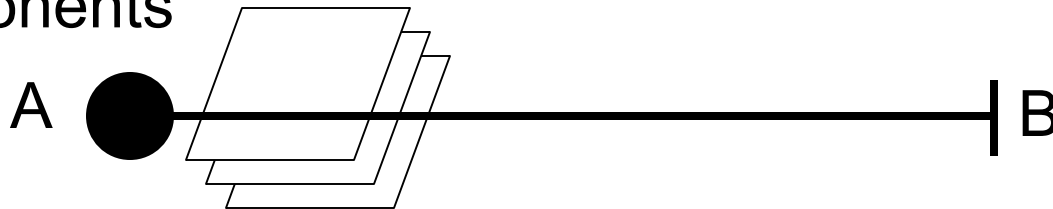


---

## Example 2

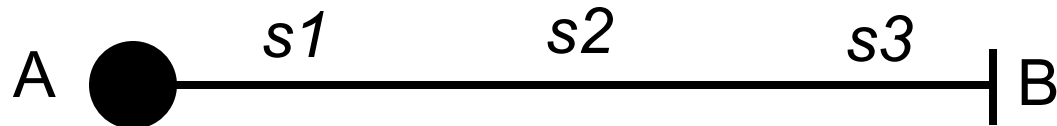


- Requirement: a new scenario along path AB can begin before another scenario along the same path finishes
- Concurrency:  $n$  active objects required for  $n$ -fold concurrency
- If r1, r2, and/or r3 are bound to passive objects, we must consider interference issues inside these components

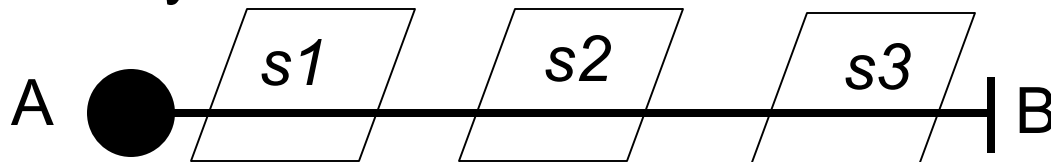


---

## *Example 3*

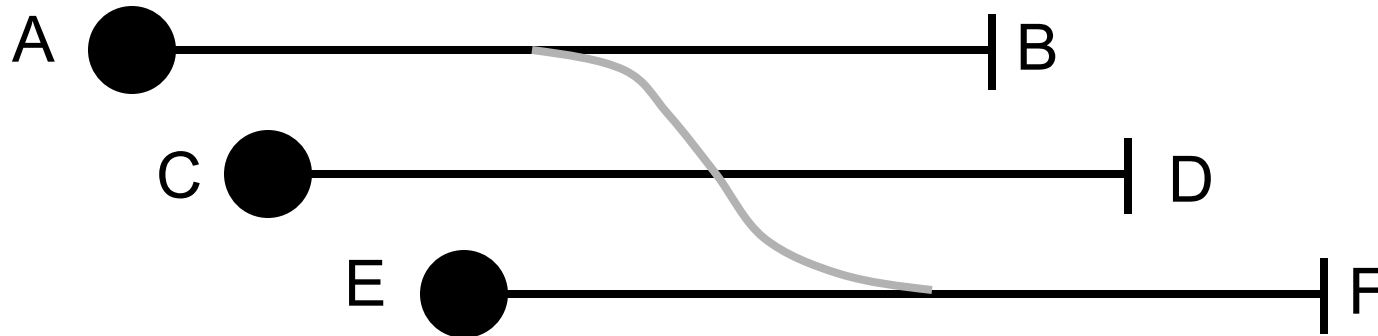


- Requirement: segments s1, s2, and s3 can proceed concurrently; a scenario along each segment must be completed before another scenario can begin along the same segment
- Concurrency: 3 active objects required for 3-fold concurrency



---

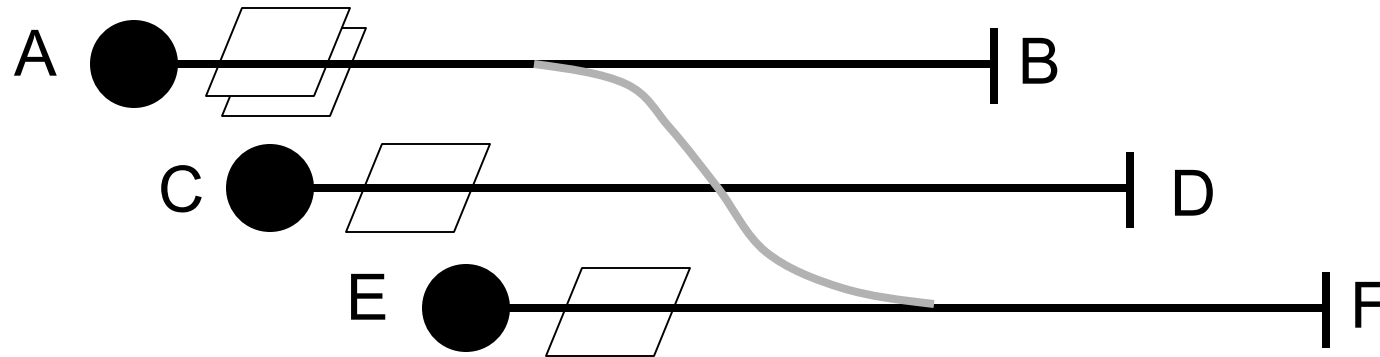
## *Example 4*



- UCM has independent paths AB, CD, EF, and AF

---

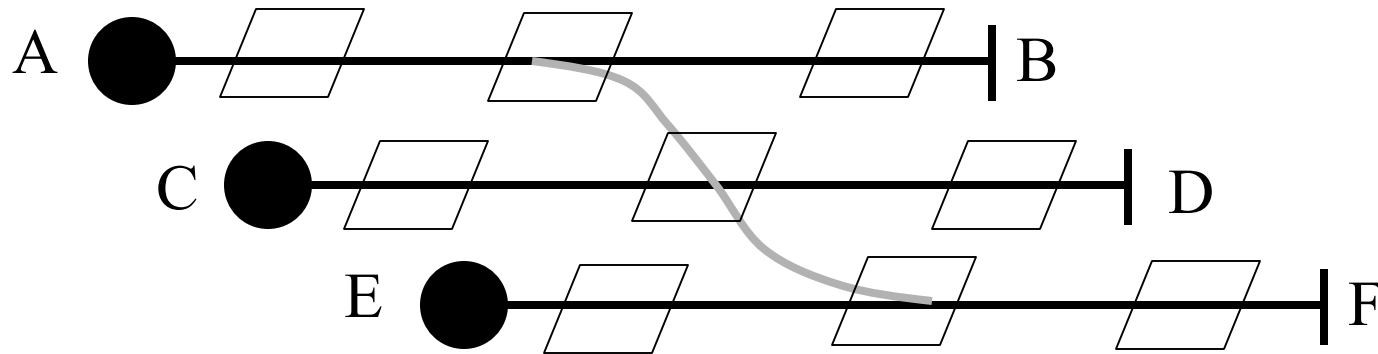
## *Example 4, cont'd*



- If preconditions on paths permit scenarios to start on any path at any time, we need at least fourfold concurrency
  - each thread handles one path, end to end

---

## *Example 4, cont'd*



- Additional threads will provide additional concurrency along paths
  - need to know more about requirements before we can identify additional threads
- Example: we determine that the concurrency along the paths justifies ninefold concurrency