BEHAVIORAL AND FLOW MODELING USING UML

UML Activity Diagrams

UML Statecharts

This slide deck assumes you have some idea of what UML is and the big ideas of modeling (abstraction). Do not start these slides until you read the introduction in the System Modeling chapter. Revisiting your SER215/SER216 notes wouldn't hurt either!

BEHAVIORAL MODELING

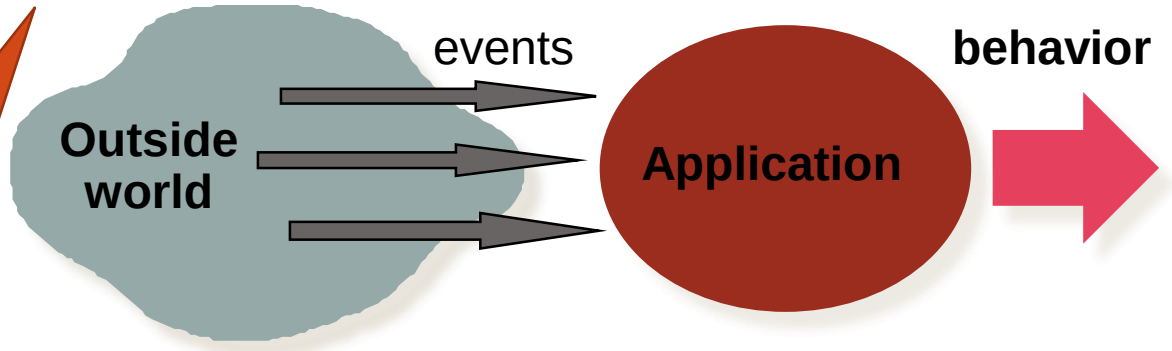
Our goal is to represent the behavior of a program as it executes, and do so in a high level way that is easy to understand. Once we have such a representation, we can manipulate it for our design instead of being stuck at the level of code or textual description.

Domain Models capture the object vocabulary of the problem space

These models describe objects, relationships, and (some) interactions *independent of a specific problem or solution*.

Examples: data dictionaries, glossaries, ontologies

Domain models are static though – we are interested in behavior at execution. Hence: behavioral modeling.



- Behavioral Models capture the *observable behaviors* of a system
- Models that capture how the system *reacts to* external stimuli.
- Examples: Stimulus-Response models, Event-oriented decomposition, **UML Statechart**

- Flow Models:
  - Capture problem-specific logic and data flows
  - Examples: Flowcharts, DataFlow Diagrams (DFDs), **UML Activity Diagrams**

Despite the different name, activity diagrams mostly work like flowcharts which are probably familiar with.

THE NEED FOR ACTIVITY DIAGRAMS

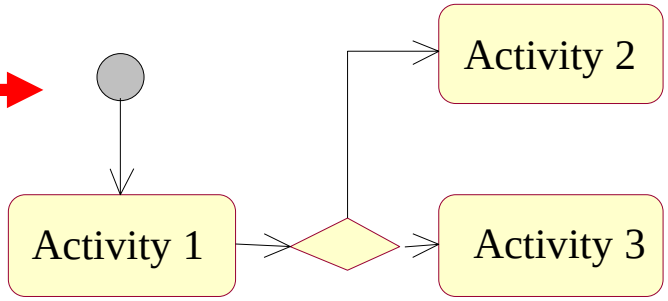
- An activity diagram in the use-case model can be used to capture the activities and actions performed in a use case.
- It is essentially a *flowchart,* showing flow of control from one activity or action to another.

Flow of Events

(This use case starts when the Registrar requests that the system close registration.)

1. The system checks to see if registration is in progress. If it is, then a message is displayed to the Registrar and the use case terminates. The Close Registration processing cannot be performed if registration is in progress.

2. For each course offering, the system checks if a professor has signed up to teach the course offering and at least three students have registered. If so, the system commits the course offering for each schedule that contains it.



Activity Diagrams show an "outside-in" view of flow – we are looking at the components of the system from outside.

THE NEED FOR STATECHARTS

Unlike activity diagrams, statecharts show an "inside-out" view of behavior– we are looking at state of components and the relative behavior.

- Definition of an object:
  - An object has state, behavior, and a unique identity
  - Object models are expressed using attributes, behaviors, & messages
    - An object has one or more attributes
    - An object exhibits one or more behaviors
    - Objects communicate via messages

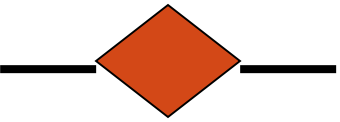
- What we need is a way to capture dynamic behaviors!
  - The way in which an object's state changes over time.
  - The object may change (transition) state for many different reasons, which we will model using events
    - One reason might be that a message (e.g., method call) arrives from another object!

- A statechart graphically captures this by representing each state as a node, with transitions between them showing how state changes in response to messages.

ACTIVITY DIAGRAMS & STATECHARTS SYNTAX

- **Action/State** – An *Action* is a step of an Activity.

- **Transition** – flow; a transition is triggered upon completion of some action, or some form of *event*.

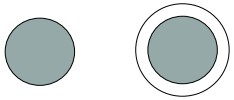


- **Decision/Merge point** – standard if-else style logic; also supports iteration. Guard conditions indicated in brackets in each transition.

- **Object node** – may be (typically not) included to show where an object's state may change. (Activity diagrams)

- **Synchronization bar** – supports fork/join semantics for concurrent processing. (Activity diagrams – usually)

- **Swimlanes** – partition actions by responsibility, not thread. (Activity diagrams)



- **States** – initial, final, pseudostates.

ACTIVITY DIAGRAM

Action
A step in the flow of events

Decision
Flows split based on a guard condition

Fork
Beginning of concurrent flows

Join
End of concurrent flow

Flow
Show the sequence of activities

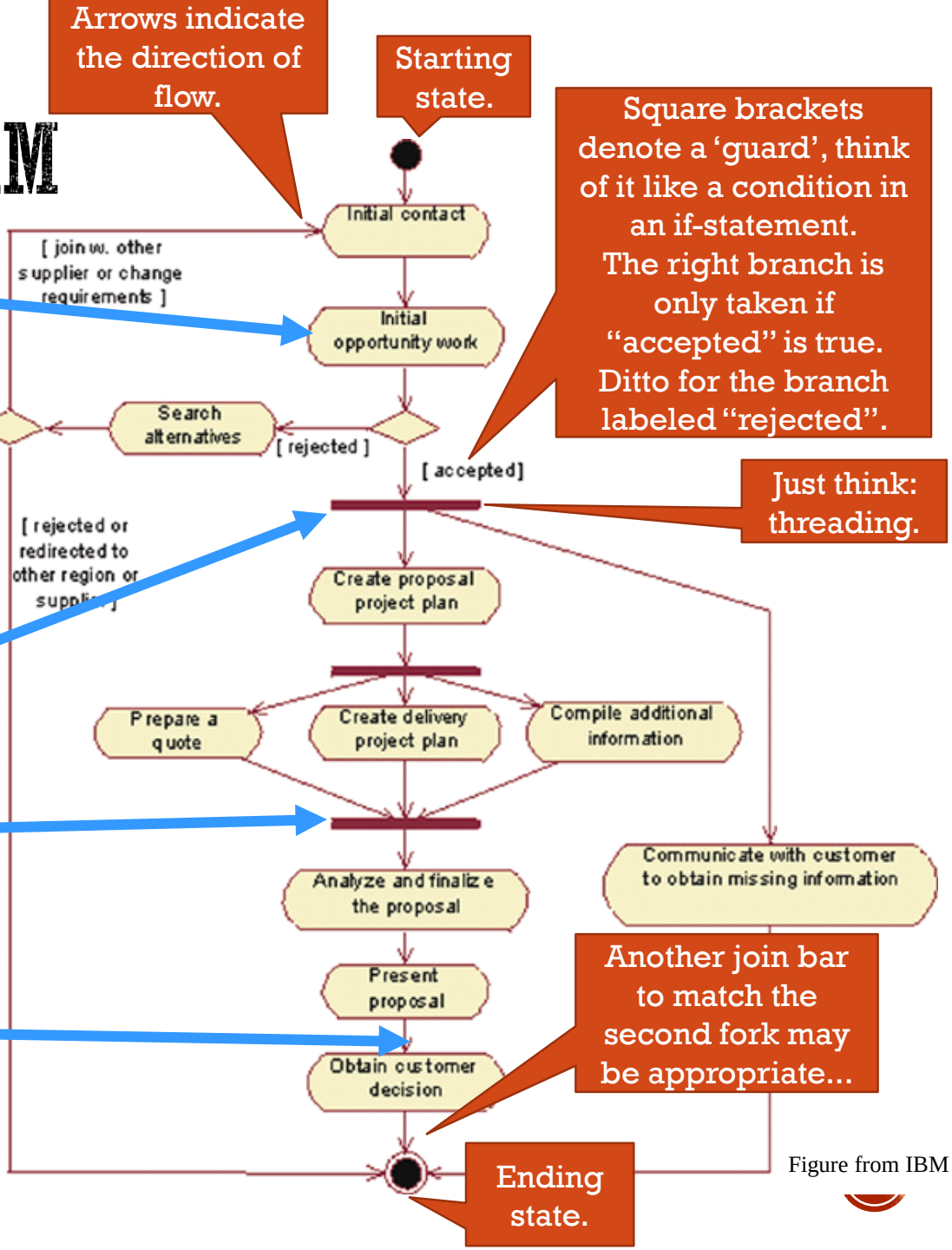
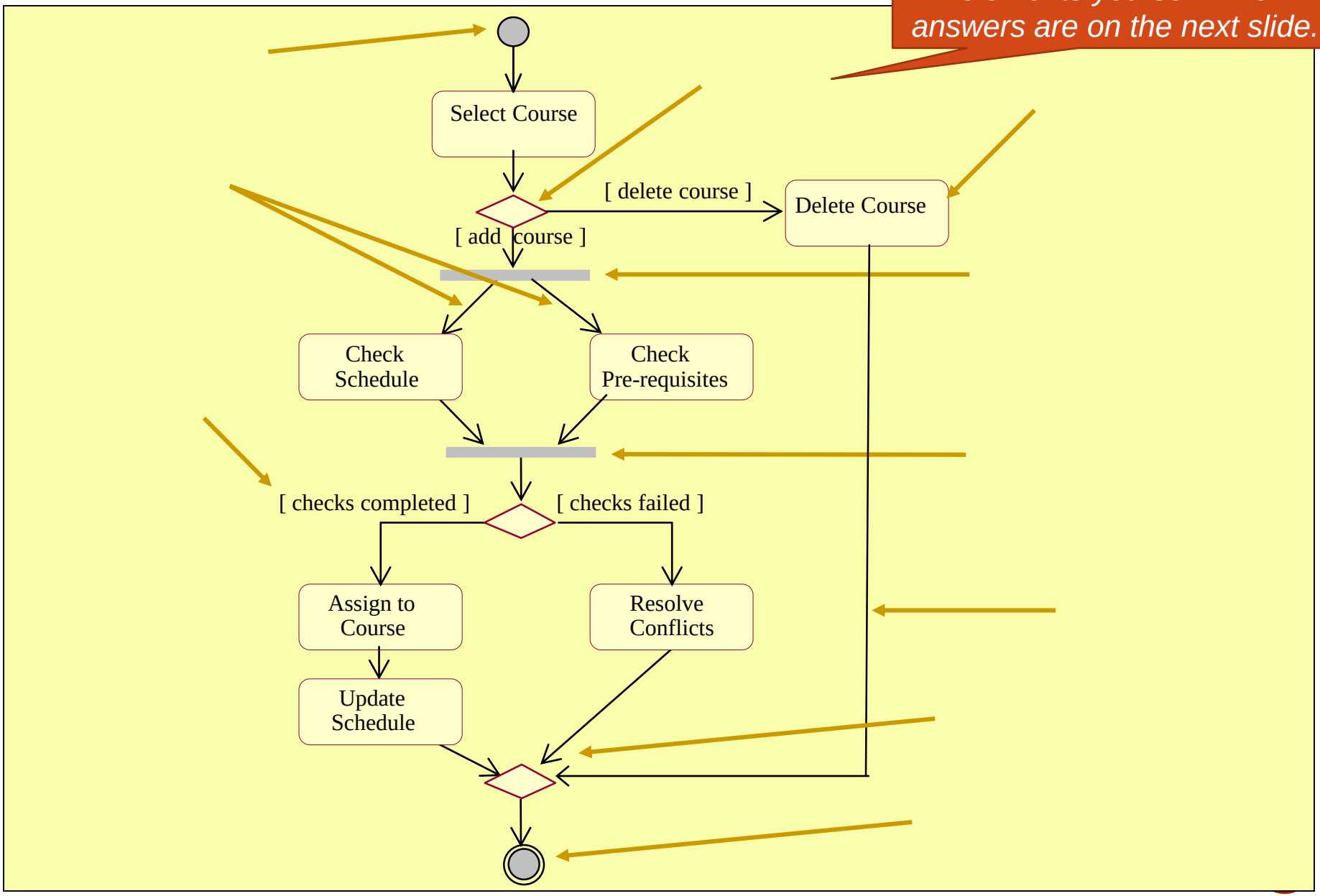


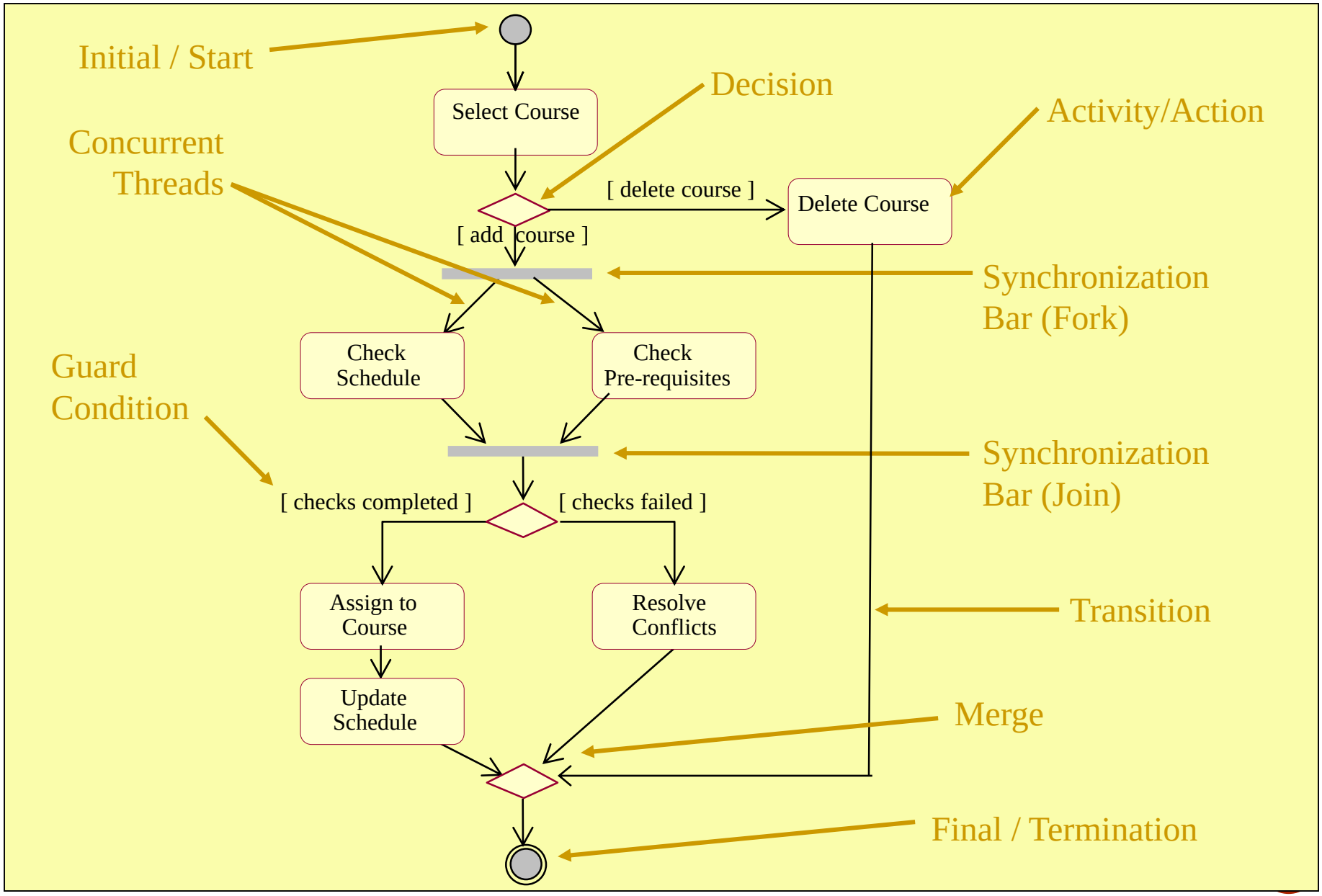
Figure from IBM

EXERCISE: ACTIVITY DIAGRAM

As an exercise, try labeling the types of these diagram elements yourself. The answers are on the next slide.



EXERCISE: ACTIVITY DIAGRAM



BASIC ACTIVITY DIAGRAMS RULES (REFERENCE)

- There should be exactly one *initial* node.
- There should be one or more *final* nodes. (Zero represents a continuous system.)
- *Decision* nodes should have one incoming transition, and any number of outgoing transitions.
- The paths out of a decision node typically converge in a *merge* node, although not immediately.
- *Fork* bars should correspond directly to *join* bars.
- **Do not mix control flow and concurrency regions!**
- No *decision/merge* node can have multiple incoming transitions and multiple outgoing transitions at the same time.
- No *fork/join* bar can have with multiple incoming transitions and multiple outgoing transitions at the same time.
- *Action* nodes must not have multiple incoming or outgoing transitions.

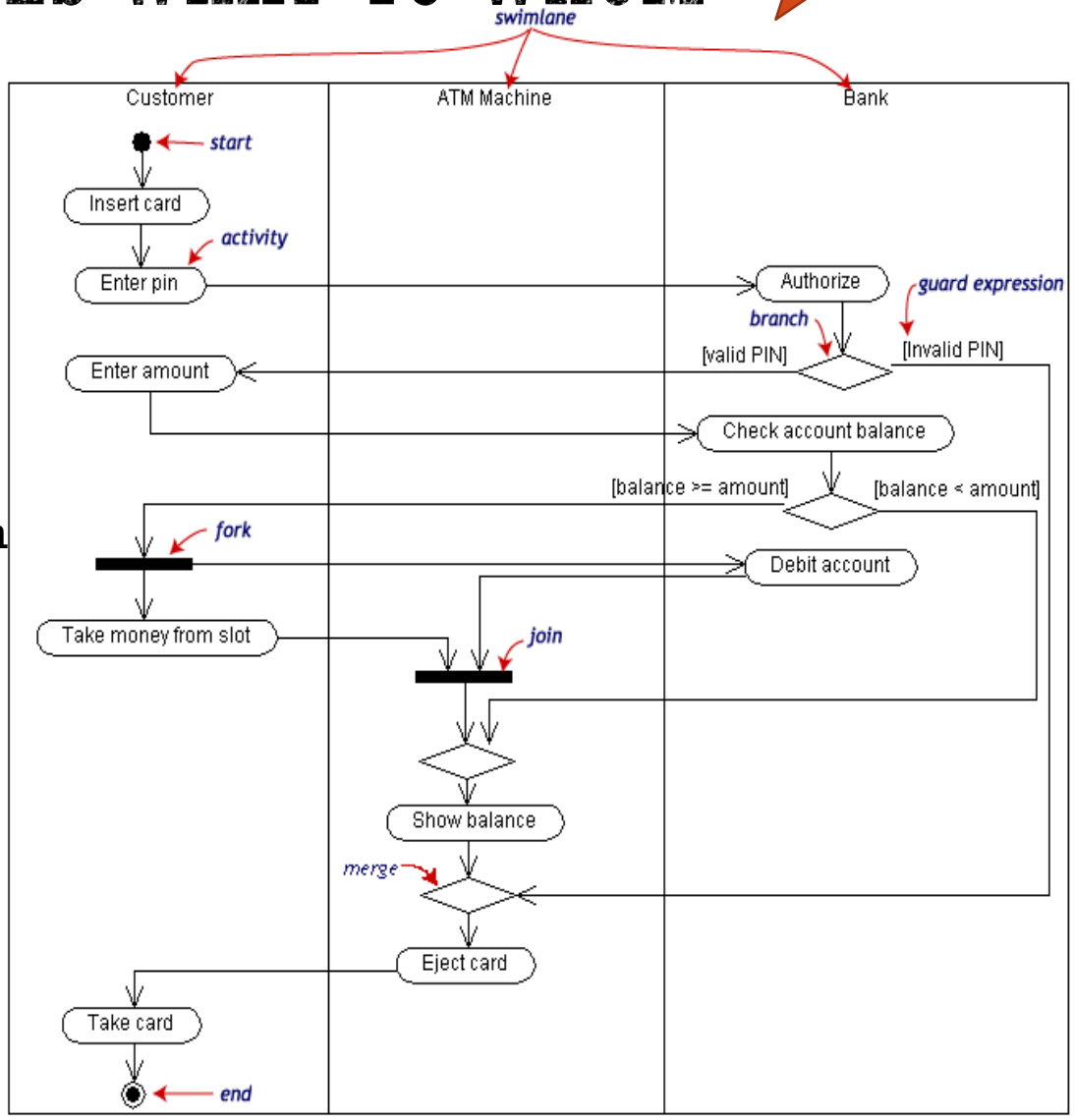
You don't have to read these right now. Attempt the assignment, and then use this as a checklist to review your submission. Repeat this with your UML diagrams until the general rules become natural.

Also, try to discover the reasoning behind these rules. E.g., a diagram should have exactly one initial node so it has an unambiguous starting place.

This diagram has three swimlanes. The syntax isn't complicated – the effort is really to map the responsibilities.

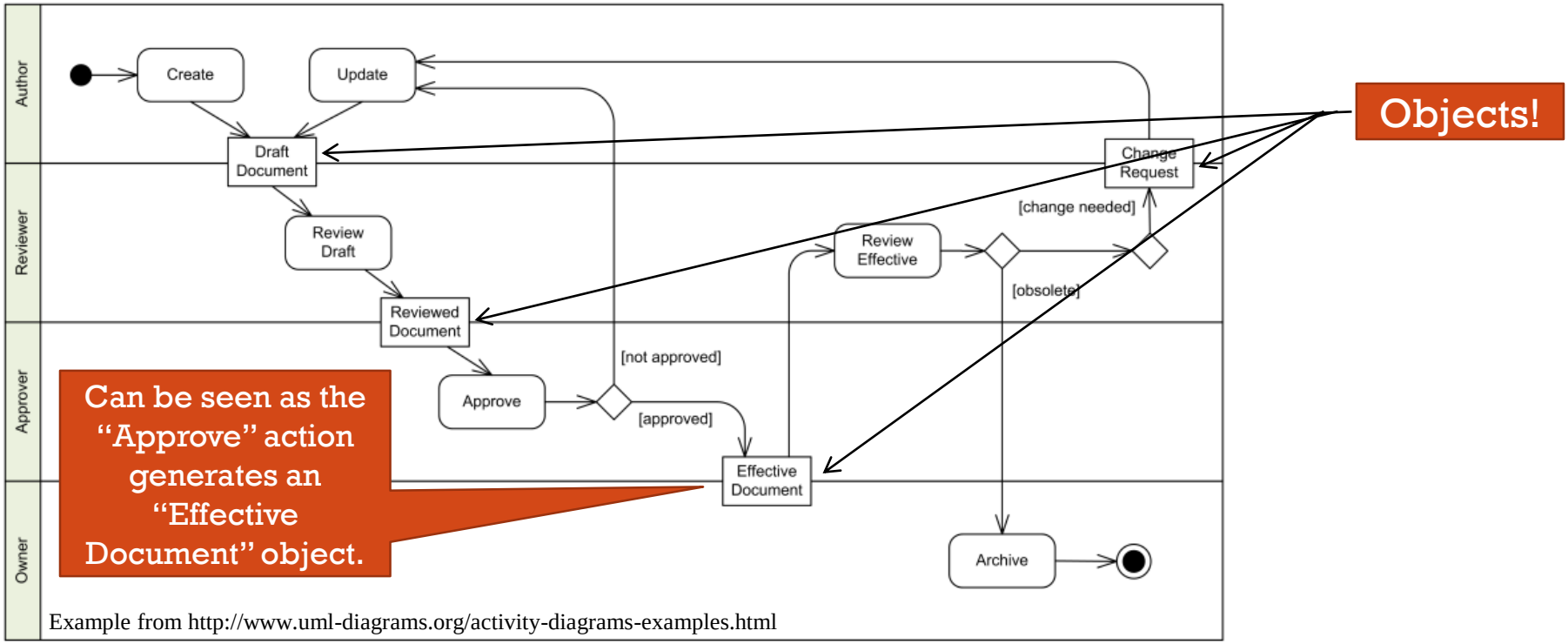
MAPPING WHO DOES WHAT TO WHOM

- Examples so far show us what actions happen - but "WHO" does each action?
- *Swimlanes:*
  - Partition activities according to who does them.
  - Who can be actors, system components – whatever.
  - When is indicated top-to-bottom or left-to-right.
- And to whom?
  - Activity diagrams can also show relationships to objects that are affected by actions – see next slide.



EXAMPLE: ACTIVITY DIAGRAM W/ OBJECTS

Document management process



Example from http://www.uml-diagrams.org/activity-diagrams-examples.html

- Rarely do we see an Object on an Activity diagram
  - Activity diagrams are for *flow* modeling
  - Object behaviors are typically modeled using an UML *Statechart*
- But, yes, objects can be shown:
  - Do so if it is important to show critical object state changes or dataflow in the activity diagram context!

ACTIVITY DIAGRAM SUMMARY

- Pros:
  - Map use case scenarios directly on to actions
  - Intuitive for most procedural programmers
  - Includes constructs for concurrency and task assignment
  - Includes constructs for top down decomposition (activity frames)
- Cons:
  - Some confusion of the relationship between activity diagrams and statecharts
    - Think: high-level process vs state of object
  - Some changes in terminology from UML 1.5 to UML 2.0
- Recommendation: Useful early in analysis, after use cases but before interaction diagrams

INTRODUCING STATECHARTS

From a previous slide, remember that our goal is to model how events change the state of an object.

- There are three elements of a statechart diagram:
  - *States*:
    - Condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event. That is: has a well-defined ongoing state.
    - Some states are "special"
  - *Events*:
    - Internal and external occurrences that impact or are generated by an object
  - *Transitions*:
    - Response of an object based on events and present state
    - May have *Guards* or *Activities* associated with them
    - On such a change of state, the transition is said to *fire*

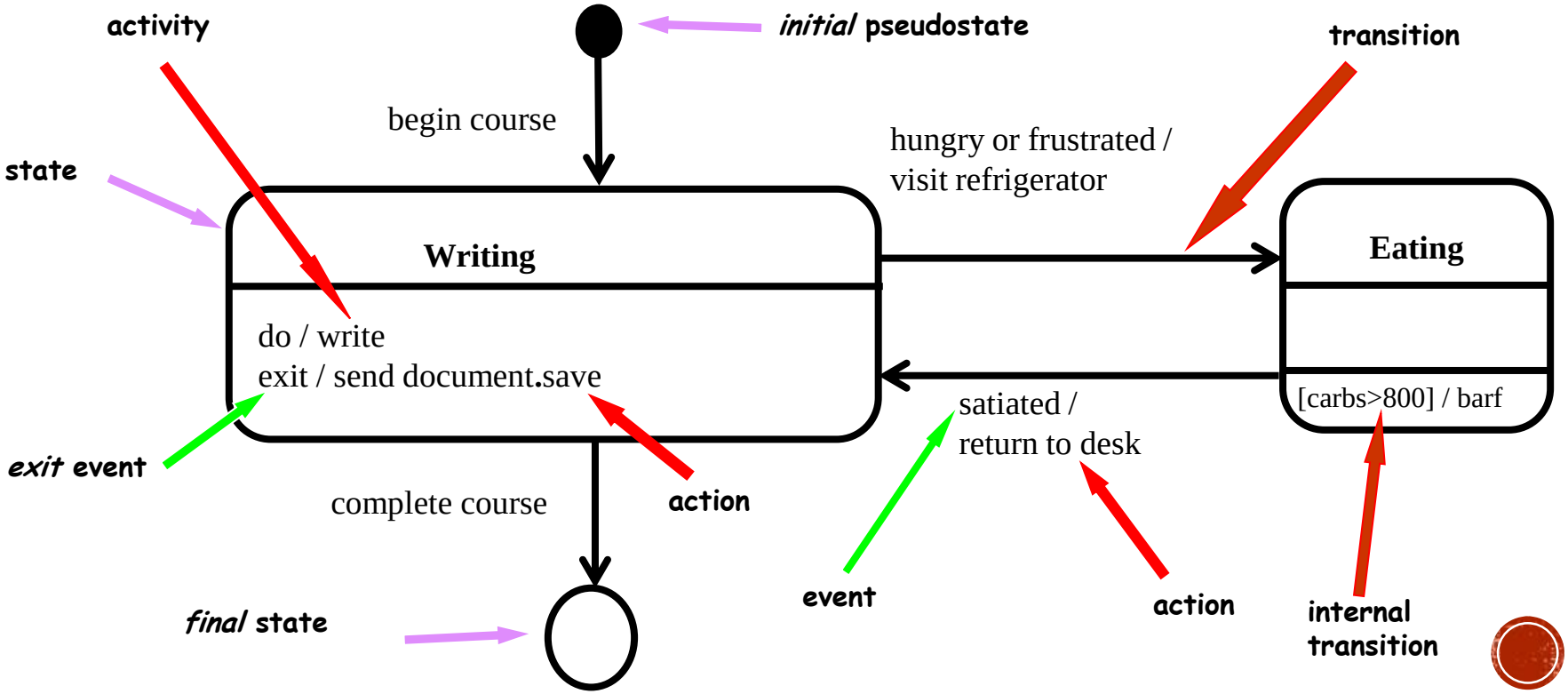
Careful – events and transition do not always go to together! There is such a thing as a transition without an event (trigger); that transition occurs when the last state completes.

Side note: some of these ideas you may already have seen. Statecharts are similar to the idea of state machines which are introduced in digital logic.

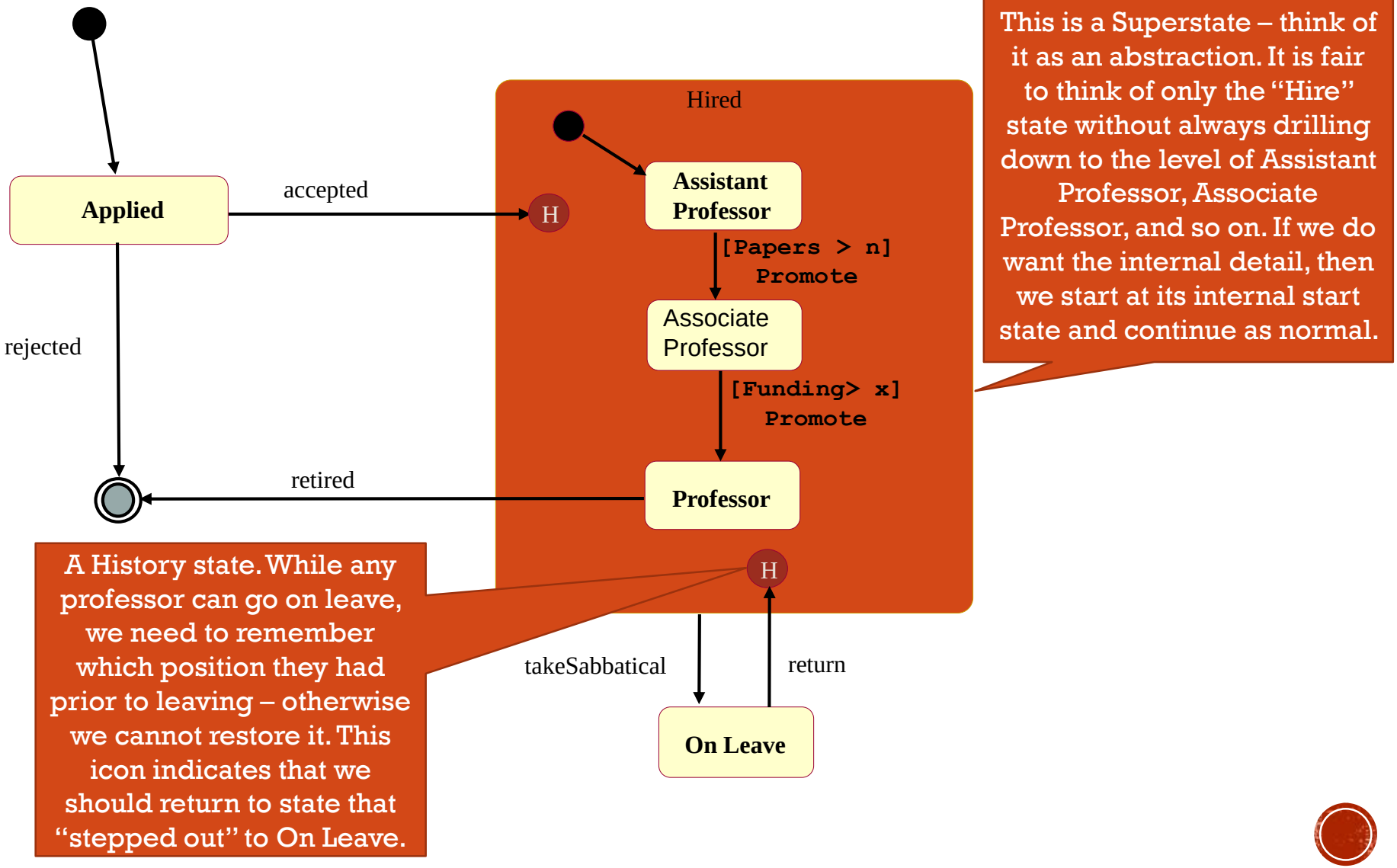
UML STATECHARTS 1-SLIDE CHEATSHEET

A statechart diagram models dynamic behavior over state. It specifies the sequence of states in which an object can exist:

- The events and conditions that cause the object to reach those states
- The actions that take place when those states are reached

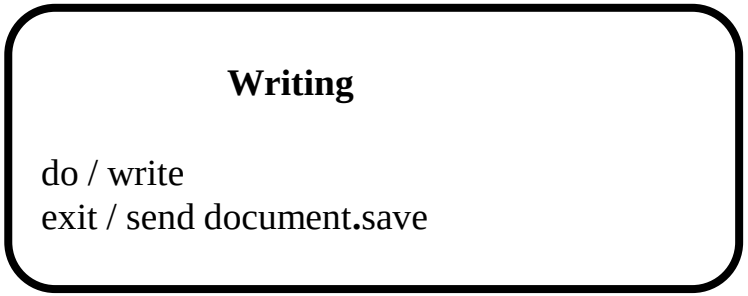


EXAMPLE: STATECHART FOR FACULTY EMPLOYMENT



ANATOMY OF A STATE

- A State can have several parts:
  - Name
  - Entry/exit actions
    - Entry action: **entry** / *action*
    - Exit action: **exit** / *action*
  - Internal transitions
    - *These bypass the entry/exit actions and guard conditions*
  - Substate, deferred events, & other things we won't use



- When in a given state, an object may be *active*
  - *Active* means it's doing something, an *internal behavior*
  - That action is interruptable, or may run to completion
    - It may generate a *completion event* which results in a transition
    - If the state exits via another event, then the action is terminated

These have the same meaning as in activity diagrams.

SPECIAL STATES

- The initial state is entered when an object is created.
  - Exactly one initial state is permitted (mandatory).
  - The initial state is represented as a solid circle.
- A final state indicates the end of life for an object.
  - A final state is optional.
  - A final state is indicated by a bull's eye.
  - More than one final state may exist.

- Other special (pseudo) states exist in UML, like the History state from before, but are not relevant to what we will do with Statecharts.

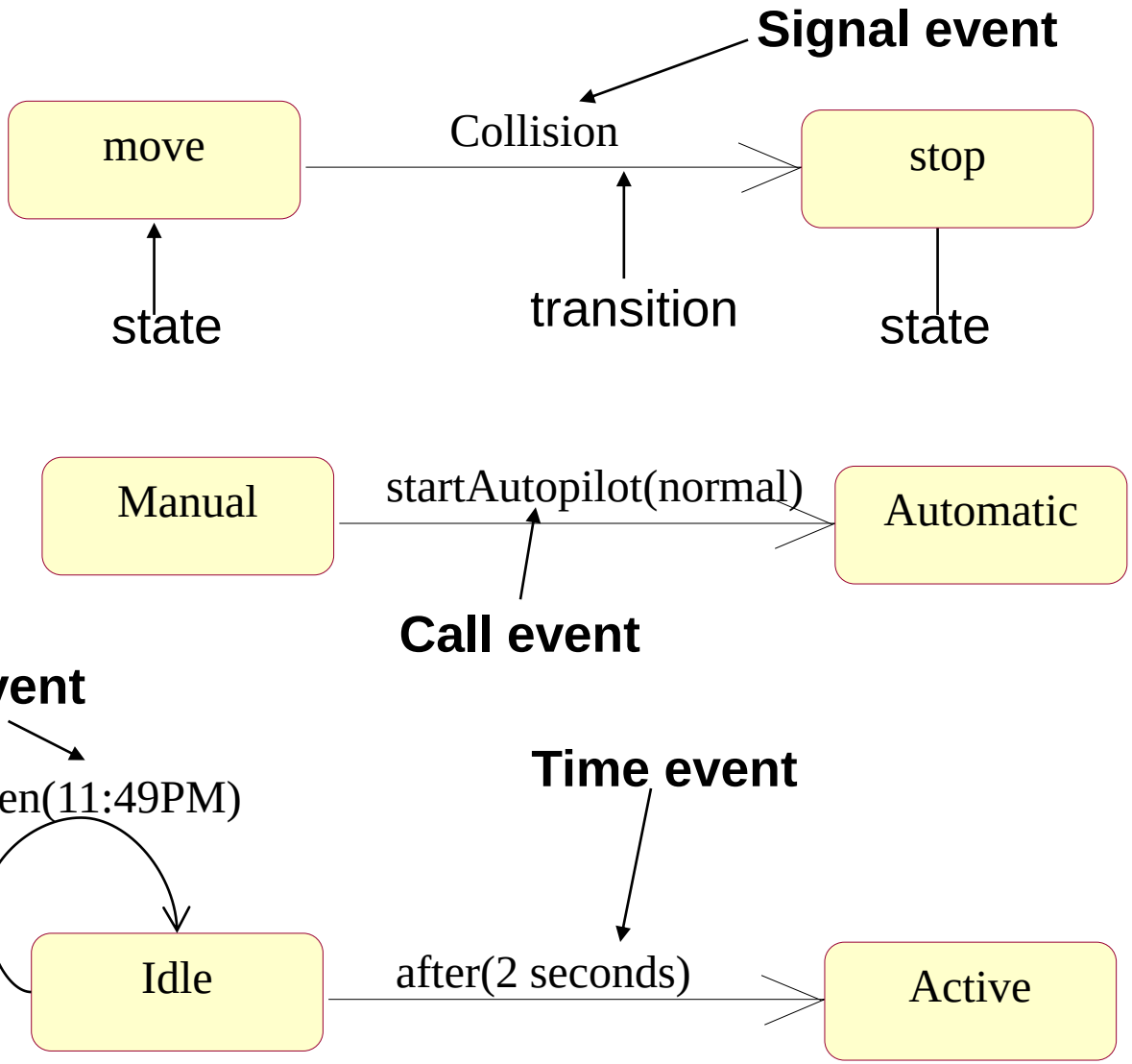
WHAT ARE EVENTS?

- A significant occurrence that has a location in time and space.
  - An event is an occurrence of a stimulus that can trigger a state change
  - Example: Temperature drops below 65 degrees

UML defines 4 event types:

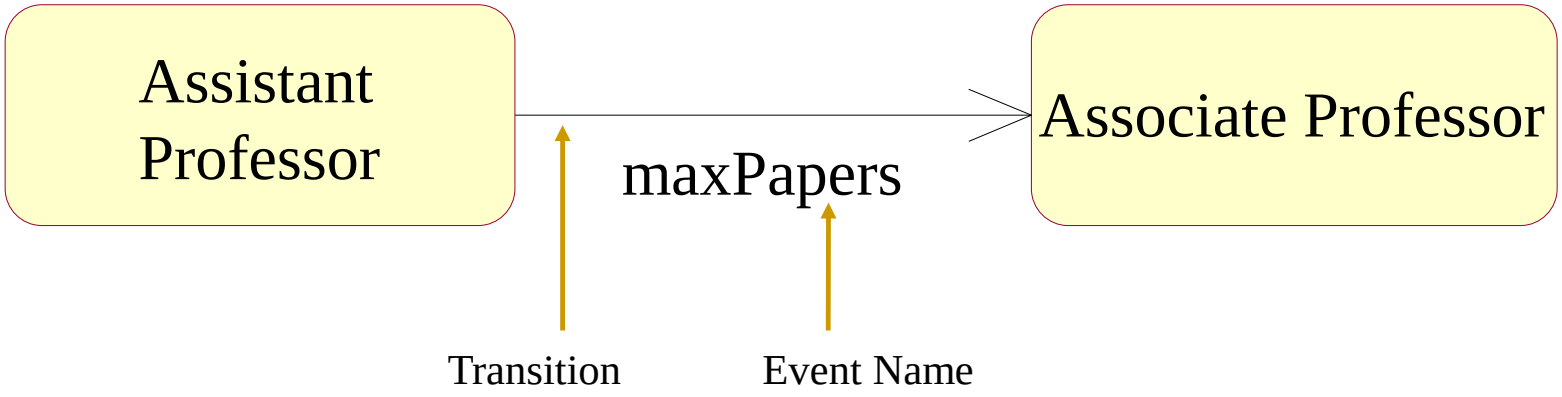
- signal event: sname(p:T)
  - Receipt of an explicit, named, asynch communication among objects
- call event: op(p:T)
  - Receipt of an explicit synchronous request among objects that wait for a response
- change event: when (exp)
  - A change in value of a Boolean expression (continuous check)
- time event: after (time)
  - Arrival of an absolute time or passage of a relative amount of time

EVENT AUTOMOTIVE EXAMPLES



WHAT ARE TRANSITIONS?

- A *transition* is a change from an originating state to a successor state as a result of some stimulus.
  - The successor state could be the originating state.
  - Transitions typically take place in response to an event.
  - Transitions may happen if an object completes an *internal behavior*.
- Transitions can be labeled with event names
  - But keep in mind that events are transitions are not the same thing!
    - Event: something happened, or changed, or was communicated in the world
    - Transition: specific to an object, indicates a change in its state



WHAT ARE GUARDS/ACTIONS/ACTIVITIES?

- **Transition:** *event* [*guard*] / *activity*

- A *Guard* is a boolean condition that may optionally be present on a transition
  - If the condition if false, no transition

- An *Activity* is an optional behavior that is executed during a transition.

- *Note: UML 1.x used the term "action" for a behavior on a transition, and used "activity" for ongoing tasks.*
  - Activity (within a state) could be interrupted
  - Actions (on a transition) could not (run-to-completion semantics)

INTERNAL VERSUS EXTERNAL TRANSITIONS

- UML also defines some special Activities:
  - An *entry activity* executes whenever you enter a state
  - An *exit activity* executes whenever you leave a state
  - An *internal activity* is one where a SC can react to an event without leaving the state
    - This is slightly different than a self-transition in that no entry/exit activities are executed and guard transition is evaluated.
      - Internal Transitions can have a different guard condition though

InReverse

onEntry: *"beep"*
onExit: *"beep off"*
Internal transition: put in reverse

HOW ALL THIS WORKS

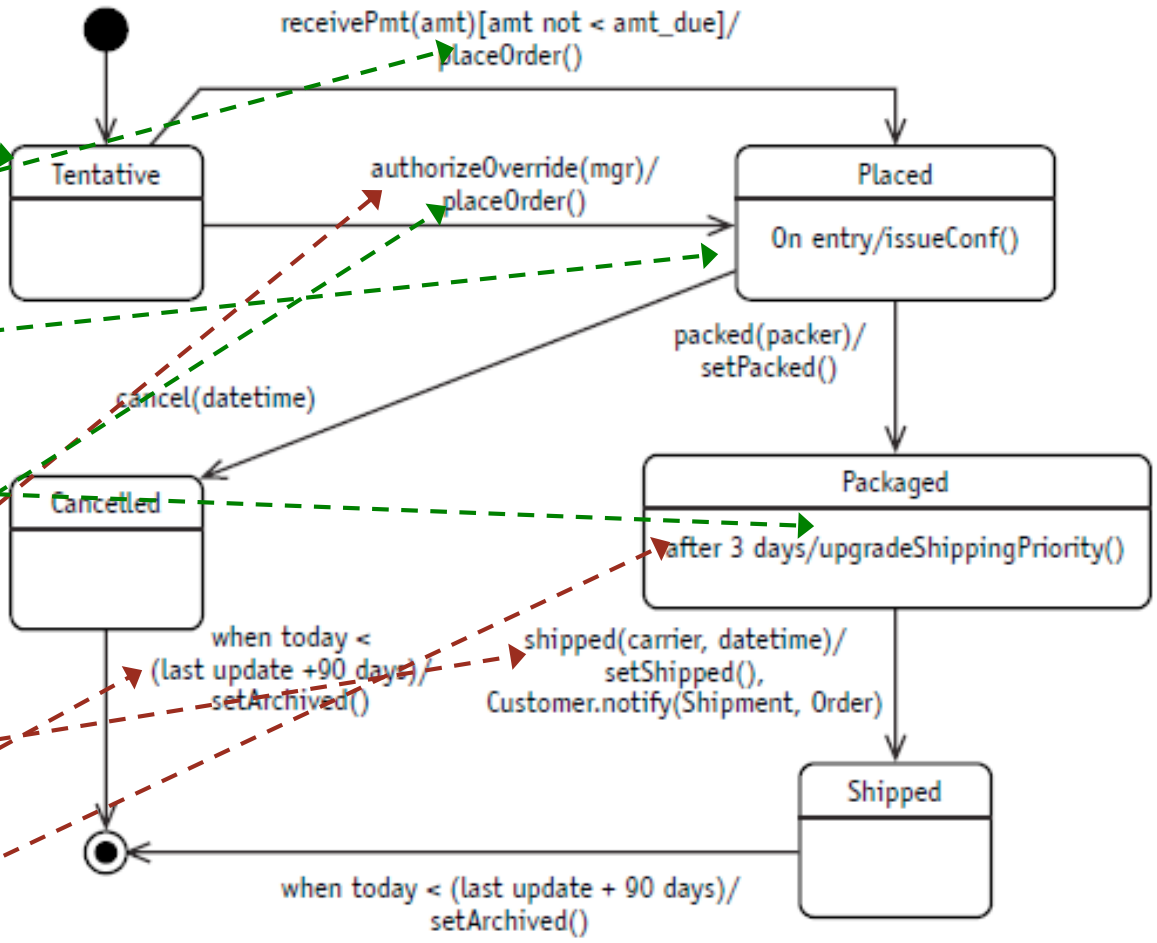
1. An instance begins life in the state pointed to by the initial pseudostate
2. Entry actions are executed every time the state is entered
3. Exit actions are executed every time the state is exited
4. The guard condition is evaluated *after* the event instance occurs
5. If an event instance matches a transition label, that transition fires if its guard condition allows
6. If an internal behavior is running when a trigging event instance occurs, the activity is terminated
7. The completion of all internal behavior in a state is considered an event—a *completion* event
8. A transition without an explicit event label is triggered by the completion event instance if its guard condition allows
9. Event instances that don't match some transition label are ignored and lost

- *Note the synergy with Activity diagrams; Statecharts use many of the same concepts and symbols, but turn an Activity diagram "inside out" to view interactions from the object's perspective!*

STATECHART EXAMPLE

This slide is more involved than our previous examples. At first your real take away from this diagram should be seeing all of the syntax in context.

- Features:
  - Multiple transitions
  - Guard conditions
  - On entry
  - Internal transition
  - Actions on transitions
  - Events
    - Signal
    - Call
    - Change
    - Timer



Statechart taken from http://blog.jstoutenburg.com/2010/12/uml-statechart-diagram.html

MORE STATECHART CONCEPTS

- Exceptional conditions:
  - Do you have to specify transitions for every state that correspond to every single possible (or impossible) thing that could ever happen in the Universe?
- Example:
  - "your Bank Account balance state goes from Good to Bad if a bunch of 3-legged martians riding flying camels descend from the sky and deep-fry your bank"
  - How would you specify a transition for this? You shouldn't!!!
  - However, you can characterize all "unexpected" states and transitions as "exceptional"
    - All transitions you do not specify as valid are assumed invalid
    - You have three options:
      - test for, and disallow, the transition
      - do not test for, and risk, an invalid transition
      - throw an exception - "I've reached an unexpected state"

A simple approach is to define an error state that invalid transitions enter.

STATECHART SUMMARY

- Think about the set of different states that object can be in at any given time – name them!
- Think about how many final states the object may have.
  - Maybe it is zero - an object that "lasts forever"
- Think about how you get from one state to another
  - This is the "how" part of an object - the dynamic behavior
  - It is important to capture all of the transitions!
  - There can be more than 1 way to get from one state to another!
  - A transition may go from one state back to itself!
- Think about what actions occur as "side-effects", and when
  - Are there expected/unexpected "weird" situations?

- <u>For your project</u>:

1. You can start modeling a domain by identifying and describing objects
2. Statecharts describe object state and behaviors
3. Nice jumping off point from Activity diagrams, syntactically & semantically