Chapter 14 – Activity Modeling and Agile (An Introduction)

Dr John H Robb, PMP

UT Arlington Computer Science and Engineering

Key Takeaway Points

- Activity modeling deals with the modeling of the information processing activities of an application or a system that exhibits sequencing, branching, concurrency, as well as synchronous and asynchronous behavior.
- Activity modeling is useful for the modeling and design of transformational systems.

What Is Activity Modeling

- Activity modeling focuses on modeling and design for
  - complex information processing activities and operations
  - information flows and object flows among the activities
  - branching according to decisions
  - synchronization, concurrency, forking, and joining control flows
  - workflow among the various departments or subsystems

Why Activity Modeling

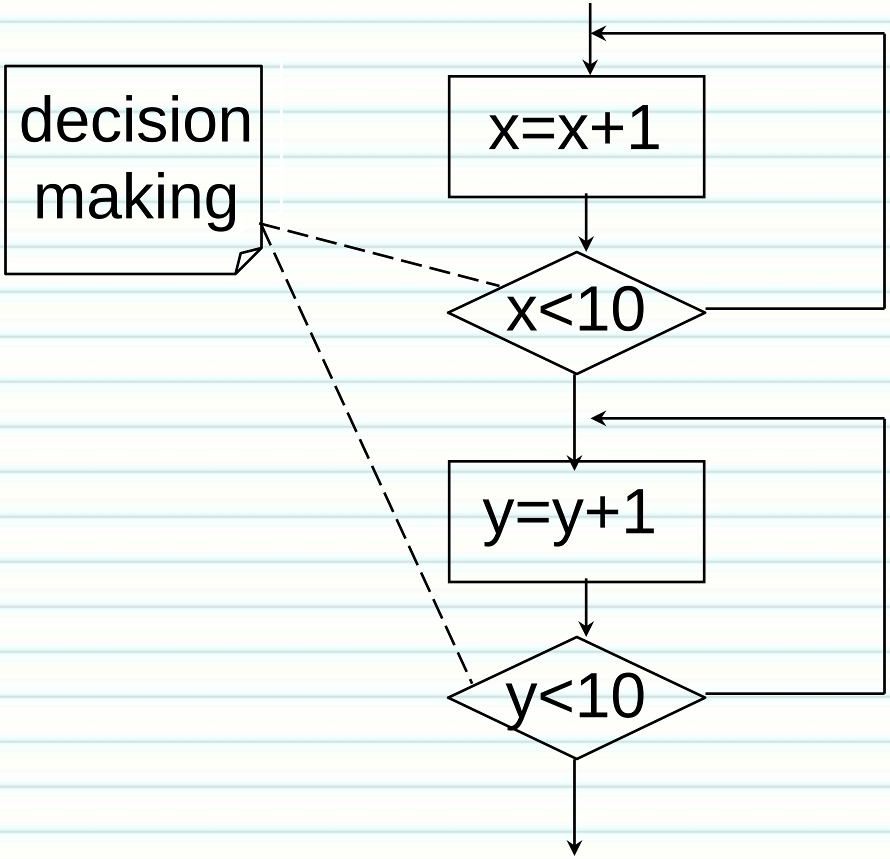
Systems analysis and design need to

- describe current information processing activities in the organization or existing system (modeling of the existing system) to help the development team understand the existing business
- describe information processing with the proposed solution (system design)

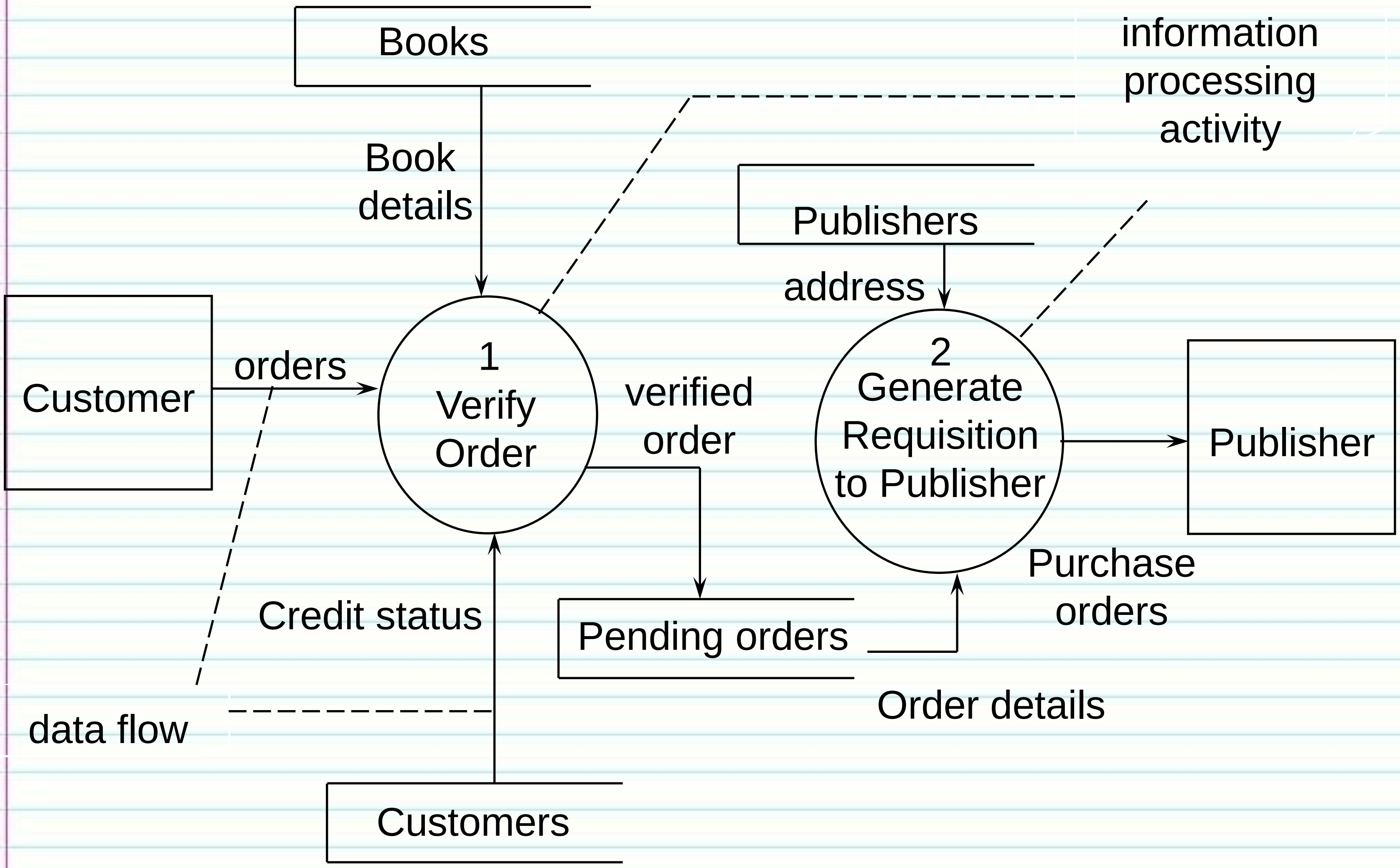
Activity Diagram

- An activity diagram models the information processing activity in the real world (analysis model) or the system (design model).
- A UML activity diagram is a combination of
  - flowchart diagram
    - for decision making or branching
  - data flow diagram
    - for information processing and data flows
  - Petri net diagram
    - for various control flows
    - for synchronization, concurrency, forking, and joining

A Flowchart



A Data Flow Diagram



Petri Nets

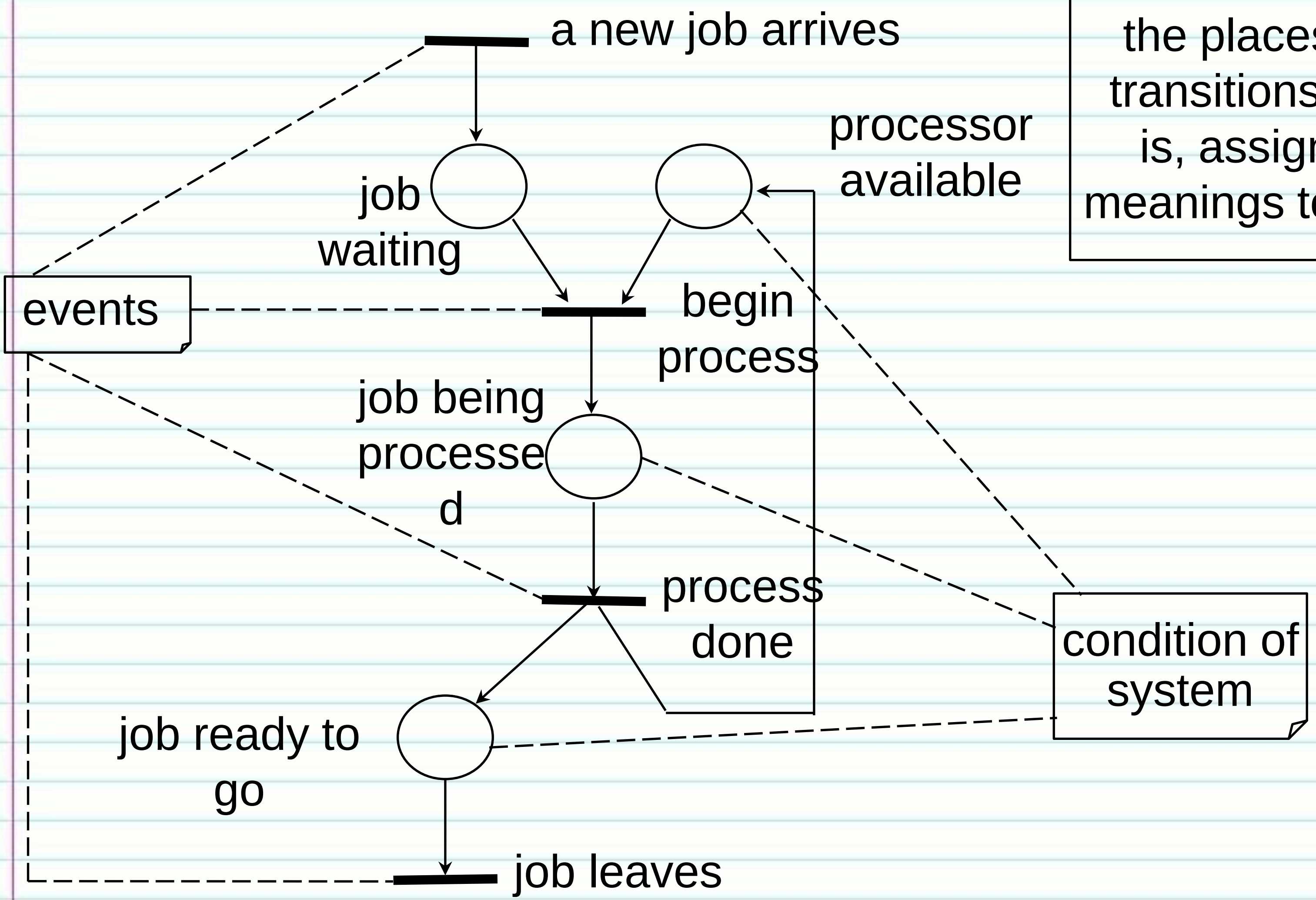
○ places, representing an abstract condition

▬ transitions, representing an event or happing

→ relationship between a place and a transition, it can only come from a place to a transition or from a transition to a place

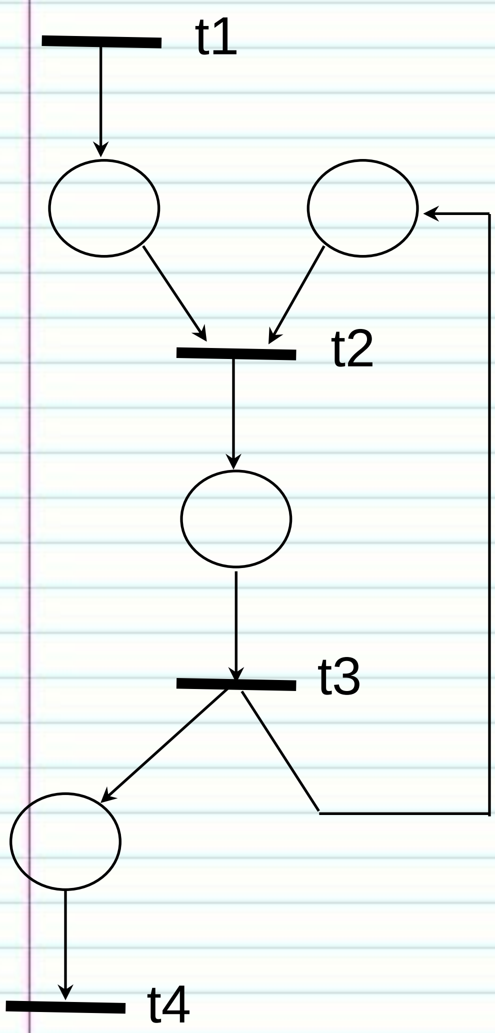
● tokens, which can be placed in places to indicate that the condition is true

A Petri Net Example



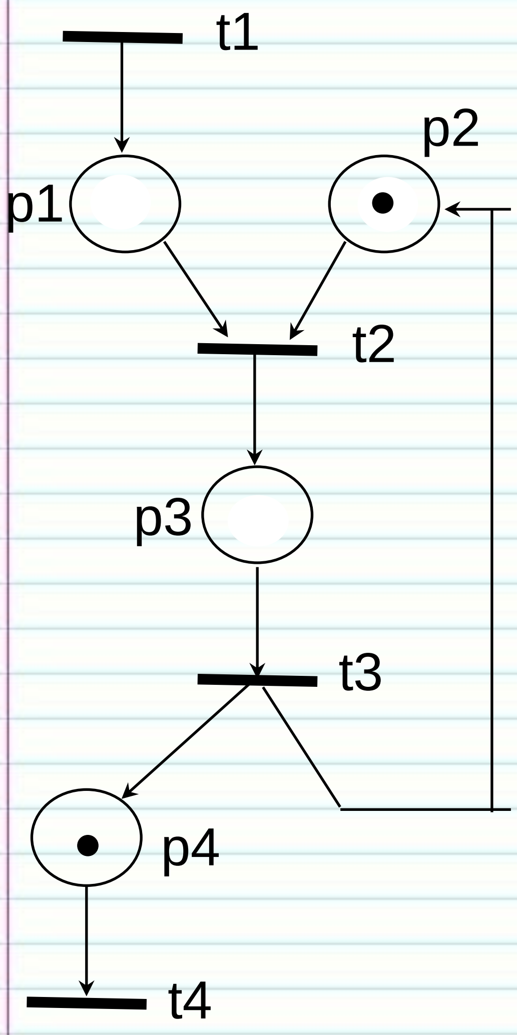
You can interpret the places and transitions. That is, assigning meanings to them.

Petri Net Execution



- A transition is enabled if and only if each of its input places contains a token.
- A transition can be fired sooner or later if it is enabled.
- Firing a transition
  - removes a token from each of its input places AND
  - places a token into each of its output places

Petri Net Marking



An initial marking is an assignment of tokens to places.

t1 is always enabled, because it does not have an input place holder.

firing t1 places a token in p1

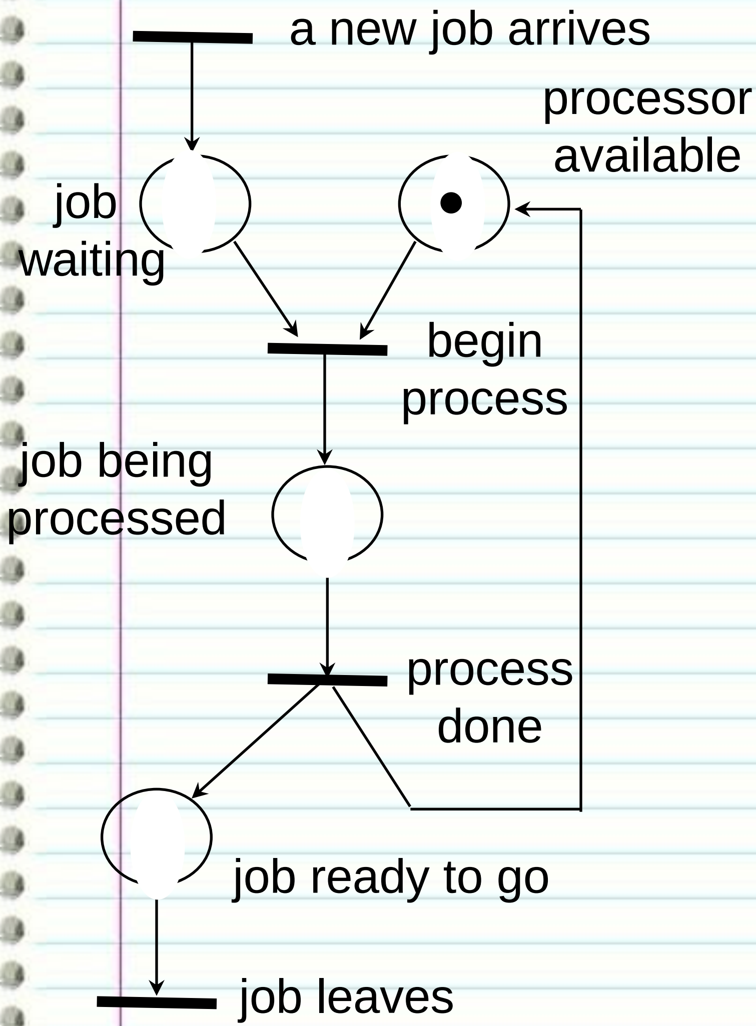
t2 is now enabled

firing t2 removes one token from each of p1 and p2 and places one token in p3

t3 is now enabled

firing t3 removes one token from p3 and places one token in p4 and p2

Petri Net Marking



"a new job arrives" is always enabled, meaning a new job can arrive anytime.

fire "a new job arrives"

"begin process" is now enabled

fire "begin process"

job is being processed

"process done" is now enabled

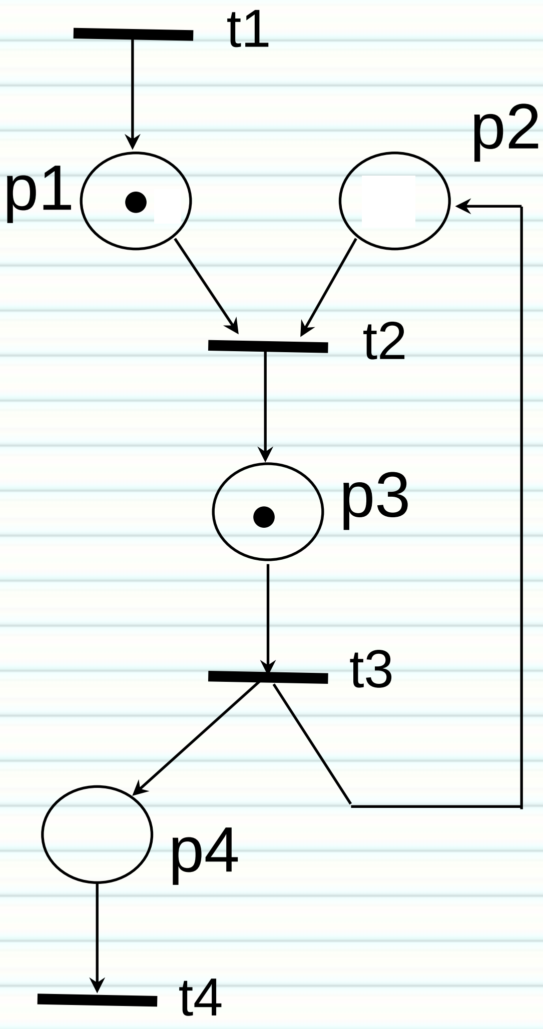
fire "process done"

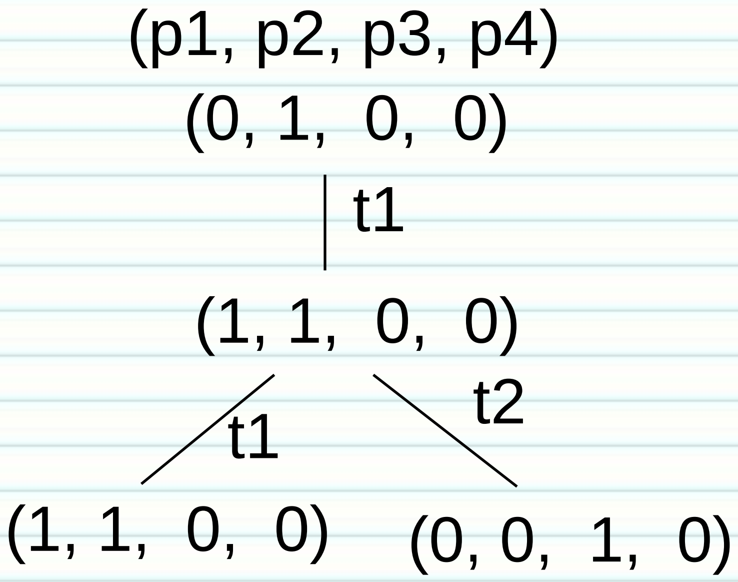
job is ready to leave & processor is available again

"job leaves" is now enabled

fire "job leaves"

Analysis of Petri Net





(p1, p2, p3, p4)

(0, 1, 0, 0)

t1

(1, 1, 0, 0)

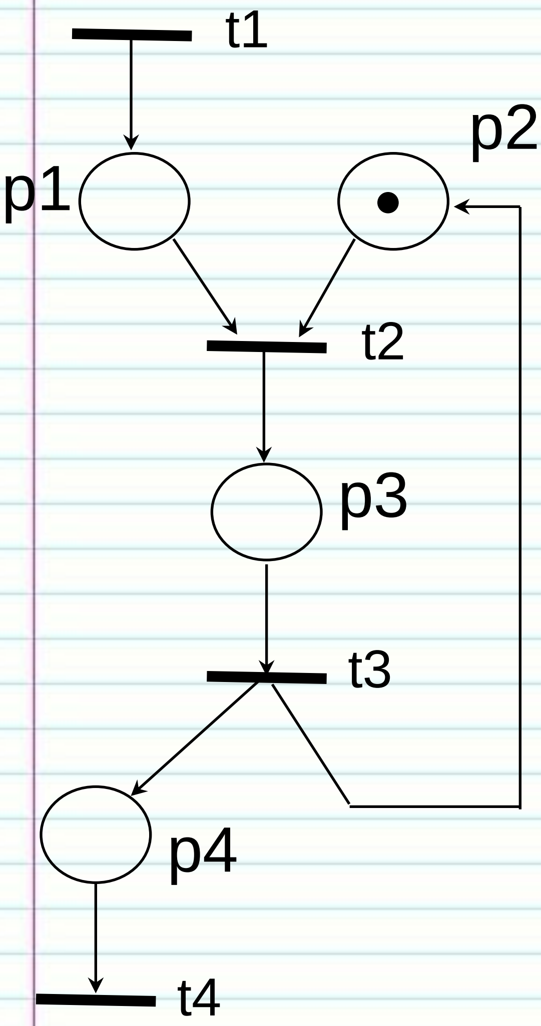
t1

t2

(1, 1, 0, 0)

(0, 0, 1, 0)

Petri Net Analysis Tree



- A marking is presented by a list of “0” and “1”:

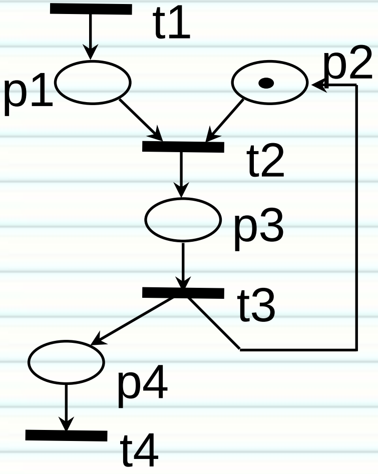
  (n1, n2, ..., nk)

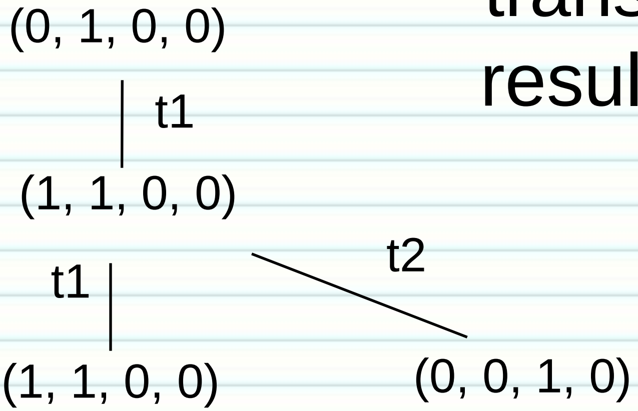
  where ni = 1 if place i contains a token
- The root of the tree denotes the initial marking (initial system state).
- Thus, the initial marking and root of tree for the Petri net on left is:

  (0, 1, 0, 0).

Petri Net Analysis Tree

- Firing a transition grows the tree with a new branch, labeled by the transition fired and the resulting new marking.




(0, 1, 0, 0)

t1

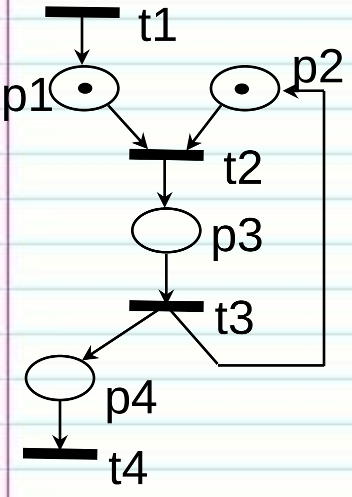
(1, 1, 0, 0)

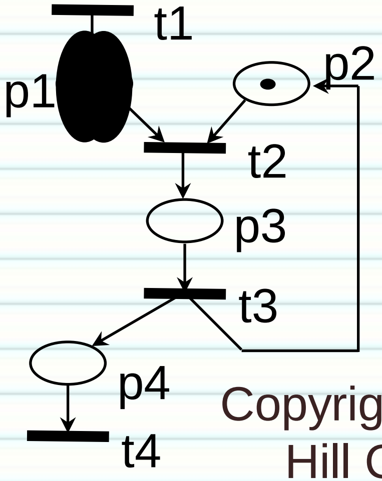
t1

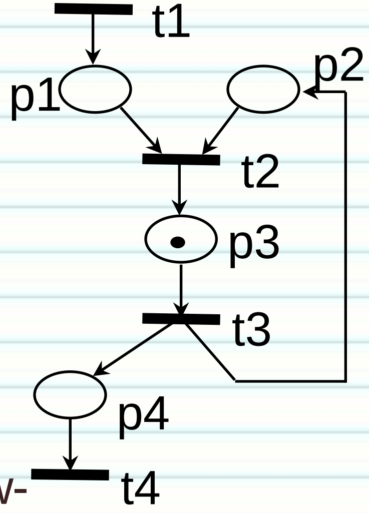
t2

(1, 1, 0, 0)

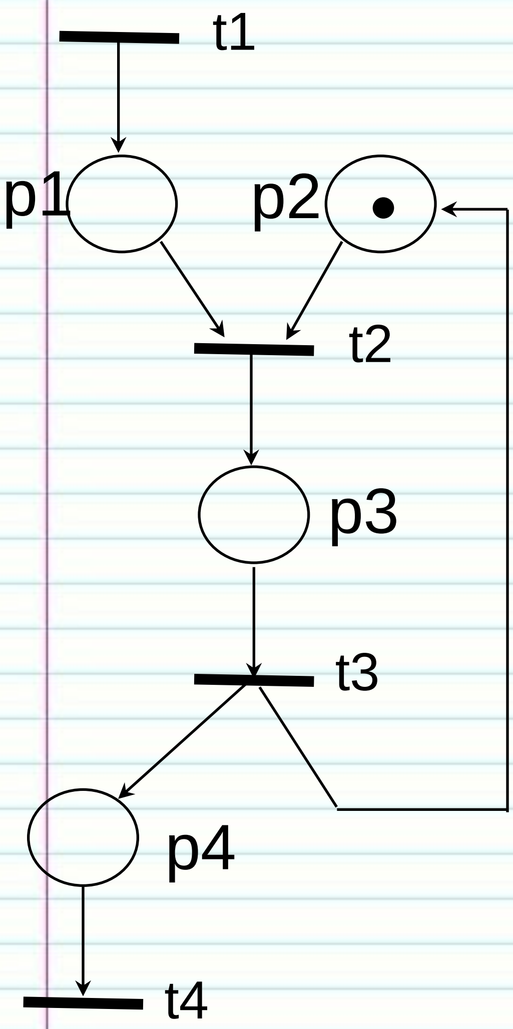
(0, 0, 1, 0)

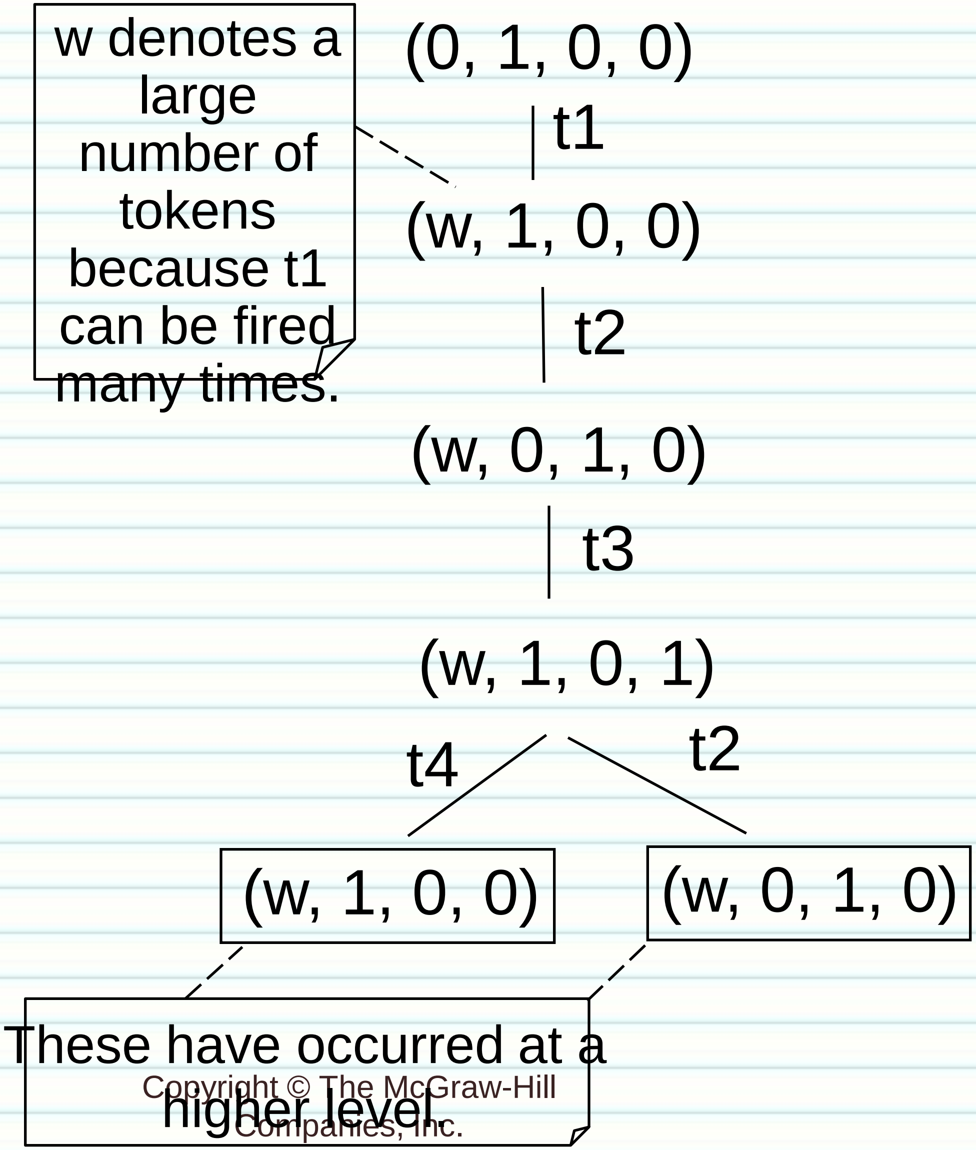






Petri Net Analysis




w denotes a large number of tokens because t1 can be fired many times.

(0, 1, 0, 0)

t1

(w, 1, 0, 0)

t2

(w, 0, 1, 0)

t3

(w, 1, 0, 1)

t4

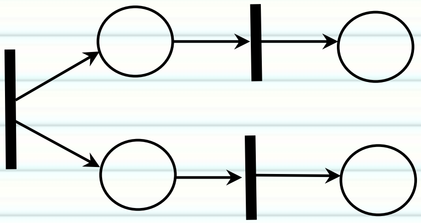
t2

(w, 1, 0, 0)

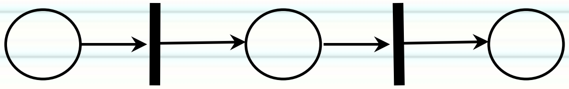
(w, 0, 1, 0)

These have occurred at a higher level.

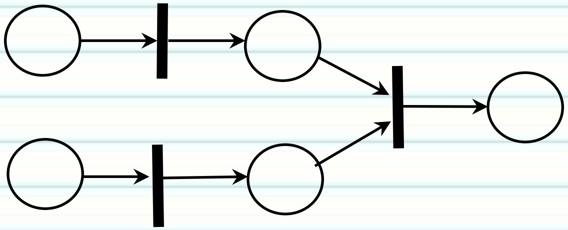
Petri Net Expressiveness



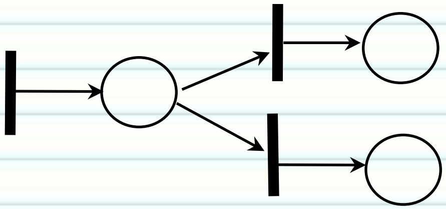
parallelism



sequencing

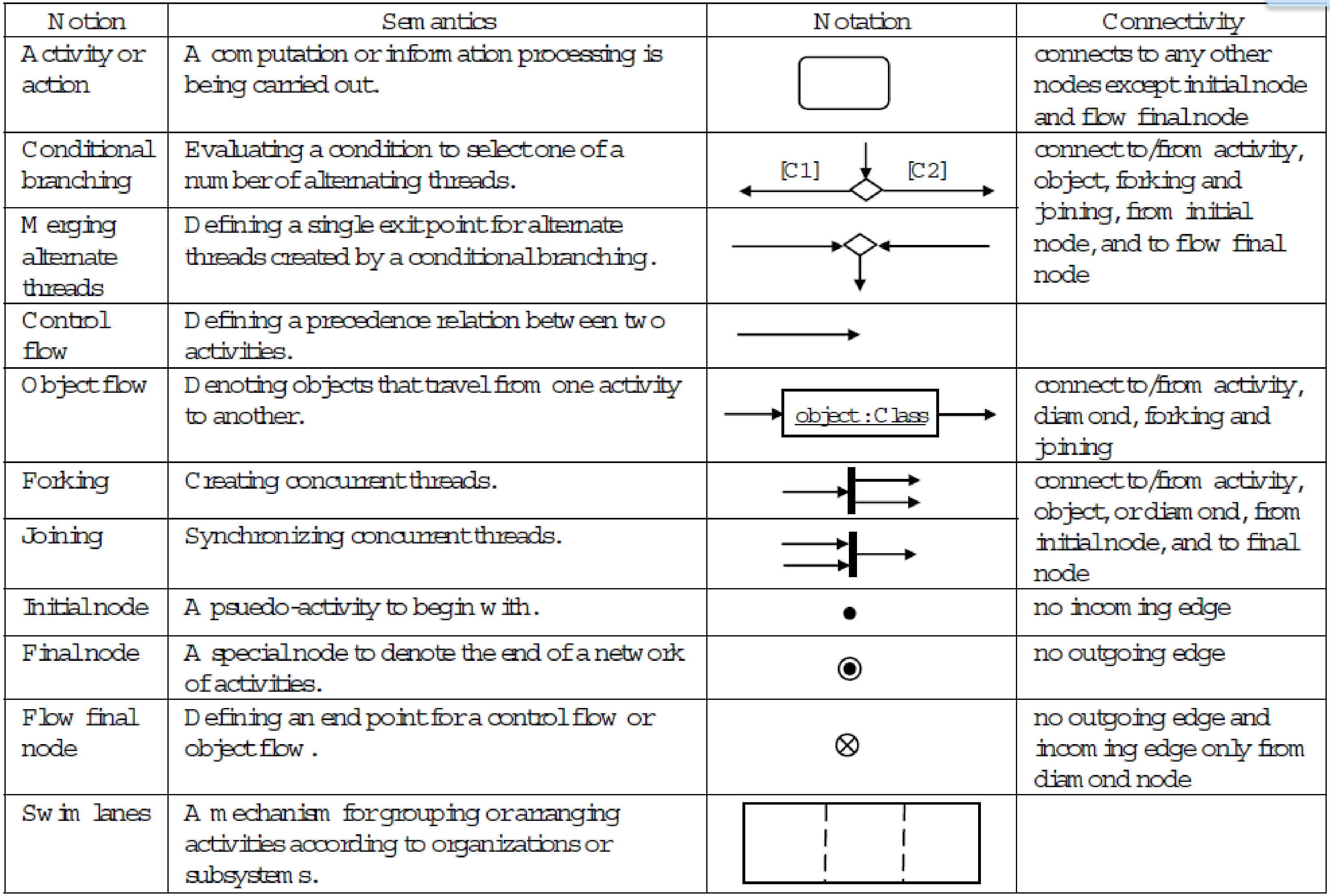


synchronization



exclusion

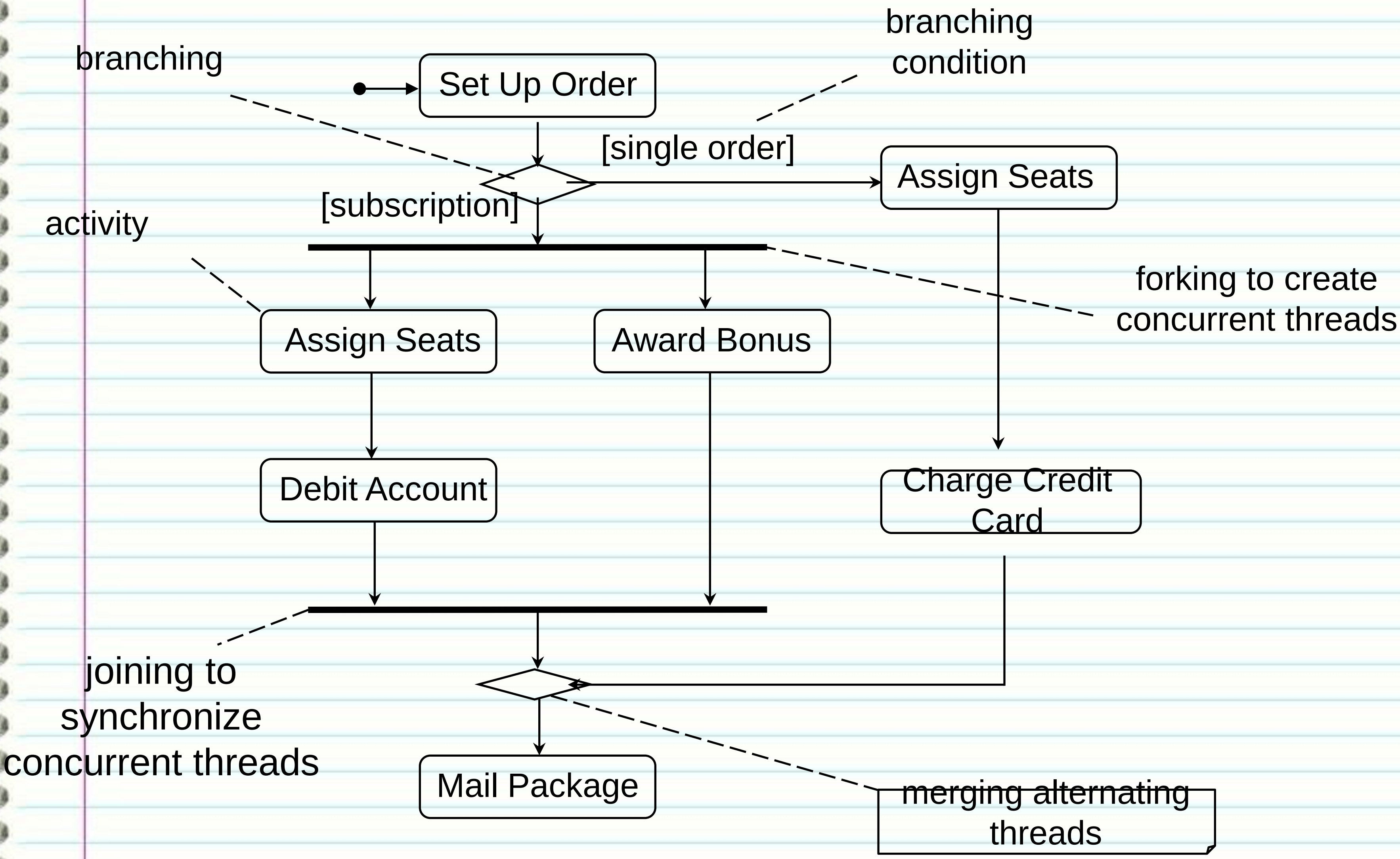
Activity Diagram Notions and Notations



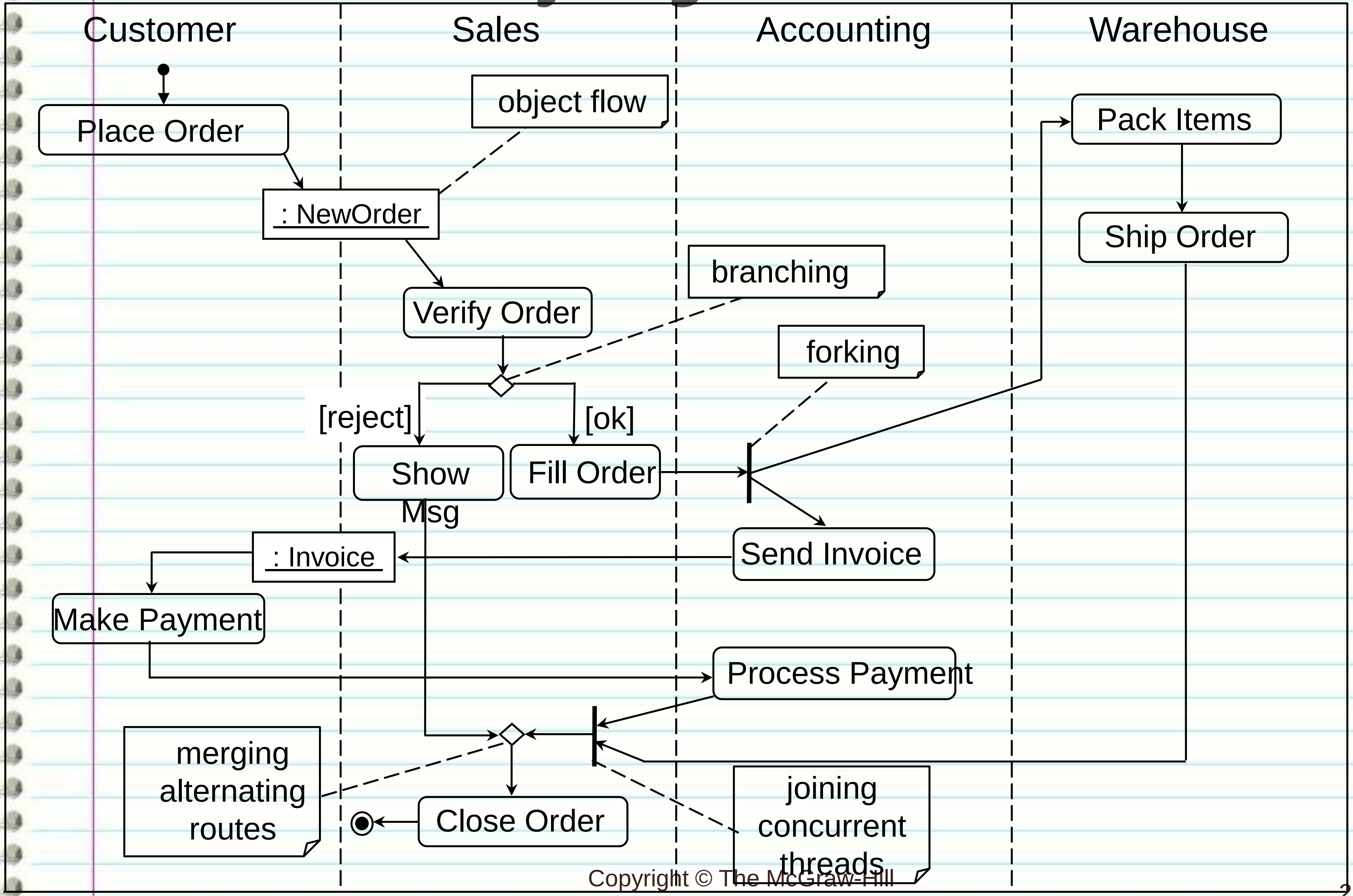
Notion	Semantics	Notation	Connectivity
Activity or action	A computation or information processing is being carried out.		connects to any other nodes except initial node and flow final node
Conditional branching	Evaluating a condition to select one of a number of alternating threads.	[C1] [C2]	connect to/from activity, object, forking and joining, from initial node, and to flow final node
Merging alternate threads	Defining a single exit point for alternate threads created by a conditional branching.		
Control flow	Defining a precedence relation between two activities.		
Object flow	Denoting objects that travel from one activity to another.	object:Class	connect to/from activity, diamond, forking and joining
Forking	Creating concurrent threads.		connect to/from activity, object, or diamond, from initial node, and to final node
Joining	Synchronizing concurrent threads.		
Initial node	A psuedo-activity to begin with.	●	no incoming edge
Final node	A special node to denote the end of a network of activities.	◉	no outgoing edge
Flow final node	Defining an end point for a control flow or object flow.	⊗	no outgoing edge and incoming edge only from diamond node
Swim lanes	A mechanism for grouping or arranging activities according to organizations or subsystems.		

Copyright © The McGraw-Hill Companies, Inc.

Box Office Order Processing

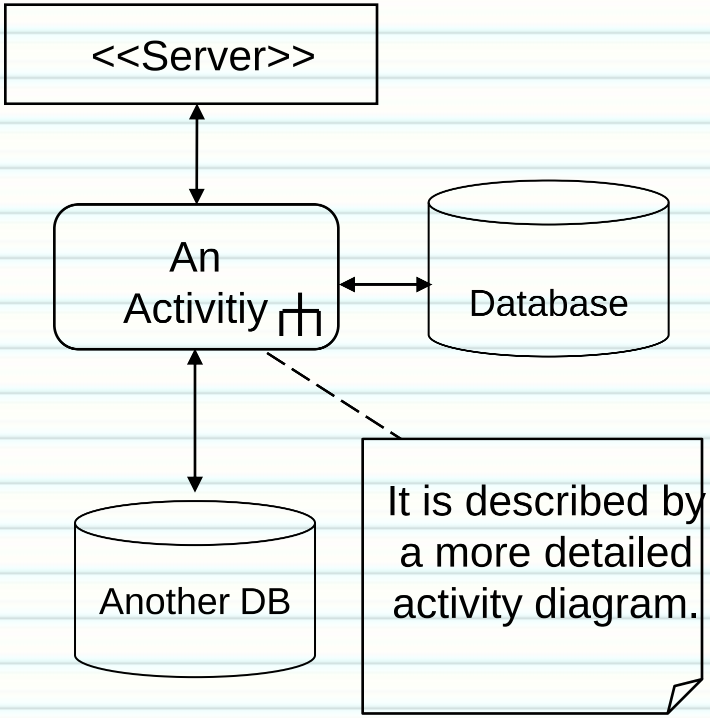


Activity Diagram: Swim Lane



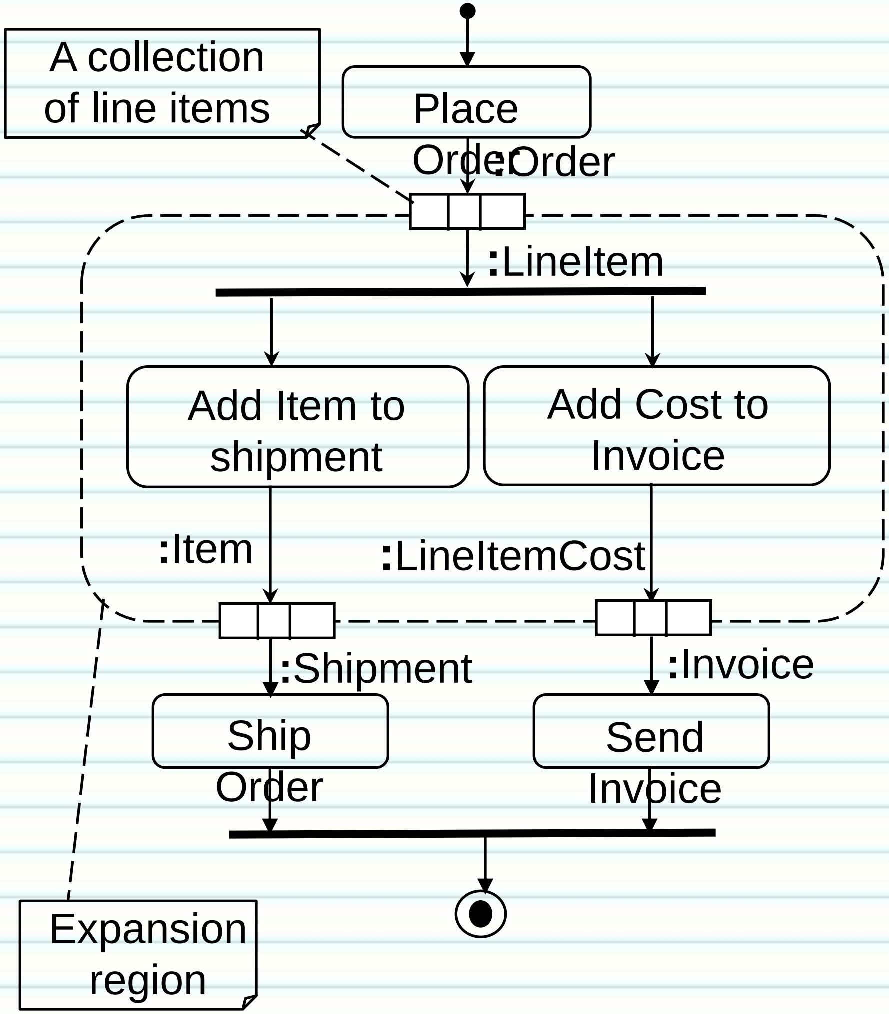
Activity Decomposition and Invocation

- A complex activity can be decomposed and represented by another activity diagram.
- The rake-style symbol is used to signify that the activity has a more detailed activity diagram.



Expansion Region

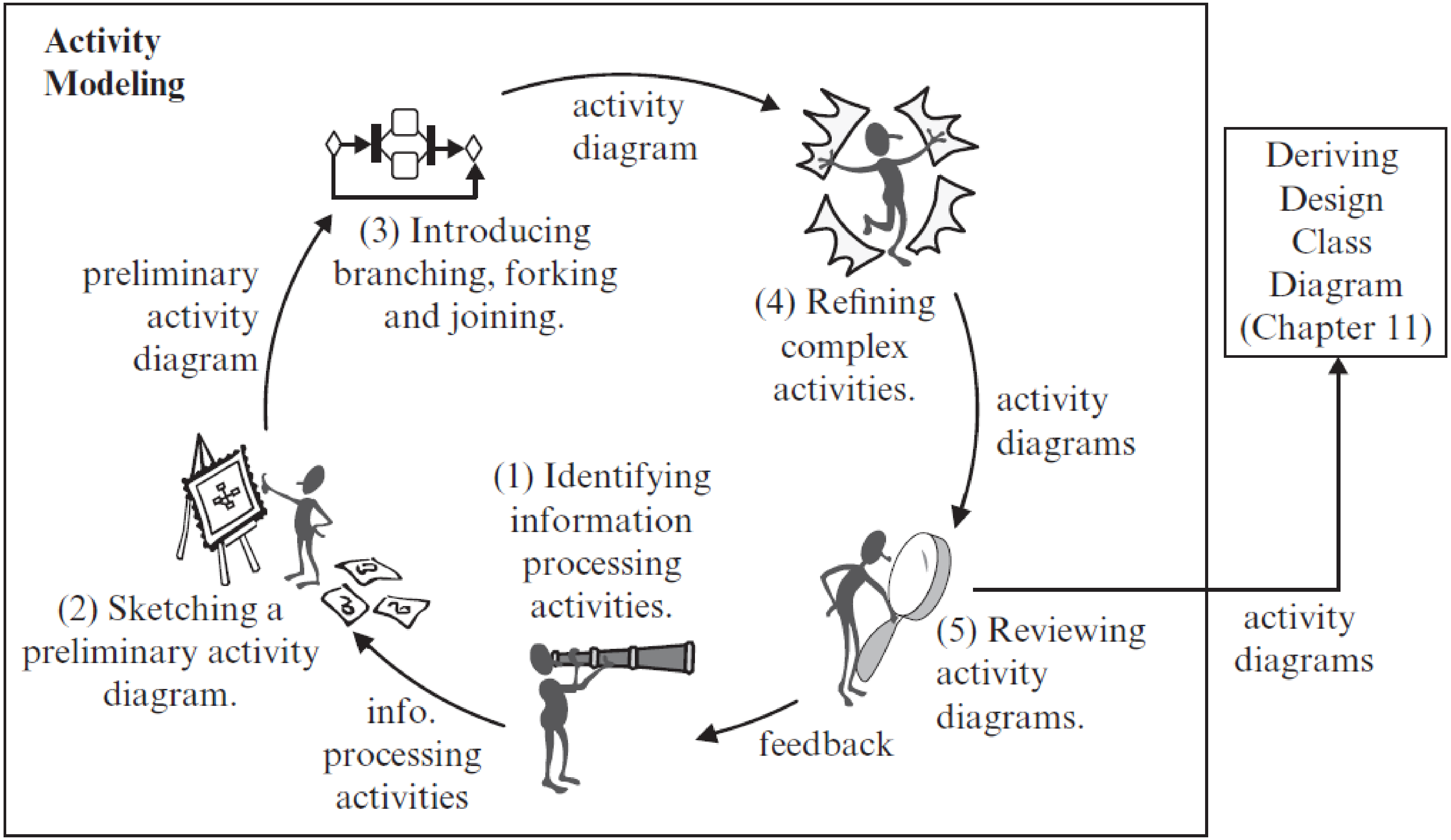
- An expansion region is a subset of activities or actions that should be repeated for each element of a collection.
- The repeated region may produce one or more collections.



Using Activity Diagram

- Modeling, analysis and design of complex information processing activities involving one or all of the following:
  - control flows
  - object flows or data flows
  - access to databases or data depositories
  - conditional branching
  - concurrent threads
  - synchronization
- Work flow among multiple organizational units and/or subsystems.
- Activity diagram can be used alone or used with the other diagrams.
- Activity diagram can be used to model the information processing activity of a system, subsystem or component, or a method of a class.

Steps for Activity Modeling


Activity Modeling
(1) Identifying information processing activities.
info. processing activities
(2) Sketching a preliminary activity diagram.
preliminary activity diagram
(3) Introducing branching, forking and joining.
activity diagram
(4) Refining complex activities.
activity diagrams
(5) Reviewing activity diagrams.
feedback
activity diagrams
Deriving Design Class Diagram (Chapter 11)

Relation to Other Diagrams

- An activity may be a use case, or suggest a use case. Therefore, activity modeling is useful for identifying use cases.
- Activity diagrams are useful for showing workflows and control flows among use cases.
- Activity modeling is useful for processing complex requests in a sequence diagram.
- An activity may exhibit state-dependent behavior, which can be refined by state modeling.
- A state may represent a complex process, which can be modeled by an activity diagram.
- Each object sent from one activity to another should appear in the design class diagram (DCD), or the domain model.
- Swim lanes may suggest object classes in the domain model, or the DCD. The activities of the swim lane identify operations for the class.
- A complex activity may decompose into lower-level activities. Some of these may be operations of the class.

The Agile Project

- The members in an Agile project communicate with each other early and frequently.

- The Agile Manifesto contains four statements of values:
  - Individuals and interactions *over* processes and tools
  - Working software *over* comprehensive documentation
  - Customer collaboration *over* contract negotiation
  - Responding to change *over* following a plan

- Individuals and Interactions
  - Agile development is very people-centered.
  - It is through continuous communication and interaction that teams work most effectively.

The Agile Project (cont.)

- Working Software
  - From a customer perspective, working software is much more useful and valuable than overly detailed documentation.
  - Working software (reduced functionality) is available much earlier in the development lifecycle provides significant time-to-market advantage.

- Customer Collaboration
  - Customers often find great difficulty in specifying the system that they require.
  - Collaborating directly with the customer improves the likelihood of understanding exactly what the customer requires.

- Responding to Change
  - Change is inevitable in software projects. These factors must be accommodated by the development process.
  - Flexibility in work practices and planning to embrace change is more important than simply adhering rigidly to a plan.

The Agile Manifesto

- Principles - the core Agile Manifesto values are captured in twelve principles:
  1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
  3. Deliver working software frequently, at intervals of between a few weeks to a few months, with a preference to the shorter timescale.
  4. Business people and developers must work together daily throughout the project.
  5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
  6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
  7. Working software is the primary measure of progress.

The Agile Manifesto (cont.)

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Is A Whole Team Approach

- Whole-Team Approach
  - The whole-team approach means involving everyone with the knowledge and skills necessary to ensure project success.
  - The team includes representatives from the customer and other business stakeholders who determine product features.
  - The team should be relatively small; successful teams have been observed with as few as three people and as many as nine.
  - The whole- team approach is supported through the daily stand-up meetings involving all members of the team, where work progress is communicated and any impediments to progress are highlighted.
  - The whole-team approach promotes more effective and efficient team dynamics.

Extreme Programming

- There are several Agile approaches, each of which implements the values and principles of the Agile Manifesto in different ways. In this presentation, three representatives of Agile approaches are considered: Extreme Programming (XP), Scrum, and Kanban.

- Extreme Programming
  - XP embraces five values to guide development: communication, simplicity, feedback, courage, and respect.
  - XP describes a set of principles as additional guidelines: humanity, economics, mutual benefit, self- similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, and accepted responsibility.
  - XP describes thirteen primary practices: sit together, whole team, informative workspace, energized work, pair programming, stories, weekly cycle, quarterly cycle, slack, ten-minute build, continuous integration, test first programming, and incremental design.
  - Many of the Agile software development approaches in use today are influenced by XP and its values and principles. For example, Agile teams following Scrum often incorporate XP practices.

Scrum

- Scrum is an Agile management framework which contains the following constituent instruments and practices:
  - Sprint: Scrum divides a project into iterations (called sprints) of fixed length (usually two to four weeks).
  - Product Increment: Each sprint results in a potentially releasable/shippable product (called an increment).
  - Product Backlog: The product owner manages a prioritized list of planned product items. The product backlog evolves from sprint to sprint (called backlog refinement).
  - Sprint Backlog: At the start of each sprint, the Scrum team selects a set of highest priority items (called the sprint backlog) from the product backlog.
  - Definition of Done: To make sure that there is a potentially releasable product at each sprint’s end, the Scrum team discusses and defines appropriate criteria for sprint completion.

Scrum (cont.)

- Timeboxing: Only those tasks, requirements, or features that the team expects to finish within the sprint are part of the sprint backlog. If the development team cannot finish a task within a sprint, the associated product features are removed from the sprint and the task is moved back into the product backlog.
- Transparency: The development team reports and updates sprint status on a daily basis at a meeting called the daily scrum.

- Scrum defines three roles:
  - Scrum Master: ensures that Scrum practices and rules are implemented and followed, and resolves any violations, resource issues, or other impediments that could prevent the team from following the practices and rules. This person is not the team lead, but a coach.
  - Product Owner: represents the customer, and generates, maintains, and prioritizes the product backlog. This person is not the team lead.
  - Development Team: develop and test the product. The team is self-organized: There is no team lead, so the team makes the decisions.

User Stories

- Collaborative User Story Creation
  - User stories are written to capture requirements from the perspectives of developers, testers, and business representatives.
  - This shared vision is accomplished through frequent informal reviews while the requirements are being written.
  - The user stories must address both functional and non-functional characteristics. Each story includes acceptance criteria for these characteristics and is complete when the criteria have been satisfied.
- According to the 3C concept, a user story is the conjunction of three elements:
  - Card, Conversation, Confirmation (acceptance criteria)

Retrospectives

- In Agile development, a retrospective is a meeting held at the end of each iteration to discuss what was successful, what could be improved, and how to incorporate the improvements and retain the successes in future iterations.

- Retrospectives can result in improvement decisions focused on effectiveness, productivity, and team satisfaction.

- They may also address the testability of the applications, user stories, features, or system interfaces. Root cause analysis of defects can drive testing and development improvements.

- In general, teams should implement only a few improvements per iteration. This allows for continuous improvement at a sustained pace.

Continuous Integration

- Continuous Integration
  - Delivery of a product increment requires reliable, working, integrated software at the end of every sprint.
  - Continuous integration addresses this challenge by merging all changes made to the software and integrating all changed components regularly, at least once a day.
  - Configuration management, compilation, software build, deployment, and testing are wrapped into a single, automated, repeatable process.
  - Since developers integrate their work constantly, build constantly, and test constantly, defects in code are detected more quickly.

Release and Iteration Planning

- For Agile lifecycles, two kinds of planning occur
  - Release planning
  - Iteration planning
- Release planning
  - Looks ahead to the release of a product, often a few months ahead of the start of a project.
  - Defines and re-defines the product backlog, and may involve refining larger user stories into a collection of smaller stories.
  - Provides the basis for a test approach and test plan spanning all iterations. Release plans are high-level.
  - Business representatives establish and prioritize the user stories for the release.
  - Based on these user stories, project and quality risks are identified and a high-level effort estimation is performed.

Iteration Planning

- Iteration planning
  - Looks ahead to the end of a single iteration and is concerned with the iteration backlog.
  - The team selects user stories from the prioritized release backlog, elaborates the user stories, performs a risk analysis for the user stories, and estimates the work needed for each user story.
  - The number of stories selected is based on established team velocity and the estimated size of the selected user stories.
  - After the contents of the iteration are finalized, the user stories are broken into tasks, which will be carried out by the appropriate team members.

Daily Stand Up Meeting

- The daily stand-up meeting includes all members of the Agile. At this meeting, they communicate their current status. The agenda for each member is [Agile Alliance Guide]:
- What have you completed since the last meeting?
- What do you plan to complete by the next meeting?
- What is getting in your way?
- Any issues that may block test progress are communicated during the daily stand-up meetings, so the whole team is aware of the issues and can resolve them accordingly.