Dynamic Aspects of UML

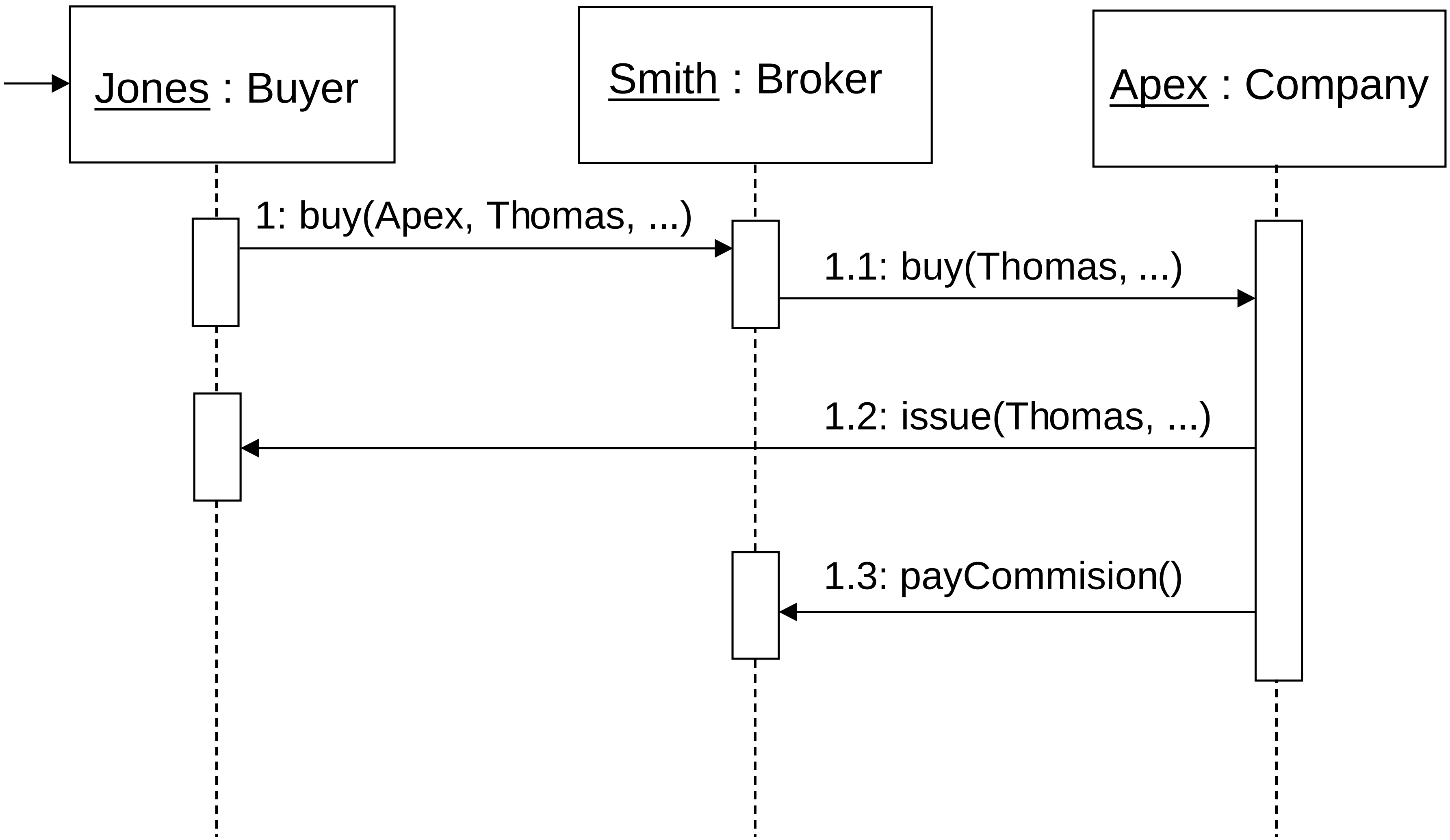
Dynamic Aspects of UML

- Additional UML diagrams
  - Collaboration diagram
  - Sequence diagram
  - Activity diagram
  - State diagram
- One or more of these diagrams would be part of a **design document** which expresses behavior prior to writing code.

Review: Sequence Diagram

- In a sequence diagram the objects' behaviors are shown as **vertical lines** and messages cross between these lines.
- The time sequence is shown by vertical position, flowing down the diagram.
- Time is not generally to scale.
- Bars indicate periods of activity.

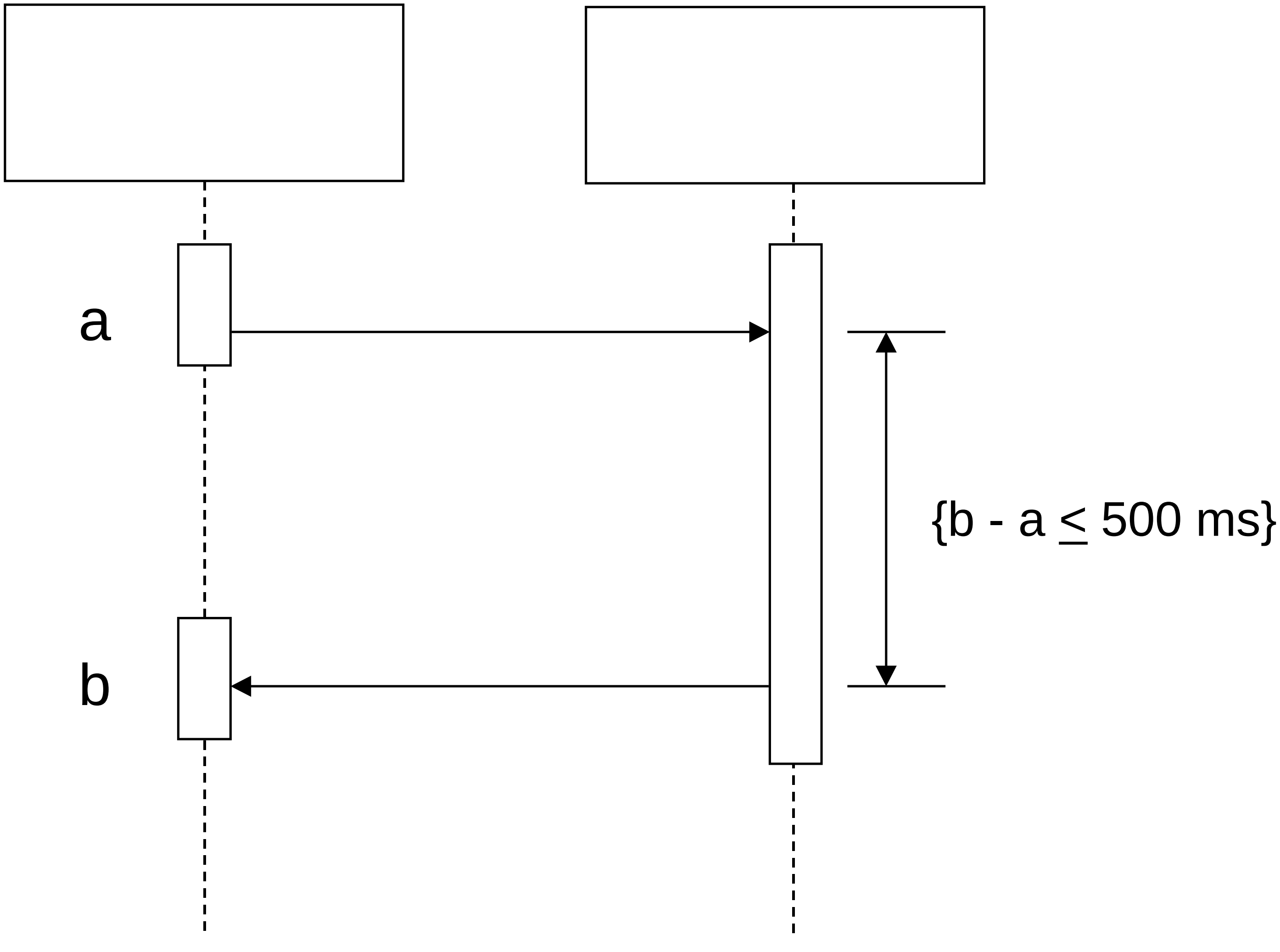
Sequence Diagram for Buying Insurance Policy



Added Facets of Sequence and Collaboration Diagrams

- Objects can send messages to **themselves** (e.g. one method implemented using another, including recursion).
- **Pre-conditions** can exist on message arcs: [... pre-condition ...]
- Timing can be indicated on the diagram
- Object destruction indicated by large X

Timing Annotation

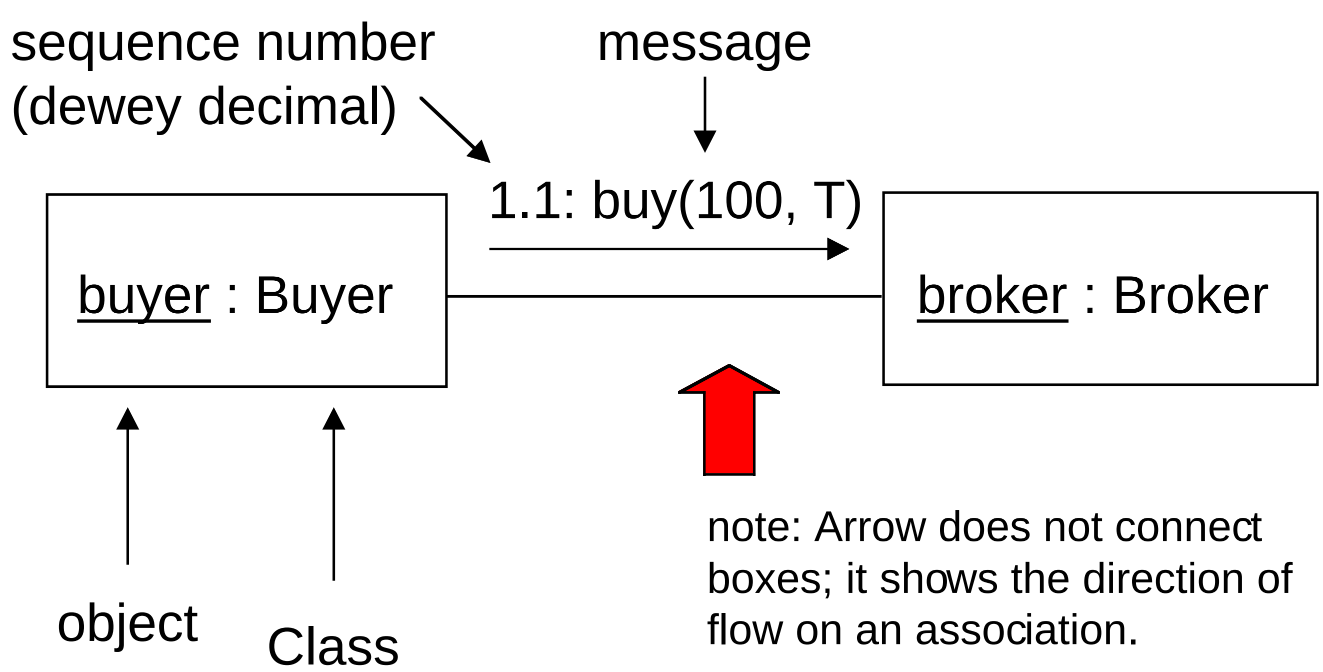


Message Types

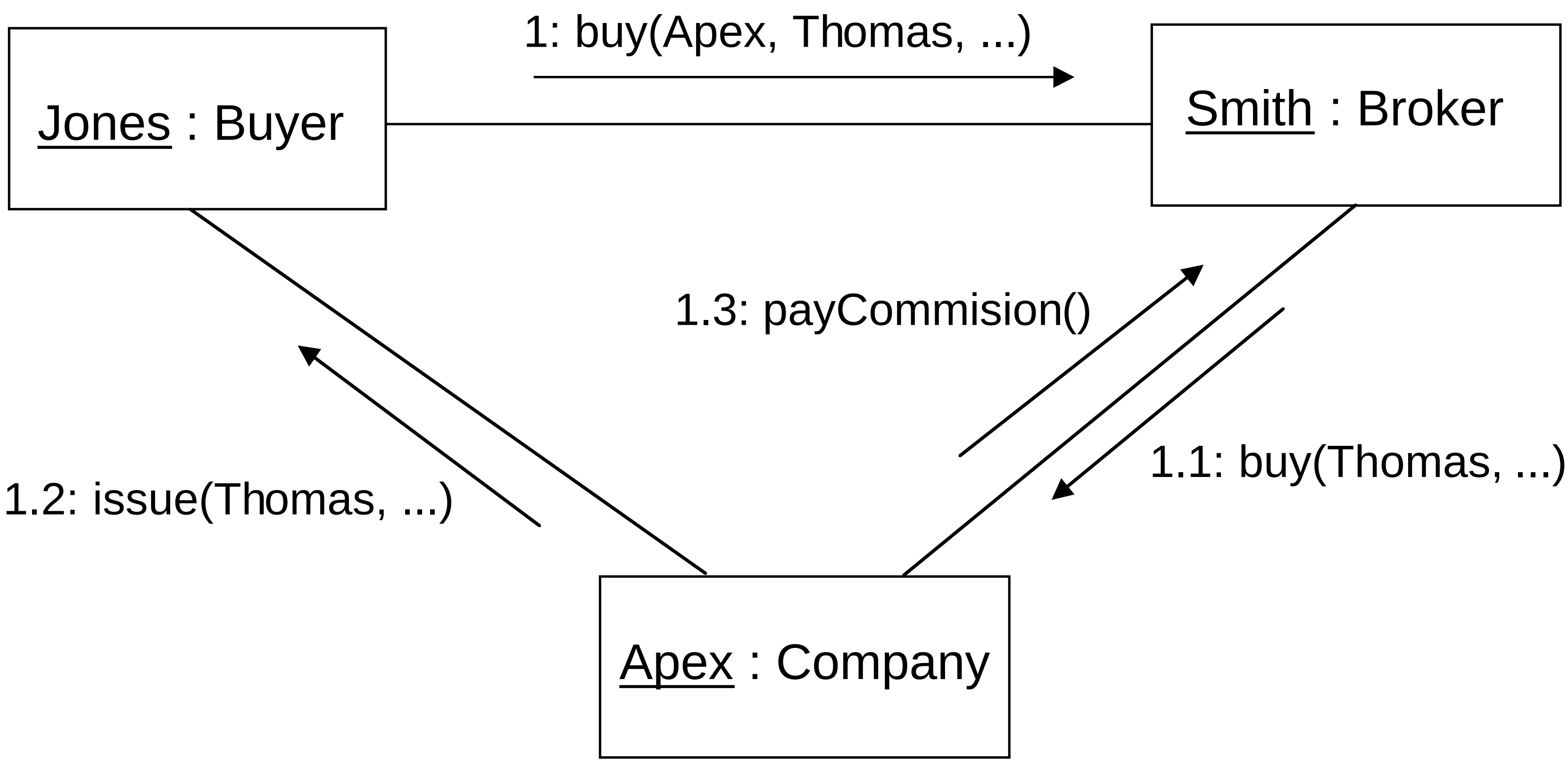
- No specific detail of control passing
- Synchronous: Nested flow, as in typical method calls; **caller waits** for reply before resuming
- Asynchronous: No explicit reply before sender resumes.

Collaboration Diagrams

- Show objects, not classes, collaborating
- "Message" flows between objects, e.g. a method call:



Message Flow for Buying an Insurance Policy



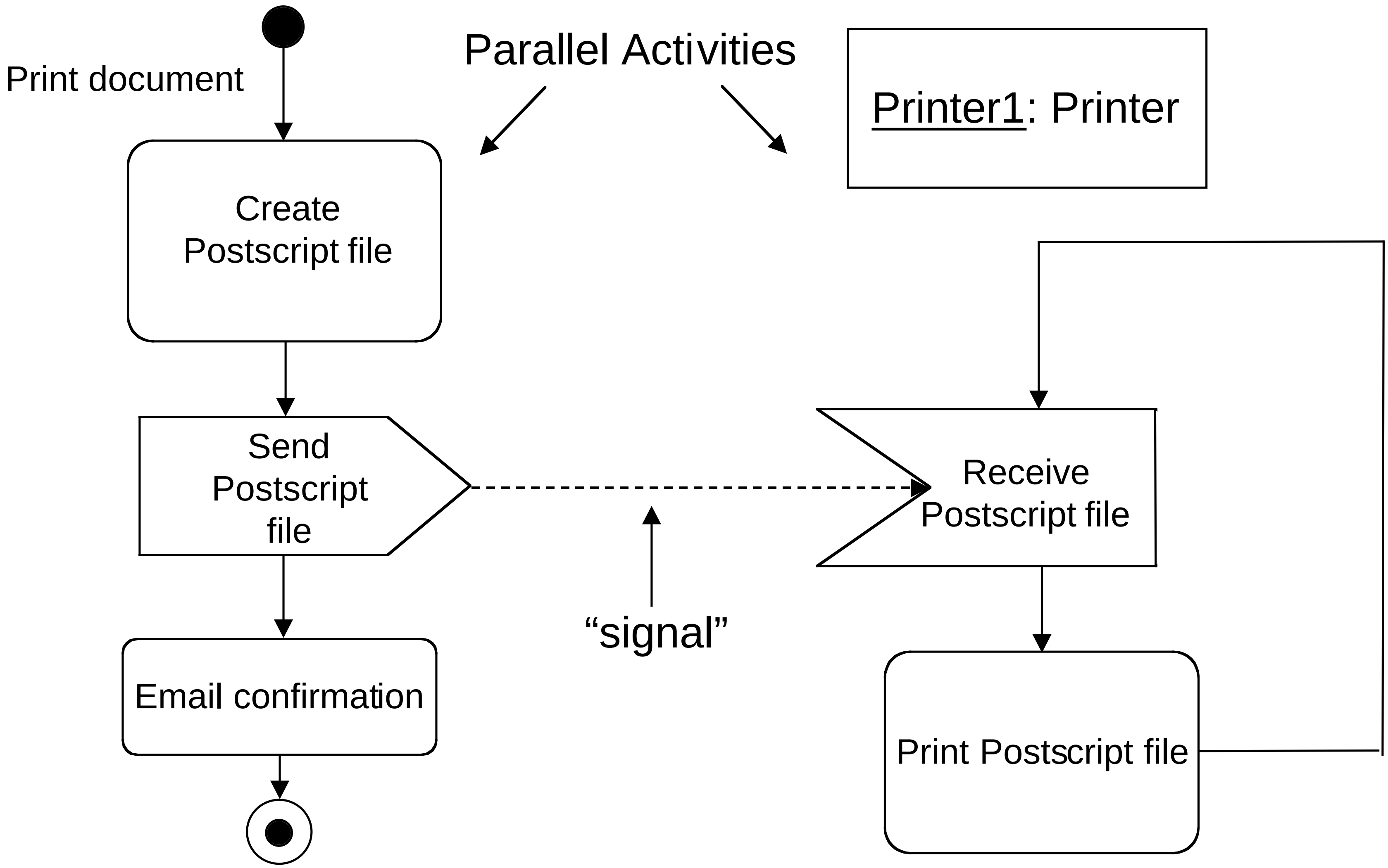
Significance of Numbering

- Numbering shows the sequencing of messages.
- Dewey-Decimal: 1.1, 1.2 are messages introduced in handling of message 1, etc.

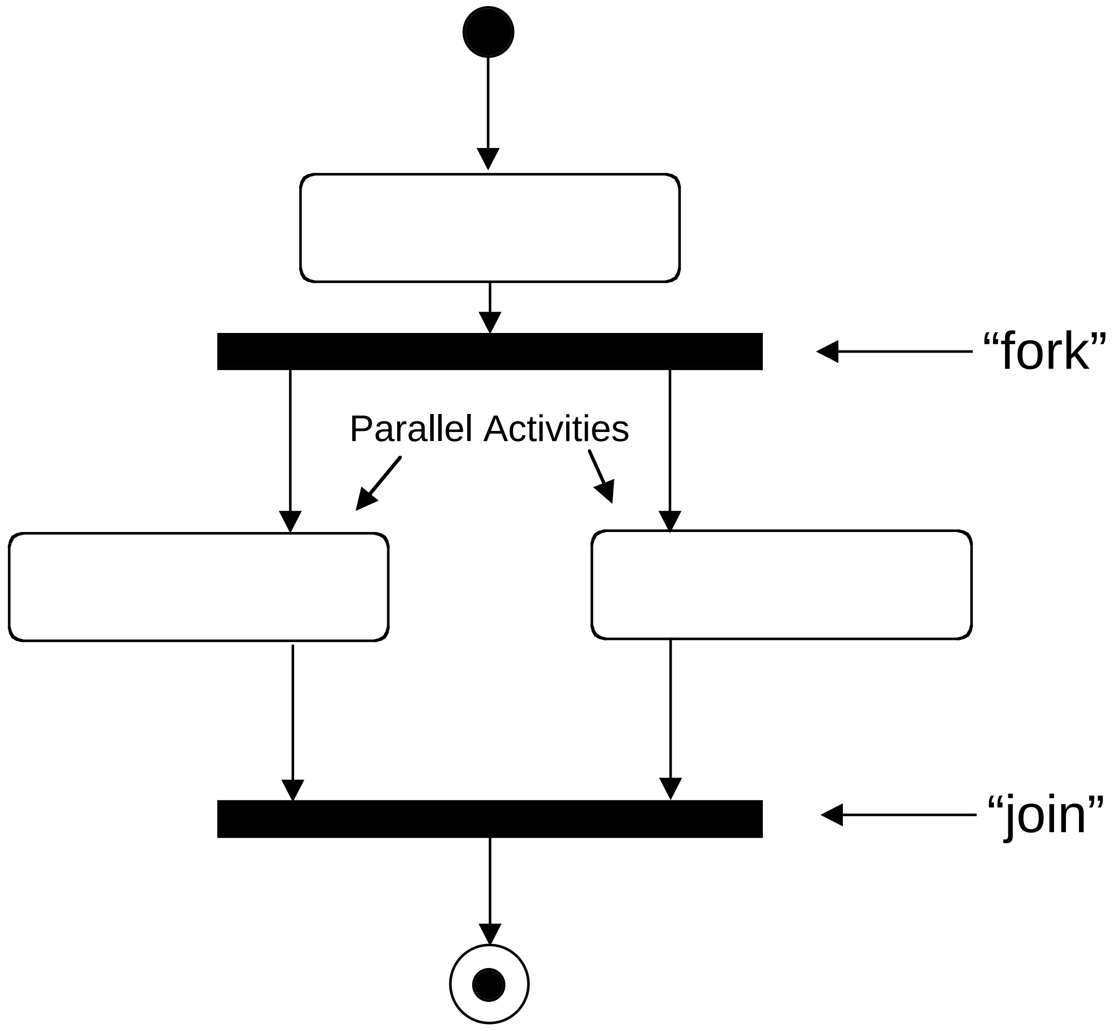
Activity Diagrams

- Activity diagrams are extensions of conventional **flowcharts**.
- **Parallel** execution is possible.
- Synchronization is a subset of Petri net (fork-join).

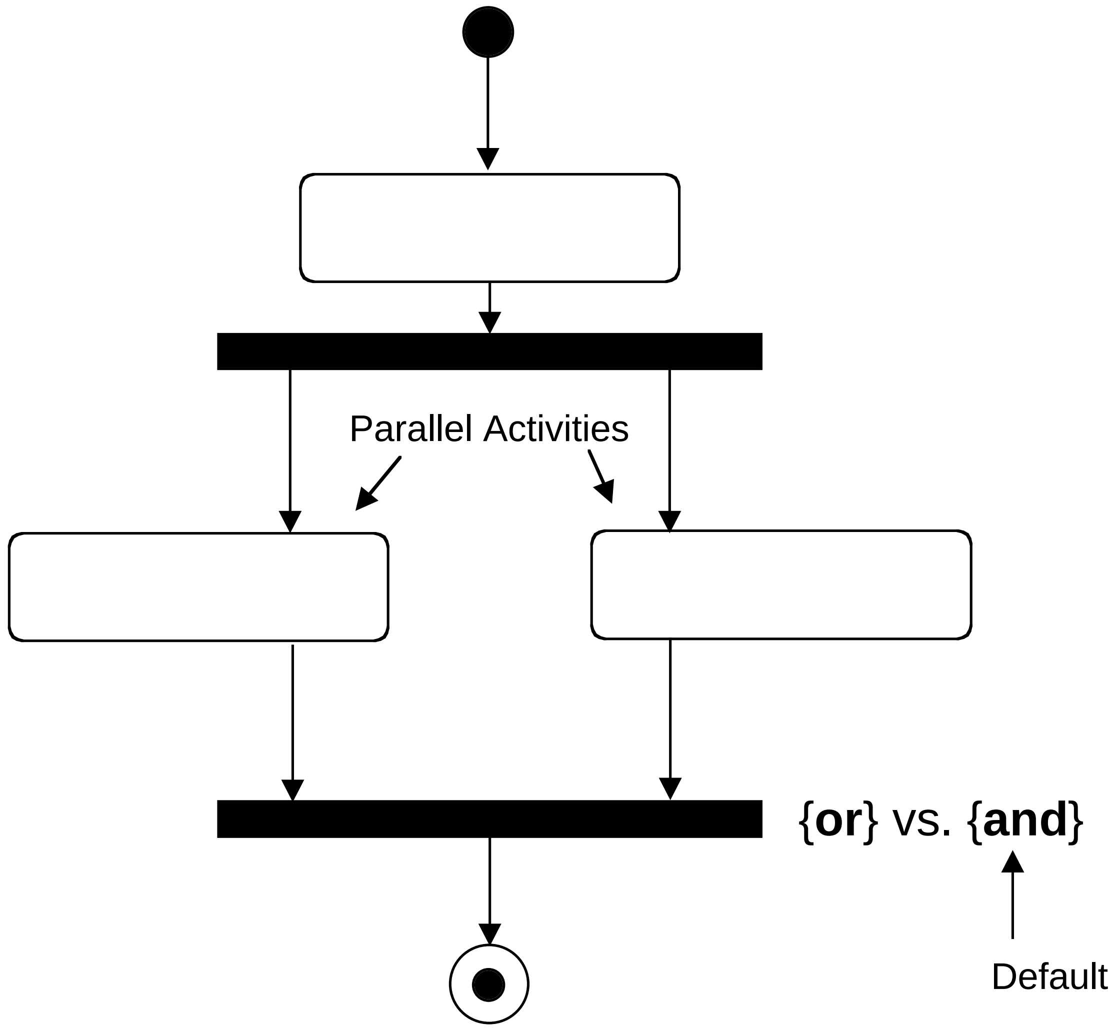
Activity Diagram Example



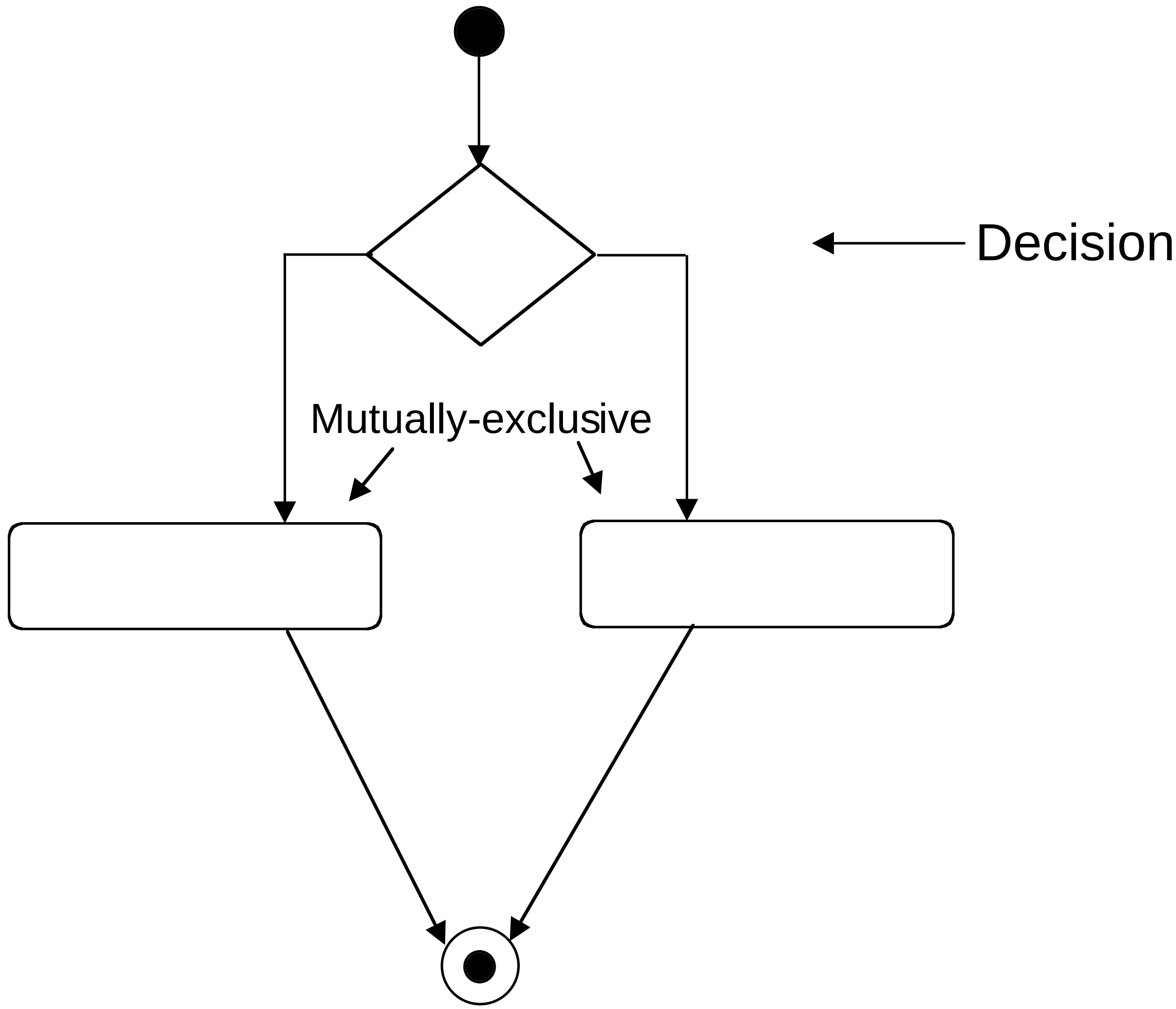
Synchronization of Activities



“And” vs. “Or” Synchronization

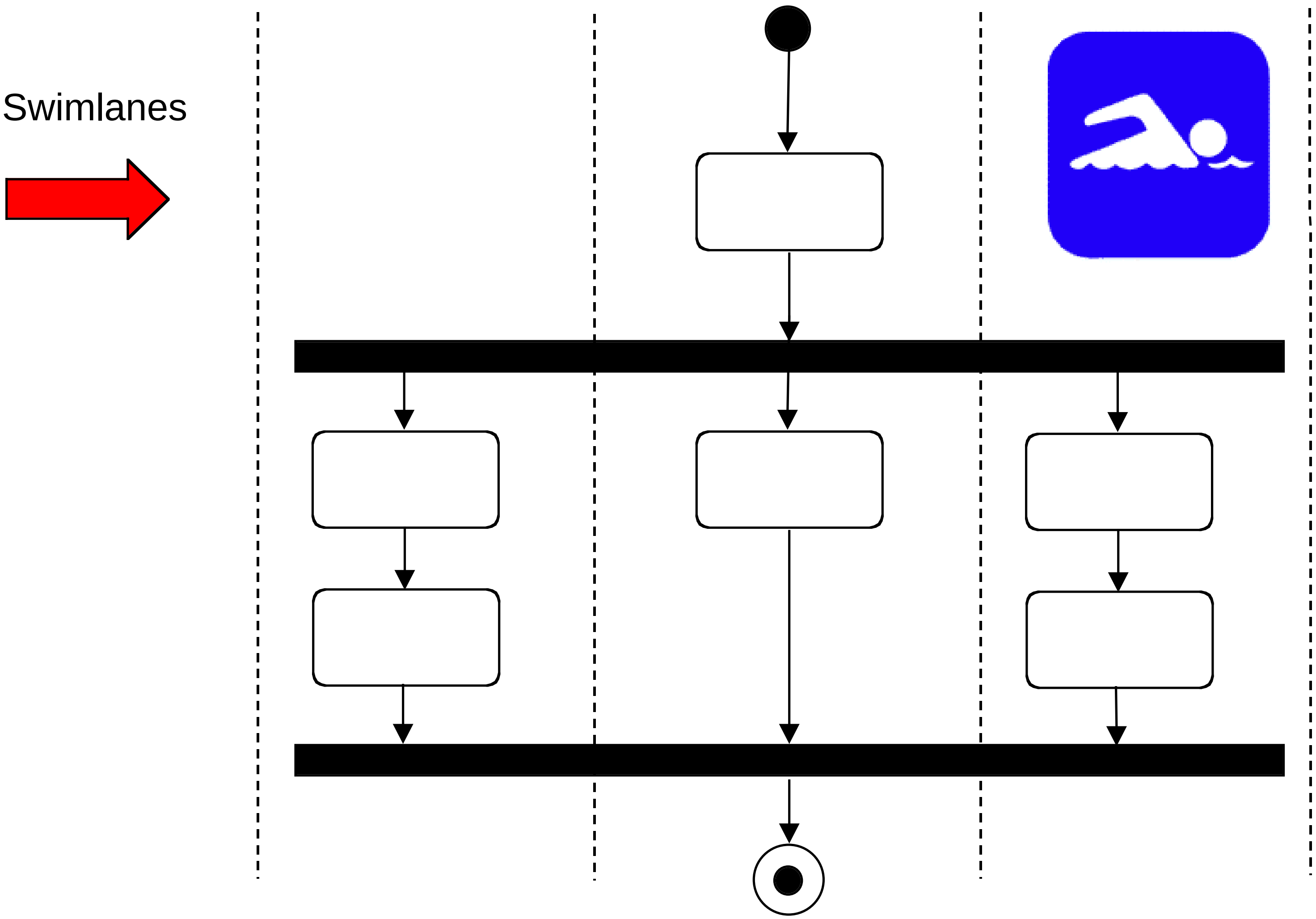


Branching of an Activity (same as in traditional flowchart)



"Swimlanes" = Threads or Processes in Activity Diagram

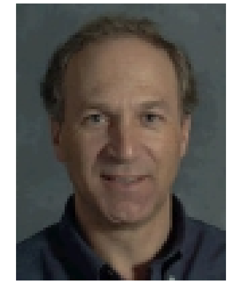
3 Swimlanes



State Diagrams

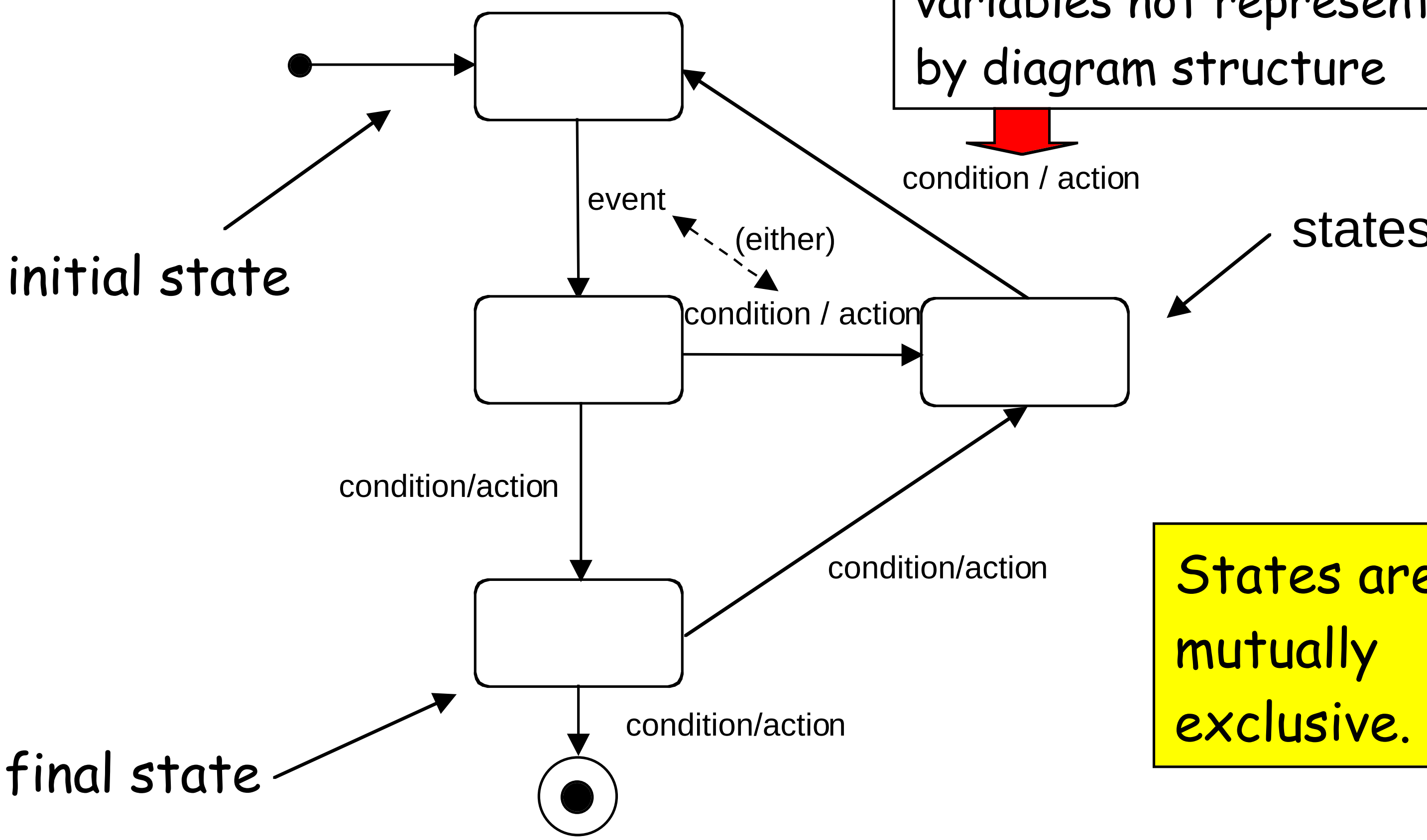
- Essentially **Statecharts**, as invented by David Harel (founder of I-logix).
- Statecharts are a structured form of **finite-state machine**.
- States can be decomposed into parallel or nested sub-systems.
- This is more economical than *enumerating* all states.

StateCharts
(David Harel, Weizmann Inst.)



Every ordinary finite-state machine diagram is a statechart.

may involve program variables not represented by diagram structure

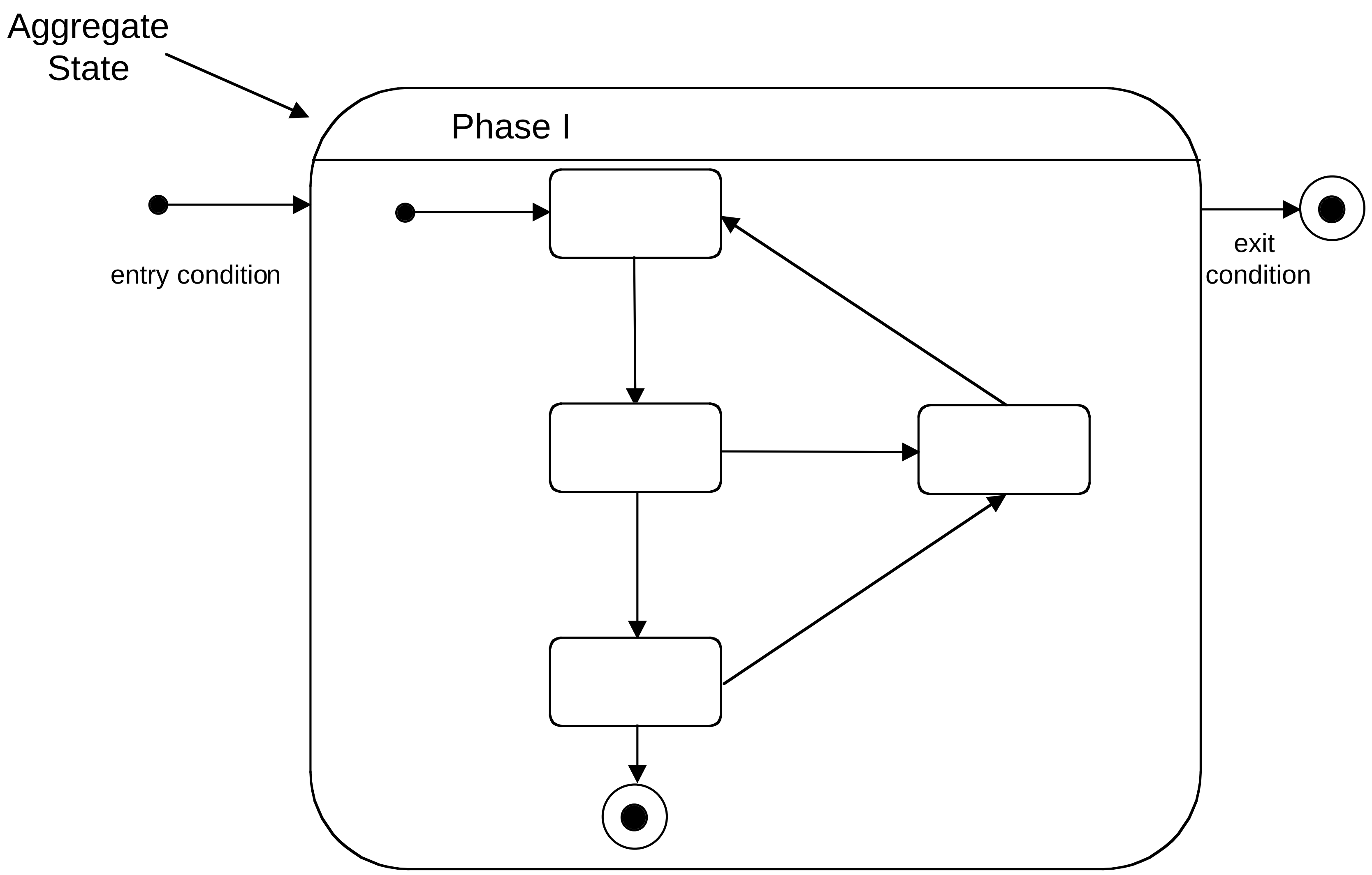


States are mutually exclusive.

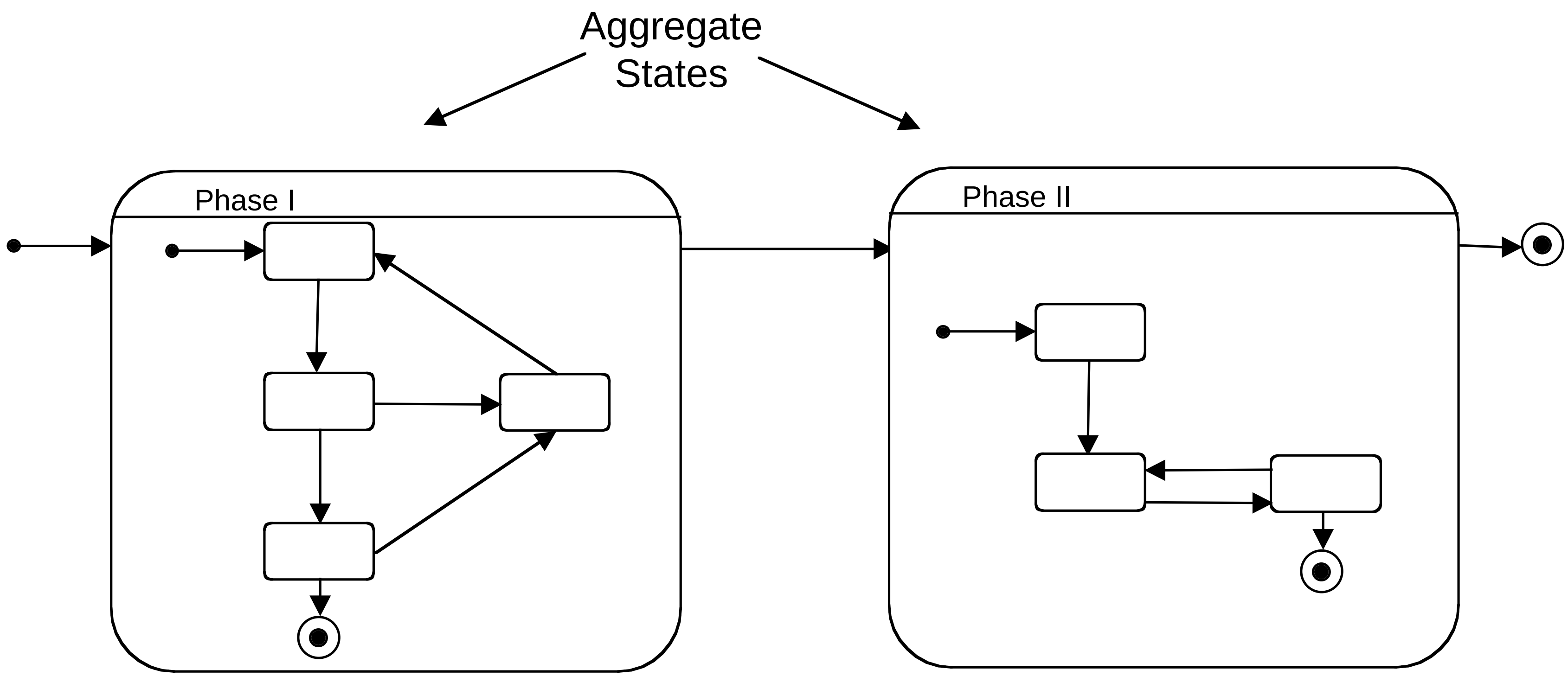
Events vs. Condition/Action pairs

- Events designate some named event occurring, such as an **external** event.
- Condition/Action represents an event **triggered** by a condition on **program variables** and an action that may assign to those variables.
- The **null condition** is the same as "true".
- The **null action** is that all variables are unchanged.

Nuance: Hierarchy in StateCharts



Hierarchy



Word descriptions signaling that an aggregate state might be appropriate:

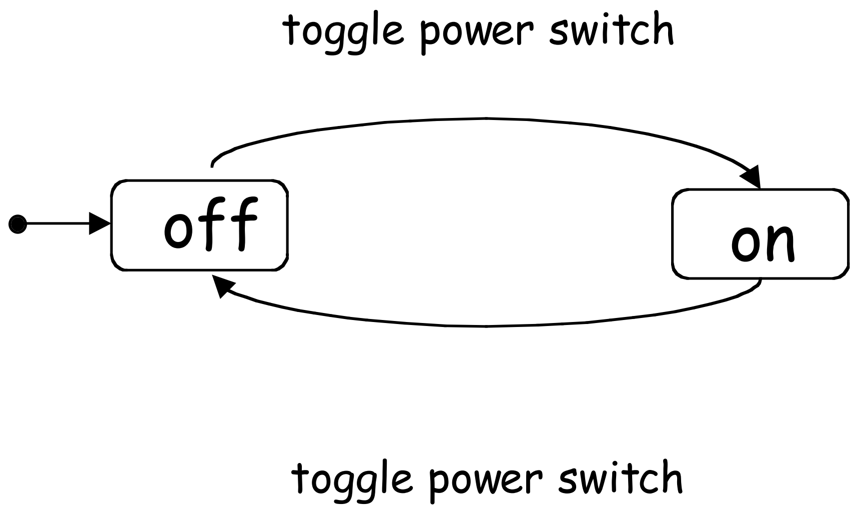
- Phase
- Mode
- Region
- Interval

Statechart Example

- Combination FM/AM Radio and Cassette Tape Player (Auto-reverse)
  - Either the radio or the tape player is playing, but not both.
  - The unit can play only if power is on.
  - The tape plays iff a tape is inserted.
  - The radio can play in AM or FM.

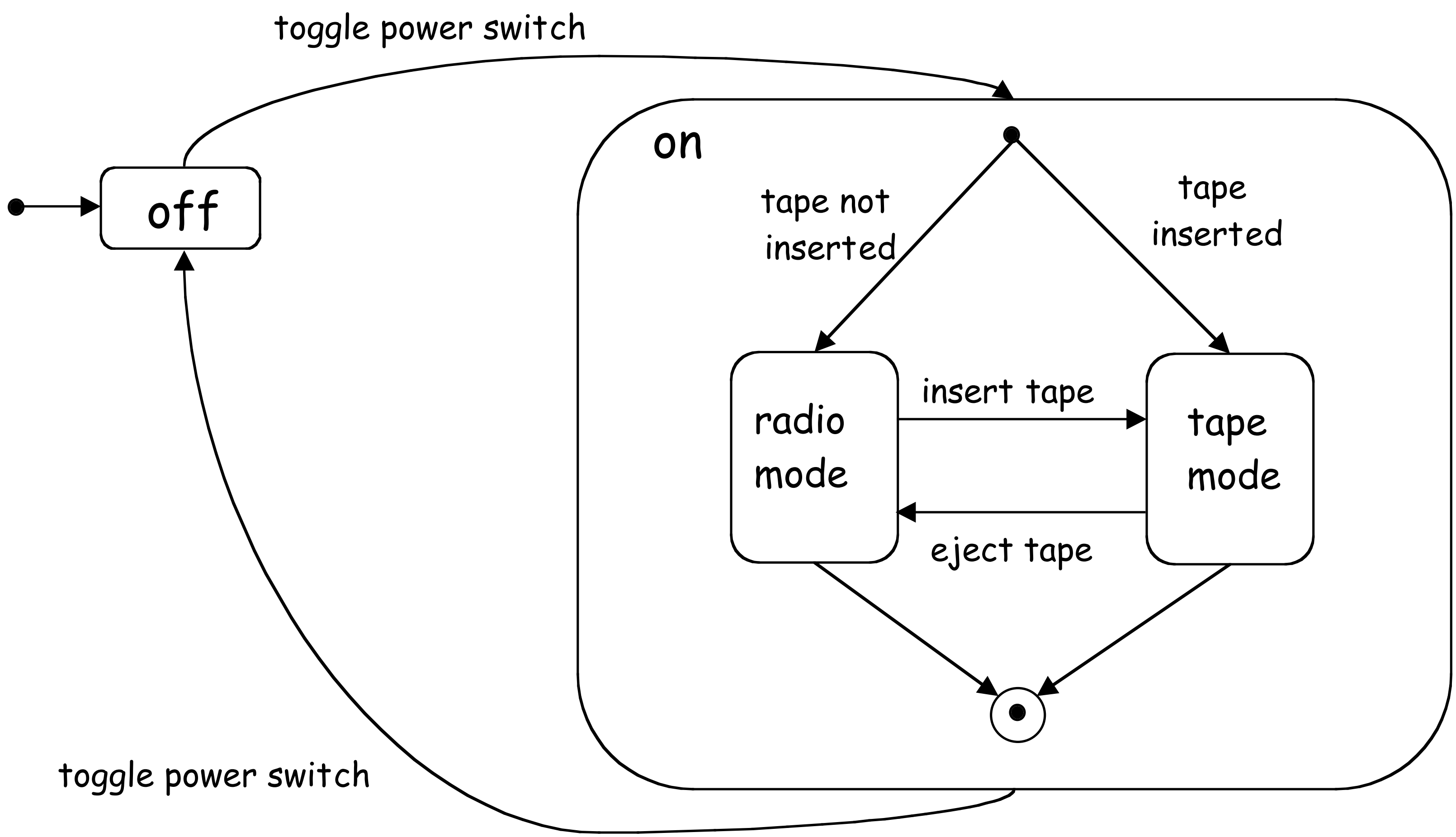
Radio/Tape Player State Diagram

Top-Level View



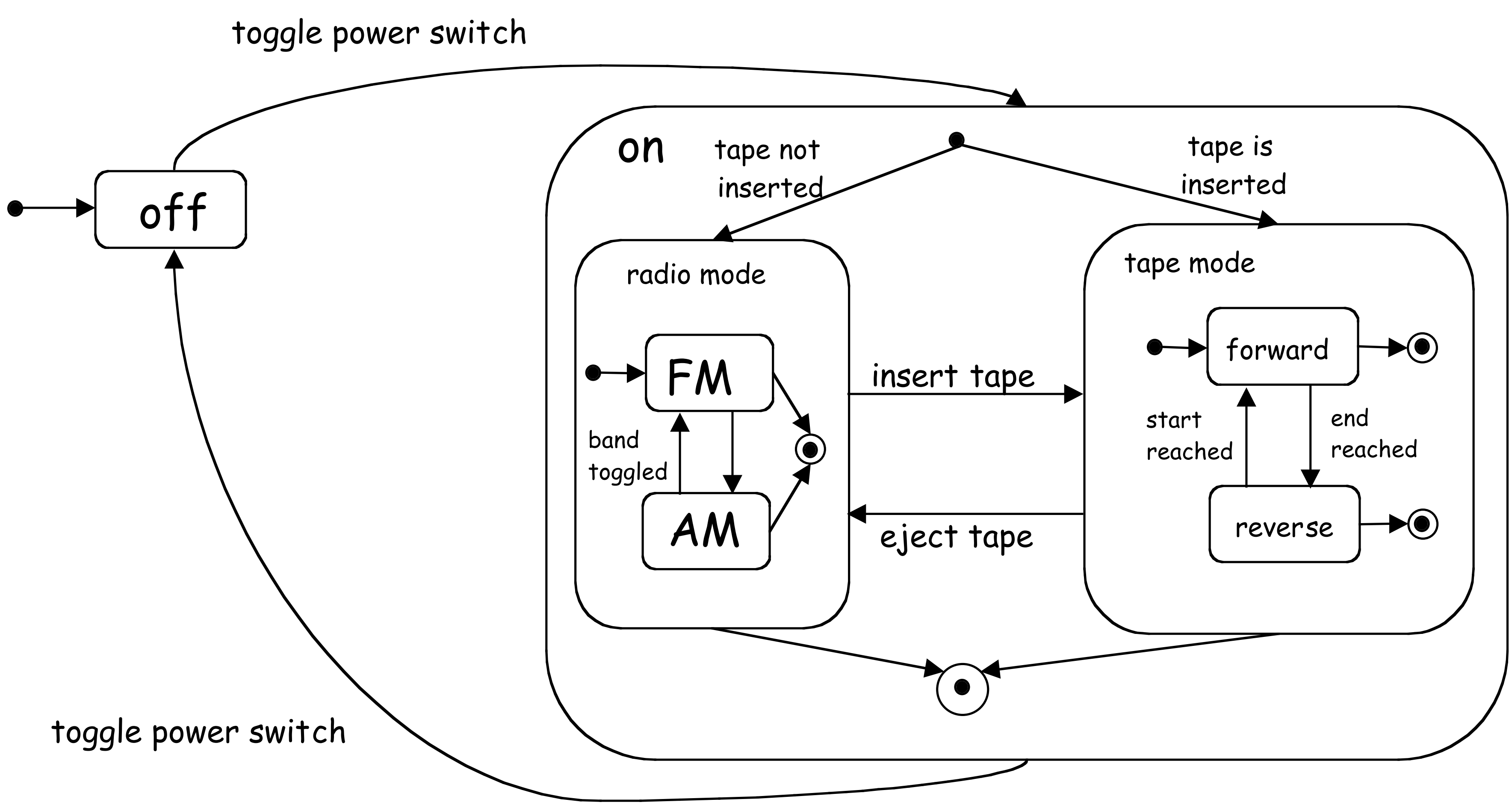
Radio/Tape Player State Diagram

First-Level Expansion



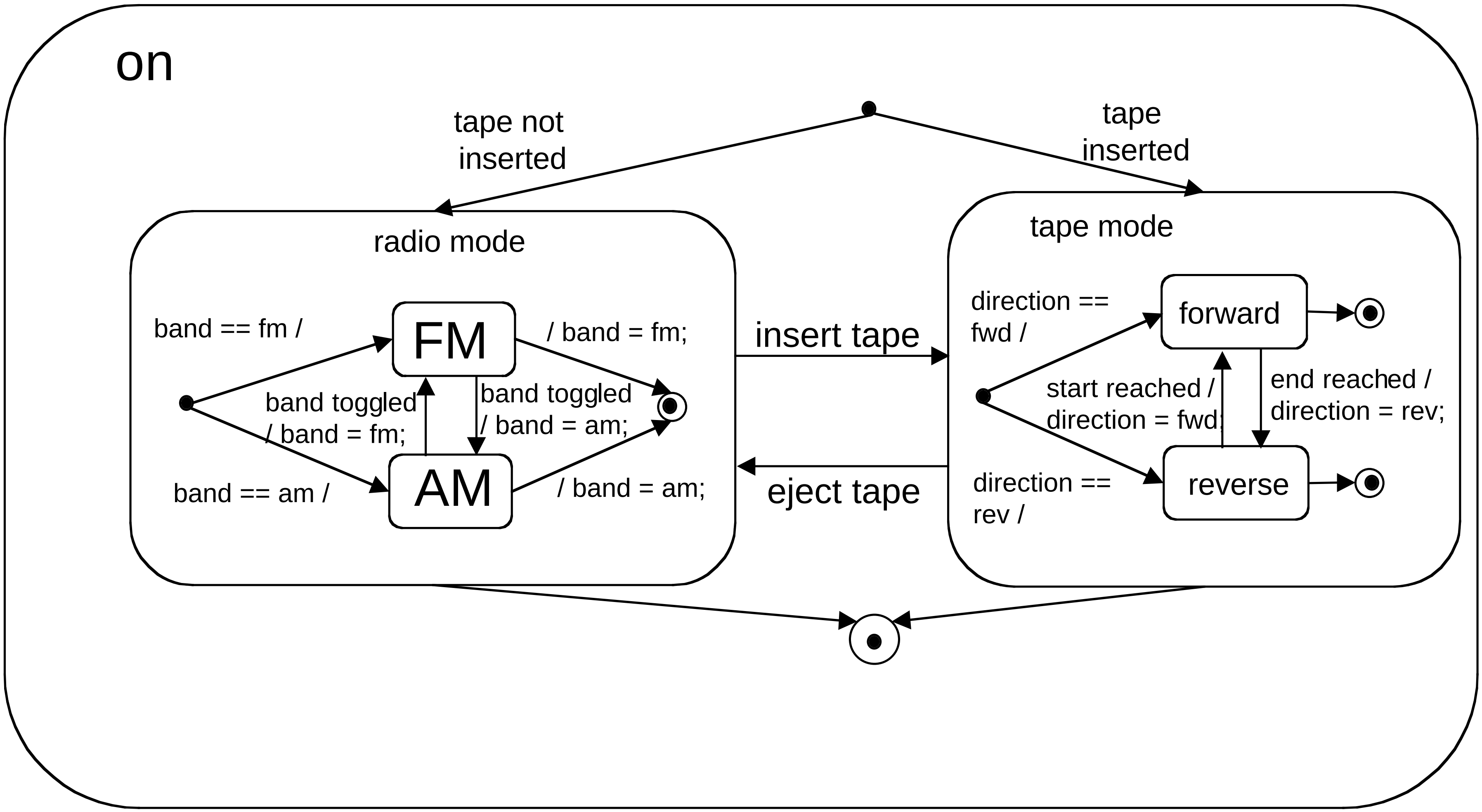
Radio/Tape Player State Diagram

Second-Level Expansion

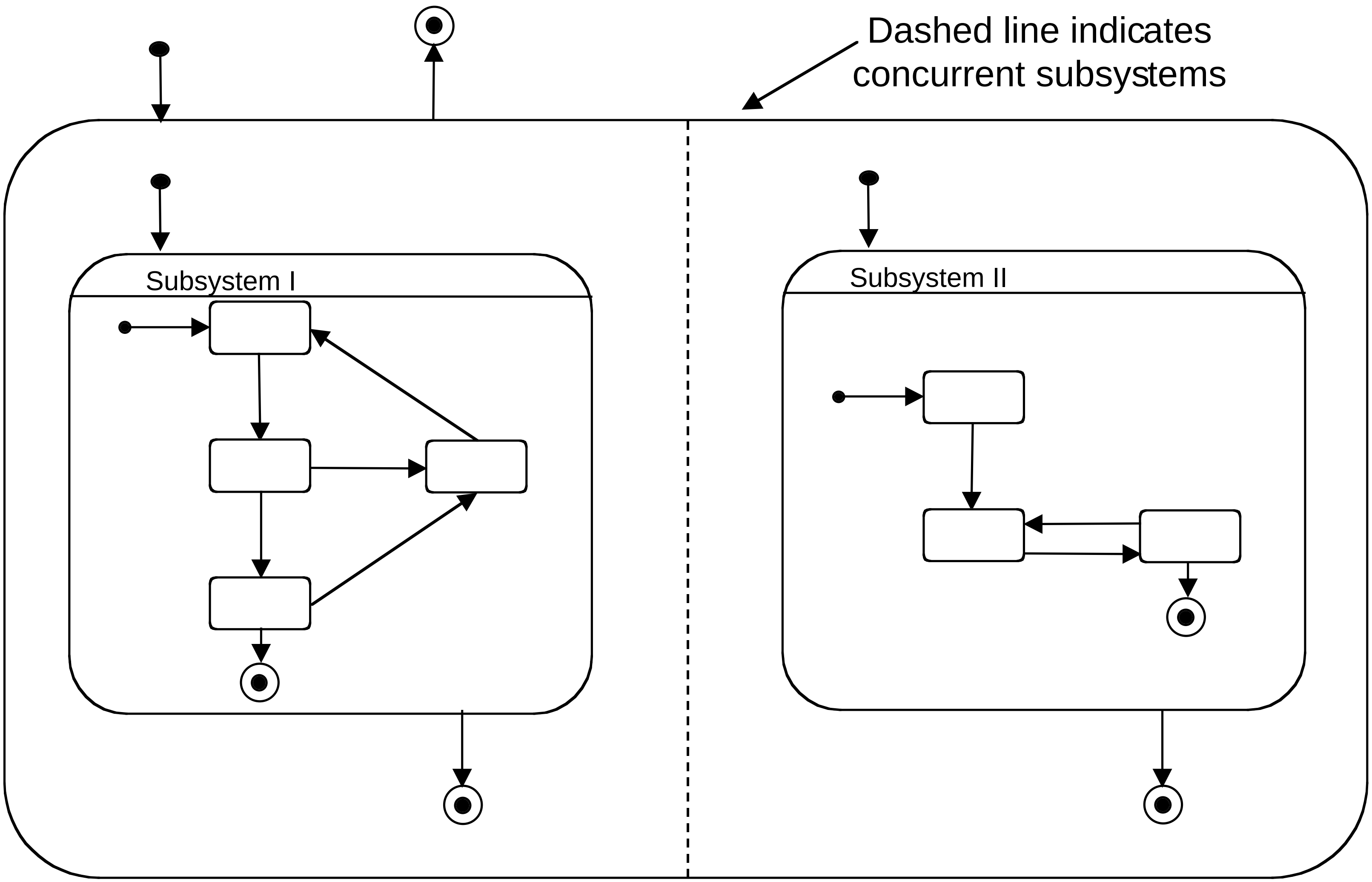


Possible Design Flaws Detected from Diagram

Redesign to Remedy Flaws



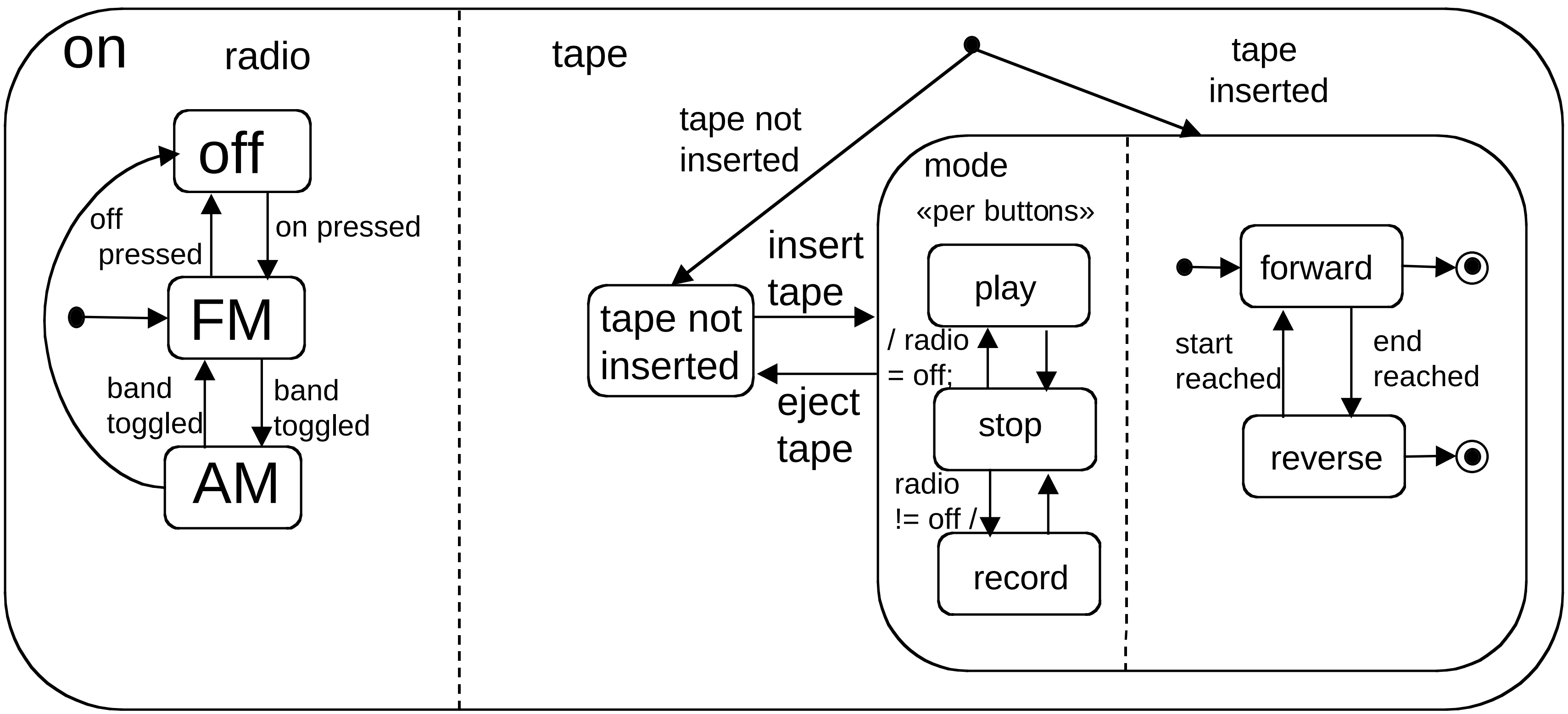
Another Nuance: Concurrency in StateCharts



Augment the Radio/Tape Design to Permit Recording from Radio

- The Radio and Tape can now operate concurrently.
- The tape may record in either direction.
- We can no longer let the presence of the tape dictate the mode; must use buttons.

Partially-Developed Radio/Tape Player/Record



Concurrency in StateCharts provide representational economy

- A statechart with concurrency provides for the representation of M x N states with only M + N bubbles.
- The **combinatorial explosion** of states is thereby reduced.

Data Flow Diagrams (DFDs)

- Classical software engineering tool
- Not part of UML (yet!)
- Data flow can be loosely handled as objects, but object diagrams are ambiguous as to whether they are representing true data flow.

Data Flow Diagrams

- The main idea:
  - Represent system as a directed graph of software modules (nodes)
  - Arcs between nodes represent the "flow" of data from one module to another

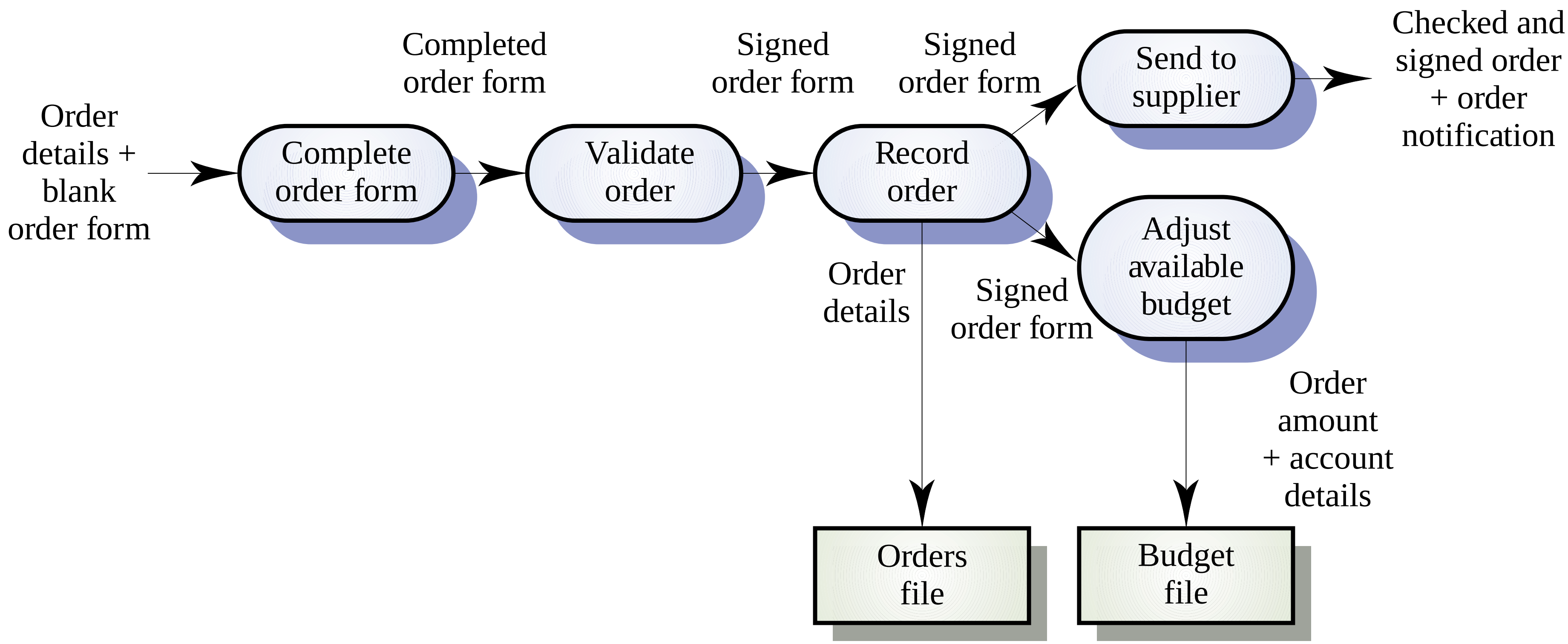
Data Flow Diagrams

- Applications that lend themselves to this model:
  - **“Pipe” composition**, as in Unix pipes
  - Problems treatable as **functional decompositions** on of streams of data
  - **File- and document-processing problems:** The “flow” is represented by intermediate immediate files that are written by one module and read by another.

Advantage of DFDs

- DFDs have an intuitive feel that is easily understood by the customer (vs. Objects, which might be less intuitive), so could be used as a sort of specification language.
- Good for "assembly-line" processing of commodities or documents.
- Data flow makes programming into plumbing.

DFD for Order Processing

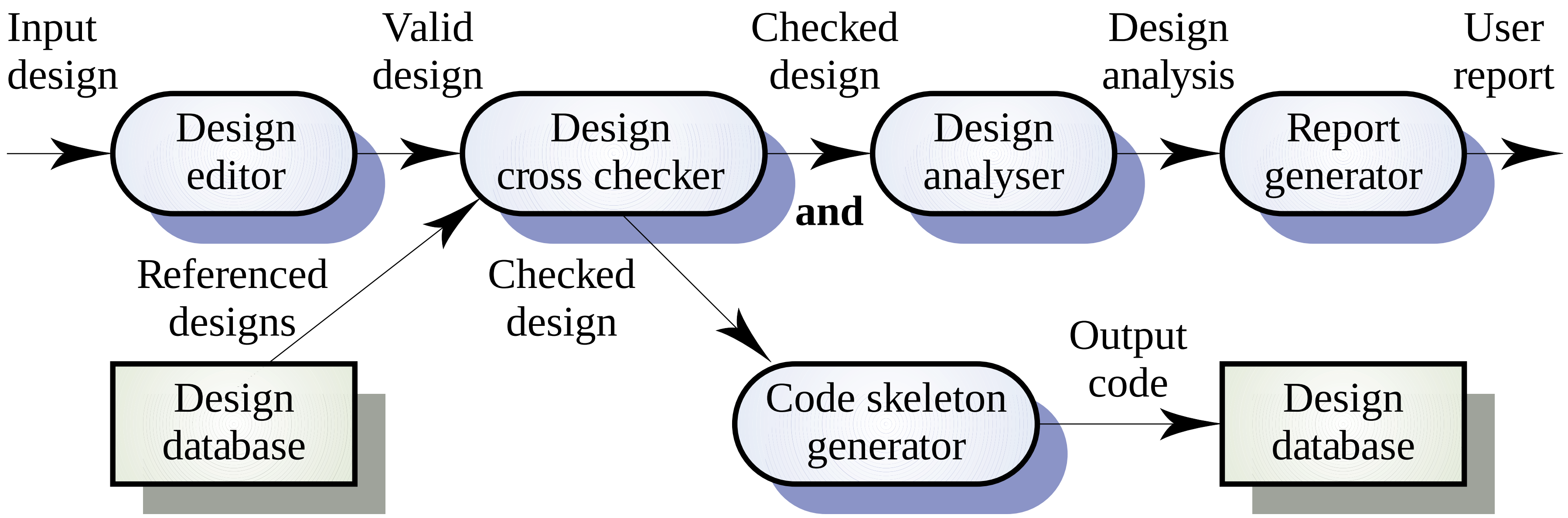


source: Ian Sommerville, Software Engineering, 5th Ed.

More DFD points

- DFD capable of showing various **levels of abstraction.**
- A processing box can be expanded into its own DFD, etc., enabling "**zooming**" into the structure of the system
- Does not particularly lend itself to showing ***interactive*** behavior, since output is separated from input.

DFD Example with Databases



source: Ian Sommerville,
Software Engineering, 5th Ed.

Data Dictionaries

Data Dictionaries

- Glossary or “database” of names and associated descriptions of entities, associations, attributes used in the system.
- represents a project-wide shared repository of system information
- provides:
  - A mechanism for name management. As a system model may be developed by different people, there is potential for name clashes
  - A link from analysis to design and implementation

Data Dictionary Example

Name	Description	Type	Date
has_labels	1:N relation between entities of type Node or Link and entities of type Label.	Relation	5.10.93
Label	Holds structured or unstructured information about nodes or links. Labels can be text or can be an icon.	Entity	8.12.93
Link	Represents a relation between design entities represented as nodes. Links are typed and may be named.	Relation	8.12.93
name (label)	Each label has a name which identifies the type of label. The name must be unique within the set of label types used in a design.	Attribute	8.12.93
name (node)	Each node has a name which must be unique within a design. The name maybe up to 64 characters long.	Attribute	15.11.93

source: Ian Sommerville, Software Engineering, 5th Ed

Data Dictionaries

- Ideally a database in their own right
- Remain intact for the lifetime of the project
- Useable by management and developers
- Useable by customer
  - if source-access is provided
  - legacy file formatting
  - as a description of processing entities (view technical details)

DD Examples on the Web

- www.cris.state.nc.us/datafrm.html
- nccs.urban.org/STalmDD.htm

- Use your browser for many other examples

Info in Data Dictionaries

- Can be handled by UML tool, e.g. in class diagrams and their documentation.
- Report generator may be able to give conventional data dictionary format.