Requirements Techniques, cont.

- Formal requirements analysis techniques include:
  - DFD (covered)
  - ERD (covered)
  - **Finite State Machines**
  - **Petri Nets**

Finite State Machines

- A requirements technique for modeling the states and transitions of a software system
- Finite state machines are used in other contexts
  - automata theory and compilers, for example
- More precise than data-flow diagrams
  - data-flow diagrams specify only the nature of a system's data flow (e.g. the what and the where)
  - finite state machines provide information on how a system progresses from state to state
    - ¤ what is "state"

Formal Definition

- The definition of a finite state machine (FSM) consists of five parts
  - a set of states
  - a set of inputs
  - a transition function
  - the initial state
  - a set of final states

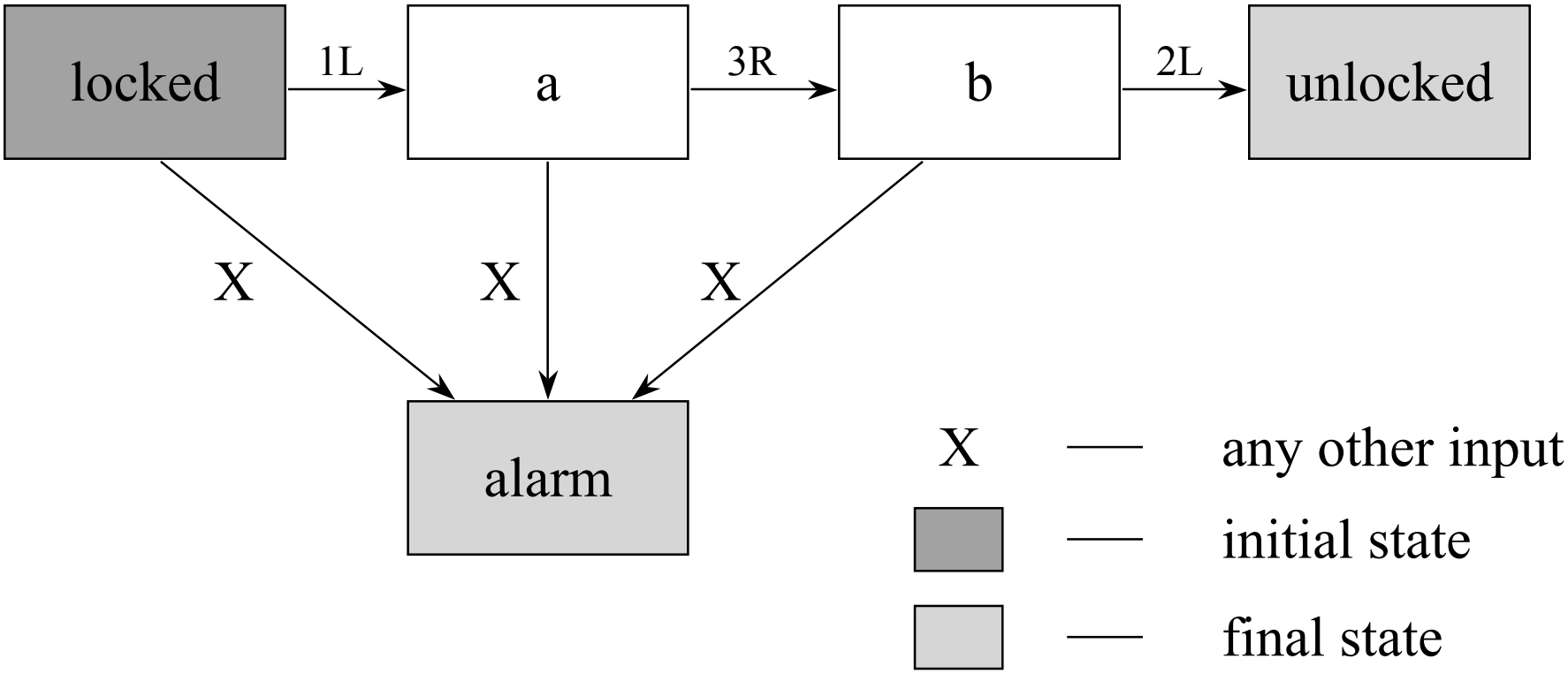
Simple Example

- States of a combination lock for a safe
  - Safe has a dial with three positions (1, 2, & 3)
  - The dial can be turned in two possible directions
    - ¤ At any point six possible motions
      - Turn left to 1 (1L)
      - Turn right to 1 (1R)
      - etc...
  - Combination is 1L, 3R, 2L
  - The possible states include the safe being locked and unlocked, sounding the alarm, and the steps along the combination (e.g. 1L and 3R)

Example, cont.

- Set of States (Locked, A, B, Unlocked, Alarm)
- Set of Inputs (1L, 1R, 2L, 2R, 3L, 3R)
- Transition Function (next two slides)
- Initial State (Locked)
- Final States (Unlocked, Alarm)

Example, cont.



Transition Table

Input	Locked	A	B
1L	A	Alarm	Alarm
1R	Alarm	Alarm	Alarm
2L	Alarm	Alarm	Unlocked
2R	Alarm	Alarm	Alarm
3L	Alarm	Alarm	Alarm
3R	Alarm	B	Alarm

Finite State Machine Wrap-Up

- More advanced examples in other textbooks
  - The infamous "Elevator Example" is a good one (Schach)
- Demonstrates
  - The specification power of FSMs
- Typical Problem
  - The number of states and transitions grows rapidly in large systems
  - Approach: decompose problem into smaller subsystems
- Tool support exists for this and related techniques (e.g. statecharts)

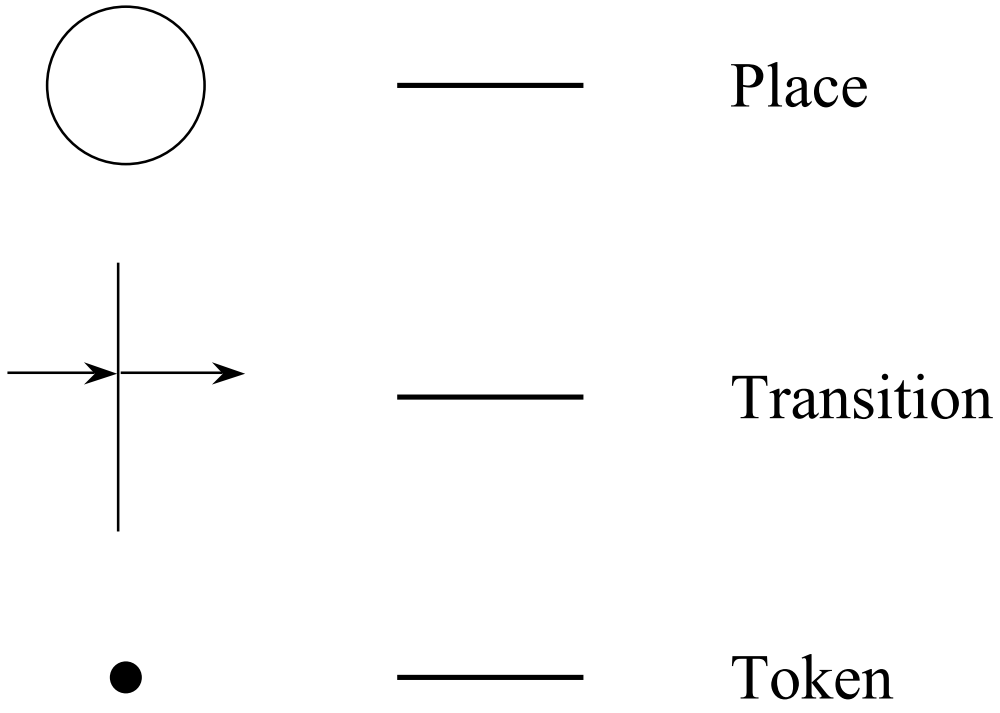
Petri Nets

- A formal technique suited for specifying the properties of concurrent or multithreaded systems
- Typical concurrency problems
  - race conditions
    - X accesses Y before Z updates it
  - deadlock
    - X is waiting on Y which is waiting on X
- Petri nets can be used to help avoid ambiguity in specifications that can lead to this class of problems in multithreaded systems

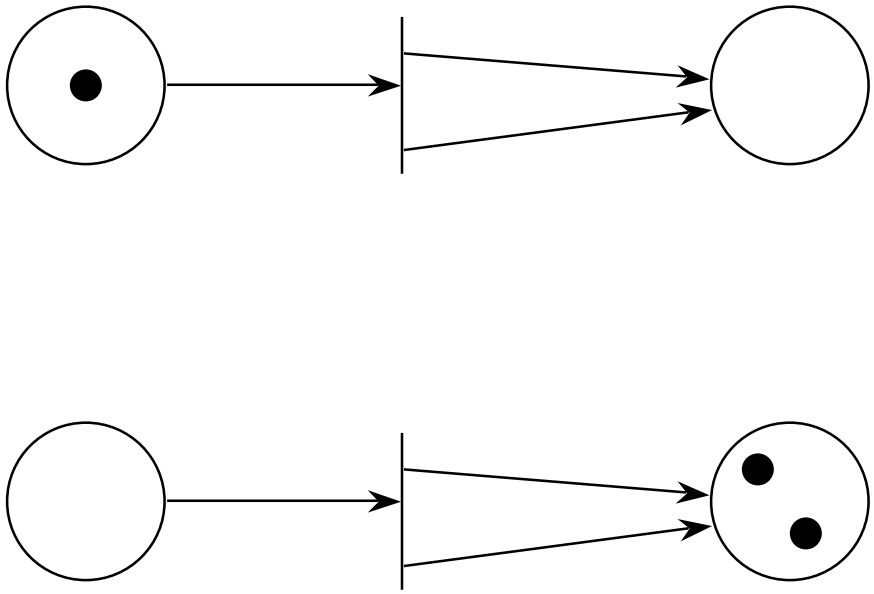
Formal Definition of Petri Nets

- A Petri net consists of four parts
  - A set of places
  - A set of transitions
  - An input function
  - An output function
- In the subsequent diagrams, the input and output functions are represented by arrows

Petri Net Parts



Firing a transition

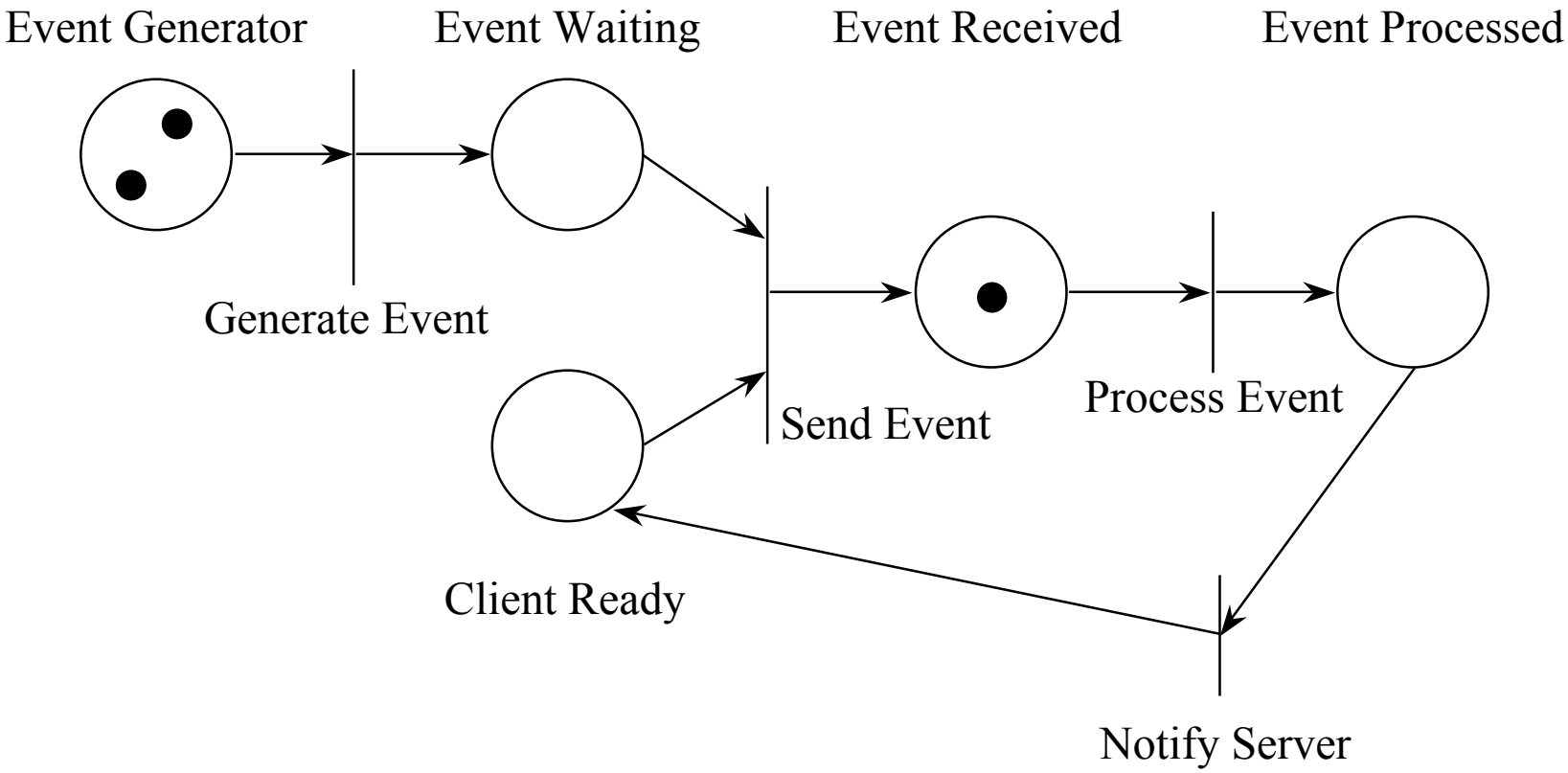


A transition fires when it has a token at each input place; as a result a token is placed at each output place.

Simple Example

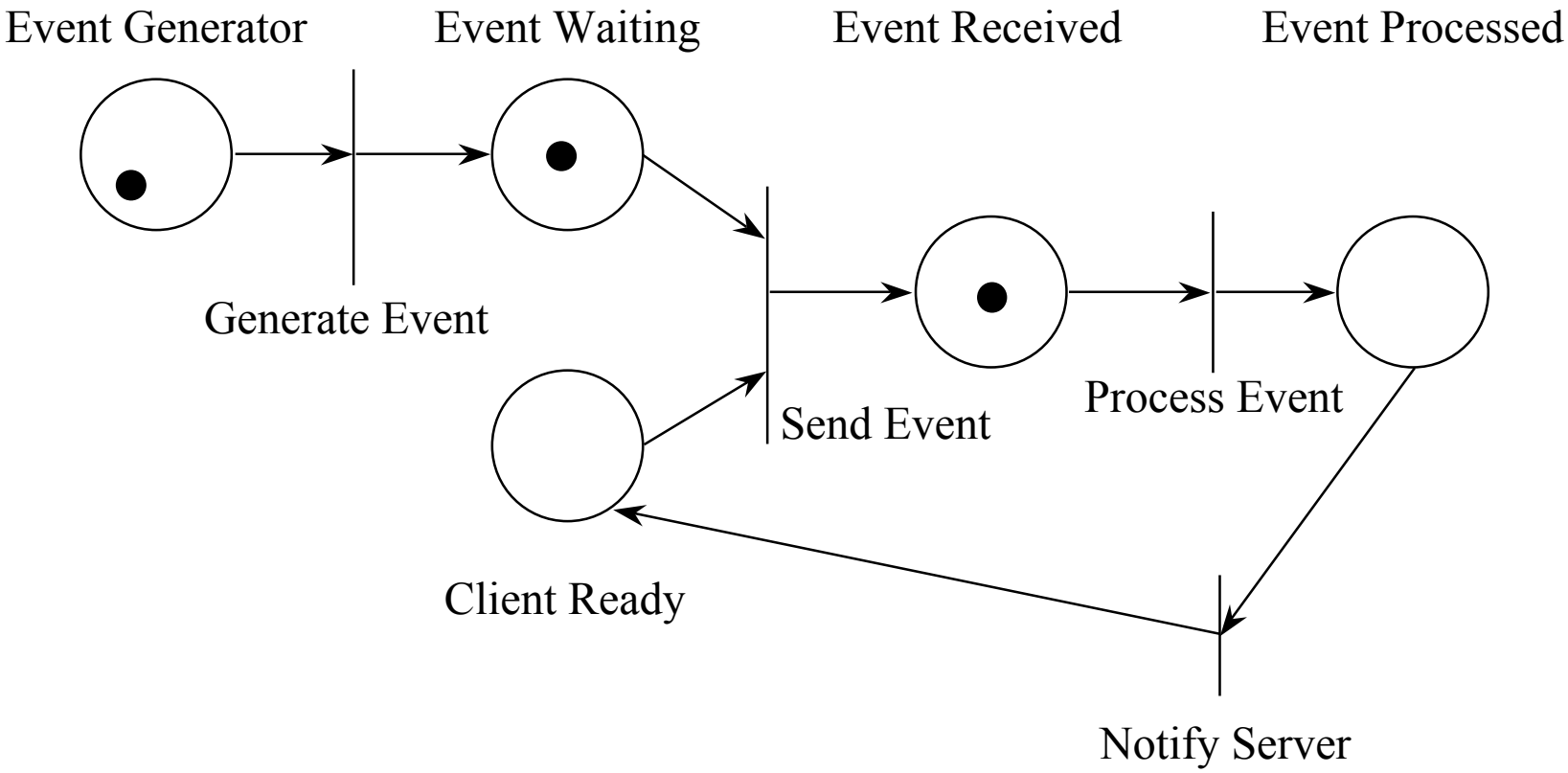
- A server cannot send an event to a client until the client has indicated that it is ready to receive one
- Approach
  - Map states into places
  - Map events into transitions
- States
  - Event waiting
  - Event received
  - Event processed
  - Client Ready
- Events
  - Generate Event
  - Send Event
  - Process Event
  - Notify server

Example, cont.



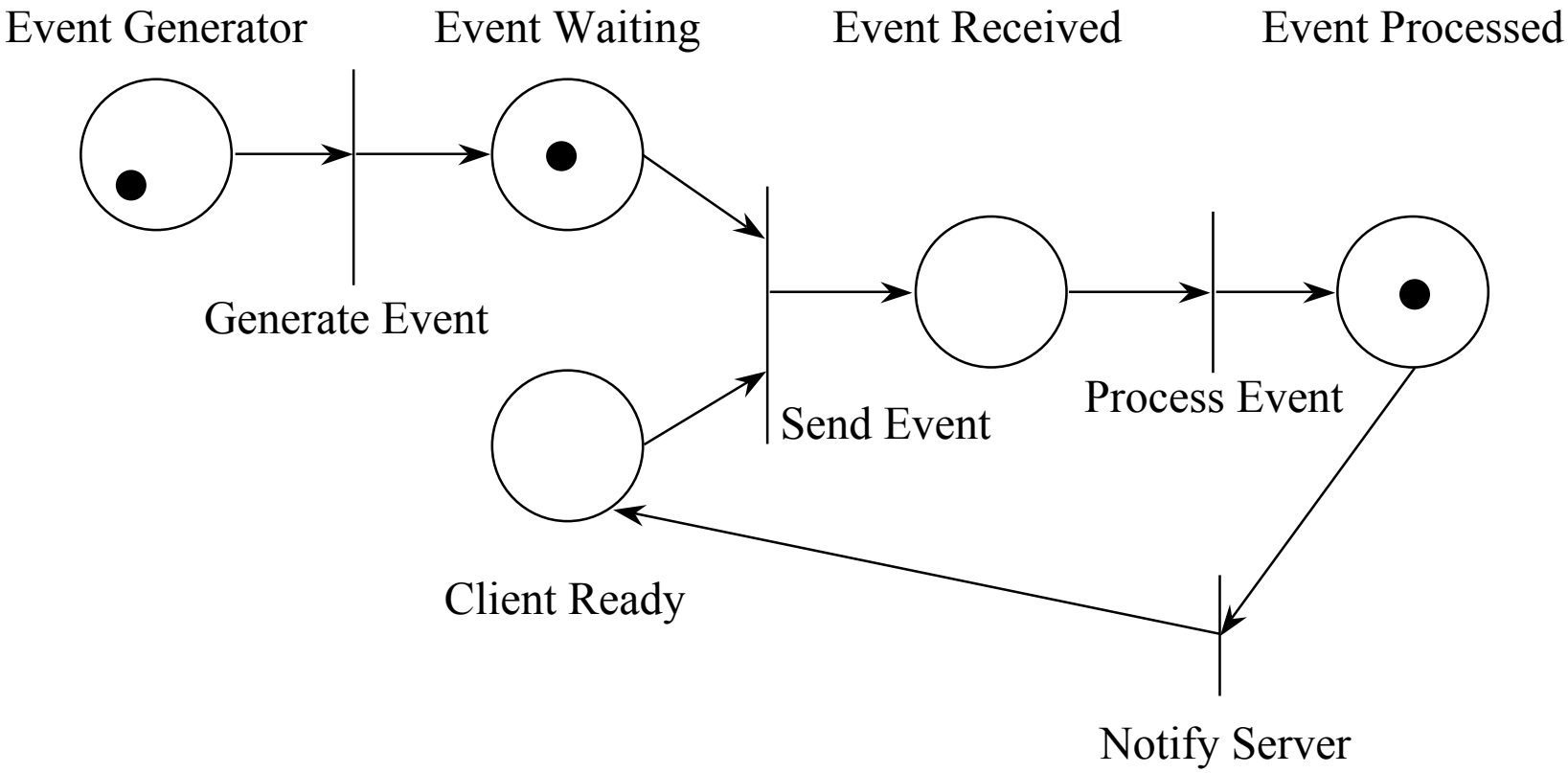
Three events; one being received

Example, cont.



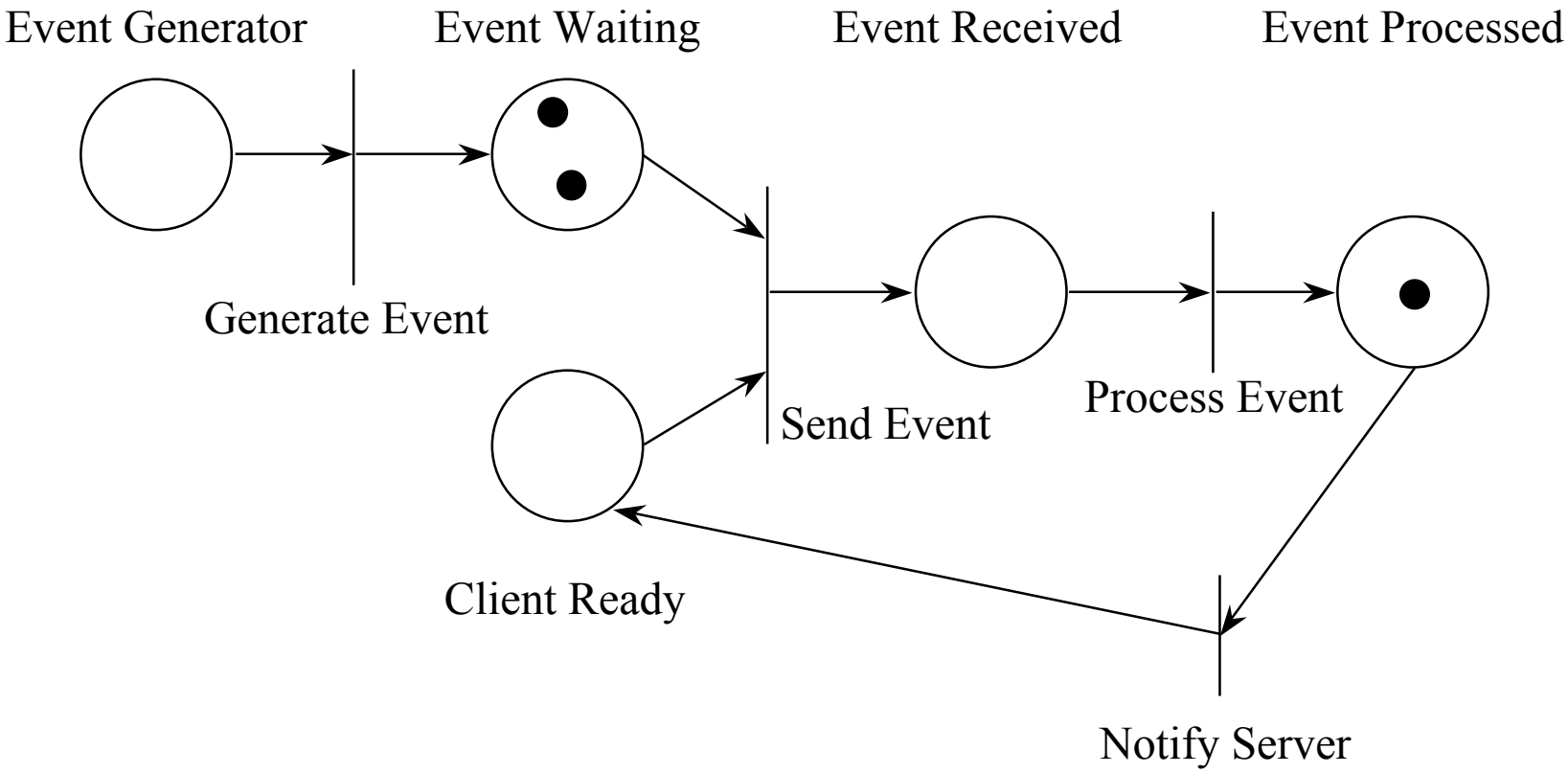
An event gets generated

Example, cont.



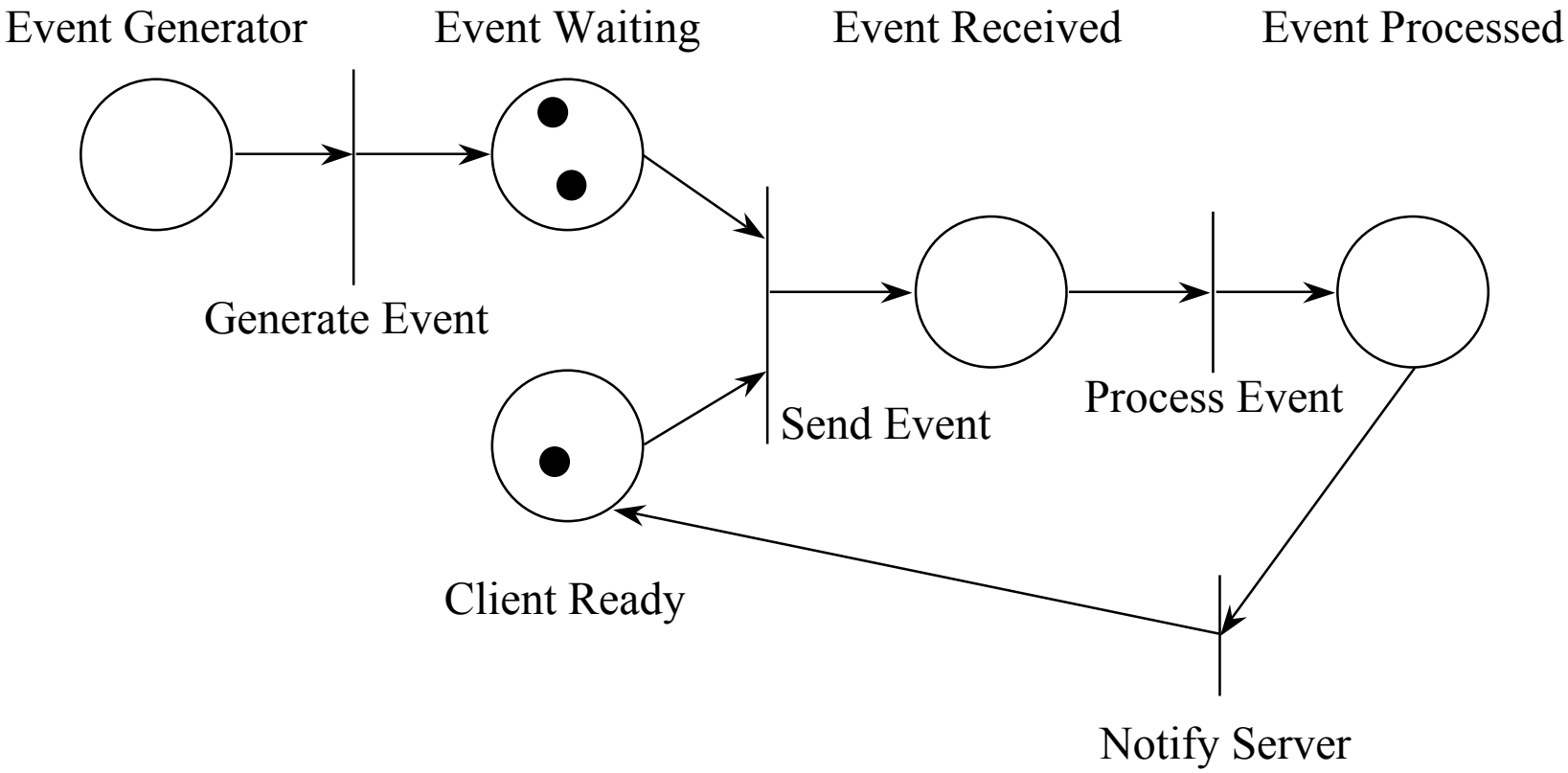
An event gets processed; waiting event must wait since “Send Event” cannot fire.

Example, cont.



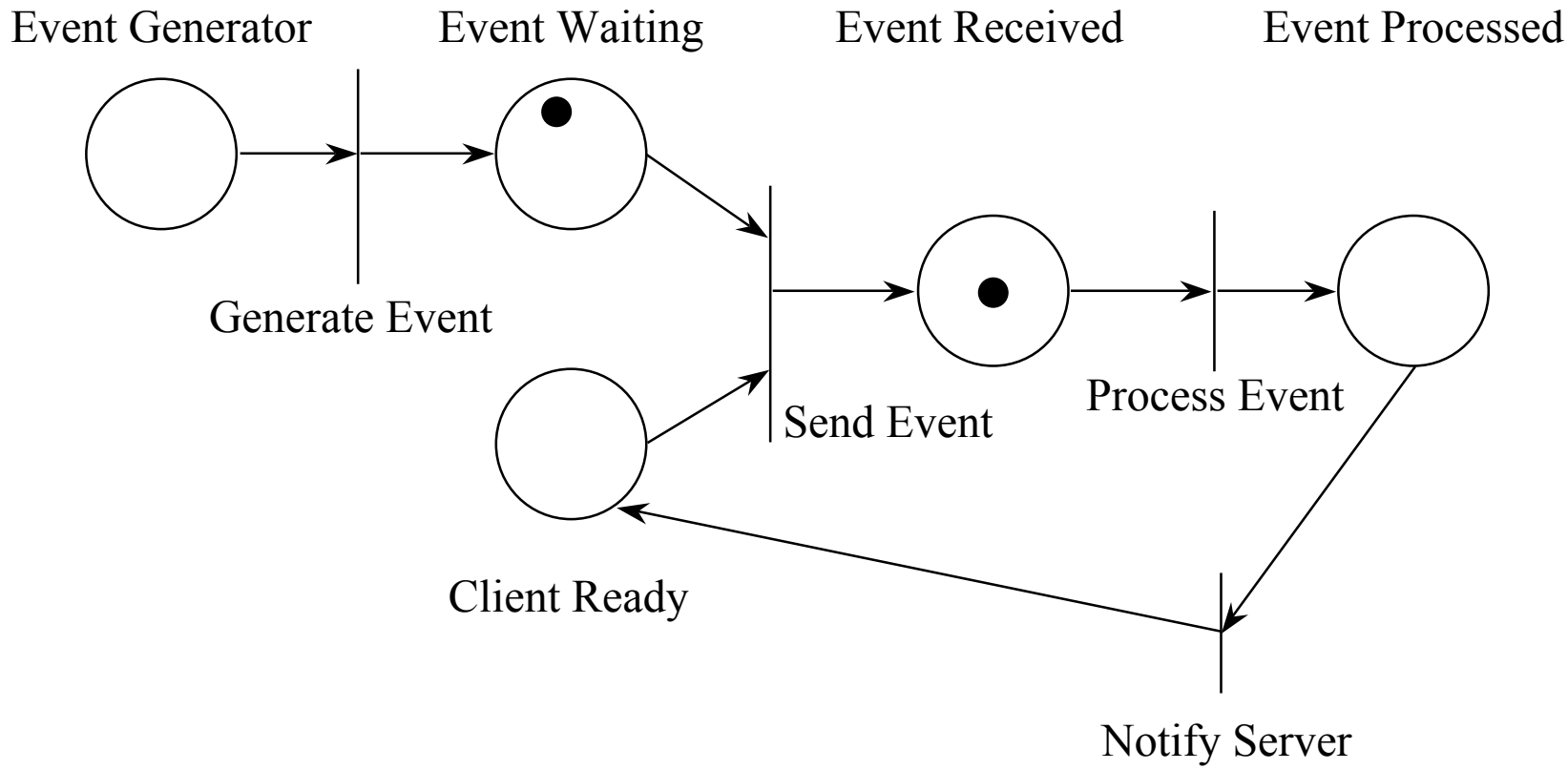
Another event gets generated

Example, cont.



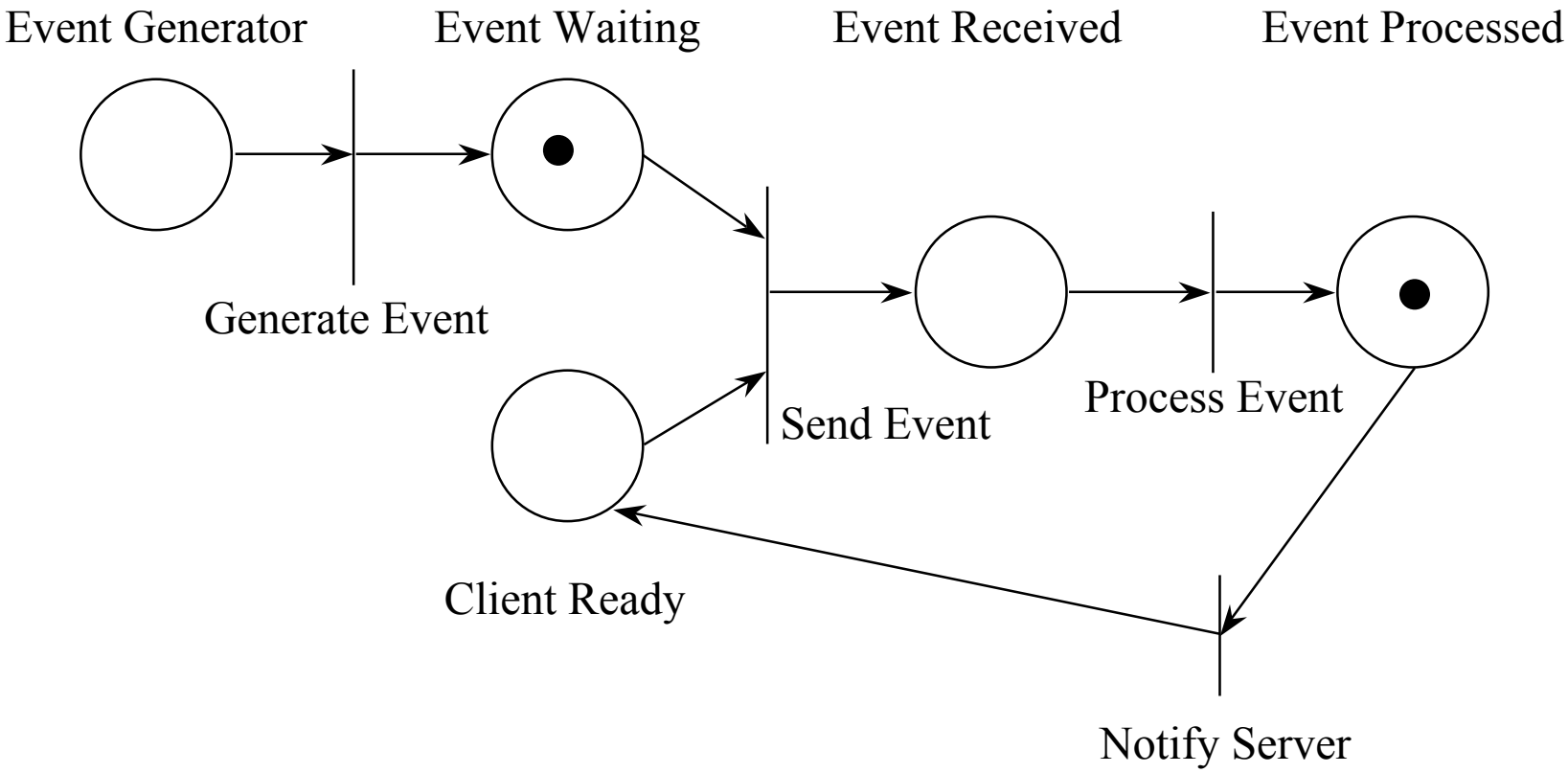
The client notifies the server

Example, cont.



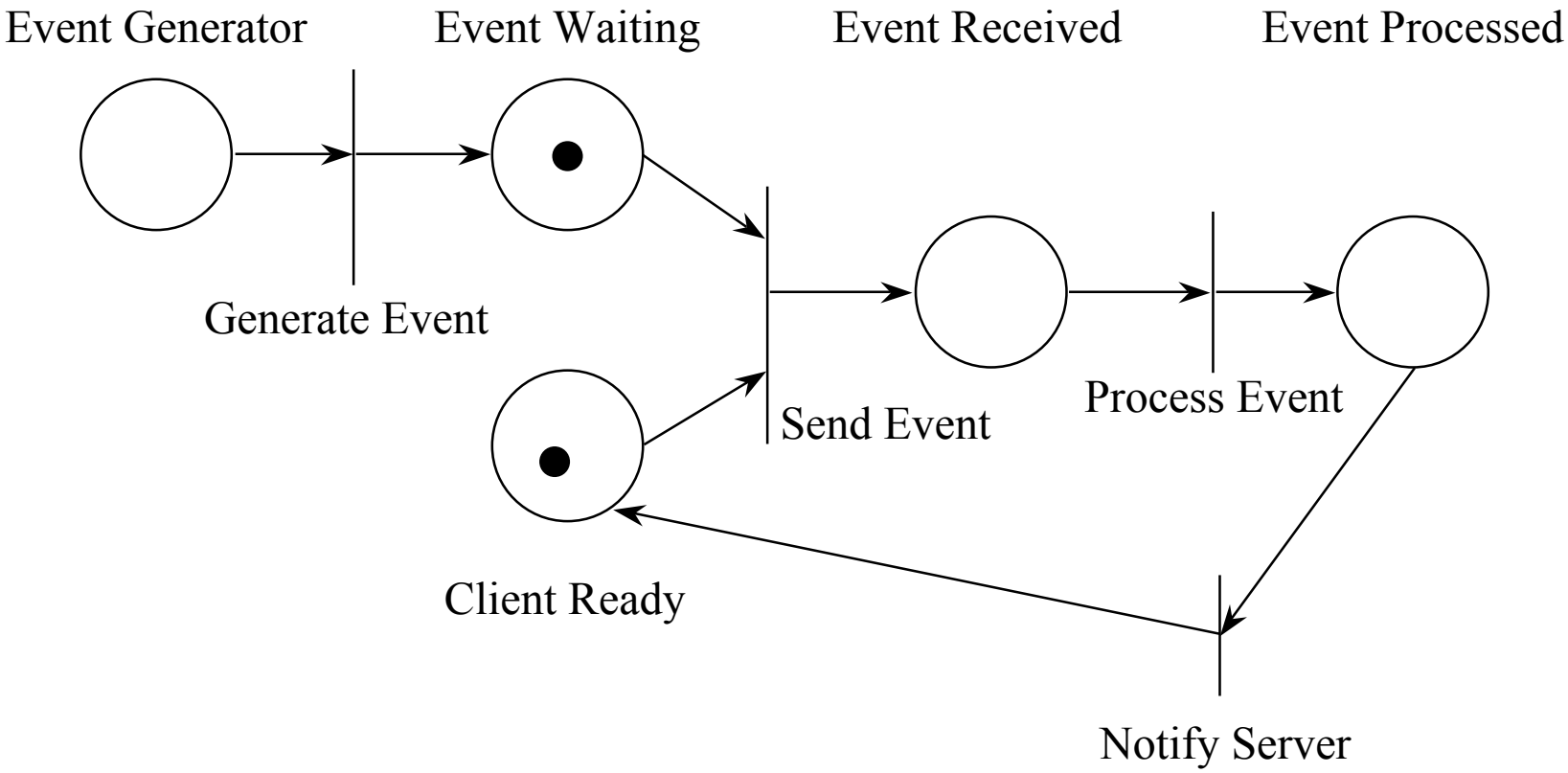
An event is sent...

Example, cont.



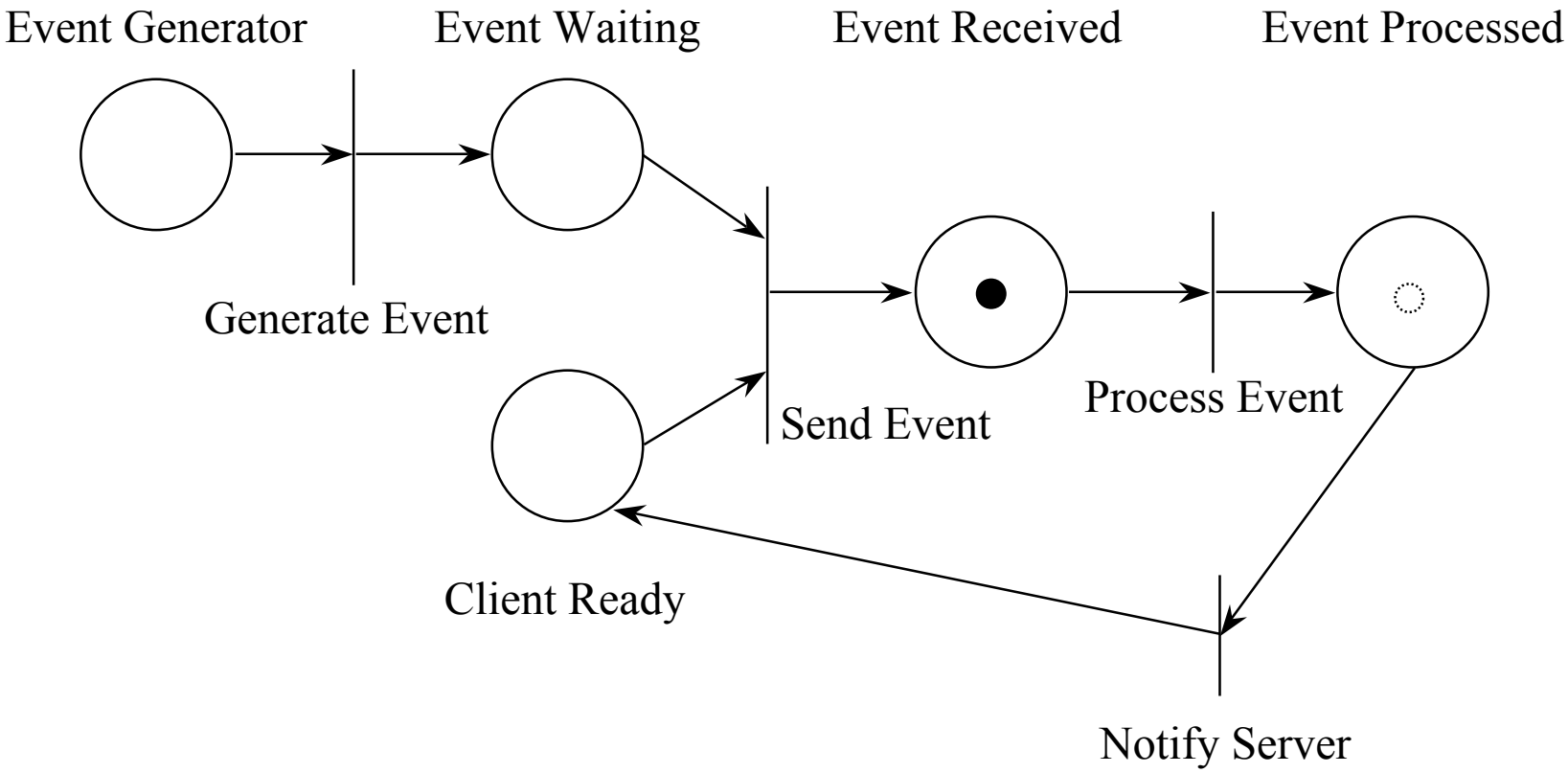
Second event is processed...

Example, cont.



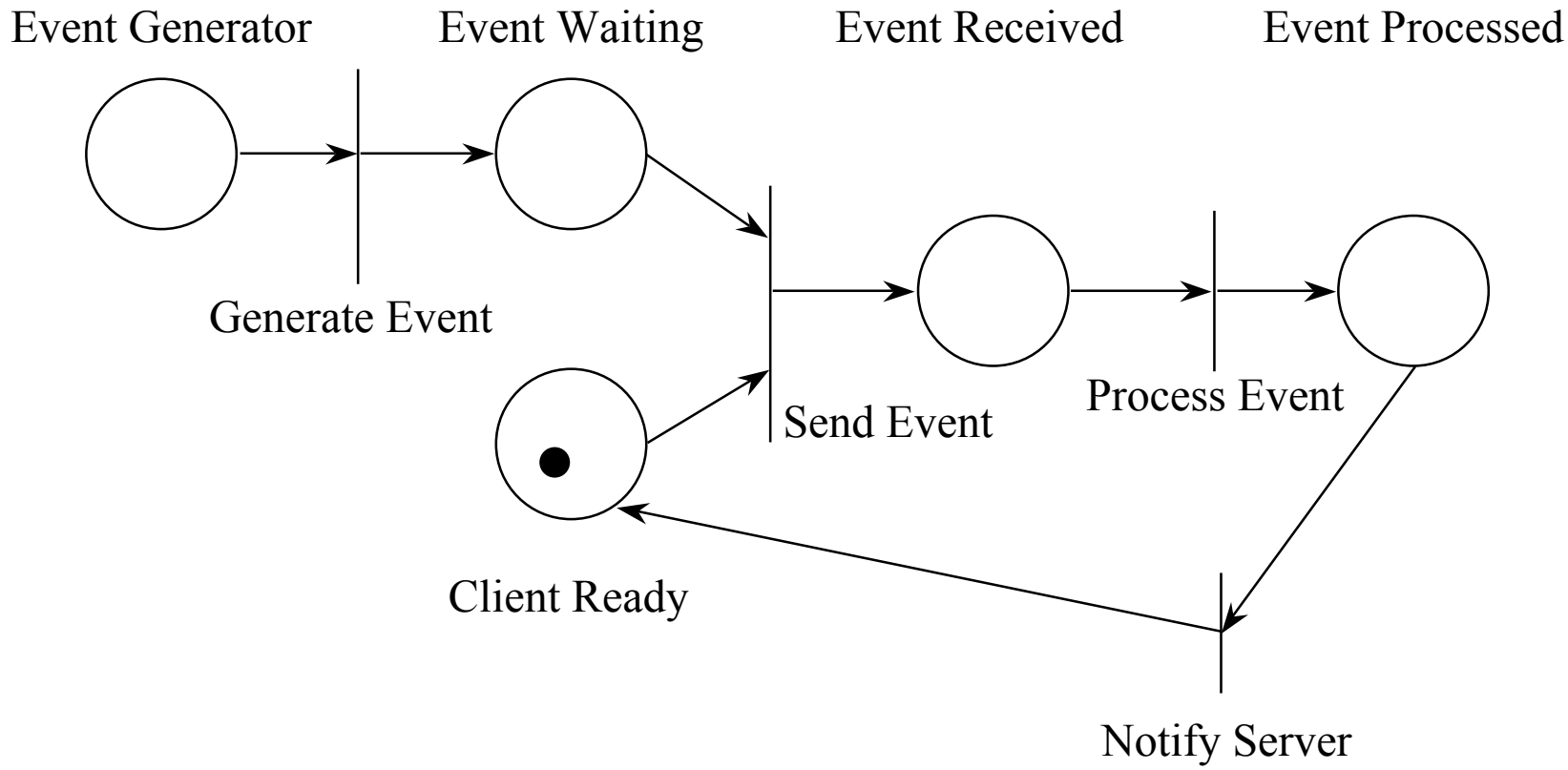
The client notifies the server

Example, cont.



Final event is sent...and
eventually processed (ghost token)

Final State



Client is ready for next event

Petri Net Wrap-Up

- Clean notation for specifying concurrent properties
- Graphical notation hides underlying formalism
  - Makes it easier to understand
- Tool support and execution engines exists for this technique
  - The latter can help in testing how well a Petri net specifies a property by running it on test cases