Consensus - GE

Executive Summary

Patrick Bouchaud

2017-05-17

Contents

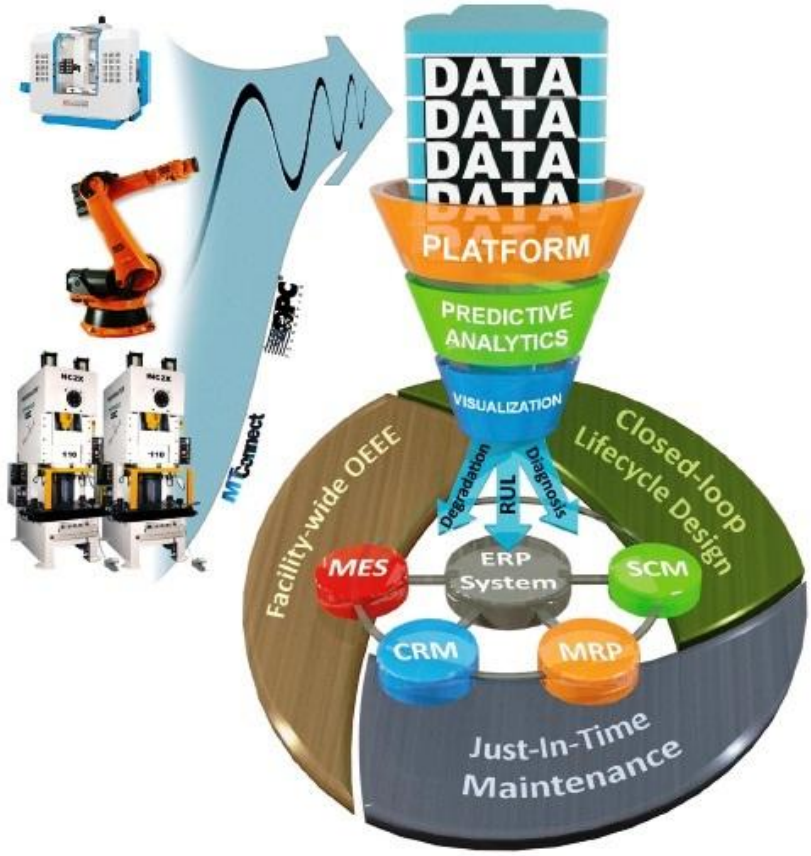
Objective

Establish GE as

The world's Digital Industrial Company, transforming industry with software-defined machines and solutions that are **connected**, **responsive** and **predictive**.

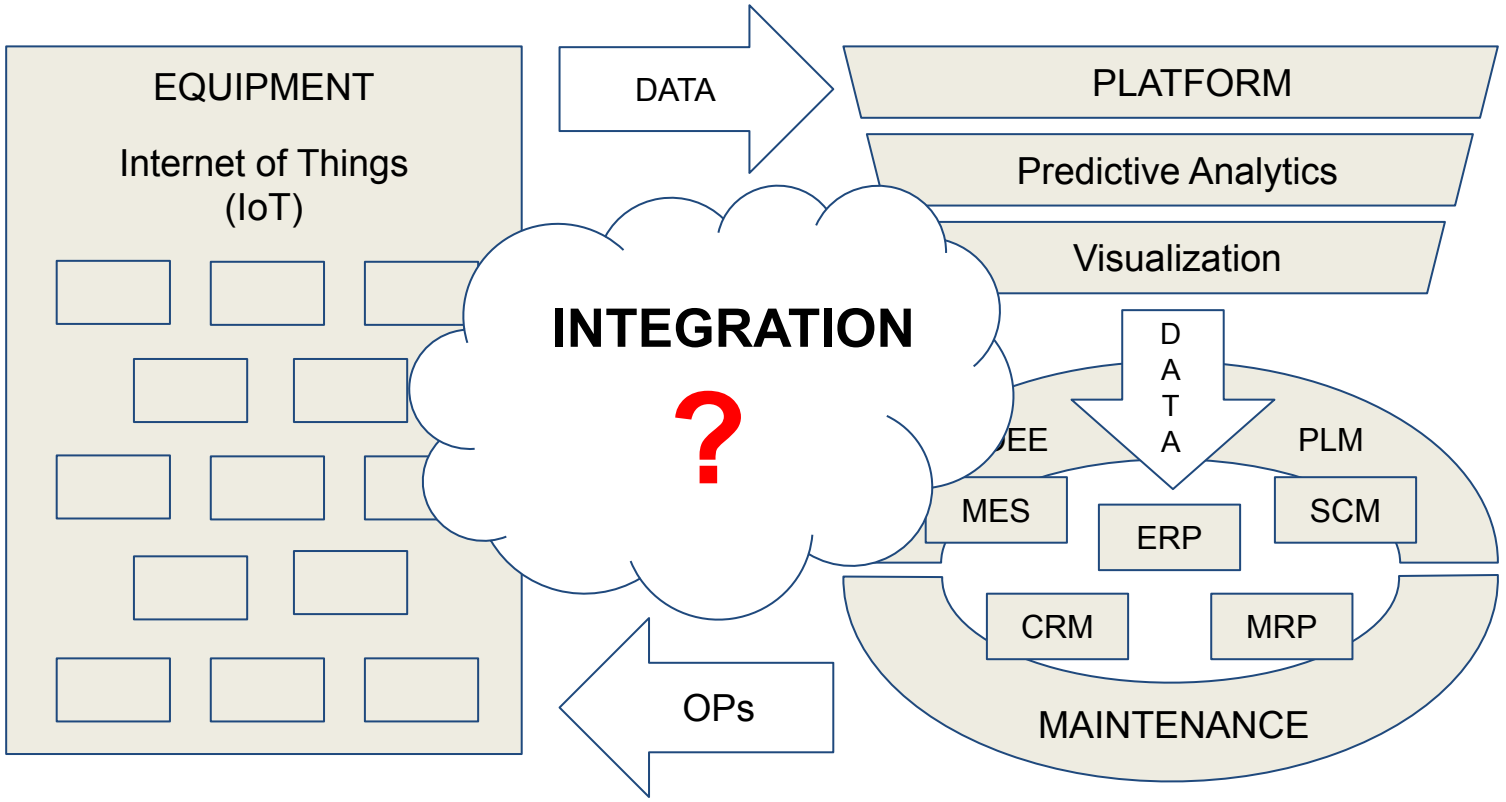
Through GE people, leadership development services, technology and scale, deliver better outcome for global customers by

speaking the language of the industry.



Sources: Text: About Us - GE advertisement. Picture: Recent Advances and Trends in Predictive Manufacturing

Challenge



Everything *is* **connected**, BUT…

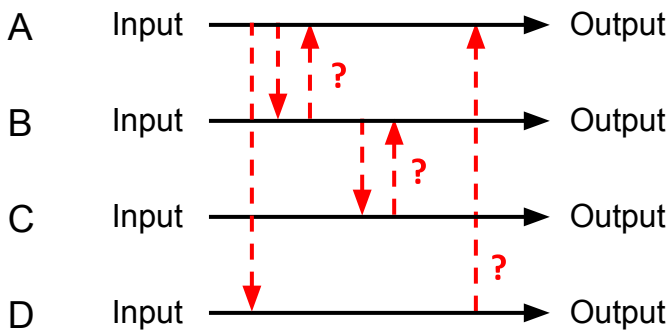
Every thing (including people) has its own, limited view of **how** ;

& domain-specific **language(s)**

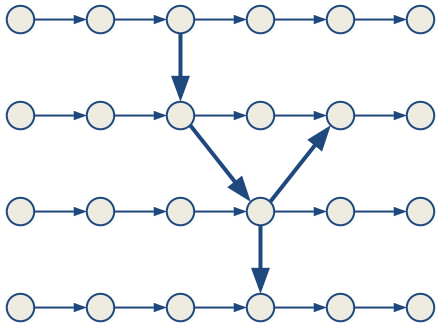
FACT

Requirements

- For the GE predictive manufacturing system to be organically viable - i.e. to ensure a proper flow of information and operations across the overall system - we need a **formalism** which allows to
  - represent and navigate the complete worldview of each participating entity using the entity's domain-specific terminology and processes
  - represent and navigate the ontological, structural and functional dependencies *across* these worldviews so as to bridge the associated information silos
  - translate any subset of these representations into executable code and feed the model with real life data so as to monitor, simulate, test or prototype the system's execution, extension, coupling or modification



Entity process / dependency mapping



cross-domain process / workflow definition

Translation

- Modern industrial challenges such as the IoT and predictive manufacturing require a **fundamental paradigm change in IT**, where existing software development methodologies and programming languages are still geared towards producing a target outcome from given inputs, in **linear** fashion.
- The predatory approach, consisting of implementing the fastest path towards a target result and worrying afterwards about side effects, leads to the constitution of **information silos** as the overall dependency map which led to the optimal path determination is abandoned as transient artifact - or, in the best case, separate documentation.
- In order to achieve, maintain and evolve the targeted predictive manufacturing closed-loop design,
    - The map, or **topology of the information network**, must change from its current *transient* to a living *persistent* status, and allow users and systems each with their own domain-specific language(s) to inform and navigate across information silos.
    - The territory, or actual tool landscape, must be coupled with the map in such a way as to **reflect any change** from either one to the other, in **real time**.

Solution

- Consensus is an Entity Behaviour & Relationship Model (EBRM) which allows to represent entities and their relationships ***as they are*** in regards to the organization of information ***in the human mind***.
- Consensus is a peer-to-peer software architecture and communication protocol which allows to apply the Consensus EBRM to the organization of data (= digital information) within and across databases and information systems.
- Not only does Consensus provide a basic yet complete formalism to represent, query and navigate everything, from the way data are organized to the way changes propagate within these data. Consensus also allows to execute the model and **bring the representation to life**.

Consensus	
Key Features	Benefits
- Consensus Entity Behaviour and Relationship Model - Programming language and command line interface - Peer to peer execution model - Automatic software parallelization and network distribution - HTML integration (Contents Management System) - Immersive user interface (3D)	- Facilitate business process description and data model specification - Facilitate creation, evolution and maintenance of automated workflow solutions - Allow evaluation and propagation of impact associated with change - Allow easy integration of new or existing software

Proposition

- Consensus is currently at the proof-of-concept / MVP stage.
- The proposition is, based on the Consensus concepts, for **GE to position itself at the forefront** and **drive** the required **IT transformation**, starting from the Visualization domain and gradually expanding to all connected areas of the predictive manufacturing system.
- The benefits for GE are to
    - Increase ownership & control - Intellectual Property, compliance certification
    - Optimize product development lifecycle - increased component re-usability
    - Strengthen existing market presence and create new market opportunities
    - Reinforce image - possibility to tackle social and environmental issues such as education and climate change, etc.

"**Conceptual simplicity**, **structural complexity** achieves a greater state of **humanity**." -Appleseed

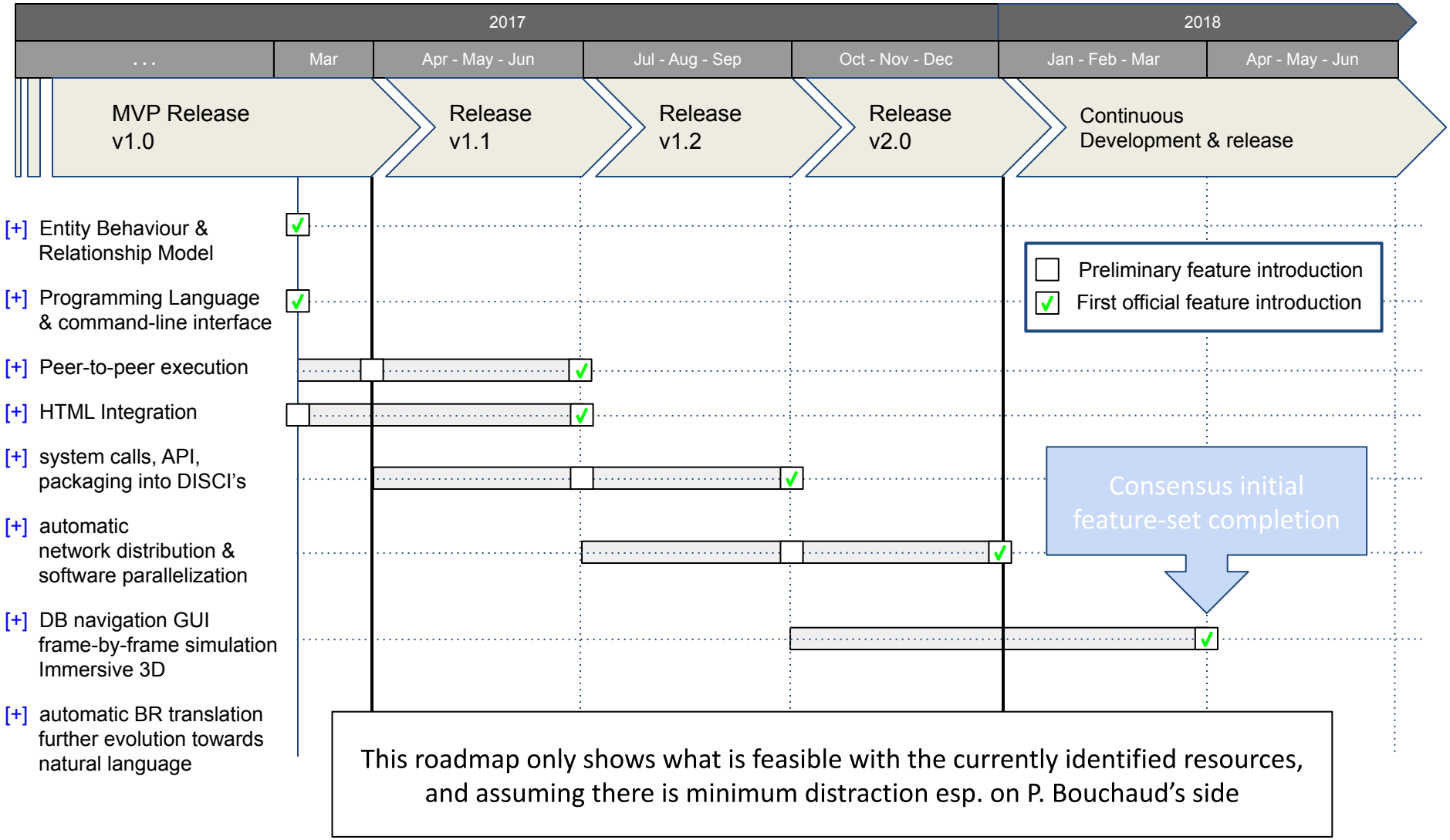
Thank You!

Backup Slides

Consensus Feature List (prioritized)

Feature	Status	Remaining	Effort
Entity Behaviour and Relationship Model	Database interface core features implementation done	. value banking operations (generalized) . Lists & file system interface (backup/restore)	S
Programming Language and command-line interface	Core architecture & featureset implementation done	. industrial use case & example . featureset completion / nice-to-haves, etc.	S
Peer-to-peer execution model	Framework is in place, implementation to be finalized	. Frame execution . interface to launch a child from parent process . external sources & inter-process communication (events, url interface)	M
HTML integration	Framework is in place, HCN-HTML output implementation done	. cgi script implementation . consensus server implementation	M
Extensibility and customization (DISCI)	syscall & API prototype implementation done	. API completion & library packaging . automatic business requirement translation . example and use cases	L
Automatic software parallelization and network distribution	not started	. external sources & event management . event client-server implementation	L
immersive user interface (3D)	not started	. Database navigation graphical user interface . frame-by-frame simulation	L

Consensus Product Roadmap 2017- 2018



Contents

Consensus entities

- Each Consensus entity is uniquely identified by an `expression` which takes the general form:

  `source-medium->target`

  where

  `source`, `medium` and `target` can be either

	nothing
`identifier`	an identifier (name) or group of identifiers enclosed in double-quotes (“)
`expression`	in the form `source-medium->target`

- identifiers are also entities.
- Each Consensus entity, once instantiated, can be in either one of two states: active, or inactive.
- Consensus entities are inactive by default.

Consensus back-pointer notation

- While entities are stored using the fore-pointer notation

  ```
  source-medium->target
  ```

  Consensus allows to query and output entities using the back-pointer notation

  ```
  target<-medium-source
  ```

  and to mix fore- and back- pointer notations within the same expression

- This feature is key with respect to Consensus current proximity and future evolution towards natural language, as illustrated by the example provided in the *Example 1. Command-line interface* section of this document.

Consensus queries

- Consensus entities can be queried by providing a `query expression` where some or all of the terms have been possibly

  replaced by the dot (**.**) character, to represent *any*

  prefixed by the tilde (~) character, to represent *not*

  prefixed by either the star (*) or the underline (_) character, to represent respectively *active* and *inactive*

- The question mark (?) character, without any prefix, may also replace one, and only one, of the terms of the expression, in which case the query will return the list of entities matching that specific term, while keeping the overall expression successful.

- Consensus allows other query expressions along the same general principles as presented here. The complete list of all possible query expressions is provided in the Appendix A of this document[1].

1. see Consensus - Reference Manual

Consensus variables

- Consensus allows to use variables in order to

    1. build complex query expressions while keeping these expressions readable.
    2. transport and reuse query results.

- Consensus variables values can be set to either one of the following:

    `: variable : %[ expression ]` sets `variable` value to the `expression` query results

    `: variable : expression` sets `variable` value to the provided expression

    the difference being that in the first case the entities corresponding to the variable value are evaluated once and for all upon the variable assignment, whereas in the second they are evaluated each and every time the variable value has to be resolved.

- `%variable` can then be used as a term of other expressions including query expressions, that is: the variable name prefixed by the character percent (%) is used to reference the variable value.

- Consensus allows other variable value assignments, as will be presented in the following section of this document[1].

1. see Consensus - Reference Manual

Consensus conditional execution mode & operations

- Consensus allows to specify a condition for the execution of subsequent commands, as follows:

`?%: expression`	allows execution of subsequent commands if the query `expression` yields result.
`?~%: expression`	disallows execution of subsequent commands if the query `expression` yields result.
`/~`	sets the current condition to its contrary, that is: disallows execution of subsequent commands if command execution was previously allowed, and allows execution of subsequent commands if command execution was previously disallowed.
`/`	exits conditional execution mode.

- This functionality in conjunction with the Consensus loop execution mode & operations is particularly useful in the context of Consensus HTML-HCN output operations, as illustrated in the example provided in the *Example 2. Consensus HTML integration* section of this document.

Consensus loop execution mode & operations

- Consensus allows to execute commands in a loop over the results of a query expression, as follows:

`?: expression`	sets start loop over the results of the specified `expression`
`/`	sets end loop

- Individual expression results can be accessed through the Consensus variator variable (`%?`) in-between the *set start loop* and *set end loop* commands.
- The Consensus variator variable (`%?`) supports all Consensus variable operations, e.g. it can be used as a term in query expressions, its value can be re-assigned to other variables, etc.

Consensus value accounts

- In the future, Consensus will allow any entity to act as a Value Bank (`$$`) and hold value accounts (`$`) for other entities.
- To date however the implementation only supports the Consensus default Value Bank, which holds Consensus system-specific value accounts.

  Consensus allows to set these accounts values for any entity, as follows:

`: %[ expression ].$( account ): value`	assigns `value` to the `account` of the entities resulting from the query `expression`
`: %[ expression ].$( account ): %variable`	assigns `variable` value to the `account` of the entities resulting from the query `expression`

  where

  `account` can be either one of the following:

`narratives`	`value`[1] then must be a list of instantiated narratives
`url`	`value`[1] then must be an identifier enclosed in double-quotes (“)
`hcn`	`value`[1] then must be an identifier enclosed in double-quotes (“)

- The purpose and usage of these accounts and of their values are explained in the following sections of this document.

1. resp. `variable` value

Consensus HTML integration

- Consensus allows to write the HTML value of an entity to standard output, as follows:

  `>: %[ expression ].$( html )` writes the HTML value of the entities resulting from the `expression` query to standard output, based on their `hcn` value.

- Consensus HCN file format consists of the standard HTML file format with the following two additions:
  - It supports Consensus tags, which are sequences of Consensus commands separated by the newline character.

    Consensus tags are specified by enclosing them in the double-characters << and >>, the same way standard HTML tags are enclosed in the single characters < and >.

    Upon encountering a Consensus tag, the Consensus HCN parser will execute the enclosed Consensus commands. The variable `%%` in this case and outside any narrative-building operation represents the entity whose HTML output is being produced.

  - It ascribes a special meaning to the character percent (`%`) used outside of Consensus tags.

    Upon encountering the character percent (`%`) in the text outside a Consensus tag, the Consensus HCN parser will interpret the subsequent characters and write the output as per its standard output operations. The variable `%%` in this case represents the entity whose HTML value is being generated.

Consensus will output all other character sequences from the HCN file to the standard output exactly as it is.

Consensus narratives

- Consensus allows to build narratives and to assign them to entities via their `narratives` value account.

  Narratives are essentially programs in the traditional sense of the term, except that, from both the programming and execution standpoints, time can no longer be relied upon to ensure that the program's instructions execute one after the other. Narrative instructions cannot be assumed to execute linearly, one after the other, in sequence. No more - nor less - than this is the way things happen in reality. Here, instead of time, we have change. And change propagates not only in one but in all directions, from one entity to all connected others, first, within the system, and then outside - and then what goes around, as the saying goes, comes around...

- To allow Consensus narratives, which are programs, to handle these changes, Consensus introduces the notion of frame.

  A Consensus frame is a snapshot at the present instant of both the current state of the system (that is, of all the entities in their present state), and of all the changes that happened to it since its own last frame. It is based on this information, and on this information only, that each entity narrative determines what changes must be performed and affect what part of the system, from the entity's standpoint, up until next frame.

- This obviously requires serious consideration up-front of all the changes susceptible to affect the entity from others, and conversely of the effects of the entity's changes on all connected others. Which is fine, as this is precisely what programmers are supposed to do, up-front, so that their program has a chance to render any significant service when entering an information highway such as the Internet, or other.

  Which is also the reason why this method is called Consensus.

Contents

Example 1.
Command-line interface

Consensus command-line interface

```
Patricks-MacBook:$ ./consensus
consensus:
consensus: !! Patrick-has->child
consensus: !! Jules-is->[ child<-has-Patrick ]
consensus: !! Charles-is->[ child<-has-Patrick ]
consensus:
consensus: ?: ?-is->[ child<-has-Patrick ]
consensus:           >:%? is Patrick's son
consensus:           /
Charles is Patrick's son
Jules is Patrick's son
consensus:
consensus: ?: [ ?-has->child ]<-is-Charles
consensus:           >:%? is Charles's father
consensus:           /
Patrick is Charles's father
consensus:
consensus: ?: ?-has->child
consensus:      ?: ?-is->[ child<-has-%? ]
consensus:           >:%?
consensus:           /
consensus:      /
Charles
Jules
consensus: _
```

Example 2.
HTML Integration

http://localhost/consensus.cgi

http://localhost/consensus.cgi?mode=edit

```
<html>
<body>
<h1 style="text-align:center; font-weight:normal;">Welcome to Consensus!</h1>
<<? %[ ?-is->Label ] >>
      <ul>
      <<? : ?-is->Label >>
      <li> <a href="consensus.cgi?label=%?">%?</a>
      <<//>>
      </ul>
<<//~>>
      No label yet.
<<//>>

<form action="consensus.cgi" method="get">
Create a new label: <input type="text" name="new_label">
<input type="submit" value="Send">
</form>
```

View | Edit

http://localhost/consensus.cgi?label=people

People

- Patrick
- Pierre
- Paul
- Jacques

Create a new entry: _ Submit

View | Edit

http://localhost/consensus.cgi?label=People&mode=edit

```
<html>
<body>
<h1 style="text-align:center; font-weight:normal;">%label</h1>
<< ? %: ?-is->[ %label-is->Label ] >>
<ul>
        << ? : ?-is->[ %label-is->Label ] >>
        <li> <a href="Consensus.cgi?label=%label&entry=%?>%?</a>
        << / >>
</ul>
<<~>>
        No entry yet.
<<>>

<form action="consensus.cgi" method="get">
Create a new entry: <input type="text" name="new_entry">
<input type="hidden" name="label" value="%label">
<input type="submit" value="Send">
</form>
```

View | Edit

http://localhost/consensus.cgi?label=People&entry=Patrick

Patrick

HAS

- Projects
  - Consensus
- Milestones
  - EOY/2016

BELONGS TO

- Teams
  - Consensus

Create a new relationship: _ Submit

View | Edit

http://localhost/consensus.cgi?label=people&entry=Patrick&mode=edit

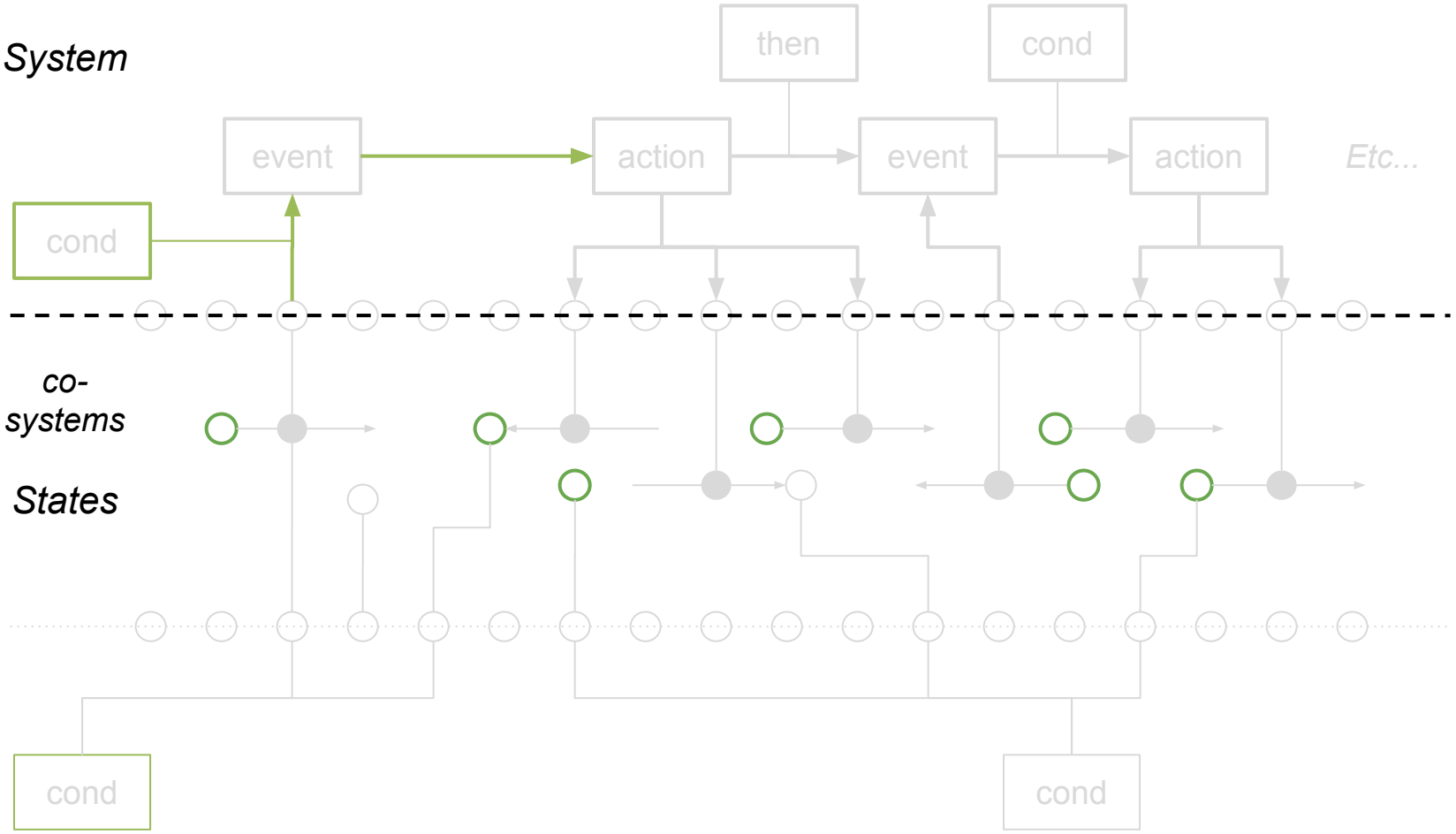
```
<h1 style="text-align:center; font-weight:normal;">%entry</h1>
<b>HAS</b>
<< ? %: [ .-is->Label ]<-has-%% >>
      <ul>
      << ? : [ .-is->Label ]<-has-%% >>
            <li> <a href= … >%[ [ ?-is->Label ]<-has-. : %? ]</a>
            <ul>
            << ? : [ .-is->[ .-is->Label ] ]-is->%? >>
                  <li> <a href = … >%[ [ ?-is->[ .-is->Label ] ]-is->. : %? ]</a>
            << / >>
            </ul>
      << / >>
      </ul>
<</~>>
        Nothing yet.
<</>>

<b>BELONGS TO</b>
```

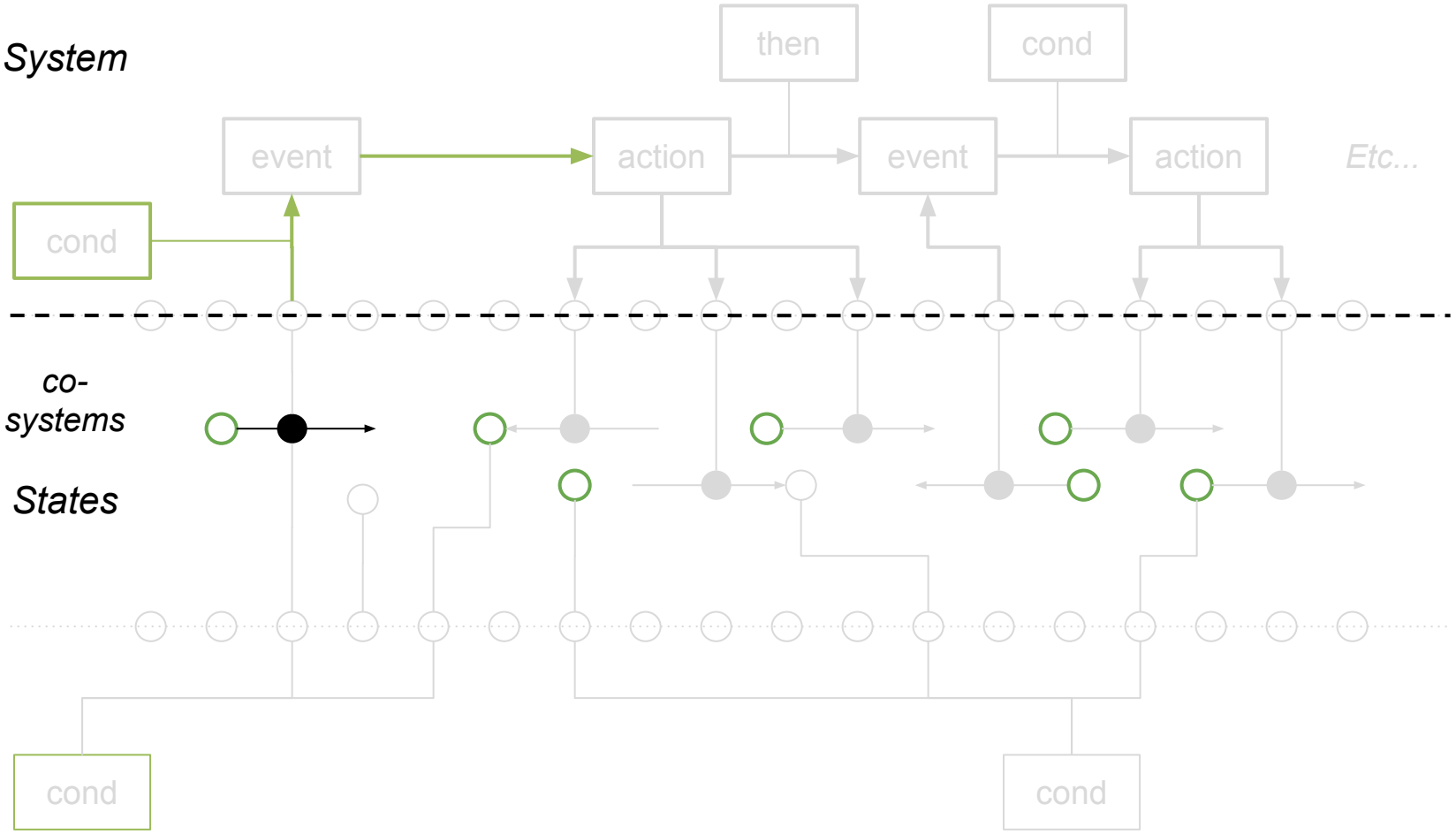
View | Edit

Example 3.
Narrative frame-by-frame simulation

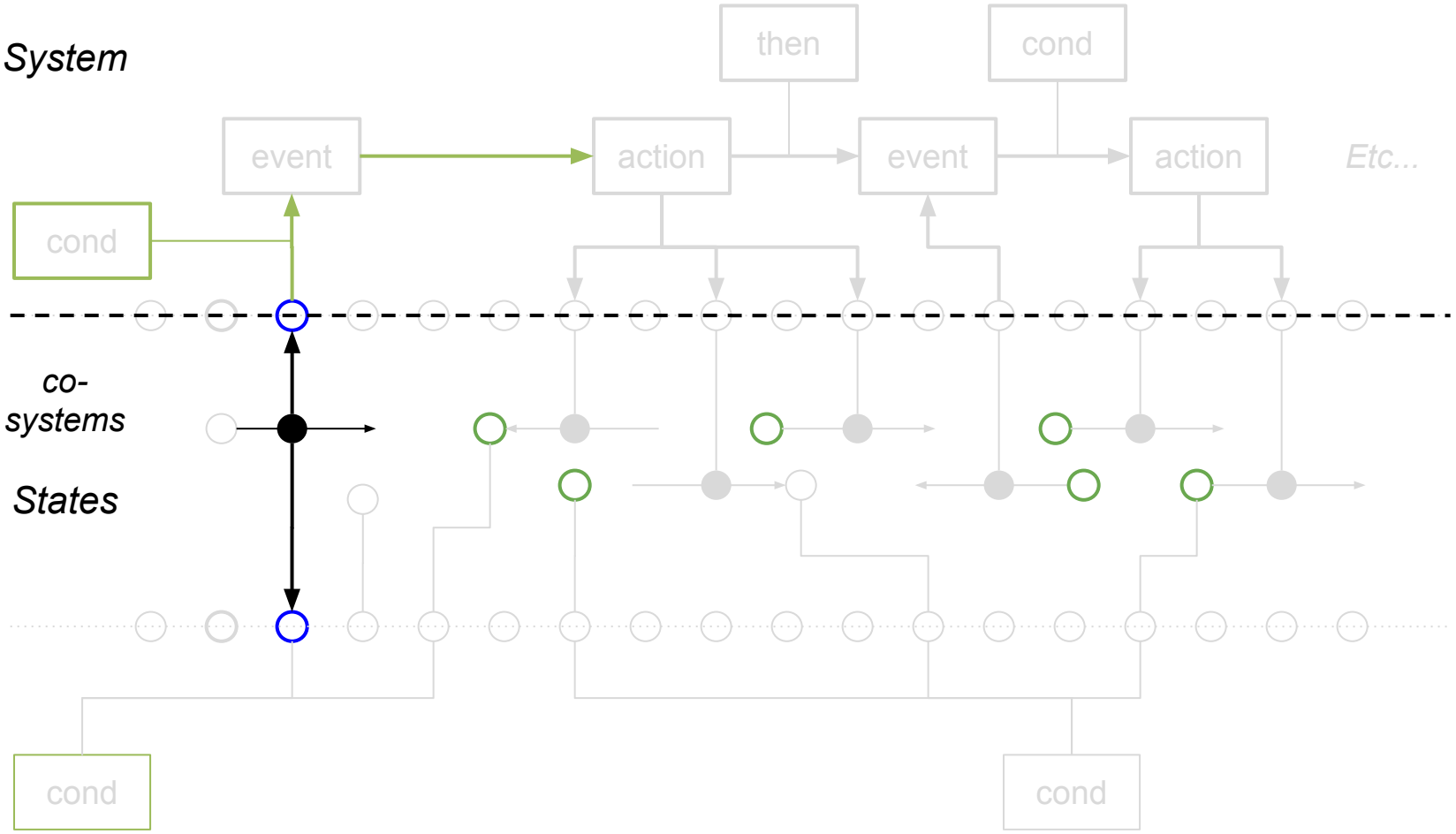
Initial State



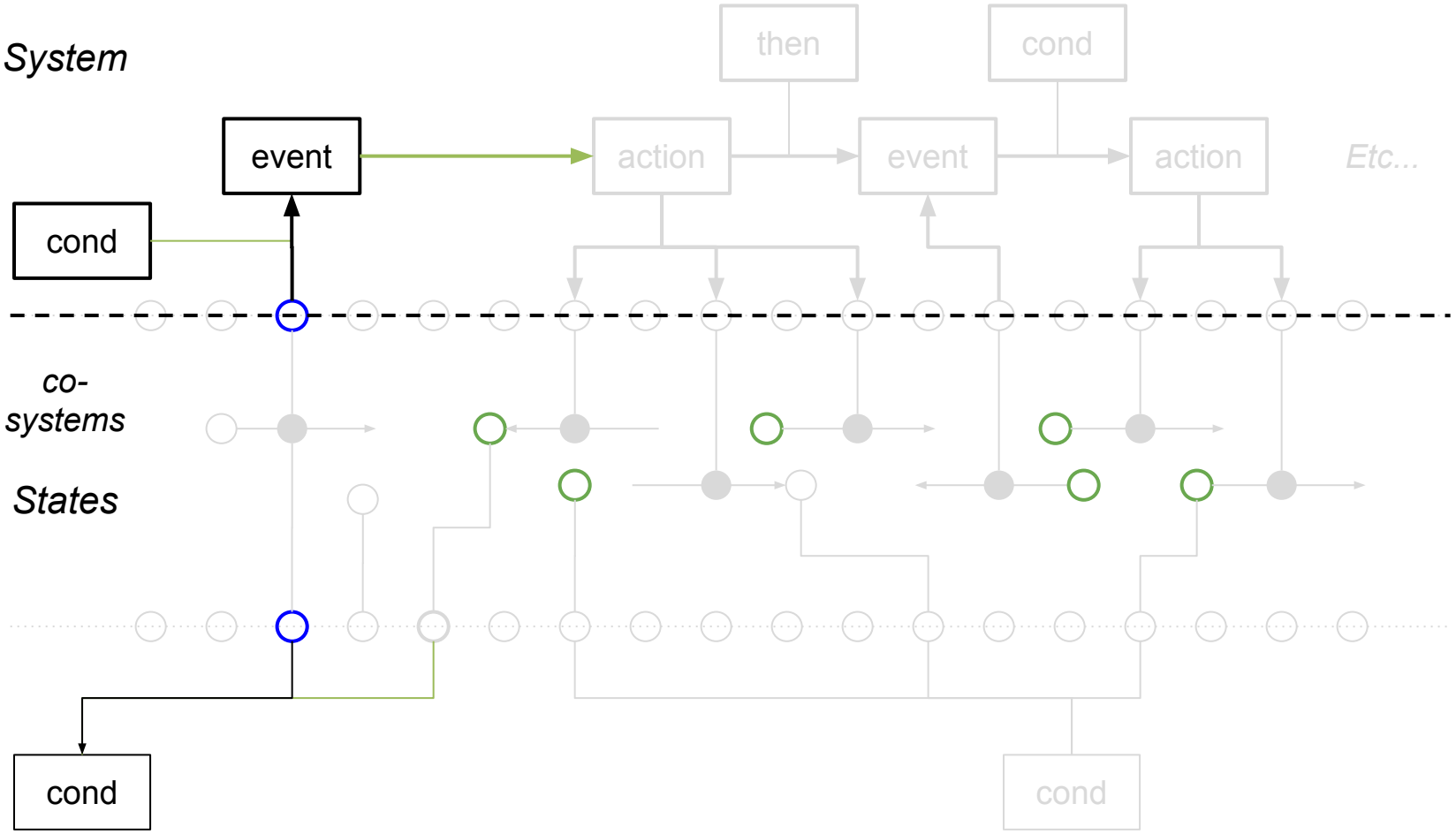
Change happens



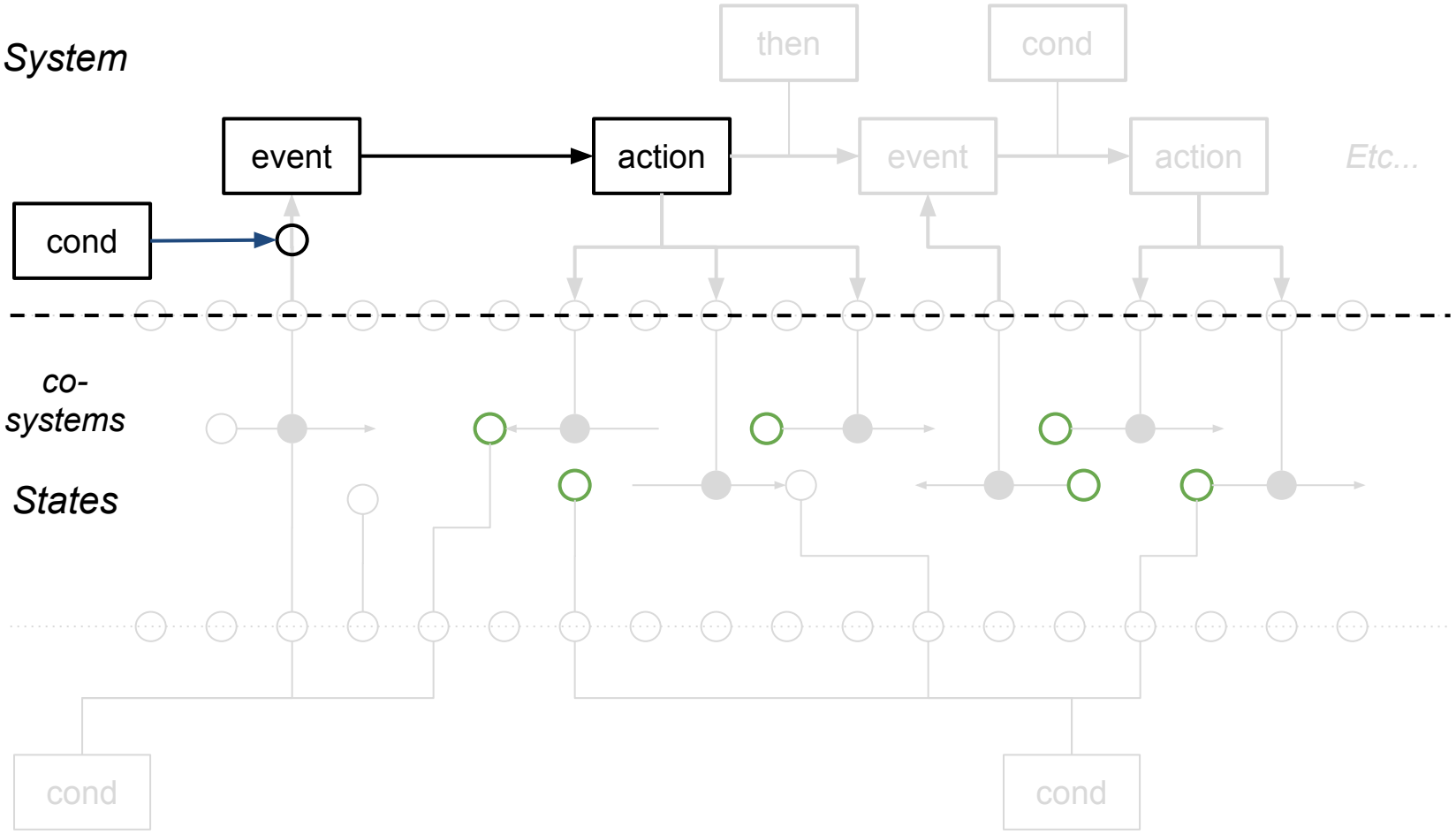
Change Propagates



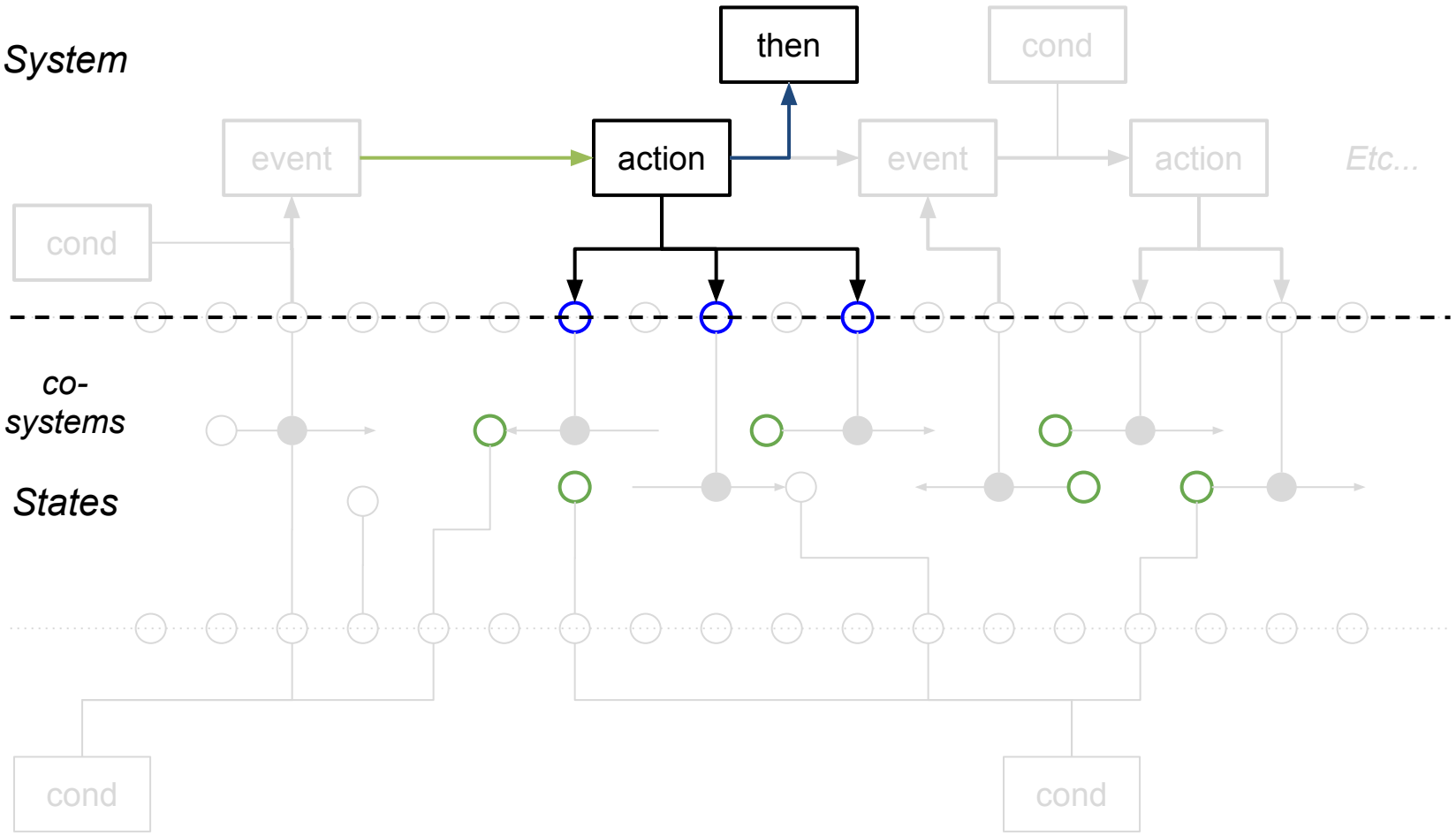
Condition and Event are notified



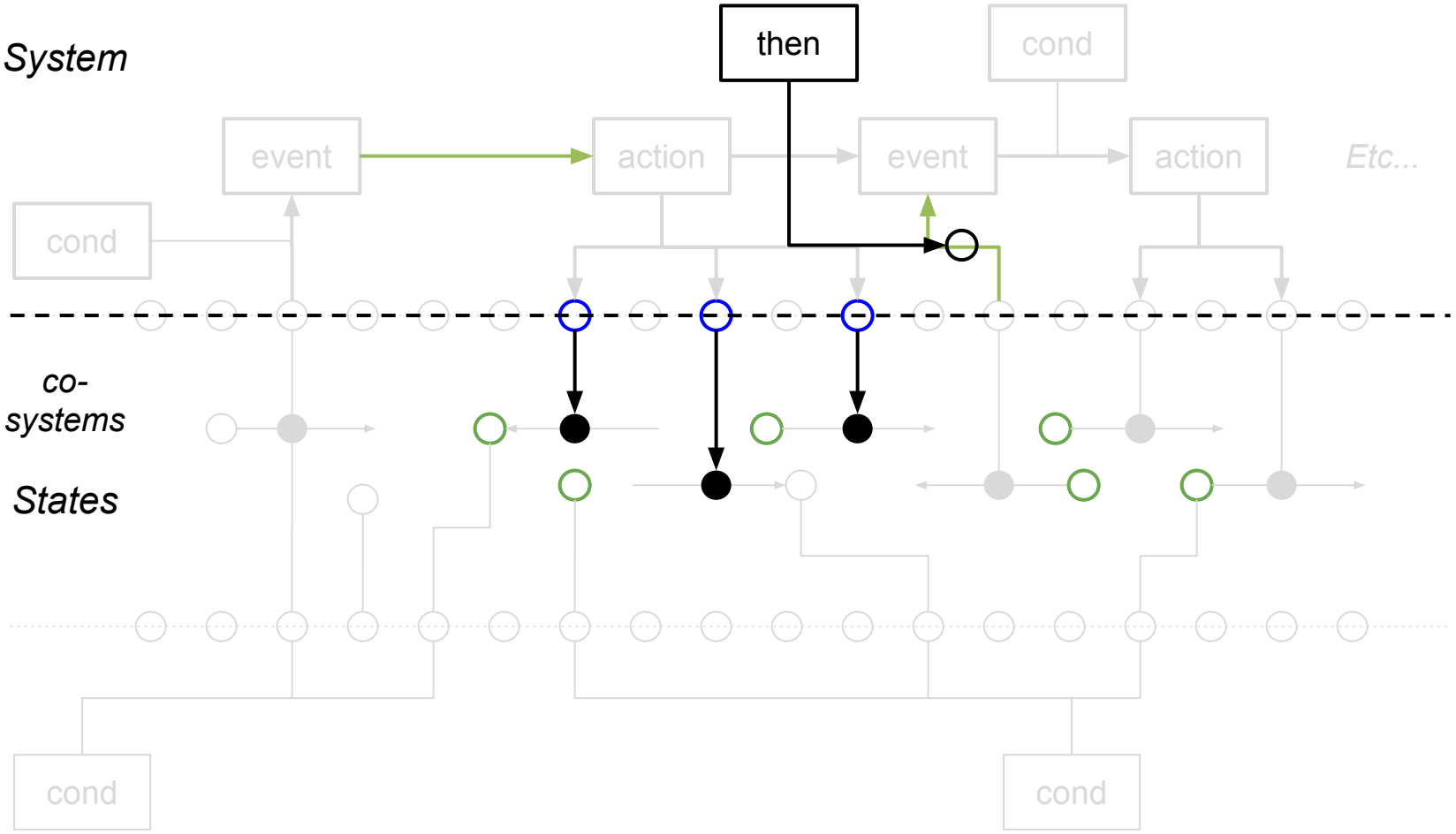
Event triggers Action, Condition applies



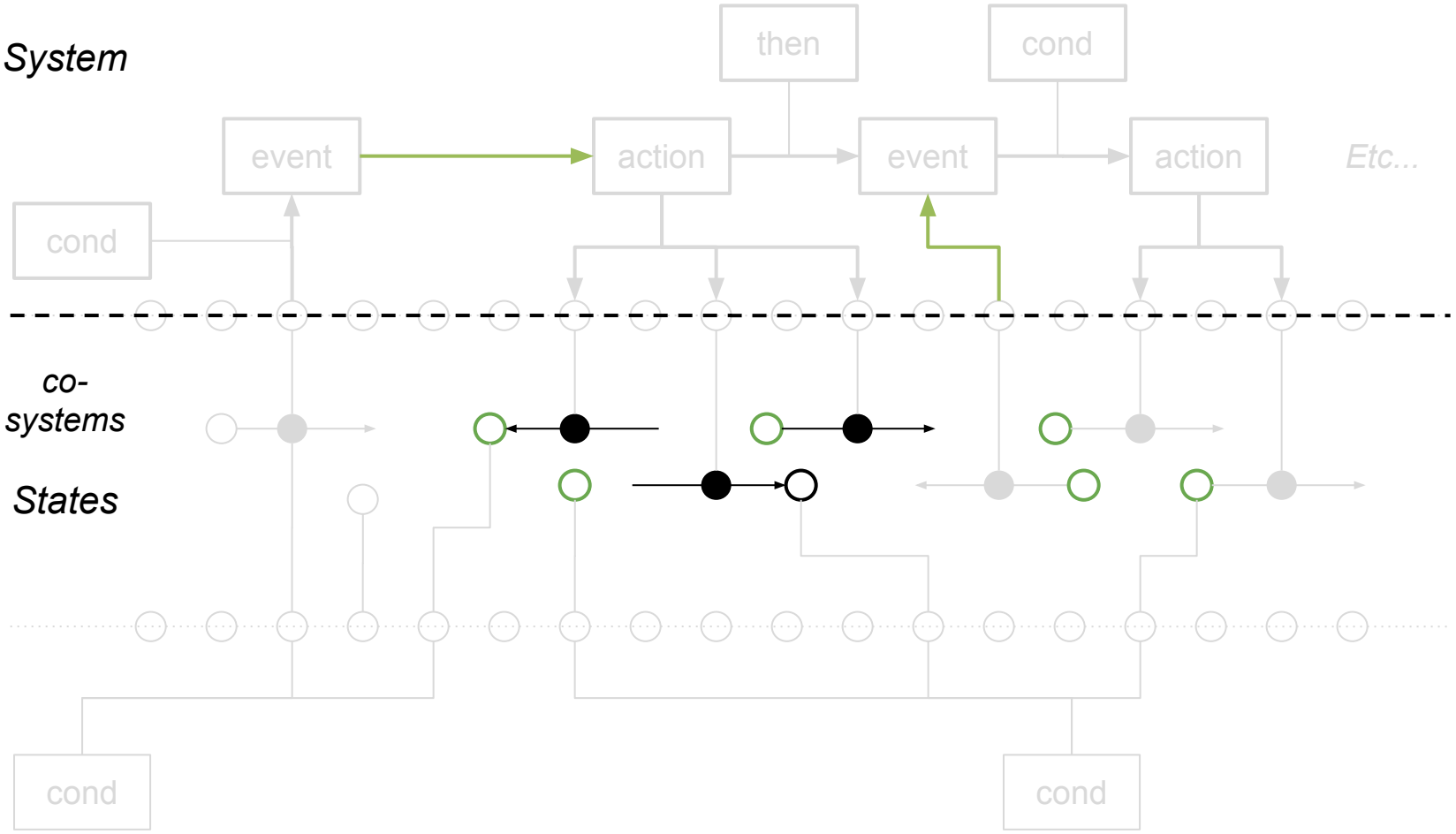
Action executes



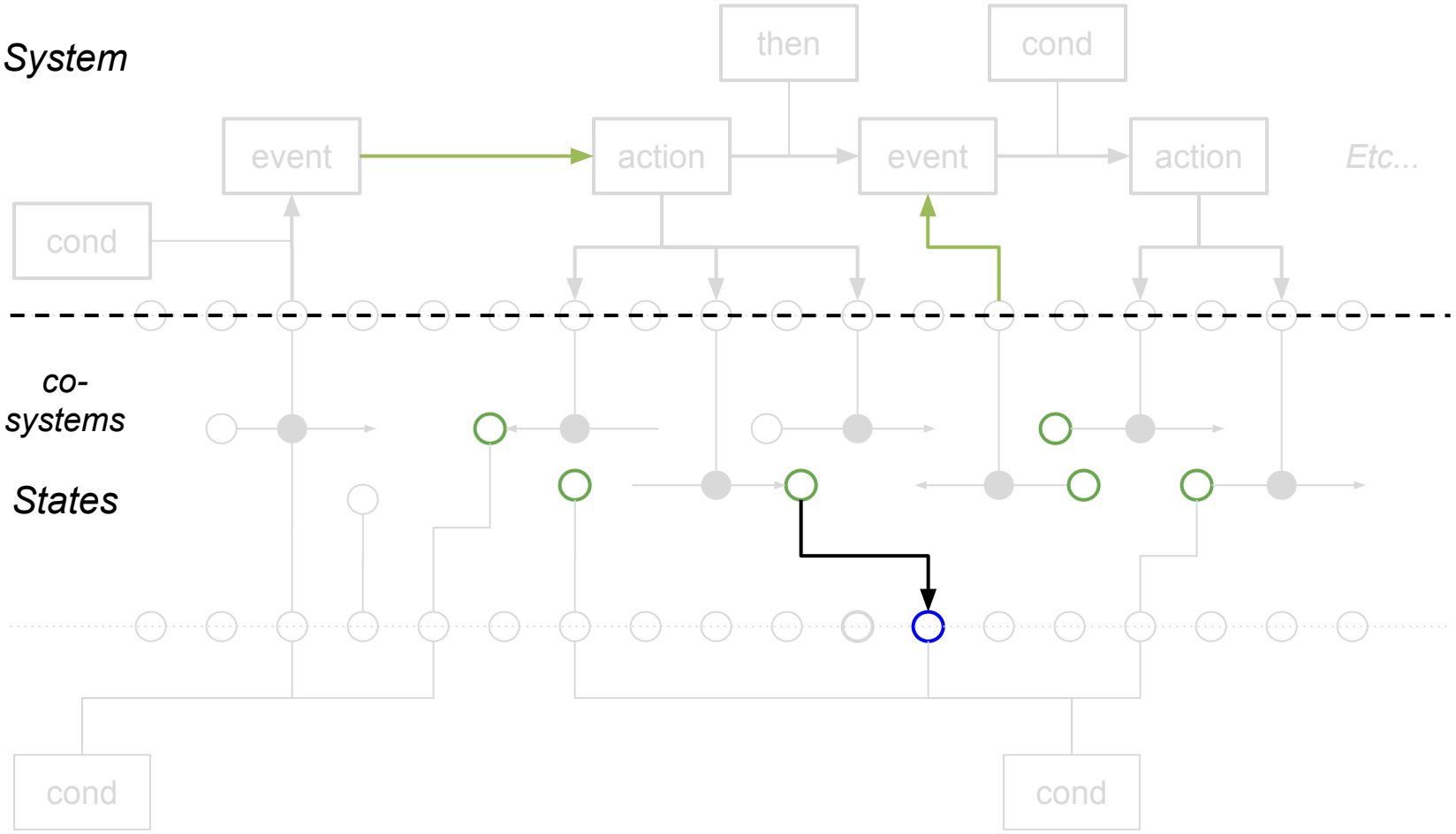
Changes are triggered, Condition applies



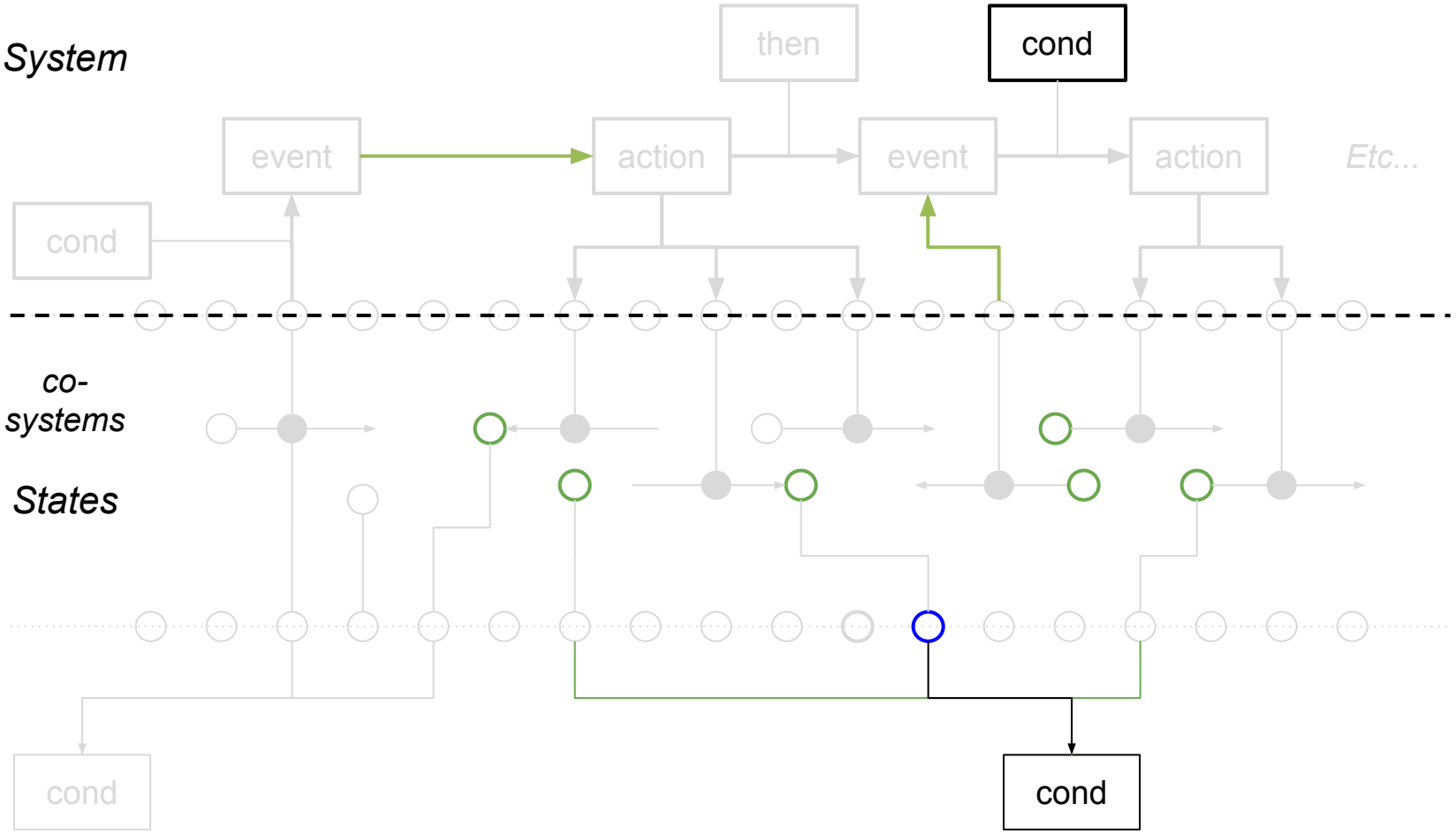
Changes happen



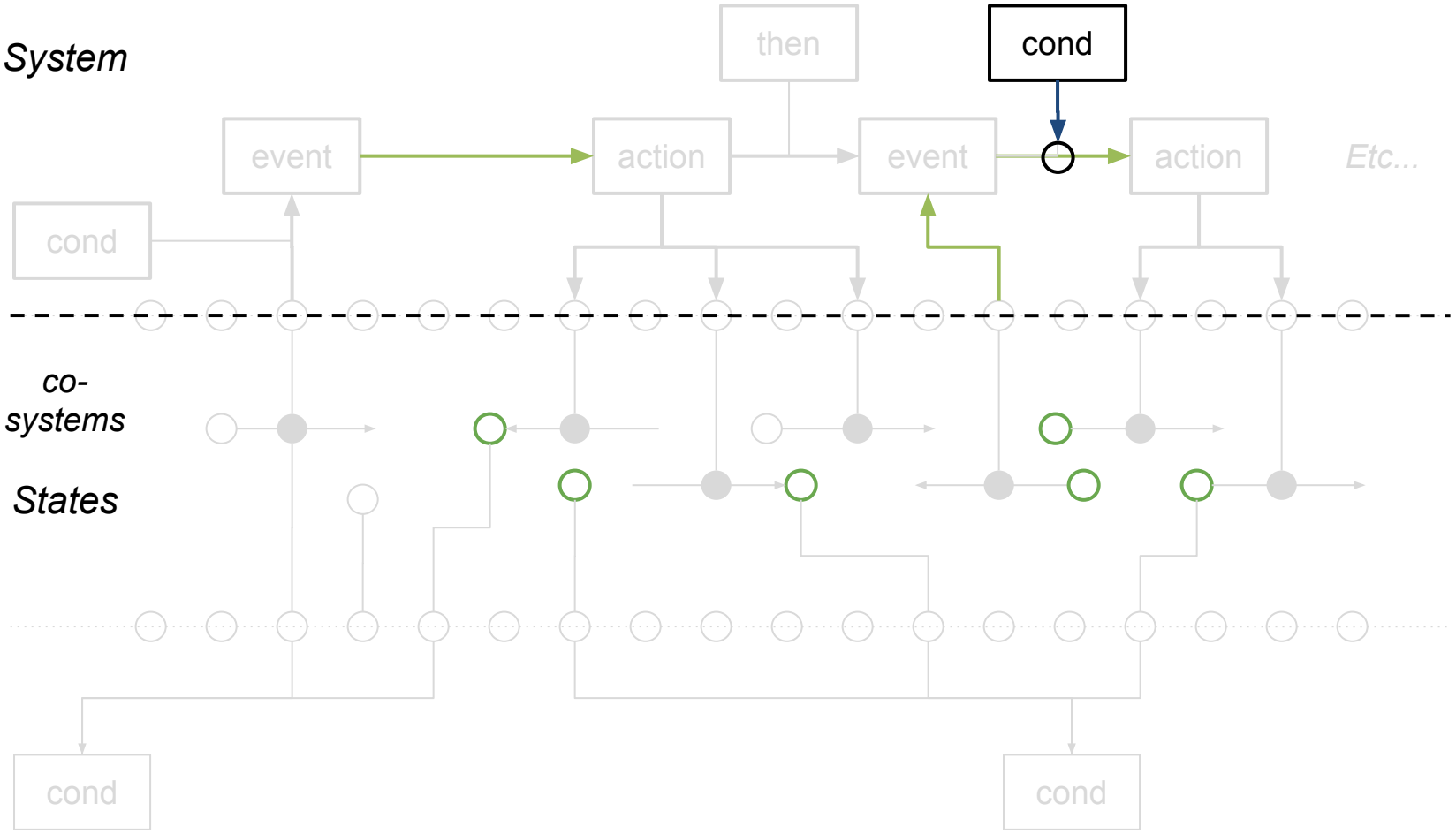
Change propagates



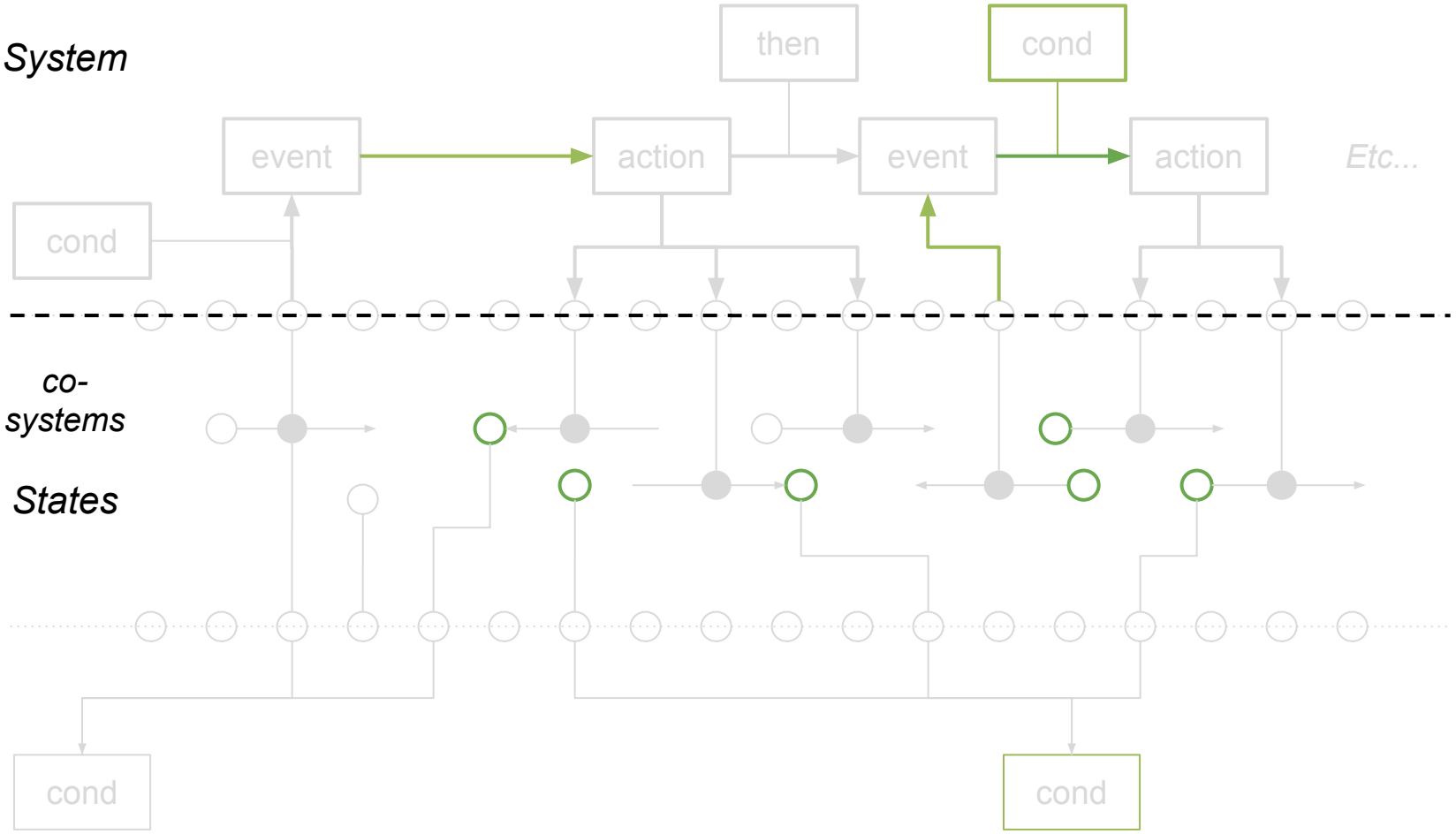
Condition is notified



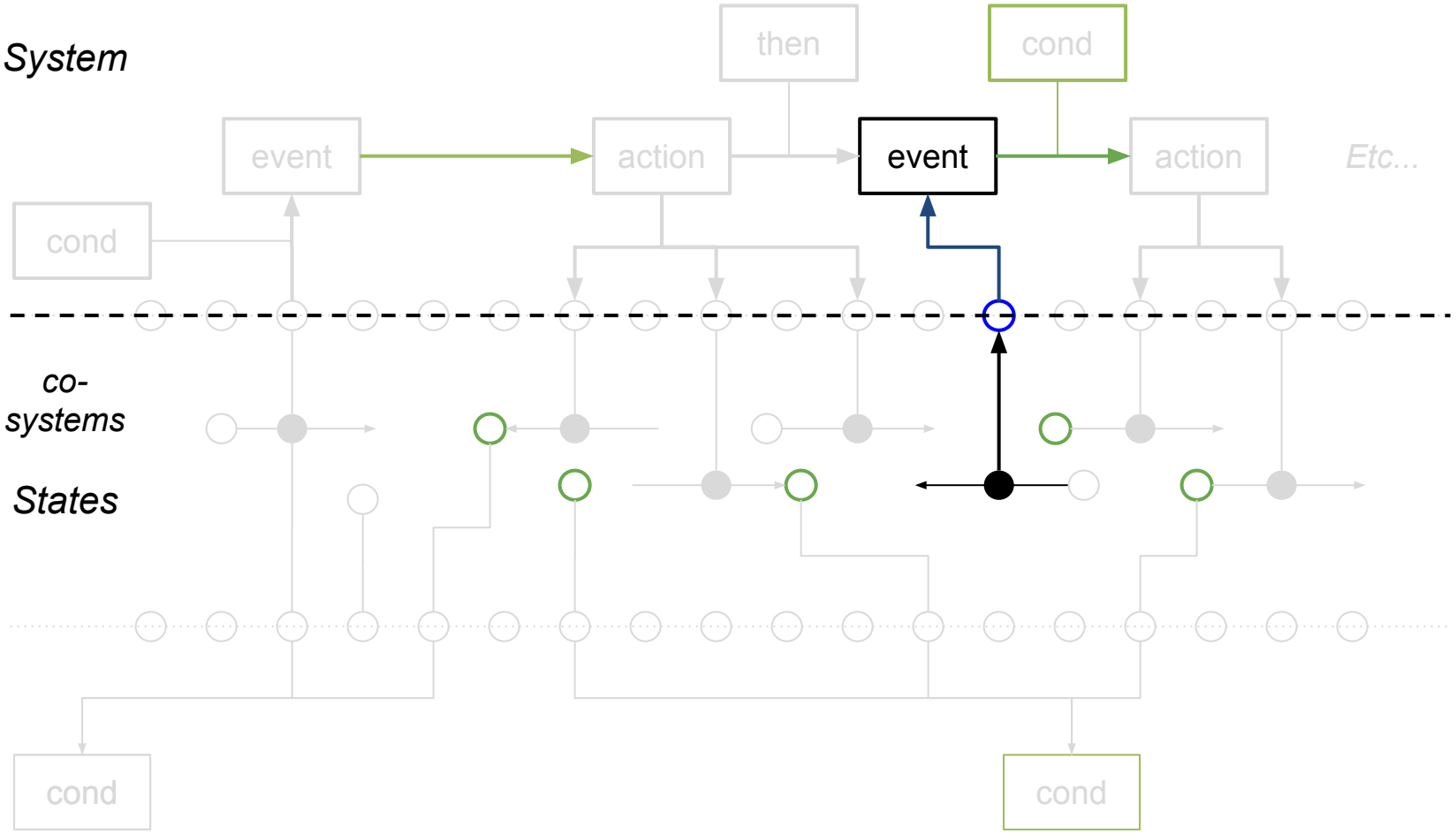
Condition applies



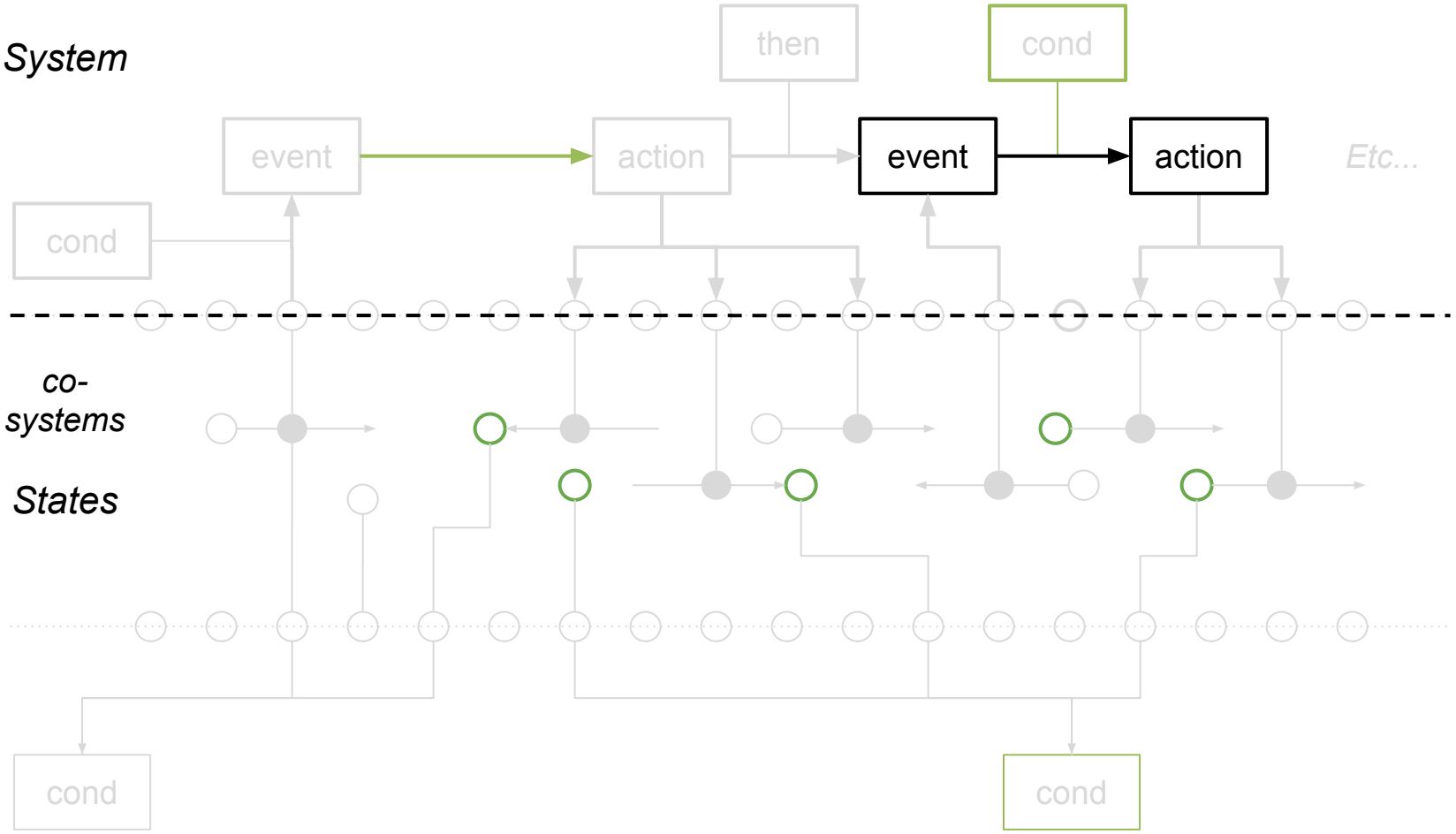
New System State



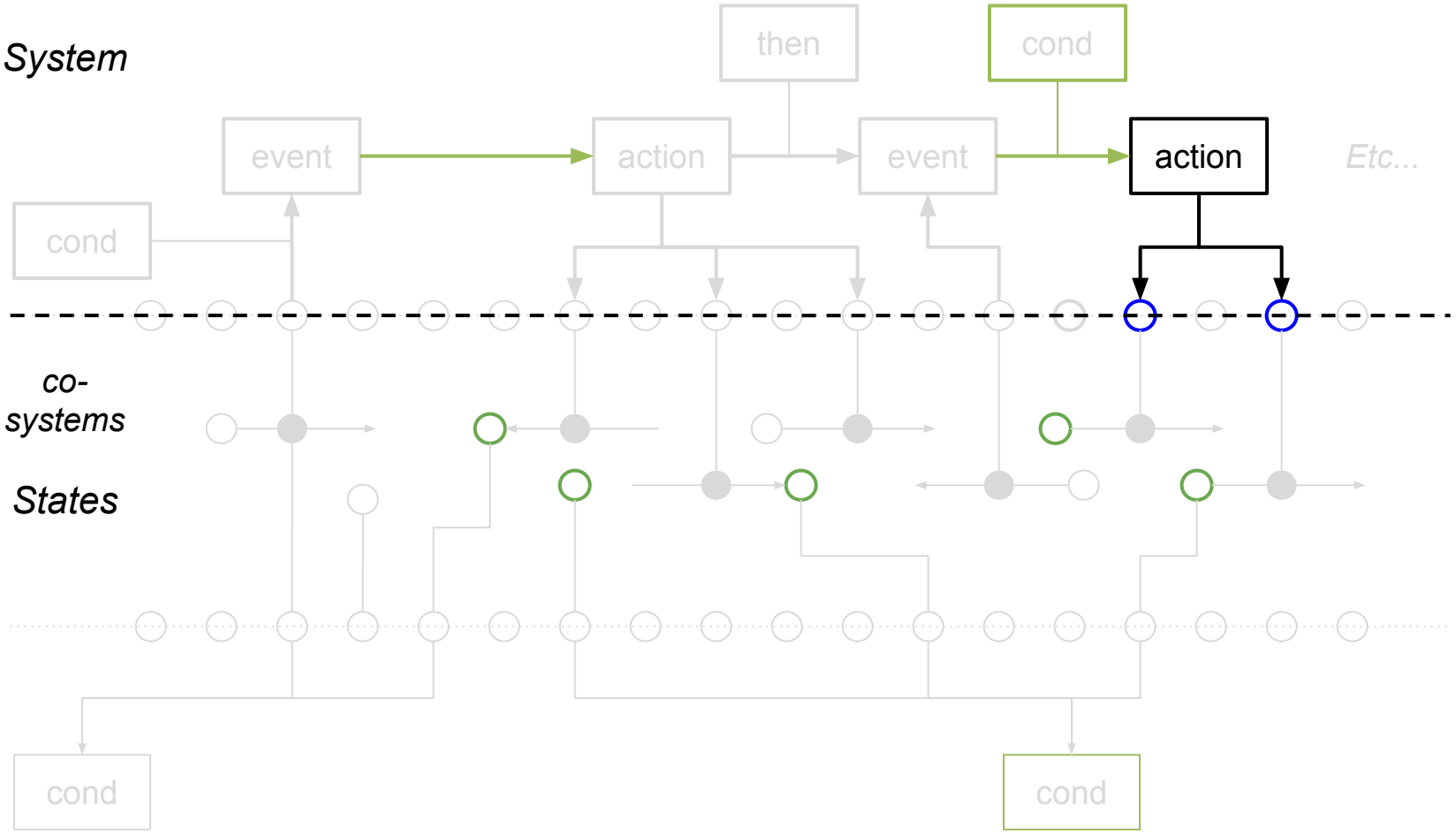
Event is notified



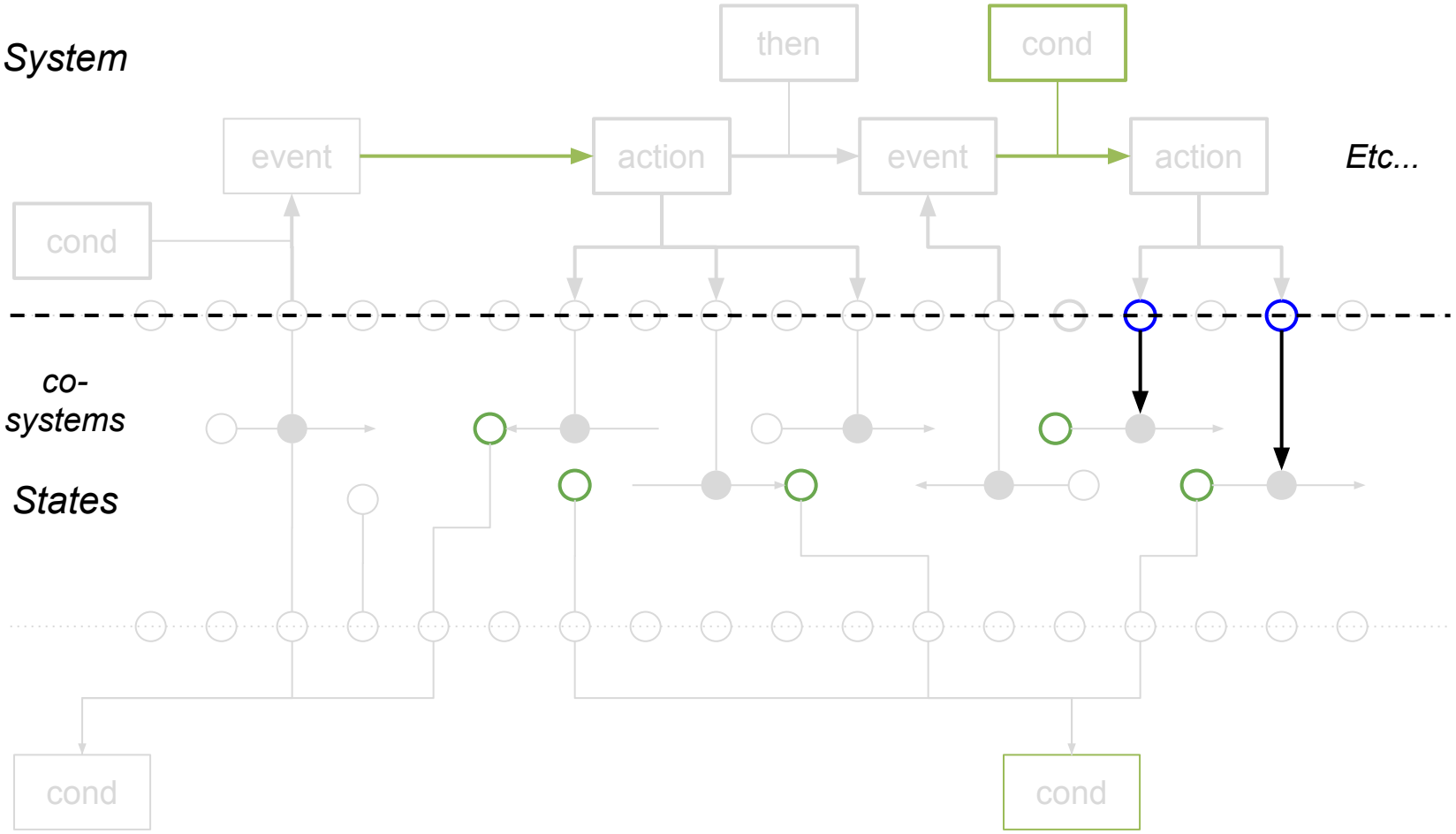
Event triggers Action



Action executes



Etc.


System
then
cond
event
action
event
action
Etc...
cond
co-
systems
States
cond
cond

Thank You!