Seminar on Complex-Event-Processing (CEP) in Distributed Systems

Complex-Event-Processing (CPE)

- -The big Why of bad things happening in IT-Systems
- -Detecting patterns of events
- -Defining and using causal relations between events
- -Aggregating events to higher-level events
- -Layered architectures and CPE
- -CPE languages and pattern matching

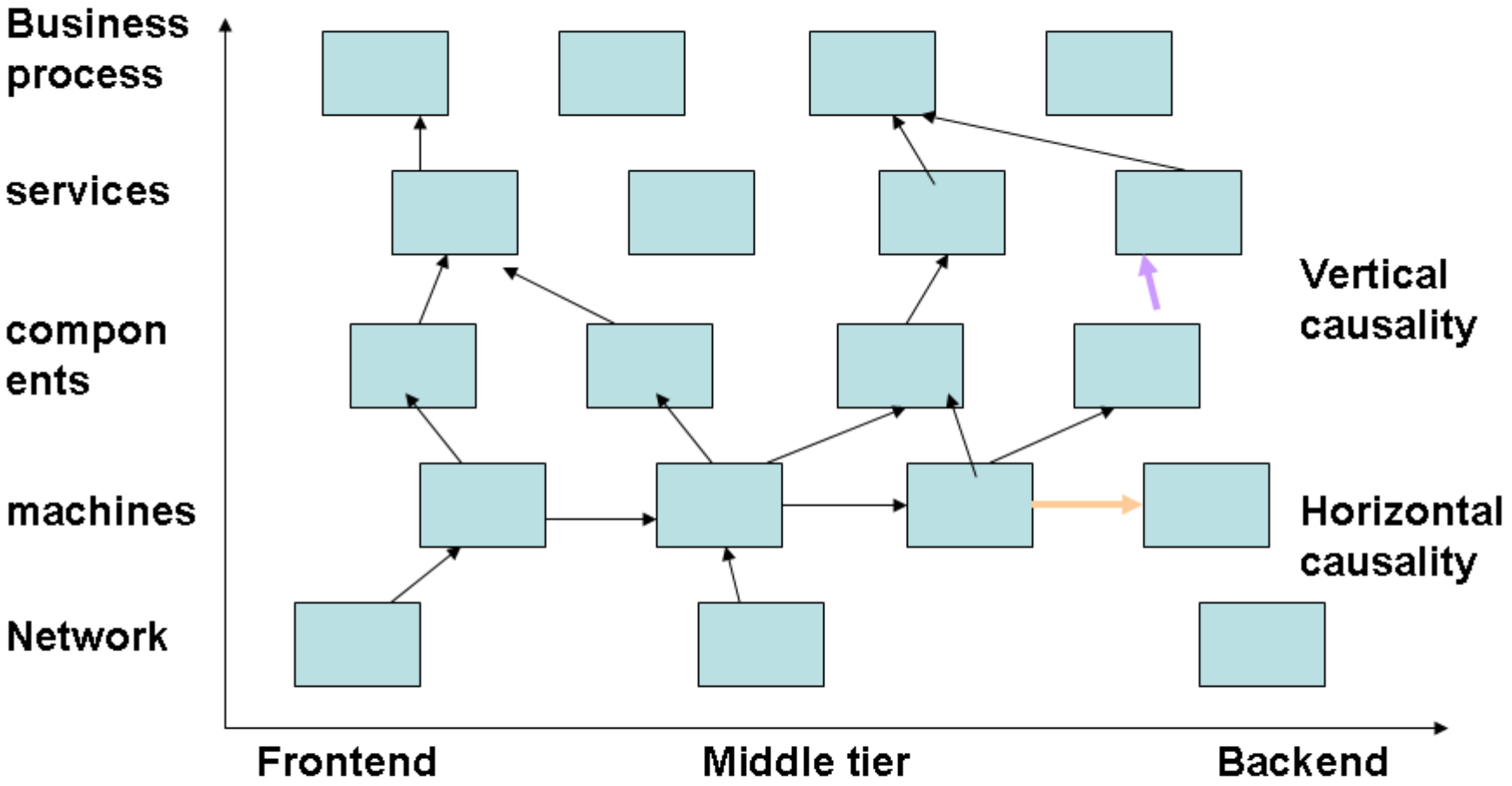
The following slides are mostly based on „complex event processing“ by Luckham and „event-driven Systems“ by Mühl et.al.

Why CEP? Case 1: Logfile Analysis of an Installation problem

1. Installation started
2. Service X: could not start
3. No license server detected
4. Component Y: f not found
5. Server Z up and running
6. Service H reports size problem
7. Service G running
8. Component U deactivated
9. Component I OK

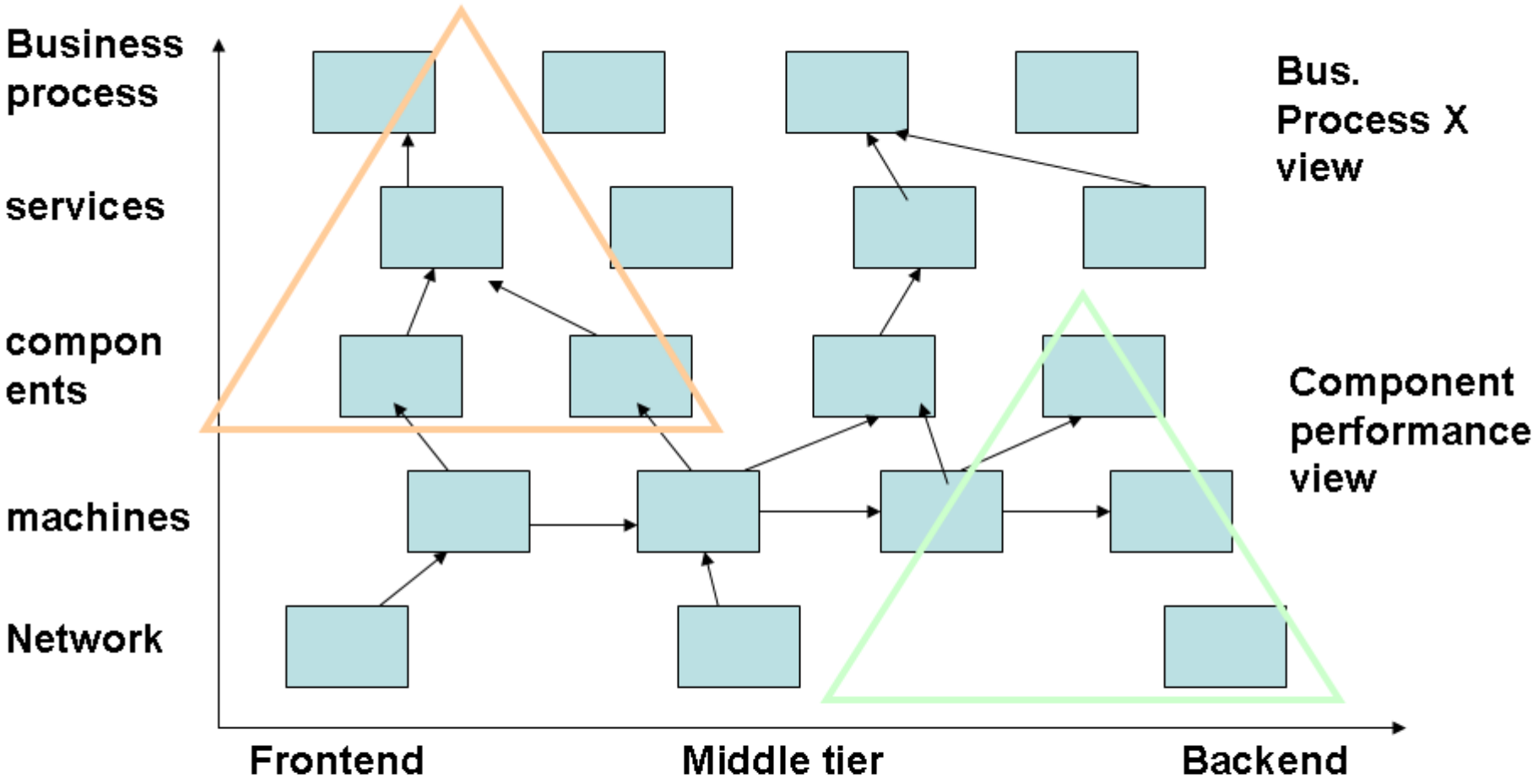
The complex event is that the newly installed system does not work. A look at the log file shows several events and comments. Some parts of the system seem to be running, some seem to be non-functional. The events shown in the log probably have some causal relationship. But we don‘t know it. It has to be recovered manually. We would like to know what caused the problems.

Case 2: End-to-End Performance and Failure Reporting and Analysis



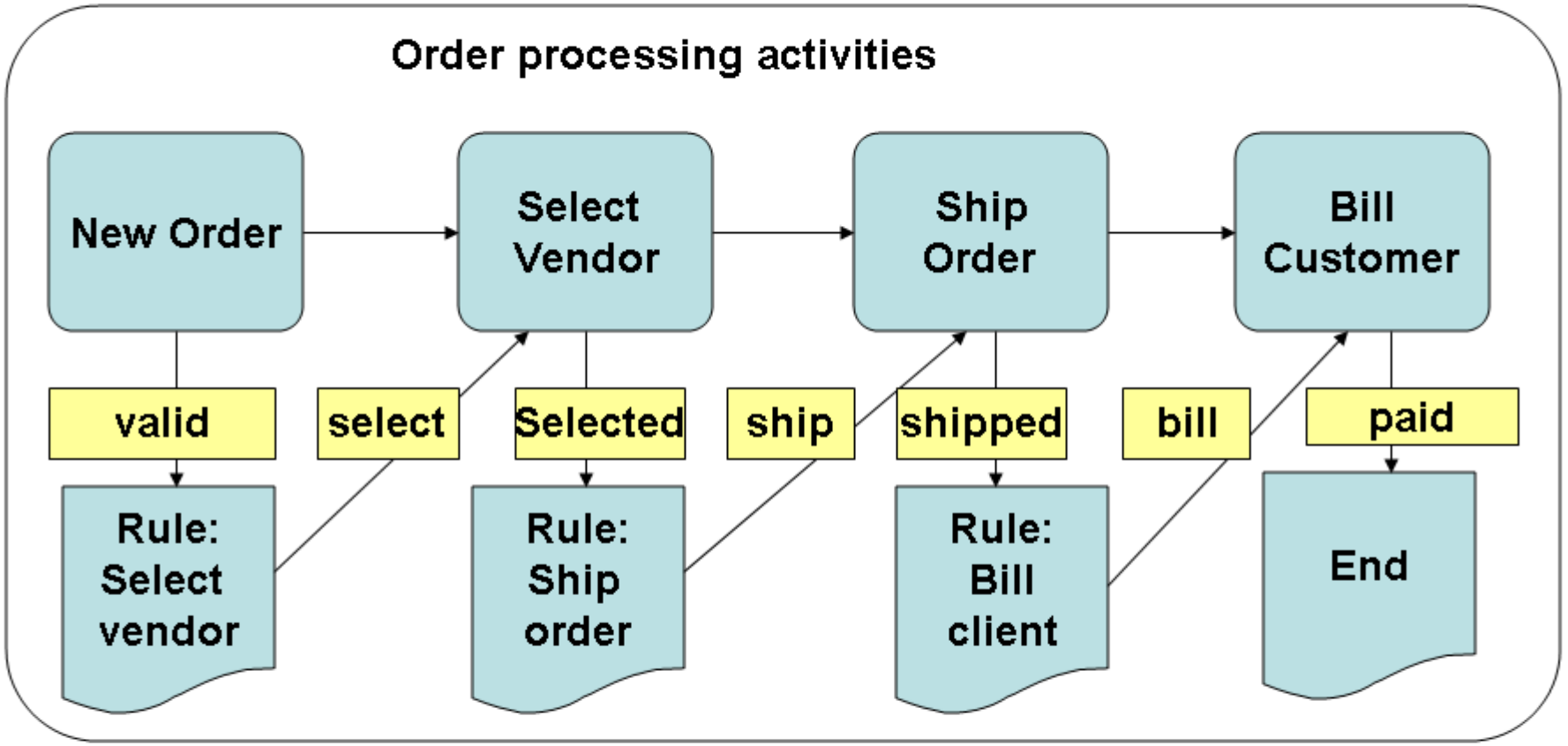
Events of different semantic level travel through layered architectures. Events at the same level express vertical causality, events between different levels express vertical causality. We want to trace the causality relation in both directions: what does an event cause? What has caused a certain event?

Hierarchical Views (looks like topic maps)



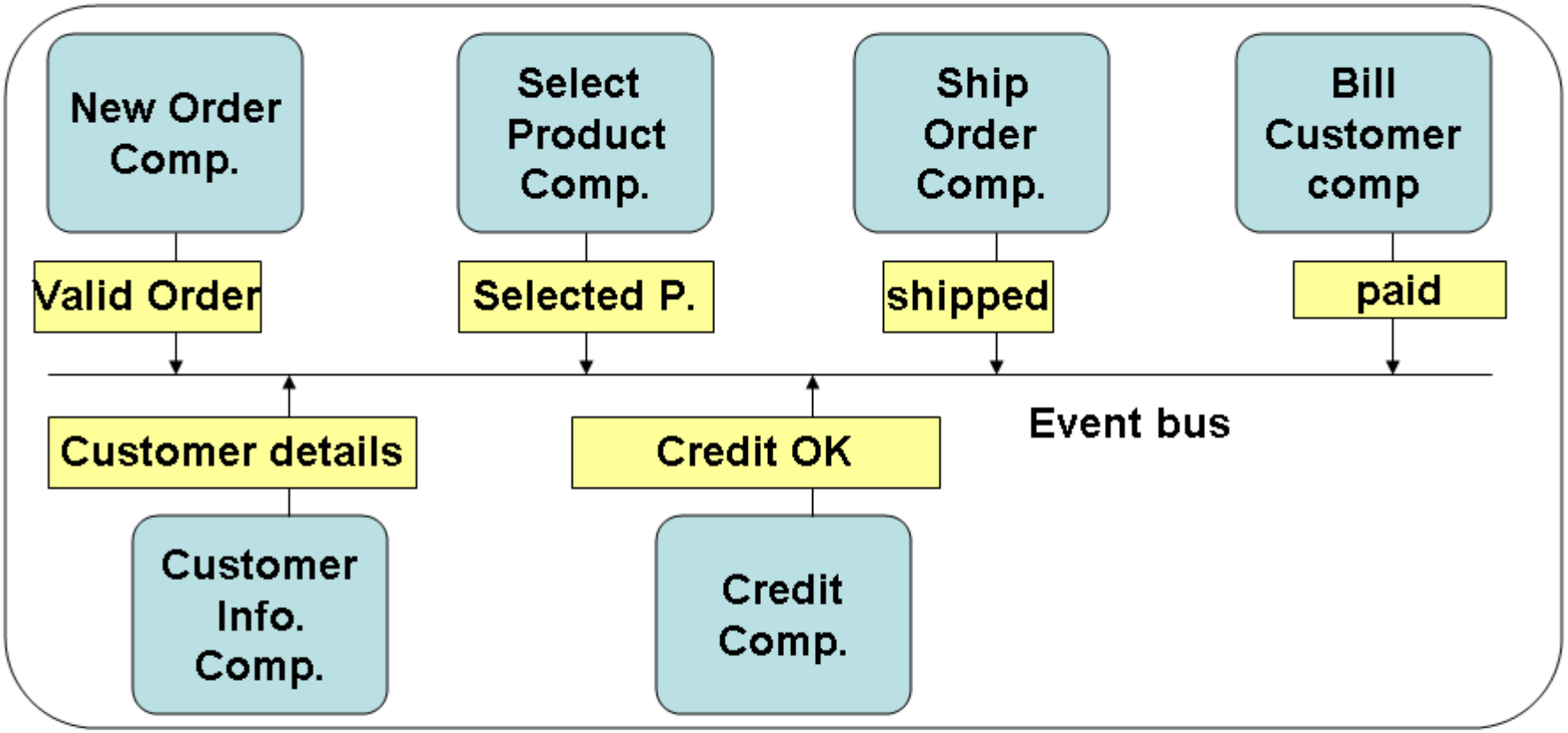
A hierarchical event view filters and categorizes events under a certain perspective. Those views can be created at all levels. Vertical navigation within a view is required to work in both ways. This implies that the dependencies between events are known (see causality later). Luckham pg. 57ff.

Sequential Busiiness Process Design



A typical sequence of activities using an imperative style: do this, do that. Steps are wired together. The relation between activity and business rule is not really clear. Are explicit command like triggers really needed (select, ship, bill)?

Event-Driven Business Process Design



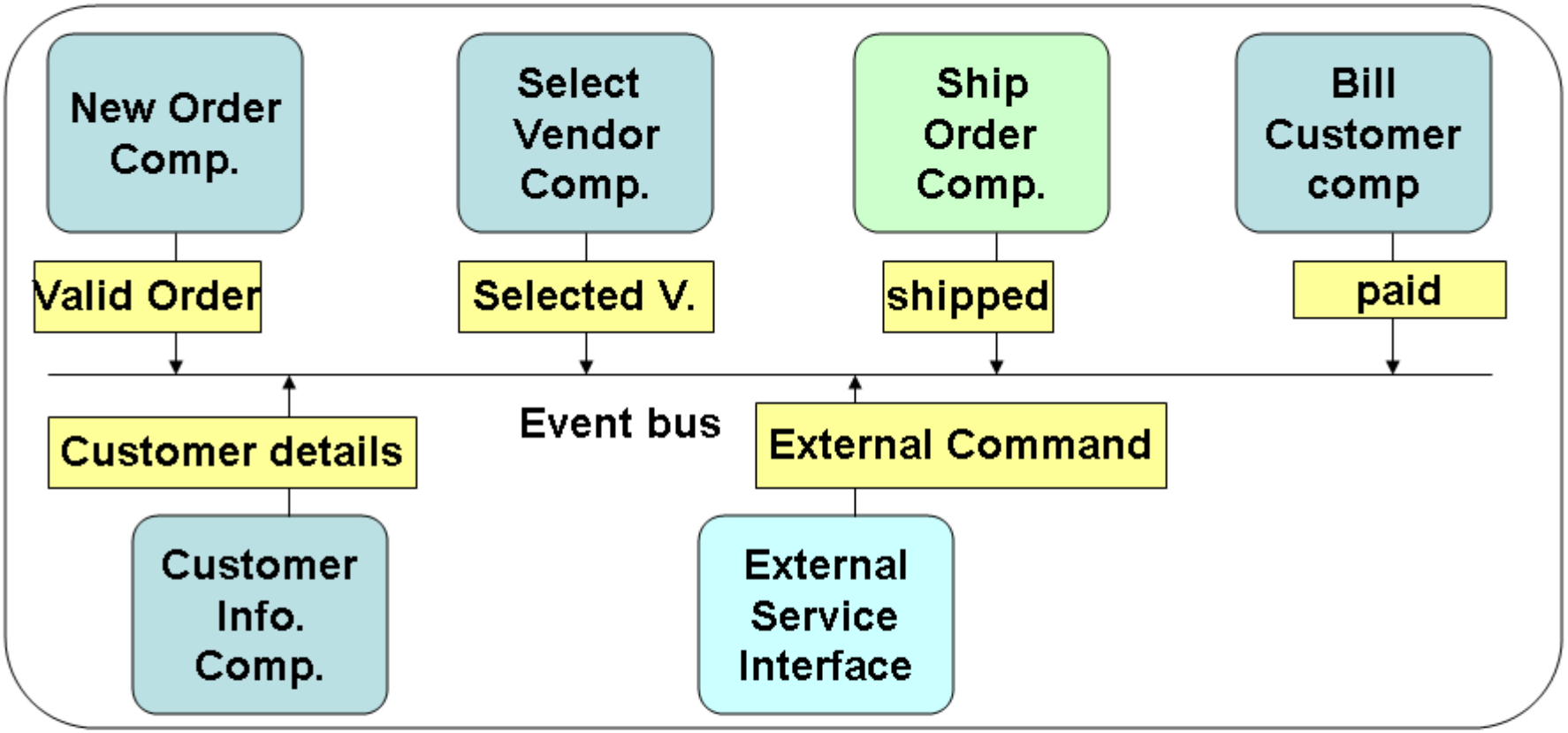
A true event-based process hides the rules within components. All actions are triggered by result-type events. Components listen for events that trigger their behavior.

Event-Driven Business Process Design

1. A new order has been validated and is posted in the system
2. A customer information component posts customer detail information including shipping address
3. A customer credit check component posts an event which states the credit rating for this customer
4. A product selection component checks type and availability of the product ordered
5. A shipping component ships the product and posts arrival at the customer
6. A billing component sends a bill and waits for payment. Then it posts that the customer paid the bill.

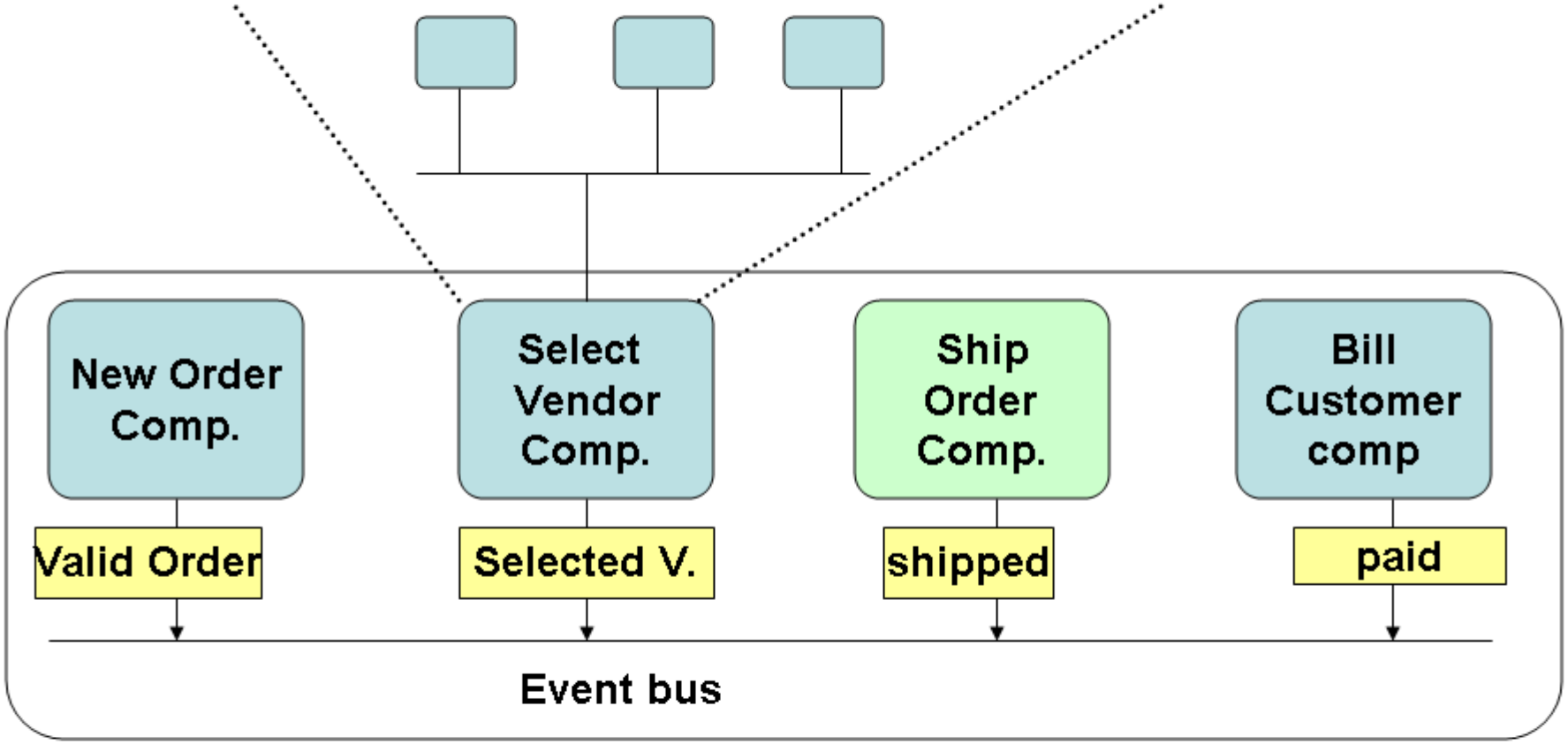
Much of the information posting is done in an asynchronous fashion in parallel and independent of other processing in the system.

Process including external service components



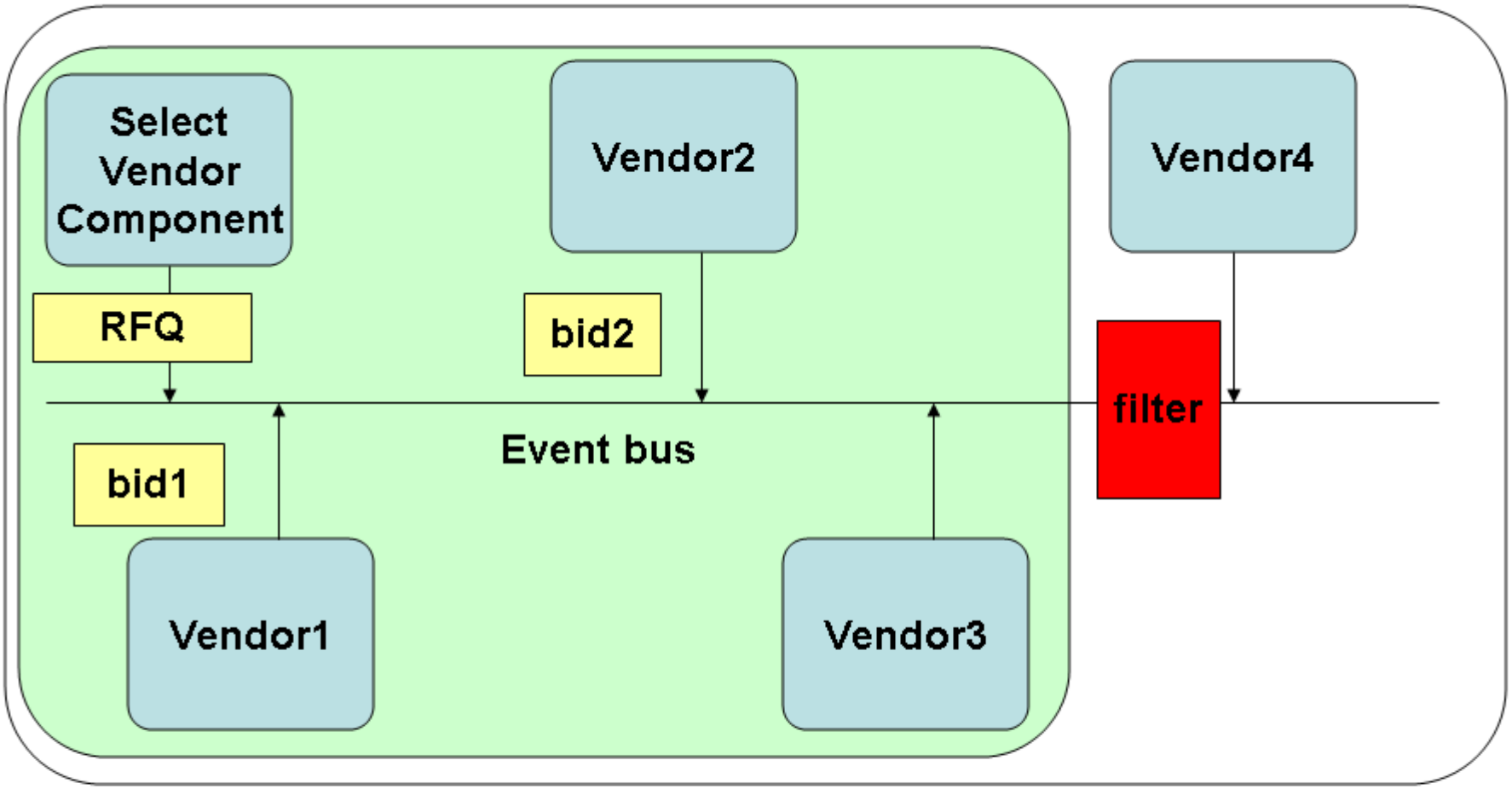
The external service interface is a technical artifact that would not be needed in a global event bus.

Asynchronous Subprocesses



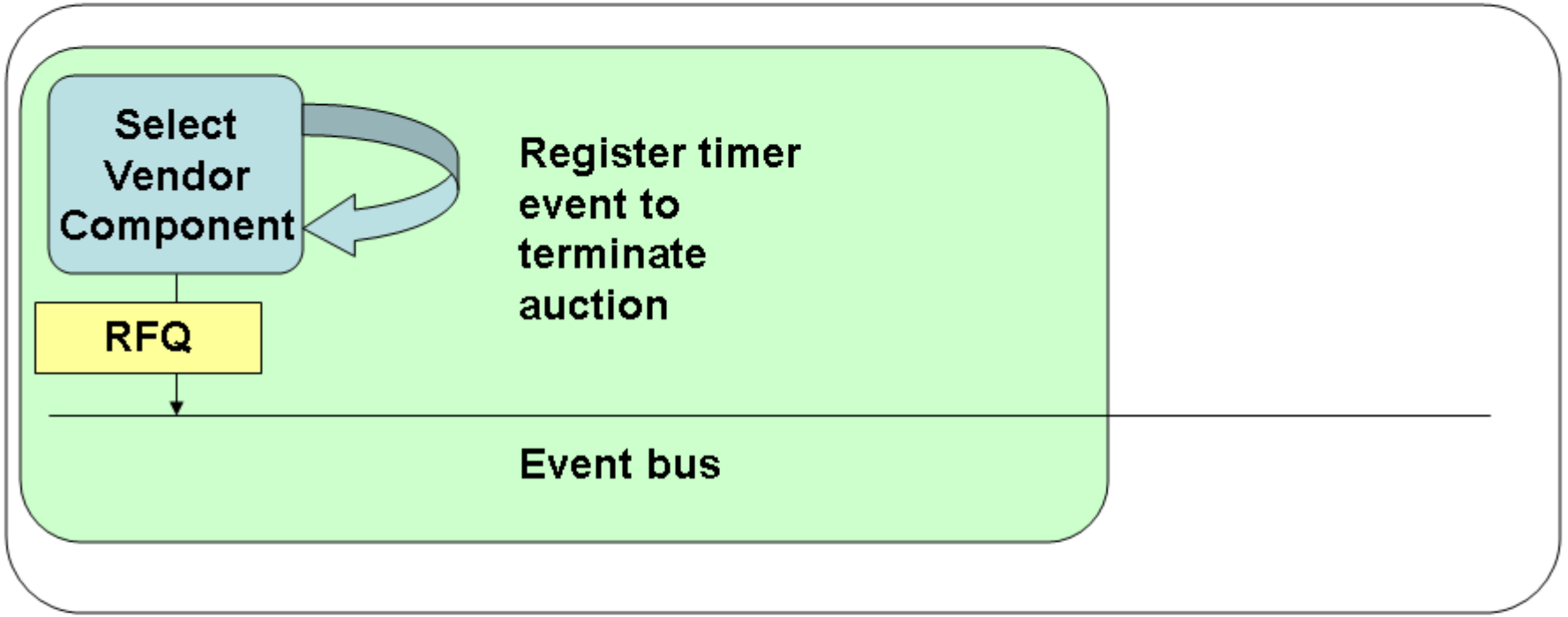
Components can internally use asynchronous events e.g. to run an auction for a bid. Sometimes the front component is called synchronously and works internally in an asynchronous way – the so called half-sync half-async pattern (Starbucks).

Select Vendor Subprocess with scope and filters



Only vendors with a certain on-time-delivery history should be selected. This can be achieved through scopes: an administrative component creates a controlled subsystem of the event bus where only selected partners can send and receive events. This can be done through the installation of filters. Vendor3 is in scope but was not able to bid. Vendor 4 did not see the RFQ.

Timing Process steps



Time limits for process steps can easily be created by registering timer events. Many systems model time based effects this way (see VRML timers for 3D effects etc.)

The Business Process Abstraction

- Business Planning frequently uses the process abstraction to describe business operations and goals.

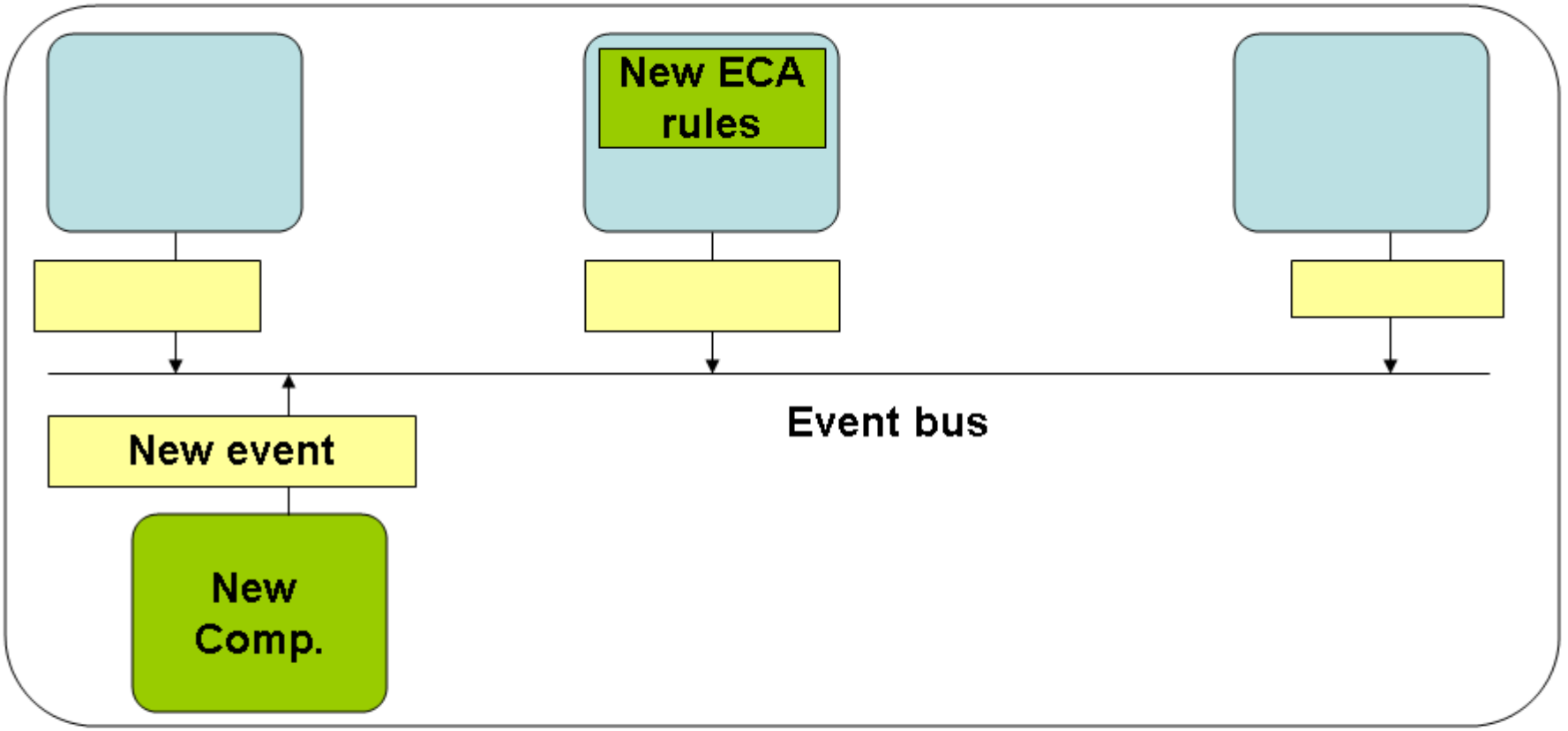
- Event-driven systems seem to dissolve the notion of a sequential process.

-The interesting question is: Is there still a concept of „business process“ in this system?

- And if not, is it needed and how could it be created?

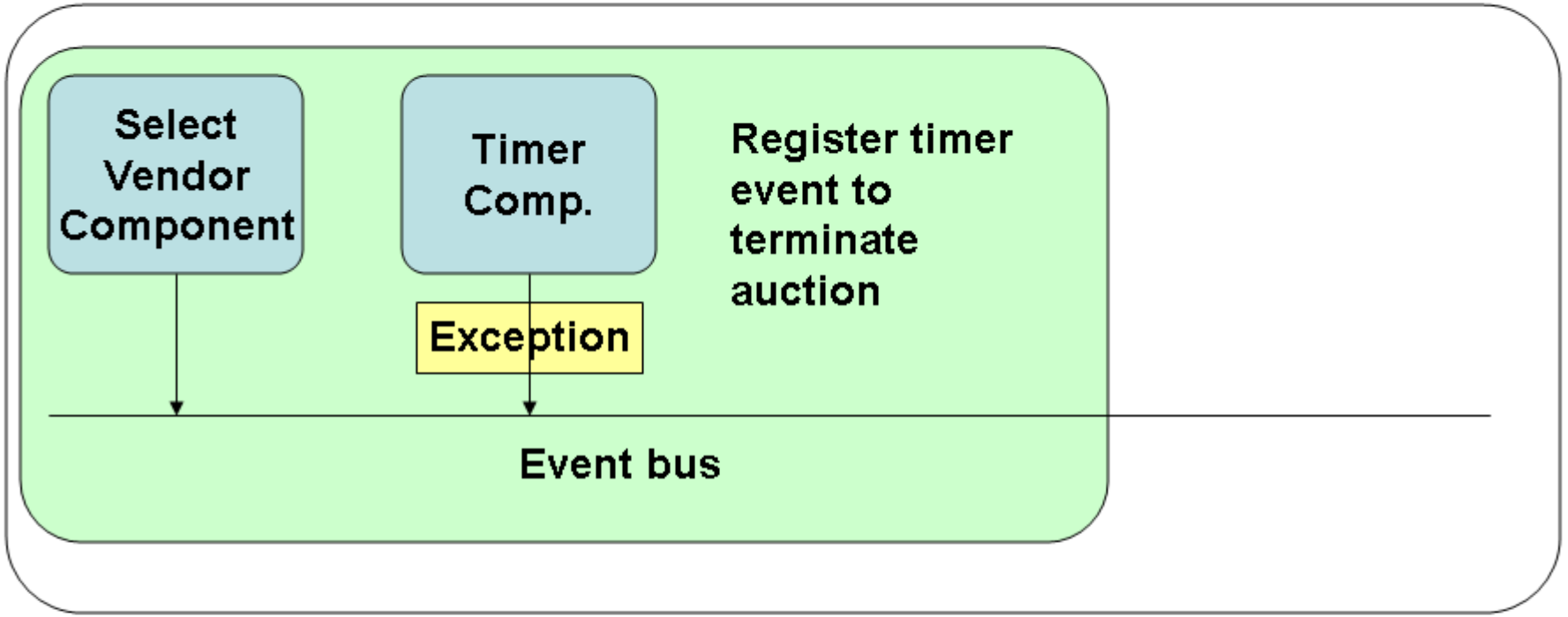
- How are events correlated to form a higher-level abstraction?

Extensions



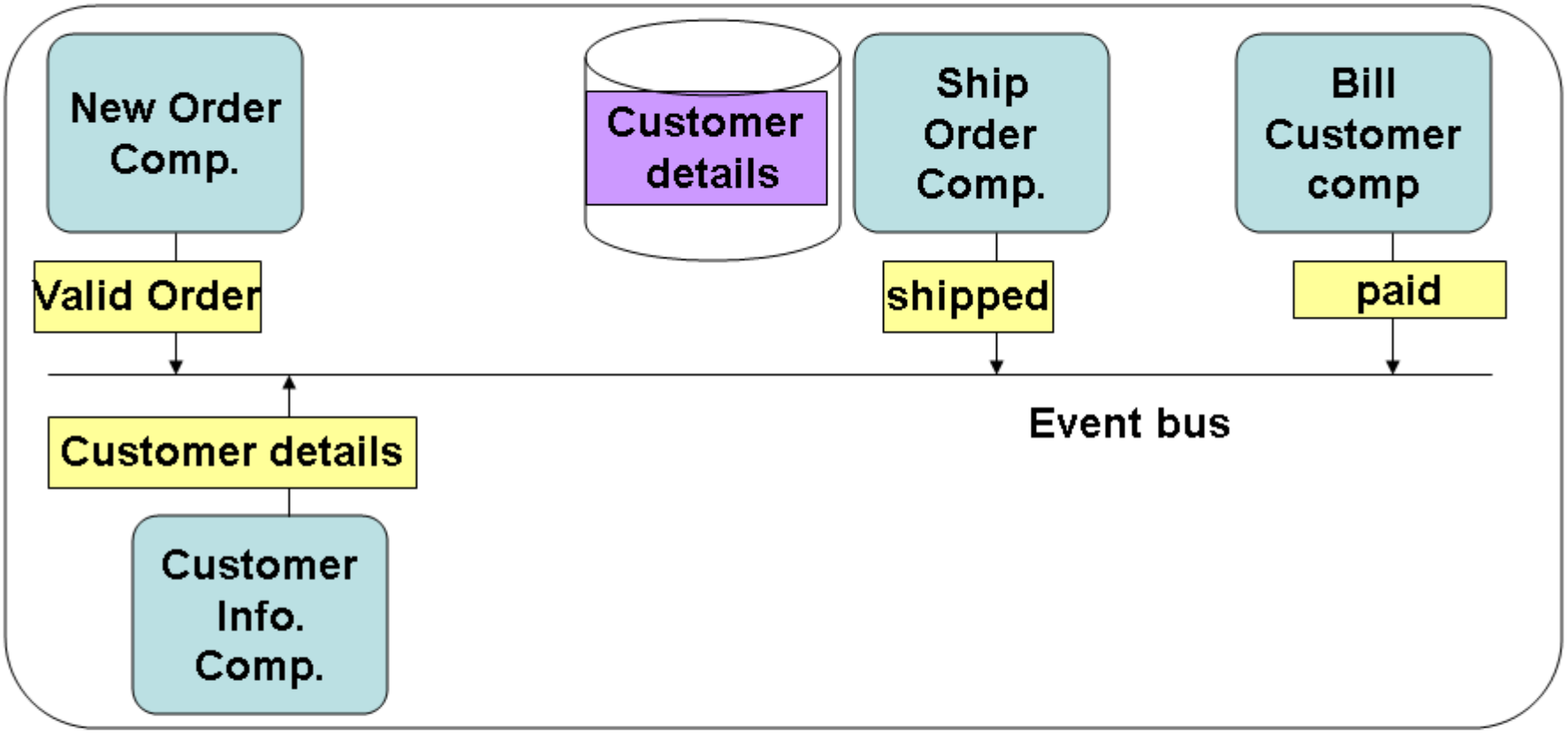
Due to their de-coupled nature event-driven systems can easily be extended with new actions. A new component can easily receive events and produce new events. Of course, the new events might need rule changes within existing components to become effective.

Exceptions



Luckham argues for exceptions as first class citizens. This means that exceptions should be handled in the same way as regular events. An external timer component can raise an event that indicates that still no proper vendor has been selected and that the criteria have to change now. The select vendor component will now change the criteria and select a vendor. (Luckham pg. 38)

State in Event-Driven Designs



Components can remember events and keep their own database. An important question: when should a component invalidate this information?

Monitoring and Causal Tracking: To Do‘s

A component or process failing needs:

1. A way to detect events (mostly lower level events) that led to the situation
2. A determination of the „root“ cause
3. A way to trace the consequences for higher level components or processes
4. A way to predict consequences on the same or higher levels as the failing components/processes

Event: A record of an Activity

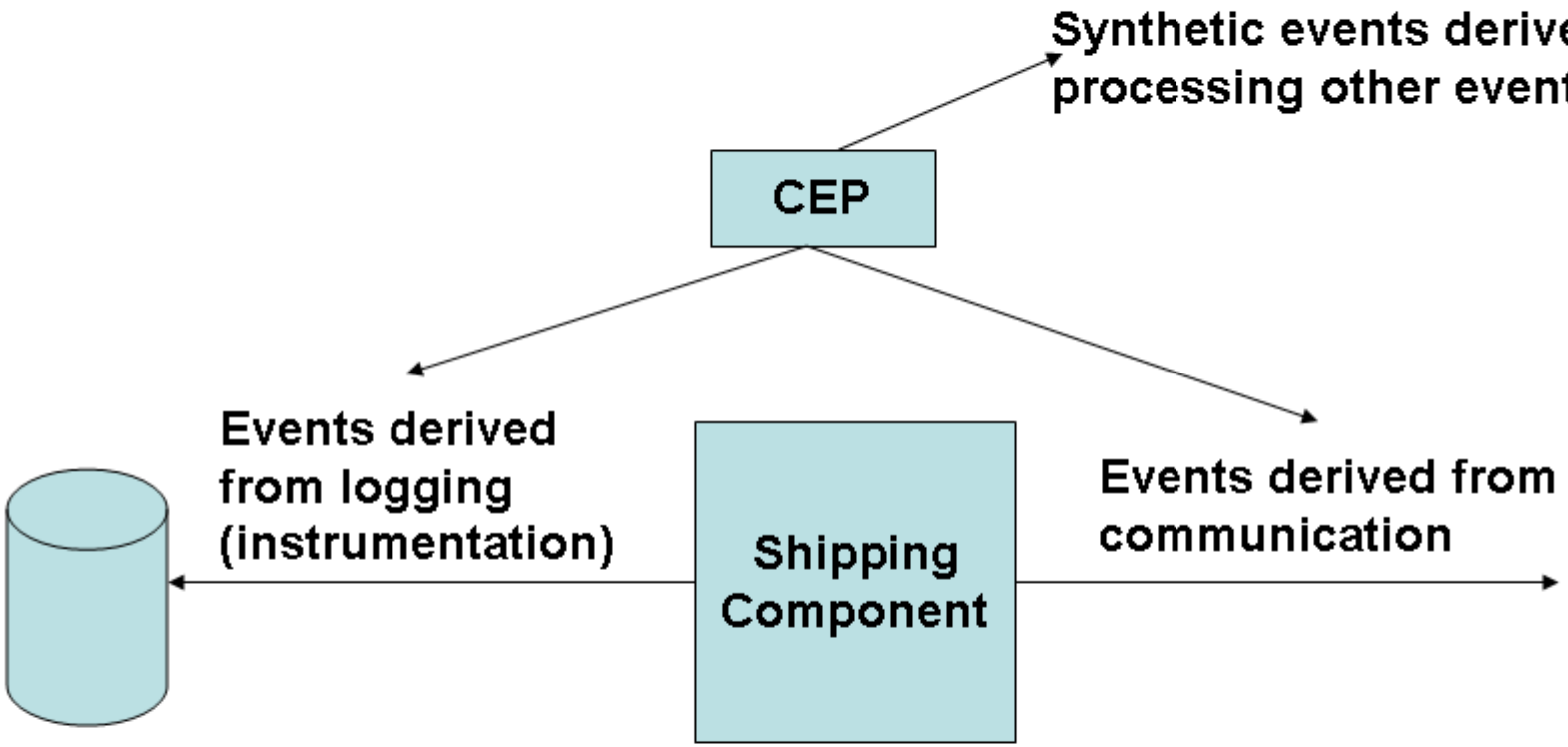
Type: shippedProductEvent

- ID: 234897895
- Time: 11.11.2011
- Customer and Product Data
- Causality Information: E3, E89, E..
- Aggregation Information

According to Luckham an event is much more than just a message. It include vital meta-data on what caused it, correlation information, time etc.

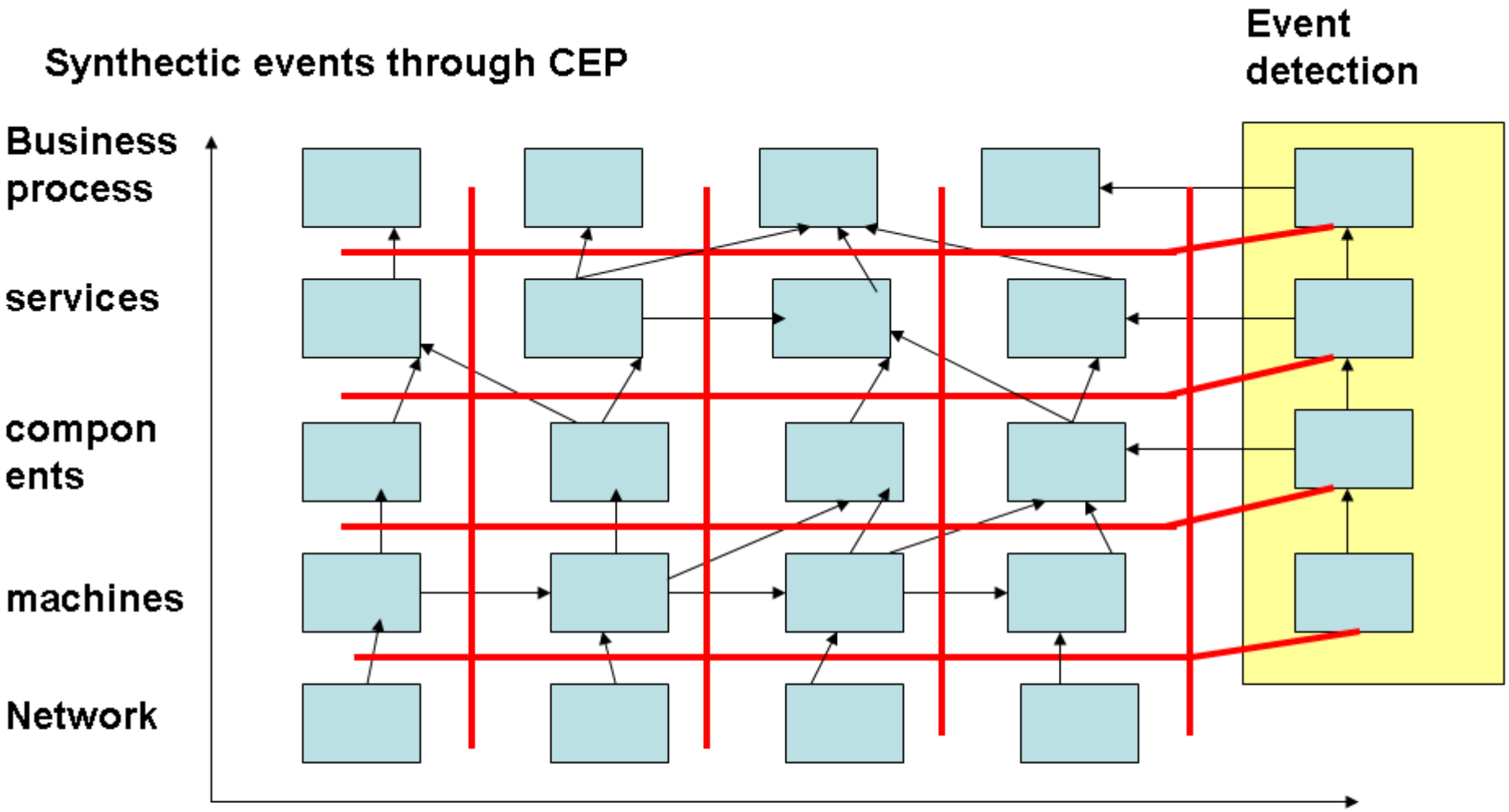
This means that events need to be specially created.

Event Sources



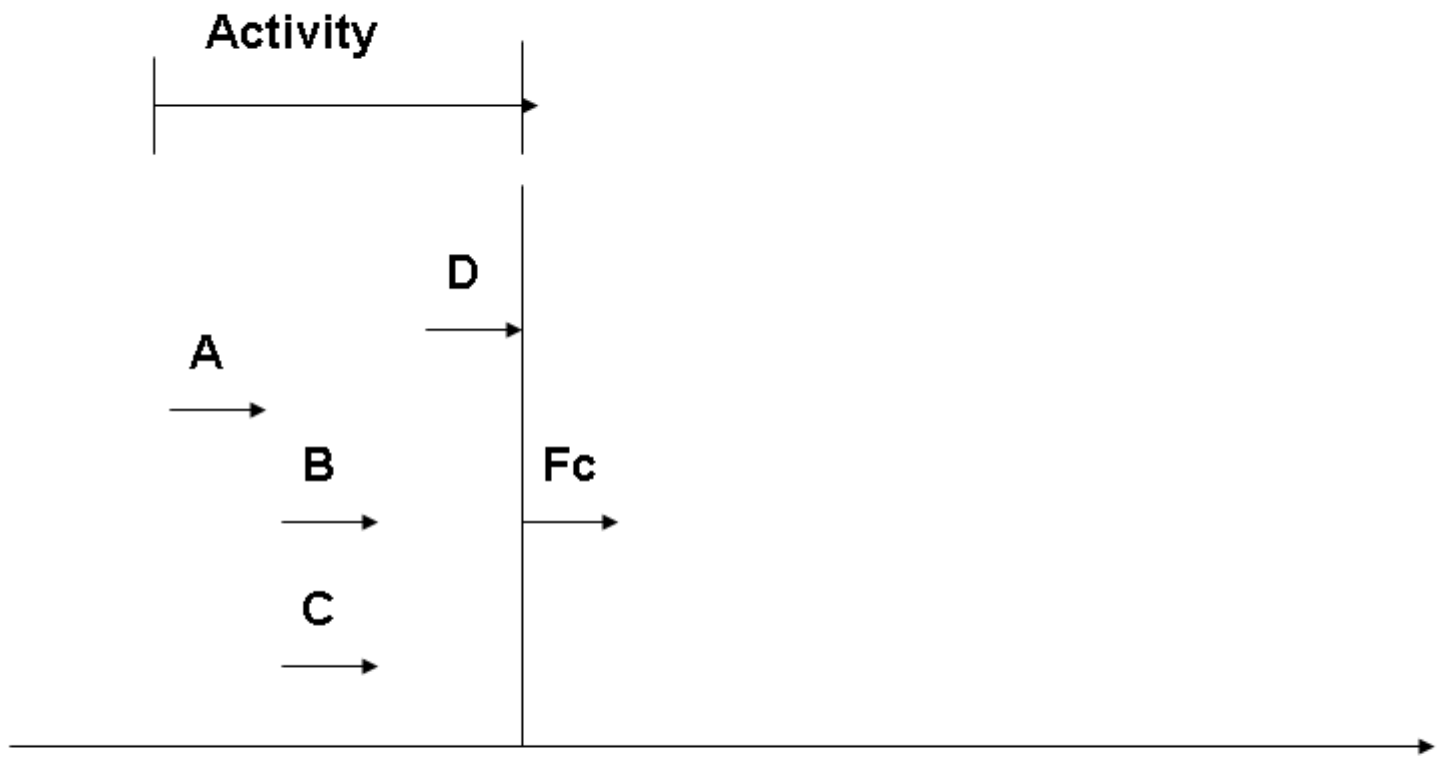
The differences in communication channels are not relevant. Messages and events can flow through the same system, e.g. a MOM (Luckham pg 91ff)

Synthectic events through CEP



Events on different levels of the architecture are aggregated into more abstract events. Those abstract events can simply be representations of a concept (e.g. an attack) which are not normally part of the event chain within a company. But they can be fed back into normal event processing, e.g. causing the shutdown of a process or machine. Should only aggregate events be forwarded? The propagation of all events will probably cause to much data.

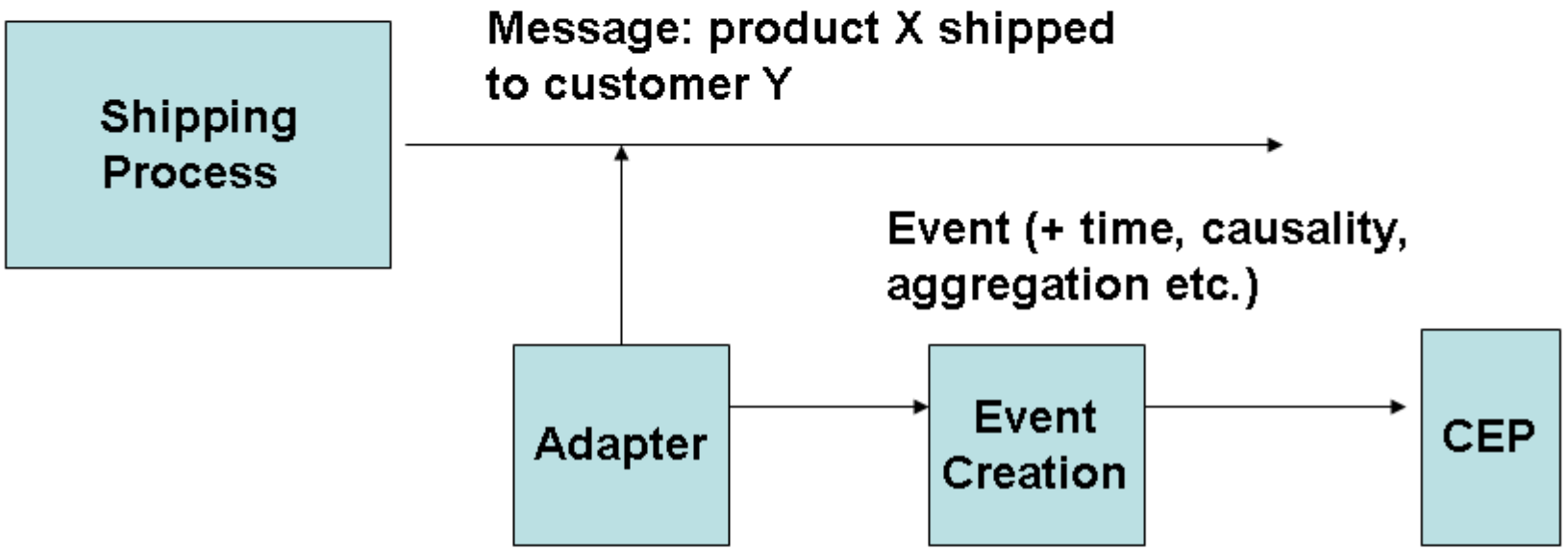
Event Aggregation and Complex Events



Fc is a complex event that represents a certain activity over a timer interval. Once detected/created, Fc is a new event in its own right. Rule: if (first(A) AND then(B parallel to C) AND then(D) create(Fc). „B parallel to C“ is a constraint for this rule. Or: Timestamp(D)-Timestamp(A) < 10sec.

Relationships between events are transitive and asymmetric (Luckham pg. 95ff) and partially ordered

Event creation: Timestamps and Causal Vectors



Event creation includes observation and enrichment. A special adapter observes regular messages and forwards it to components which add vital information that finally creates an event. This event is then forwarded to further CEP processing. Timestamp and causality information are called „genetic parameters“. (Luckham). A causal vector is the set of events that caused the current event. It allows causal tracing between events.

Computational Causality

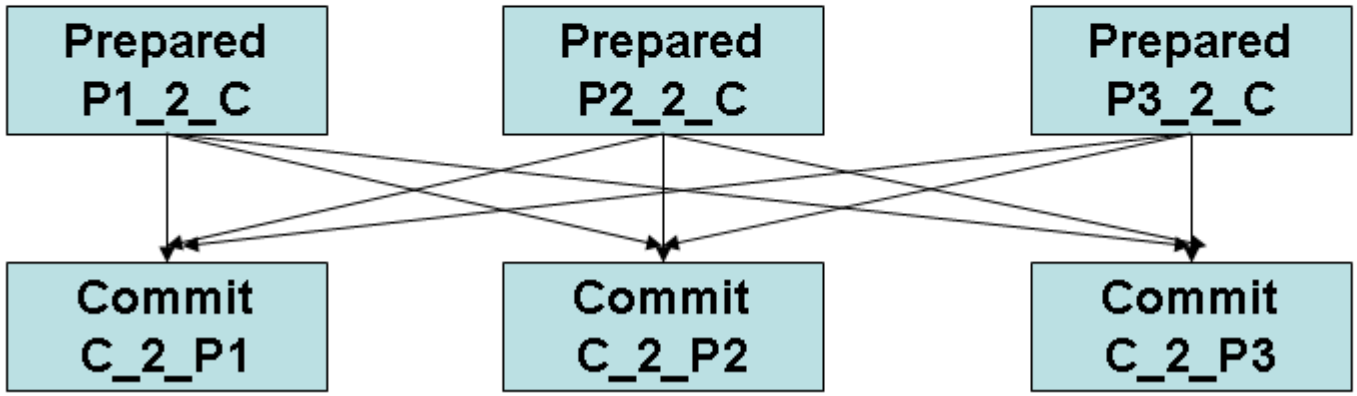
„Cause: If the activity signified by event A had to happen in order for the activity signified by event B to happen, then A caused B. Causality as defined here is a dependency relationship between activities in a system. [..] An event depends upon other events if it happened only because the other events happened. [..] if event B depends upon event A, the A caused B (This is computational causality which depends only upon properties of the target system. It is a much more limited concept than philosophical or statistical notions of cause and effect“ (Luckham, 95)

The dependency relationships are defined by the properties of the target system. This means that the causal rules used to create the event sets for historical causality express the BELIEVES of those people who built the system. Actual system behavior may be based on different rules.

Another point: Each event has a causal vector with events that were instrumental in causing the event. But what about different perspectives for system observations? Theoretically we should be able to keep the causal vectors also independent of the events.

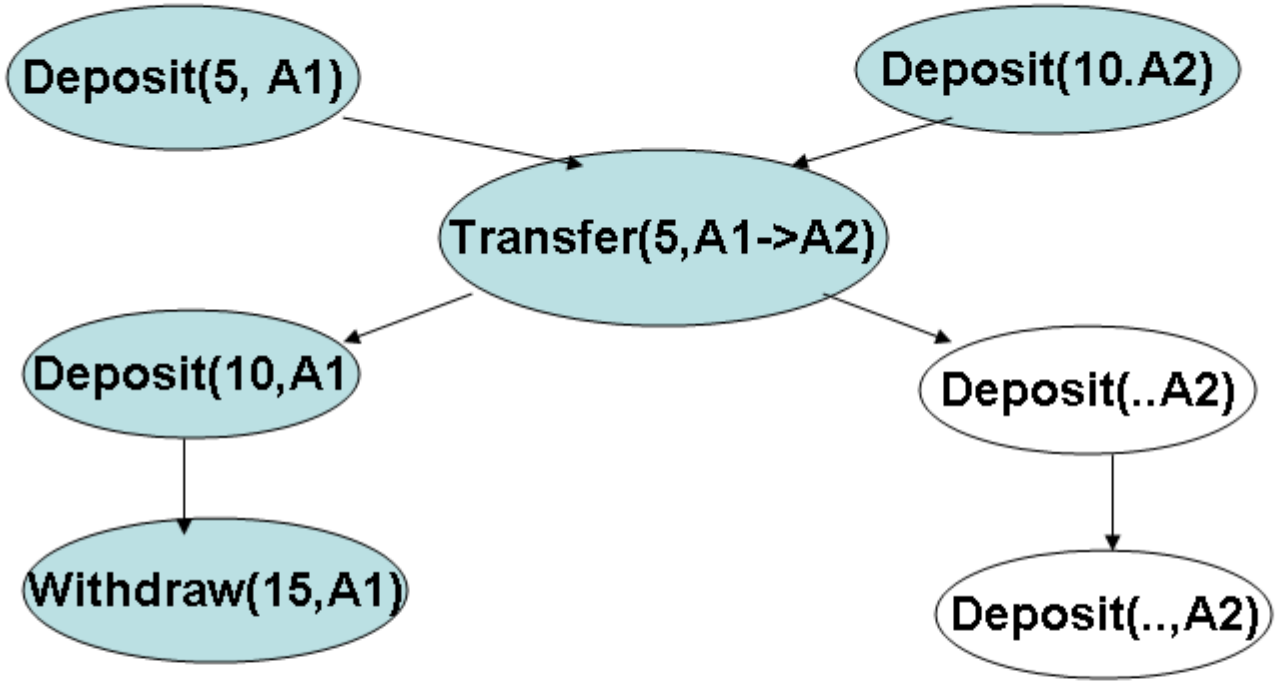
The CPE examples below: are they really convincing with respect to causality?

Additional causal information



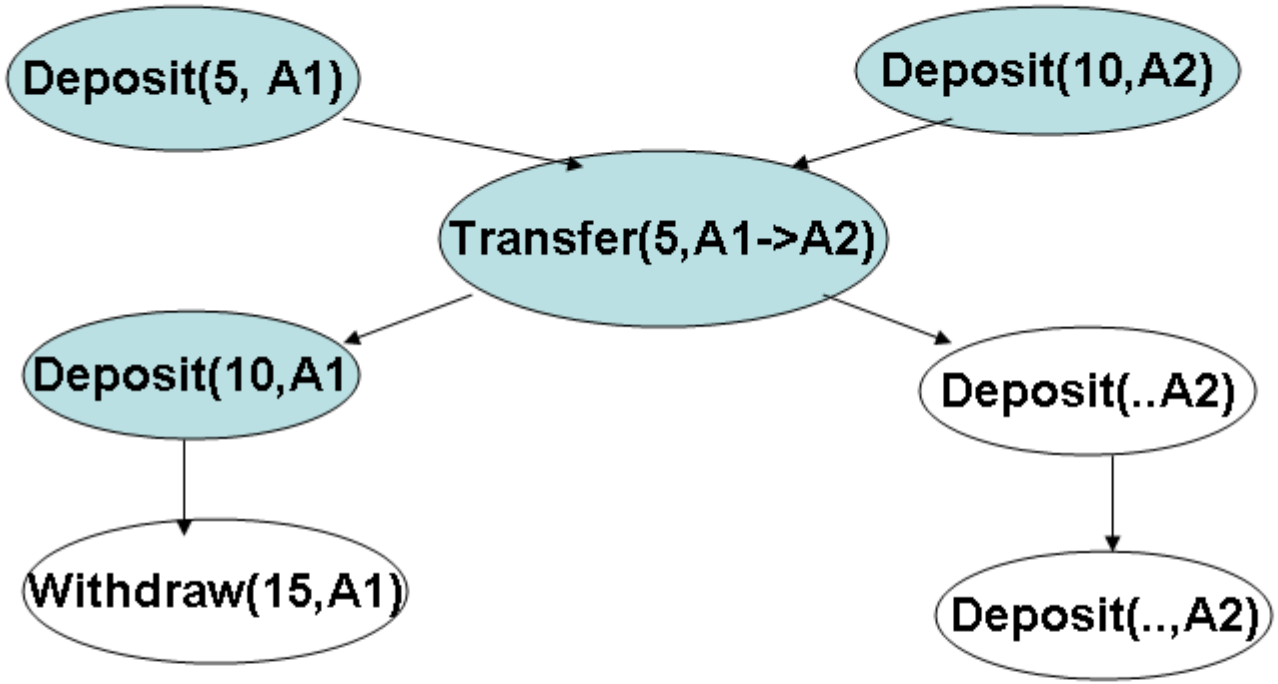
Each input event to the coordinator C is causally responsible for the commit command to every process. Otherwise the coordinator would send a commit command to one process without having the agreement of all three processes. Luckham argues that without the causal relations made explicit a defective logic in the system would not be detected because the system itself behaves correctly in this case. Example form Luckham, pg. 48

Partially Ordered Sets of Events (Posets)



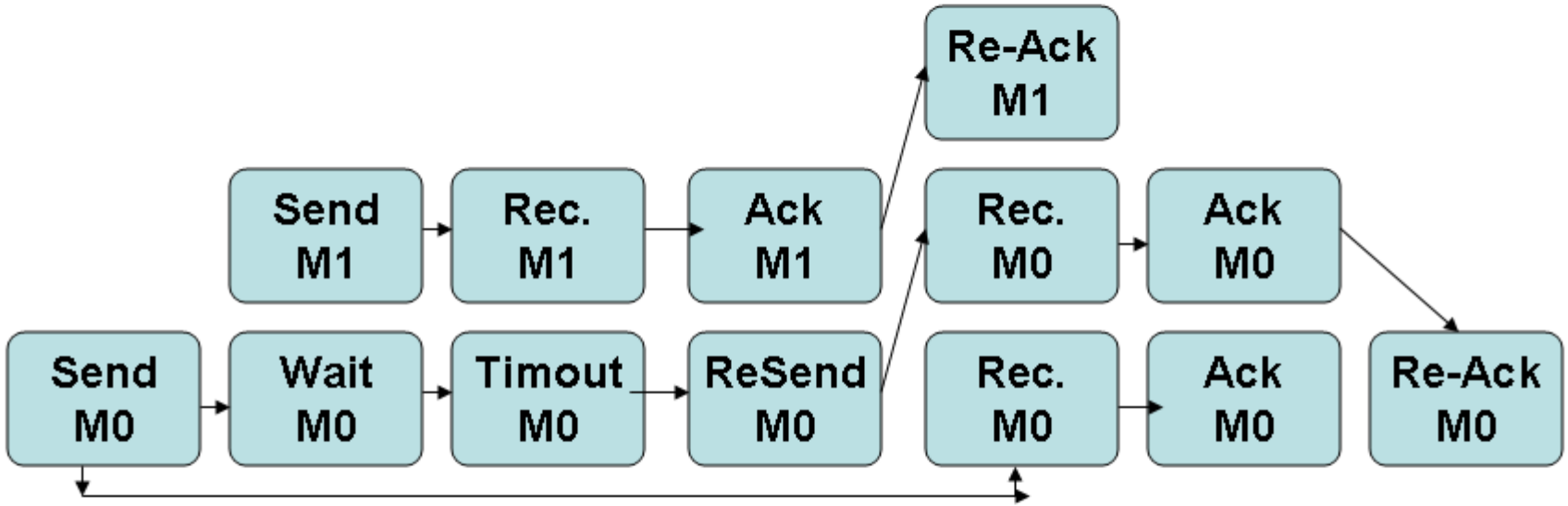
Posets express causality relations between events. This is mostly done using a DAG. Once causality is expressed it can help to reduce the huge numbers of events to find those who are responsible for a certain effect. The causal history of an event includes its ancestors and their causal history. (Luckham 101)

Explanations from causal history



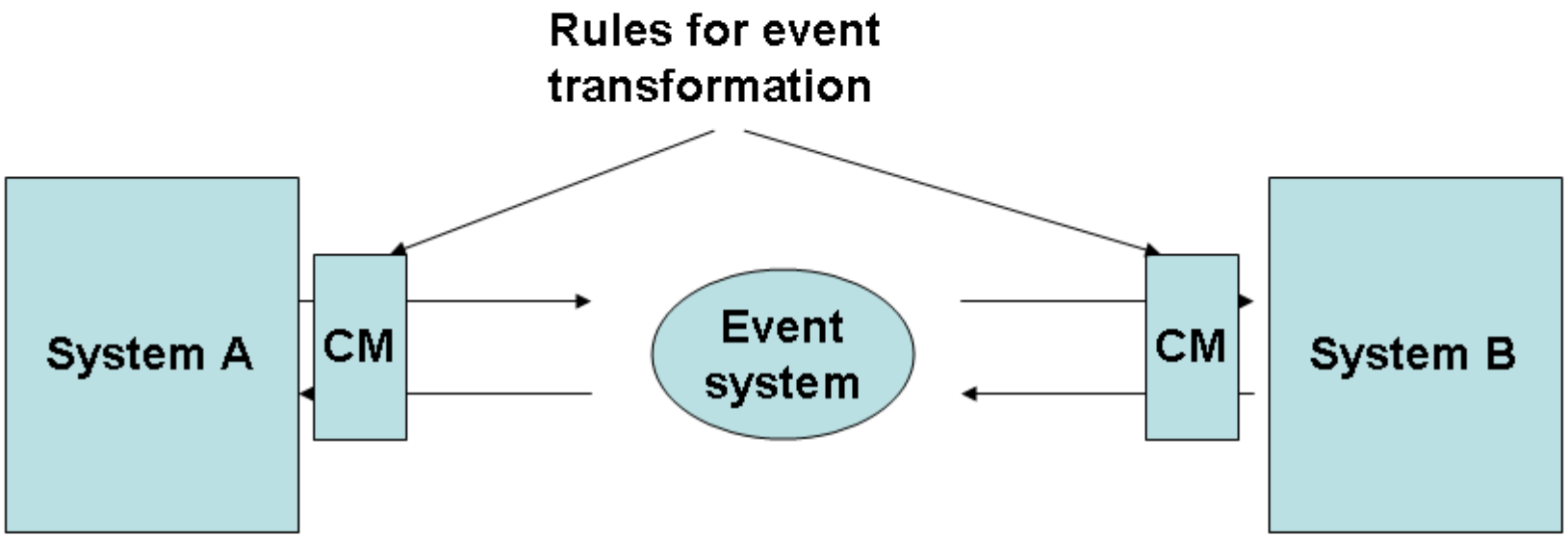
The withdraw event fails because of the previous transfer. Luckham (102) argues that the Deposit(10,A2) belongs to the history of A1 because of the transfer request and the account being a sequentially accessed ressource. The later deposits on A2 are „causally after“ the transfer (why?). Even with an earlier timestamp they would clearly not belong to the causes of the failure. The example is not really convincing.

Alternating Bit Protocol Example



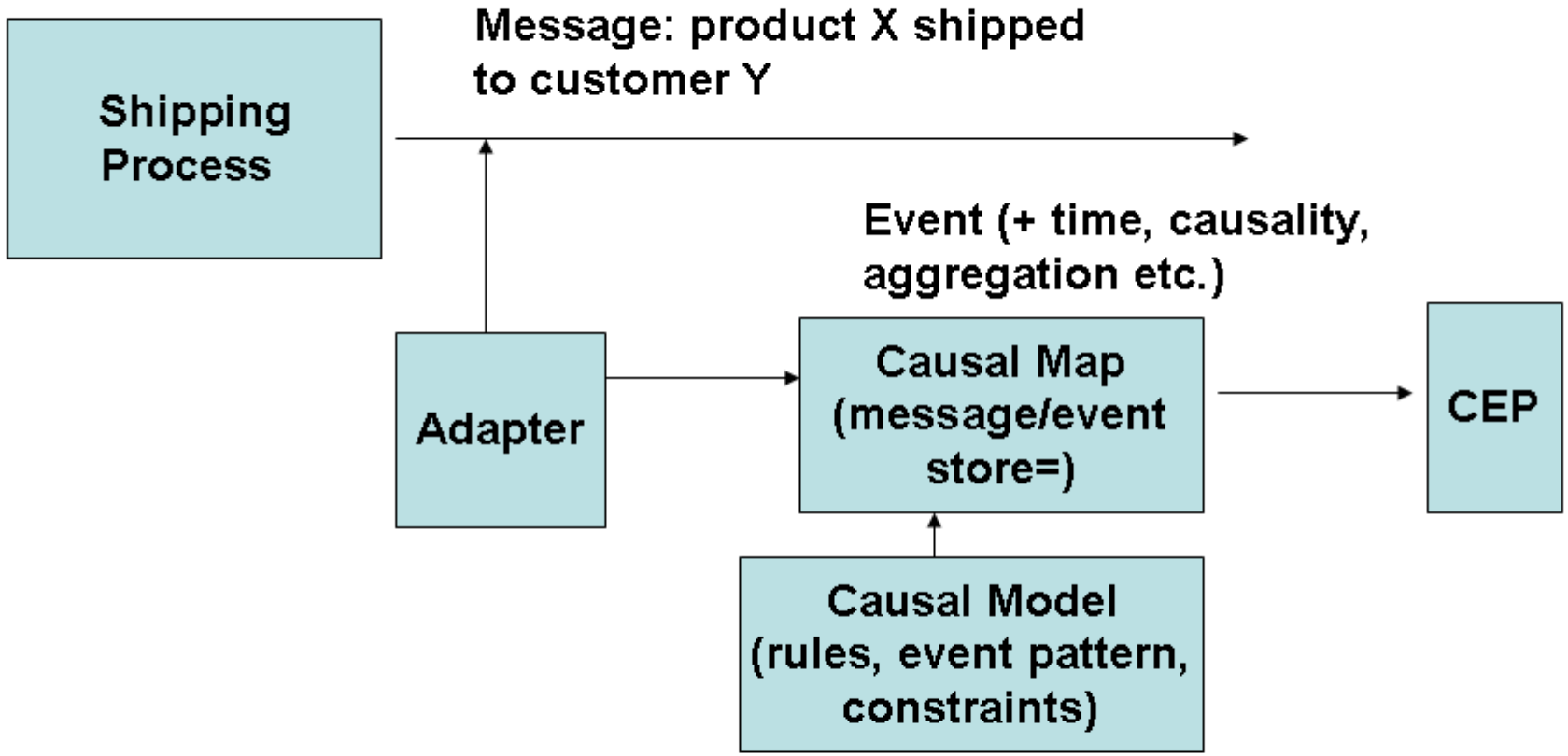
Luckham pg. 103ff. But was the second RecM0 a duplicate during the re-send or a late arriving package? Should the timeout value change?

Causal Models



Causal models can be retrofitted to monitor incoming and outgoing requests between collaborating enterprises. Models can use correlation ideas if they exist or create their own causal identifiers.

Event creation II: Causal Models and Maps



A causal model allows a causal map to process messages and create events. Depending on the available message information and the requested patterns this can be very demanding.

Event Patterns

-Content sensitive (look into event text and timestamp etc.)

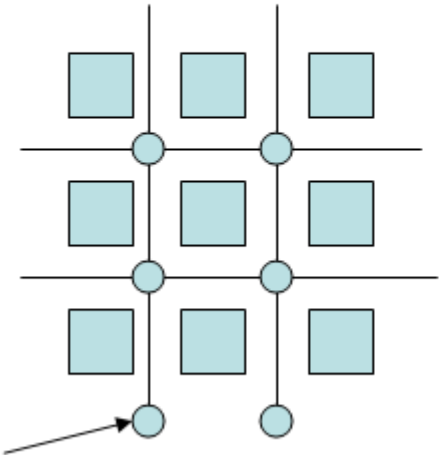
-Context sensitive (the matching state plays a role, e.g. a database lookup is needed)

- Causal filter (e.g. use a reference ID to select only certain events – think about an advertisement which requires to mention a ref. Number in order to get a discount) The ref.Number connects the advertisement with the following orders causally (computational causality)

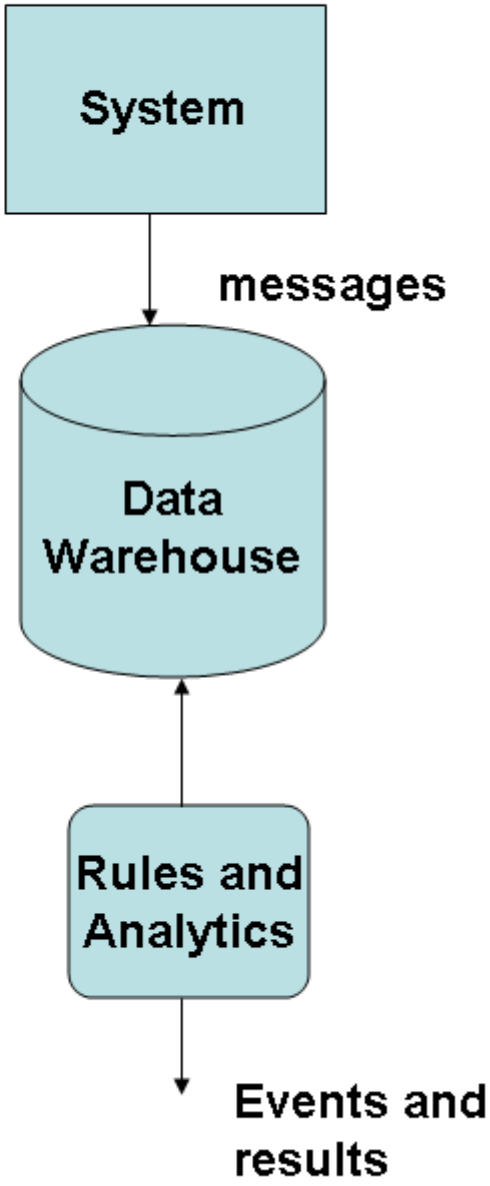
- complex event-relationship patterns (an order, a discount announcement and following that a request for reduction from customers who ordered before the discount was announced

Luckham, pg. 115ff.

OLAP vs. CEP



Causal
Models and
Maps



-Both need causal rules and theories

-CEP can do real-time analysis and feedback to the system

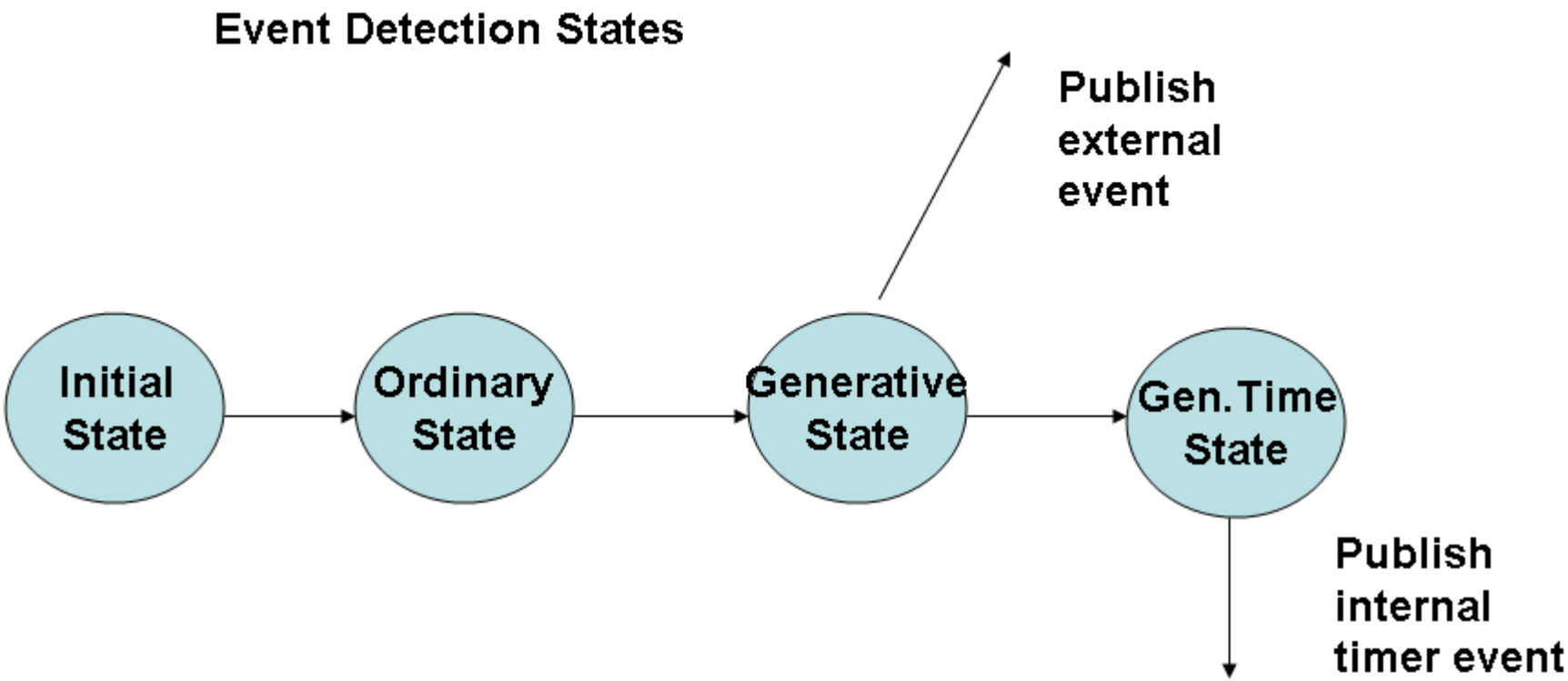
- Both can store the events

Interval Timestamps

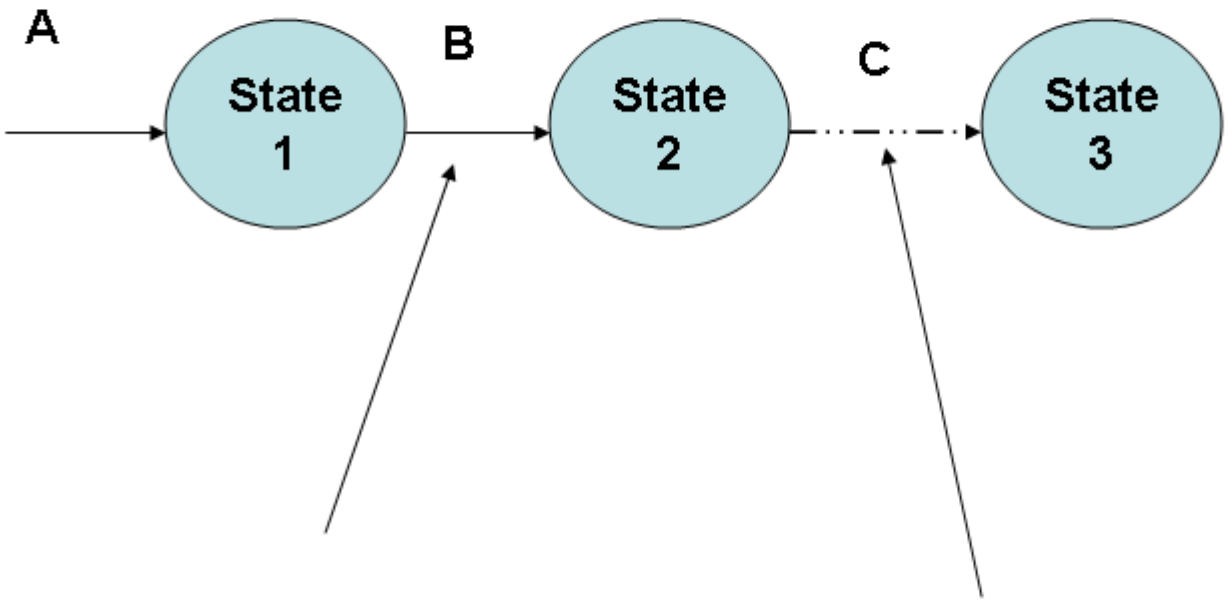
$T = |T_{low}; T_{high}|$

Total order: $T1 << T2$ if $T1_{high} < T2_{low}$

Partial order: $T1 < T2$ if $(T1_{high} < T2_{high})$ V $(T1_{high} = T2_{high}$ AND $T1_{low} < T2_{low})$


Event Detection States
Publish external event
Initial State
Ordinary State
Generative State
Gen. Time State
Publish internal timer event

Weak and Strong Event Transitions



Strong transition with timestamps of B events following totally ordered those of A

Weak transition with timestamps of C events following partially ordered those of A

Composite Event Language

Atom: detect individual events in the input stream and switch to a generative state

Negation: detect all events except the ones negated and switch to a generative state

Concatenation: a composite event is detected where e2 weakly follows e1

Sequence: a composite event is detected where e2 strongly follows e1 (timestamps do not overlap)

Iteration: a composite event is detected where a generative state repeatedly transitions back to the intital state

Alternation: One of two possible composite events is detected. The e-transitions introduce non-determinism.

Timing: a certain time after a composite event a timer event is generated. This timer event is then detected by another composite event.

Parallelization: two composite events are detected in parallel

See Mühl, pg. 238ff.

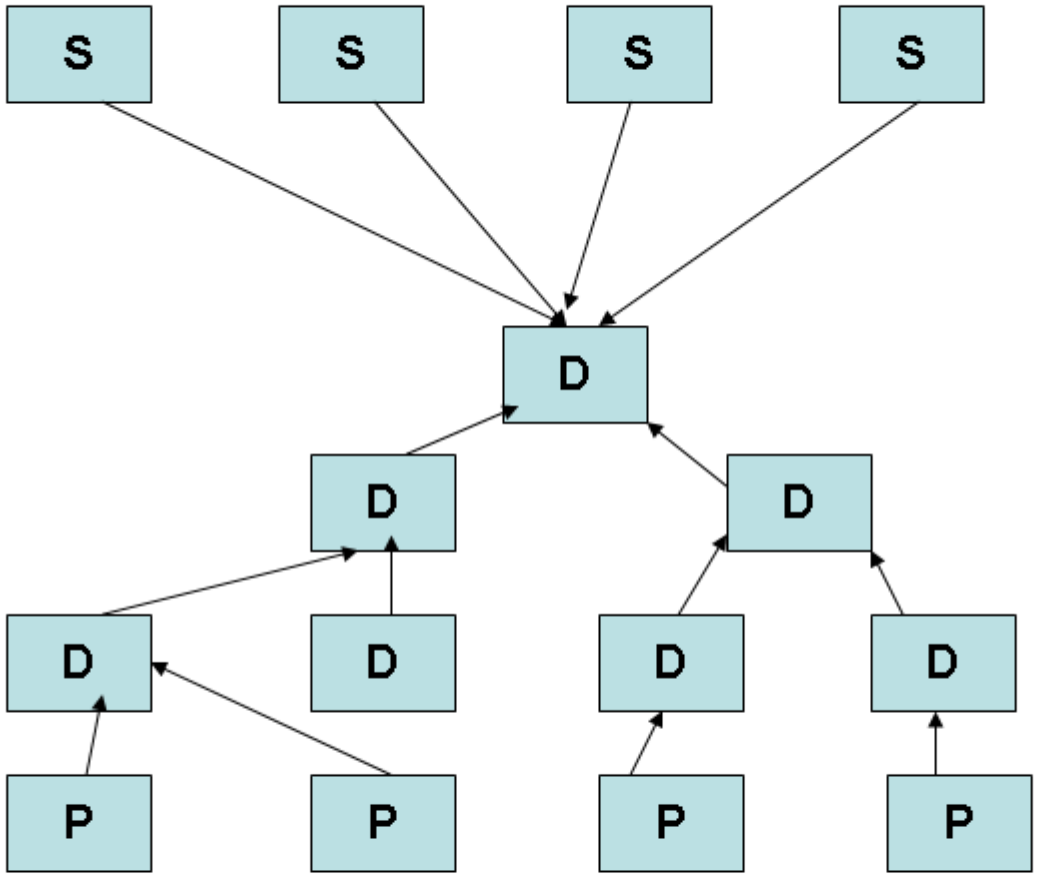
Detection Policies

Early consumption of events:

best effort. Can cause wrong or failed detections because events with older timestamps are still being delayed e.g. in the network

Guaranteed Detection: must wait for totally ordered events. Lost events can cause the detector to wait forever in asynchronous, distributed systems because the delay is unbound. If the network does not re-order events the detection of an event with a newer timestamp allows the detector to conclude that no older events are still in the network.

Distributed Event Detection



Complex event expressions are de-composed and distributed to individual detectors which can be much closer to event sources. Unneeded event atoms can be thrown away close to the source.

Resources

-Judea Pearl, Causality, http://bayes.cs.ucla.edu/BOOK-2K/index.html

- David Luckham, The power of events,

- CEP Portal http://www.complexevents.com/

- Mühl et.al. Distributed Event-Based Systems

- John Sowa, Process and Causality, http://www.jfsowa.com/ontology/causal.htm

- Ken Birman, Reliable Distributed Systems