Course Materials

Course Materials

- Will be posted as the course progresses
- This is new material that is evolving as we go (I will give you preprints of chapters of a book I'm writing)
- Because the materials are in part unpublished, they are protected by password on our web site

  User: CEP

  Password: 6230 (internal phone number of my sec.)
- Other materials will be posted (papers, URLs, etc.)
- Some books are either available in the library (or can be easily purchased via amazon.com or similar)

Books and Overview

- Hinze, A., Sachs, K., Buchmann, A.; Event-based applications and enabling technologies, Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, Nashville, TN, July 2009 (keynote)
- Chandy, K.M., Schulte, R.; Event Processing: Designing IT Systems for Agile Companies, McGraw Hill, 2009
- Taylor, H., Yochem, A., Phillips, L., Martinez, F.; Event-Driven Architecture: How SOA Enables the Real-Time Enterprise, Pearson Education, 2009
- Faison, T., Event-Based Programming: Taking Events to the Limit, Apress, 2006
- Luckham, D.; The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison Wesley, 2002
- Mühl, G., Fiege, L., Pietzuch, P.; Distributed Event-Based Systems, Springer, 2006
- Hinze, A., Buchmann, A. (eds.); Principles and Applications of Distributed Event-based Systems (to appear)
- Buchmann, A., Georgakopoulos, D., Hinze, A.; Complex Event Processing, Morgan-Claypool Lecture Series, 2010 (to appear)

Articles architecture

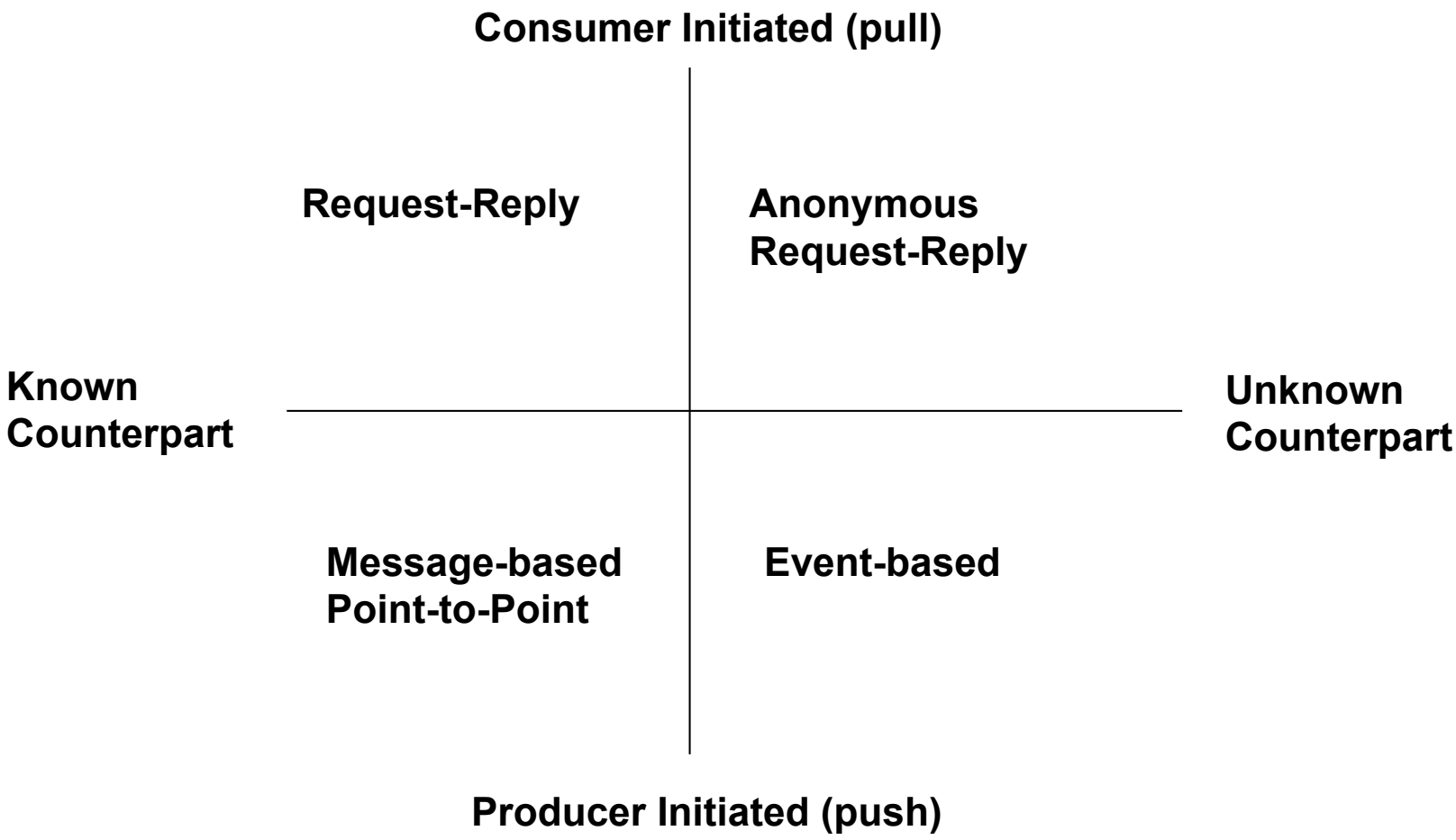
- Dustdar, S., Gall, H., Hauswirth, M.; Software-Architekturen für Verteilte Systeme, Springer, 2003
- Jean-Louis Maréchaux; http://www.ibm.com/developerworks/webservices/library/ws-soa-eda-esb/
- Michelson, B., Event-Driven Architecture Overview http://www.omg.org/soa/Uploaded%20Docs/EDA/bda2-2-06cc.pdf
- http://www.drdobbs.com/architecture-and-design/208801141
- Etzion, O.; Towards an Event-Driven Architecture: An Infrastructure for Event Processing, Position Paper, Springer LNCS 3791, 2005

References

- Lots more to come as we go along
- Web site of the Event Processing Technical Society
  http://www.ep-ts.com/

Be prepared to read a lot!

Interactions



Interactions

- Request-reply: user/consumer initiated, known counterpart
- Anonymous request-reply: user/consumer initiated, unknown counterpart
- Point-to-Point messaging: producer initiated, known counterpart
- Event-based: producer initiated, unknown counterpart

Who controls the interaction?

- Invocation semantics
- Termination semantics
- Quality of service

Interactions

- Request-initiated interactions
  - Initiation?
  - Duration?
  - Termination?
- Time-initiated interactions

- Event-initiated interactions

Interactions

- Request-initiated interactions
  - Initiated by an explicit request
  - Ends immediately after completion (1 round trip)
- Time-initiated interactions
  - Initiated at predetermined time (absolute or relative)
  - Ends immediately
- Event-initiated interactions
  - Initiated by an event (the triggering event)
  - Open ended or with validity interval

Question

- Are time-initiated interactions event based?
- Are time-initiated interactions request based?

Time-initiated interactions

- Some authors classify time-initiated interactions as a separate class
- I prefer to consider them a special case of event-basedinteractions

➔ Temporal events are first class events

What is an event?

- An event is a happening of interest (David Luckham)
  - Intuitive
  - Describes situations
  - Can't be used computationally
  - Need a measurable quantity

➔ This definition of an event is useless as a basis for computational systems

What is an event?

- An event is a (meaningful) change of state (Mani Chandy)
- Requires a model of reality
- Changes are with respect to the model of reality
- Attributes of the model of reality are readily measurable (e.g. RFID detection at a gate, temperature value)
- Since this definition is predicated on change, it is not obvious how to handle situations of interest where the environment has not changed (e.g. temperature has not changed in the last hour)

What is an event?

- An event is a detectable condition that can trigger a notification (Ted Faison - Event-Based Programming: Taking Events to the Limit)
- Reporting-based view of events
- More general since it considers the absence of change a detectable condition
- Depends on a reporting capability in the form of notifications, defined as an event-triggered signal sent to a run-time recipient
- What is not detected did not happen

What is an event?

- An event is a meaningful change of state (Buchmann)
- Based on a model of reality
- Not every measurement is an event
- Interest (what is meaningful) is determined by the producers and consumers
- Producers can advertise the events they detect
- Consumers subscribe to events of interest
- Connection is made by notification service
- Time is an integral part of the definition of state
- Since time advances, two observations of the same value are two distinct events
- This makes it easy to deal with status events

Event and event objects

- Events have a representation
- The event representation is generally known as an event object
- Minimally, an event object has
  - An identifier
  - A type
  - A timestamp
  - Optional attributes
- Events are typically represented as tuples

Temp, 13-2456, 090221112345,24.6

- As in a database schema, the attributes of an event object carry their specific semantics
- Design of event types can have far-reaching effects

Types of events

- Simple event: any discrete event that is directly detectable and not the result of a composition
  - Change event: detected change of state
  - Status event: meaningful observation that the state has not changed between distinct detections
  - Note: by introducing time as an integral dimension of event characterization, status events and change events are treated uniformly
- Complex event: any event that is produced by composing two or more simple events through operators of an event algebra and/or enriching an event with external information

Types of compositions

- Aggregation: typically understood as the application of relational aggregation operators, such as max, min, sum, count or avg

Note: it is generally assumed/accepted that the events that are aggregated are homogeneous.

- Composition: combination of two or more events (simple or complex) through the use of an operator of the event algebra. Participating events can be heterogeneous.

Note: the definition of the EPTS requires in its definition of a composite event that the instances of the constituting events be preserved and carried with the composite event. This has not been universally accepted.

Types of events – derived events

- Derivation: derived events are typically events of higher level of abstraction. In the derivation process we apply domain semantics to define new events based on observed events.

Example: 5 simple temperature-reading events taken every 5 minutes and showing that the temperature is monotonically increasing lead to an air-conditioner-failed event

Assumes: Temperature readings are ambient temperature readings, uses domain knowledge about location of sensor(s), fact that AC is on, etc.

- This corresponds to the orientation phase in the OODA cycle.

Complex events

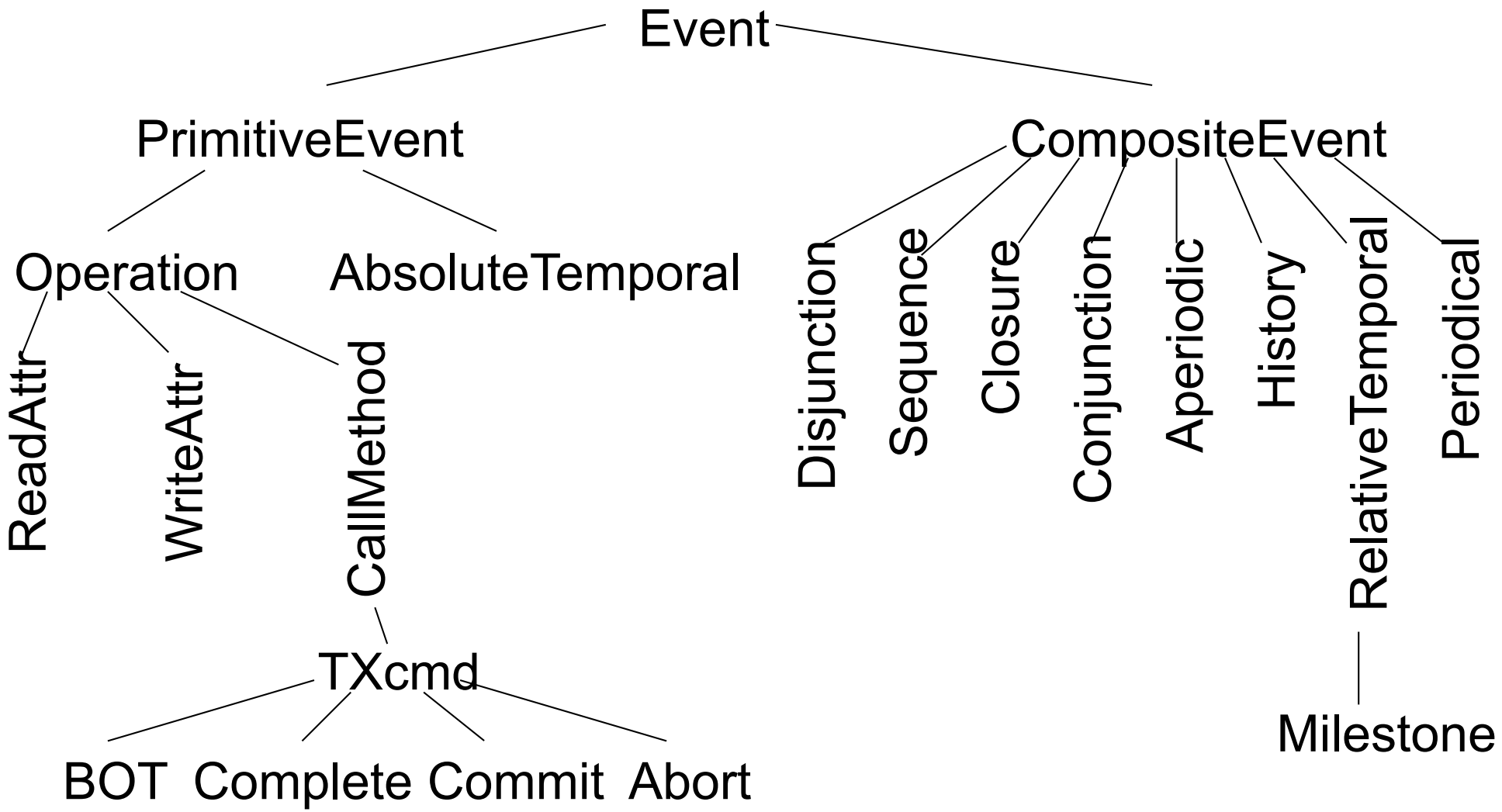
- Complex events are all events that were obtained through the application of aggregation, composition or derivation.

Note: The process of obtaining a complex event may be much more complex than the simple application of an operator of an event algebra, i.e. it could be a whole program that detects an interesting event pattern.

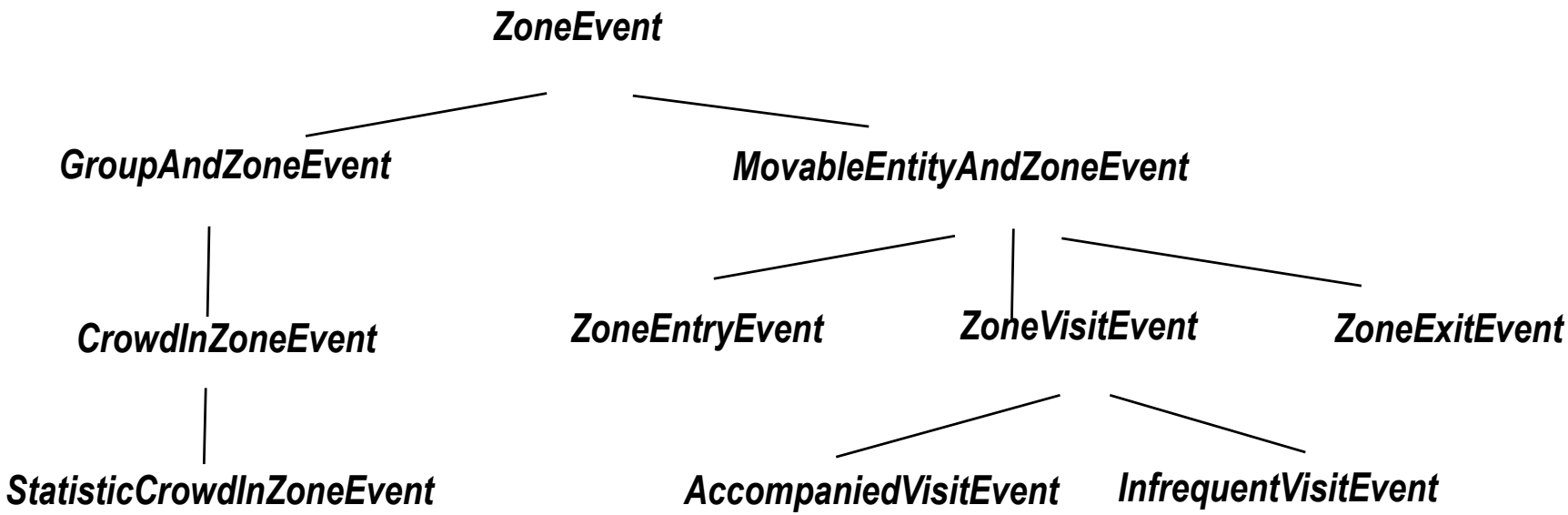
Event type hierarchies

- Events can be arranged in event type hierarchies (typically with the same abstractions of generalization, specialization and aggregation used in object systems)
- Event type hierarchies are domain dependent
  - Type hierarchies for active databases
  - Type hierarchies for surveillance systems
  - Type hierarchies for RFID-based supply-chain management
  - Type hierarchies for Ambient Intelligence (and its specific domains)
  - …

REACH type hierarchy



Surveillance system event type hierarchy



Event Driven Architecture (EDA)

- Goals:
  - Provide agility
  - Provide flexibility
  - Support celerity

- Event producers need not be aware of who will eventually consume the events
- Event producers and consumers should be decoupled in space and time
- Event consumers are notified as soon as possible about events of interest

Properties of EDA – Timeliness/Celerity

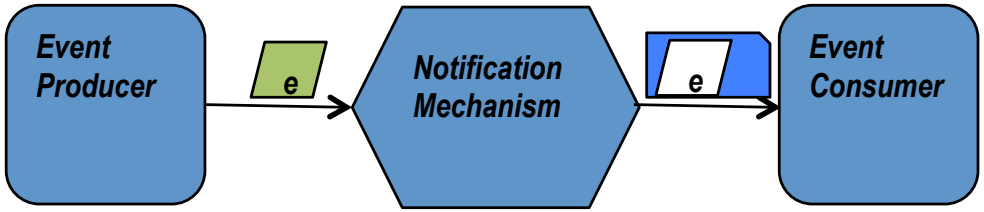
- Reporting of current events as they happen
  - No store and forward
- Pushing notifications of events from the producer to the consumer
  - Events are packaged as notifications and delivered to consumer
- Responding immediately to recognized events
  - Deferred responses are immediate responses to a complex event involving some other event (temporal or any other demarcation event)

Properties of an EDA – agility and flexibility

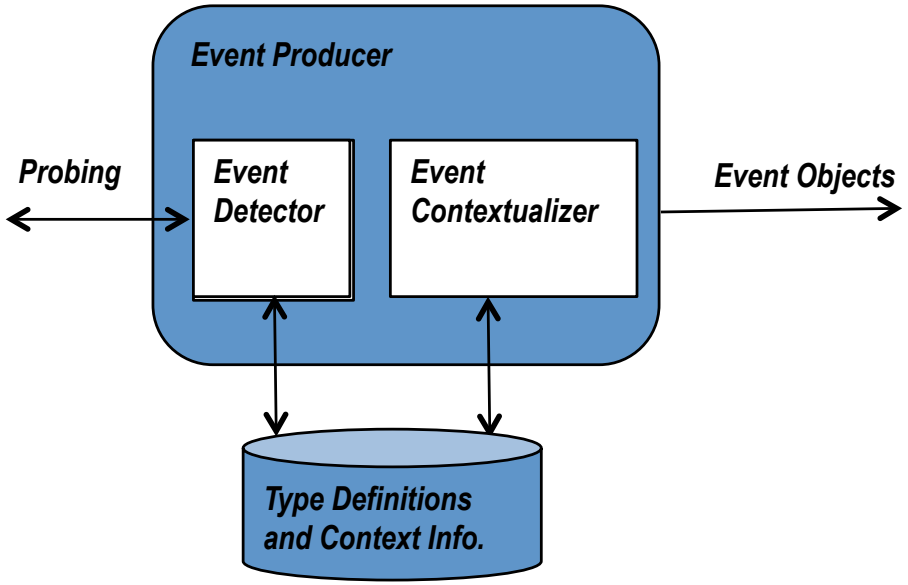
- Communicating one-way without need for acknowledgements
  - Event-based interaction pattern does not require an answer, i.e. the event producer will not block
- Reacting to event notifications and not to commands
  - Interested parties must subscribe to events
  - Subscribers may unsubscribe and this does not affect the event detection, just the notification process
  - Because the event producer is not aware of the consumers, it cannot ask them to execute any actions
  - Event-based systems are consumer controlled
  - New consumers can be added without affecting the rest of the system (except the notification mechanism)

Components of an Event Driven Architecture

- Minimally, an EDA consists of event producers, event consumers and a notification mechanism

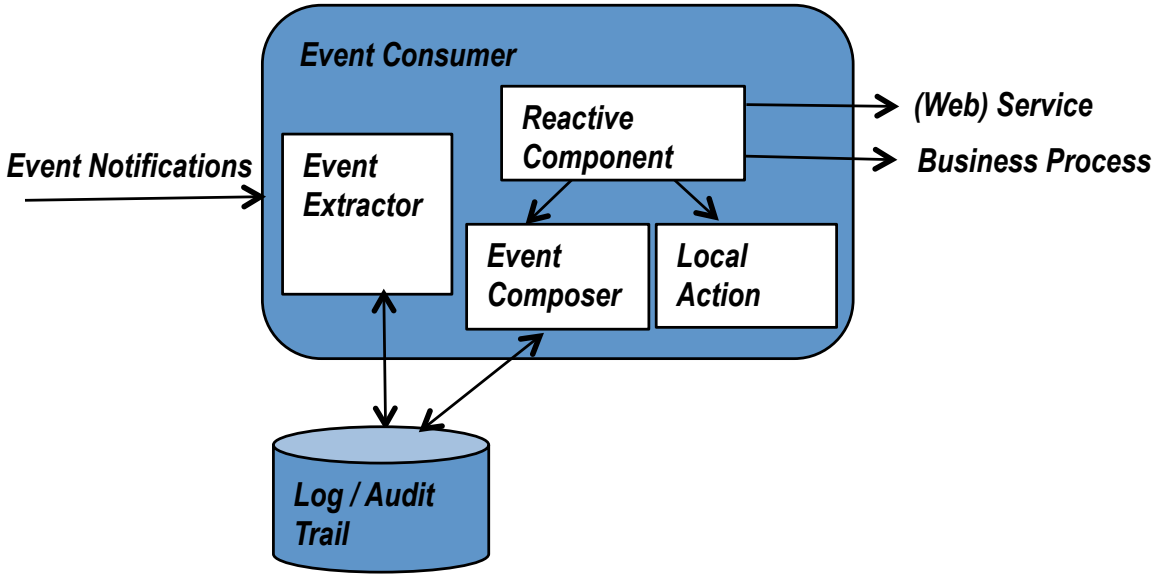


Event Producers



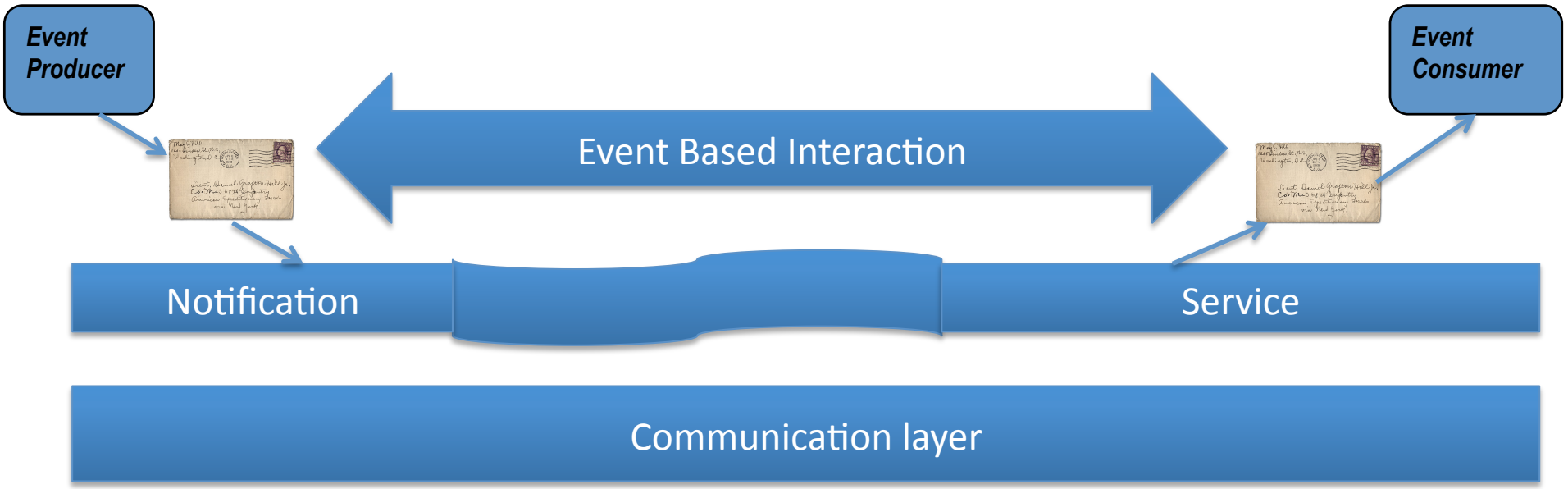
- Detect events and produce event objects. Their structure is defined by the event type and contains the necessary event parameters
- Event parameters are instantiated by the event detection process and the event contextualization process.
- The event detection process typically will probe the environment. The event contextualization process may rely on external data sources, type and context information may be held locally.

Event Consumers



- Receive event notifications from the notification mechanism/service.
- Must unpack the event notification, extract the event object and execute an action in response to the received event.
- The response may be a local action, the invocation of a (remote) service or business process, an event composition or storage of the event for logging.

Event Notification Service



- Key questions:
- How are producers and consumers brought together?
- Does the channel deliver all messages or does it filter?
- If filtering is done, on what criteria and where are the filters placed?
- Are events only routed by the notification mechanism or are they transformed?
- If transformations are applied, where are they applied and what are they?

Dedicated Channel

- The simplest channel is a dedicated line between producers and consumers of events.
- Routing is implicit and hard-wired.
- This is a valid option for applications handling very high volumes of events, with few event producers and event consumers willing to pay for the dedicated communication channel.
- A dedicated channel does not make any assumptions about filtering. All events could be transmitted to the consumer, leaving the filtering to that component or filters could be placed at the source.

Common Area Channels (push?)

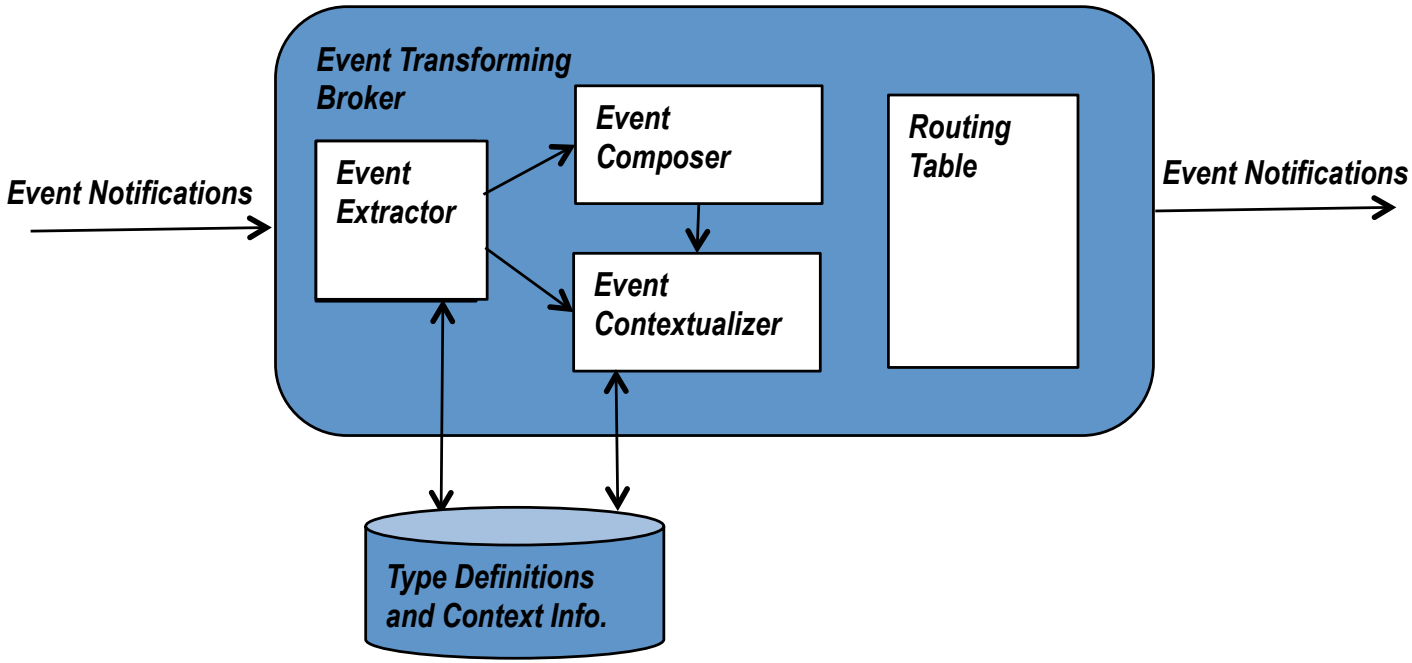
- Two kinds of common area channels are popular:
  - blackboards
  - queues
- In a blackboard-type channel publishers post their detected events to a common area and consumers pick up events from there.
  - tuple-spaces (extreme form of a persistent tuple-space is a relational database)
  - queues are message buffering structures administered by a queue manager. Can be persistent vs. non.persistent, transactional vs. non-transactional.

Common-area channels provide asynchrony and loose coupling, ***do not fulfill end-to-end push-style notification*** required for EDA. Consumers must ***pull*** events from common area.

Notification Routing Channels

- Notification routing channels contain one or more brokers that implement some form of routing table.
- Routing tables are built based on event subscriptions.
- Consumers declare their interest in certain (types of) events, the brokers mediate between producers and consumers.
- There exists an N:1 relationship between producers and broker and a 1:N relationship between broker and consumers.
- If no filtering occurs in the broker, we speak about flooding.
- The power of notification routing channels, lies in the ability to filter event notifications and deliver only relevant notifications to the consumer.
- Criteria for filter definition and placement discussed later

Event Transforming Channels



- Network of brokers, brokers act as consumers/producers
- Simplest form: take in an event and change the structure of the event object and/or the format of one or more parameters
- Can aggregate, enrich or compose events
- Notification Transforming Broker receives event notification, unpacks it, transforms the event, repacks it into a notification and routes the new notification according to its routing table.

Different types of Publish/Subscribe

- Channel-based Pub/Sub
- Subject-based Pub/Sub
- Type-based Pub/Sub
- Topic-based Pub/Sub
- Content-based Pub/Sub
- Concept-based Pub/Sub

Channel-based Pub/Sub

- In this type of Pub/Sub system, subscribers subscribe to a named channel.
- All the events of a given type are dumped into the channel.
- The subscriber receives all the notifications published to this channel and must apply the filters.
- This is the approach used in early middleware platforms, e.g. CORBA Notification Service

Subject-based Pub/Sub

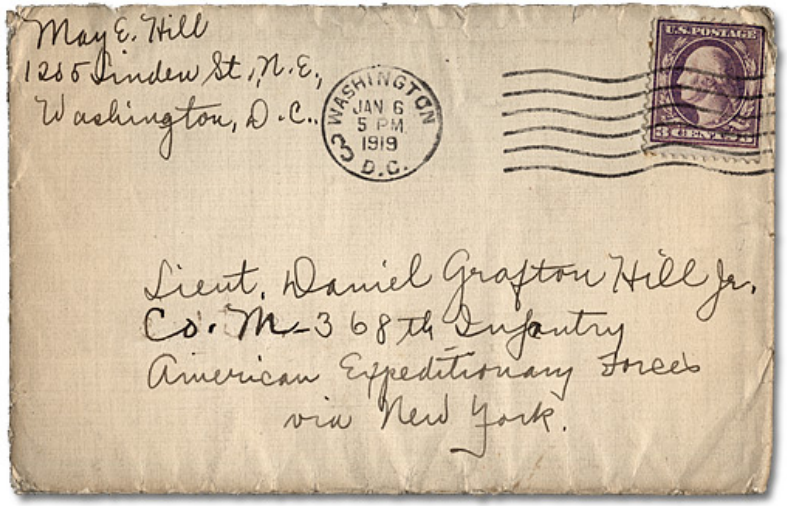
- Notifications are annotated with subjects.
- Subjects are expressed as strings and organized in (static) subject hierarchies.
- Filtering occurs through string-matching along the paths of the subject hierarchy.
- Wildcards are commonly used.
- This approach is simple yet powerful.
- Limitations in expressiveness because each match occurs strictly along a single path of the subject hierarchy.
- Limitations in scalability, since alternative classifications require the permutation of the subjects in the subject tree.
- Some of the early MOM products pioneered this approach

Type-Based Pub/Sub

- Type-based pub/sub uses path expressions and subtype inclusion tests to select notifications.
- Through multiple inheritance the subject tree can thus be converted into a type lattice with multiple rooted paths to the same node.
- This approach circumvents some of the limitations of simple subject hierarchies.
- Used in Hermes

Topic-based Pub/Sub

- Topic-based pub/sub is the approach associated with the Java Messaging Service (JMS)
- Topic-based pub/sub uses channels that include filters.
- Filtering is done through Boolean predicates defined on the envelope of the notification.
- While filters that can be expressed are quite flexible their expressiveness is limited by the fact that they can only be defined over the metadata contained in the header.



Content-based Pub/Sub

- Content-based pub/sub extends matching to the content of the body of the message rather than limiting it to the header.
- Expressiveness of this approach is determined to a large extent by the data model used for the content and the corresponding formalism for expressing the filter predicates.
- Systems have been proposed based on simple template matching (JEDI), filter expressions on name/value pairs (REBECA) and Xpath expressions on XML documents (Franklin VLDBJ, Rebeca reimplementation).
- A tacit assumption is the existence of a common name space and context used by publishers and subscribers. This is often the case in small systems or subsystems of larger systems but rarely in large, heterogeneous environments.

Content-Based Publish/Subscribe

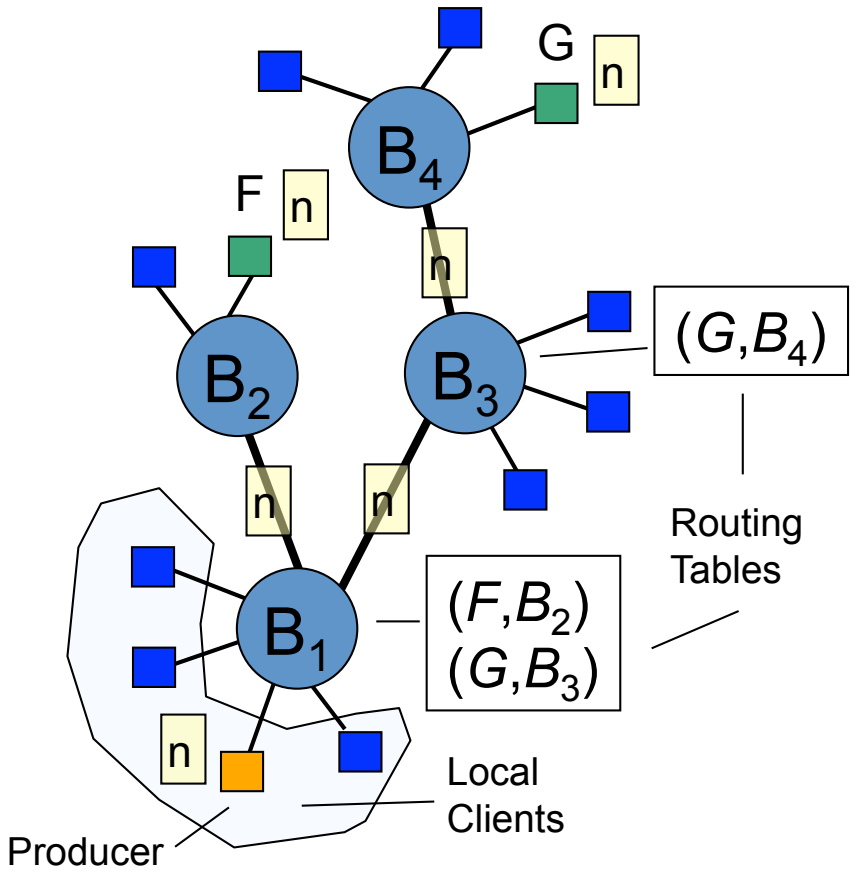
- A *content-based filter F*
  - is a predicate on the content of notifications
  - induces the set of matching notifications $N(F)=\{n \mid F(n) = true\}$
- Content-Based filtering is flexible but complex
- Centralized implementations not scalable to wide-area scenario
- powerful distributed infrastructure required

Problems derived from scale

- Flooding of notifications is not an applicable solution ==> need strategies for filter placement to optimize bandwidth and size of routing tables
- Need to cache/store semi-composed events
- Need to scope events and subdivide event space

Content-Based Routing

- Cooperating brokers
  - Local clients
  - Notification forwarding
  - Filter-Based Routing Tables
- Tradeoff:
  Network resource waste vs. filtering overhead (processing and delay)

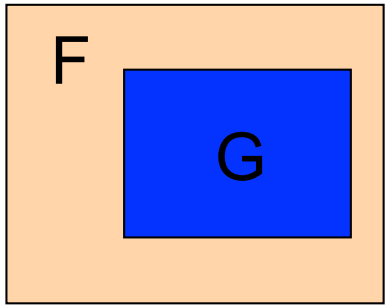


Content-Based Routing II

- Size of routing tables crucial for scalability
  global knowledge about all active subscriptions not feasible
- Solution: reduce size of routing tables and overhead to update them by
  - exploiting similarities among filters
    - identity tests
    - covering tests
  - merging of filters
  - trading accuracy vs. efficiency

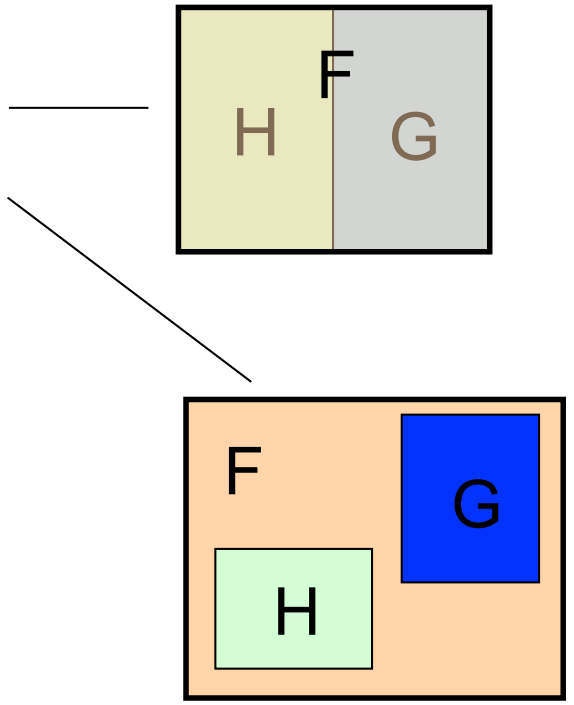
Covering

- Filters can cover each other
  - *F covers G* = *N*(*F*)supersetequal*N*(*G*)
- Covering can decrease
  - size of routing tables
  - filter forwarding overhead



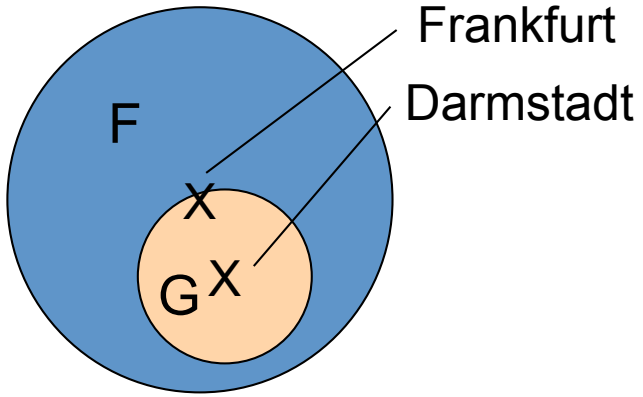
Merging of Filters

- Filters can be merged
  - *perfect* $N(F)=N(G) \vee N(H)$
  - *imperfect* $N(F) \geq N(G) \vee N(H)$
- Merging generates new covers
- similar benefits as covering

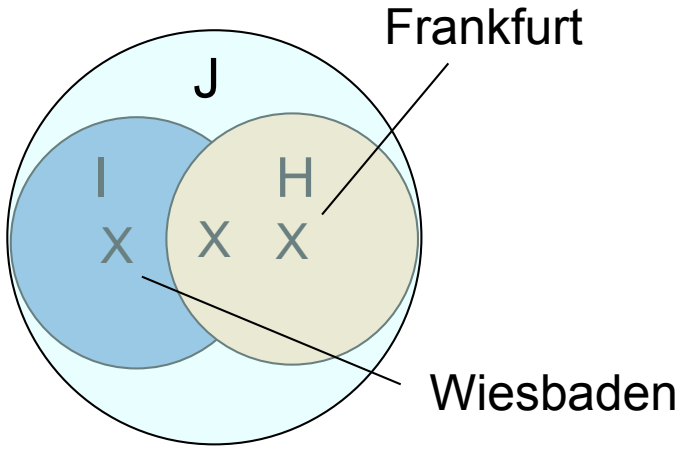


Example: GIS

- *F* = {(*Type* = *TrafficInformation*) ∧ (*Location around*(*Frankfurt*,50km))}
- *G* = {(*Type* = *TrafficJam*) ∧ (*Length* = 5km) ∧ (*Location around*(*Darmstadt*,20km))}
- *F covers G*



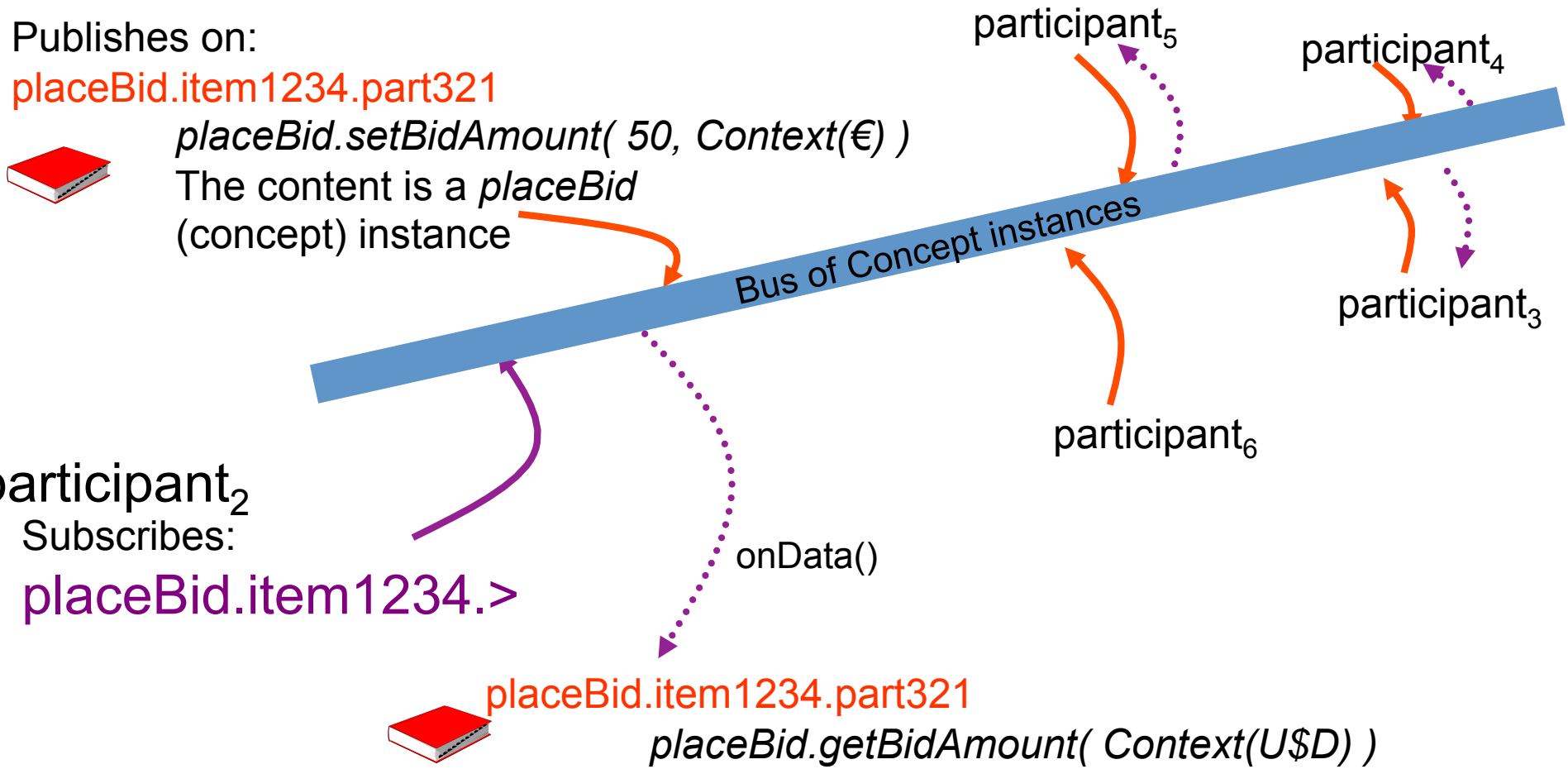
- *H* = {(*Type* = *TrafficJam*) ∧ (*Location around*(*Frankfurt*,40km))}
- *I* = {(*Type* = *TrafficJam*) ∧ (*Location around*(*Wiesbaden*,40km))}
- *H* and *I* can be merged imperfectly to *J*



Concept-based Pub/Sub

- Concept-based pub/sub addresses the problem of implicitly assumed semantics
- Makes the semantics of advertisements, subscriptions and notifications explicit.
- Concept-based pub/sub associates a context with each notification and filter.
- Matching compares first, if the contexts of filter and notification are the same. If true, the normal content-based pub/sub approach kicks in. If the contexts of filter and notification are different, an ontology service is invoked to resolve the different semantics
  - E.g. converting currencies or formats of dates.
- Can be used in conjunction with many of the other pub/sub approaches in large, heterogeneous environments.

Concept-based addressing



Advanced Message Queuing Protocol AMQP

- Emerging standard (financial domain)
- Need for an open standard to provide interoperability between MOM providers
- Wire level protocol encompassing JMS semantics
- Specifies exact semantics of services in its queuing model (P2P, pub/sub, request/response), transaction management, distribution, security, clustering.

Hybrid Architectures

- Real-life systems rarely conform to the pure reference architecture.
- Necessary to combine different invocation styles in the form of hybrid architectures.
- The event consumer may trigger a request-driven interaction to a business process or a service in a SOA, resulting in an event-driven SOA.
- Likewise, an event may trigger a request to a queue to pick up something posted there.

Event detection

- (continuous) signal must be discretized
  - Polling cycle
  - Sensor sends signal periodically
- Event is provided with ID, timestamp, parameter values
- Event object is produced
- Notification is produced by packaging event object into a message

Signal → Event → Event object → Notification

Assumptions

- Duration of events
- Type of clock
- Type of processing environment
- Type of communication channel
- Semantics of detection
- Semantics of composition
- Semantics of consumption

Duration of events

- Events were originally considered to be instantaneous and have no duration
- Instantaneous events can be associated with exactly one point in time
- Time of occurrence vs. time of detection
  - Is there a difference?

Duration of events

- Are events truly instantaneous?
  - Consider a method execution to be an event
  - Is method-event detected when method is invoked or when method returns?
  - Event modifiers: Before, After, (Instead)
    - Before method1 – detects event before method is executed
    - After method 1 – detects event on method return
    - Instead method1 – detects event on method invocation and executes alternative action
  - Event modifiers first used in Postgres
  - Event modifiers combine event detection semantics with control flow

Duration of events

- Events can be long lived (e.g. runway closing)
- Long lived events have a validity interval
  - Begin of interval
  - End of interval
- Long lived events can be modeled as

  BeginInt ∧NOT EndInt ∧ (event-specific-condition)
- Time of definition is often different from Begin of Interval

Type of Clock and Environment

- First generation event systems considered
  - Single central clock
  - Single CPU that could only detect a single event at a time
- With single clock and single event source a total order of events is possible
- With multiple CPUs, events can occur simultaneously
  - No total order
  - How to disambiguate simultaneous events

Type of communication channel

- Instantaneous communication – cero latency between detection and consumption
- Channel with bounded latency – there is a finite and upward bounded delay between event detection and event consumption
- Unbounded delay – delay is finite and unbounded

Semantics of composition

- When composing an event, there is a finite time between the occurrence of the first event in the composition and the occurrence of the last event
- So-called detection-based semantics consider
  - Simple events occur instantaneously
  - Composite/complex events occur when the last event completing a composition occurs

IBM ; Sun

IBM occurs at 10:00

Sun occurs at 11:00

IBM ; Sun occurs at 11:00

Interval semantics

- Interval semantics consider the duration of the interval between the first and the last event in a composition

  IBM ; Sun

  IBM occurs at 10:00

  Sun occurs at 11:00

  IBM ; Sun occurs at [10:00,11:00]

Example composition interval vs. detection semantics

If Dow Jones Industrial Average (DJIA) increases by 5% followed by a 5% price increase of Sun Microsystems (Sun) and a 2% increase in IBM (IBM), trigger an action

IBM occurs at 10:00

Sun occurs at 11:00

DJIA occurs at 10:30

Composite event: DJIA ; (Sun $\wedge$ IBM)

Under detection semantics → TRUE

Under interval semantics → FALSE

Why???

Example composition interval vs. detection semantics

Composite event: DJIA ; (Sun ∧ IBM)

This is a double composition. (Sun ∧ IBM) is a composite event that will be timestamped as having occurred at 11:00 , the time the IBM primitive event was detected

Since DJIA was detected at 10:30, under detection semantics DJIA occurred before the composite event (Sun ∧ IBM)

Under interval semantics, (Sun ∧ IBM) occurred during the interval [10:00,11:00] and DJIA at 10:30 and the sequence does not evaluate TRUE

Semantics of event consumption

- Event instances can be consumed, i.e. used in compositions, in different orders:
  - Combine the most recent instances that fit a type
  - Combine the instances in chronological order
  - Consume all instances within a (sliding) window
  - Consume all instances up to a cut-off event and start with a clean slate

Simple event algebras

- The first event algebras were quite simple
  - Single event source
  - Single clock
  - Total order of events was assumed
- All event algebras of the first generation fall under this category:
  - HiPAC, SNOOP, REACH, ADAM, SAMOS, etc.*
- Events have associated parameters (attributes of the event object)
- Will assume an ECA rule environment in discussion

Ref. to Event Algebras

- Dayal, U., Buchmann, A., McCarhty, D.; “Rules are Objects Too”, 2nd Intl. Workshop on Object Oriented Databases, Bad Münster am Stein, Aug. 1988, LNCS
- Branding et al, “Rules in an open system: the REACH Rule System”, Proc. 1st Intl. Workshop on Rules in Database Systems, Edinburgh, Scotland, Sept. 1993
- Chakravarthy, S., Mishra, D.; “Snoop: an expressive event specification language for active databases”, IEEE Data and Knowledge Engineering, 14 (10), 1994
- Díaz, O., Paton, N., Gray, P.; “Rule management in object-oriented databases: a unified approach”, Proc. VLDB 1991, Barcelona
- Gatziu, S., Dittrich, K.; “Events in an Active Object Oriented Database System”, Proc. 1st Intl. Workshop on Rules in Database Systems, Edinburgh, Scotland, Sept. 1993

Basic Event Composition - HiPAC

- Three constructors
  - Disjunction
  - Sequence
  - Closure

Basic Event Composition – HiPAC - Disjunction

- $E = (E_1 \mid E_2)$
  - Raised when either E_1 or E_2 is raised
  - Parameters of either E_1 or E_2 bound to E
  - Parameters of non-occurring event set to null
- Useful for triggering rule applicable to many situations
  - E.g. relative distance check fired by movement of object A or B

Basic Event Composition – HiPAC - Sequence

- $E = (E_1 ; E_2)$
  - E is signaled when E_2 is raised, provided E_1 has been raised before
  - Parameters are the union of E_1 and E_2
  - Essential for composing events for flow control rules

Basic Event Composition – HiPAC – Closure

- E_1^*
  - Used to fire rule once considering the cumulative effect of multiple operations raising the event E_1
  - For definiteness another terminating event needed
    - $E = (E_1^* ; E_2)$
  - E.g. evaluate average once after all components have been modified

Additional Event Composition - SAMOS

- Same event constructors as in HiPAC
  - Closure with different semantics
    - Relative to time interval raised first time primitive event is detected
- Extends the HiPAC event algebra with
  - Conjunction
  - Negation
  - History

Additional Event Composition: SAMOS – Conjunction

- (E_1, E_2)
  - Raised when E_1 and E_2 are detected
  - Order of detection is irrelevant
  - Composition of arguments is the union

Additional Event Composition: SAMOS – Negation

- NOT E,t_1-t_2
  - Raised when event is not detected within time interval
  - Modifier ANY allows firing when none of the registered events has occurred within time interval

Additional Event Composition: SAMOS – History

- Times(n, E)
  - Counts number of occurrences of primitive event *E*
  - Raised when specified number of *n* occurrences is detected
- Times($[n_1\text{-}n_2]$,E,$[t_1\text{-}t_2]$)
  - Fires if E occurs at least n_1 and at most n_2 times within time interval
  - Raised at end of time interval if condition is met

SNOOP Extensions

- Also evolved from the HiPAC event algebra
- Additional Operators
  - Conjunction
  - Aperiodic

SNOOP Extensions - Conjunction

- Any(i, E_1, E_2, …, E_n) where i <= n
  - Fires when i distinct events from e_1..e_n are detected

- All(E_1, E_2, …, E_n)
  - Fires when an occurrence of each event in the list is detected

SNOOP Extensions – Aperiodic

- $A(E_1, E_2, E_3)$
  - E_1 and E_3 define interval
    - this interval is not limited to temporal events
  - Fires whenever E_2 is detected in interval
- $A^*(E_1, E_2, E_3)$
  - Similar semantics to HiPAC's E* within interval

Event Algebra Issues

- The more powerful the algebra
  - The more complex to implement and understand
- More complexity
  - Loss of light-weight nature of events
  - Monitoring becomes the bottleneck
  - Semantic ambiguities

Event Algebra Issues – Semantic Ambiguities

- For instance sequence E_1 ; E_2
  - With event occurrences

    E_1, E_1', E_2
  - Which occurrence of E_1 should be composed with E_2?
  - If the second event of the sequence does not occur, how long should the other occurrences be held?

Event Algebra Issues – Semantic Ambiguities (cont.)

- Solution I
  - Fixed semantics
    - Last occurrence prevails
    - First occurrence prevails (FIFO)
    - But semantics depend on the application

Event Algebra Issues – Semantic Ambiguities (cont.)

- Solution II
  - Contexts (SNOOP/Sentinel)*
    - Four possible contexts (application dependent)
      - Recent
      - Chronicle
      - Continuous
      - Cumulative
    - Also known as consumption modes

- Chakravarthy et al. “Composite events for active databases: semantics, contexts and detection, VLDB 1994, Santiago de Chile

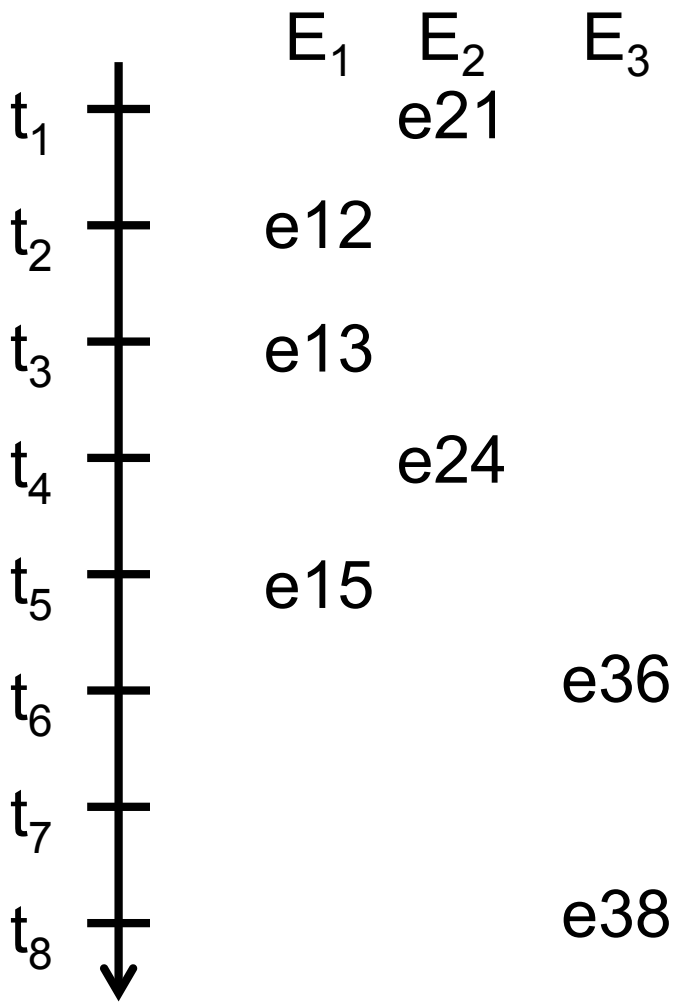
Event Algebra Issues – Semantic Ambiguities (cont.)

- Solution II (cont.)
  - Consumption modes
    - Recent: most recent occurrence that allows to proceed to the next step used in composition
      - E.g. sensor monitoring
    - Chronicle: primitive events are consumed in chronological order by composer
      - E.g. flow-control applications

Event Algebra Issues – Semantic Ambiguities (cont.)

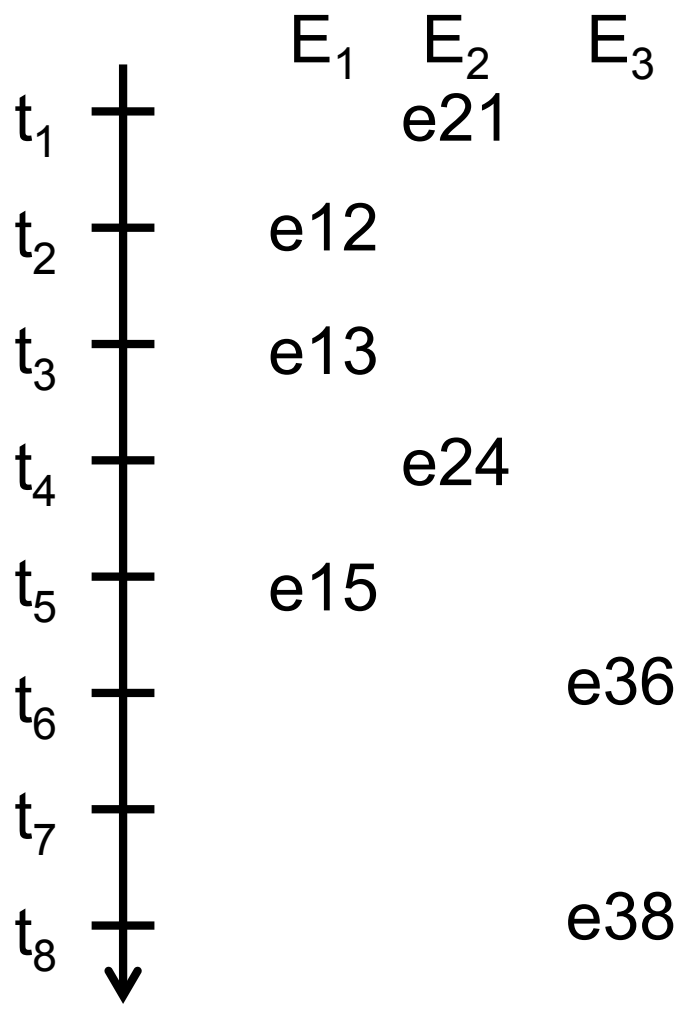
- Solution II (cont.)
  - Continuous: each occurrence start a new “window” of evaluation continuous monitoring
    - E.g. finance applications (tracking the DAX over 2 hours)
  - Cumulative: all occurrences are used up to the point composite event is raised
    - E.g. constraints on aggregates based on updates within a transaction

Consumption Modes - Example



A = any(2, E1,E2) ; E3

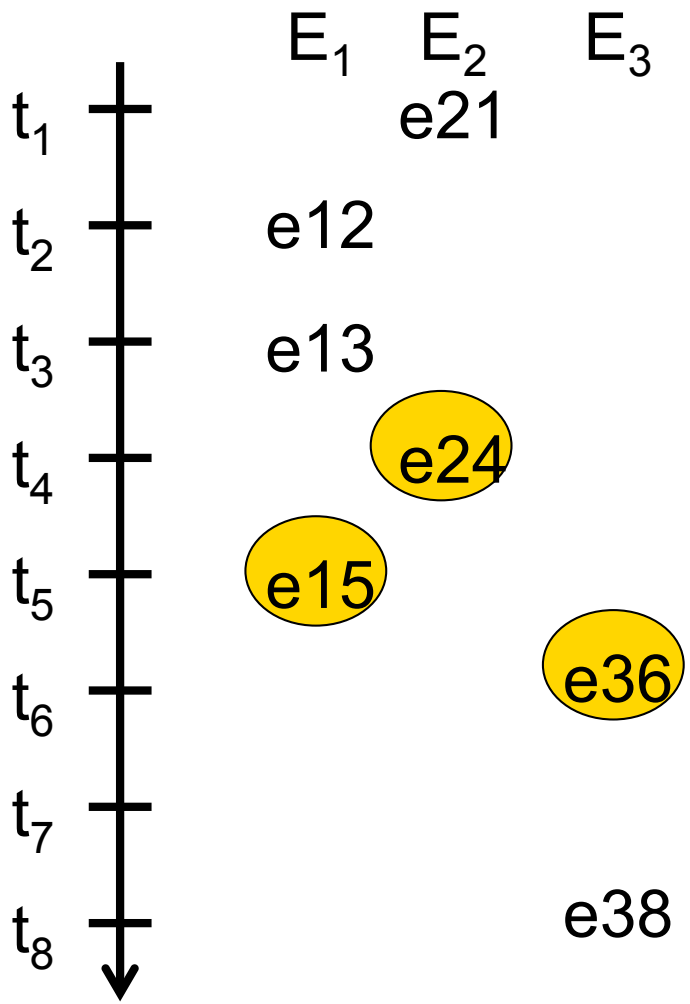
Consumption Modes – Example Recent



A = any(2, E1,E2) ; E3

Recent:
A= ?

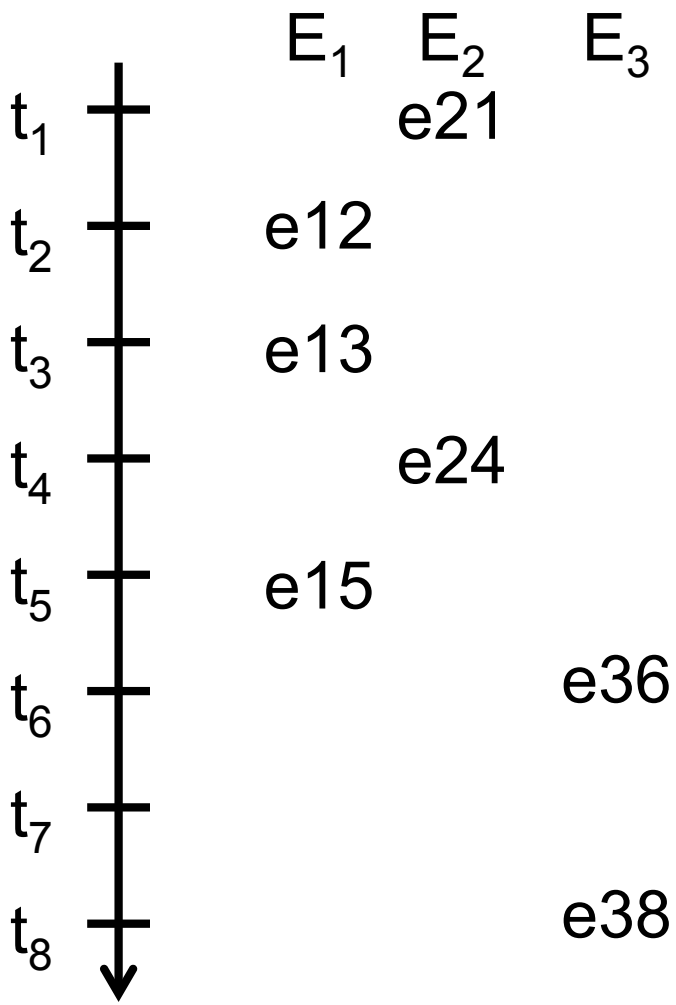
Consumption Modes – Example Recent



A = any(2, E1,E2) ; E3

Recent:
A= e15 e24 e36

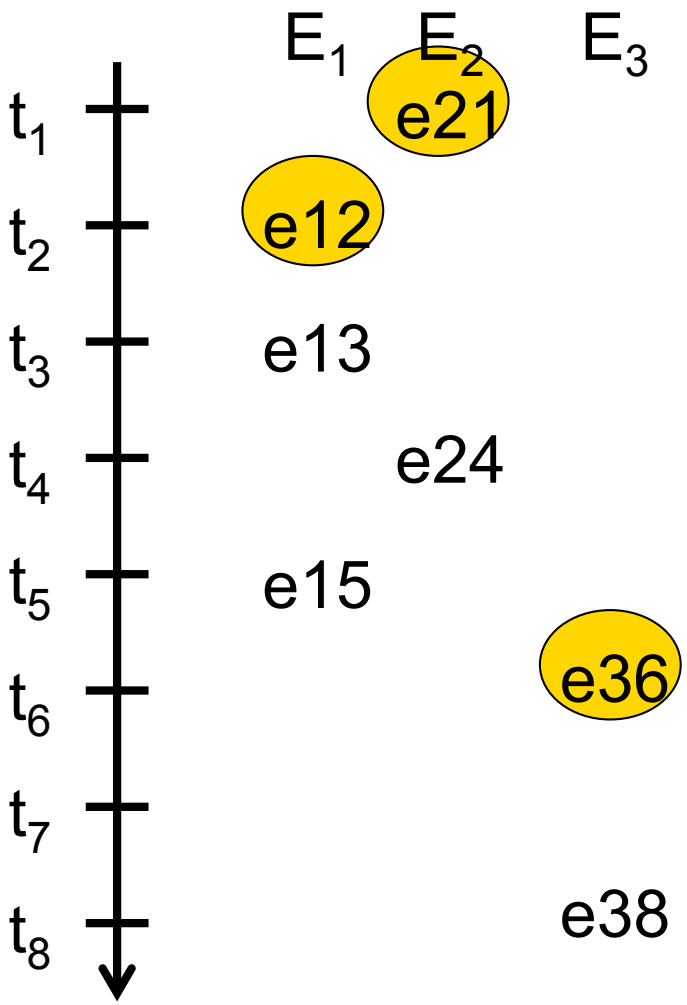
Consumption Modes – Example Chronicle



A = any(2, E1,E2) ; E3

Chronicle:
A= ?

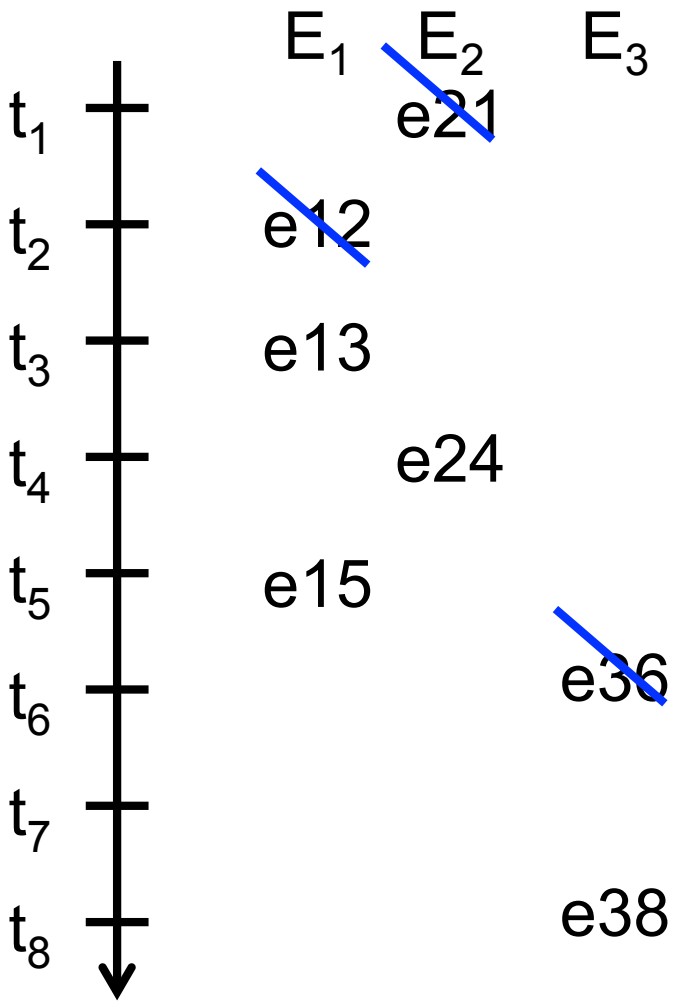
Consumption Modes – Example Chronicle



A = any(2, E1,E2) ; E3

Chronicle:
A= e12 e21 e36

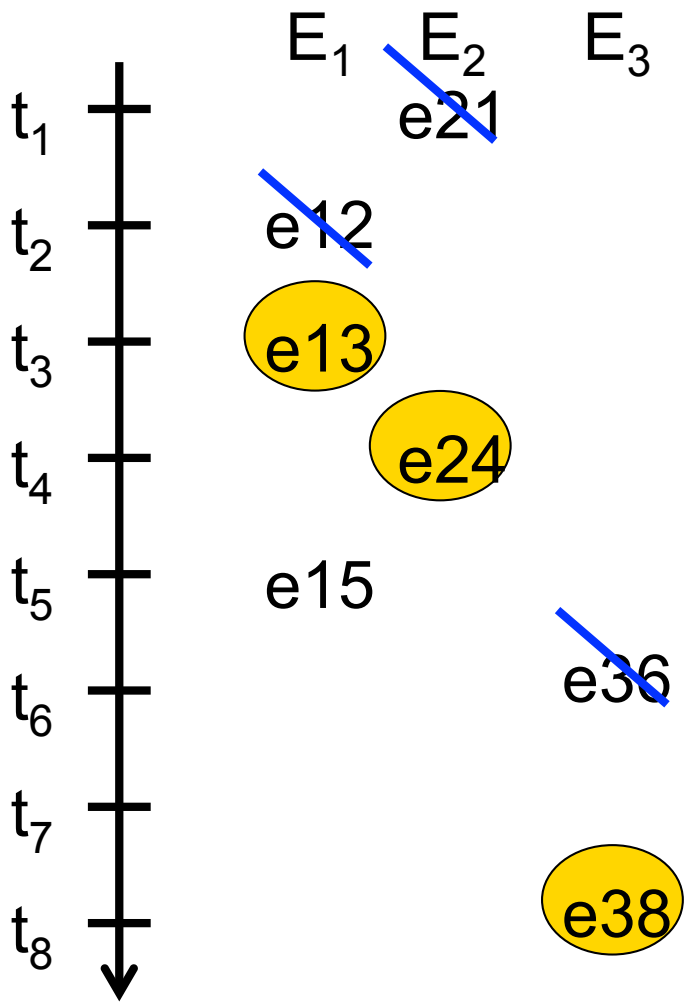
Consumption Modes – Example Chronicle



A = any(2, E1,E2) ; E3

Chronicle:
A= e12 e21 e36

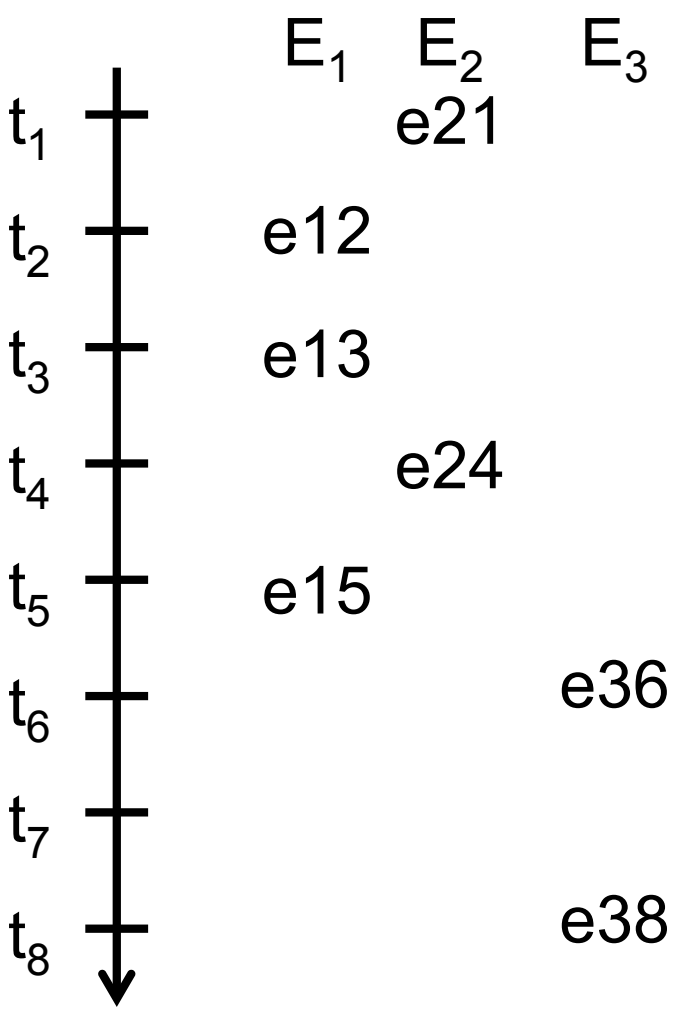
Consumption Modes – Example Chronicle



A = any(2, E1,E2) ; E3

Chronicle:
A= e12 e21 e36
A= e13 e24 e38

Consumption Modes – Example Continuous

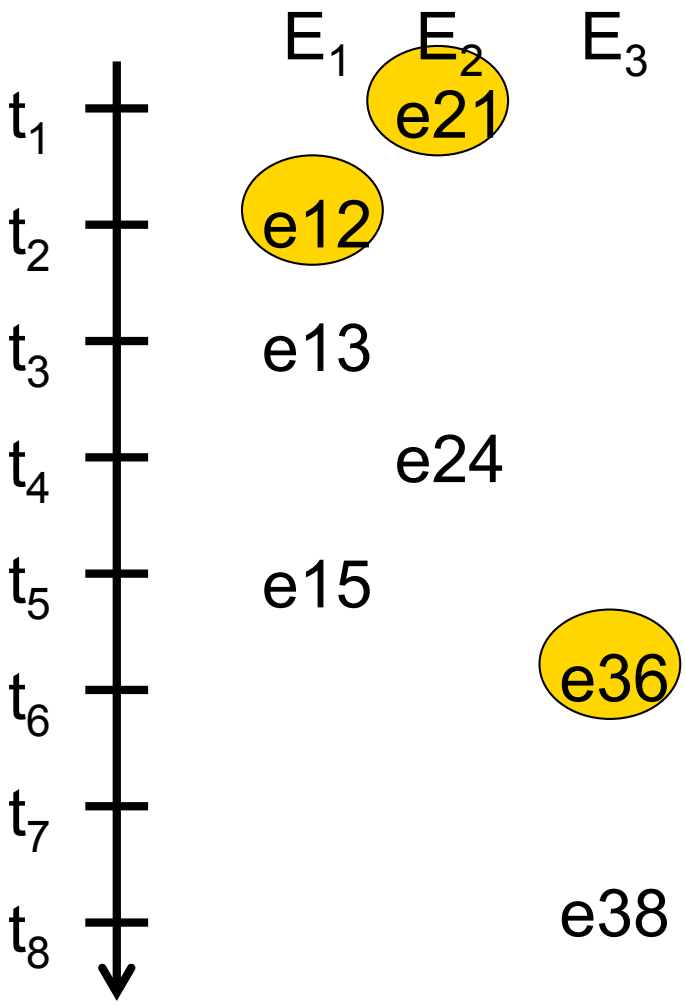


A = any(2, E1,E2) ; E3

Continuous:
A= ?

Each e1 and e2 starts an event composition

Consumption Modes – Example Continuous

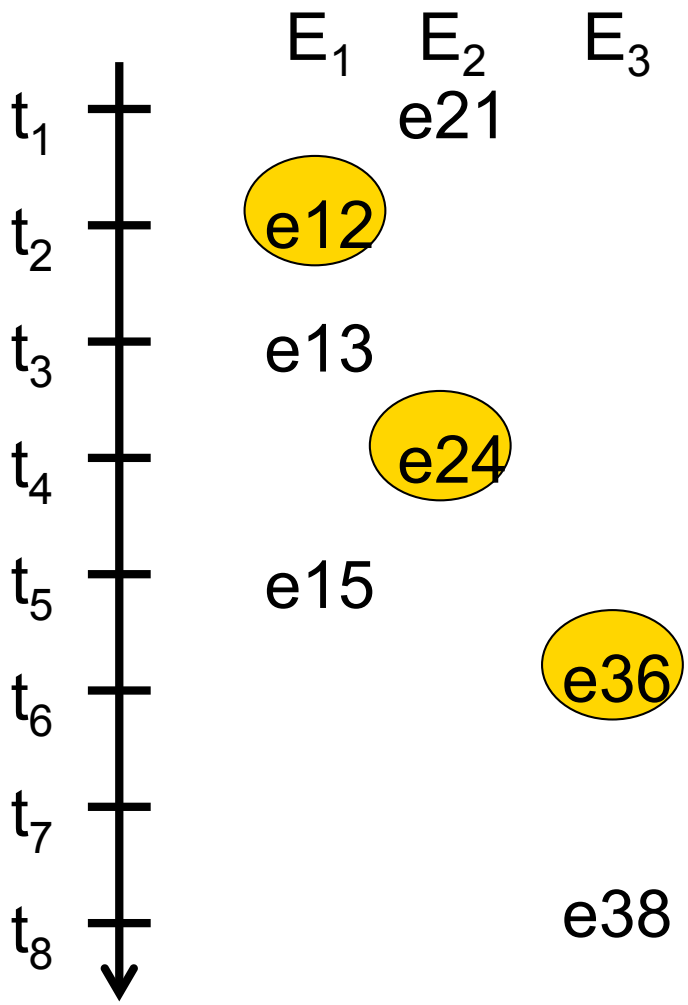


A = any(2, E1,E2) ; E3

Continuous:
A= e21 e12 e36

Each e1 and e2 starts an event composition

Consumption Modes – Example Continuous

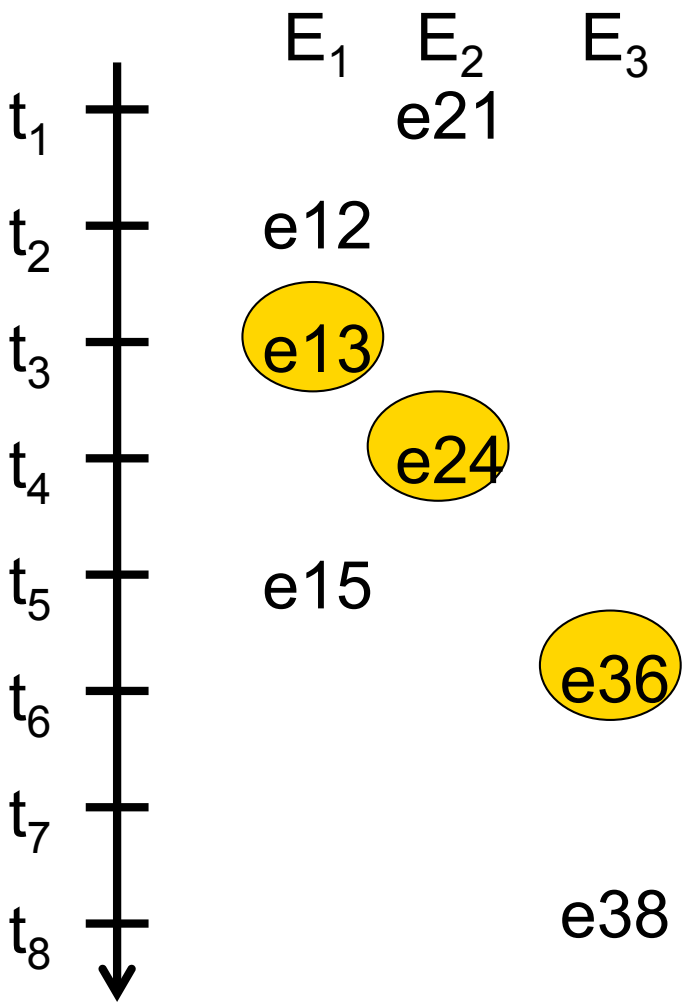


A = any(2, E1,E2) ; E3

Continuous:
A= e21 e12 e36
A= e12 e24 e36

Each e1 and e2 starts an event composition

Consumption Modes – Example Continuous

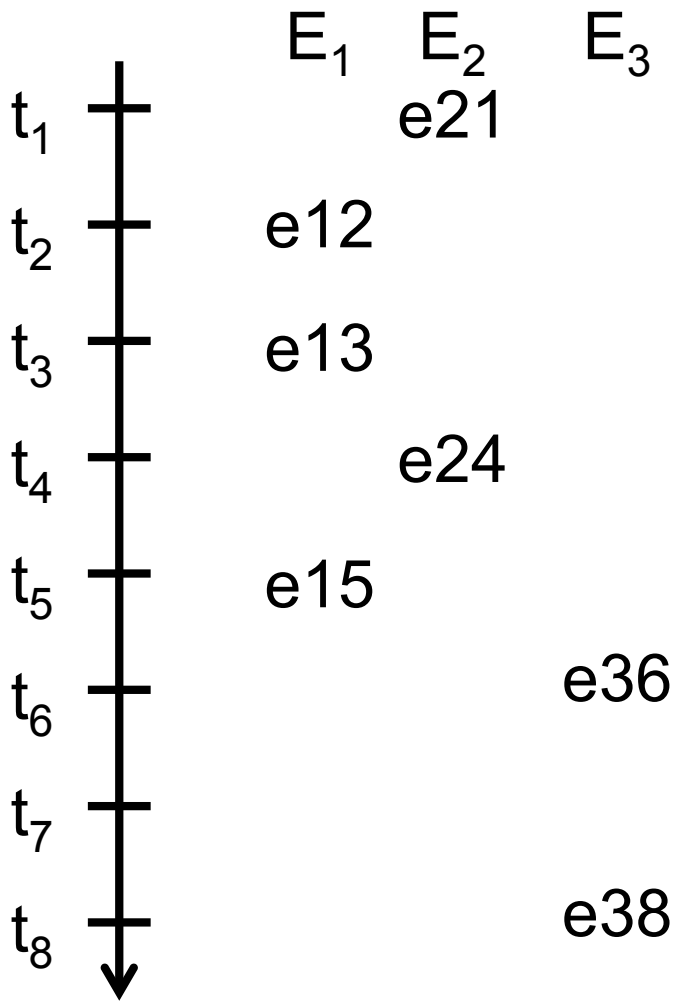


A = any(2, E1,E2) ; E3

Continuous:
A= e21 e12 e36
A= e12 e24 e36
A= e13 e24 e36

Each e1 and e2 starts an event composition

Consumption Modes – Example Cumulative

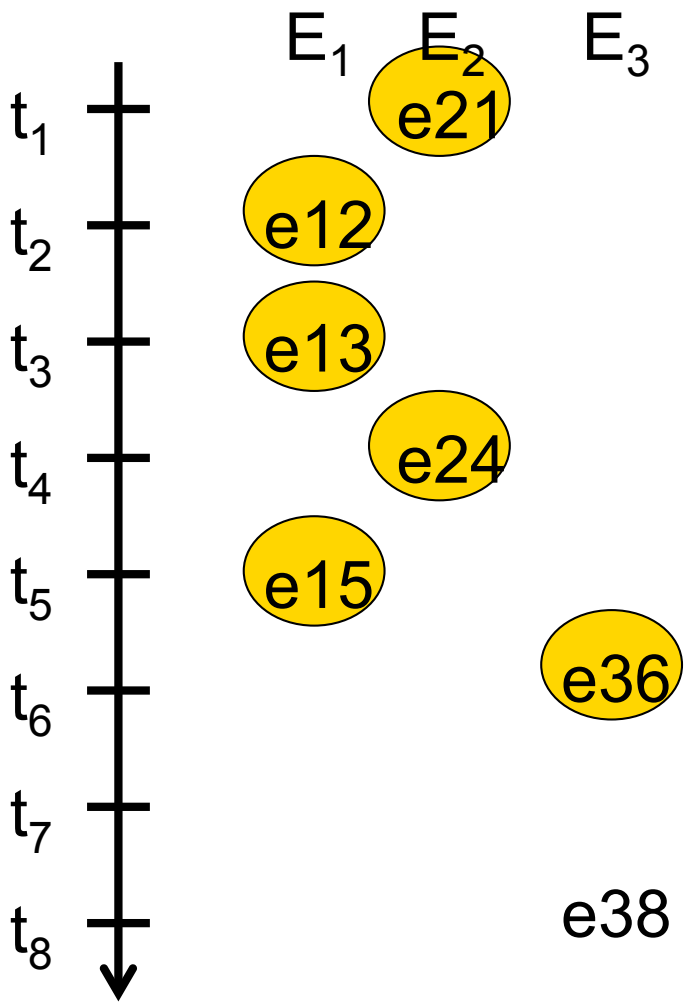


A = any(2, E1,E2) ; E3

Cumulative:
A= ?

Each partial event is accumulated until A is raised

Consumption Modes – Example Cumulative



A = any(2, E1,E2) ; E3

Cumulative:
A= e12 e13 e15 e21 e24 e36

Each partial event is accumulated until A is raised

Interval semantics with single clock

- Will look at interval composition semantics*
- Assume equidistant discrete time domain having 0 as origin and each time point being a non-negative integer
- The granularity of time is domain-specific
- Event occurrence is denoted O(E[t1,t2]) with t1 being the start time of the interval and t2 the end time
- Primitive events are considered to be detected atomically such that O(E[t,t']) with t=t'

Note: even though interval semantics are proposed, primitive events have cero duration (e.g. a method invocation would also have cero duration)

* Adaikkalavan, R., Chakravarthy, S.; "SnoopIB: Interval-based event specification and detection for active databases", Data and Knowledge Engineering 59 (2006) 139-165

Interval composition semantics

- Composite event consists of multiple events
  - Composite event occurs over an interval
  - Composite event is detected when last constituent event is detected

  ➔Occurrence and detection semantics are separated *

  - Composite events have
    - an initiator event,
    - a terminator event
    - a detector event

*Galton, A., Augusto, J.; “Two approaches to event definition”, 13th Intl. Conf. on Database and Expert Systems Applications, Aix en Provence, 2002

Composite events under interval semantics

- Binary Snoop operators — E1 op E2 [t1,t2]
- Ternary Snoop operators — op (E1,E2,E3), [t1,t2]

- t1 is the start time of the initiator, t2 is the end time of the detector/ terminator as well as the composite event

- Conjunction O(E1 AND E2, [t1,t2])
  Composite event occurs when both E1 and E2 occur in any order
  either E1 or E2 can act as initiator or terminator

Composite events under interval semantics

- Disjunction O(E1 OR E2, [t1,t2])

  Occurs when either E1 or E2 occurs, the event that occurs acts both as initiator and terminator

- Negation O(NOT (E3) [E1;E2], [t1,t2])

  defines the non-occurrence of event E3 in the interval marked by the end time of E1 and the start time of E2 which is the interval [t1,t2]

Composite events under interval semantics

- Aperiodic event operator O(A(E1,E2,E3),[t1,t2])

  Event is signaled every time the event E2 is detected within the interval denoted by E1 and E3

  E1 is the initiator, E2 is the detector, and E3 is the terminator

  Time of occurrence is the time of detection of E2

  Note: paper is wrong (typos in the indices)

Composite events under interval semantics

- Cumulative Aperiodic $O(A^*(E_1,E_2,E_3),[t_{s2},t_{e2}]$
- Detector event is accumulated in the interval formed by the events E_1 and E_3
- Occurrence time is the time between the first detection of an instance of E_2 in the interval and the detection of the last instance of E_2 in the interval

Note: The semantics of this composition is tricky, since E_2 is detected every time it occurs, but the cumulative detection is only possible when E_3 occurs. Otherwise we can't say if a detection of E_2 is the last in the interval. The time of occurrence is given by the interval $[t_{s2},t_{e2}]$ but the time of detection is t_3

Composite events under interval semantics

- Periodic events $O(P(E_1,E_2,E_3),[t_2,t_2])$
- Periodic events are signaled every time a fixed, finite amount of time elapses
- E_2 should be a time string while E_1 and E_3 can be arbitrary events

- Cumulative variant of Periodic $O(P^*(E_1,E_2,E_3),[t_{s2},t_{e2}])$
- Signaled only when E_3 occurs

Note: in the cumulative version of periodic we can see clearly that it makes sense to view event compositions as traces (or histories) of events

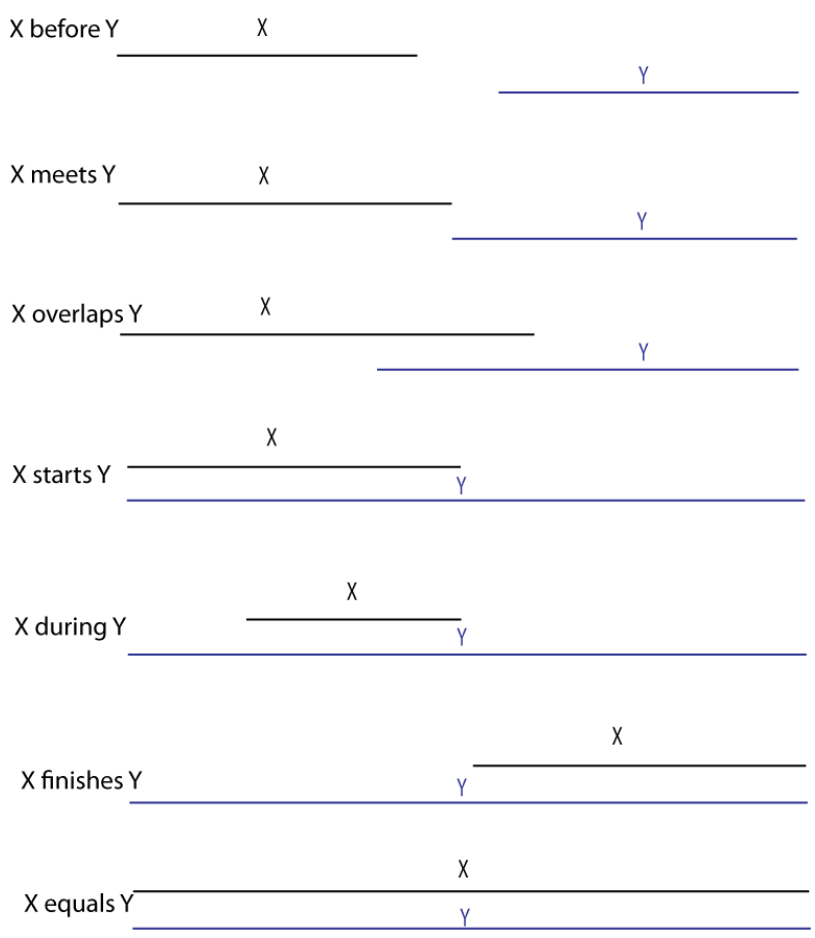
Composite events under interval semantics

- PLUS operator $O(PLUS(E_1,E_2)\ ,\ [t_2,t_2])$
- E_2 is a time event and E_1 is an arbitrary event expression
- Event is signaled when E_2 is detected after an arbitrary event E_1 is detected

Discussion point: the PLUS operator is not needed imho since it can be expressed as any other sequence

Event overlap

- If we accept that events can have a duration, then the temporal relationship between events becomes more complex
- Allen's interval logic defines the valid relationships
- 13 possible combinations, reduce to 7 because of symmetry



Non-overlapping events

- When events are not allowed to overlap, only 3 relationships are possible (reducible to 2 because of symmetry):
- X before Y, and X meets Y are possible

X before Y

X meets Y

Traces of events

- In monitoring situations we are interested in traces or histories of events
- Traces can be built independently of the semantics (point or interval)
- Under point semantics the trace or history of an event of type E_1 would look like

  $E_1[H] = e_1^1[3] , e_1^2[4] , e_1^3[7]$

  subscript is event type, superscript is occurrence ordinal and value in square brackets is timestamp
- Under interval semantics (overlapping) it could be

  $E_1[H] = e_1^1[1,3] , e_1^2[2,4] , e_1^3[5,7]$

Compositions with interval semantics

- Compositions can be made under different consumption modes
- Unrestricted consumption mode will generate all possible compositions matching the defining pattern
- Unrestricted composition requires large amounts of storage, additional computation and generates many useless events
- Better to apply the known consumption modes, especially chronicle, recent and sliding window

Compositions with interval semantics

$E_1[H] = e_1^1[3,5] , e_1^2[4,6] , e_1^3[8,9]$

$E_2[H] = e_2^1[1,2] , e_2^2[7,10] , e_2^3[11,12]$

Sequence under unrestricted consumption mode

$\{(e_1^1,e_2^2) , [3,10] , (e_1^2,e_2^2) [4,10] , (e_1^1,e_2^3) [3,12], (e_1^2,e_2^3) [4,12] , (e_1^3,e_2^3) [8,12]\}$

Sequence under recent consumption mode

$\{(e_1^3,e_2^3) [8,12]\}$

Ordering of events

- Ordering of events is basic functionality
- Order is typically considered based on time
- Some spatial applications or location-sensitive apps. order according to spatial relationships
- Allen's relationships apply to time as well as space
- Causality as ordering criterion
  - Lamport's happened before relationship a→b
  - If neither a → b nor b→a then events are concurrent and independent
  - Logical clocks assign running integers to events
  - Causality is compatible with logical clocks

Logical Clocks

- **Problem**: What is **ordering?**
  - Is agreement on time needed (clock sync) or just the relative order of occurrence
- The **happened-before relation** on the set of events in a distributed system is the relation satisfying:
  - If **a** and **b** are two events in the same process, and **a** comes before **b,** then **a** → **b**. (a happened before b)
  - If **a** is the sending of a message, and **b** is the receipt of the message: **a** → **b**.
  - If **a** → **b** and **b** → **c**, then **a** → **c**. (transitive relation)

- If two events, **x** and **y,** happen in different processes that do not exchange messages, then they are **concurrent**.

- This mechanism introduces a **partial ordering** of events in a system with concurrently operating processes.

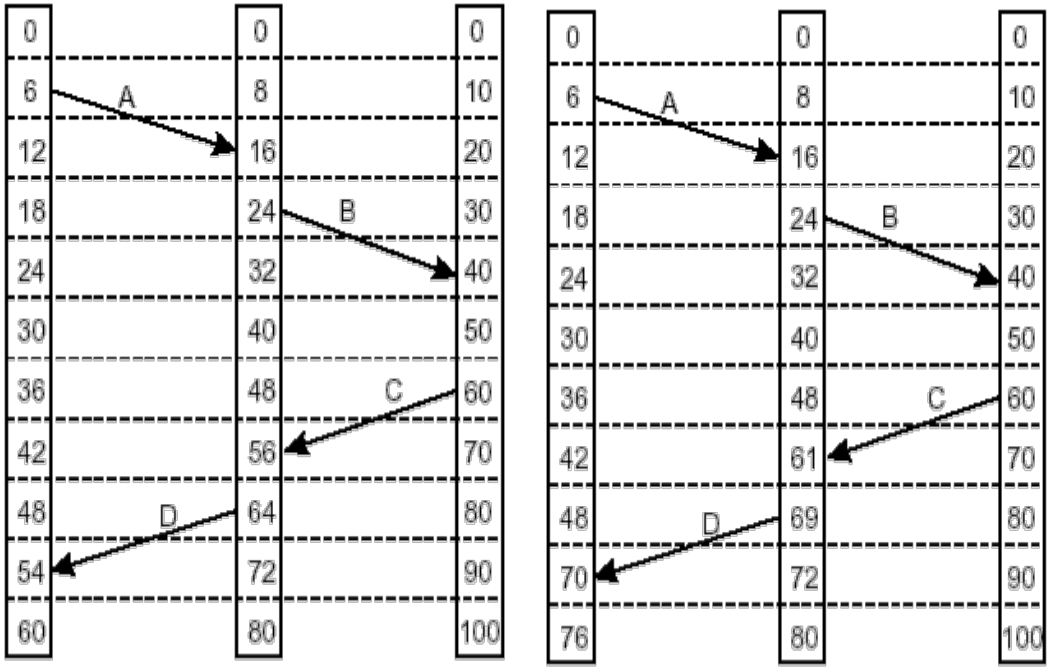
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Logical Clocks

- **Problem:** How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?
- **Solution:** attach a timestamp $C(e)$ to each event e, satisfying the following properties:
- **P1:** If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$
- **P2:** If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$
- **Problem:** How do we attach a timestamp to an event when there's no global clock? ➔ maintain a **consistent** set of logical clocks, one per process.

Logical Clocks – Lamport's algorithm

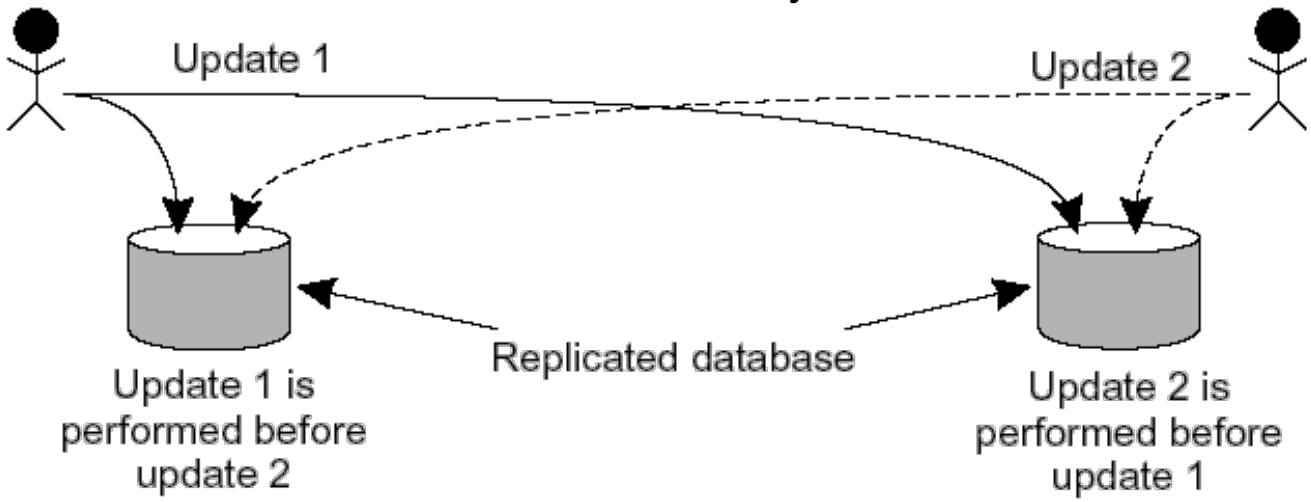
- Lamport's algorithm well known from other lectures
  - (KN1, KN2, Middleware)
- **Question**: can it cope with clock drift? Can it be used to correct the drift?
  - Can it implement Global Time?
- Total Ordering
  - Each message carries sender's clock
  - Upon arrival the receiver's clock = sender's clock +1
- Global Time:
  - between every two events the clock ticks at least once
  - events don't occur simultaneously (attach process number to event)



Situation: 3 Clocks + Drift **Solution: Lamport Clocks**

Example: Totally-Ordered Multicast

- **Problem: G**uarantee that concurrent updates on a replicated database are seen in the same order by all replicas. Example with 2 replicas:
  - Update1: Process P1 adds €100 to an account (initial value: €1000)
  - Update2: Process P2 increments account by 1%



- **Result: i**f NO synchronization → Replica1 = €1111, Replica2 = €1110
  - **Solution**: upon receipt of msg → place in queue that is ordered according to timestamp, every recipient (incl. sender) sends an ack. Eventually all queues converge. A message is only executed if it is at the head of the queue and was confirmed by all other recipients.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Vector Timestamps

- Lamport's algorithm does not capture causality:
  - If Timstamp(A) < Timstamp(B) → does this imply that A happen before B?
  - Consider posting messages on a forum and replying
    - Is the posting of a new message B a result of message A?
- Causality: Reaction to msg A should always follow the receipt of A
  - If msg A and msg B are independent → order of delivery irrelevant
- Vector Clocks [Fidge]
  - A vector timestamps VT(E), VT(F) for events E,F → VT(E) < VT(F) if causal dept.
  - Each process P has a vector V
    - V[i] – number of events occcured at P
    - If V[j] = K → P knows that K events have occured
    - Important to know how many messages at other nodes/process P had seen when he sent his message → send vectors with message
    - Receiving process knows now for how many events at a node it must wait to be in the same state as the sender

Colin Fidge, "Logical Time in Distributed Computing Systems", *IEEE Computer*, Vol. 24, No. 8, pp. 28-33, 1991

Vector Timestamps

The Fidge logical clock is maintained as follows:

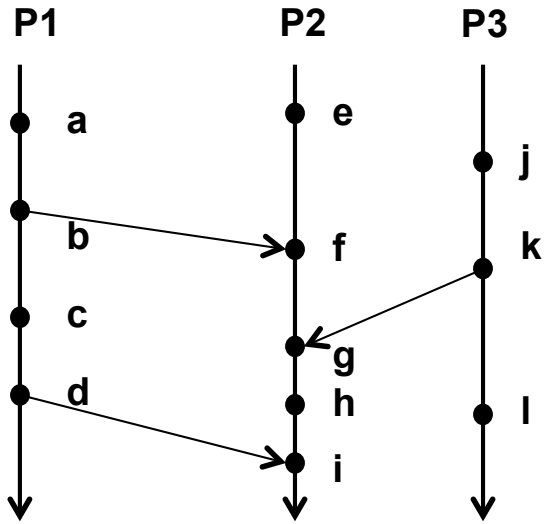
1. Initially all clock values are set to the smallest value.
2. The local clock value is incremented at least once before each primitive event in a process
3. The current value of the entire logical clock vector is delivered to the receiver for every outgoing message.
4. Values in the timestamp vectors are never decremented.
5. Upon receiving a message, the receiver sets the value of each entry in its local timestamp vector to the maximum of the two corresponding values in the local vector and in the remote vector received.
6. The element corresponding to the sender is a special case; it is set to one greater than the value received, but only if the local value is not greater than that received.

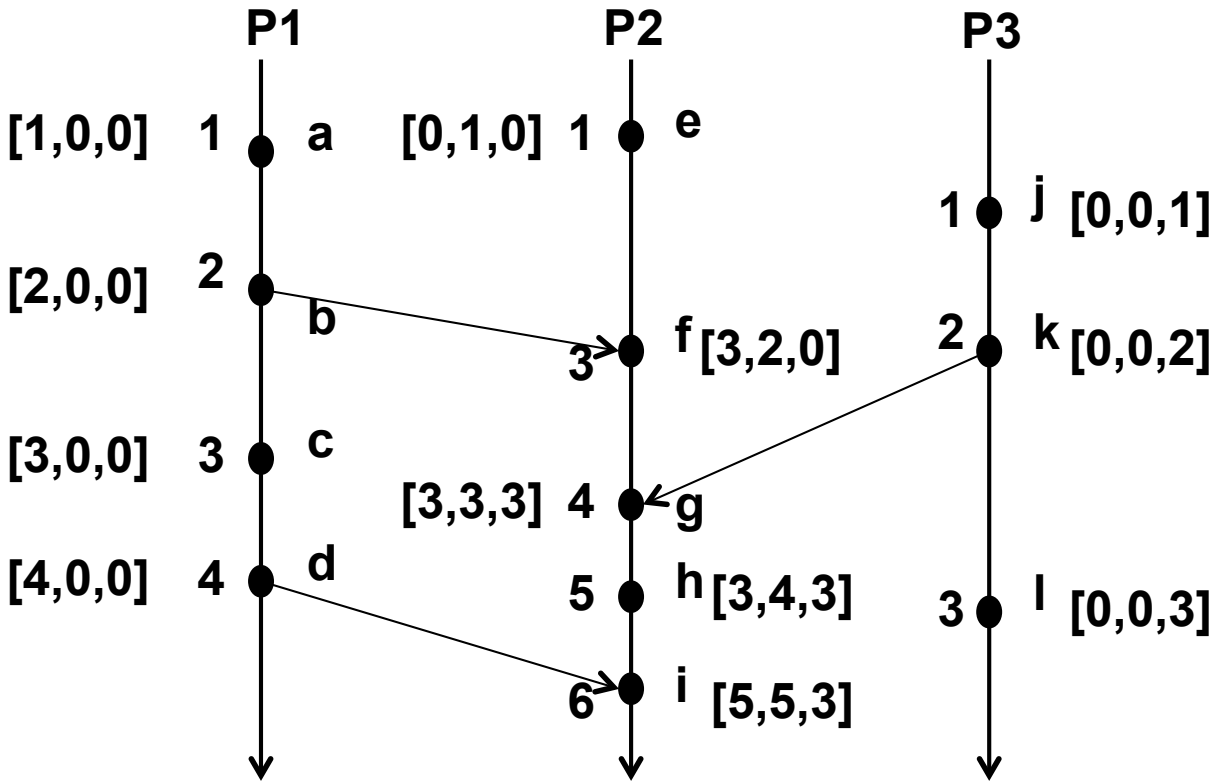
Colin Fidge, "Logical Time in Distributed Computing Systems", *IEEE Computer*, Vol. 24, No. 8, pp. 28-33, 1991

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Vector Timestamps

- Assign the Lamport and Fidge logical clock values for all the events.
  - The logical clock of each process is initially set to 0





Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Vector Timestamps

- The above diagram shows both Lamport timestamps (an integer value) and Fidge timestamps (a vector of integer values) for each event.

- Lamport clocks:
  - 2 < 5 since b → h,
  - 3 < 4 but c <!> g.

- Fidge clocks:
  - f → h since 2 < 4 is true,
  - b → h since 2 < 3 is true,
  - h <!> a since 4 < 0 is false,
  - c <!> h since (3 < 3) is false and (4 < 0) is false.

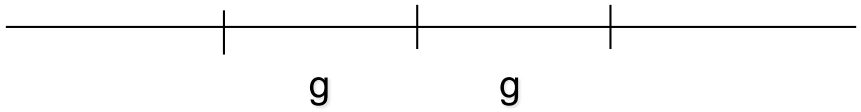
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Ordering of events – vector time

- Vector time characterizes causality (Schwarz & Mattern proposed it same time as Fidge)
- Two events are ordered by vector time iff they are causally dependent
- Shortcomings of logical and vector time:
  - Can’t deal with physical time
  - Can’t deal with causal relations that are established through hidden channels

Ordering of events – 2g precedence

- 2g precedence is a global time approximation introduced by Kopetz and Verissimo
- Assumes that the maximum difference between 2 clocks is δ at any given instant
- The granularity g of the time base should not be smaller than δ, i.e. $g > \delta$
- If the granularity condition is met, then the global clocks cannot overlap
- If two events are at least 2g apart, then it is possible to establish a global total order of events
- If the 2g precedence condition does not hold at all times, we must face partial orderings of events



Event ordering under 2g precedence

- Schwiderski (dissertation 96) proposed a distributed event composer based on an event tree and the 2g precedence model
- Was able to define sequence operation
- Defined concurrency operators
  - Event consumption is non-deterministic for concurrent or unrelated events
- Chakravarthy applied the 2g precedence model to event compositions in distributed environments (ICDE99)

Event ordering under unbounded imprecision

- Local clocks can produce local orderings of events
  - All events detected by the systems with the same clock can be locally ordered
- Imprecision introduced by delays (e.g. between occurrence and detection)
  - Single local logical clock may produce only partial orders when multiple detectors are involved
- Physical time is the time the event actually occurred in the real world
- Reference time (e.g. GPS time) is an artifact and a granular representation of dense physical time
- Clock synchronization depends on the upper bound of the communication delay

Time Service

- Network Time Protocol defines architecture for a time service and protocol to distribute accurate time information in an unmanaged global internet environment
- Participating nodes form logical synchronization subnets organized in strata
- Primary servers at stratum 1 are directly connected to a source that provides accurate UTC reference time (error in microsec range)
  UTC = Coordinated Universal Time
- Servers at stratum 2 synchronize wrt server at stratum 1 and peers at stratum 2, etc.
- Peer selection and calculation of offset (Cristian algorithm)
- NTP provides a reliable error bound (synchronization distance, accounts for skew and estimation inaccuracy)

Timestamping of events

- See “Event Composition in Time-dependent Distributed Systems” C. Liebig, M. Cilia, A. Buchmann, CoopIS'99, Edinburgh, Sept 1999

- Local detection delay: the delay between actual occurrence of the event and its timestamping
- NTP provides a reliable error bound, the synchronization distance
- Accuracy interval with reference time t_{ref} : $I(t_{ref}) = [t_{ref} - \alpha^-; t_{ref} + \alpha^+]$

- Global timestamp

  $ts(e) = [C(t_{det}) +- \alpha]$

  $\alpha = [\ synchdist_{tdet} + ldd;\ synchdist_{tdet}\ ]$

- Accuracy Interval Order

  $I_j=[r_j+-\alpha_j)$ $I_k=[r_k+-\alpha_k)$ $I_j < I_k$ < == > forall s in I_j

  and forall t in I_k : $s < t$ < == > $r_j + \alpha_j^+ < r_k - \alpha_k^-$

  – Partial order based on the accuracy intervals for the events
  – If the accuracy intervals are too large, the order of events cannot be determined
  – Predefined actions should be taken

Ordering of events with Accuracy Intervals

- It can be guaranteed that using accuracy intervals
  - Situations of uncertain event order are detected
  - Events are not erroneously ordered

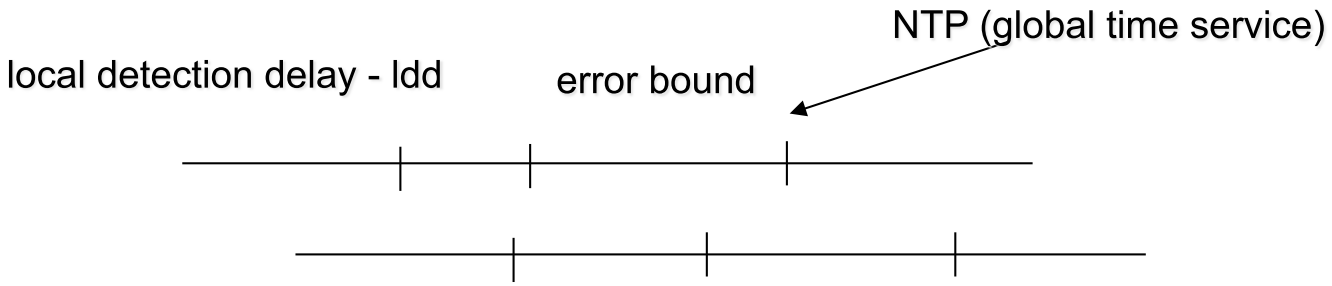
- Time Consistent Order:
  - Accuracy interval order is consistent with physical time order
  - Given events e_j, e_k and

    $ts(e_j) = I_j\ (t_{det}(e_j))$ $\qquad$ $ts(e_k) = I_k\ (t_{det}\ (e_k))$

    then $I_j < I_k \rightarrow t_{occ}(e_j) < t_{occ}(e_k)$

Notification with Imprecision

- With unbounded delays in the communication channel the 2g precedence approach can't be used
- Need to generalize and use Accuracy Interval Ordering
- Problem:
  - if an event has not appeared at the composing node, did it not occur or is it just late?
  - When can composition begin/end without violating the event consumption policies?

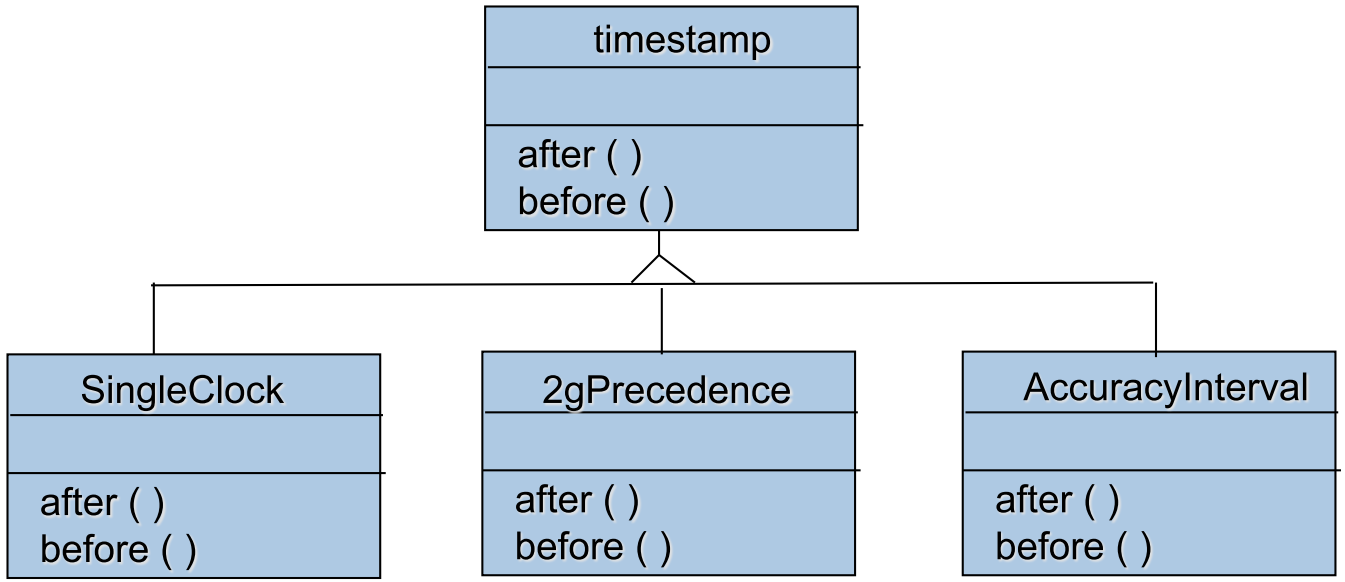


Composing with known accuracy

- By injecting a heartbeat event and requiring the channel to send events in FIFO order we can
  - Know with certainty that no event previously put in the channel has not been delivered
  - The NTP heartbeat event, once read, makes the history stable
  - Events can be ordered according to AIO in the stable past
  - If no ordering is possible, resolution can be attempted at a semantically higher level

Note: For general purpose software (e.g. middleware) it is essential that the system does not misrepresent the achievable level of certainty

Timestamp based ordering – application dependent



- Must provide interval-based time model with explicit (in)accuracy intervals
- Allen‘s interval semantics ==> indeterminacy
- Stable past vs. unstable past and present
- Middleware must expose indeterminacy, resolve through application semantics

Approaches to Event Processing and Composition

- Event processing occurs on streams of event objects
  - Event objects could be represented as
    - Tuples
    - Objects
    - XML documents
    - …
  - The representation of the event objects will determine to some extent the processing of the events
  - We will abstract from the actual event object representation and deal with filters and continuous queries
  - Filters and continuous queries can then be represented in a matching language (StreamSQL or CSQL for tuples, Xpath expressions for XML docs., (continuous)OQL / objects

Query vs. Graph-based Complex Event Detection

- Events can be composed by one of two basically different methods
  - Queries expressed in some from of continuous query language
  - Graphs that are traversed by some form of token and signal the complex event when the proper number and order of tokens has traversed an instance of the graph
- Both forms of complex event detection offer advantages and disadvantages
- We will discuss both

Stream Abstraction

- Event objects arrive continuously and sequentially as a stream
- Streams are usually ordered according to some criterion, often timestamp
- If streams are unordered or events arrive out of order (in a bounded time) special treatment is required
- The rate of the stream is determined by the source and cannot be controlled by the receiver (the event composer)
- The size of an event stream is potentially unbounded
- The event objects are typed or have a known structure
- Event representations may be corrupted

Processing Requirements for Streams

- Continuous processing of newly arriving event objects is required
- Queries must evaluate continuously and evaluate repeatedly on new event objects (as opposed to ad hoc pull-based queries)
- Approximate answers are acceptable (mostly)
- Completeness/accuracy is often traded for timeliness
- The result of a continuous query is a complex event that may trigger some pertinent action

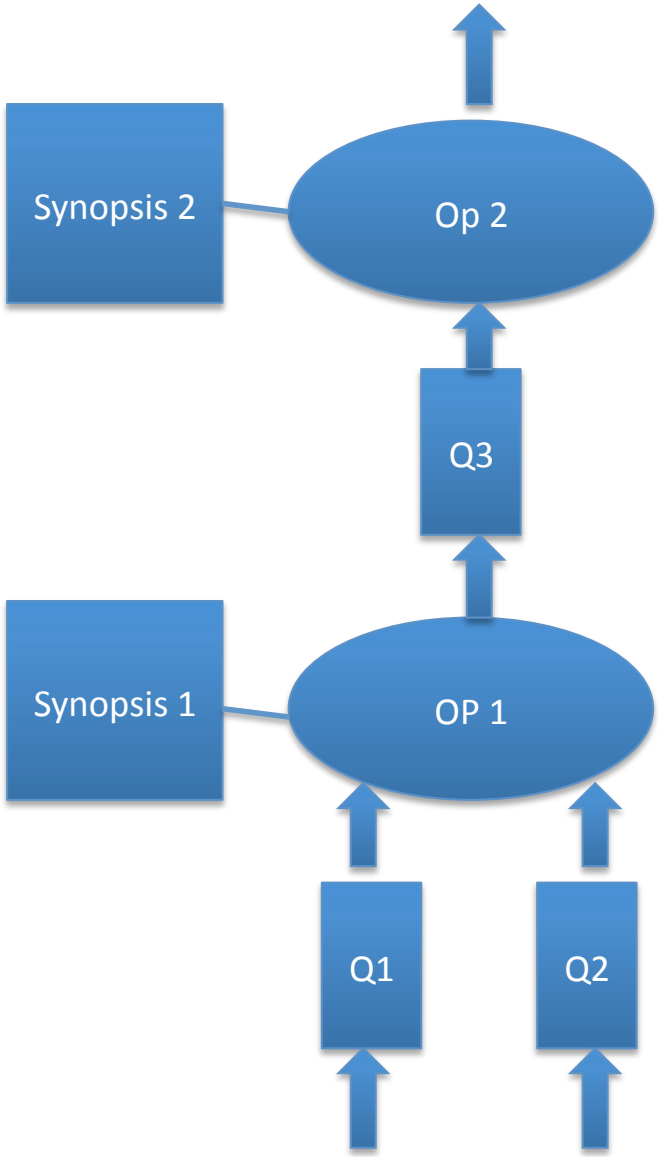
Blocking and non-blocking operators on streams

- Streams cannot be assumed to be finite
- Select (and project without duplicate elimination) can be executed in non-blocking manner
- Many other operators, such as sort, projection with duplicate elimination, max, min, join, etc. can't be completed without processing the whole input set → unless the whole input set is available the operation blocks
- To avoid blocking, finite portions of the stream are defined
- Windows allow to convert blocking operations into non-blocking ones
- A window specification is added to the continuous query specification and produces time-varying finite relations out of the stream

Processing continuous query plans

- Continuous queries are processed from the leaves to the root
- Each operator takes one or more streams as input and produces an output stream that can be fed to one or more operators
- Each operator may have to buffer streams during bursty input
- Each operator is associated with a main memory queue or buffer (non-persistent) to store the incoming or partially processed streams
- Output streams are directly sent to the next operator
- A synopsis (memory resources) is associated with each operator

Continuos Query Pans



Operators are non-blocking, adapted for window processing if needed

Results are directly forwarded to the next queue

Synopsis represents the resources for Storage of intermediate results

Window Definitions

- Windows are defined as a historical snapshot of a finite portion of a stream at any point in time
- A window defines a meaningful set of instances required by an operator to compute its functions
- Windows can be
  - Time based (physical windows)
  - Tuple or instance based (logical window)
- Windows are delimited by a window start (ws) and a window end (we)
- Two successive windows may be overlapping or disjoint
  - Overlapping (sliding): ws2 < we1 shares a portion of window
  - Disjoint (tumbling): ws2 >= we1 instances in window disjoint

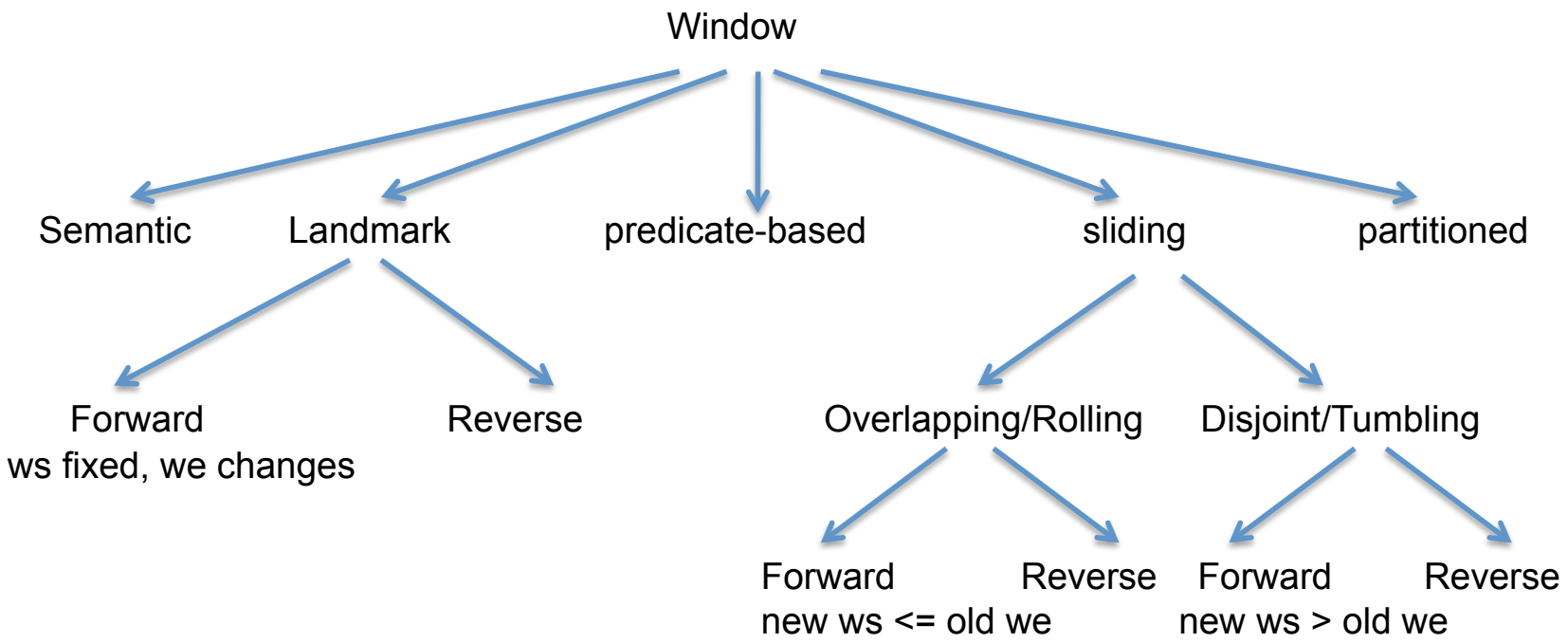
Window Definitions 2

- Window specification
  - Physical: [Range N time units, Advance M time units]
  - Logical: [Row N tuples, Advance M tuples]

- Range specifies the size of the first window
- Advance specifies how the begin of the next window is advanced, absence of Advance parameter implies a disjoint window

- A window definition is associated to a stream in a query
  - Same stream has same window in all the operators of a query using it
  - Stream can have different windows associated in different queries

Window Types proposed in the Literature



See Chakravarthy and Jian, Stream Data Processing: A Quality of Service Perspective, Chapter 2

Examples of Continuous Queries

- Scenario:
  - Streams of Call Data Records representing start or end event representations for phone calls
  - Definition of each CDR is
    - Call_ID
    - Caller for outgoing, Callee for incoming calls
    - Time
    - Event (Start or End)
  - Multiple central offices, each producing its CDR stream

Continuous Queries – Example 1

Find all outgoing calls from central office 1 longer than 10 minutes over a 24 hour window (assume no call is longer than 24 hrs)

```
SELECT    O1.call_ID, O1.caller
FROM      Outgoing O1 [Range 24 hours],
          Outgoing O2 [Range 24 hours]
WHERE     (O2.time – O1.time) > 10 minutes
          AND O1.call_ID == O2.call_ID
          AND O1.event == START
          AND O2.event == END
```

Continuous Queries – Example 1 (cont)

```
SELECT    O1.call_ID, O1.caller
FROM      Outgoing O1 [Range 24 hours],
          Outgoing O2 [Range 24 hours]
WHERE     (O2.time – O1.time) > 10 minutes
          AND O1.call_ID == O2.call_ID
          AND O1.event == START
          AND O2.event == END
```

Self-join query continuously outputs all outgoing calls longer than 10 minutes from central office 1 within a 24 hour disjoint sliding window

Results form a new stream

Continuous Queries – Example 2

Find all pairs of callers and callees between two central offices over 24 hour period

```
SELECT    DISTINCT O.caller, I.callee
FROM      Outgoing O [Range 24 hours],
          Incoming I [Range 24 hours]
WHERE     O.call_ID == I.call_ID
          AND O.event == START
          AND I.event == START
```

Continuous Queries – Example 3

Find the user who made the longest single outgoing call in the past 24 hours

```
SELECT     O1.caller, MAX(O2.time – O1.time)
FROM       Outgoing O1 [Range 24 Hours]
           Outgoing O2 [Range 24 hours]
WHERE      O1.call_ID == O2.call_ID
           AND O1.event == START
           AND O2.event == END
GROUP BY   O1.caller
```

PSoup: streaming queries on arriving and old streams

- Most systems produce results continuously and generate a new stream that feeds either other operators or the application
- If periods of disconnection occur, streaming over past events may be necessary
- PSoup addresses that problem:
  - Combines ad hoc and continuous queries
  - Treats them symmetrically
  - Allows applying new data to old queries and new queries to old data
  - Supports periods of disconnection by separating the computation of results from the delivery of results

Chandrasekaran, S., Franklin, M.; Streaming Queries over Streaming Data, VLDB 2002, Hong Kong

Graph-based Event Composition

- As an alternative to query-based event composition, graph based mechanisms can be used
- Many of the early active OODBMSs used some form of graph based composition
  - SAMOS used (colored) Petri Nets
  - HiPAC and REACH used event graphs
  - Different kinds of state machines have been proposed (e.g. Pablo's approach based on a Mealy Machine)

Composition using Petri Nets

- Following discussion is based on the approach used by SAMOS

Gatziu, S., Dittrich, K.;" Events in an Active Object-Oriented Database System", Proc. 1st International Workshop on Rules in Database Systems, Edinburg, August 1993.

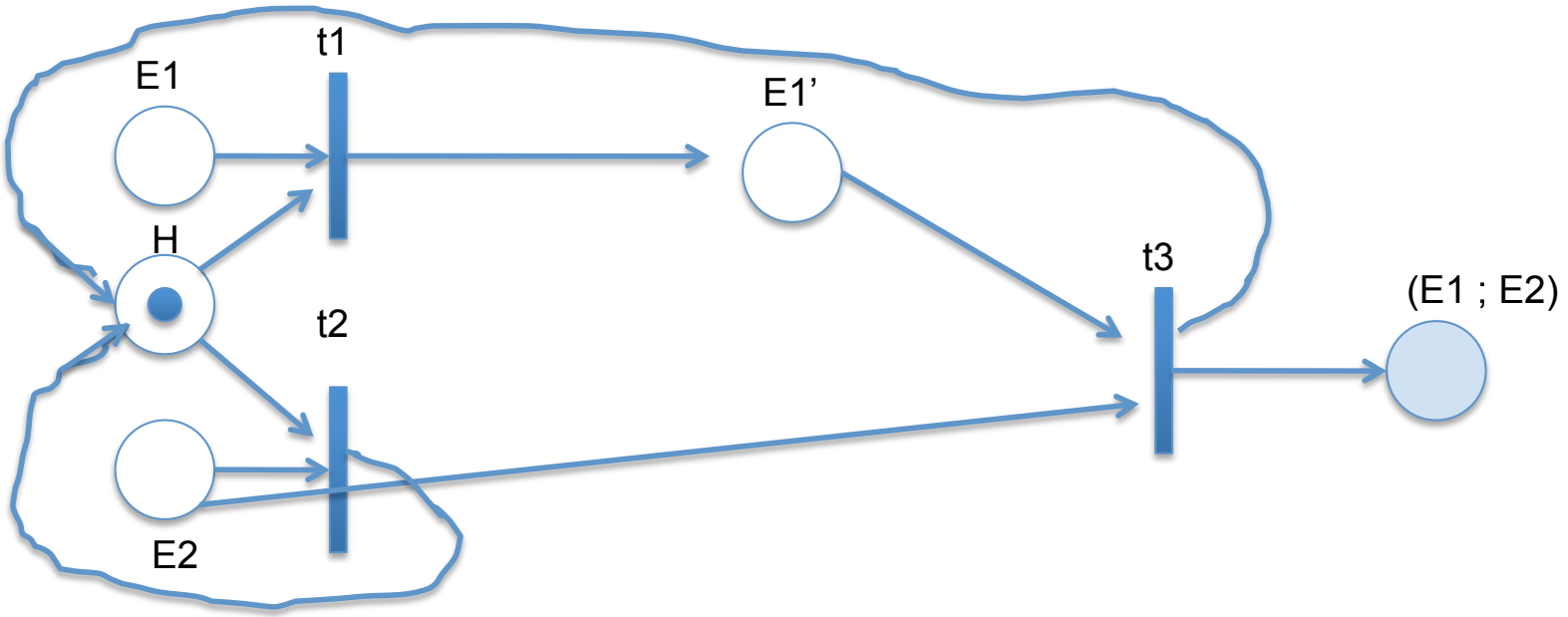
- A Petri Net (PN) is a tuple (N,M_0)
  - N is the static structure of the Petri Net
  - M_0 is the initial marking of the PN
- N is a 5-tuple (P,T,A,P_s,P_e)
  - P is a finite set of places
  - T is a finite set of transitions
  - A is a subset of (P x T U T x P), a finite set of arcs denoting connections between places and transitions
  - P_s input places which are marked as soon as the corresponding event appears
  - P_e output places which are marked as soon as the corresponding composite event is produced

Composition using Petri Nets (cont)

- A PN in an actual state has a non-negative number of tokens assigned to places
- Individual tokens are used to represent actual parameters and can be assigned to places
- A PN has a marking M. M_0 is the initial marking
- A transition t fires when all its input places are marked
- After the firing of a transition t all output places have to be marked
  - The value of the tokens of all input places flow into the tokens of the output places of t
  - The values of the parameters are assigned in accordance with the event algebra (e.g. union of params. in disjunction)

Composition using Petri Nets (cont)

- A distinct Petri Net pattern is defined for each operator of the event algebra
- Definition of the sequence pattern (E1;E2)



Event Composition with Petri Nets

- When user defines a new composite event pattern, the appropriate patterns for the constructs are retrieved and assembled into the new PN
- A new PN is instantiated for each composite event
- SAMOS combined all individual PNs into a single PN
  - The reason for combining multiple elementary PNs into a single PN was to reuse events in multiple compositions
    - Only one place has to be marked
    - If a composite event takes part in further compositions the output place doesn't have to be remarked as input place in a different PN
  - In practice this did not work out as published as semicomposed events could not be eliminated and bloated the PN

Composition with Petri Nets (cont)

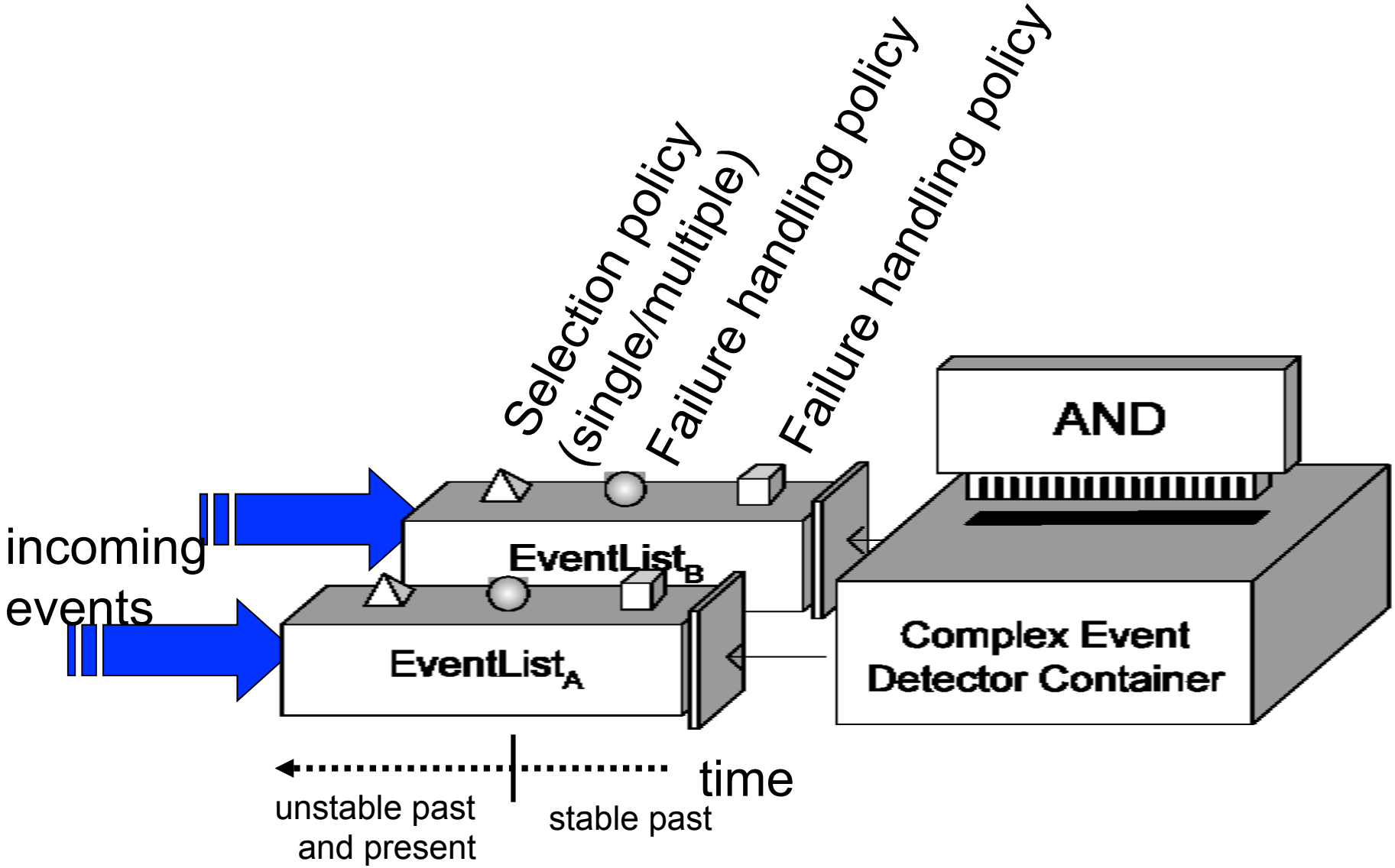
- Use of Petri Nets has following advantages:
  - Composite events can be detected step by step and do not require the search of a large register of (primitive) events
  - PNs are expressive and allow the definition of all defined patterns
    - Some modifications / transformations are required, e.g. an event E with monitoring interval [s-e] can be transformed into the sequence TS ; (Not TE; E)
    - Multiple instances of the same pattern can be handled by analyzing the parameters

Event Composition - Our Approach (cont.)

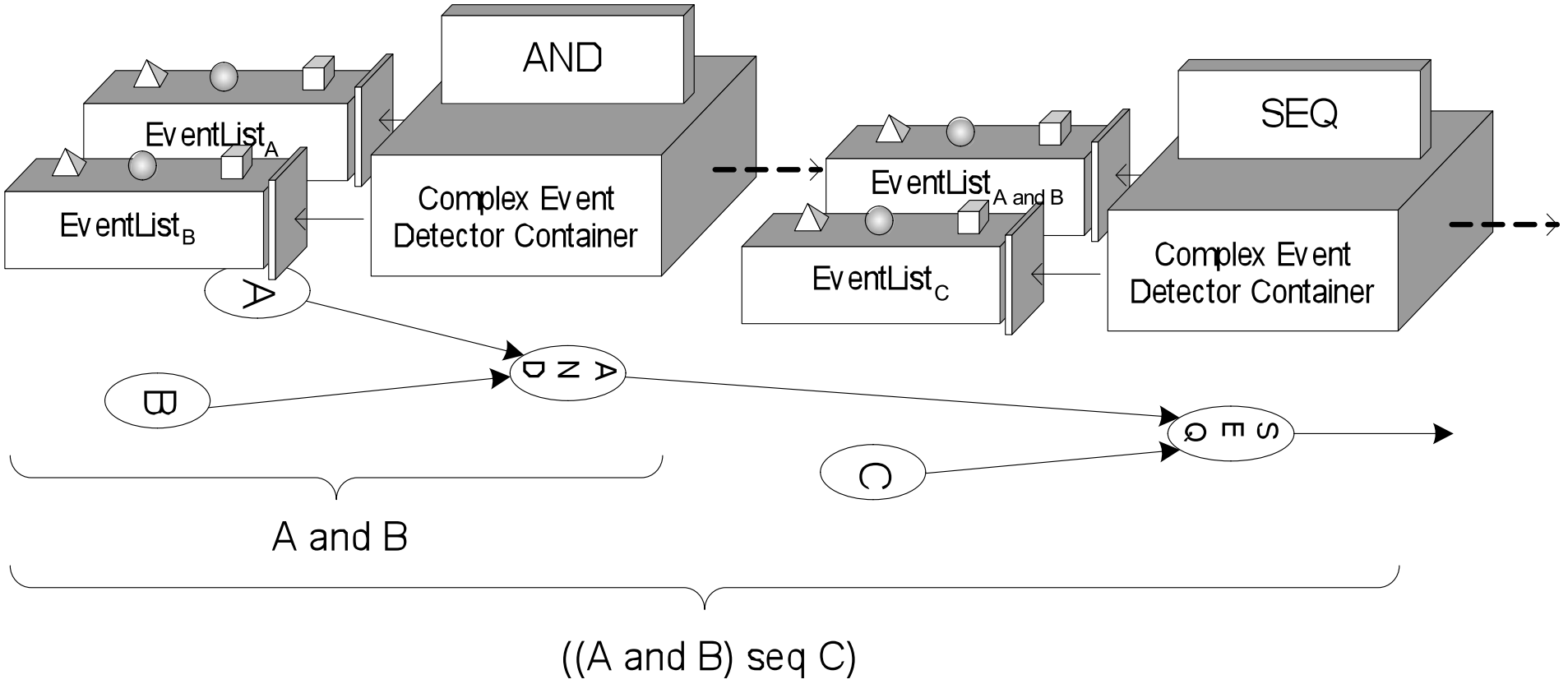
- Complex event detection
  - container and component model
  - flexible definition of event operators
  - concentrates on simplifying operator's logic
  - no hidden semantics (all decisions are made explicit)
  - configurable policies
    - failures
    - selection
    - uncertainty

Mariano Cilia "An Active Functionality Service for Open Distributed Heterogeneous Environments", Dissertation 2002 (Shaker Verlag)

Event Composition – Our Approach (cont.)



Event Composition – Example



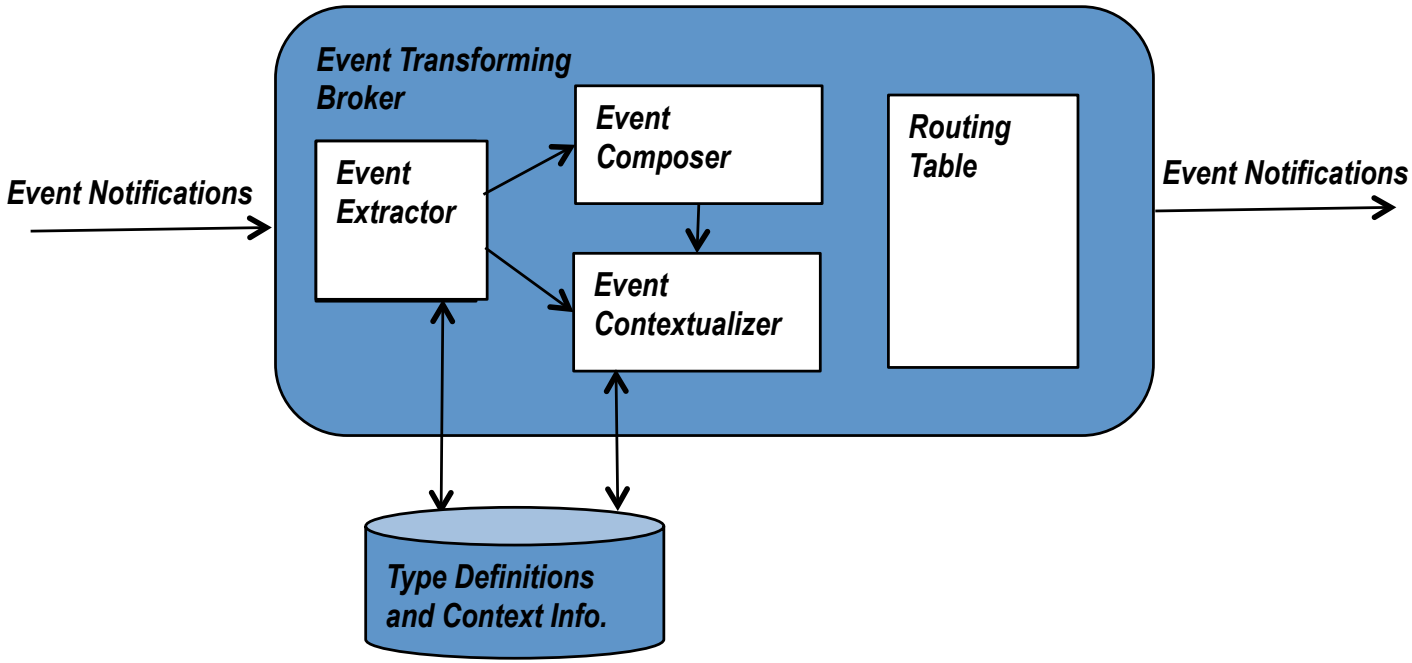
Event Processing Revisited

NOTE: After Opher Etzion's talk and after discussions with him and reading his book I came to the following conclusions that merit to revisit the topic of architecture, streams and operation on streams

The main points I want to make are:

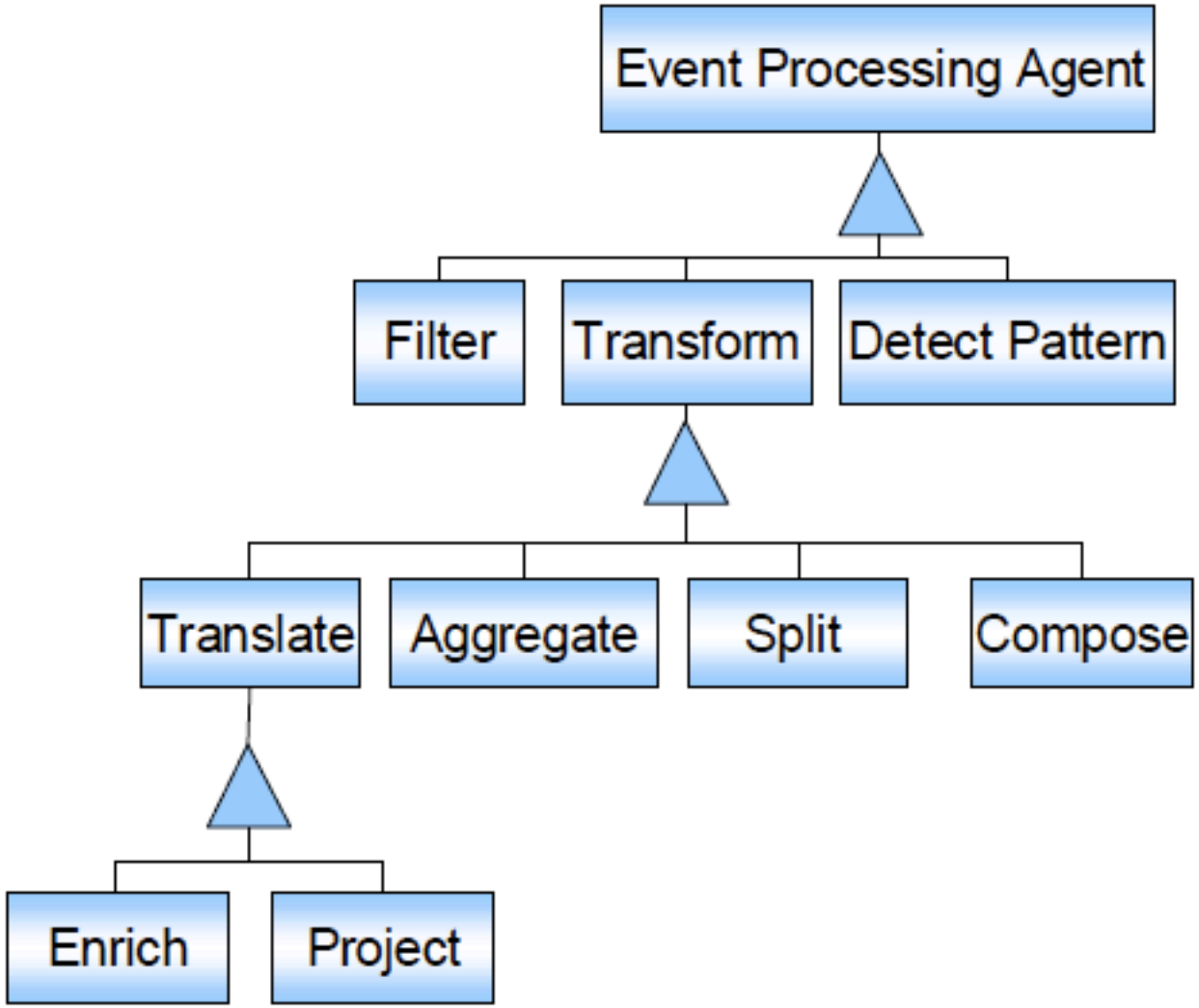
- While Opher's Event Processing Agents are a useful concept (not unlike brokers in the classical Pub/Sub literature), I dislike OE & PN architecture
- OE does not use the event oriented concepts cleanly and mixes push and pull
- The idea of having abstract, i.e. implementation independent, event processing functionality is good
- The notion of streams that we used is powerful and must be preserved
- We must clarify the distinction of events/messages, EPAs/brokers, routing and event modifying channels.

Event Transforming Channels revisited



- Network of brokers, brokers act as consumers/producers
- Simplest form: take in an event and change the structure of the event object and/or the format of one or more parameters
- Can aggregate, enrich or compose events
- Notification Transforming Broker receives event notification, unpacks it, transforms the event, repacks it into a notification and routes the new notification according to its routing scheme.

Abstract Operations on Events



Filter

- Filtering acts on a single event (out of an event stream)
- Stateless – neither previous nor future events influence the filtering
- Filters are typically Boolean predicates
  - Select operation in relational algebra
  - Where clause in SQL dialects

- Filters are pass/fail and do not change the event type
- Events that pass a filter are placed in an output stream
- Events that fail are either discarded (default) or put in another stream to apply a different filter

Transformation - Translate

- Translate transforms attribute values from one context to another context
  - Euro to $
  - cm to inch
- Not all attribute values must/can be translated
- Values that are not translated are just copied
- Translation is usually effected through scripting languages, stylesheets, functions in a general purpose programming language or simple table look-up
- Translate does not necessarily change the type of an event (depends on how context is handled)

Transformation - project

- Project eliminates attributes of an event object
- The attributes that are not eliminated are simply copied
- The type of the event changes under projection

Transformation - Enrich

- Enrich can copy, modify or insert attributes
- Inserted attributes typically come from external context (information sources)
- Sources are typically stateful external components
- Enrich may involve a translation (e.g. GPS location to street name) and can be complemented by additional information (e.g. one-way street)

Transform - Split

- Split produces multiple events from a single input event
- The resulting events may all have the same type or may be of different types
- Each event that results from a split is put on an event stream
- Split can be seen as a generalization of translate or project with multiple output streams
- Static splitting produces always same number of output events
- Iterative splitting produces a number of output events dependent on the input event
- Split of a composite event produces the corresponding simple events

Transform - Aggregate

- Stateful operation on a single stream
- Examples are:
  - MAX, MIN, AVG, SUM, COUNT
- Must define a buffer or window for these operations to be well defined
- Some authors would consider composition (building a set of homogeneous events) an aggregate
- Self-joins can be considered a case of aggregation (I'm not sure where to classify them)

Transform - Compose

- Compose acts on two or more streams
- Stateful
- Typically must specify
  - Buffer or window
  - One or more match expressions (join over two or more windows, places in a Petri-net)
  - Type-conformity
  - Consumption policy
  - Policy for treating non-matches (drop vs. forward)

Pattern detection

- Patterns can be of many types
- Saw basic patterns under aggregation (max, min)
- Many more patterns possible
  - Monotonic increasing
  - All events in a window
  - Fixed number of events
  - Top k
  - Outlier detection (2 std. dev.)
  - Thresholds, min distance, etc.
- Often these patterns can be combinations of filtering with dynamic filter predicates (e.g. to select increasing values)

Execution environments

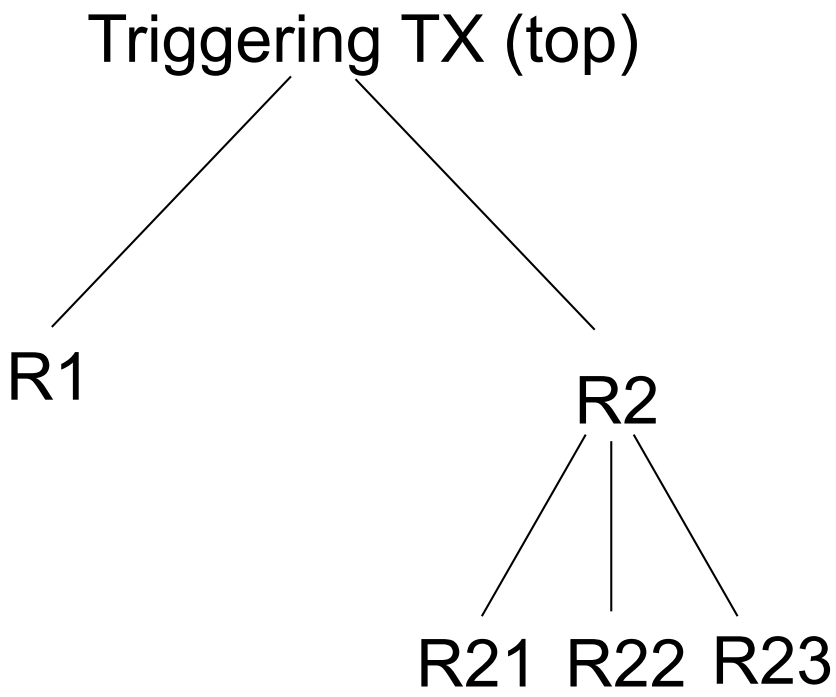
- In a database or TPM environment the natural execution environment is a transaction
- Transaction boundaries determine
  - when compositions are to terminate
  - when reaction is to occur

- In a generalized distributed system there are no natural execution boundaries and appropriate points for termination of composition and execution of reaction must be defined (explicitly), middleware mediated transactions

Transactions in Active Databases – the Origins

- In a situation where the events originate in a DBMS, every event occurs within a transaction
- Rules that are triggered by database events are executed as transactions
- Coupling Modes determine
  - Execution of rule(s) relative to triggering transaction
  - Whether rule is executed as one unit or if condition and/or action may execute as distinct processing units

Transaction Structure



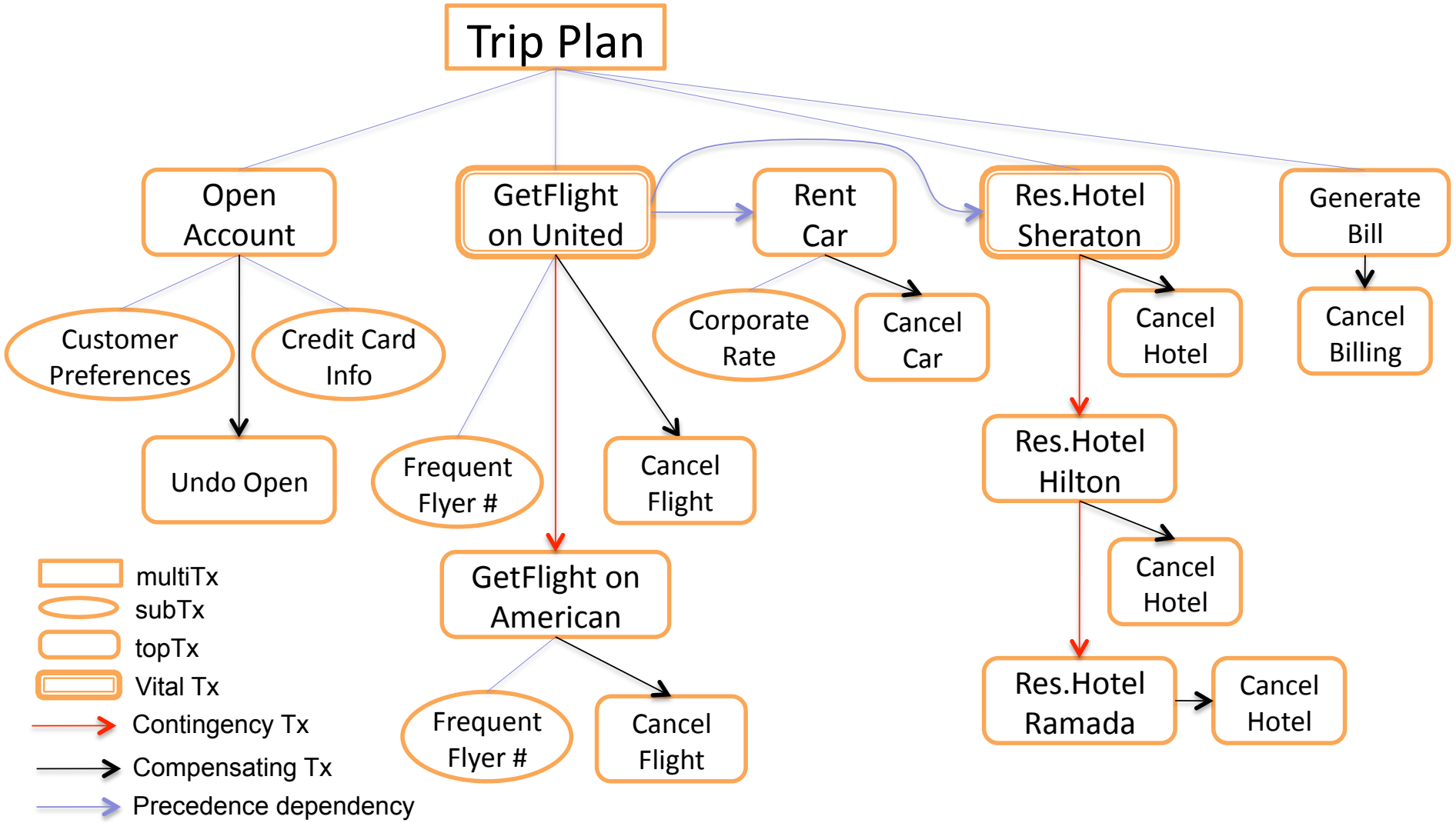
- Execution of rules in subtransactions results in nested TX
- If the actions of rules may cause other rules to fire, a deeper TX tree results
- Closed nested TXs
  - Commitment through the top
  - Siblings may not see results of each other until committed through the parent
  - Top can only commit if all children have aborted or committed

Open Nested Transactions

- Open nested transactions also have tree structure
- Visibility rules (isolation) are relaxed: results of subtransactions may be visible to the outside
- Because subtransactions may commit independently of the parent transaction, rollback is no longer possible, instead compensating transactions are used
  - compensating transaction is semantically inverse transaction
  - compensating transactions may not always exist (irreversible side effects)
- Subtransactions may be vital or not (abort dependency of parent transaction on vital subtransaction)

DOM Transaction Model, Buchmann, Özsu, Georgakopoulos and Hornick, in Elmagarmid „Transaction Models“

Example of DOM a Transaction



TX - Coupling modes (cont.)

- Execution of rules depends on:
  - Type of rule
  - Application domain
- As introduced in HiPAC subsume most others, sufficient for closed systems
- Extensions proposed in REACH to deal with open systems
- In distributed event-based systems, Middleware Mediated Transactions are required

Coupling modes for Consistency Rules

- Example
  - Consistency rule over simple attribute in single object
    - Evaluate immediately
  - Consistency rule over multiple, individually changed objects
    - Evaluate at end of transaction (deferred)
  - Consistency rule over multiple objects in CAD environment
    - Evaluate on demand or outside design transaction for first instantiated objects (decoupled)

TXs - Issues Relevant for Rule Execution in aDBMS

- When should rule execution begin?
  - Immediately after event detection
  - At end of user transaction but before commit
  - After triggering transaction committed
  - In parallel with triggering user-transaction, i.e. exact beginning is unimportant

TXs - Issues Relevant for Rule Execution (cont.)

- When should rule execution finish?
  - Before execution of triggering transaction resumes
  - Before commit of triggering transaction
  - After commit of triggering transaction
  - After abort of triggering transaction
- Must rule execute within triggering TX or may it execute as a separate TX?
  - As subtransaction
  - As separate transaction

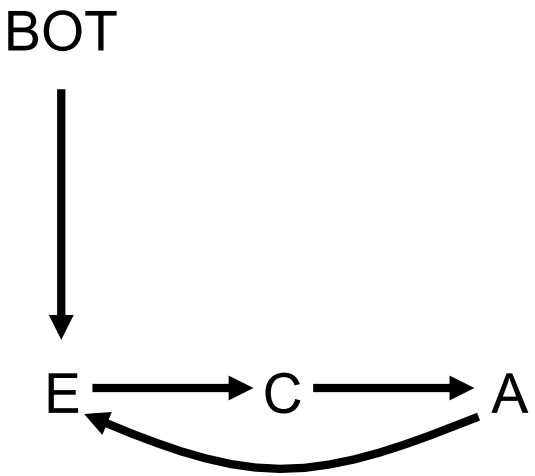
Coupling modes - Options

- Possible CM result from various combinations of above options
  - Immediate
  - Deferred
  - Detached
  - Detached causally dependent
    - Parallel
    - Sequential
    - Exclusive

Coupling Modes - Immediate

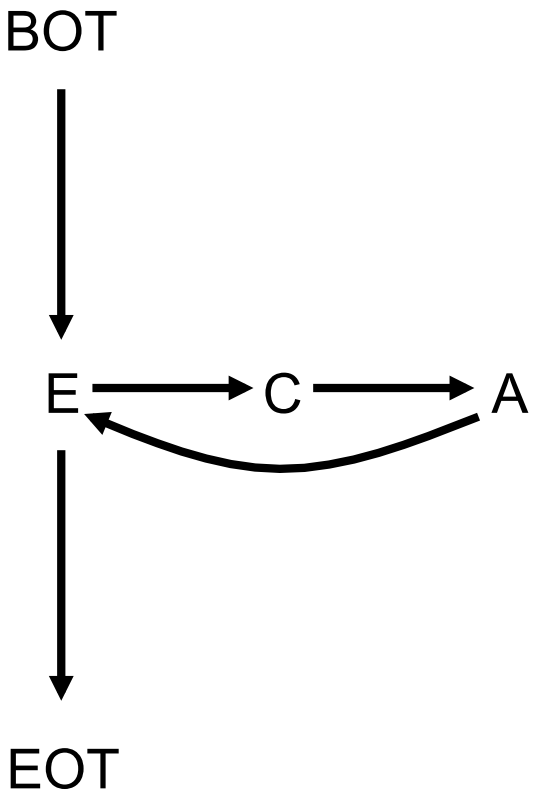
- Rule executed as soon as event is detected
- Execution as nested subtransaction
- Condition and Action may be specified with different CM

Coupling Modes - Immediate



- Rule executed as soon event is detected
- Execution as nested subtransaction
- Condition and Action may be specified with different CM

Coupling Modes - Immediate

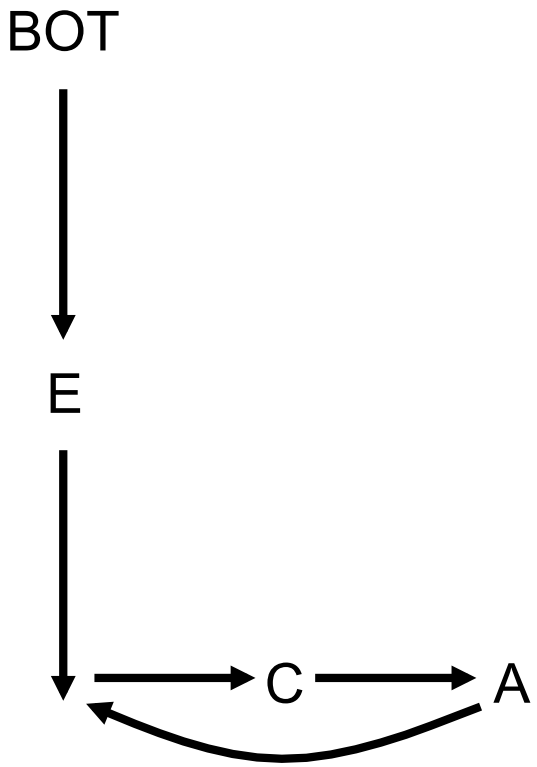


- Rule executed as soon as event is detected
- Execution as nested subtransaction
- Condition and Action may be specified with different CM (usually not)

Coupling Modes - Deferred

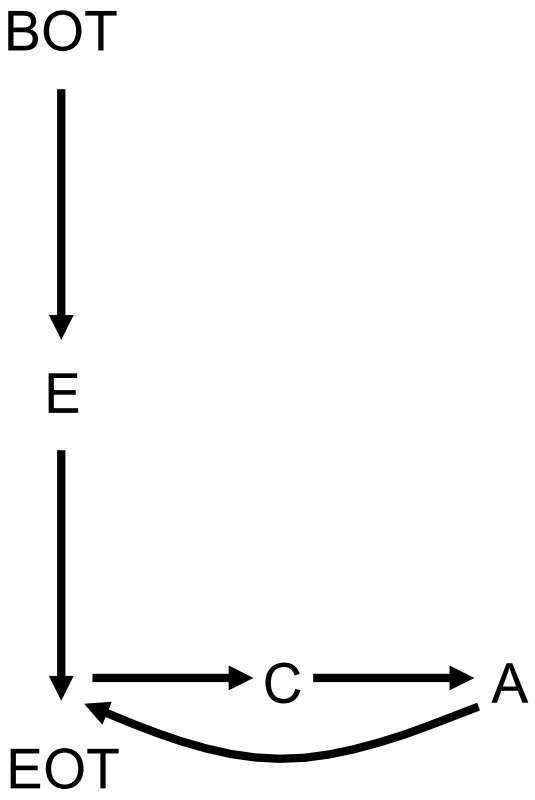
- Rule executed after user-submitted transaction but before it commits
- Condition and Action may have different CM
  - E.g. deferred condition and detached action (unusual)

Coupling Modes - Deferred



- Rule executed after user-submitted transaction but before it commits
- Condition and Action may have different CM
  - E.g. deferred condition and detached action

Coupling Modes - Deferred

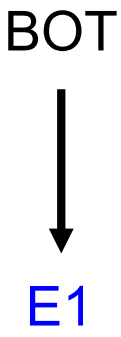


- Rule executed after user-submitted transaction but before it commits
- Condition and Action may have different CM
  - E.g. deferred condition and detached action (unusual)

Coupling Modes – Deferred (cont.)

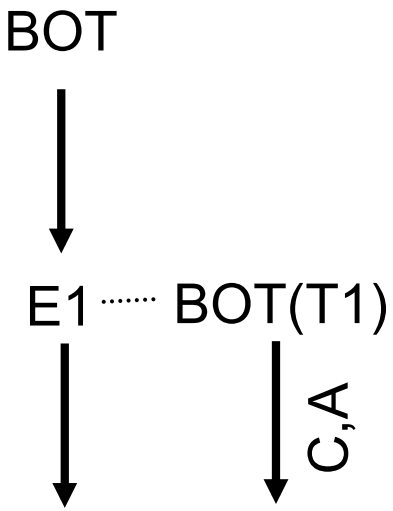
- Execution of deferred rules occurs in cycles
- All fired (deferred) rules are delayed until the end of user's top transaction (end cycle-0)
- At cycle-0 end
  - Begin (parallel) execution of all deferred rules fired so far
  - If new rules are fired, delay them until end-of-cycle (cycle-1 end)
- At cycle-1 end
  - Repeat until no more delayed rules
  - If a deferred rule triggers an immediate rule, it is executed as a nested TX

Coupling Modes - Detached



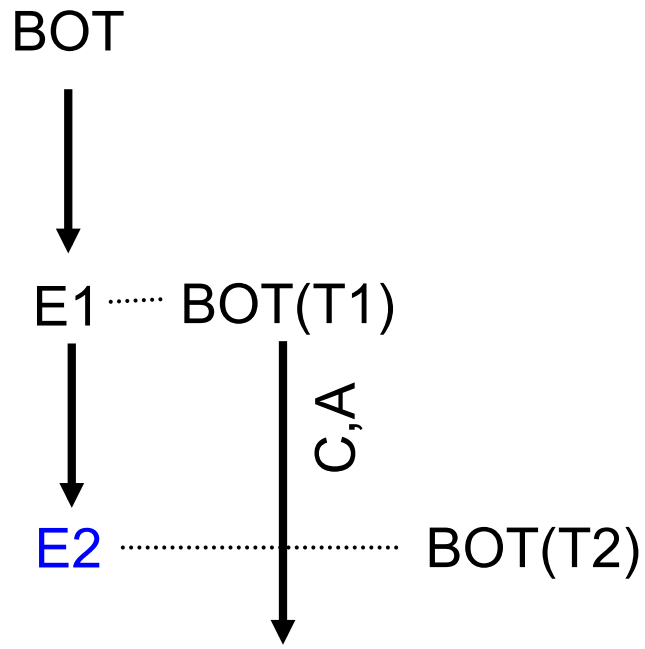
- For efficiency rules may be executed in separate transactions (if semantically correct)
- Rules execute in separate TXs and have no dependencies on triggering TX
- If triggering TX aborts, detached rule execution continues

Coupling Modes - Detached



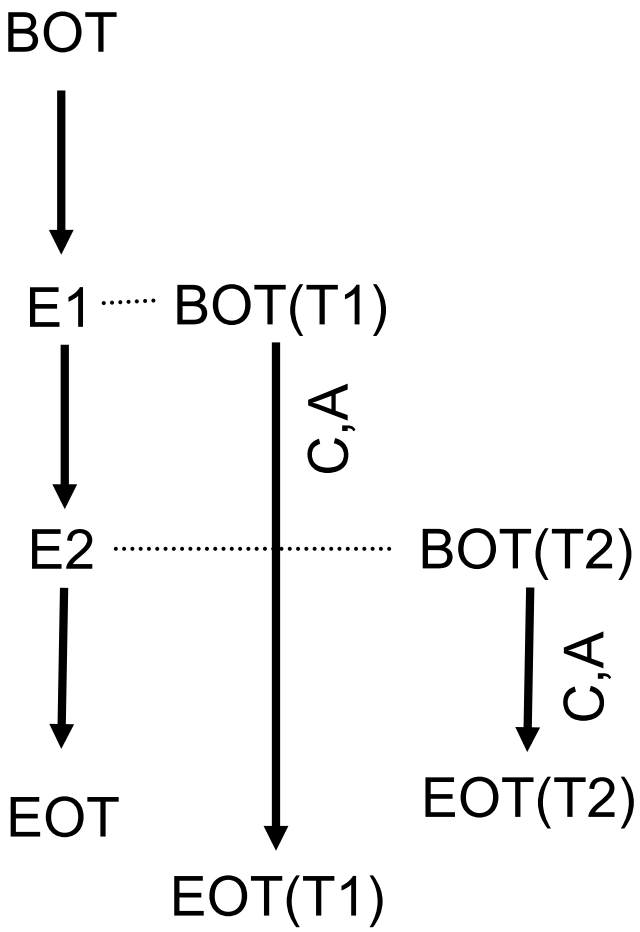
- For efficiency rules may be executed in separate transactions (if semantically correct)
- Rules execute in separate TXs and have no dependencies on triggering TX
- If triggering TX aborts, detached rule execution continues

Coupling Modes - Detached



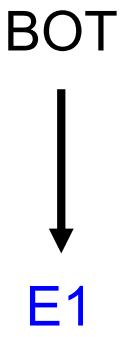
- For efficiency rules may be executed in separate transactions (if semantically correct)
- Rules execute in separate TXs and have no dependencies on triggering TX
- If triggering TX aborts, detached rule execution continues

Coupling Modes - Detached



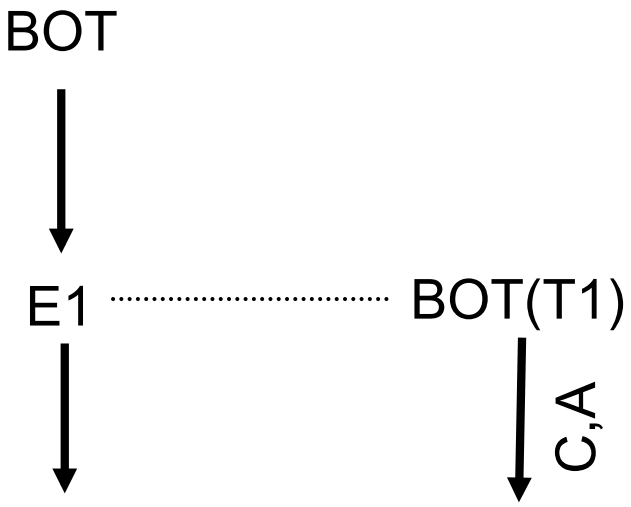
- For efficiency rules may be executed in separate transactions (if semantically correct)
- Rules execute in separate TXs and have no dependencies on triggering TX
- If triggering TX aborts, detached rule execution continues

Coupling Modes – Parallel Causally Dependent



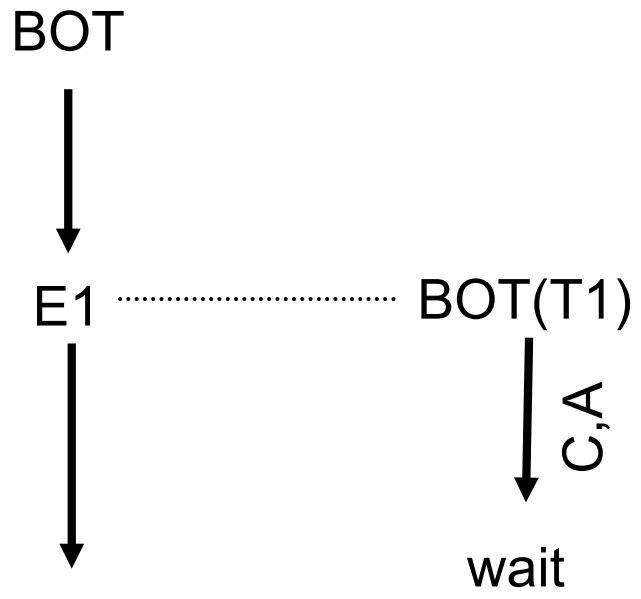
- Rule executes in separate TX
- Rule execution may begin in parallel
- Triggered TX (rule) may not commit until user-TX commits

Coupling Modes – Parallel Causally Dependent



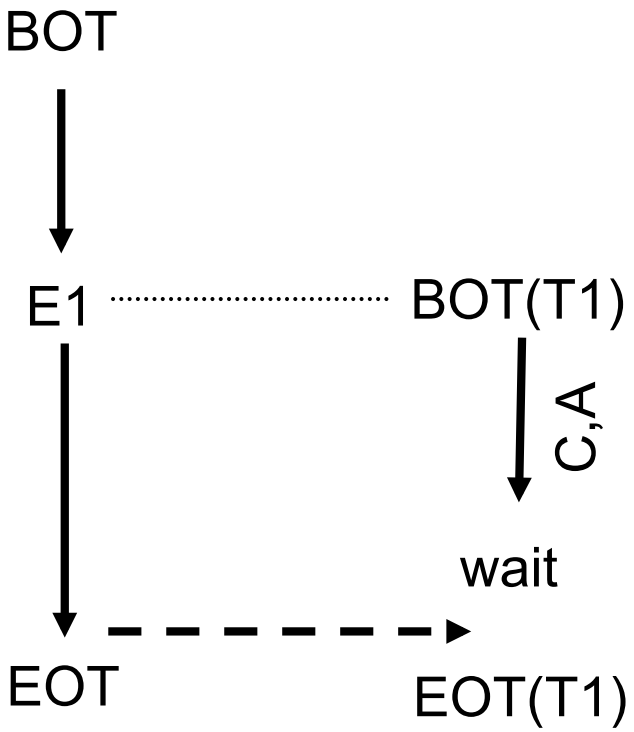
- Rule executes in separate TX
- Rule execution may begin in parallel
- Triggered TX (rule) may not commit until user-TX commits

Coupling Modes – Parallel Causally Dependent



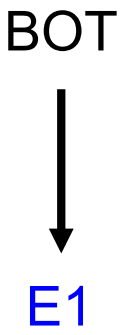
- Rule executes in separate TX
- Rule execution may begin in parallel
- Triggered TX (rule) may not commit until user-TX commits

Coupling Modes – Parallel Causally Dependent



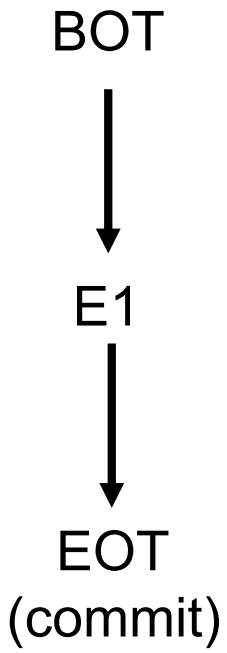
- Rule executes in separate TX
- Rule execution may begin in parallel
- Triggered TX (rule) may not commit until user-TX commits

Coupling Modes – Sequentially Causally Dependent



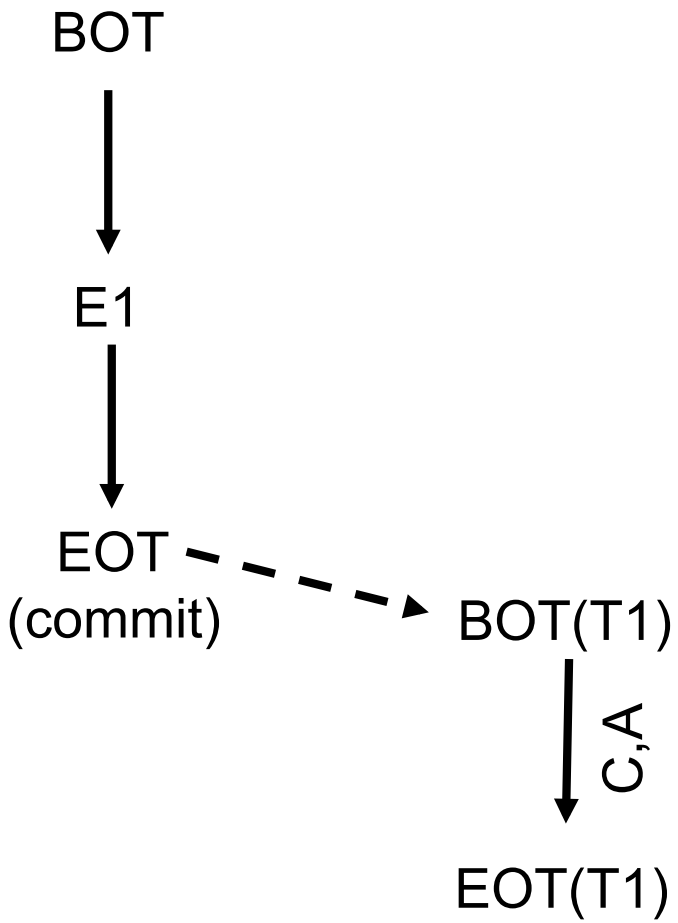
- Rule executes in detached TX
- Execution may not begin until triggering TX commits
- Necessary in environments with irreversible actions executed by rules

Coupling Modes – Sequentially Causally Dependent



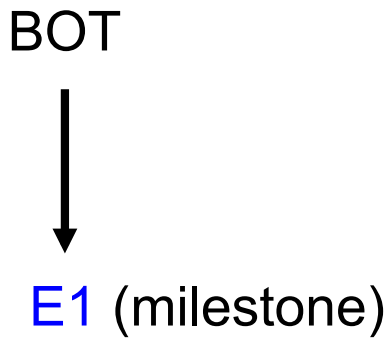
- Rule executes in detached TX
- Execution may not begin until triggering TX commits
- Necessary in environments with irreversible actions executed by rules

Coupling Modes – Sequentially Causally Dependent



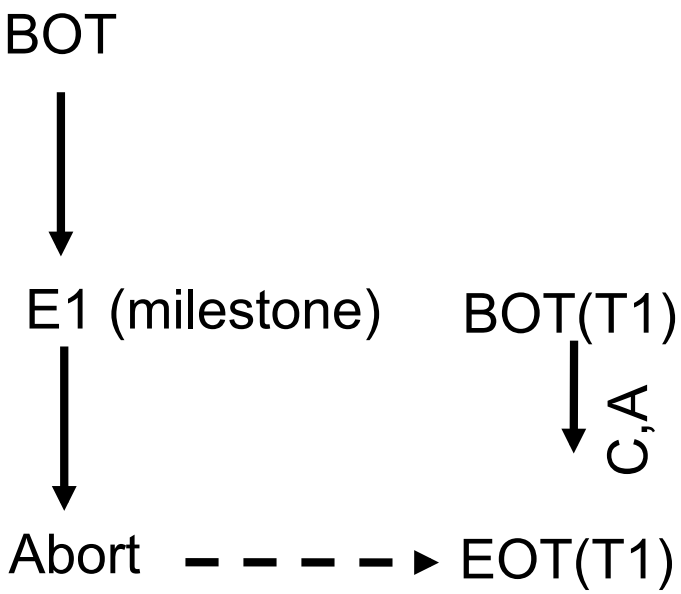
- Rule executes in detached TX
- Execution may not begin until triggering TX commits
- Necessary in environments with irreversible actions executed by rules

Coupling Modes – Exclusive Causally Dependent



- Rule may start execution in parallel but may not finish unless triggering TX aborts
- Useful in time-constrained processing
- Rule used for expressing contingency plans

Coupling Modes – Exclusive Causally Dependent



- Rule may start execution in parallel but may not finish unless triggering TX aborts
- Useful in time-constrained processing
- Rule used for expressing contingency plans

Detached and Causally Dependent Execution

- Detached rules are executed in their own top TX
- Top TXs are independent from each other
- Causally dependent rules are executed in separate TXs subject to dependencies
  - Abort dependencies
  - Commit dependencies
  - Exclusion dependencies

Coupling Modes – Summary

	Begin				Completion				TX	
	Event detec.	EOT	After commit	Parallel 2 user-TX	Before resume U-TX	Before commit U-TX	After commit U-TX	After abort	Sub-TX	Separate -TX
immediate	X				X				X	
deferred		X				X			X	
detached				X						X
Parallel causally dep.				X			X			X
Sequential causally dep.			X				X			X
Exclusive causally dep.				X				X		X

Transactions in event based systems

- In a pub/sub system the producers and consumers of events are decoupled through the mediator
  - transactions must include the mediator
  - transactional behavior (if needed) must be controlled on consumer side
- Challenge: how to combine notifications with conventional object requests into middleware mediated transactions (MMT)

Middleware Mediated Transactions

- MMTs extend atomicity sphere of transactional object requests to include mediators and/or final recipients of notifications and depend on:
  - topology: fixed vs. variable (1:1, 1:n, n:m)
  - binding: reference-based vs. mediator-based
  - Temporal availability: time-independent (may not be available at same time) vs. time-dependent
  - synchronicity: blocking vs. non-blocking
  - delivery: reliable (at least once, exactly once) vs. unreliable (at most once, best effort)
  - processing: best effort or atomic transactional
  - recovery: forward vs. backward

Example

- OO communication and transactions

  1:1, fixed topology, reference-based, time-dependent, synchronous, atomicity only

- (Traditional) Pub/sub

  n:m, variable topology, mediator-based, time-independent, asynchronous, transactional delivery to mediator but not to final recipients

Coupling Modes in X²TS

Visibility	immediate, on commit, on abort, deferred
Context	non, shared, separate top
Forward dependency	none, commit, abort (transitive)
Backward dependency	none, vital, mark-rollback
Production	**transactional, independent**
Consumption	**on delivery, on return, atomic, explicit**

Visibility

- Visibility determines when an event can be seen
  - immediate: as soon as they arrive at consumer and independent of outcome of triggering TX
  - on commit: event is only notified (delivered) when event-generating TX committed
  - on abort: event is only notified (delivered) when event-generating TX is aborted
  - deferred: consumer is notified as soon as producer starts commit processing

Coupling Modes in X²TS

Visibility	immediate, on commit, on abort, deferred
Context	non, shared, separate top
Forward dependency	none, commit, abort (transitive)
Backward dependency	none, vital, mark-rollback
Production	**transactional, independent**
Consumption	**on delivery, on return, atomic, explicit**

Transaction Context

- Transactional context determines whether a reaction executes in the same transaction as the event generator
  - none
  - shared: executes on behalf of triggering TX
  - separate: executes as ist own separate top TX

Coupling Modes in X²TS

Visibility	immediate, on commit, on abort, deferred
Context	non, shared, separate top
Forward dependency	none, commit, abort (transitive)
Backward dependency	none, vital, mark-rollback
Production	**transactional, independent**
Consumption	**on delivery, on return, atomic, explicit**

Forward Dependency

- Forward dependency determines when a triggered reaction may commit
  - none: it will execute independently
  - commit: it executes only if triggering transaction commits
  - abort: it executes only if triggering transaction aborts

Coupling Modes in X²TS

Visibility	immediate, on commit, on abort, deferred
Context	non, shared, separate top
Forward dependency	none, commit, abort (transitive)
Backward dependency	none, vital, mark-rollback
Production	**transactional, independent**
Consumption	**on delivery, on return, atomic, explicit**

Backward Dependency

- Backward dependency constrains the commit of the triggering transaction
  - vital: the triggering transaction may only execute if triggered TX executed successfully
  - mark-rollback: triggered TX is independent of triggering TX but if triggering TX is marked-rollback it will abort if the event could not be delivered

Coupling Modes in X²TS

Visibility	immediate, on commit, on abort, deferred
Context	non, shared, separate top
Forward dependency	none, commit, abort (transitive)
Backward dependency	none, vital, mark-rollback
Production	transactional, independent
Consumption	on delivery, on return, atomic, explicit

Event Production

- Event production may occur in a transactional context that includes the notification to the mediator or it may occur independently

Coupling Modes in X²TS

Visibility	immediate, on commit, on abort, deferred
Context	non, shared, separate top
Forward dependency	none, commit, abort (transitive)
Backward dependency	none, vital, mark-rollback
Production	**transactional, independent**
Consumption	on delivery, on return, atomic, explicit

Event Consumption

- Event consumption determines when an event is considered to have been delivered.
- Once an event has been consumed the notification is considered to have been delivered and will not be replayed if the consumer crashes subsequently
  - on delivery: consumed by accepting notification
  - on return: consumed when returning from reaction
  - atomic: bound to consumer‘s atomicity sphere
  - explicit: by explicit action of consumer

X²TS: An Event-Driven Transaction and Notification Service

- Integration of notification service (NOS) and transaction service (OTS) for CORBA
- X²TS realises an event-action model with
  - transactional behavior
  - flexible visibilities of events
  - coupling modes and asynchronous reaction to events
- Prototype based on COTS pub/sub MOM, current version open source
- Quality of service determined by consumer rather than producer ==> support for multiple requirements
- XA-adapters

Notification Service Overview

- Event-based, pub/sub decoupled communication
  - event channel (untyped, typed and structured events)
  - administrative objects (proxy factories)
  - proxies (publisher and subscriber, push and pull)
- Push and pull style event communication
- Filters, may be attached to administrative objects (applicable to all proxies) or to individual proxies
- Filters are disjunctions of predicates
- Filters can be combined (AND/OR)

OTS Overview

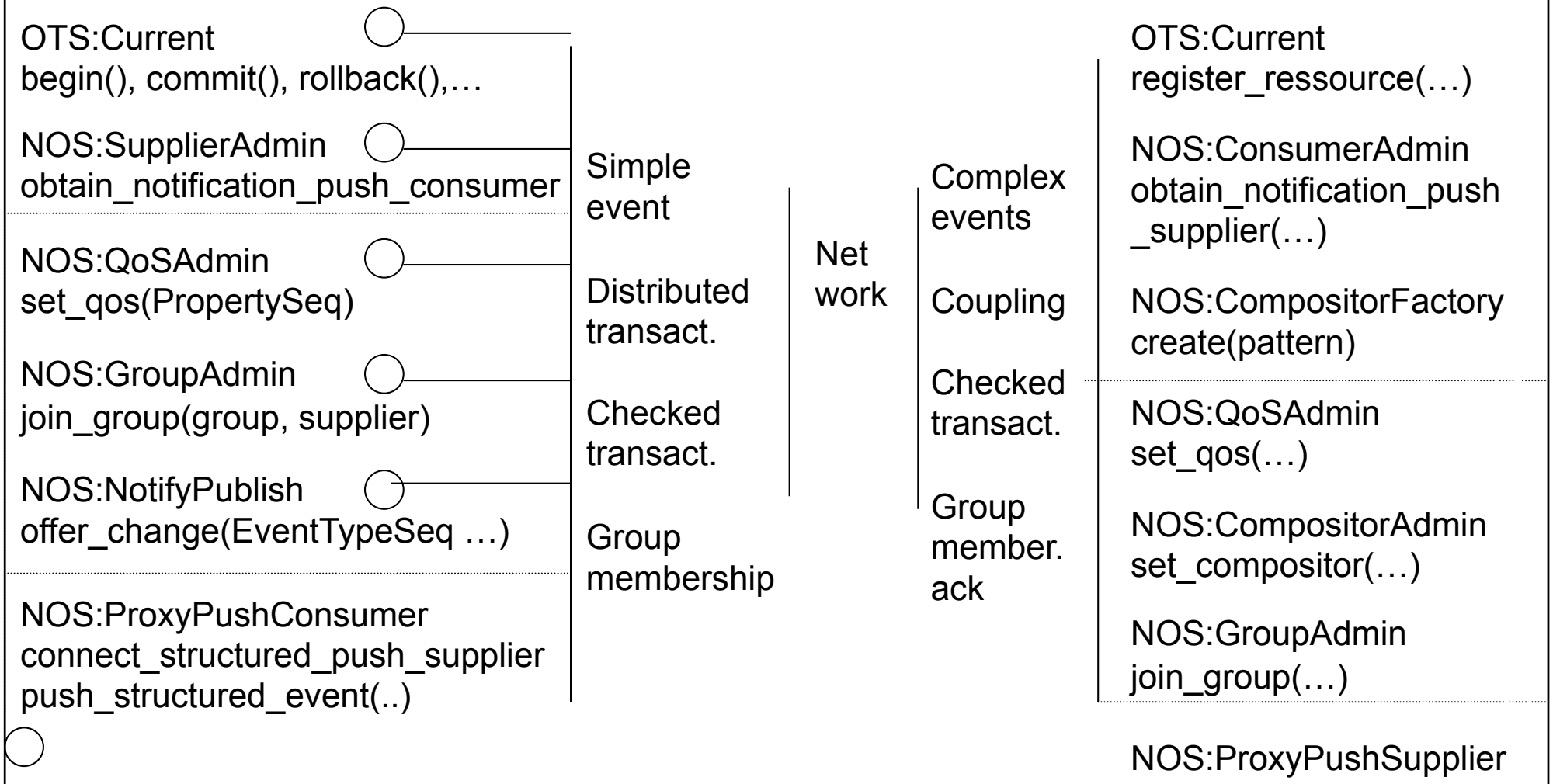
- All-or-nothing execution (atomicity spheres, 2PC)
- Isolation provided by the concurrency service (2PL)
- Event-based style results in asynchronous transaction branches and dynamic atomicity spheres
- In event-based style dependencies between publisher and subscriber expressed by
  - visibility (when should events be notified to subscribers)
  - transaction context of the reaction
  - commit/abort dependencies between atomicity spheres of action and reaction

Services Provided by X²TS Prototype

X²TS prototype implements subset of possible (and desirable) options

- push/push Notification Service
- StructuredEvent one-at-a-time notifications
- subscriptions to events of specific types and patterns of events
- patterns may be simple filters or complex event compositors
- indirect context management
- implicit context propagation
- interposition with interceptors

X²TS Architecture



Context

- Definition of context according to Wikipedia
  - **Context is the surroundings, circumstances, environment, background, or settings which determine, specify, or clarify the meaning of an event.**
- Alternative definition from Etzion and Niblett
  - A context is a named specification of conditions that groups together event instances for the purpose of processing them together. A context may have one or more context dimensions and consist of one or more context instances
- Distinguish semantic context vs. operational context

Semantic Context

- Semantic context is necessary to assign meaning to an event
- Knowledge of the source context and the target context is needed to translate events from one context to another
  - Example: A sale of Siemens stock on the NY stock exchange vs. a sale of Siemens stock in Ffm may be expressed in different currency, dates may have different format, etc.
- Context is needed to convert absolute or physical geographical position into logical location
  - 45^{o} 52’ N, 5^{o} 26’ E corresponds to Hochschulstr 10 in Darmstadt
  - TUD context needed to interpret S2|02

Semantic Context

Semantic context is the set of metadata needed to interpret an event unambiguously

Context information may be implicit or explicit

Implicit context information could be that prices are always given in Euro and temperatures in deg C

If the event producer and the event consumer operate in the same context, implicit context is enough

Explicit context is needed when subscribing to events from a foreign context

Generalization/Aggregation of Context

- Generalization and Aggregation operations can be applied to context
  - Metric System is a set (aggregate) of valid units
  - Euro Zone is a set of countries using the same currency
  - Airline Context is a generalization of the contexts used by different airlines on their web sites

Personal and Social Context

- Personal context can be defined by profiles
- Profiles may have many different degrees of detail (dependent on the application domain)
  - Basic information: id, phone number, email address, …
  - Device information: iphone, …, resolution, comm. Protocols
  - Preferences: music, food, gas stations, loyalty progs.
  - Constraints: allergies, food, …
- Social context includes friends, enemies, …
- Sometimes intersection of different contexts: friends and proximity

Operational Context

- A context is a named specification of conditions that groups together event instances for the purpose of processing them together. A context may have one or more context dimensions and consist of one or more context instances (Etzion)

- Dimensions of context serve for grouping
- Windows are a form of operational context definition
  - A window is thus a context instance
  - Windows can be defined based on number of instances or timestamps, can overlap or be disjunct, …
- Contexts also can be defined based on location (spatial context) or on some other partitioning criterion (segmentation-oriented context)

Temporal context (based on Etzion and Niblett)

From an operational point of view temporal contexts can be:

- Fixed interval: starting time, end time, period, order relation
  (24 hr interval 00:00 to 23:59 to limit withdrawals from ATM)
- Sliding fixed interval: interval starting on first event
  (24 hr interval after first withdrawal from ATM)
- Event interval: specific begin event and specific end event
  (From patient admittance to patient release)
- Sliding event interval: every n event instances
  (every 10 transactions)

In event intervals both initial and ending event may be qualified by a predicate
Order relation indicates if order is based on position in stream, timestamp, etc.

Temporal context as constraint

- Temporal context may be used as a selection criterion (or a constraint).
  - Absolute temporal consistency (based on Ramamritham)
    - Event is discarded if $t_{now} - t_{timestamp} >$ threshold
    - Stale events are discarded, relates to QoS freshness
  - Relative temporal consistency (based on Ramamritham)
    - All the events that are to be correlated should be within a "time-band"
    - For all events that are correlated

      $|timestamp_i - timestamp_j| <$ threshold
    - Also relates to QoS since the tighter the time-band, the better the quality of the derived event

Location or spatial context

- Location is one of the most frequently used contexts
- Location can be based on
  - Physical location (coordinates)
  - Logical location (address)
- Mappings between physical and logical location are often needed
- Low level processes and sensors typically use coordinates while users are more comfortable with logical location
- Location can refer to
  - The location of the sensor
  - The location of a reference entity
  - The location of an explicit event

Grouping and deriving events based on location

- Grouping events based on location requires
  - A reference point
  - A distance from the reference point
    - Typical grouping is all events that occur within circle with center at reference point and radius = distance from ref. pt.
  - The reference point can be either a physical location, an entity that has a location attribute or an event whose location can be derived
  - Bounding boxes are often used (end points of diagonal)
- Deriving events based on location requires multiple event instances (time series)
  - Entering or leaving requires at least 2 location events and their ordering

State dependent context

- Context may be defined based on the state of a particular object
  - Alert levels of a security system may determine which events must be monitored
  - Parameters are the reference entity, the possible states and the temporal ordering (based on time of occurrence or time of arrival of the event)

Segmentation oriented context

- Segmentation oriented context is defined based on the value of an event's attribute(s)
  - Generalization of temporal and location contexts
  - Any attribute in the payload can be used for segmentation given the proper segmentation expression
  - Can be used for banding (all instances of product entering warehouse referring to supplier XYZ, or all RFID tags within certain band)

Context Initiator Policies

- Whenever events of a certain type are responsible for initiating a new window (context instance) we must deal with the question: what happens if a new instance of the initiating event appears in the event stream?
- Possible options are:
  - Ignore (nothing happens, initiating event is ignored)
  - Add (opens a new window while preserving the old one, includes the new event in both windows)
  - Refresh (closes old window and opens a new one)
  - Extend (adds the new initiator to the old window and moves the end event if this is an offset time)

Context Composition

- Contexts can be defined combining two or more contexts
  - Spatio-temporal contexts
  - Spatial and segmentation (e.g. cars of a type in areas of the city)
  - Temporal and segmentation (e.g. hour windows and customer type)

Discussion

- The notion of context as used by Etzion and Niblett (operational context) is quite similar to the original contexts of SNOOP
  - They represent selection and grouping policies for events
- This is in contrast to the broader notion of context (semantic context) as used e.g. in concept based pub/sub
  - Here context provides the metadata for a correct semantic interpretation of an event
- In context aware services or ambient-aware pervasive environments the notion of context appears to span the whole range (location, time, intent, profile, etc.)

Mobility in Event Based Systems

- Three kinds of mobility
  - Movement of the source of events (any location based events such as GPS equipped devices, derived location e.g. based on dead reconning)
  - Movement of the recipient of events (plane in flight, mobile subscriber)
  - Movement of the relaying brokers (VANETs - vehicular ad hoc networks, mobile phones as concentrators, planes as mobile sensor platforms and relay nodes)

Areas Contributing

- Mobile Pub/Sub
  - Supporting mobility in content-based Pub/Sub middleware (Fiege, Gärtner, Kasten, Zeidler)
  - Mobility support with REBECA (Zeidler, Fiege)
  - Looking into the past: enhancing mobile Pub/Sub MW (Cilia, Fiege, Haul, Zeidler, Buchmann)
  - Aeronautical Pub/Sub (Christian Grothe diss.)
- VANETs
  - Pub/Sub Notification Middleware for vehicular networks (Leontiadis)
- Wireless ad hoc networks, STEAM (Miller and Cahill)
- Time-space event correlation (Eiko Yoneki)
- Pocket switched networks (Yoneki, Baltopoulos, Crowcroft)
- Human Dynamic Networks (Yoneki, Hui, Crowcroft)

Problems derived from mobility

- Mobility makes event processing more complex
  - Wireless communication
  - Intermittent communication
  - Resource limitation
    - Energy
    - Bandwidth
    - Compute power and storage
  - Handover from one edge broker to another
  - Loss or delay of events and fault management
  - Speed of movement determines timing constraints
  - Others?

Location of the source (adapted repetition last class)

- Location of the source of events is the simplest problem
- Location can be based on
  - Physical location (coordinates)
  - Logical location (address)
- Mappings between physical and logical location are often needed
- Low level processes and sensors typically use coordinates while users are more comfortable with logical location
- Hybrid forms of location determination (GPS outdoors, dead-reconning or control-point/beacon based indoors)

Proximity determination

- Same as grouping events based on location requires
  - A reference point (typically the location of the observer or the ME in ambient computing)
  - A distance from the reference point
    - Typical grouping is all events that occur within circle with center at reference point and radius = distance from ref. pt.
  - The reference point can be either a physical location, an entity that has a location attribute or an event whose location can be derived
  - Bounding boxes are often used (end points of diagonal)
- Typical situations:
  - Raise alarm when sex-offender wearing electronic bracelet is within restricted distance to school
  - Raise alarm when number of known hooligans per unit area increases above x (fixed area not good enough, moving centroid)

Proximity determination

- Same as grouping events based on location requires
  - A reference point (typically the location of the observer or the ME in ambient computing)
  - A distance from the reference point
    - Typical grouping is all events that occur within circle with center at reference point and radius = distance from ref. pt.
  - The reference point can be either a physical location, an entity that has a location attribute or an event whose location can be derived
  - Bounding boxes are often used (end points of diagonal)
- Typical situations:
  - Raise alarm when sex-offender wearing electronic bracelet is within restricted distance to school
  - Raise alarm when number of known hooligans per unit area increases above x

Service delivery to mobile entities

- Scenario: mobile user wants to be notified of “interesting” things in his proximity
  - Determining position of mobile user
  - Determining what is of interest to mobile user
    - Profile
    - Subscriptions (key question: when should I check user preferences against environment or what subscriptions should be active at what time?)
  - Answering efficiently the continuous query
  - Notifying the mobile user

Notifying mobile users of events of interest

- Some key considerations when dealing with mobile users
  - Speed of movement
  - Need for handover
  - Possibility of disconnection
- Need to support roaming clients
  - Buffering of notifications when disconnected
  - Rerouting and replay of messages at different locations
  - Goal: exploit loose coupling and allow applications developed for stationary subscribers to be used in mobile environments

Physical and Logical Mobility

- Physical mobility is intended to provide location transparency
  - A physically mobile client is disconnected certain periods of time
  - Physically mobile client has different border brokers along his itinerary through the infrastructure
- Logical mobility is intended to provide automated location awareness
- Physical and logical mobility are orthogonal aspects of mobility
  - Phys. Mobility deals with rerouting subscriptions to new locations
  - Logical mobility automates the adaptation of subscriptions

Physical Mobility

- Desirable features in support of physical mobility
  - No change of interface between pub/sub middleware and application for mobile or non-mobile users
  - No events should be lost during periods of disconnection, i.e. pub/sub middleware delivers eventually all messages
  - Sender FIFO order is guaranteed
  - System should be responsive (however, due to reconfiguration of the network topology delays are possible)

Possible solutions to physical mobility

- Naïve solution: sequence of subscribe/unsubscribe/subscribe
  - Very costly since each subscription is a new entry in the routing table
  - Since moving subscriber does not know a priori when disconnection will occur, this turns into a subscribe/subscribe pattern
  - Without proper support, because of different travel times through the network,
    - A user moving from b1 to b2 after notification has been delivered at b1 but not yet at b2 may receive a duplicate
    - A user moving from b1 to b2 before delivery at b1 but after delivery at b2 may miss a notification

Possible solutions

- Mobile IP is intended for hiding mobility in the network layer
  - Complete hiding of mobility in the network layer facilitates physical mobility
  - Makes exploitation of location information impossible for the support of logical mobility

Extending REBECA for mobility

- Prerequisites:
  - Each border broker has a buffer for undelivered messages
  - Advertisements are used (simplifies the trace-back from the new border broker to the publisher)
  - Simple routing is used as routing strategy
  - Border brokers can recognize new subscribers entering their range

Algorithm outline

- Subscriptions are automatically reissued at reconnection by the local broker installed on the client
- Broker network configuration is updated to reflect relocation instead of plain resubscription
  - Advertisements are followed back from new border broker to first junction (recognized because it already has an entry for that client)
- Notifications that were forwarded to the old border broker must be replayed to the new border broker
  - Junction broker sends a fetch along old path. Entries in routing table are now reversed. Border broker replays buffer. Messages in transit are collected on the way to junction. After replay path is closed down
- Delivery of new notifications is suspended until replay is complete. This guarantees sender FIFO
  - From junction broker messages are sent to new border broker and properly ordered with other (later messages) to ensure sender FIFO

Dealing with disconnection

- Clients (subscribers) may not move but just suffer disconnection
  - Buffer at border broker maintains undelivered messages
    - Subject to buffer size
    - Timeouts for limiting periods of disconnection
  - At reconnection simply replay messages
  - If client connects to different border broker, eventually a request for replay will arrive, execute step three of generic algorithm

Dealing with filter merging

- If filters were merged (e.g. covering was used) replay will reach the node where the merged filter is but it is not clear where the exact filter is located (could be in the direction of advertisement or in direction of old border broker)
- Must follow possible branches (somewhat like flooding)
- Eventually the right junction broker is reached, revert to base algorithm

Effect of speed and cache sizes

- Effect of speed of movement
  - Slow movement (or equivalently a large broker range) are basically assumed
  - High speed may result in reconnection and replay not having finished before the next separation occurs
  - Problem can be addressed by proper relocation counters that are reset after a successful replay
- Effect of limited cache
  - Especially relevant for small edge brokers
  - Periods of disconnection that are tolerable are influenced by cache size and message traffic

Logical mobility

- Contrary to physical mobility, the subscriber remains attached to the same broker but has an application-dependent notion of location
  - E.g. parking spot finder
  - Being connected to one broker in the house but wanting only notifications referring to the room I am in
- Logical mobility depends on location-dependent subscriptions
- Depends on location determination of user and the relationship of that location to other objects/services/ events

Solution approaches for logical mobility

- Naïve approach
  - Building a wrapper on top of standard pub/sub system
  - Suffers from blackout periods
    - Tdel for one way and at least tdel to new location
    - 2 tdel is blackout time, may lead to starvation
- Naïve approach 2
  - Avoid starvation by flooding
  - Flooding is too expensive to be useful in pub/sub systems of meaningful size

- Proposed solution: since user does not physically disconnect from border broker, deliver “as if” flooding was used → use epochs

Solution for logical mobility

- Assume a chain of filters f3, f2, f1, f0
- Assume inclusion relationship (i.e. f0 is the most specific)
- Can simplify to equality
  - This has the effect that edge broker always does perfect filtering
  - Brokers down the line receive approximations
  - If approximations are based on position of the consumer, then brokers along the path receive messages “in the neighborhood”
- Trick is to establish a movement graph that expresses the possible locations as a function of time (T → L)
  - Movement graph is subject to restrictions (e.g. max speed)

Solution for logical mobility (cont.)

- Messages are sent along the path according to the table derived from the movement graph
  - The further ahead I look, the farther I can be in location
  - Send messages anticipating movement to potential brokers
- This behavior has the behavior of perfect flooding without incurring the cost of flooding

Architecture of Context Aware Systems

- Different approaches
  - Middleware
  - Frameworks

- Middleware based
  - Reconfigurable Context-Sensitive Middleware (RCSM) – Yau, Karim, et al. Pervasive Computing 1,3
  - CORTEX – from the Lancaster group (Gordon Blair, Paul Grace, etc.)
- Framework based
  - Context Toolkit
  - Hydrogen project

RCSM

- RCSM is adaptive
- Depending on context-sensitive behavior of application it adapts
  - Object discovery
  - Connection management

CORTEX

- Introduced sentient objects
- Sentient objects are responsible for receiving, processing and providing context-related information
- Sentient objects are autonomous objects capable of sensing their environment and acting accordingly
- Sentient objects can be organized according to their (primary) tasks

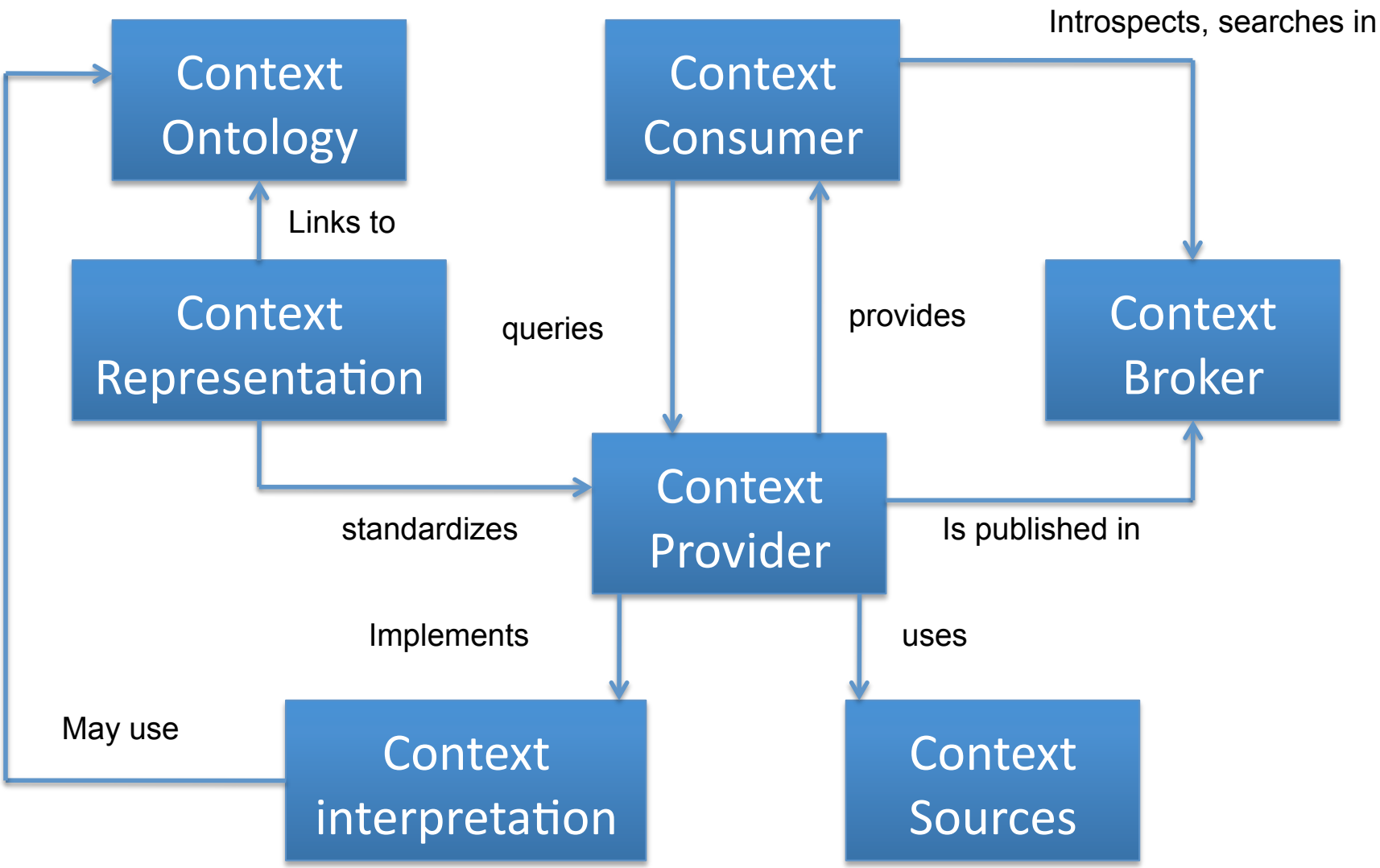
Frameworks

- Frameworks can be blackboard based
  - Blackboard is managed by context manager
  - Everything else acts as a client to the blackboard

- or widget based (Context Toolkit)
  - Widgets separate acquisition and presentation of context from the applications requiring it
- Tries to emphasize higher level inference of context information vs. low level sensor data

- Project Hydrogen uses 3 layer context architecture: adaptor, management, application layers, comm. via XML

Context Management Framework



Context Provider

- Produces new context information derived from internal or external context info
- Adheres to published interfaces to provide context to context consumers