Correctness and consistency of event-based systems
Tutorial in DEBS'2019

Speaker: Opher Etzion

The Max Stern Yezreel Valley College

Some of the work was done in collaboration with Adi Fux, Ella Rabinovich, Elior Malul, Sitvanit Ruach, Tali Yazkar-Haham.

Correctness

Dictionary definition: The quality or state of being free from error; accuracy.

An execution of a system is correct, if it conforms with the intentions of the designers.

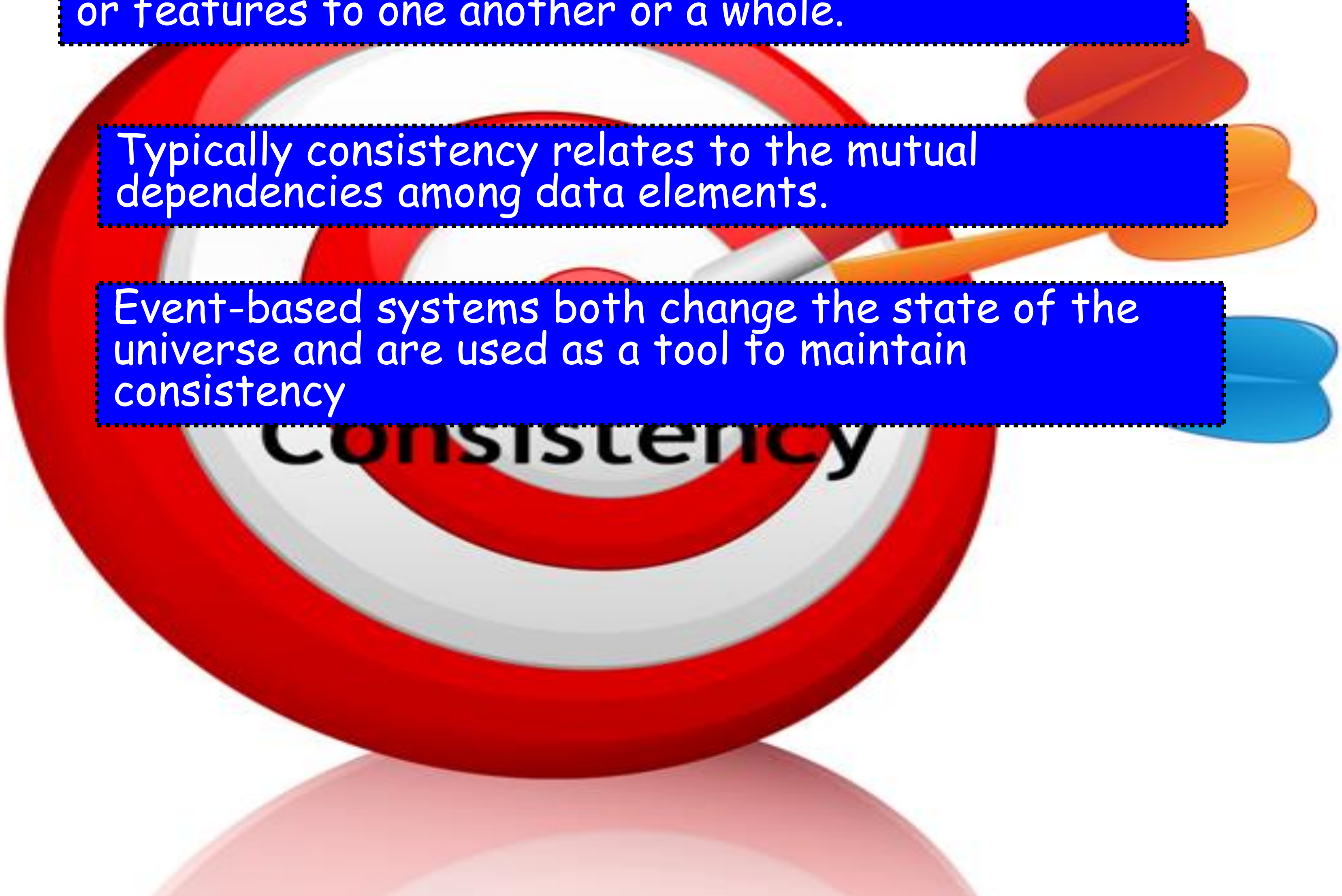
The temporal nature of event-based systems provides some correctness challenges

Consistency

Dictionary definition: agreement or harmony of parts or features to one another or a whole.

Typically consistency relates to the mutual dependencies among data elements.

Event-based systems both change the state of the universe and are used as a tool to maintain consistency

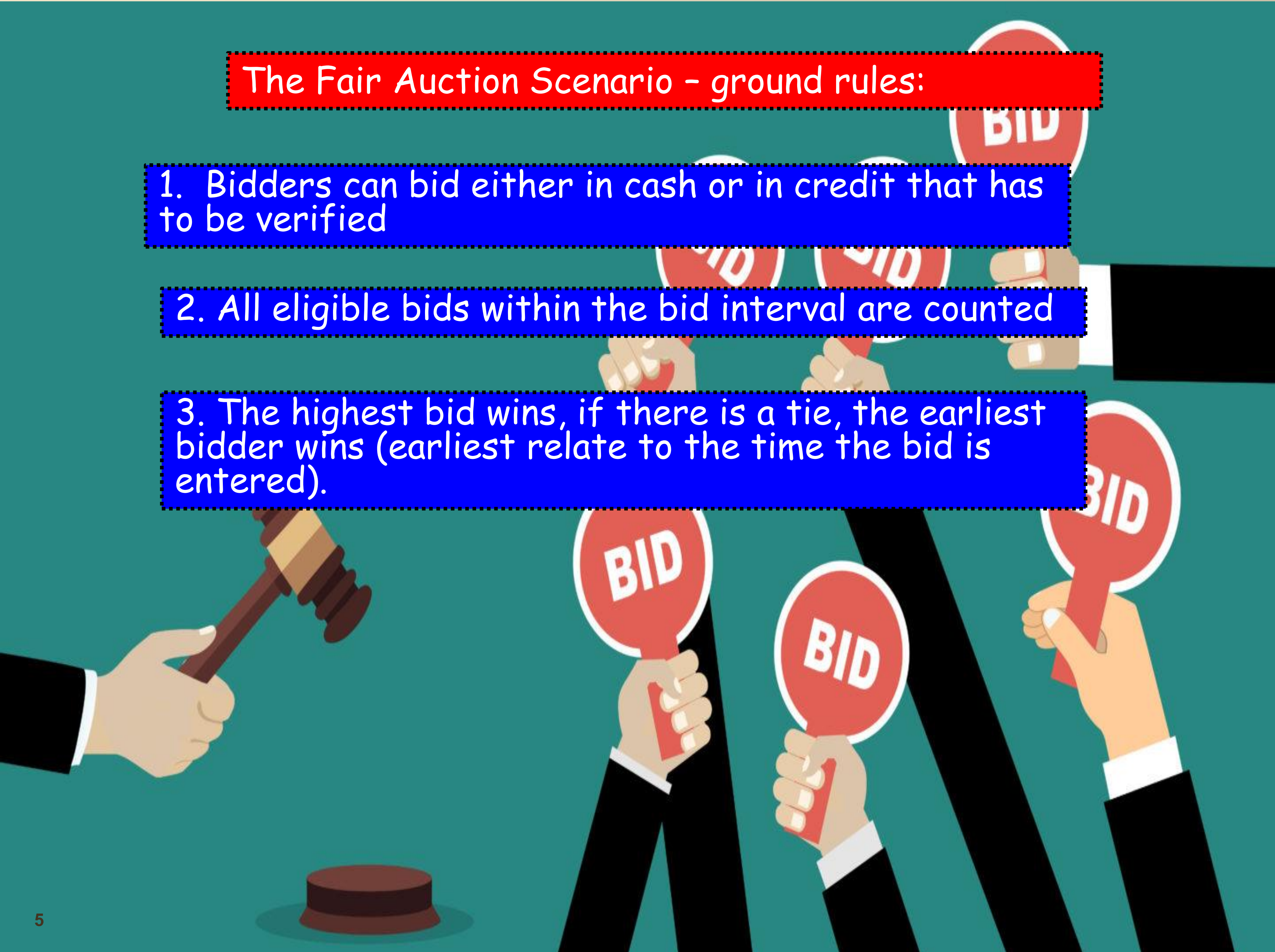


The Fair Auction Scenario – ground rules:

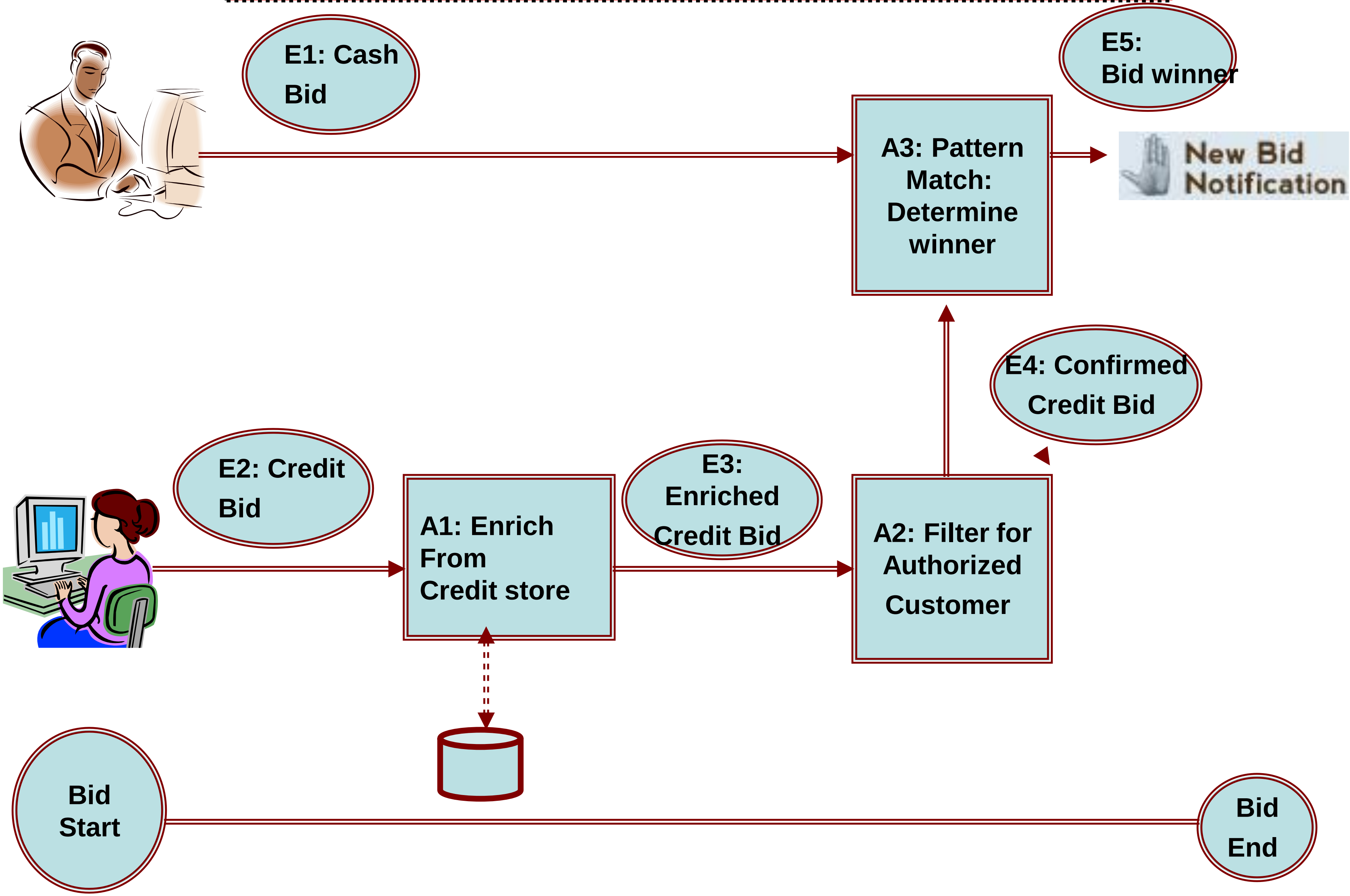
1. Bidders can bid either in cash or in credit that has to be verified

2. All eligible bids within the bid interval are counted

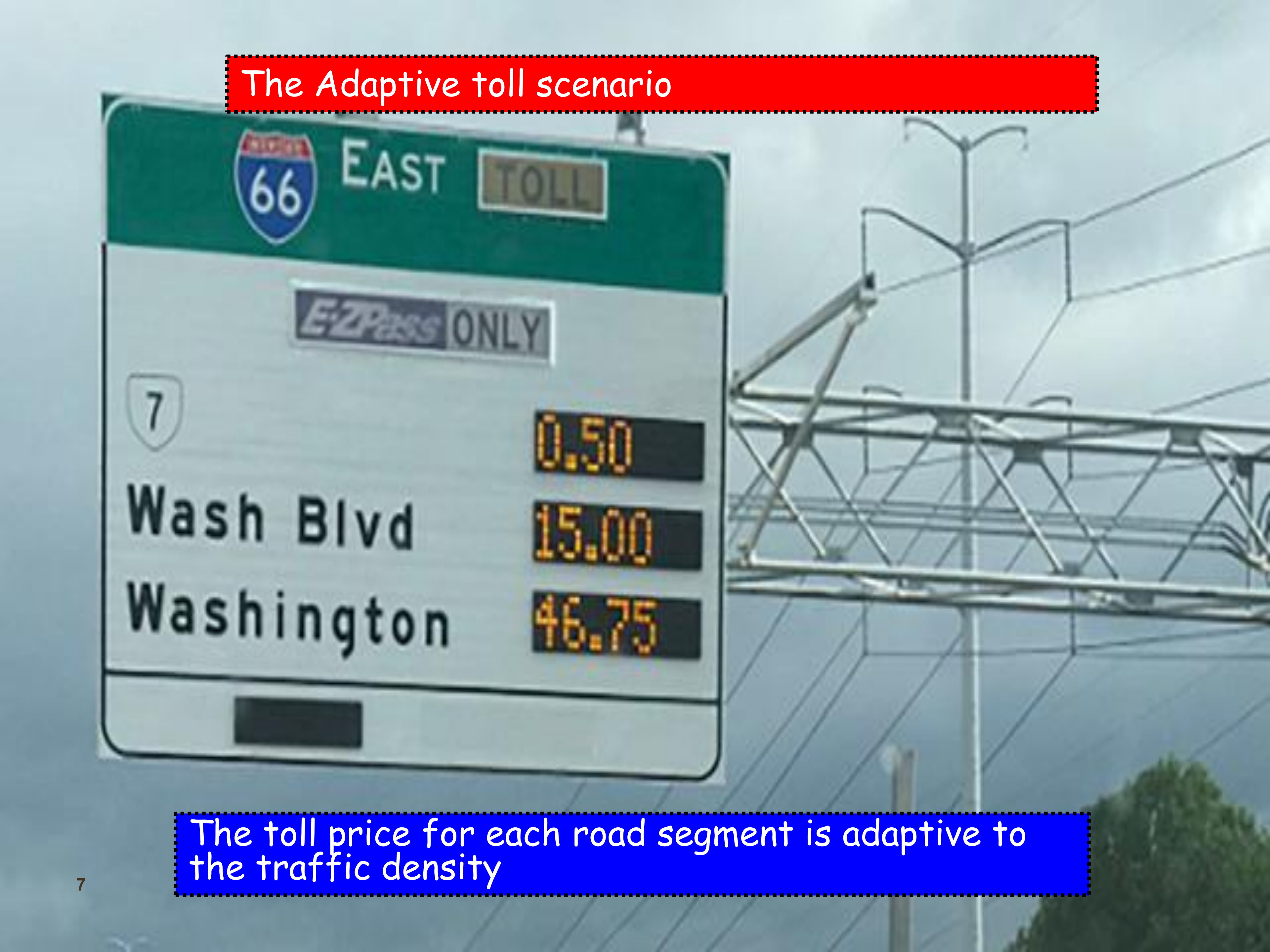
3. The highest bid wins, if there is a tie, the earliest bidder wins (earliest relate to the time the bid is entered).



The Fair Auction Scenario



The Adaptive toll scenario



The toll price for each road segment is adaptive to the traffic density

The Adaptive Toll Scenario

E1: Car in segment

A1: Segment Aggregator: calculate number of cars as entries and exits for time interval. Derive price by formula.

E2: number of cars in segment for time interval

A3: Price aggregator/ splitter: total price for next segments

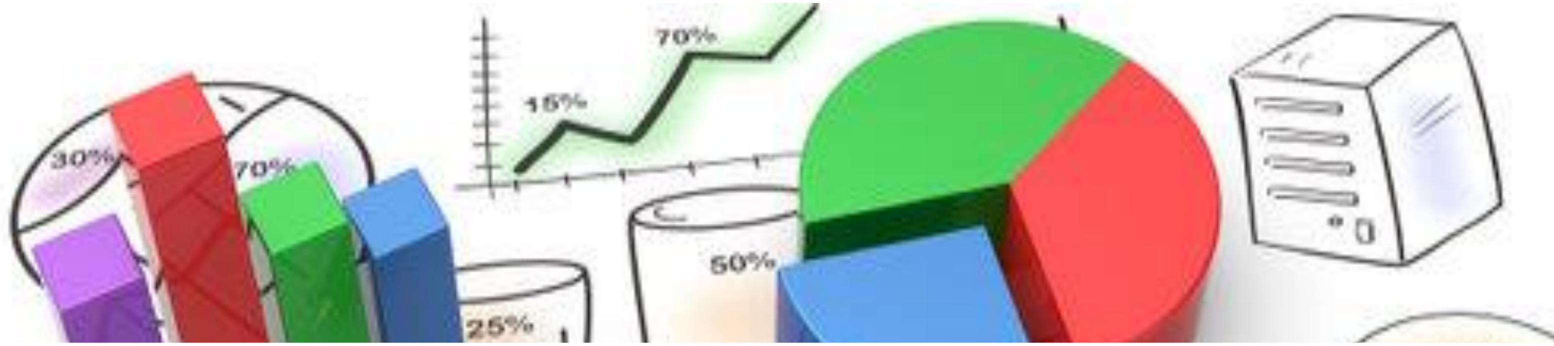
E5: next segments price

E3: Price for segment in next time interval

A2: Billing composer: Matches the driver with the price for segment in the first time driver is identified in the segment

E4: Debit account

The budget management scenario



Schema:

Project: is an entity with attributes: Overhead-Cost, Budget, Total-Cost[PDI].

Activity: is an entity with attributes: Project-Affiliation, Activity-Type, Duration, Activity-Cost[PDI].

Activity-Type: is an entity with attributes: Cost-Per-Day.

Invariants:

Activity-Cost := Duration $\times$ Cost-Per-Day + Overhead-Cost

Total-Cost := sum (Activity-Cost)

Assertion-1 Budget $\geq$ Total-Cost

Agenda

- Temporal correctness
- Tuning the semantics of event-based applications
- Data consistency and event-based systems
- Validation of event-based systems
- Conclusion

Agenda

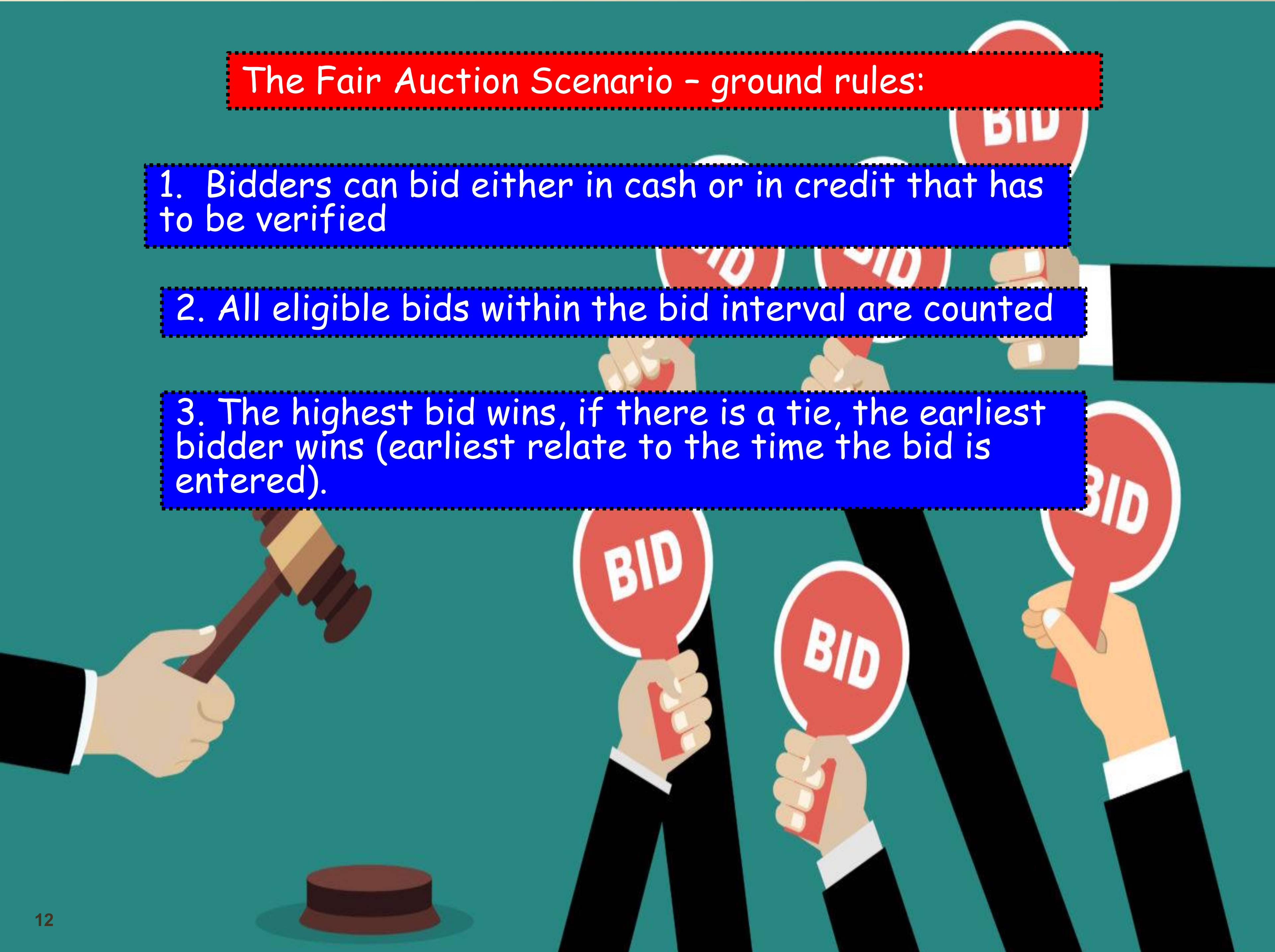
- Temporal correctness
- Tuning the semantics of event-based applications
- Data consistency and event-based systems
- Validation of event-based systems
- Conclusion

The Fair Auction Scenario – ground rules:

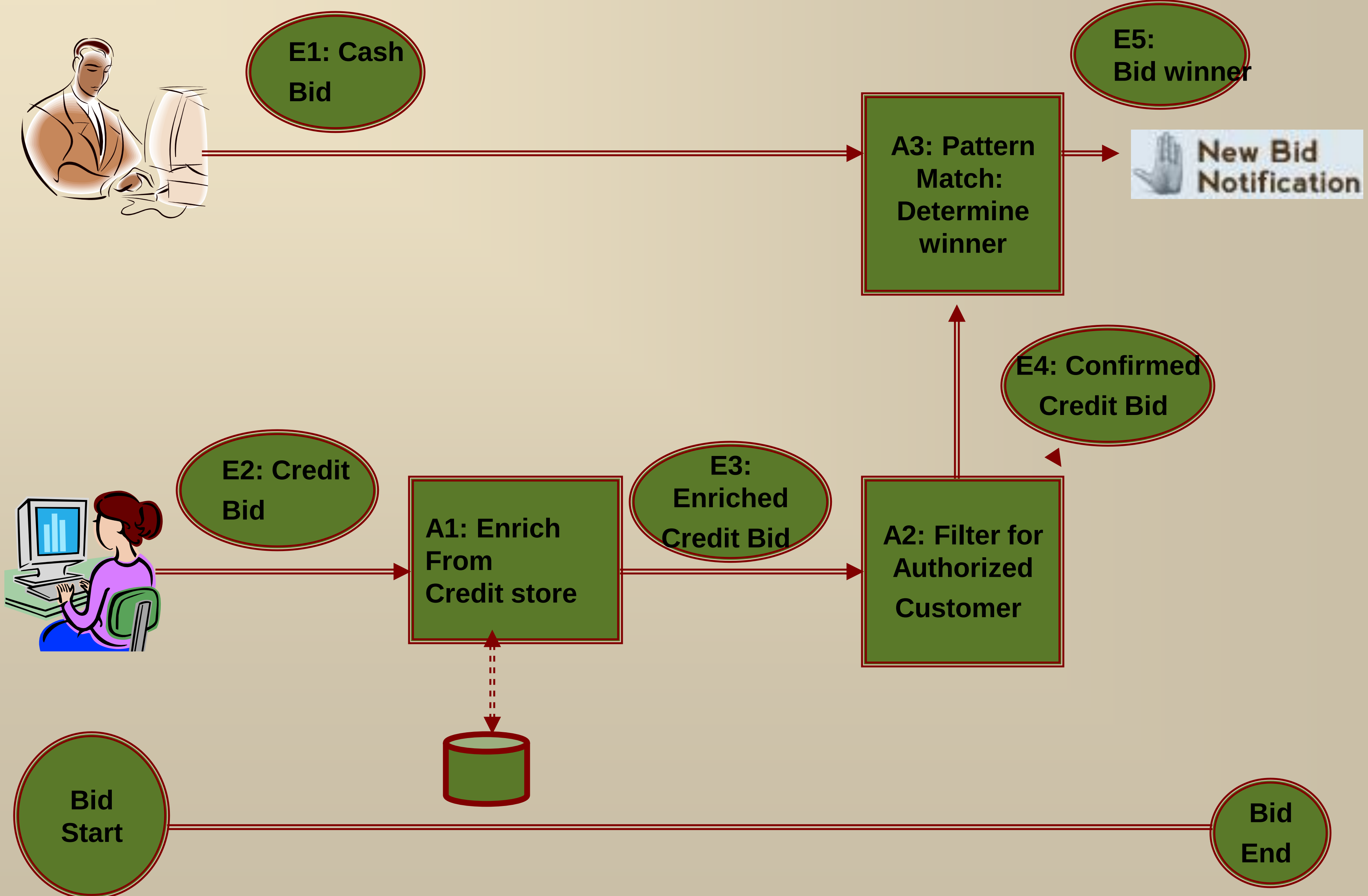
1. Bidders can bid either in cash or in credit that has to be verified

2. All eligible bids within the bid interval are counted

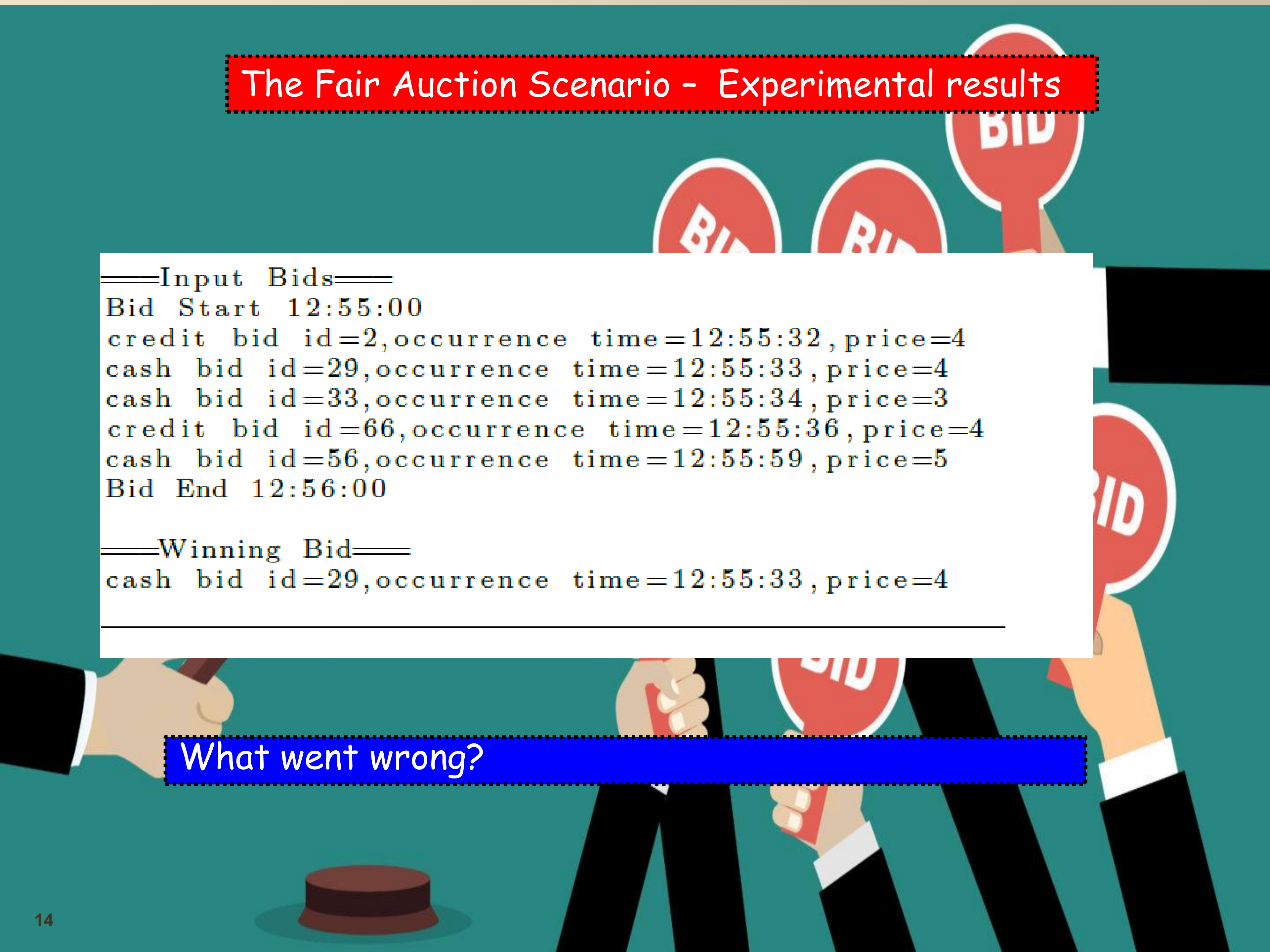
3. The highest bid wins, if there is a tie, the earliest bidder wins (earliest relate to the time the bid is entered).



The Fair Auction Scenario



The Fair Auction Scenario - Experimental results




```
===Input Bids===
Bid Start 12:55:00
credit bid id=2,occurrence time=12:55:32,price=4
cash bid id=29,occurrence time=12:55:33,price=4
cash bid id=33,occurrence time=12:55:34,price=3
credit bid id=66,occurrence time=12:55:36,price=4
cash bid id=56,occurrence time=12:55:59,price=5
Bid End 12:56:00

===Winning Bid===
cash bid id=29,occurrence time=12:55:33,price=4
```

What went wrong?

Issues related to the order of events

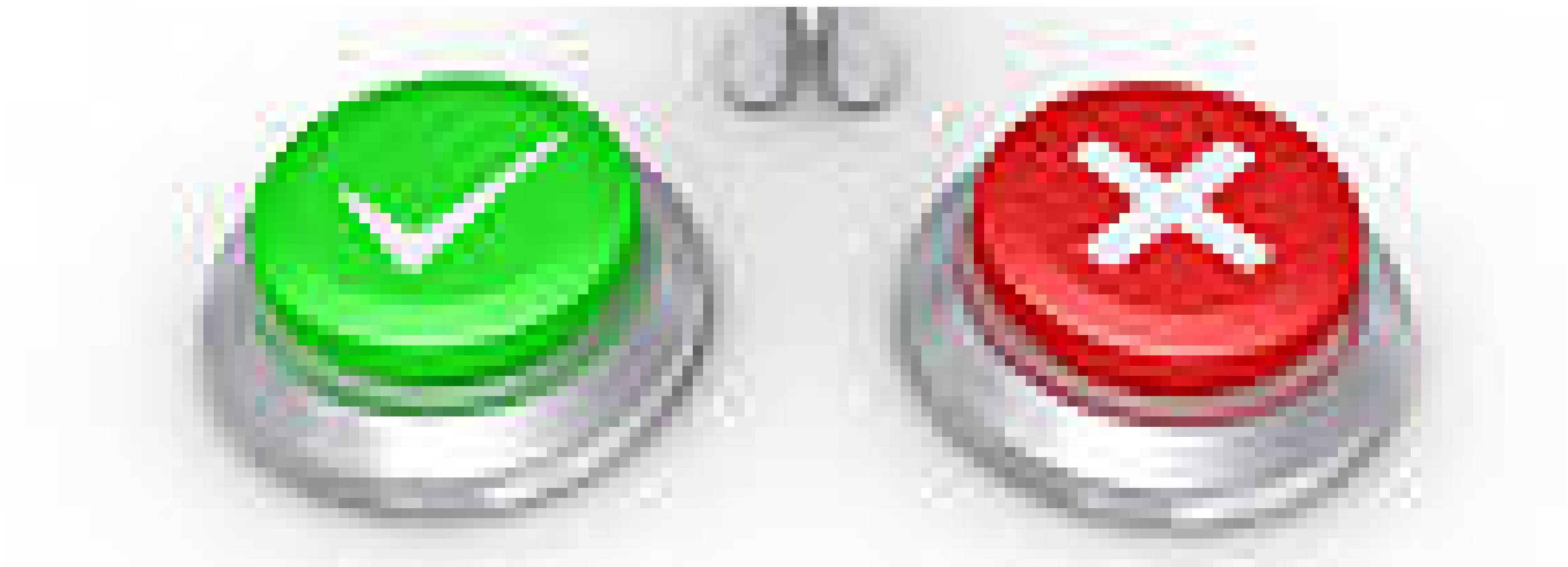
When does an event occur?

1. *Detection time based policy*—the timestamp assigned to the event is the time in which it was detected by an EPA instance. This policy is the one most commonly used by systems today.
2. *Occurrence time based policy*—the timestamp is copied from the deriver event to the derived event.
3. *End of window*—the timestamp of the event is set to be the timestamp of the context terminator in which the event was processed.

Late events arrive out-of-order

Types of windows

Window start	Window end	Inclusion condition
Open	Open	$T_s < \tau < T_e$
Open	Closed	$T_s < \tau \leq T_e$
Closed	Open	$T_s \leq \tau < T_e$
Closed	Closed	$T_s \leq \tau \leq T_e$



Issues related to the time window boundaries

Participant arrives after the terminator but with earlier occurrence time (should be part of the window)

Participant arrives before the terminator but the terminator has earlier occurrence time (should not be part of the window)

Participant arrives before the initiator, but the initiator has earlier occurrence time (the participant should be part of the window)

Participant arrives after the initiator but with earlier occurrence time (the participant should not be part of the window)

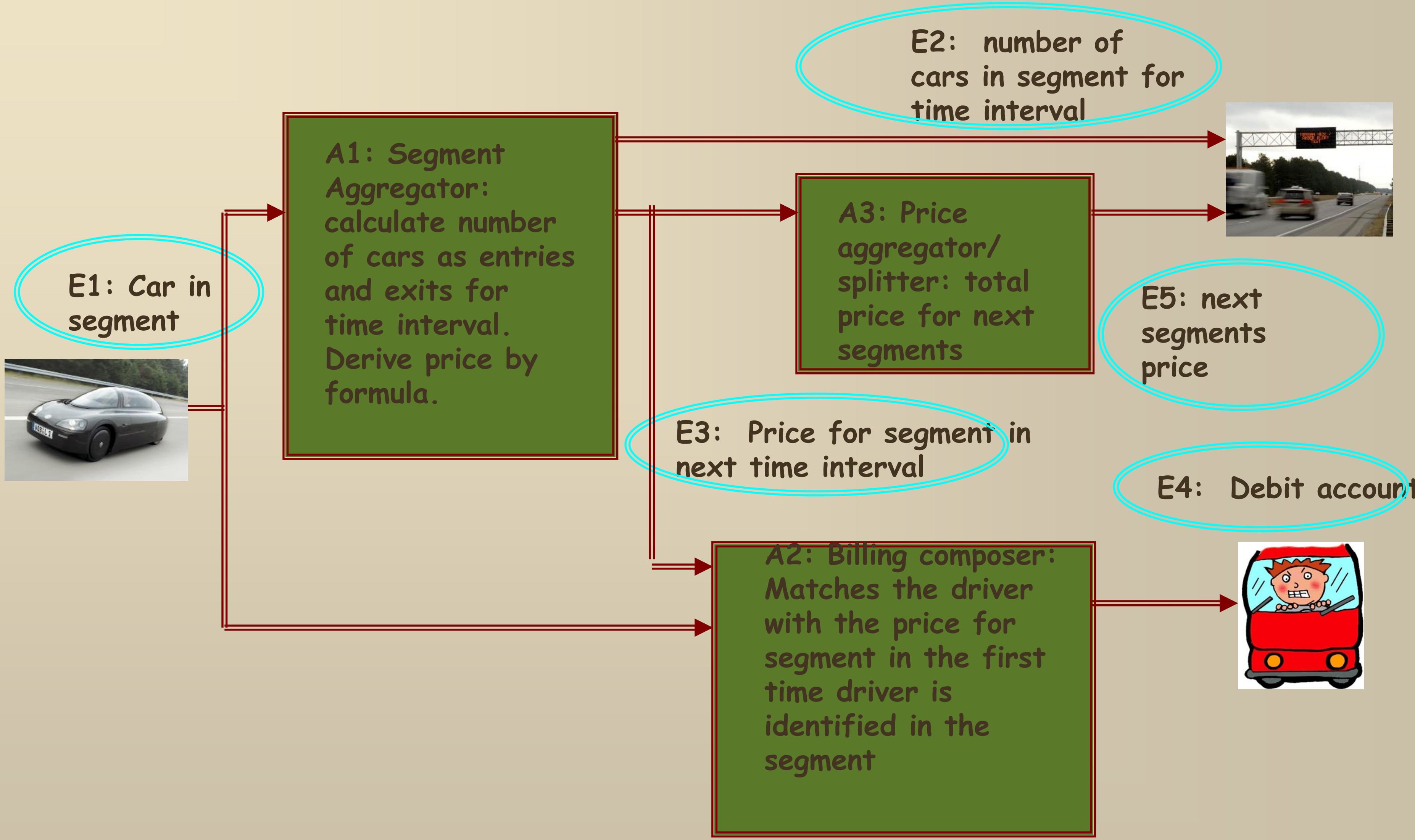
Correctness schemes (back to the Fair Auction Scenario)

The fairness scheme : Priority orders of transformed/filtered events when compared both to other derived events and to raw events can be determined according to the timing of the transformed event (In the fair bid scenario, for A3 – timing of events of type E3 are determined according to their E2 deriver).

The relative inclusion in window: An EPA that lives within a time window may still process derived events after the window ends, but not process raw events anymore, even if they arrive earlier then the derived events (In the fair bid scenario,

for A3 – at the end of the bid interval, accept events of type E3 that are still in process, but don't accept events of type E1).

The Adaptive Toll Scenario



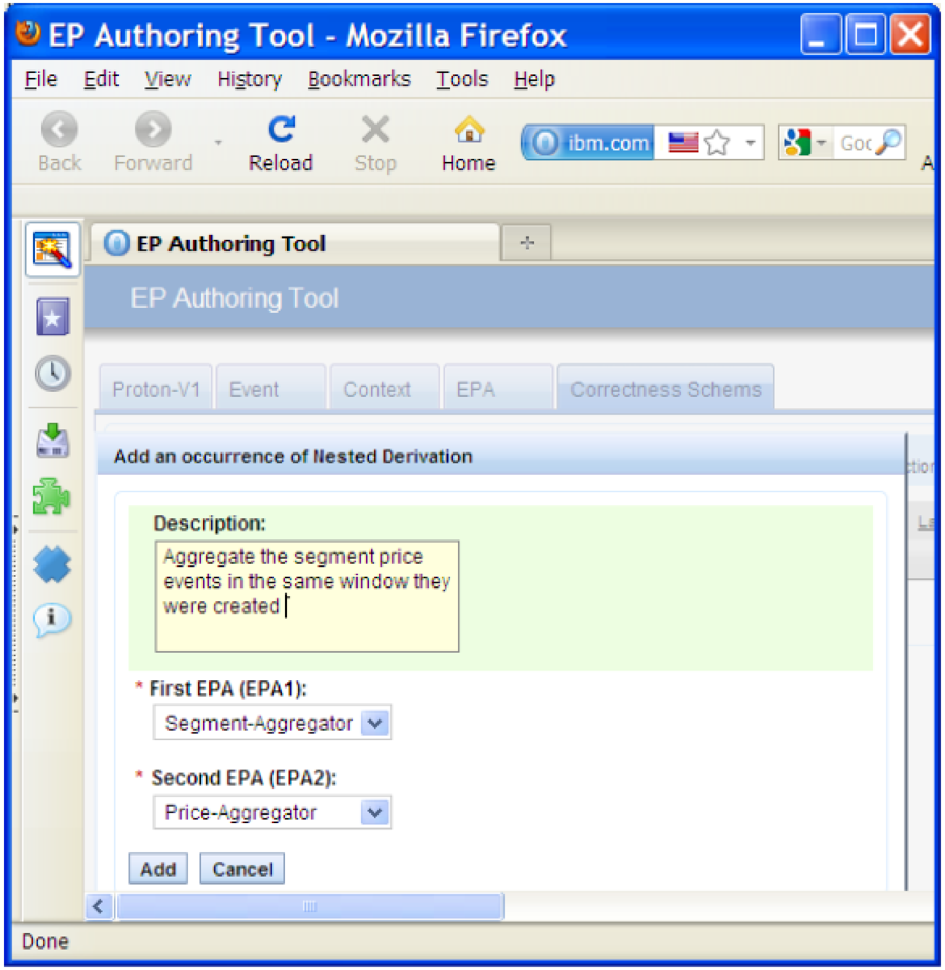
Correctness schemes (Related to the Adaptive Toll Scenario)

The nested aggregation scheme: An aggregator which aggregates events that are derived from a collection of aggregators, should ensure that its input events are temporally consistent with each other, and with the window of aggregated events (A3 for a certain window aggregates all E3 instances of the same window, though they might have been created after the window ended).

The consecutive derivation scheme: A derived event should be processed in the consecutive time window relative to its deriver.

Enforcement of schemes in run-time

The ability to add a definition of GUARD, a guard is used to enforce a scheme.



Enforcement of schemes in run-time

Buffering technique to wait for late events.

Method similar to linear 2PC, designating each EPA as "safe" relative to all guards.

Summary of temporal correctness

Temporal correctness clarity, and avoiding run-time fallacies of race conditions are very delicate and difficult issues in event-based system where the logic us temporally oriented .

Adding temporal correctness schemes and enforcing them are vital for correctness of the results.

Agenda

- Temporal correctness
- Tuning the semantics of event-based applications
- Data consistency and event-based systems
- Validation of event-based systems
- Conclusion

Who?

What Doing?

A simple example: heavy trading scenario

Given:

A stream of events of a single type, about the activity in the stock market for a certain stock.

An event is produced every 10 minutes when there is trade in the stock.

Each event consists of: quote (current stock-quote), volume (an accumulated volume of traded events within these 10 minutes).

A selection specification: “trigger an automatic trade program if the volume exceeds 300,000 3 times within an hour; pass as an argument the last quote and the sum of the 3 volume values”.

Why defining patterns is not that easy? Because we need to tune up the semantics

Event-Id	Time-Stamp	Quote	Volume
E1	9:00	33.23	
E2	9:10	33.04	320,000
E3	9:20	33.11	280,000
E4	9:30	33.01	400,000
E5	9:40	32.90	315,000
E6	9:50	33.04	320,000
E7	10:00	33.20	303,000
E8	10:10	33.33	219,000
E9	10:20	33.11	301,000
E10	10:40	33.00	210,000
E11	10:50	32.78	400,000
E12	11:00	32.70	176,000

How many times the trade programming is triggered ;
Which arguments are used in each triggering?

Semantic Tuning decisions

Decision 1: When is the pattern detection applicable?

Decision 2: Single or multiple time windows?

Decision 3: When a detected event should be acted upon?

Decision 4: Within an interval – one or many results?

Decision 5. How repeated events of the same type are handled?

Decision 6: Can a consumed event be re-consumed?

Decision 7: What determines the order of events?

Context in life

Decision 1: When is the pattern detection applicable?

In the play "The Tea house of the August Moon" one of the characters says: Pornography question of geography

- This says that in different geographical contexts people view things differently
- Furthermore, the syntax of the language (no verbs) is typical to the way that the people of Okinawa are talking

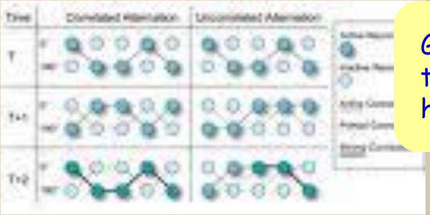
When hearing concert people are not talking, eating, and keep their mobile phone on "silent".

Context has three distinct roles (which may be combined)

Partition the incoming events

The events that relate to each customer are processed separately

Grouping events together



Grouping together events that happened in the same hour at the same location

Determining the processing

Different processing for Different context partitions

Context Types Examples

Context

Segmentation Oriented

"All Children 2-5 years old"
"All platinum customers"

Temporal

"Every day between 08:00 and 10:00 AM"

"A week after borrowing a disk"

"A time window bounded by TradingDayStart and TradingDayEnd events"

Spatial

"3 miles from the traffic accident location"

"Within an authorized zone in a manufactory"

State Oriented

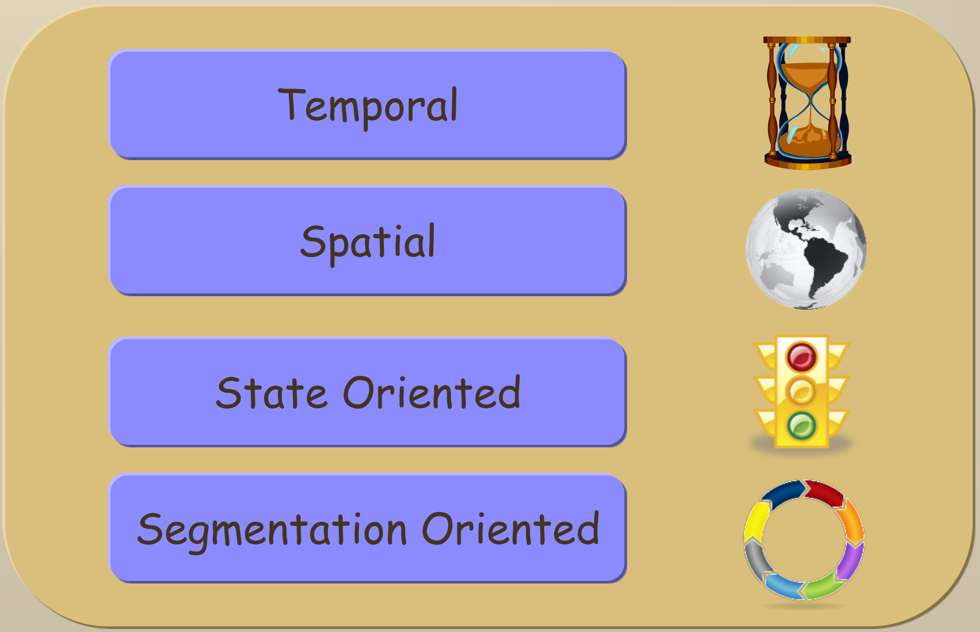
"Airport security level is red"
"Weather is stormy"

Context Definition

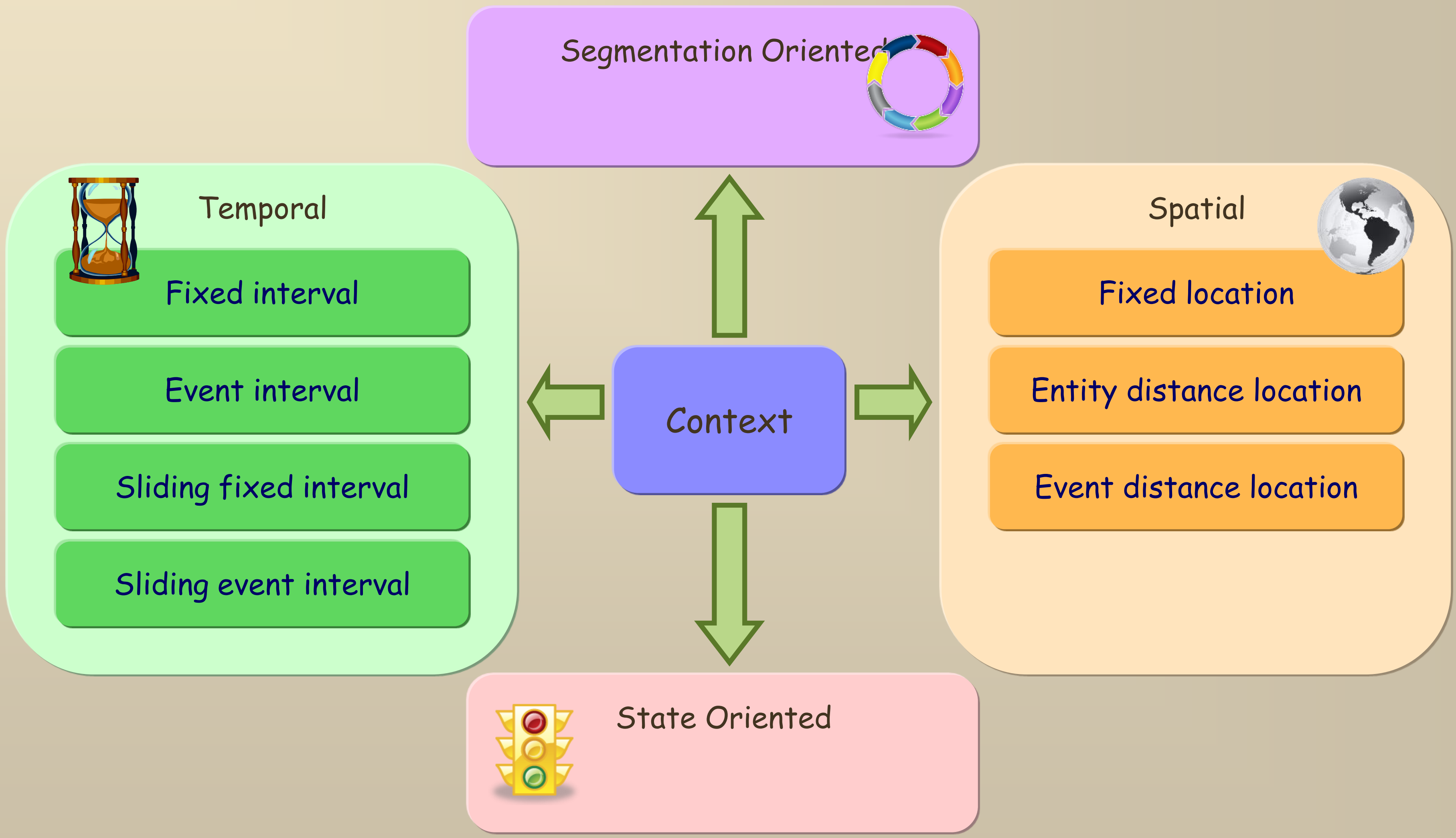
Decision 2: Single or multiple time windows?

A context is a named specification of conditions that groups event instances so that they can be processed in a related way. It assigns each event instance to one or more context partitions.

A context may have one or more context dimensions.

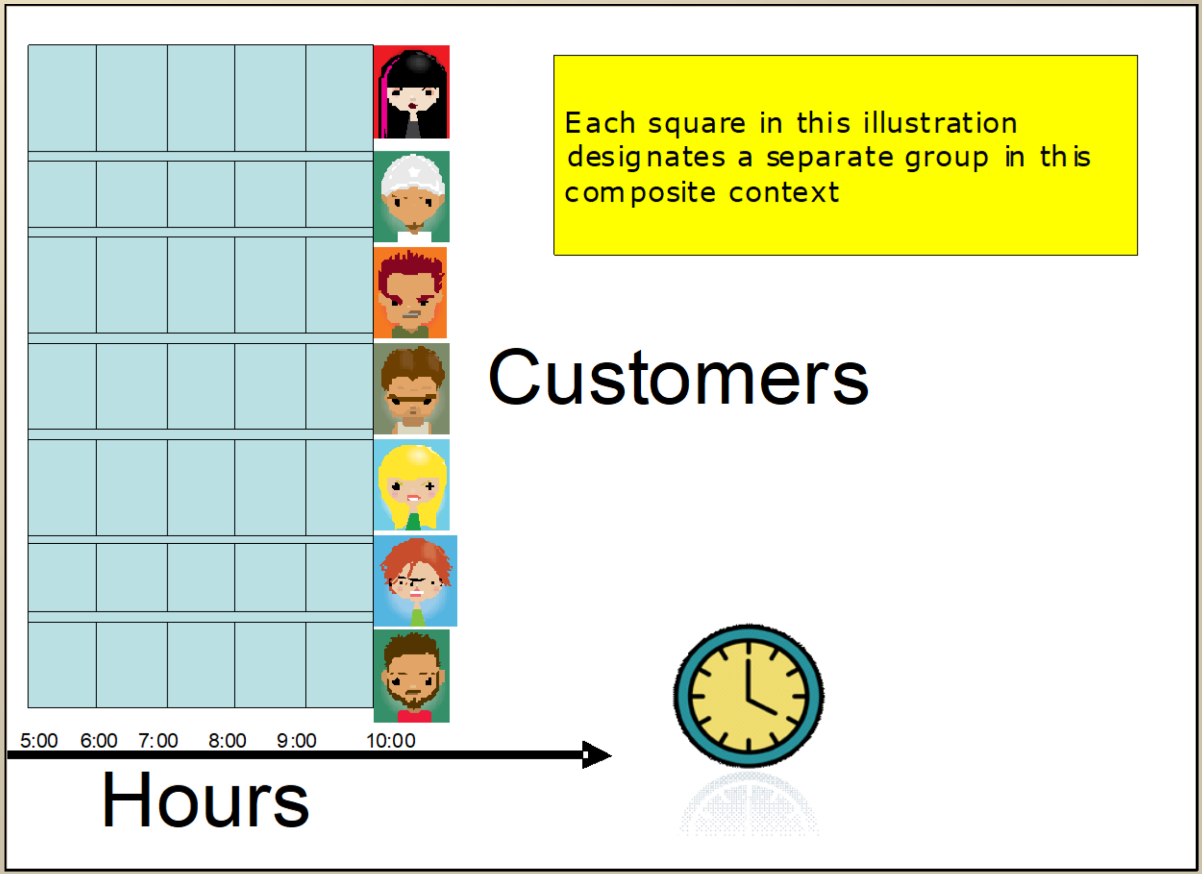


Context Types



Composite context

A composite context is a context that is composed from two or more contexts. Example: the set of context partitions for the composite context is the Cartesian product of the partition sets of its constituent contexts.



Composition of context – some observations:

The most common combination is: segmentation and temporal

Segment: customer
Temporal: Every 10 orders

The relations between the composed contexts can be – union or intersection (intersection is the more common)

State: rainy union
Temporal: every day between midnight and 6am

There may be multiple composition participants

Segment: driver
Temporal: Within 1 hour from an accident
Spatial: within 5KM from the accident

In some cases a priority is needed to disambiguate the context affiliation

Segment: customer
Temporal: Every 10 orders

Priorities in event composition

Temporal: Every 10 orders
Segment: customer

The temporal context has higher priority

First create group of 10 orders and this group by customer

Priorities in event composition

Segment: customer
Temporal: Every 10 orders

The segmentation context has higher priority

First group of by customer and then count 10 orders for each customer

Policies

A policy is a tool to tune up the semantics and disambiguate semantic decisions

Pattern policies

***Evaluation policy*—This determines when the matching sets are produced.**

***Cardinality policy*—This determines how many matching sets are produced within a single context partition.**

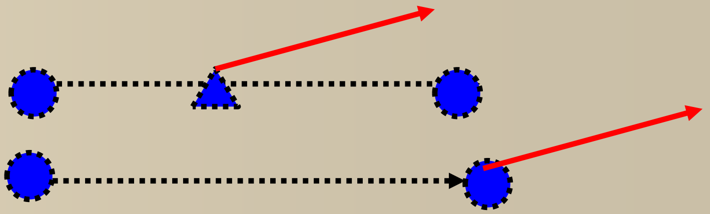
***Repeated type policy*—This determines what happens if the matching step encounters multiple events of the same type.**

***Consumption policy*—This specifies what happens to a participant event after it has been included in a matching set.**

***Order policy*—This specifies how temporal order is defined.**

Decision 3: When a detected event should be acted upon?

Evaluation policy



An *evaluation policy* is a semantic abstraction that determines when the matching process is to be evaluated.

The evaluation policy lets you choose whether a *Pattern detect* agent generates output incrementally, or only at the end of the temporal context. The two policies are:

Immediate—The pattern is tested for each time a new event is added to the participant event set.

Deferred—The pattern is only tested for when the agent's temporal context partition (window) closes.

Cardinality policies

Decision 4: Within an interval – one or many results?

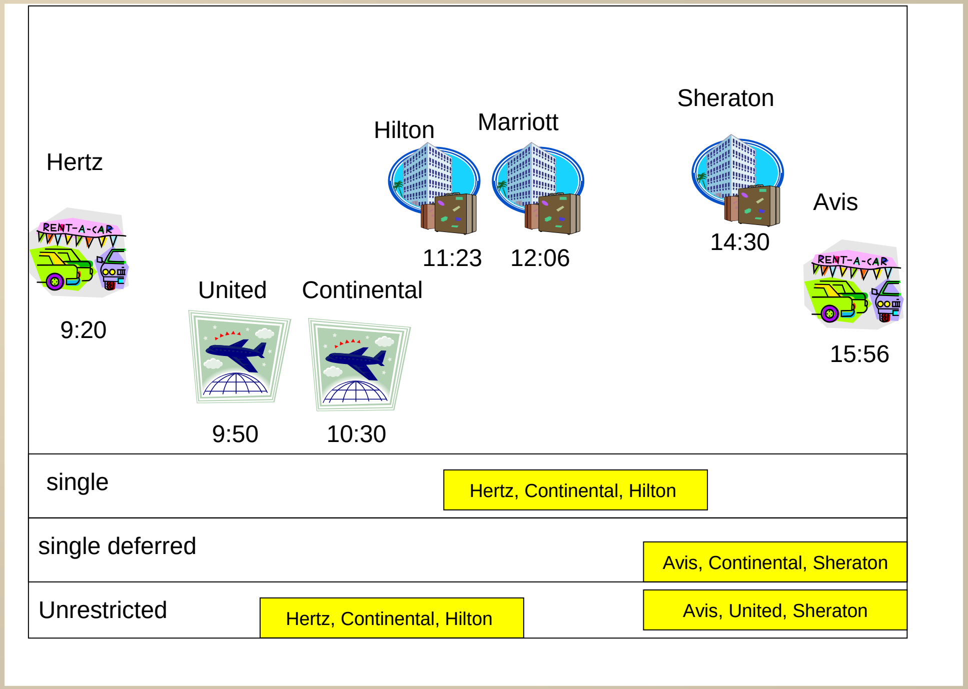
A ***cardinality policy*** **is a semantic abstraction that controls how many matching sets are created. The possible policies are:** ***single*****,** ***unrestricted*** **and** ***bounded*****.**

The various policies are:

Single**—Only one matching set is generated. When this has been done no further action is performed within this context partition, so no more matching sets are generated.**

Unrestricted**—Under this policy there are no restrictions on the quantity of matching sets that can be generated.**

Bounded**—This policy specifies an upper bound on the number of matching sets that can be generated within a context partition. The** ***Pattern detect*** **agent continues generating matching sets until it reaches this bound.**



Repeated type policies

Decision 5. How repeated events of the same type are handled?

A repeated type policy is a semantic abstraction that defines the behavior of a Pattern detect agent when an excess type condition occurs. The possible policies are: override, every, first, last, with maximal value, with minimal value.

Override The participant event set keeps no more instances of any event type than the number implied by the relevant event types list. If a new event instance is encountered and the participant set already contains the required number of instances of that type, then the new instance replaces the oldest previous instance of that type.

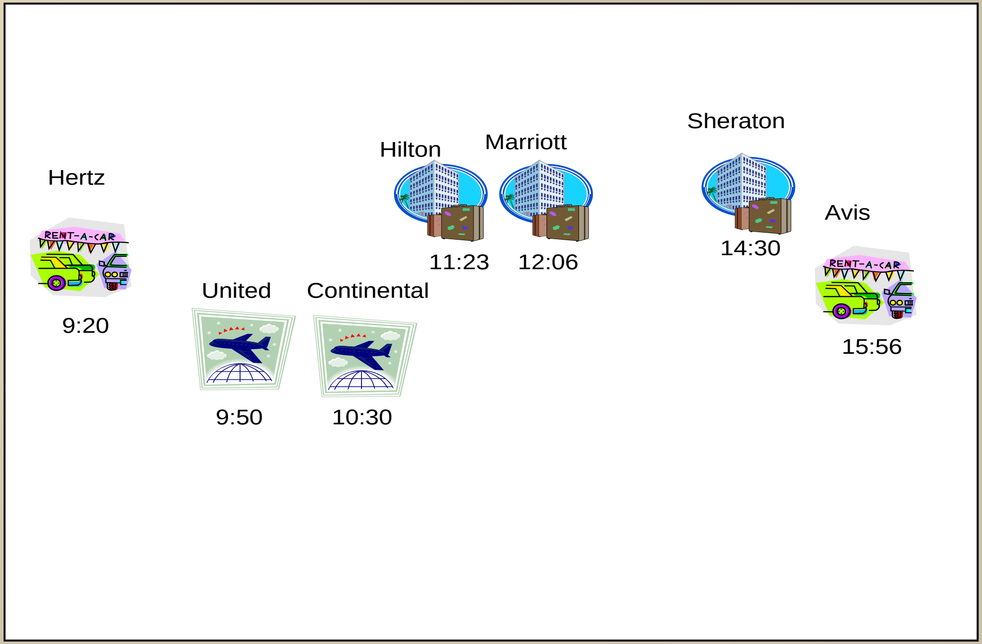
Every: Every instance is kept in the participant event set, so that all possible matching sets can be produced.

First Every instance is kept in the participant event set, but only the earliest instances of each type are used for matching.

Last Every instance is kept, but only the latest instances of each type are used for matching.

With maximal value <attribute name> Every instance is kept, but only the event or events with the maximal value of the specified attribute are used for matching.

With minimal value <attribute name> Every instance is kept, but only the event or events with the minimal value of the specified attribute are used for matching.



Consumption policies

Decision 6: Can a consumed event be re-consumed?

A *consumption policy* is a semantic abstraction that defines whether an event instance is consumed as soon as it is included in a matching set, or whether it can be included in subsequent matching sets. Possible consumption policies are: consume, reuse and bounded reuse.

The consumption policies are quite straightforward:

Consume—Under this policy each event instance is removed from the participant event set once it has been included in a matching set. This means that it cannot take part in any further matching for this particular pattern within the same context.

Reuse—under this policy, an event instance can participate in an unrestricted number of matching sets.

Bounded reuse—under this policy, you can specify the number of times that an event can be used in matching sets for this particular pattern within the same context.

Order policies

Decision 7: What determines the order of events?

An *order policy* is a semantic abstraction that defines the meaning of the << temporal order of the event instances in the participant event set. The possible policies are: by occurrence time, by detection time, by user-defined attribute, or by stream position.

The order policy is applicable to all temporal or spatiotemporal patterns. The possible policies are:

By occurrence time—The order of events in the participant event set is determined by comparing their *occurrence time* attributes, so that the order reflects the order in which the events happened in reality (as accurately as the temporal granularity allows).

By detection time—The order of events in the participant event set is determined by comparing their *detection time* attributes, that is the order in which events are detected by the event processing system. Note that this order may not be identical to the order in which events happened in reality.

By user-defined attribute—Some event payloads contain a timestamp, sequence number or some other attribute that increases over time, and this can be used to determine the order. For example the Delivery Request events in the Fast Flower Delivery application could be ordered using their Delivery Time attribute.

By stream position—In this case the order to be used is the order in which the events are delivered to the event processing agent from the channel that feeds it. Some channel implementations are designed so that this order is the same as the order in which events were delivered to the channel

Agenda

Temporal correctness

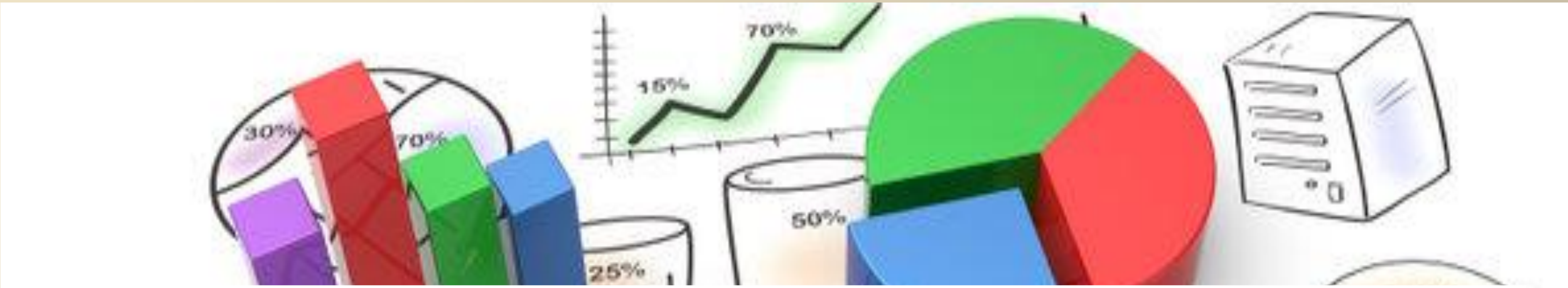
Tuning the semantics of event-based applications

Data consistency and event-based systems

Validation of event-based systems

Conclusion

The budget management scenario



Schema:

Project: is an entity with attributes: Overhead-Cost, Budget, Total-Cost[PDI].

Activity: is an entity with attributes: Project-Affiliation, Activity-Type, Duration, Activity-Cost[PDI].

Activity-Type: is an entity with attributes: Cost-Per-Day.

Invariants:

Activity-Cost $:=$ Duration $\times$ Cost-Per-Day $+$ Overhead-Cost

Total-Cost $:=$ sum (Activity-Cost)

Assertion-1 Budget $\geq$ Total-Cost

Using event processing to maintain data dependency

Event is a database event: Insert, modify, delete.

According to the dependency formulae – a derived event is created to maintain the dependency.

Example: The affiliation of Activity A is modified from project P1 to project P2 is modified.   There are two events created:

Modify P1 to subtract the cost of A to Total-Cost

Modify P2 to add the cost of A to Total-Cost

Data dependencies

Unconditional Direct Dependency:

The value d2 is derived directly and unconditionally from the value of the corresponding d1.
Example*: d2:= d1 * 1.17,* this is the case of calculating prince under tax.

Data dependencies

Conditional Direct Dependency:

The value d2 is derived directly from the value of d1, if a condition is satisfied. Example: d2:= d1 * 1.17 when d3 = 'taxable". In this case, another data item (d3) participates and stands for the taxable / non-taxable property of the product that d1 refers to.

Data dependencies

Indirect dependency:

A data element that participates in a condition issues another type of indirect dependency. Example: if d1:= d2 + d3 when d4 > d5. In this case d1 and d2 issue conditional direct dependencies, while d4 and d5 issue indirect dependencies.

Data dependencies

Aggregated dependency:

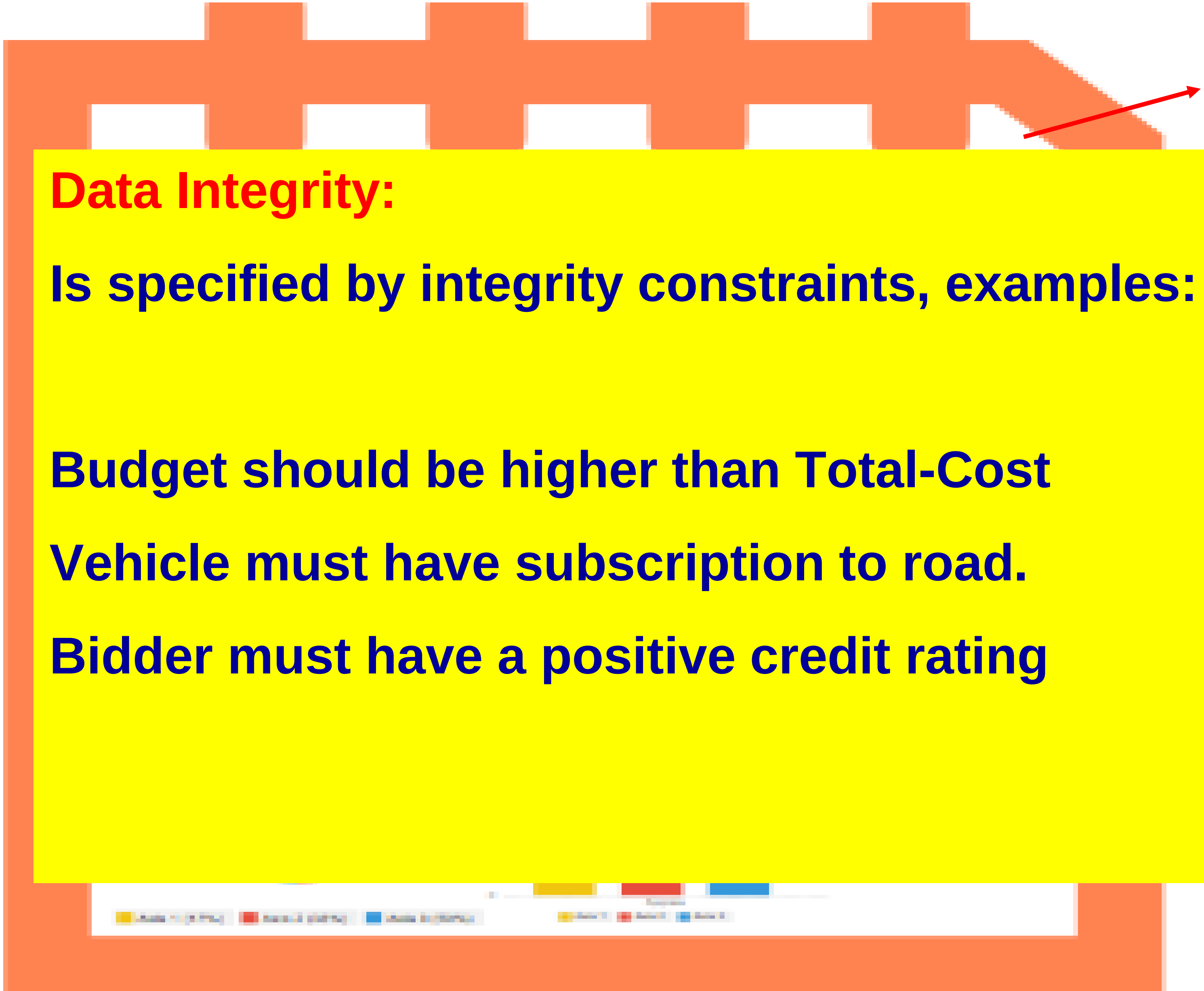
The value of d2 is an aggregation of the value of a collection of values of d1. For example, d2:= count (d1). Again, this dependency may be conditioned or unconditioned

Data dependencies

Aggregated dependency:

The value of d2 is an aggregation of the value of a collection of values of d1. For example, d2:= count (d1). Again, this dependency may be conditioned or unconditioned

Data Integrity



Data Integrity:

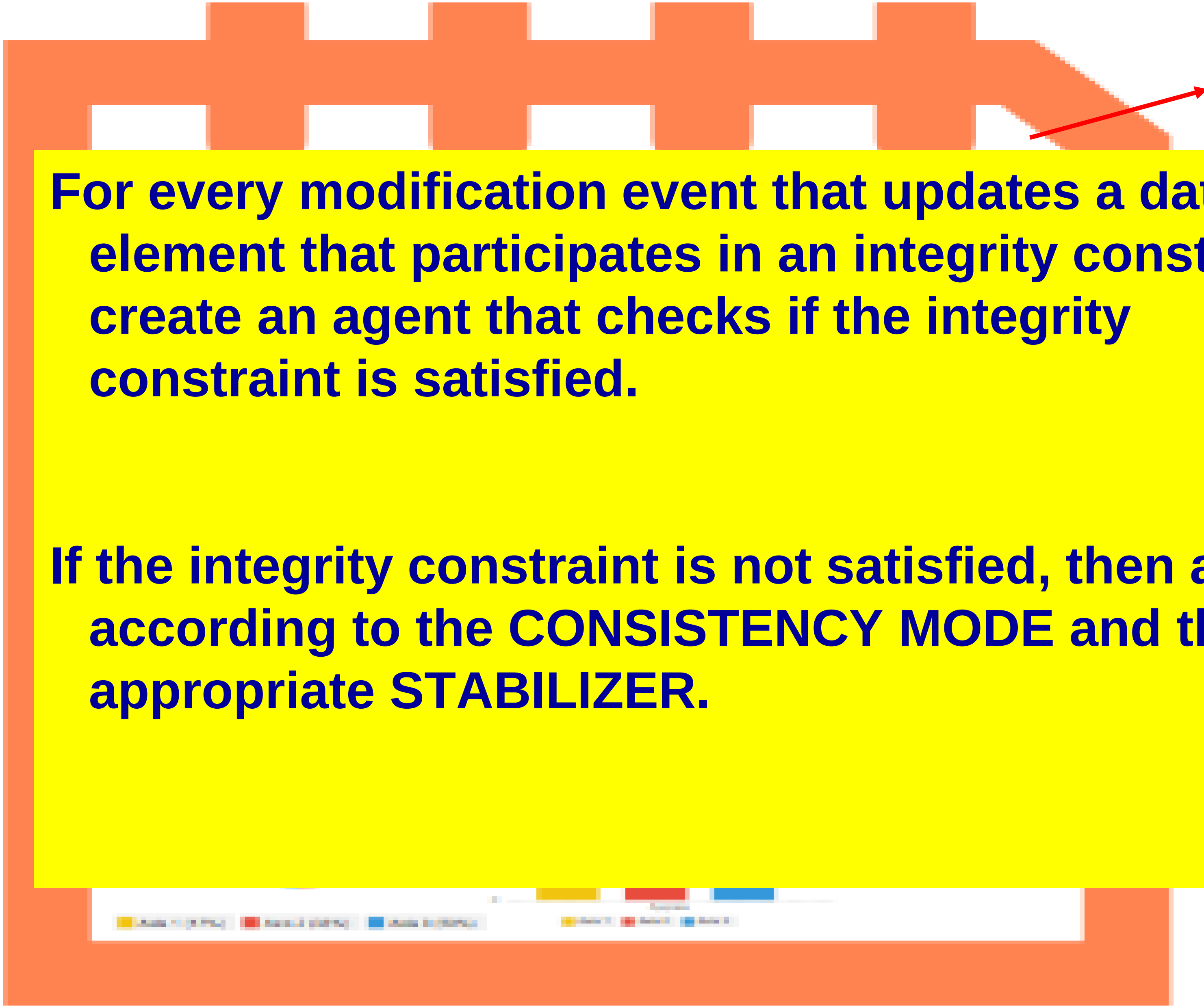
Is specified by integrity constraints, examples:

Budget should be higher than Total-Cost

Vehicle must have subscription to road.

Bidder must have a positive credit rating

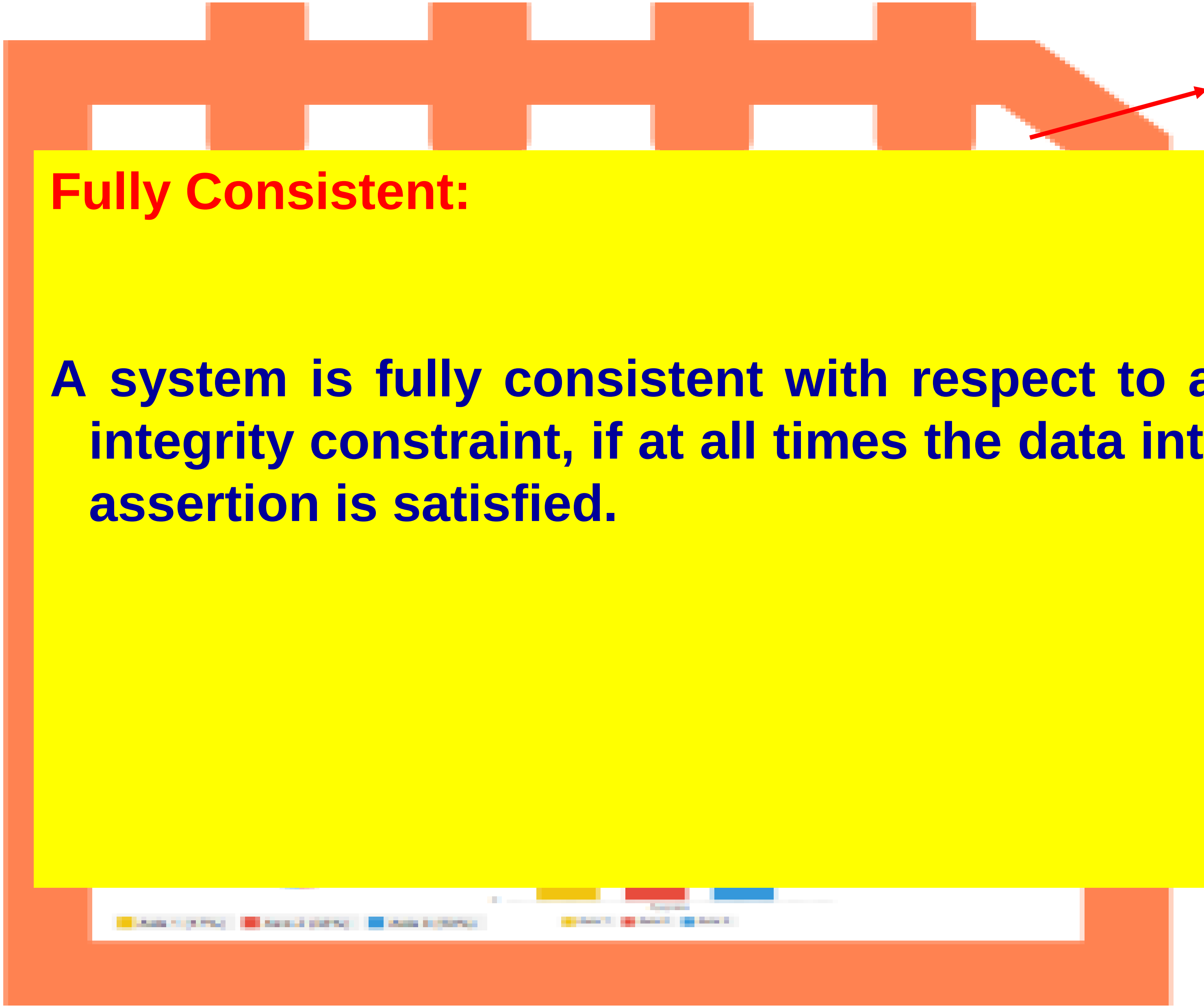
Data Integrity and event processing



For every modification event that updates a data element that participates in an integrity constraint, create an agent that checks if the integrity constraint is satisfied.

If the integrity constraint is not satisfied, then act according to the CONSISTENCY MODE and the appropriate STABILIZER.

Consistency Modes



Fully Consistent:

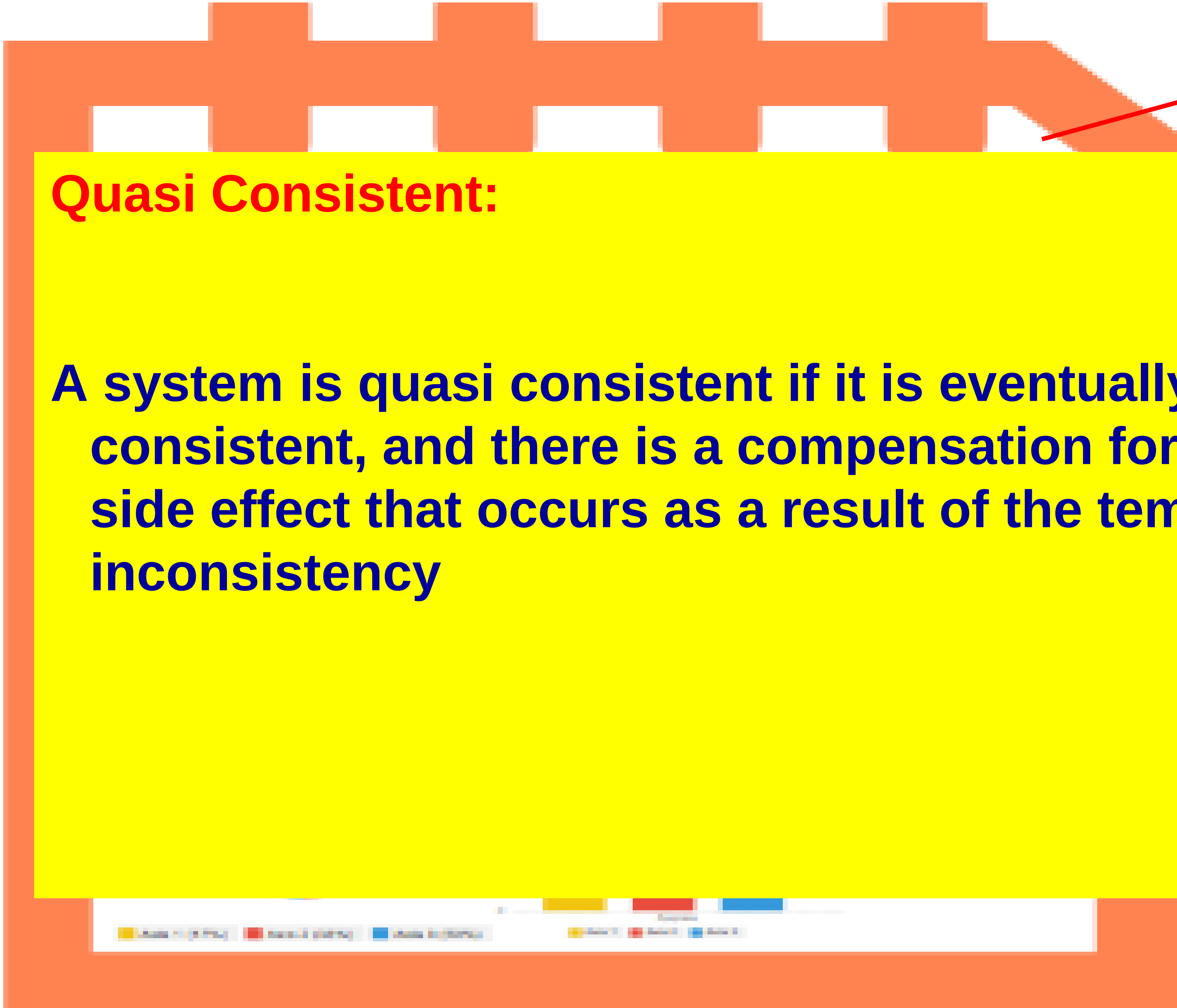
A system is fully consistent with respect to a data integrity constraint, if at all times the data integrity assertion is satisfied.

Consistency Modes

Eventually Consistent:

A system is eventually consistent with respect to a data integrity constraint, if eventually the data integrity assertion is satisfied. There can be a time interval in which the assertion is not satisfied.

Consistency Modes



Quasi Consistent:

A system is quasi consistent if it is eventually consistent, and there is a compensation for any side effect that occurs as a result of the temporary inconsistency

Consistency Modes

Loosely Consistent:

A system is loosely consistent with respect to a data integrity constraint, if the assertion is not necessarily enforced.

Stabilizers

Self Stabilization policies:

While the classic consistency theory advocates rollback of any update that causes violation, there are several other possibilities denoted as STABILIZERS

Stabilizers

A simple example:

Consistency Assertion:

Sum of salaries for the department must not exceed the department budget.

A new employee E is hired in department D.

The budget for the department is 2M$, with total salaries of $1.9M. The salary of employee E is assigned as $110K, thus with the addition of this employee, the consistency assertion is violated.

Stabilizers

Restrict (the conservative stabilizer):

Rollback to erase any effect of the update by aborting the transaction or initiating a compensation transaction.

In our example: the employee hiring is denied.

Stabilizers

Repair Transaction:

Repair the input transaction in a minimal way, such that the consistency assertion would be satisfied.

In this case the employee can be hired with a salary of $100K, the minimal change that does not violate the consistency assertion.

Stabilizers

Cascade

Leave the input transaction as is, and minimally change the value of any other data item to satisfy the consistency assertion.

In our example, raise the department salaries budget to $2.01M

Stabilizers

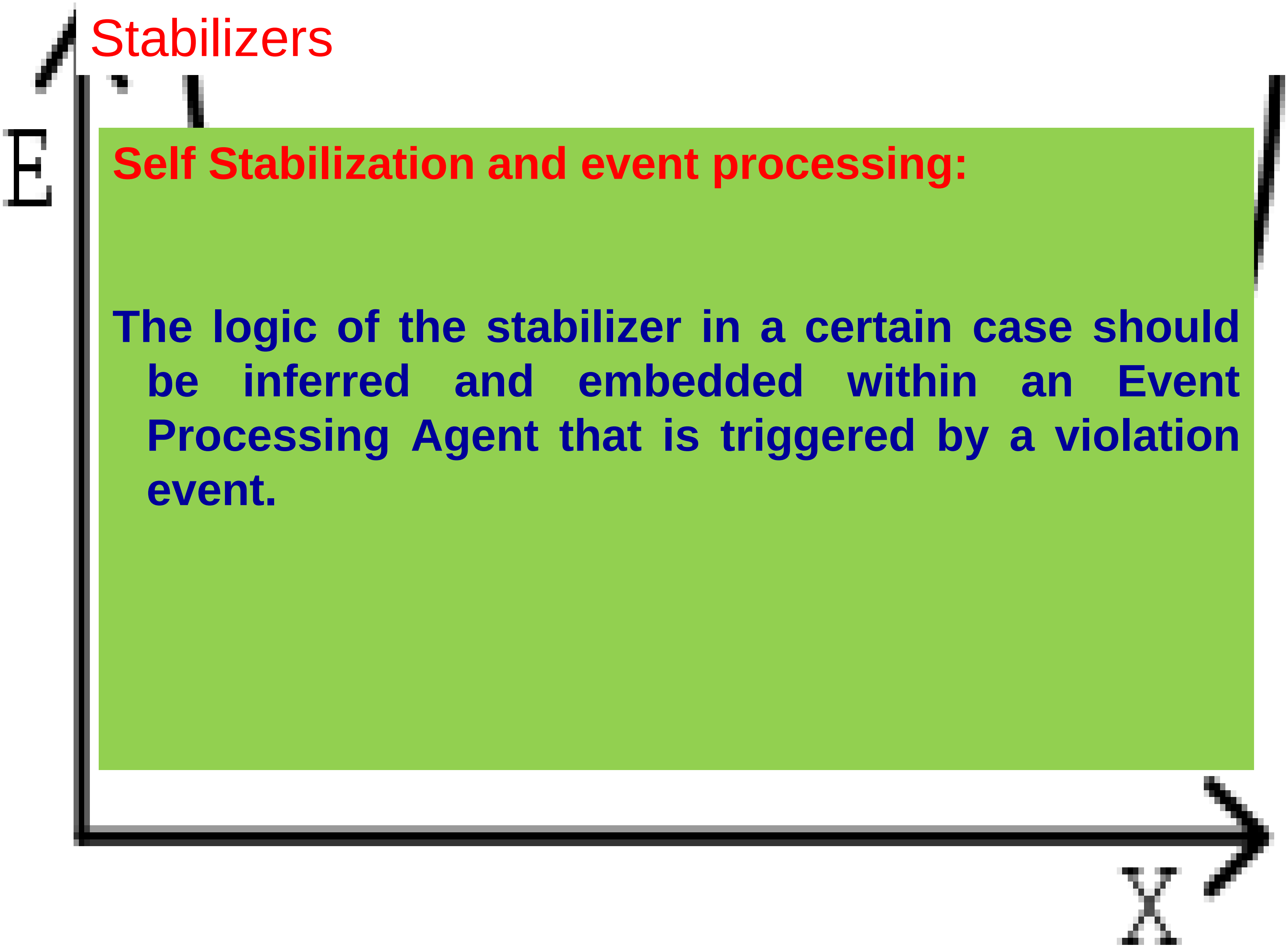
Forgive:

Allow the violation to persist

Stabilizers

Self Stabilization and event processing:

The logic of the stabilizer in a certain case should be inferred and embedded within an Event Processing Agent that is triggered by a violation event.



Agenda

- Temporal correctness
- Tuning the semantics of event-based applications
- Data consistency and event-based systems
- Validation of event-based systems
- Conclusion

Validation and verification

Event processing applications development is an evolutional process, often done bottom-up

- Modifications and extensions to existing application are very common → continuous validation and verification is required

Event processing poses challenges when applying state-of-the-art software verification techniques

- Comprises strong temporal semantics
- …

Analyzing the behavior of big applications (hundreds of assets) by manual inspection is often impractical

What are the validation point?

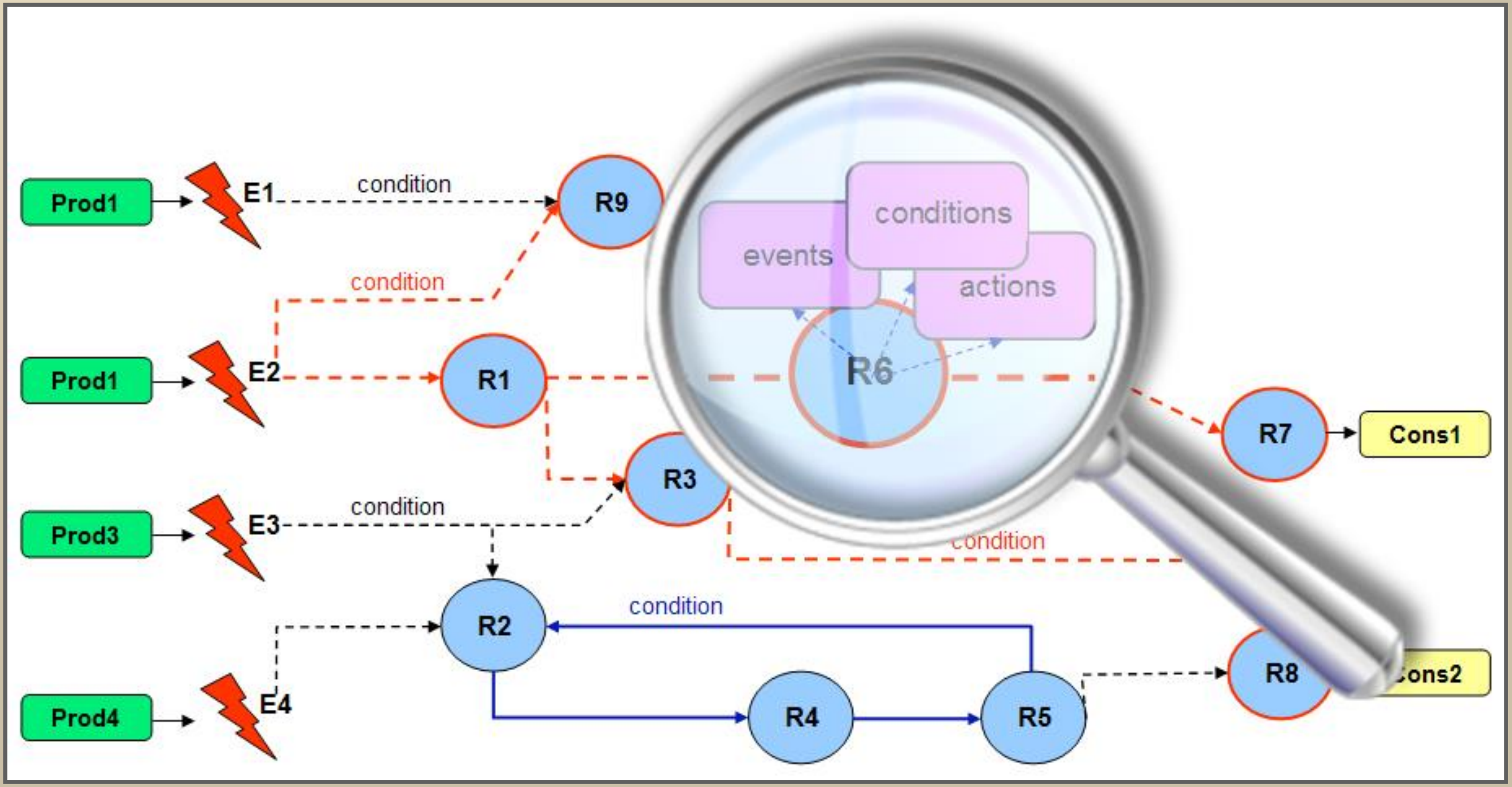
Changing a certain event, what are the application artifacts affected?

What are all possible ways to produce a certain action (derived event)?

There was an event that should have resulted in a certain action, but that never happened!

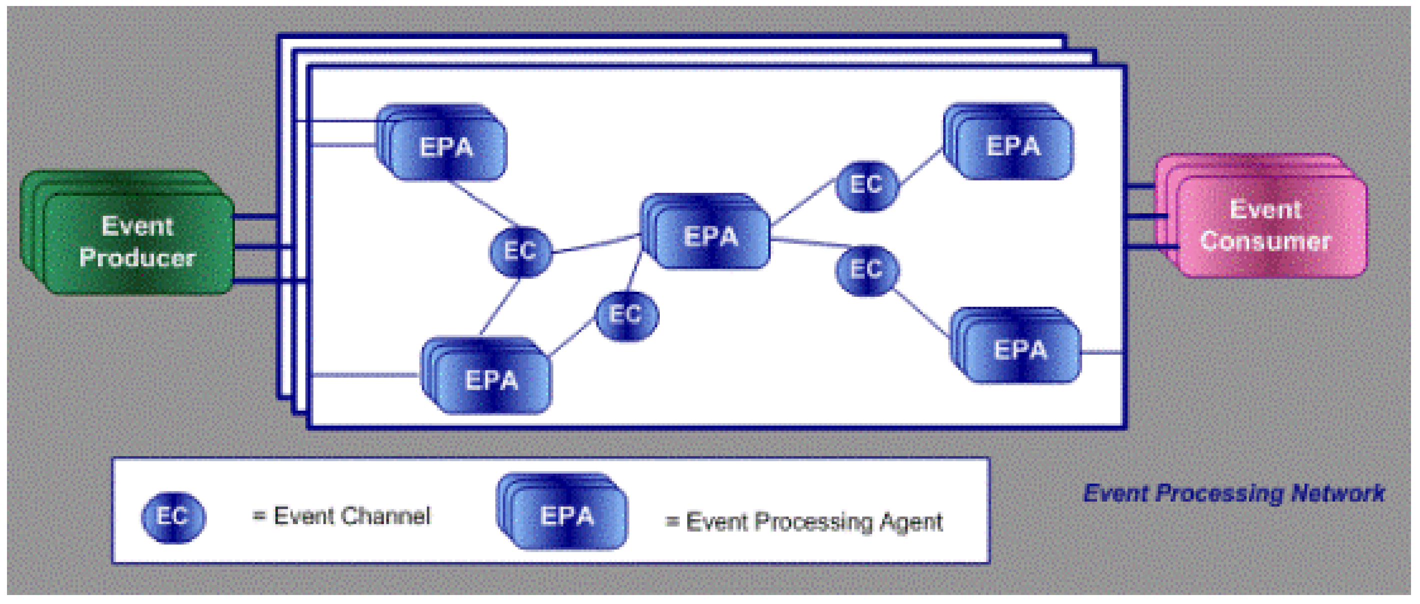
“Wrong” action was taken, how did that happen?

…



The Verification Model

- Event type
- Event Processing Agent (EPA)
- Producer, Consumer
- Channel



Moxey C. et al: A Conceptual model for Event Processing Systems, an IBM Redguide publication.
http://www.redbooks.ibm.com/abstracts/redp4642.html?Open

Analysis Techniques

Static Analysis

Navigate through mass of information wisely
Discover event processing application artifacts dependencies and changing rules with confidence

Build-time
Development phase

Dynamic Analysis

Compare the actual output against the expected results
Explore rules coverage with multiple scenarios invocation
System consistency tests

Run-time
Development & production phases

Analysis with Formal Methods

Advanced correctness and logical integrity observations

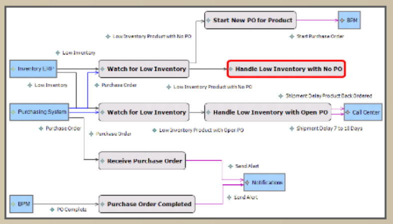
Build-time
Development phase

Static Analysis Observations

Disconnected agents

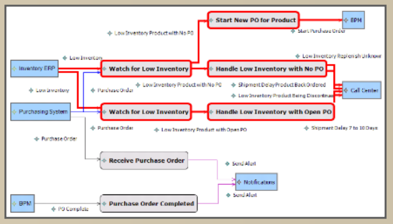
EPA does not produce any derived event or produces a derived event that is not consumed – “dead end”

EPA’s input event(s) are never produced



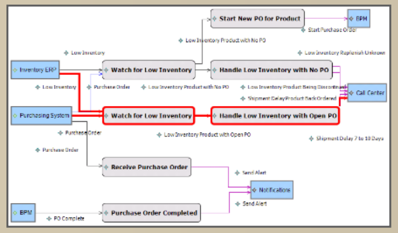
Event consequences

All application assets (events and EPAs) directly or indirectly affected by event



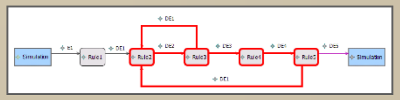
Event provenance

All possible ways to emit an event (set of paths in the application network)



Potential infinite cycles detection

Event that belongs to its own consequences



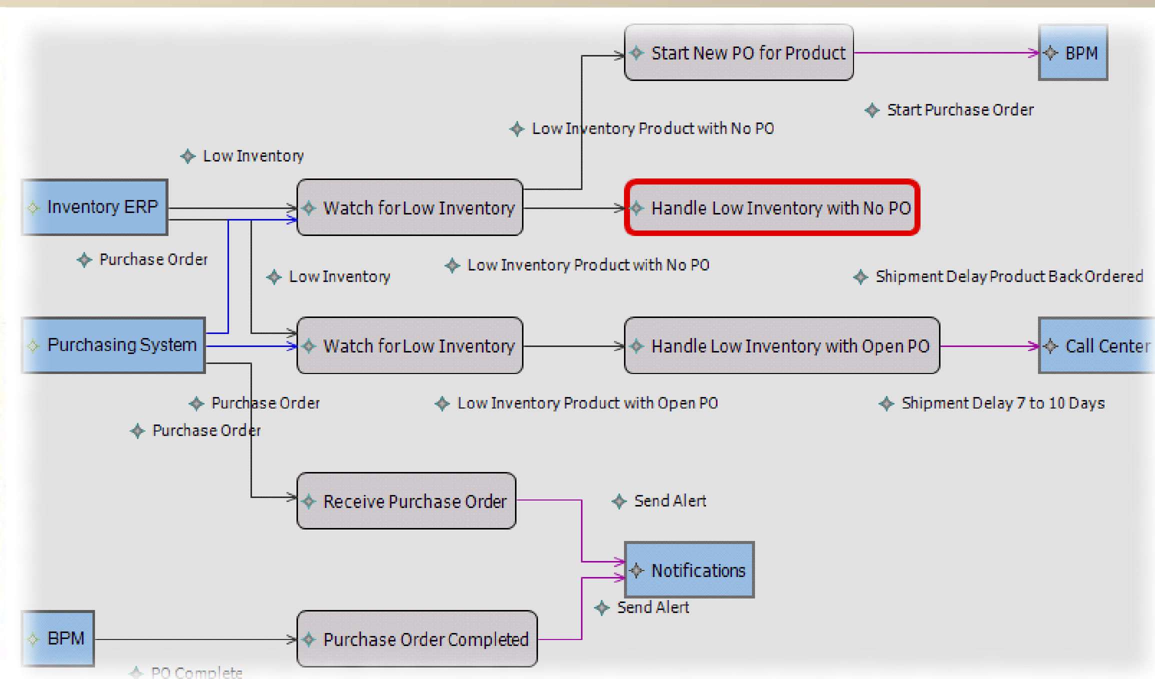
Static Analysis – Disconnected Agents

EPA is disconnected with respect to its input in case the input event(s) are not defined or never produced. EPA is disconnected with respect to its output in case it does not produce a derived event or produces a derived event that is never consumed.

Letf ET be a set of event types, s.t.$|ET| = N$ and let A be a set of EPAs, s.t. $|A| = M$.

*Agent A_i is disconnected with respect to its **output** if for each A_j, s.t. $0<=j<=M-1$ and $j != i$, it holds that $Dist(A_i, A_j) = \infty$.*

*Agent A_i is disconnected with respect to its **input** if for each A_j, s.t. $0<=j<=M-1$ and $j != i$, it holds that $Dist(A_j, A_i) = \infty$.*



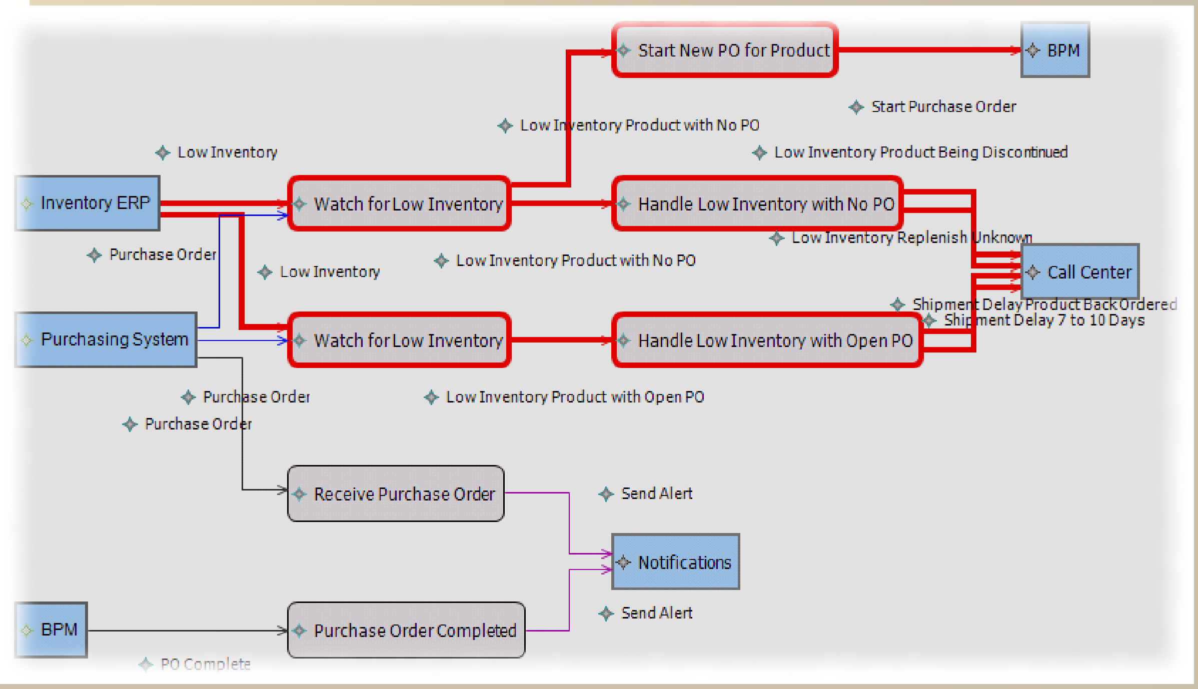
Static Analysis – Event Consequences

Event type consequences are all event types and EPAs found in the transitive closure of the event type that is a subject for a change.

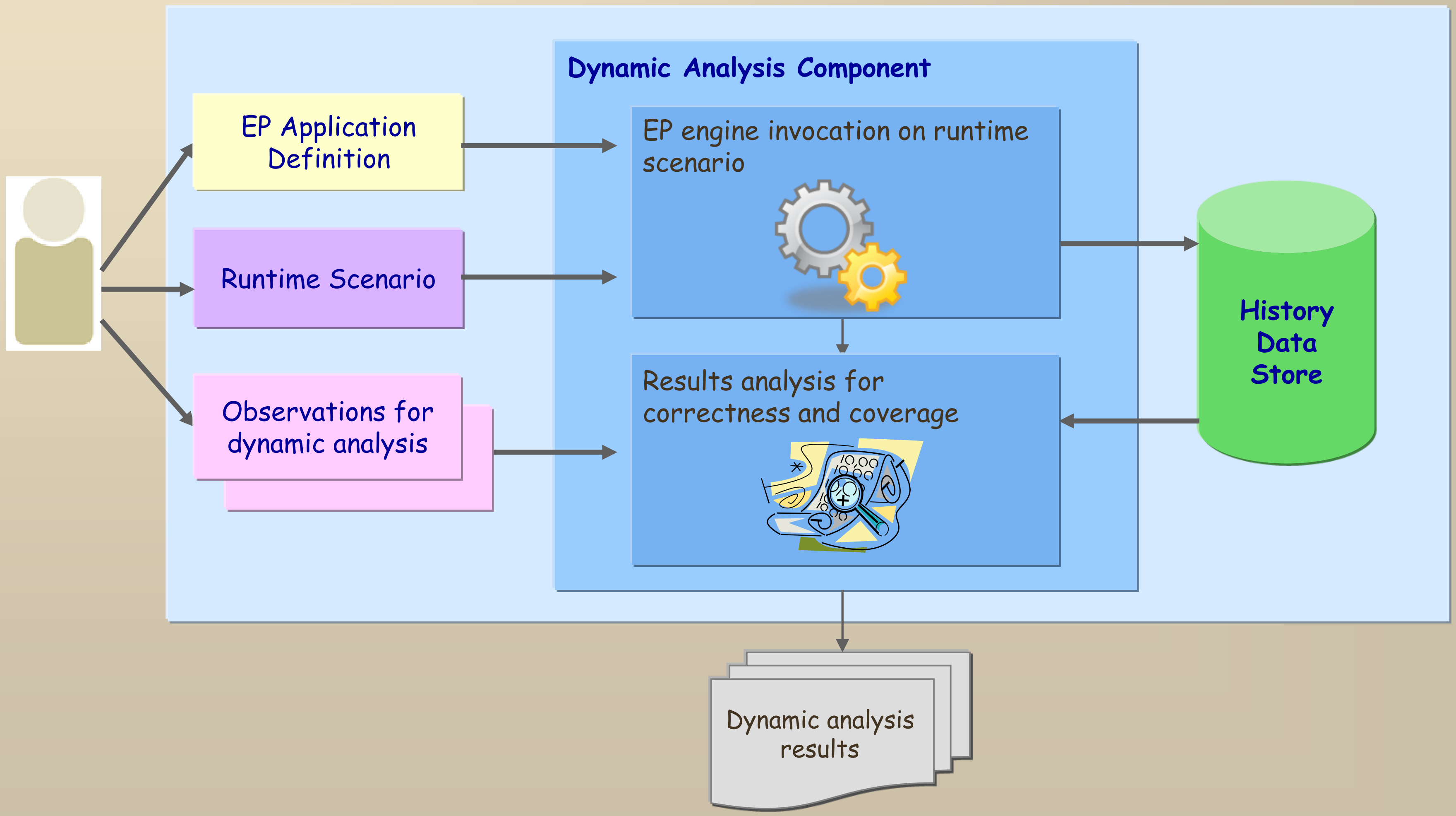
Event type ET_i consequences by events **EventsCons(ET_i)** is {ET_j, $0 \le j \le N-1$, s.t. there exists a path <ET_i, …, ET_j> in the application dependency graph} **U** ET_i

Event type ET_i consequences by agents **AgentsCons(ET_i)** is {A_j, $0 \le j \le M-1$, s.t. there exists a path <ET_i, …, A_j> in the application dependency graph}

Cons(ET_i) = EventsCons(ET_i) **U** AgentsCons(ET_i).



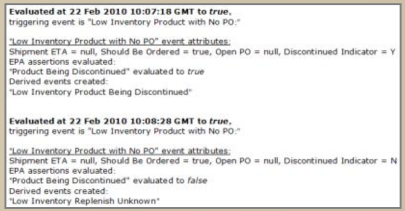
Dynamic Analysis - Approach



Dynamic Analysis Observations

EPA evaluation in context

Tracing an EPA behavior within a certain context partition, e.g. for specific Customer ID



Evaluated at 22 Feb 2010 10:07:18 GMT to *true*,
triggering event is "Low Inventory Product with No PO:"

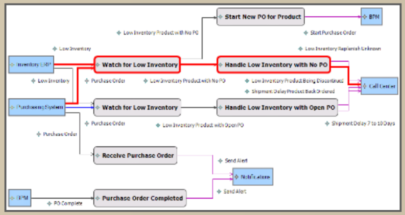
"Low Inventory Product with No PO" event attributes:
Shipment ETA = null, Should Be Ordered = true, Open PO = null, Discontinued Indicator = Y
EPA assertions evaluated:
"Product Being Discontinued" evaluated to *true*
Derived events created:
"Low Inventory Product Being Discontinued"

Evaluated at 22 Feb 2010 10:08:28 GMT to *true*,
triggering event is "Low Inventory Product with No PO:"

"Low Inventory Product with No PO" event attributes:
Shipment ETA = null, Should Be Ordered = true, Open PO = null, Discontinued Indicator = N
EPA assertions evaluated:
"Product Being Discontinued" evaluated to *false*
Derived events created:
"Low Inventory Replenish Unknown"

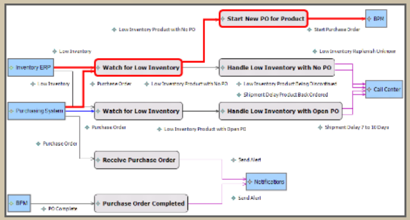
Event instance forward trace

EPAs executed and derived events fired as a result of an event instance arrival



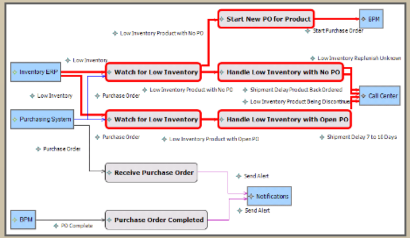
Event instance backward trace

EPAs, raw and derived events that caused the firing of an observed event



Application coverage by scenario execution

Events arrived and EPAs detected as a result of a scenario execution



Dynamic Analysis – Forward Trace

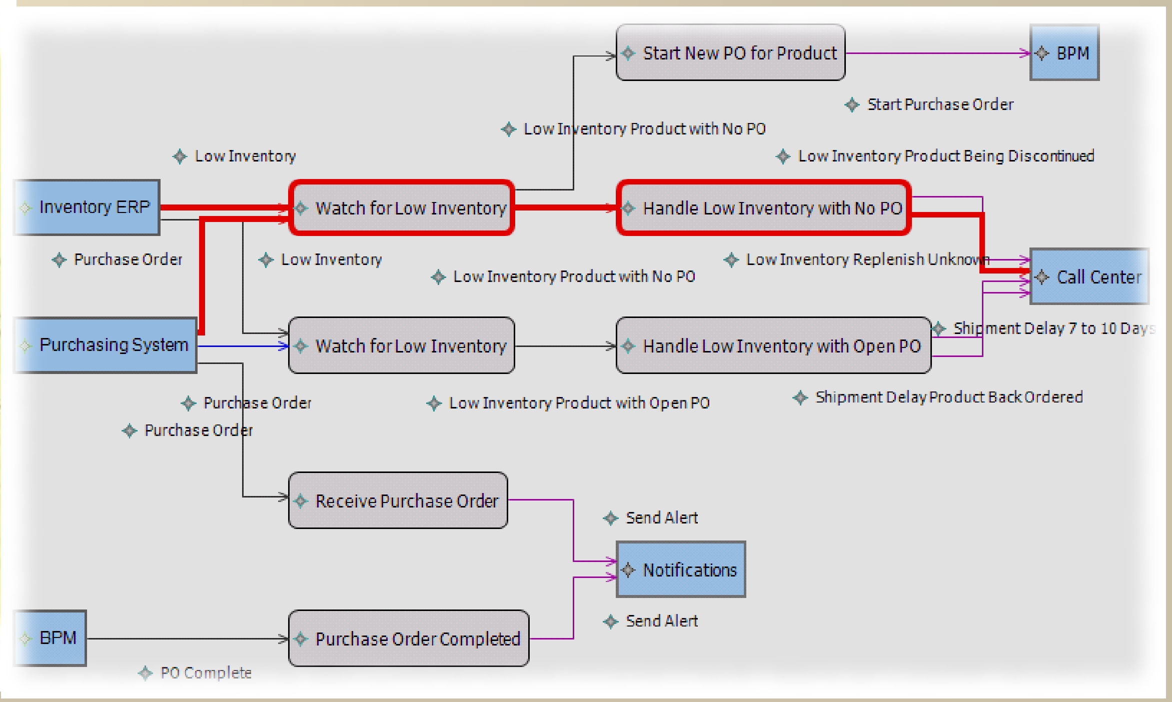
An event instance forward trace is defined as a set of EPAs executed and derived events fired as a result of a certain event instance arrival.

Event instance EI_i forward trace by events ***FEventsTrace (EIi)*** *is $\{EI_j, 0<=j<=K-1$, s.t. EI_j was fired as a result of EI_i arrival$\}$*

Event instance EI_i forward trace by agents ***FAgentsTrace(EIi)*** *is $\{A_j, 0<=j<=M-1$, s.t. A_j was detected as a result of EI_i arrival$\}$*

FTrace(EIi) *= FEventsTrace(EI_i)* **U** *FAgentsTrace(EI_i).*

FTrace(EIi) ⊆ Cons(Type(EIi))

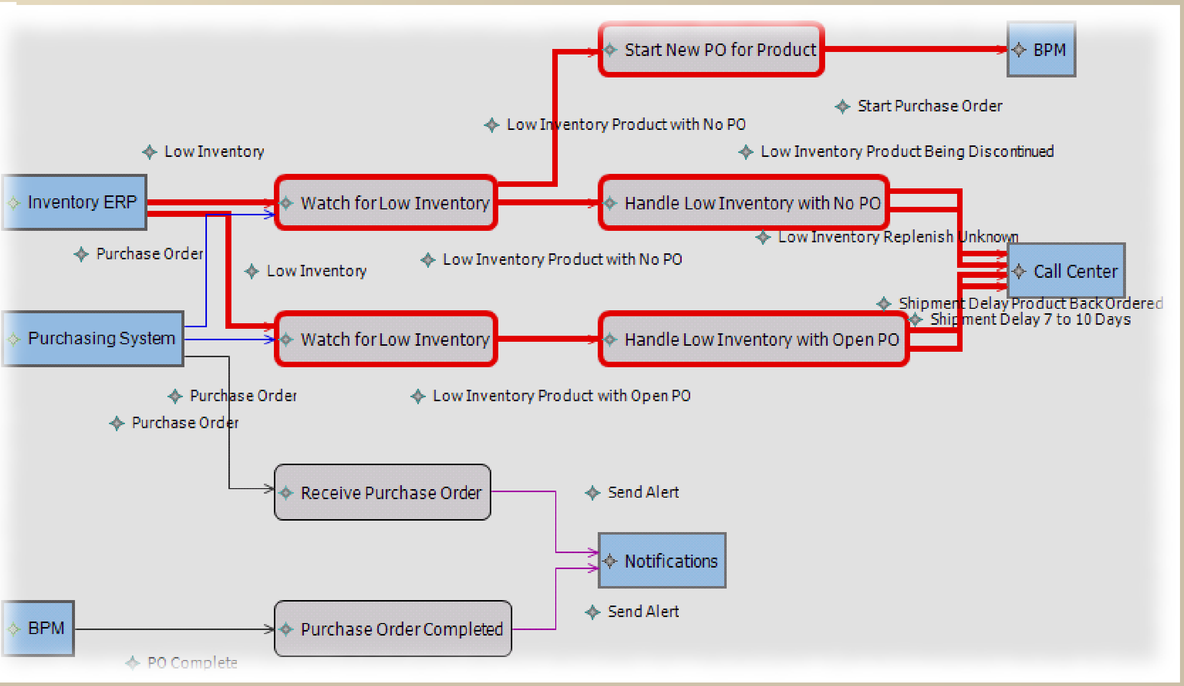


Dynamic Analysis – Coverage

The coverage of event processing application's artifacts by scenario, is a collection of all event instances arrived and EPAs detected, as a result of a scenario execution, i.e., the union of forward traces of all raw event instances.

Let RawEI be a set of raw events instances in a given scenario execution;
$RawEI \subseteq EI.$

$Cov = \bigcup(FTrace(EI_i)),$ *s.t.*
$EI_i \in RawEI.$

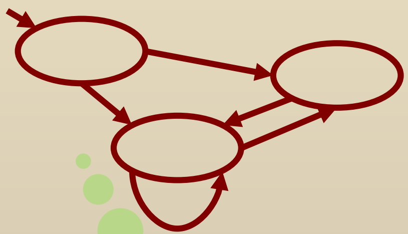


Formal Verification (aka Model Checking)

is the system correct?

A mathematical model of the system **M** (an FSM):

A formal specification of system property ***p***



a labeled state-transition graph

does M satisfy p?

the system is correct!

no

yes

counter example

Analysis Using Formal Methods - Motivation

Static analysis methods enable to derive a set of "shallow" observations on top of the application graph

- A derived event can be physically connected to the graph, but not reachable during the application runtime

Advanced logical integrity observations are beyond the capabilities of current event processing tools

- ✓ Formal verification techniques are optimized for these kind of tasks, using exhaustive exploration of the entire application model

Employing formal methods by event processing is feasible

- ✓ Strong temporal nature
- ✓ Relatively free model (events arrival is not constraint)
- ✓ Relatively small number of assets → formal verification is efficient

Analysis Using Formal Methods Observations

Derived event unreachability

A derived event will never be produced due to logical contradictions in its provenance paths

Logical equivalence of two EPAs

For a given scenario, EPA1 is detected iff EPA2 is detected

Mutual exclusion of two EPAs

For a given scenario, EPA1 is detected iff EPA2 is **not** detected

Automatic generation of a scenario for application coverage

Using the model checking “counter example” feature

Agenda

- Temporal correctness
- Tuning the semantics of event-based applications
- Data consistency and event-based systems
- Validation of event-based systems
- Conclusion

Conclusion

Most efforts in event-based systems are invested towards non-functional requirements such as throughput and latency

Consistency and correctness are often achieved with manual "workarounds", especially around temporal consistency issues

More research and engineering efforts should be invested in tools.

The main challenge is how to use the power of events to make the world become better