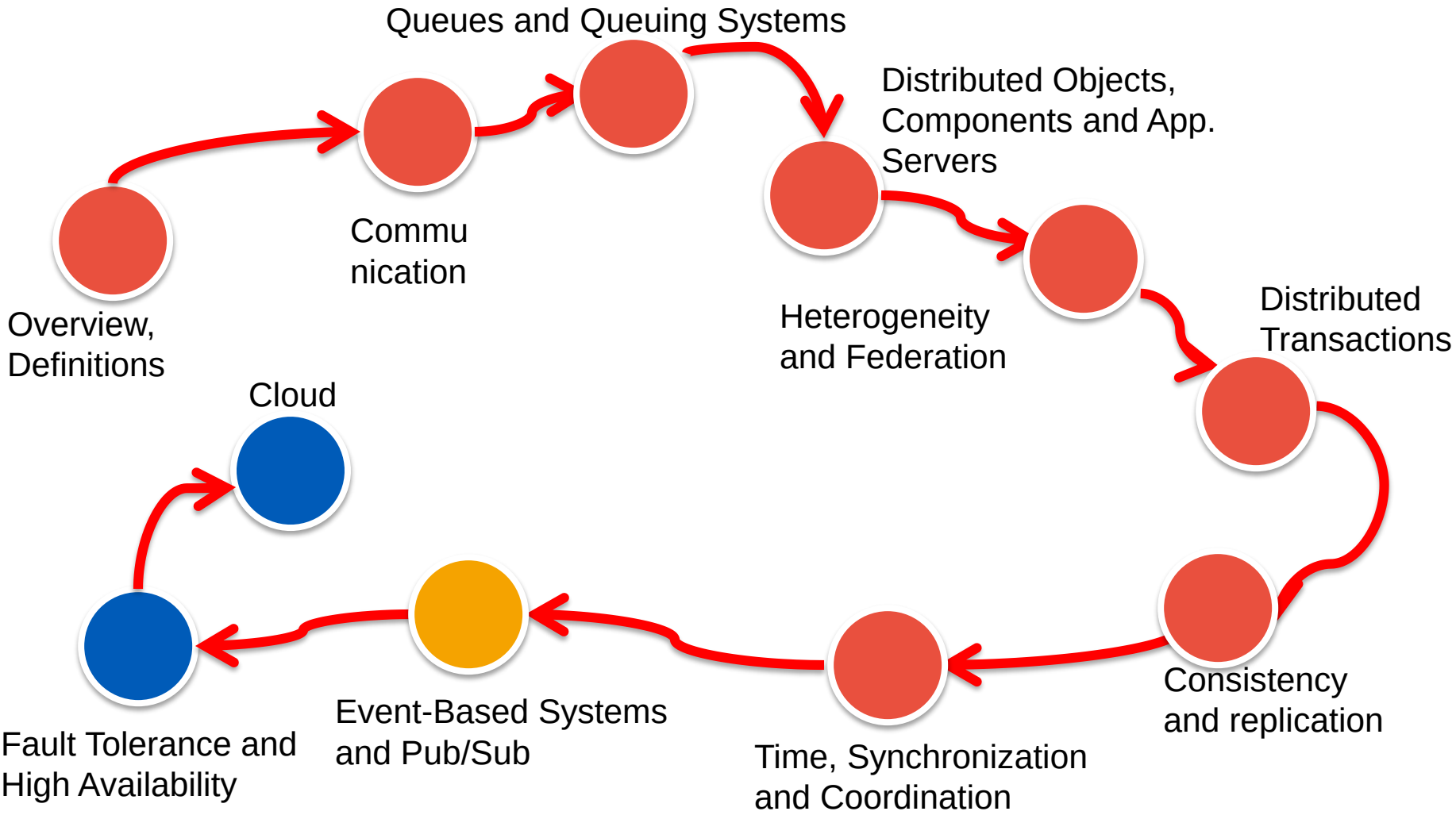
Middleware:
9. Event Processing and Pub/Sub

A. Buchmann
Wintersemester 2014-2015

TECHNISCHE UNIVERSITÄT DARMSTADT

DVS

Topics

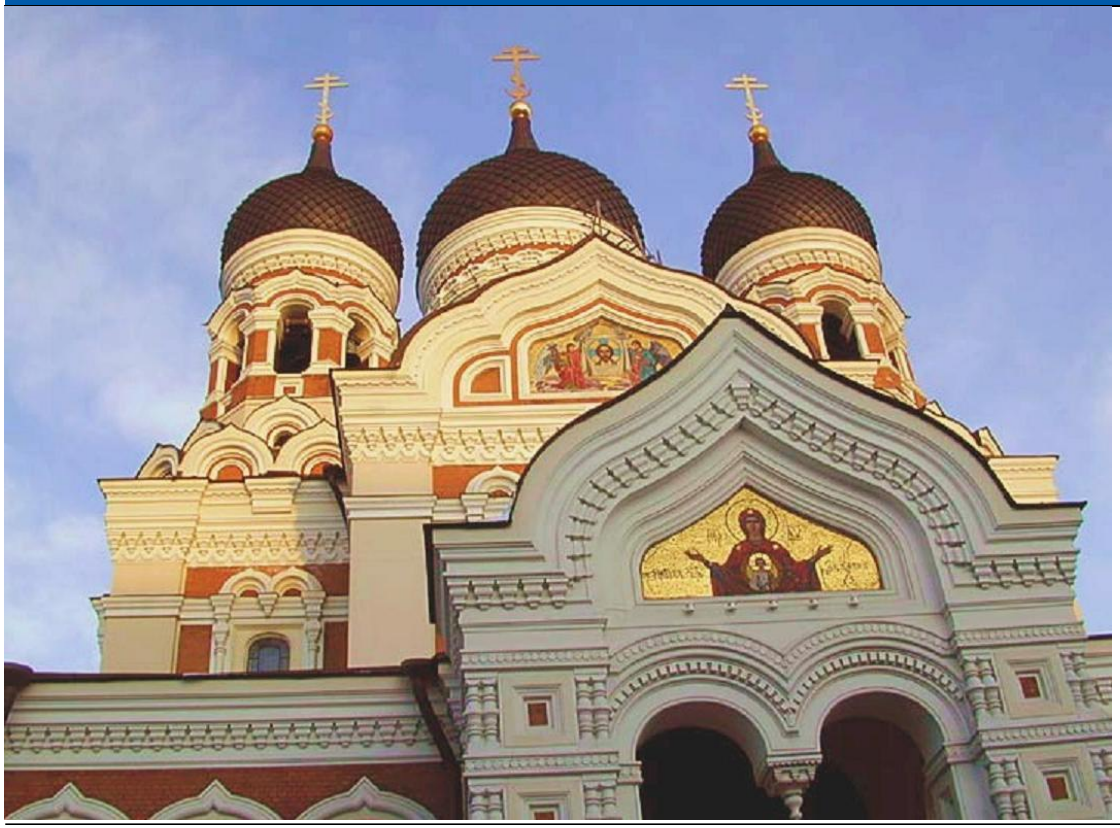

Queues and Queuing Systems
Distributed Objects, Components and App. Servers
Commu nication
Overview, Definitions
Heterogeneity and Federation
Distributed Transactions
Cloud
Consistency and replication
Fault Tolerance and High Availability
Event-Based Systems and Pub/Sub
Time, Synchronization and Coordination

From Calls to Events: Architecting Future BPM Systems

Alejandro Buchmann
buchmann@informatk.tu-darmstadt.de

TECHNISCHE UNIVERSITÄT DARMSTADT

DVS

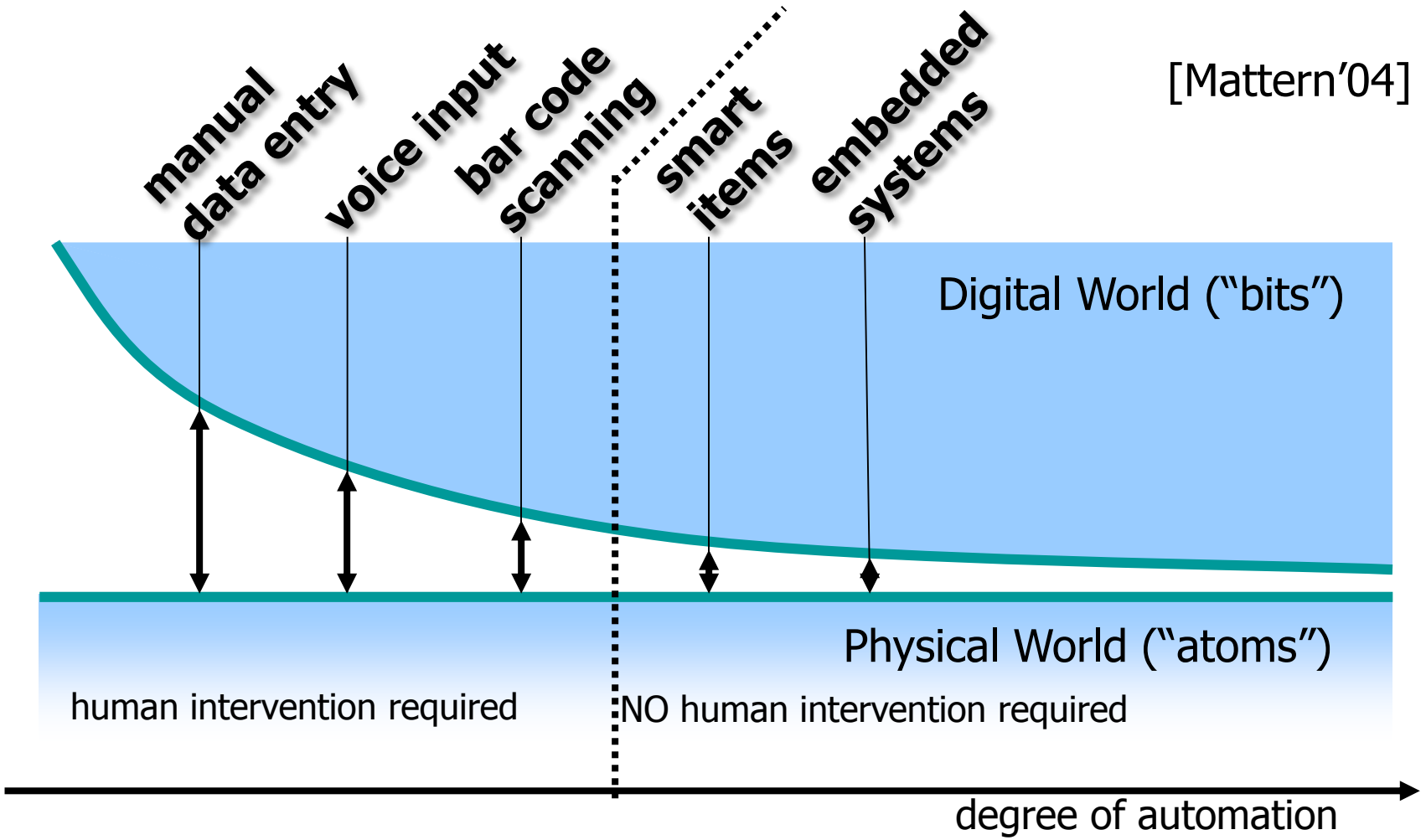


BPM – 2012

Tallinn, Estonia

Merging of Physical and Digital Worlds

TECHNISCHE UNIVERSITÄT DARMSTADT



Smart Environments Produce Streams of Events

- RFID tag detection
- Sensor readings
- Images from surveillance cameras
- Bio-data from wearable sensors
- Vehicle count/identification
- Position/context data
- Radar/Lidar, IR, imaging in autonomous vehicles
- Stock ticker
- Blog postings

Stream Processing requires New Approach

TECHNISCHE UNIVERSITÄT DARMSTADT

Data in database

- Stationary
- *Pull* = Query
- Easy to control

Streaming data

- Flowing
- *Push* = Filter & Aggregate
- Harder to control

The Real-Time Enterprise

TECHNISCHE UNIVERSITÄT DARMSTADT

From the Gartner Web Site:

- The Real-time Enterprise is Event-driven

  Enterprises that want to operate in real time must expand their use of event-oriented design, message-oriented middleware and publish-and-subscribe communication.

http://www.gartner.com/pages/story.php.id.2632.s.8.jsp

Business Events may be unpredictable

How to deal with unpredictability

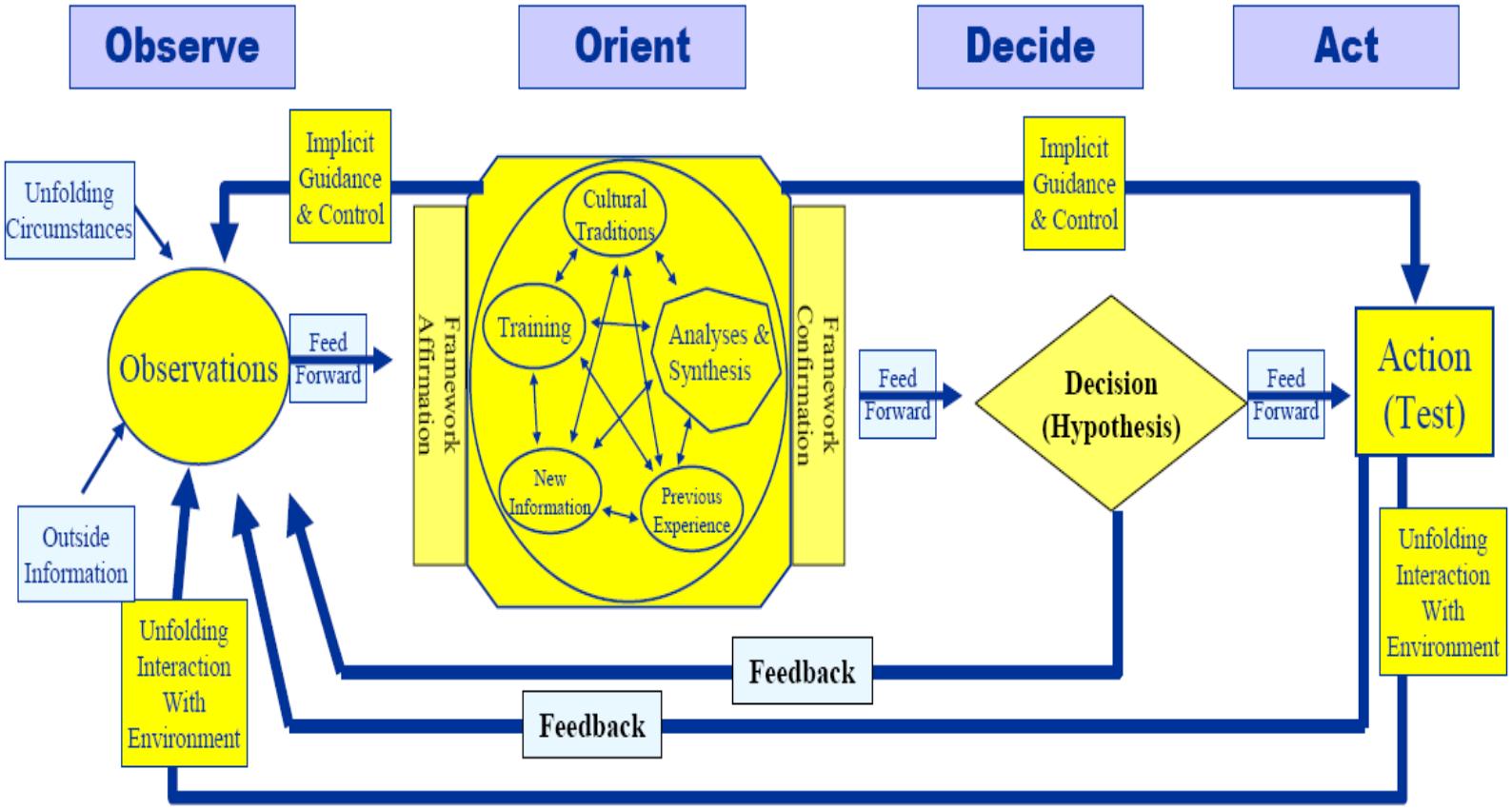
TECHNISCHE UNIVERSITÄT DARMSTADT

- Predictable events are easy to deal with → planning
- Software for planned processes has well-established information flows and invocation mechanisms

- Unpredictable events require monitoring and reacting
- Reactive processes must be activated when a situation arises

The OODA Cycle (John “40 second” Boyd) (MAPE is the IBM buzzword)

TECHNISCHE UNIVERSITÄT DARMSTADT

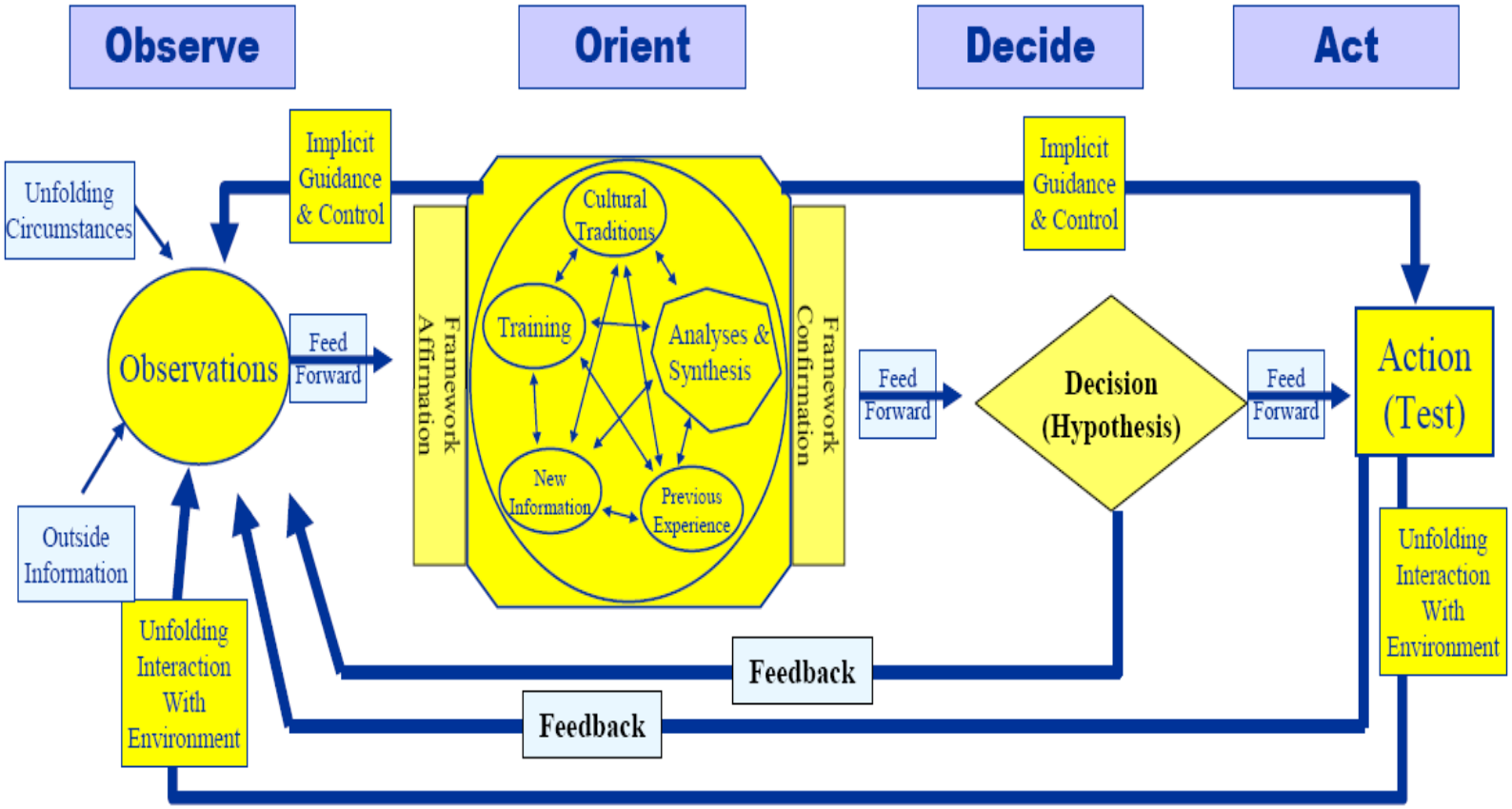


DVS

The OODA Cycle (John “40 second” Boyd)

TECHNISCHE UNIVERSITÄT DARMSTADT

Event Detection Event Contextualization Relevance Test Process Invocation



DVS

The phases of the OODA Cycle

TECHNISCHE UNIVERSITÄT DARMSTADT

- Observation deals mostly with detection of simple events
  - From sensor signals to discrete events (event objects)
- Orientation deals with complex events
  - Event aggregation
  - Event composition
  - Event derivation
  - Contextualization of events → all together CEP
- Decision entails the comparison with (mental) model and the decision to act if deviations exceed thresholds
- Act is the reactive component that itself may impact the surroundings causing new events

Growing Number of Event-Triggered Applications

TECHNISCHE UNIVERSITÄT DARMSTADT

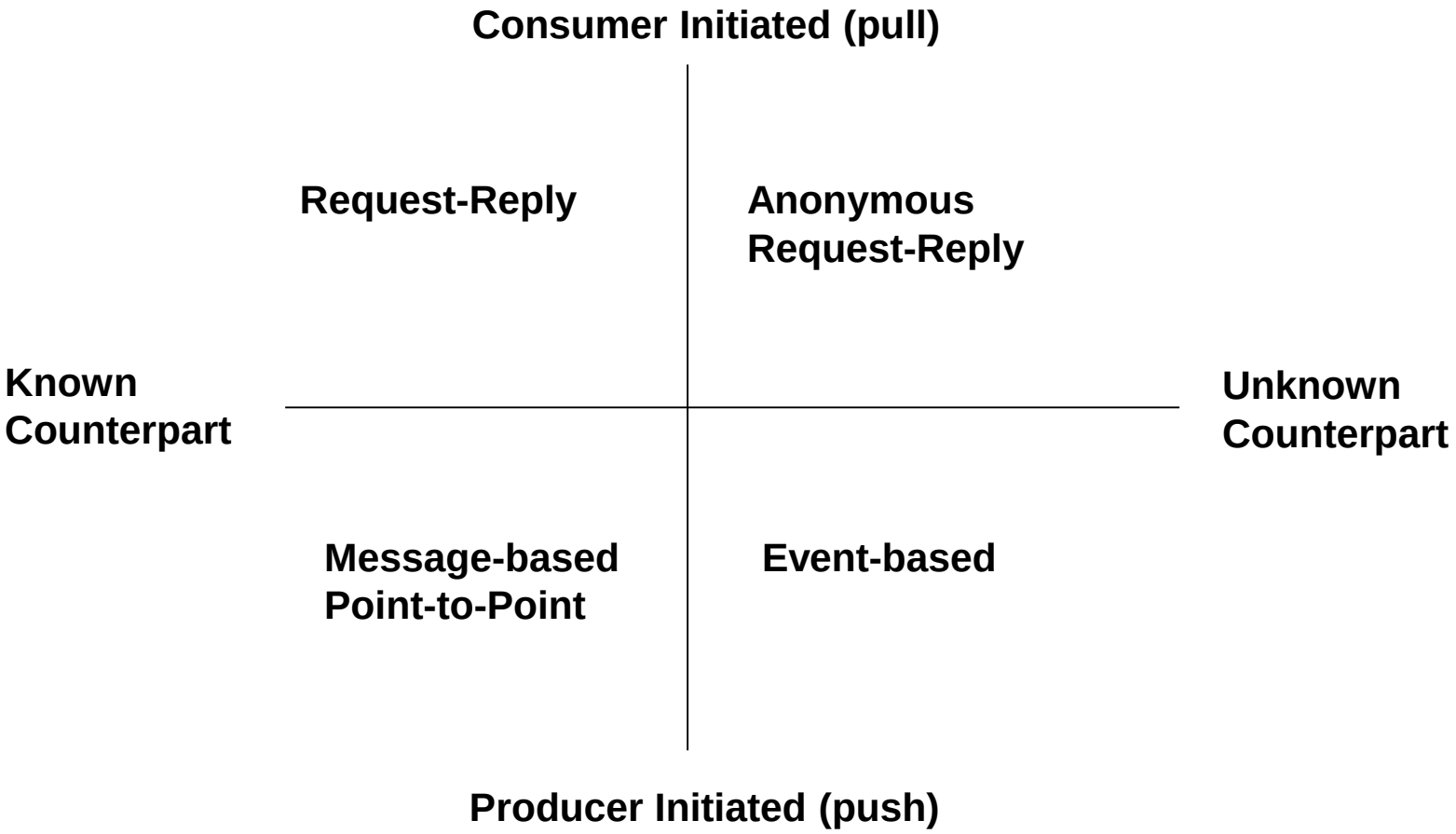
- Automated trading
- Fraud detection (financial, mobile telephony)
- Baggage/package transportation and tracking
- Logistics (WSN in container, spanning containers, fleet management)
- Traffic control and management
- Piping information to the cockpit
- Environmental monitoring
- Integrated health monitoring from home to hospital
- Ambient assisted living, smart homes
- Infrastructures for smart cities
- Massively parallel multiplayer online games (MMOGs)

Growing Number of Event-Triggered Applications

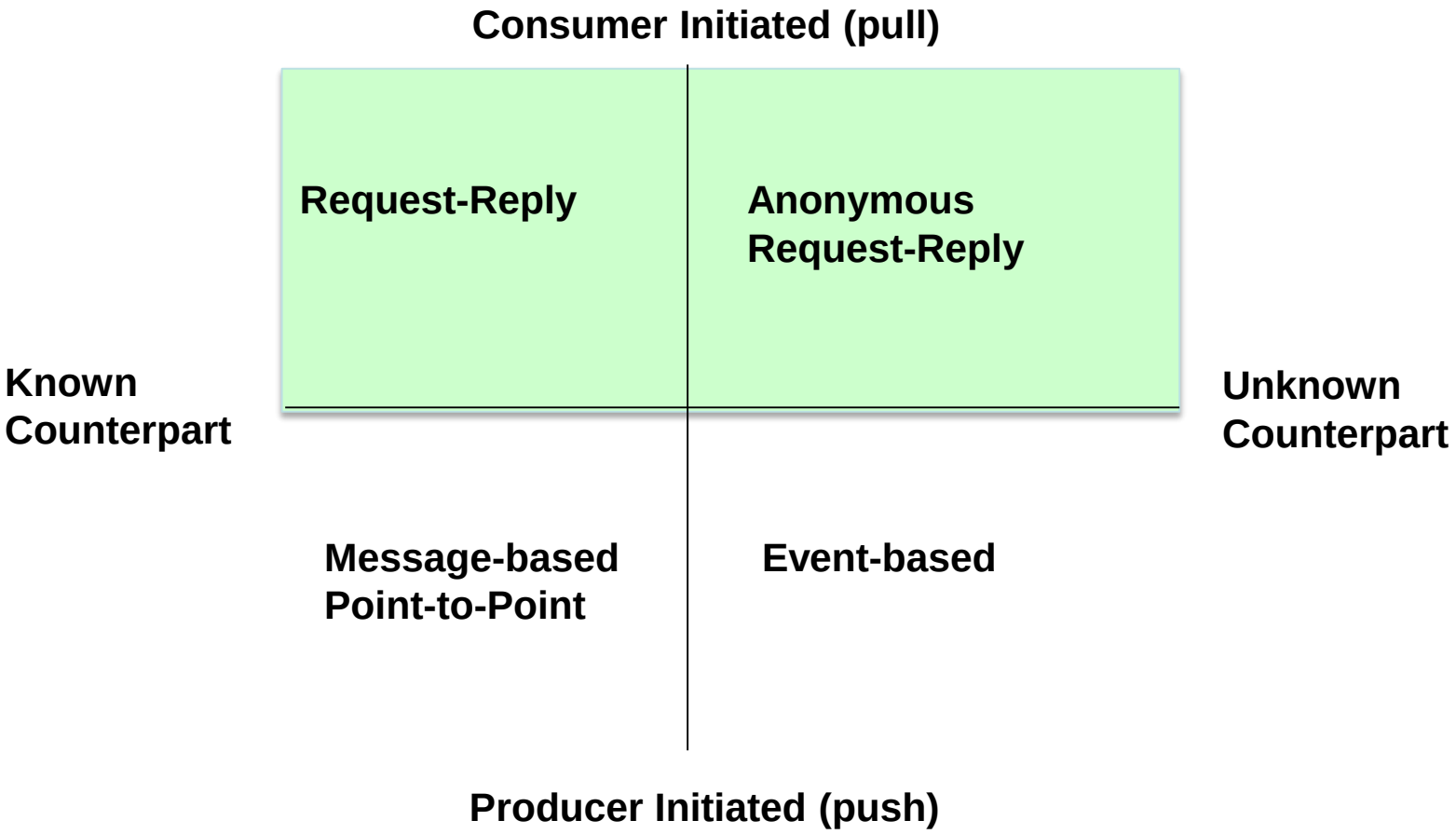
TECHNISCHE UNIVERSITÄT DARMSTADT

DVS

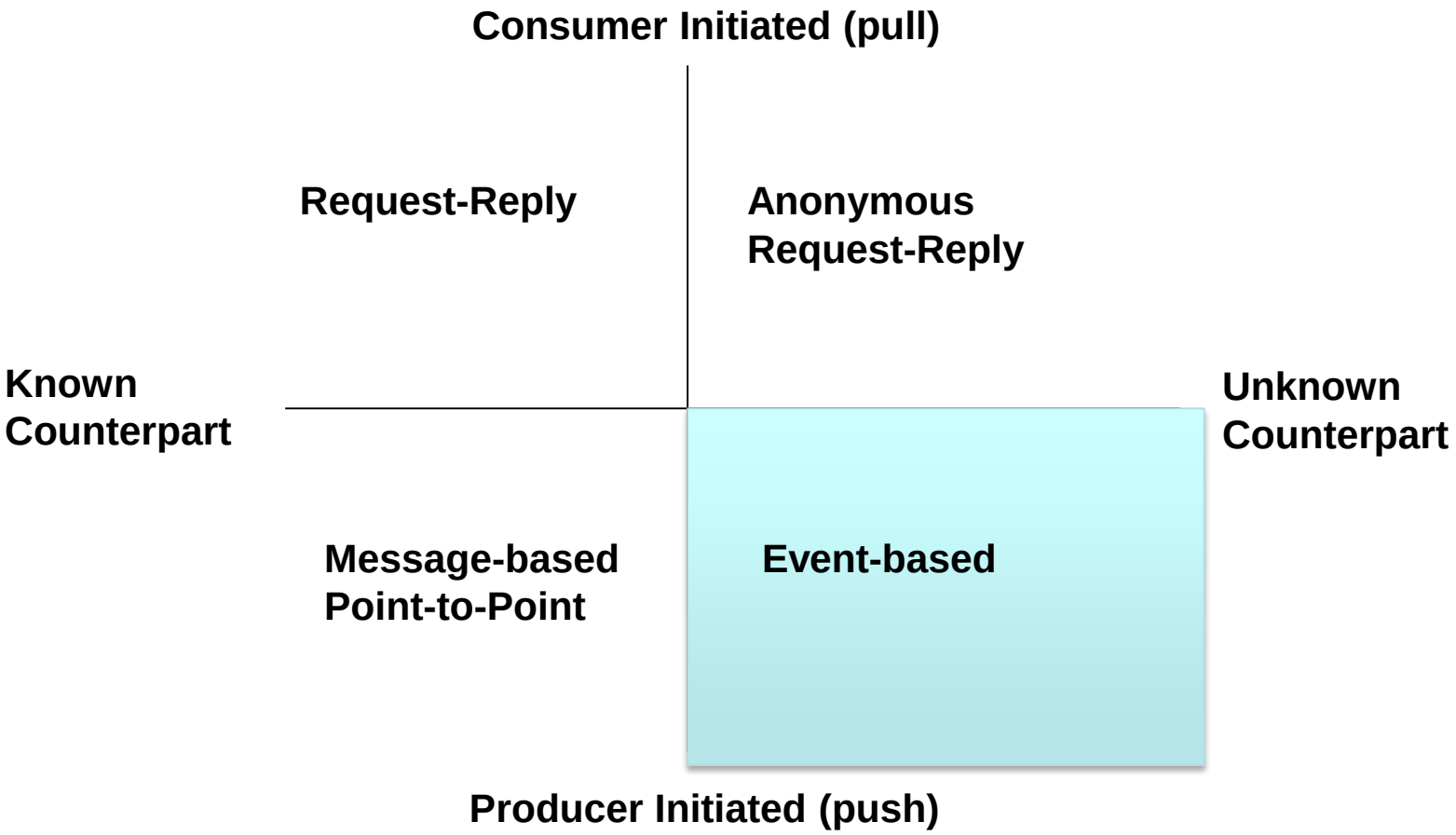
Invocation


Consumer Initiated (pull)
Request-Reply
Anonymous Request-Reply
Known Counterpart
Unknown Counterpart
Message-based Point-to-Point
Event-based
Producer Initiated (push)

Invocation


Consumer Initiated (pull)
Request-Reply
Anonymous Request-Reply
Known Counterpart
Unknown Counterpart
Message-based Point-to-Point
Event-based
Producer Initiated (push)

Invocation


Consumer Initiated (pull)
Request-Reply
Anonymous
Request-Reply
Known
Counterpart
Unknown
Counterpart
Message-based
Point-to-Point
Event-based
Producer Initiated (push)

What is an event?

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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?

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

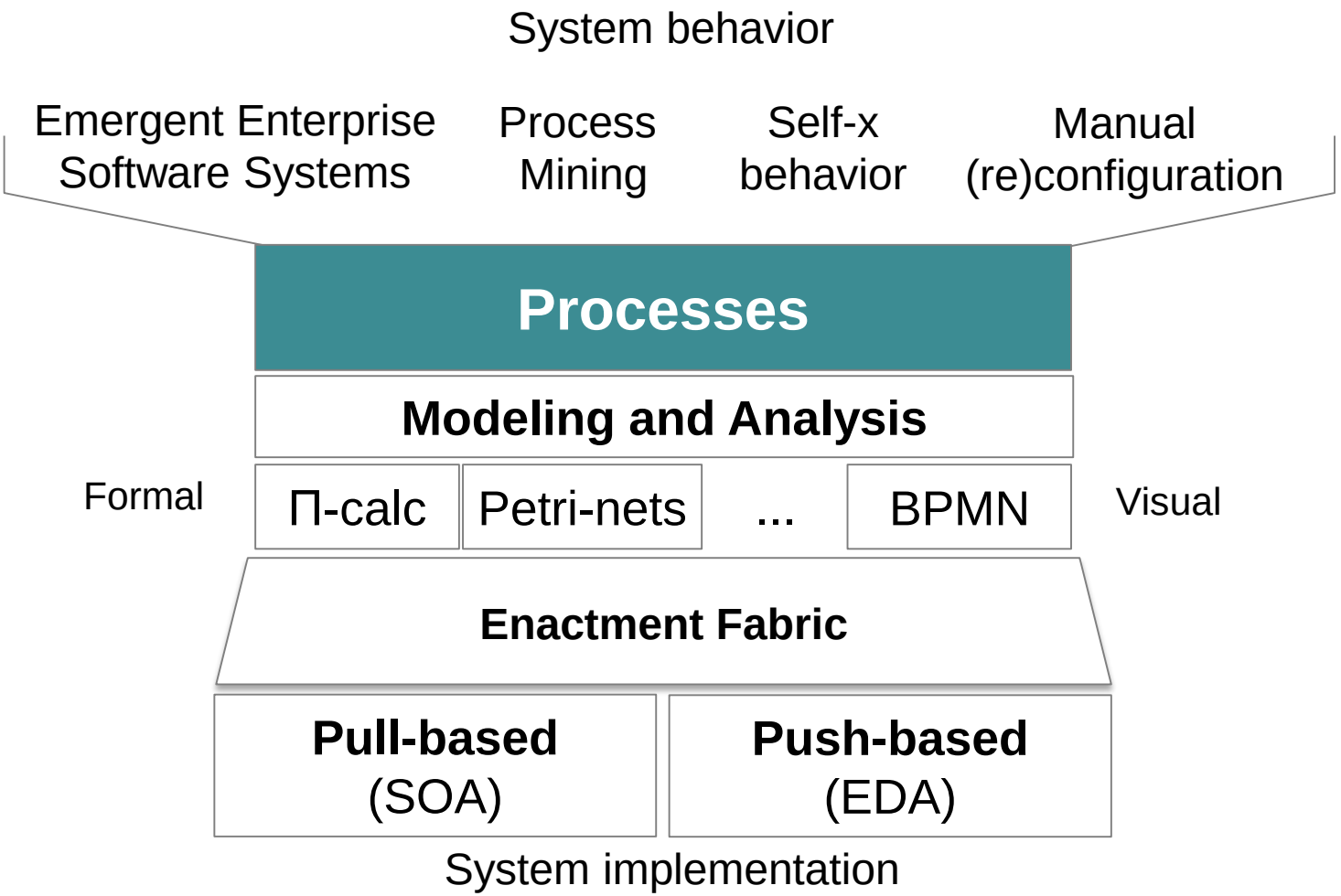
Event-based Processing

TECHNISCHE UNIVERSITÄT DARMSTADT

- The key to event-based processing is the invocation
- Producer of events does not know who will consume them
- Consumer of events can’t give imperative commands to producer
- Middleware isolates producers and consumers in space and time
  - Neither producer nor consumer are blocked
  - New consumers (applications) can be added without touching the event producers
- No application-level polling

Global Architecture for Future BPM

TECHNISCHE UNIVERSITÄT DARMSTADT



Event Driven Architecture (EDA)

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Properties of EDA – Timeliness/Celerity

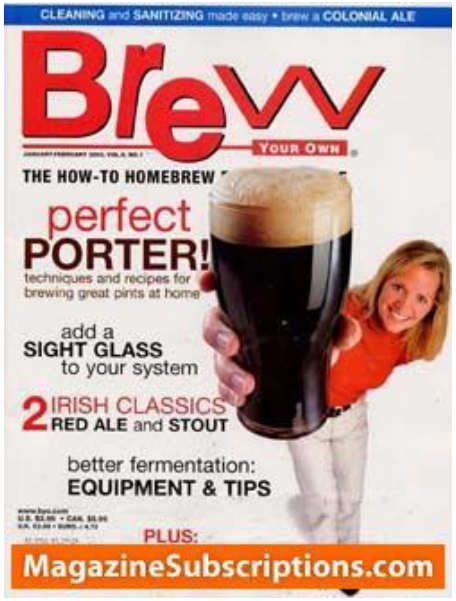
TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- Communication is one-way without need for acknowledgements
  - Event-based interaction pattern does not require an answer, i.e. the event producer will not block



- Reaction to event notifications (no commands)
  - Interested parties must subscribe to events
  - Subscribers may unsubscribe without affecting the producers, just the notification process
  - New consumers (i.e. apps) can be added without affecting the rest of the system (except the notification mechanism)

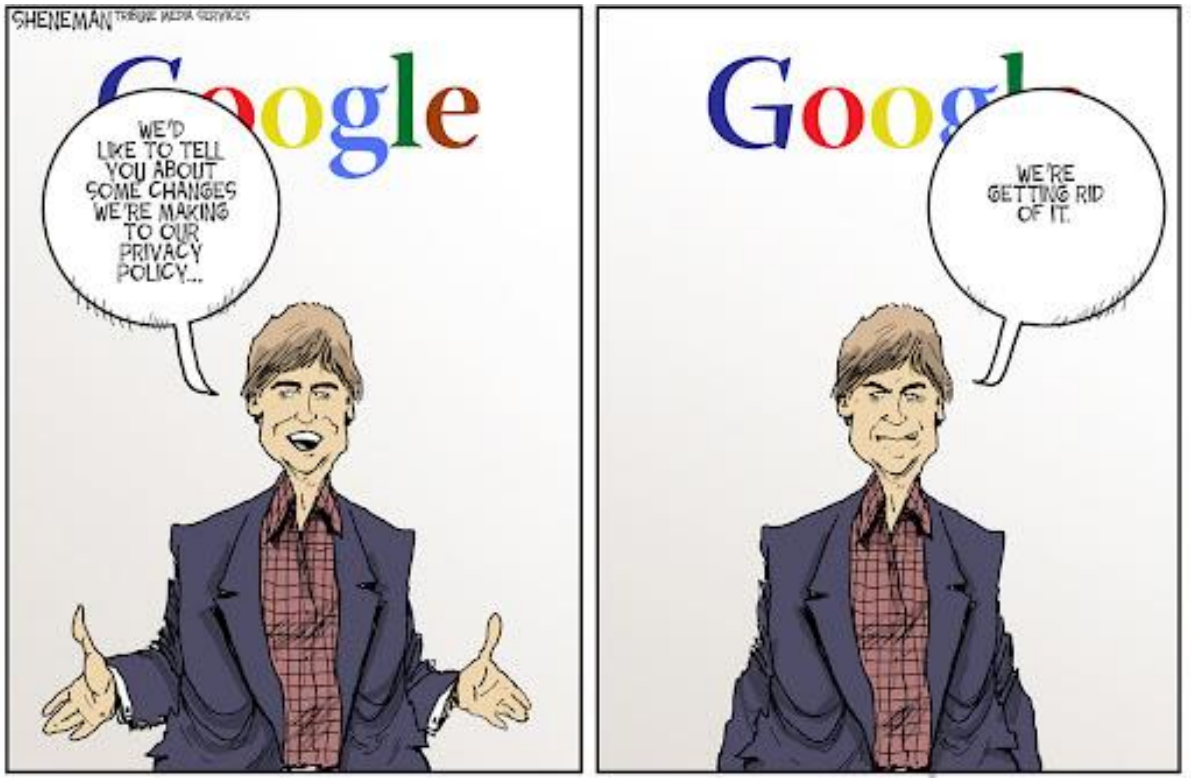
DVS

Control in EDA

TECHNISCHE UNIVERSITÄT DARMSTADT

Because the event producer is not aware of the consumers, it cannot ask them to execute any actions

Event-based systems are consumer controlled

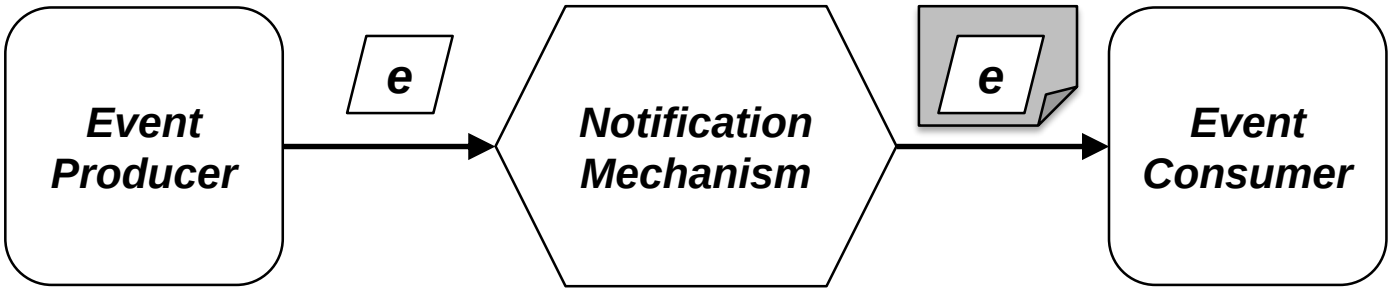


DVS

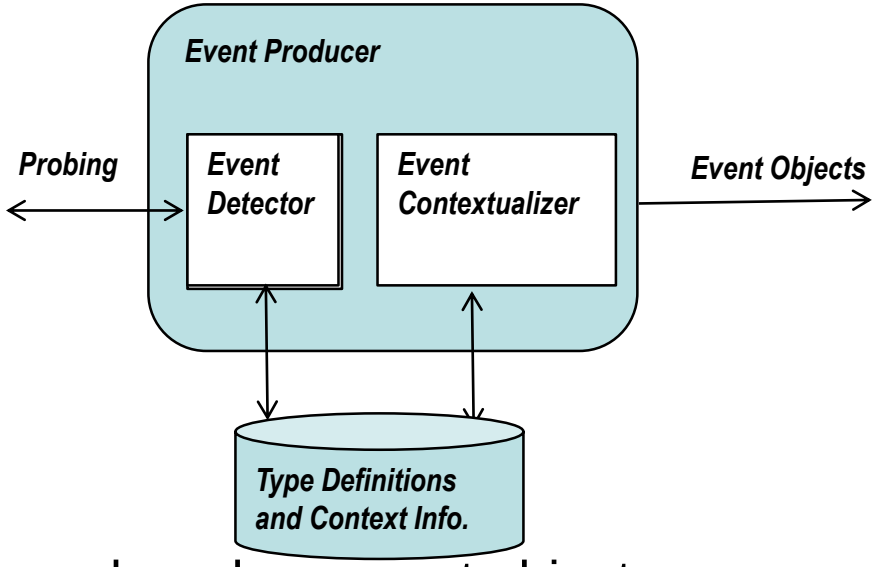
Components of an Event Driven Architecture

TECHNISCHE UNIVERSITÄT DARMSTADT

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

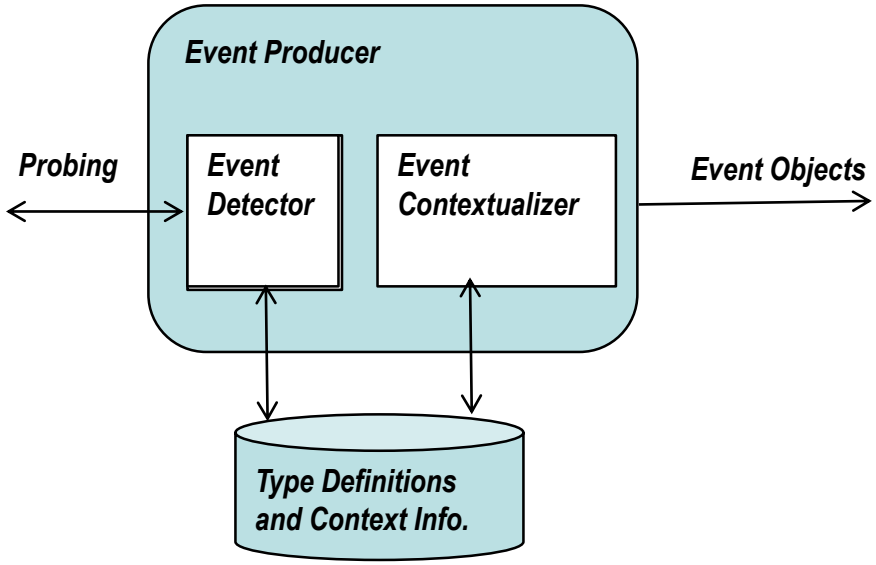


Event Producers



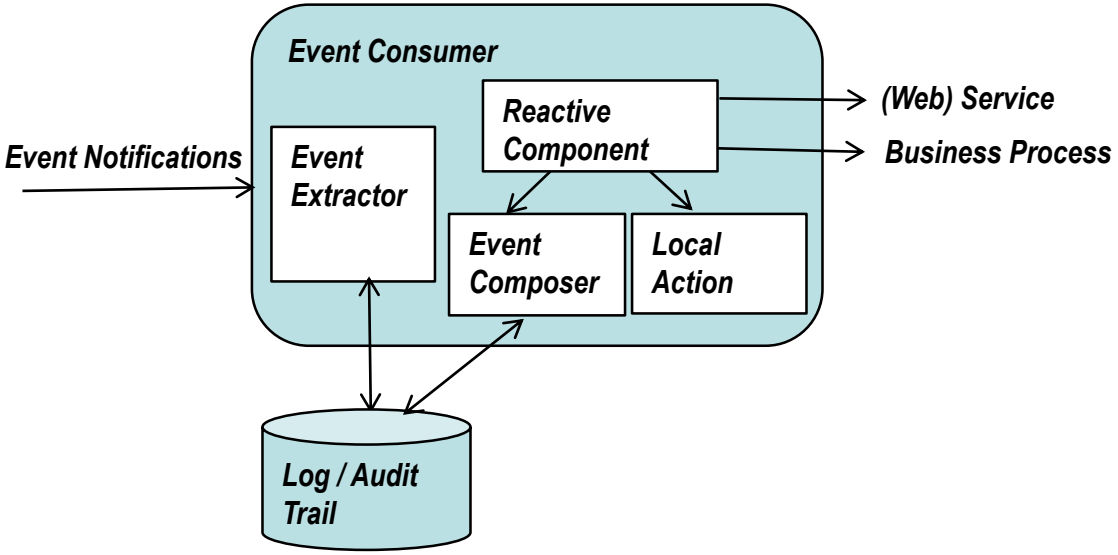
- Detect events and produce event objects.
- The event detection process typically will probe the environment.
- Structure of event objects is defined by event type and contains the necessary event parameters.

Event Producers (cont.)



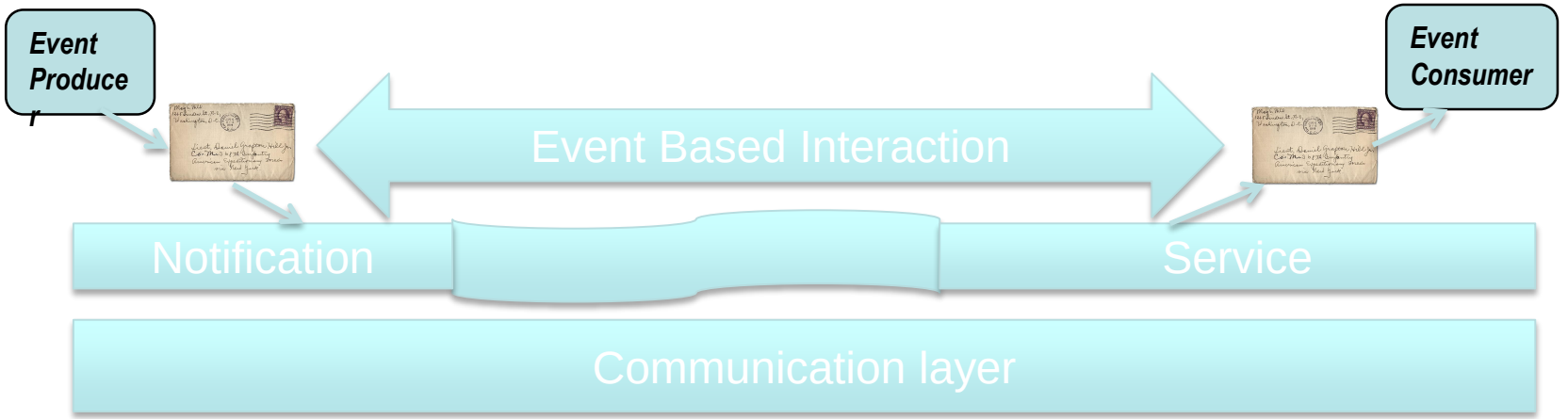
- Event parameters are instantiated by the event detection process and the event contextualization process.
- The event contextualization process may rely on external data sources, type and context information may be held locally

Event Consumers



- Receive event notifications from notification mechanism/service.
- Must unpack event notification, extract event object and execute an action in response to the received event.
- Response may be: local action, invocation of a (remote) service or BP, 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 notification mechanism or transformed?
- If transformations are applied, where and how are they applied?

The Notification Mechanism

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- Notifications are annotated with subjects.
- Subjects are strings 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.

  Sport.Football.NFL.NewEnglandPatriots.SuperBowl

  Sport.Football.NFL.SuperBowl.Winner
- 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 (TIBCO)

Type-Based Pub/Sub

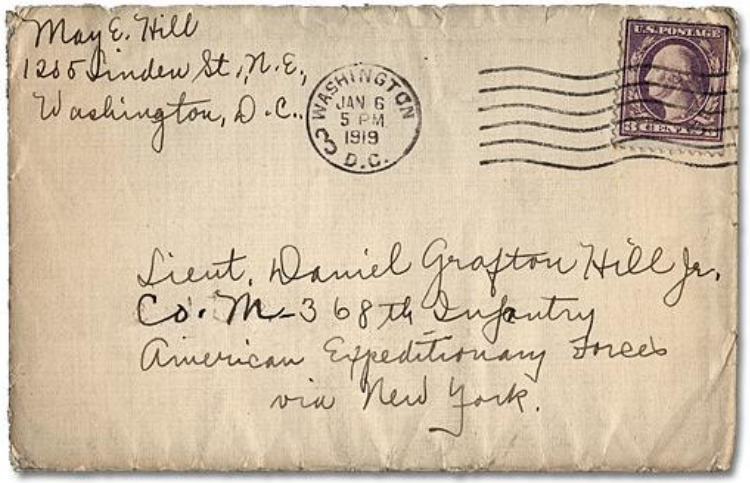
TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

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



DVS

Content-based Pub/Sub

TECHNISCHE UNIVERSITÄT DARMSTADT

- Matching on content of the body of the message
- Expressiveness is determined by the data model used for the content and the formalism for expressing the filter predicates
  - Simple template matching (JEDI)
  - Filter expressions on name/value pairs (REBECA)
  - Xpath expressions on XML documents (Franklin VLDBJ, REBECA-2)

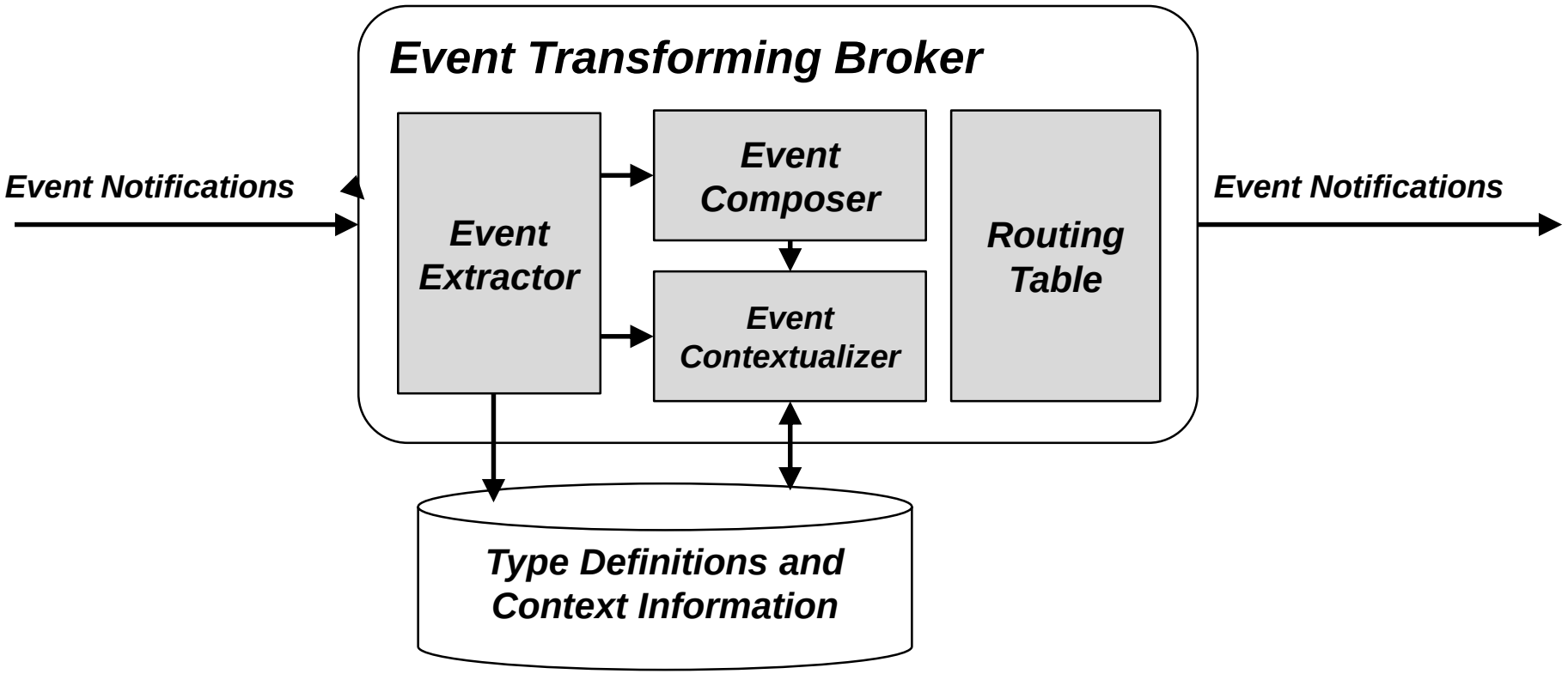
- A tacit assumption is the existence of a common name space used by publishers and subscribers. This is often the case in small systems but rarely in large, heterogeneous environments.

Concept-based Pub/Sub

TECHNISCHE UNIVERSITÄT DARMSTADT

- Makes semantics of advertisements, subscriptions and notifications explicit
- Associates a context with each notification and filter
- Matching compares first, if context of filter and notification are the same
  - If true, use normal content-based pub/sub
  - If the contexts of filter and notification are different, an ontology service is invoked to resolve the different semantics
  - Can be used in conjunction with many of the other pub/sub approaches in large, heterogeneous environments
  - Example ACTRESS, Cilia et al.

Event Transforming Brokers



Event Types – Complex Events

TECHNISCHE UNIVERSITÄT DARMSTADT

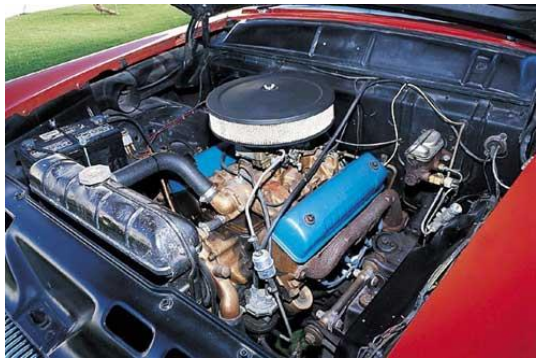
- Event is a happening of interest = a meaningful change of state
- Simple events (temperature readings, RFID tag detection)
- Composite/complex events combine multiple events
  - Aggregation (average temperature)
  - Sequences or time series (trends in temp, 2 RFID readings)
  - Combinations of events of different type (T, P, conc. @ time t)
- Derived events combine with additional (external) information
  - 2 RFID readings from 2 readers at warehouse gate can indicate a higher level event, i.e. pallet came in or moved out
  - Contextualization is essential (situation awareness)

Complex Event Processing Systems

- CEP Engines are simply event composers (i.e. event transforming brokers)
- They take in (streams of) events, operate on them and produce new events
- Output events are usually aggregations, correlations or some derivation that are used to trigger some action (BP)

The Inner Workings of a CEP Engine

TECHNISCHE UNIVERSITÄT DARMSTADT



- Event processing occurs on streams of event objects
  - Event objects could be represented as
    - Tuples
    - Objects
    - XML documents …
  - Representation of the event objects influences event processing
  - 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- Events can be composed by one of two basically different methods
  - Queries expressed in a continuous query language
  - Graphs that are traversed by tokens 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

Continuous Queries

TECHNISCHE UNIVERSITÄT DARMSTADT

- Continuous queries are defined in a derivative of SQL or Xpath
- Windows are the basic construct (sliding/tumbling, instances/time)
- Operators of the algebra are redefined for windows (e.g. instead of relations)
- Particularly good for high-volume streams of regular and homogeneous event objects
- Not so good for event composition with few heterogeneous events

Event Graphs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Complex events are defined via event algebra
- Complex event can be expressed as an event tree (similar to query tree) with events at the leaves and operators of the event algebra at the internal nodes
- Can be evaluated directly rolling up the graph bottom up or by mapping to another graph, e.g. colored Petri net
- Particularly good for heterogeneous event correlation

Issues to consider

TECHNISCHE UNIVERSITÄT DARMSTADT

- Event model and semantics:
  - Are events instantaneous? (point semantics vs. interval semantics)
  - Sequencing of events? (single clock vs. distributed, network delays)
  - Consumption of events? (chronological, recent, …)

Life cycle of events:

- How long are events relevant? (partial composition, garbage collection)

Quality of Service:

- What QoS is required? (timeliness, completeness, uniqueness, …)

Mobility:

- How to deal with mobility? (intermittent connectivity, event staging, …)

Catching an Event

TECHNISCHE UNIVERSITÄT DARMSTADT

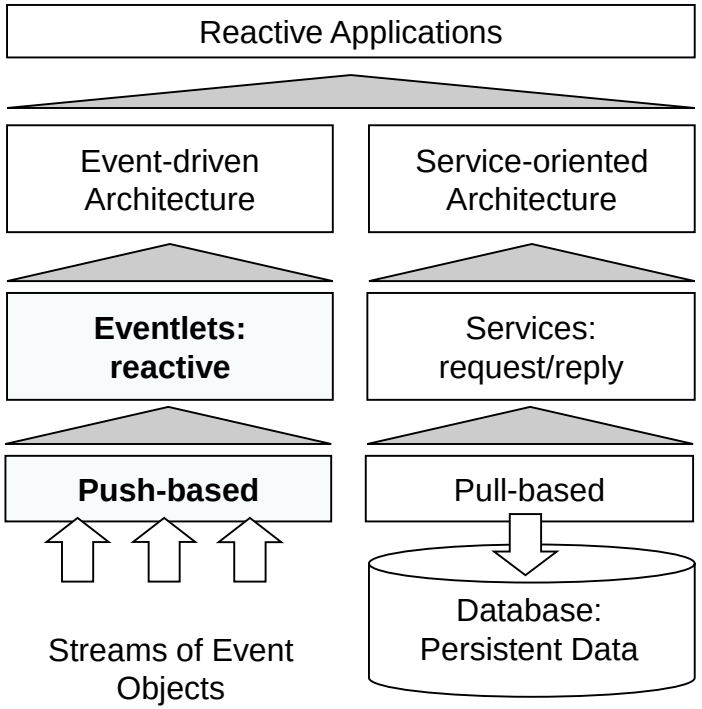
- Business Processes must be enabled to react to events

- Several options:
- BPs run inside a CEP engine
- BPs implement the event catching function as application code
- BPs rely on middleware → Eventlets

Eventlets – an Event Execution Framework

TECHNISCHE UNIVERSITÄT DARMSTADT

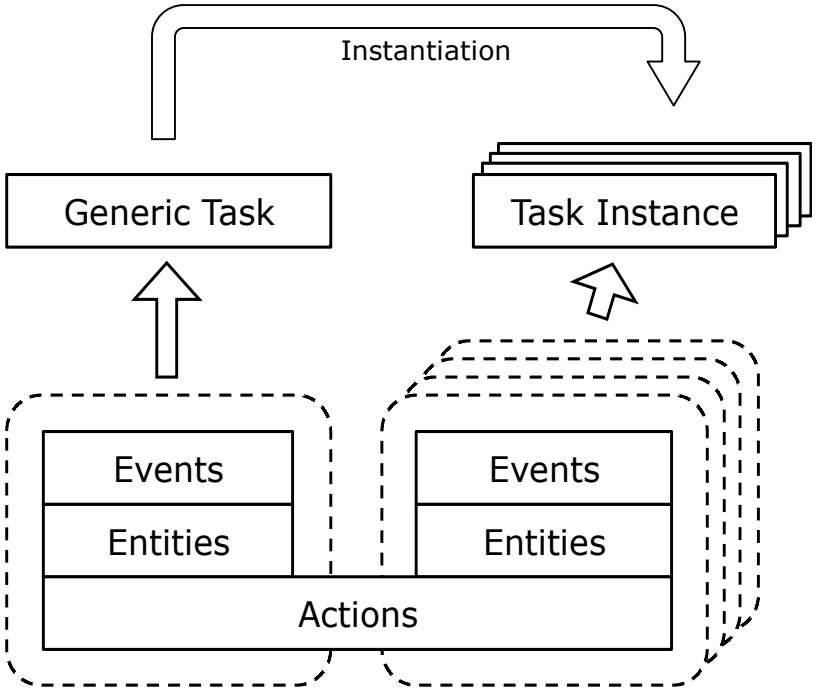
- Eventlets are the equivalent to services in an EDA
- Eventlets can encapsulate existing services
- Eventlets are a component model for the processing of event streams



Abstract View of Eventlets

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eventlets encapsulate application logic
- Containers for generic, reactive application logic (tasks)
- Generic task that is applied to individual entities

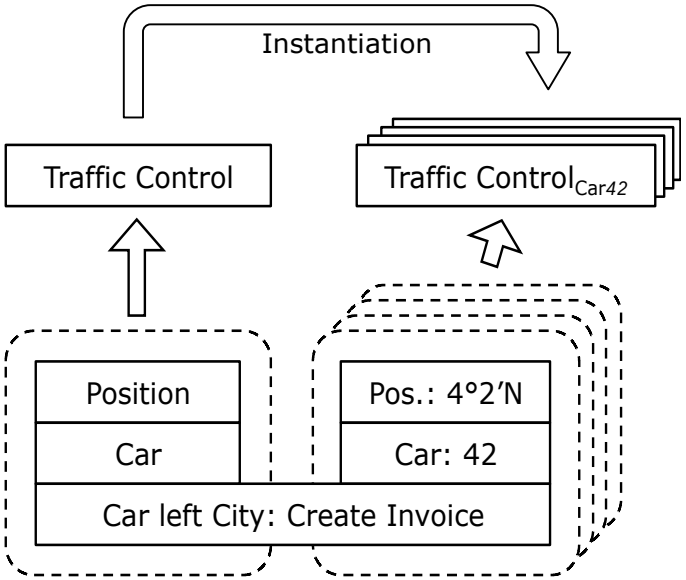


(a) Abstract View

Instantiation of Eventlets

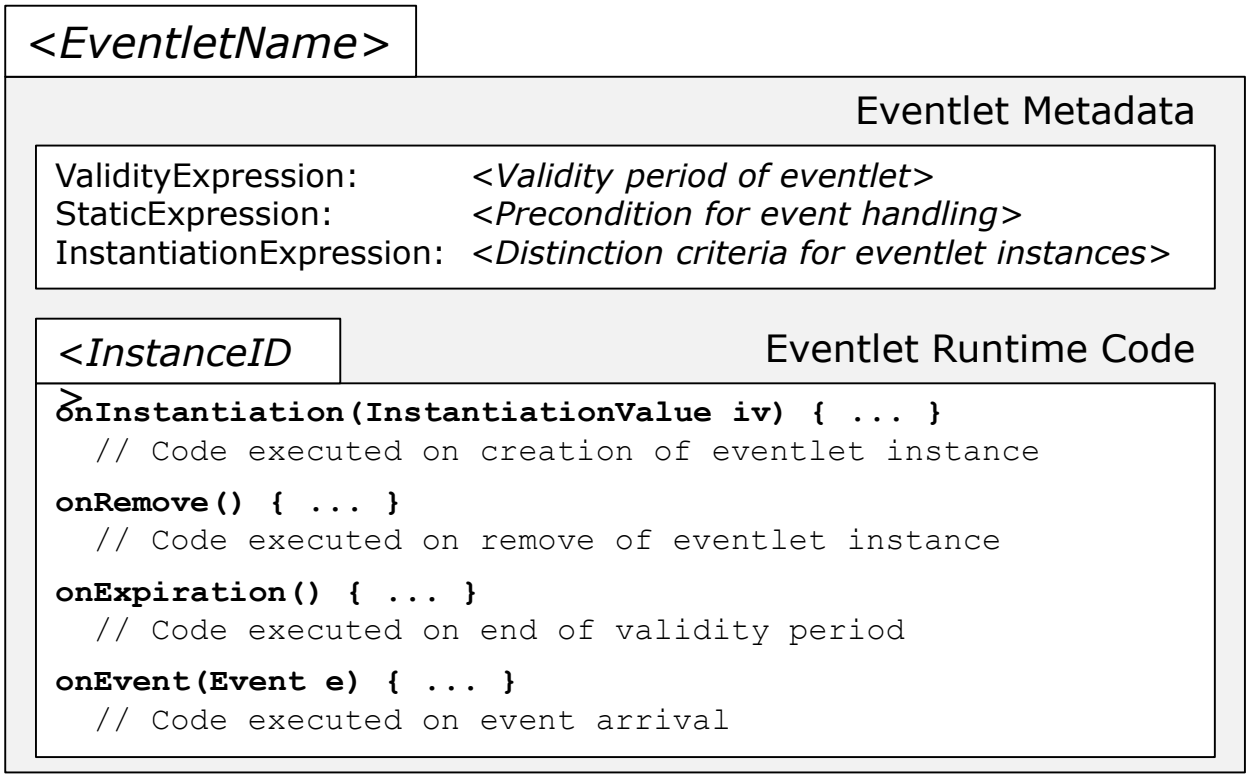
TECHNISCHE UNIVERSITÄT DARMSTADT

- For each entity, a task instance is maintained
- Actions are generic and applied per instance automatically
- Eventlets are invoked implicitly and distributed automatically upon arrival of appropriate event
- Middleware platform provides life cycle management



(b) Example

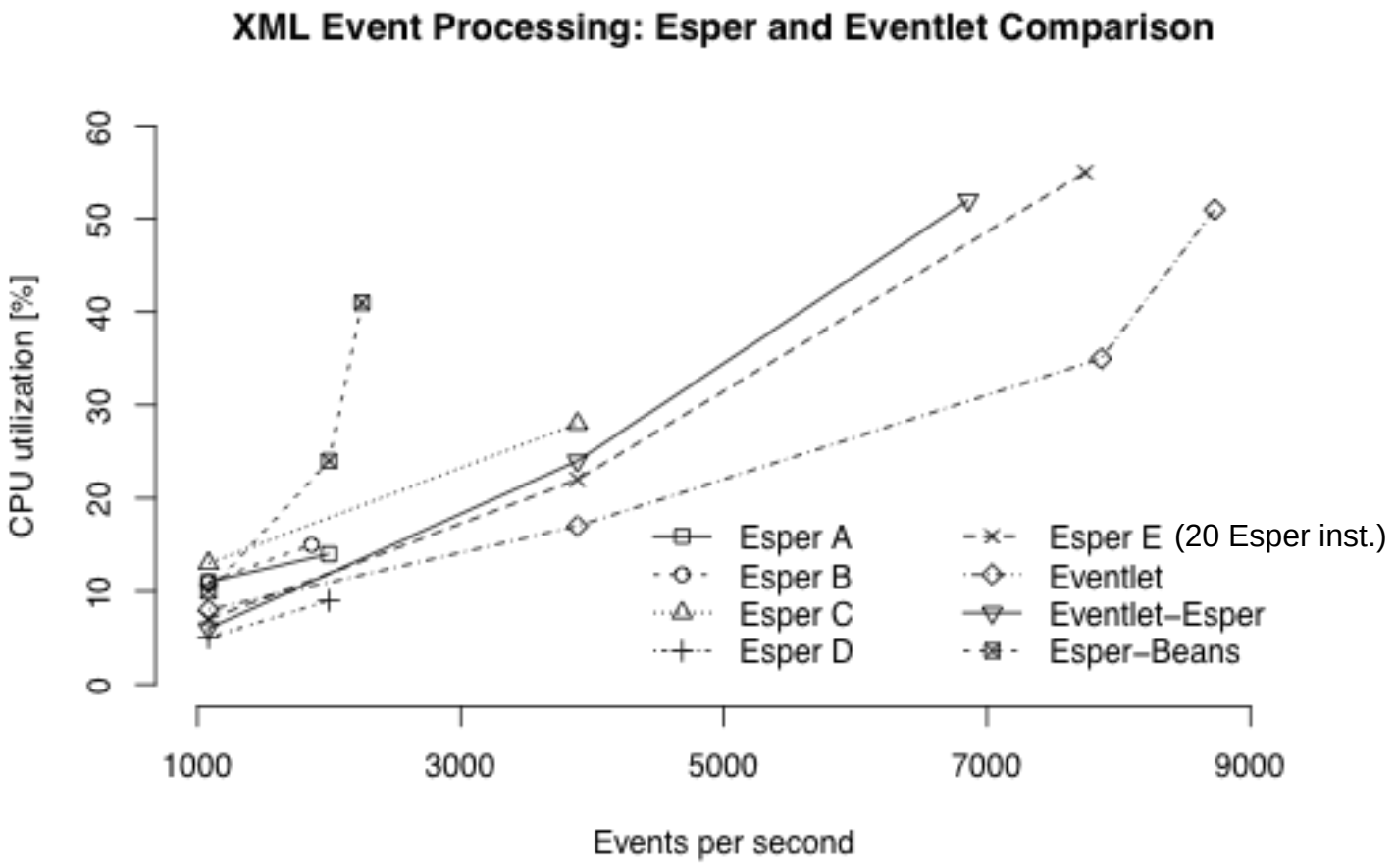
Eventlet Structure



Benefits of Eventlets in comparison: Performance

TECHNISCHE UNIVERSITÄT DARMSTADT

- CEP alone vs. CEP + message beans vs. CEP + eventlets vs. eventlets alone



Benefits of Eventlets in comparison: Programming Effort

TECHNISCHE UNIVERSITÄT DARMSTADT

Component	Esper	Esper-Beans	Esper-Eventlets
EL Prototype App. Logic			20
EL Prototype Metadata			2
EL Registration			1
CEP Queries/Results			18
Main Class App. Logic	39		
Event Listener	10		
Distr. Instantiotion (JMS)	61		
CEP Queries/Results	18		
Bean (generic/specific)		17/5	
Bean Config. (gen/Spec)		2/1	
CEP Queries/Results		18	
Total	128	157 (37+20*6)	41

DVS

Conclusions

- Events are important. Mishandling them leads to missed opportunities
- SOA and EDA are complementary architectures that fit like hand in glove to conform the enactment fabric for Business Processes

- We must promote the understanding of Event Processing and raise the maturity level of EDA to that of SOA

Acknowledgements

- Cristoph Liebig, Bianca Bösling, Christian Grothe: cockpit information systems
- Samuel Kounev, Kai Sachs: SPECjAppServer2004, SPECjms2007 benchmarks, performance
- Mariano Cilia, Pablo Guerrero, Kai Sachs, Christof Bornhoevd: supply chain management, rule processing @ periphery
- Patric Kabus, Christof Leng, Bettina Kemme: MMOG on P2P substrate
- Pablo Guerrero, Daniel Jacobi, Arthur Herzog, Jan Stefan: mixed mode sensor systems, search and rescue applications, sensor networks in logistics
- Gero Mühl, Ludger Fiege, Mariano Cilia: Pub/Sub, scopes
- Andreas Zeitler, Ludger Fiege, Gero Mühl, Felix Gärtner: mobility, ubiquitous computing
- Roger Kilian-Kehr, Ludger Fiege: security, privacy in pub/sub
- Stefan Appel, Tobias Freudenreich, Sebastian Frischbier: performance, eventlets, contexts, emerging software
- Mani Chandy: discussions on SW Engineering, enrichment, web-apps, etc.

Topics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Time is an integral part of the world, models, systems and events
- **Time**
  - Issues in distributed systems
  - Global time vs. local time, physical time vs. logical time
  - Influence on messaging mechanisms or guarantees (ordering, multicast, at-most-once)
- **Synchronization** in middleware and distributed systems
- **Coordination**

Reading for THIS Lecture

TECHNISCHE UNIVERSITÄT DARMSTADT

- The slides for the lecture are based on material from:
  - Andrew S. Tanenbaum and Maarten Van Steen. 2001. **Distributed Systems: Principles and Paradigms.** Prentice Hall.
    - Chapter 5
  - George Coulouris, Jean Dollimore, and Tim Kindberg. 2005. **Distributed Systems: Concepts and Design.** Addison-Wesley Longman.
    - Chapter 11

Synchronization

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Why do processes synchronize in middleware?**
  - To coordinate access of shared resources
  - To coordinate/synchronize execution
  - To order events

- Example
  - How do processes appoint a coordinator?
  - How can mutual exclusion be implemented?
  - How can events be ordered?

Time and Clocks

TECHNISCHE UNIVERSITÄT DARMSTADT

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

DVS

Time, Clocks and Clock Synchronization

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Time**
  - Why is time important in middleware?
  - consider distributed Version Control or the UNIX Make tool
- **Clocks (Timer)**
  - Physical clocks
  - Logical clocks (introduced by Leslie Lamport)
  - Vector clocks (introduced by Collin Fidge)
- **Clock Synchronization**
  - How do we synchronize clocks with real-world time?
  - How do we synchronize clocks with each other?

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Physical Clocks

TECHNISCHE UNIVERSITÄT DARMSTADT

Problem: Clock Skew – clocks gradually get out of synch and report different values

Solution: Universal Coordinated Time (UTC, formerly called GMT):

- Based on the number of transitions per second – atomic clock.
  - International Atomic Time (TAI) – std. sec = 9 192 631 770 oscilations Cs^{133}
- Introduces a leap second from time to time to compensate that days are getting longer.

- UTC is **broadcast**
  - Land-based short wave radio → accuracy of +/- 1 msec (MAX +/- 10 msec) and
  - satellite (GEOS) →accuracy of +/- 0.1 msec , GPS +/- 0.5 msec

- **Question:** What problems does this solve/not solve?
  - too coarse for synchronization of many kinds of event (e.g. 1 000 000 ops/sec)

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

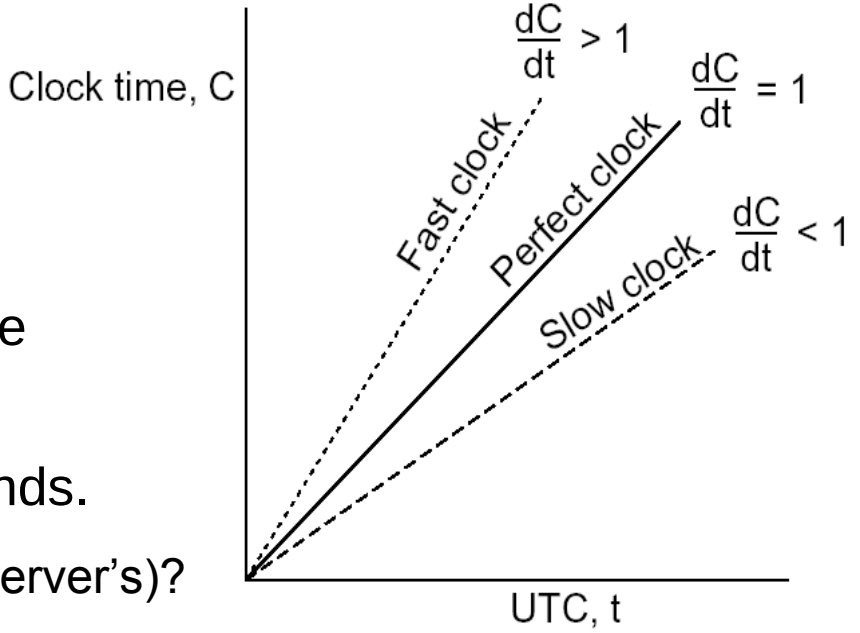
DVS

Physical Clocks – Basic Principle

TECHNISCHE UNIVERSITÄT DARMSTADT

- Every machine has a timer
  - generates an interrupt H times (typically 60) per second
- There is a clock in machine p that ticks on each timer interrupt. Denote the value of that clock by Cp(t) , where t is UTC time.
- Ideally, we have that for each machine p, Cp(t) = t → dC/dt = 1
  - In reality → relative error of 10^{-5}
- If ρ **is the max. drift rate**

$$1-\rho \leq \frac{dC}{dt} \leq 1+\rho.$$



- **Goal:** Never let two clocks differ by more than δ time units
- → synchronize at least every $\delta/2\rho$ seconds.
- Negative time (machine's clock faster than server's)?

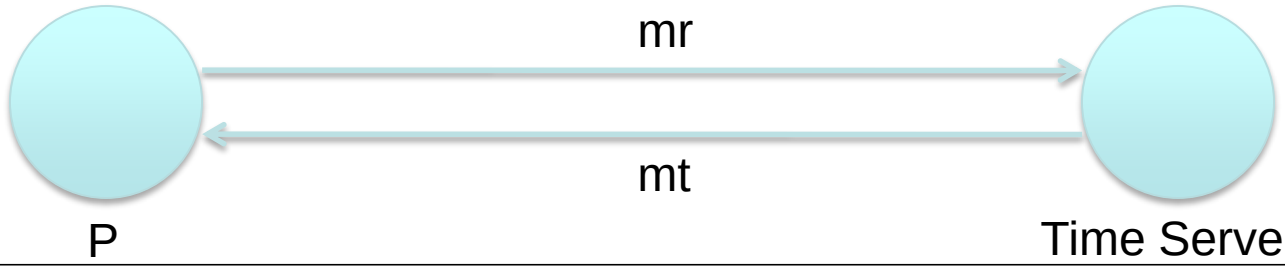
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Clock Synchronization – Cristian's Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

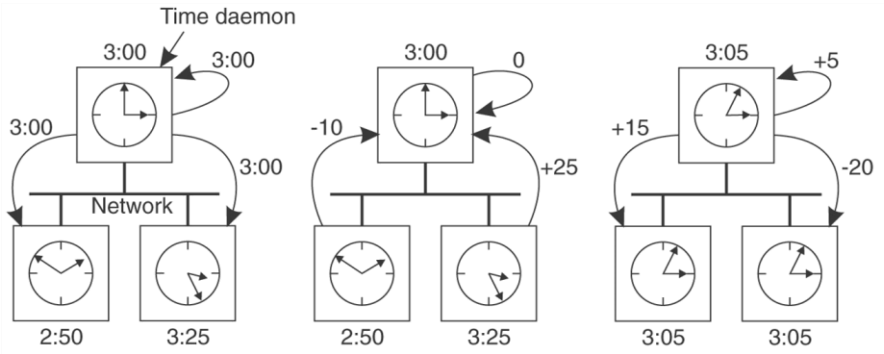
- Time adjustment
  - Slow clock → match UTC
  - Fast clock → slow down clock till match occurs
- Cristian's algorithm → Every machine asks a **time server** for the accurate time at least once every $\delta/2\rho$ seconds
  - Handling round-trip delays?
  - Delays occur because of contention at server and network
    - Ideally: P sends message at t, receives message at t + T_{round}
    - Really: delay going = min, delay coming = min → accuracy = +/- ($T_{round}/2$ – min)



DVS

Clock Synchronization – Berkeley Algorithm

- The Berkeley Algorithm addresses single point of failure and malicious node issues
  - The time server (master) polls slaves periodically → Received times are averaged and each machine is notified about individual compensation
  - Centralized algorithm



- Decentralized Algorithm
  - Every machine broadcasts its time periodically for fixed length synchronization interval
  - Averages the values from all other machines
- Network Time Protocol (NTP) → +/- 50msec
  - the most popular one used by the machines on the Internet
  - uses an algorithm that is a combination of centralized/distributed

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Logical Clocks

TECHNISCHE UNIVERSITÄT DARMSTADT

- **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. Why partial?

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Logical Clocks

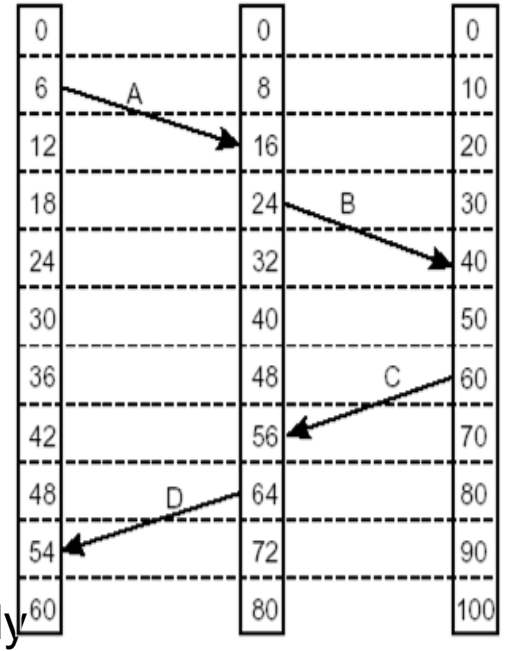
TECHNISCHE UNIVERSITÄT DARMSTADT

- **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.
- Application:
  - **totally ordered** and **causally ordered multicast** → Lect. 2!

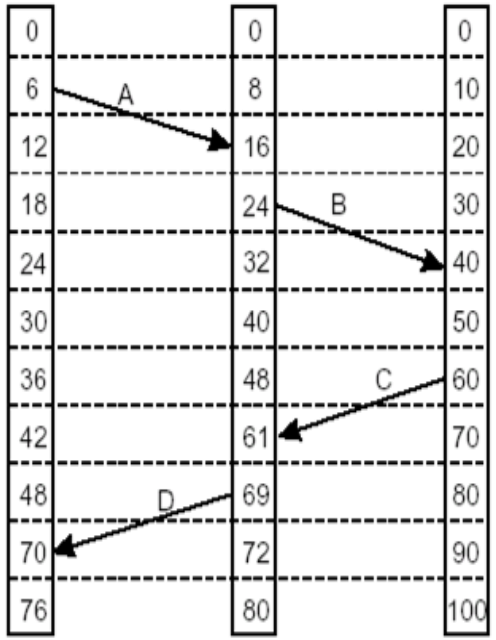
Logical Clocks – Lamport's algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lamport's algorithm well known from other lectures
  - (KN1, KN2, Prof. Steinmetz)
- **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

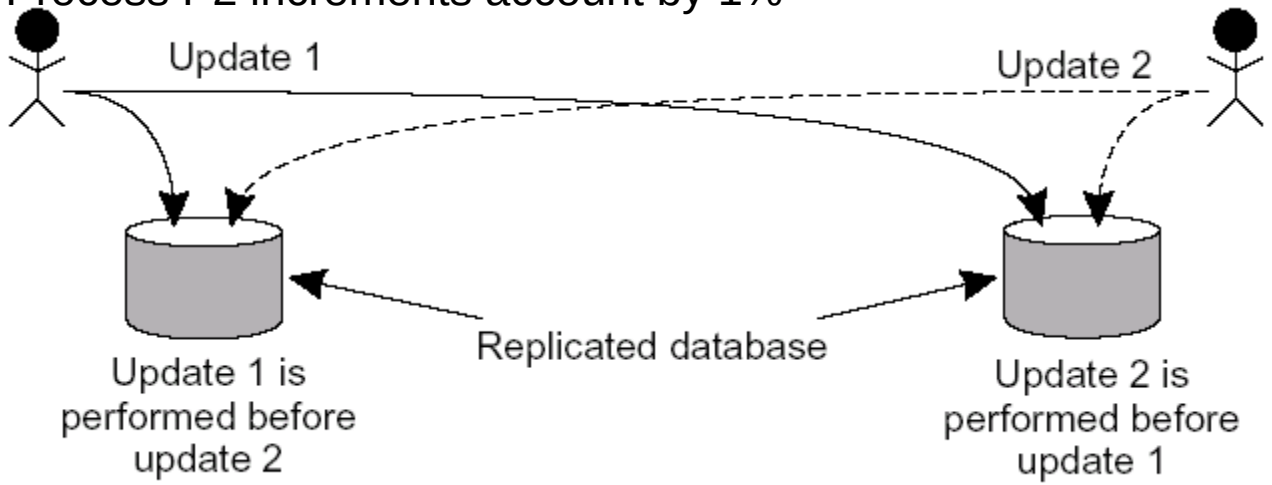
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Example: Totally-Ordered Multicast

TECHNISCHE UNIVERSITÄT DARMSTADT

- **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

DVS

Vector Timestamps

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Vector Timestamps

TECHNISCHE UNIVERSITÄT DARMSTADT

The Fidge logical clock is maintained as follows:

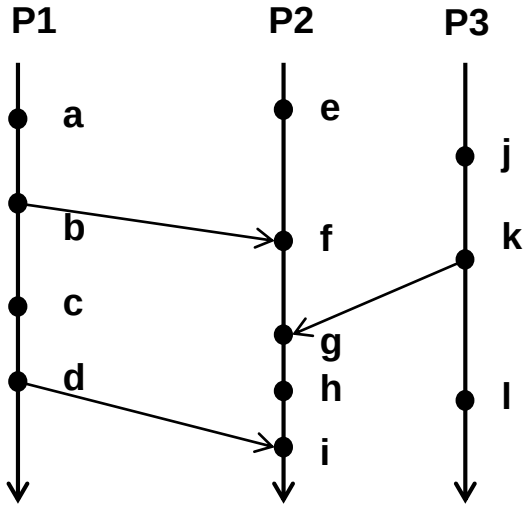
1. Initially all clock values are set to the smallest value.
2. The local clock value is incremented at least once before each primitive event in a process
3. The current value of the entire logical clock vector is delivered to the receiver for every outgoing message.
4. Values in the timestamp vectors 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.

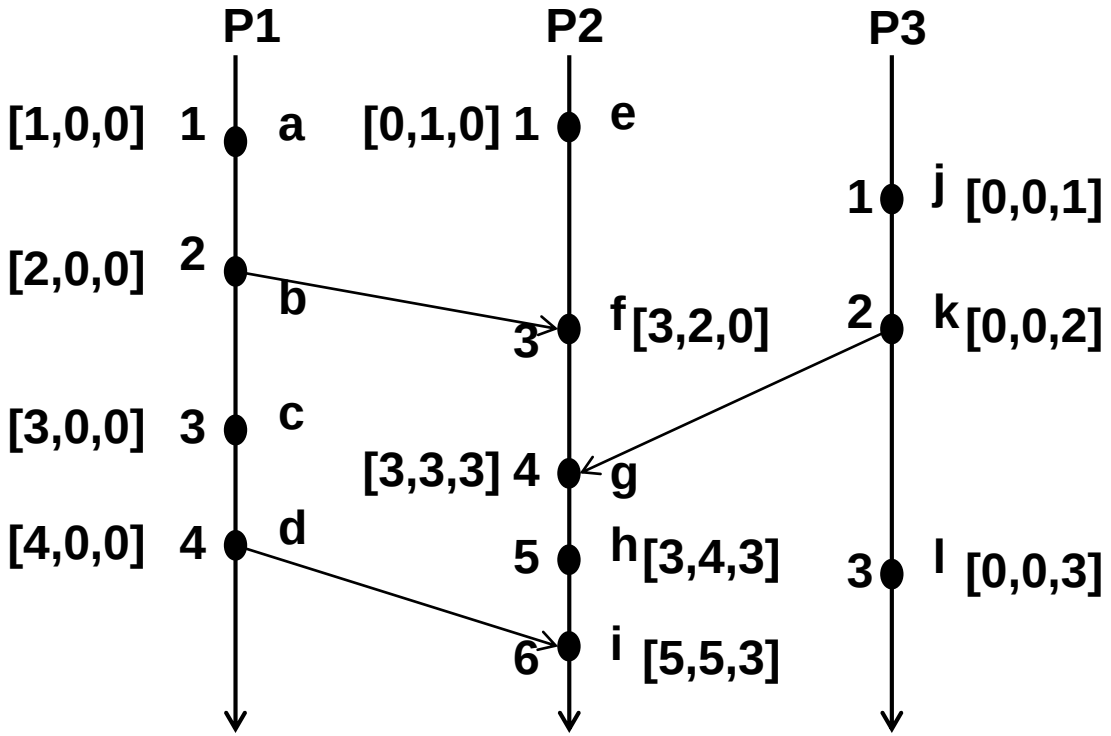
Colin Fidge, "Logical Time in Distributed Computing Systems", *IEEE Computer*, Vol. 24, No. 8, pp. 28-33, 1991

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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 $\rightarrow$ h,
  - 3 < 4 but c <!> g.

- Fidge clocks:
  - f $\rightarrow$ h since 2 < 4 is true,
  - b $\rightarrow$ 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.

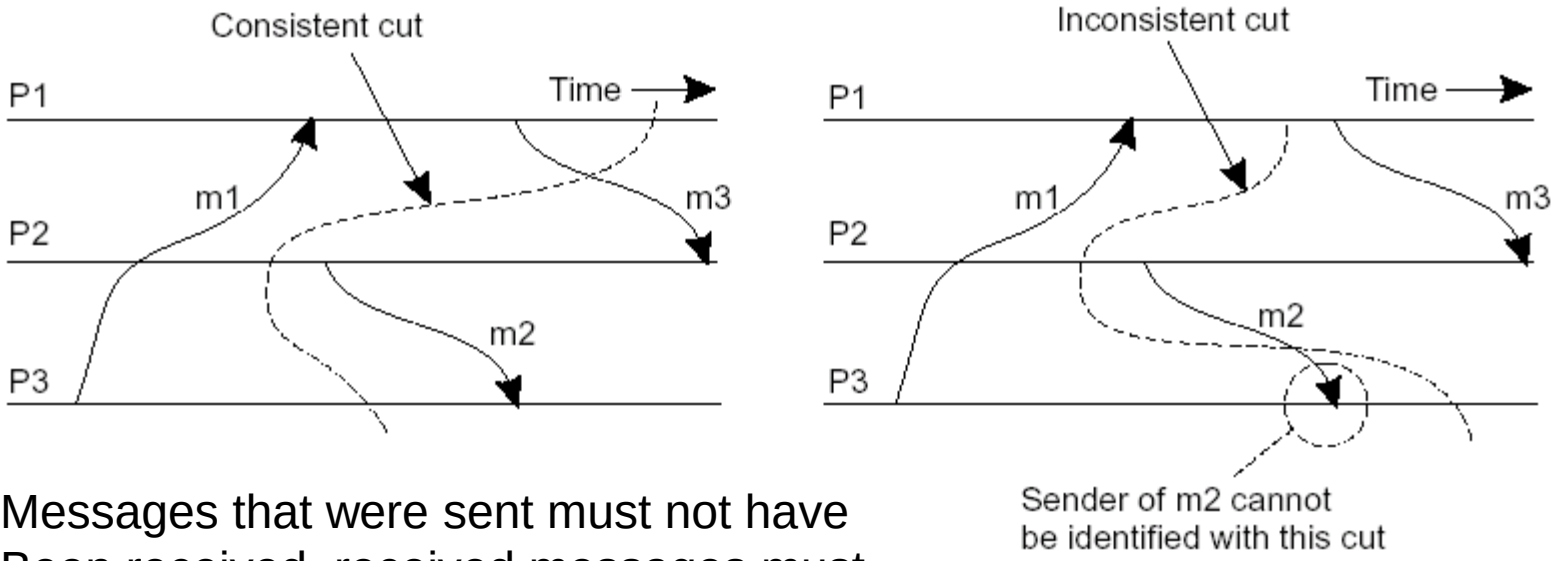
Global State

TECHNISCHE UNIVERSITÄT DARMSTADT

DVS

Global State

- **Basic Idea:** Sometimes you want to collect the current state of a distributed computation, called a **distributed snapshot**. It consists of all local states and messages in transit.
- **Important:** A distributed snapshot should reflect a **consistent** state:



Messages that were sent must not have
Been received, received messages must
Have been recorded as sent!

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

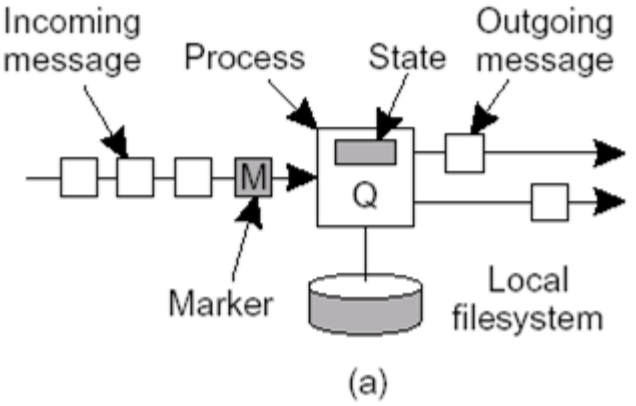
Global State

TECHNISCHE UNIVERSITÄT DARMSTADT

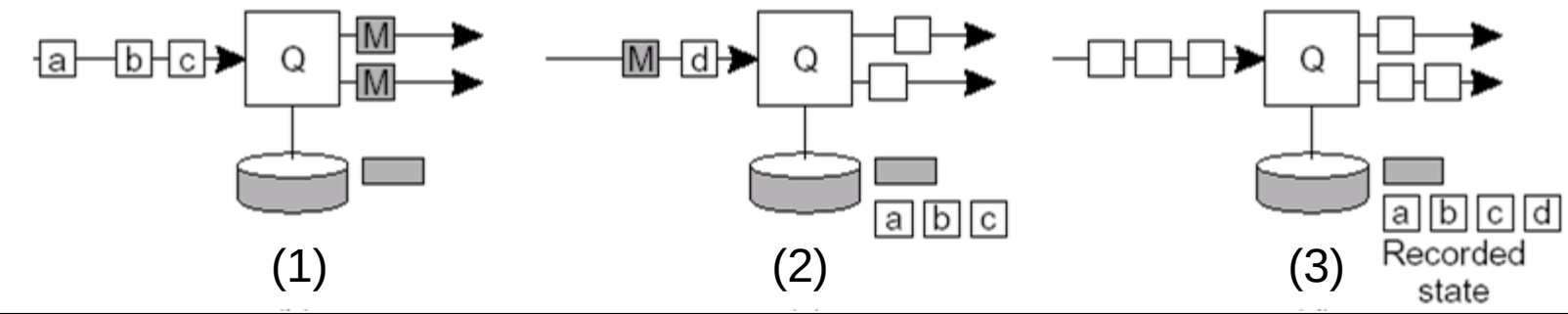
- **Note:** Processes are connected by unidirectional channels, any process P can initiate taking a distributed snapshot
- P starts by recording its own local state and subsequently sends a marker along each of its outgoing channels
- When Q receives a marker through inbound channel C, its action depends on whether it had already recorded its local state:
  - Not yet recorded: it records its local state, and sends the marker along each of its outgoing channels
  - Already recorded: the marker on C indicates that the channel's state should be recorded: all messages received before this marker and the time Q recorded its own state.
- Q is finished when it has received a marker along each of its incoming channels
- Parallel snapshots are possible, therefore markers have initiator ID

Global State

- Organization of a process and channels for a distributed snapshot



1. Process Q receives a marker for the first time and records its local state
2. Q records all incoming messages
3. Q receives a marker for its incoming channel and finishes recording the state of the incoming channel



Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Election Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

DVS

Election Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Principle:** Many distributed algorithms require that some process acts as a coordinator. The question is how to select this special process **dynamically**.

- **Note:** In many systems the coordinator is chosen by hand (e.g., file servers, DNS servers). This leads to centralized solutions => single point of failure.
- **Question:** If a coordinator is chosen dynamically, to what extent can we speak about a centralized or distributed solution?
- **Question:** Is a fully distributed solution, i.e., one without a coordinator, always more robust than any centralized/coordinated solution?

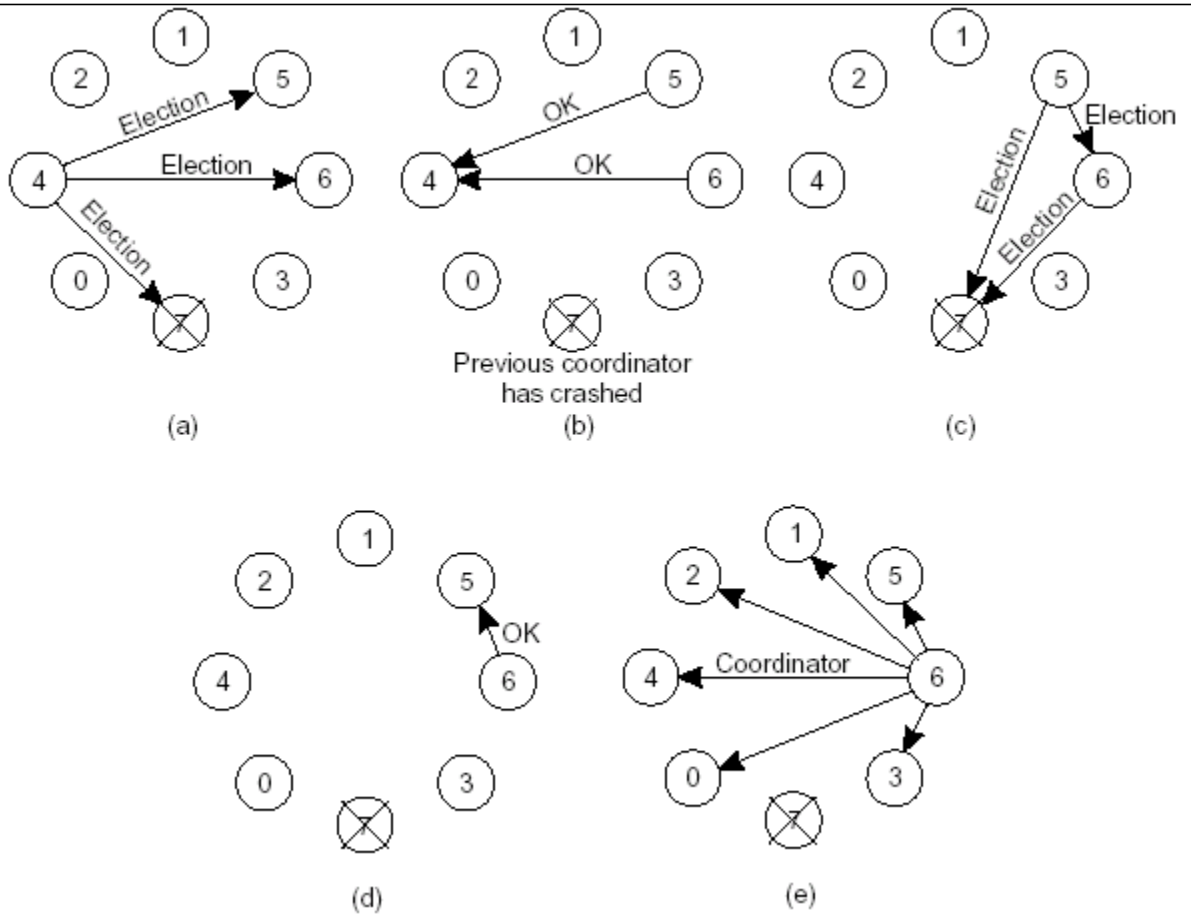
The Bully Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Principle:** Each process has an associated priority (weight). The process with the highest priority should always be elected as the coordinator.
- **Issue:** How do we find the heaviest process?

- Any process can just start an election by sending an election message to all other processes (assuming you don't know the weights of the others).
- If a process *P*heavy receives an election message from a lighter process *P*light, it sends a take-over message to *P*light. *P*light is out of the race.
- If a process doesn't get a take-over message back, it wins, and sends a victory message to all other processes.

The Bully Algorithm



Question: We're assuming something very important here – what?

Assumption: Each process knows the process number of other processes

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

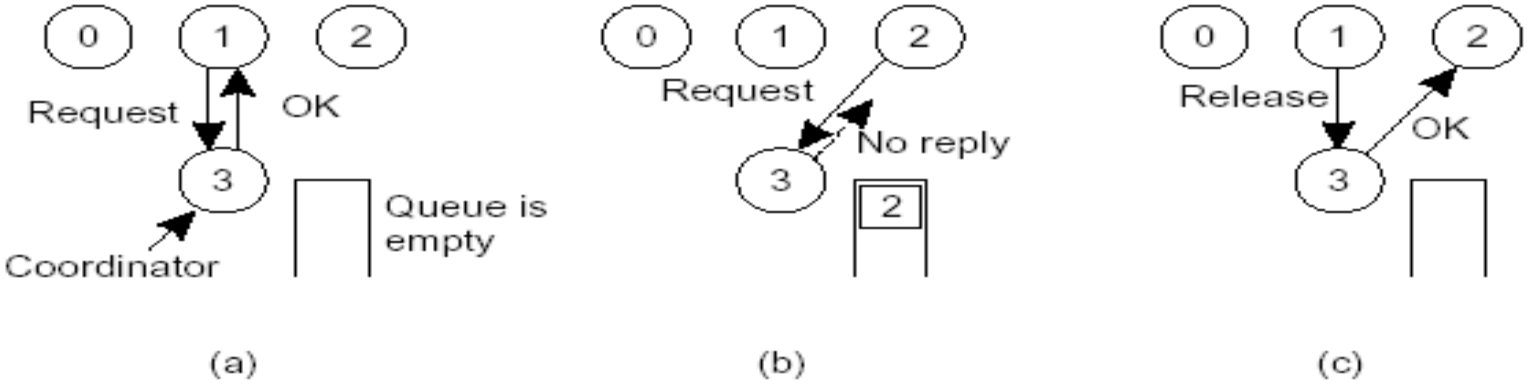
Election in a Ring

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Principle**: Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator.
- Any process can start an election by sending an election message to its successor. If a successor is down, the message is passed on to the next successor.
- If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known.
- The initiator sends a coordinator message around the ring containing a list of all living processes. The one with the highest priority is elected as coordinator.
- **Question**: Does it matter if two processes initiate an election?
- **Question**: What happens if a process crashes during the election?

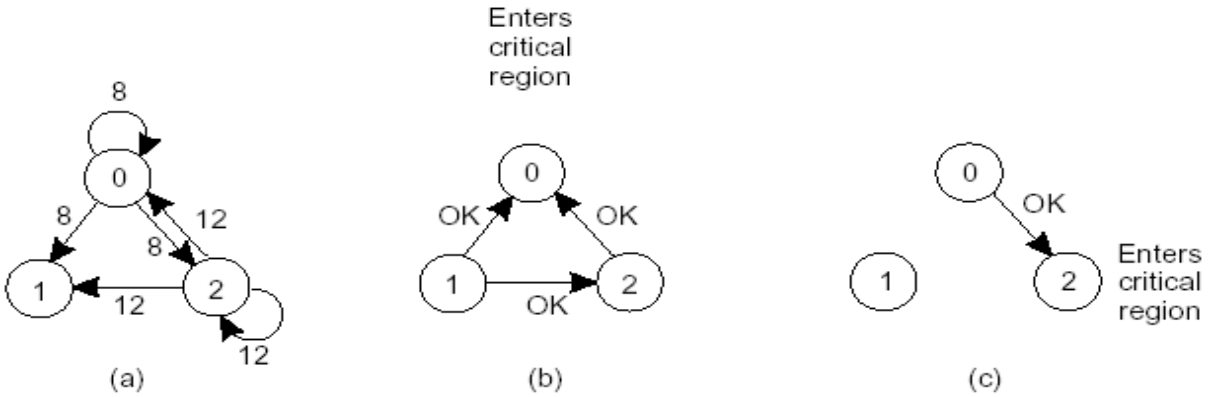
Mutual Exclusion

- **Problem:** A number of processes in a distributed system want exclusive access to some resource.
- **Basic solutions:**
- Via a centralized server.
- Completely distributed, with no topology imposed.
- Completely distributed, making use of a (logical) ring.
- **Centralized:** Really simple:



Mutual Exclusion: Lamport

- A process wanting to enter a critical region
  - Sends a (reliable) message with resource sought, own PID, timestamp
  - Since messages are reliable an ack must be received
  - Upon receipt (depending on recipients status)
    - Not in critical region and not interested → send OK
    - Recipient in critical region → queue incoming message
    - Waiting to enter → compare TS of own message with received message, lowest wins
  - Sending process waits till it gets the OK from all other processes

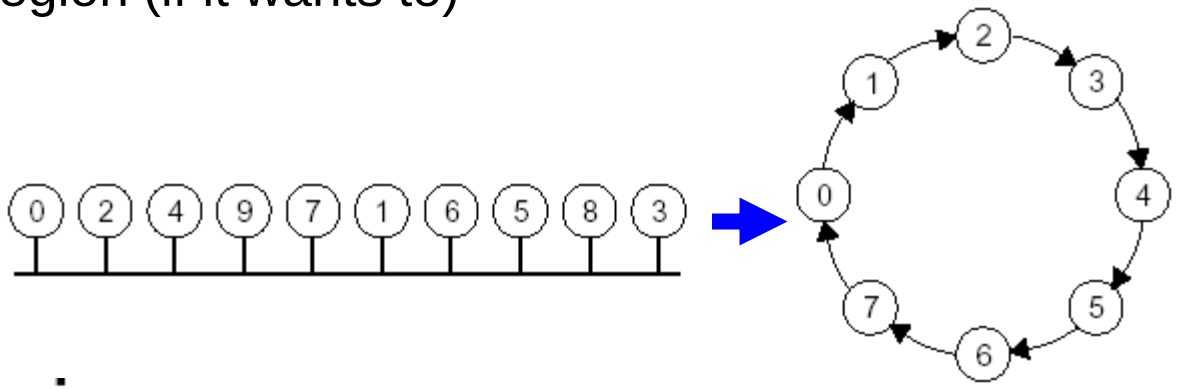


Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Mutual Exclusion: Token Ring Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Essence:** Organize processes in a **logical** ring, and let a token be passed between them. The one that holds the token is allowed to enter the critical region (if it wants to)



(a) (b)

Comparison:

Algorithm	# msgs	Delay	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Time issues
- Clocks
  - Physical Clocks
  - Logical Clocks
  - Synchronization
- Global State
- Election algorithms
- Mutual Exclusion

Thank You!

TECHNISCHE UNIVERSITÄT DARMSTADT

Questions?

DVS