Mobile and Ubiquitous Computing
2022-23
MEIC/METI

Context-Awareness I

Context-Awareness (1/2)

- What is Context?
- What is the role and components of a Context-Aware Systems?
- Distributed Context-Aware Systems
- Local vs. Distributed Context-Aware Systems
- Collaborative vs . Non-collaborative Distributed Context-Aware Systems
- Examples of Sensors and Types of Context

Definition of Context

- The word “context” is subject to multiple interpretations and has been researched in various fields like psychology, philosophy, and computer science

- In the area of CS, context was initially perceived as user location

- In the last years, it has been enriched with other sources of information such as identity, activity, and state of people, groups, and objects

- *Context is any information that can be used to characterize the situation of an entity:*
  - *an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

Definition of Context (cont.)

- Consider only relevant information w.r.t. interaction between user and application
  - thus, application developers can focus on a subset of the user environment data for a given application scenario

- Example:
  - context needed by museum guide app to assist museum visitors is their location and native language
  - other environmental information (e.g. body temperature, marital status) is not relevant

- In mobile and ubiquitous systems, the idea is:
  - context supports systems to be cognizant of its user's state and surroundings and to modify their behavior based on this information

- A user's context can be quite rich, consisting of attributes such as:
  - physical location,
  - physiological state (such as body temperature and heart rate),
  - emotional state (such as angry, distraught, or calm),
  - personal history,
  - daily behavioral patterns

Definition of Context (cont.)

- Another definition (by enumeration of examples):
  - **computing context**, such as network connectivity, communication costs, and communication bandwidth, and nearby resources such as printers, displays, and workstations;
  - **user context**, such as the user's profile, location, people nearby, even the current social situation;
  - **physical context**, such as lighting, noise levels, traffic conditions, and temperature;
  - **temporal context**, such as time of a day, week, month, and season of the year

- Possible refinement:
  - there are certain types of context that are, in practice, more important than others for characterizing the situation of a particular entity:
    - *location*, *identity*, *activity*, and *time*.

- These context types not only answer the questions of **who, what, when, and where** but also act as identity indexes into other sources of contextual information.

Context-Aware Systems

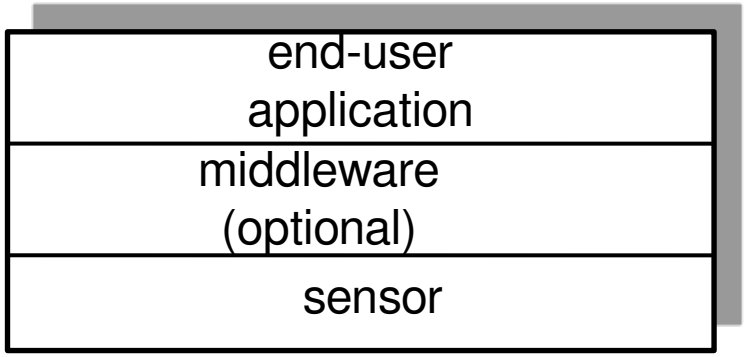
- *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

- This results in three main features that context-aware systems may provide:
  - **presentation of information and services to a user**
    - systems that provide information to the user augmented with contextual information (e.g., phone's contact list enhanced with location information) or that provide services based on the current user's context (e.g., show me restaurants nearby);
  - **automatic execution of a service**
    - systems that execute a service automatically based on the current context (e.g., automatically updating my status on a social network based on accelerometer data—sleeping, walking, and running);
  - **tagging of context to information for later retrieval**
    - systems that are able to associate digital data with the user's context (e.g., virtual notes that are attached to certain locations, for others to see).
  - **System reconfiguration**

Context-Aware Systems

- These systems can be described as:
  - *Centralized or distributed system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task;*
  - *including end-user applications* that use context information provided by *sensors*.

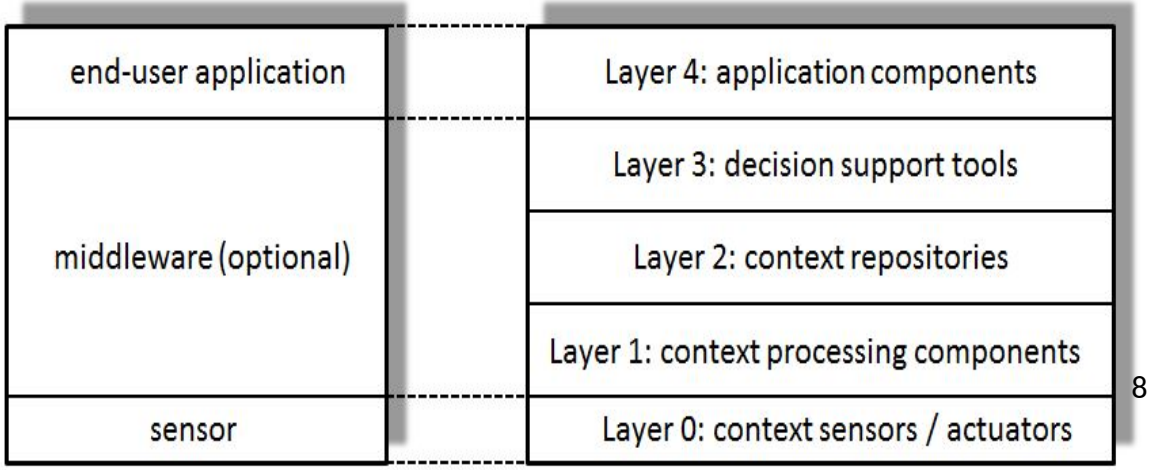
- From an architectural point of view:
  - end-user applications layer - consumers of context information
  - middleware layer - communication and coordination issues between distributed components
  - sensors layer - producers



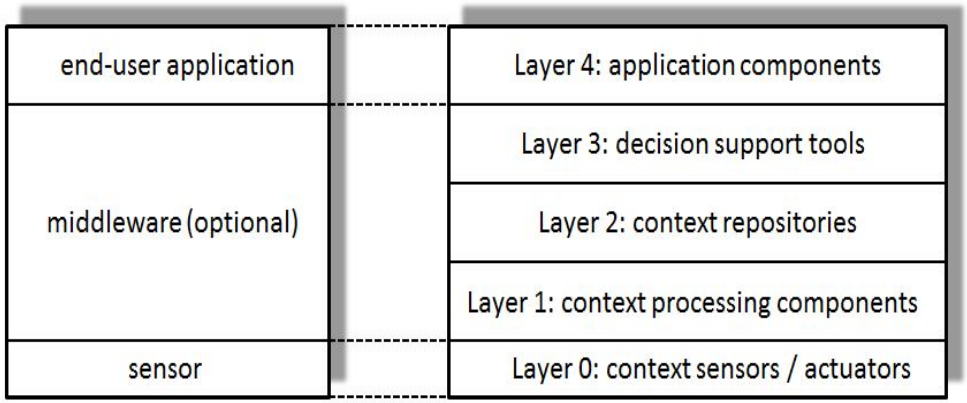
Context-Aware Systems (cont.)

- In simple context-aware systems, the end-user application communicates directly with the sensor:
  - e.g., a mobile phone that automatically switches off GPS when battery is below a certain capacity

- Today, it is widely acknowledged that additional infrastructural components are desirable:
  - to reduce the complexity of distributed context-aware applications,
  - improve maintainability, and
  - promote reuse.



Context-Aware Systems: Typical Stack



- layer 0 – context sensors and actuators that:
  - provide the interface with the environment, either by capturing it (sensors) or by modifying it (actuators);
- layer 1 – components that:
  - (i) assist with processing sensor outputs to produce context information that can be used by applications, and
  - (ii) map update operations on the higher-order information back down to actions on actuators;
- layer 2 - context repositories that:
  - provide persistent storage of context information and advanced query facilities;
- layer 3 – decision support tools that:
  - help applications to select appropriate actions and adaptations based on the available context information;
- layer 4 – application components that:
  - are integrated in client applications using programming toolkits.

Local vs. Distributed Context-Aware Systems

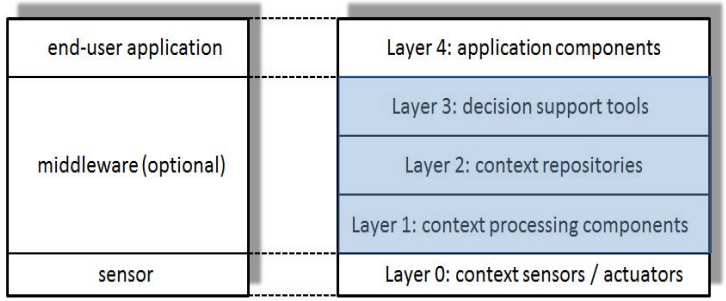
- Local:
  - sensors and applications are **tightly coupled** (usually through a direct physical connection)
  - e.g. a mobile phone application that sets the mode to silent, while its owner is jogging

  (the accelerometer that provides the “running” context is directly attached to the mobile phone as well as the application that uses that context to activate the silent mode)

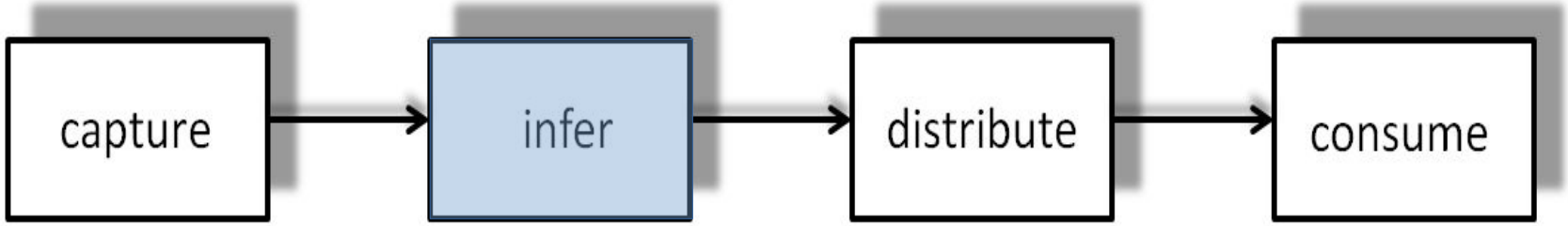
- Distributed:
  - do **not have a direct physical connection** between the sensor and the application.
  - thus, it is possible to have multiple applications receiving information from the same sensor.
  - it is also possible that multiple dispersed sensors produce information to be consumed by a single application
  - e.g. a mobile phone may broadcast to a group of friends that its owner is currently walking, to decrease the probability of incoming calls during that period.
  - e.g. multiple thermometers in a home send periodic updates to a central thermostat.
  - in this case, the thermometers are not tightly coupled to the recipient of its information.

Context Processing: Taxonomy

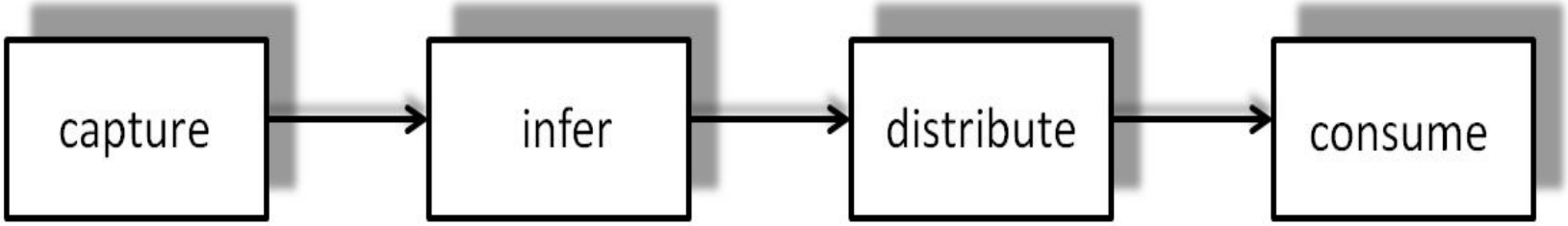
- Clearly categorizes the architectural options available to the application developer, explaining the advantages and disadvantages of each approach;
- Analyzes the problems that are specific to distributed context-aware systems, whose (distributed) components have to communicate with each other, and how that affects availability and scalability of such systems;



- Four layers:



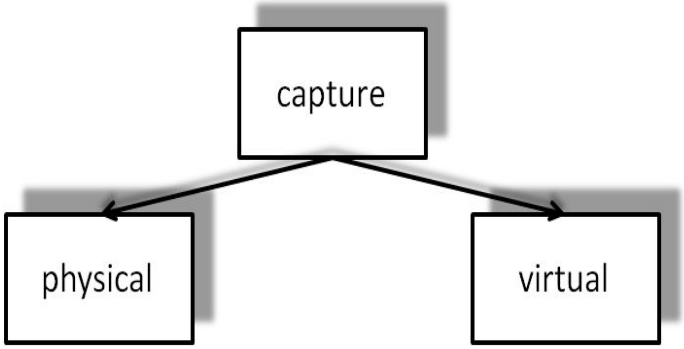
Context Processing: Taxonomy (cont.)



- Context data is **captured** from the environment using sensors
- A **context inference step** is needed to obtain higher-level aggregated data:
  - e.g. GPS device captures geographical coordinates from which a place (city, building, etc.) is inferred.
  - this step is also known as *feature extraction*.
- **Distribution of context data** among its components in an efficient and scalable manner
- Client applications **consume** this information in order to provide relevant services to their users.

Capture

- The Capture layer is responsible for acquiring context data from the environment using sensors.



- The word “sensor” not only refers to sensing hardware but also to every data source which may provide usable context information:
  - *physical,* and v*irtual*

- **Physical sensors** are hardware sensors capable of capturing physical data such as light, audio, motion, and location.
  - location is, by far, the most researched type of physical sensor

- **Virtual sensors** acquire context data from software applications, operating systems, and networks.
  - detecting new appointments on an electronic calendar and watching the file system for changes are examples of virtual sensors

Examples of Sensors and Types of Context

- Examples of *location*, *identity*, *activity*, and *time sensors.*

	Physical sensors	Virtual sensors
Location	*Outdoor*: global positioning system (GPS), global system for mobile communications (GSM); *indoor*: Bluetooth, 802.11 cells	Networked calendar system, travel-booking system, user's login on location-aware computer, IP subnet
Identity	*Based on something you are*: fingerprint reader, retina scanner, microphone; *based on something you have*: smart card reader, RFId	Various authentication schemes at the operating system or application level
Activity	Mercury switch, accelerometer, motion detector, thermometer, UV sensors, camera	Keyboard or mouse activity, application usage
Time	Clock	Operating system timer

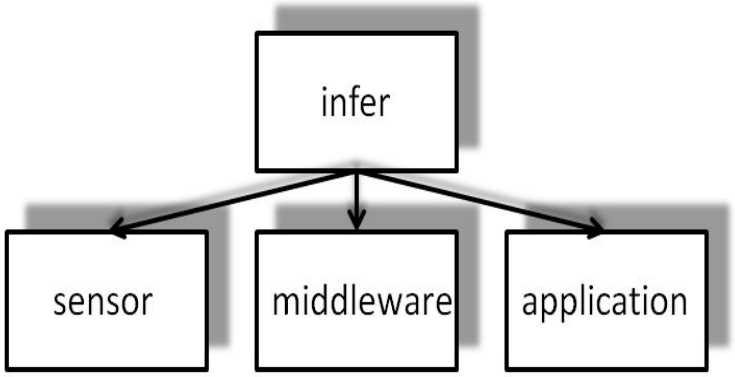
Infer

- It is responsible for reasoning and interpreting raw context information provided by sensors on the Capture Layer:
  - e.g. end-user applications do not need to know the exact latitude and longitude of an entity, being much more interested in knowing the place (city, street, etc.) in which the entity is located

- The *Context-Inference* layer, is also known as the *Preprocessing Layer*

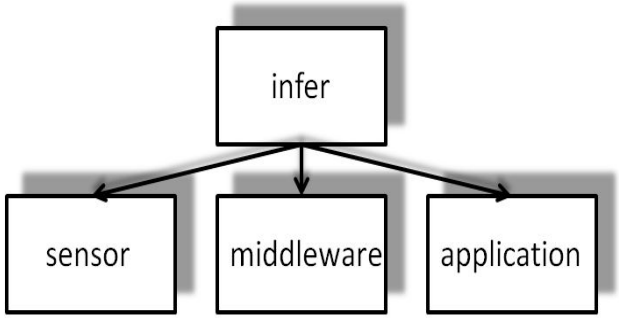
- Context inference usually involves some kind of reasoning:
  - a transformation is needed to reach a higher level of abstraction
  - it is normally associated with classification techniques
  - e.g. inferring the user activity (walking, running) from an accelerometer

- Where should the Inference layer be placed?



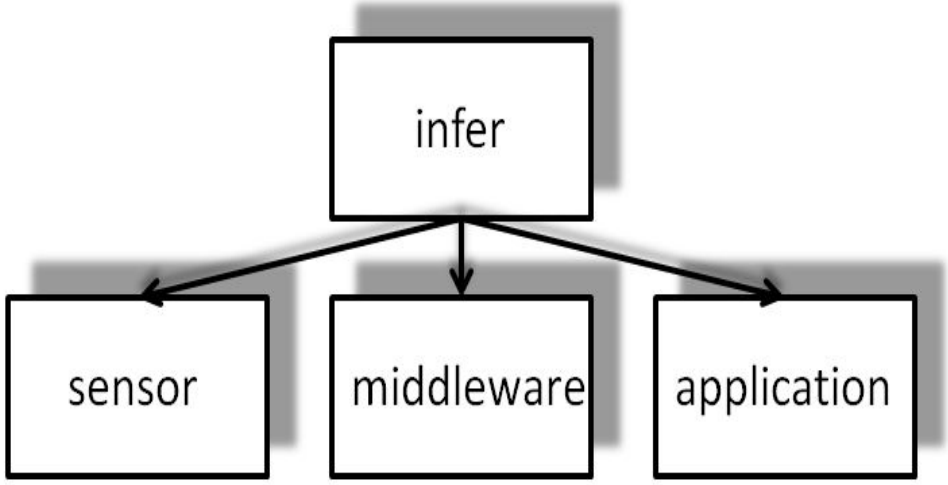
Infer (cont.)

- The type of information inferred from sensorial data is **specific** to each application:
  - e.g. taking the data provided by an **accelerometer**, an application might use that information to know whether a user is **walking** or running, while another may be more interested in detecting climbing stairs or rising in elevators.
  - it may make sense to move the inferring task to the application because of the specific semantic needs it tries to accomplish.
  - transforming raw sensorial data into high-level context information can be a resource-demanding task
    (unsuitable for applications on constrained devices such as mobile phones)
  - some systems use a middleware component running in a machine with higher capability



Infer (cont.)

- The three locations for inference are intimately related to the following properties of a context-aware system:
  - Network bandwidth consumption
  - Complexity (CPU/memory consumption)
  - Reusability
  - Personalization



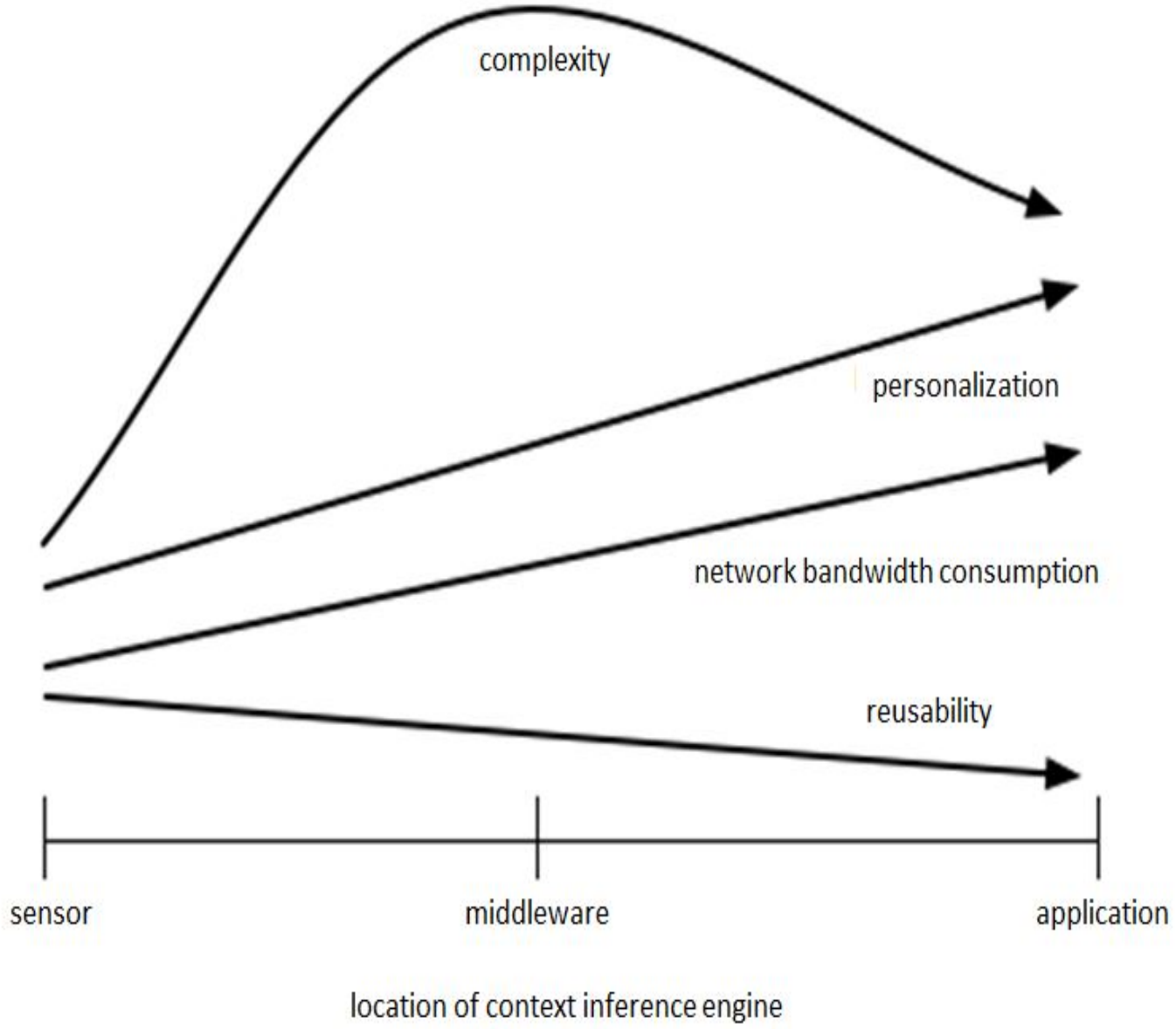
Infer (cont.)

- Network bandwidth consumption
  - context inference transforms fine grained sensorial information into coarse-grained high-level data
  - it reduces the amount of information needed to represent a context message
  - moving the inference layer closer to the sensor results in less network bandwidth consumption
- Complexity (CPU/memory consumption)
  - context inference mechanisms can lead to high CPU and RAM consumption
  - even if the device is capable of computing the information, it can lead to excessive and unsustainable battery consumption
  - thus, the developer has to deploy the inference engine in a resourceful machine (e.g., a server)
  - most physical sensors are attached to low-capability devices (to increase mobility) and do not provide context-inference mechanisms.
  - virtual sensors often run on servers or desktop computers (where they can access virtual context information from databases, shared calendars, etc.), so they are able to provide higher-level context information than their physical counterparts.

Infer (cont.)

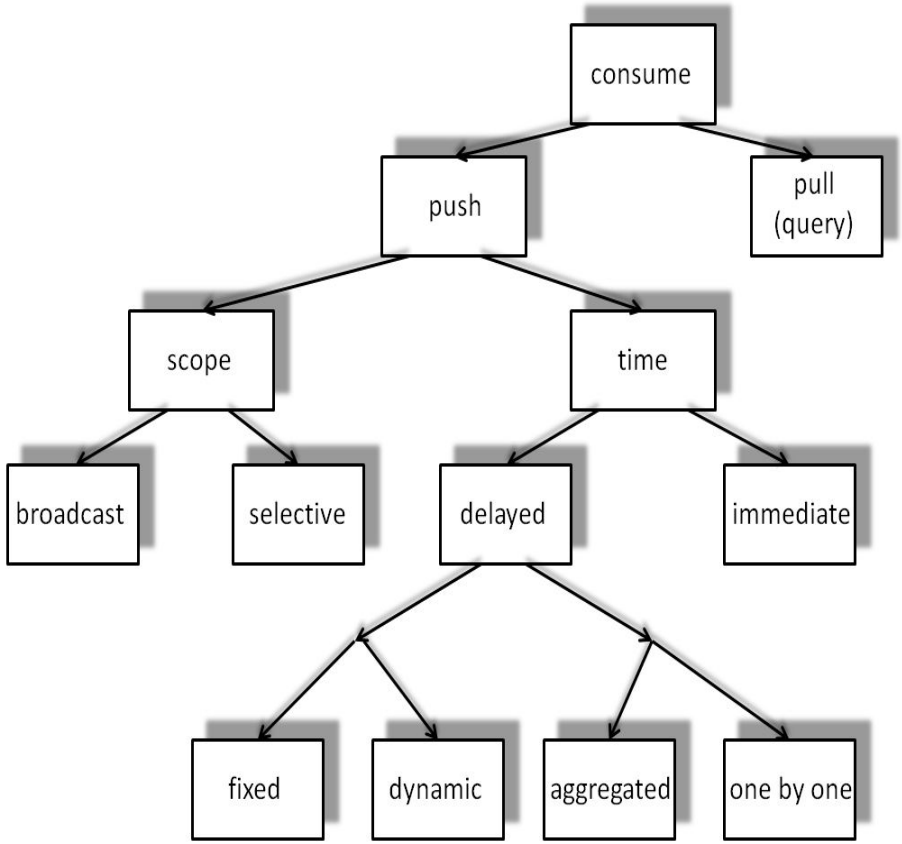
- Reusability
  - makes sense for context types as location, where the output is sufficiently *standard* to be used by multiple applications
  - e.g. inferring the street name from geographical GPS coordinates is a recurring requirement in location-aware systems and does not make sense to (re)implement in every end-user application
  - moving the inference layer closer to the sensor increases reusability
- Personalization
  - ability to personalize the inference engine to better suit individual needs
  - e.g. a phone application that allows its users to associate a certain movement (like drawing an imaginary circle with the phone) to some meaning or activity (e.g., going to lunch)
  - mediation of ambiguity - context inference includes dealing with ambiguous data and explicit user mediation is needed (thus adapting the inference mechanism to suit individual needs)
  - Can be circumvented by API with varied levels of abstraction.

Infer (cont.)


complexity
personalization
network bandwidth consumption
reusability
sensor
middleware
application
location of context inference engine

Consume

- After context information has been:
  - captured from sensors,
  - inferred into higher-level data,
  - distributed through the network,
  - it will be consumed by client applications.

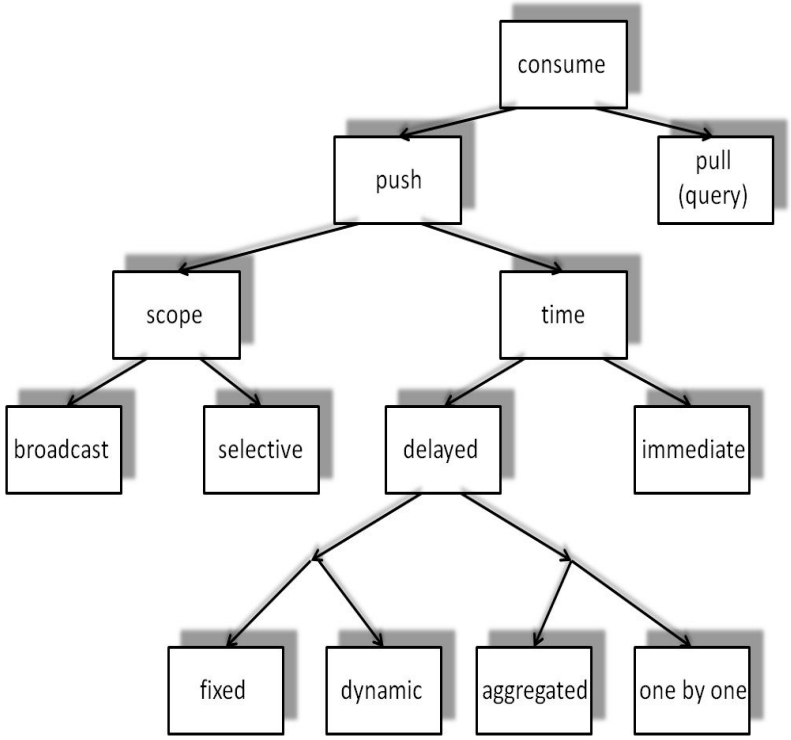


Consume (cont.)

- Pull-based (or query) systems:
  - *pull-based systems* - the consumer (client application) queries the component that has the information (sensor or middleware) for updates
  - this can be done manually (e.g., the user clicks the “refresh” button) or periodically (e.g., checking for updates every 5 min)

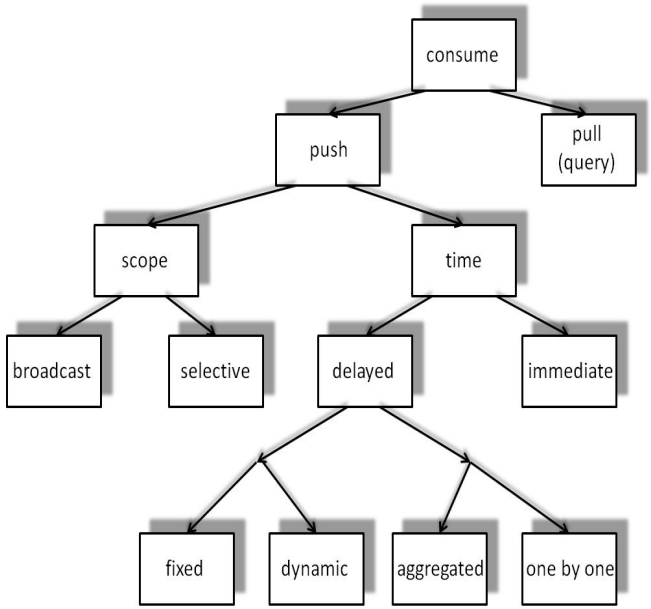
- Push-based systems:
  - the component that has the information is responsible for delivering it to interested client applications.

- The main distinction between these approaches lies on who initiates the communication:
  - if a client application initiates the communication, the system is pull-based;
  - otherwise it is push-based.



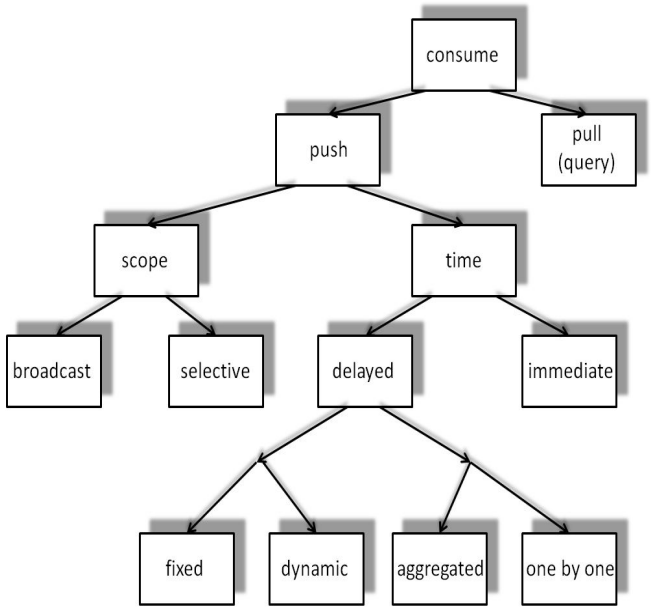
Consume (cont.)

- Pull-based mechanisms are simple to implement:
  - the client application only has to query a certain component with a certain periodicity.
  - the application can simply delegate the responsibility of refreshing the information to the user.
  - the polling interval can change according to the application current needs.
  - e.g., an application running on a mobile phone can increase the polling interval to save battery life, when the battery power falls below a certain threshold.



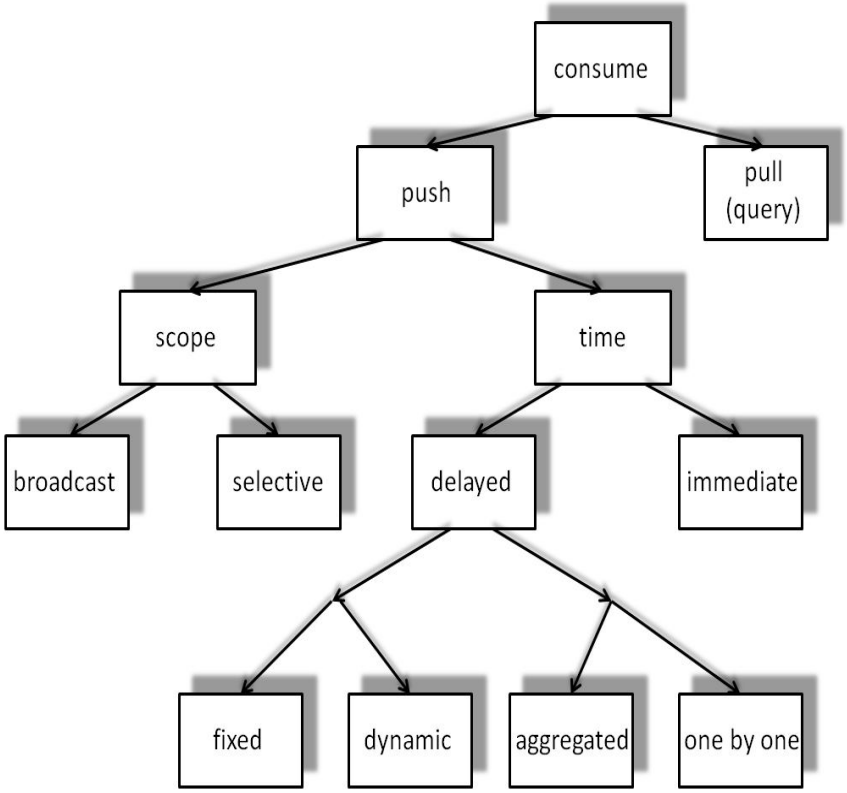
Consume (cont.)

- However, this mechanism suffers from two problems that can be critical:
  - context information reaches the application with a delay (may equal the polling interval)
  - if polling interval is 5 min**, the application may receive context information from 5 min ago**, compromising its usefulness.
  - the application can minimize this disadvantage by **reducing the polling interval**, but then, the second problem may start **occurring—most queries will be unnecessary** since there is no new context information available
  - since the application does not know when there will be updates, it has no alternative as to keep asking until there is new information, leading to **wasted network bandwidth and CPU consumption**



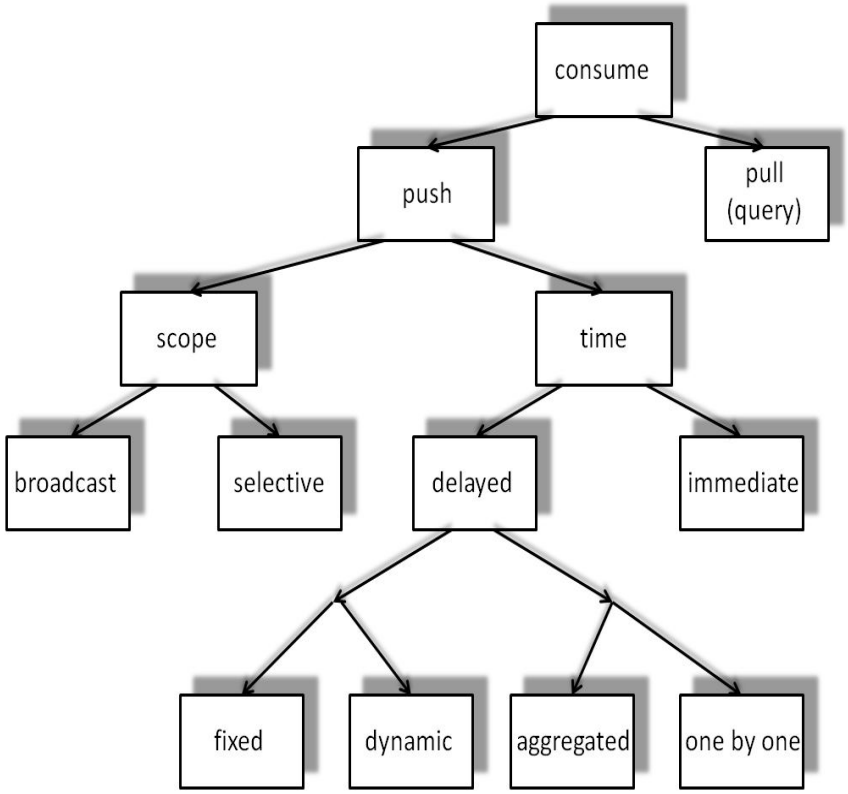
Consume (cont.)

- Push-based mechanisms are **more complex** than pull-based mechanisms.
- The component that has the information is also responsible for delivering it:
  - this component has to **know how to reach every possible consumer** that is interested in that information.
- Usually, a permanent connection with all the consumers is necessary.
- If the system has a large number of consumers, its **performance may degrade**.
- Furthermore, these systems have **poor scalability** as every new consumer degrades the performance.
- There are two measures to overcome the scalability problem: *reduce scope* and/or *relax delivery time*.



Consume (cont.)

- Users may be interested only in a **subset** of the system's available context information:
  - e.g. an application that enhances the traditional phone contact list with contextual information (such as location and availability) is **only interested in context updates for the people in its contact list**.
  - thus, if person A has person B in his list of contacts, A is notified if B moves from one location to another, but other people in the system without B in their contact list are not notified.

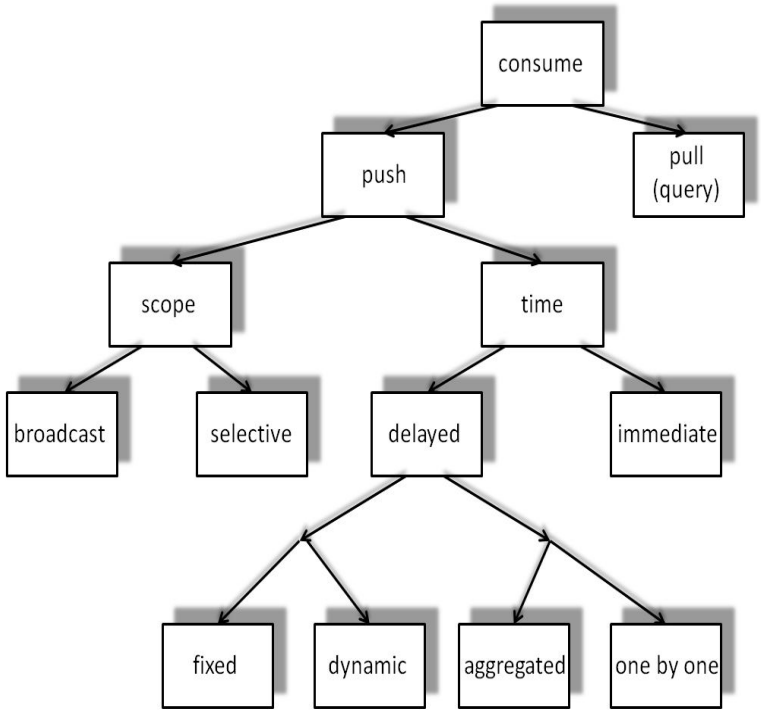


Consume (cont.)

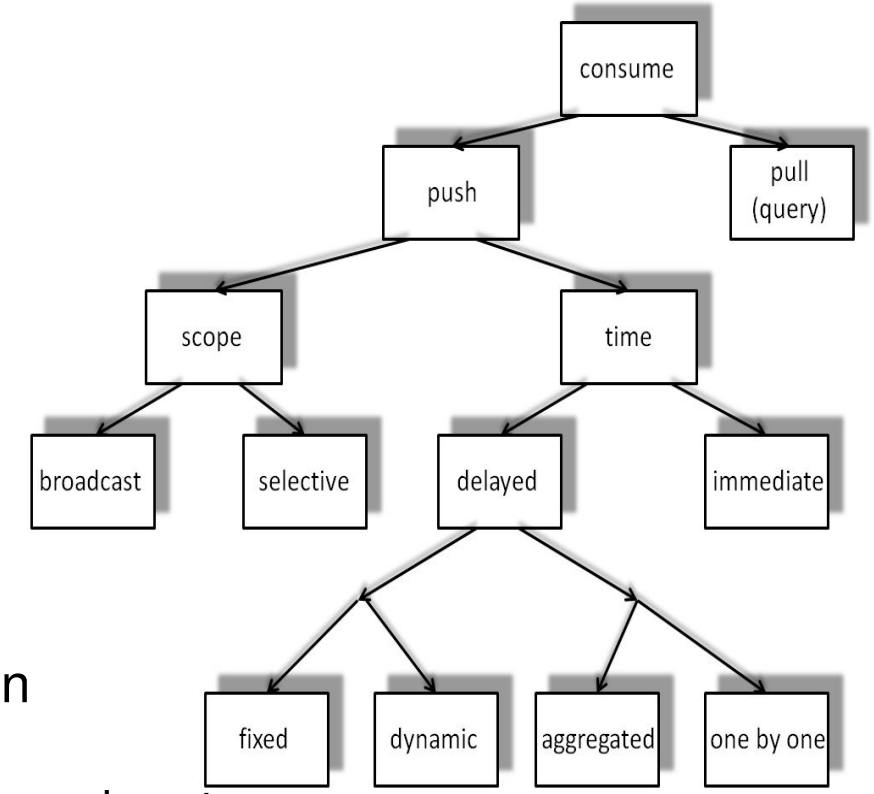
Selective scope systems propagate only a **subset of the available context information** based on user's selection.

Most context-aware systems propagate all the available context information, thus falling into the broadcast scope category

- Using scope is an effective technique to reduce the number of pushed messages in the system:
  - reducing the required **outbound network bandwidth** on the component that has the information and improving overall scalability.
  - *quenching* - a mechanism that allows sensors to know whether client applications are interested in their information and only sending information in that case



Consume (cont.)



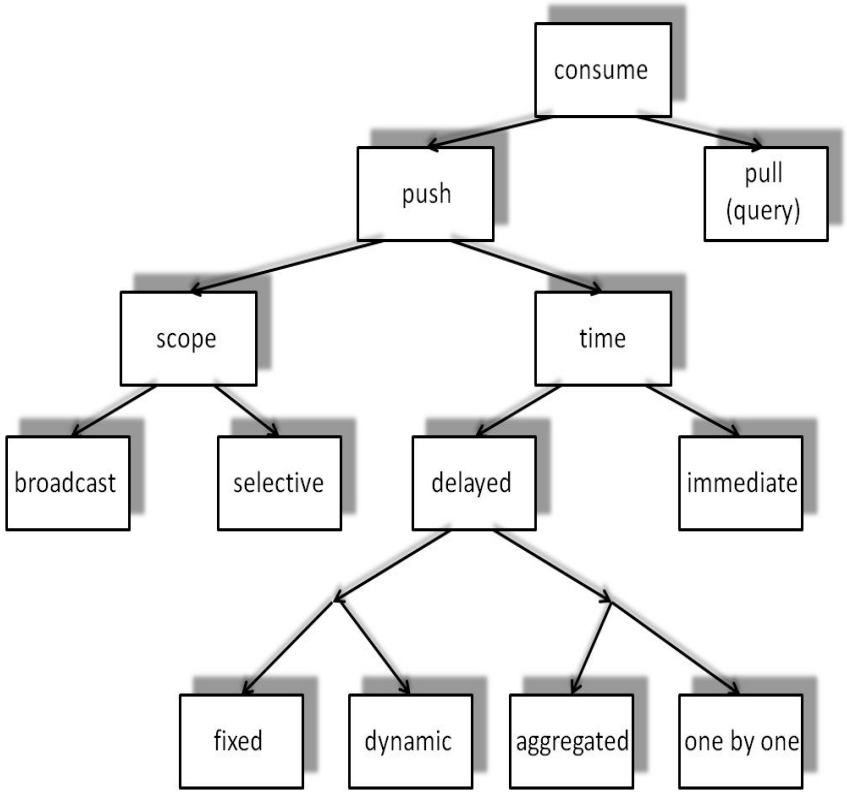
- Time:
  - to improve push-based systems scalability,
  - assuming that certain context information **does not need to be immediately delivered**

- Users **tolerate some lag** as long as the information does not require urgent attention.
  - e.g., a context-aware system that shows friends near me is not required to be continuously up to date.
  - on the other hand, an application that shows the products around a user in a supermarket loses its usefulness if context information is not propagated as soon as possible

- Some of these issues have been studied in the context of *optimistic replication algorithms:*
  - increase availability and scalability of distributed data sharing systems by allowing replica contents to diverge in the short term

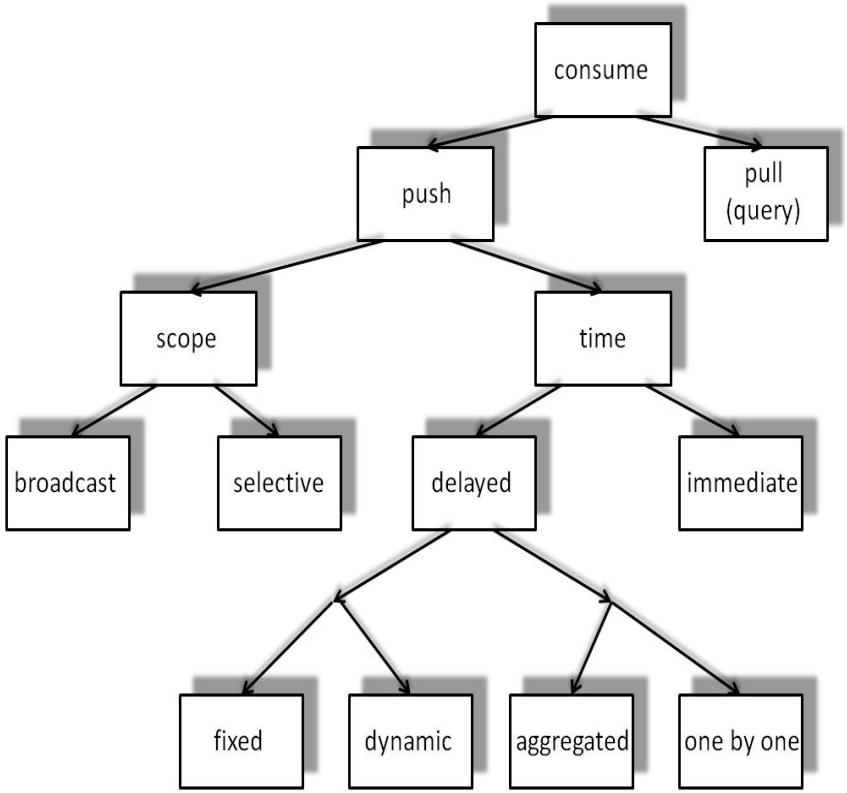
Consume (cont.)

- Fixed vs. dynamic:
  - must decide when to actually push the updates.
  - decision can be made based on a **fixed criteria such as time period** (push the update every x seconds) or **amount of retained information** (push the update when there is more than x Kbytes of context information)
  - obviously, both criteria can be used: e.g. to improve the scalability of a context-aware message application



Consume (cont.)

- Aggregated vs. one-by-one:
  - since there is a delay in context propagation, information must be retained and may start accumulating
  - when the system decides to push the update, there may be a considerable number of messages to transmit
  - simple approach is to **transmit the messages one-by-one**, as would be the case if there was not any delay (this can be extremely inefficient)
  - alternative solution is to **aggregate all delayed information in just one** (e.g. GPS locations during a certain amount of time)
  - even when all the retained messages are relevant, they can still be **aggregated in one big message**, achieving **higher levels of compression** and much **less round-trips** in the connection path



Consume (cont.)

- Note that:
  - in context-aware related literature, the term “aggregation” is often associated with the combination of information from different sensors, along with conflict resolution.
  - do not confuse both uses of the same term!

- More to read:
  - See several examples of context aware systems in the bibliography (Distributed Context-Aware Systems.

  Paulo Ferreira, Pedro Alves. SpringerBriefs in Computer Science, 2014)

Example (1/2)

Estimote Smart Beacons - welcome to the contextual computing era-
https://www.youtube.com/watch?v=SrsHBjzt2E8

Example (2/2)

Estimote Sticker Beacons - Introducing Nearables - https://www.youtube.com/watch?v=JrRS8qRYXCQ

Examples of Context-Aware Applications and Sensors

	Description	Sensors used
GUIDE [25]	Information for city visitors	WiFi (location)
Welbourne [97]	Mode of transit: walking, running, riding a vehicle	Clock, GSM/WiFi (location), accelerometer
ContextContacts [62]	Enhanced phone's contact list	GSM, phone activity, Bluetooth (nearby environment)
UbiqMuseum [15]	Assists museum visitors	Bluetooth (location)
AwareMedia [9]	Support coordination at an operation ward	Bluetooth (location), video camera, shared calendar
Stiefmeier [91]	Help workers perform critical and complex assembly tasks in a car production environment	Body-worn, car-mounted, and tool-mounted accelerometers
BikeNet [36]	Cyclist experience mapping	Magnetometer, inclinometer, speedometer, microphone, GPS, GSR stress monitor, CO^2 meter
CenceMe [59]	Social activities (dancing, lunching)	Microphone, GPS, camera, Bluetooth (nearby environment), accelerometer
SoundSense [57]	Music events' sharing	Microphone, camera, GPS
Upcase [82]	Daily activities (working, driving, sleeping, resting, walking outside)	Luminosity, microphone, temperature, accelerometer