



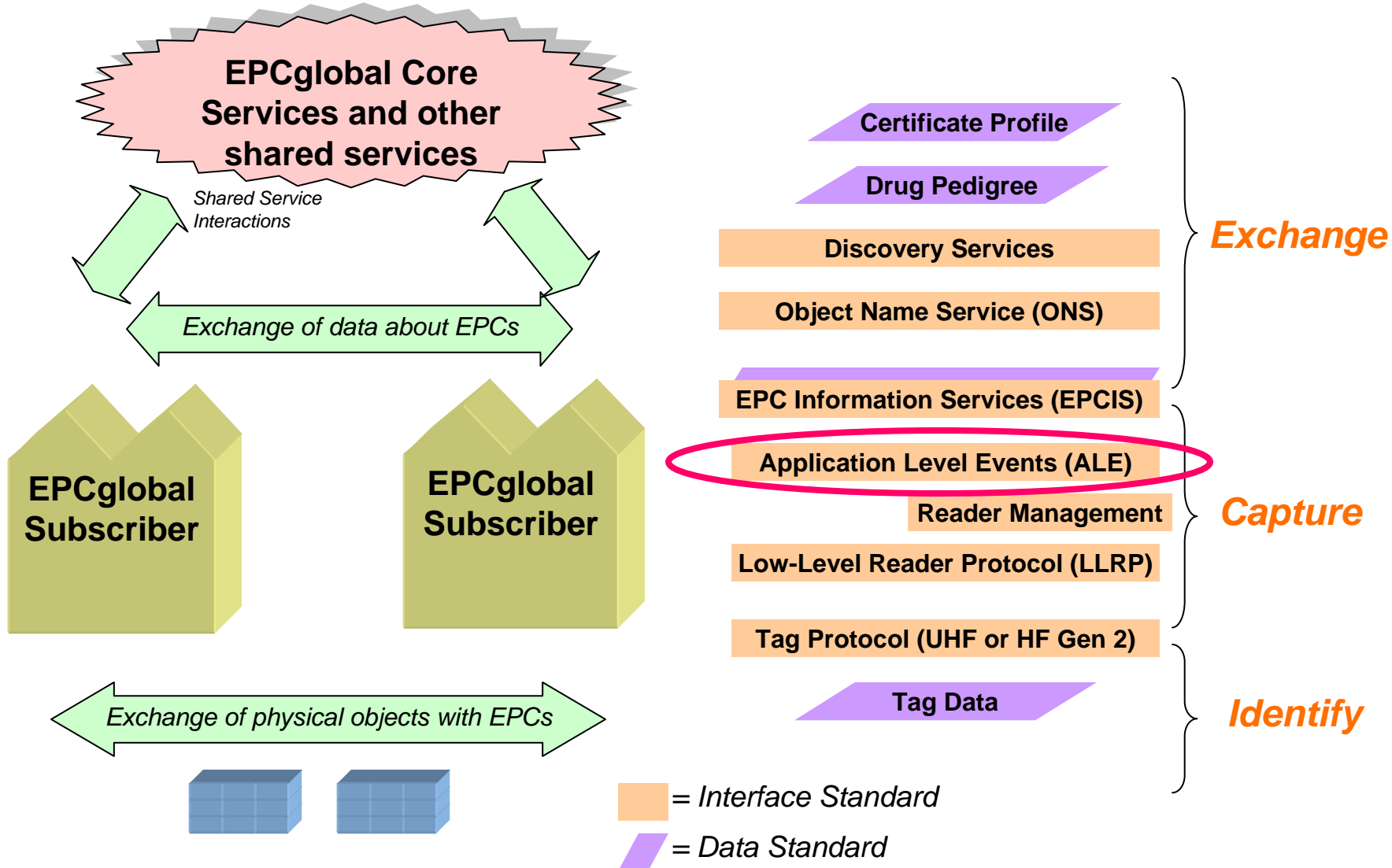
*Application Level Events 1.1(ALE 1.1) Overview
Filtering & Collection WG, EPCglobal
March 5, 2008*

Identify individual products, cases, loads, assets, etc, so that they can be tracked individually

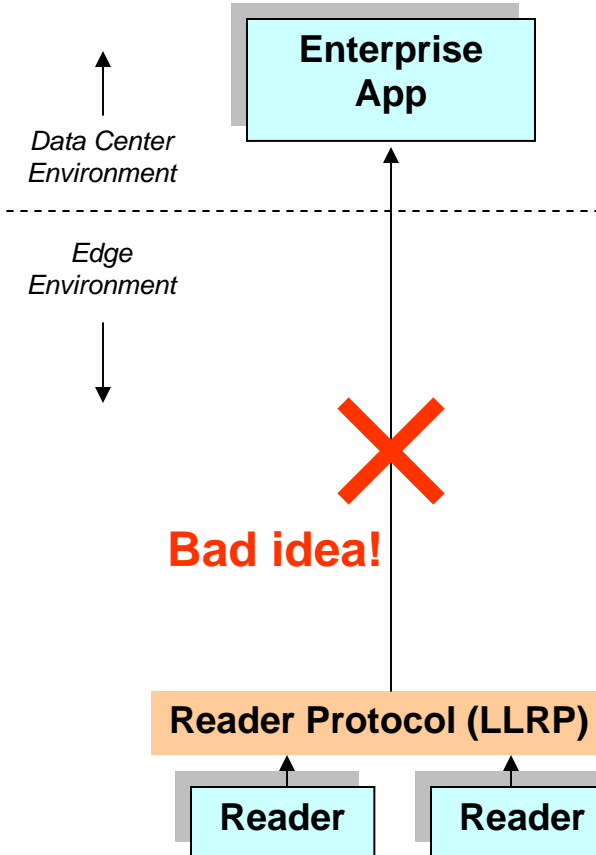
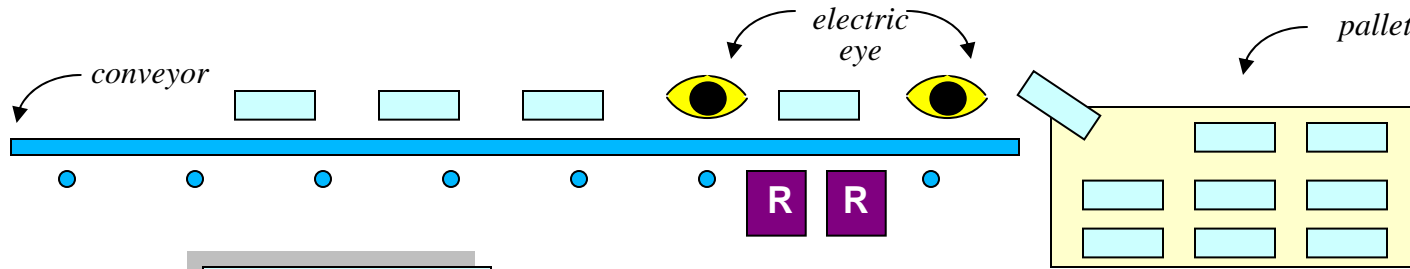
Capture data about the movement of physical assets, creating visibility

Exchange data with IT applications and trading partners, to turn visibility into information and action

EPCglobal Standards Overview



Data Capture Example – Palletizer



Too much data
No local business logic

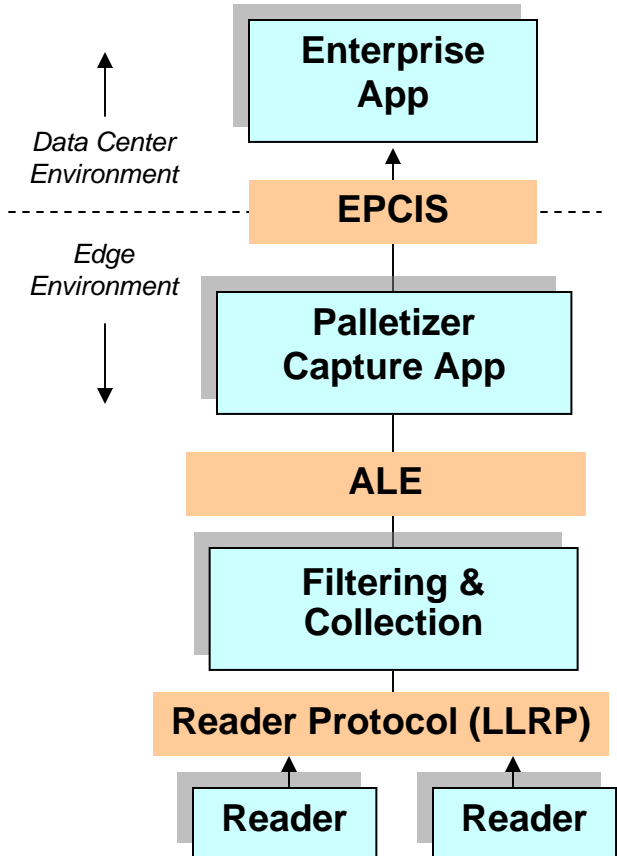
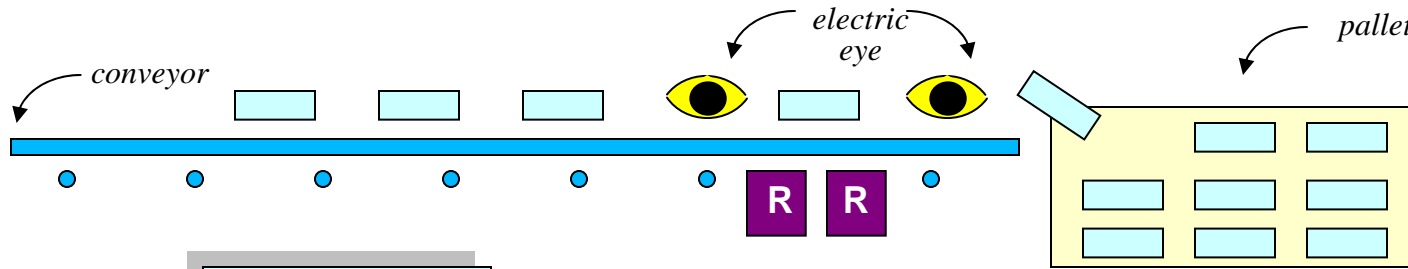
- E.g., what if no tag read? Stop? Divert? Continue?

Physical details not hidden from enterprise application

- Hard to know what data means
- Can't change the operational process

dozens of individual tag read and sensor events from specific antenna/device

Data Capture Example – Palletizer



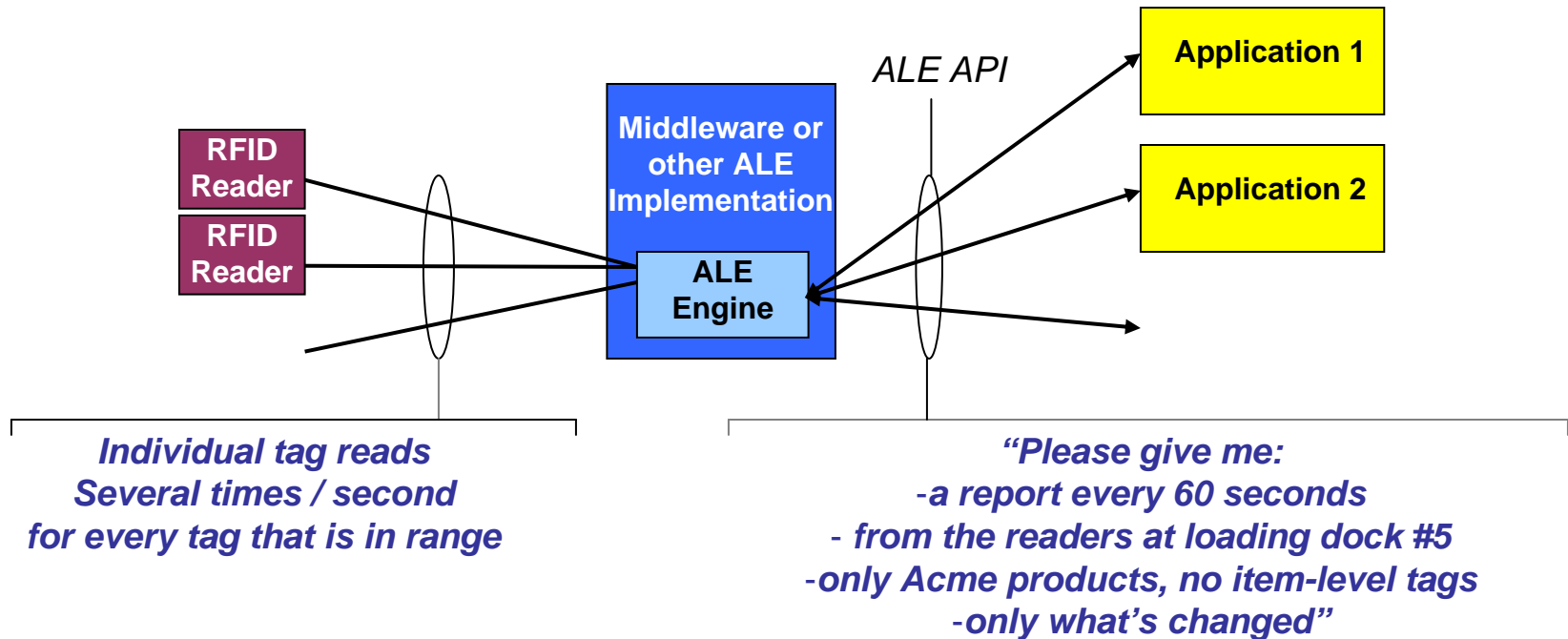
“at time T , the association of the following case tags to the following pallet tag was created at palletizer #3, to fulfill order #1234”

Service consumed by enterprise -- operational details hidden

“between the time the case crossed the two beams at location L , the tag X was read with temperature T ”

Service consumed by local business logic -- device details hidden

dozens of individual tag read and sensor events from specific antenna/device



Reduces volume of data from readers to applications

Elevates level of abstraction for application writers

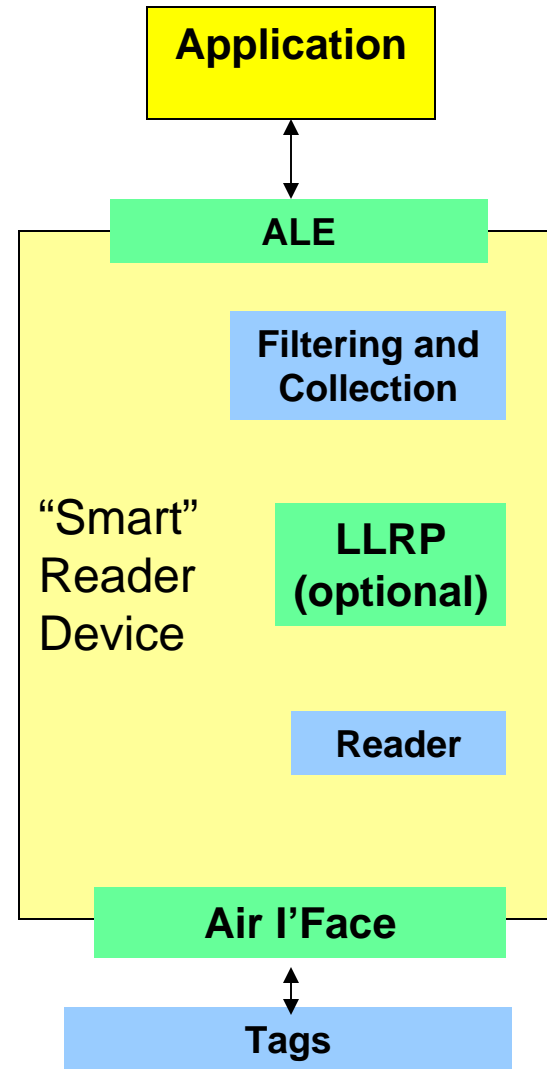
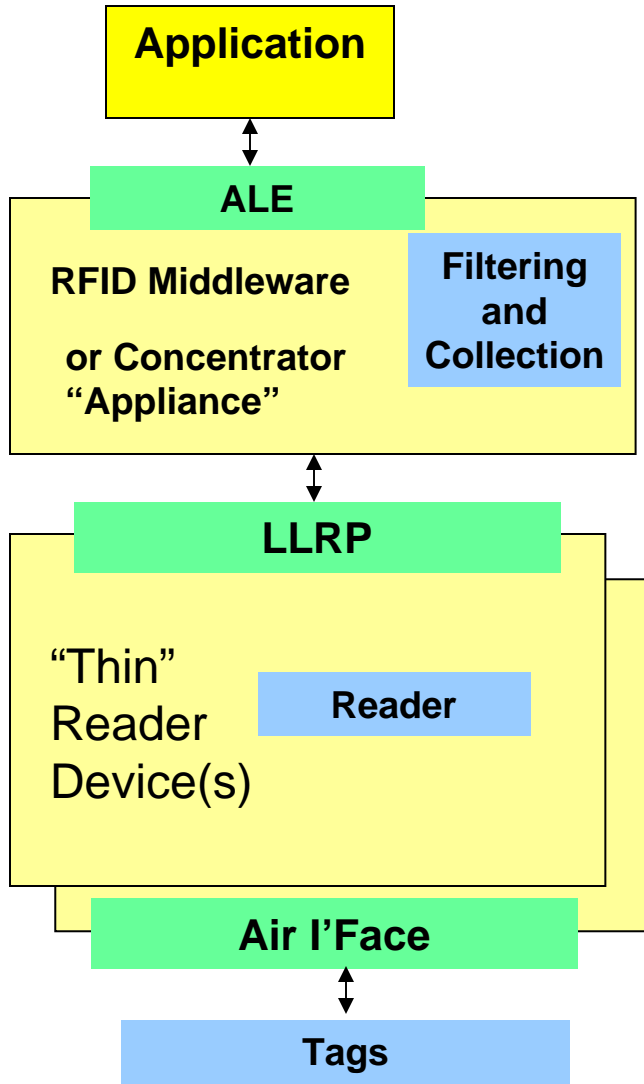
Insulates applications from device details

Shares data among multiple applications

Extensible to vendor changes

Integrates easily using standard XML / Web Services technology

ALE: an Interface, not a software component



Things that might implement ALE

- **Middleware:**
 - Software system that interfaces to readers and other devices via network protocols
 - Exposes ALE to other software applications that wish to read and write tags
- **Concentrator:**
 - Similar to middleware, but embedded in a network appliance
- **“Smart” Reader:**
 - Embedded ALE implementation provides a high-level interface for applications that want to interact with that reader
- **Printer:**
 - Similar to smart reader
- Most of these are commercially available today (ALE 1.0)

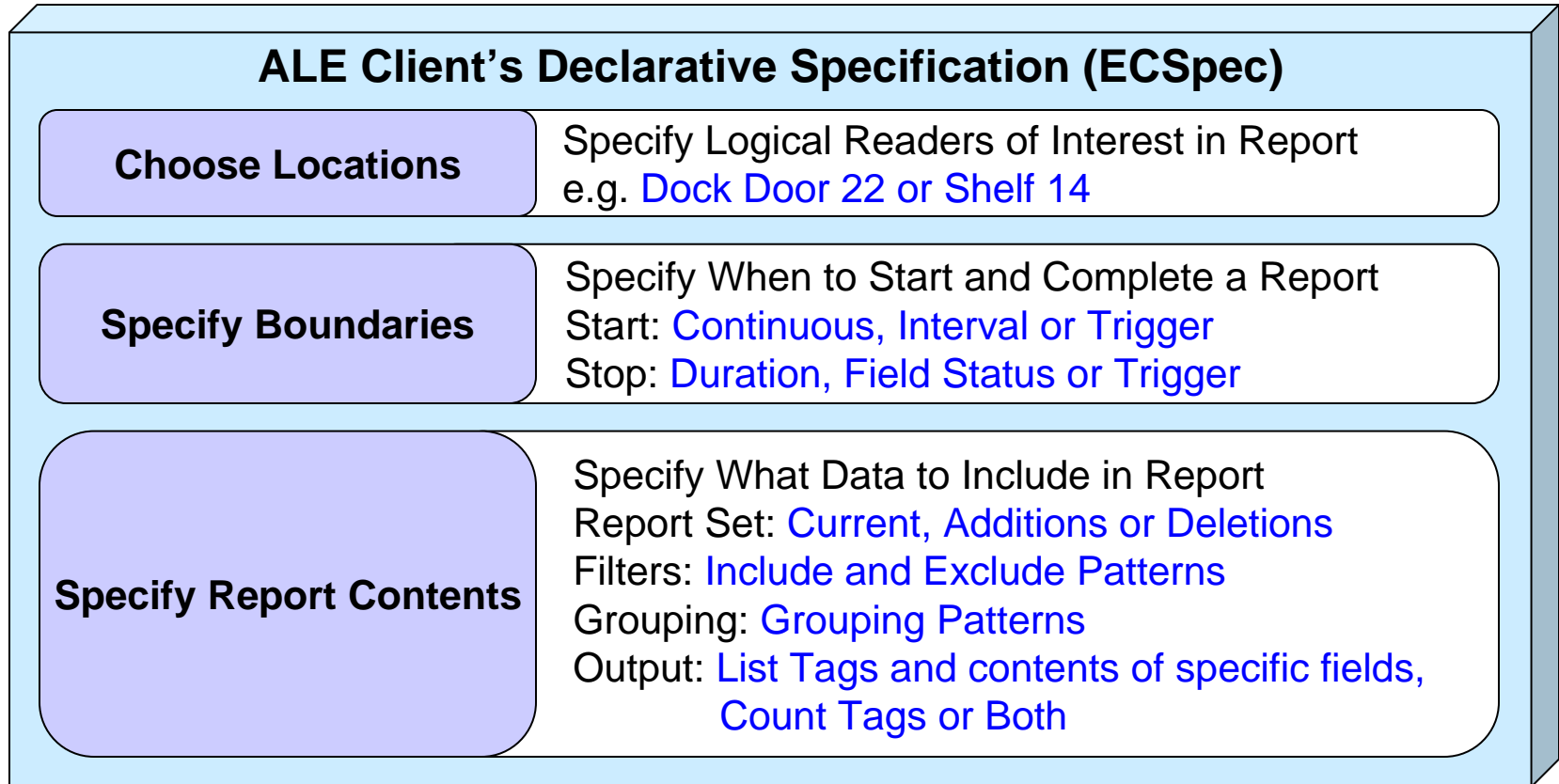
- ALE 1.0 ratified September 2005
- 15 products certified as of June 2007 (many more on market)
- ALE 1.1 ratified February 2008
 - full support for UHF Class 1 Gen 2 / ISO 18000-6C features: memory banks, kill/lock, etc
 - support for ISO 15962 encoding (which will eventually encompass EPCglobal Tag Data Standards for user memory)
 - new API for writing tags and doing other operations
 - new API for defining named memory fields
 - new API for configuring logical to physical reader mappings
 - security features
- ALE 1.1 intended to be suitable both as an interface to “middleware” as well as a “high level reader protocol”

- Reading API
 - Reads tags, reports in variety of ways
- Writing API
 - Initialize, read, write, lock, kill
- Tag Memory API
 - Define symbolic names for memory fields, for use by Reading & Writing APIs
- Logical Reader API
 - Define symbolic names for reader/device resources, for use by Reading & Writing APIs
- Access Control API
 - Control access by clients to other API features

Primarily used by applications
(data plane)

Primarily used for setup and administration
(control plane)

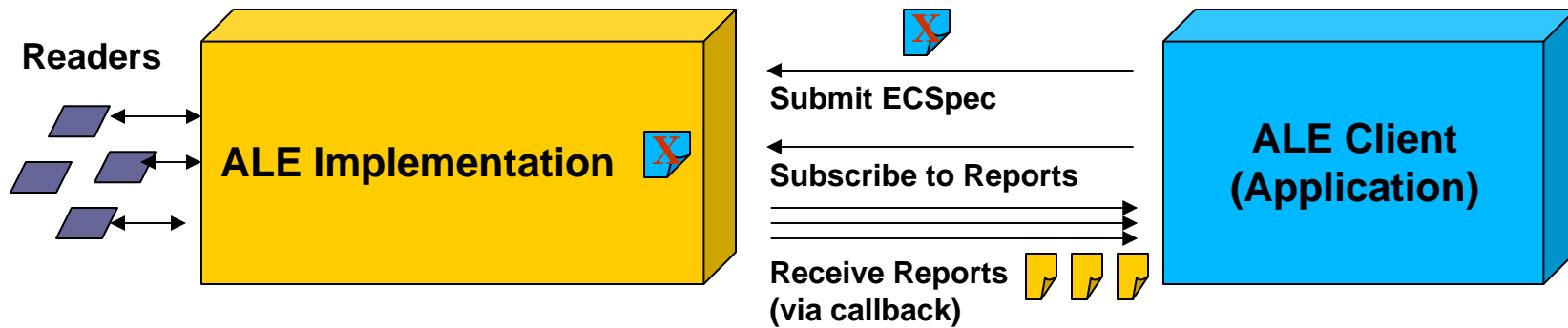
- ALE specifies an *interface*
 - ALE Client – application or other system component that wants to operate upon Tags
 - ALE Implementation – system component that implements the ALE APIs, and carries out client requests by interacting with readers or other devices (or it may be embedded in a reader itself)
 - The design of an ALE implementation is outside the scope of the spec
- ALE is *declarative*
 - ALE Client says *what* it wants done
 - ALE Implementation figures out *how* best to carry out that request
 - ALE Implementation has great freedom to push processing down to the reader or even the tag, to combine simultaneous requests, and otherwise optimize the use of resources
- ALE interface centers around “specs” and “reports”
 - Event Cycle Spec (ECSpec) = ALE Client request in Reading API
 - Event Cycle Report (ECReport) = ALE Implementation’s response
 - Command Cycle Spec / Report (CCSpec & CCReport) = corresponding things in Writing API



↓ Client presents to ALE API
in one of three ways

Subscribe Mode:

Asynchronous (“push”) reports from a standing request (ECSpec)

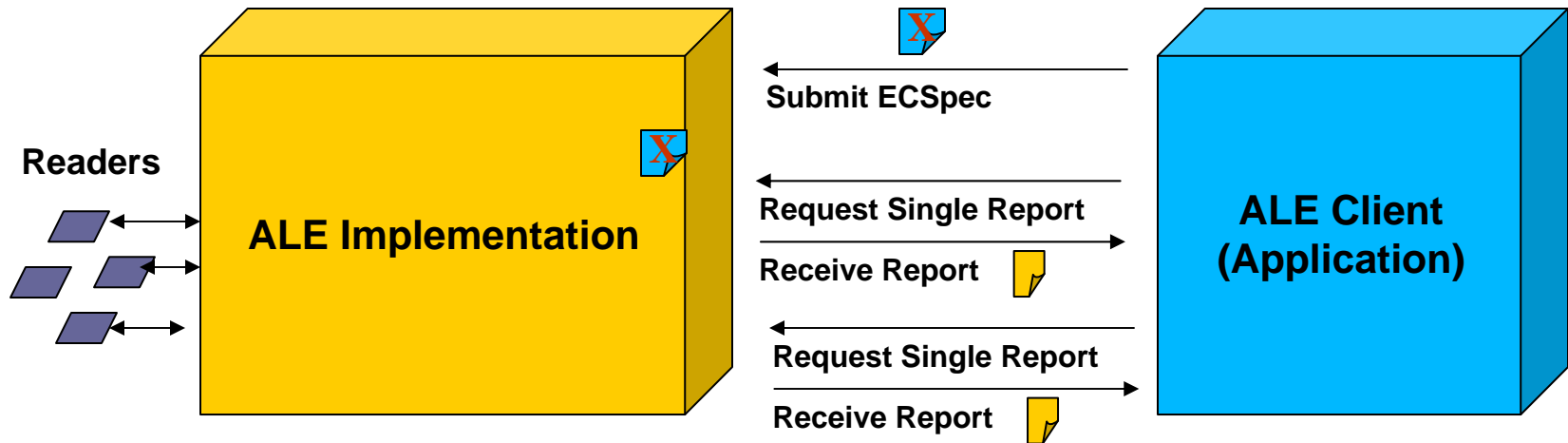


Most often used for:

- Continuous operation; or
- Triggered by time, external events

Poll Mode:

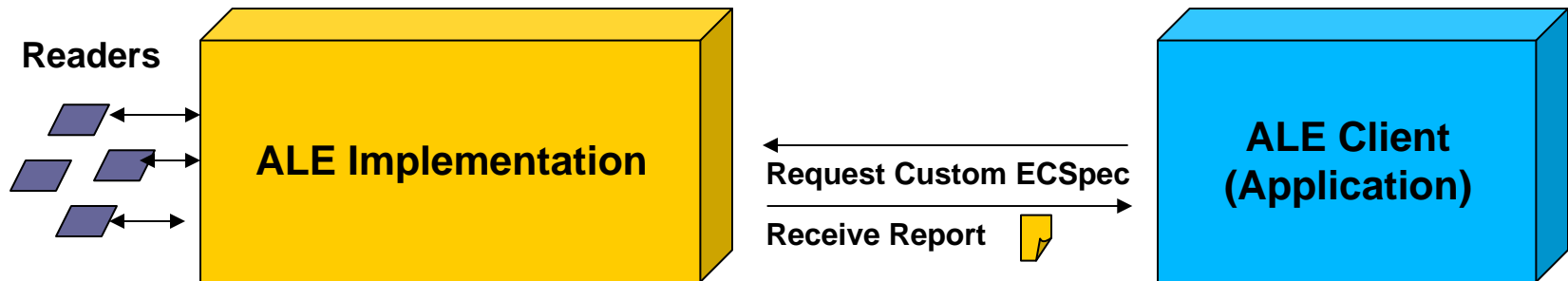
Synchronous (on-demand, “pull”) report from a standing request



Most often used for operations triggered programmatically (e.g., GUI-driven)

Immediate Mode:

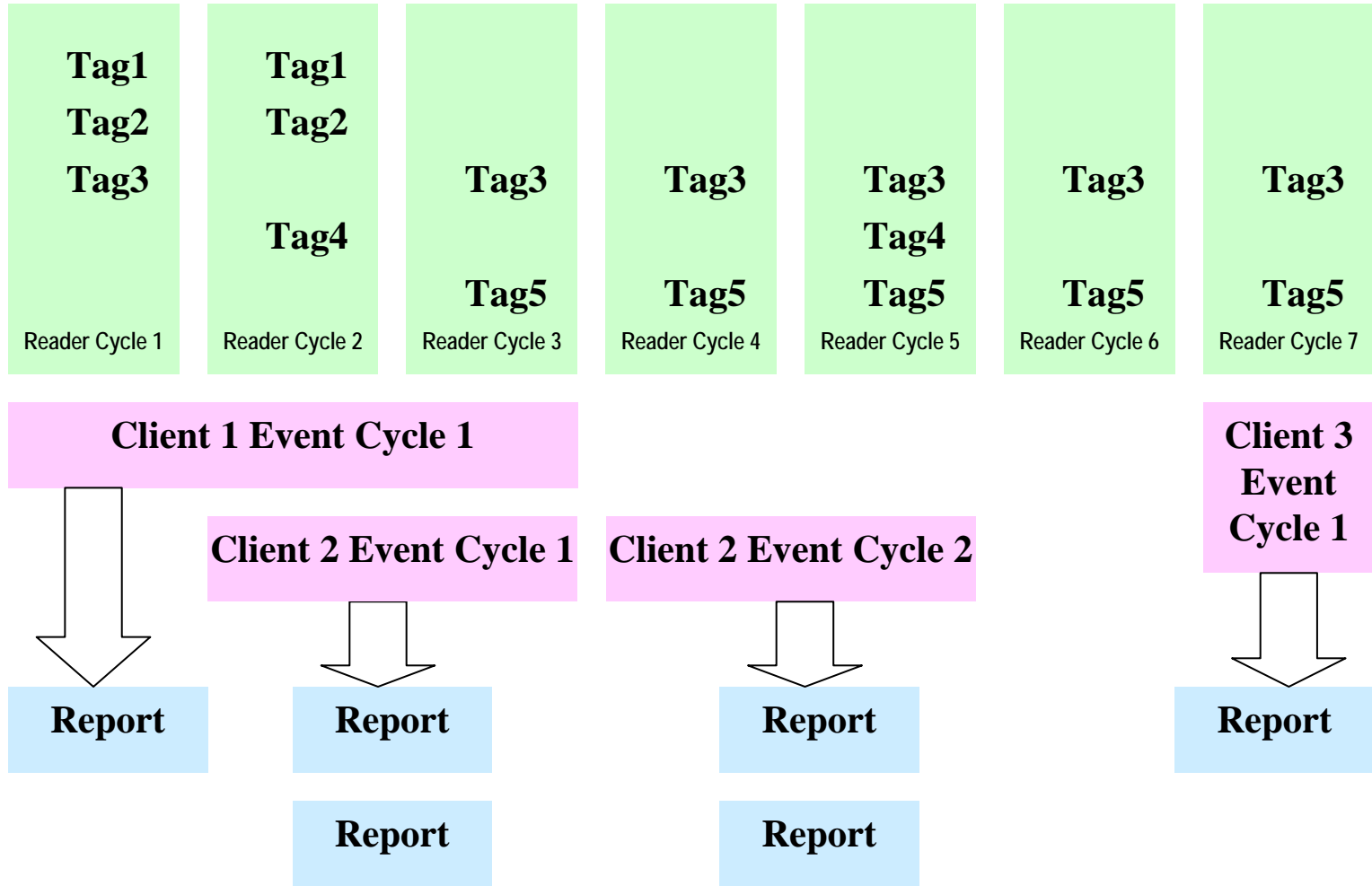
Synchronous report from one-time request



Equivalent to define + poll + undefine

- An *event cycle* is the smallest unit of interaction with an ALE Reading API client
- An ALE client describes what data it wants from an event cycle through an ECSpec (Event Cycle Specification)
- Multiple ECSpecs sharing data from underlying sources may be active simultaneously
- ALE implementation chooses best way to complete request(s)
 - May involve many interactions with underlying readers or other data sources
 - May aggressively read tags, removing any duplicates encountered
 - May delegate filtering or other processing to lower layers (e.g., to a smart reader, or to the UHF Class 1 Gen 2 / ISO 18000-6C air interface “select” command)

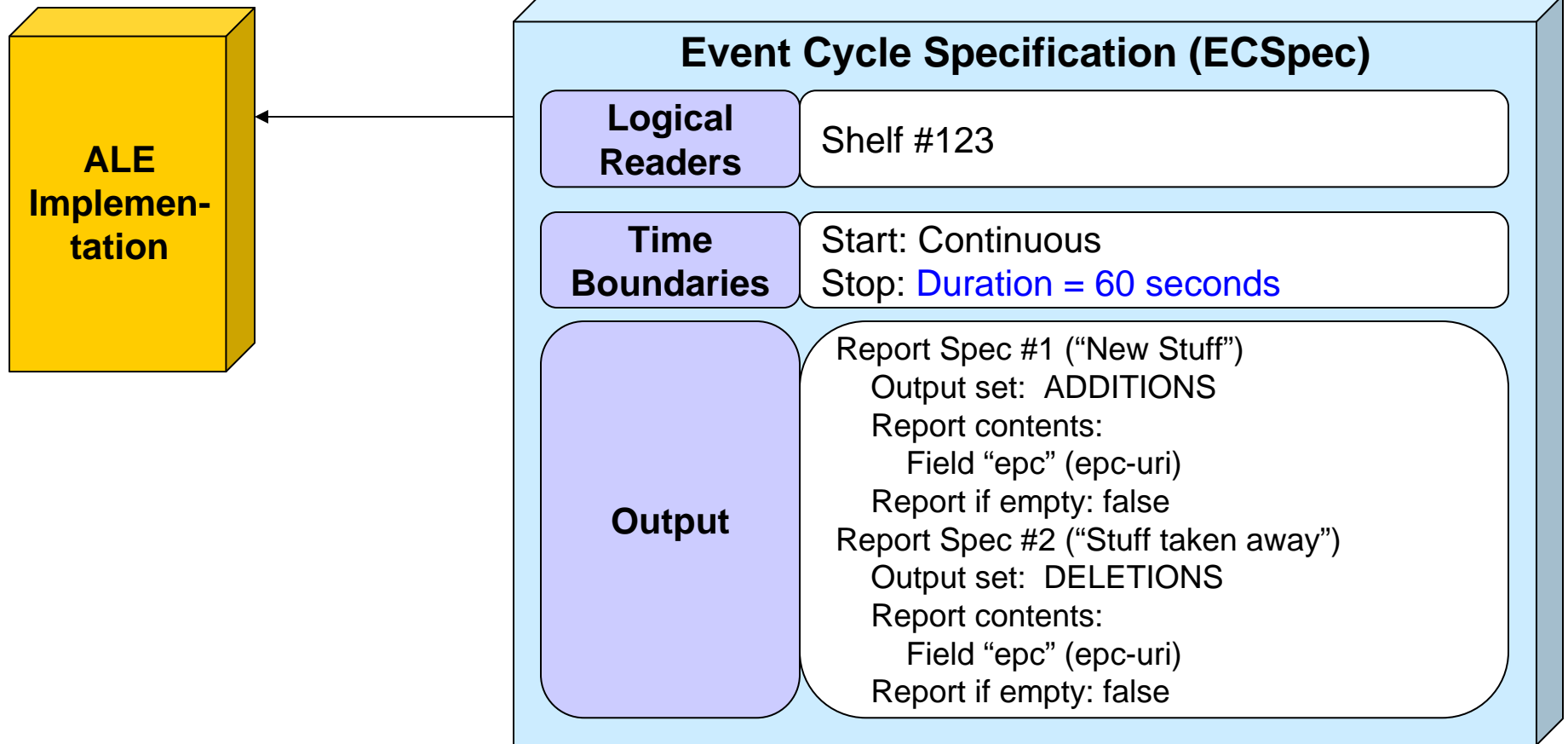
Reading API – Event Cycles



- Logical Readers
 - What readers or other devices to use to read tags
 - *Logical* names, not physical; e.g., “Dock Door #5” – ALE client doesn’t know (or care) how many readers or what make/model of readers are in use at Dock Door #5”
- Time Boundaries
 - Start condition: continuous, periodic, external trigger
 - Stop condition: duration, trigger, “stable set interval” (no new tags within a time interval), “when data available” (as soon as new tags are seen)
- Output (report) Specification(s)
 - What filters to apply – include or exclude tags based on field contents
 - Output set: CURRENT, ADDITIONS, DELETIONS
 - What fields to read & report (or, just count tags)
 - Data format to use for reporting
 - Report if empty (true/false)
 - Grouping (optional)

Example #1

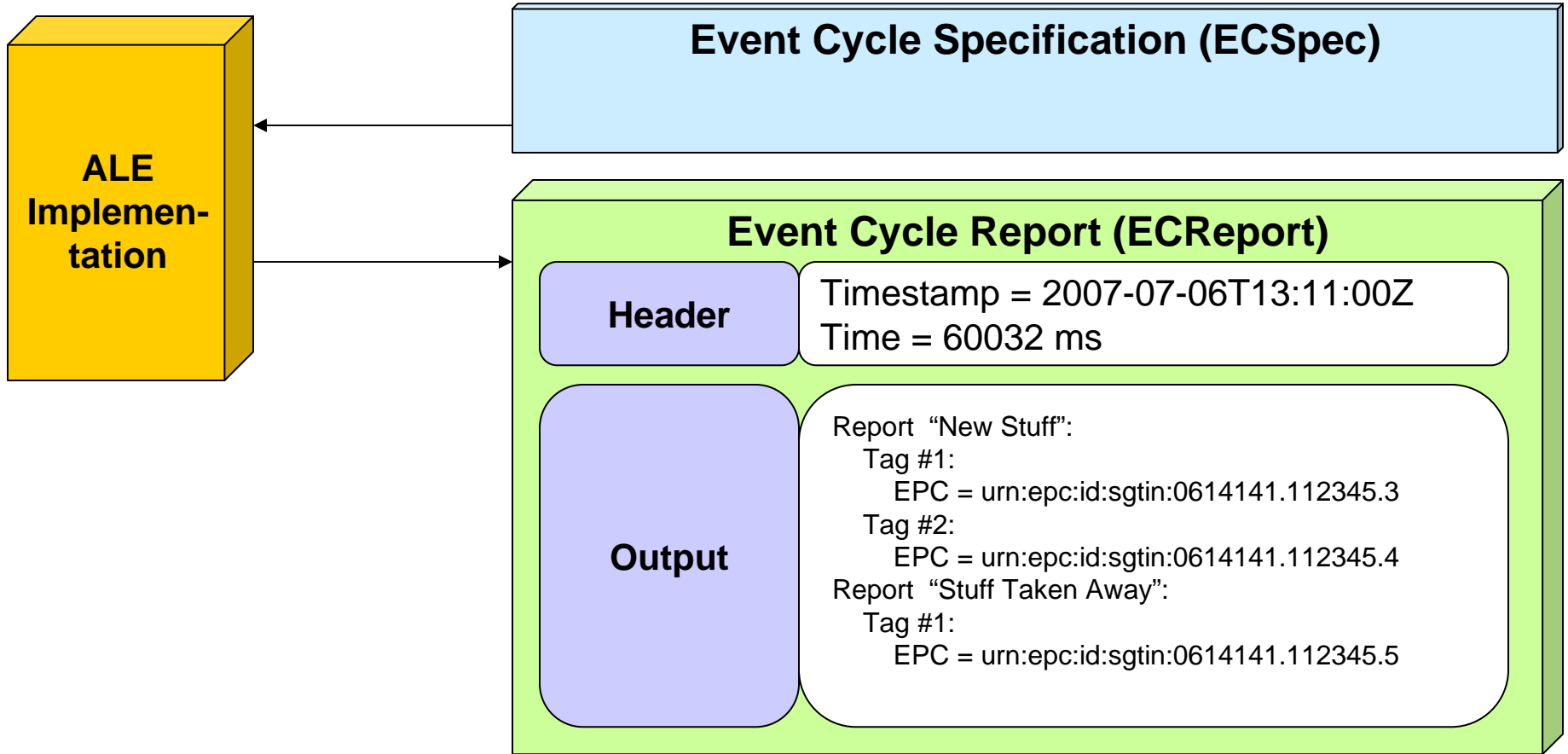
Smart Shelf: “Report once per minute about things added and removed from Shelf #123”



“Report once per minute about things added and removed from Shelf #123”

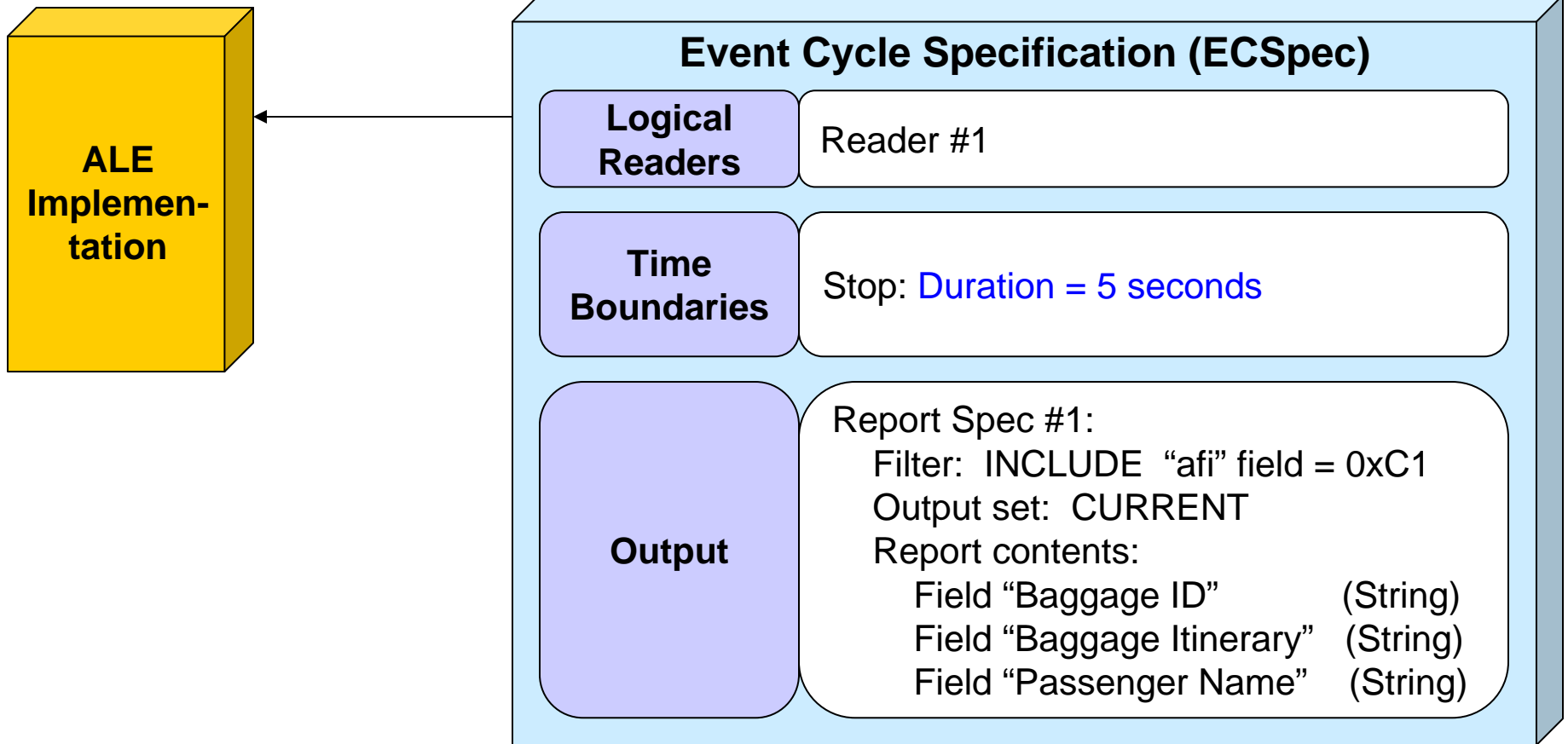
```
<ale:ECSpec ...>
  <logicalReaders>
    <logicalReader>shelf123</logicalReader>
  </logicalReaders>
  <boundaries>
    <duration unit="MS">60000</duration>
  </boundaries>
  <reportSpecs>
    <reportSpec name="New Stuff">
      <reportSet set="ADDITIONS"/>
      <output includeEPC="true"/>
    </reportSpec>
    <reportSpec name="Stuff Taken Away">
      <reportSet set="DELETIONS"/>
      <output includeEPC="true"/>
    </reportSpec>
  </reportSpecs>
</ale:ECSpec>
```

Example #1 (cont'd)



```
<ale:ECReports ...>
  <reports>
    <report name="New Stuff">
      <group>
        <groupList>
          <member>
            <epc>urn:epc:id:sgtin:0614141.112345.3</epc>
          </member>
          <member>
            <epc>urn:epc:id:sgtin:0614141.112345.4</epc>
          </member>
        </groupList>
      </group>
    </report>
    <report name="Stuff Taken Away">
      <group>
        <groupList>
          <member>
            <epc>urn:epc:id:sgtin:0614141.112345.5</epc>
          </member>
        </groupList>
      </group>
    </report>
  </ale:ECReports>
```

“Read the Bag ID, Itinerary, and Passenger Name from all the IATA Baggage Tags at Reader #1”



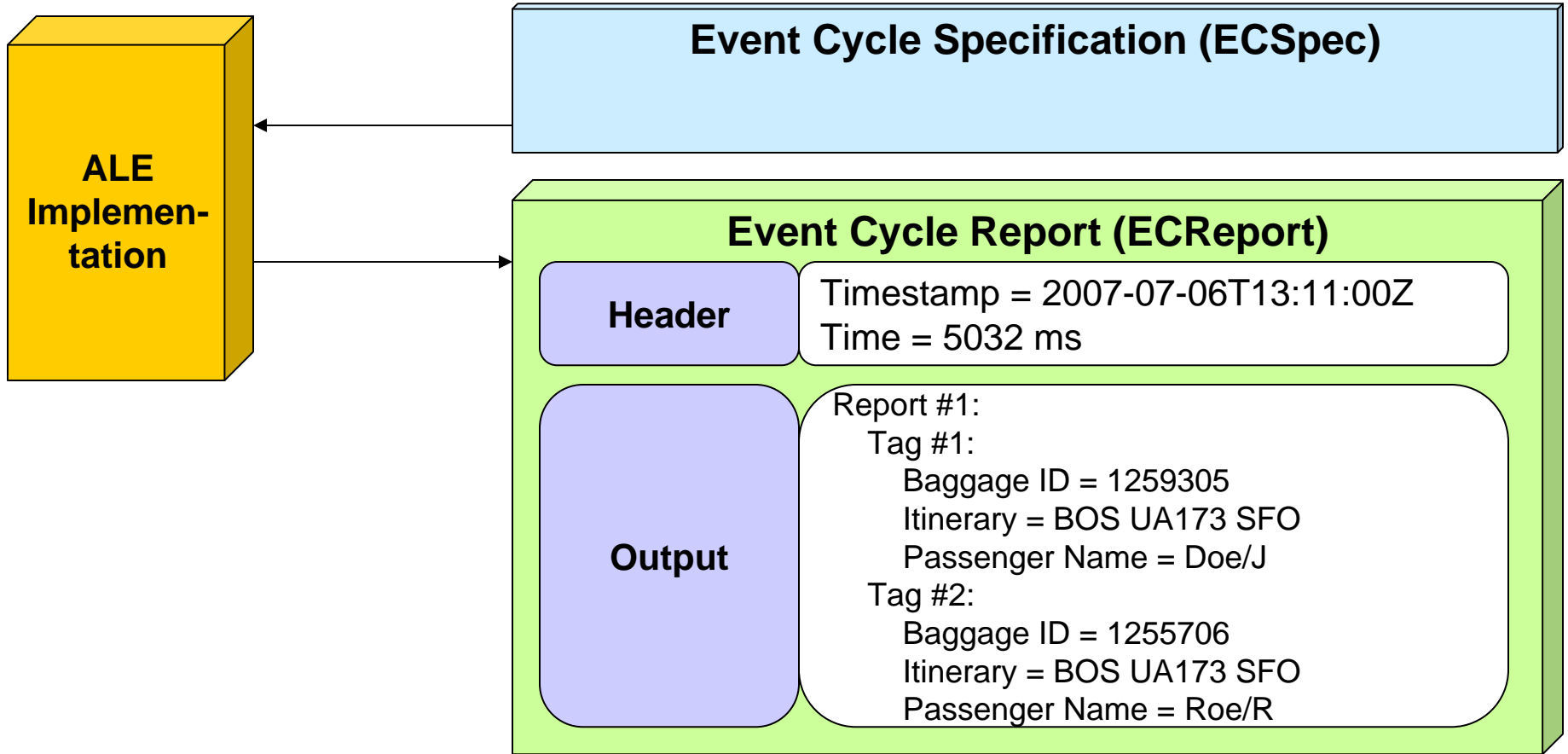
“Read the Bag ID, Itinerary, and Passenger Name from all the IATA Baggage Tags at Reader #1”

```

<ale:ECSpec ...>
  <logicalReaders>...</logicalReaders>
  <boundaries>...</boundaries>
  <reportSpecs>
    <reportSpec name="report1">
      <reportSet set="CURRENT"/>
      <filterSpec>...
        <filter>
          <includeExclude>INCLUDE</includeExclude>
          <fieldspec><fieldname>afi</fieldname></fieldspec>
          <patList><pat>xCl</pat></patList>
        </filter>...
      </filterSpec>
      <output>...
        <fieldList>
          <field><fieldspec><name>@3.urn:oid:1.0.15961.12.3</name></fieldspec></field>
          <field><fieldspec><name>@3.urn:oid:1.0.15961.12.4.*</name></fieldspec></field>
          <field><fieldspec><name>@3.urn:oid:1.0.15961.12.11</name></fieldspec></field>
        </fieldList>
      ...</output>
    ...</reportSpec>
  ...</ale:ECSpec>

```

Example #2 (cont'd)



```
<ale:ECReports ...>
  <reports>
    <report name="report1">
      <group>
        <groupList>
          <member>
            <tag>urn:epc:raw:96.x01234567890123467890123</tag>
            <fieldList>
              <field>
                <name>@3.urn:oid:1.0.15961.12.3</name>
                <value> 1259305 </value>
              </field>
              <field>
                <name>@3.urn:oid:1.0.15961.12.4.1</name>
                <value>BOS UA173 SFO</value>
              </field>
              <field>
                <name>@3.urn:oid:1.0.15961.12.4.2</name>
                <value>SFO UA425 PSP</value>
              </field>
              <field>
                <name>@3.urn:oid:1.0.15961.12.11</name>
                <value>Doe/J</value>
              </field>
            </fieldList>
          </member>
          <member>
            <tag>urn:epc:raw:96.x01234567890123467890123</tag>
            <fieldList>
              ...</member>
            ...</ale:ECReports>
```

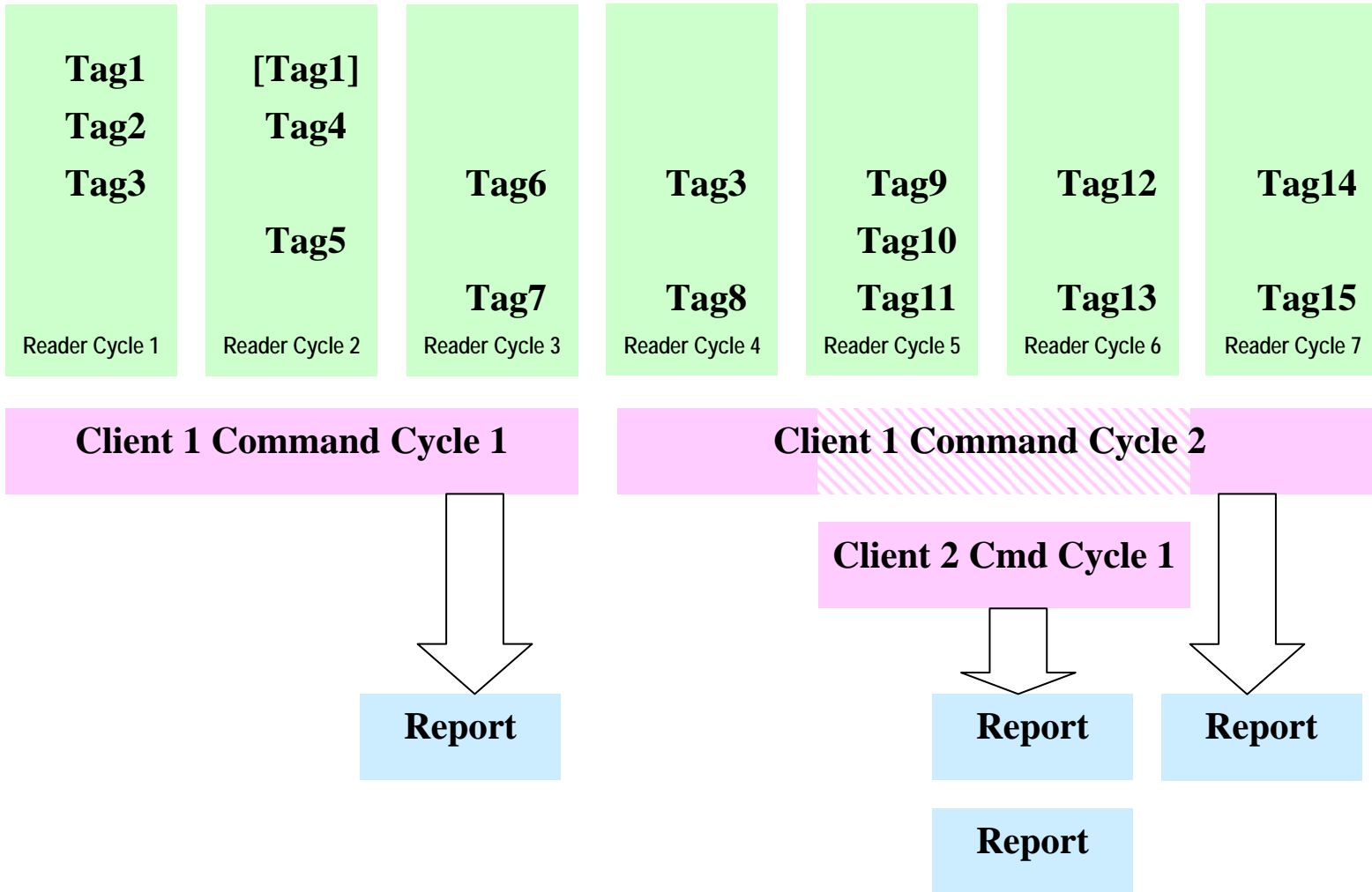
“Read the Bag ID, Itinerary, and Passenger Name from all the IATA Baggage Tags at Reader #1”

```

<ale:ECSpec ...>
  <logicalReaders>...</logicalReaders>
  <boundaries>...</boundaries>
  <reportSpecs>
    <reportSpec name="report1">
      <reportSet set="CURRENT"/>
      <filterSpec>...
        <filter>
          <includeExclude>INCLUDE</includeExclude>
          <fieldspec><fieldname>afi</fieldname></fieldspec>
          <patList><pat>xCl</pat></patList>
        </filter>...
      </filterSpec>
      <output>...
        <fieldList>
          <field><fieldspec><name>BaggageID</name></fieldspec></field>
          <field><fieldspec><name>Itinerary</name></fieldspec></field>
          <field><fieldspec><name>PassengerName</name></fieldspec></field>
        </fieldList>
      ...</output>
    ...</ale:ECSpec>
  
```

- A *command cycle* is the smallest unit of interaction with an ALE Writing API client
- An ALE client describes what operations to perform on tags during a command cycle through a CCSpec (Command Cycle Specification)
- Each active CCSpec gets exclusive access – pre-emption possible
- ALE implementation chooses best way to complete request(s)
 - May involve many interactions with underlying readers or other data sources
 - Should employ air interface mechanisms to insure each tag acted upon only once
 - May delegate filtering or other processing to lower layers (e.g., to a smart reader, or to the UHF Class 1 Gen 2 / ISO 18000-6C air interface “select” command)

Writing API – Command Cycles



- Logical Readers
 - What readers or other devices to use to read tags
 - *Logical* names, not physical; e.g., “Dock Door #5” – ALE client doesn’t know (or care) how many readers or what make/model of readers are in use at Dock Door #5”
- Time Boundaries
 - Start condition: continuous, periodic, external trigger
 - Stop condition: duration, trigger, tag count, error
- Command Specification(s)
 - What filters to apply – include or exclude tags based on field contents
 - List of operations to be performed, each specifying:
 - Operation type: initialize, read, write, add, delete, check, lock, kill, password
 - Bank or field to operate upon
 - Data input (if applicable), output format (if applicable)
 - Report if empty (true/false)

Op Type	Description	Field spec	Data spec
INITIALIZE	Prepare a bank for using variable fields	Bank	Init info (e.g., DSFID)
READ	Read contents of a field	Fieldname	[none]
WRITE	Write contents of a field	Fieldname	Value to write
ADD	Add new field (for fixed field, same as WRITE)	Fieldname	Value to write
DELETE	Remove field (for fixed field, same as WRITE zero)	Fieldname	[none]
LOCK	Change access permissions	Bank (or field, if tag permits)	Permission controls
KILL	Kill tag	[none]	Kill password
PASSWORD	Provide password to enable subsequent ops	[none]	Access password
CHECK	Check memory state consistency	Bank	Format code

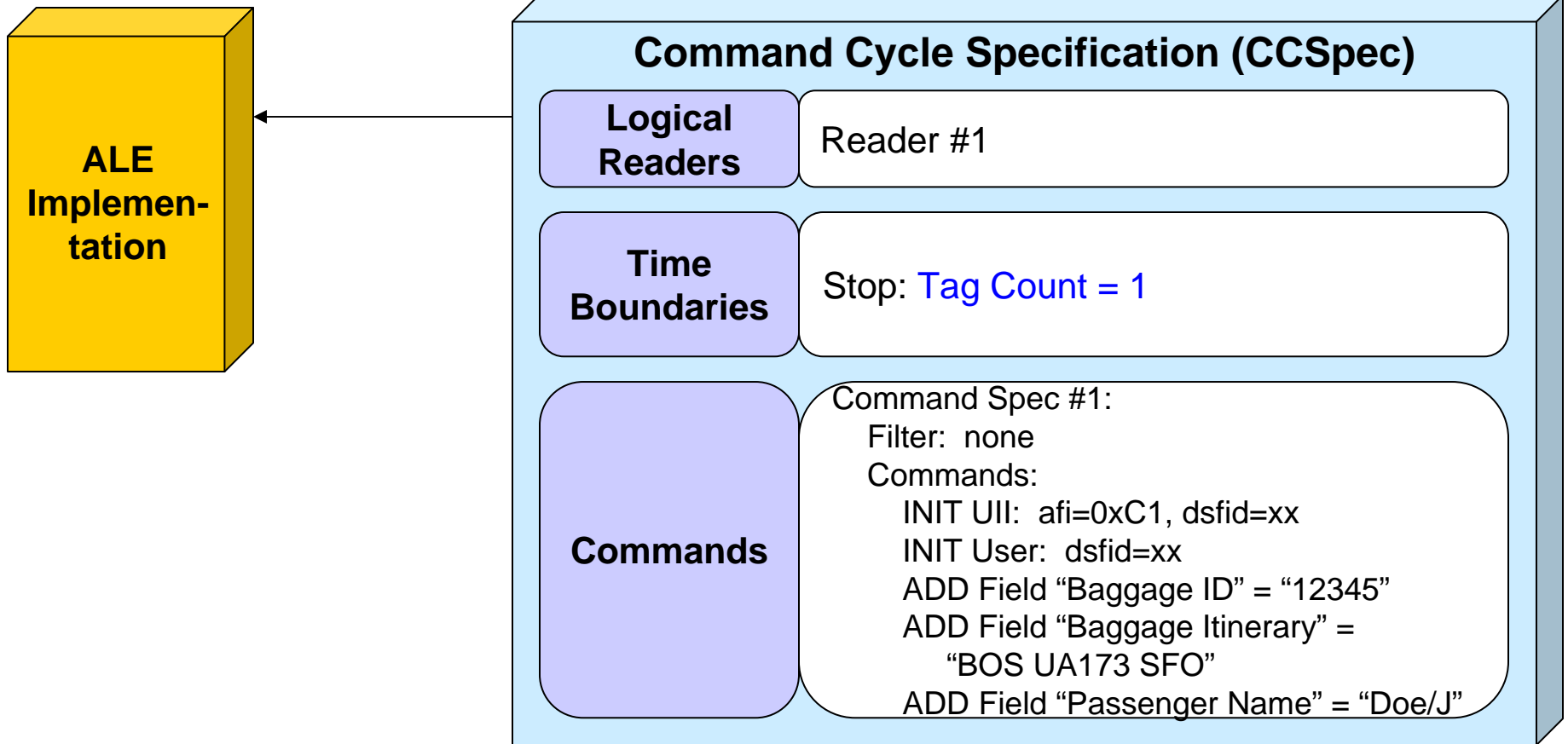
For an ADD or WRITE command, data can be specified in several ways:

- Literal: specified directly in the CCSpec
- Parameter: client provides a different value on each poll() operation
- EPC cache: a new value taken from a previously-defined list of available EPCs
- Association table: looked up in a table, indexed by EPC
- Random # generator: randomly-generated value

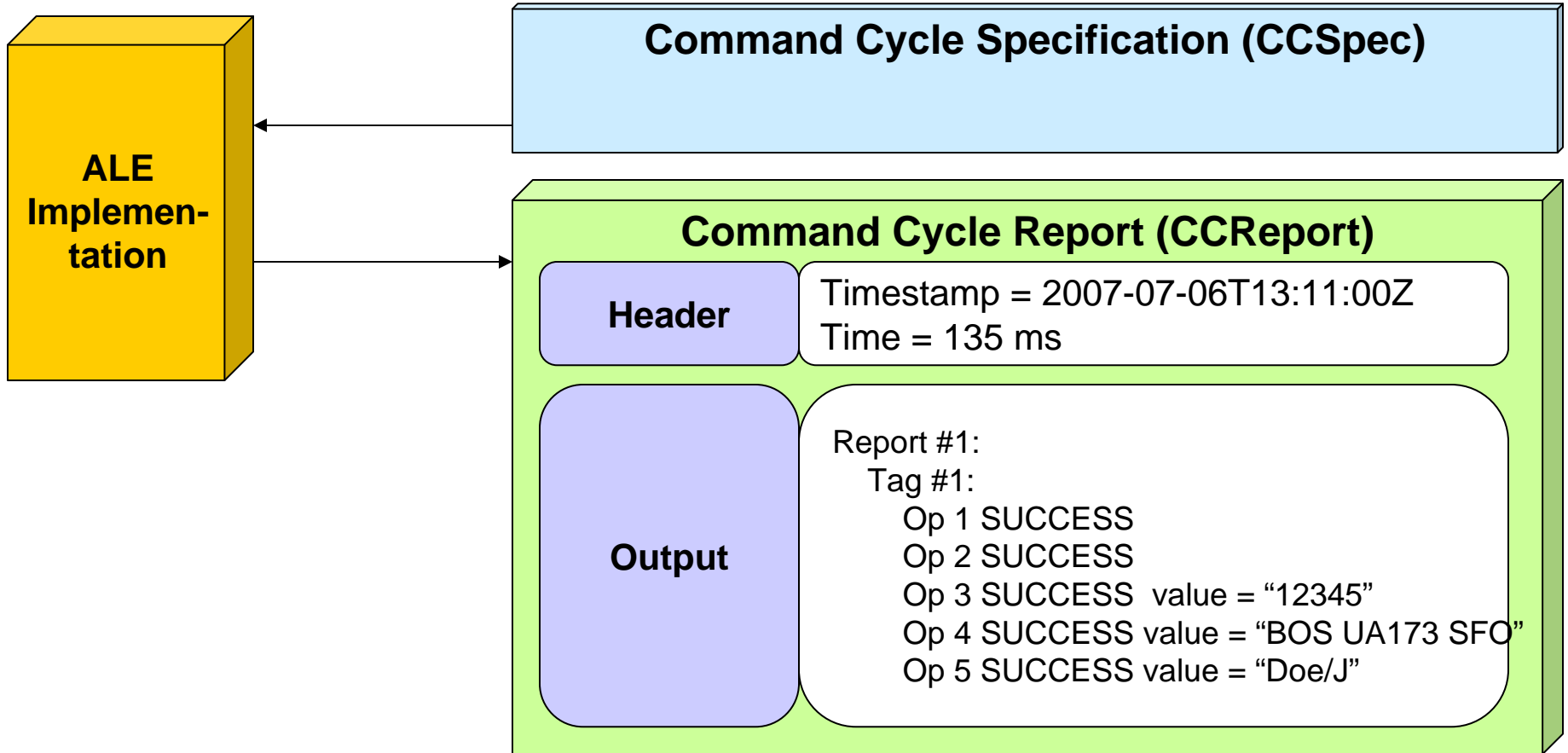
Use cases:

- Commission new EPC in all 12 item-level tags when a case passes a reader on a conveyor
- Look up kill passwords and kill all 20 tags in field at a POS terminal
- Generate new kill passwords when 12 tags are commissioned

“Write John Doe’s Baggage Tag at Reader #1”



Example #3 (cont'd)



- ECSpec/CCSpec refer to a field by name; E.g., “baggageID”, “passengerName”, “epc”, ...
- Fieldname tells ALE where to find the data.
 - Could be fixed address (bank/offset) or variable location (e.g., ISO 15962 data set)
 - Could be different depending on tag type (e.g., on a Gen1 tag vs a Gen2 tag)
- ECSpec/CCSpec may also specify
 - Datatype – how to interpret the bits (e.g., as an unsigned integer)
 - Format – how to present the data across the ALE interface (e.g., hex or decimal)
 - Either or both may be omitted: each fieldname has defaults for both
- Extensible
 - ALE API specifies certain built-in names (next slide)
 - ALE implementation may provide other built-in fieldnames
 - Clients may define fieldnames using Tag Memory API
- ALE implementation considers totality of fields required by ECSpec/CCSpec, and accesses the tags accordingly.

Built-in fieldnames:

- “epc”
- Fixed fieldnames of the form *@bank.length[.offset]*
- ISO 15962 variable fieldnames of the form *@bank.oid*

Built-in datatypes:

- “epc”
- “unsigned integer”
- “iso 15962-encoded string”

Built-in formats:

- For the “epc” type: epc, epc-tag, epc-hex, epc-decimal
- For the “unsigned integer” type: hex
- For the “iso 15962-encoded string” type: string

Vendor extensions and extensions in future ALE versions likely

- Built-in fieldnames:
 - epcBank, tidBank, userBank
 - accessPwd, killPwd
 - afi, nsi
 - epc
- Absolute Address fieldnames:
 - @*bank.length[.offset]*
 - E.g. @1.8.16 – 8-bit field starting at bit 10h in Bank 1
- ISO 15962 dataset fieldnames:
 - @*bank.oid*
 - E.g. @3.urn:oid:1.0.15961.12.11
- Symbolic names
 - Defined by client using the Tag Memory API
 - Alias for Absolute Address or ISO 15962 dataset fieldname

Fixed fields:

- Always at same address in tag memory
- Always there (add/delete not relevant)
- No INIT command needed

Variable fields:

- Address varies
- Not always there (add/delete useful)
- INIT command needed

Fixed or variable

“Read the Bag ID, Itinerary, and Passenger Name from all the IATA Baggage Tags at Reader #1”

```

<ale:ECSpec ...>
  <logicalReaders>...</logicalReaders>
  <boundaries>...</boundaries>
  <reportSpecs>
    <reportSpec name="report1">
      <reportSet set="CURRENT"/>
      <filterSpec>...
        <filter>
          <includeExclude>INCLUDE</includeExclude>
          <fieldspec><fieldname>afi</fieldname></fieldspec>
          <patList><pat>xCl</pat></patList>
        </filter>...
      </filterSpec>
      <output>...
        <fieldList>
          <field><fieldspec><name>BaggageID</name></fieldspec></field>
          <field><fieldspec><name>Itinerary</name></fieldspec></field>
          <field><fieldspec><name>PassengerName</name></fieldspec></field>
        </fieldList>
      ...</output>
    ...</ale:ECSpec>
  
```

- All five request/response APIs
 - SOAP/HTTP (XML over HTTP)
- Reading and Writing Callback APIs
 - XML over raw TCP
 - XML over HTTP
 - XML over HTTPS (new for ALE 1.1)



The global language of business

www.gs1.org