



## Application Level Events (ALE)

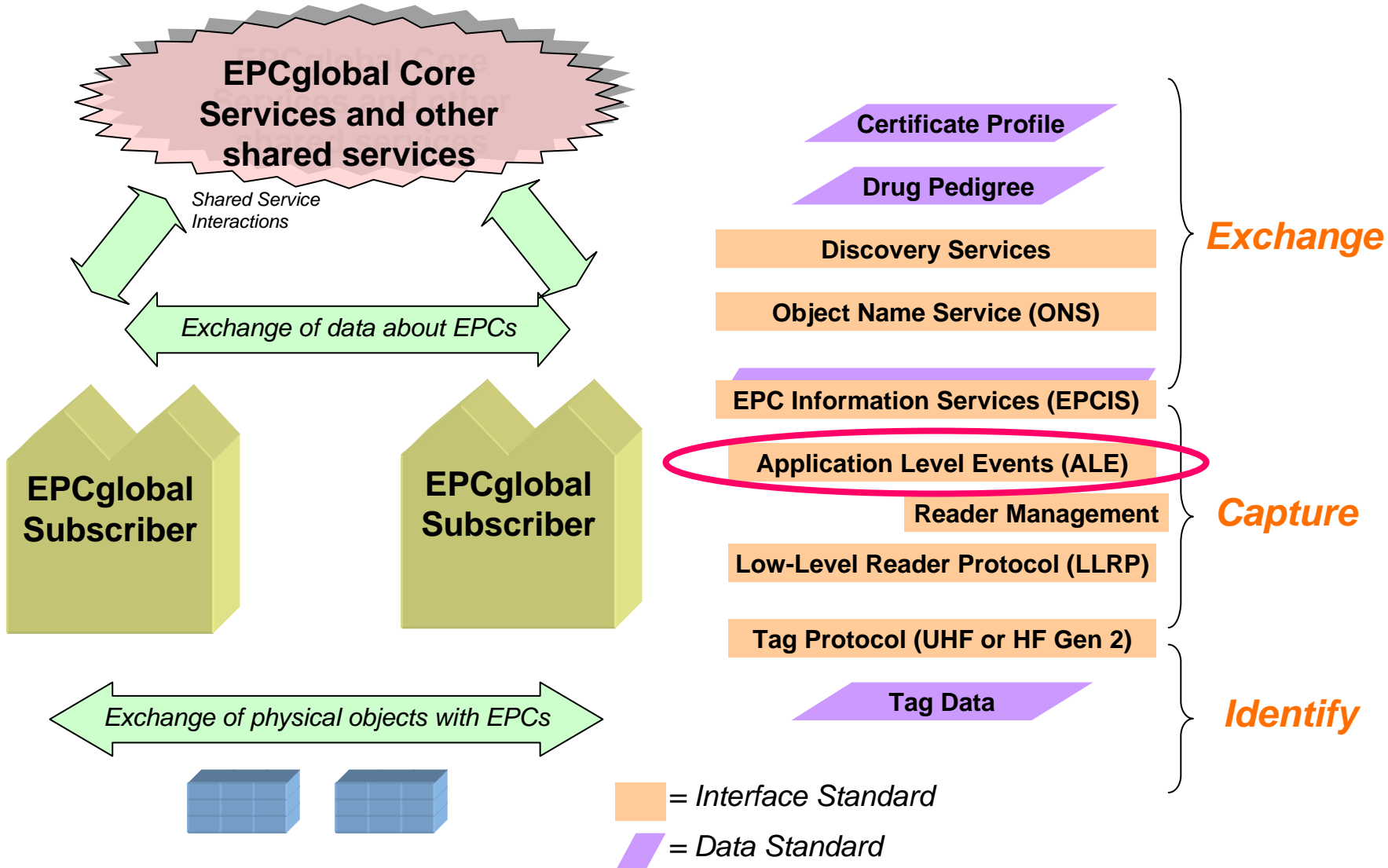
*Filtering & Collection WG*

*18 June 2007*

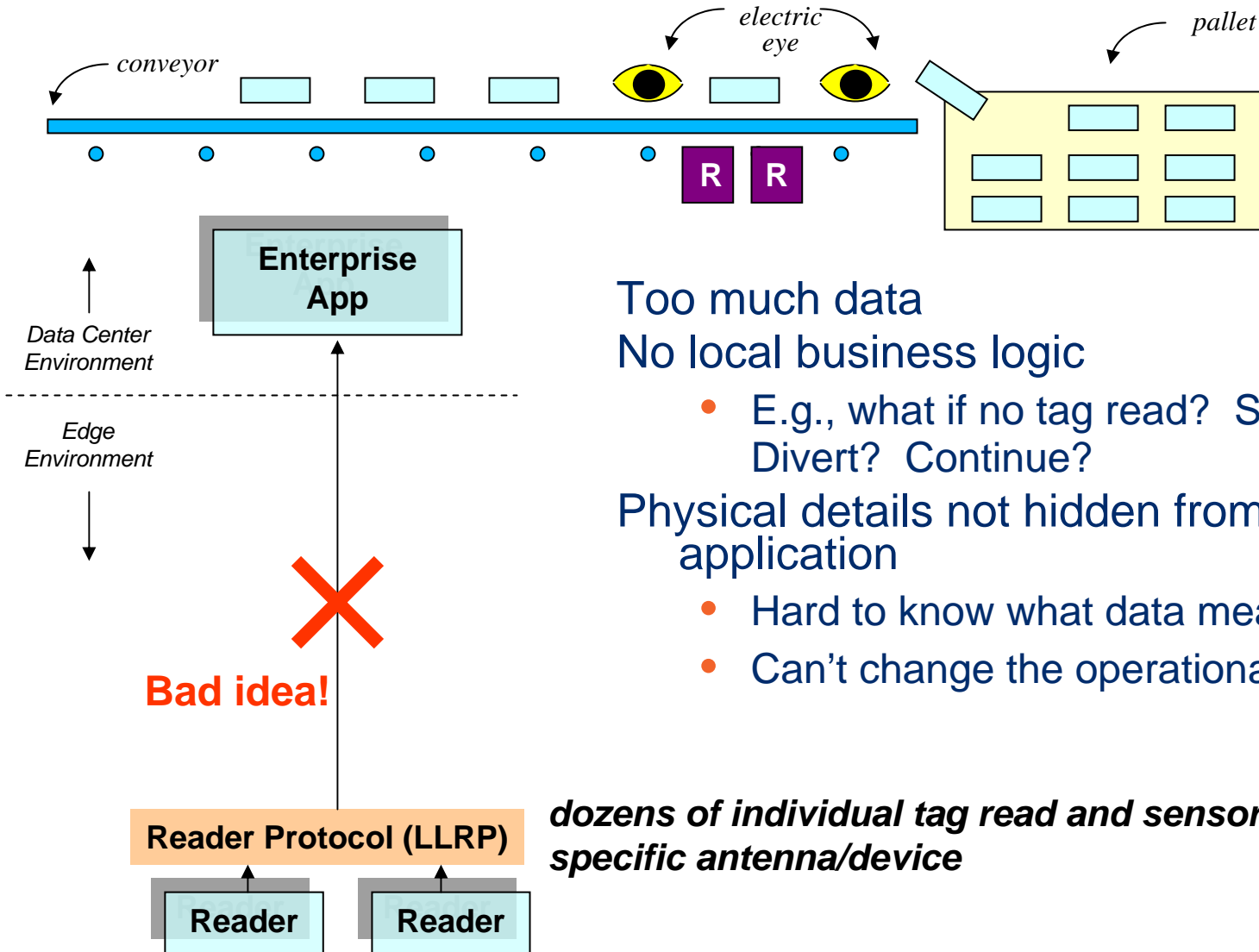
**Identify** individual products, cases, loads, assets, etc, so that they can be tracked individually

**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



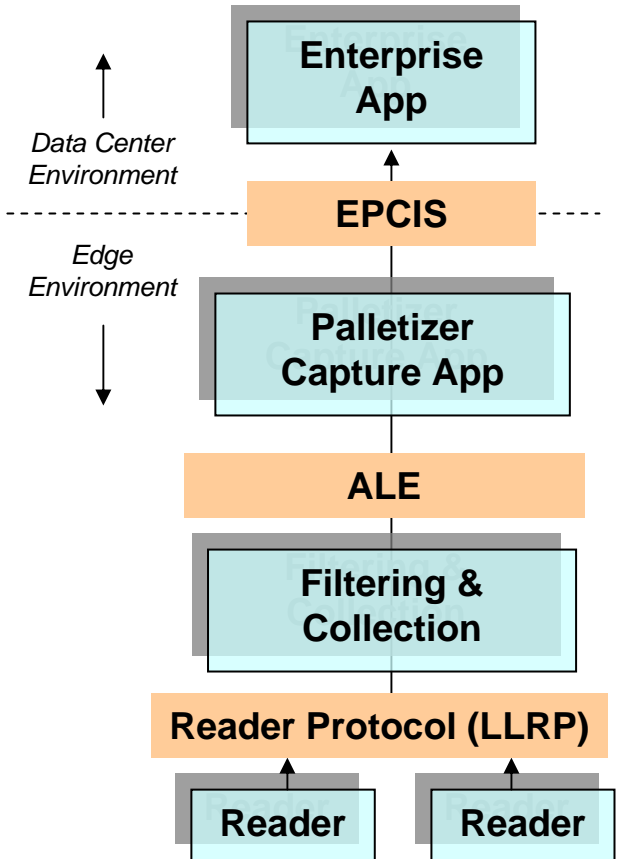
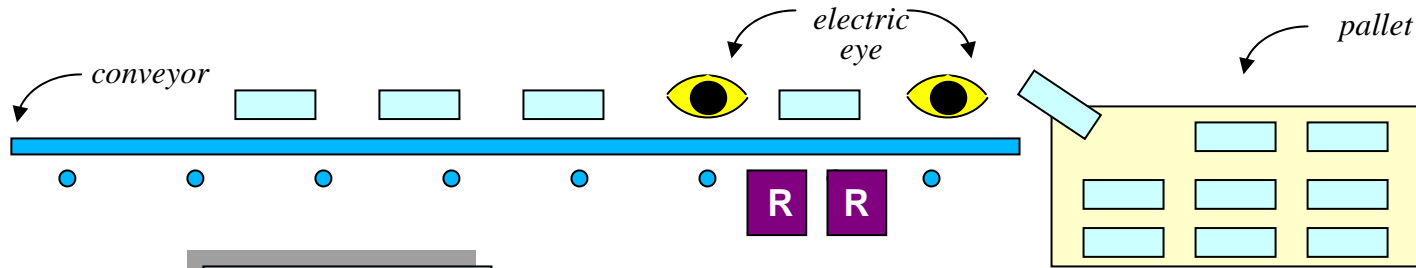
# Data Capture Example – Palletizer



- 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



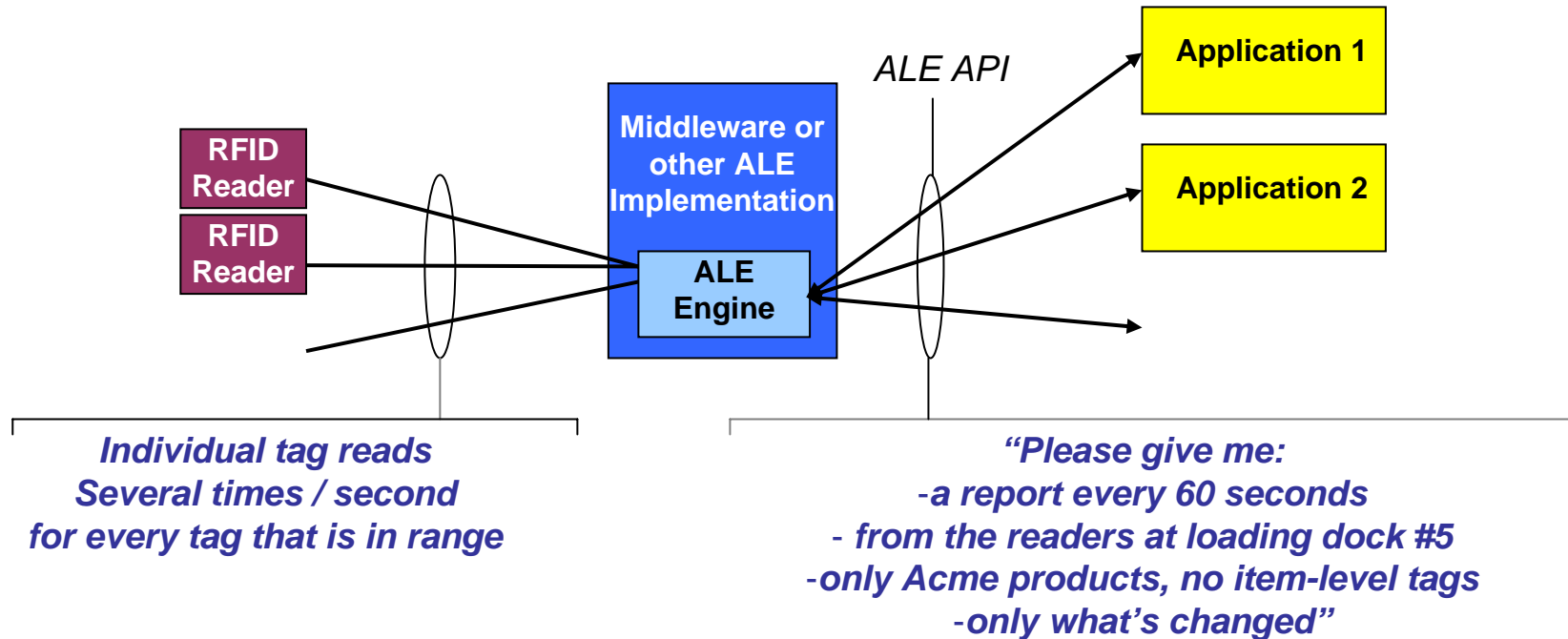
***“at time T, the association of the following case tags to the following pallet tag was created at palletizer #3, to fulfill order #1234”***

**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*

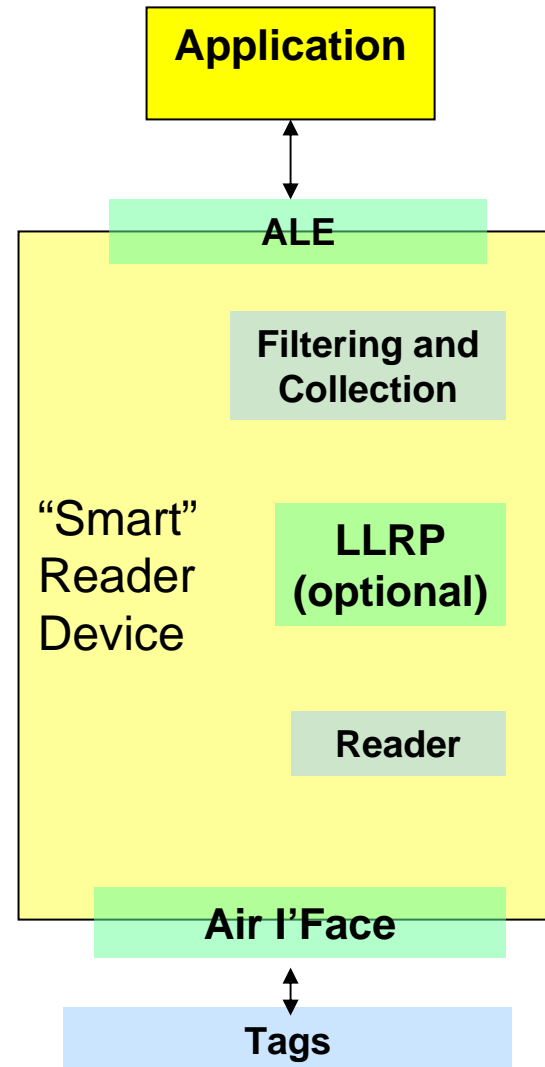
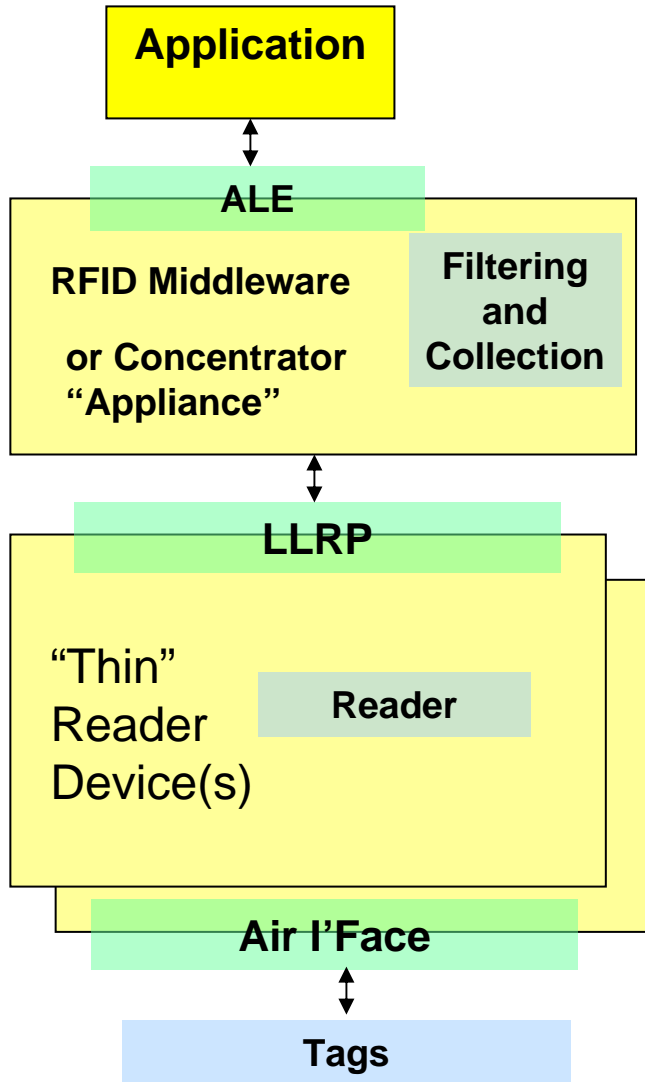
*Insulates applications from device details*

*Shares data among multiple applications*

*Extensible to vendor changes*

*Integrates easily using standard XML / Web Services technology*

# ALE: an Interface, not a software component



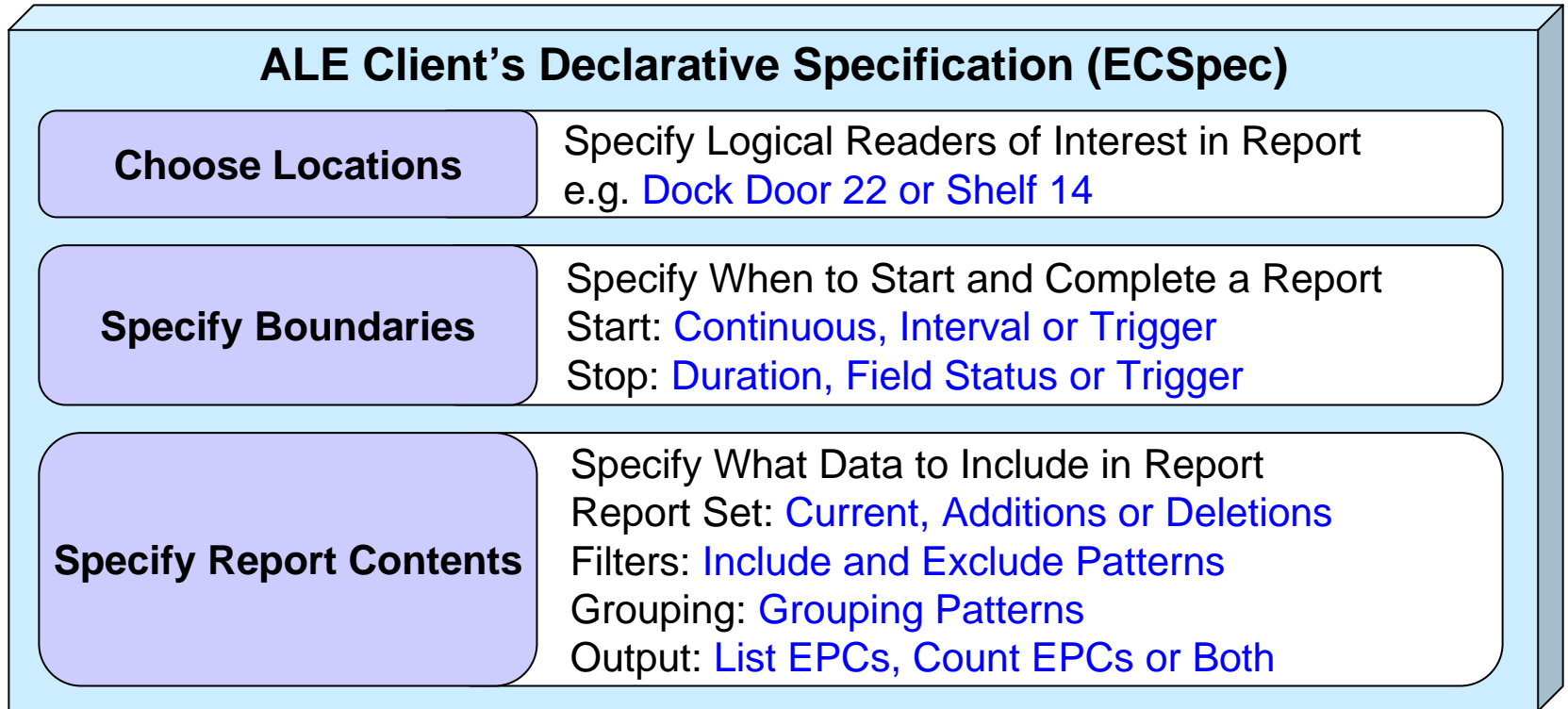
ALE 1.0 ratified September 2005

15 products certified as of June 2007

ALE 1.1 underway (Last Call Working Draft 18 May 2007)

- full support for Gen2 features: memory banks, kill/lock, etc
- new API for writing tags
- new API for defining named memory fields
- new API for configuring logical to physical reader mappings
- security features
- binary binding

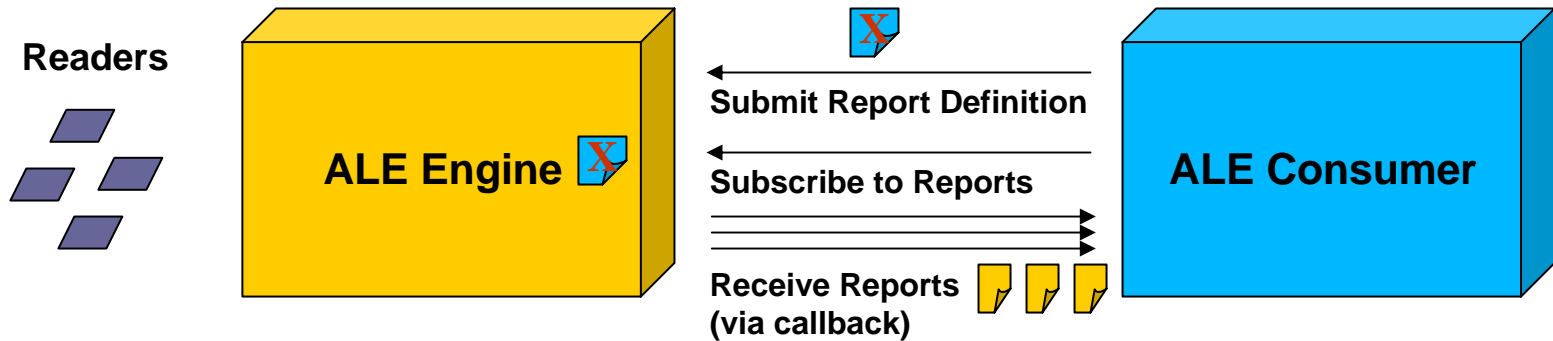
ALE 1.1 intended to be suitable both as an interface to “middleware” as well as a “high level reader protocol”



Client presents to ALE API  
in one of three ways

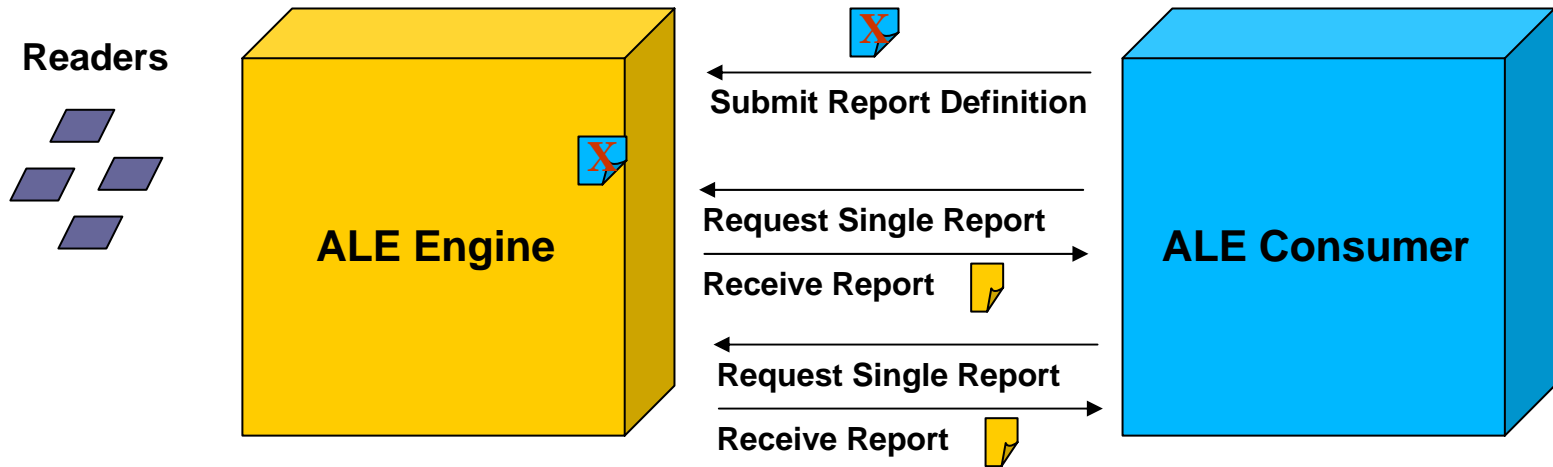
## Subscribe Mode:

Asynchronous reports from a standing request (ECSpec)

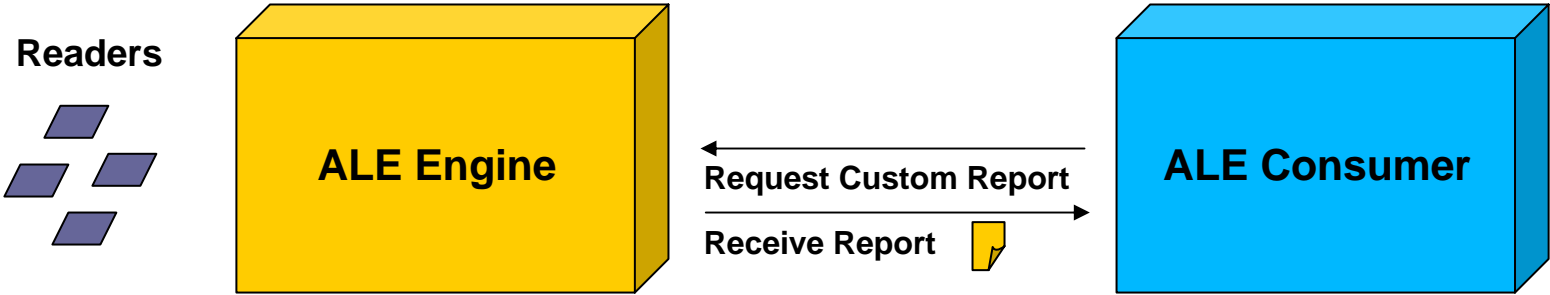


## Poll Mode:

Synchronous (on-demand) report from a standing request

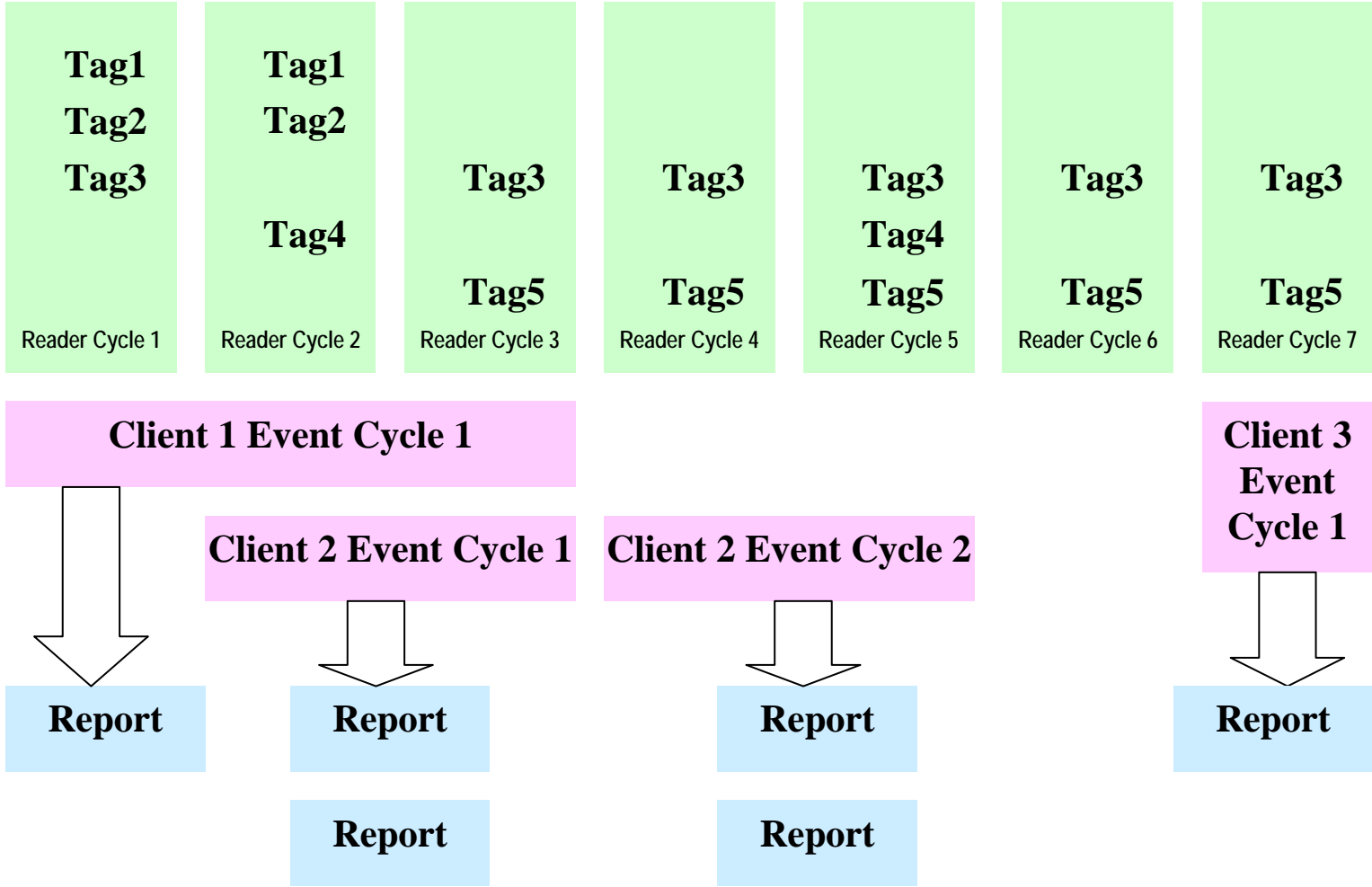


## Immediate Mode: Synchronous report from one-time request



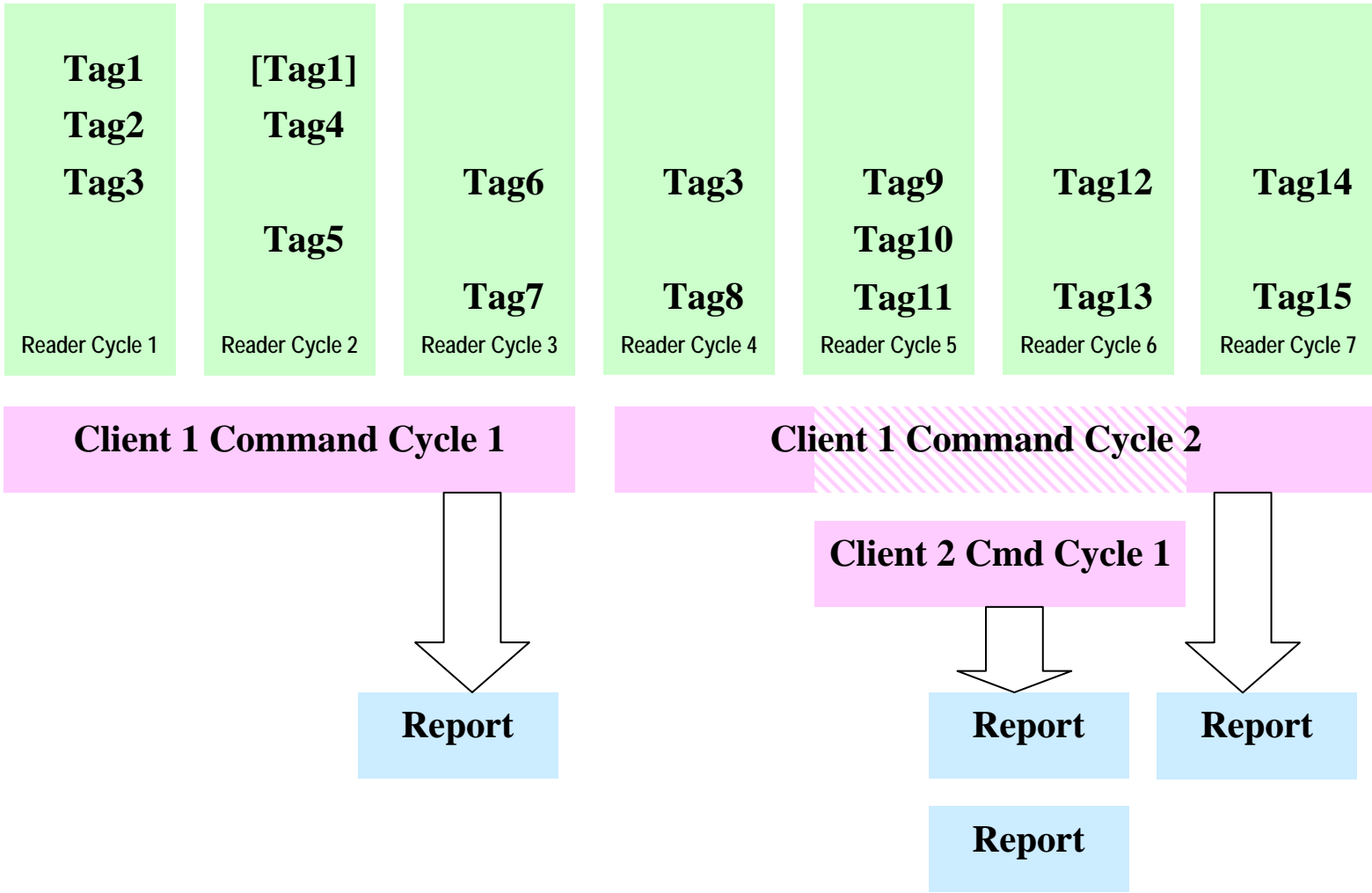
- An *event cycle* is the smallest unit of interaction with an ALE Reading API client
- An ALE client describes what data it wants from an event cycle through an ECSpec (Event Cycle Specification)
- Multiple ECSpecs sharing data from underlying sources may be active simultaneously
- ALE implementation 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 Gen2 air interface)

# Reading API – Event Cycles



- 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 Gen2 air interface)

# Writing API – Command Cycles



For a 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

- ECSpec/CCSpec refer to a field by name; E.g., “epc”, “lot”, “expiration”, ...
- Fieldname tells ALE where to find the data.
  - Could be fixed (bank/offset) or variable location (e.g., use ISO 15962 directory)
  - 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”
- Fieldnames of the form *@bank.length[.offset]*

## Built-in datatypes:

- “epc”
- “unsigned integer”

## Built-in formats:

- For the “epc” type: epc, epc-tag, epc-hex, epc-decimal
- For the “unsigned integer” type: hex

Vendor extensions and extensions in future ALE versions likely

For a 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

- All five request/response APIs
  - SOAP/HTTP (XML over HTTP)
  - Binary (new for ALE 1.1)
- Reading and Writing Callback APIs
  - XML over raw TCP
  - XML over HTTP
  - XML over HTTPS (new for ALE 1.1)
  - Binary (new for ALE 1.1)
- Binary bindings to be based on LLRP serialization
  - Goal is to facilitate sharing of code between LLRP and ALE implementations



The global language of business

[www.gs1.org](http://www.gs1.org)