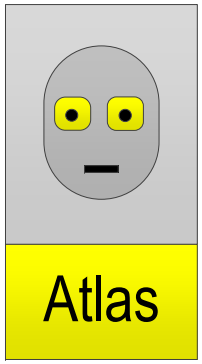
What is Atlas?

Atlas is a **Cognitive System** comprised of:

- A cognitive agent (**CA**).
- A cognitive environment (**CE**).

Atlas **Cognitive Agent** can:

- Learn
- Apply knowledge
- Communicate



Atlas and its environment are created from a user application by:

- Loading Atlas library (DLL/so)
- Calling method **A0:Start**

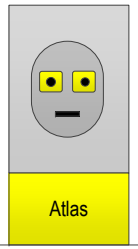
What is a cognitive agent?

- A cognitive agent is an logical processor of information described in a structured cognitive environment.

- Cognitive agents are capable of:
  - Learning rules
  - Understanding natural language (semantic)
  - Inferring from observed phenomena (semiotic)
  - Simulating a situation in past, present, and future.
  - Communicating results back to the agent's principal.
  - Managing the cognitive environment.

- Atlas cognitive agent is abstract, but its clones are instantiated within the environment.

Legend



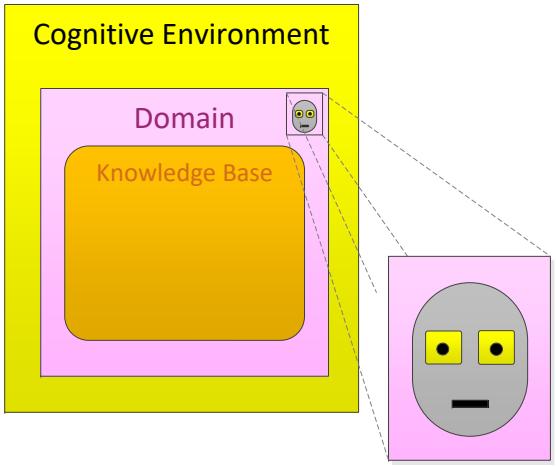
What is a cognitive environment?

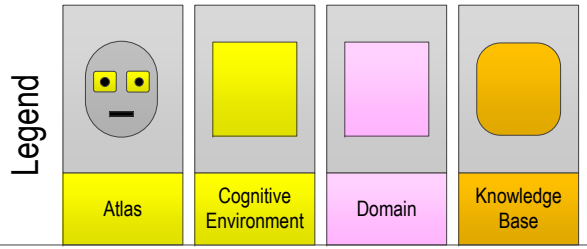
- Atlas cognitive environment is collection of one or multiple **domains**.

- Domains are disjointed pools of knowledge and the application of that knowledge.
- All domains are accessible to Atlas Agent.

- Each domain contains:
  - Knowledge base that describes the rule set of the domain.
  - Simulation that describes a state of the world subjected to the rule set.
  - One or multiple managing agents.

Domain Interface		
D1	Create
D2	_Create
D3	Destroy
D4	CreateContext
D5	_CreateContext
D6	DestroyContext
D7	CreateSimulation
D8	_CreateSimulation
D9	DestroySimulation

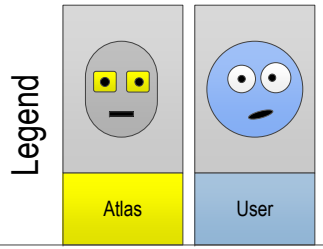




How does Atlas learn?

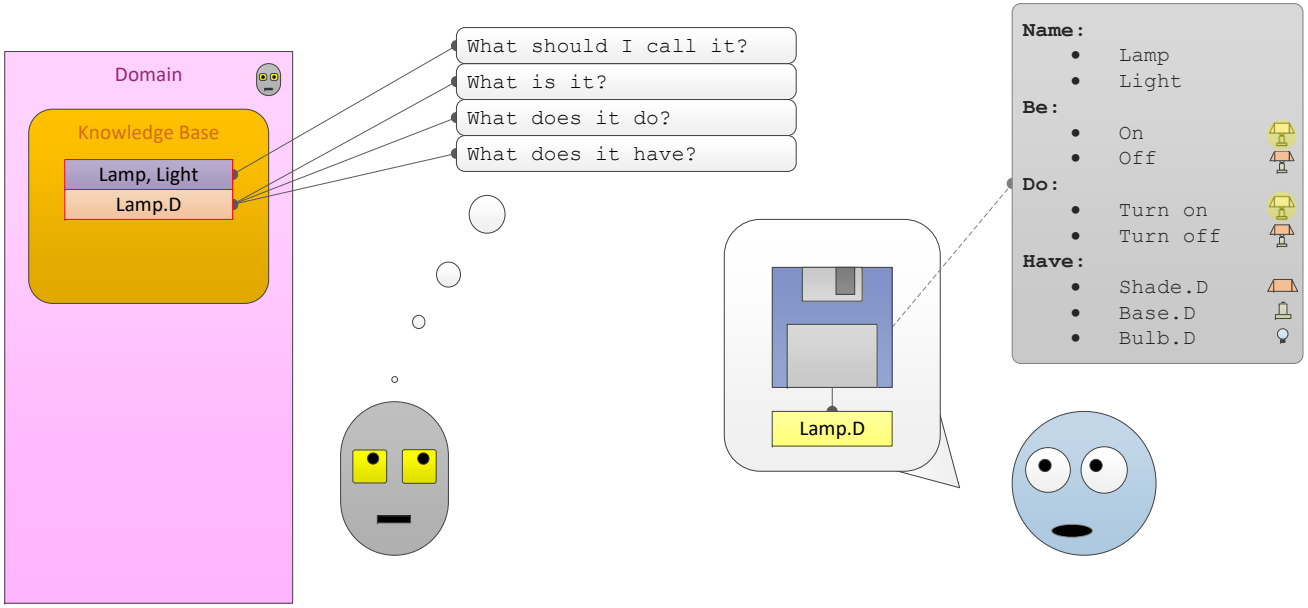
- Atlas learns by loading **cognitive containers**.
- A cognitive container describes a **concept** to Atlas by its:
  - Declaration (Name)
  - Definition
  - Interpretation
  - Instantiator
- A cognitive interface allows mapping of source code, referenced by the interpretation, to the definition.

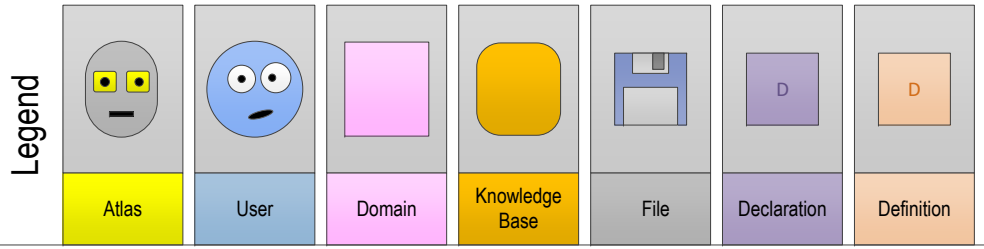
Cognitive Interface		
	C1	Get Interface
	C2	Set Instance Manager
	C3	Do
	C4	Be
	C5	Have
	C6	Get Concept
	C7	Get Subconcept
	C8	IsCompatible
	C9	Set Sub Instance Manager
	CA	AddAttribute
	CB	Pause
	CC	Resume
	CD	Shutdown



How is a concept defined?

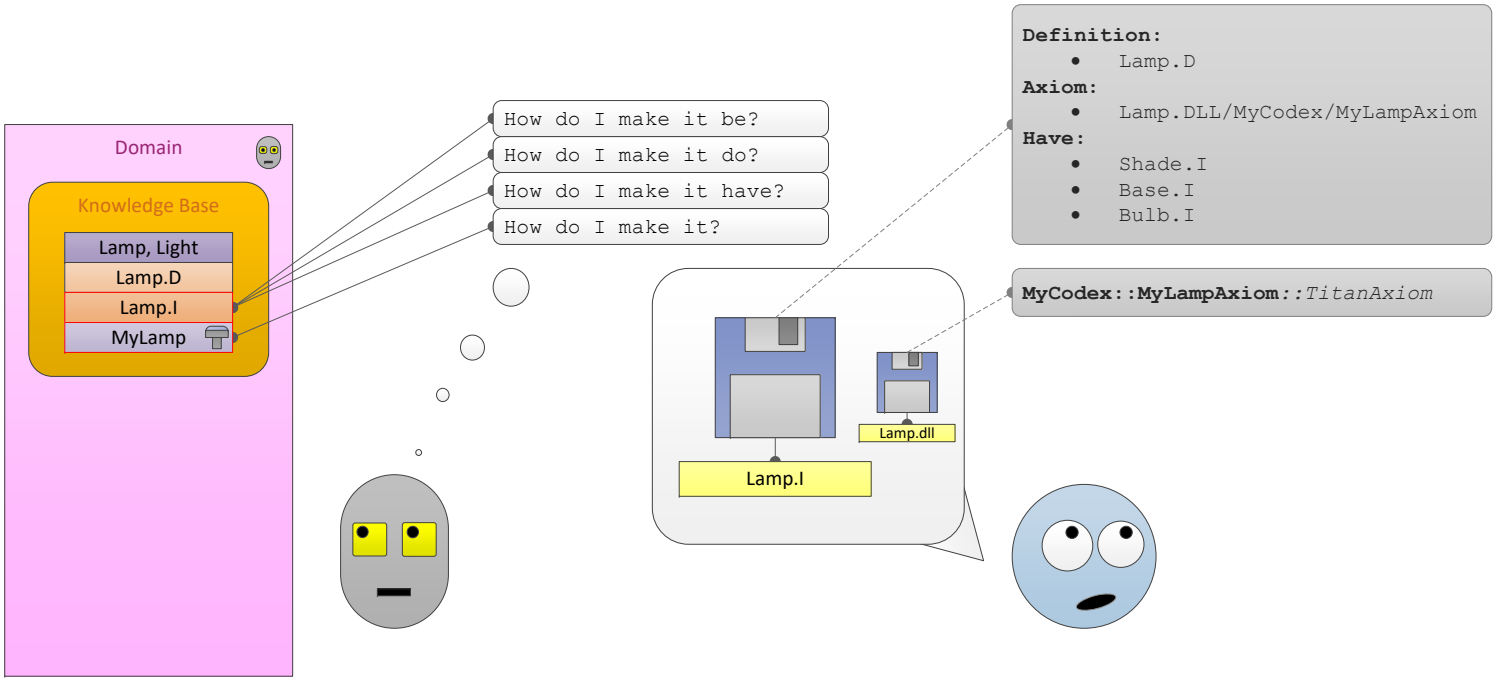
- A concept name is known as a **declaration**.
- Answers to Atlas' questions come in the form of a definition file.
- A **definition** is a plain text file with a (.D) extension.
- For the lamp concept, *Lamp.D* is supplied to Atlas as a definition file.
- The definition file helps Atlas understand the aspects of the concept and how the concept relates to other current or future concepts.

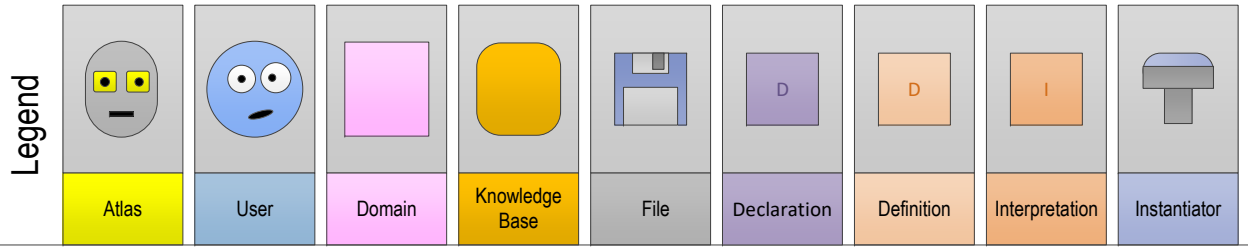




How is a concept interpreted?

- Once a concept is defined, it needs to be interpreted. The **interpretation** file (.I) directs Atlas to the axioms (rules) found in codices (books) of published libraries to learn the new interpretation.
- The interpretation file helps Atlas understand how the interpret and manipulate the aspects of the concept described in the definition file.



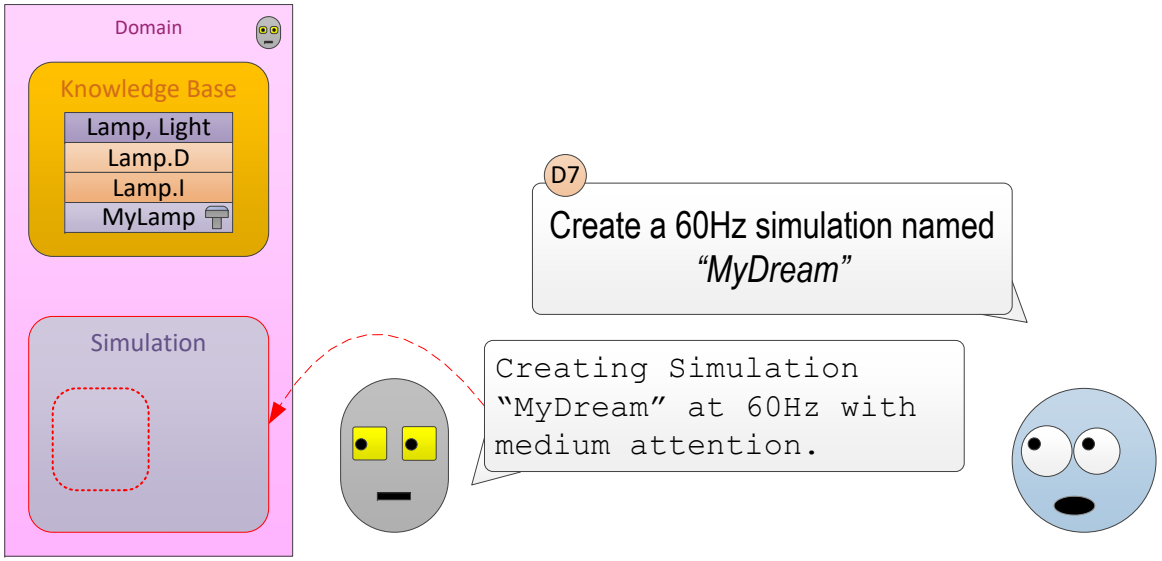


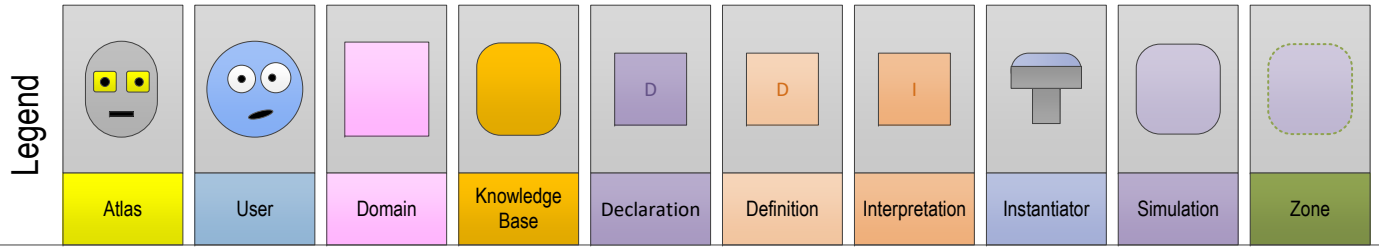
How does Atlas apply knowledge?

- To apply the knowledge it has learned, Atlas needs to create a **simulation**.
- Atlas is required to create a simulation (or scenario) of a set complexity (speed and accuracy) in order for it to apply the knowledge it has acquired.

- The simulation is created by:
  - A domain manager

Simulation Interface		
	S1	GetTime
	S2	Create*Atlas*Clone
	S3	Destroy*Atlas*Clone
	S4	CreateSimulationObject
	S5	DestroySimulationObject
	S6	SetSimulationObjectParent
	S7	LoadSimulationObject
	S8	SaveSimulationObject
	S9	ActivateSimulationObject
	SA	DeactiveSimulationObject

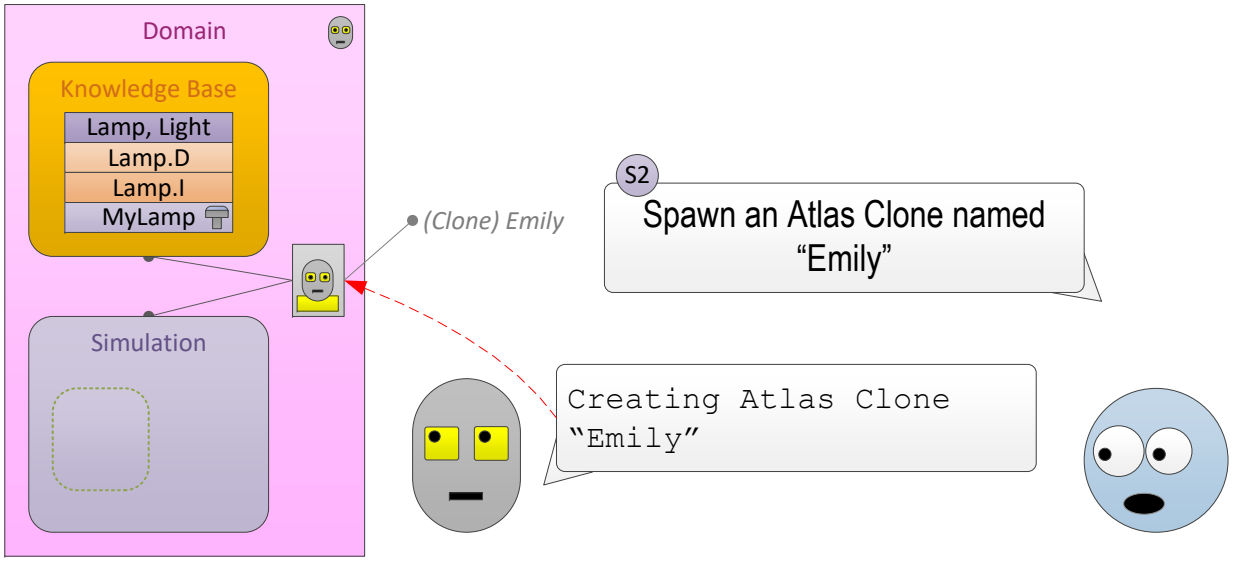


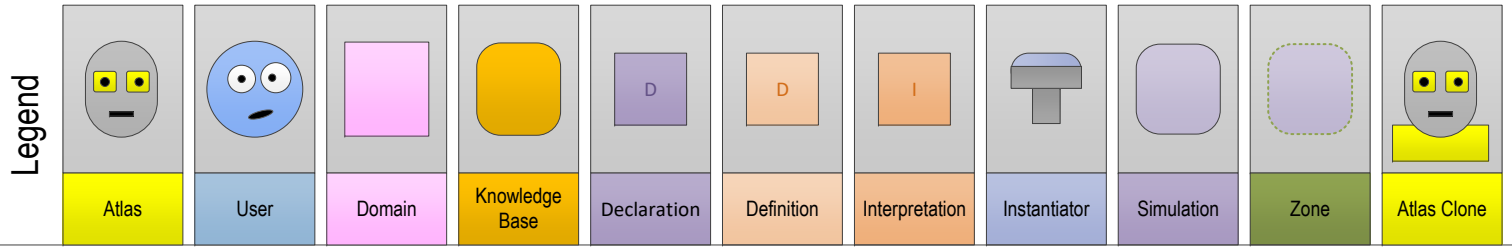


How does Atlas interact with the knowledge?

Atlas interacts with all knowledge through its **agent clones**:

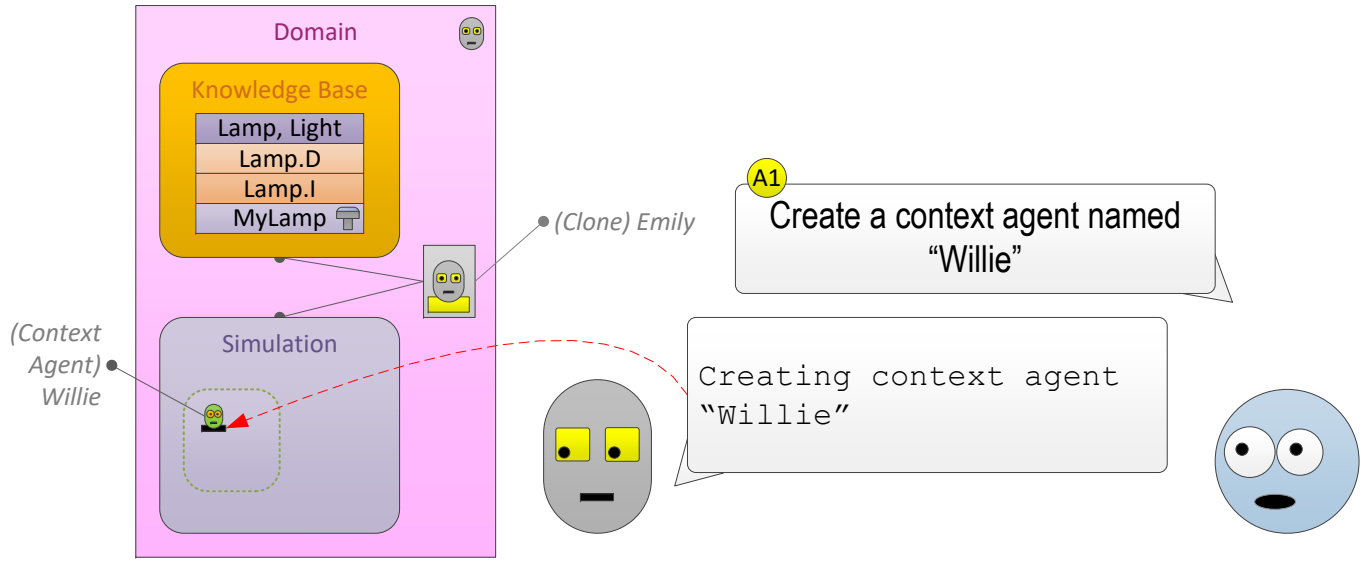
- One or more agent clones can be spawned to perform within a domain.
- All agent clones share the knowledge base of the domain in which they operate.
- Atlas clones reside within the domain; however, they can communicate across domains.
- Each spawned clone has a personalized attention span and activity frequency.
  - Attention span is the total number of concepts tracked during a clone's reasoning.
  - Activity frequency is the maximum number of times the clone processes the activities of its context agents within one second.

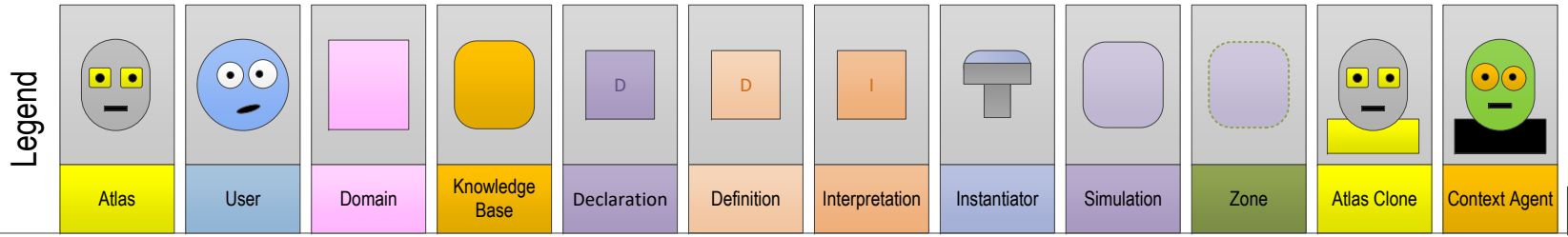




How do clones work?

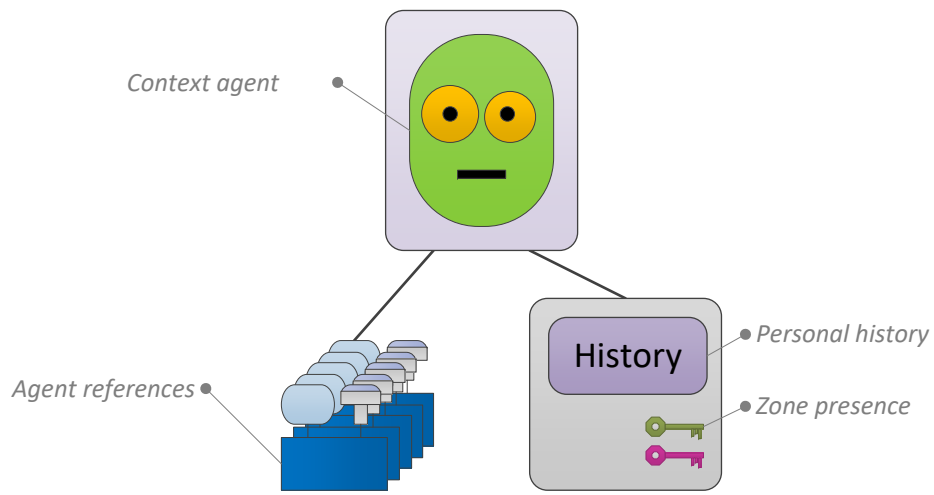
- Clones work on behalf of context agents.
- Clones borrow the context agent's context for a period of time to perform the tasks requested by that context agent.

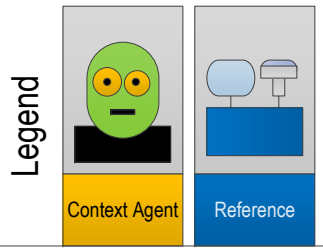




What are context agents?

- Context agents are independent identities created by the developer inside the simulation.
- A context agent is a cognitive representative of one or multiple references.
- All references of a context agent share the context agent's:
  - Personal history of observations and actions.
  - Presence in one or multiple context zones.
- There is no theoretical limit to the number of concurrent agents in a domain. Total activity in the domain is distributed between the agents.





Agent Interface

A1	Create
A2	_Create
A3	Destroy
A4	RegisterManager
A5	Join*Atlas*Clone
A6	Leave*Atlas*Clone
A7	EnterZone
A8	_EnterZone
A9	ExitZone
AA	_ExitZone
AB	Tell
AC	Ping
AD	Transmit
AE	ExecuteScript
AF	GetAttachment
AG	_GetAttachment
AH	RegisterExpert
AI	UnregisterExpert
AJ	Observe

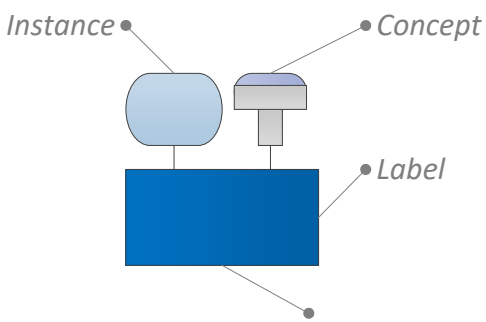
What are references?

- References are cognitive pointers to **instances**.
- References help the developer/user label and categorize information so it can be **inferred** by Titan upon request.

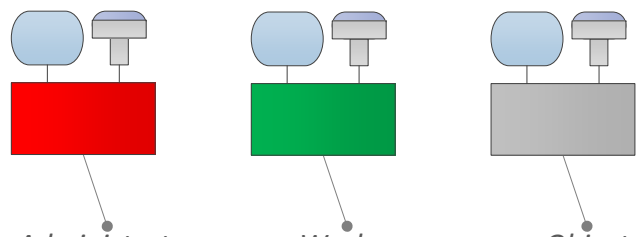
- A reference has:
  - Instance/Element ID.
  - Concept interpreting the instance.
  - One or multiple labels (names).

- A reference can be one of three types:
  - **Administrator** of a context agent.
  - **Worker**.
  - **Object**.

Reference Interface		
	R1	Add
	R2	_Add
	R3	Create
	R4	_Create
	R5	Destroy
	R6	RegisterManager
	R7	GetPublicHandle
	R8	Load
	R9	Save
	RA	Observe
	RB	GetInfo
	RC	GetTransform

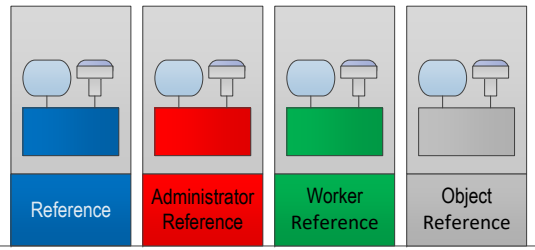


Generic Reference Icon for administrator, worker, or objects.



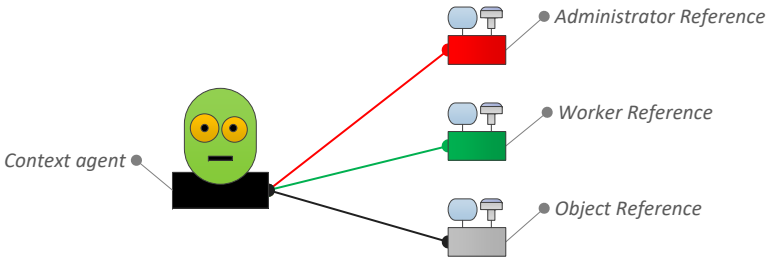
Administrator Reference *Worker Reference* *Object Reference*

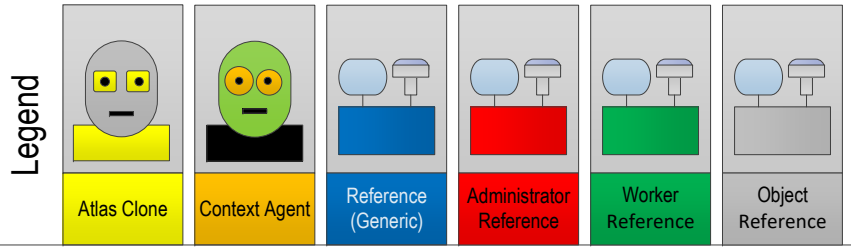
Legend



How do references become part of a context agent?

- When a reference is created, it joins a context agent as one of the following:
  - **Administrator**, which can:
    - Modify context agent parameters such as name, hosting clone, permissions, etc.
    - Enter/Exit context agent domains and context zones.
    - Register/Unregister context agent as an expert.
    - Have worker privileges.
  - **Worker**, which can:
    - Communicate on behalf of the context agent.
    - *Observe* directives.
    - Have prop privileges.
  - **Object**, which can:
    - *Ping* directives allowed by context agent.
    - Is visible in zones entered by context agent.

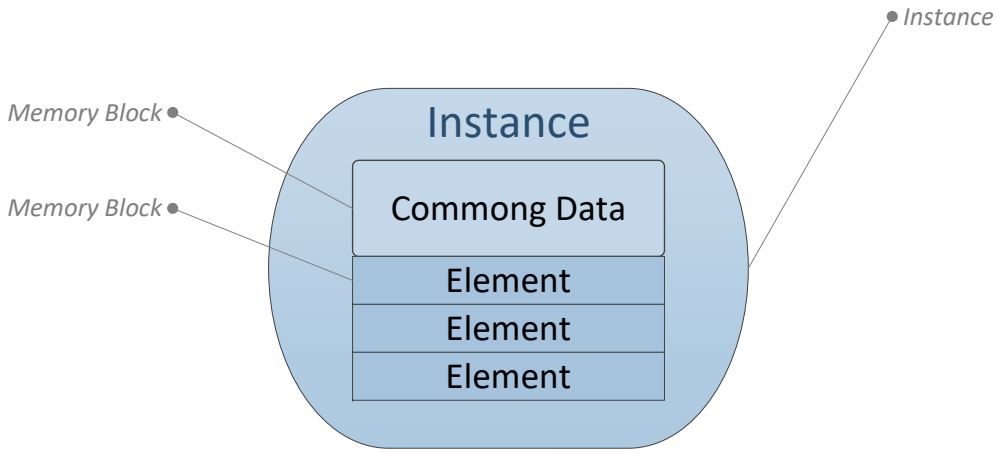


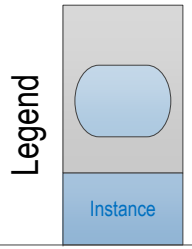


What are instances?

- An instance is a simulated example of a concept interpretation in a particular situation.
- Technically, an instance of a concept is a memory reference to:
  - A single block of common data.
  - One or multiple blocks of element data.
- Multiple references can point to the same instance.

Instance Interface		
	M1	Create
	M2	Destroy
	M3	Load
	M4	Save
	M5	GetValue
	M6	SetValue
	M7	GetSimulationObject

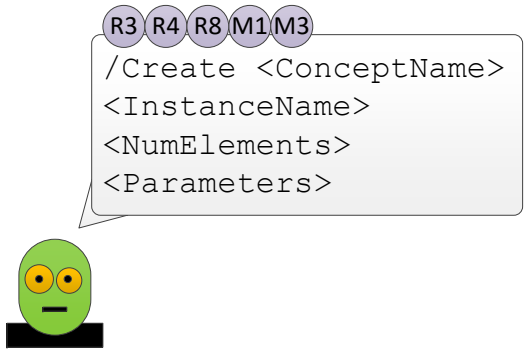


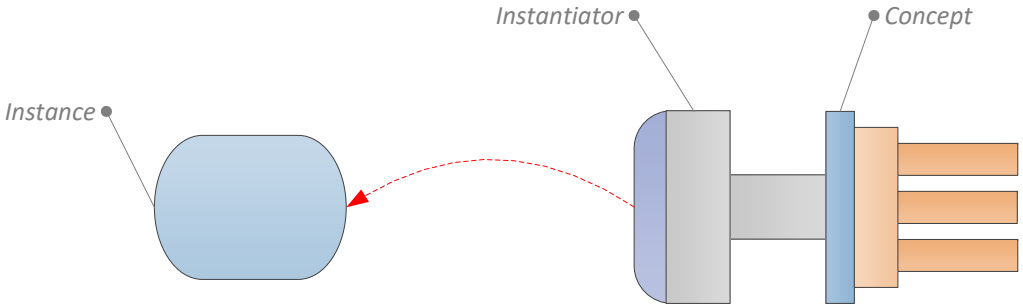


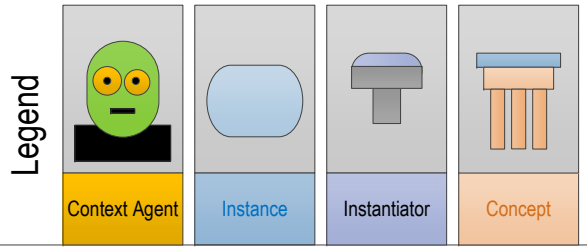
How are instances created?

- Instances can be created directly by:
  - Instance *Create* method **M1**
  - Instance *Load* method **M3**

- Instances can also be created indirectly through references by:
  - Reference *Create* method **R3**/**R4**
  - Reference *Load* method **R8**
  - */Create* command



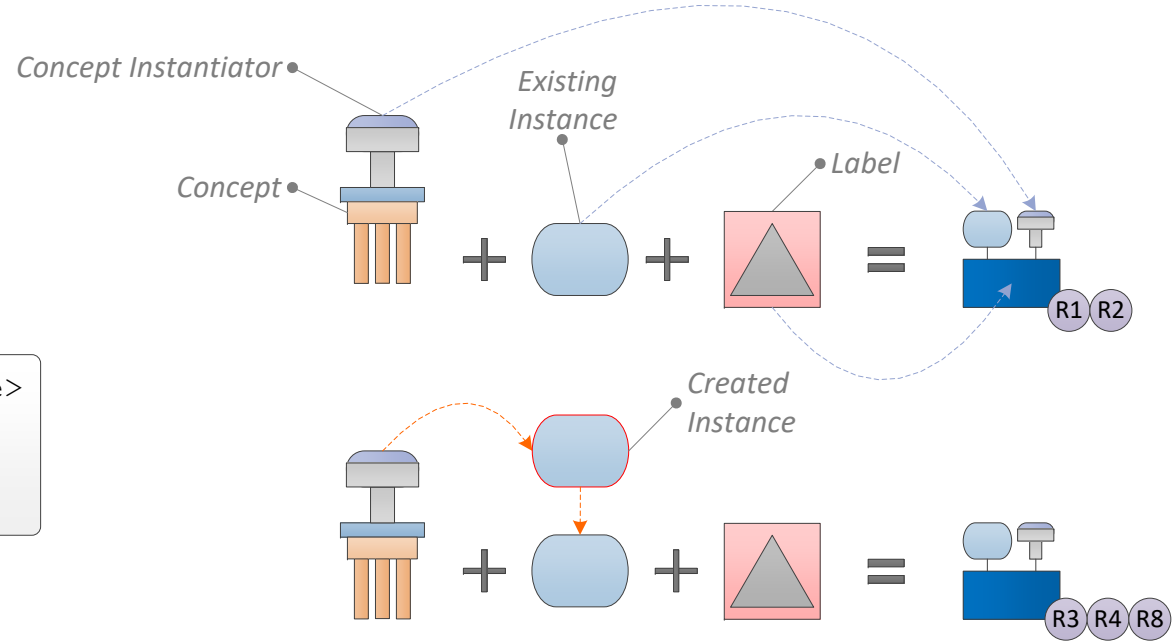


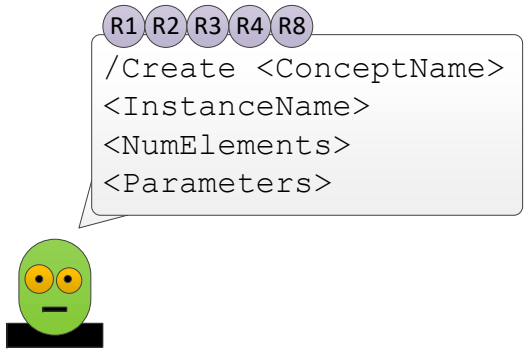


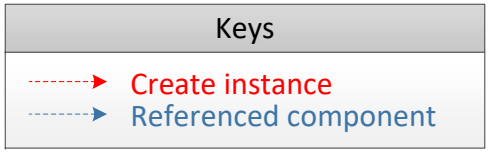
How are references created?

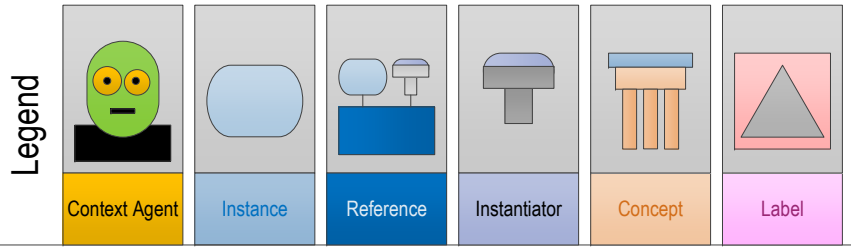
- References can be created by:
  - Adding an existing instance of a specified concept to a label using method **R1/R2**
  - Creating a new instance from a concept and assigning it a label using method **R3/R4**
  - Loading reference data from a file using method **R8**
  - */Create* command

- All references created by a (caller) reference will automatically join the context agent of that reference.





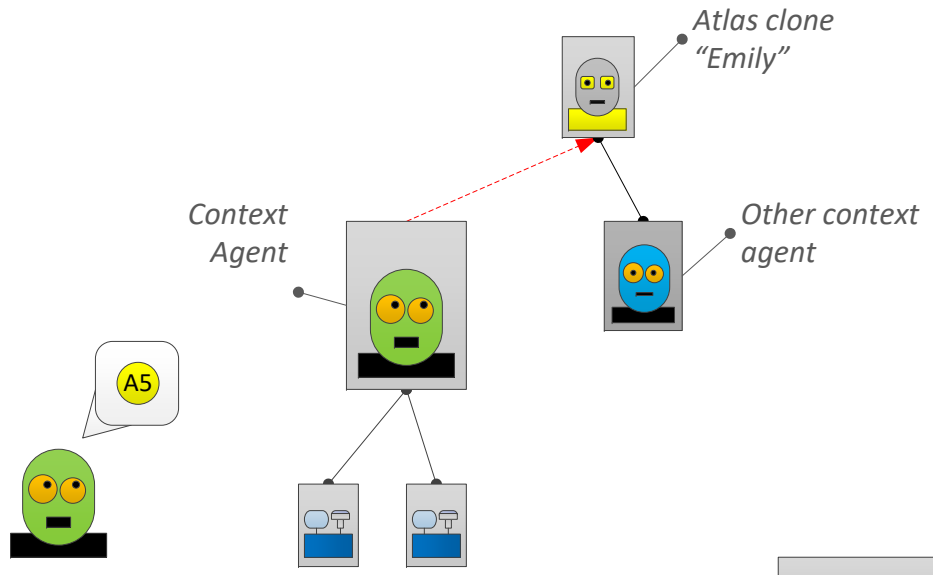


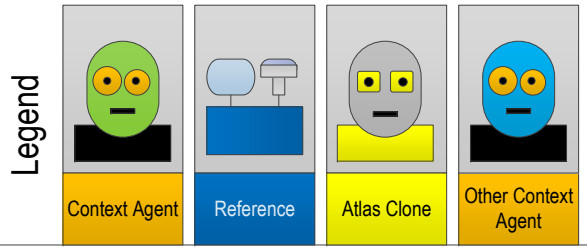


How do context agents perform work?

- Context agents can join an Atlas clone to be activated:
  - At a specific time.
  - At regular time intervals.
  - By observation (Hear, Observe, Receive).
  - By spatial-temporal rendezvous.

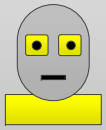
- Context agents can join Atlas Clones by name using method **A5**.
- The simulation agent regulates the joining process.





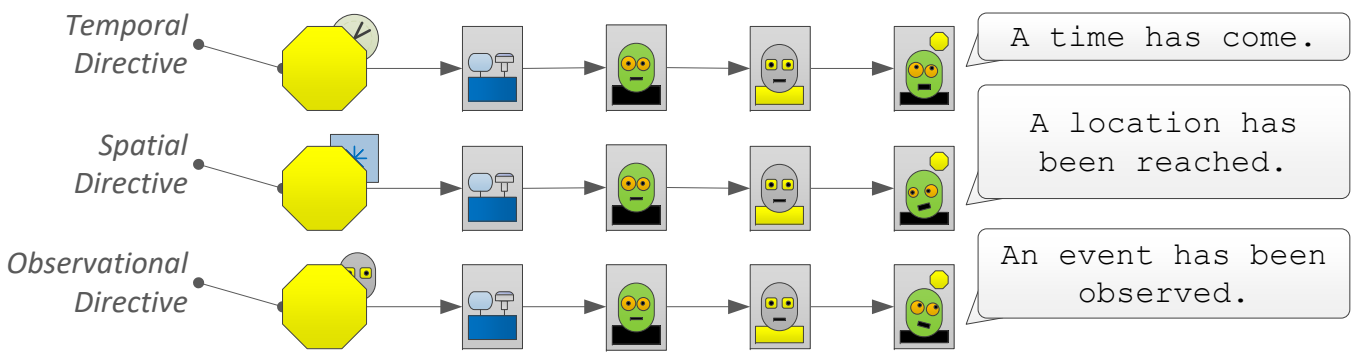
Keys
Join clone

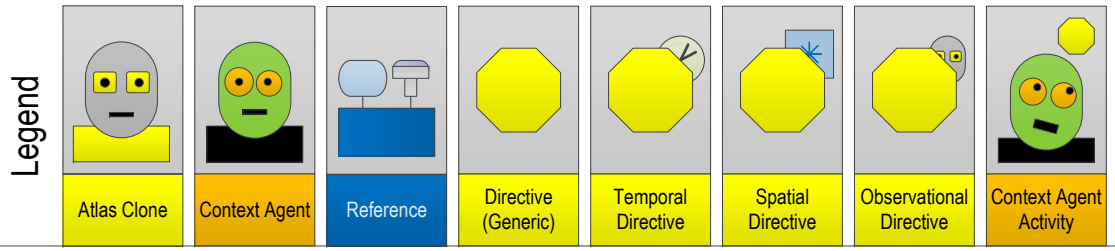
What are the agent roles?

Icon	Name	Role
	Atlas	• *Cognitive Agent*
	Atlas Clone	• *Spawn of Atlas* • Works on behalf of context agents
	Simulation Manager	• Regulates creation and destruction of *Atlas* agent clones • Regulates manipulation of simulation objects • Regulates reference manipulation
	Domain Manager	• Regulates domain or context creation and destruction • Regulates domain access and configuration • Regulates concept learning and forgetting
	Zone Manager	• Observes zone references • Regulates access to zones
	Guide	• *Special zone manager* • Welcomes and guides agents around the domain • Answers questions about the environment

How is a context agent activated?

- A context agent is activated when one of its references is invoked in a directive.
- Three ways a directive reaches a context agent:
  - Temporal:
    - At a scheduled moment or regular intervals .
  - Spatial:
    - When a simulation object has reached a defined position.
  - Observational:
    - When one of the agent's hosted references is involved in the directive as a subject, object, or observer.
- During activation, the Atlas clone temporarily assumes a context agent's identity to perform its cognitive work.





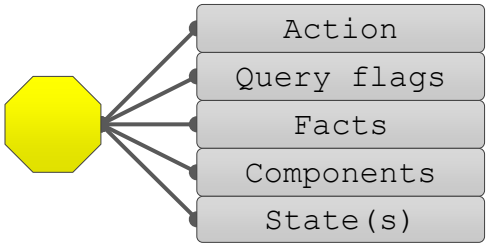
What is a directive?

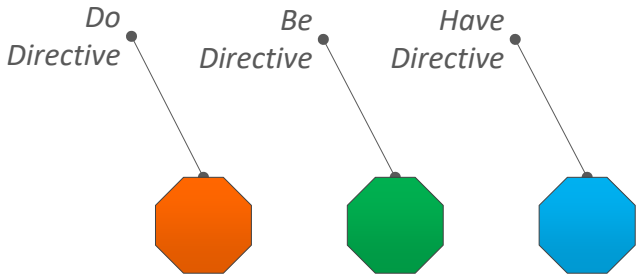
- The **directive** is the main packet of activity in the simulation.
- Directives represent a single **statement** or **query**.

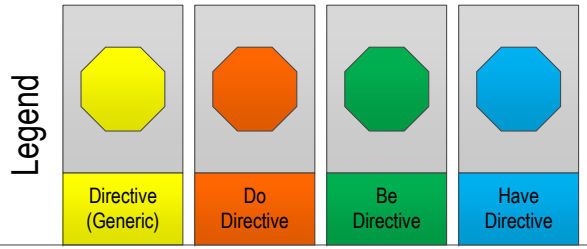
- They can take one of three forms:
  - A state (Be)
  - A state change (Do)
  - A relationship (Have)

- A directive is composed of:
  - An **action**
  - One or multiple **query flags**
  - One or multiple **states**
  - One or multiple **opinions**
  - One or multiple **components**

Directive Interface		
	X1	GetNumComponents
	X2	GetComponent
	X3	GetNextComponent
	X4	GetComponentByName
	X5	RephraseComponent
	X6	GetAction
	X7	GetState
	X8	SetComponentHandshake
	X9	ClearHandshakes
	XA	Remember
	XB	Forget

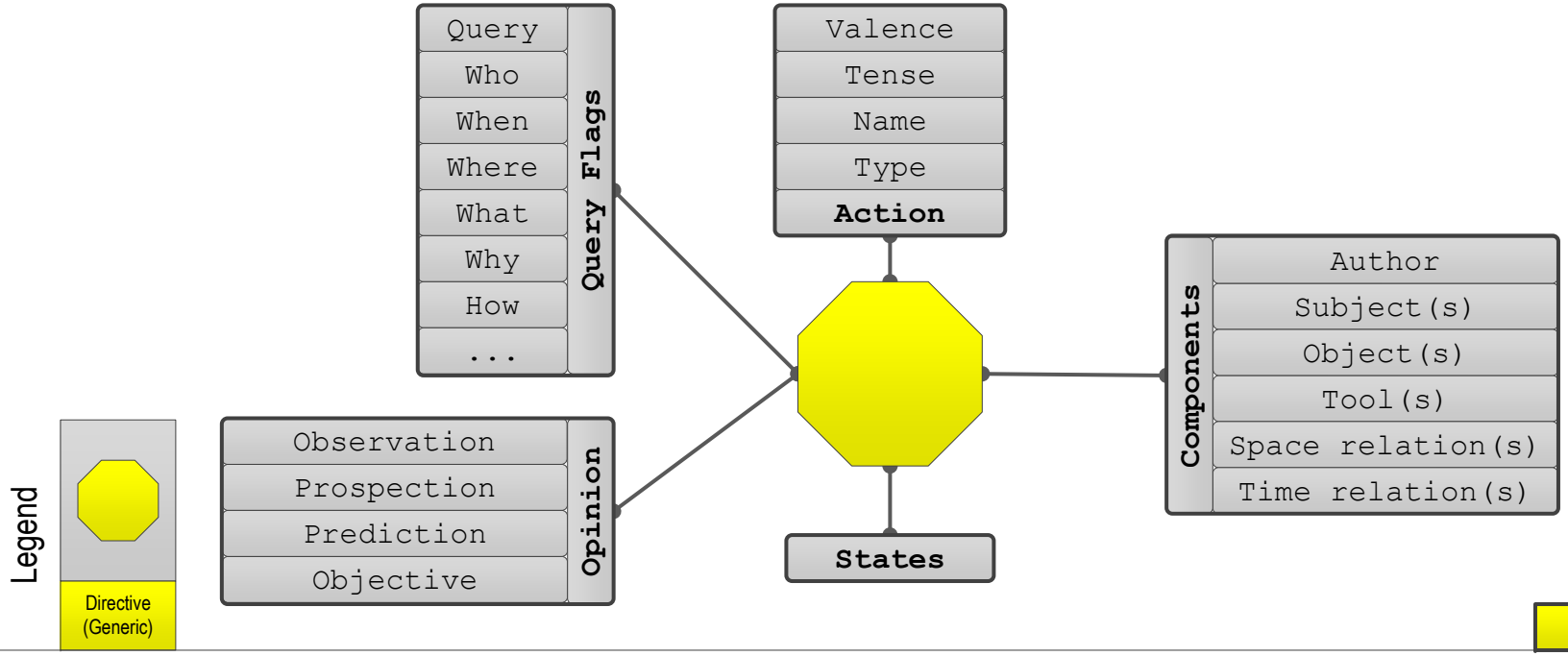






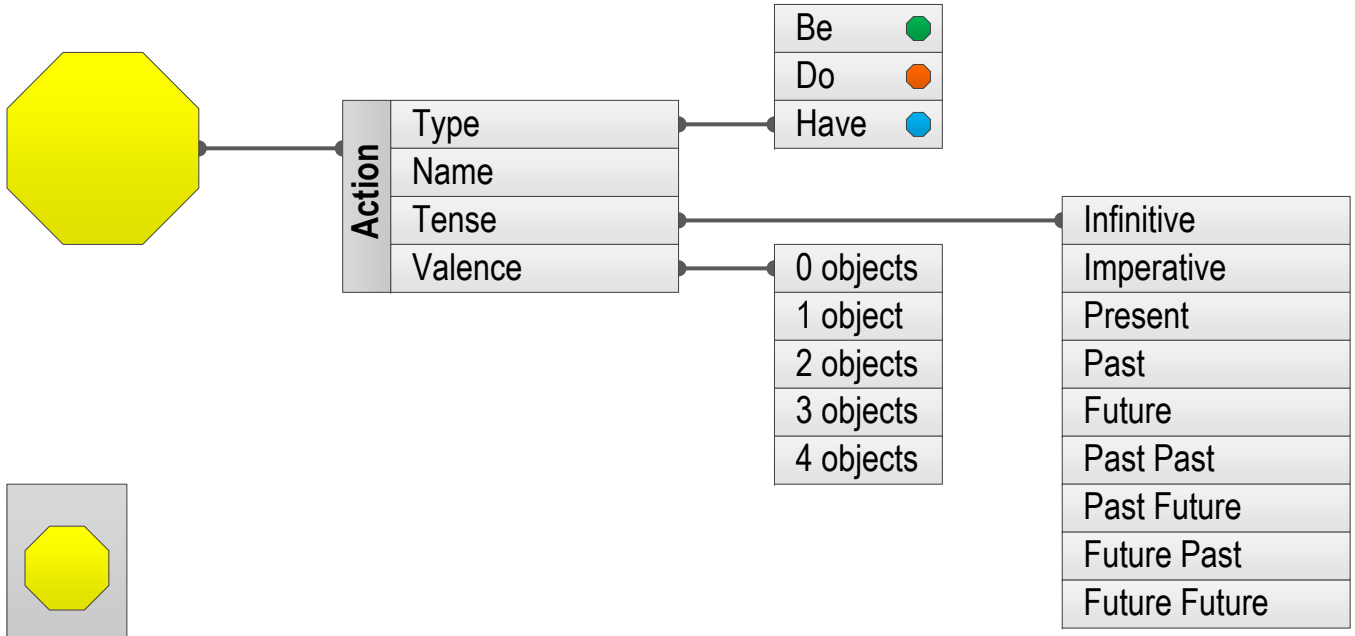
About directive **parts**:

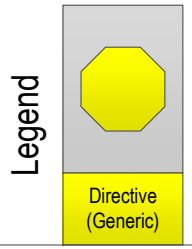
- Directives have the following parts:
  - **Action:**
    - Describes the name and tense of the directive statement.
  - **State:**
    - Describes the modifier of the action. (Adverb)
  - **Query Flags:**
    - A set of flags describing the type of query. If all flags are turned off, then the directive is a statement.
  - **Components:**
    - A statement representing the references involved in the action.
  - **Opinion :**
    - A set of values representing observation, possibility, prediction, and goal of the action.



About the directive **action**:

- The directive **action** block describes:
  - Type of action:
    - Do: The action is a state change where the action name is a verb.
    - Be: The action is a state where the action name is an adjective.
    - Have: The action is a relationship where the action name is a noun.
  - Name of action:
    - The name of the action is a text string containing a word phrase describing the directive.
    - Depending on the action type, the name can be an treated as adjective, verb, or noun.
  - Tense of action:
    - Tense (temporal referent) in which the action is happening.
  - Valence of action:
    - Number of objects involved in the action.

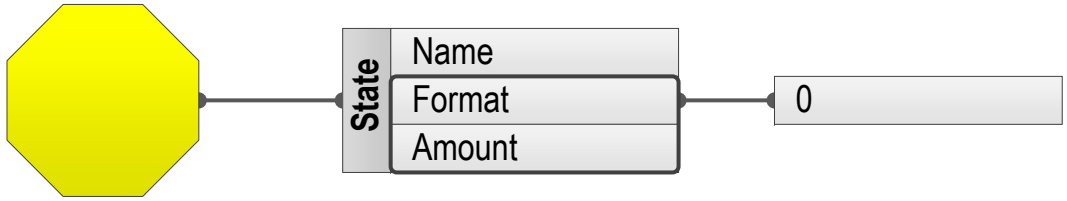




About the directive **state**:

- The directive **state** block describes:
  - Name of the state
  - Amount format (unit) of the state
    - This value should be set to 0
  - Amount of the state
    - Amount of the state where:
      - -100 maximum negation of state
      - 0 no state (should not be a valid value)
      - 100 maximum agreement with state

States are retrieved using method **X7**.



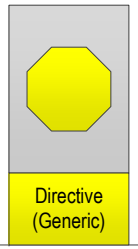
Example state:
I ate the sandwich **happily**.

Name	Format	Amount
Happily	0	50

Example state:
I ate the sandwich **very happily**.

Name	Format	Amount
Happily	0	100

Legend



About the directive **component**:

- The directive **component** block describes:
  - Component **reference**
  - Component **reference name**
  - Component reference **concept name**
  - **Relationship** to reference:
    - Relationship **name**
    - Relationship **format** (unit)
      - This value should be set to 0
    - **Amount** of the relationship in units
  - **Quantity** of reference:
    - Quantity **format** (unit)
    - Quantity **amount** (in units)

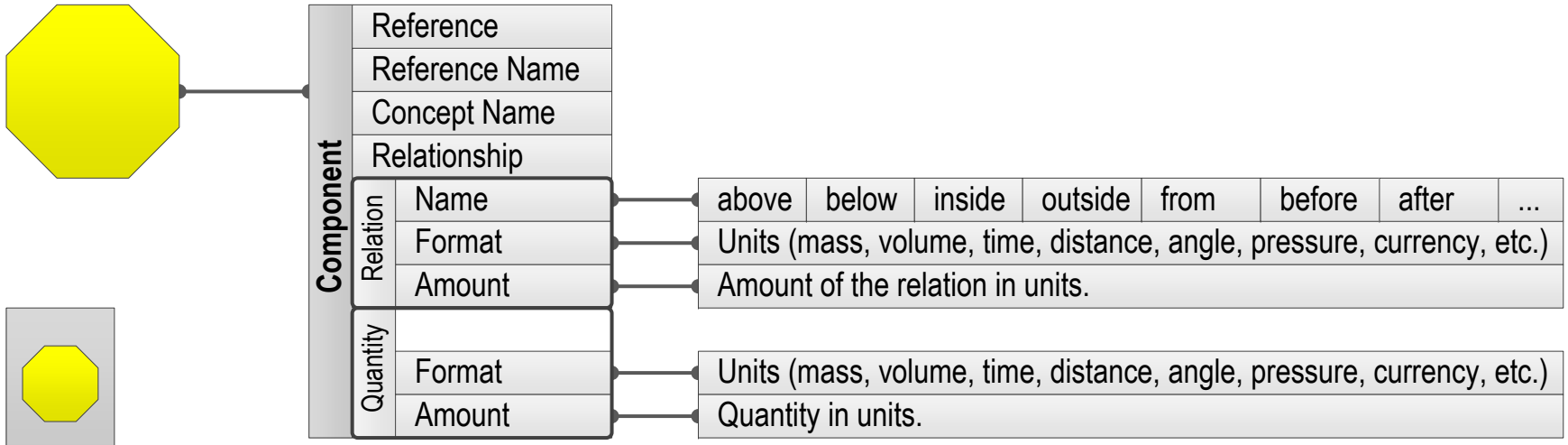
Components are retrieved using methods **X1** and **X2**.

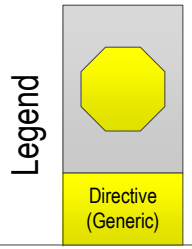
Example component relation:
1.5 meters under the **house**.

Name	Relation	Format	Amount
house	under	meter	1.5

Example component quantity:
3.45kg of **sand**.

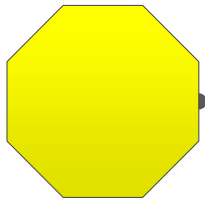
Name	Format	Amount
sand	kilogram	3.45





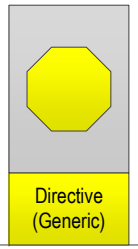
About the directive **query flags**:

- The directive **query flags** block is a multi-hot bitmap describing:
  - **Expression Type**:
    - Type of directive (query, statement, etc.)
  - **Available** expression **components**:
    - Available components or references in the directive.
- Typically, the directive expression type is processed first to determine the course of action of extracting the components, and spatial/temporal references.



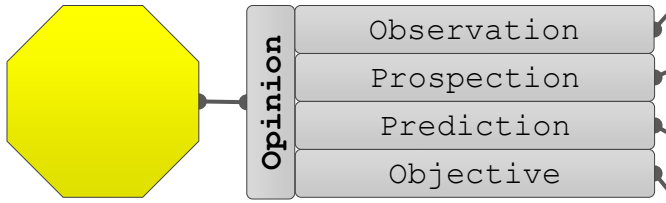
Query Flags		
Expression Type	Query	*True/False question*
	Who	*Query objects in the action*
	When	*Query time of the action*
	Where	*Query location of the action*
	What	*Query type/name of action*
	Why	*Query cause of action*
	How	*Query action state/process*
	Cause	*Action is a cause*
	Consequence	*Action is a consequence*
	Condition	*Action is a condition (if)*
	Reaction	*Action is a reaction(then)*
Available Components	Subject	*Action has a subject*
	Next Subject	*Action has more subjects*
	Object	*Action has an object*
	Next Object	*Action has extra objects*
	Space Object	*Action has a spatial object*
	Next Space Object	*Action has extra spatial objects*
	Time Object	*Action has a temporal object*
	Next Time Object	*Action has extra temporal objects*
	Tool Object	*Action has a tool object (using)*
	Next Tool Object	*Action has extra tool objects*

Legend



About the directive **opinions:**

- The directive **facts** block describes the facts about the action, where:
  - **Observation** describes the action as a perceived fact. (Happened)
  - **Prospection** describes the action as a possibility. (Can happen)
  - **Prediction** describes the action as a probability. (Should happen)
  - **Objective** describes the action as a goal. (Want to happen)



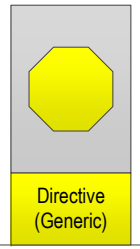
-100	-50	0	50	100
Untrue	Mostly untrue	Unknown	Most true	True

-100	-50	0	50	100
Impossible	Mostly impossible	Unknown	Mostly possible	Always possible

-100	-50	0	50	100
Improbable	Mostly improbable	Unknown	Probable	Absolutely probable

-100	-50	0	50	100
Do not want	Do not need	Unknown	Want	Need

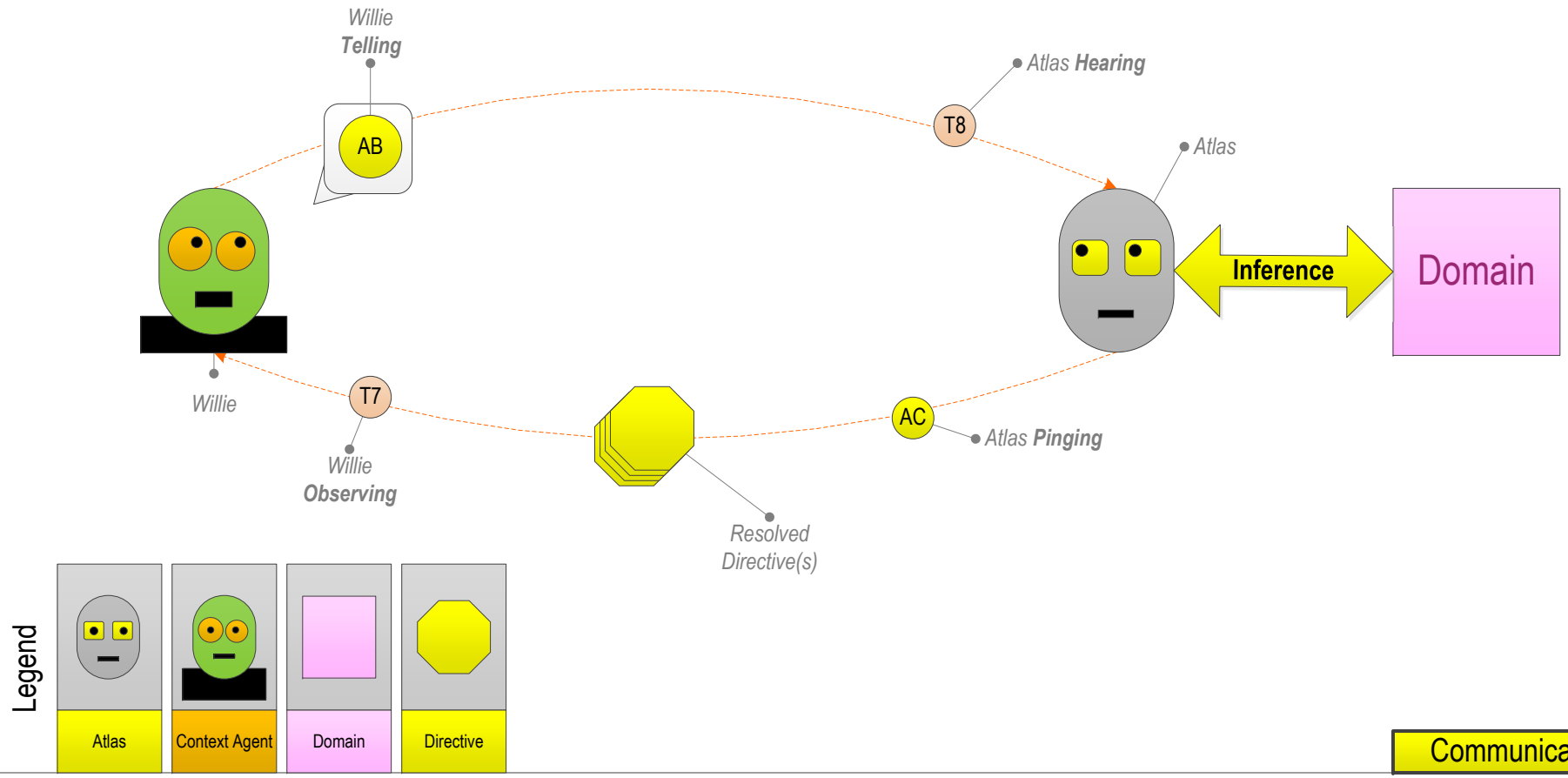
Legend



How is a directive generated?

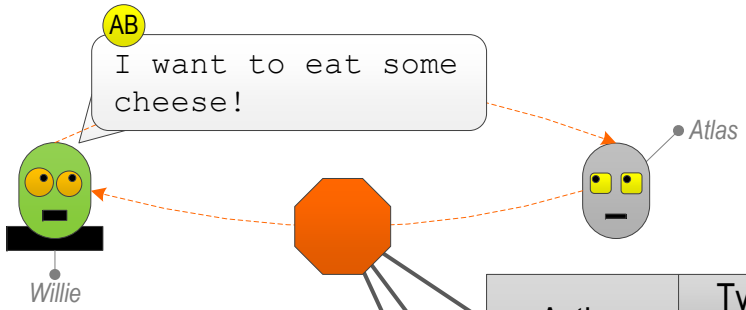
- The directive is generated using the **Tell** method **AB** where:
  - The context agent *Tells* a natural language statement to Atlas.
    - Atlas hears the natural language expression through method **T8**.
    - Atlas **infers** the statement and generates the corresponding directive(s).

- Atlas responds by *Pinging*, **AC**, all the corresponding directive(s) back to the source context agent.
- Calling context agent receives the pinged directive through the **Observe** method **T7**.



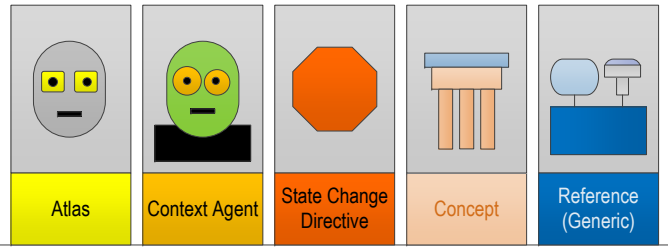
How is a directive processed?

- Once a directive is received by a context agent from Atlas, it can be:
  - Executed in the simulation by *Pinging* (**AC)** the directive back to Atlas.
    - Executed directives trigger observations and simulation methods in the domain.
  - **Remembered**/stored for future use through method **XA**.
    - Remembered directives can be managed, modified, and re-pinged by the context agent that remember them.



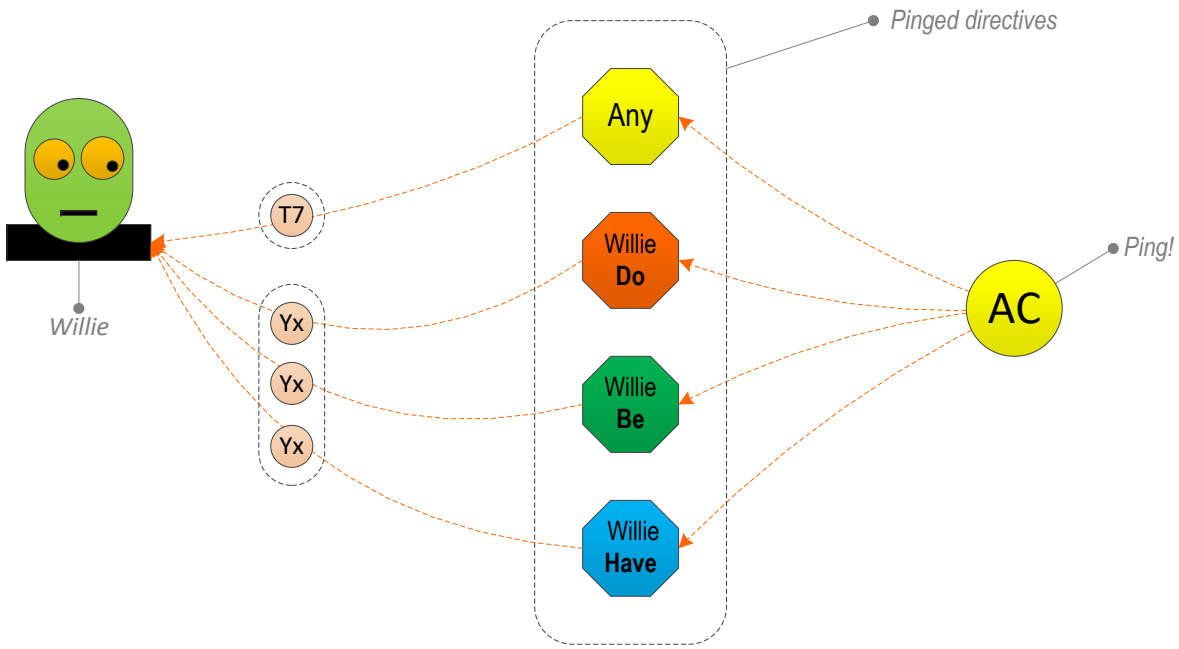
Action	Type	Name	Tense	Valence				
	Do	Eat	Present	1				
Query Flags	Bitmap							
	Subject, Object							
Component	ID	Reference	Name	Relation			Quantity	
	Subject		Willie	0	0	0	0	0
	Object		N/A	0	0	0	0	0

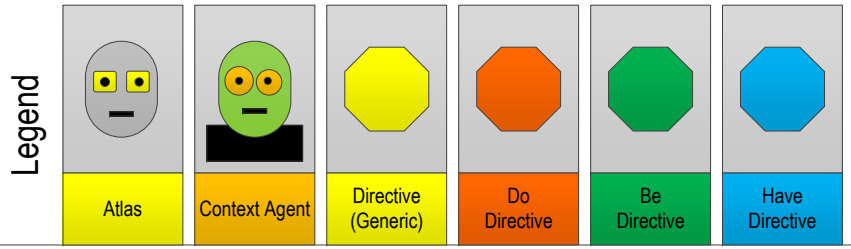
Legend



Types of pinged directives:

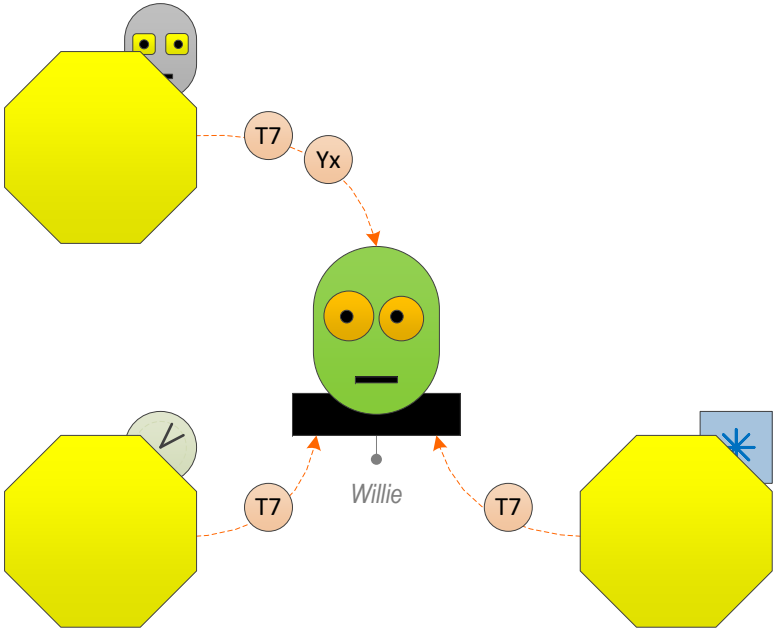
- There two main paths to receiving a pinged directive by a context agent:
  - As a parameter to the **Observe** method **T7:**
    - Context agent registered as an observer for a particular action of another reference.
    - Context agent performed a Tell (AB) and is receiving resolved directive(s).
  - As a parameter in a **Be**/**Do**/**Have** simulation method **Yx**.
    - Context agent is the subject, object, or tool of a triggered action in the simulation.

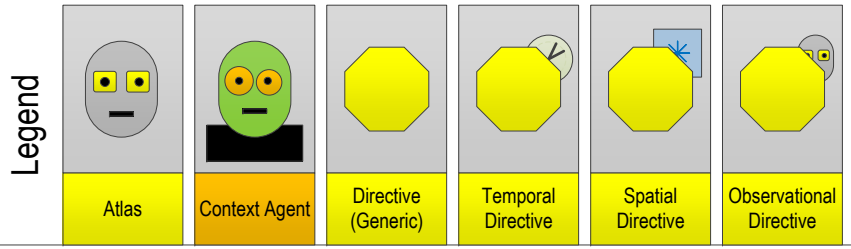




How is a directive used?

- Directives drive the activity of the cognitive system.
- Directives represent discreet events that Atlas tracks to infer a particular idea.
- Activity is triggered in three ways:
  - Expressions being **told** by context agents**.**
  - Timers **expiring.**
  - Spatial locations being **reached**.





Communication outside of the domain:

- Context agents can interact with other context agents in a global domain known as Titan Exchange, where they can be one of two performers:
  - Assistants:
    - Can *Tell* expressions into Titan Exchange.
      - These expressions are *Translated* and *Pinged* as directives to registered observing experts.
    - Can ping directives to Titan Exchange.
  - Experts:
    - Experts are context agents that can *Observe Pinged* directives in the Titan Exchange domain and respond to the directive's author.
    - Can be assistants.
- Assistants can ask questions on behalf of the user.
- Experts can ask and answer questions of behalf of the user.