



Weak Slot and Filler Structures

Introduction Semantic Nets Frames





- Slot-and-filler structures are...

1. Weak?

- We do not say about the specific knowledge that the structures should contain. (**knowledge-poor structures**)

2. Strong?

- specific commitments to the content of the representation are made.





2 views of this kind of structure

- **semantic nets**
- **frames**

Discussion about ...

representations

techniques for reasoning, about them.





Semantic networks:

- Semantic networks are a powerful tool in the field of artificial intelligence (AI), used to represent knowledge and understand relationships between different concepts.
- They are graphical representations that connect nodes (representing concepts) with edges (representing relationships).
- Semantic networks are widely used in natural language processing (NLP), knowledge representation, and reasoning systems.





SEMANTIC NETS:

Definition:

The main idea behind semantic net is the meaning of a concept comes from the ways in which it is connected to other concepts.

A semantic network is a form of knowledge representation that visually illustrates how concepts are related to each other.

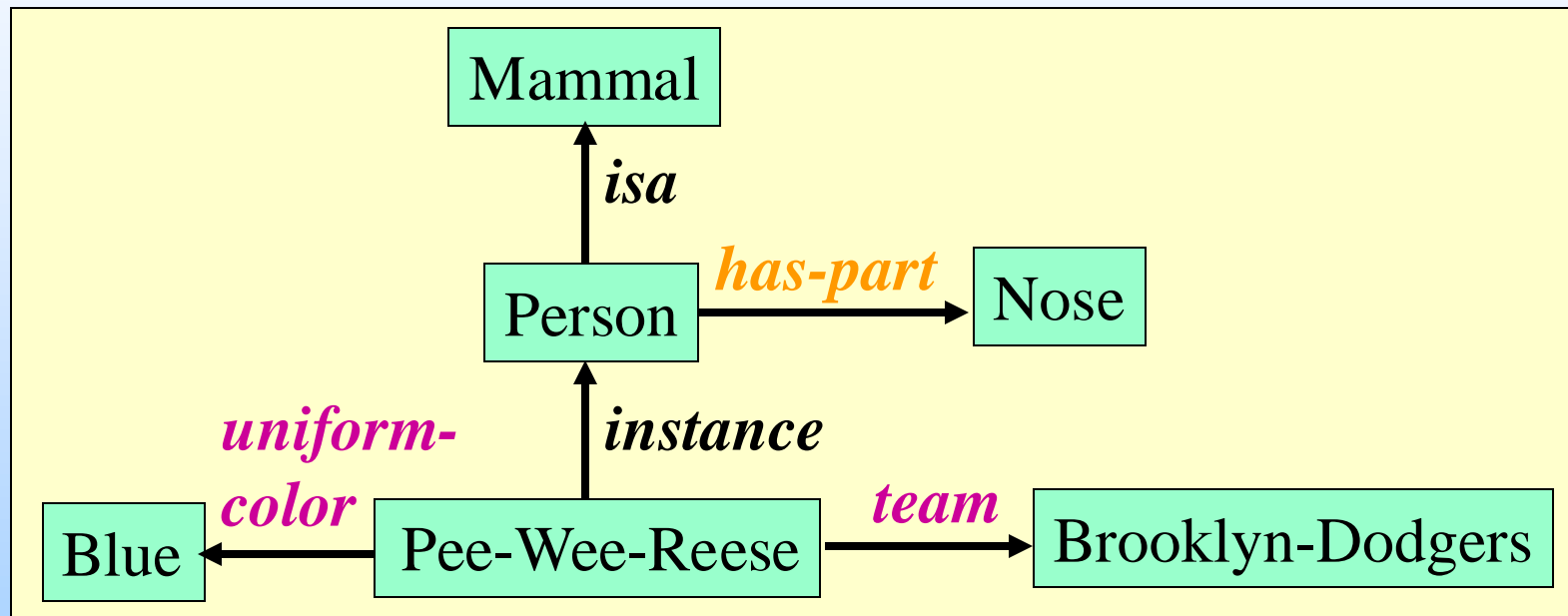
Components in semantic net:

In a semantic net information is represented as a **set of nodes** connected to each other by a **set of labeled arcs**, which represent relationship among nodes.





EXAMPLE FOR SEMANTIC NET:





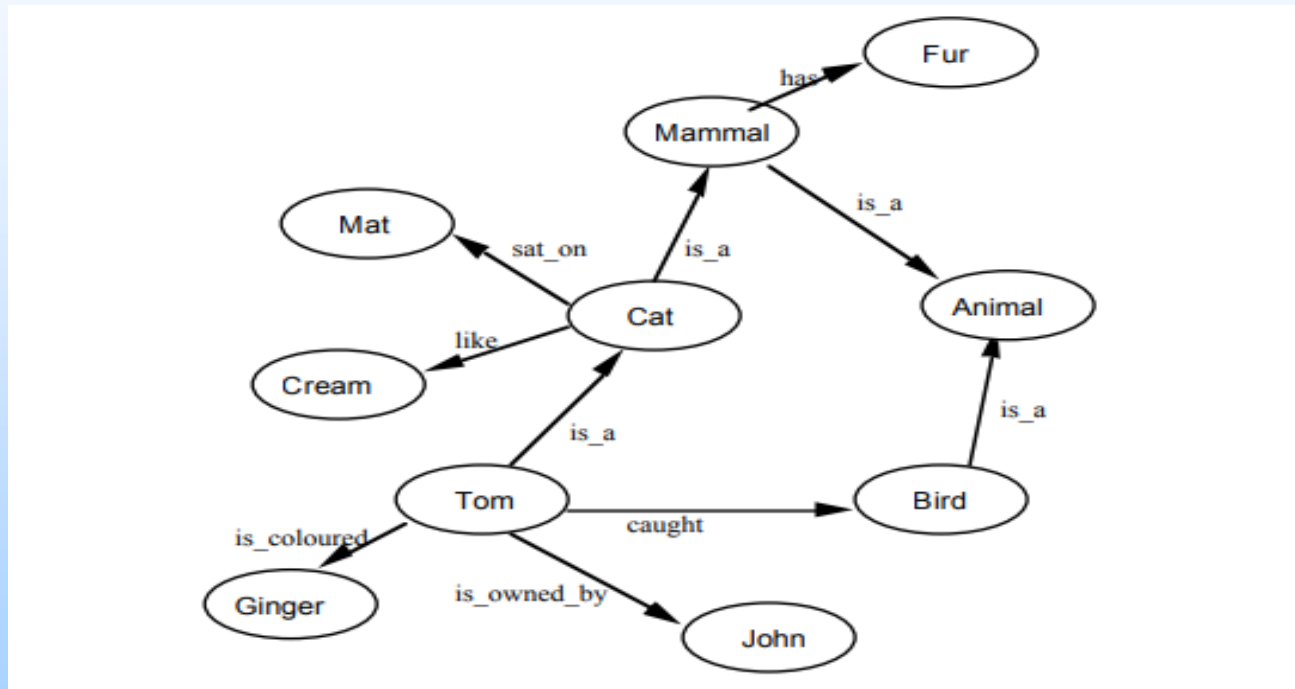
EXAMPLE FOR SEMANTIC NET:

- Tom is a cat.
- Tom caught a bird.
- Tom is owned by John.
- Tom is ginger in colour.
- Cats like cream.
- The cat sat on the mat.
- A cat is a mammal.
- A bird is an animal.
- All mammals are animals.
- Mammals have fur

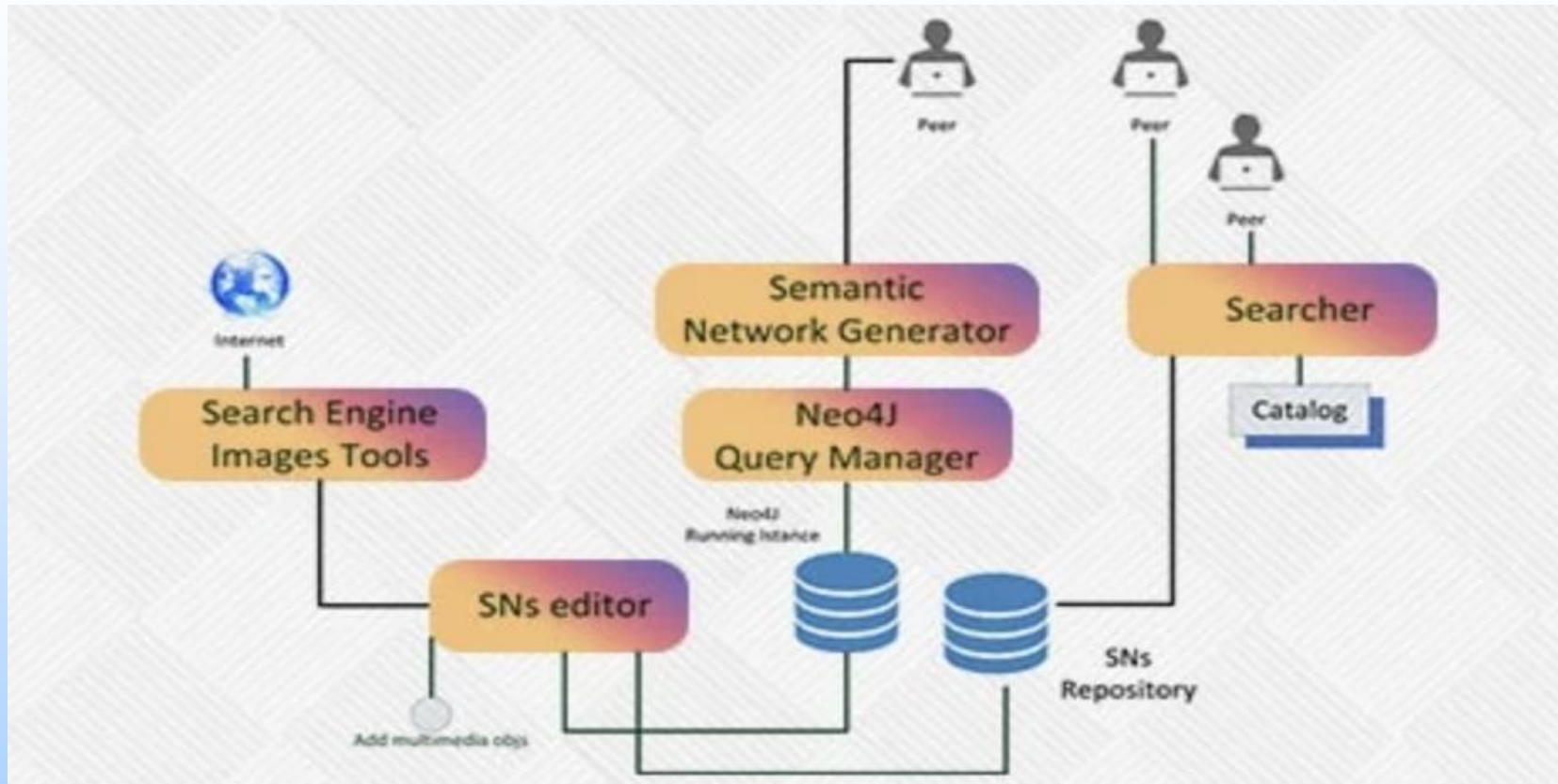




EXAMPLE FOR SEMANTIC NET:



Department of Computer Science and Engineering



Architecture of semantic net



- **Reasoning in semantic net : Intersection search**
- The process of finding relationships among objects by spreading activation out from each of 2 nodes and seeing where the activation meet.
- **Advantage** of slot-and-filler structures over logical representations.?
entity based organization of knowledge, makes this reasoning possible.





Representing Binary predicates:

Semantic nets are a natural way to represent relationships that would appear as ground instances of *binary predicates* in predicate logic.

For the given example Represented in logic as...

- *isa(Person, Mammal)*
- *instance(Pee-Wee-Reese, Person)*
- *team(Pee-Wee-Reese, Brooklyn-Dodgers)*
- *uniform-color(Pee-Wee-Reese, Blue)*





Representing Non-Binary predicates:

•1. Unary predicates in logic:

represented as binary predicates using general purpose predicates. *Isa, instance.*

Ex: **man(Marcus)** is written as...

instance(Marcus, Man)

now it is easy to represent in a semantic net.





Representing Non-Binary predicates:

•1. Unary predicates in logic:

represented as binary predicates using general purpose predicates. *Isa, instance.*

Ex: **man(Marcus)** is written as...

instance(Marcus, Man)

now it is easy to represent in a semantic net.





2. 3/more place predicates:

can also be converted to a binary form by creating one new object representing the entire predicate statement and then introducing binary predicates to describe the relationship to this new object of each of the original arguments.

Ex: **score(Cubs, Dodgers, 5-3)**

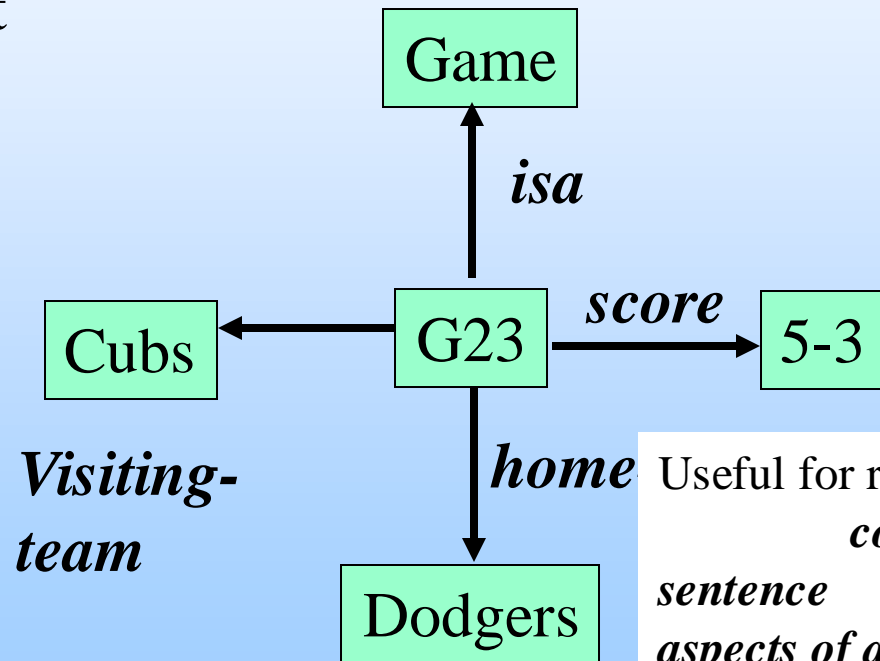
is represented in a semantic net by creating a node to represent the specific game then relating each of the three pieces of information to it





Ex: **score(Cubs, Dodgers, 5-3)**

is represented in a semantic net by creating a node to represent the specific game then relating each of the three pieces of information to it

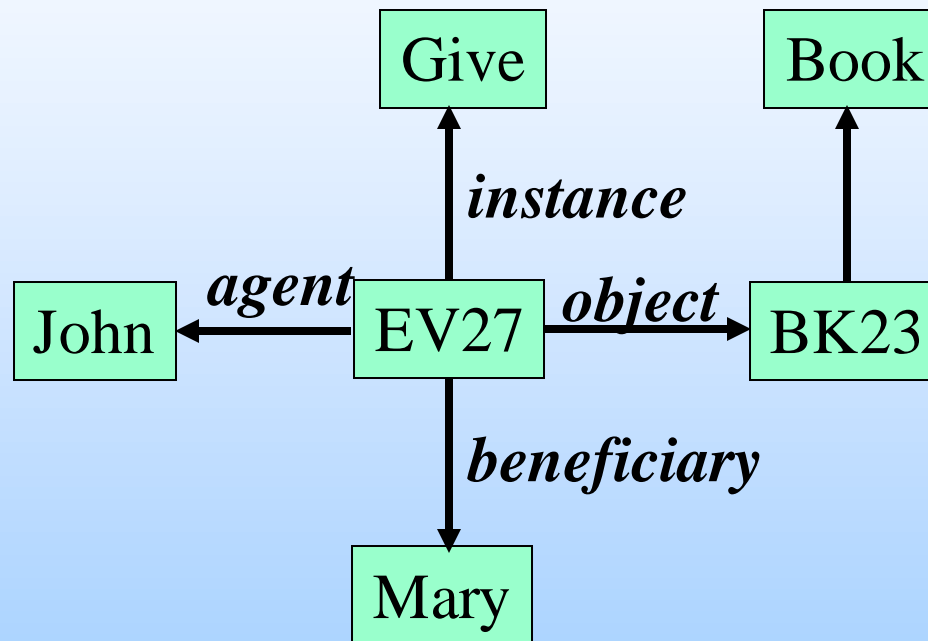


Useful for representing the
*contents of a declarative
 sentence that describes several
 aspects of a particular event.*






Ex: **john gave the book to Mary**

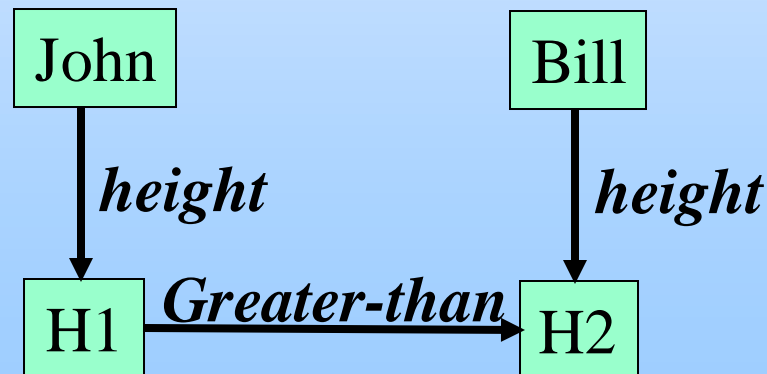


Making some important distinctions

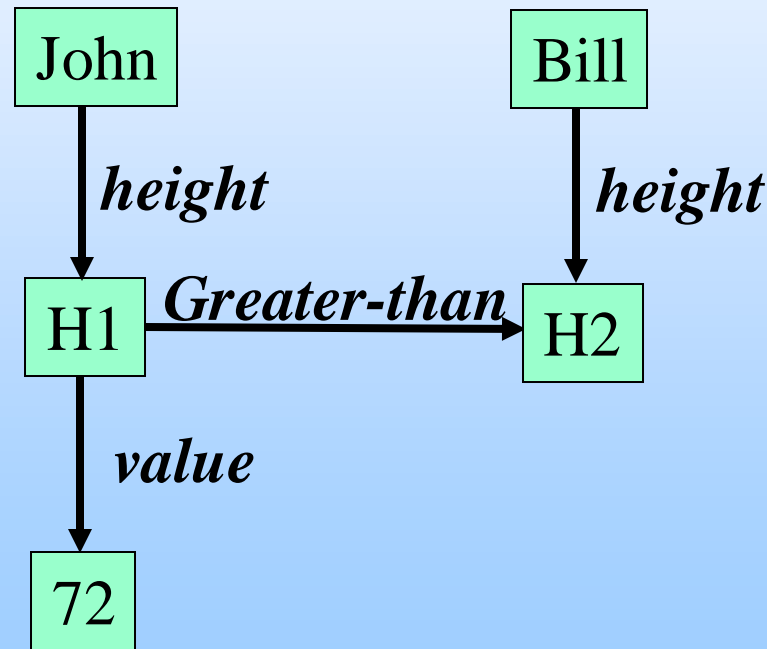
1. There should be a difference between a link
 - that defines a new entity and
 - one that relates 2 existing entities.

Ex:  *Both nodes represent **objects** that exist independently of their relationship to each other*

Representing the fact that “**John is taller than Bill**” using the net



Nodes **H1** and **H2** are new **concepts** representing
John's height & Bill's height respectively.
They are defined by their relationships to the nodes John & Bill.
Using these defined **concepts**,
it is possible to represent such facts as that john's height
increased.(which we could not do before)



To make the distinction clear, it is useful to introduce the arc value.
arc values...

height: defines new entities

greater-than, value: describe relationship among existing entities.

2. The difference between

-the properties of a node itself &

-the properties that a node simply holds and passes on
to its instances.

It is very difficult to capture these distinctions without assigning more structure to our notions of node, link, value.

We do that, when we talk about **frames**.

A network oriented solution...to a simple problem

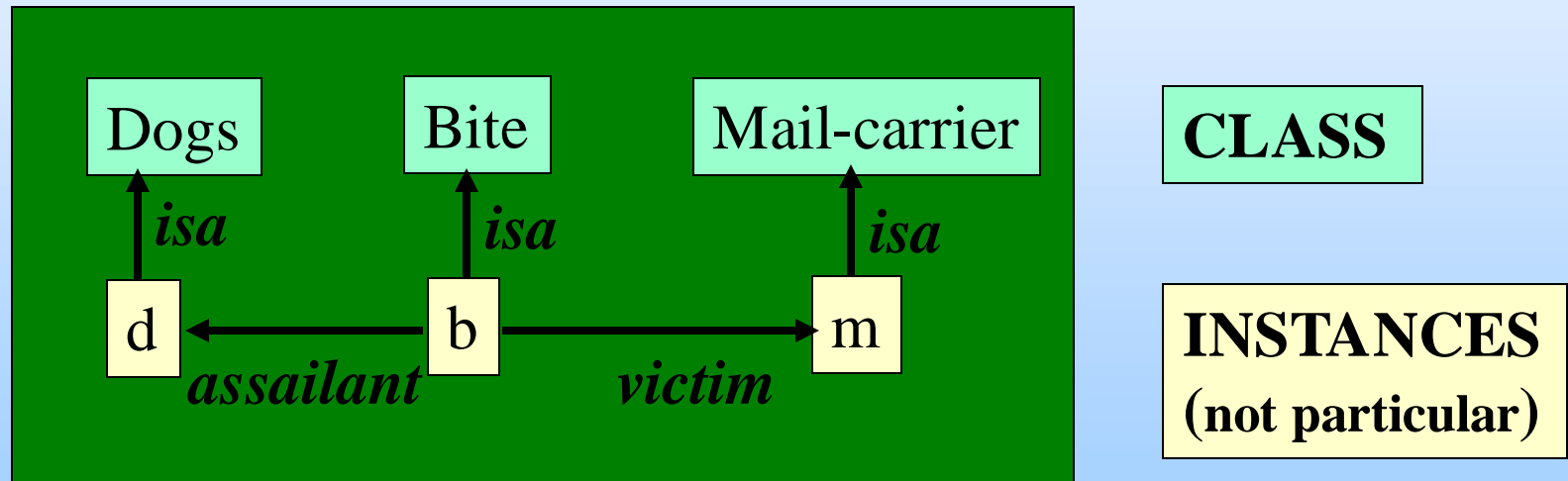
Partitioned Semantic Nets

To represent simple **quantified expressions** in semantic nets.

Partition the semantic net into a hierarchical set of spaces.

‘**space**’ corresponds to the scope of 1 or more variables.

Ex: **The dog bit the mail carrier.**

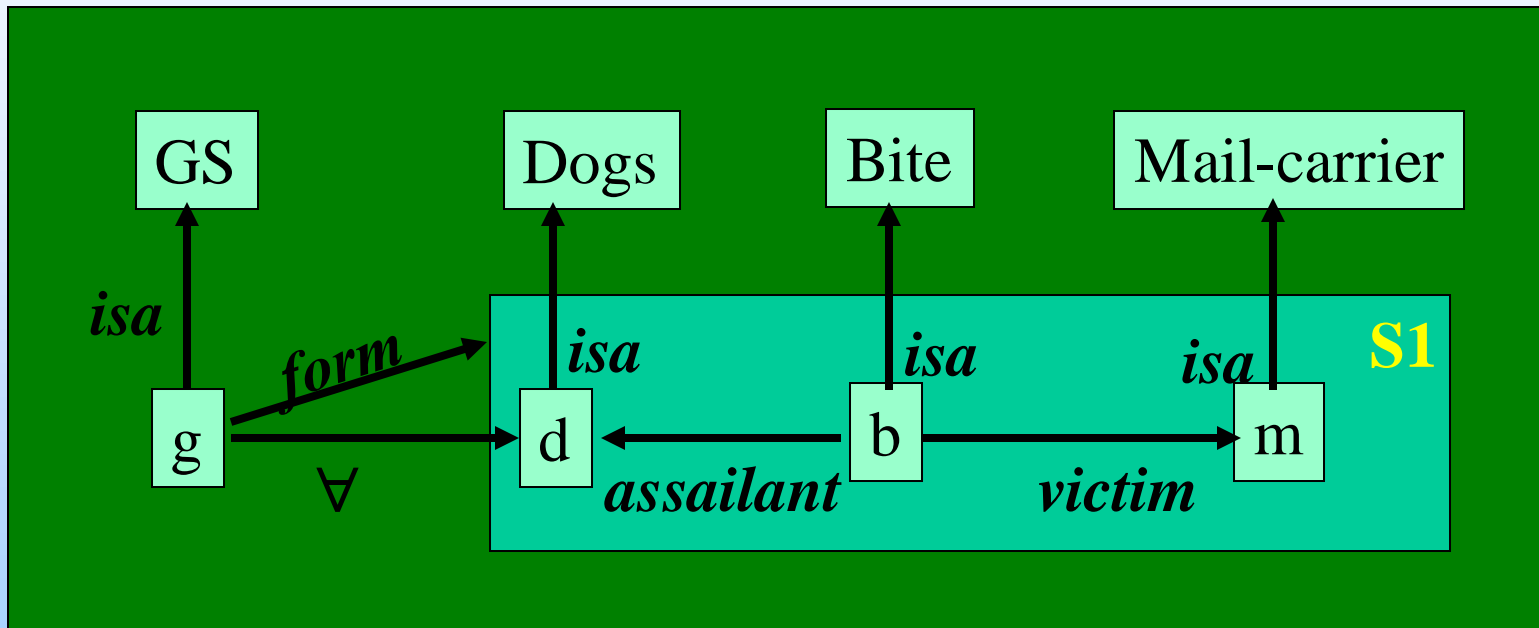


Fact,

represented by a single net with no partitioning.

To represent the fact, **Every dog has bitten a mail carrier**
in logic, $\forall x: \text{dog}(x) \rightarrow \exists y: \text{Mail-Carrier}(y) \wedge \text{Bite}(x, y)$

encode the scope of universally quantified variable 'x',
using partitioning



d : universally quantified
b, m : existentially quantified.



: the assertion given.

An instance of the special class **GS** of g
general statements about the world.

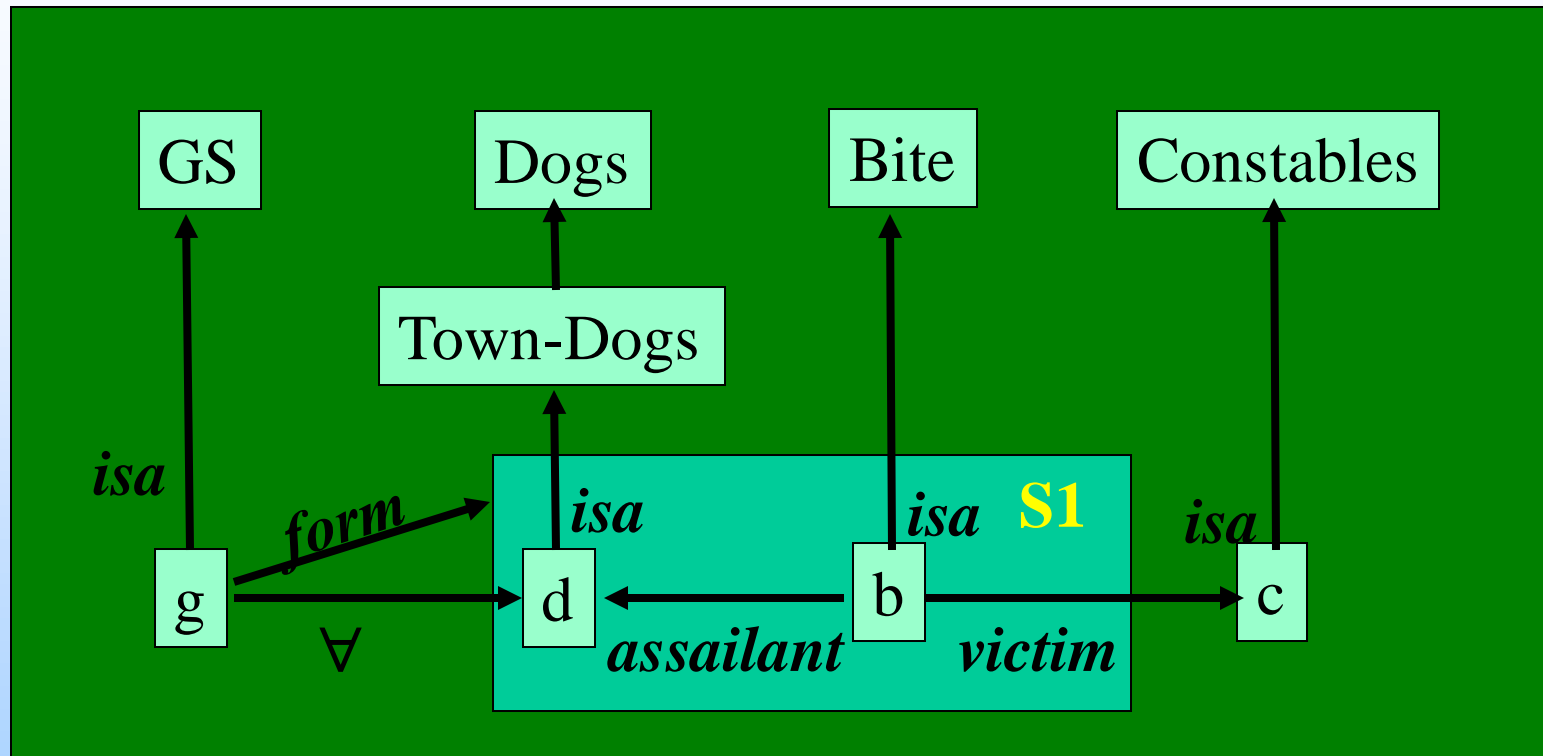
- Every element of GS** has at least 2 attributes,
- a form**, which states the relation that is being asserted.
 - 1/more \forall connections**, one for each of the universally quantified variable.

For every dog ‘**d**’
there exists a biting event ‘**b**’ and
a mail carrier ‘**m**’,
such that d is the assailant of b and m is the victim.



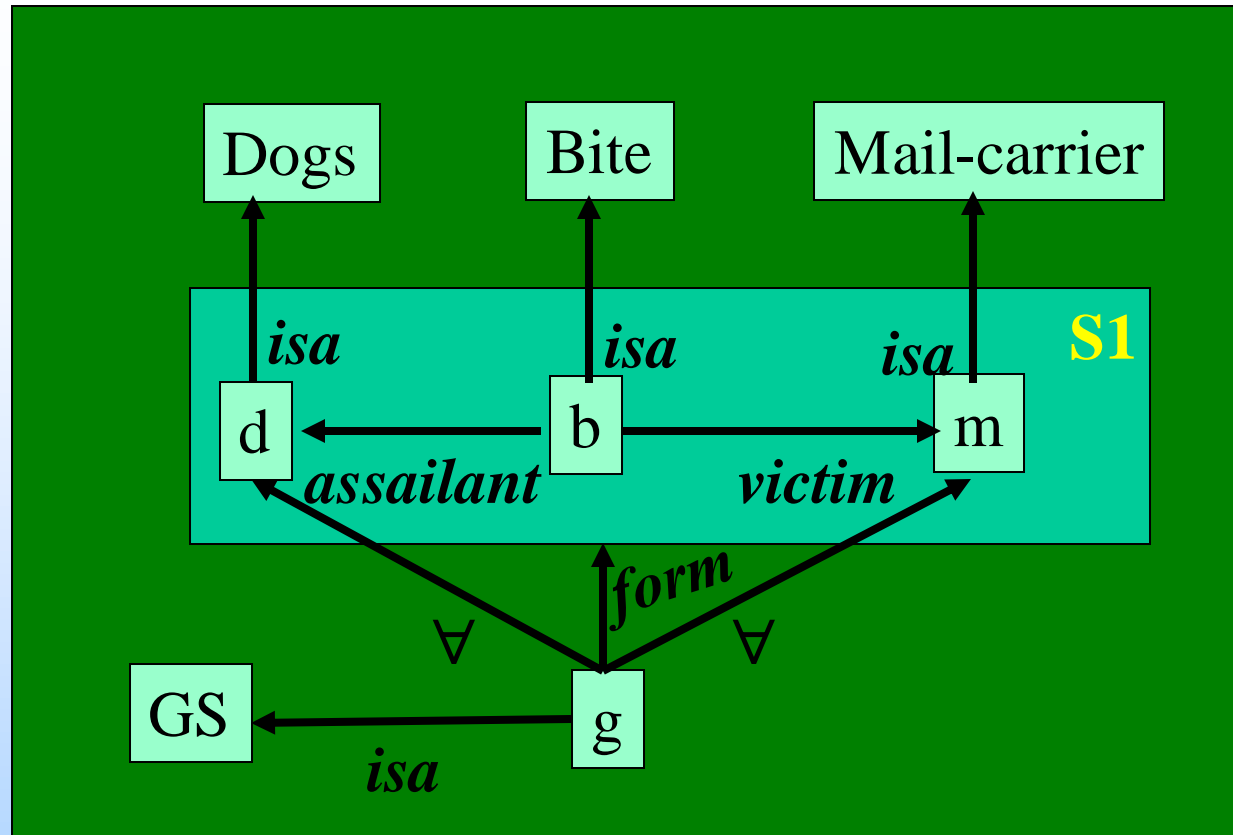
How partitioning makes variable quantification explicit ?

Ex: **Every dog in the town has bitten the constable.**



Node 'c' lies outside the form of the general statement.
Thus it is not viewed as an existentially quantified variable
whose value may depend on the value of d.
-interpreted as a specific entity.

Ex: **Every dog has bitten every mail carrier.**



The spaces of partitioned semantic net are relate to each other by an **inclusion hierarchy**.



When ever a **search process** operates in a partitioned semantic net,
it can explore nodes and arcs in the space from which it
starts and in other spaces that contain the starting point.
but it *cannot go downward*, except in special
circumstances, such as when a **form** arc is being traversed.





References

<https://www.geeksforgeeks.org/semantic-networks-in-artificial-intelligence/>

<https://www.techtarget.com/searchcontentmanagement/definition/semantic-network-knowledge-graph>

<https://intellipaat.com/blog/what-is-semantic-network-in-artificial-intelligence/>





The evolution into frames

Semantic net started

as a way to represent labeled connections among entities.

But, became complex to represent complex problems.

i.e assign more structure to nodes as well as to links.

Semantic net Vs Frame

-no distinction but,

-The more structure the system has, the more likely it is to be termed a frame system.



Properties of Knowledge Representation

*The following properties should be possessed by a **knowledge representation system**.*

Representational Adequacy –

- The ability to **represent** all kinds of knowledge that are needed in that domain.

Inferential Adequacy

- The ability to **manipulate** the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.

Inferential Efficiency

- The ability to **incorporate** into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions.

Acquisitional Efficiency

- The ability to **acquire** new information easily.
 - 1)Direct insertion by person.
 - 2)Program control knowledge acquisition.

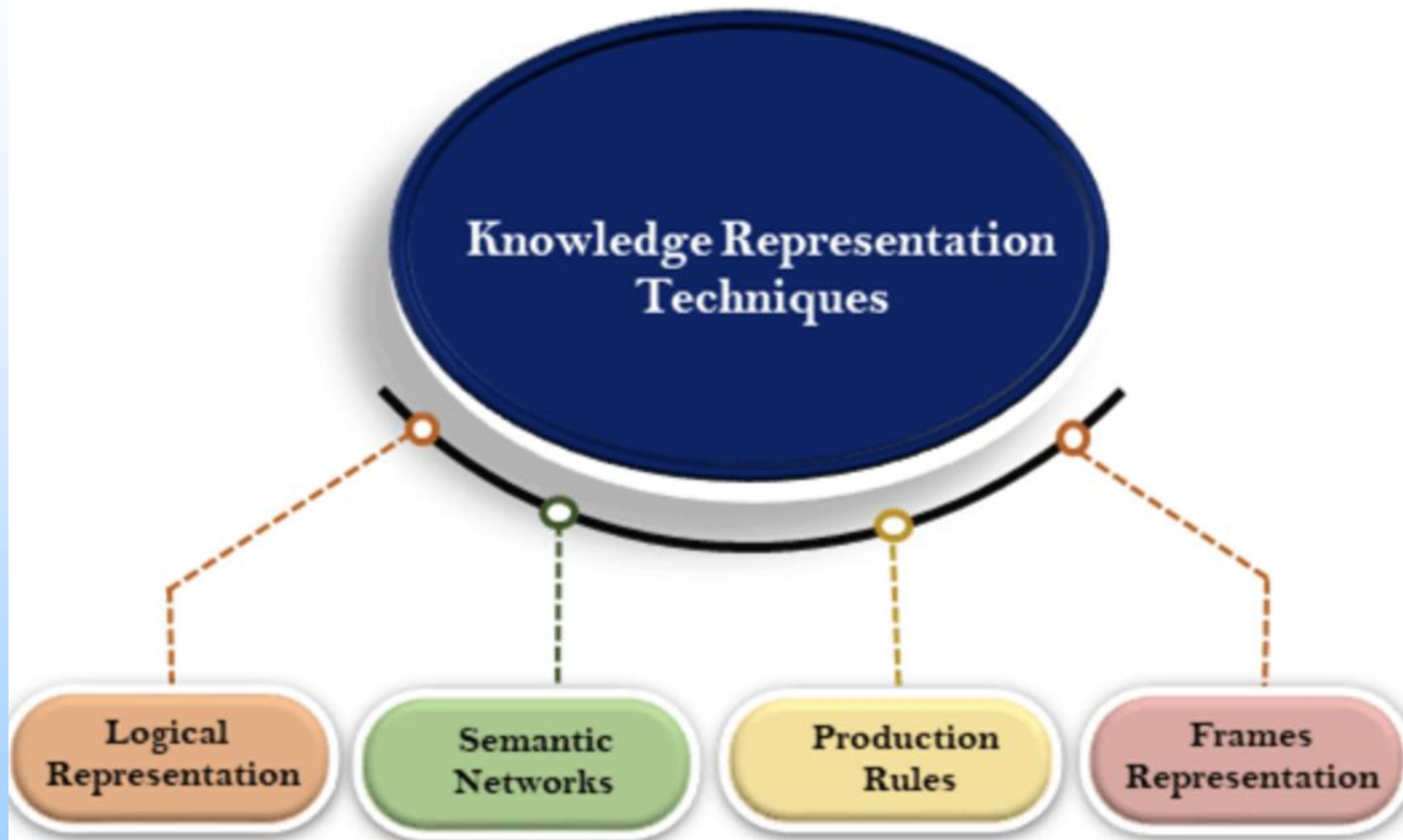


VRSEC

VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE

(Sponsored by Siddhartha Academy of General & Technical Education)

Department of Computer Science and Engineering



VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE
(AUTONOMOUS)

Course Code: 20CS6404A
Course Name: Artificial Intelligence



Dr. M. Sobhana
Associate Professor,
V. R. Siddhartha Engineering College



Frames

Frame: is a collection of attributes(called slots) and associated values (possibly constraints on values) that describe some entity in the world.

Frame system: is a collection of frames that are connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame.

Frame systems can be used to

encode knowledge and support reasoning.





Frames as Sets and Instances

- Each frame represents a **class**(a set) or
an instance(an element of a class)
- *isa* relation: is a **subset** relation.
instance relation: is a **element-of** relation.
- *isa* and *instance* relations have inverse attributes,
subclasses
all-instances





Frames as Sets and Instances

- Because a class represents a set, there are 2 kinds of attributes that can be associated with it.
 - There are attributes **about the set itself**, and
 - there are attributes **that are to be inherited by each element of the set**.





Example of a frame for a book

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152



Person

CLASS

Mammal
6,000,000,000
Right

Adult-Male

CLASS

person
2,000,000,000
5-10

ML-Baseball-Player

isa:
cardinality:

Adult-Male
624
61

CLASS

Default values

} attributes *about the set itself*

} attributes *that are to be inherited by each element of the set*

team:
*uniform-color:

Fielder

CLASS

ML-Baseball-Player
376
.262

Pee-Wee-Reese

instance:

Fielder
5-10
Right
309

INSTANCE

team:
uniform-color:

Brooklyn-Dodgers
Blue

ML-Baseball-team

CLASS

Team
26
24

Brooklyn-Dodgers

INSTANCE

ML-Baseball-Team
24
Leo-Durocher
{Pee-Wee-Reese,...}



Classes and Metaclasses

regular classes: whose elements are individual entities.

metaclasses: are special classes, whose elements are themselves classes

A **class** is

- an element(instance) of some class or classes*
- a subclass of one or more classes.*

In all the frame systems

all classes are *instances* of the metaclass **Class**.



Representing the class of all teams as a metaclass

Class

instance	:	Class
isa	:	Class
*cardinality	:	

Team

instance	:	Class
isa	:	Class
cardinality	:	{ the number of teams that exist }
*team-size	:	{ each team has a size }

ML-Baseball-Team

instance	:	Class
isa	:	Team
cardinality	:	26 { the number of baseball teams that exist }
*team-size	:	24 { default 24 players on a team }
*manager	:	

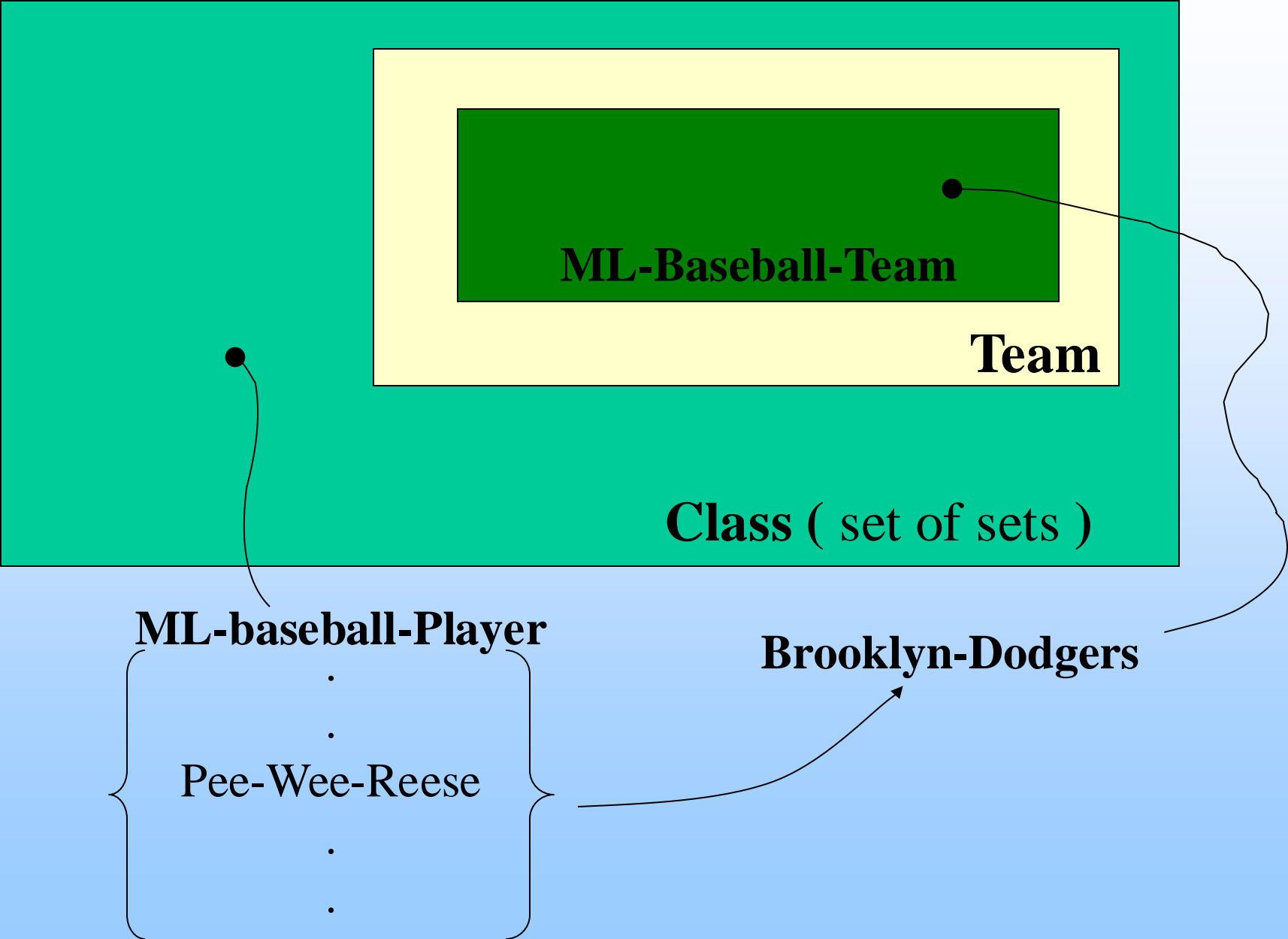
Brooklyn-Dodgers

instance	:	ML-Baseball-team
isa	:	ML-Baseball-player
team-size	:	24
manager	:	Leo-Durocher
*uniform-color	:	Blue

Pee-Wee-Reese

instance	:	Brooklyn-Dodgers
instance	:	Fielder
uniform-color	:	Blue
batting-average	:	.309

Graphic view of the same classes...





Other ways of relating classes to each other

Upto now, classes can be related to each other in 2 ways

1. class1 can be a **subset** of class2
2. if class2 is a metaclass, then
class1 can be an **instance** of class2.





Other ways of relating classes to each other

Other relationships...

Mutually-disjoint-with:

which relates a class to one or more other classes that are guaranteed to have no elements in common with it.

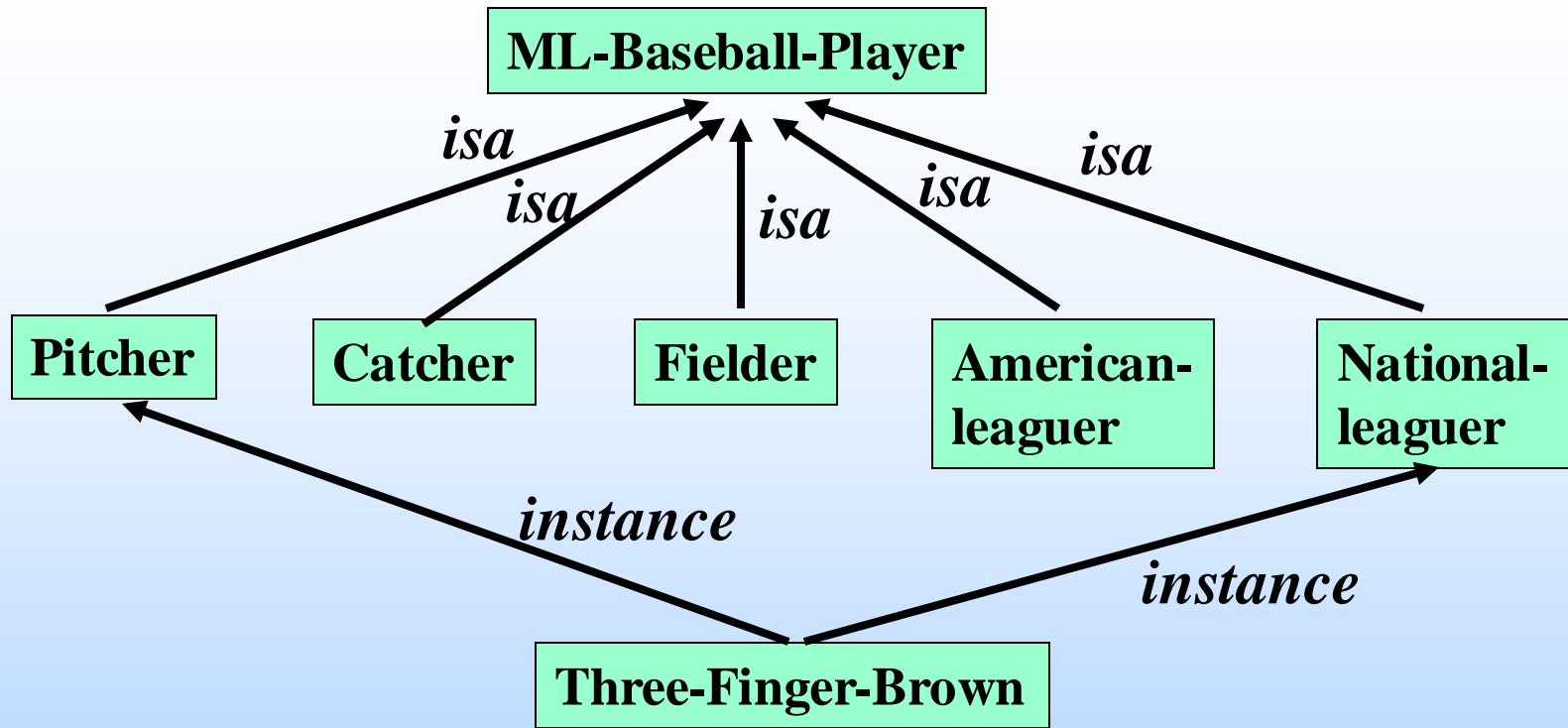
Is-covered-by:

which relates a class to a set of subclasses, the union of which is equal to it.

If a class is-covered-by a set S of mutually disjoint classes, then S is called a **partition** of the class.



Representing relationships among classes



ML-Baseball-Player

is-covered-by : { pitcher, Catcher, Fielder },
{ American-Leaguer, National-Leaguer }



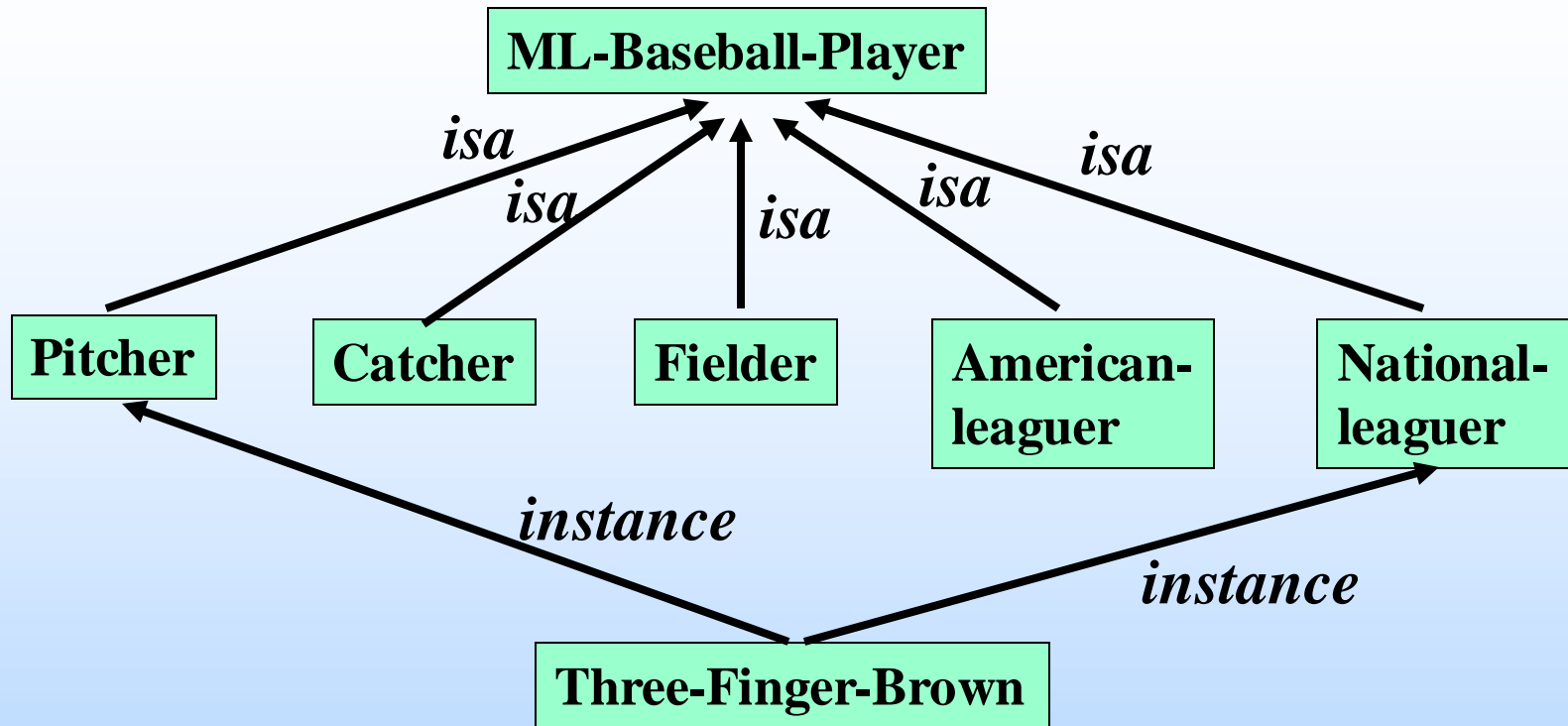
VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE
(AUTONOMOUS)

Course Code: 20CS6404A
Course Name: Artificial Intelligence



Dr. M. Sobhana
Associate Professor,
V. R. Siddhartha Engineering College

Representing relationships among classes



Pitcher

isa : ML-Baseball-Player
mutually disjoint-with : { Catcher, Fielder }



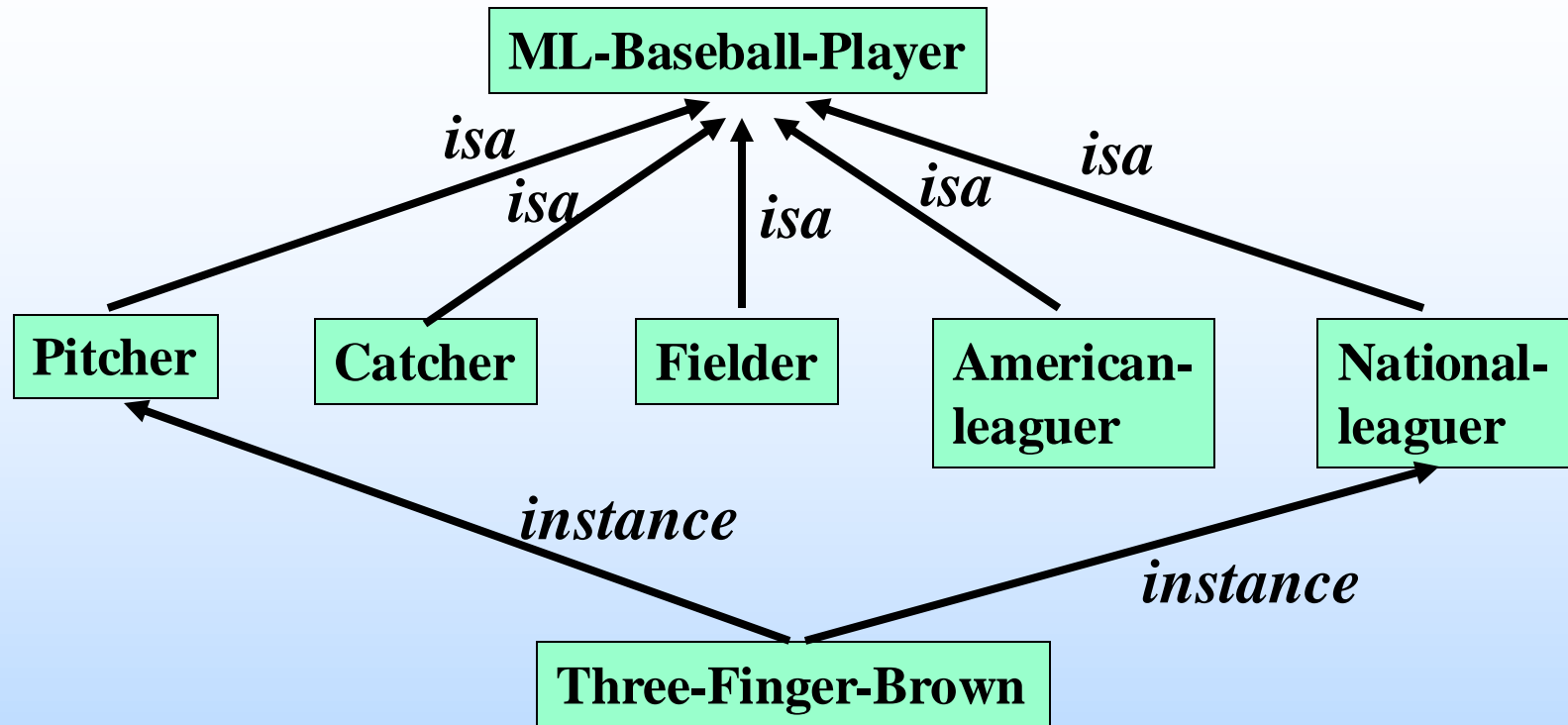
VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE
(AUTONOMOUS)

Course Code: 20CS6404A
Course Name: Artificial Intelligence



Dr. M. Sobhana
Associate Professor,
V. R. Siddhartha Engineering College

Representing relationships among classes

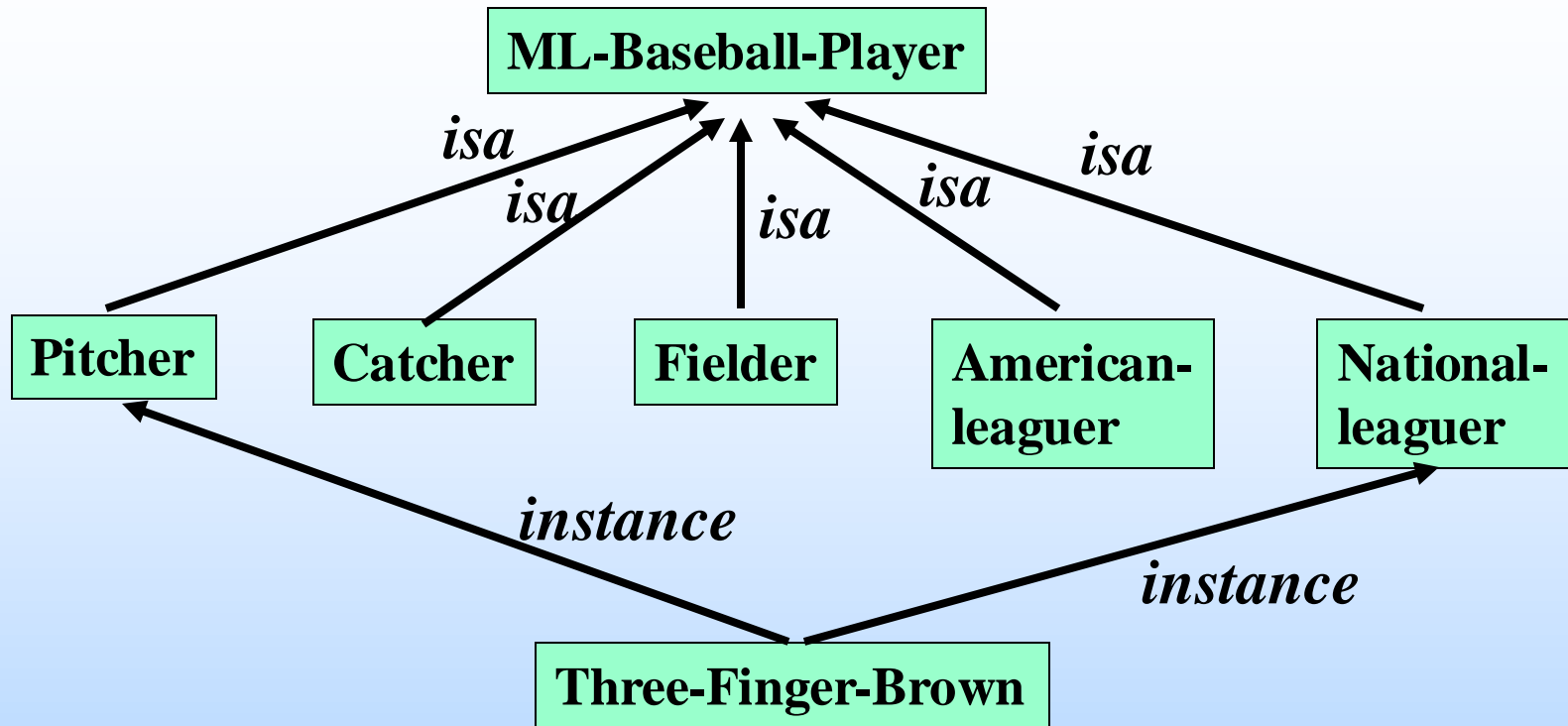


Catcher

isa : ML-Baseball-Player
mutually disjoint-with : { Pitcher, Fielder }



Representing relationships among classes



Fielder

isa : ML-Baseball-Player
mutually disjoint-with : { Pitcher, Catcher }



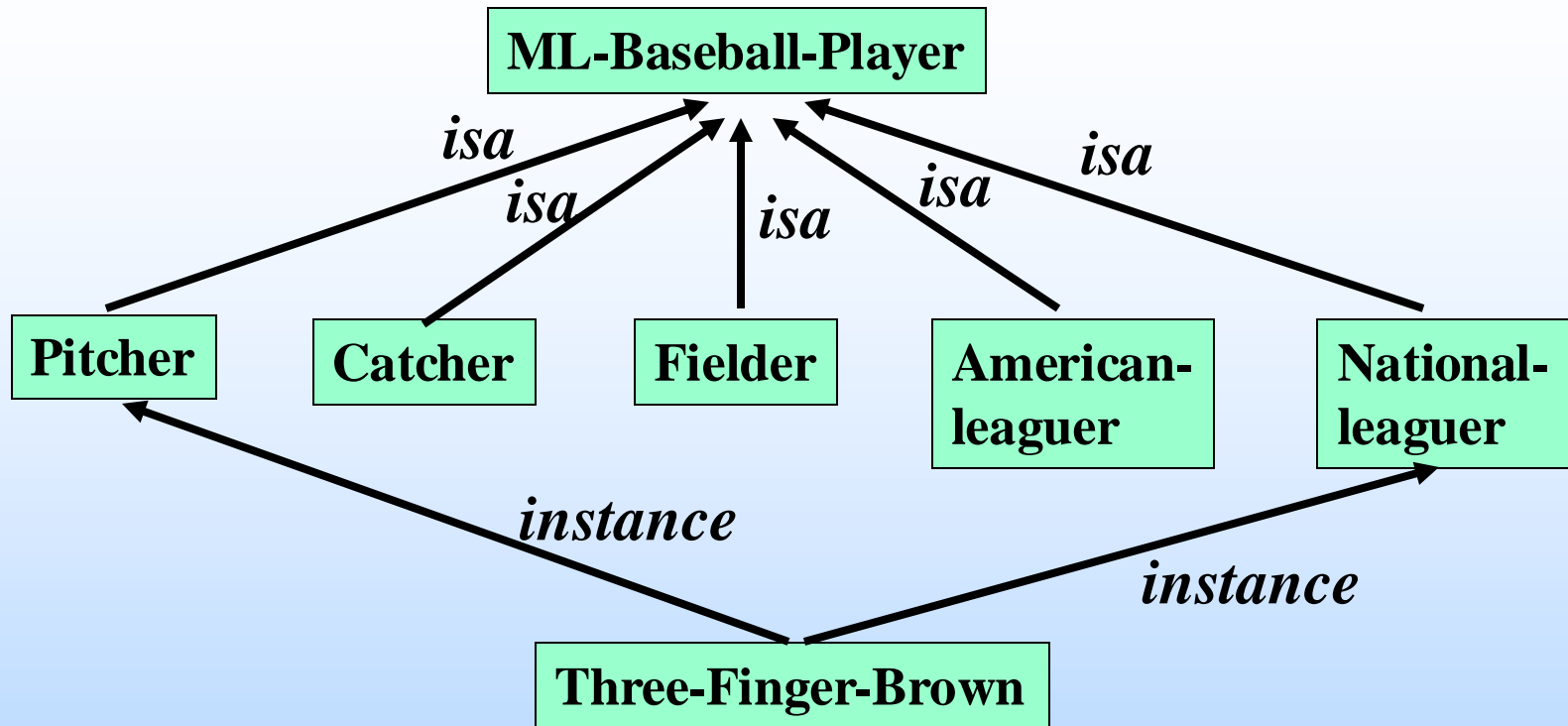
VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE
(AUTONOMOUS)

Course Code: 20CS6404A
Course Name: Artificial Intelligence



Dr. M. Sobhana
Associate Professor,
V. R. Siddhartha Engineering College

Representing relationships among classes



American-leaguer

isa

:

ML-Baseball-Player

mutually disjoint-with

:

{National-Leaguer}



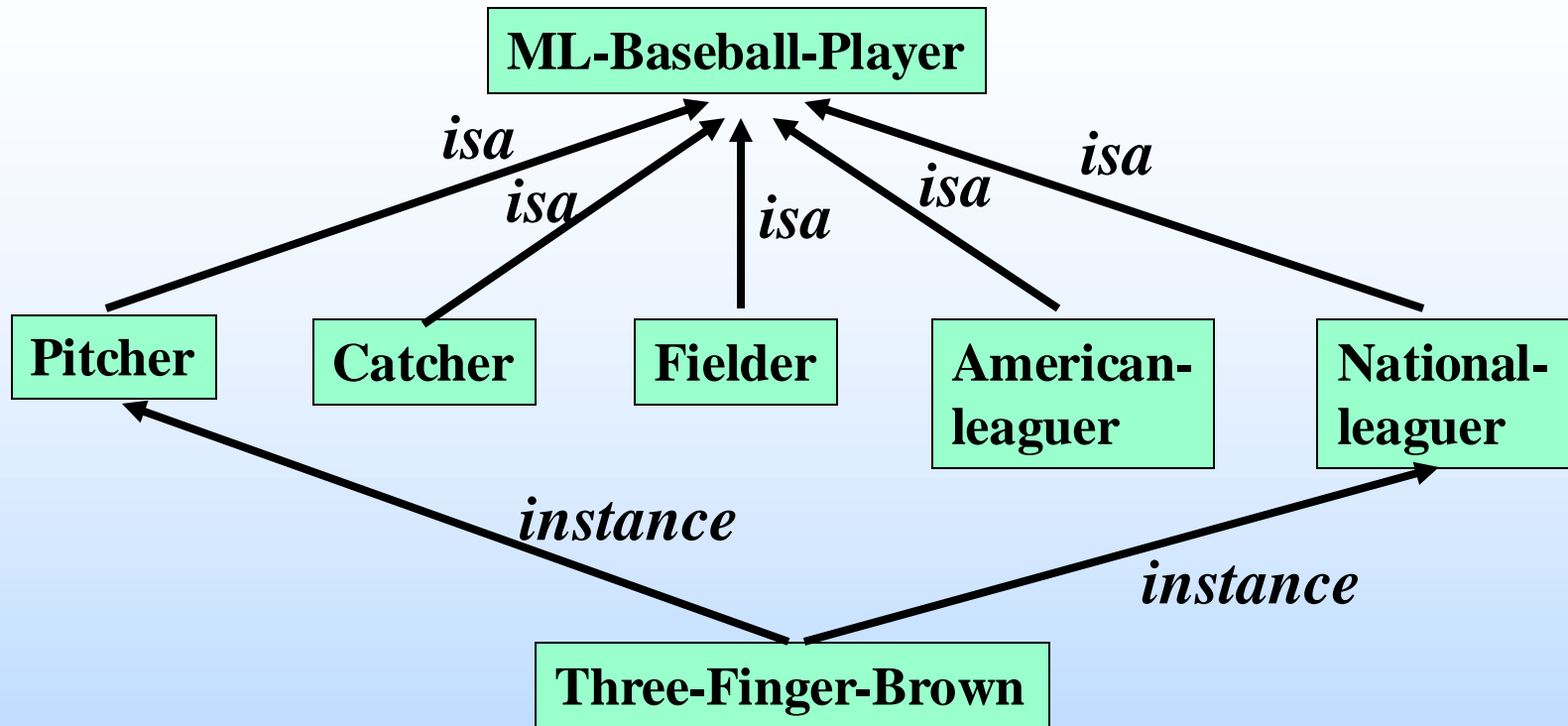
**VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE**
(AUTONOMOUS)

Course Code: 20CS6404A
Course Name: Artificial Intelligence



Dr. M. Sobhana
Associate Professor,
V. R. Siddhartha Engineering College

Representing relationships among classes



National-Leaguer

isa

:

ML-Baseball-Player

mutually disjoint-with

:

{ American-Leaguer }



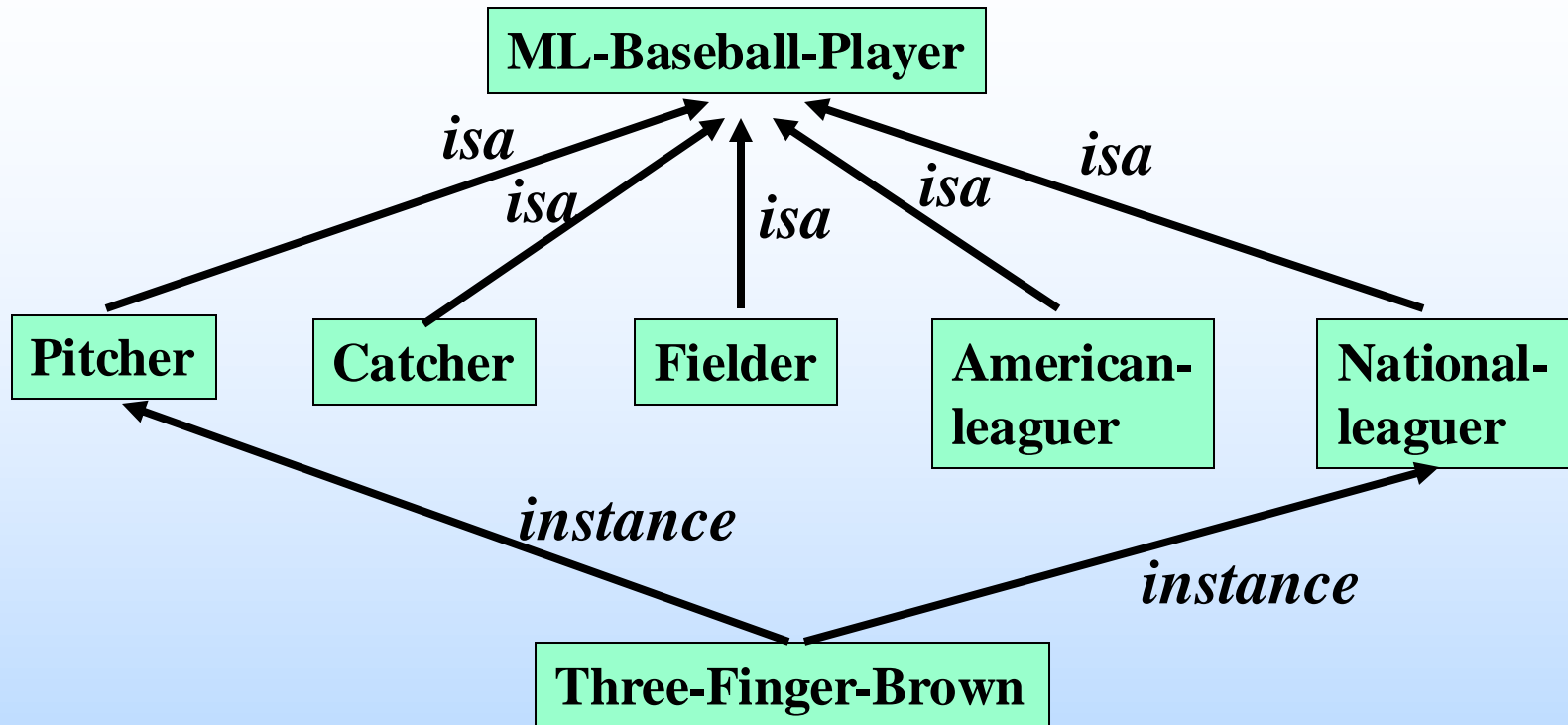
VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE
(AUTONOMOUS)

Course Code: 20CS6404A
Course Name: Artificial Intelligence



Dr. M. Sobhana
Associate Professor,
V. R. Siddhartha Engineering College

Representing relationships among classes



Three-Finger-Brown

instance : Pitcher
instance : National-Leaguer





Key Takeaways:

- Frames are a structured knowledge representation technique used in AI to model concepts, objects, and their attributes.
- They consist of slots (attributes) and fillers (values) that organize information.
- Frames can have frames as their fillers, forming hierarchical structures that facilitate inheritance.
- Inheritance simplifies knowledge representation by reducing redundancy, ensuring consistency, and supporting extensibility.





Key Takeaways:

- Frames are used in various AI applications, including expert systems, natural language processing, and robotics.
- Challenges include scalability issues, complex relationships, and inefficiency in retrieval.
- Solutions involve careful planning, organization, and the use of techniques like property chains.





Slots as Full-Fledged objects

Sets of objects and individual objects are described both in terms of **attributes** and **values**.

Attributes are represented as **slots** attached to Frames.

But, there are several reasons that, to represent attributes explicitly and describe their properties.





Some of the **properties** we would like to be able to represent and use in reasoning include...

- The classes to which the attribute can be attached.
- Constraints on either the type or the value of the attribute.
- A value that all instances of a class must have by the definition of the class.
- A default value for the attribute.
- Rules for inheriting values for the attribute.
- Rules for computing a value separately from inheritance.
- An inverse attribute.
- Whether the slot is single-valued or multivalued.





To represent these attributes of attributes, we need to describe attributes(slots) as frames.

These frames will be organized into an **isa hierarchy**, and that hierarchy can then be used to support inheritance of values for attributes of slots.



Representing Slots as Frames

Slot

<i>isa</i>	:	Class
<i>instance</i>	:	Class
<i>*domain</i>	:	
<i>*range</i>	:	
<i>*range-constraint</i>	:	
<i>*definition</i>	:	
<i>*default</i>	:	
<i>*transfers-through</i>	:	
<i>*to-compute</i>	:	
<i>*inverse</i>	:	
<i>*single-valued</i>	:	

manager

<i>instance</i>	:	Slot
<i>domain</i>	:	ML-Baseball-Team
<i>range</i>	:	Person
<i>range-constraint</i>	:	x(baseball-experience x.manager)
<i>default</i>	:	
<i>inverse</i>	:	manager-of
<i>single-valued</i>	:	TRUE



Since a slot is a set, the set of all slots, which we will call ‘***Slot***’, is a **metaclass**.

A slot is a relation, has a **domain** and **range**.

- Represent the domain in the slot labeled ***domain***.
- Braking up the representation of range into 2 parts:
range, gives the class of which elements of the range must be elements.
Range-constraint, contains a logical expression that further constrains the range to be elements of range that also satisfy the constraint.



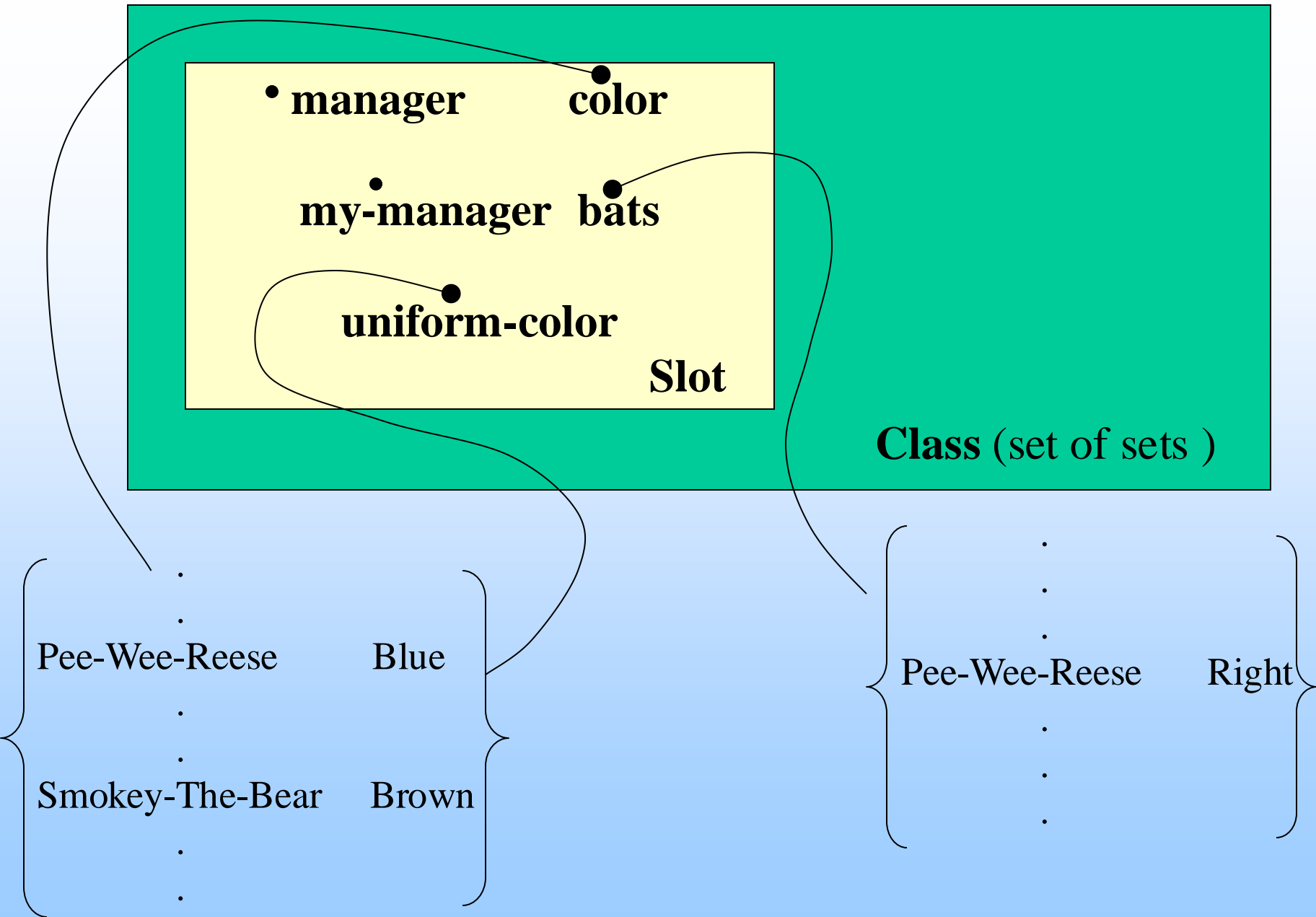


What other slots do...?

<i>definition</i>	:	it must be propagated to all instances of the slot.
<i>default</i>	:	value is inherited to all instances of the slot unless there is an overriding value.
<i>transfers-thorough</i> :		lists other slots from which values for this slot can be derived through inheritance.
<i>to-compute</i>	:	contains a procedure for deriving its value.
<i>inverse</i>	:	contains the inverse of the slot.
<i>single-valued</i>	:	used to mark the special cases in which the slot is a function and so can have only one value.



my-manager	<i>instance</i>	:	Slot
	<i>domain</i>	:	ML-Baseball-Team
	<i>range</i>	:	Person
	<i>range-constraint</i>	:	$\lambda x(\text{baseball-experience } x.\text{my-manager})$
	<i>to-compute</i>	:	$\lambda x(x.\text{team}).\text{manager}$
	<i>single-valued</i>	:	TRUE
color	<i>instance</i>	:	Slot
	<i>domain</i>	:	Physical-Object
	<i>range</i>	:	Color-Set
	<i>transfers-through</i>	:	top-level-part-of
	<i>visual-sailence</i>	:	high
	<i>single-valued</i>	:	FALSE
uniform-color	<i>instance</i>	:	Slot
	<i>domain</i>	:	color
	<i>range</i>	:	team-player
	<i>transfers-through</i>	:	Color-set
	<i>range-constraint</i>	:	not pink
	<i>visual-sailence</i>	:	high
bats	<i>instance</i>	:	Slot
	<i>domain</i>	:	ML-Baseball-Team
	<i>range</i>	:	{ Left, Right, Switch }
	<i>to-compute</i>	:	$\lambda x x.\text{handed}$
	<i>single-valued</i>	:	TRUE





We assume that the frame system **interpreter** knows how to reason with all of these slots of slots as part of its built-in reasoning capability.

We assume that it is capable of performing the following reasoning actions:





Associating defaults with slots

earlier we listed 2 defaults for the *batting-average* slot
one, associated with **ML-Baseball-Player** &
one, associated with **Fielder**.

They can be represented correctly, by

creating a specialization of *batting-average* that can be
associated with a specialization of ML-Baseball-Player to
represent the more specific information.



batting-average

<i>instance</i>	:	Slot
<i>domain</i>	:	ML-Baseball-Player
<i>range</i>	:	Number
<i>range-constraint</i>	:	$\lambda x(0 \leq x.\text{range-constraint} \leq 1)$
<i>default</i>	:	.252
<i>single-valued</i>	:	TRUE

fielder-batting-average

<i>instance</i>	:	Slot
<i>domain</i>	:	batting-average
<i>range</i>	:	Number
<i>range-constraint</i>	:	$\lambda x(0 \leq x.\text{range-constraint} \leq 1)$
<i>default</i>	:	.262
<i>single-valued</i>	:	TRUE



Although this model of slots is
simple and
it is internally consistent,
it is not easy to use.

Therefore, we introduce some notational **shorthand** that allows the *four most important properties of a slot* (**domain**, **range**, **definition**, **default**) to be defined implicitly by how the slot is used in the definitions of the classes in its domain.





We describe...

- domain** implicitly to be the class where the slot appears.
- range** and **range-constraints** with the clause **MUST BE** as the value of inherited class.
- definition & default**, as they appear.

Ex:

ML-Baseball-Player

bats : MUST BE { *Left, Right, Switch* }



Slot values as objects

Representing slot-values...

John		
	height :	72
Bill		
	height :	

All we know that **‘John is taller than Bill’**

with **lambda notation**

John		
	height :	72; $\lambda x(x.\text{height} > \text{Bill}.\text{height})$
Bill		
	height :	$\lambda x(x.\text{height} < \text{John}.\text{height})$

Inheritance revisited

To support flexible representations of knowledge about the world, it is necessary to allow the hierarchy(isa) to be an arbitrary **directed acyclic graph**. (rather than as a tree).

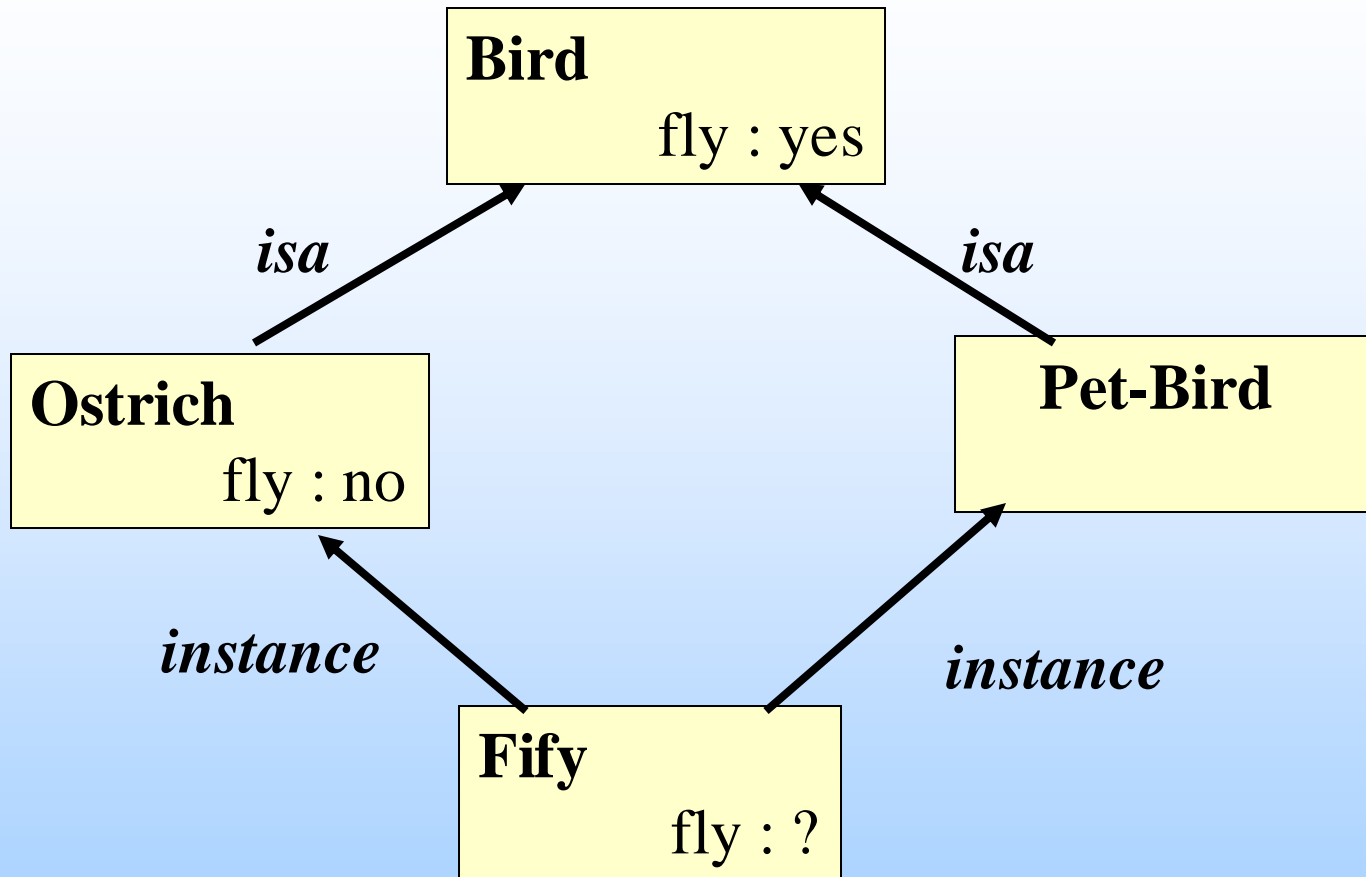
Hierarchies that are not trees are called *tangled hierarchies*. Which require a new inheritance algorithm.

Algorithm for

- inheriting values for single-valued slots in a tangled-hierarchy. (discussing)
- inheriting multivalued slots. (exercise)

Example: CASE 1

Tangled hierarchies

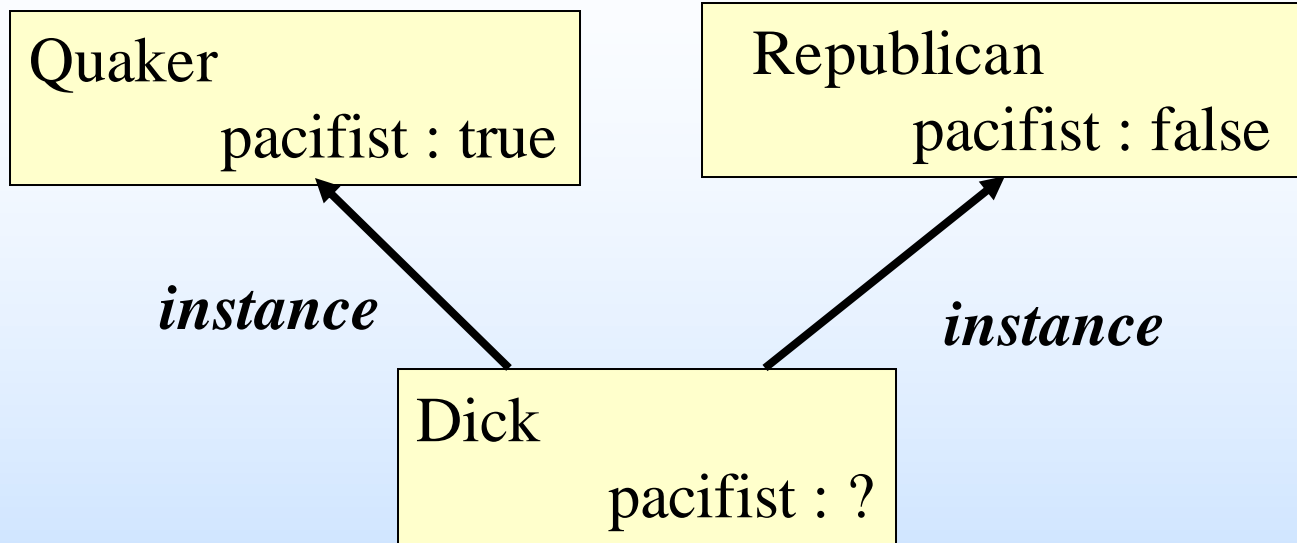


We want to decide whether **'fify'** can fly?

Ans: *no*

Example: CASE 2

Tangled hierarchies



Determining whether **‘Dick’** is pacifist ?

Ans: no answer (since, Ambiguity)

One possible basis for new inheritance algorithm is *‘path length’*.

Algorithm is implemented by executing a **breadth-first search**, *starting with the frame for which a slot value is needed*. Follow its *instance* links, then follow *isa* links upward. If a path produces a value it can be terminated, as can all other paths once their length exceeds that of the successful path.

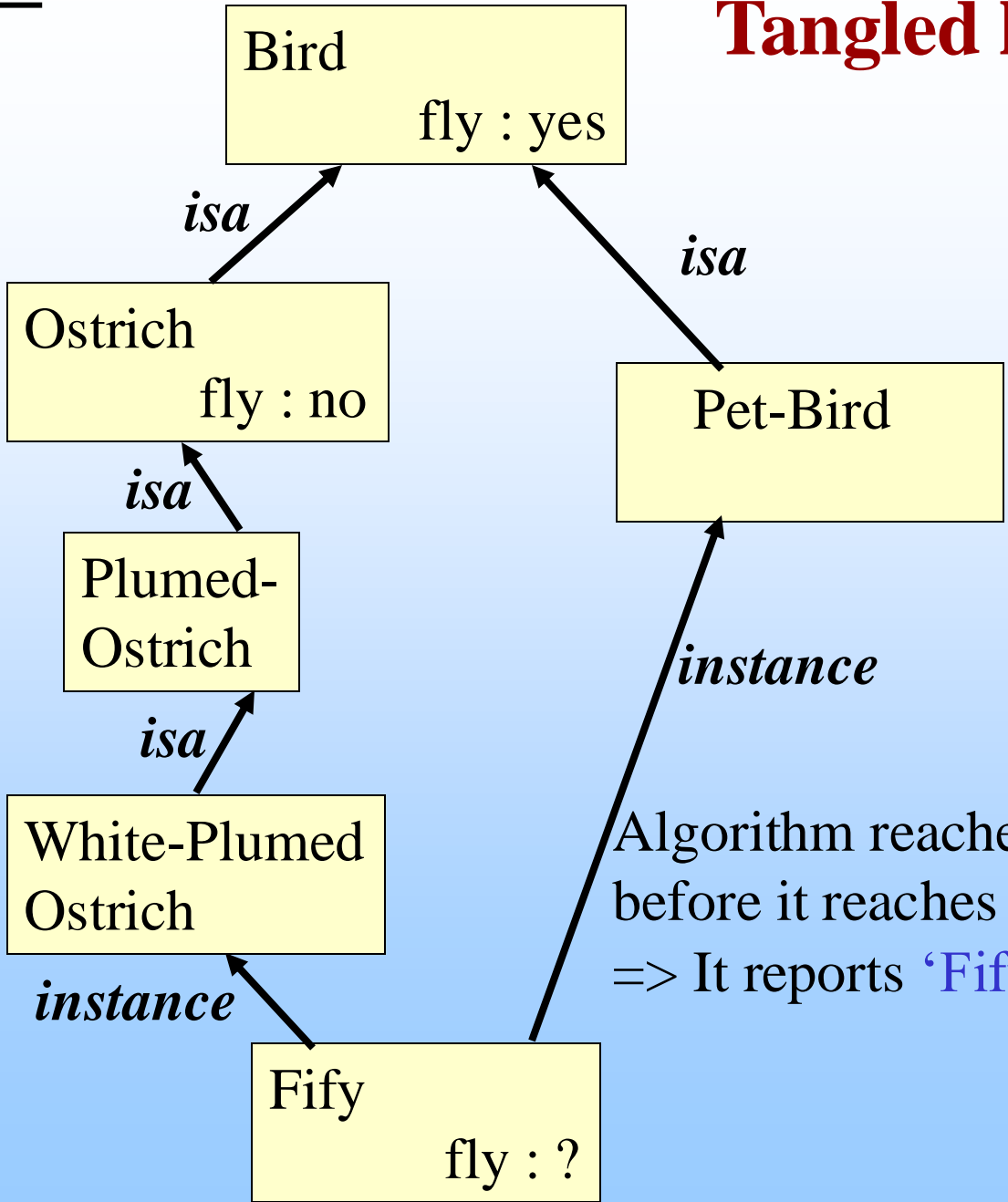
This algorithm

For CASE 1: reports no.

For CASE 2: reports contradiction.

Example: CASE 3

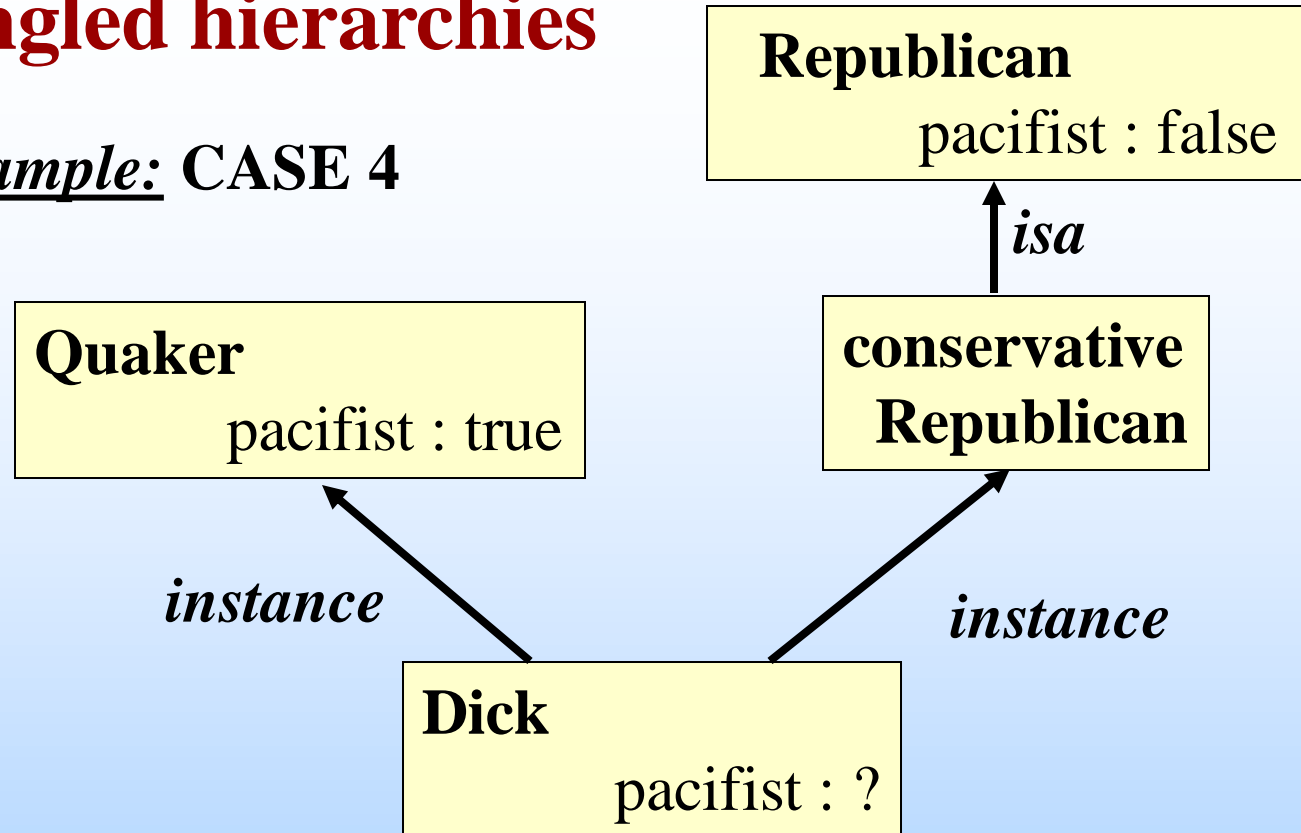
Tangled hierarchies



Algorithm reaches **Bird**(via Pet-Bird)
before it reaches **Ostrich**
=> It reports 'Fify' can fly.

Tangled hierarchies

Example: CASE 4



Algorithm reaches '**Quaker**' and stops without noticing a contradiction

Path length doesn't always correspond to the level of generality of a class. Sometimes it corresponds to is the degree of elaboration of classes in the knowledge base.

Therefore, Base inheritance algorithm *not on the path length* but on the notion of *inferential distance*, which can be defined as follows.

Class1 is closer to **Class2** than to **Class3** if and only if **Class1** has an inference path through **Class2** to **Class3**

defining the result of inference as ...

The set of competing values for a Slot **S** in a frame **F** contains all those values that

- can be derived from some frame **X** that is above **F** in the *isa* hierarchy.
- Are not contradicted by some frame **Y** that has a shorter inferential distance to **F** than **X** does.

Using this definition,

looking at the example **CASE1**

Ostrich has a shorter inferential distance to **Fifi** than **Bird** does, so we get the single answer ‘**no**’.

looking at the example **CASE2**

we get 2 answers, and neither is closer to Dick than the other, so we correctly identify a **contradiction**.

looking at the example **CASE3**

we get 2 answers, but again **Ostrich** has a shorter inferential distance to **Fifi** than **Bird** does.

(the way we have defined inferential distance is that as long as Ostrich is a subclass of Bird, it will be closer to all its instances than Bird is, no matter how many other classes are added to the system.)

looking at the example **CASE4**

we get 2 answers and again neither is closer to **Dick** than the other.

Algorithm: property inheritance

to retrieve a value **V** for slot **S** of an instance **F** do:

1. Set CANDIDATES to empty.
2. Do breadth-first or depth-first search up the isa hierarchy from F, following all instance and isa links. At each step, see if a value for S or one of its generalizations is stored.
 - A) if a value is found, add it to CANDIDATES and terminate that branch of the search.
 - B) if no value is found but there are instance or isa links upward, follow them.
 - C) otherwise, terminate the branch.
3. For each element C of CANDIDATES do:
 - a) see if there is any other element of CANDIDATES that was derived from class closer to F than the class from which C came.
 - B) if there is, remove C from CANDIDATES.
4. Check the cardinality of CANDIDATES.
 - A) if it is 0, then report that no value was found.
 - B) if it is 1, then return the single element of CANDIDATES as V.
 - c) if it is greater than 1, report a contradiction.



VRSEC

VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE

(Sponsored by Siddhartha Academy of General & Technical Education)

Department of Computer Science and Engineering



The End



VELAGAPUDI RAMAKRISHNA
SIDDHARTHA ENGINEERING COLLEGE
(AUTONOMOUS)

Course Code: 20CS6404A
Course Name: Artificial Intelligence



Dr. M. Sobhana
Associate Professor,
V. R. Siddhartha Engineering College