

# Video Databases

---

## Introduction

---

Retrieval requests may take one of two forms.

- **Retrieving a Specified Video:** User specifies the video he wants to see, e.g. “Show me *The Sound of Music*”.
- **Identifying and Retrieving Video Segments:** User might express a query such as *Find all videos in which John Wayne appears with a gun*. This query requires that we:
  - identify the movies in which John Wayne appears with a gun and
  - identify the segments within those movies in which John Wayne appears with a gun.
- Once we can organize the content of a single video, we can organize the content of a set of videos.

## Organizing Content of a Single Video

---

We must ask ourselves the following questions:

1. *Which* aspects of the video are likely to be of interest to the users who access the video archive ?
2. *How* can these aspects of the video be stored efficiently, so as to minimize the time needed to answer user queries ?
3. What should *Query Languages* for video data look like and how should the relational model of data be extended to handle video information ?
4. Can the *Content Extraction* process be automated, and if so, how can the reliability of such content extraction techniques be taken into account when processing queries?

## Video Content: Which Aspects of a Video To Store?

---

Example: An 8-hour, one day lecture of a short course given by a professor on the topic *Multimedia Databases*. In this case, the video contains a set of “items of interest.” These items of interest could include:

1. *People* such as the professor, any guest lecturer (or lecturers ) who speak at selected times in the course, and any students who might ask questions and/or distinguish themselves in other ways; For instance, **Prof. Felix** might be one such person, while **Erica** might be a student.
2. *Activities* that occur in the class such as *lecturing* (on a particular topic, by a particular individual), or *questioning* (by a particular student), or *answering* a question posed by a particular student. Other activities could involve general group discussions, and/or coffee breaks.

In addition, activities have attributes, e.g. *lecturing(quadtrees, Prof. Felix)* indicates an activity involving Prof. Felix lecturing on quadtrees, and *questioning(Erica, Prof. Felix)* indicating that Prof. Felix was questioned by Erica.

## Movie Example

---

- Consider the movie, *Sound of Music*.
- Items of interest include:
  1. *People* such as Maria, Count Von Trapp, and others;
  2. *Inanimate objects* such as the piano in Count Von Trapp's house;
  3. *Animate objects* such as the ducks and birds in the pond;
  4. *Activities* such as singing and dancing, with their associated list of attributes. For example, the activity *singing* may have two attributes:
    - (a) **Singer** specifying which person is singing and
    - (b) **Song** specifying the name of the song and
- Certain common characteristics occur. Given any frame  $f$  in the video, the frame  $f$  has a set of associated objects and associated activities.
- Objects/activities have certain properties, and these properties may vary from one frame to another.
- **Creating a video database means we should be able to index all these associations.**

## Properties

---

- **Property:** Consists of a pair ( $\text{pname}$ ,  $\text{Values}$ ) where:
  - $\text{pname}$  is the *Name* of the property,
  - $\text{Values}$  is a set.
- **Property Instance:** An expression of the form  $\text{pname} = v$  where  $v \in \text{Values}$ .
- Example properties:
  1. ( $\text{height}$ ,  $\mathbf{R}^+$ ) consists of the “height” property with real-values;
  2. ( $\text{primarycolors}$ ,  $\{\text{red}, \text{green}, \text{blue}\}$ ) consists of a property called *primarycolors* with values red, green, blue.

## Object Scheme

---

- **Object Scheme:** A pair  $(fd, fi)$  where:
  1.  $fd$  is a set of *frame-dependent* properties;
  2.  $fi$  is a set of *frame-independent* properties.
  3.  $fi$  and  $fd$  are disjoint sets.

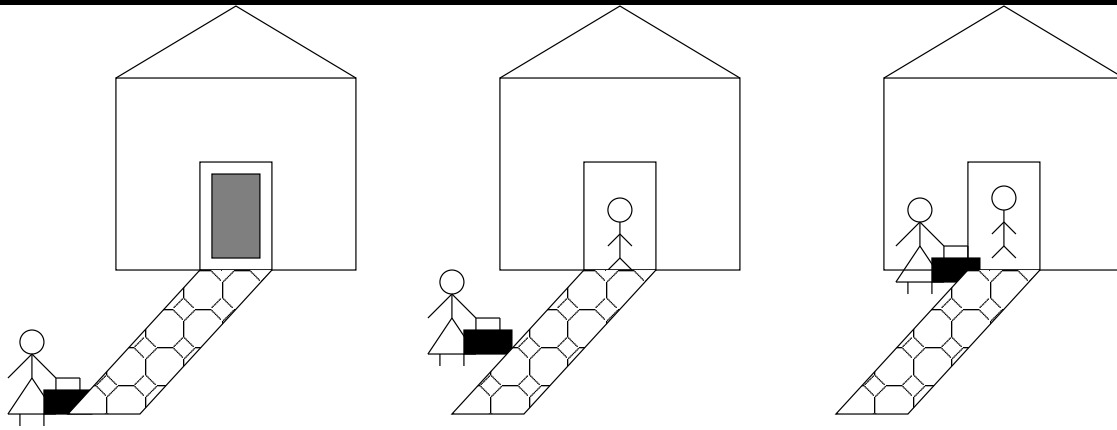
$fd$  and  $fi$  are disjoint.
- If  $(pname, Values)$  is a property in  $fd$ , then this means that the property named  $pname$  may assume different instances, depending upon the video frame being considered. E.g. the property *shirtcolor* varies from frame to frame.
- **Object Instance:** An *Object Instance* is a triple  $(oid, os, ip)$  where:
  1.  $oid$  is a string called the object-id and
  2.  $os = (fd, fi)$  is an object structure and
  3.  $ip$  is a set of statements such that:
    - (a) for each property  $(pname, Values)$  in  $fi$ ,  $ip$  contains *at most* one property instance of  $(pname, Values)$  and;
    - (b) for each property  $(pname, Values)$  in  $fd$ , and each frame  $f$  of the video,  $ip$  contains *at most* one property instance of  $(pname, Values)$  – this property instance is denoted by the expression  $pname = v \text{ IN } f$ .

## Example

---

- Surveillance video of 5 frames.
- Show surveillance video of the house of Denis Dopeman.
- Frame 1: We see Jane Shady at the path leading to Mr. Dopeman's door. She is carrying a briefcase.
- Frame 2: She is halfway on the path to the door. Door opens. Mr. Dopeman appears at the door.
- Frame 3: Jane Shady and Denis Dopeman are ext to each other at the door; Jane Shady is still carrying the briefcase.
- Frame 4: Jane Shady is walking back, and Denis Dopeman has the brief case.
- Frame 5: Jane Shady is at the beginning of the path to Denis Dopeman's door. Door is shut.

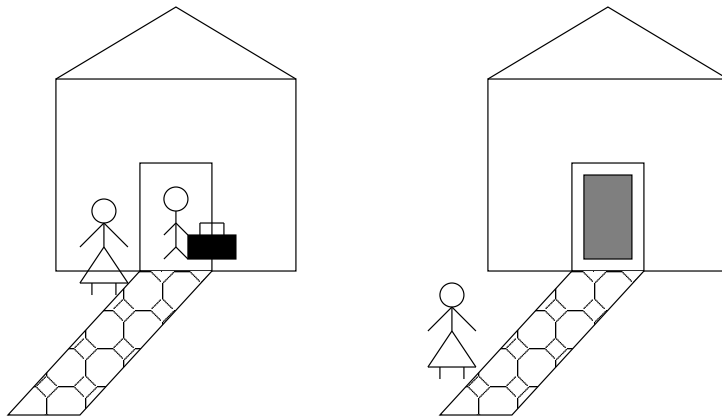
# Example, contd.



Frame 1

Frame 2

Frame 3



Frame 4

Frame 5

## Frame-dependent properties

---

Frame	Objects	Frame-dependent properties
1	Jane Shady	<code>has(briefcase), at(path_front)</code>
	dopeman_house	<code>door(closed)</code>
	Briefcase	
2	Jane Shady	<code>has(briefcase), at(path_middle)</code>
	Denis Dopeman	<code>at(door)</code>
	dopeman_house	<code>door(open)</code>
	Briefcase	
3	Jane Shady	<code>has(briefcase), at(door)</code>
	Denis Dopeman	<code>at(door)</code>
	dopeman_house	<code>door(open)</code>
	Briefcase	
4	Jane Shady	<code>at(door)</code>
	Denis Dopeman	<code>has(briefcase), at(door)</code>
	dopeman_house	<code>door(open)</code>
	Briefcase	
5	Jane Shady	<code>at(path_middle)</code>
	dopeman_house	<code>door(closed)</code>
	Briefcase	

## Frame independent properties

---

Object	Frame-independent property	Value
Jane Shady	age	35
	height	170 (cms)
dopeman_house	address	6717 Pimmit Drive Falls Church, VA 22047.
	type	brick
	color	brown
Denis Dopeman	age	56
	height	186
briefcase	color	black
	length	40 (cms)
	width	31 (cms)

## Activity Schema

---

- An *Activity Scheme*, **ACT\_SCH**, is a finite set of properties such that if  $(\text{pname}, \text{Values}_1)$  and  $(\text{pname}, \text{Values}_2)$  are both in **ACT\_SCH**, then  $\text{Values}_1 = \text{Values}_2$ .
- **Example:** Consider the activity **ExchangeObject** such as the exchange of objects between Jane Shady and Denis Dope-man. his activity has the three-pair scheme consisting of the pairs:
  1. **(Giver, Person)**: This pair specifies that the activity **ExchangeObject** has a property called **Giver** specifying who is transferring the object in question. This says that the property **Giver** is of type **Person**. **Person** is the set of all persons.
  2. **(Receiver, Person)** This pair specifies that the activity **ExchangeObject** has a property called **Receiver** specifying who is receiving the object in question.
  3. **(Item, Thing)**: This pair specifies the item being exchanged. **Thing** is the set of all “exchange-able” items.

Thus, the exchange of the briefcase that occurred between Jane Shady and Denis Dopeman can be captured as an activity scheme with **Giver** = Jane Shady, **Receiver** = Denis Dopeman, and **Item** = briefcase.

## Activity/Event

---

- An *Activity* is a pair:
  1. **AcID**: the “name” of the activity of scheme **ACT\_SCH** and
  2. for each pair  $(\mathbf{pname}, \mathbf{Values}) \in \mathbf{ACT\_SCH}$ , an equation of the form  $\mathbf{pname} = v$  where  $v \in \mathbf{Values}$ .
- Any activity has an associated activity scheme, and each property of the activity has an associated value from its set of possible values.
- **Example:**

1. The activity **Lecturing** may have the scheme

$$\{(\mathbf{Lecturer}, \mathbf{Person}), (\mathbf{Topic}, \mathbf{String})\}$$

and may contain the equations:

$$\mathbf{Lecturer} = \mathit{Prof. Felix}.$$

$$\mathbf{Topic} = \mathit{Video Databases}.$$

2. Likewise, the activity **Questioning** may have the scheme

$$\{(\mathbf{Questioner}, \mathbf{Person}), (\mathbf{Questionee}, \mathbf{Person}),$$

$$(\mathbf{Question}, \mathbf{String}), (\mathbf{Answer}, \mathbf{String})\}$$

and may contain the equations:

$$\mathbf{Questioner} = \mathit{Erica}.$$

$$\mathbf{Questionee} = \mathit{Prof. Felix}.$$

$$\mathbf{Question} = \text{How many children does a quadtree node have?}$$

$$\mathbf{Answer} = \text{At most 4.}$$

## Video Content

---

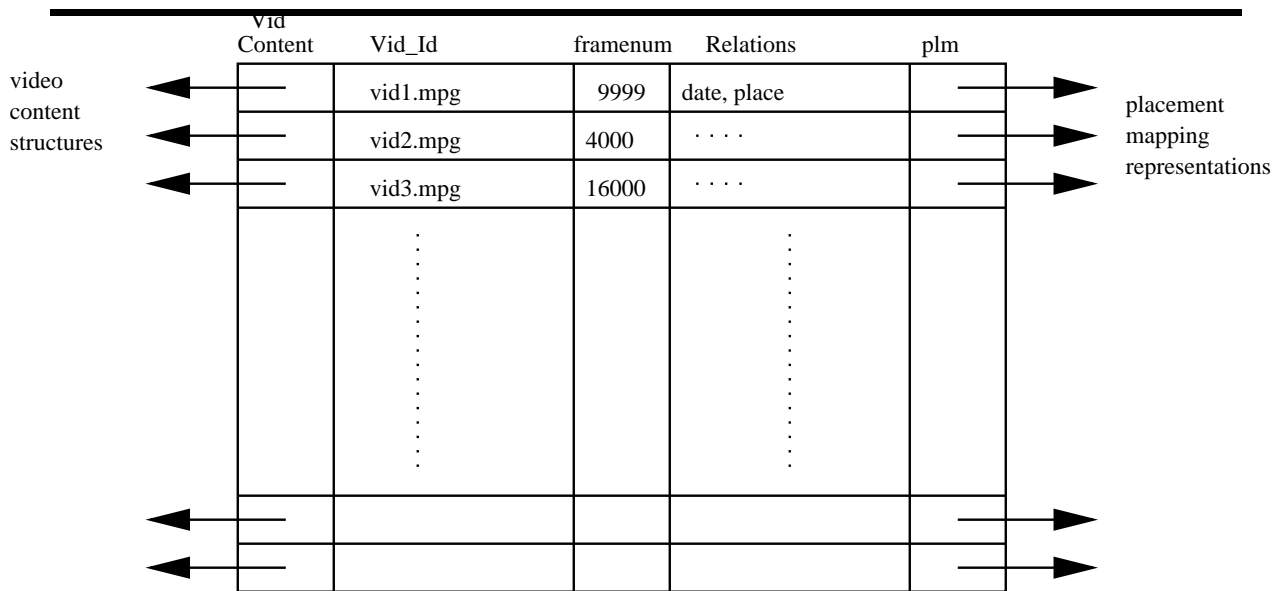
- Suppose  $v$  is a video.
- Let  $\text{framenum}(v)$  specify the total number of frames of video  $v$ .
- The *content* of  $v$  consists of a triple  $(\text{OBJ}, \text{AC}, \lambda)$  where:
  1.  $\text{OBJ} = \{\text{oid}_1, \dots, \text{oid}_n\}$  is a finite set of object instances;
  2.  $\text{AC} = \{\text{AcID}_1, \dots, \text{AcID}_k\}$  is a finite set of activities/events and
  3.  $\lambda$  is a map from  $\{1, \dots, \text{framenum}(v)\}$  to  $2^{\text{OBJ} \cup \text{AC}}$ .
- Intuitively,
  1.  $\text{OBJ}$  represents the set of objects of interest in the video and
  2.  $\text{AC}$  represents the set of activities of interest in the video and
  3.  $\lambda$  tells us which objects and which activities are associated with any given frame  $f$  of the video.
- Though this definition assumes that  $\lambda$  will be specified on a frame by frame basis, this is not required, as we will see later.

## Video Library

---

- A *video library*, VidLib, consists of a finite set of 5-tuples (Vid\_Id, VidContent, framenum, plm,  $\mathfrak{R}$ ) where:
  1. Vid\_Id is the *Name* of the video and
  2. VidContent is the *Content* of the video and
  3. framenum is the number of frames in the video and
  4. plm is a *placement mapping* that specifies the address of different parts of the video.
  5.  $\mathfrak{R}$  is a set of relations about videos “as a whole”.

# Organization of a simple video library



## Query Languages for Video Data

---

Querying video involves the following types of queries.

- **Segment Retrievals:** Find all segments, from one or more videos in the library, that satisfy a given condition.
- **Object Retrievals:** Given a video  $v$  and a segment  $[s, e]$  (start frame through end frame) of the video, find all objects that occurred in:
  - all frames between  $s$  and  $e$  (inclusive),
  - some frame between  $s$  and  $e$  (inclusive).
- **Activity Retrievals:** Given a video  $v$  and a segment  $[s, e]$  (start frame through end frame) of the video, find all activities occurred in:
  - all frames between  $s$  and  $e$  (inclusive),
  - some frame between  $s$  and  $e$  (inclusive).
- **Property-based Retrievals:** Find all videos, and video segments in which objects/activities with certain properties occur.

## Video Functions

---

- **FindVideoWithObject(o)**: Given the name of a data object  $o$ , this function returns as output, a set of triples of the form:

**(VideoId, Startframe, EndFrame)**

such that if  $(v, s, e)$  is a triple returned in the output, then video  $v$ 's segment starting at frame  $s$  and ending at frame  $e$  has the object  $o$  in all frames between and including  $s, e$ .

- **FindVideoWithActivity(a)**: This does exactly the same as above, except that it returns all triples  $(v, s, e)$  such that video  $v$ 's segment starting at frame  $s$  and ending at frame  $e$  has the activity  $a$  in it. For each property  $p$ , the notation  $a.p$  specifies the value of that property.
- **FindVideoWithActivityandProp(a,p,z)**: This does exactly the same as above, except that it returns all triples  $(v, s, e)$  such that video  $v$ 's segment starting at frame  $s$  and ending at frame  $e$  has the activity  $a$  in it with  $z$  as the value of property  $p$ .
- **FindVideoWithObjectandProp(o,p,z)**: This does exactly the same as above, except that it returns all triples  $(v, s, e)$  such that video  $v$ 's segment starting at frame  $s$  and ending at frame  $e$  has the object  $o$  in it with  $z$  as the value of property  $p$ .

- **FindObjectsInVideo( $v, s, e$ )**: Given the name of a video, and a start and end frame, this returns all objects that appear in all segments of the video between  $s$  and  $e$  (inclusive).
- **FindActivitiesInVideo( $v, s, e$ )**: Identical to the above, except it applies to activities, not objects.
- **FindActivitiesAndPropsInVideo( $v, s, e$ )**: Given the name of a video, a start and end frame, this returns a set of records of the form

activityname : prop1 = entity1; prop2 = entity2; ...  
propk = entityk

comprising all activities, and their associated roles, that occur in all times between  $s$  and  $e$  of video  $v$ .

- **FindObjectsAndPropsInVideo( $v, s, e$ )**: Identical to the above, except that it applies to objects, not to activities.

## Video Query Languages

---

- Standard SQL query has the form:

```

SELECT      field1,...,fieldn
FROM        relation1 <R1>,
            relation2 <R2>, ...,
            relationk <Rk>
WHERE       Condition.

```

- Expand this so that:

1. The **SELECT** statement may contain entries of the form

$$\text{Vid\_Id} : [s, e]$$

denoting the selection of a video with id, **Vid\_Id**, and with the relevant segment comprised of frames between *s* and *e* inclusive.

2. The **FROM** statement may contain entries of the form:

$$\text{video}\langle\text{source}\rangle\langle V \rangle$$

which says that *V* is a variable ranging over videos from the source named.

3. The **WHERE** condition allows statements of the form

$$\text{term} \underline{\text{IN}} \text{func\_call}$$

where:

- (a) *term* is either a variable or an object or an activity, or a property value and
- (b) *func\_call* is any of the eight video functions listed above.

## Examples

---

- “Find all videos and their relevant segments from video library VidLib<sub>1</sub> that contain Denis Dopeman.”

```
SELECT      vid:[s,e]
FROM        video:VidLib1
WHERE       (vid, s, e) IN
            FindVideoWithObject(Denis Dopeman).
```

- “Find all videos and their relevant segments from video library VidLib<sub>1</sub> that contain Denis Dopeman and Jane Shady.”

```
SELECT      vid:[s,e]
FROM        video:VidLib1
WHERE       (vid, s, e) IN
            FindVideoWithObject(Denis Dopeman) AND
            (vid, s, e) IN
            FindVideoWithObject(Jane Shady).
```

## Examples, Continued

---

- “Find all videos and their relevant segments from video library VidLib<sub>1</sub> that contain Denis Dopeman and Jane Shady exchanging a briefcase.”

```

SELECT      vid:[s,e]
FROM        video:VidLib1
WHERE       (vid, s, e) IN
            FindVideoWithObject(Denis Dopeman)
            AND
            (vid, s, e) IN
            FindVideoWithObject(Jane Shady)
            AND
            (vid, s, e) IN
            FindVideoWithActivityandProp
                (ExchangeObject, Item, Briefcase)
            AND
            (vid, s, e) IN
            FindVideoWithActivityandProp
                (ExchangeObject, Giver, Jane Shady)
            AND
            (vid, s, e) IN
            FindVideoWithActivityandProp
                (ExchangeObject, Receiver, Denis Do

```

## Indexing Video Content

---

- Now that we have defined content, we need to index it.
- We have 8 types of video retrieval functions. Indexing must support efficient execution of these 8 function types.
- It is impossible to store video content on a frame by frame basis due to the fact that a single 90 minute video contains close to ten million frames.
- We need Compact Representations to store video content.
- Two such data structures:
  - Frame Segment Tree
  - R-Segment Tree

## Frame Segment Trees

---

- **Frame-sequence** is a pair  $[i, j)$  where  $1 \leq i, \leq j \leq n$ .  $[i, j)$  represents the set of all frames between  $i$  (inclusive) and  $j$  (non-inclusive), i.e.

$$[i, j) = \{k \mid i \leq k < j\}.$$

- EX:  $[6, 12)$  denotes the set of frames  $\{6, 7, 8, 9, 10, 11\}$ .
- **Frame-sequence Ordering:**  $[i_1, j_1) \sqsubseteq [i_2, j_2)$  iff  $i_1 < j_1 \leq i_2 < j_2$ .
- $[i_1, j_1) \sqsubseteq [i_2, j_2)$  means that the sequence of frames denoted by  $[i_1, j_1)$  precedes the sequence of frames denoted by  $[i_2, j_2)$ .
- EX: Consider frame-sequences  $fs_1 = [10, 15)$ ,  $fs_2 = [8, 10)$  and  $fs_3 = [11, 13)$ .

- $fs_2 \sqsubseteq fs_1$
- $fs_2 \sqsubseteq fs_3$
- $fs_1 \not\sqsubseteq fs_3$ .

- **Well-Ordered Set of Frame-sequences:** A set,  $X$ , of frame-sequences is said to be *well-ordered* iff:
  1.  $X$  is finite, i.e.  $X = \{[i_1, j_1), \dots, [i_r, j_r)\}$  for some integer  $r$ , and
  2.  $[i_1, j_1) \sqsubseteq [i_2, j_2) \sqsubseteq \dots \sqsubseteq [i_r, j_r)$ .
- EX:  $X = \{[1, 4), [9, 13), [33, 90)\}$  is a well-ordered set of frame-sequences because  $[1, 4) \sqsubseteq [9, 13) \sqsubseteq [33, 90)$ .

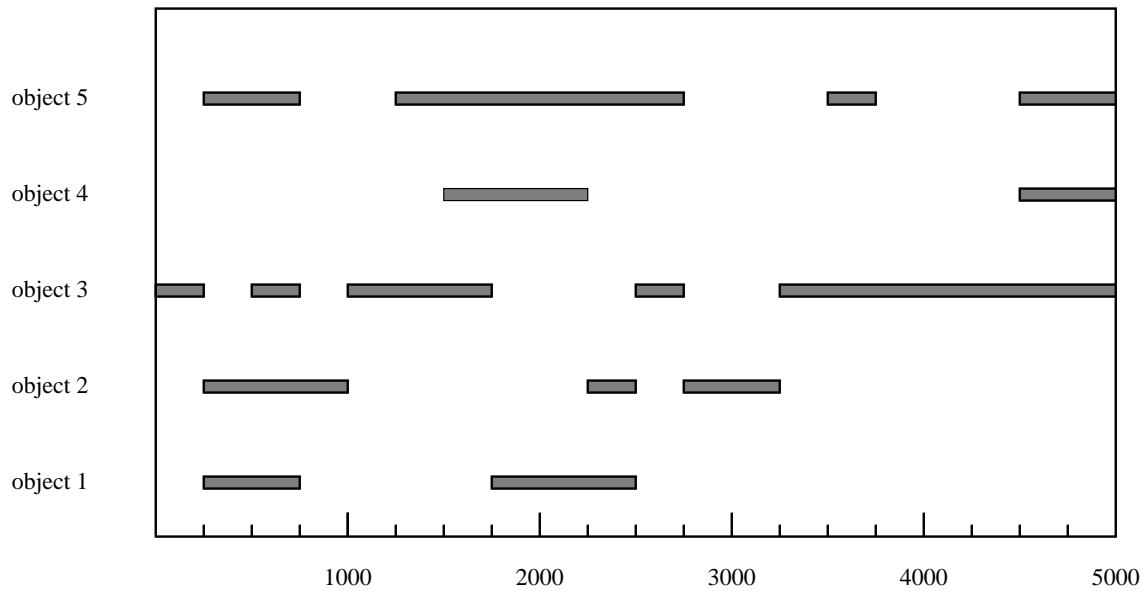
## Frame Segment Trees, Continued

---

- **Solid Set of Frame-sequences:** A set,  $X$ , of frame-sequences is said to be *solid* iff
  1.  $X$  is well-ordered, and
  2. there is no pair of frame-sequences in  $X$  of the form  $[i_1, i_2)$  and  $[i_2, i_3)$ .
- Take  $X = \{[1, 5), [5, 7), [9, 11)\}$ .
- $X$  is not solid. Why?
- Take  $Y = \{[1, 7), [9, 11)\}$ . This is solid.
- **Segment Association Map:** Suppose  $(\text{OBJ}, \text{AC}, \lambda)$  represents the content of a video  $v$ . A *Segment Association Map*  $\sigma_v$  associated with video  $v$  is the map defined as follows:
  1.  $\sigma_v$ 's domain is  $\text{OBJ} \cup \text{AC}$  and
  2.  $\sigma_v$  returns, for each  $x \in \text{OBJ} \cup \text{AC}$ , a *solid* set of frame-sequence, denoted  $\sigma_v(x)$  such that:
    - (a) if  $[s, e) \in \sigma_v(x)$ , then for all  $s \leq f < e$ , it is the case that  $x \in \lambda(f)$  and
    - (b) for all frames  $f$  and all  $x \in \text{OBJ} \cup \text{AC}$ , if  $x \in \lambda(f)$ , then there exists a frame-sequence  $[s, e) \in \sigma_v(x)$  such that  $f \in [s, e)$ .

## An example of a video's content

---



## Example, continued

---

- 5000 frames in example.
- The table below shows how many frames each object appears in:

Object	Number of frames
object1	1250
object2	1500
object3	3250
object4	1000
object5	2750

- To explicitly represent the mapping  $\lambda$  associated with the content of this video, we would need to have a total of 9750 tuples.
- Instead, represent information with 16 tuples as shown below.

### Segment Table:

Object	Segment
object1	250–750
object1	1750–2500
object2	250–1000
object2	2250–2500
object2	2750–3250
object3	0–250
object3	500–750
object3	1000–1750
object3	2500–2750
object3	3250–5000
object4	1500–2250
object4	4500–5000
object5	250–750
object5	1250–2750
object5	3500–3750
object5	4500–5000

## Frame-segment tree structure

---

- Suppose there are  $n$  objects  $o_1, \dots, o_n$  in our video  $v$  and  $m$  activities  $a_1, \dots, a_m$ .

- Then we have a total of:

$$\sum_{i=1}^n (\text{card}(\sigma_v(o_i))) + \sum_{j=1}^m (\text{card}(\sigma_v(a_j)))$$

entries in the table *just for one single video*.

- FS-trees use the following components:
  - **OBJECTARRAY**: specifies, for each object, an *ordered linked list* of pointers to nodes in the frame segment tree specifying which segments the object appears in.
  - **ACTIVITYARRAY**: specifies, for each activity, an *ordered linked list* of pointers to nodes in the frame segment tree specifying which segments the activity occurs in.
  - The FS-tree is now constructed from the segment table.

## Frame-segment trees, continued

---

- **Step 1:** Let  $[s_1, e_1), \dots, [s_w, e_w)$  be all the intervals in the “Segment” column of the segment table. Let

$$q_1, \dots, q_z$$

be an enumeration, in ascending order, of all members of  $\{s_i, e_i \mid 1 \leq i \leq w\}$  with duplicates eliminated.

If  $z$  is not an exponent of 2, then do as follows: let  $r$  be the smallest integer such that  $z < 2^r$  and  $2^r > \text{framenum}(v)$ . Add new elements  $q_{z+1}, \dots, q_{2^r}$  such that  $q_{2^r} = \text{framenum}(v) + 1$  and  $q_{z+j} = q_z + j$  (for  $z + j < 2^r$ ).

*By virtue of the above argument, we may proceed under the assumption that  $z$  is an exponent of 2, i.e.  $z = 2^r$  for some  $r$ .*

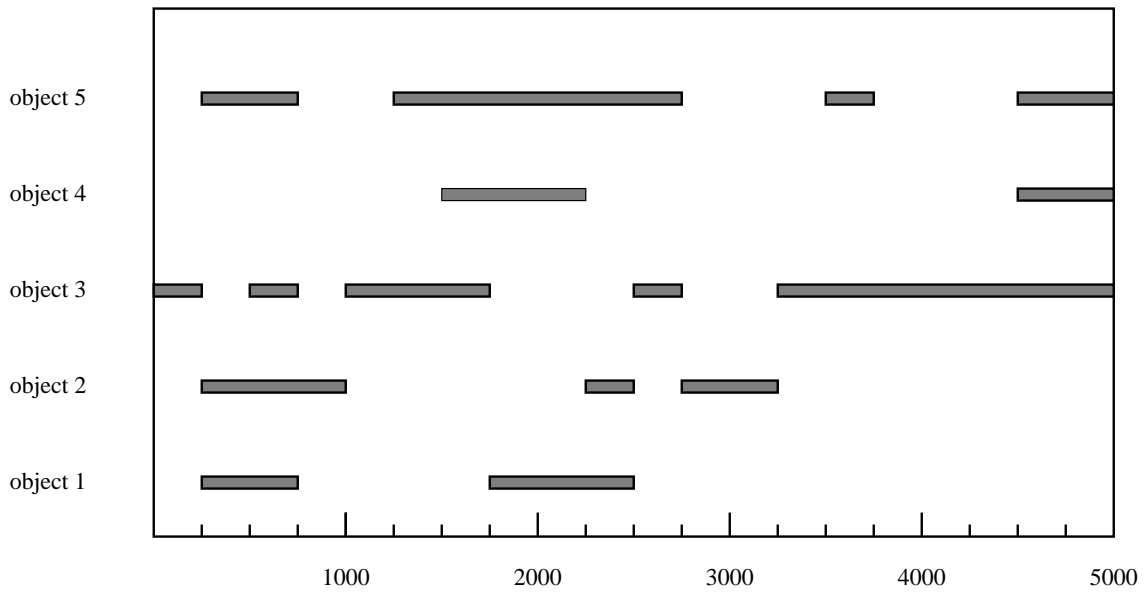
- **Step 2:** The frame-segment tree is a binary tree constructed as follows.
  1. Each node in the frame segment tree represents a *frame-sequence*  $[x, y)$  starting at frame  $x$  and including all frames up to, but not including, frame  $y$ .
  2. Every leaf is at level  $r$ . The leftmost leaf denotes the interval  $[z_1, z_2)$ , the second from left-most represents the interval  $[z_2, z_3)$ , the third from left-most represents the interval  $[z_3, z_4)$  and so on. If  $N$  is a node with two children

representing the intervals  $[p_1, p_2)$ ,  $[p_2, p_3)$ , then  $N$  represents the interval  $[p_1, p_3)$ . Thus, the root of the segment tree represents the interval  $[q_1, q_z)$  if  $q_z$  is an exponent of 2; otherwise it represents the interval  $[q_1, \infty)$ .

3. The number inside each node may be viewed as the address of that node.
4. The set of number placed next to a node denotes the id-numbers of video objects and activities that appear in the entire frame-sequence associated with that node. Thus, for example, if a node  $N$  represents the frame sequence  $[i, j)$  and object  $o$  occurs in all frames in  $[i, j)$ , then object  $o$  labels node  $N$  (unless object  $o$  labels an ancestor of node  $N$  in the tree).

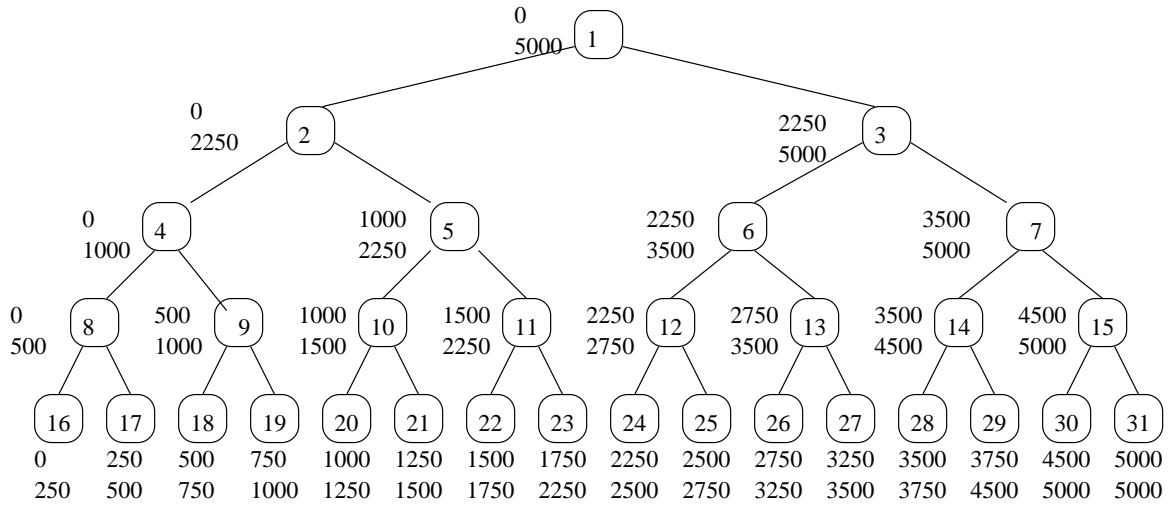
# Example

---



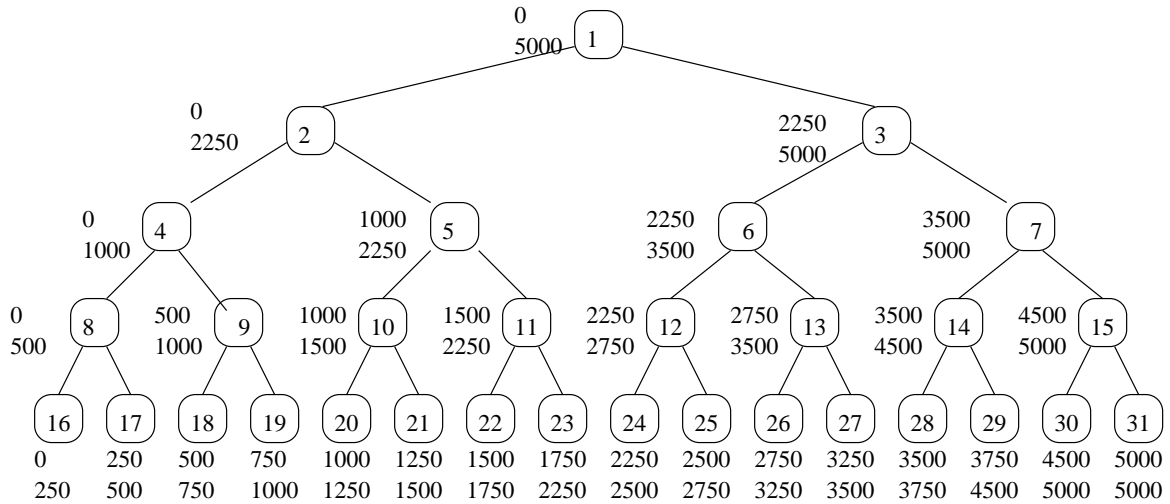
# Example, continued

---



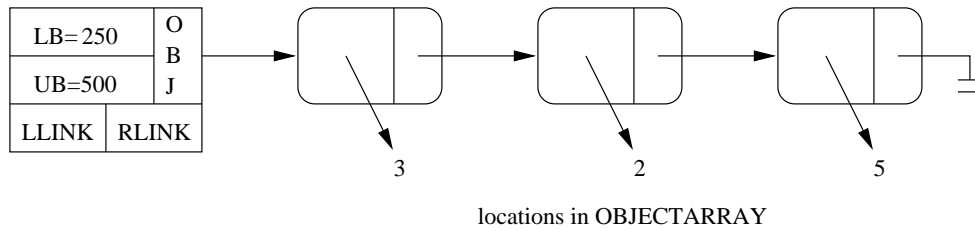
# Example, continued

---



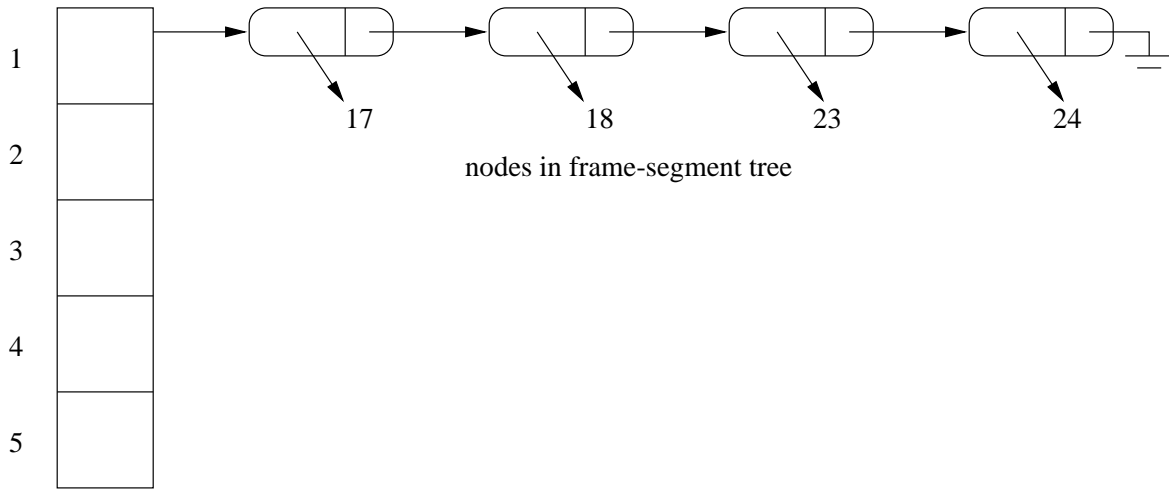
## Example, continued

---



# Example, continued

---



## Video Library

---

- Suppose our video library, VidLib, contains videos  $v_1, \dots, v_n$ .
- Create a table called **INTOBJECTARRAY** having the scheme **(VID\_ID, OBJ, PTR)**.
- Tuple  $(v, o, ptr)$  is in **INTOBJECTARRAY** iff the pair  $(o, ptr)$  is in the **OBJECTARRAY** associated with video  $v$ .
- Create a table called **INTACTIVITYARRAY** having the scheme **(VID\_ID, ACT, PTR)**.
- Tuple  $(v, a, ptr)$  is in **INTACTIVITYARRAY** iff the pair  $(a, ptr)$  is in the **ACTIVITYARRAY** associated with video  $v$ .
- For each  $v_i$ , a frame segment tree,  $\text{fst}(v_i)$  is associated with video  $v_i$ .
- Only difference from before is that pointers from the frame segment tree point to locations in **INTOBJECTARRAY** and **INTACTIVITYARRAY** rather than to **OBJECTARRAY** and **ACTIVITYARRAY** as described earlier.

## Implementing Video Operations

---

- FindVideoWithObject(o):

```
SELECT      VIDEO_ID
FROM        INTOBJECTARRAY
WHERE       OBJ = o.
```

- FindVideoWithActivityandProp(a,p,z):

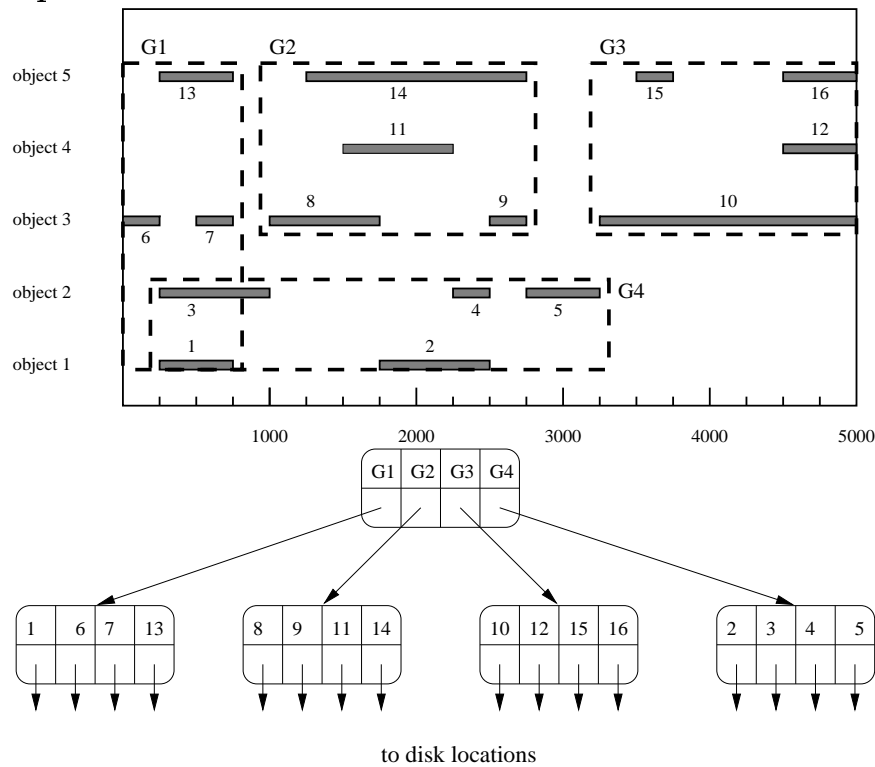
```
SELECT      VIDEO_ID
FROM        INTOBJECTARRAY t
WHERE       OBJ = o AND t.p = z.
```

- FindObjectsInVideo(v,s,e):

```
Algorithm 4 FindObjectsInVideo(R,s,e)
S = NIL; (* no objects found so far *)
if R = NIL then { Return S; Halt }
else
  { if [R.LB, R.UB] ⊆ [s,e] then S = append(S,preorder(R))
  else
    { if [R.LB, R.UB] ∩ [s,e] ≠ ∅ then
      { S=append(S,R.obj);
        S=append(S,FindObjectsInVideo(R.LLINK,s,e));
        S=append(S,FindObjectsInVideo(R.RLINK, s,e));
      }
    }
  }
return(S); end
```

## RS-Trees

- Very similar to the frame segment tree.
- The concepts of **OBJECTARRAY** and **ACTIVITYARRAY** remain the same as before.
- Instead of using a segment tree to represent the frame sequences (such as those shown in Figure ??), we take advantage of the fact that a sequence  $[s, e)$  is a *rectangle* of length  $(e - s)$  and of width 0.
- We already know how to represent a set of rectangles using an R-tree.
- Example:



## Video Segmentation

---

- We have assumed a *logical delineation* of video data – video is brokenup into homogeneous segments.
- Usually, a video is created by taking a set of *shots*.
- These shots are then composed together using specified *composition operators*.
- Shots are usually taken with a fixed set of cameras each of which has a constant relative velocity.
- A *shot composition* operator, often referred to as an *edit effect* is an operation that takes as input, two shots,  $S_1, S_2$ , and a duration,  $t$ , and *merges* the two shots into a composite shot in within time  $t$ .
- Thus, for example, suppose we wish to compose together, two shots  $S_1, S_2$  and suppose these two shots have durations  $t_1, t_2$  respectively. If  $f$  is a shot composition operator, then

$$f(S_1, S_2, t)$$

creates a segment of video of length  $(t_1 + t_2 + t)$ .  $S_1$  is first shown and then undergoes a continuous transformation over a time interval  $t$  leading to the presentation of  $S_2$  next.

- $f(S_1, S_2, t)$  then is a continuous sequence of video.
- In general, a video as a whole may be represented as:

$$(f_n(\dots f_2(f_1(S_1, S_2, t_1), S_3, t_2) \dots, S_n, t_n).$$

## Shot Composition Operators

---

- **Shot Concatenation:** Concatenates the two shots (even if the transition is not smooth). If `shotcat` is a shot concatenation operator, then  $t$  must be zero, i.e. whenever we invoke `shotcat( $S_1, S_2, t$ )`, the third argument  $t$  must be set to 0.
- **Spatial Composition:** Examples include: a *translate* operation which causes two successive shots to be overlaid one on top of the other. For instance, suppose we want to show shot  $S_1$  first, followed by shot  $S_2$ . This is done by first overlaying shots  $S_1$  *on top* of shot  $S_2$  and then moving (i.e. translating) shot  $S_1$  away, thus exposing shot  $S_2$ .
- **Chromatic Composition:** *fades* and *dissolves*. Both these operations are chromatic scaling operations that try to continuously transform each pixel  $(x, y)$  in the first shot into the corresponding pixel in the second shot.

## Video Segmentation Problem

---

- Given a video  $V$ , express the video  $V$  in the form:

$$V = f_n(\dots f_2(f_1(S_1, S_2, t_1), S_3, t_2) \dots, S_n, t_n).$$

- That is, given video  $V$ , find  $n$  and shots  $S_1, \dots, S_n$ , times  $t_1, \dots, t_n$ , and composition operations  $f_1, \dots, f_n$  such that the above equation holds.

## Video Standards

---

- All video compression standards attempt to compress videos by performing an *intra-frame* analysis.
- Each frame is divided up into blocks. Compare different frames to see which data is “redundant” in the two frames.
- Drop redundant data to compress.
- Compression quality is measured by:
  - the fidelity of the color map – how many colors of the original video occur when the compressed video is decompressed ?
  - the pixel resolution per frame – how many pixels per frame of the video have been dropped ?
  - the number of frames per second – how many frames have been dropped?
- Compression Standards: MPEG-1,2,3, Cinepak, JPEG video etc.

## MPEG-1

---

- Stores videos as a sequence of *I*, *P* and *B* frames.
- *I* frames are independent images called “intra frames”. Basically a still image.
- P-frame is computed from the closest I-frame *preceding* it by interpolation (using DCT).
- B-frames are computed by interpolating from the two closest P or I frames.

# MPEG-1, Continued

---

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I	B	B	P	B	B	I	B	B	P	B	B	I	B	B	P	B	B	I	B

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I	B	B	P	B	B	I	B	B	P	B	B	I	B	B	P	B	B	I	B

## Other Compression Standards

---

- MPEG-2 uses higher pixel resolution and a higher data rate, thus making it superior to MPEG-1 in terms of the quality of the video as seen by the user. However, it requires higher bandwidth thus making it feasible for some, but not all applications.
- MPEG-3 s even higher sampling rates and frames per second than MPEG-2.