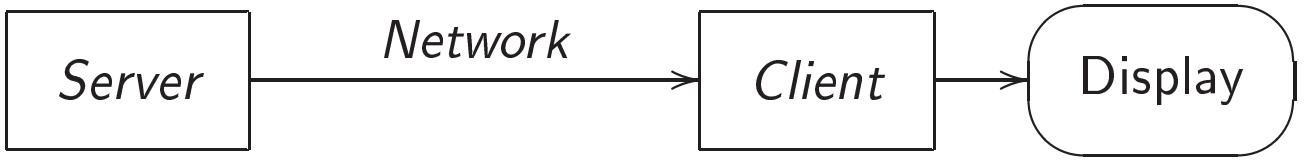
Prototyping a Video Pipeline in Timber

Magnus Carlsson 2004

Background to Streaming Video

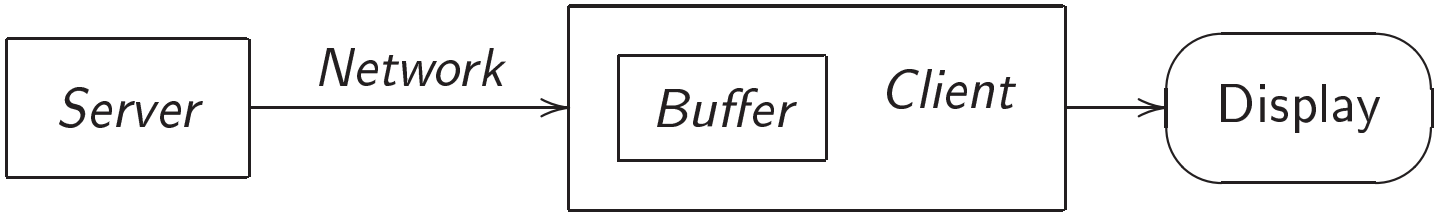


Problems:

- Limited network bandwidth
- Network latency
- Variations in bandwidth and latency

How to avoid dropouts?

Background to Streaming Video



Solution: *Buffer* fluctuations in bandwidth and latency.

Background to Streaming Video

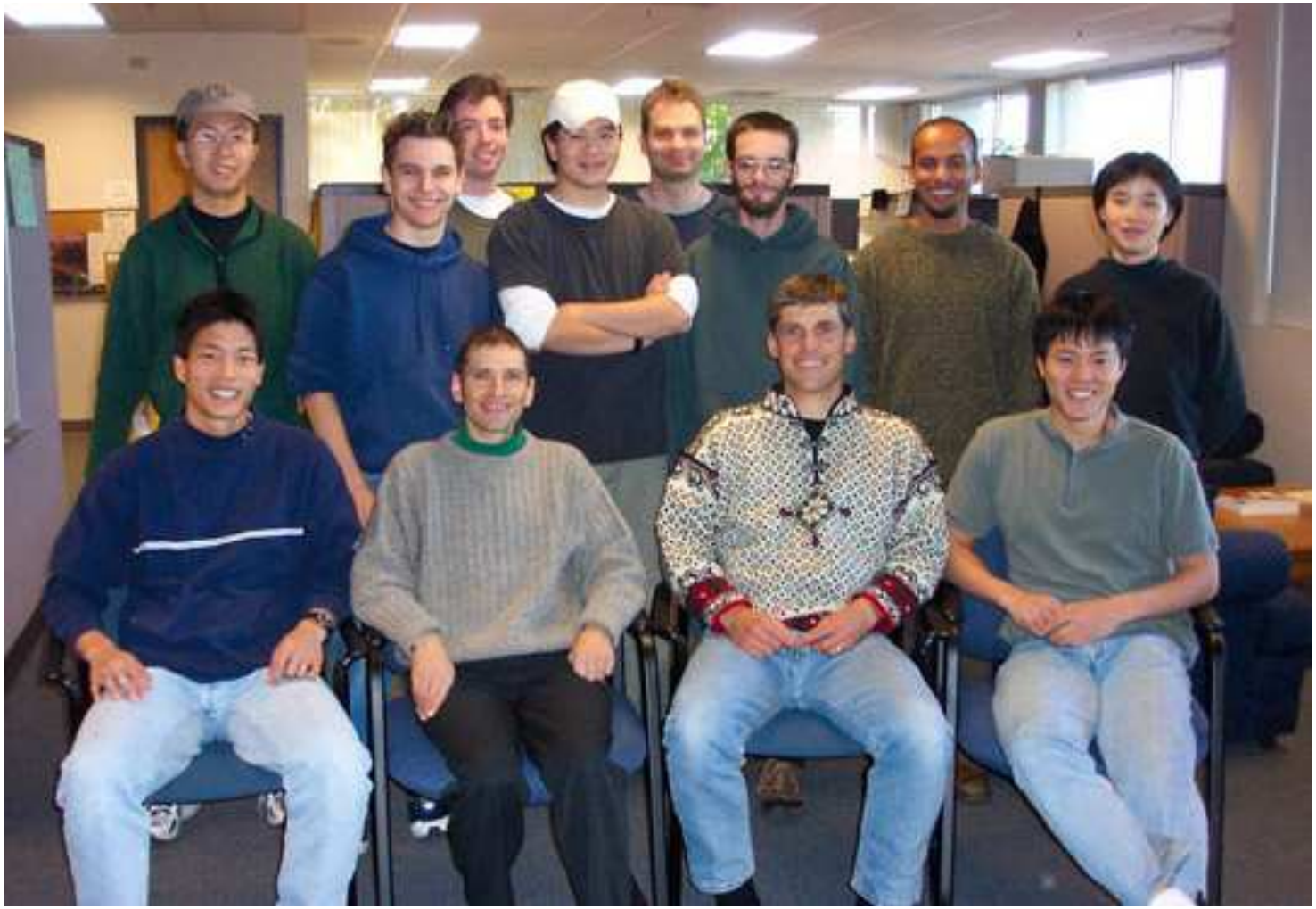
Filling the buffer can take a long time!

Background to Streaming Video

Also: the buffer can become empty during streaming!

Priority progress streaming

Developed at the Systems Software Lab, Oregon Graduate Institute



by Buck Krasic, Jon Walpole and others

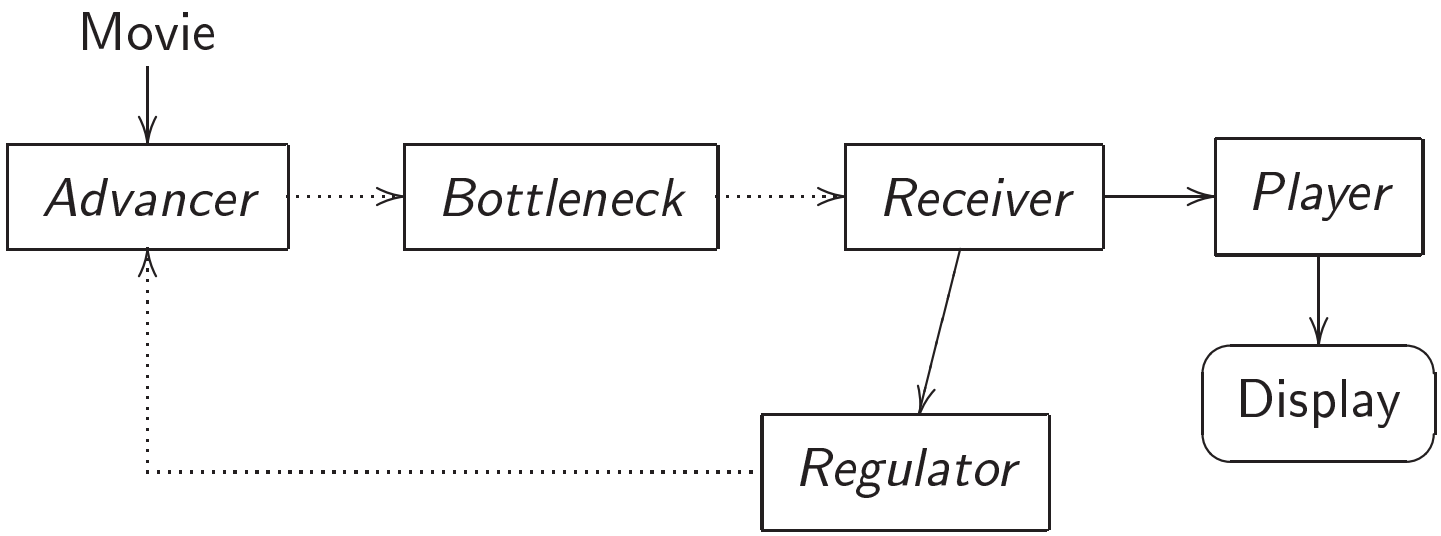
Priority progress streaming

The basic idea: **send most important information first**

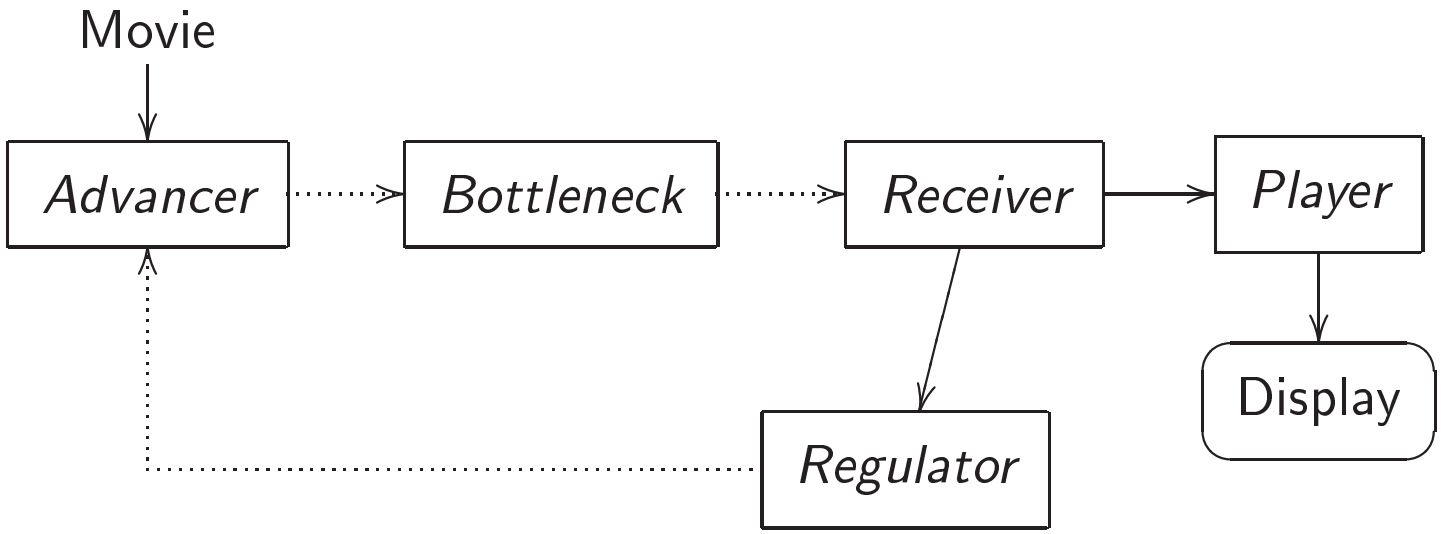
(Compare to *progressive JPEG*)

Priority progress streaming

From "Quality-Adaptive Media Streaming by Priority Drop", by Krasic and Walpole (ftp://cse.ogi.edu/pub/tech-reports/2002/02-015.pdf):



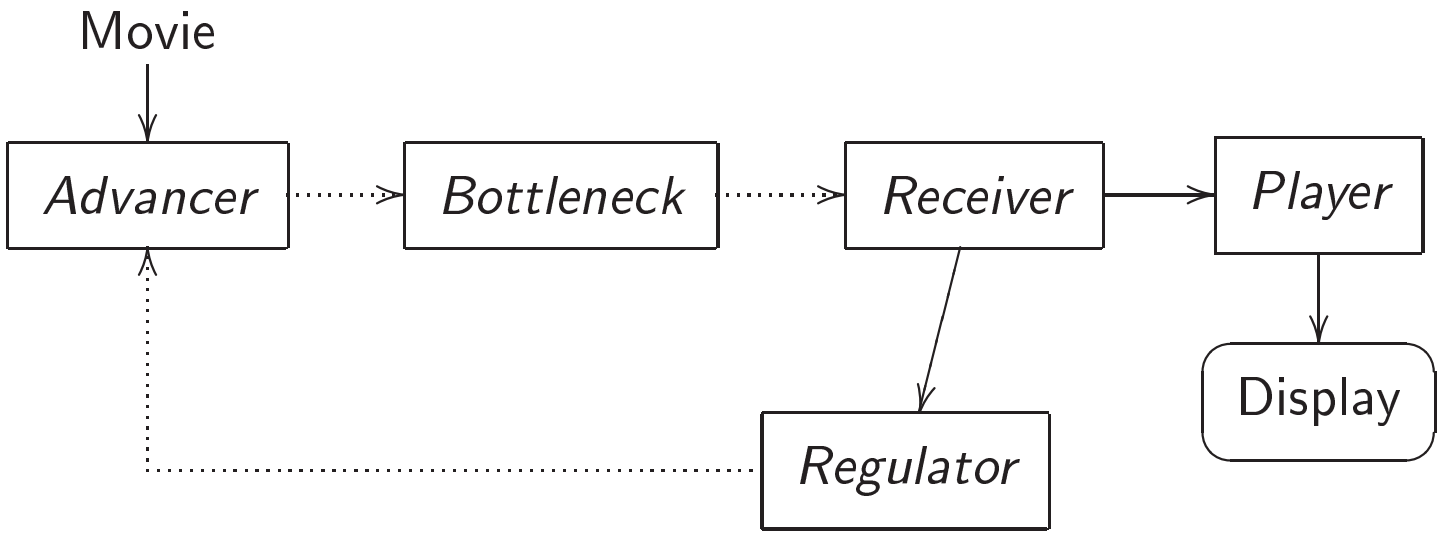
Priority progress streaming



Advancer:

- Collect one *time window*'s worth of frames of movie
- Sort the frame collection according to priority
- Output frame collection to bottleneck as quickly as it permits
- Interrupt when the window has passed

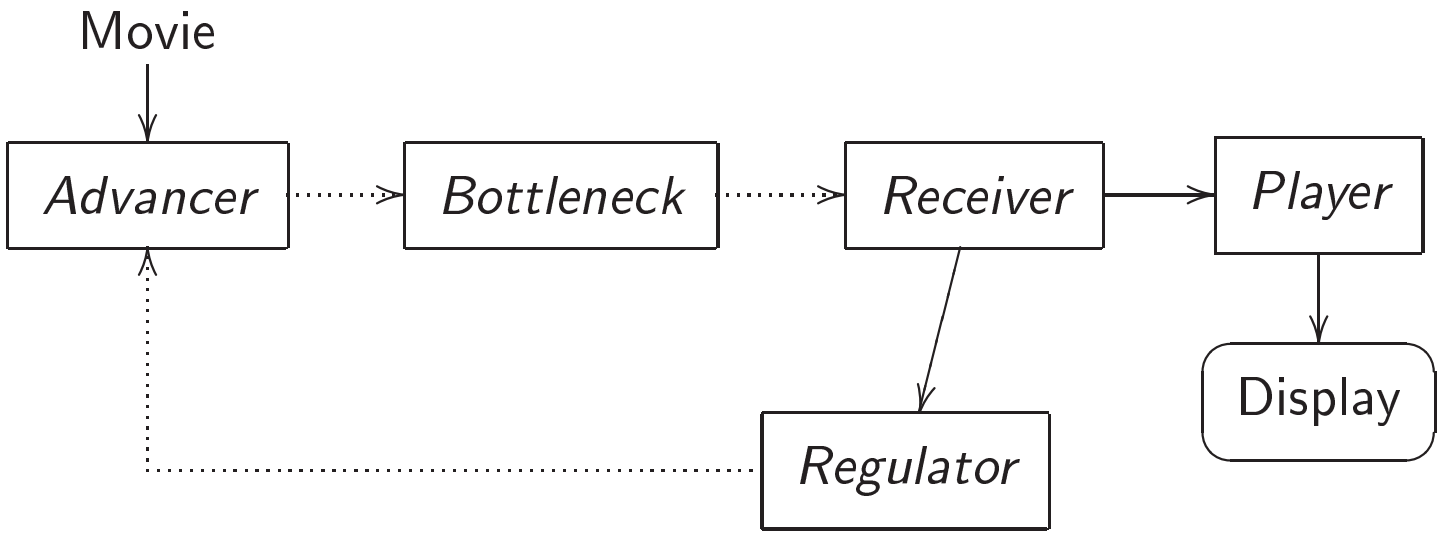
Priority progress streaming



Bottleneck:

- Transmit data at varying bandwidth and latency
- Note: good match with TCP

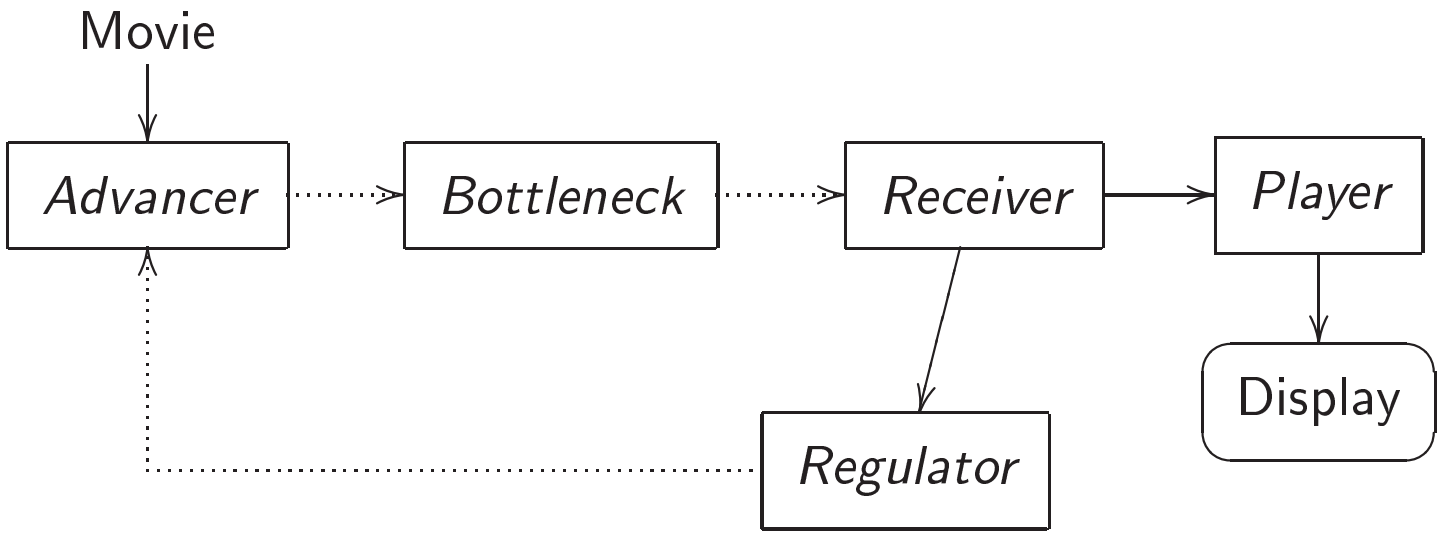
Priority progress streaming



Receiver:

- Collect frames from bottleneck until window has passed
- Sort frame collection according to time stamps
- Pass frame collection to player

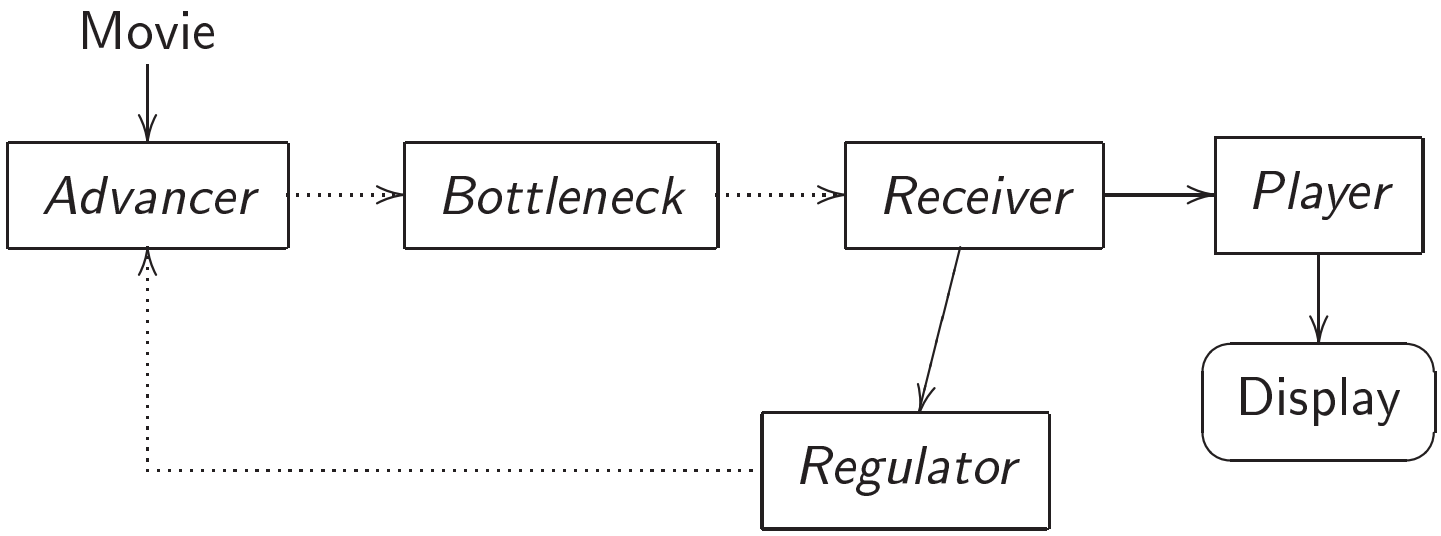
Priority progress streaming



Player:

- Output frames to display without jitter

Priority progress streaming

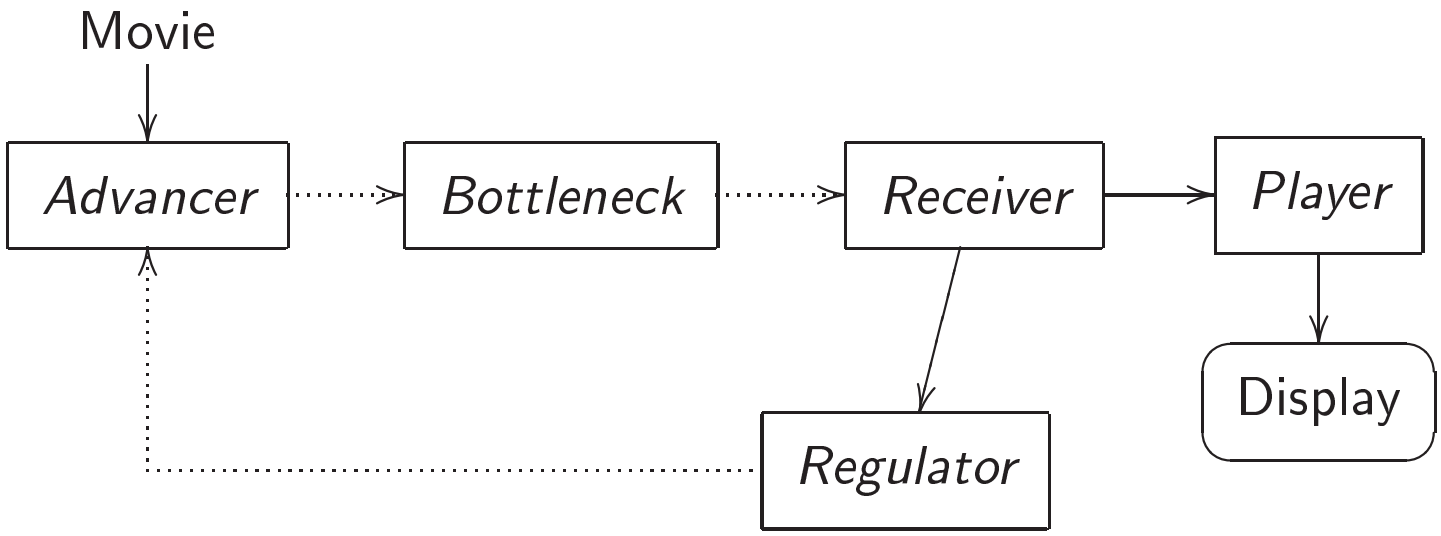


What happens if frames are delayed so that they arrive *after* the window has passed?

Receiver:

- Report late frames to Regulator

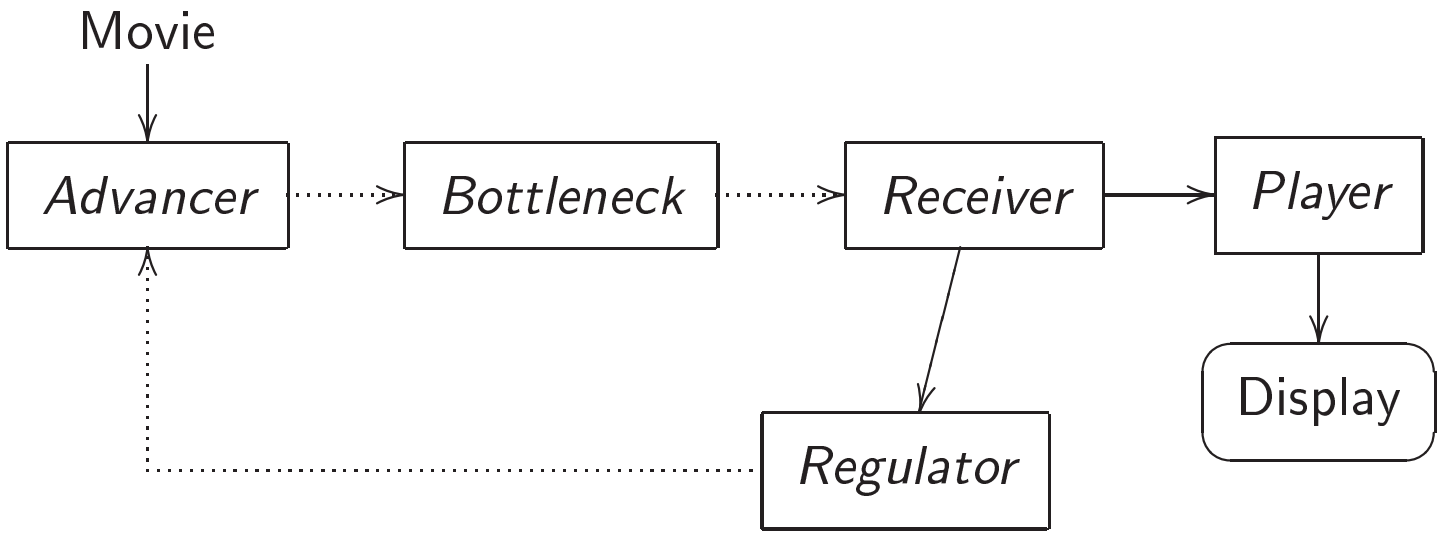
Priority progress streaming



Regulator:

- Maintain maximum frame-underrun time (*phase*)
- Report increased phase values to Advancer

Priority progress streaming



Advancer:

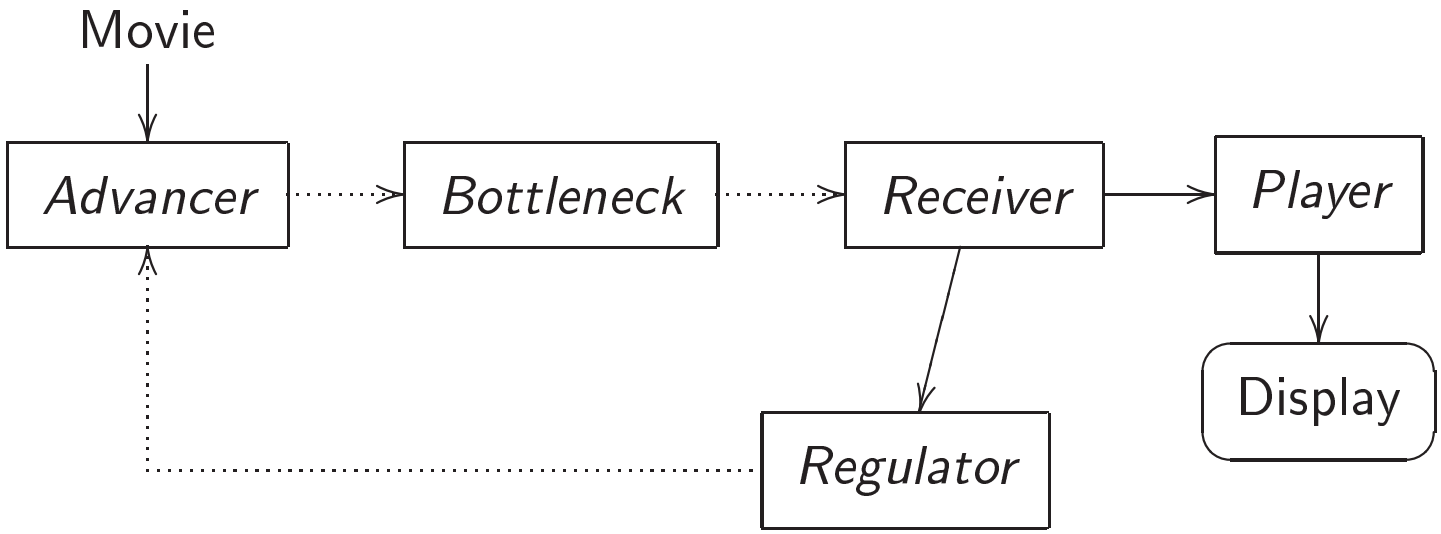
- Receive increased phase values p from Regulator
- Start window number i at time $iw - p$, where w is window size

PPS in Timber

- Simplified prototype, running in the *TI* interpreter
- Streams a "movie" with three objects of different priority:
  - Highest priority: Big black blob
  - Medium priority: Small green blob
  - Low priority: Tiny red blob
- Every 10th frame is more important than every 5th, which is more important than every second:

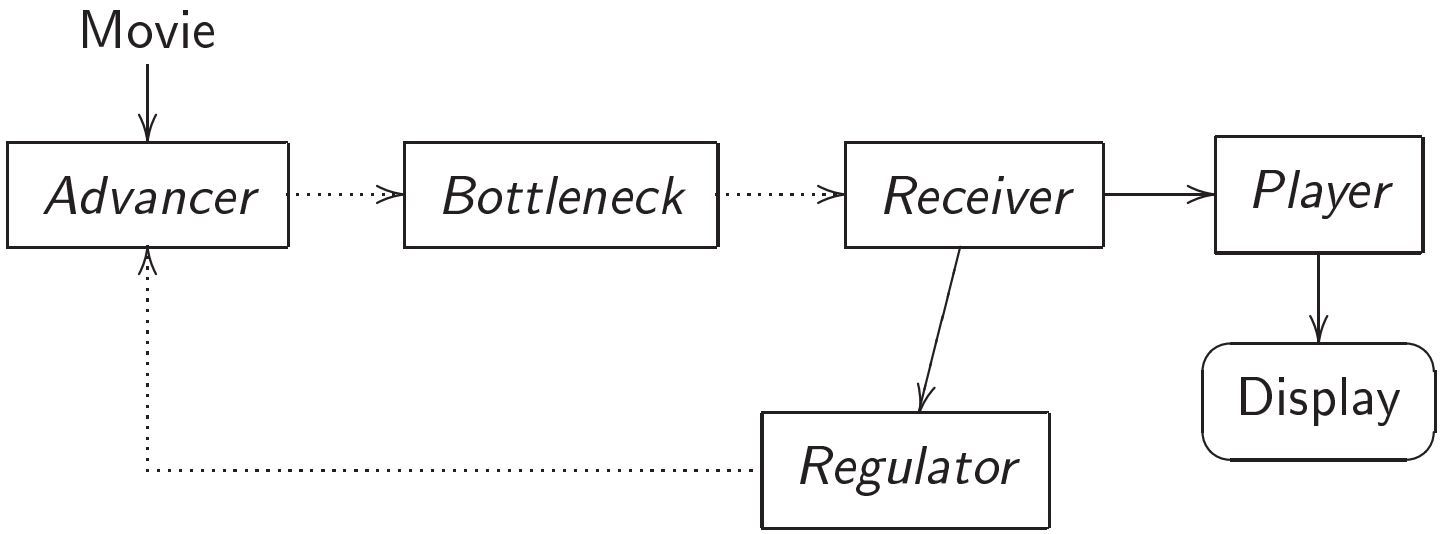
```
priority f | f `mod` 10 == 0 = 3
           | f `mod` 5  == 0 = 2
           | f `mod` 2  == 0 = 1
           | otherwise       = 0
```

PPS in Timber




```
record Advancer =
  start    :: Time -> Action  -- start playing from given time
  stop     :: Action
  adjust   :: Time -> Action  -- adjust for increased latency
```

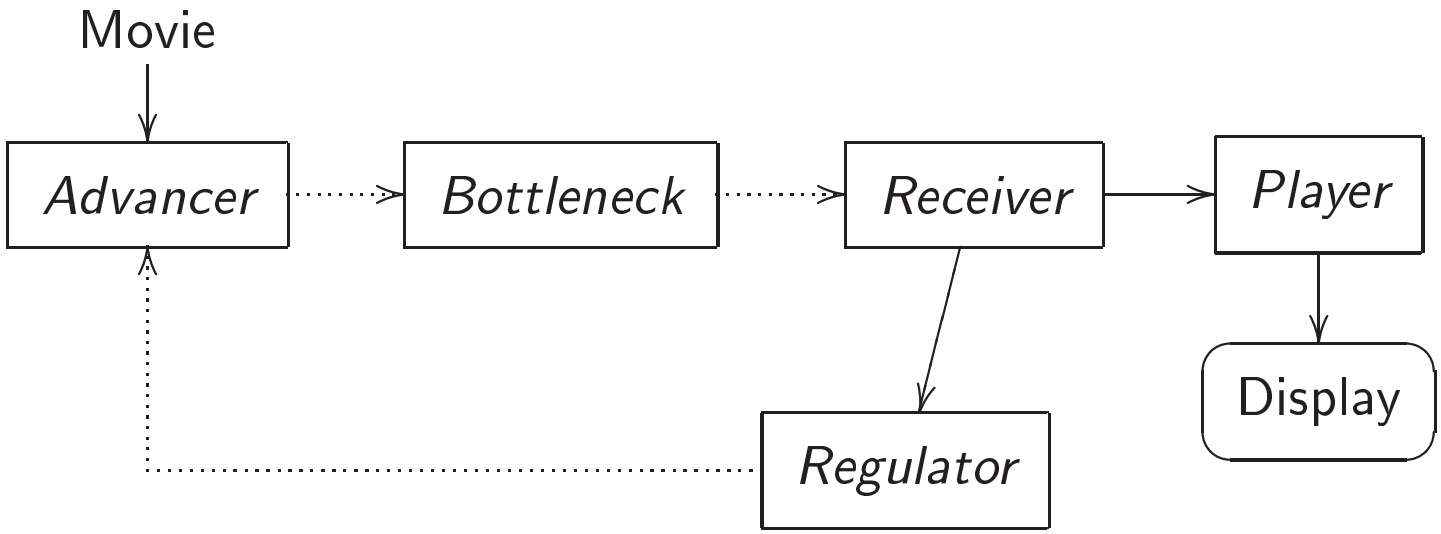
PPS in Timber




```
record Movie a =
  frames :: Time -> Time -> Request [MovieFrame a]
```

Picks up all frames in given time interval

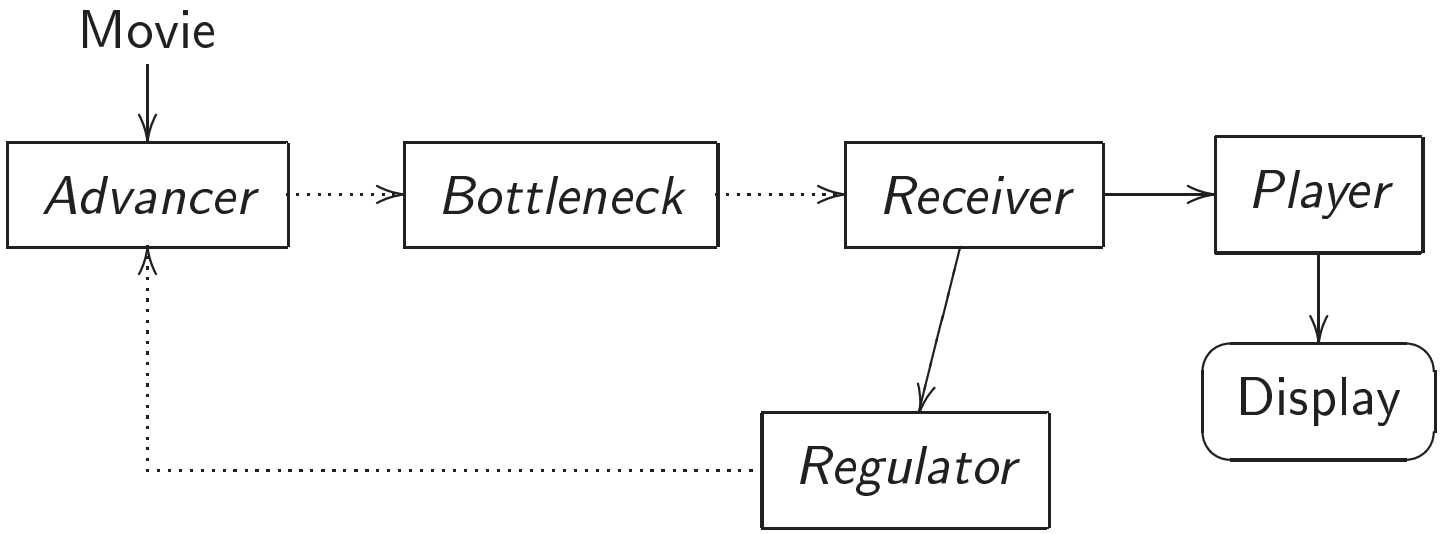
PPS in Timber




```
record Bottleneck a =
   send :: [a] -> Action -> Action
```

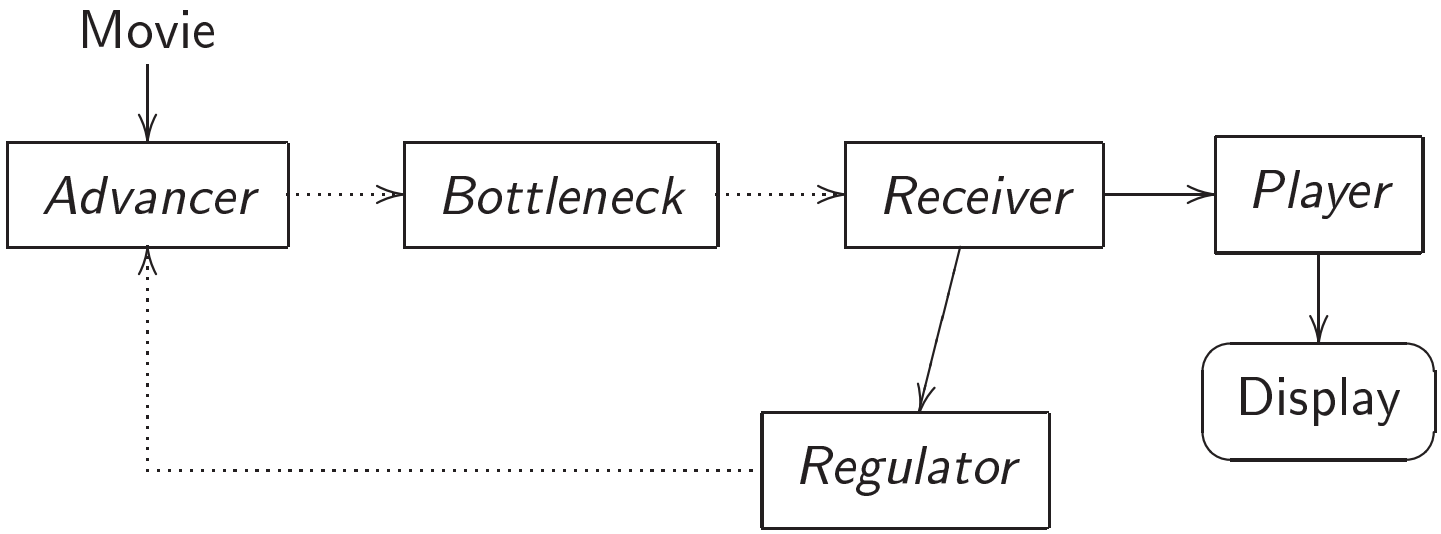
b.send l c sends the message list l on the bottleneck b, which in turn announces completion on the action c

PPS in Timber




```
record Receiver a =
  receive :: a -> Action
```

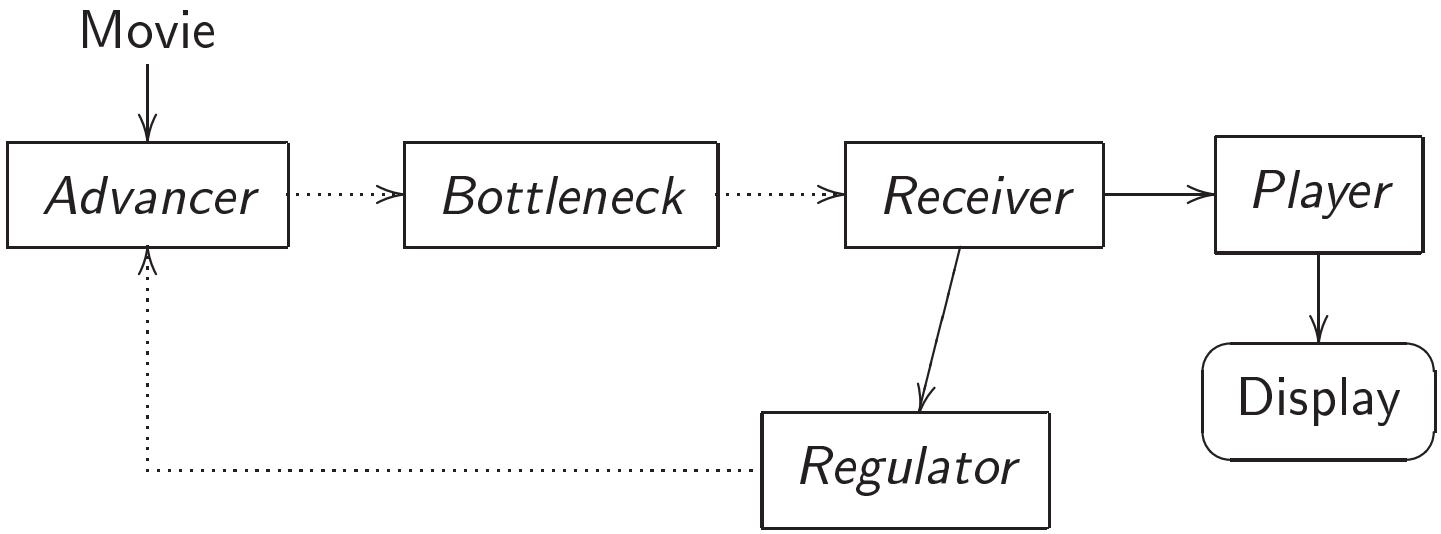
PPS in Timber




```
record Player a =
  play :: [MovieFrame a] -> Action

data MovieFrame a =
  Frame Integer -- priority
        Time    -- deadline
        a
```

PPS in Timber




```
record Regulator =
  setphase      :: Time -> Time -> Request ()
```

r.`setphase` p o informs the regulator r that a frame sent using phase p came late by the amount of o

PPS in Timber

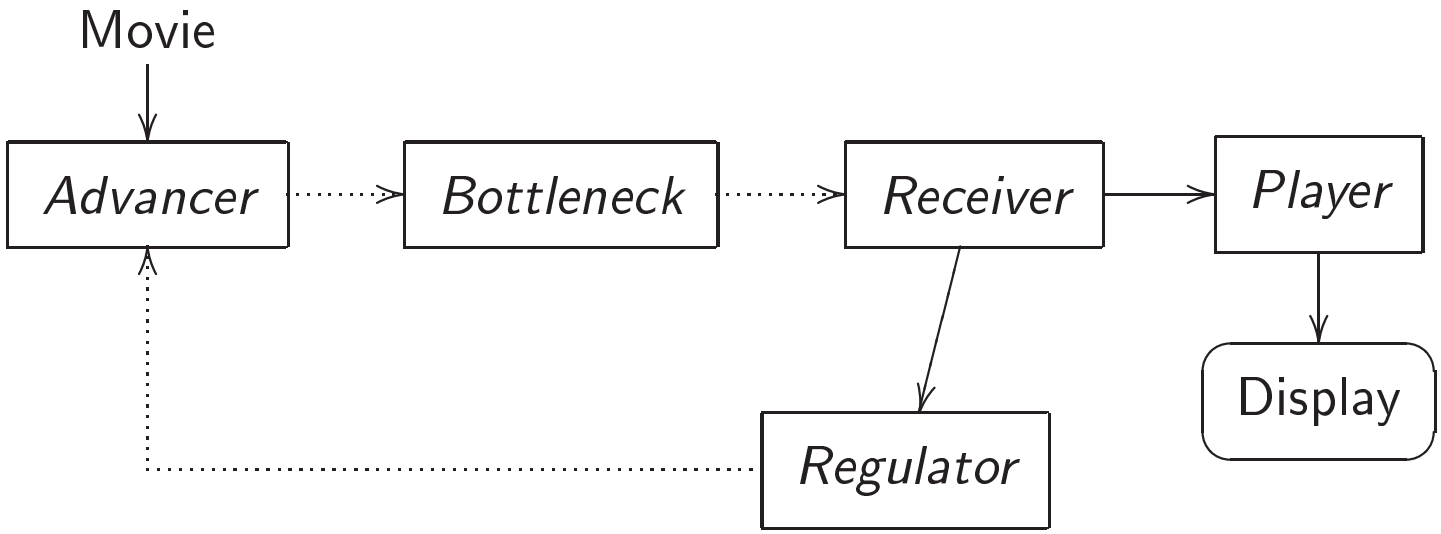
```
regulator :: Advancer -> Template Regulator
regulator adv =
  template
    s := 0
  in record
       setphase p' o =
         request
           let s' = p' + o
           if s' > s then
             s := s'
             adv.adjust s'
```

(Suggested improvement over Krasic and Walpole)

```
data ForwardMsg a =
   NewWindow Time Time -- current phase and window size
 | Data (MovieFrame a)
 | Stop
```

PPS in Timber

Putting everything together:

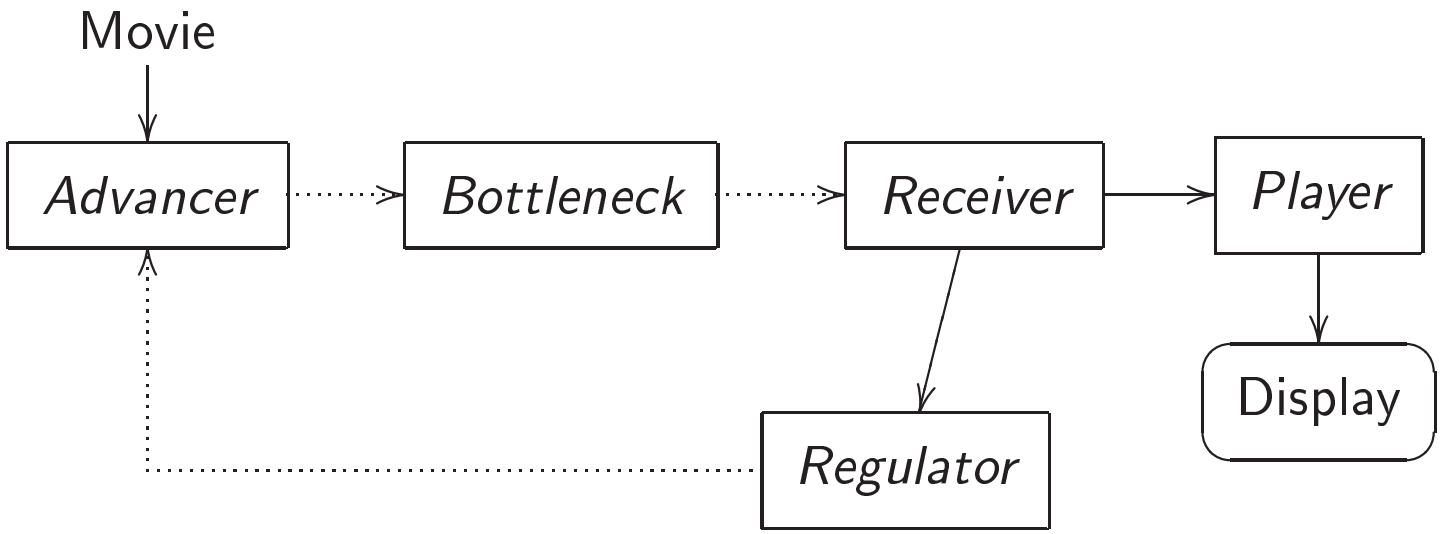



```
regulator :: Advancer -> Template Regulator

advancer  :: Bottleneck (ForwardMsg a)
          -> Movie a -> Template Advancer

receiver  :: Regulator -> Player a
                  -> Template (Receiver (ForwardMsg a))
```

PPS in Timber




```
tie :: Movie a -> Player a -> Template Advancer
tie m p =
  template
    adv <- advancer bn m reg
    bn  <- bottleneck r
    reg <- regulator adv
    r   <- receiver reg p
    return adv
```