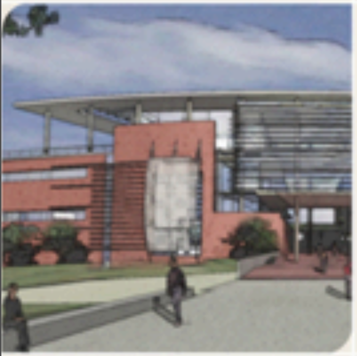


Dynamic Tuning of Feature Set in Highly Variant Interactive Applications

Tushar Kumar, Romain Cledat,

and Santosh Pande

Georgia Tech College of Computing

EMSOFT 2010

Motivation

Motivation

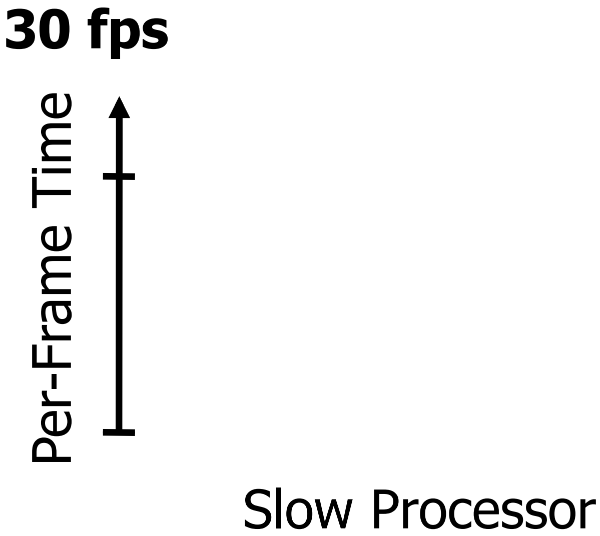
- An entire class of **frame-oriented, interactive applications** currently lack a systematic optimization methodology

Motivation

- An entire class of **frame-oriented**, **interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***

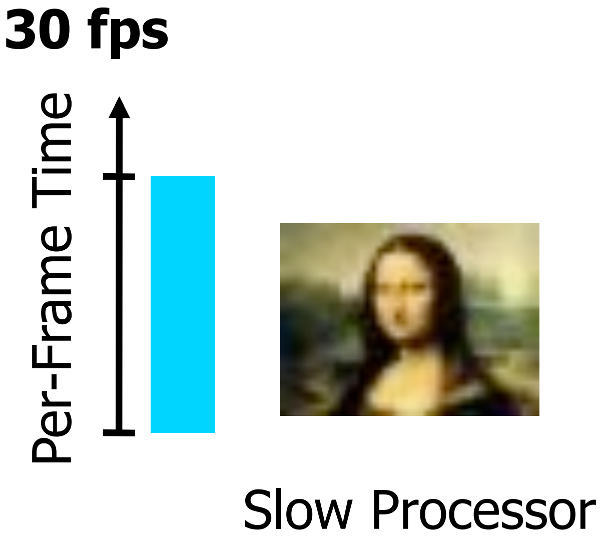
Motivation

- An entire class of **frame-oriented**, **interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***



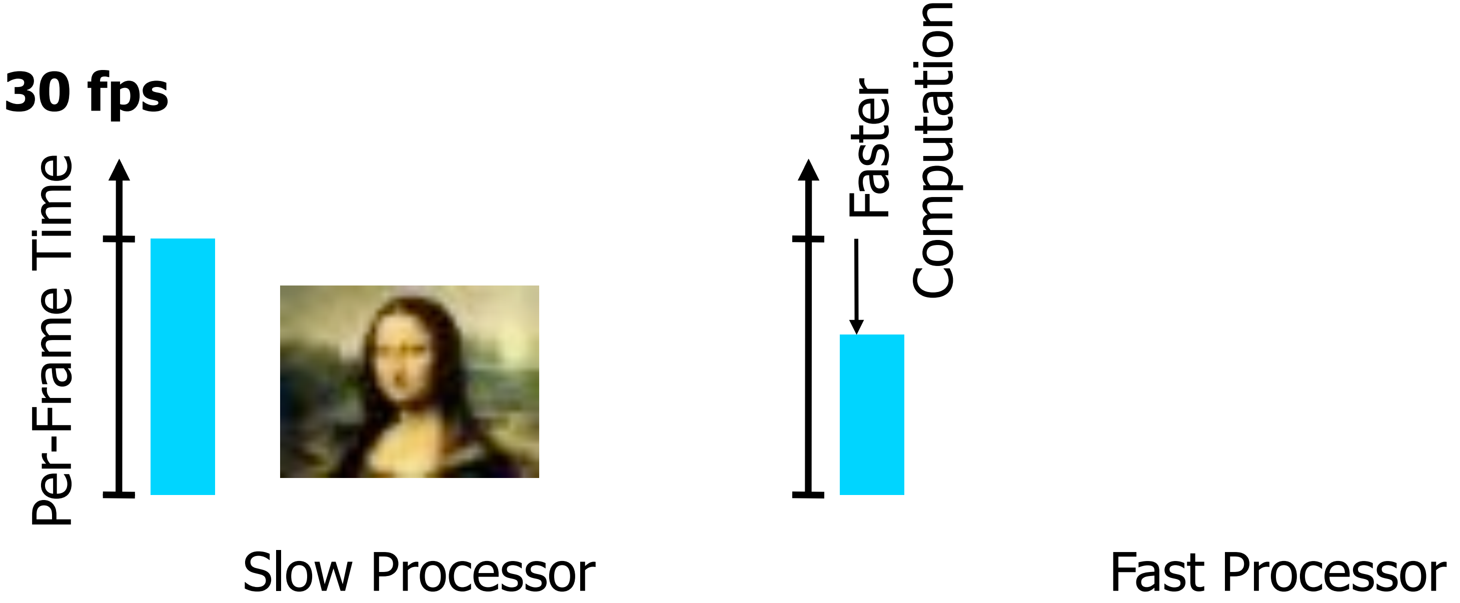
Motivation

- An entire class of **frame-oriented**, **interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***



Motivation

- An entire class of **frame-oriented, interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***



30 fps

Per-Frame Time

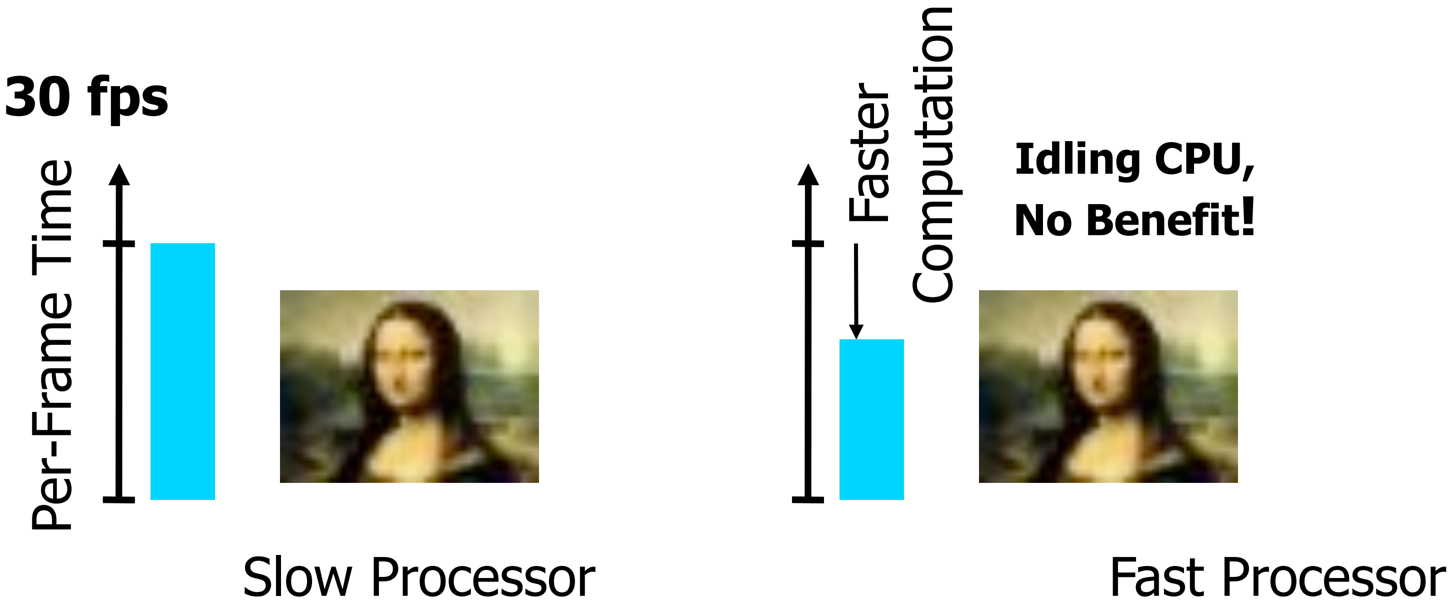
Slow Processor

Faster Computation

Fast Processor

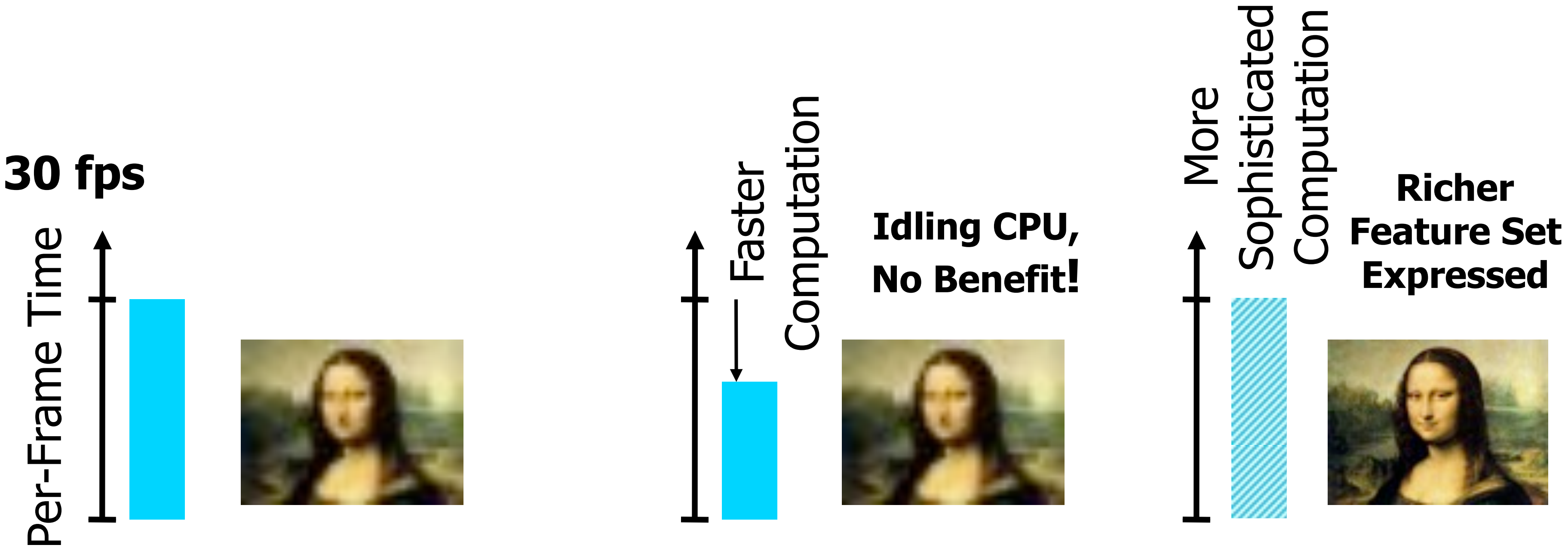
Motivation

- An entire class of **frame-oriented, interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***



Motivation

- An entire class of **frame-oriented**, **interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***

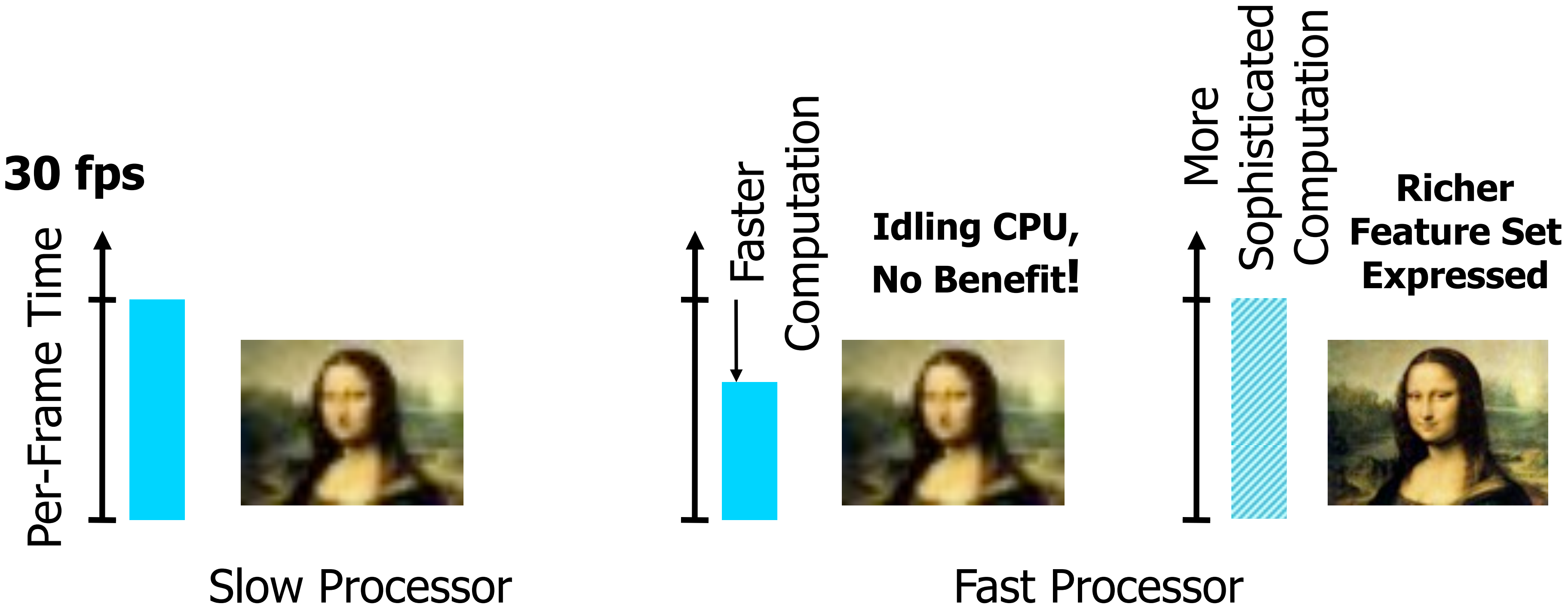


Slow Processor

Fast Processor

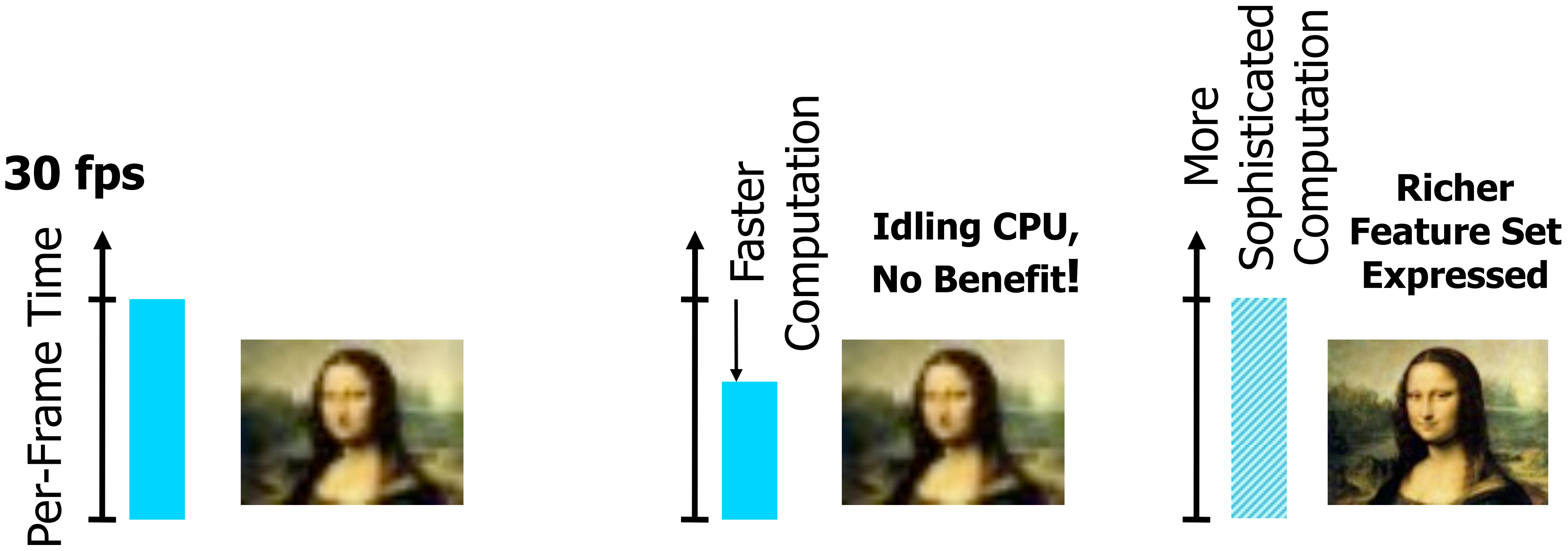
Motivation

- An entire class of **frame-oriented**, **interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***
- Design goal



Motivation

- An entire class of **frame-oriented, interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***
- Design goal
  - Provide the **richest, most engrossing experience** possible to the **interactive user**



30 fps

Per-Frame Time

Faster Computation

Idling CPU, No Benefit!

More Sophisticated Computation

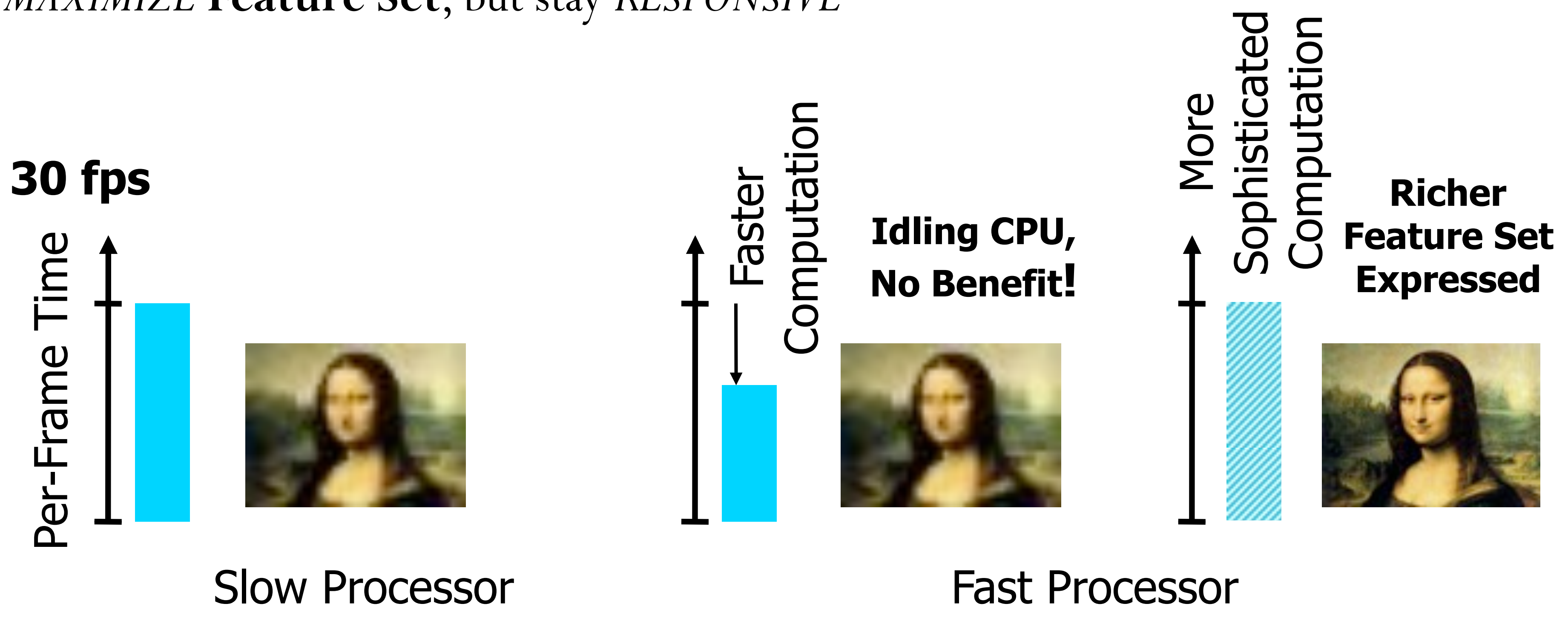
Richer Feature Set Expressed

Slow Processor

Fast Processor

Motivation

- An entire class of **frame-oriented, interactive applications** currently lack a systematic optimization methodology
  - Gaming, Multimedia, Interactive Visualization
  - Conventionally developed C/C++/Java applications ” ***lack analyzable semantics***
- Design goal
  - Provide the **richest, most engrossing experience** possible to the **interactive user**
  - *MAXIMIZE* **Feature Set**, but stay *RESPONSIVE*



30 fps

Per-Frame Time

Slow Processor

Faster Computation

Idling CPU, No Benefit!

More Sophisticated Computation

Richer Feature Set Expressed

Fast Processor

Representative Applications

Representative Applications

- MPEG2 Encoder

Representative Applications

- MPEG2 Encoder
  - Motion Estimation algorithm dominates per-frame time
  - ***Search-Window-Size*** parameter dramatically ***scales***:
    - Per-frame encoding time

Representative Applications

- MPEG2 Encoder
  - Motion Estimation algorithm dominates per-frame time
  - ***Search-Window-Size*** parameter dramatically ***scales***:
    - Per-frame encoding time
    - Achieved compression of video

Representative Applications

- MPEG2 Encoder
  - Motion Estimation algorithm dominates per-frame time
  - ***Search-Window-Size*** parameter dramatically ***scales***:
    - Per-frame encoding time
    - Achieved compression of video

- Torque Game Engine (www.torquepowered.com)
  - AI algorithms dominate per-frame time, when there are large number of simulated enemies (bots)
  - Multiple AI parameters ***scale***:

Representative Applications

- MPEG2 Encoder
  - Motion Estimation algorithm dominates per-frame time
  - ***Search-Window-Size*** parameter dramatically ***scales***:
    - Per-frame encoding time
    - Achieved compression of video

- Torque Game Engine (www.torquepowered.com)
  - AI algorithms dominate per-frame time, when there are large number of simulated enemies (bots)
  - Multiple AI parameters ***scale***:
    - Frame-rate
    - Intelligence exhibited by bots

Representative Applications

- MPEG2 Encoder
  - Motion Estimation algorithm dominates per-frame time
  - ***Search-Window-Size*** parameter dramatically ***scales***:
    - Per-frame encoding time
    - Achieved compression of video

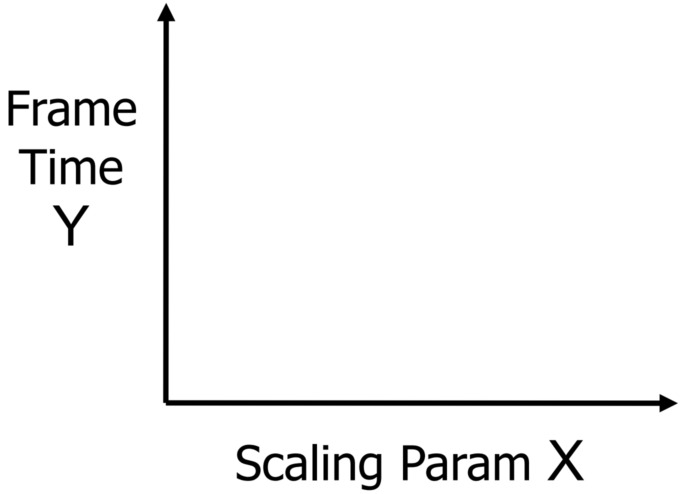
- Torque Game Engine (www.torquepowered.com)
  - AI algorithms dominate per-frame time, when there are large number of simulated enemies (bots)
  - Multiple AI parameters ***scale***:
    - Frame-rate
    - Intelligence exhibited by bots

- Scalable Algorithms are common and dominant

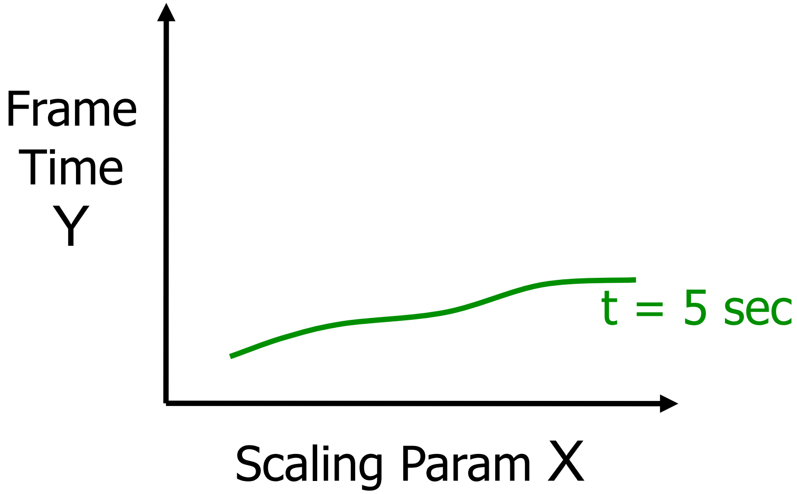
Representative Applications

- MPEG2 Encoder
  - Motion Estimation algorithm dominates per-frame time
  - ***Search-Window-Size*** parameter dramatically ***scales***:
    - Per-frame encoding time
    - Achieved compression of video
- Torque Game Engine (www.torquepowered.com)
  - AI algorithms dominate per-frame time, when there are large number of simulated enemies (bots)
  - Multiple AI parameters ***scale***:
    - Frame-rate
    - Intelligence exhibited by bots
- Scalable Algorithms are common and dominant
  - Let X denote the *scaling parameter* (application-specific choice)

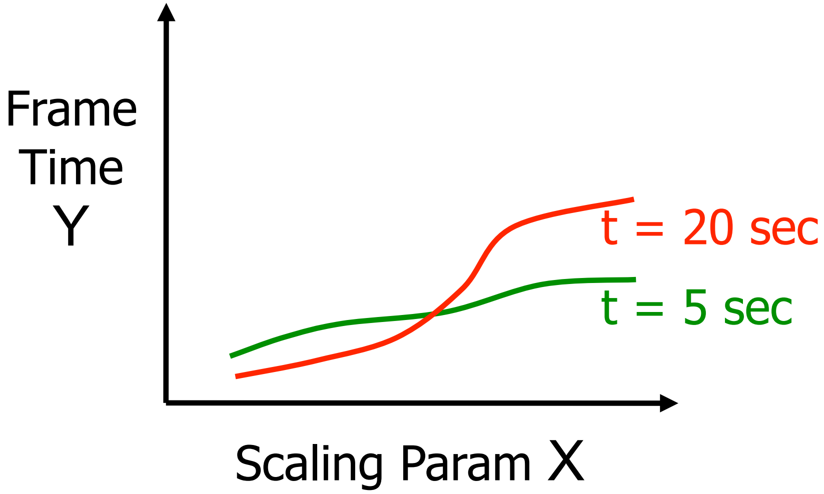
Challenges in Tuning Feature Set



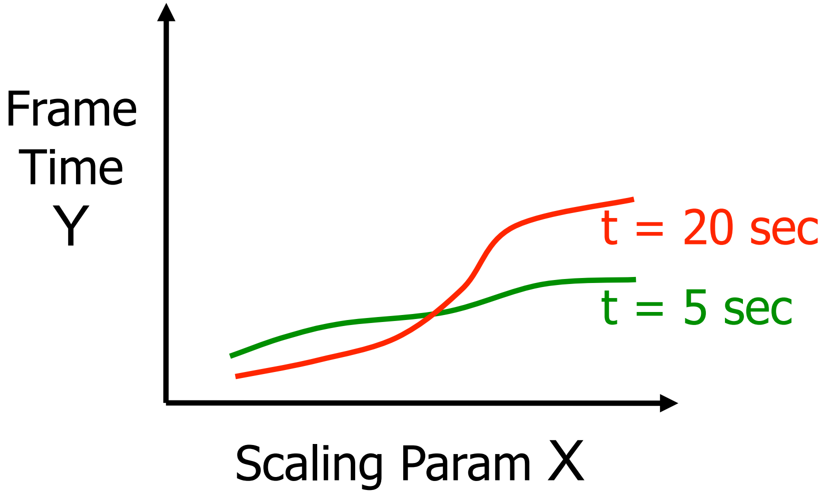
Challenges in Tuning Feature Set



Challenges in Tuning Feature Set

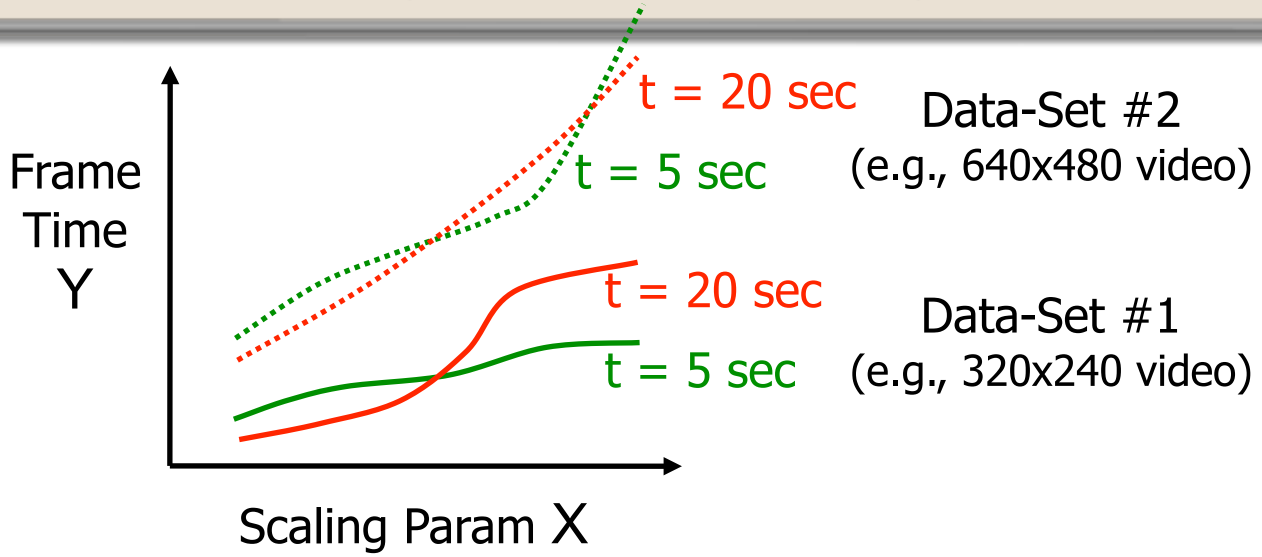


Challenges in Tuning Feature Set

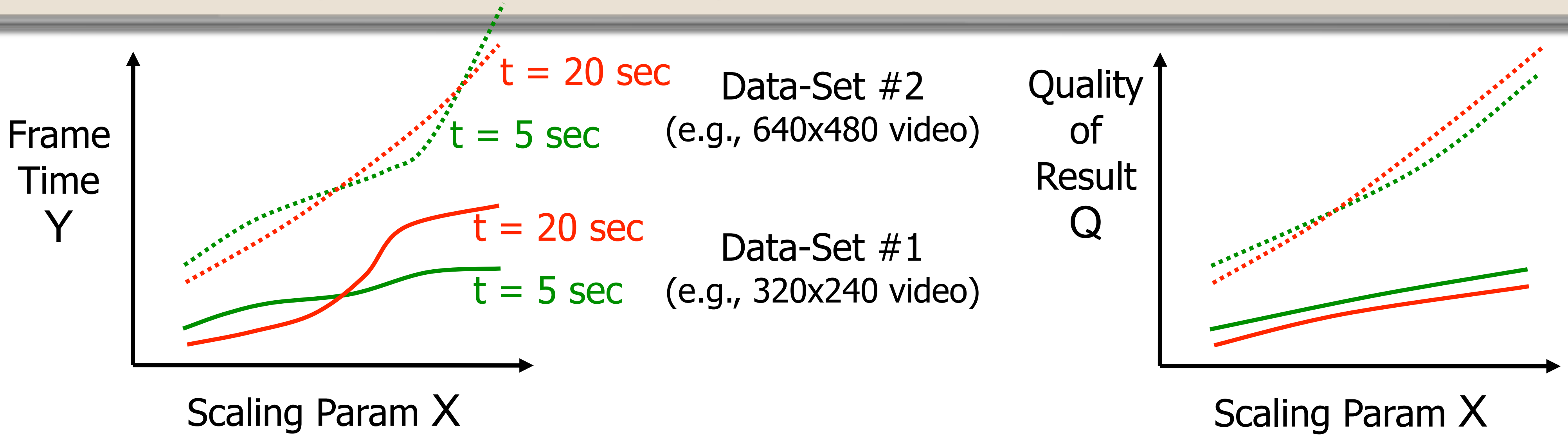


Data-Set #1
(e.g., 320x240 video)

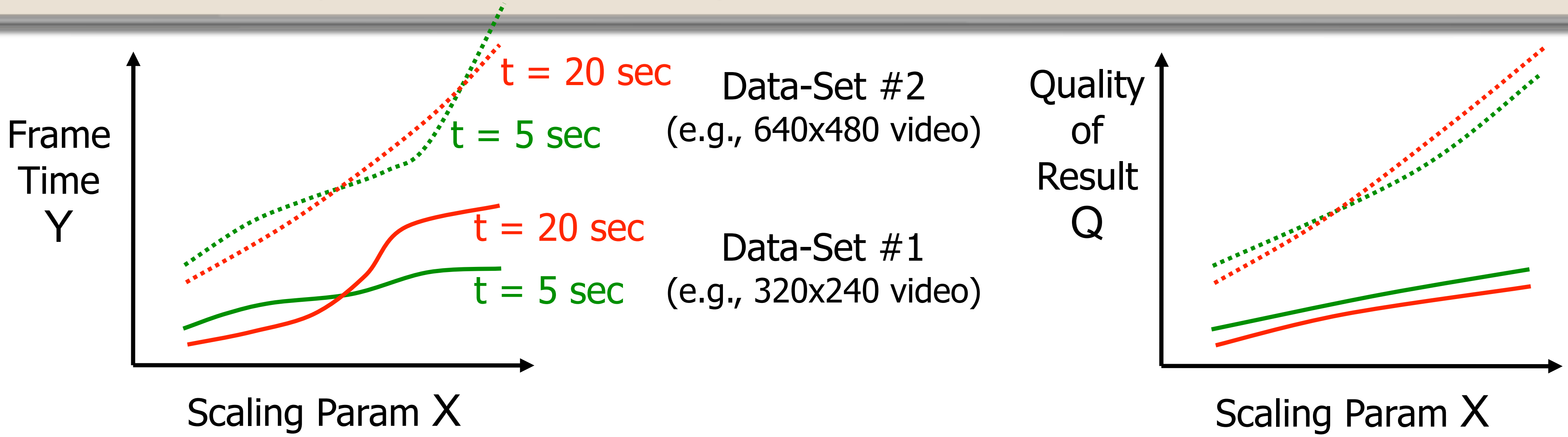
Challenges in Tuning Feature Set



Challenges in Tuning Feature Set



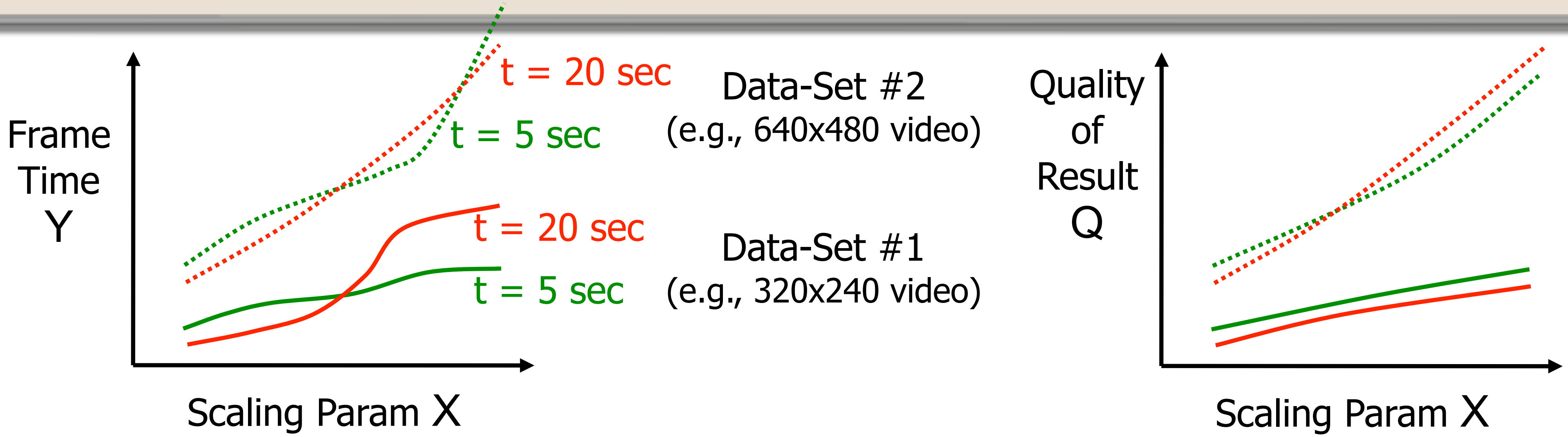
Challenges in Tuning Feature Set



X-Y, X-Q Response Characteristics

- Highly time-varying, data-set dependent ” ***No fixed relationships***

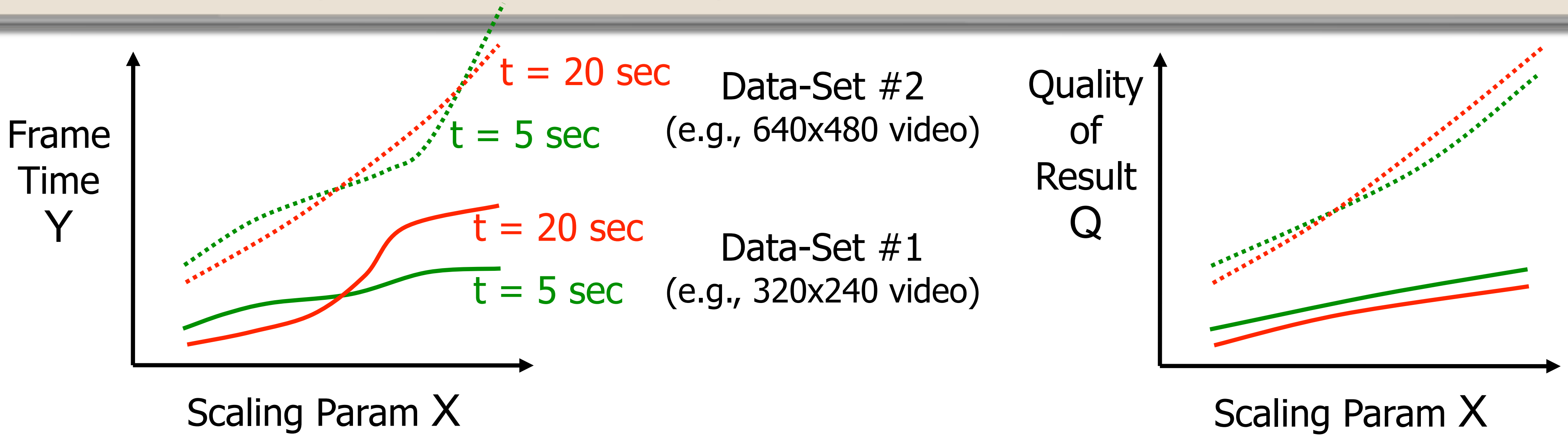
Challenges in Tuning Feature Set



X-Y, X-Q Response Characteristics

- Highly time-varying, data-set dependent ” ***No fixed relationships***
- For practical purposes: UNKNOWN, entirely EMERGENT behavior

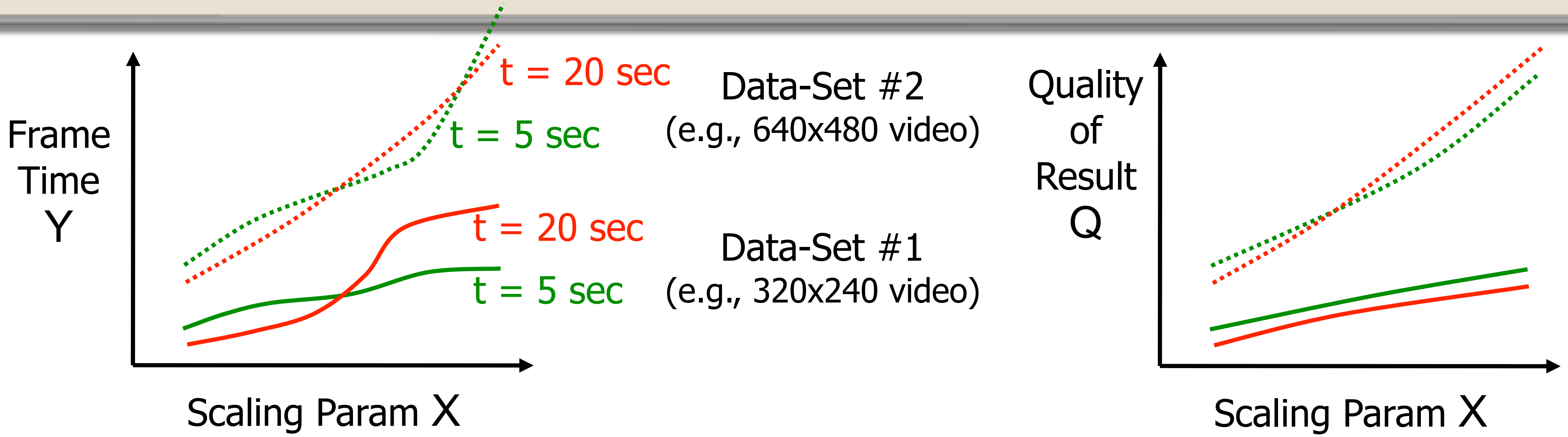
Challenges in Tuning Feature Set



X-Y, X-Q Response Characteristics

- Highly time-varying, data-set dependent ” ***No fixed relationships***
- For practical purposes: UNKNOWN, entirely EMERGENT behavior

Challenges in Tuning Feature Set



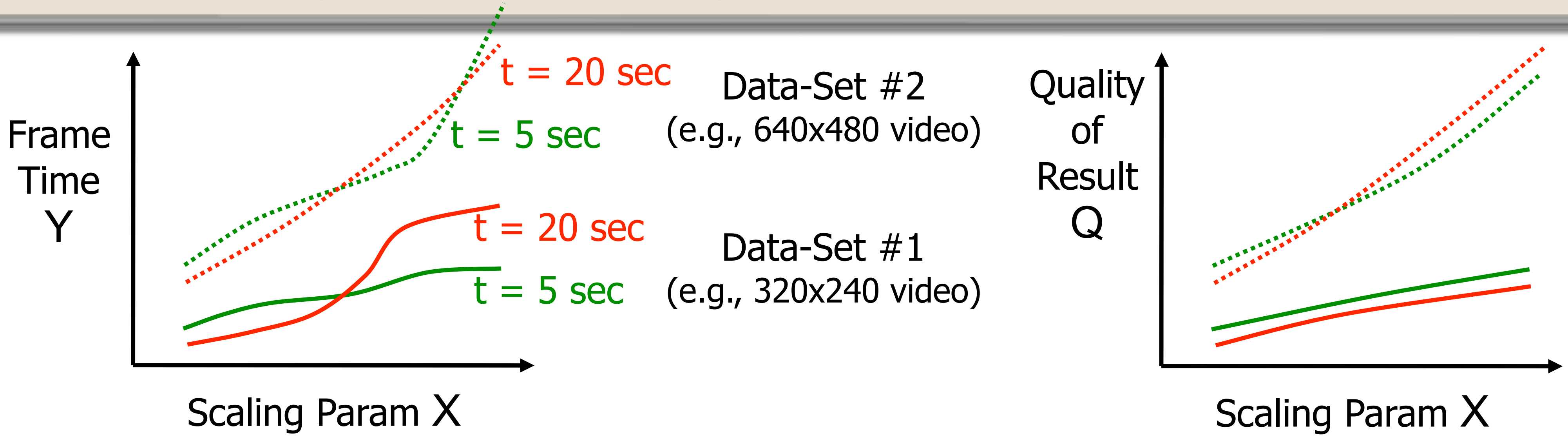
X-Y, X-Q Response Characteristics

- Highly time-varying, data-set dependent ” ***No fixed relationships***
- For practical purposes: UNKNOWN, entirely EMERGENT behavior

So how do programmers currently tune?

- Impose ***severe limitations***:
  - Video surveillance (slower motion, fixed background), at 320x240 resolution
  - Then: Fix X to ***least harmful*** value

Challenges in Tuning Feature Set



X-Y, X-Q Response Characteristics

- Highly time-varying, data-set dependent ” ***No fixed relationships***
- For practical purposes: UNKNOWN, entirely EMERGENT behavior

So how do programmers currently tune?

- Impose ***severe limitations***:
  - Video surveillance (slower motion, fixed background), at 320x240 resolution
  - Then: Fix X to ***least harmful*** value
- Or, manually tune to ***each game-play scenario***, tune for Xbox vs PS3

Our Approach

Our Approach

- Premise: Modeling emergent response behavior is hard!

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
  - Identifies application-specific scale parameter X (WHAT to scale)

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
  - Identifies application-specific scale parameter X (WHAT to scale)
  - Specifies desired frame-rate Y to achieve (WHAT to achieve)

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
  - Identifies application-specific scale parameter X (WHAT to scale)
  - Specifies desired frame-rate Y to achieve (WHAT to achieve)

    (Implicit: Quality-of-Result Q scales with Y)

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
  - Identifies application-specific scale parameter X (WHAT to scale)
  - Specifies desired frame-rate Y to achieve (WHAT to achieve)

    (Implicit: Quality-of-Result Q scales with Y)

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
  - Identifies application-specific scale parameter X (WHAT to scale)
  - Specifies desired frame-rate Y to achieve (WHAT to achieve)

    (Implicit: Quality-of-Result Q scales with Y)

- HOW to achieve?

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
  - Identifies application-specific scale parameter X (WHAT to scale)
  - Specifies desired frame-rate Y to achieve (WHAT to achieve)

    (Implicit: Quality-of-Result Q scales with Y)

- HOW to achieve?
  - *HARD*: Usually beyond scope of most programmers and even domain experts

Our Approach

- Premise: Modeling emergent response behavior is hard!
  - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
  - Identifies application-specific scale parameter X (WHAT to scale)
  - Specifies desired frame-rate Y to achieve (WHAT to achieve)

    (Implicit: Quality-of-Result Q scales with Y)

- HOW to achieve?
  - *HARD*: Usually beyond scope of most programmers and even domain experts

What will it take for a ***generic, broadly applicable Runtime Controller*** *to effectively control frame-QoS?*

Our Approach

- Premise: Modeling emergent response behavior is hard!
    - Infeasible for programmers and even domain experts

- Place only an *EASY* burden on the Programmer:
    - Identifies application-specific scale parameter X (WHAT to scale)
    - Specifies desired frame-rate Y to achieve (WHAT to achieve)

      (Implicit: Quality-of-Result Q scales with Y)

Main Contribution

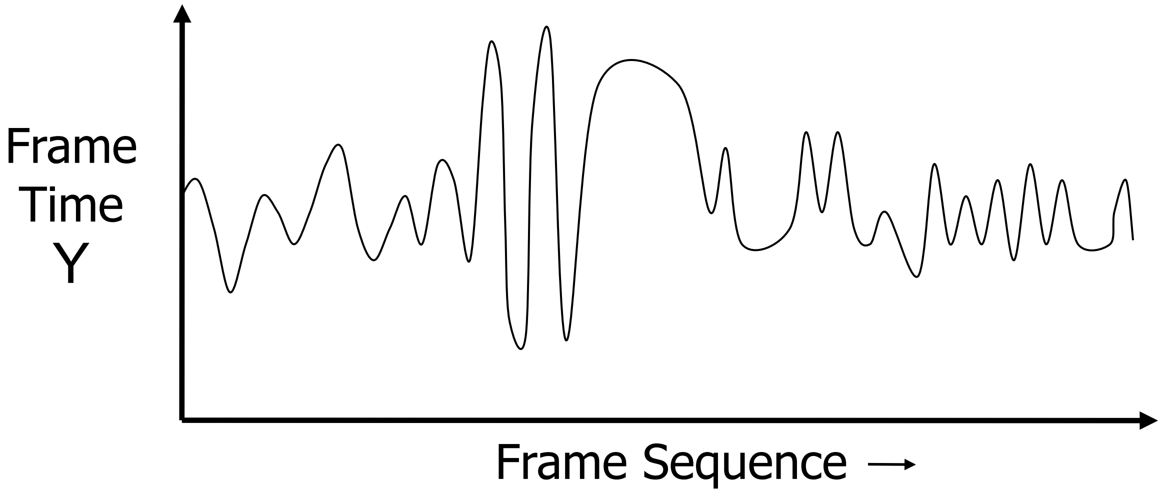
Simplicity of Use,

Generality of Application

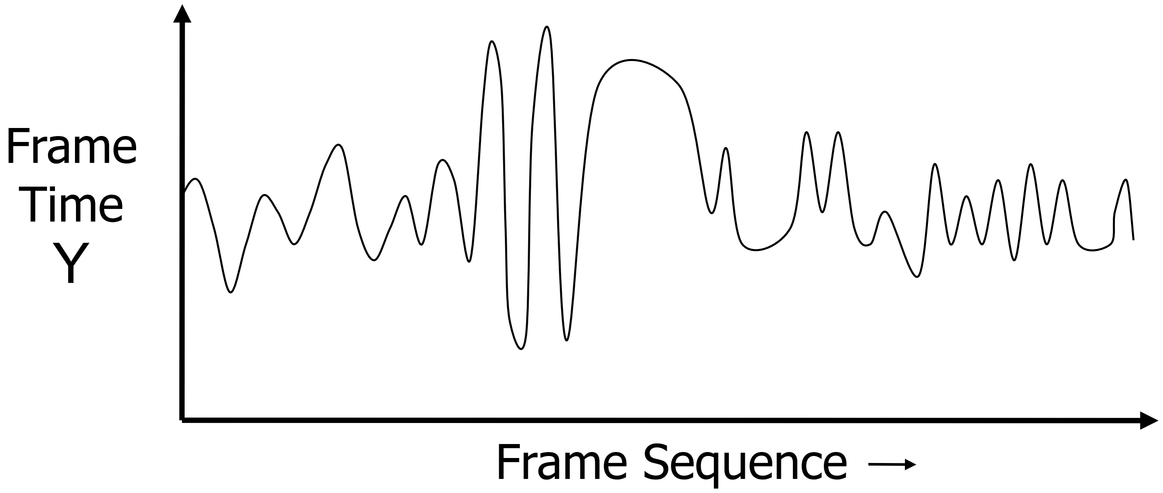
- HOW to achieve?
    - *HARD*: Usually beyond scope of most programmers and even domain experts

What will it take for a ***generic, broadly applicable Runtime Controller*** *to effectively control frame-QoS?*

Control Problem

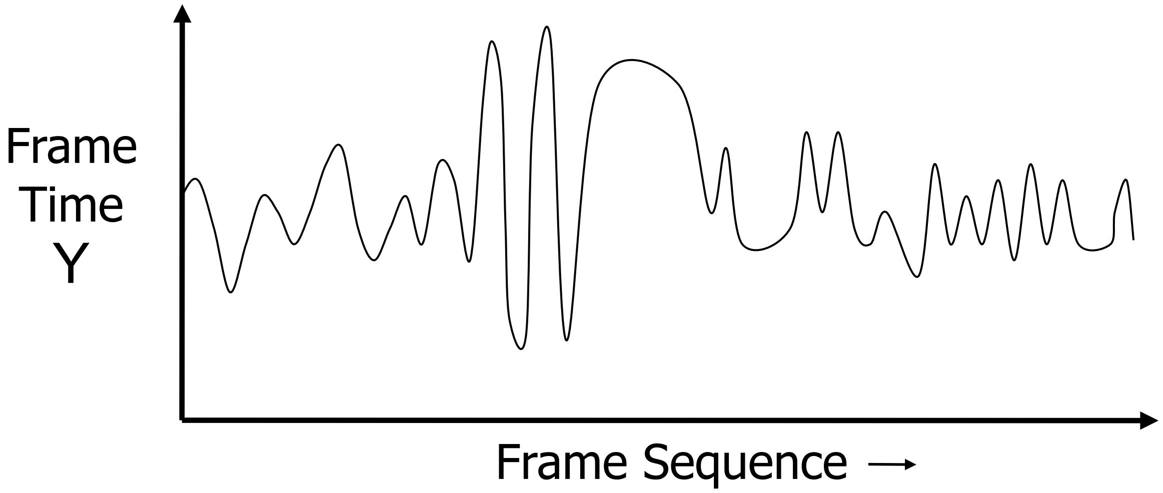


Control Problem



- Soft Real Time nature

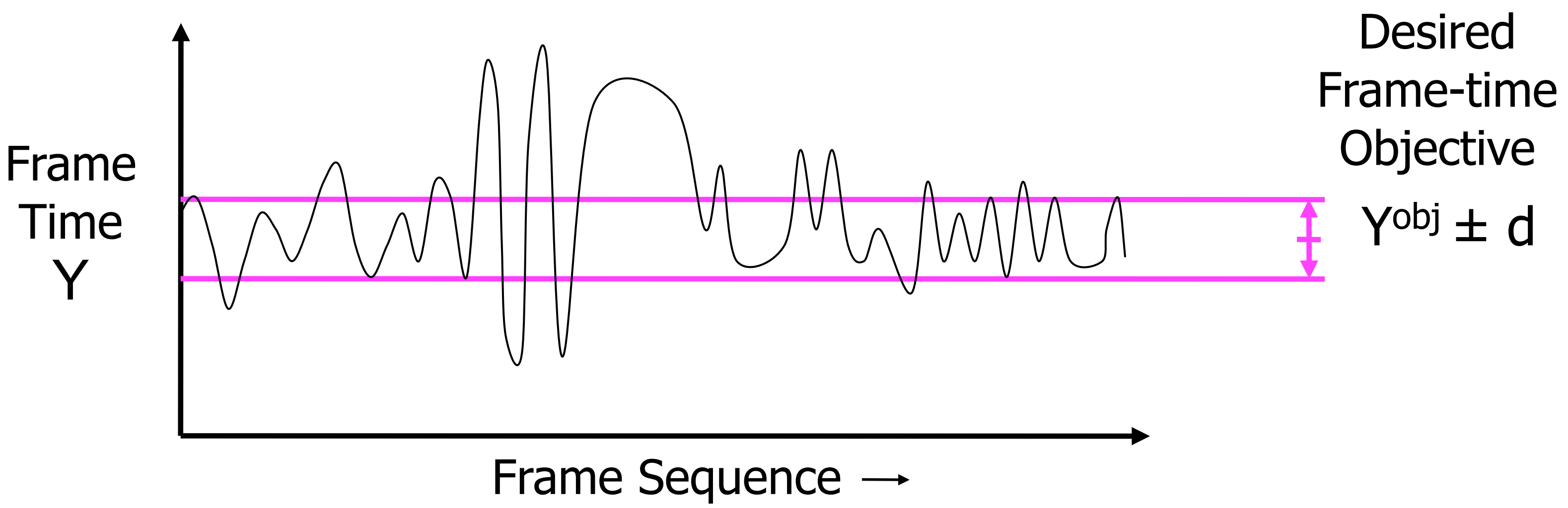
Control Problem



- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

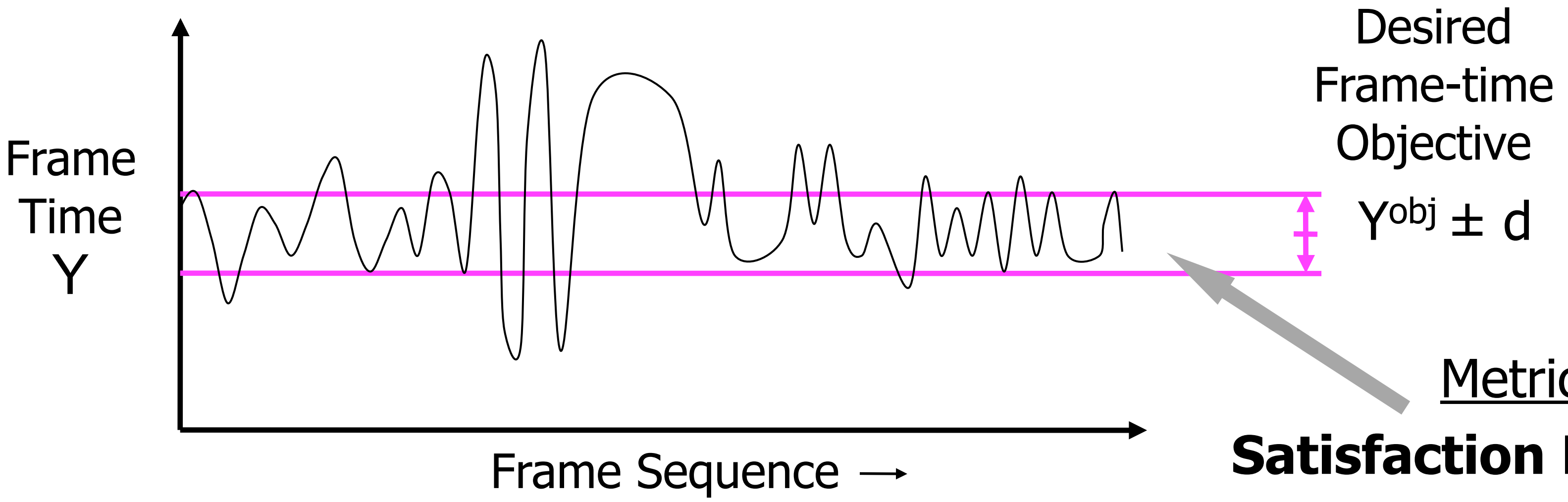
Control Problem



- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

Control Problem



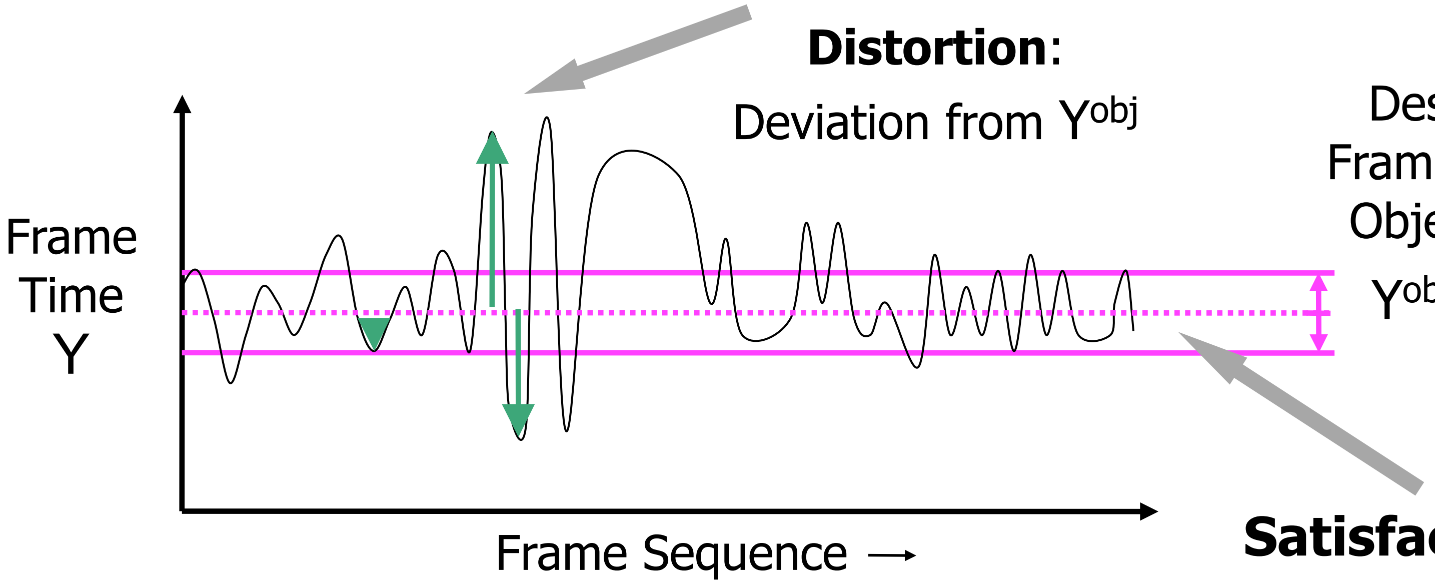
<u>Metric#1</u>

Satisfaction Ratio (SR):

Fraction of frames with Y within $Y^{obj} \pm d$

- Soft Real Time nature
  - Probabilistic/best-effort suffices
    "Maintain 25-35 frames-per-second with high probability"

Control Problem



Metric#2

Distortion:

Deviation from Y^{obj}

Desired Frame-time Objective

$Y^{obj} \pm d$

Metric#1

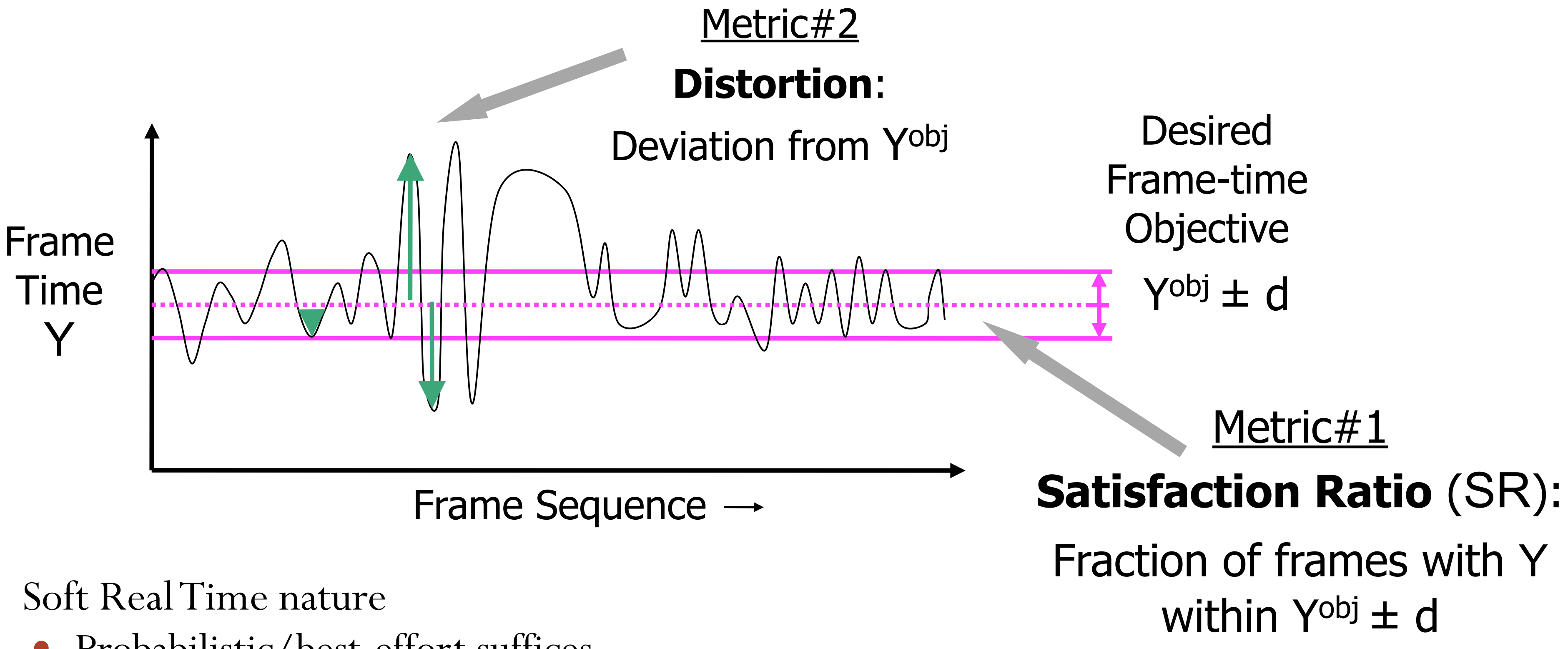
Satisfaction Ratio (SR):

Fraction of frames with Y within $Y^{obj} \pm d$

- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

Control Problem



<u>Metric#2</u>

Distortion:

Deviation from Y^{obj}

Desired Frame-time Objective

$Y^{obj} \pm d$

<u>Metric#1</u>

Satisfaction Ratio (SR):

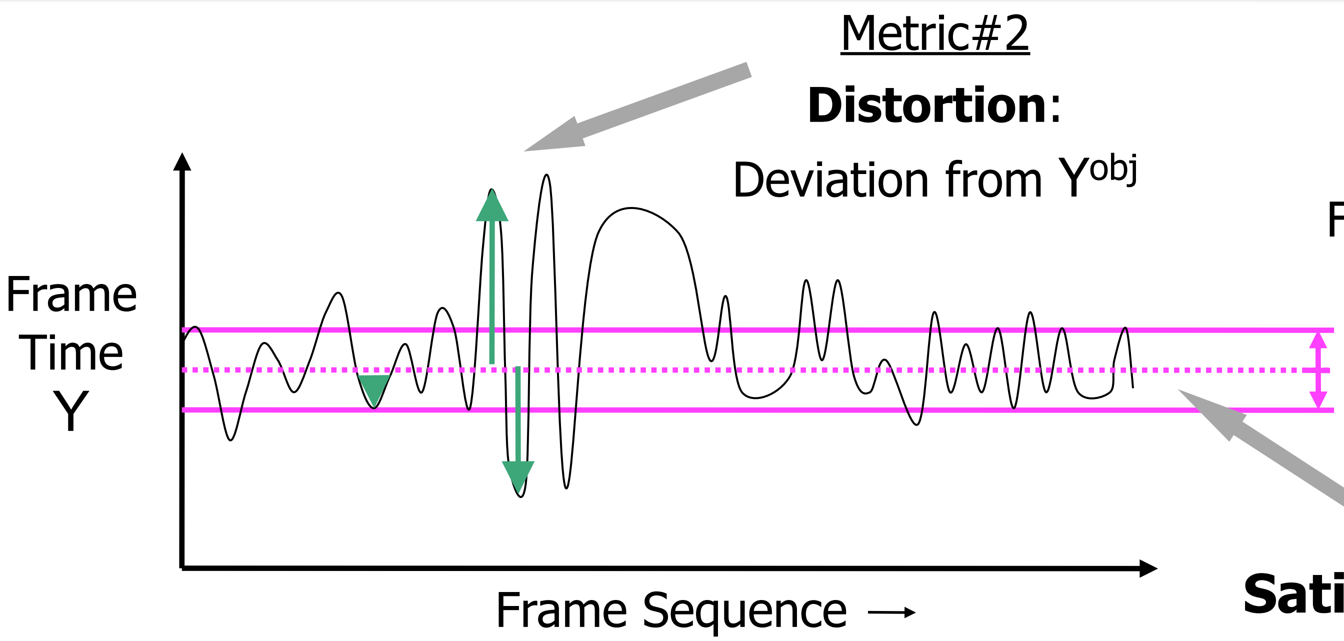
Fraction of frames with Y within $Y^{obj} \pm d$

- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

- What's a good SR on a data-set?

Control Problem



Metric#2

Distortion:

Deviation from Y^{obj}

Desired Frame-time Objective

Y^{obj} ± d

Metric#1

Satisfaction Ratio (SR):

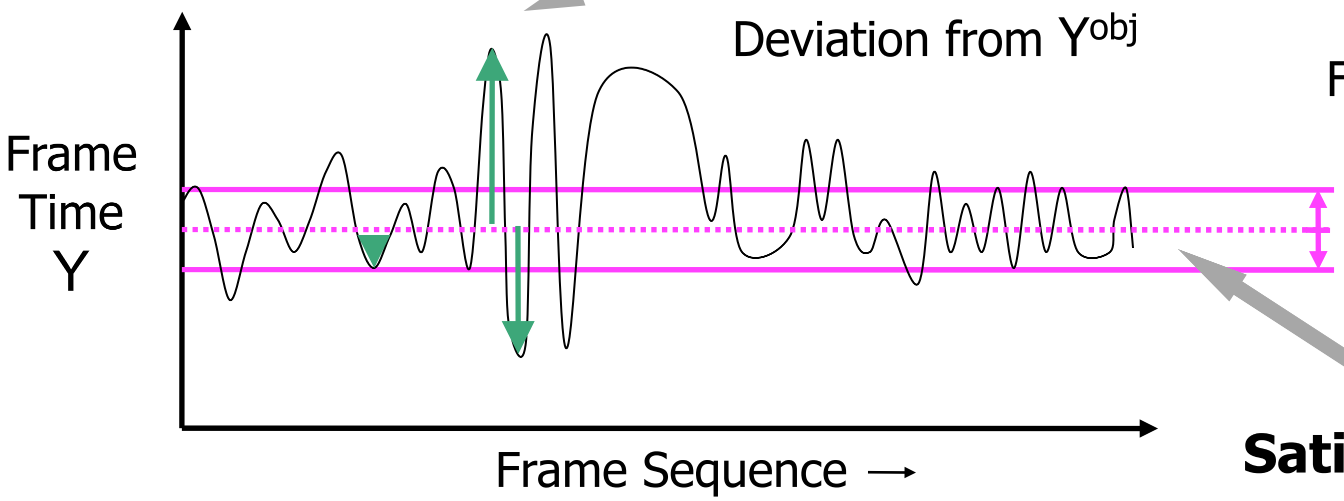
Fraction of frames with Y within Y^{obj} ± d

- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

- What's a good SR on a data-set?
  - *30%, 50%, 90%, 99%?*

Control Problem



Metric#2

Distortion:

Deviation from Y^{obj}

Desired Frame-time Objective

Y^{obj} ± d

Metric#1

Satisfaction Ratio (SR):

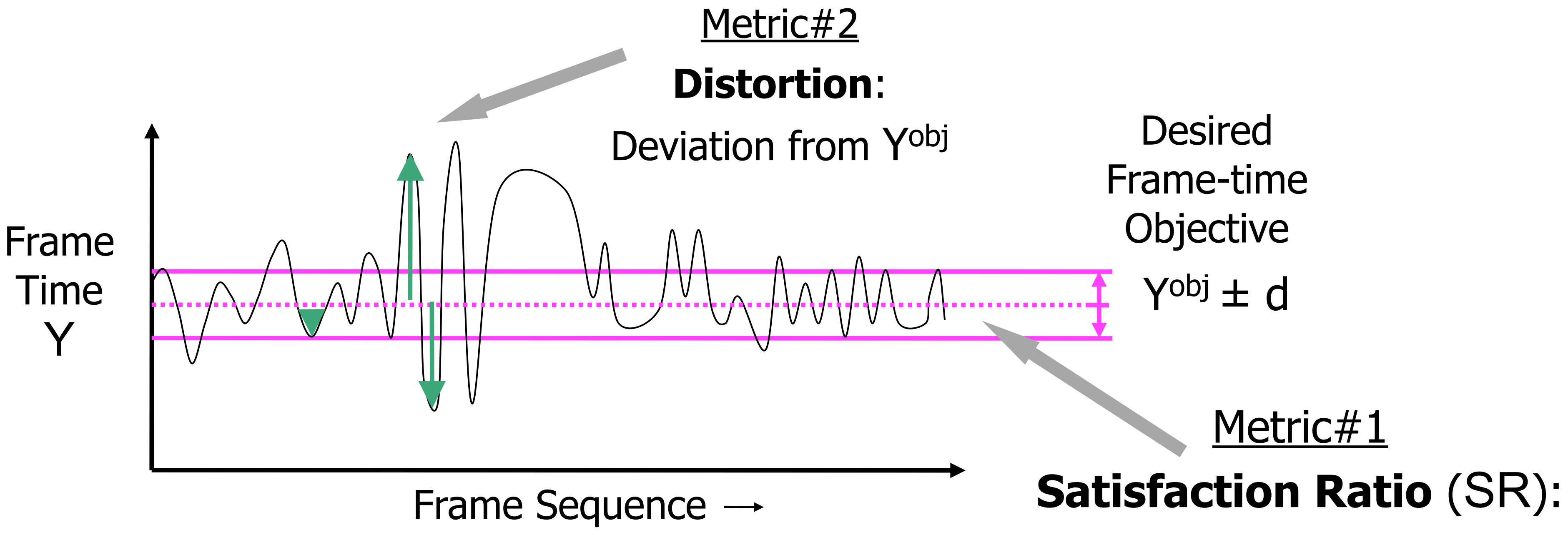
Fraction of frames with Y within Y^{obj} ± d

- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

- What's a good SR on a data-set?
  - *30%, 50%, 90%, 99%?*
  - A specified objective Y^{obj} ± d may be ***impossible*** to achieve

Control Problem



Metric#2

Distortion:

Deviation from Y^{obj}

Desired Frame-time Objective

Y^{obj} ± d

Metric#1

Satisfaction Ratio (SR):

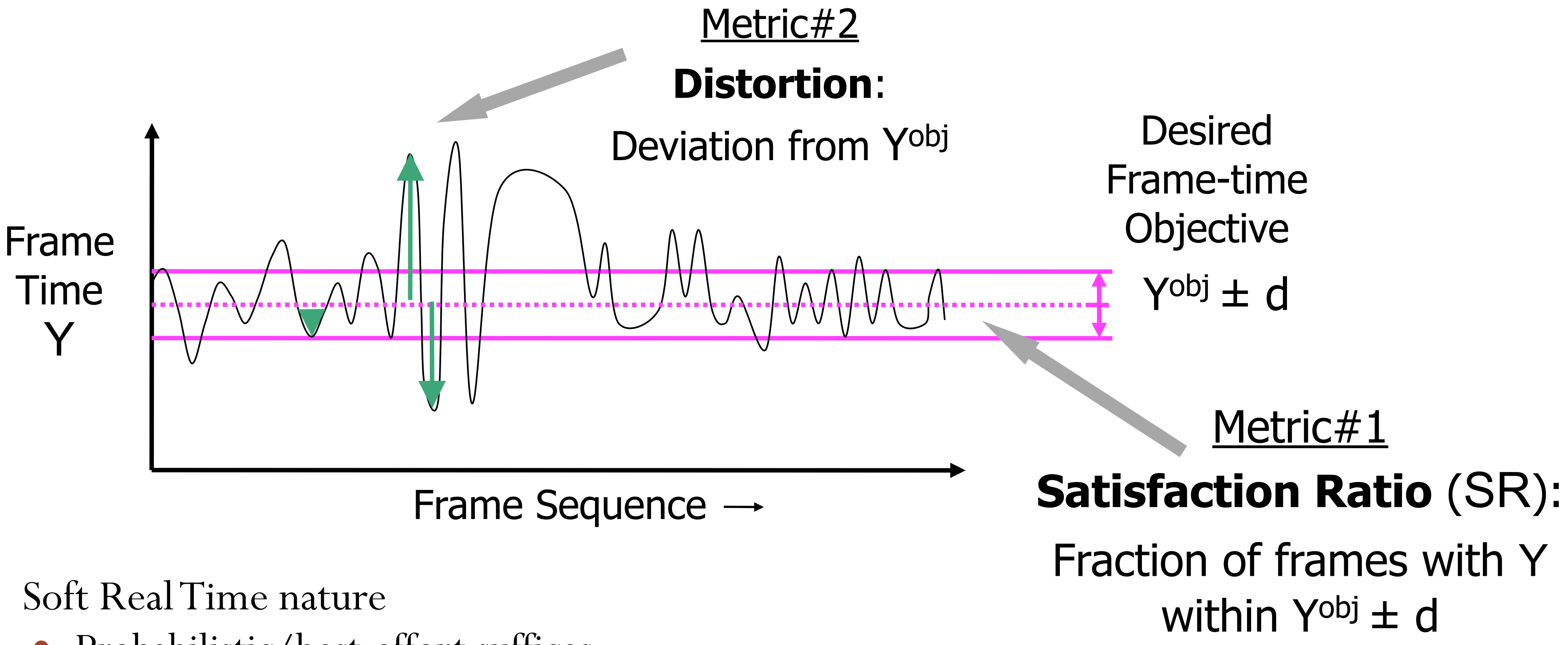
Fraction of frames with Y within Y^{obj} ± d

- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

- What's a good SR on a data-set?
  - *30%, 50%, 90%, 99%?*
  - A specified objective Y^{obj} ± d may be ***impossible*** to achieve
  - In general, application characteristics are unknown, \ best feasible SR is unknown

Control Problem



Metric#2

Distortion:

Deviation from Y^{obj}

Desired Frame-time Objective

Y^{obj} ± d

Metric#1

Satisfaction Ratio (SR):

Fraction of frames with Y within Y^{obj} ± d

- Soft Real Time nature
  - Probabilistic/best-effort suffices

    "Maintain 25-35 frames-per-second with high probability"

- What's a good SR on a data-set?
  - *30%, 50%, 90%, 99%?*
  - A specified objective Y^{obj} ± d may be ***impossible*** to achieve
  - In general, application characteristics are unknown, \ best feasible SR is unknown

” Only option: compare against SR when X has ***best fixed setting***

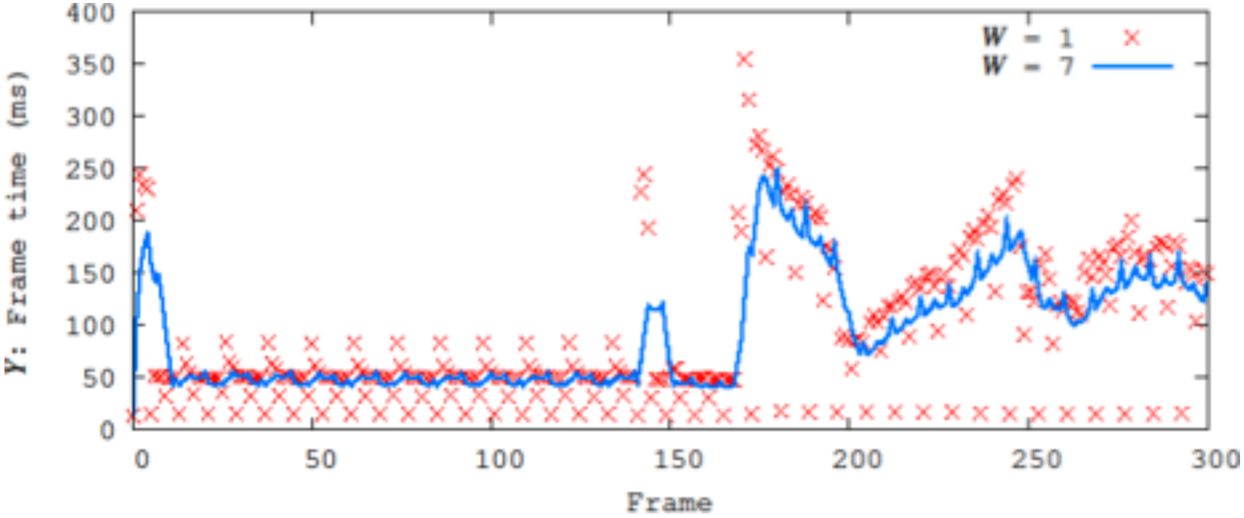
Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

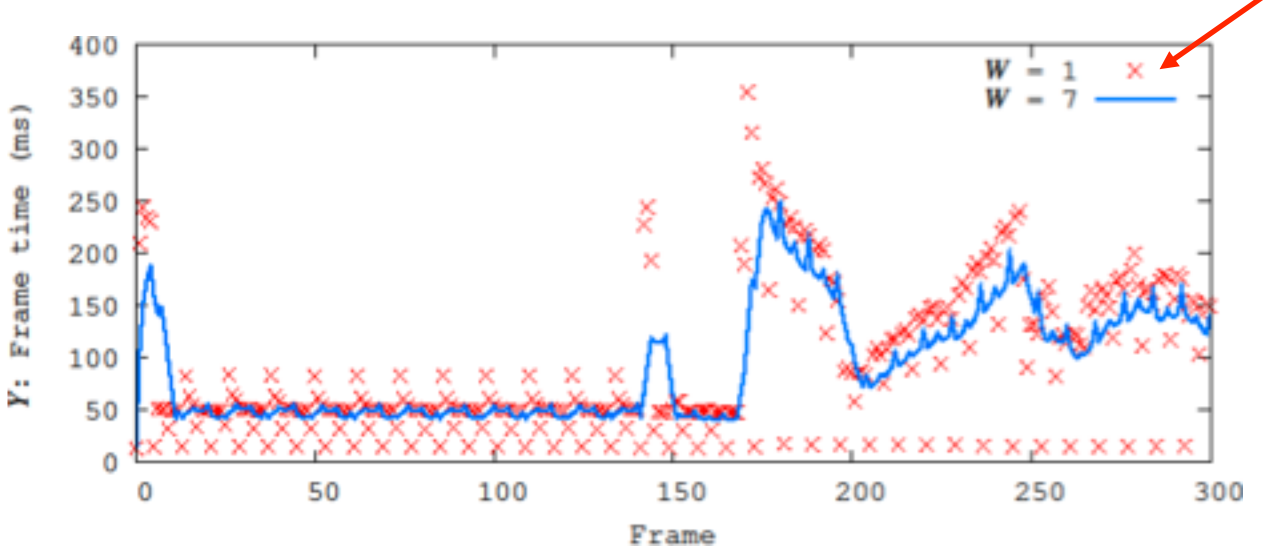
MPEG2 Encoder Frame Sequence (Fixed X)



Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

MPEG2 Encoder Frame Sequence (Fixed X)

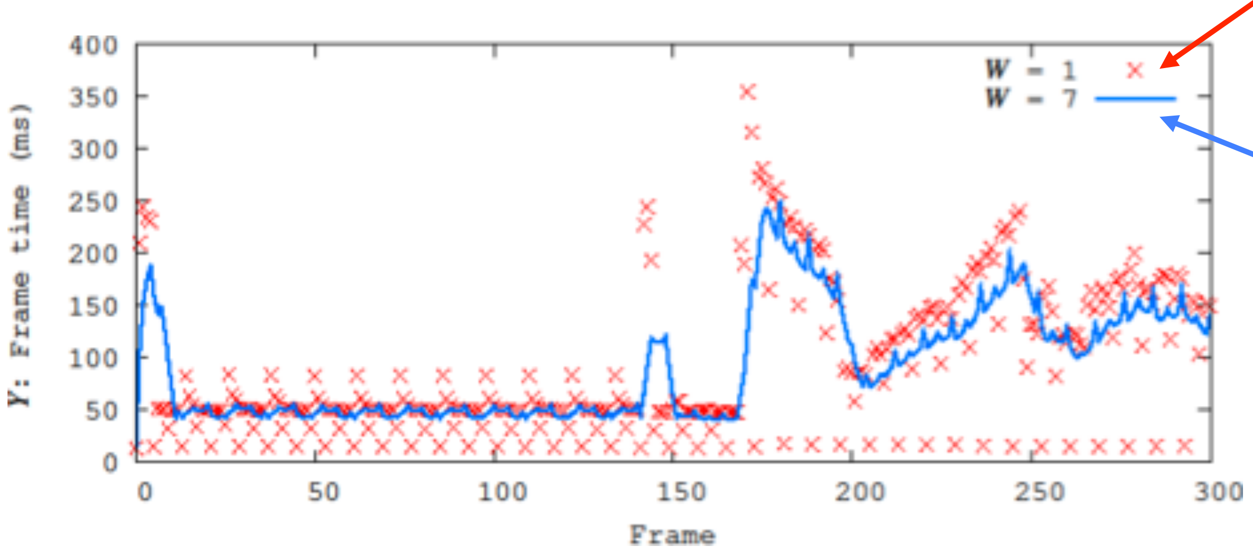


Instantaneous frame-times have significant transients *(Not perceived by user!)*

Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

MPEG2 Encoder Frame Sequence (Fixed X)



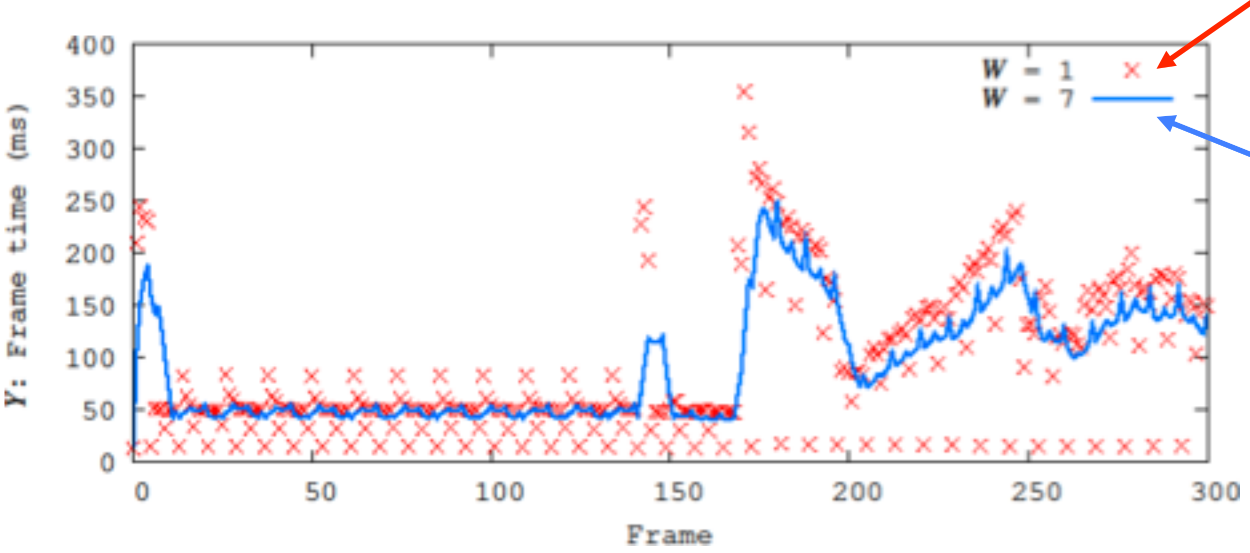
Instantaneous frame-times have significant transients *(Not perceived by user!)*

Moving average of previous 7 frames is mostly smooth *(Closer to user perception)*

Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

MPEG2 Encoder Frame Sequence (Fixed X)



Instantaneous frame-times have significant transients *(Not perceived by user!)*

Moving average of previous 7 frames is mostly smooth *(Closer to user perception)*

Lesson#1

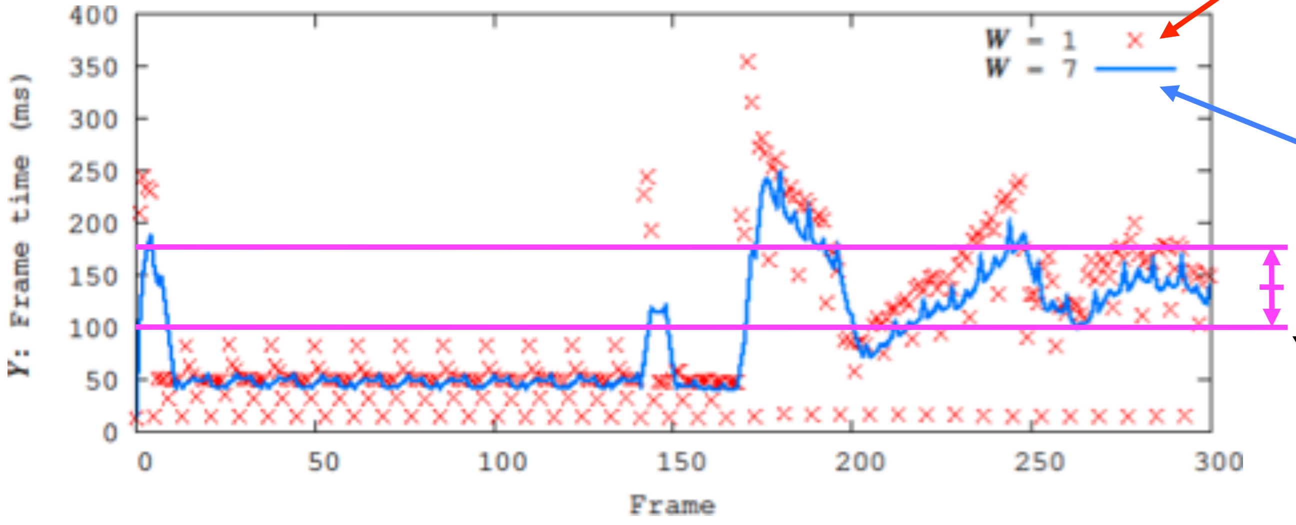
-Control **perceived** frame-time (much easier)

-Programmer specifies: Window of Perception W

Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

MPEG2 Encoder Frame Sequence (Fixed X)



Instantaneous frame-times have significant transients *(Not perceived by user!)*

Moving average of previous 7 frames is mostly smooth *(Closer to user perception)*

$Y^{obj} \pm d$

Lesson#1

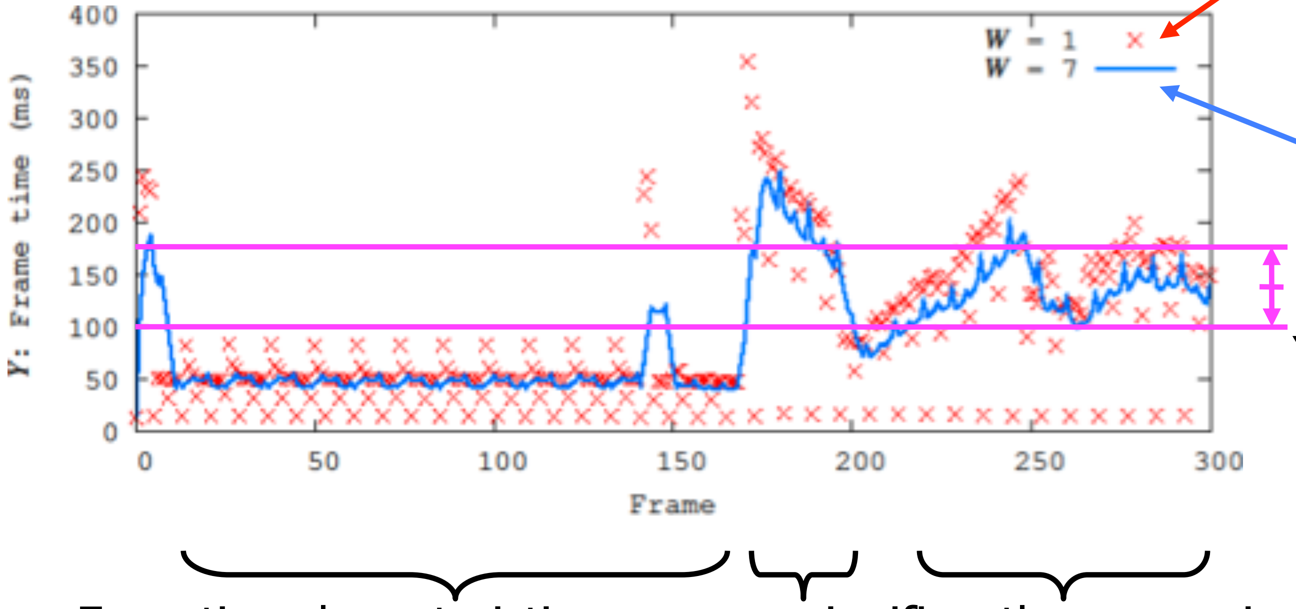
-Control **perceived** frame-time (much easier)

-Programmer specifies: Window of Perception W

Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

MPEG2 Encoder Frame Sequence (Fixed X)



Instantaneous frame-times have significant transients *(Not perceived by user!)*

Moving average of previous 7 frames is mostly smooth *(Closer to user perception)*

$Y^{obj} \pm d$

Execution characteristics can vary significantly over regions (**Regional Dependence**)

Lesson#1

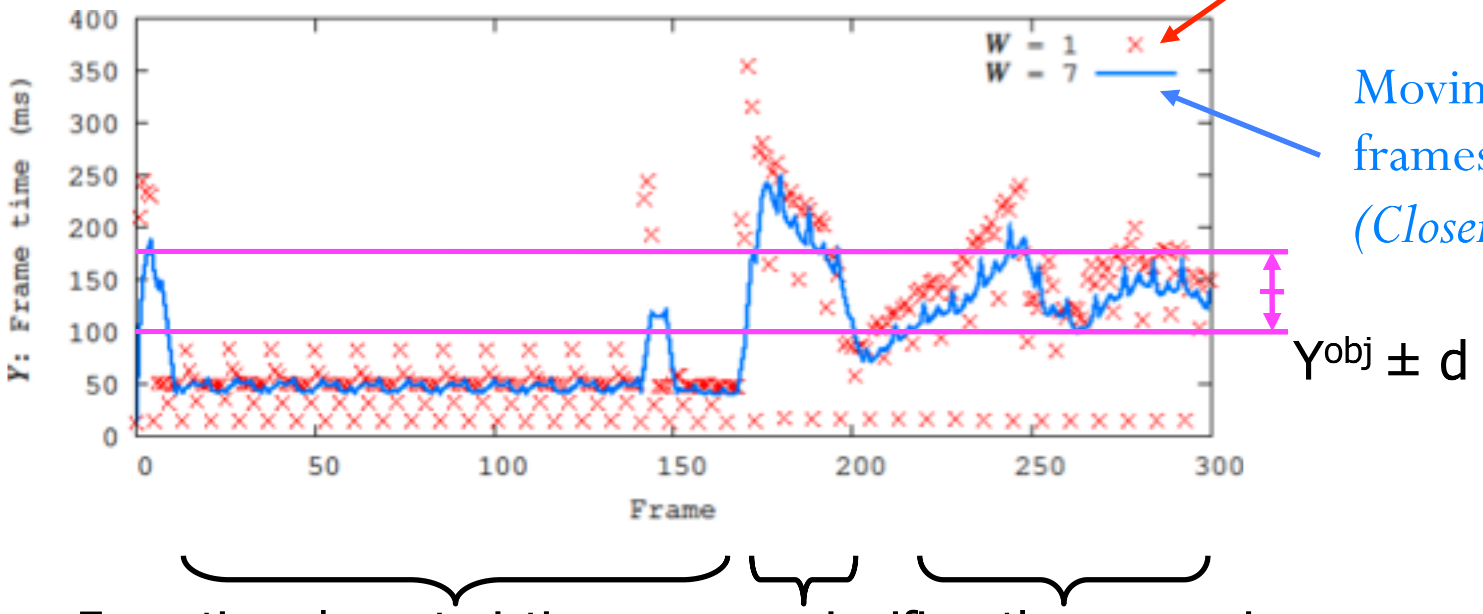
-Control **perceived** frame-time (much easier)

-Programmer specifies: Window of Perception W

Runtime Controller: Attempt#1

- Goal: Find ***best fixed*** X for each data-set
  - Highly Sub-Optimal!

MPEG2 Encoder Frame Sequence (Fixed X)



Instantaneous frame-times have significant transients *(Not perceived by user!)*

Moving average of previous 7 frames is mostly smooth *(Closer to user perception)*

Execution characteristics can vary significantly over regions (**Regional Dependence**)

Lesson#2

Train X to each region!

Lesson#1

-Control **perceived** frame-time (much easier)

-Programmer specifies: Window of Perception W

Runtime Controller: Attempt#2

Runtime Controller: Attempt#2

- Goal: Train X to each region

Runtime Controller: Attempt#2

- Goal: Train X to each region
- Reinforcement learning of X-Y relationship
  - Too slow!

Runtime Controller: Attempt#2

- Goal: Train X to each region
- Reinforcement learning of X-Y relationship
  - Too slow!

    X-Y relationships in MPEG2 Encoder and Torque Game usually change before learning can be used to subsequently control Y
- Find good X for region before X-Y relationship changes significantly

Runtime Controller: Attempt#2

- Goal: Train X to each region

- Reinforcement learning of X-Y relationship
  - Too slow!

    X-Y relationships in MPEG2 Encoder and Torque Game usually change before learning can be used to subsequently control Y

- Find good X for region before X-Y relationship changes significantly
- Then exploit good X for rest of (hopefully long) region ” achieves high SR

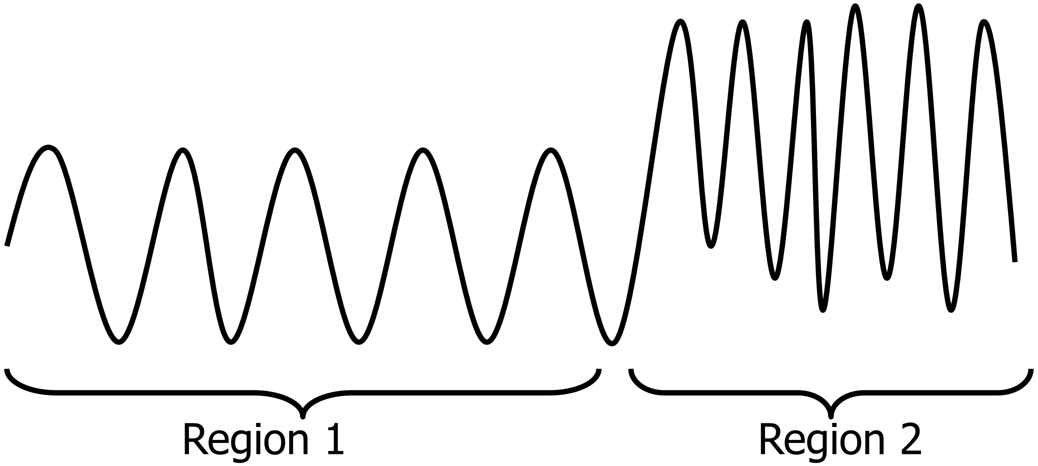
Runtime Controller: Attempt#2

- Goal: Train X to each region

- Reinforcement learning of X-Y relationship
  - Too slow!

    X-Y relationships in MPEG2 Encoder and Torque Game usually change before learning can be used to subsequently control Y

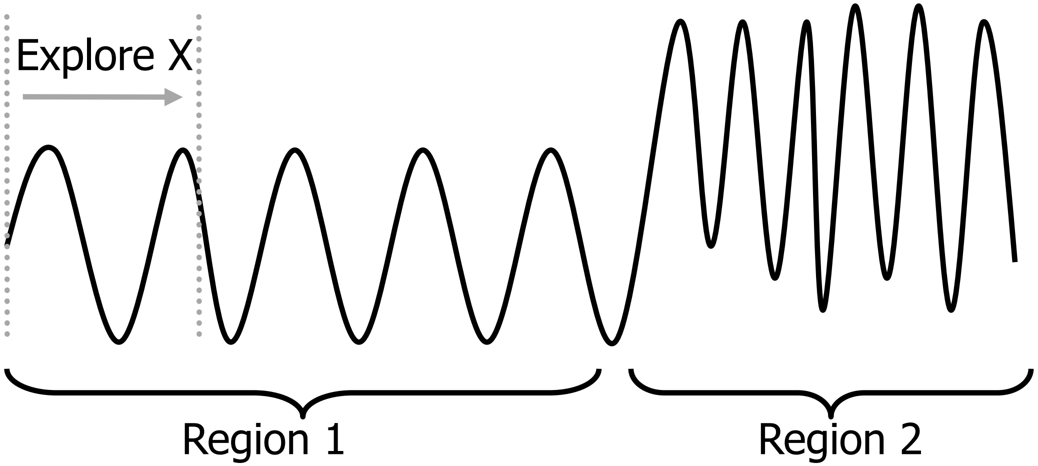
- Find good X for region before X-Y relationship changes significantly
- Then exploit good X for rest of (hopefully long) region ” achieves high SR



Runtime Controller: Attempt#2

- Goal: Train X to each region
- Reinforcement learning of X-Y relationship
  - Too slow!

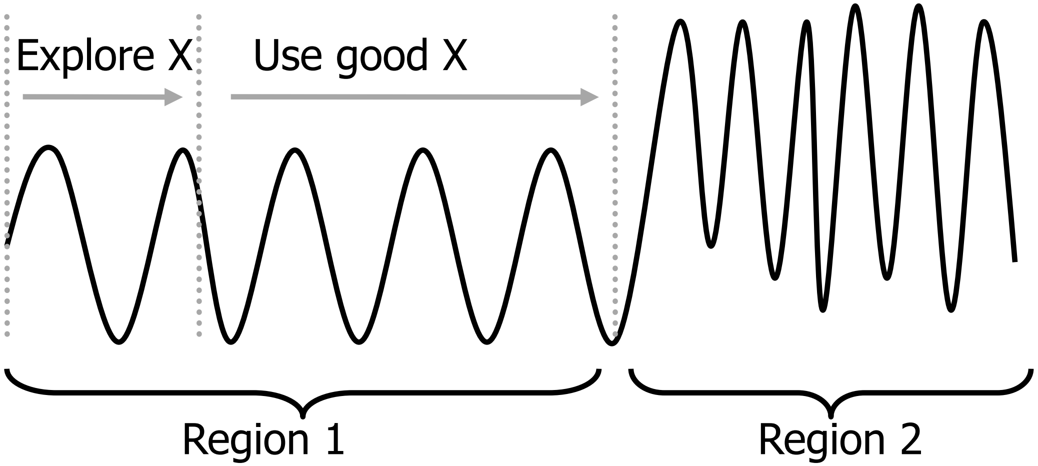
    X-Y relationships in MPEG2 Encoder and Torque Game usually change before learning can be used to subsequently control Y
- Find good X for region before X-Y relationship changes significantly
- Then exploit good X for rest of (hopefully long) region ” achieves high SR



Runtime Controller: Attempt#2

- Goal: Train X to each region
- Reinforcement learning of X-Y relationship
  - Too slow!

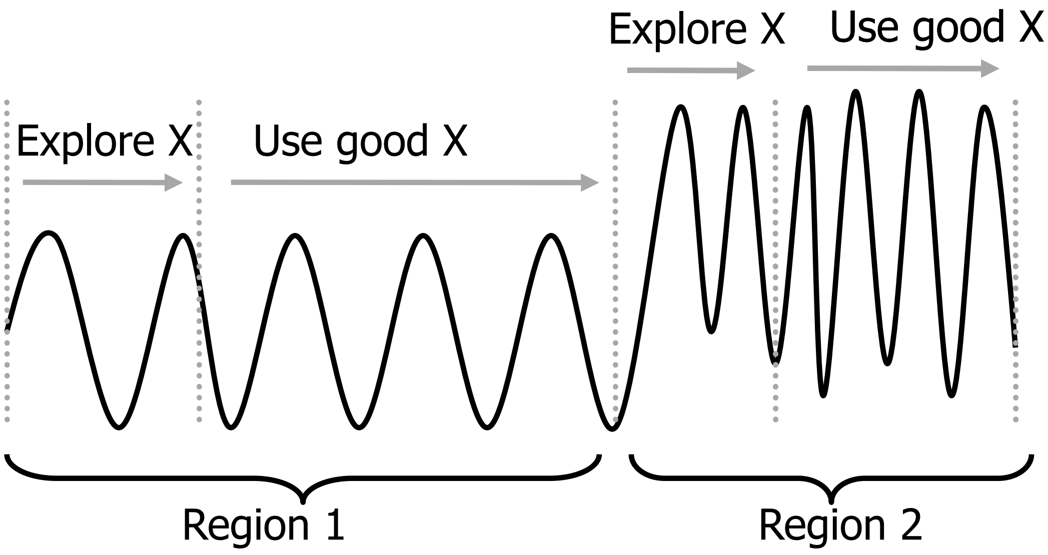
    X-Y relationships in MPEG2 Encoder and Torque Game usually change before learning can be used to subsequently control Y
- Find good X for region before X-Y relationship changes significantly
- Then exploit good X for rest of (hopefully long) region ” achieves high SR



Runtime Controller: Attempt#2

- Goal: Train X to each region
- Reinforcement learning of X-Y relationship
  - Too slow!

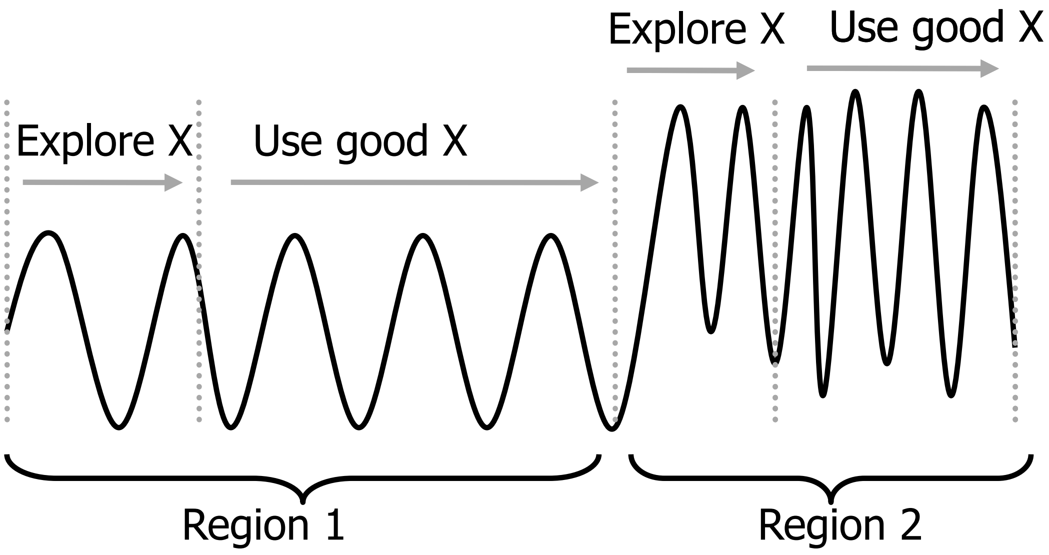
    X-Y relationships in MPEG2 Encoder and Torque Game usually change before learning can be used to subsequently control Y
- Find good X for region before X-Y relationship changes significantly
- Then exploit good X for rest of (hopefully long) region ” achieves high SR



Runtime Controller: Attempt#2

- Goal: Train X to each region
- Reinforcement learning of X-Y relationship
  - Too slow!

    X-Y relationships in MPEG2 Encoder and Torque Game usually change before learning can be used to subsequently control Y
- Find good X for region before X-Y relationship changes significantly
- Then exploit good X for rest of (hopefully long) region ” achieves high SR



<u>Observation</u>

Application “world state” does not change dramatically faster than user perception time

<u>Lesson#3</u>

Stable Region Length >> W, with high probability

Runtime Controller: Attempt#3

Runtime Controller: Attempt#3

- Use Feedback Controller: adjust X based on observed error ΔY

Runtime Controller: Attempt#3

- Use Feedback Controller: adjust X based on observed error ΔY
  - Need to assume ***monotonicity*** in X-Y relationship

Runtime Controller: Attempt#3

- Use Feedback Controller: adjust X based on observed error ΔY
  - Need to assume ***monotonicity*** in X-Y relationship
  - But X-Y relationship can remain time-varying

- Simplest: P controller

Runtime Controller: Attempt#3

- Use Feedback Controller: adjust X based on observed error ΔY
  - Need to assume ***monotonicity*** in X-Y relationship
  - But X-Y relationship can remain time-varying

- Simplest: P controller

  $\Delta X \leftarrow (1/\alpha)\, \Delta Y$

Runtime Controller: Attempt#3

- Use Feedback Controller: adjust X based on observed error ΔY
  - Need to assume ***monotonicity*** in X-Y relationship
  - But X-Y relationship can remain time-varying

- Simplest: P controller

$\Delta X \leftarrow (1/\alpha)\ \Delta Y$

(Controller Gain: $1/\alpha$)

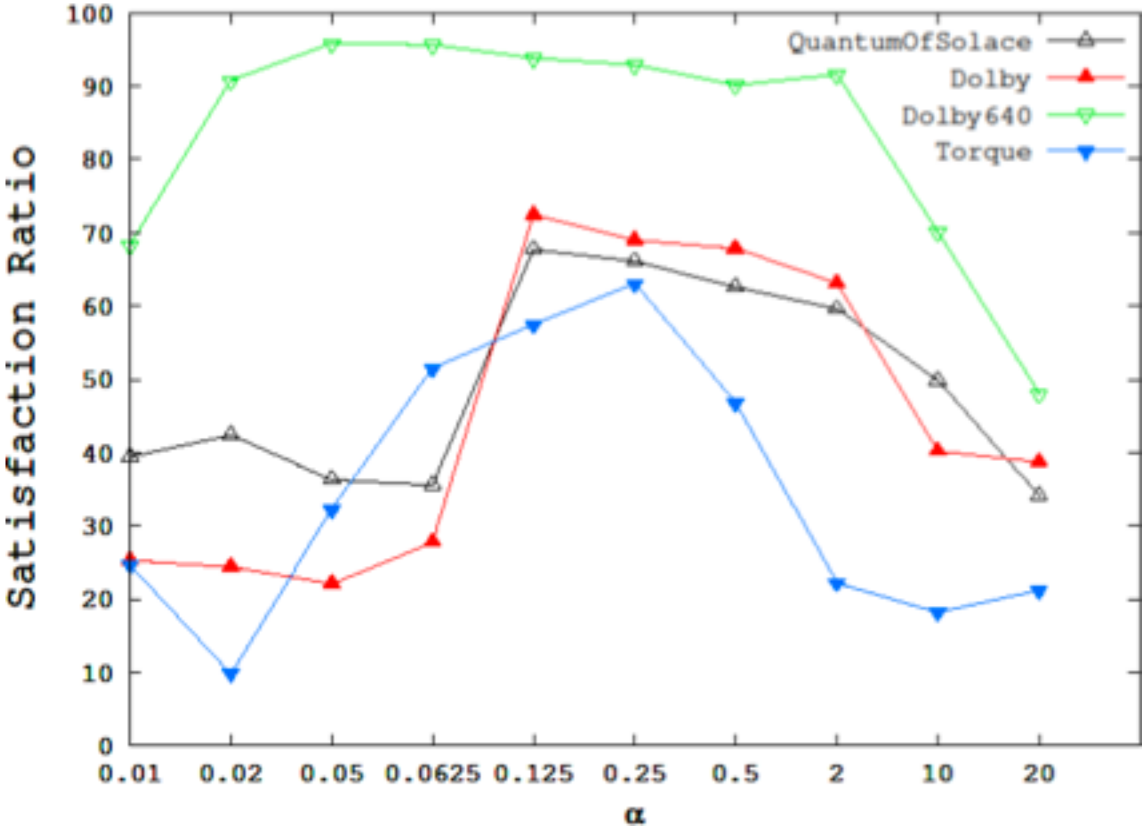
Runtime Controller: Attempt#3

- Use Feedback Controller: adjust X based on observed error ΔY
  - Need to assume ***monotonicity*** in X-Y relationship
  - But X-Y relationship can remain time-varying

- Simplest: P controller

  ΔX ← (1/α) ΔY

  (Controller Gain: 1/α)



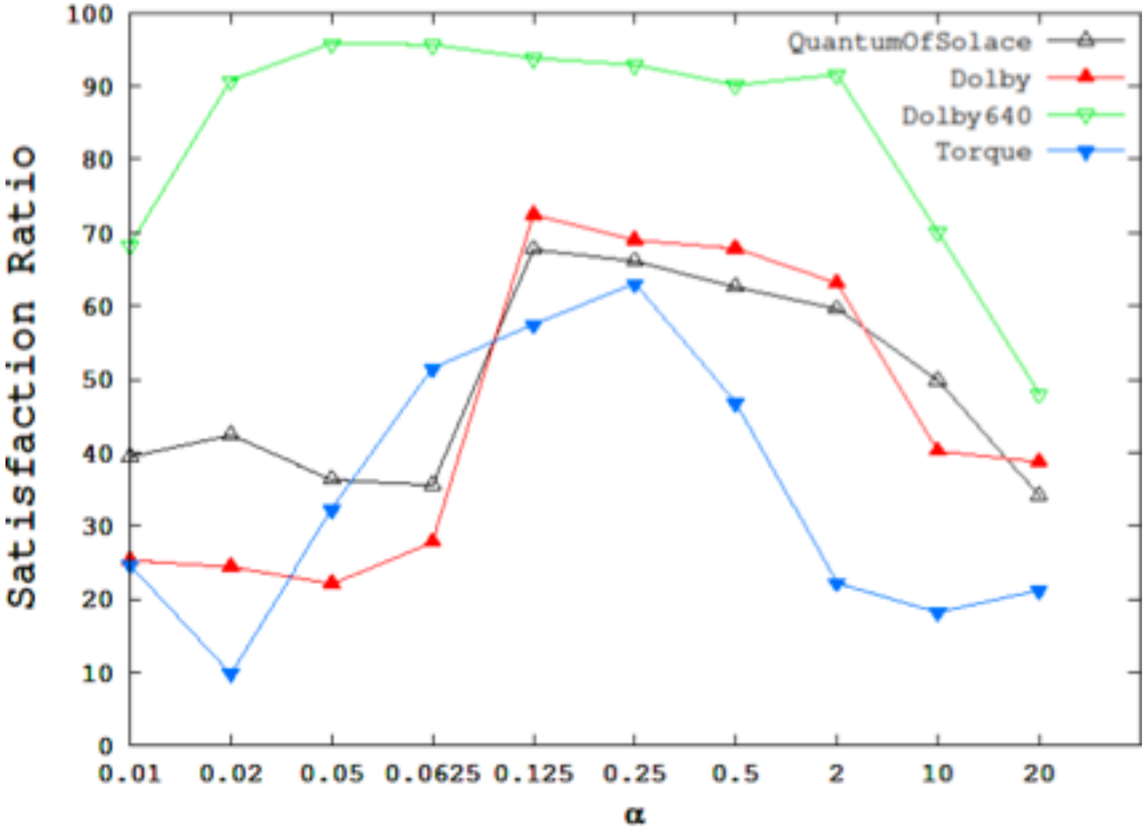
Runtime Controller: Attempt#3

- Use Feedback Controller: adjust X based on observed error ΔY
  - Need to assume ***monotonicity*** in X-Y relationship
  - But X-Y relationship can remain time-varying

- Simplest: P controller

  $\Delta X \leftarrow (1/\alpha)\,\Delta Y$

  (Controller Gain: $1/\alpha$)



<u>Lesson#3</u>

Feedback Control works poorly, unless α is in a narrow range determined by data-set

Adaptive Feedback Controller

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y

Adaptive Feedback Controller

- Programmer specifies: X, Y^{obj} ± d, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y

  - Makes rapid feedback-control feasible, dramatically simplifies learning

Adaptive Feedback Controller

- Programmer specifies: X, Y^{obj} ± d, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y

  - Makes rapid feedback-control feasible, dramatically simplifies learning

  #2. Only perceived frame-rate matters, W specified

Adaptive Feedback Controller

- Programmer specifies: X, Y^{obj} ± d, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

#1. Monotonic X-Y

- Makes rapid feedback-control feasible, dramatically simplifies learning

#2. Only perceived frame-rate matters, W specified

- Minimizes influence of frequent transients

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

#1. Monotonic X-Y

- Makes rapid feedback-control feasible, dramatically simplifies learning

#2. Only perceived frame-rate matters, W specified

- Minimizes influence of frequent transients
- When is "failure" too long? → Perceptible to user

#3. Stable Region length >> W

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y
  - Makes rapid feedback-control feasible, dramatically simplifies learning

  #2. Only perceived frame-rate matters, W specified
  - Minimizes influence of frequent transients
  - When is "failure" too long? → Perceptible to user

  #3. Stable Region length >> W
  - Determines quantitative metrics and tests to adapt policy

Adaptive Feedback Controller

- Programmer specifies: $X, Y^{obj} \pm d, W$
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y
  - Makes rapid feedback-control feasible, dramatically simplifies learning

  #2. Only perceived frame-rate matters, W specified
  - Minimizes influence of frequent transients
  - When is "failure" too long? → Perceptible to user

  #3. Stable Region length >> W
  - Determines quantitative metrics and tests to adapt policy

- Simplicity in Feedback Law:

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y

  - Makes rapid feedback-control feasible, dramatically simplifies learning

  #2. Only perceived frame-rate matters, W specified

  - Minimizes influence of frequent transients
  - When is "failure" too long? → Perceptible to user

  #3. Stable Region length >> W

  - Determines quantitative metrics and tests to adapt policy

- Simplicity in Feedback Law:

$$\Delta X \leftarrow (1/\alpha)\, \Delta Y$$

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y
  - Makes rapid feedback-control feasible, dramatically simplifies learning

  #2. Only perceived frame-rate matters, W specified
  - Minimizes influence of frequent transients
  - When is "failure" too long? → Perceptible to user

  #3. Stable Region length >> W
  - Determines quantitative metrics and tests to adapt policy

- Simplicity in Feedback Law:

$$\Delta X \leftarrow (1/\alpha)\,\Delta Y$$

- Sophistication in Adaptive Policy:

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y
  - Makes rapid feedback-control feasible, dramatically simplifies learning

  #2. Only perceived frame-rate matters, W specified
  - Minimizes influence of frequent transients
  - When is "failure" too long? → Perceptible to user

  #3. Stable Region length >> W
  - Determines quantitative metrics and tests to adapt policy

- Simplicity in Feedback Law:

  $\Delta X \leftarrow (1/\alpha)\, \Delta Y$
- Sophistication in Adaptive Policy:

  How/When to adjust α?

Adaptive Feedback Controller

- Programmer specifies: X, $Y^{obj} \pm d$, W
- Objective: maintain a high SR

- **Domain Assumptions** help us design generic controller

  #1. Monotonic X-Y
  - Makes rapid feedback-control feasible, dramatically simplifies learning

  #2. Only perceived frame-rate matters, W specified
  - Minimizes influence of frequent transients
  - When is "failure" too long? → Perceptible to user

  #3. Stable Region length >> W
  - Determines quantitative metrics and tests to adapt policy

- Simplicity in Feedback Law:

  $\Delta X \leftarrow (1/\alpha)\, \Delta Y$
- Sophistication in Adaptive Policy:

  How/When to adjust α?

} Sufficient to deliver significant improvements in SR

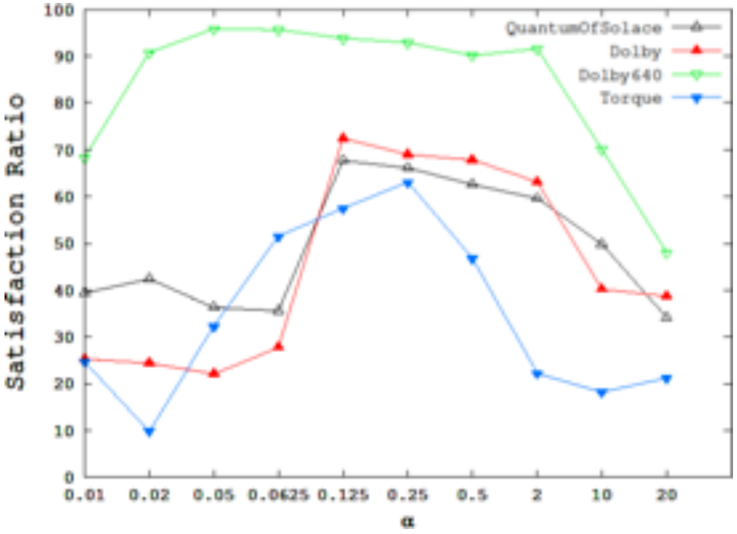
Adaptive Policy Design

Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*

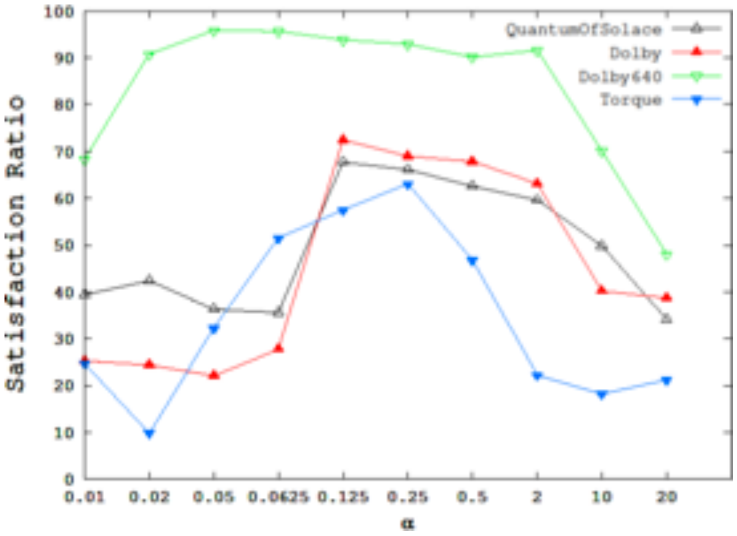
Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to $2*d$ *(error of excessive magnitude)*
  - Y outside $Y^{obj} \pm d$ for more than W frames *(error of excessive duration)*



Adaptive Policy Design

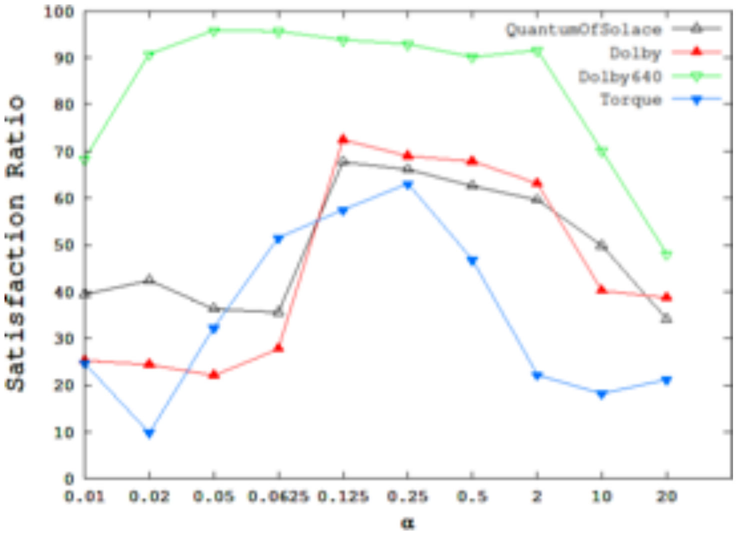
- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside $Y^{obj} \pm d$ for more than W frames *(error of excessive duration)*
- Strategy



Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside $Y^{obj} \pm d$ for more than W frames *(error of excessive duration)*

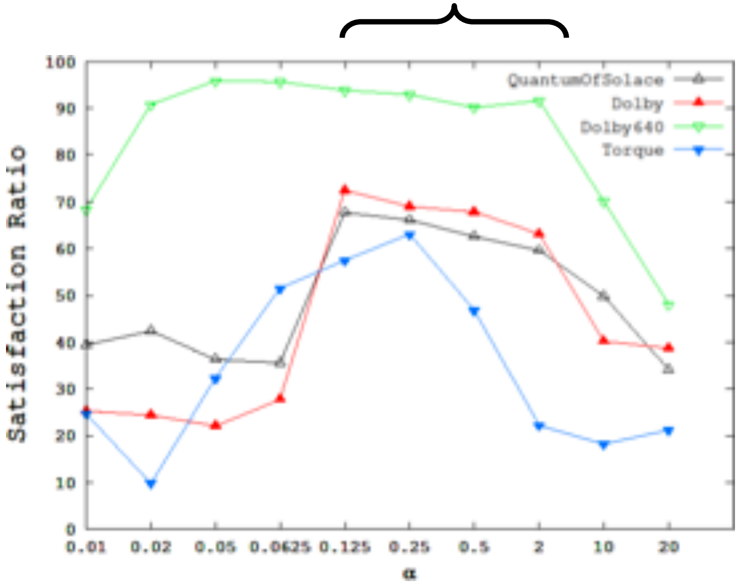
- Strategy
  - P controller's simplicity ” clearly identifiable **Failure Modes**



Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside $Y^{obj} \pm d$ for more than W frames *(error of excessive duration)*

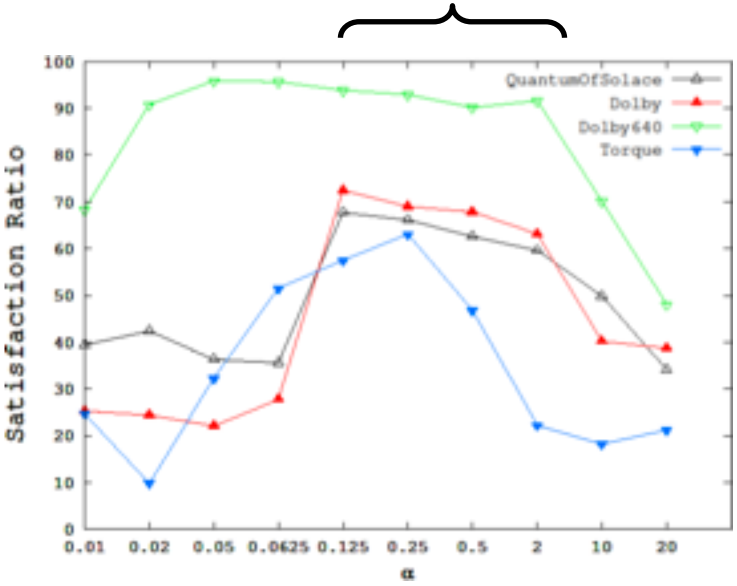
- Strategy
  - P controller's simplicity ” clearly identifiable **Failure Modes**
  - α needs to be corrected within narrow data-set dependent range (for high SR)



Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside Y^{obj} ± d for more than W frames *(error of excessive duration)*
- Strategy
  - P controller’s simplicity ” clearly identifiable **Failure Modes**
  - α needs to be corrected within narrow data-set dependent range (for high SR)

Global Failure (e.g., 320x240 vs 640x480 data-sets):
orders-of-magnitude correction in α in single step

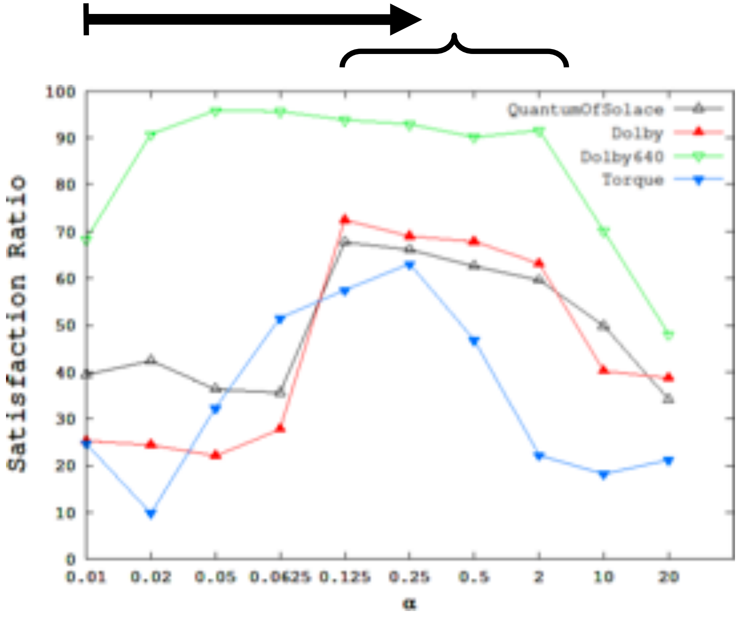


Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside $Y^{obj} \pm d$ for more than W frames *(error of excessive duration)*

- Strategy
  - P controller's simplicity ” clearly identifiable **Failure Modes**
  - α needs to be corrected within narrow data-set dependent range (for high SR)

Global Failure (e.g., 320x240 vs 640x480 data-sets):
orders-of-magnitude correction in α in single step

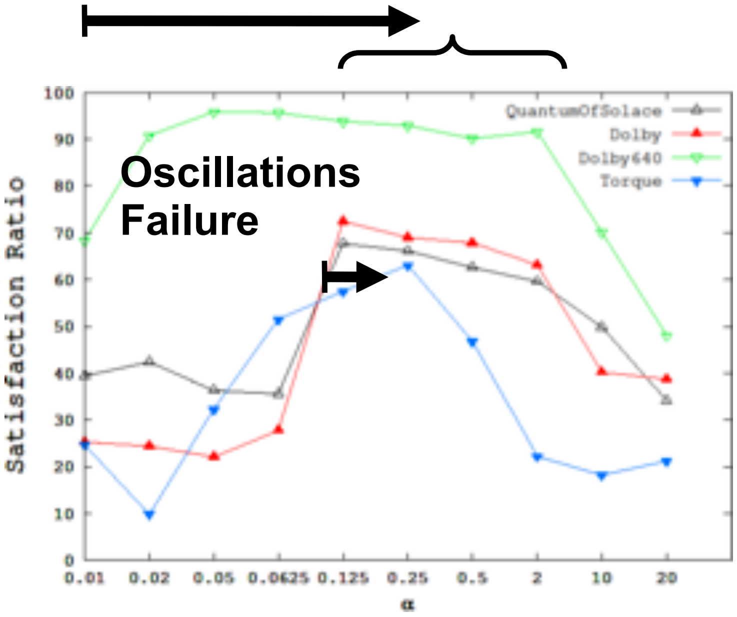


Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside $Y^{obj} \pm d$ for more than W frames *(error of excessive duration)*

- Strategy
  - P controller's simplicity ” clearly identifiable **Failure Modes**
  - α needs to be corrected within narrow data-set dependent range (for high SR)

Global Failure (e.g., 320x240 vs 640x480 data-sets):
orders-of-magnitude correction in α in single step



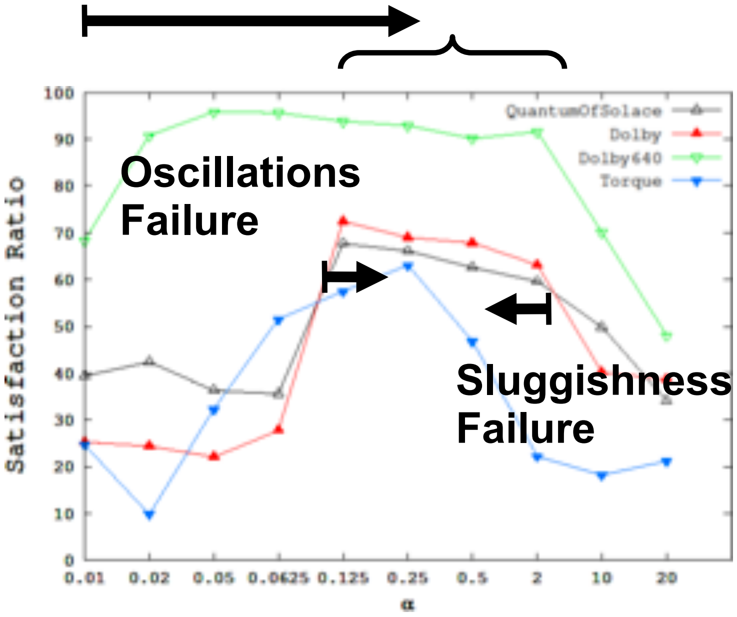
Oscillations Failure

Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside Y^{obj} ± d for more than W frames *(error of excessive duration)*

- Strategy
  - P controller's simplicity ” clearly identifiable **Failure Modes**
  - α needs to be corrected within narrow data-set dependent range (for high SR)

Global Failure (e.g., 320x240 vs 640x480 data-sets):
orders-of-magnitude correction in α *in single step*



Oscillations Failure

Sluggishness Failure

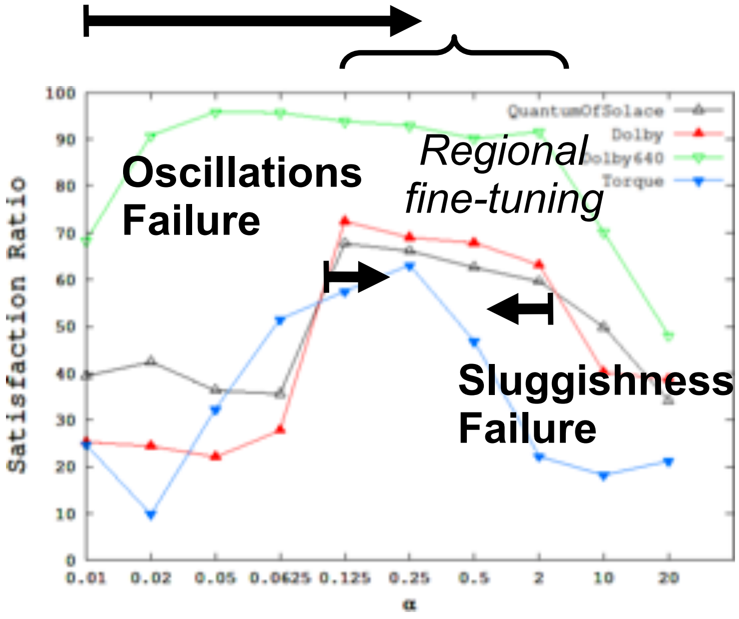
Adaptive Policy Design

- Criteria for ***Significant Policy failure***
  - ΔY compared to Y^{obj}, and compared to 2*d *(error of excessive magnitude)*
  - Y outside Y^{obj} ± d for more than W frames *(error of excessive duration)*

- Strategy
  - P controller's simplicity ” clearly identifiable **Failure Modes**
  - α needs to be corrected within narrow data-set dependent range (for high SR)

Global Failure (e.g., 320x240 vs 640x480 data-sets):

orders-of-magnitude correction in α *in single step*



Oscillations Failure

Regional fine-tuning

Sluggishness Failure

Illustration of Oscillation Failure Mode

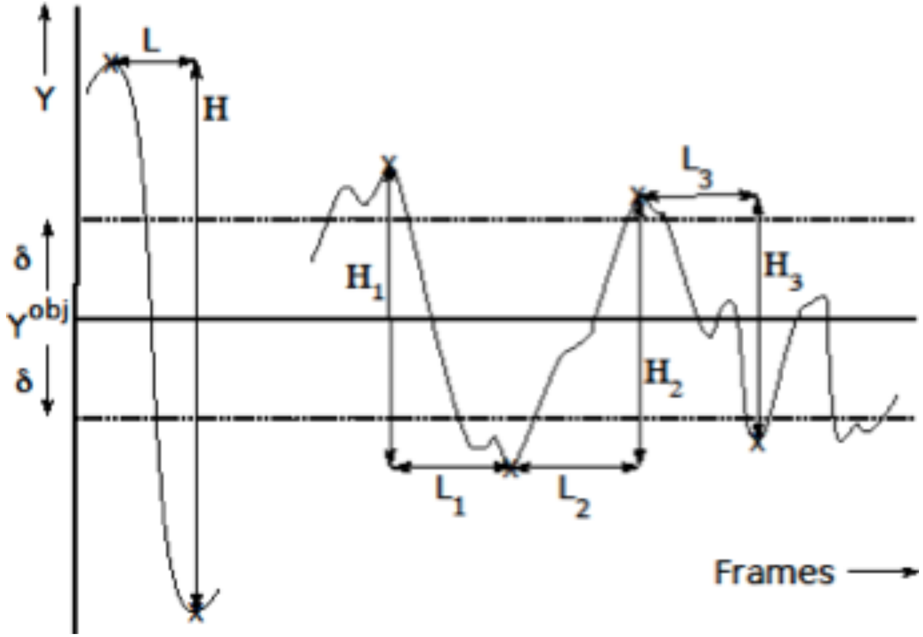


Illustration of Oscillation Failure Mode

- Failure Metrics
  - H, L
  - $\eta^{\sim}\ d * \eta + H * W / L, \ (0 < d < 1)$

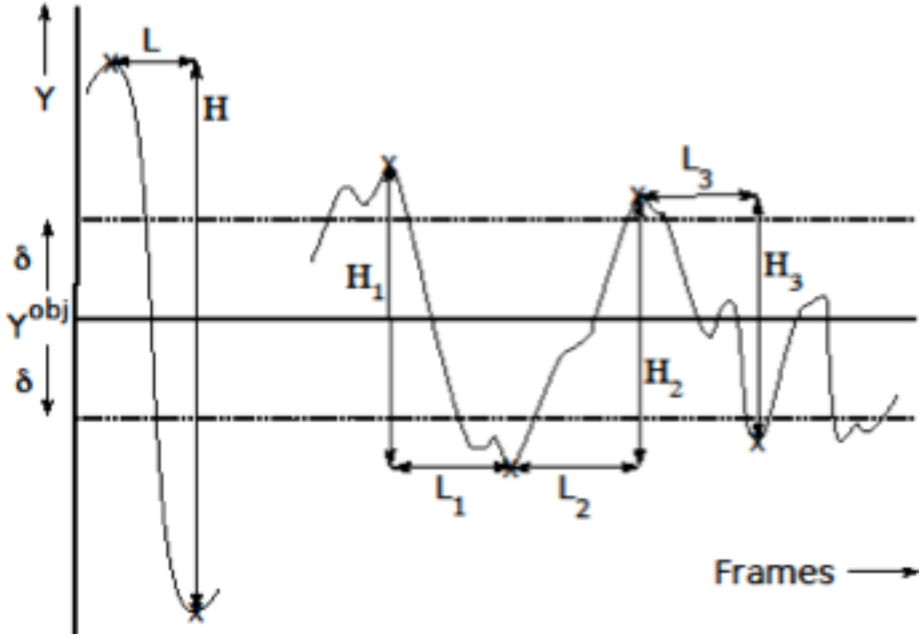


Illustration of Oscillation Failure Mode

- Failure Metrics
  - H, L
  - $\eta^{\sim}\ d * \eta + H * W / L$, $(0 < d < 1)$

- Failure Detection
  - $\eta > t$

    with $t = 1/(1-d) * 2d * 1.0$

- Policy Adaptation
  - $\alpha^{new\sim}\ \alpha * \eta / t$

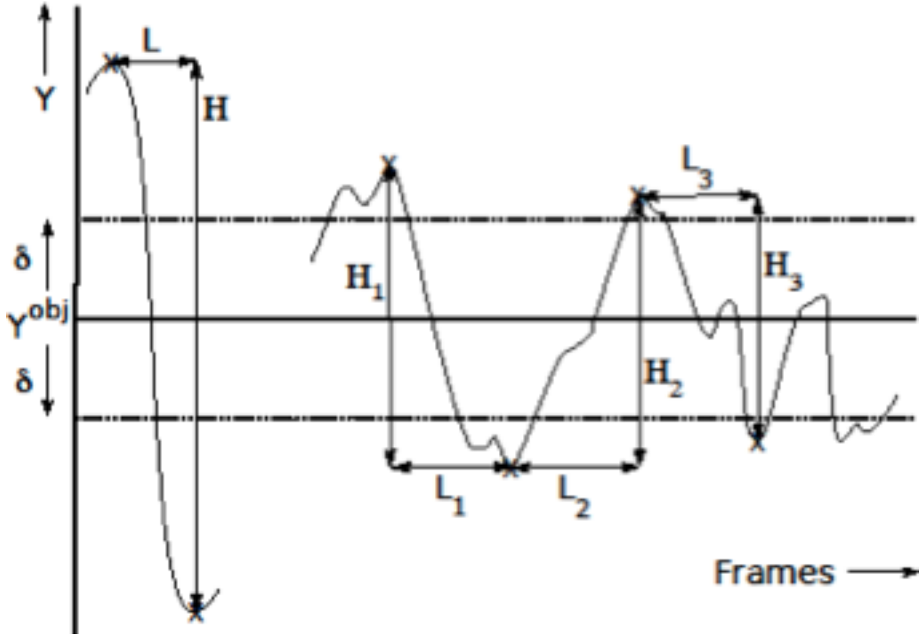
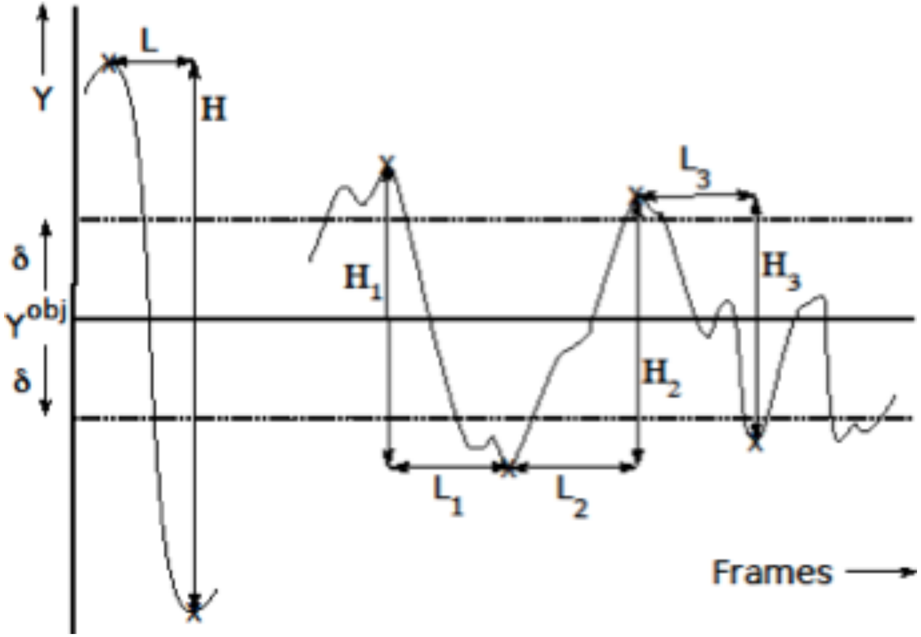


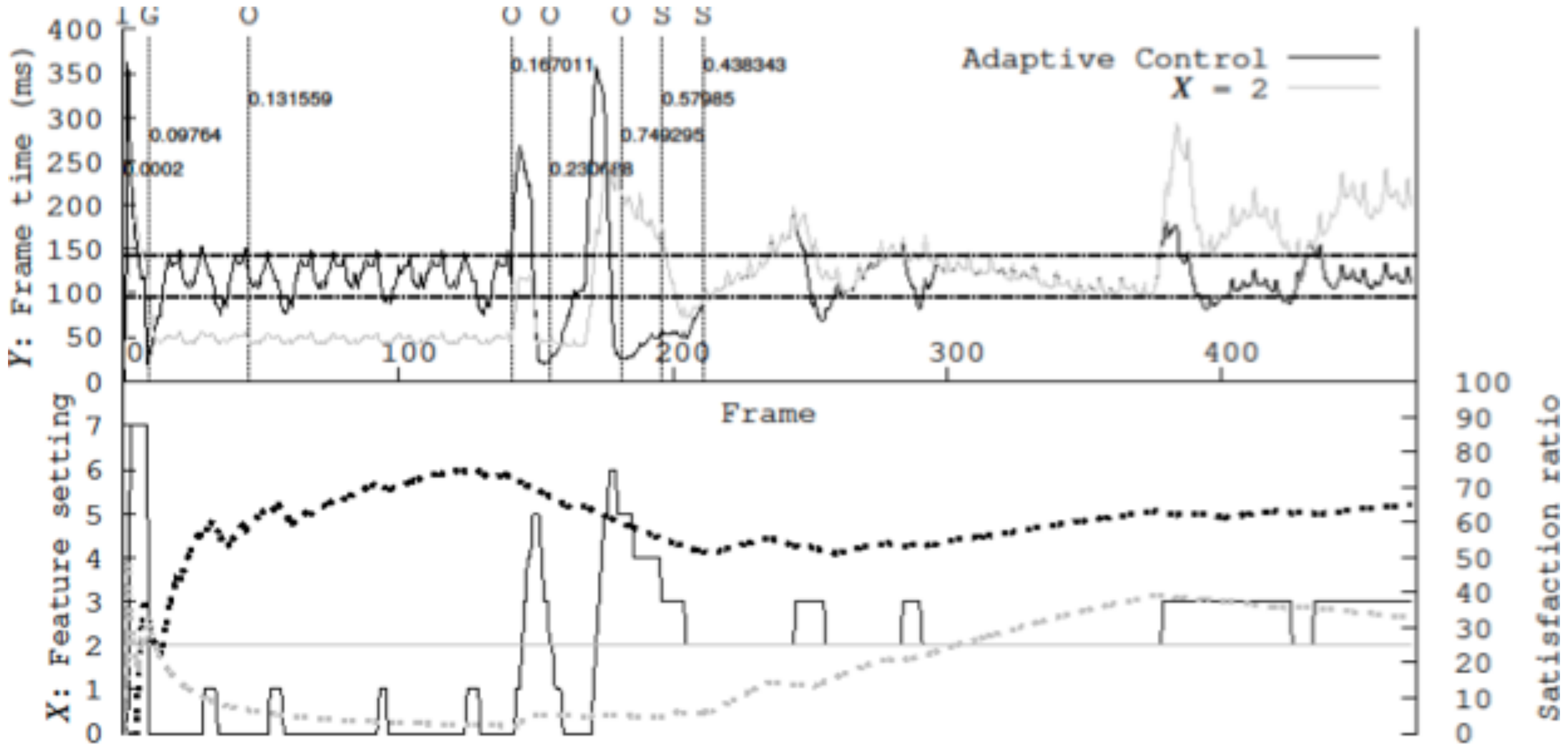
Illustration of Oscillation Failure Mode

- Failure Metrics
  - H, L
  - $\eta^{\sim} d * \eta + H * W / L, \ (0 < d < 1)$
- Failure Detection
  - $\eta > t$

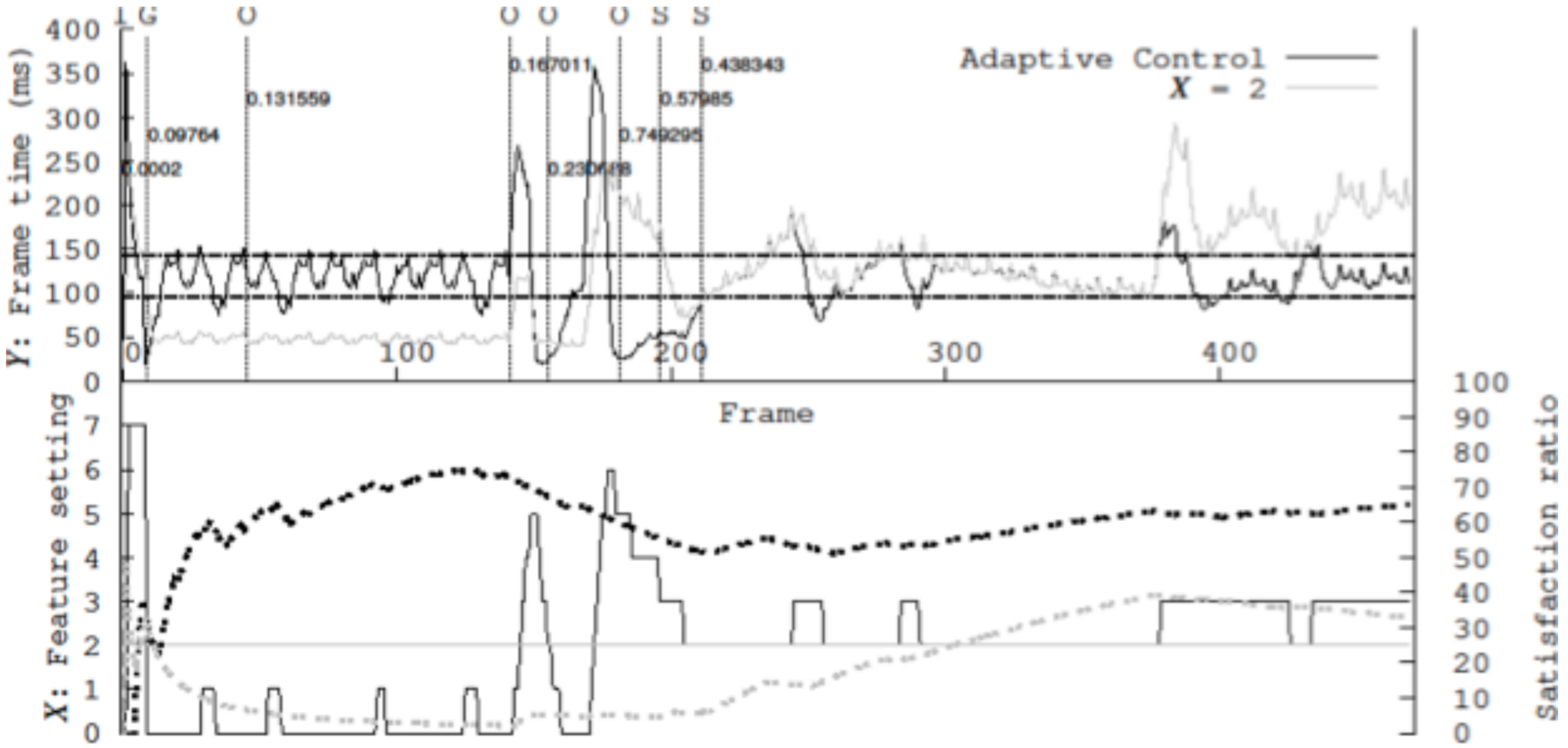
    with $t = 1/(1-d) * 2d * 1.0$
- Policy Adaptation
  - $\alpha^{new\sim} \alpha * \eta / t$
- ***Accumulative metrics*** for failure
  - Weighted to gradually forget old learning, at rate d
  - Robustness against transients
  - Responsive to persistent changed behavior
  - Constant state ” very low runtime overhead of controller



Execution Trace: MPEG2 Encoder



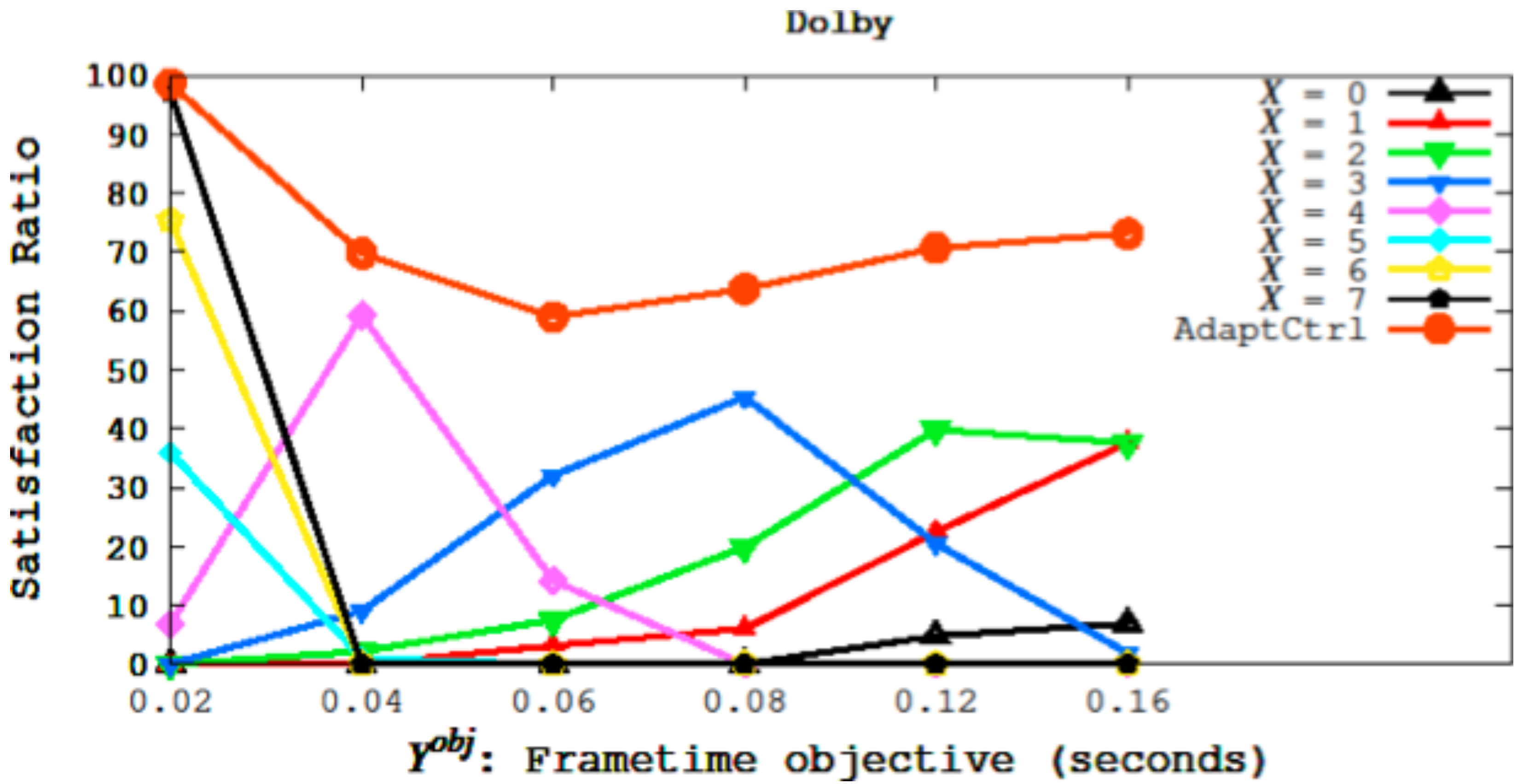
Execution Trace: MPEG2 Encoder



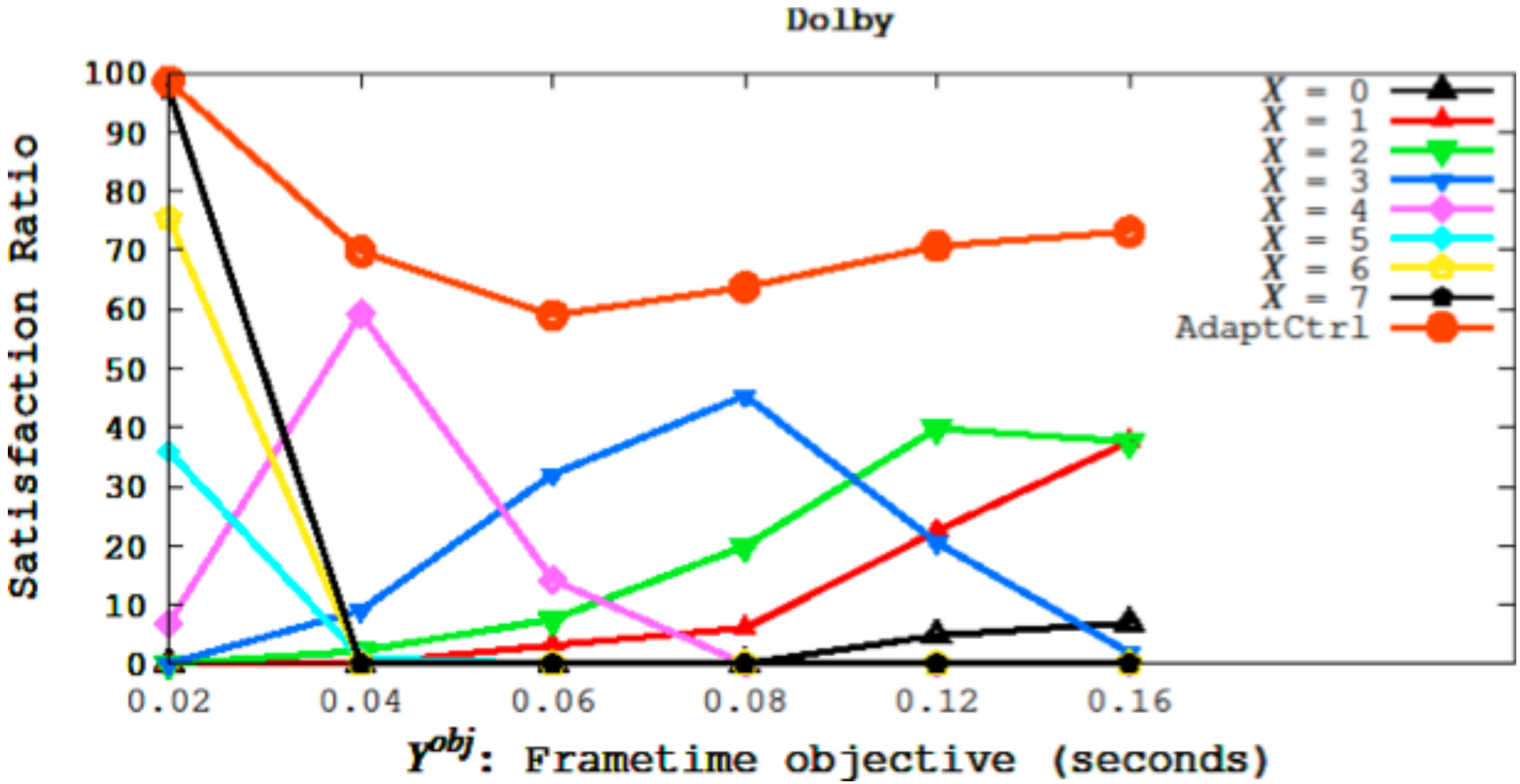
X → *Search Window Size* : $\{0 \to 30, 1 \to 20, 2 \to 15, 3 \to 10, 4 \to 5, 5 \to 2, 6 \to 1, 7 \to 0\}$

Integral X: Easier for programmer to just **sample** ***Search Window Size*** over sufficient range

Benchmark Result: MPEG2 Encoder

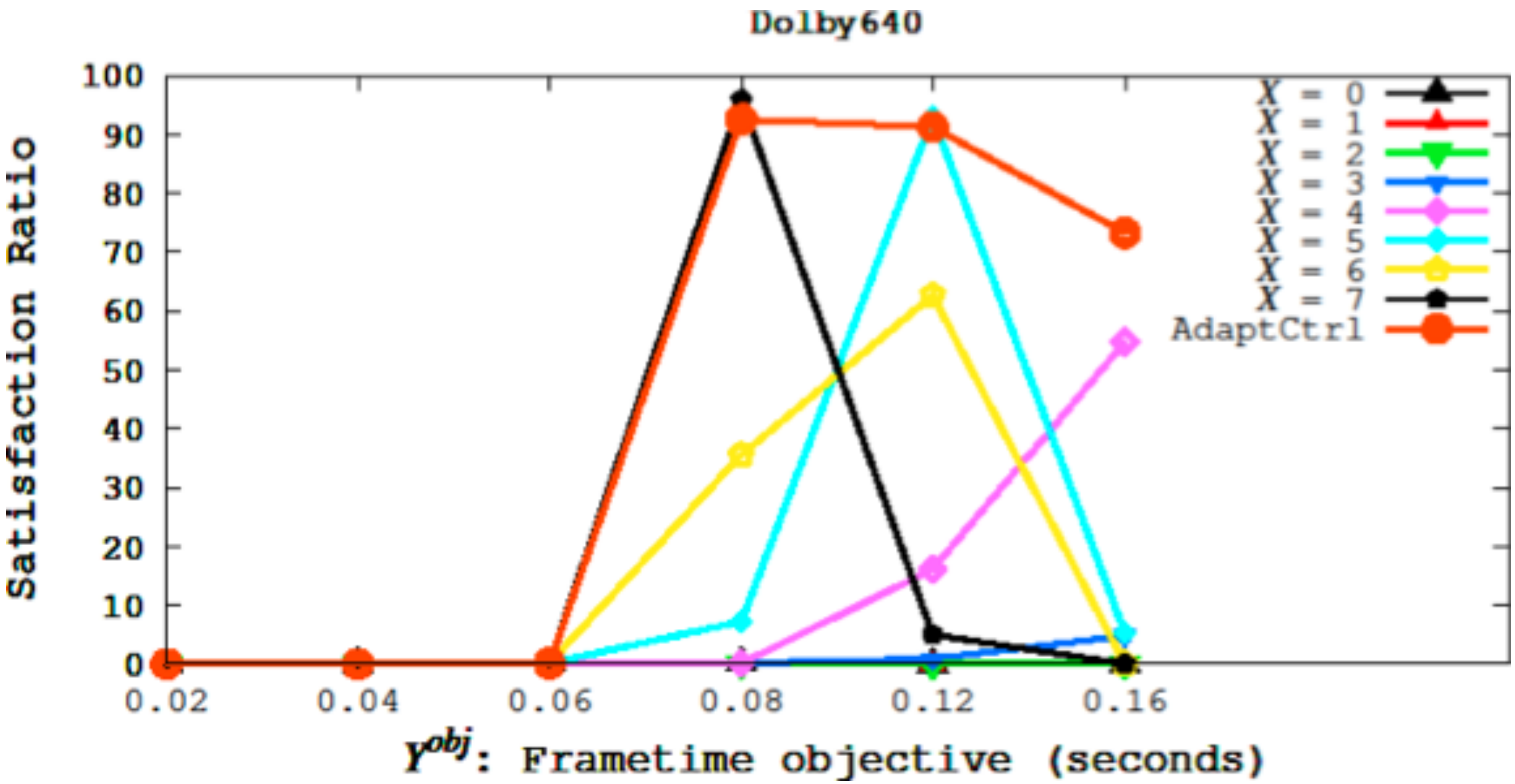


Benchmark Result: MPEG2 Encoder

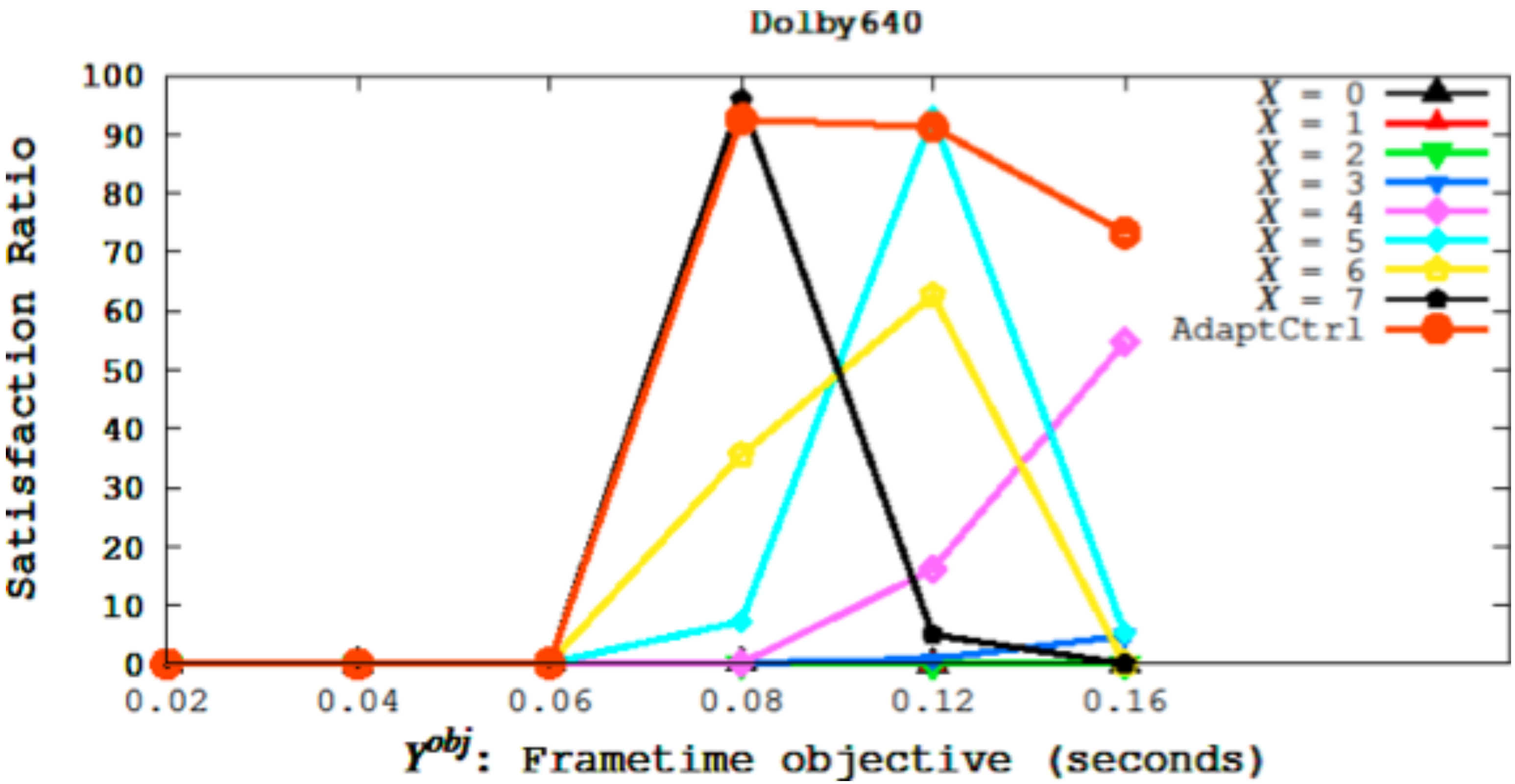


- Video sequence: dolbycity320x240
- Adaptive Controller better than ***envelope of best fixed*** Xs
  - Due to Regional tuning of X vs only global tuning

Benchmark Result: MPEG2 Encoder

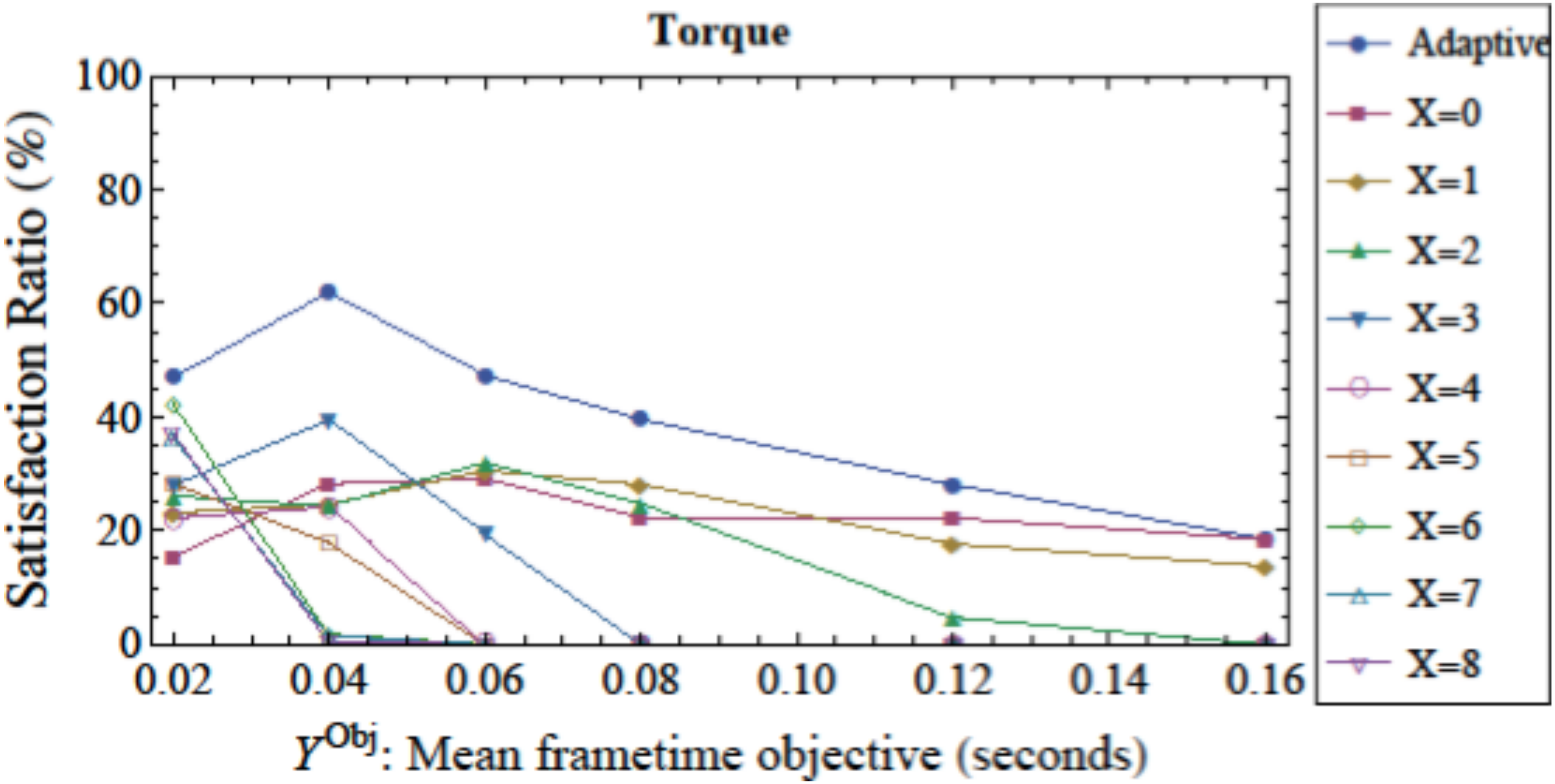


Benchmark Result: MPEG2 Encoder

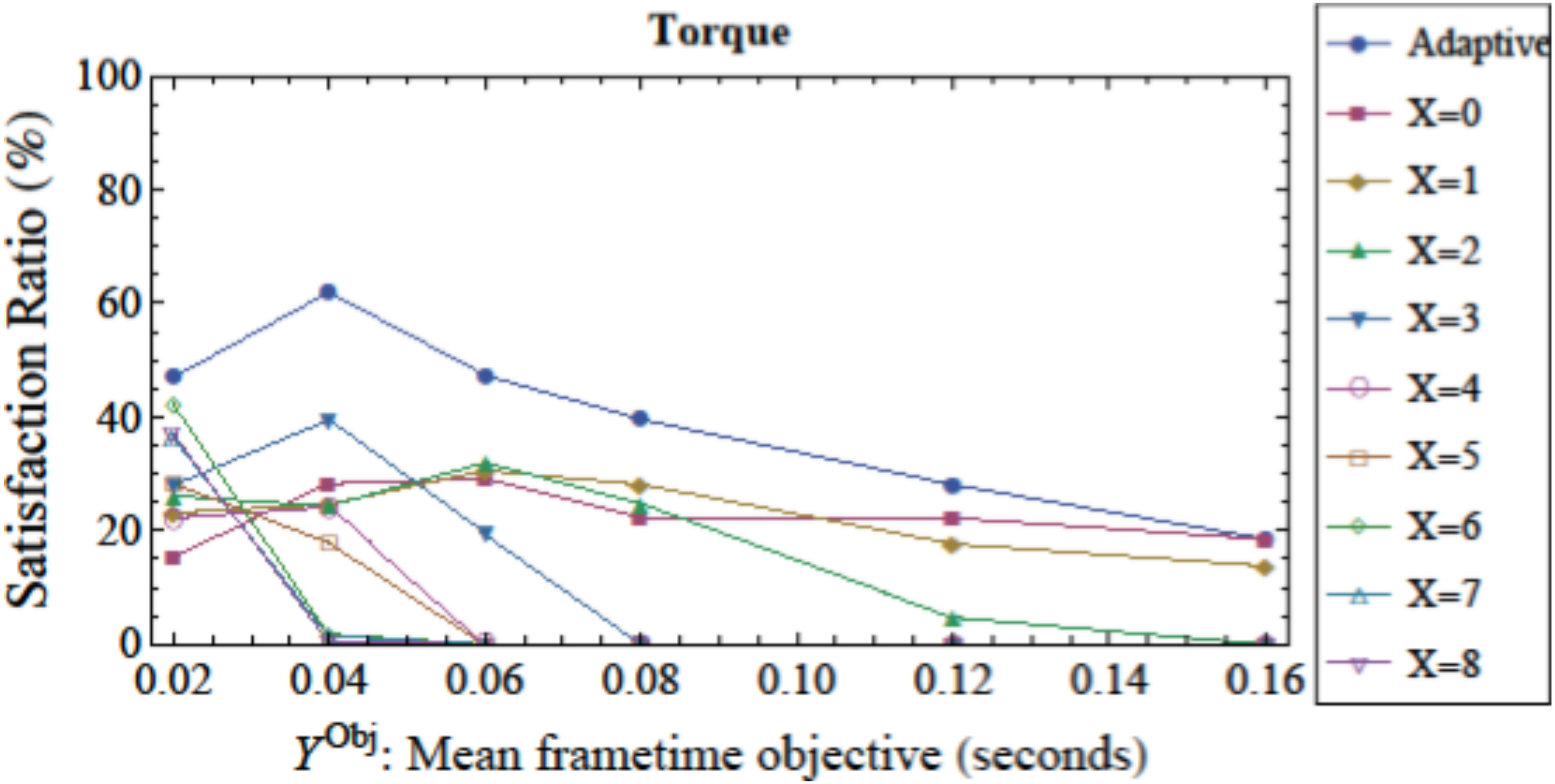


- Video sequence: dolbycity 640x480
- Adaptive better than every Fixed X case overall
  - Even though for a given Y^{obj}, a particular Fixed X might match or exceed Adaptive

Benchmark Result: Torque Game Engine

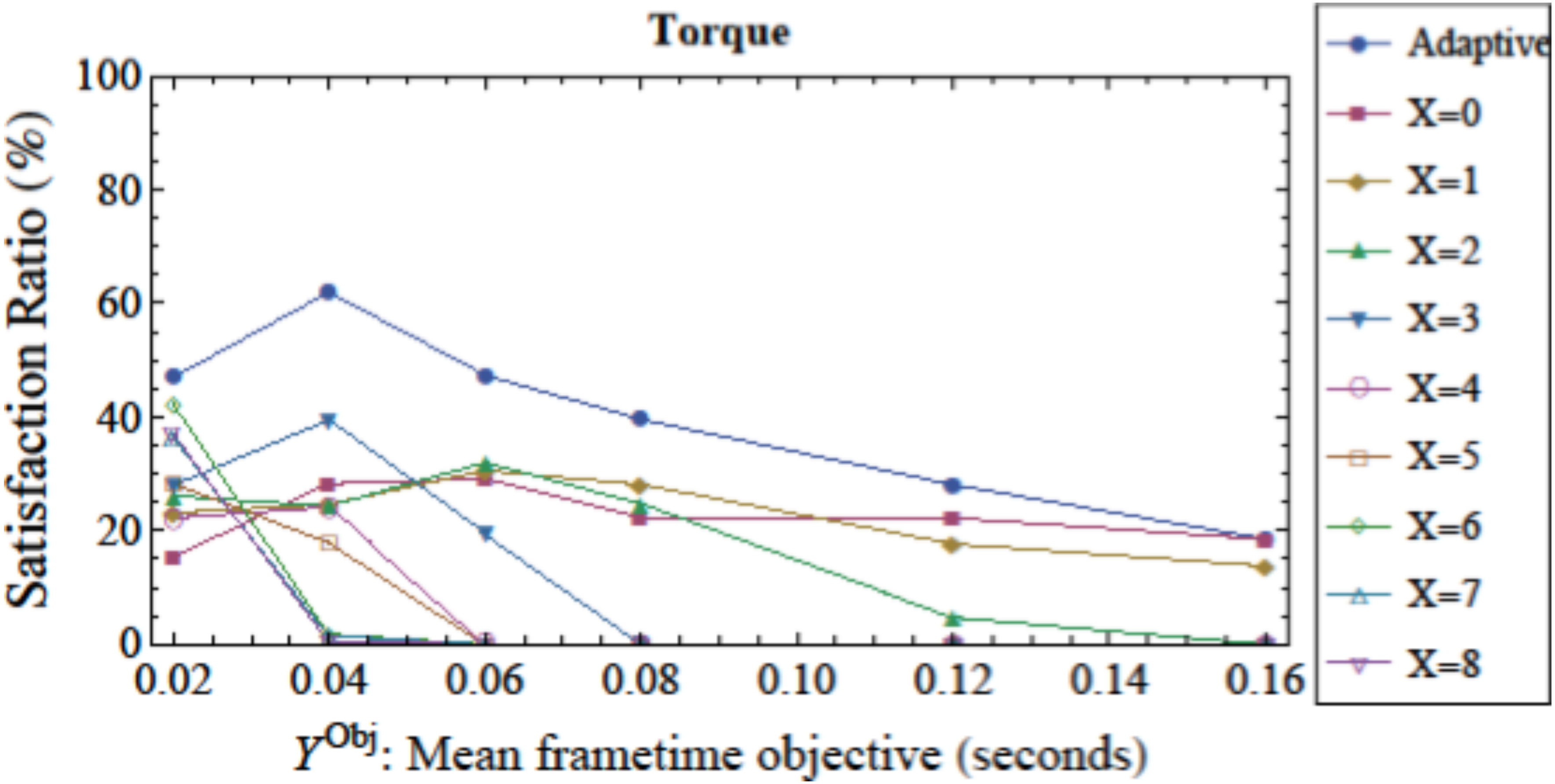


Benchmark Result: Torque Game Engine



- Quality of Result: ***Gameplay Intelligence***

Benchmark Result: Torque Game Engine



- Quality of Result: ***Gameplay Intelligence***
- For $Y^{obj} = 0.04$ secs
  - Adaptive: 24ms of AI/frame
  - Best fixed X: 14ms of AI/frame

Related Work

Related Work

- Real-time techniques already address QoS and Soft Real-time. BUT:
  - Application needs to be implemented as Task-Graphs
  - With execution-time properties specified for nodes [Mejia-Alvarez et al. RTSS 2000]
  - And, Utility functions for QoS provided [Block et al. ECRTS 2008]

- Application-specific techniques
  - Rate-distortion control in H.264 encoder
  - QoS control in MPEG2 ***decoder*** (easier than ***encoder*** due to video-control-sequence information) [Roitzsch et al. RTSS'06] [Huang et al. MULTIMEDIA'07] [Wust et al. ECRTS'04]

Related Work

- Real-time techniques already address QoS and Soft Real-time. BUT:
  - Application needs to be implemented as Task-Graphs
  - With execution-time properties specified for nodes [Mejia-Alvarez et al. RTSS 2000]
  - And, Utility functions for QoS provided [Block et al. ECRTS 2008]

- Application-specific techniques
  - Rate-distortion control in H.264 encoder
  - QoS control in MPEG2 ***decoder*** (easier than ***encoder*** due to video-control-sequence information) [Roitzsch et al. RTSS'06] [Huang et al. MULTIMEDIA'07] [Wust et al. ECRTS'04]

- "Ready-made" Adaptive techniques
  - Illustrated by: Web-cache QoS optimization [Lu et al. IWQoS 2002]

    Adaptive Pole-placement controller-design, based on periodically re-estimating a 2nd order parametric LTI model

Related Work

- Real-time techniques already address QoS and Soft Real-time. BUT:
  - Application needs to be implemented as Task-Graphs
  - With execution-time properties specified for nodes [Mejia-Alvarez et al. RTSS 2000]
  - And, Utility functions for QoS provided [Block et al. ECRTS 2008]

- Application-specific techniques
  - Rate-distortion control in H.264 encoder
  - QoS control in MPEG2 ***decoder*** (easier than ***encoder*** due to video-control-sequence information) [Roitzsch et al. RTSS'06] [Huang et al. MULTIMEDIA'07] [Wust et al. ECRTS'04]

- "Ready-made" Adaptive techniques
  - Illustrated by: Web-cache QoS optimization [Lu et al. IWQoS 2002]

    Adaptive Pole-placement controller-design, based on periodically re-estimating a 2nd order parametric LTI model

- Our work is an ***Adaptive Gain-Scheduling*** controller
  - Adjust control-law directly to overcome observed failures

Related Work

- Real-time techniques already address QoS and Soft Real-time. BUT:
  - Application needs to be implemented as Task-Graphs
  - With execution-time properties specified for nodes [Mejia-Alvarez et al. RTSS 2000]
  - And, Utility functions for QoS provided [Block et al. ECRTS 2008]

- Application-specific techniques
  - Rate-distortion control in H.264 encoder
  - QoS control in MPEG2 ***decoder*** (easier than ***encoder*** due to video-control-sequence information) [Roitzsch et al. RTSS'06] [Huang et al. MULTIMEDIA'07] [Wust et al. ECRTS'04]

- "Ready-made" Adaptive techniques
  - Illustrated by: Web-cache QoS optimization [Lu et al. IWQoS 2002]

    Adaptive Pole-placement controller-design, based on periodically re-estimating a 2nd order parametric LTI model

- Our work is an ***Adaptive Gain-Scheduling*** controller
  - Adjust control-law directly to overcome observed failures
- Controller is ***domain-specific*** rather than ***application-specific***
  - Based on broad domain-assumptions, rather than linear dynamical models of plant
  - ***Semi-physical System-Identification***

Related Work

- Real-time techniques already address QoS and Soft Real-time. BUT:
  - Application needs to be implemented as Task-Graphs
  - With execution-time properties specified for nodes [Mejia-Alvarez et al. RTSS 2000]
  - And, Utility functions for QoS provided [Block et al. ECRTS 2008]

- Application-specific techniques
  - Rate-distortion control in H.264 encoder
  - QoS control in MPEG2 ***decoder*** (easier than ***encoder*** due to video-control-sequence information) [Roitzsch et al. RTSS'06] [Huang et al. MULTIMEDIA'07] [Wust et al. ECRTS'04]

- "Ready-made" Adaptive techniques
  - Illustrated by: Web-cache QoS optimization [Lu et al. IWQoS 2002]

    Adaptive Pole-placement controller-design, based on periodically re-estimating a 2nd order parametric LTI model

- Our work is an ***Adaptive Gain-Scheduling*** controller
  - Adjust control-law directly to overcome observed failures
- Controller is ***domain-specific*** rather than ***application-specific***
  - Based on broad domain-assumptions, rather than linear dynamical models of plant
  - ***Semi-physical System-Identification***
- Very effective, but Best-Effort:
  - No guarantees on Safety, Robustness, Reachability

Conclusion

Conclusion

- Our controller greatly simplifies frame-QoS control
  - Programmers avoid having to model emergent behaviors
  - Incorporate controller as a library
    - exceedingly light-weight: < 0.05% overhead

Conclusion

- Our controller greatly simplifies frame-QoS control
    - Programmers avoid having to model emergent behaviors
    - Incorporate controller as a library
        - exceedingly light-weight: < 0.05% overhead

- Generality
    - Any application that satisfies Domain Assumptions
        - Monotonic response, perceived frame-rate, Stable response periods >> W frames
    - Graceful degradation when not

- No general-purpose alternatives available that work well for interactive applications whose behavior is *UNKNOWN*, *EMERGENT* and *NON-ANALYZABLE*

Conclusion

- Our controller greatly simplifies frame-QoS control
  - Programmers avoid having to model emergent behaviors
  - Incorporate controller as a library
    - exceedingly light-weight: < 0.05% overhead

- Generality
  - Any application that satisfies Domain Assumptions
    - Monotonic response, perceived frame-rate, Stable response periods >> W frames
  - Graceful degradation when not

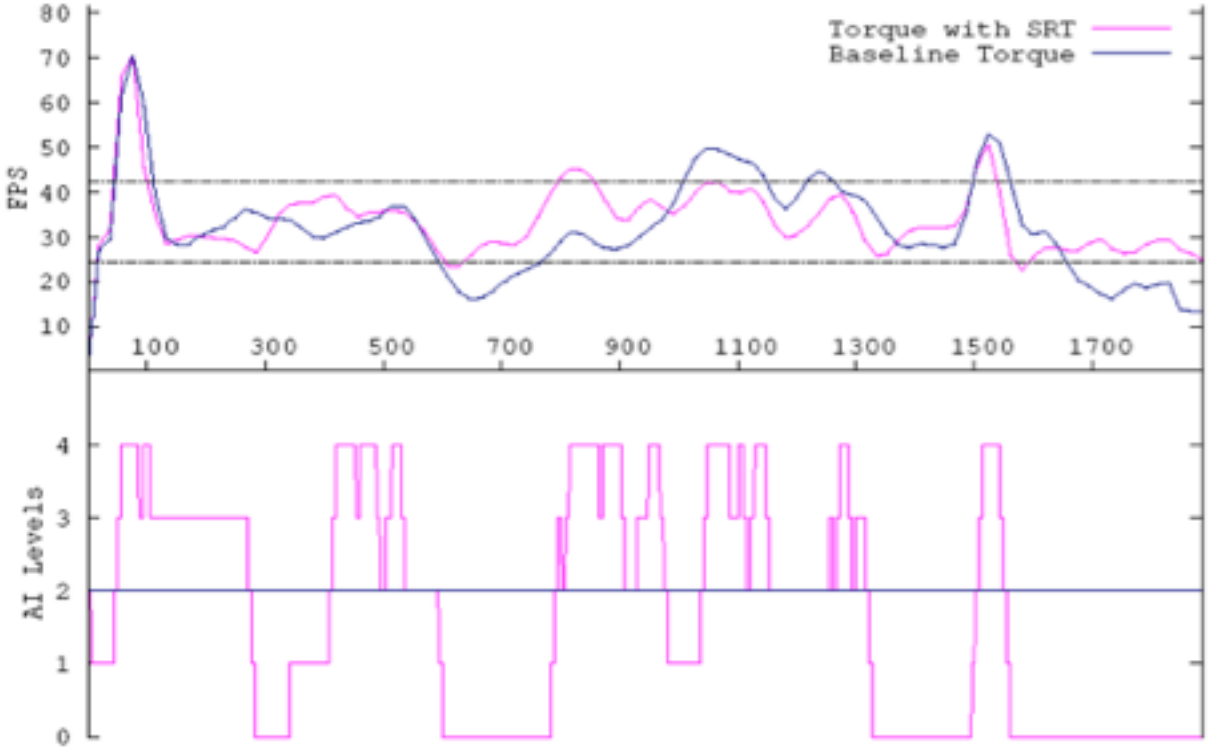
- No general-purpose alternatives available that work well for interactive applications whose behavior is *UNKNOWN*, *EMERGENT* and *NON-ANALYZABLE*

- Future Work
  - Multiple X, Multiple prioritized Y, Explicit Q
  - Domain Observations ”Adaptive Control + Least-Squares Function Estimation
  - But, much more compute intensive controller

Thank you!

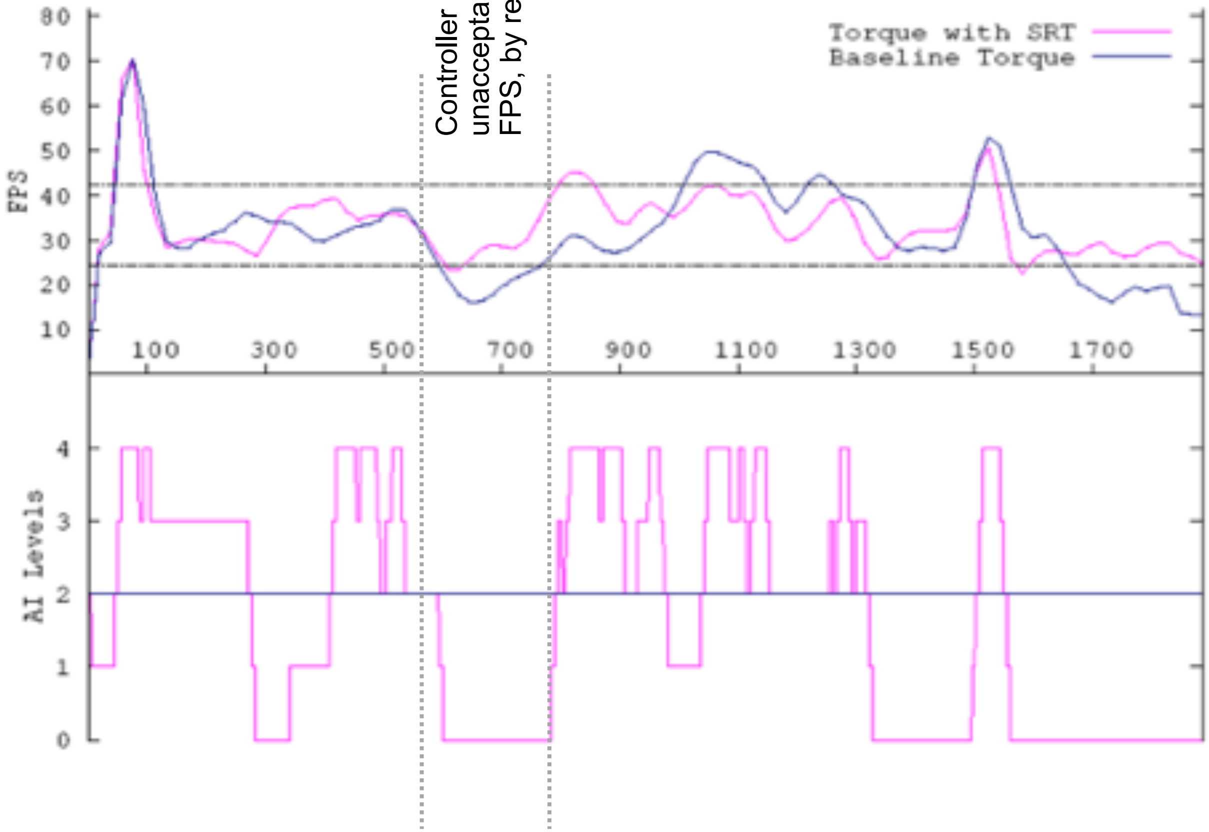
- Questions?

Torque Game Engine: Measured Behavior



objective:
25 to 42 fps

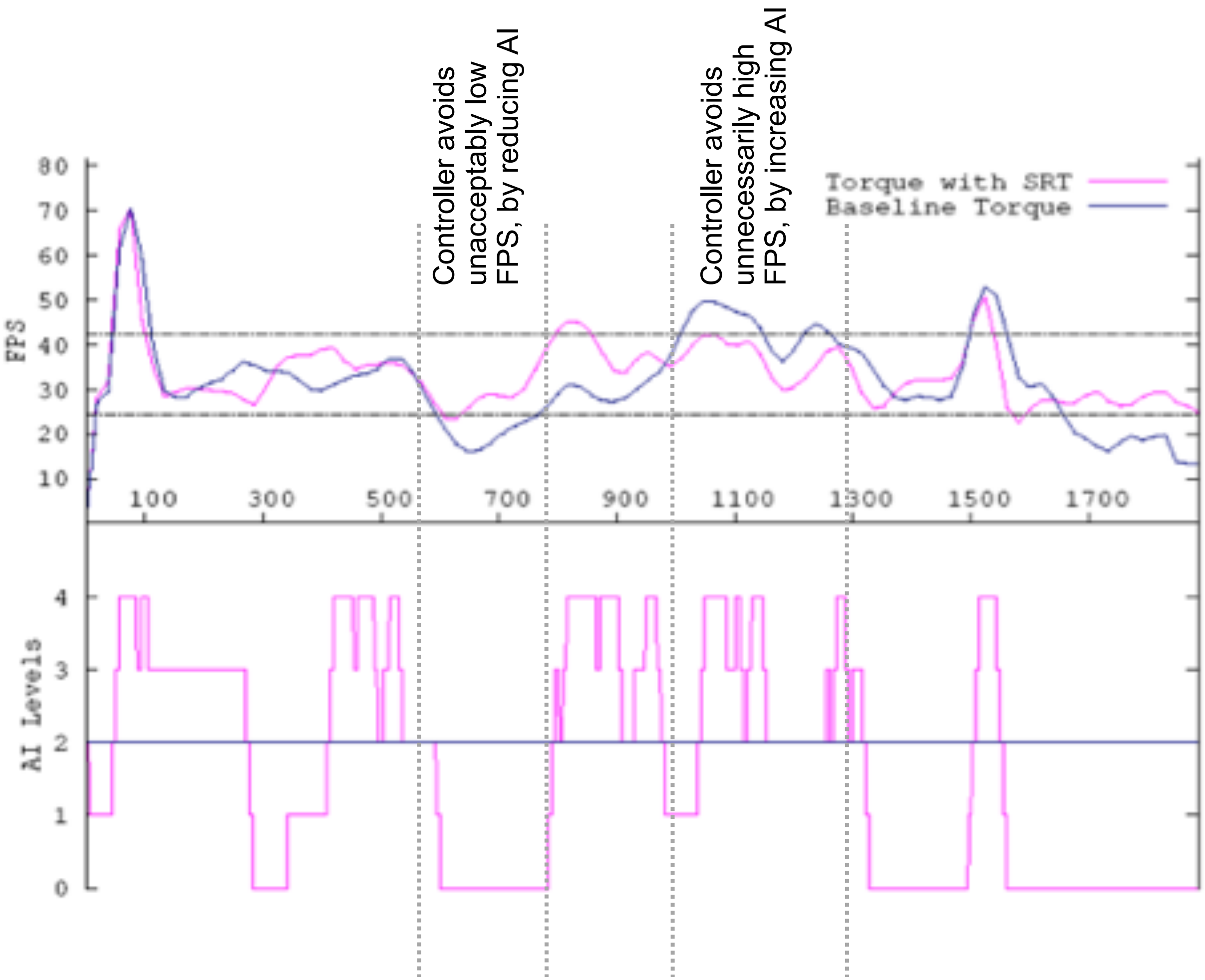
Torque Game Engine: Measured Behavior



Controller avoids unacceptably low FPS, by reducing AI

objective:
25 to 42 fps

Torque Game Engine: Measured Behavior

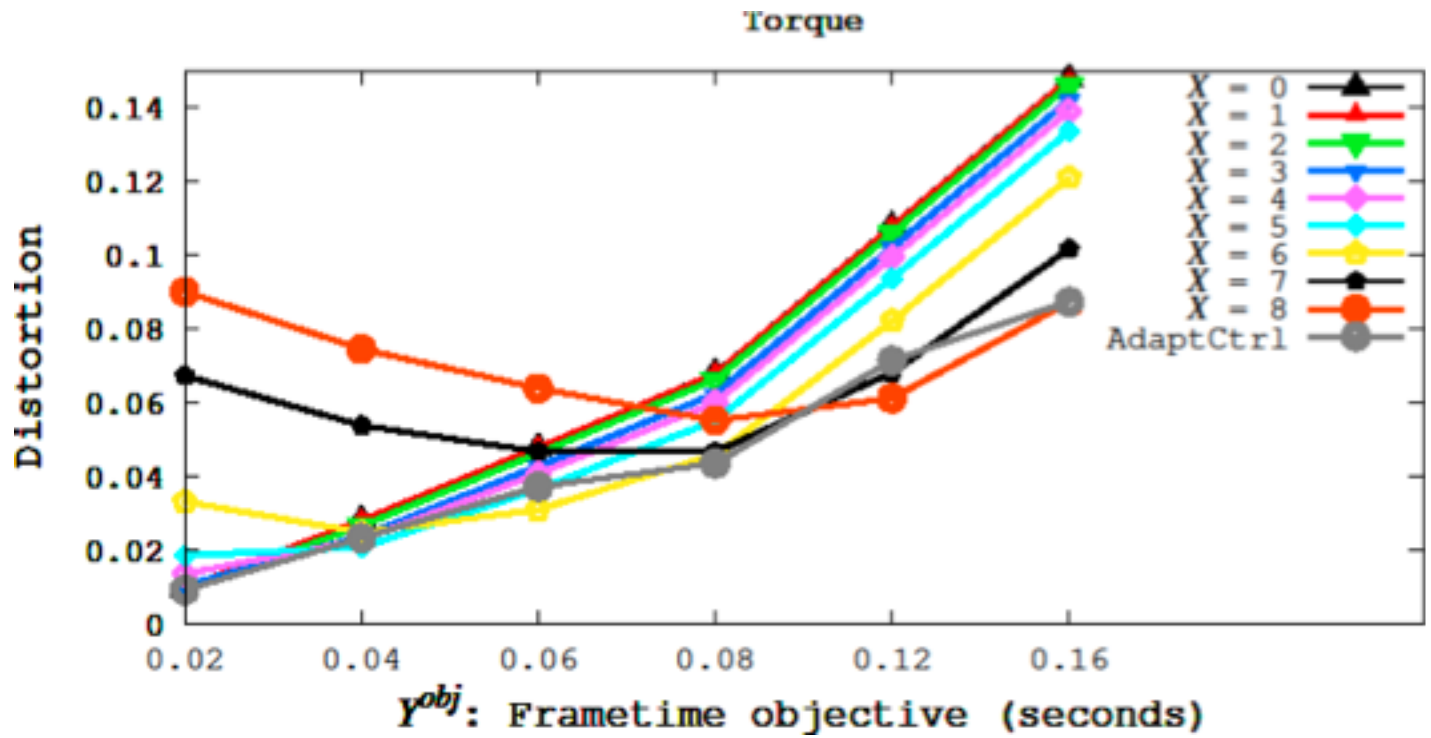


Controller avoids unacceptably low FPS, by reducing AI

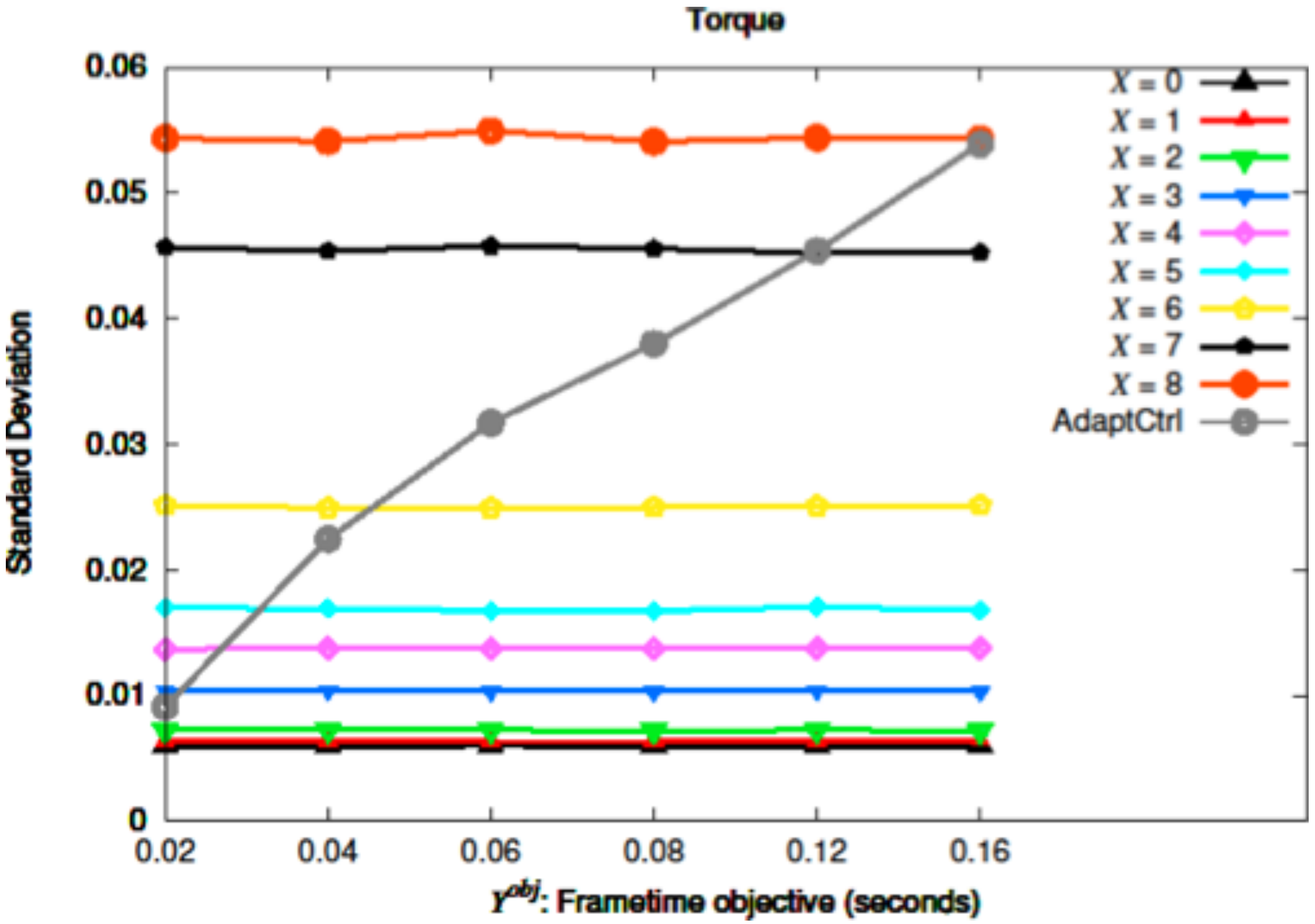
Controller avoids unnecessarily high FPS, by increasing AI

objective: 25 to 42 fps

Distortion in Torque



Standard Deviation in Torque



Don’t Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz

Don’t Real-Time Methods Solve This Already?

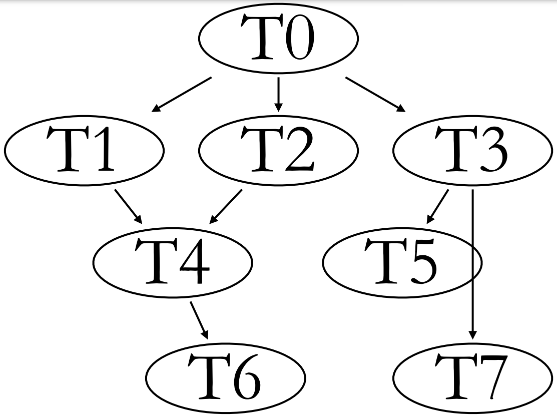
**Games, Multimedia,
Interactive Viz**

**Implement as
a Real-Time App**

Don’t Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz

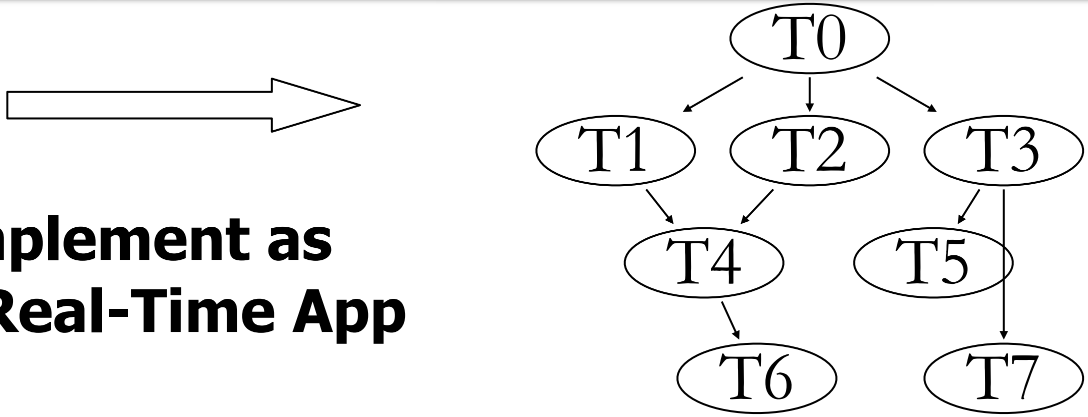
Implement as a Real-Time App



Don’t Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz

Implement as a Real-Time App



<u>Real-Time Task-Graph</u>

- Application decomposed into Tasks and Precedence Constraints

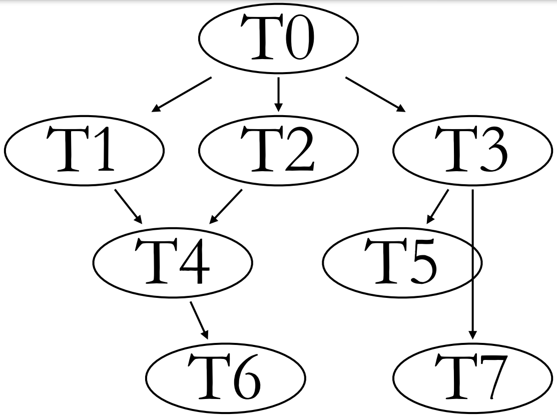
- Responsiveness guaranteed by Real-time semantics (hard or probabilistic)

Don’t Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz

Implement as a Real-Time App

Implement with High-Productivity, Large Scale Programming flows



<u>Real-Time Task-Graph</u>

- Application decomposed into Tasks and Precedence Constraints

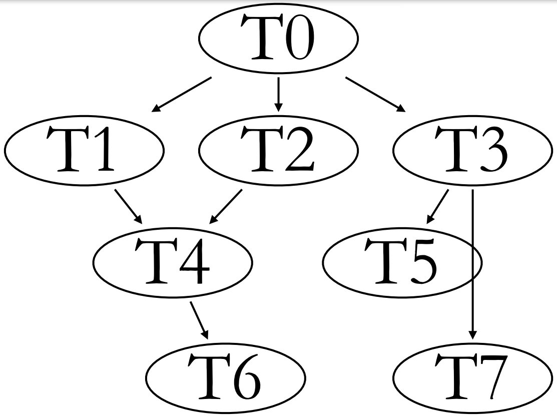
- Responsiveness guaranteed by Real-time semantics (hard or probabilistic)

Don’t Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz

Implement as a Real-Time App

Implement with High-Productivity, Large Scale Programming flows



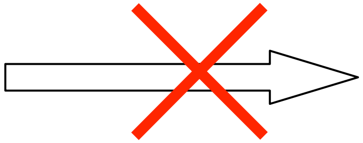
<u>Real-Time Task-Graph</u>

- Application decomposed into Tasks and Precedence Constraints

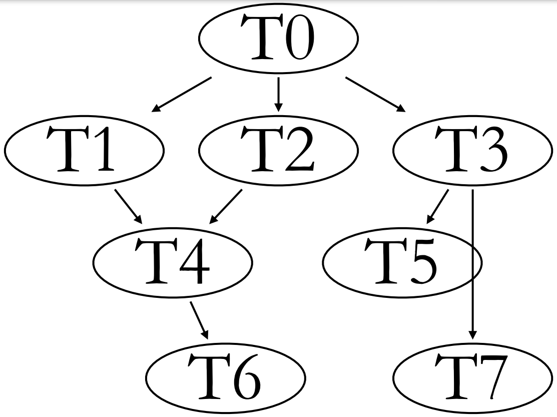
- Responsiveness guaranteed by Real-time semantics (hard or probabilistic)

Don’t Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz



Implement as a Real-Time App



Implement with High-Productivity, Large Scale Programming flows

<u>C, C++, Java: Monolithic App</u>

- 100Ks to Millions of LoC

- No analyzable structure for responsiveness and scaling

- Responsiveness and Quality entirely **emergent** attributes (currently tuning this is an art)

<u>Real-Time Task-Graph</u>

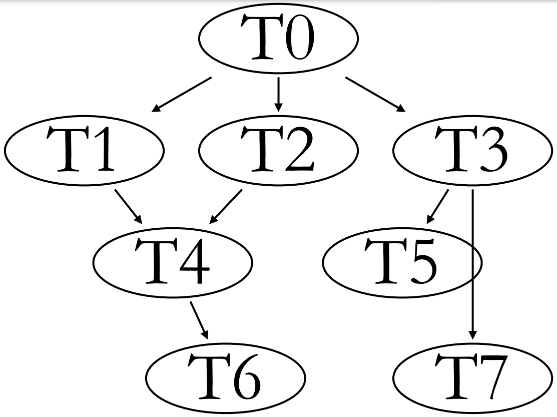
- Application decomposed into Tasks and Precedence Constraints

- Responsiveness guaranteed by Real-time semantics (hard or probabilistic)

Don’t Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz

Implement as a Real-Time App



Implement with High-Productivity, Large Scale Programming flows

<u>C, C++, Java: Monolithic App</u>

- 100Ks to Millions of LoC

- No analyzable structure for responsiveness and scaling

- Responsiveness and Quality entirely **emergent** attributes (currently tuning this is an art)

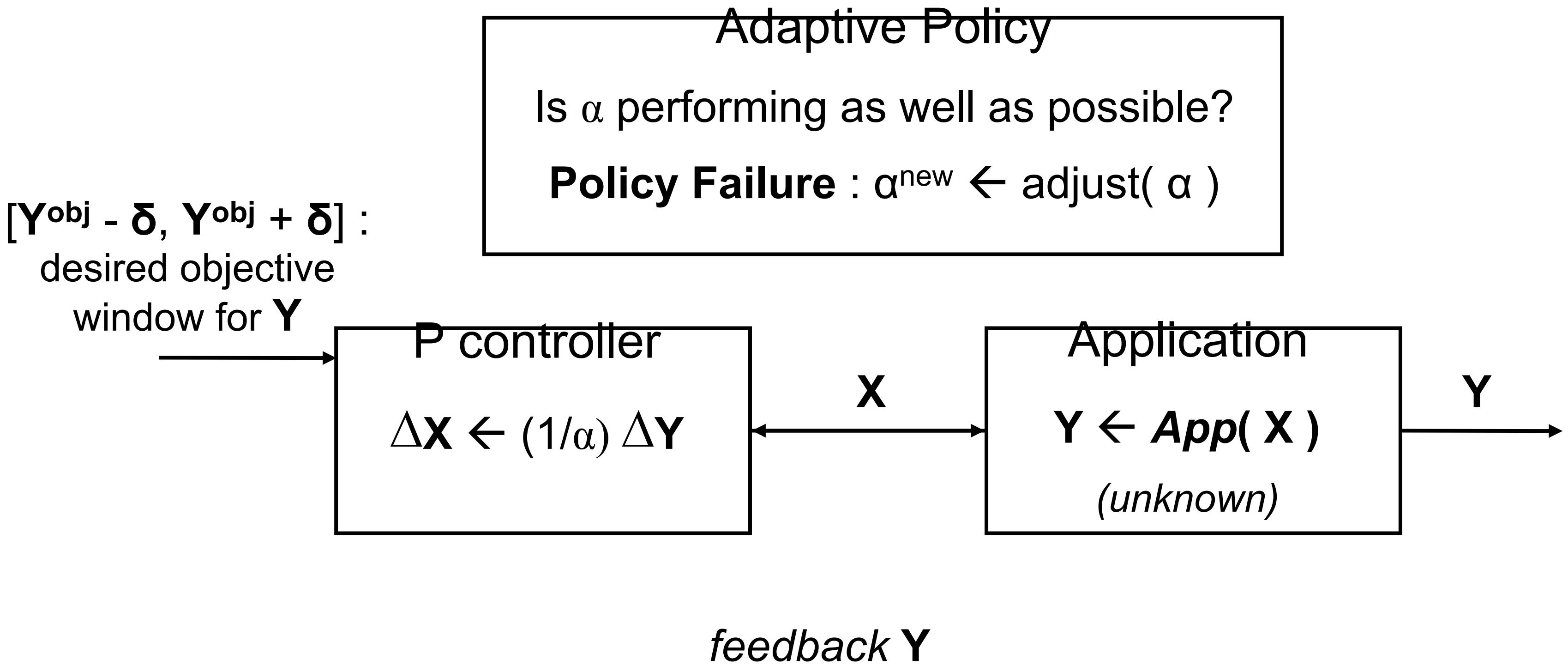
<u>Real-Time Task-Graph</u>

- Application decomposed into Tasks and Precedence Constraints

- Responsiveness guaranteed by Real-time semantics (hard or probabilistic)

Need a new bag of tricks to Scale Semantics in Monolithic Applications

Runtime Controller



Adaptive Policy

Is α performing as well as possible?

Policy Failure : α^{new} ← adjust(α)

[Y^{obj} - δ, Y^{obj} + δ] : desired objective window for **Y**

P controller

Δ**X** ← $(1/\alpha)$ Δ**Y**

X

Application

Y ← ***App*(X)**

(unknown)

Y

feedback **Y**