



ARTIST2 Summer School 2008 in Europe  
*Autrans (near Grenoble), France*  
*September 8-12, 2008*

# Feedback Performance Control of Distributed Computing Systems: A Real-time Perspective

Invited Speaker: Tarek F. Abdelzaher  
Department of Computer Science  
University of Illinois



# Feedback Performance Control of Distributed Computing Systems

- Feedback control has been a great success story in performance management of engineering and physical artifacts.
- It is time to advance a branch of theory that addresses feedback control of distributed **software** systems!





## Why Feedback Control of Software?

- **Claim 1:** In 10 years, most computing innovation will be focused on *distributed systems that interact with the physical world*
- **Claim 2:** They will operate under *increased uncertainty*
- **Claim 3:** They will be burdened with *increased autonomy*



# Where is Computer Science Research Going?

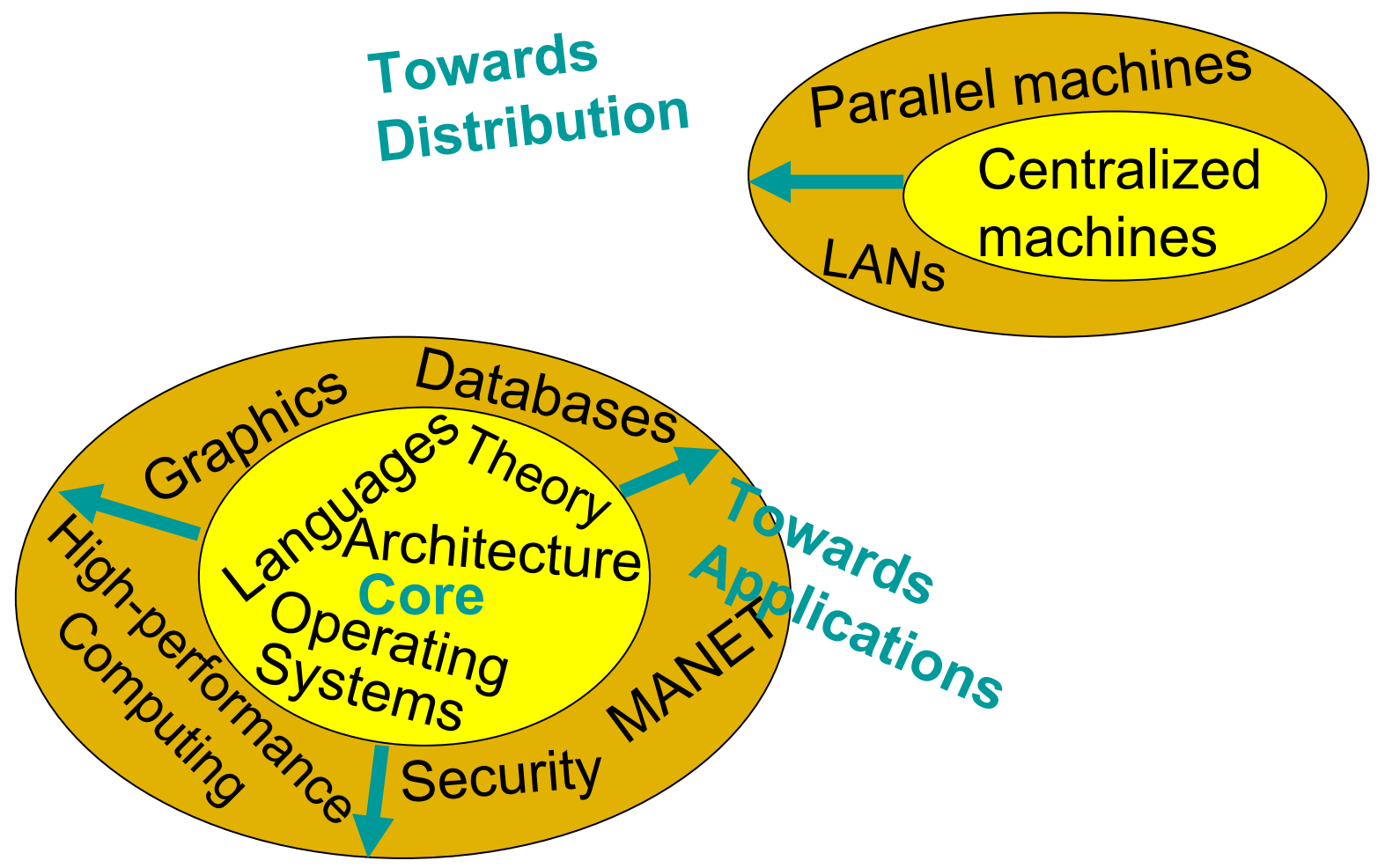
The beginning:

Centralized machines

Languages Theory  
Architecture  
**Core**  
Operating Systems

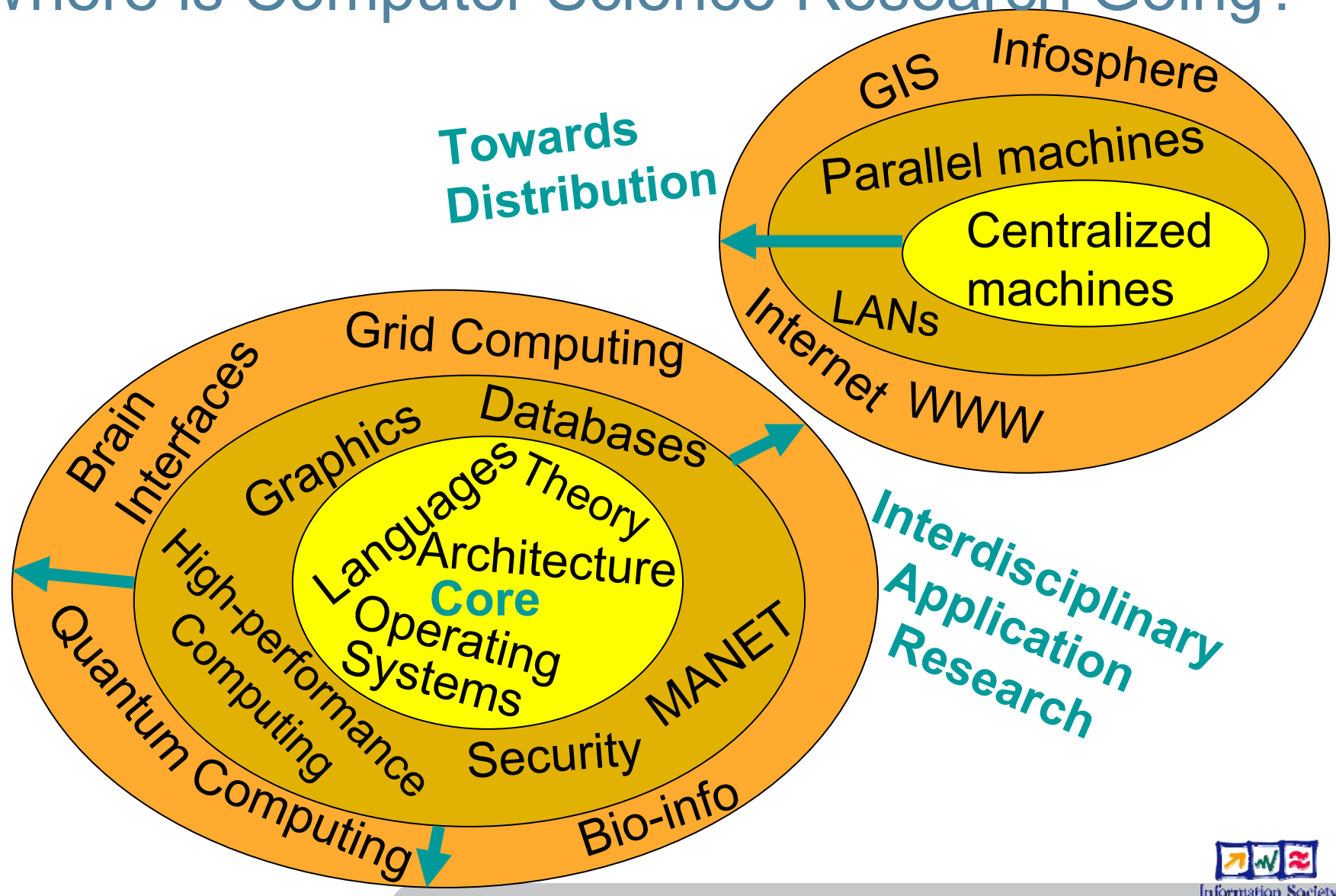


# Where is Computer Science Research Going?



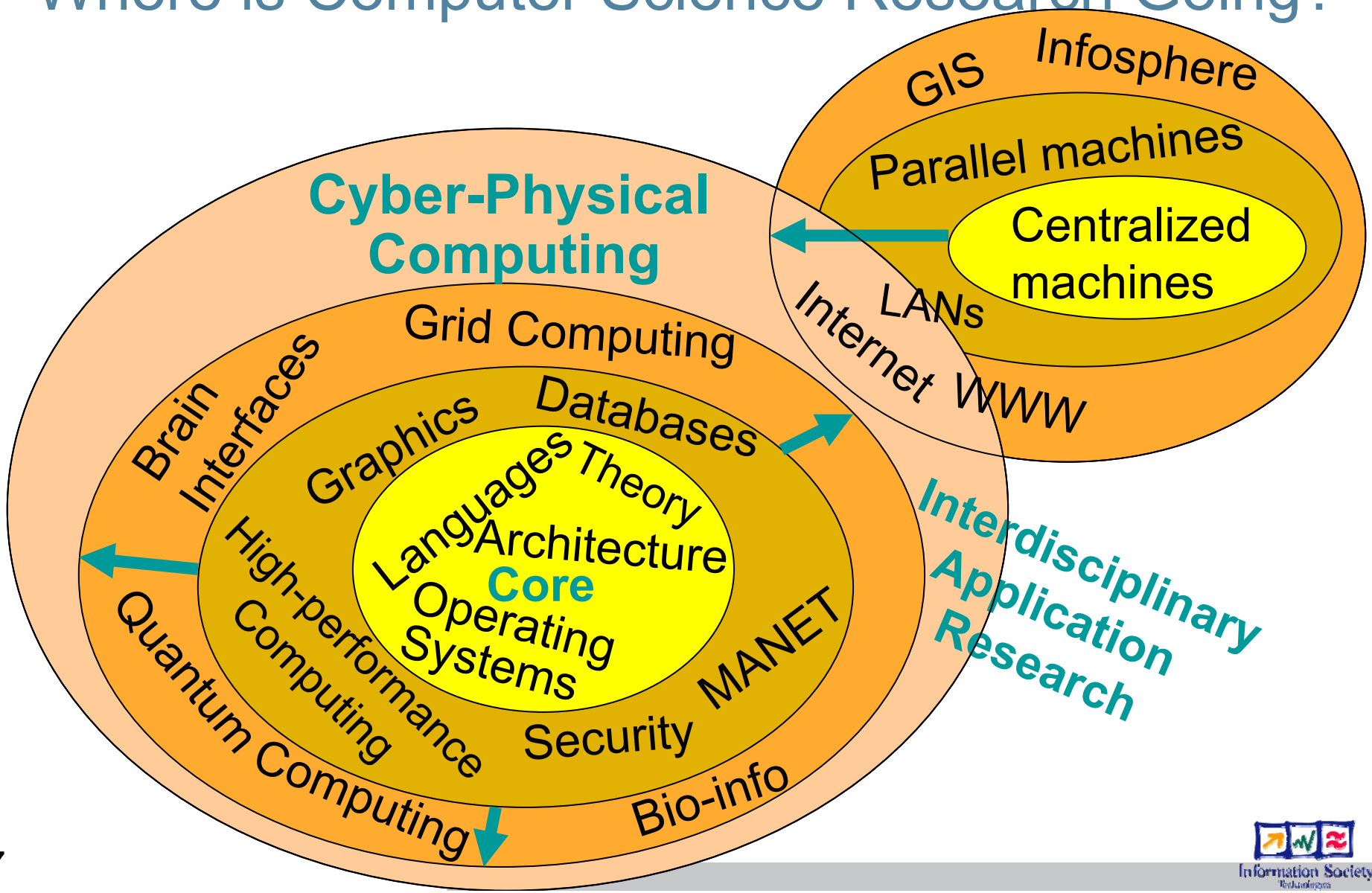


# Where is Computer Science Research Going?



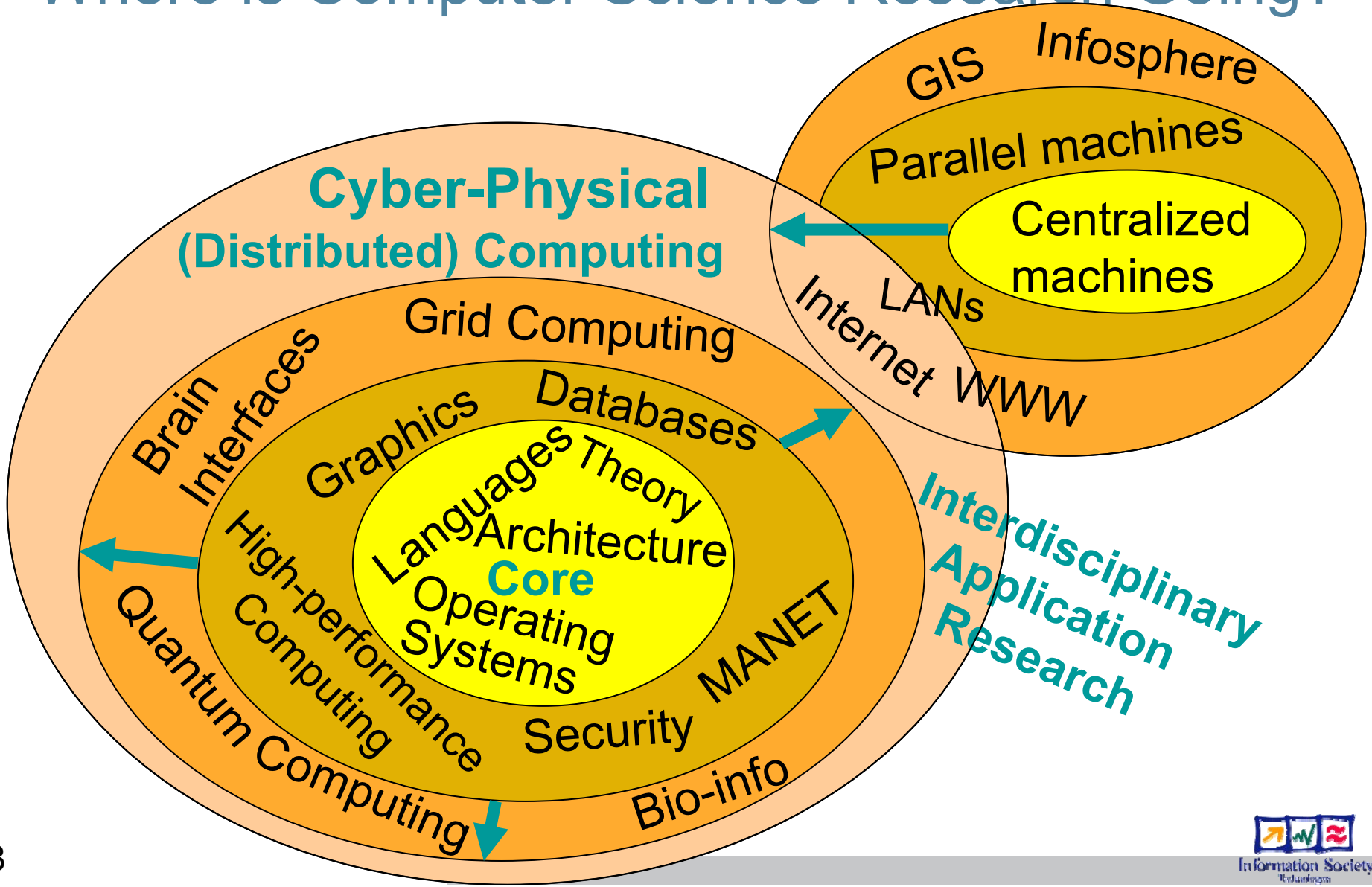


# Where is Computer Science Research Going?





# Where is Computer Science Research Going?





# Where is Computer Science Research Going?

**Claim 1:** In 10 years, most computing innovation will be focused on *distributed systems that interact with the physical world*

## Cyber-Physical Distributed Systems

For example:

In the US, the **Presidential Council of Advisors in Science and Technology** named systems that interact with the physical world the

**#1 Research Priority in the US**

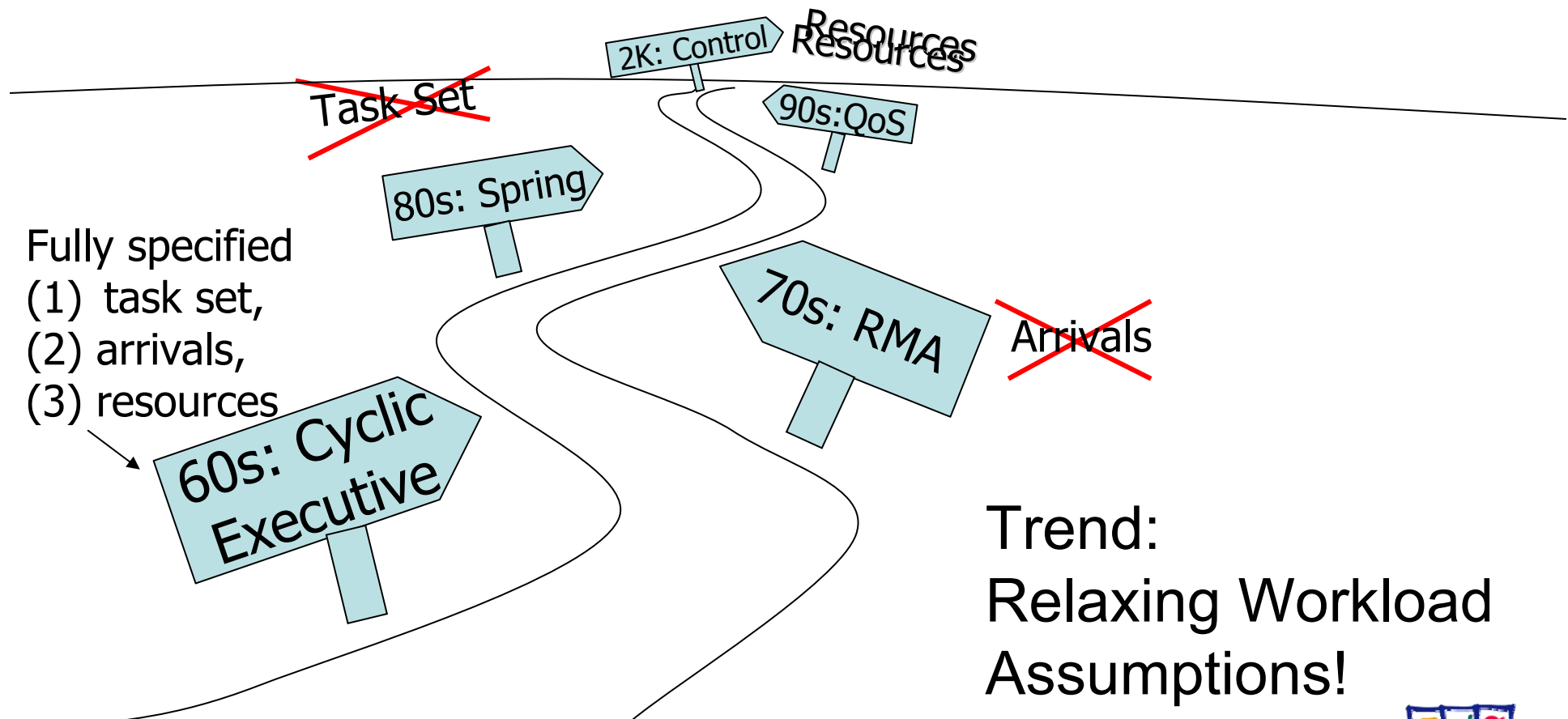
## Why Feedback Control of Software?

- **Claim 1:** In 10 years, most computing innovation will be focused on *distributed systems that interact with the physical world*
- **Claim 2:** They will operate under *increased uncertainty*
- **Claim 3:** They will be burdened with *increased autonomy*

## The Mounting Uncertainty

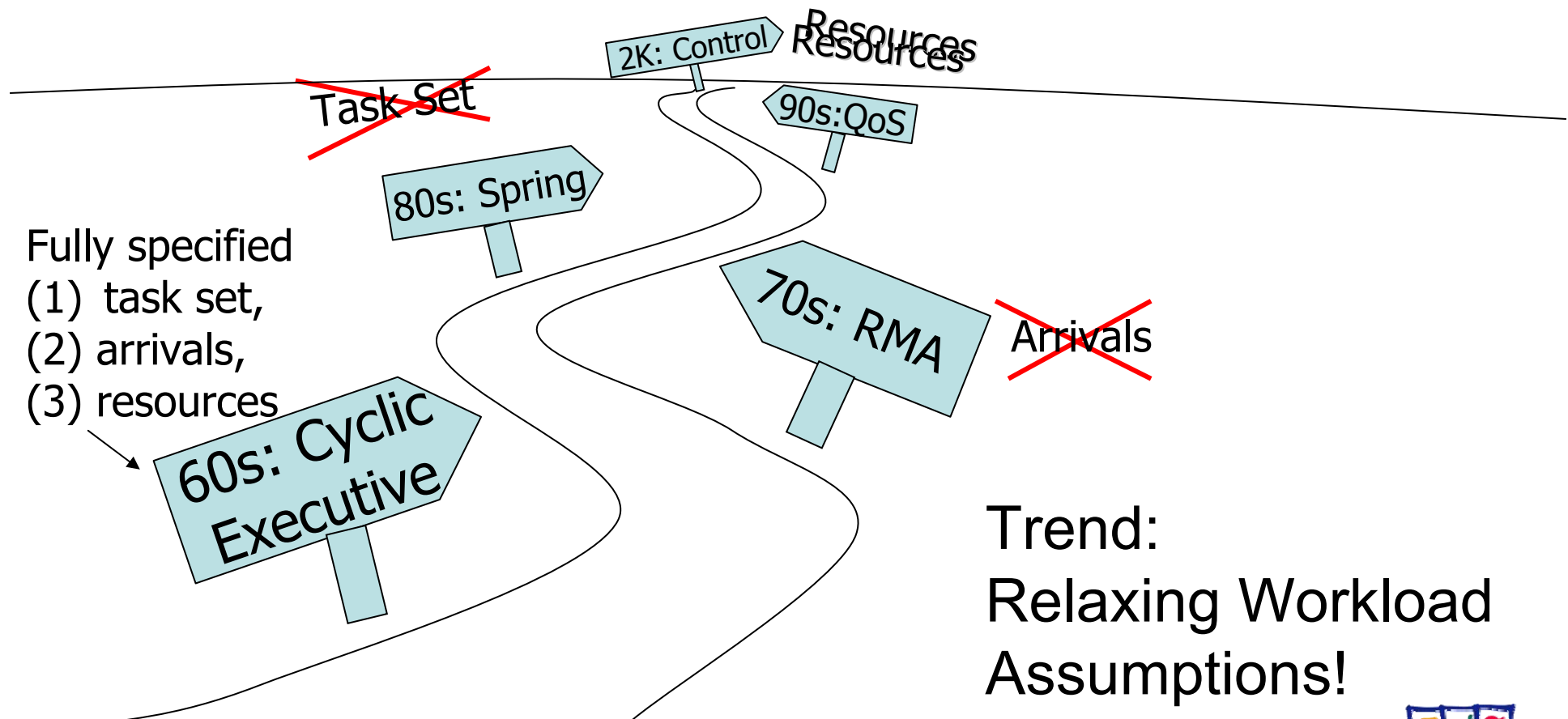
- Larger (distributed) systems
- Higher connectivity and interactive complexity
- Increasingly data-centric nature
  - Time it takes to execute is driven by data
  - Worst case is too pessimistic or unbounded
- More complex sensing at higher-level of abstractions
  - Data mining engines to convert data to actionable information
- Increasingly hybrid nature
  - Fusion of digital, analog, social, and biological models to understand overall behavior

# Evidence by Example: The Real-Time Scheduling Theory Roadmap



# Evidence by Example: The Real-Time Scheduling Theory Roadmap

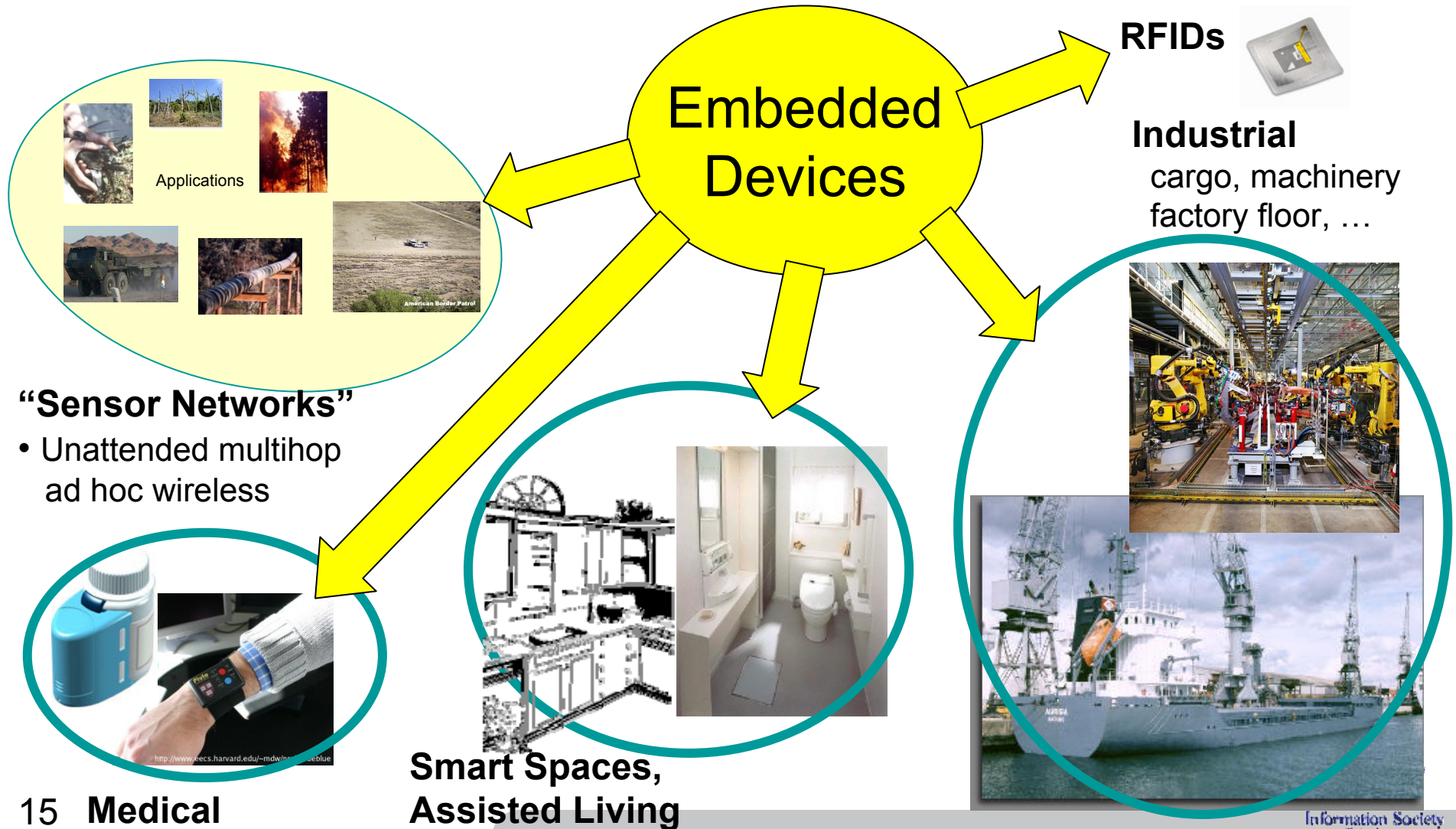
**Claim 2:** A significant challenge will be one of providing *guarantees under uncertainty*



## Why Feedback Control of Software?

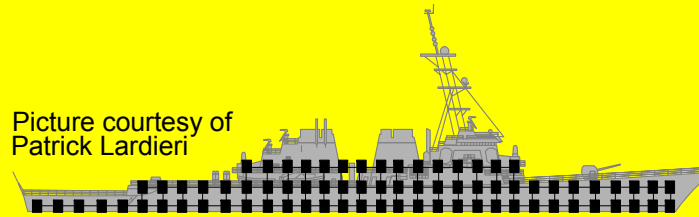
- **Claim 1:** In 10 years, most computing innovation will be focused on *distributed systems that interact with the physical world*
- **Claim 2:** They will operate under *increased uncertainty*
- **Claim 3:** They will be burdened with *increased autonomy*

# Factor #1: Device Proliferation (By Moore's Law)



# Factor #2: Integration at Scale (Isolation costs!)

- Low end: ubiquitous embedded devices
  - Large-scale networked embedded systems
  - Seamless integration with a physical environment



Picture courtesy of Patrick Lardieri

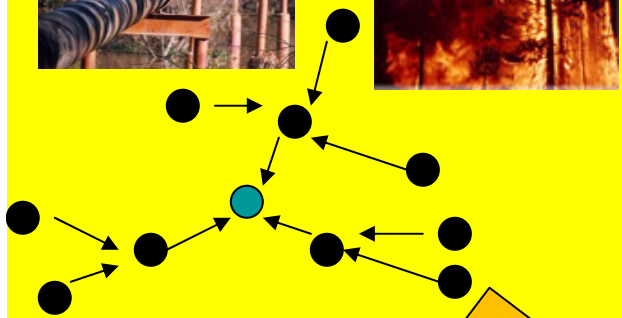
Total Ship Computing Environment (TSCE)

- High end: complex systems with global integration
  - Examples: Global Information Grid, Total Ship Computing Environment

Global Information Grid



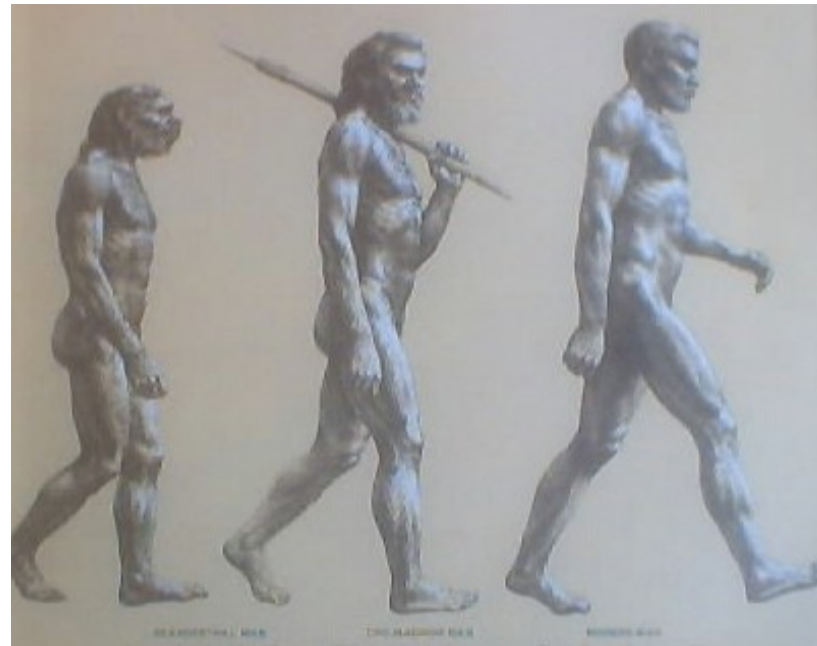
Future Combat System (Rob Gold)



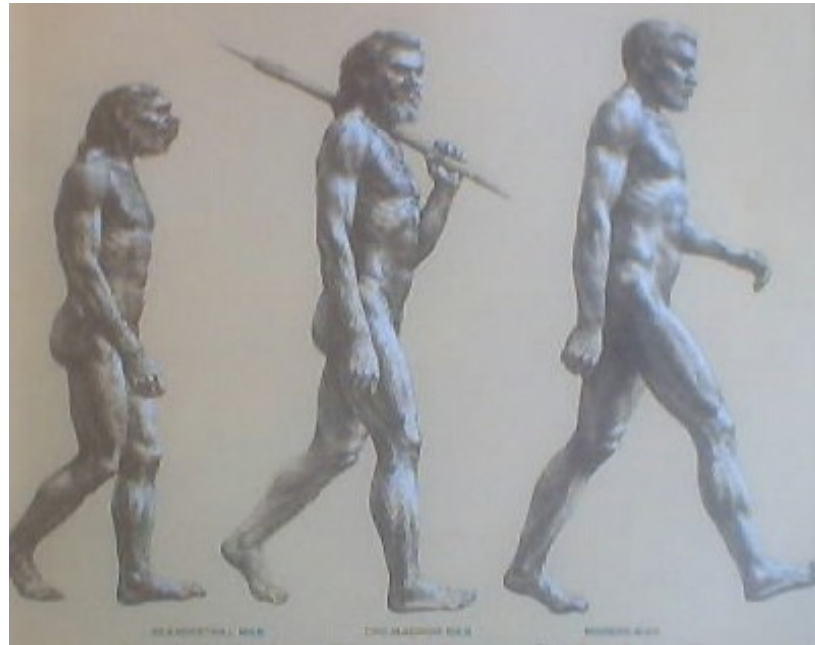
World Wide Sensor Web (Feng Zhao)

Integration and Scaling Challenges

## Factor #3: Biological Evolution



## Factor #3: Biological Evolution

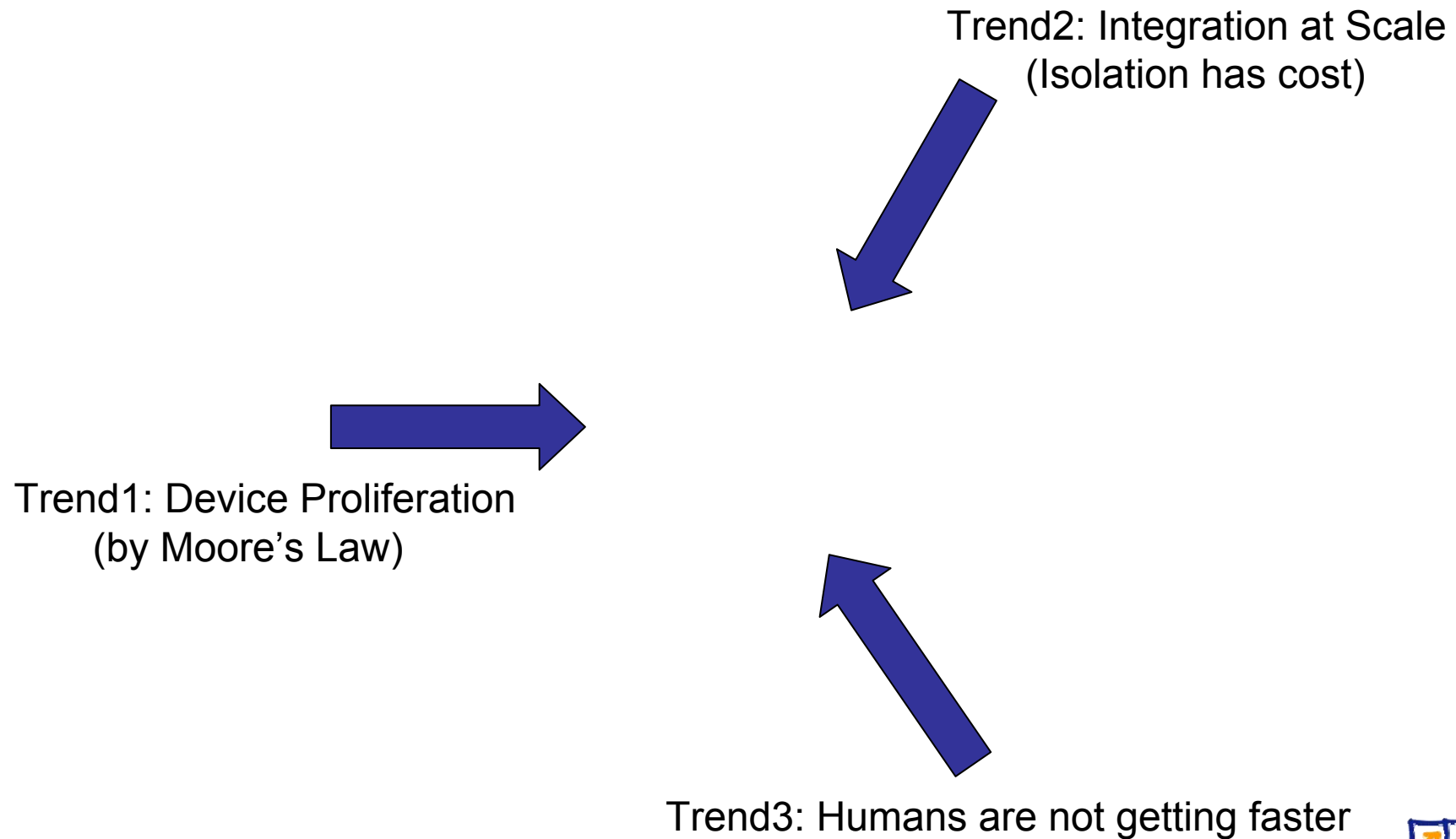


- **It's too slow!**

- The exponential proliferation of data sources (afforded by Moore's Law) is **not** matched by a corresponding increase in human ability to consume information!

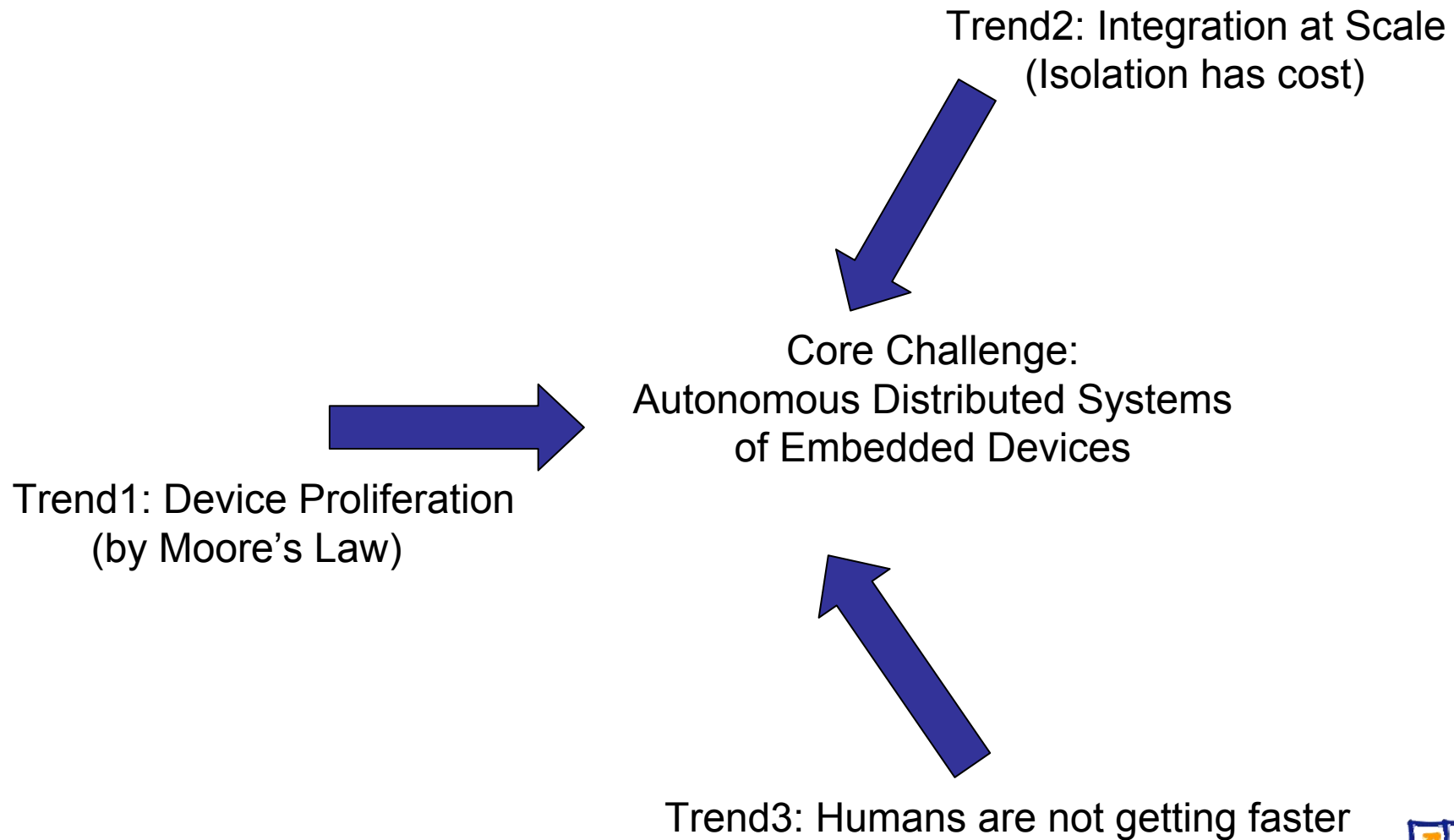


# Confluence of Trends The Overarching Challenge



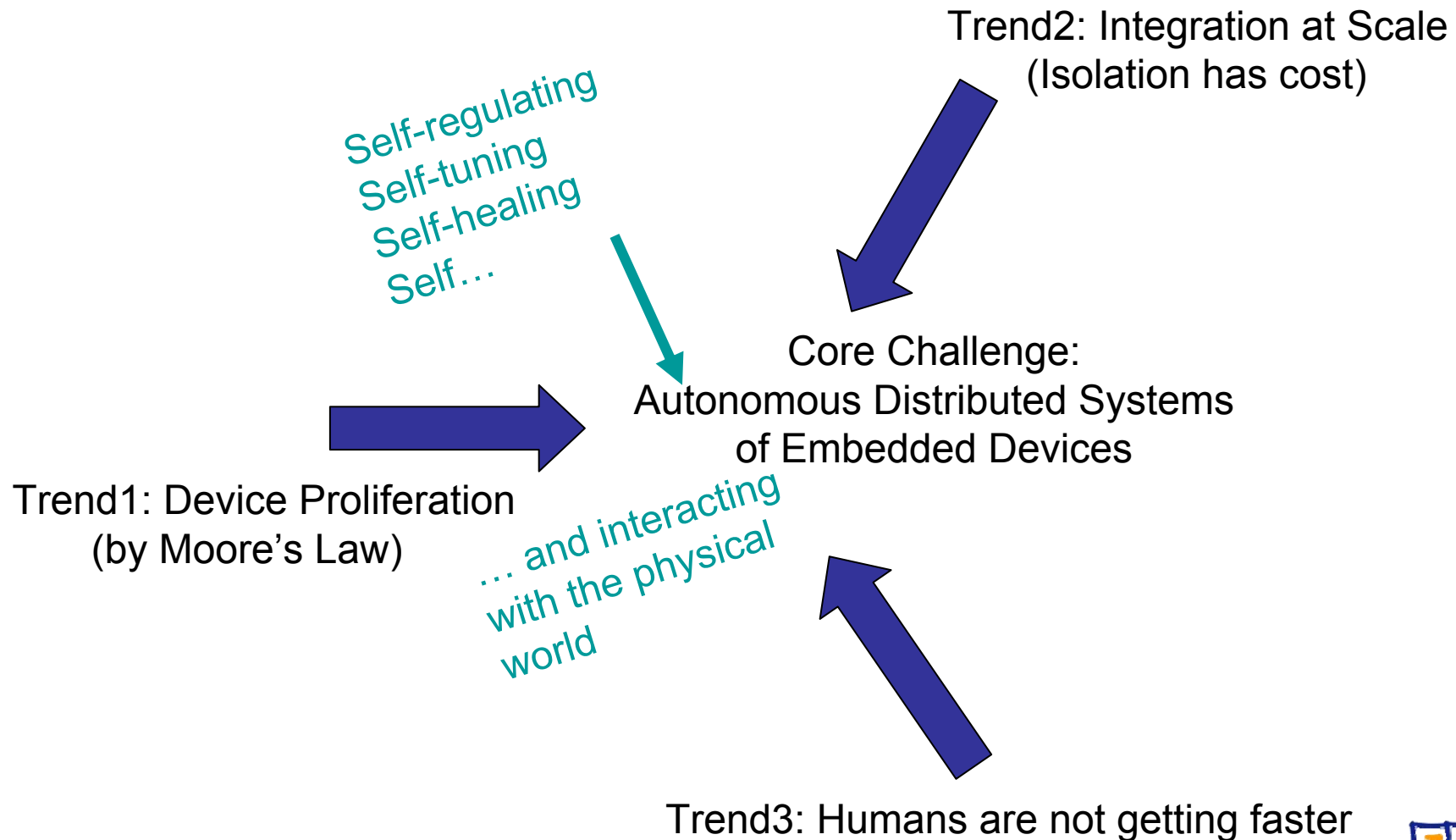
# Confluence of Trends

## The Overarching Challenge



# Confluence of Trends

## The Overarching Challenge



## Why Feedback Control of Software?

- **Claim 1:** In 10 years, most computing innovation will be focused on *distributed systems that interact with the physical world*
- **Claim 2:** They will operate under *increased uncertainty*
- **Claim 3:** They will be burdened with *increased autonomy*

# Examples

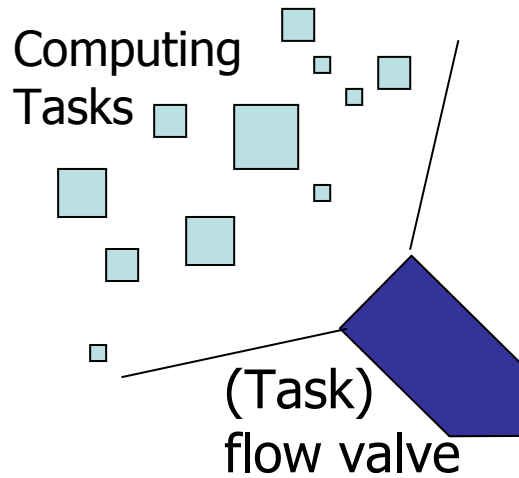
- Distributed vehicular traffic control algorithms for mass (e.g., city scale) evacuation in disaster scenarios
  - Hundreds of thousands of vehicles
  - Poor communication infrastructure (mostly vehicle to vehicle communication)
  - Complex time-varying underlying topology (accidents, gridlock, obstructions, ...)
  - Need for personalized directions to balance load, optimize throughput, etc.
- Large energy-optimal data centers
  - Tens of thousands of machines
  - Hundreds of performance management knobs, including computing and cooling
  - Time-varying demand on multiple time scales
  - Need to minimize energy (cost) while meeting service-level agreements
- Utility optimizing wireless ad hoc networks

# Feedback Performance Control of Distributed Software Systems

- Software performance optimization is a **resource management problem**
  - Resource allocation and scheduling are adjusted dynamically in response to external stimuli to optimize an objective while meeting constraints
- A key challenge: **bridge the levels of abstraction**
  - Software: tasks, priorities, deadlines, queues, arrival times, precedence constraints, ...
  - Control: state variables, deviations, dynamic models, stability conditions, ...
- A key challenge: **formulate and solve a control problem**

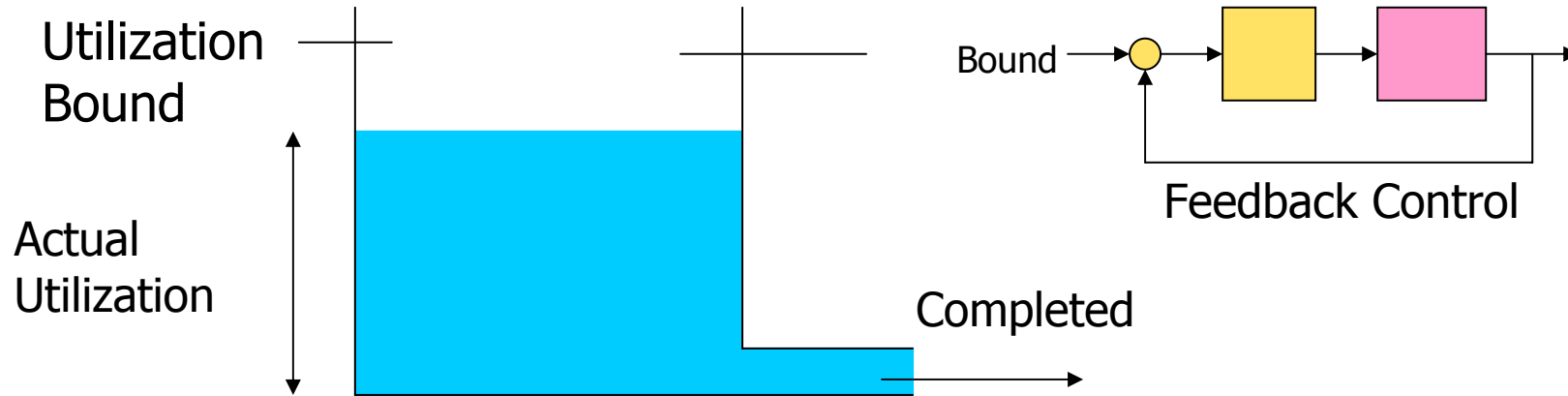


# Bridging the Levels of Abstraction: An Example



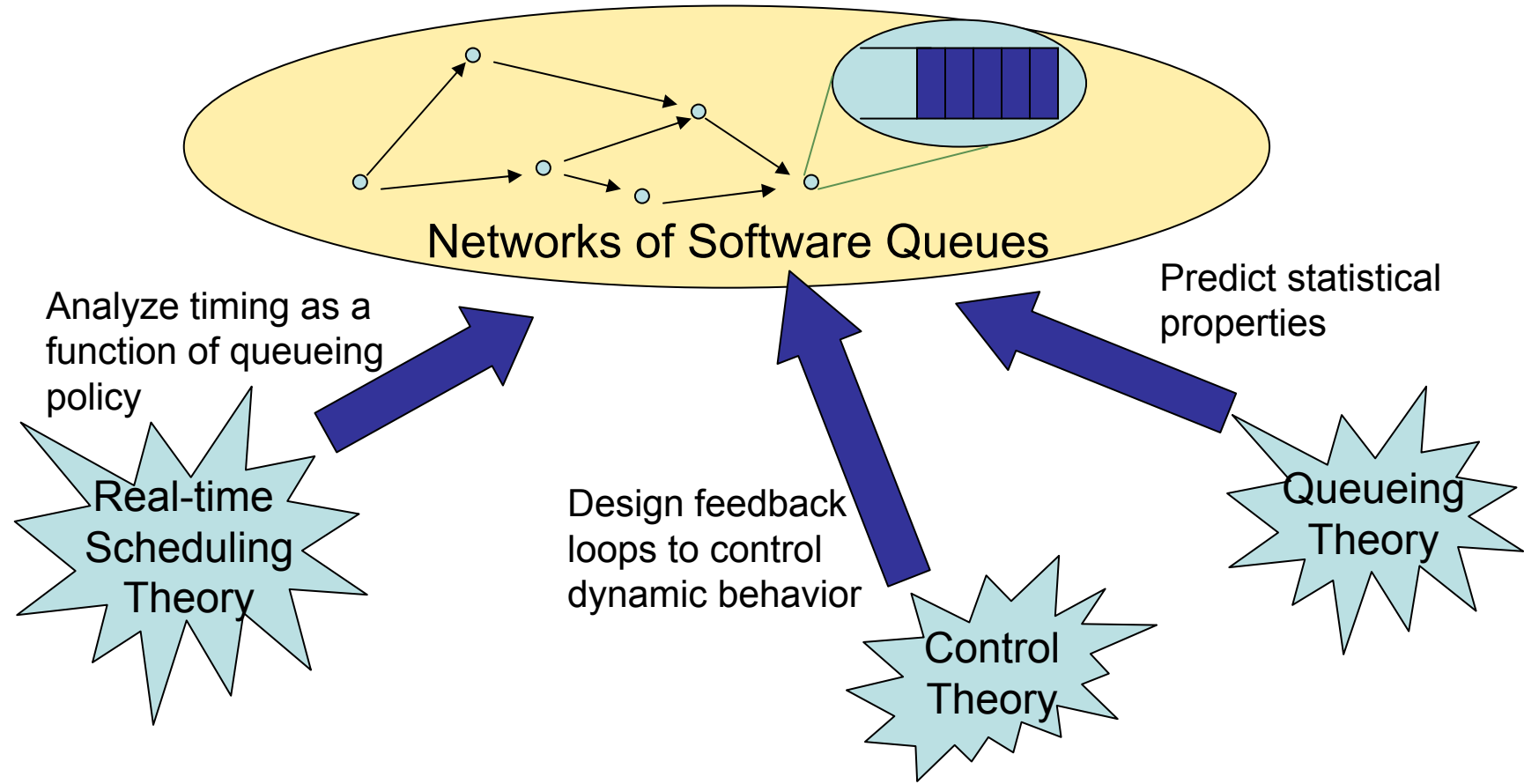
Using feedback control to remove deadline misses while maximizing throughput:

- Compute an aggregate workload or utilization bound such that all tasks are schedulable if bound is not exceeded.
- Control the aggregate workload not to exceed the bound (a form of level control).



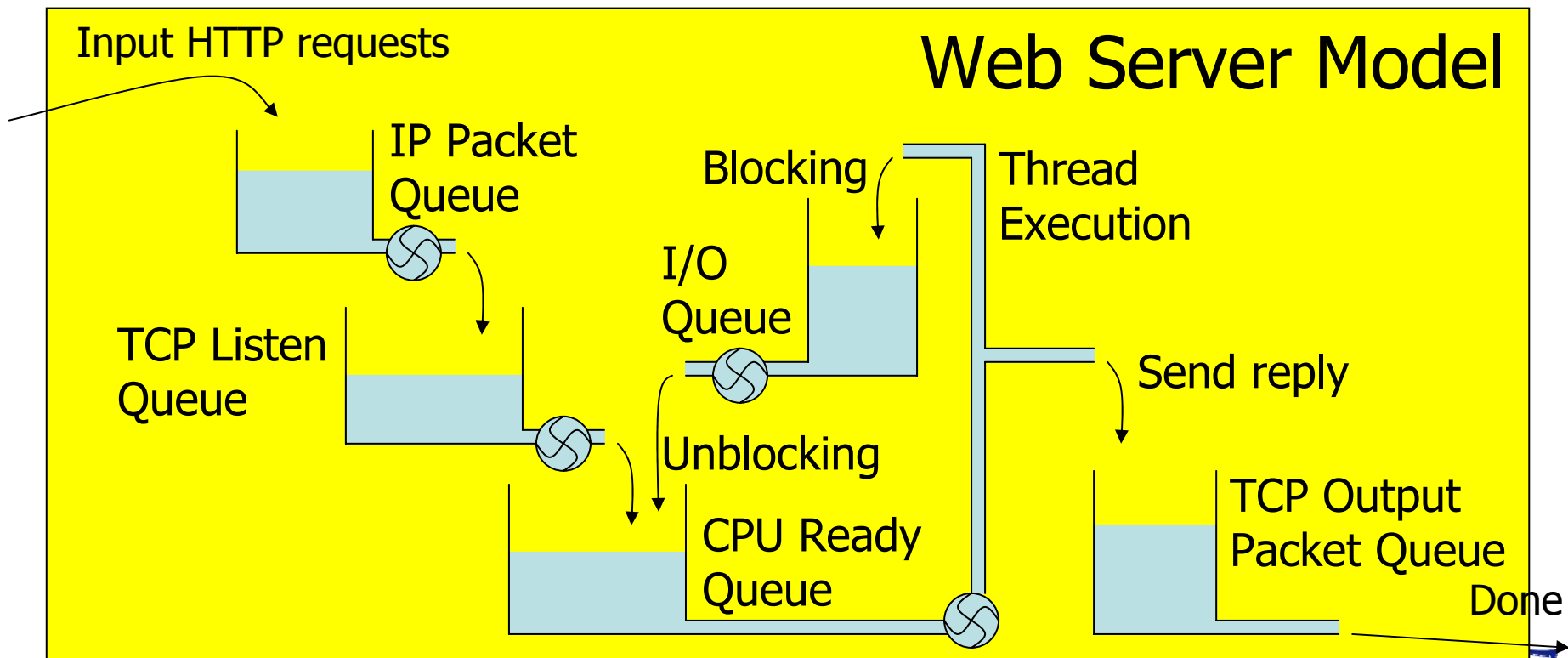


# Software Queues: A Unifying Construct in Software Performance Control



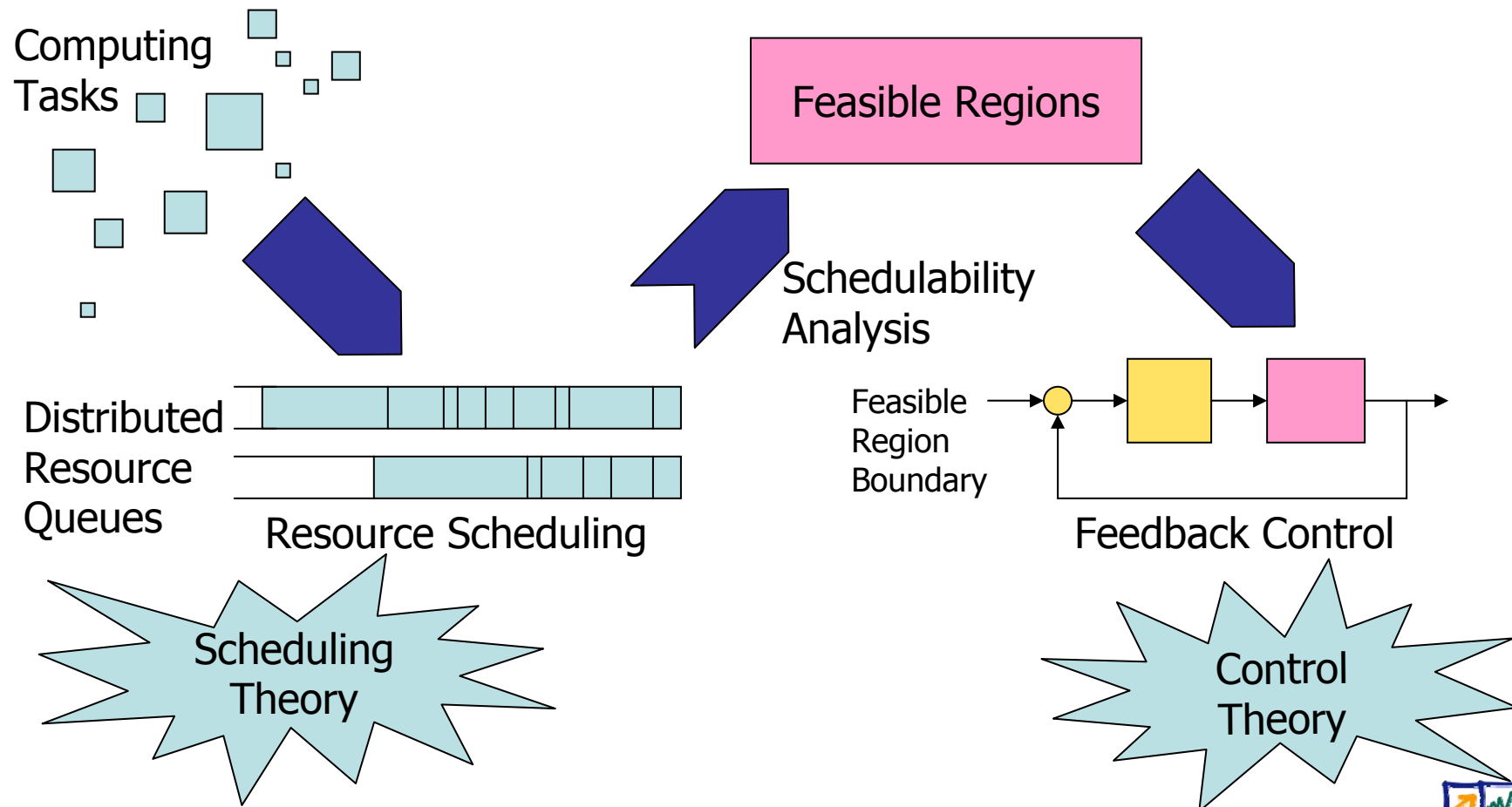
# Example: A Web Server Model

Why web servers can be modeled by difference equations!



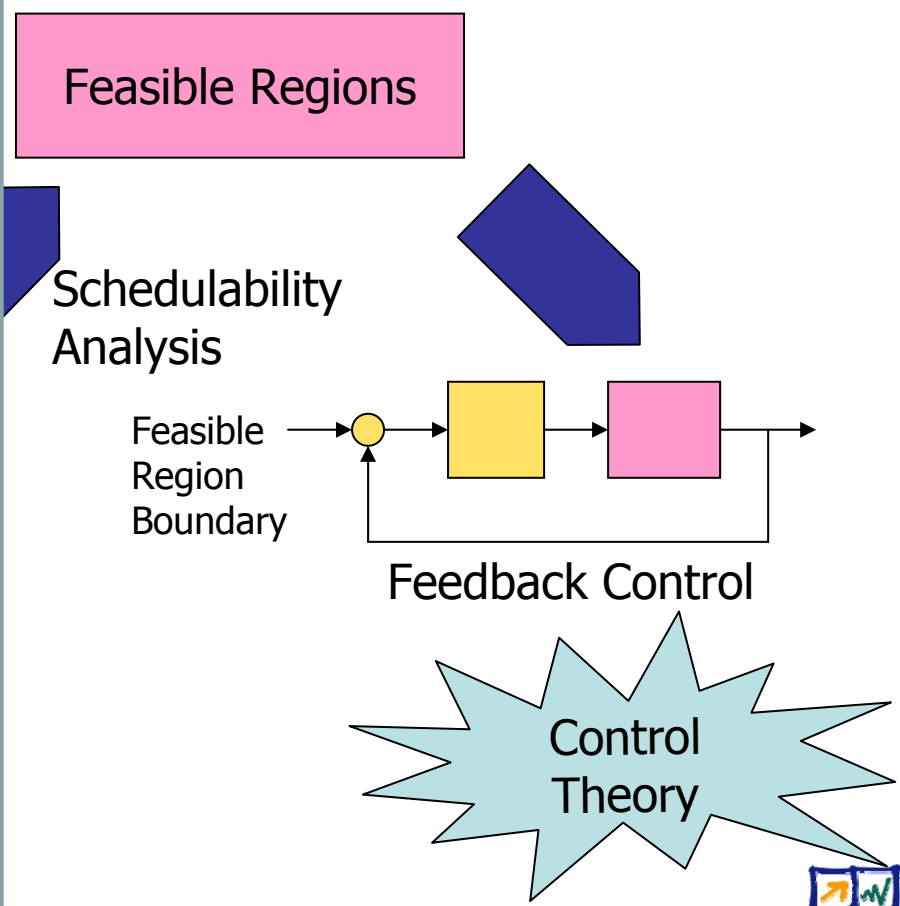
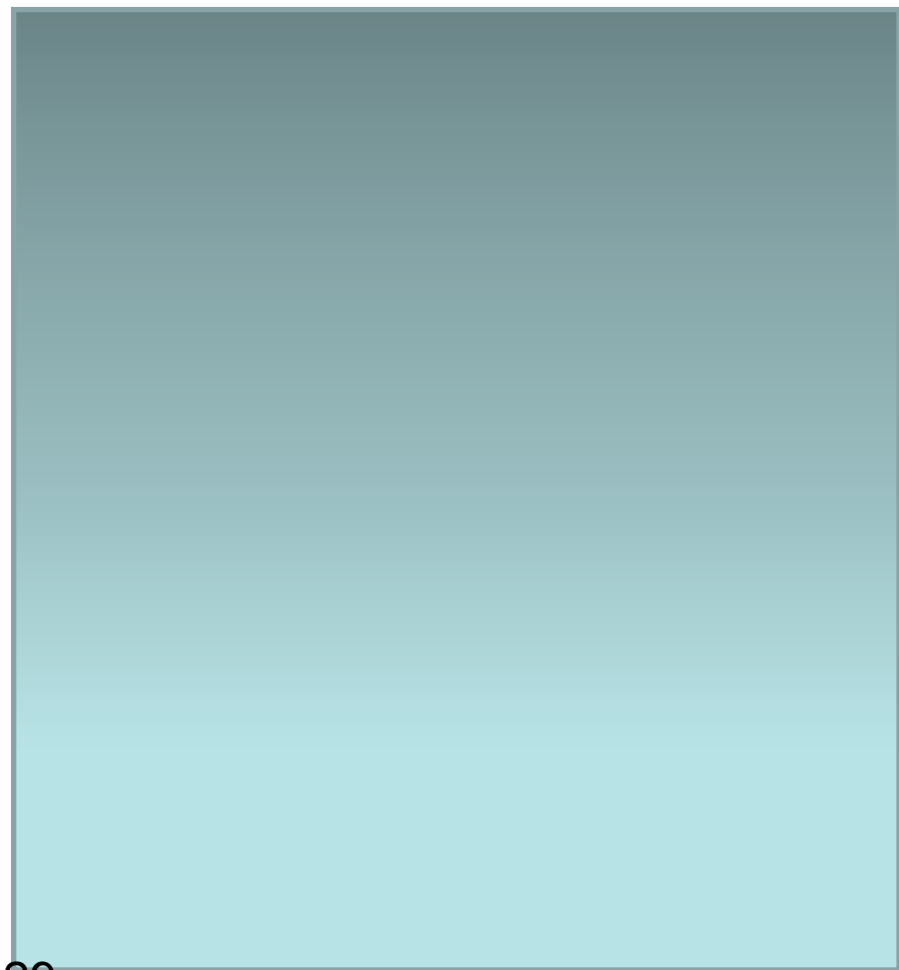


# Distributed Software Performance Control



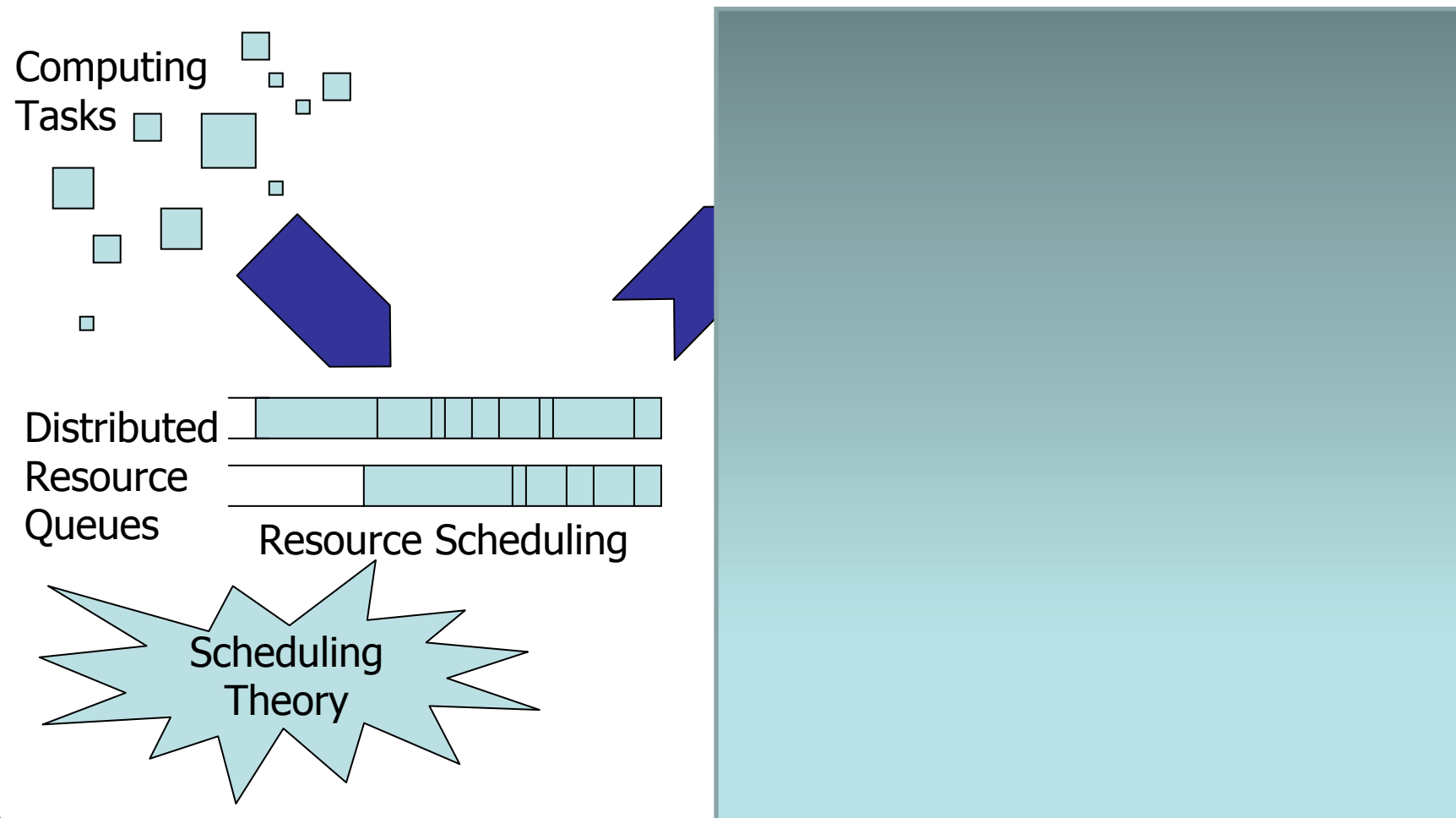
# Distributed Software Performance Control

Part I: Bridge the abstractions: low-level metrics  $\rightarrow$  queue state



# Distributed Software Performance Control

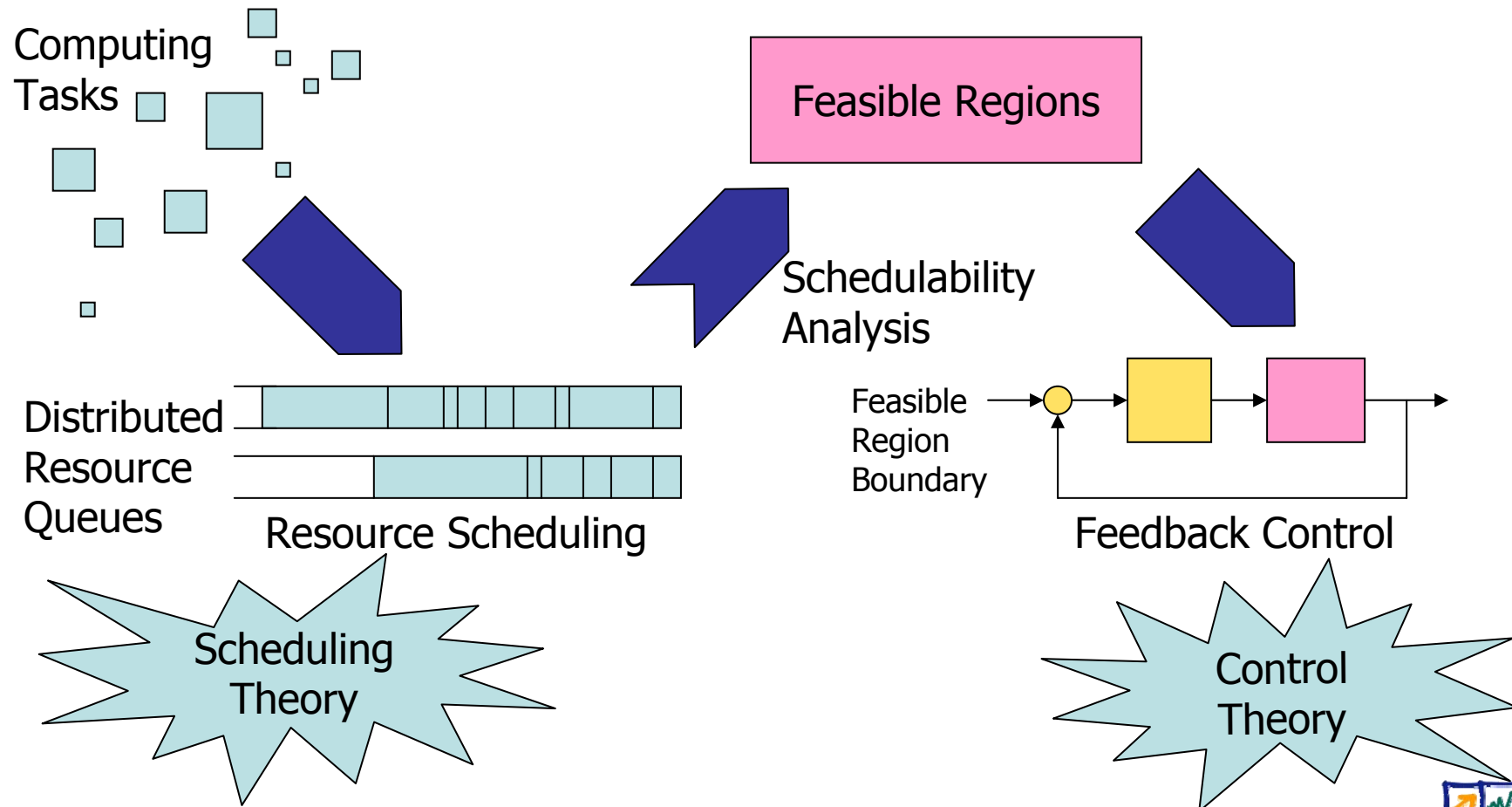
## Part II: Formulate and solve a (queue state) control problem





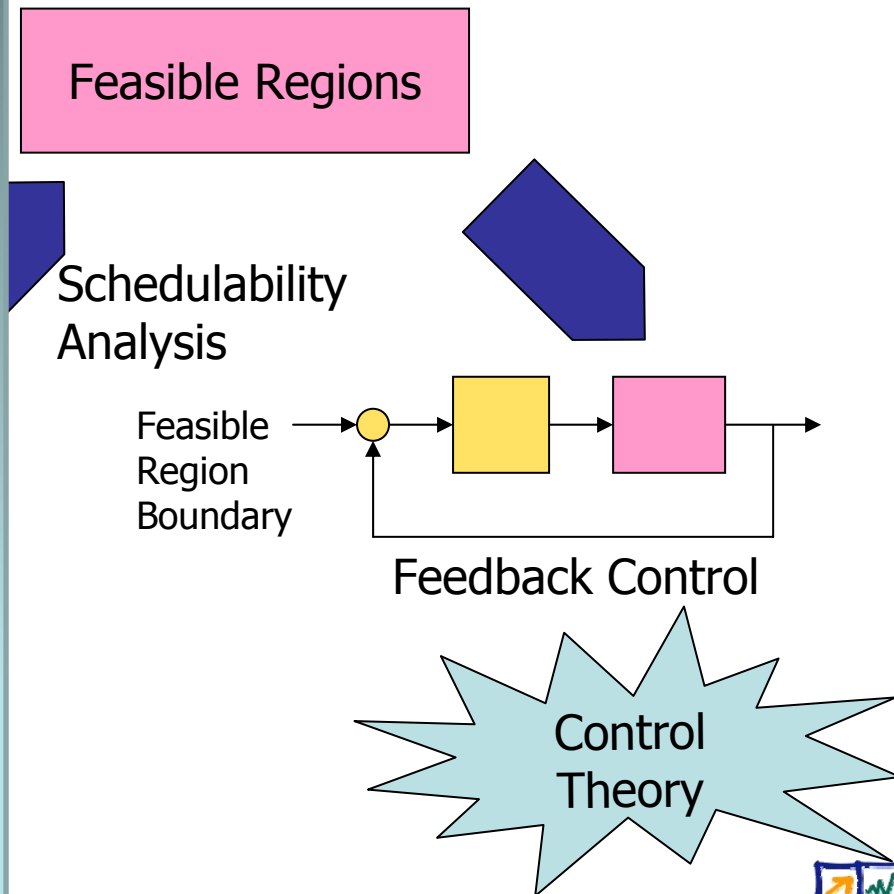
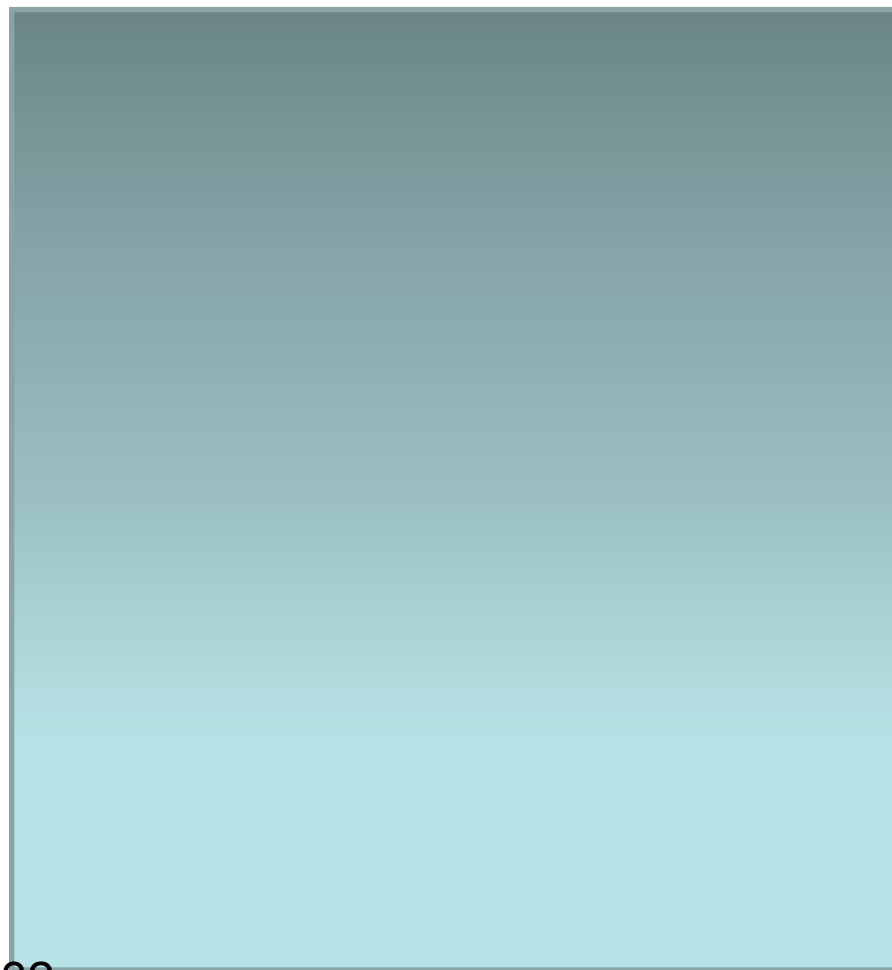
# Bridging the Abstractions: A Schedulability Study

Requirements on meeting deadlines → utilization (queue state) → performance control



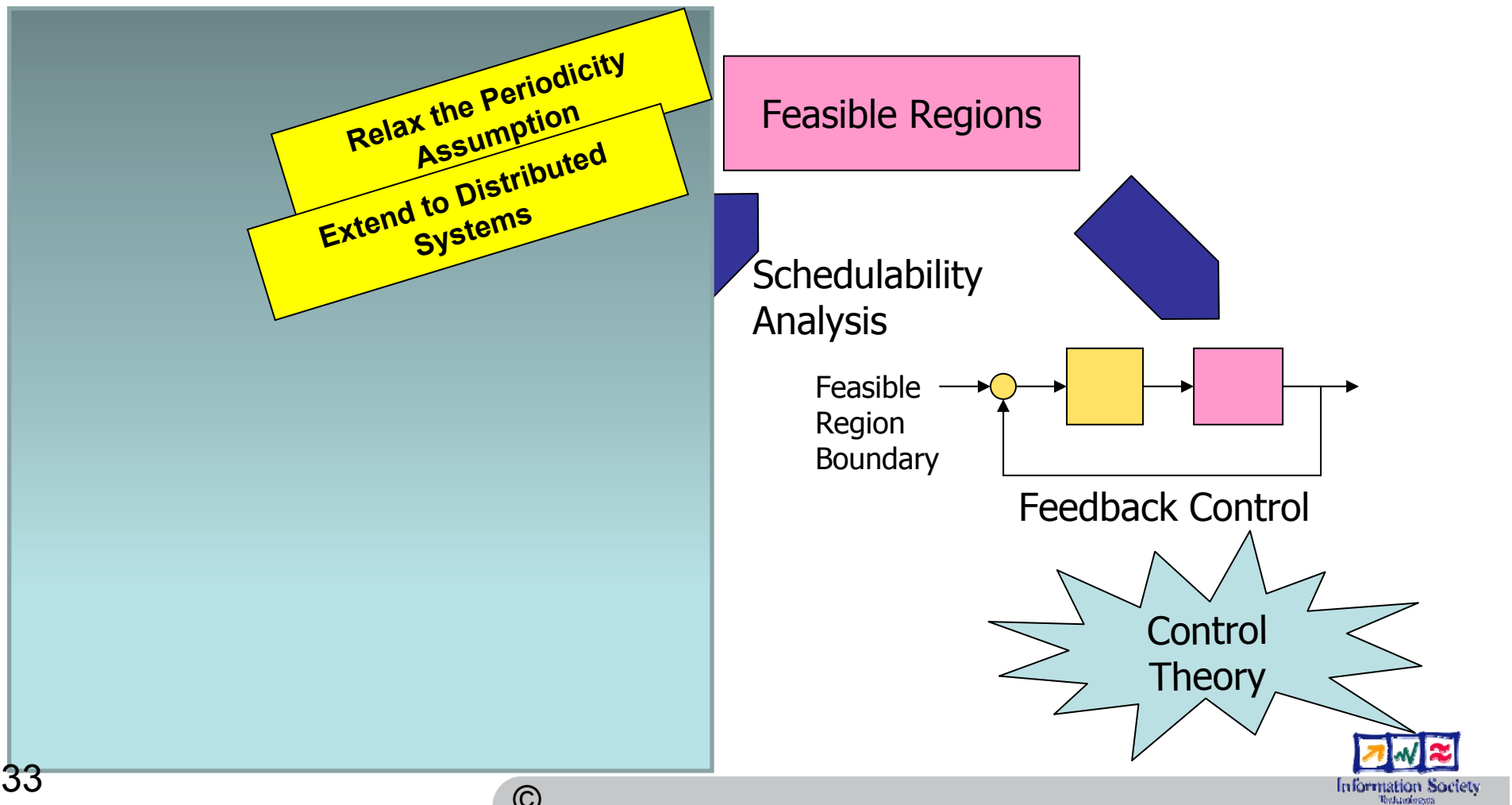
# Bridging the Abstractions: A Schedulability Study

Requirements on meeting deadlines → utilization (queue state) → performance control



# Bridging the Abstractions: A Schedulability Study

Requirements on meeting deadlines → utilization (queue state) → performance control



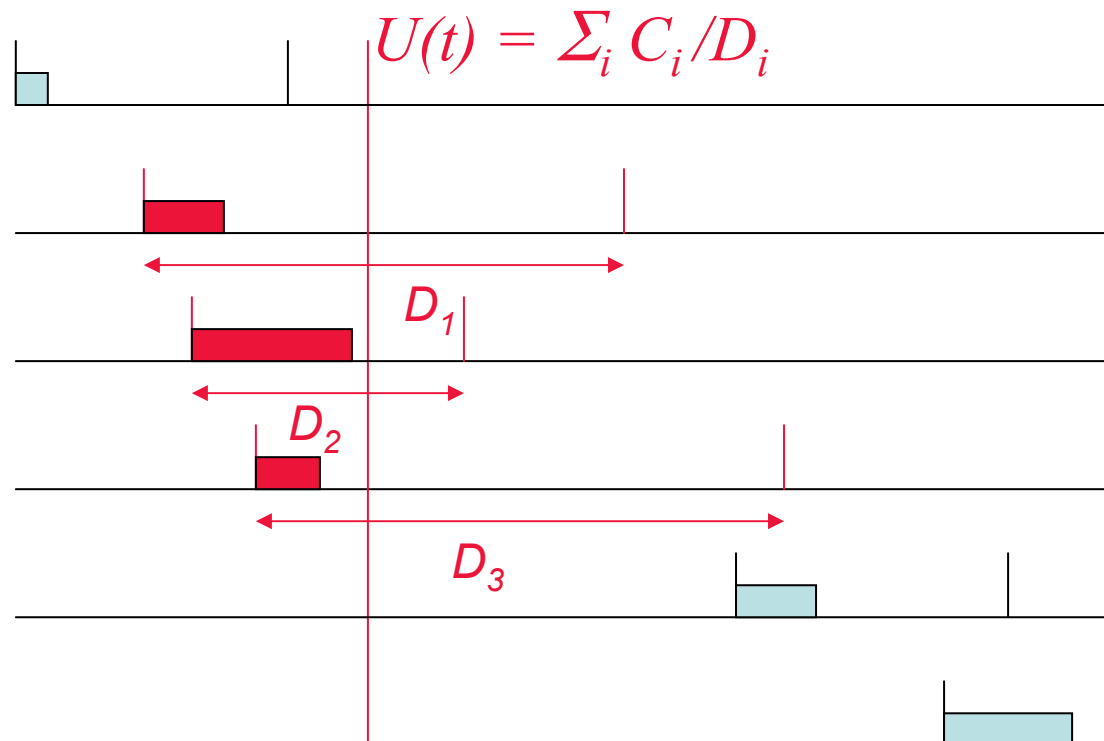


## Relaxing the Periodicity Assumption

- Is there a utilization bound such that a system of aperiodic tasks arriving at arbitrary time instances meets all deadlines as long as the bound is not exceeded?
- If so, this bound would make a good control set point.

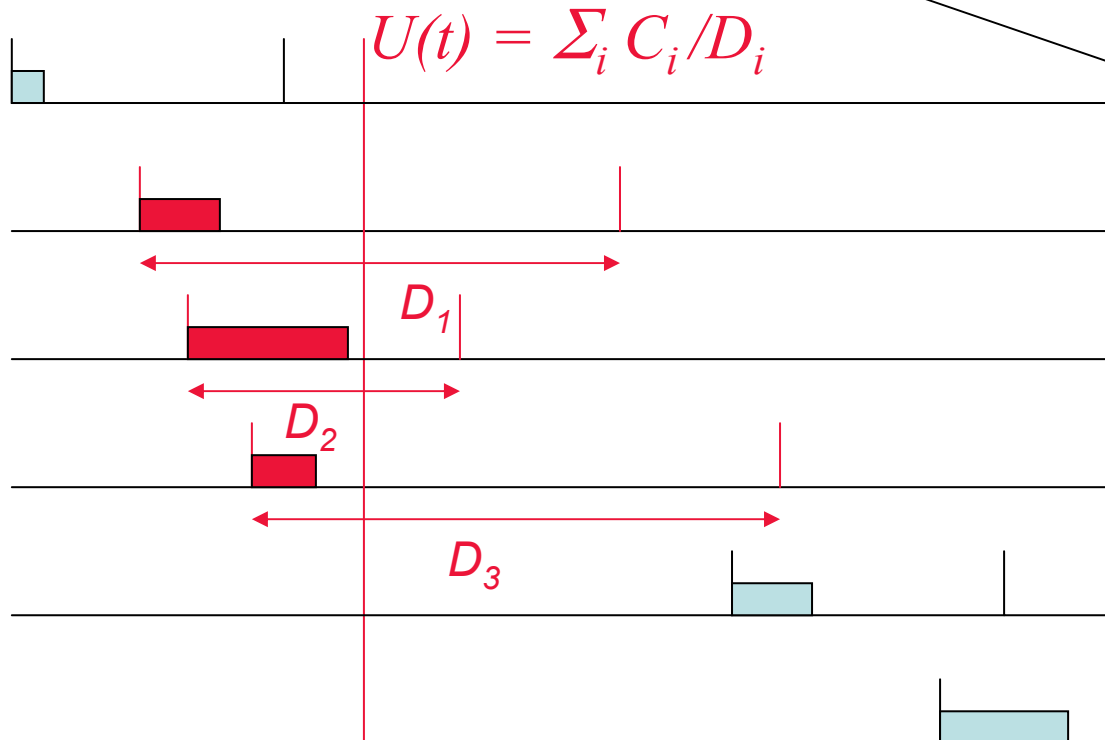
# Aperiodic Tasks and *Instantaneous* Utilization

- Instantaneous utilization  $U(t)$  is a function of time,  $t$
- $U(t)$  is defined over the *current* invocations



# Aperiodic Tasks and *Instantaneous* Utilization

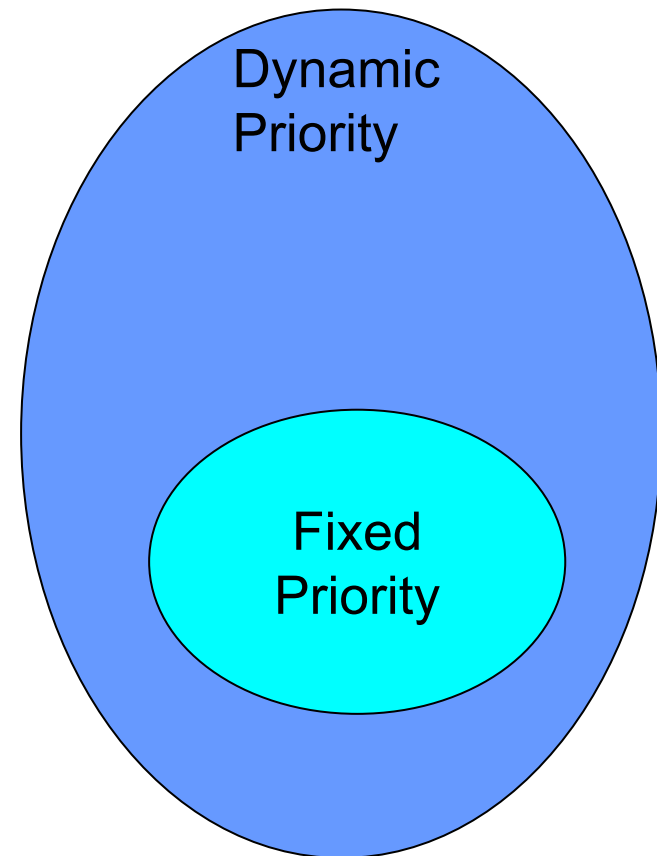
- Instantaneous utilization  $U(t)$  is a function of time,  $t$
- $U(t)$  is defined over the **current** invocations



Arrived but deadline has not expired

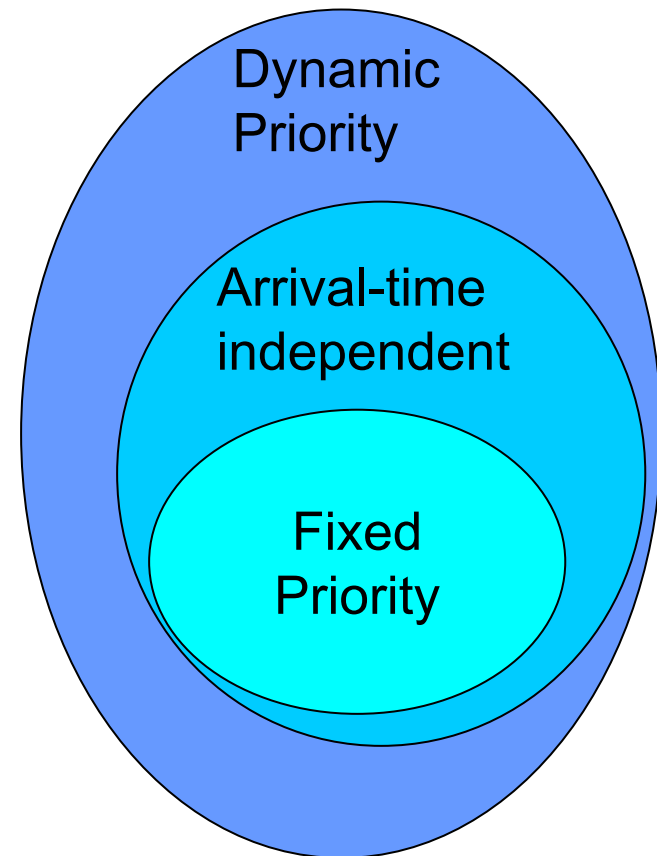
# Fixed versus Dynamic Priority Scheduling

- **Fixed-priority scheduling:**
  - All invocations of a task have same priority
- **Dynamic-priority scheduling:**
  - Invocation priorities may not be the same
- **What about Aperiodic Tasks?**
  - Equivalent for fixed priority scheduling?



# Arrival-Time-Independent Scheduling

- **Fixed-priority scheduling:**
  - All invocations of a task have same priority
- **Dynamic-priority scheduling:**
  - Invocation priorities may not be the same
- **Arrival-time-independent scheduling:**
  - Invocation priorities are not a function of invocation arrival times



## Why Arrival-Time Independent Scheduling?

- Easy to implement on current non-real-time operating systems with fixed-priority support (e.g., UNIX, the #1 OS for web servers)
  - Requires a finite number of priority levels
  - Priorities are statically assigned to threads

## A Sense of Optimality

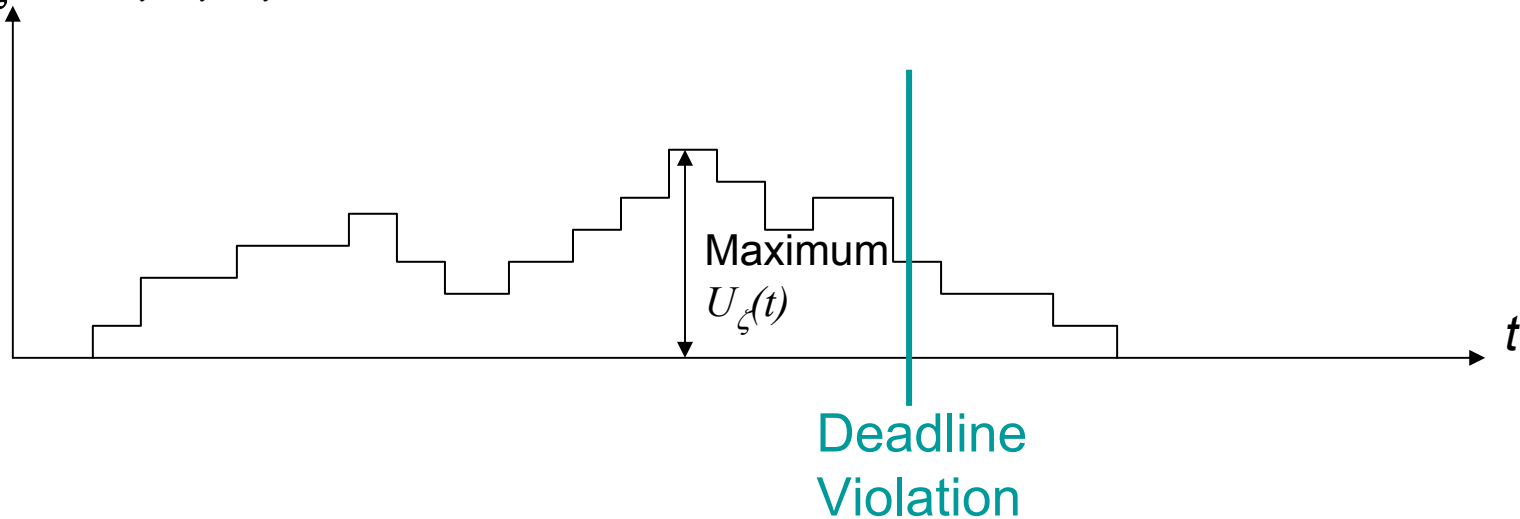
- A scheduling policy is optimal in a class if it maximizes the schedulable utilization bound among all policies in the class
- “Backward Compatibility”:
  - Rate monotonic is the optimal fixed-priority policy (for periodic tasks)
  - EDF is optimal dynamic-priority policy
  - **New:** Deadline monotonic is the optimal arrival-time independent policy

# Deriving a Utilization Bound for Aperiodic Tasks

## Main idea:

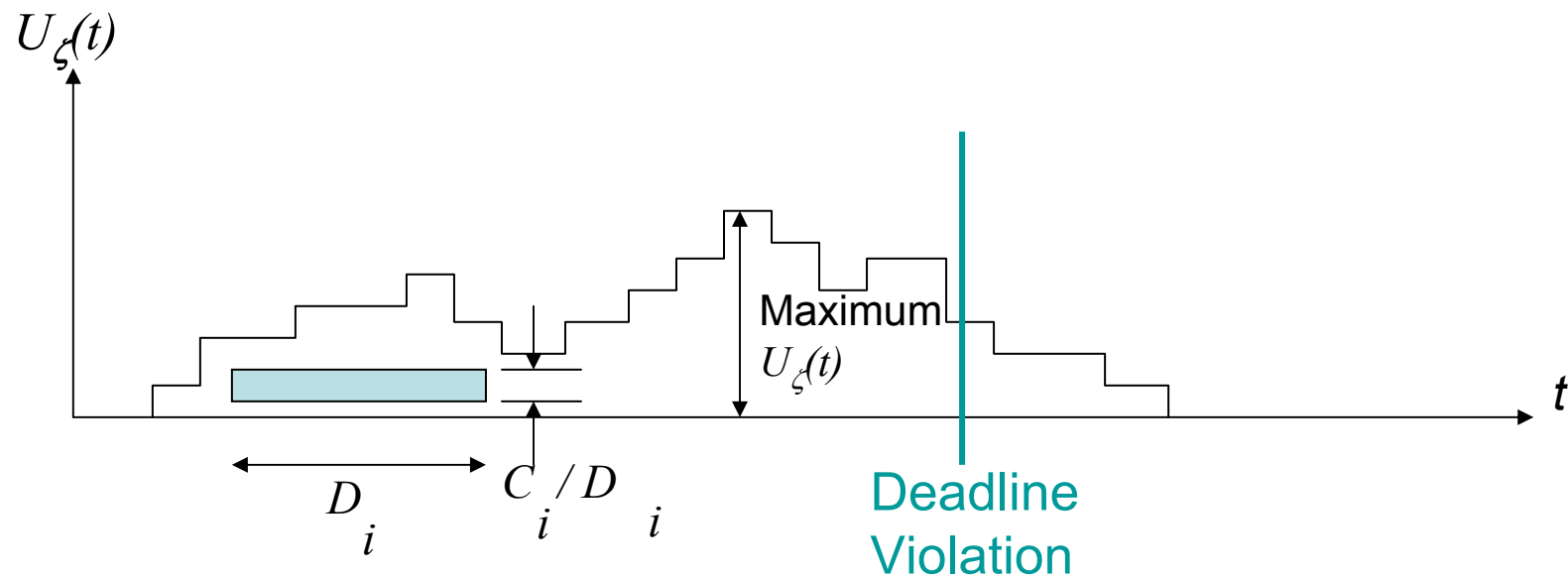
- **Minimize**, over all arrival patterns  $\zeta$ , the maximum  $U_\zeta(t)$  that precedes a deadline violation

$$U_\zeta(t) = \sum_i C_i/D_i$$



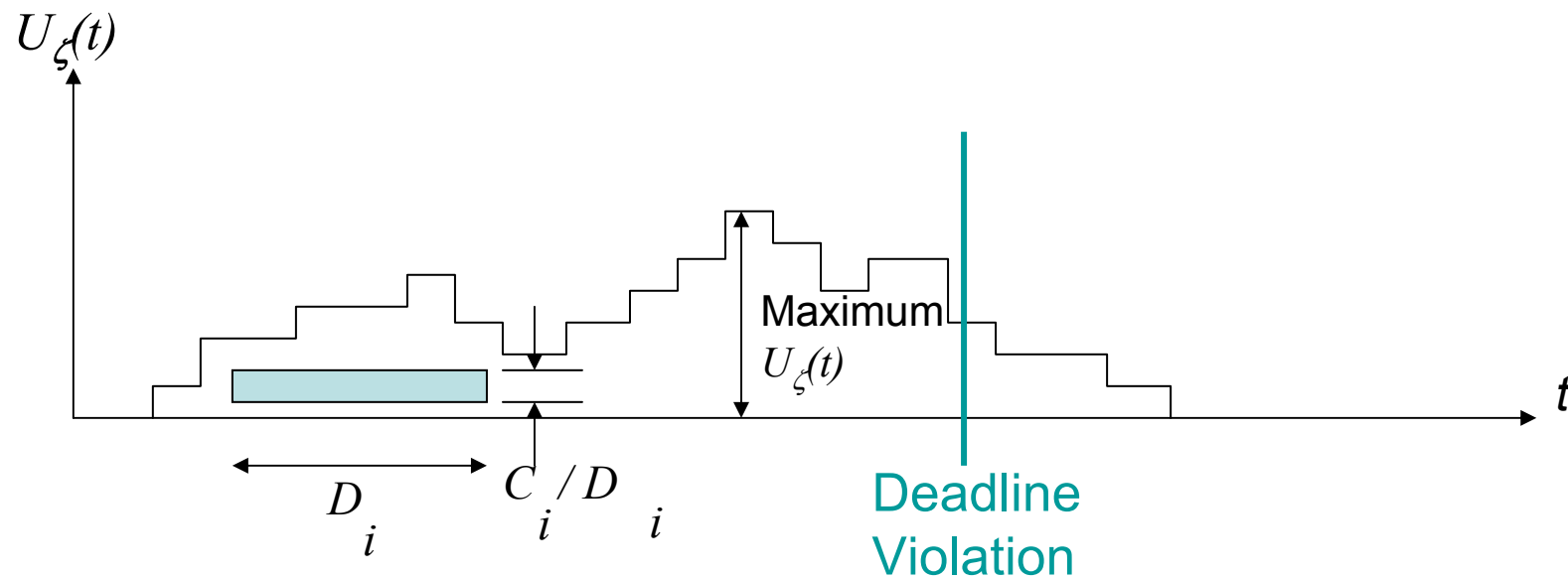
# Quick-and-Dirty Derivation

- Observe that each task  $i$  contributes  $C_i$  to the area under the  $U_\zeta(t)$  curve – see figure below.



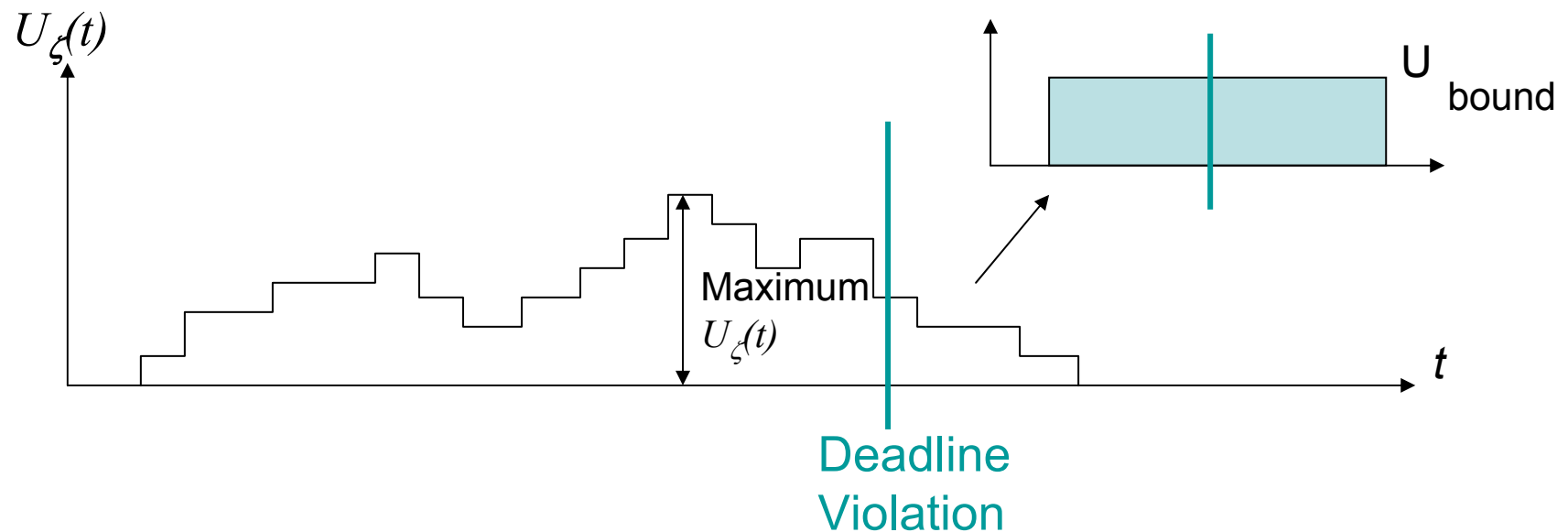
# Corollary

- The total area under the  $U_{\zeta}(t)$  curve is  $\Sigma C_i$  carried over all arrived tasks



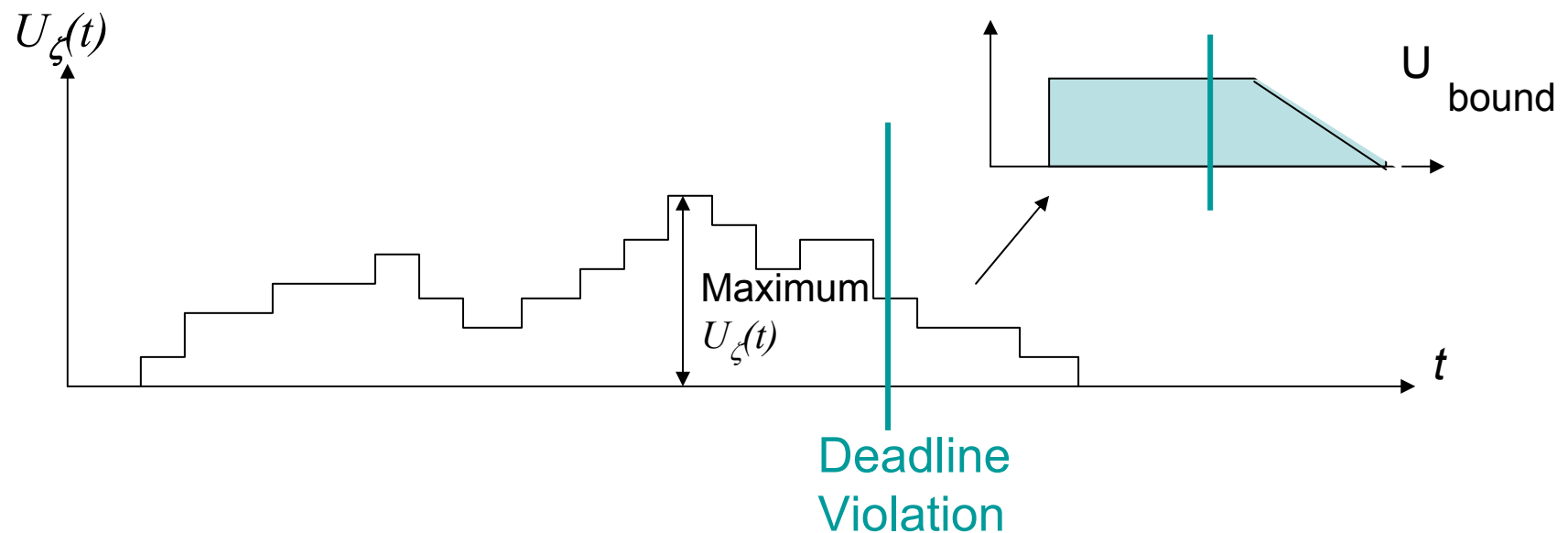
# A Geometric Interpretation

- **Minimize**, the sum  $\Sigma C_i$  across all unschedulable patterns. Say minimum is  $C_{\min}$
- Minimize curve height while area =  $C_{\min}$



# A Geometric Interpretation

- **Minimize**, the sum  $\sum C_i$  across all unschedulable patterns. Say minimum is  $C_{\min}$
- Minimize curve height while area =  $C_{\min}$



## Main Result

- A set of aperiodic tasks is schedulable using an optimal fixed-priority policy if:

$$U(t) \leq \frac{1}{1 + \sqrt{\frac{1}{2}}}$$

## Main Result

- A set of aperiodic tasks is schedulable using an optimal fixed-priority policy if:

$$U(t) \leq \frac{1}{1 + \sqrt{\frac{1}{2}}}$$

58.6%

A red arrow originates from the text '58.6%' and points towards the denominator of the fraction in the equation above, specifically to the term  $\sqrt{\frac{1}{2}}$ .

## More Main Results

- A set of  $n$  recurrent tasks is schedulable using an optimal arrival-time-independent policy if:

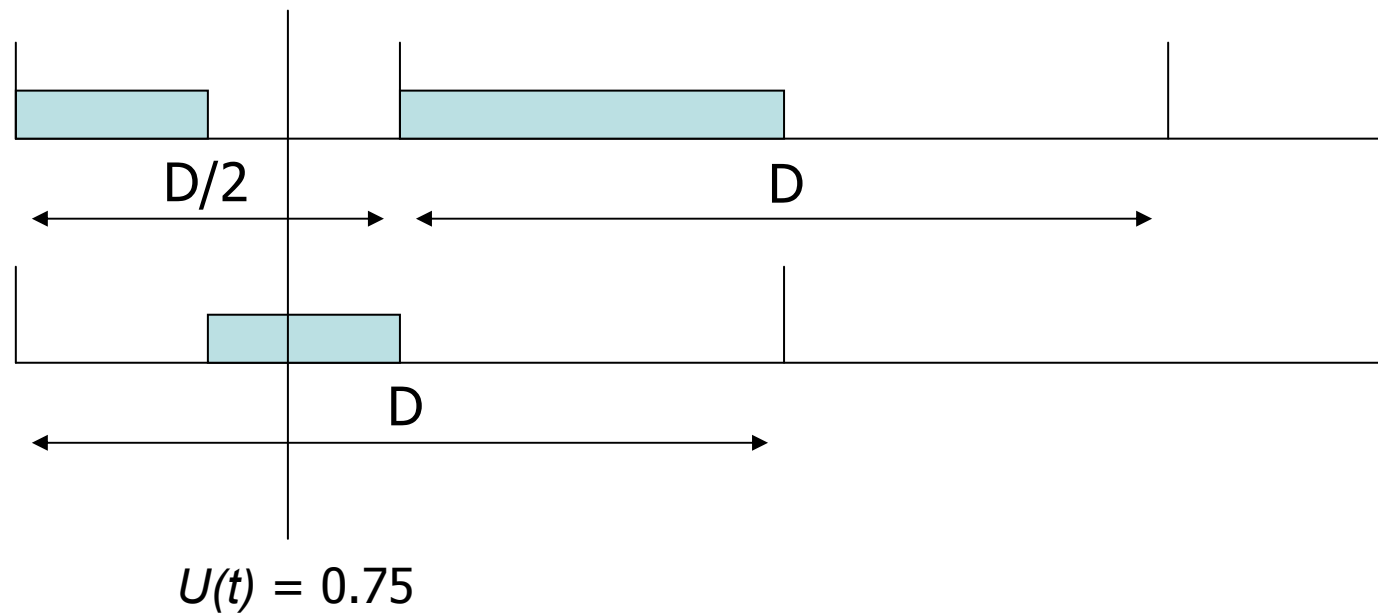
$$U(t) \leq \frac{1}{2} + \frac{1}{2n} \quad n < 3$$

$$U(t) \leq \frac{1}{1 + \sqrt{\frac{1}{2} \left(1 - \frac{1}{n-1}\right)}} \quad n \geq 3$$

- This bound is tight

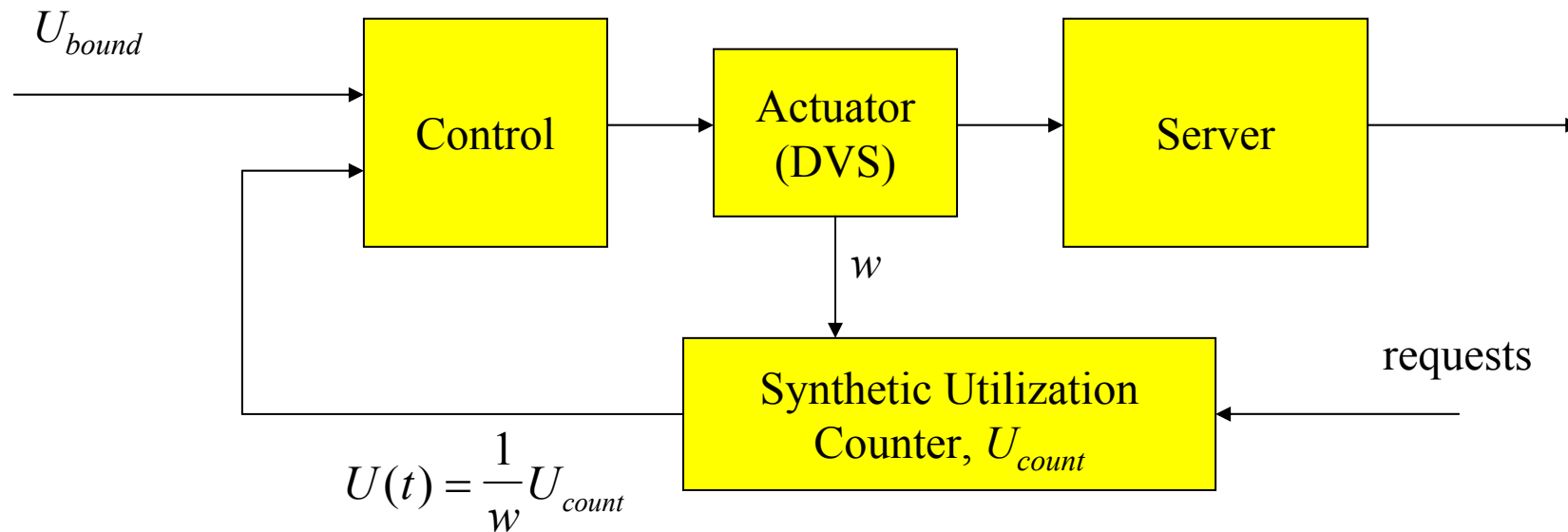
# Example of Tightness

- Consider  $n=2$



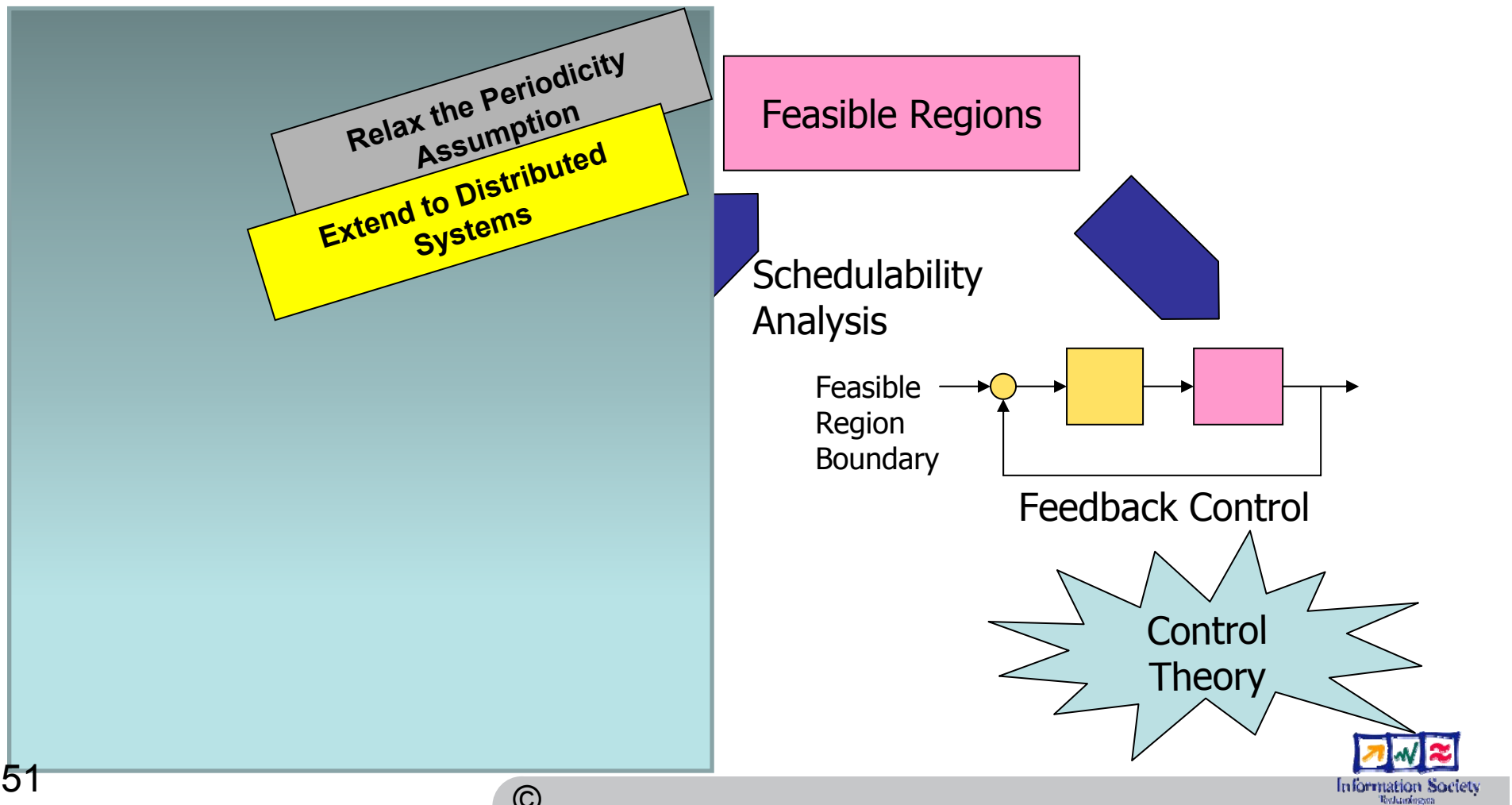
# Utilization Control

- Controller scales CPU speed,  $w$ , to the slowest value that keeps measured instantaneous utilization below the bound



# Bridging the Abstractions

Transform fine-grained performance requirements into aggregate state variables that are easy to control

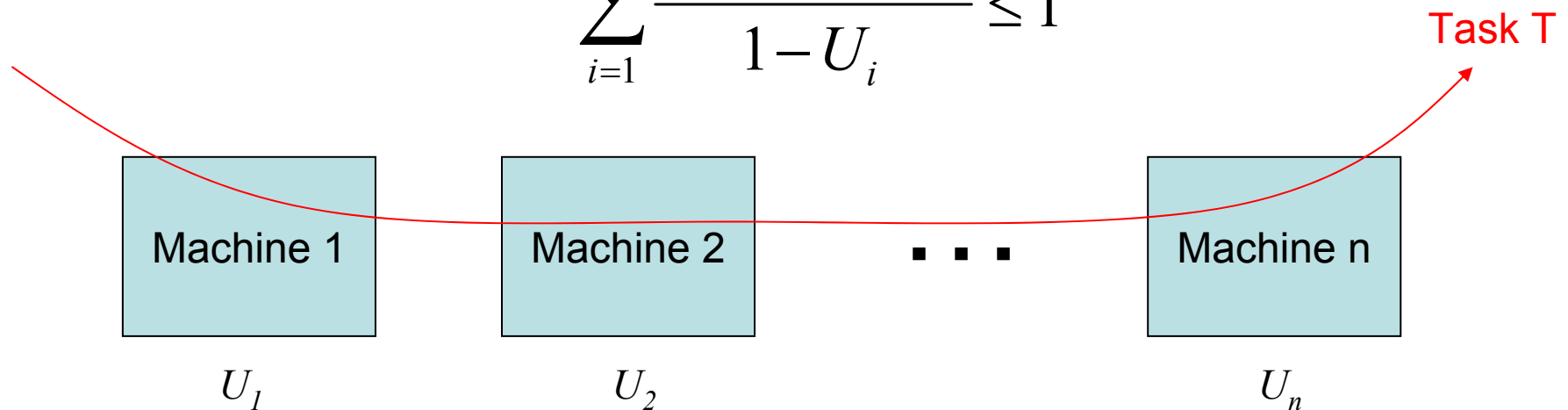




# Main Results in Schedulability of Multistage Execution

- Let  $U_1, U_2, \dots, U_n$  be the utilization values of  $n$  machines
- The end-to-end deadline of a task  $T$  traversing the  $n$ -machine path is met if:

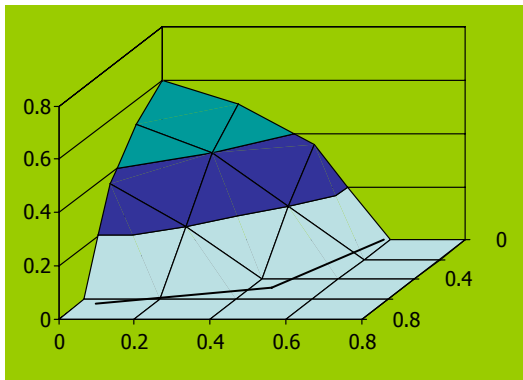
$$\sum_{i=1}^n \frac{U_i(1-U_i/2)}{1-U_i} \leq 1$$



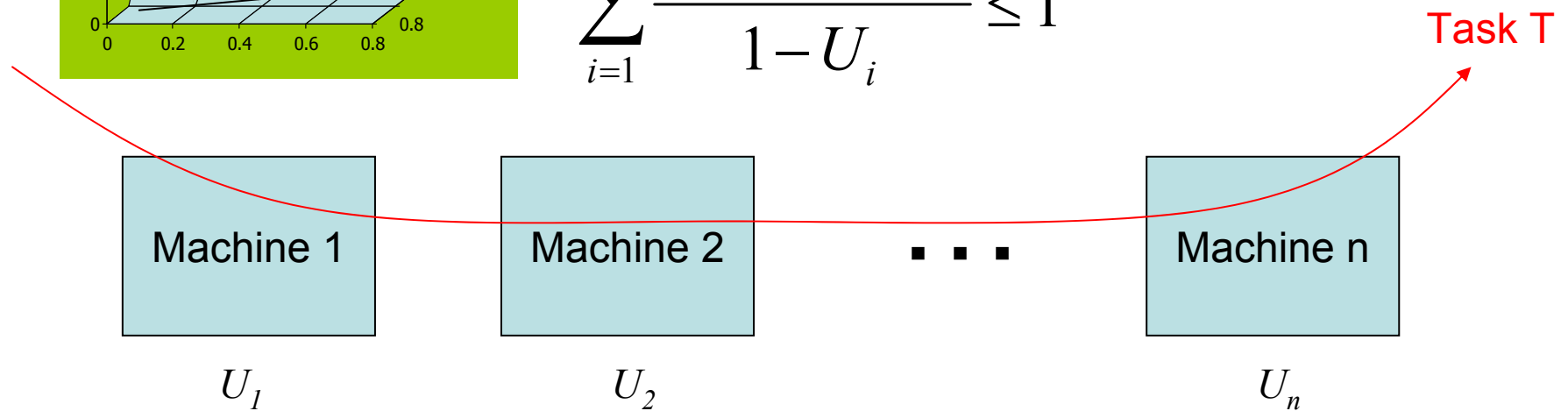


# Main Results in Schedulability of Multistage Execution

- A Multidimensional utilization bound



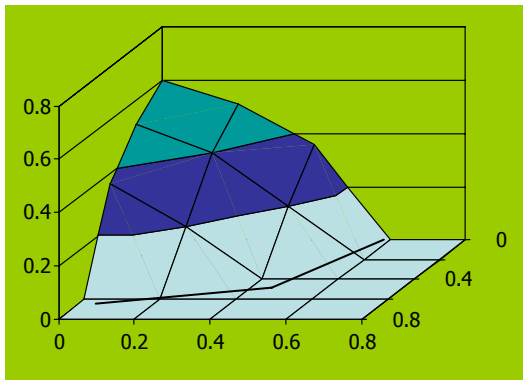
$$\sum_{i=1}^n \frac{U_i(1-U_i/2)}{1-U_i} \leq 1$$





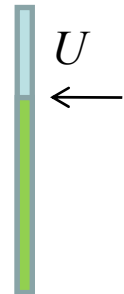
# Main Results in Schedulability of Multistage Execution

- A Multidimensional utilization bound

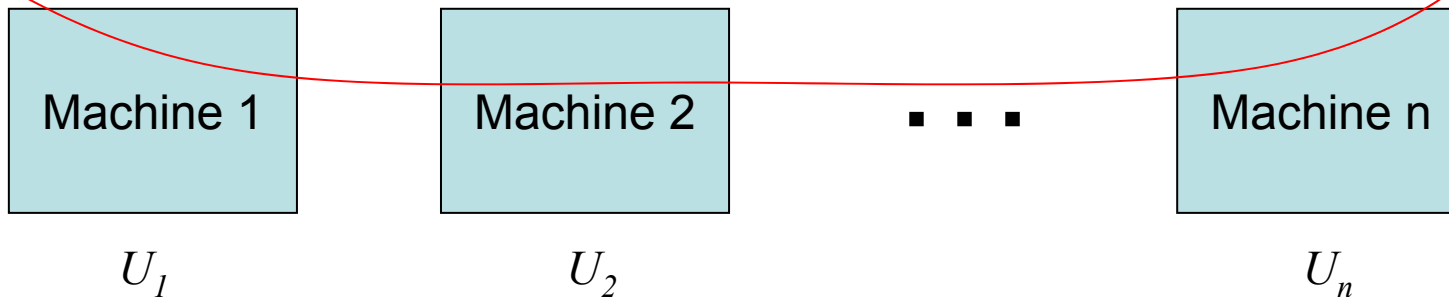


$$\sum_{i=1}^n \frac{U_i(1 - U_i/2)}{1 - U_i} \leq 1$$

Reduces to  $U(t) \leq \frac{1}{1 + \sqrt{\frac{1}{2}}}$  for a single stage



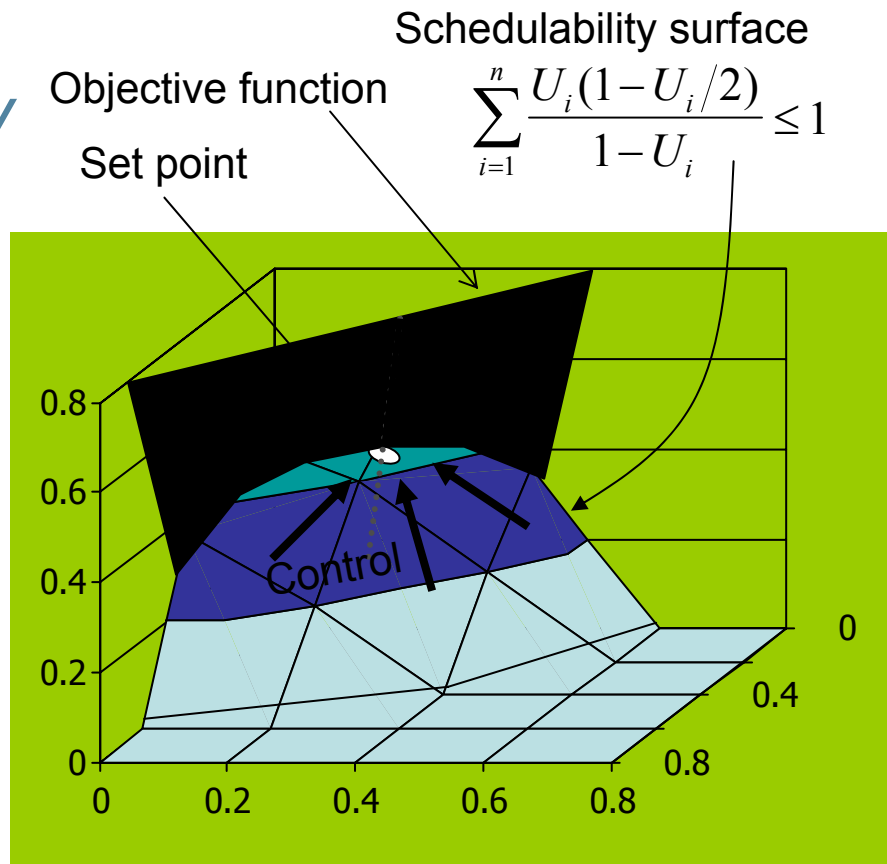
Task T





# Optimization Subject to Schedulability Constraints

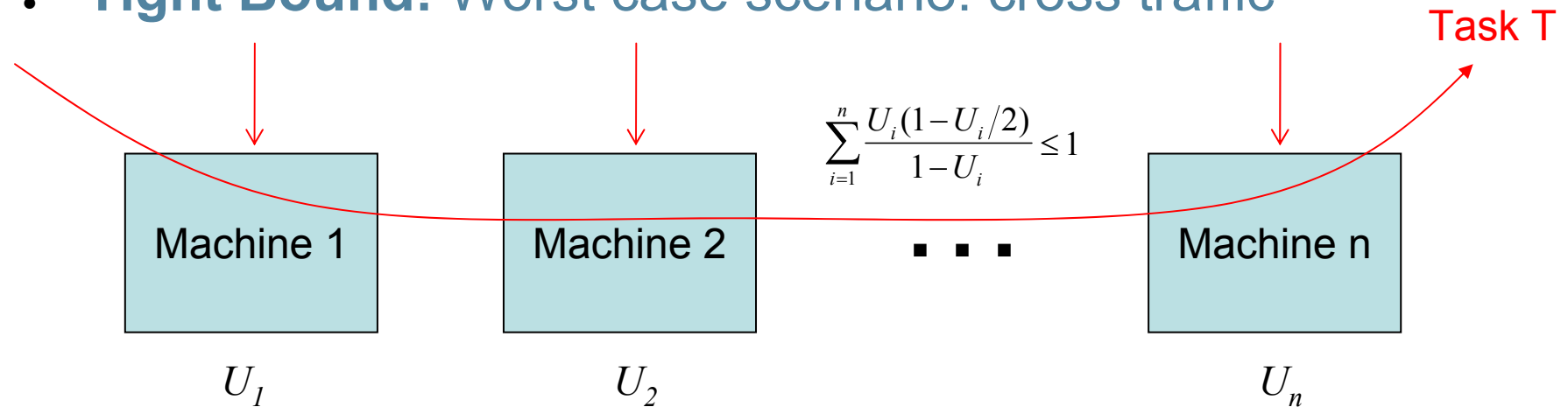
- *Intersect the schedulability surface of the system with objective function*
- *Find optimal point on intersection curve*
- *Use run-time feedback control to measure and correct performance dynamically to reach optimal.*





# Main Results in Schedulability of Multistage Execution

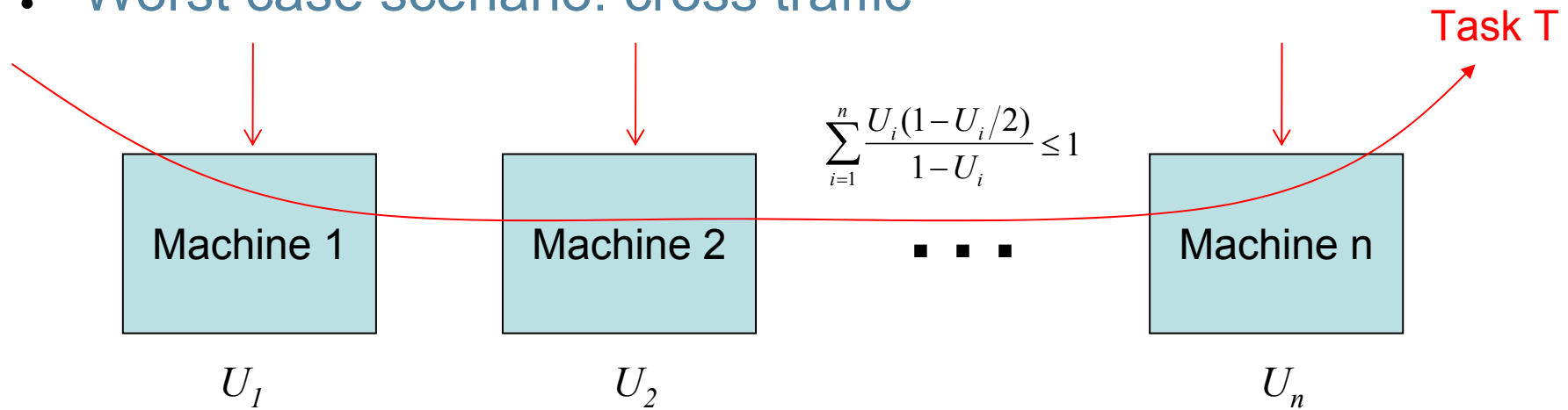
- **Tight Bound!** Worst case scenario: cross traffic



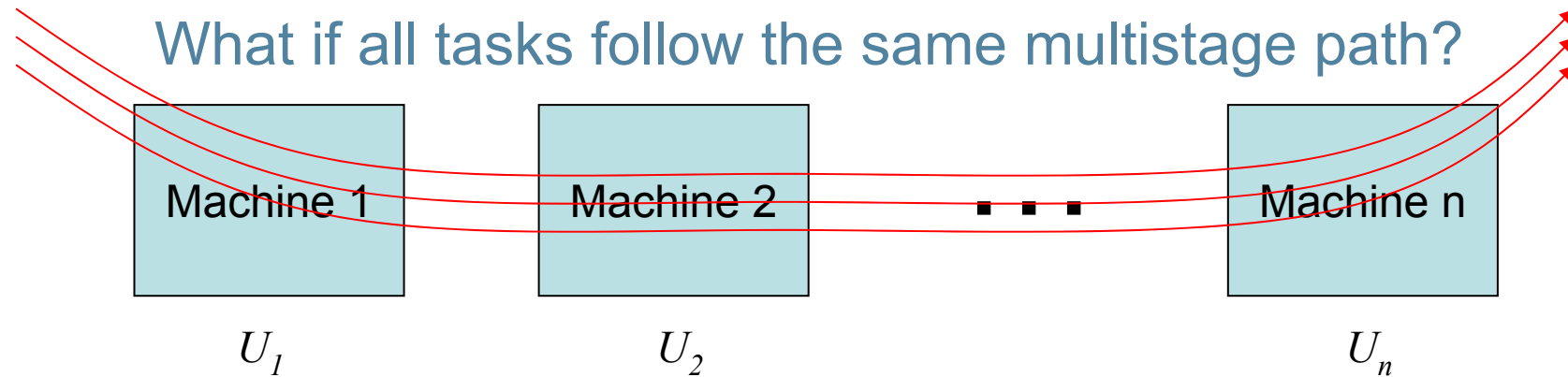


# Main Results in Schedulability of Multistage Execution

- Worst case scenario: cross traffic

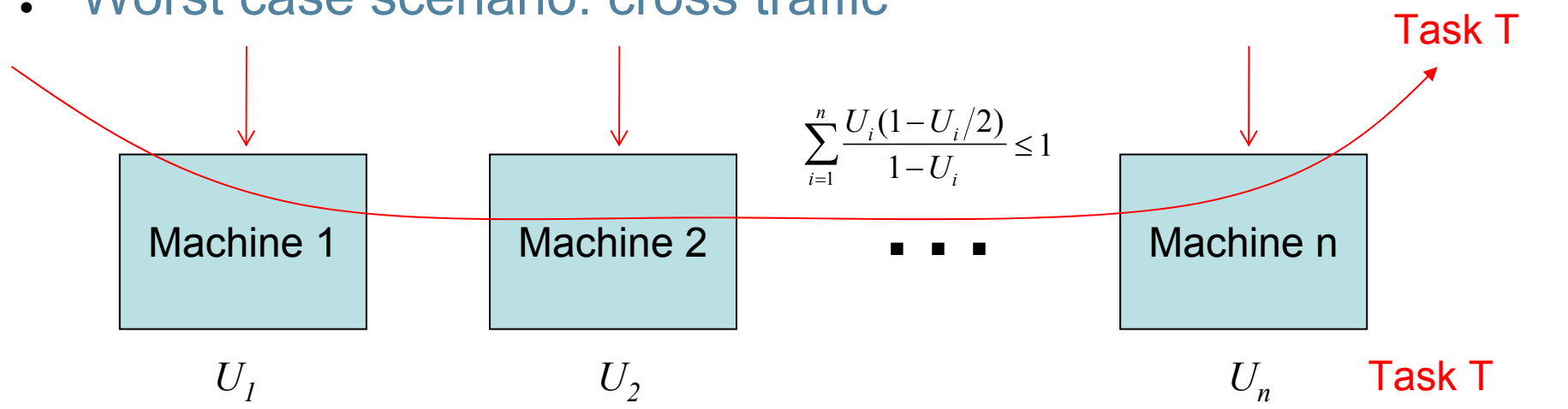


What if all tasks follow the same multistage path?



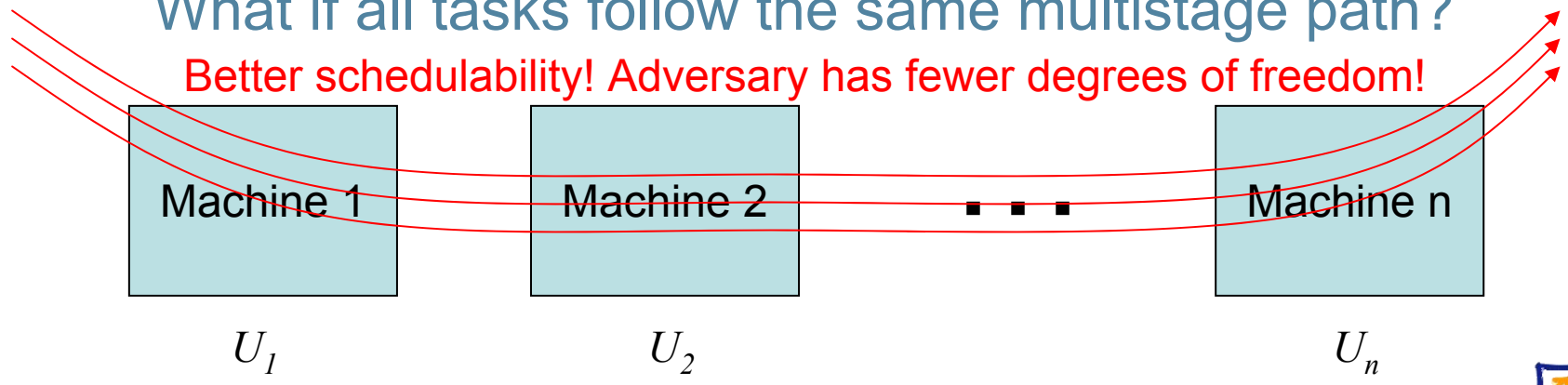
# Main Results in Schedulability of Multistage Execution

- Worst case scenario: cross traffic



What if all tasks follow the same multistage path?

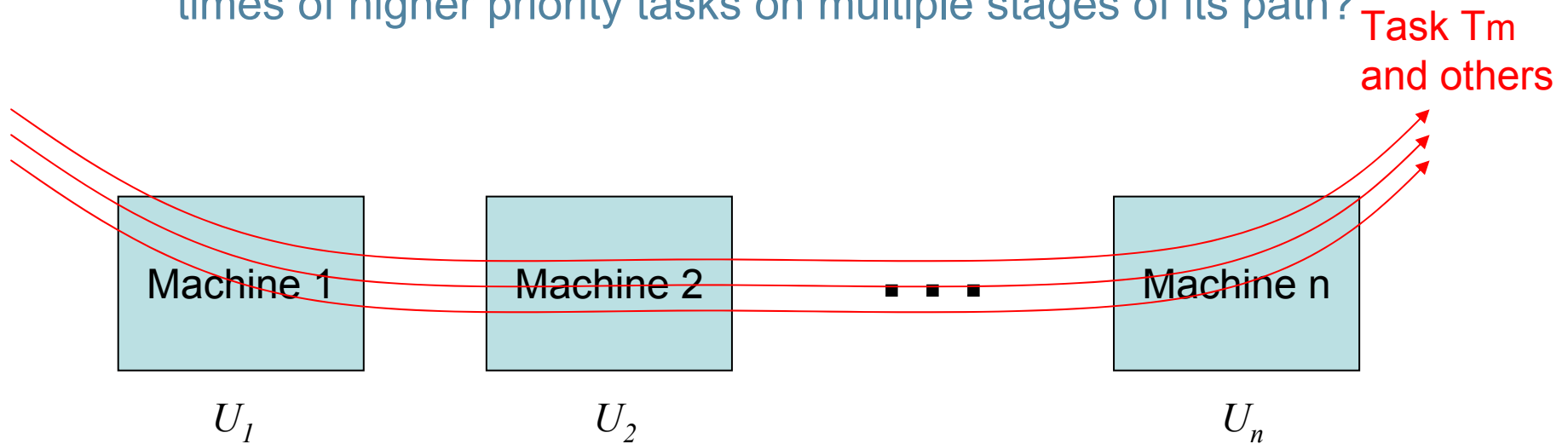
Better schedulability! Adversary has fewer degrees of freedom!





# Main Results in Schedulability of Multistage Execution

- A fundamental question:
  - How does delay of a low priority task  $T_m$  depend on execution times of higher priority tasks on multiple stages of its path?





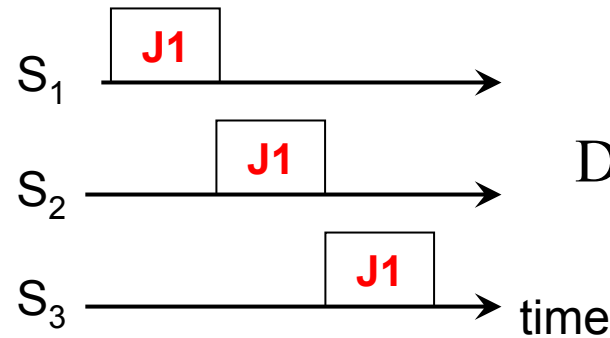
# Delay Composition in Pipelined Execution

Many jobs, one stage



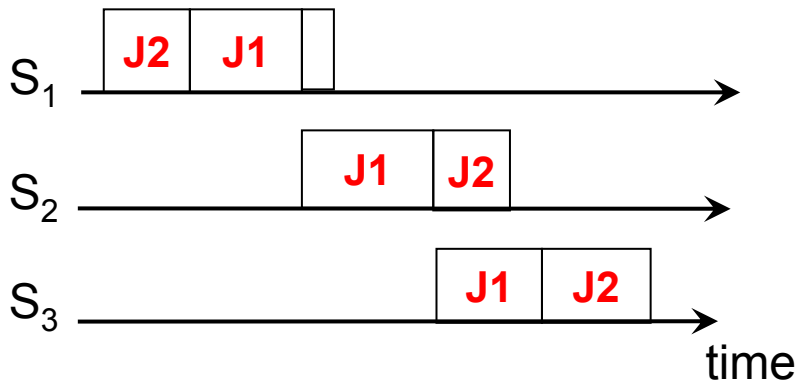
$$\text{Delay} = \sum_{i \in \text{jobs}} C_{i,1}$$

Many stages, one job



$$\text{Delay} = \sum_{j \in \text{stages}} C_{1,j}$$

Many jobs, many stages

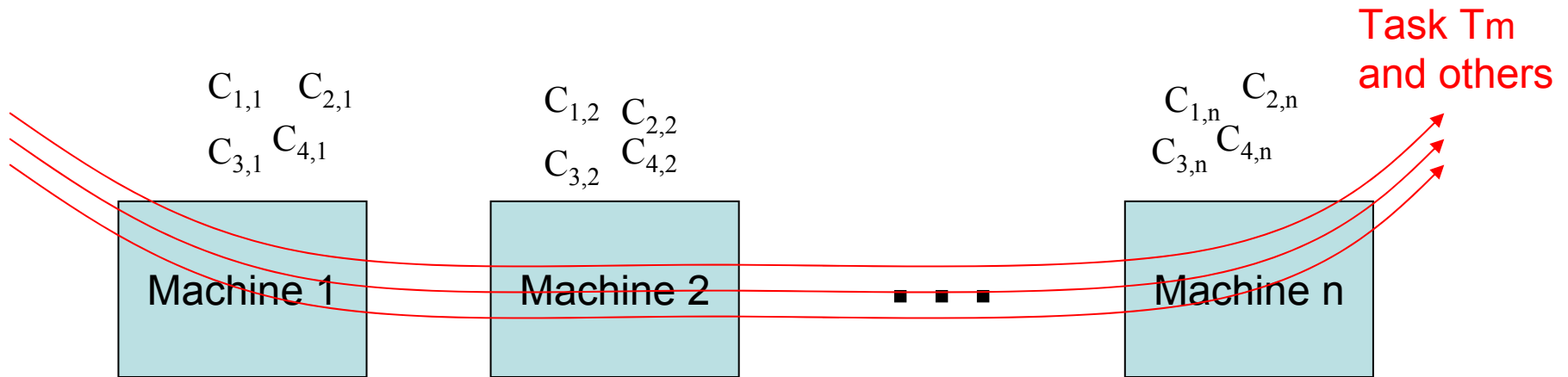


$$\text{Is Delay} = \sum_{i \in \text{jobs}} \sum_{j \in \text{stages}} C_{i,j} ?$$

$$\text{Is Delay} < \sum_{i \in \text{jobs}} \sum_{j \in \text{stages}} C_{i,j} ?$$



# Delay Composition in Pipelined Execution

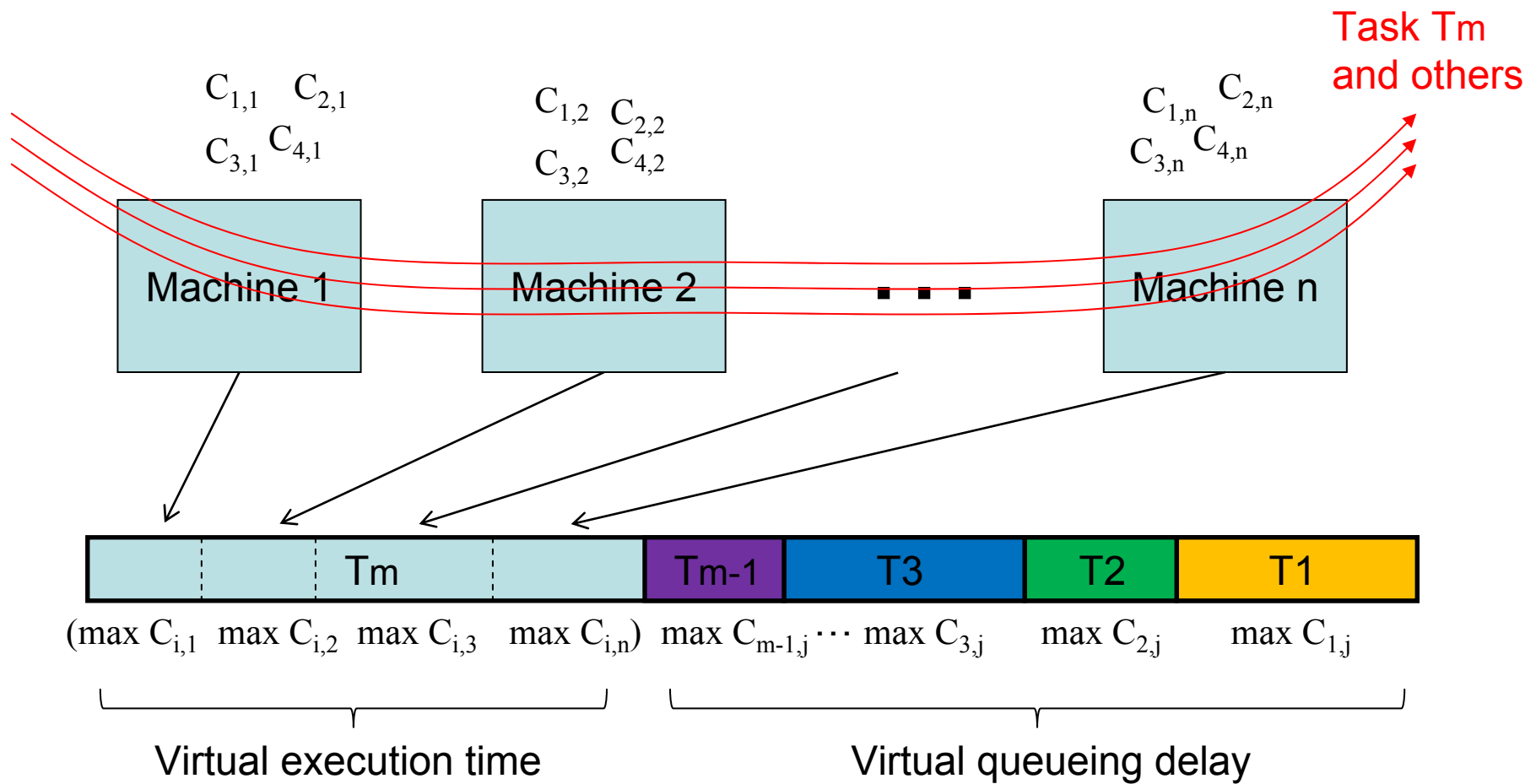


$$\text{Delay} \leq \sum_{\text{allstages}} \max_{\substack{\text{higher} \\ \text{priority} \\ \text{jobs}}} C_{i,j} + \sum_{\text{higher} \\ \text{priority} \\ \text{jobs}} \max_{\text{allstages}} C_{i,j}$$

$C_{1,1}$	$C_{2,1}$	$C_{3,1}$	$C_{4,1}$
$C_{1,2}$	$C_{2,2}$	$C_{3,2}$	$C_{4,2}$
$C_{1,3}$	$C_{2,3}$	$C_{3,3}$	$C_{4,3}$
$C_{1,4}$	$C_{2,4}$	$C_{3,4}$	$C_{4,4}$



# Delay Composition in Pipelined Execution



Task  $T_m$   
and others

$(\max C_{i,1} \max C_{i,2} \max C_{i,3} \max C_{i,n})$ 
 $\max C_{m-1,j}$ 
 $\dots$ 
 $\max C_{3,j}$ 
 $\max C_{2,j}$ 
 $\max C_{1,j}$

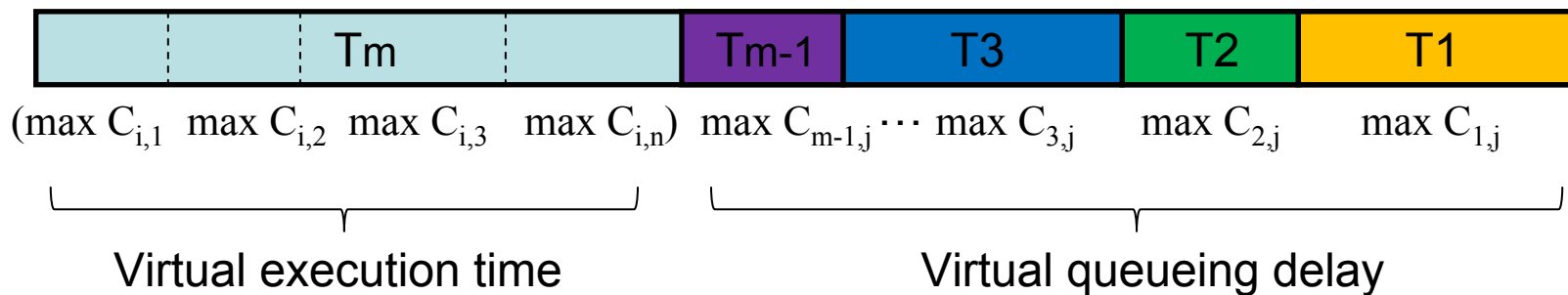
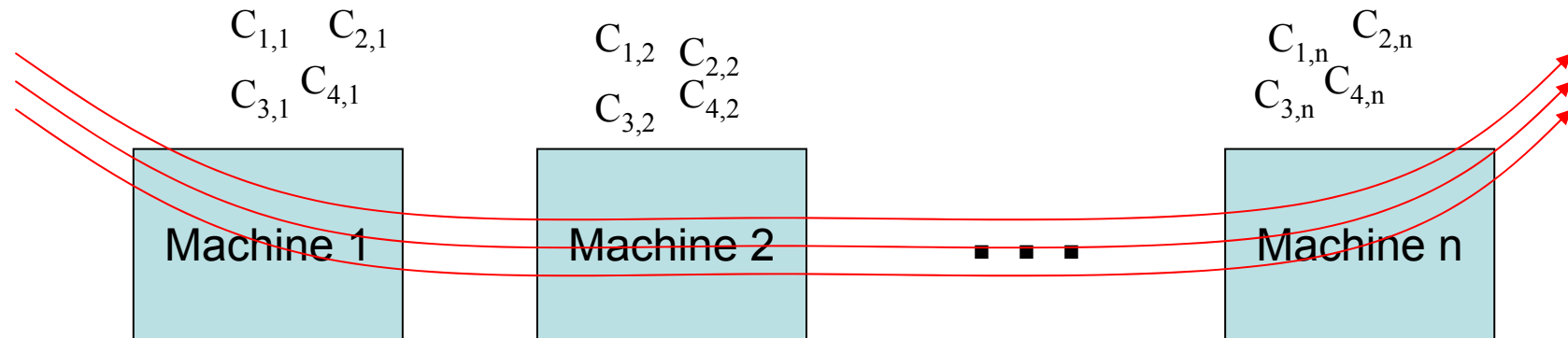
Virtual execution time

Virtual queuing delay

$$\text{Delay} \leq \sum_{\text{allstages}} \max_{\text{jobs}} C_{i,j} + \sum_{\text{jobs}} \max_{\text{allstages}} C_{i,j}$$

# Properties

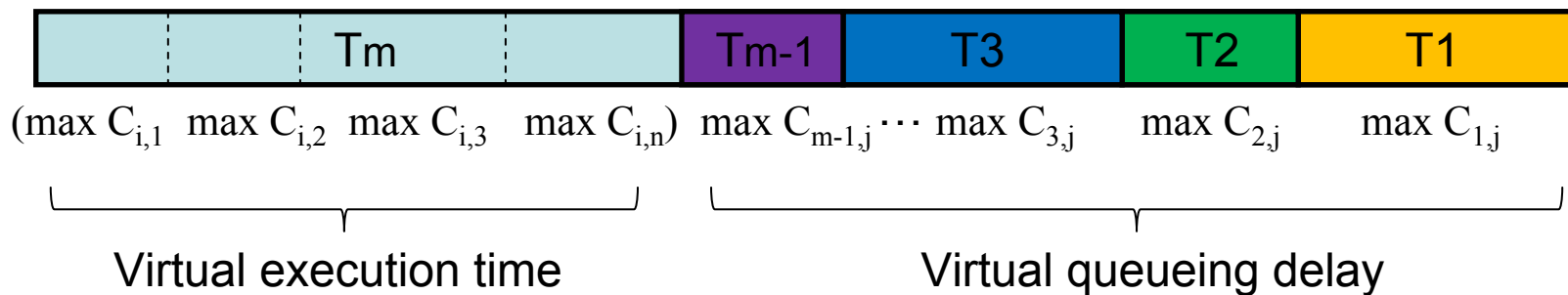
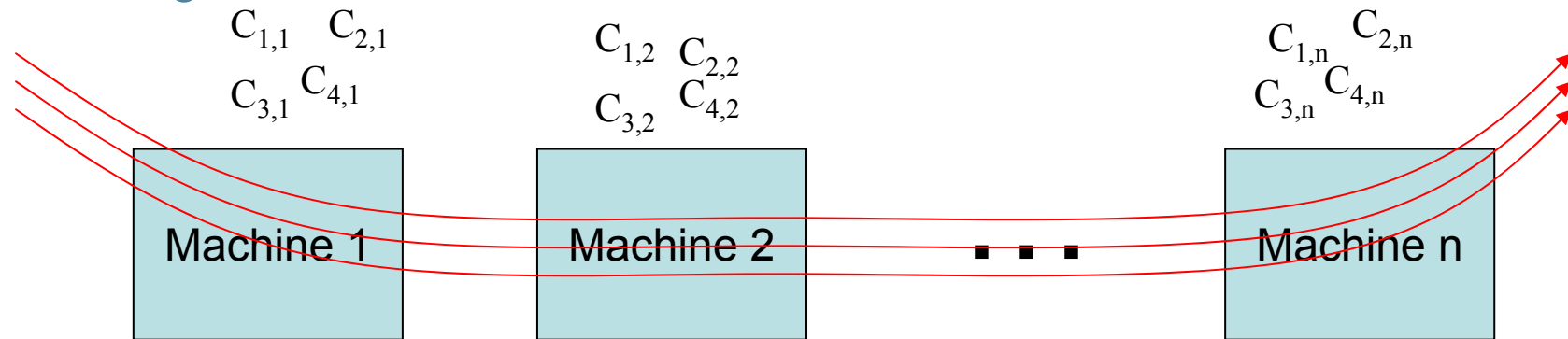
- Results applicable to both periodic and aperiodic tasks
- Results applicable under any priority-based scheduling policy where prioritization is consistent across stages



$$\text{Delay} \leq \sum_{\text{allstages}} \max_{\text{jobs}} C_{i,j} + \sum_{\text{jobs}} \max_{\text{allstages}} C_{i,j}$$

# General Observation

- For each shared segment of the path that a higher priority task shares with a lower priority one in a distributed system, the former delays the latter by at most its max computation time over the segment.

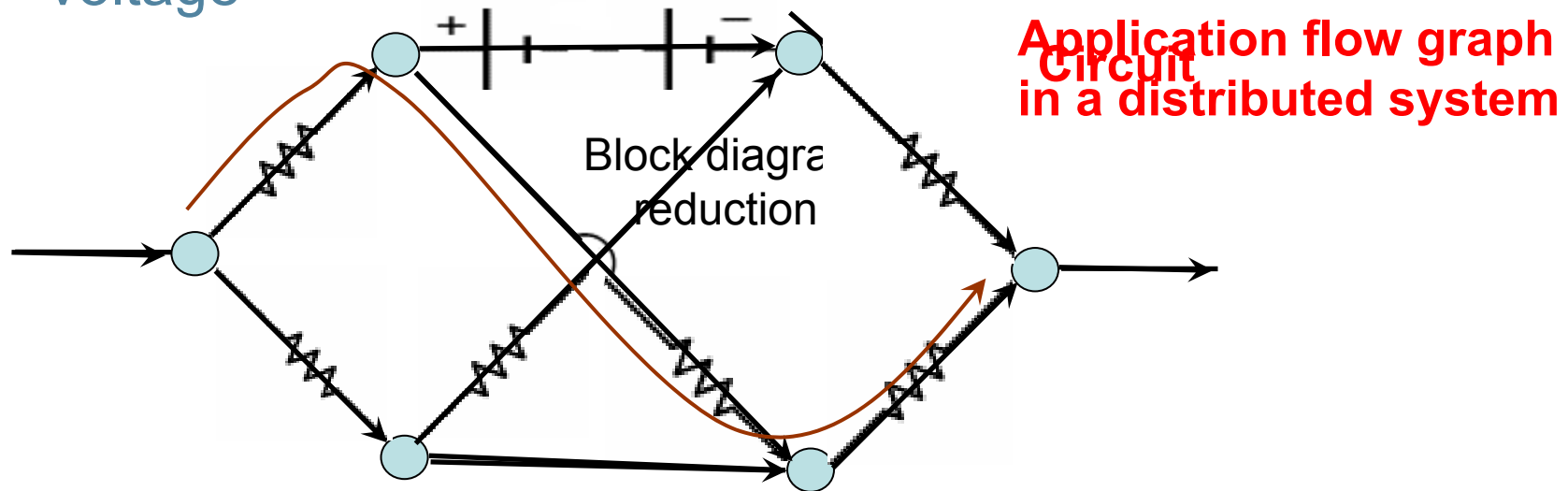


$$\text{Delay} \leq \sum_{\text{allstages}} \max_{\text{jobs}} C_{i,j} + \sum_{\text{jobs}} \max_{\text{allstages}} C_{i,j}$$



# A Reduction-based Approach to Distributed System Analysis?

- Control theory
- Circuit theory – Kirchoff's laws - analyze current and voltage



- Reduce schedulability problem on distributed system to problem on uniprocessor?

## A Recent Result: Delay Composition Algebra

- A compositional algebraic framework to analyze timing issues in distributed systems
  - Operands represent workload on composed sub-systems
  - Set of operators to systematically transform distributed real-time task systems to uniprocessor task-systems
  - Traditional uniprocessor analysis can be applied to infer schedulability of distributed tasks
- Less pessimistic than traditional analysis with increasing system scale

## Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

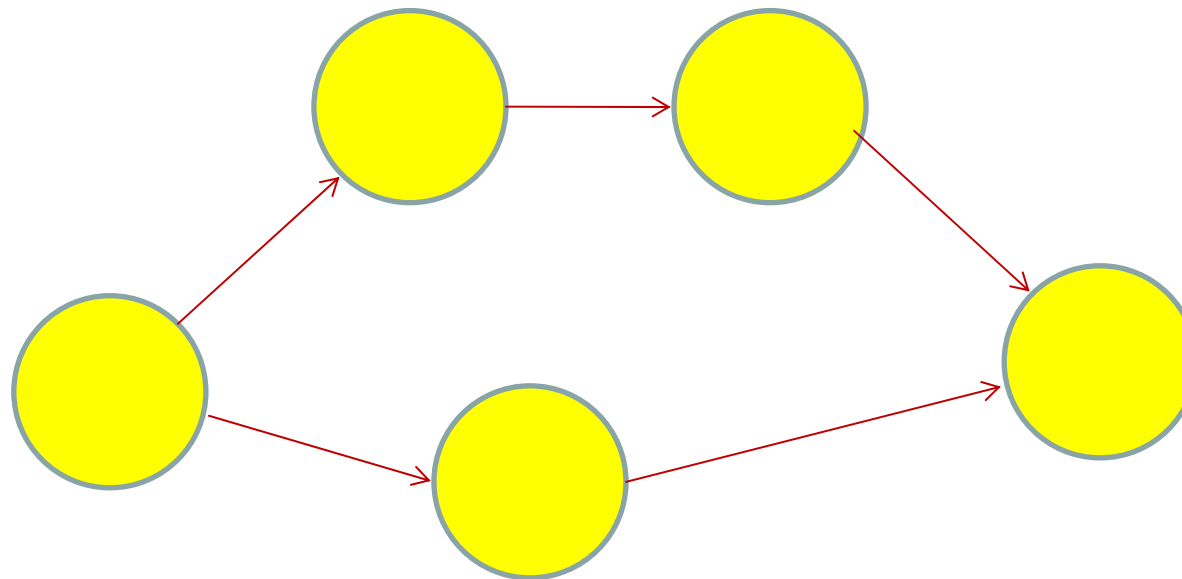
- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

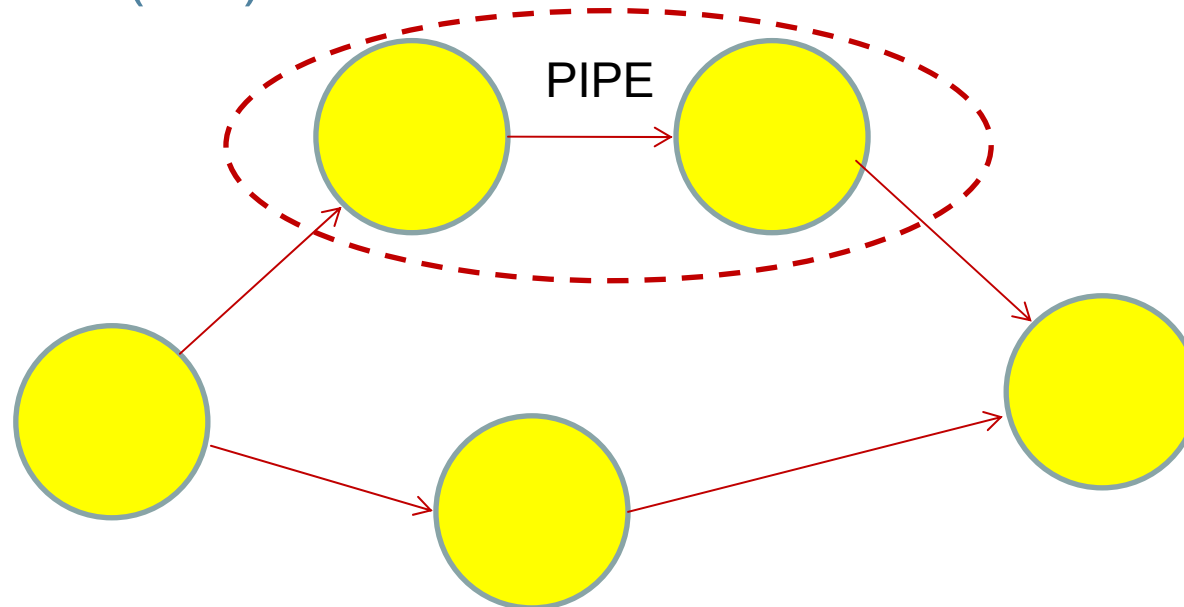


# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

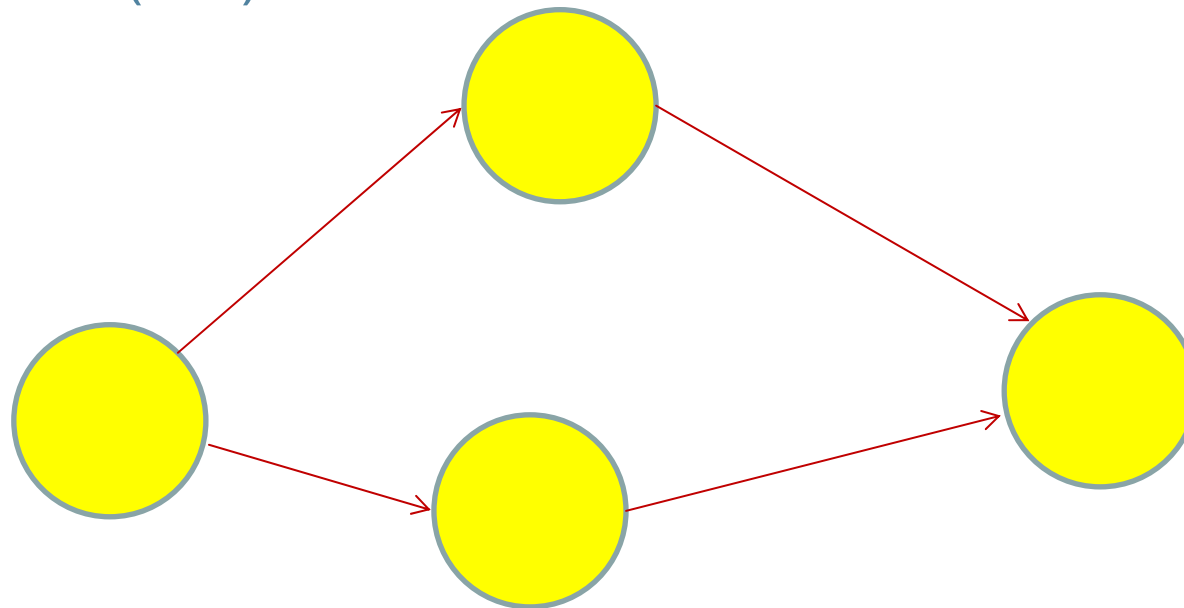


# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

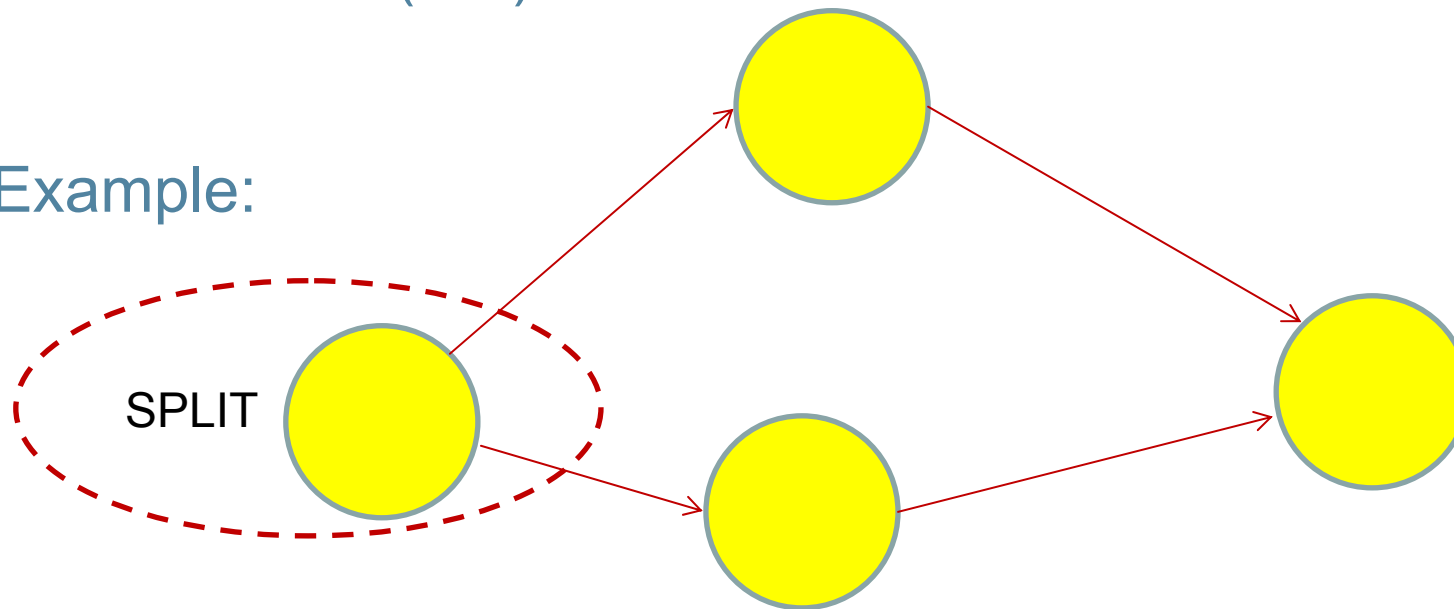


# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

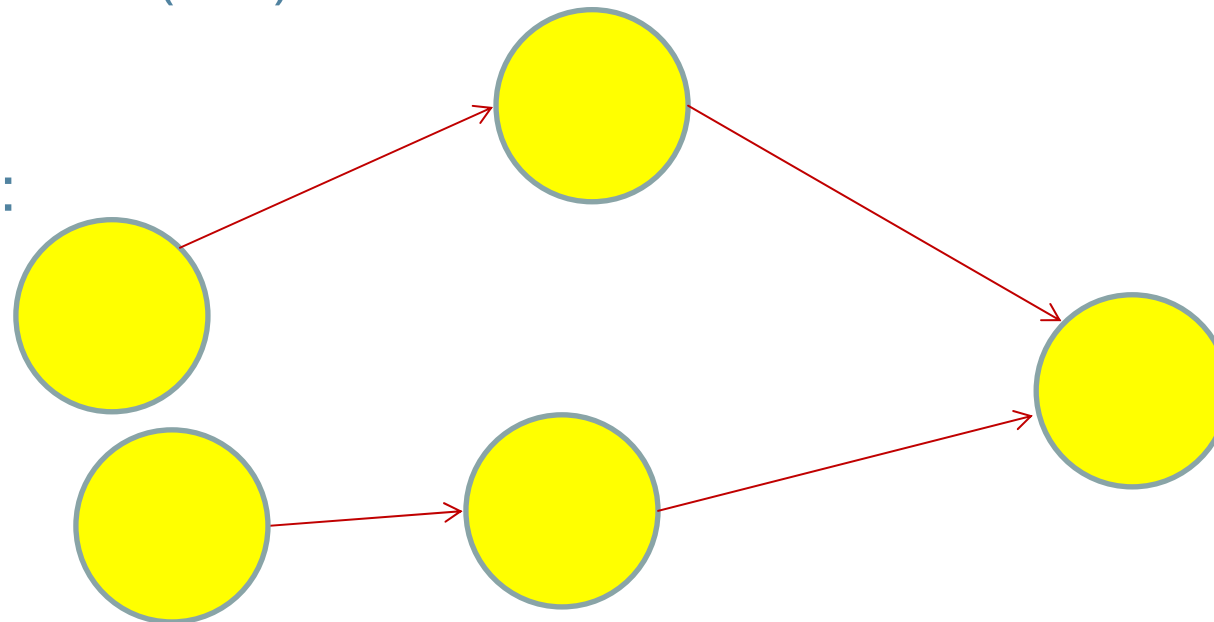


# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

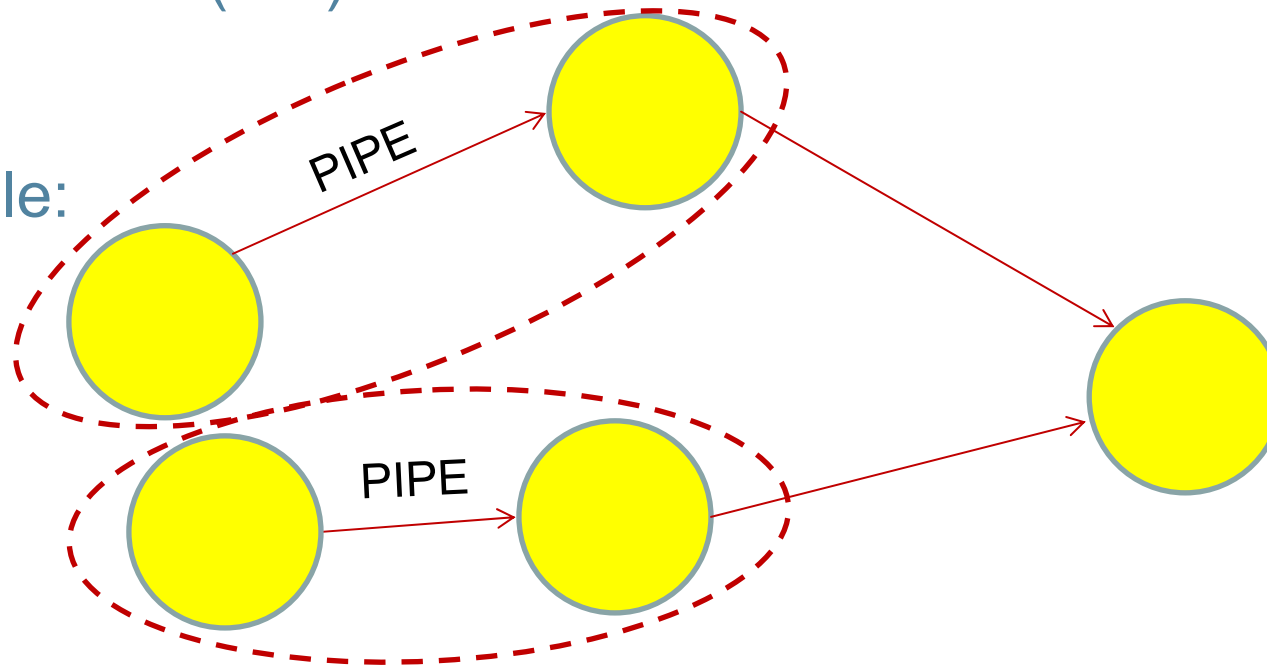


# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

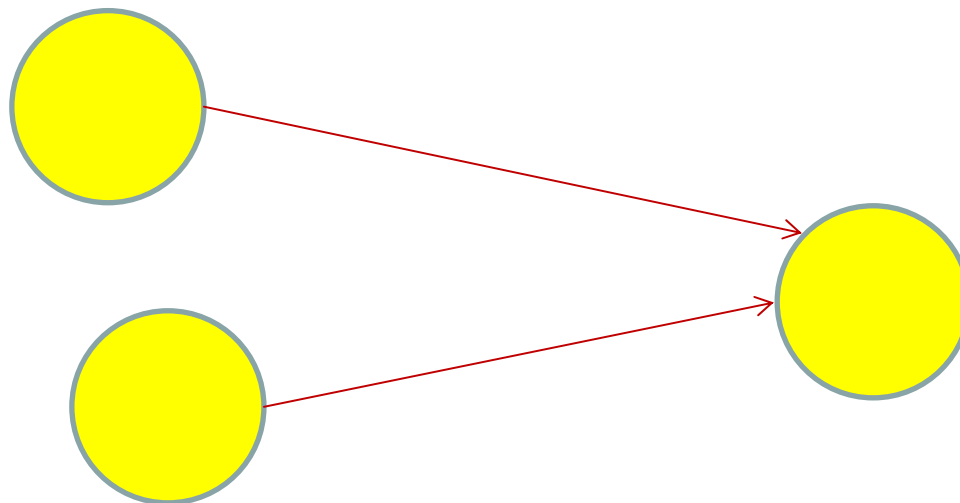


# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

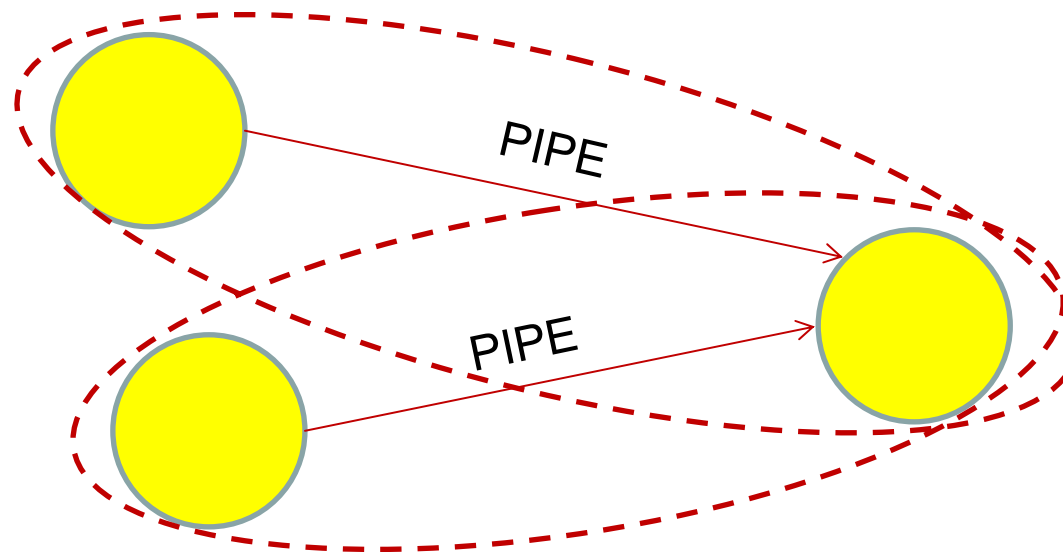


# Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:

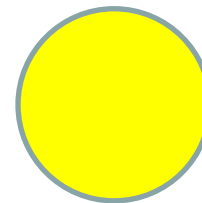


## Delay Composition Algebra

Two operators that apply to node workloads while simplifying the resource graph

- $W = \mathbf{PIPE} (W1, W2)$
- $W = \mathbf{SPLIT} (W1)$

Example:



# The Workload Matrix

- Each cell  $i,j$  holds the delay that job  $i$  imposes on job  $j$  in the subsystem that the matrix represents.
- Example: System consisting of four jobs ( $J_1$ - $J_4$ ) of which only  $J_1$ ,  $J_2$ , and  $J_4$  execute on the stage

	$J_1$	$J_2$	$J_3$	$J_4$
$J_1$	$(C_{1,j}, 0)$	$(C_{1,j}, 0)$	$(0, 0)$	$(C_{1,j}, 0)$
$J_2$	$(0, 0)$	$(C_{2,j}, 0)$	$(0, 0)$	$(C_{2,j}, 0)$
$J_3$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
$J_4$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(C_{4,j}, 0)$
.....				
	$C_{1,j}$	$\max(C_{1,j}, C_{2,j})$	0	$\max(C_{1,j}, C_{2,j}, C_{4,j})$



# The Operators

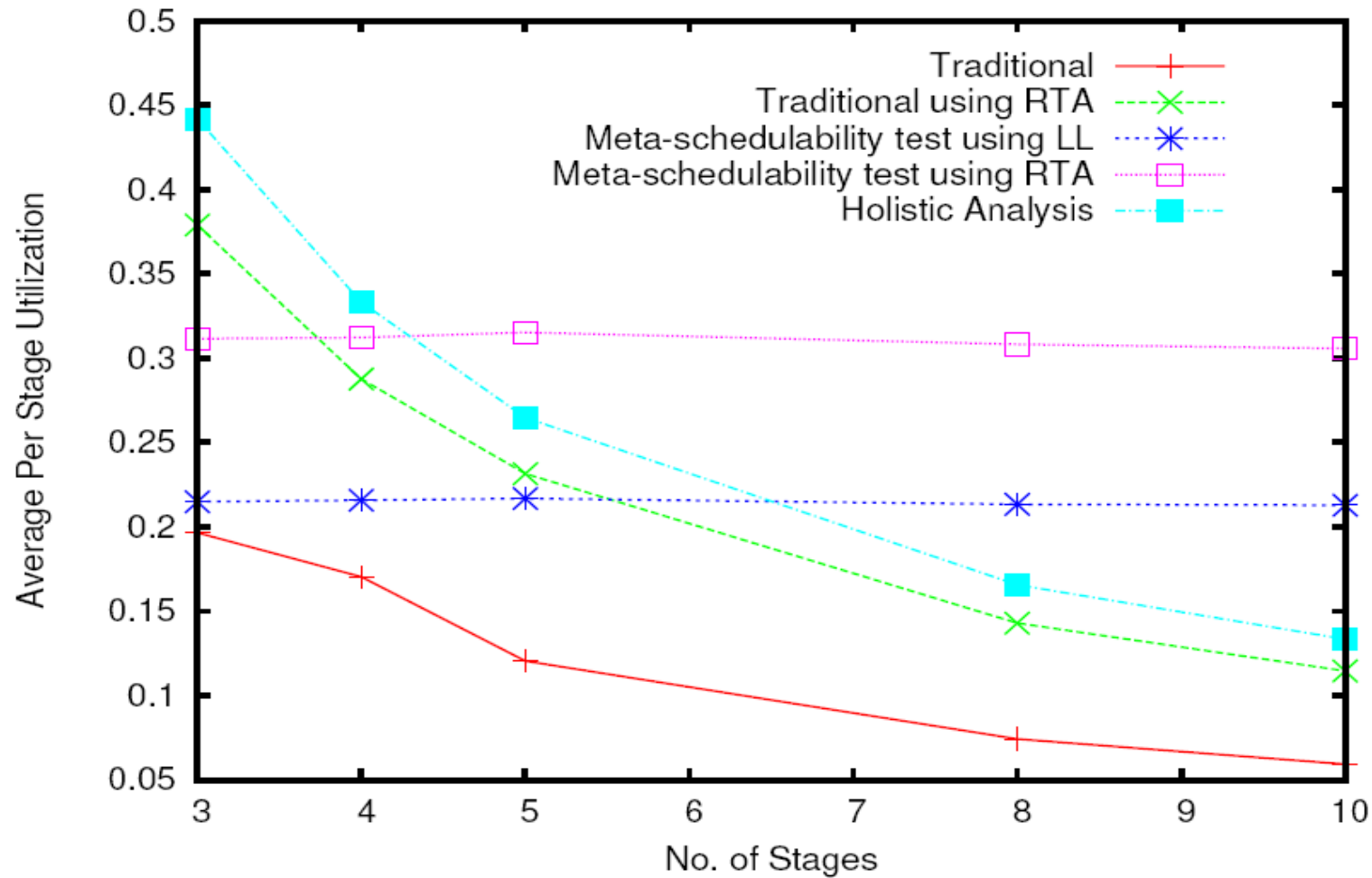
$$\left( \begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (q_{1,1}^A, r_{1,1}^A) & (q_{1,2}^A, r_{1,2}^A) \\ J_2 & (0, 0) & (q_{2,2}^A, r_{2,2}^A) \\ \dots & \dots & \dots \\ & s_1^A & s_2^A \end{array} \right) \text{PIPE} \left( \begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (q_{1,1}^B, r_{1,1}^B) & (q_{1,2}^B, r_{1,2}^B) \\ J_2 & (0, 0) & (q_{2,2}^B, r_{2,2}^B) \\ \dots & \dots & \dots \\ & s_1^B & s_2^B \end{array} \right)$$

$$= \left( \begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (\max(q_{1,1}^A, q_{1,1}^B), \max(r_{1,1}^A, r_{1,1}^B)) & (\max(q_{1,2}^A, q_{1,2}^B), \max(r_{1,2}^A, r_{1,2}^B)) \\ J_2 & (0, 0) & (\max(q_{2,2}^A, q_{2,2}^B), \max(r_{2,2}^A, r_{2,2}^B)) \\ \dots & \dots & \dots \\ & s_1^A + s_1^B & s_2^A + s_2^B \end{array} \right)$$

$$\text{SPLIT} \left( \begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (q_{1,1}^A, r_{1,1}^A) & (q_{1,2}^A, r_{1,2}^A) \\ J_2 & (0, 0) & (q_{2,2}^A, r_{2,2}^A) \\ \dots & \dots & \dots \\ & s_1^A & s_2^A \end{array} \right) \equiv \left( \begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (q_{1,1}^A, r_{1,1}^A) & (0, 0) \\ J_2 & (0, 0) & (0, 0) \\ \dots & \dots & \dots \\ & s_1^A & 0 \end{array} \right), \left( \begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (0, 0) & (0, q_{1,2}^A + r_{1,2}^A) \\ J_2 & (0, 0) & (q_{2,2}^A, r_{2,2}^A) \\ \dots & \dots & \dots \\ & 0 & s_2^A \end{array} \right)$$



# Schedulability Analysis Results

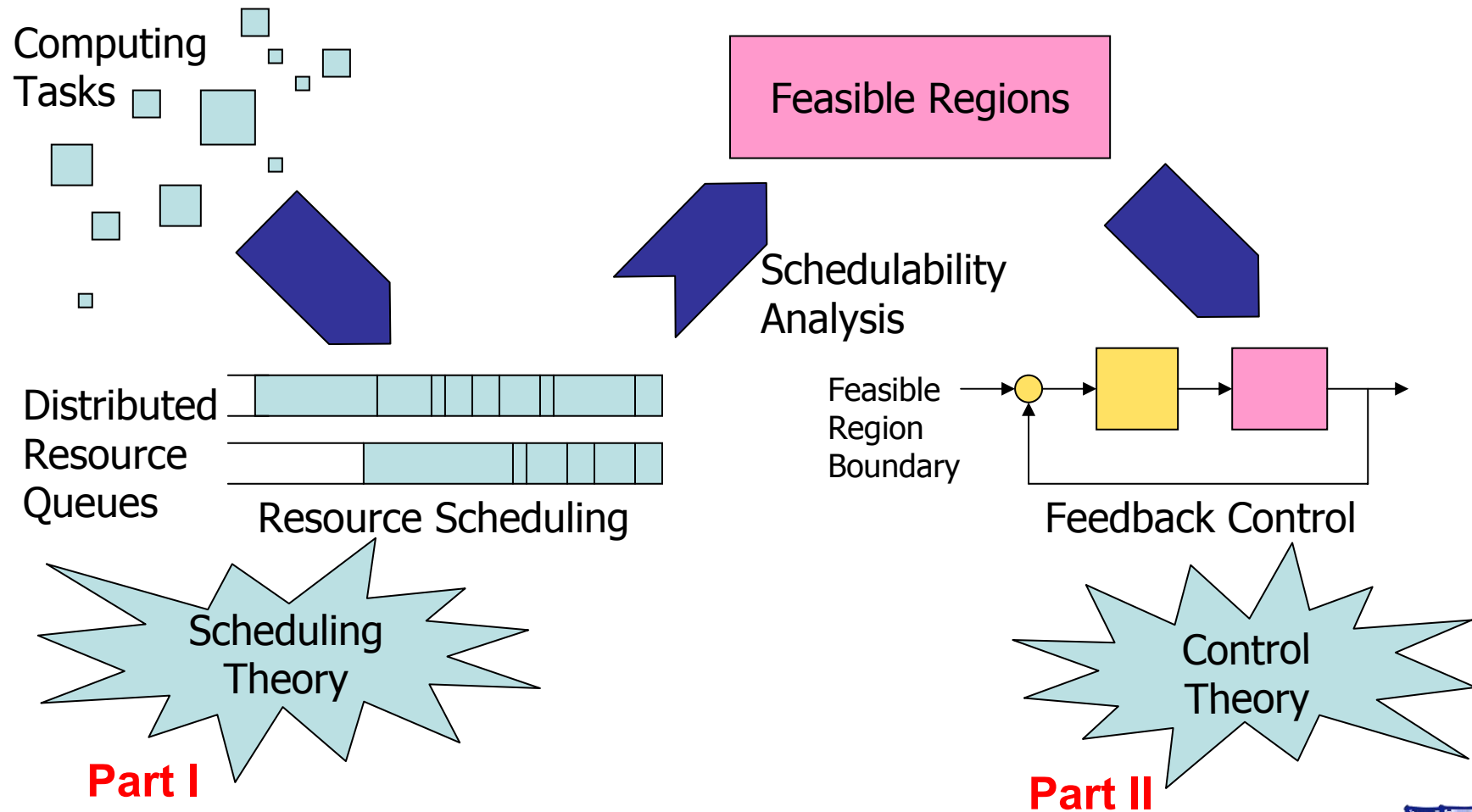


## Impact of Delay Composition Algebra

- A simple approach to transform arbitrary task graphs into equivalent uniprocessor workloads amenable to existing analysis techniques
- Transforms schedulability conditions of a distributed task set into a utilization bound (queue state) amenable to control
- Opportunities for future research:
  - Needs extension to resource blocking (mutual exclusion constraints), spatially partitioned resources (as opposed to prioritized), transactions (multiple resource acquired in an all-or-nothing fashion), ...
  - Advantages: simplicity, scalability.

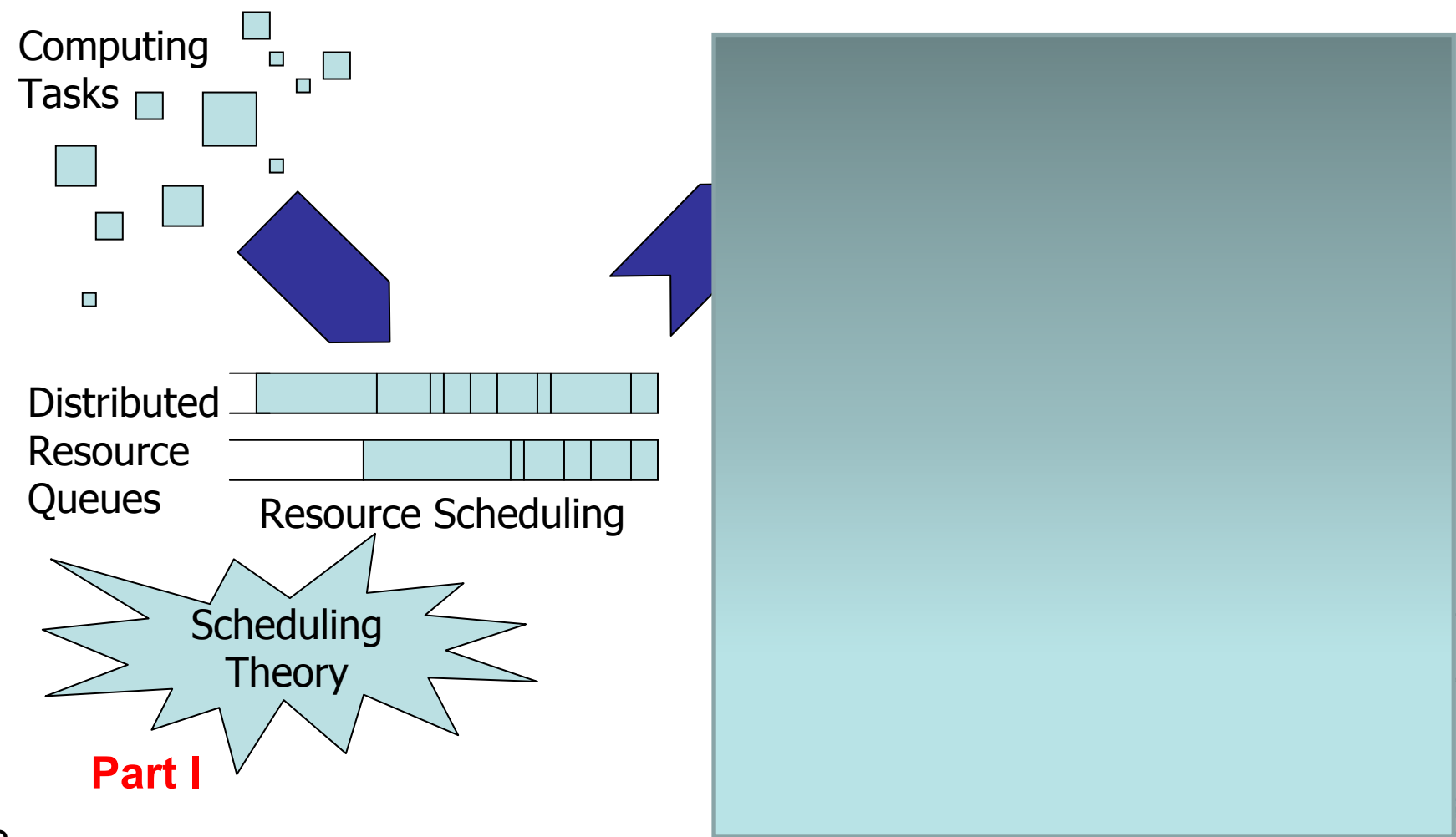
# Distributed Software Performance Control

Requirements on meeting deadlines  $\rightarrow$  utilization (queue state)  $\rightarrow$  performance control



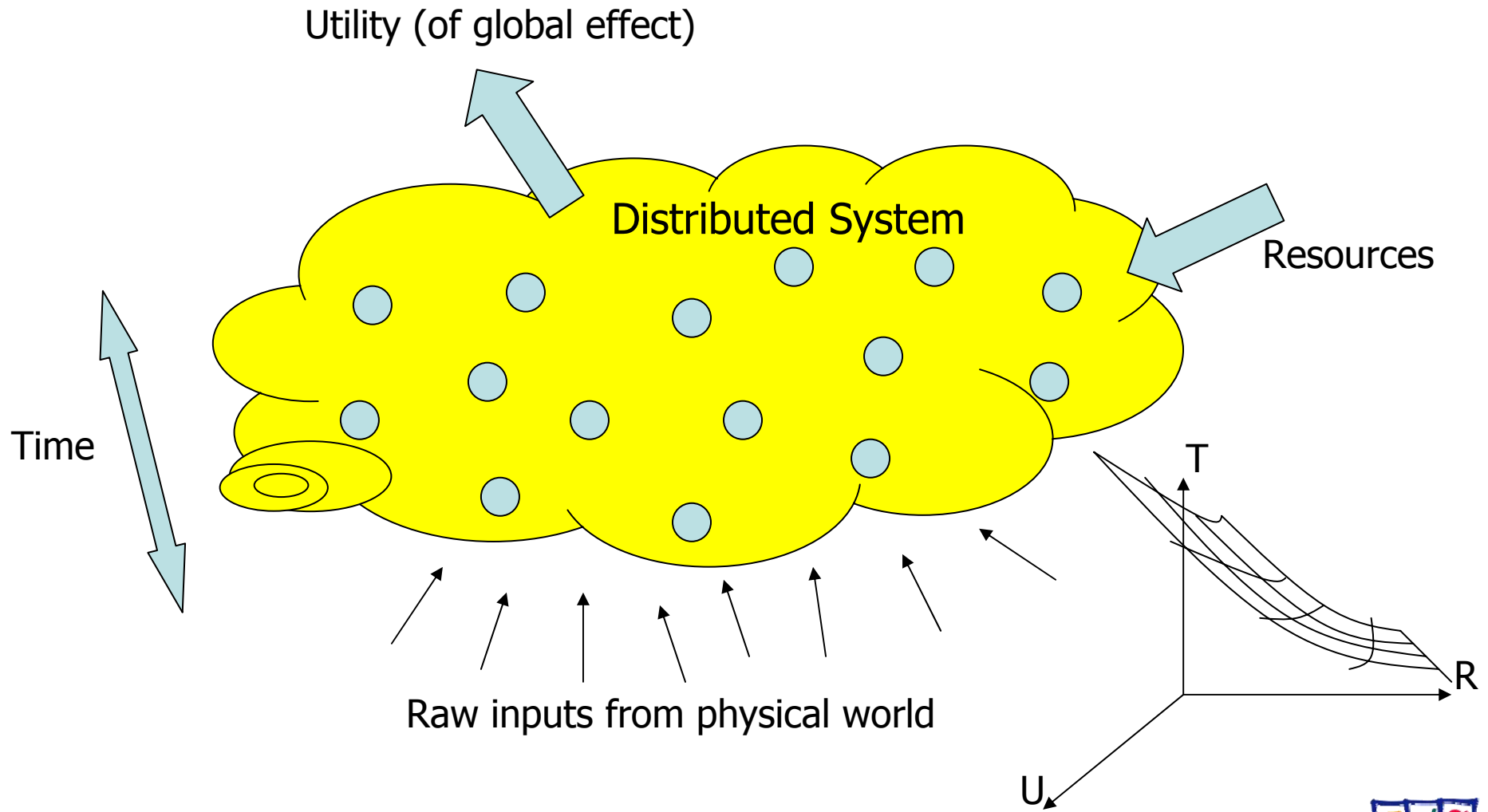
# Distributed Software Performance Control

Requirements on meeting deadlines → utilization (queue state) → performance control

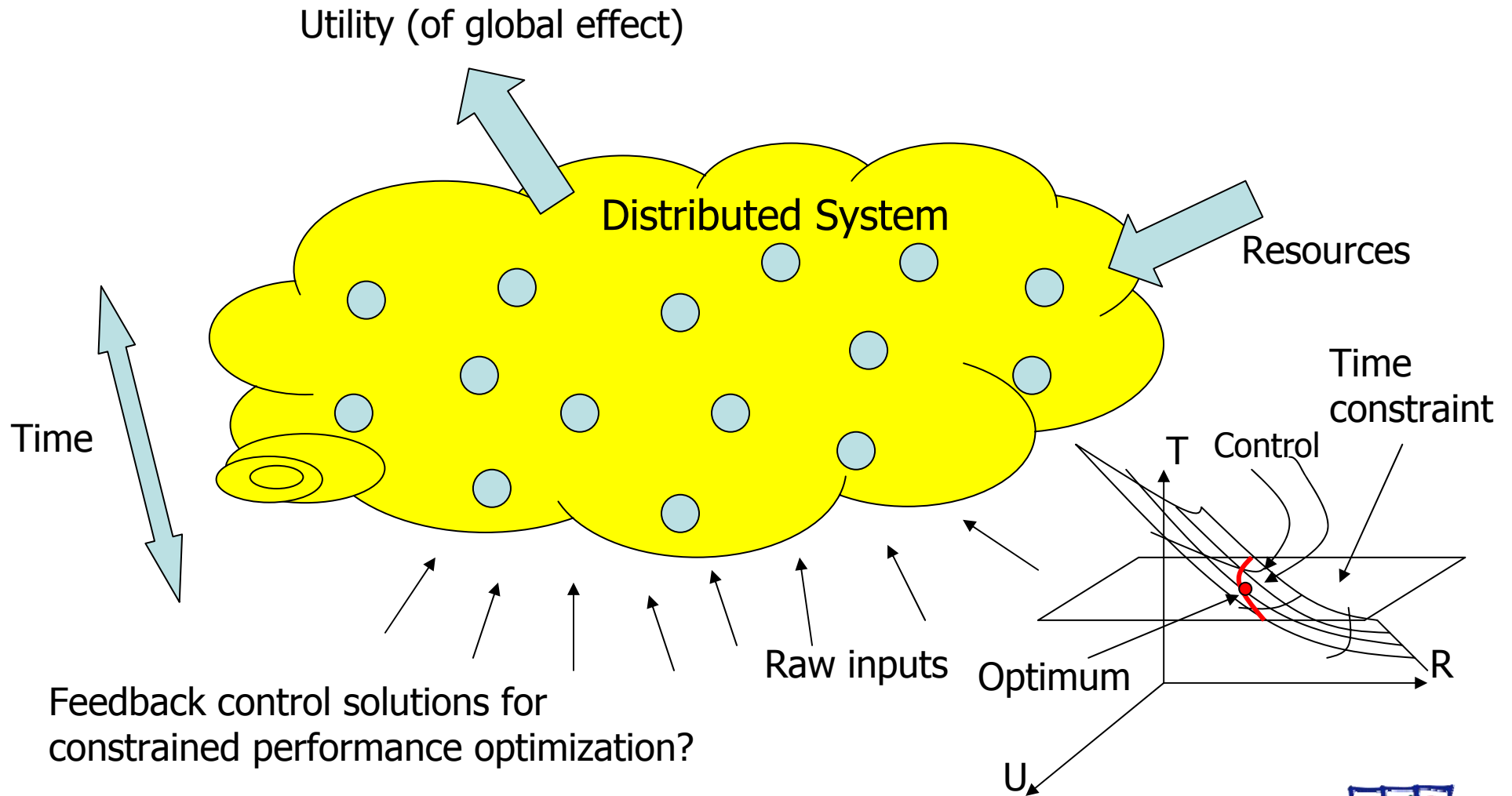


**Part I**

# Control of Performance Tradeoffs in Distributed Systems



# Control of Performance Tradeoffs in Distributed Systems





# A Methodology for Applying Feedback Control to Software

- Mapping: Performance management problem → Feedback problem
  - Determine the controlled performance metric (*output*) and its desired value (*set point*)
  - Determine the available actuators (*input*)
  - Map the performance control problem into a set of control loops
- Modeling:
  - Model the input-output relation as a difference equation:
 
$$Output_k = \sum_i a_i Output_{k-i} + \sum_i b_i Input_{k-i}$$
- Controller design:
  - Use control theory to find function  $f$ , such that:
 
$$Input = f(set\ point - output) \text{ makes } output \rightarrow set\ point$$



# Control Examples in Distributed Systems

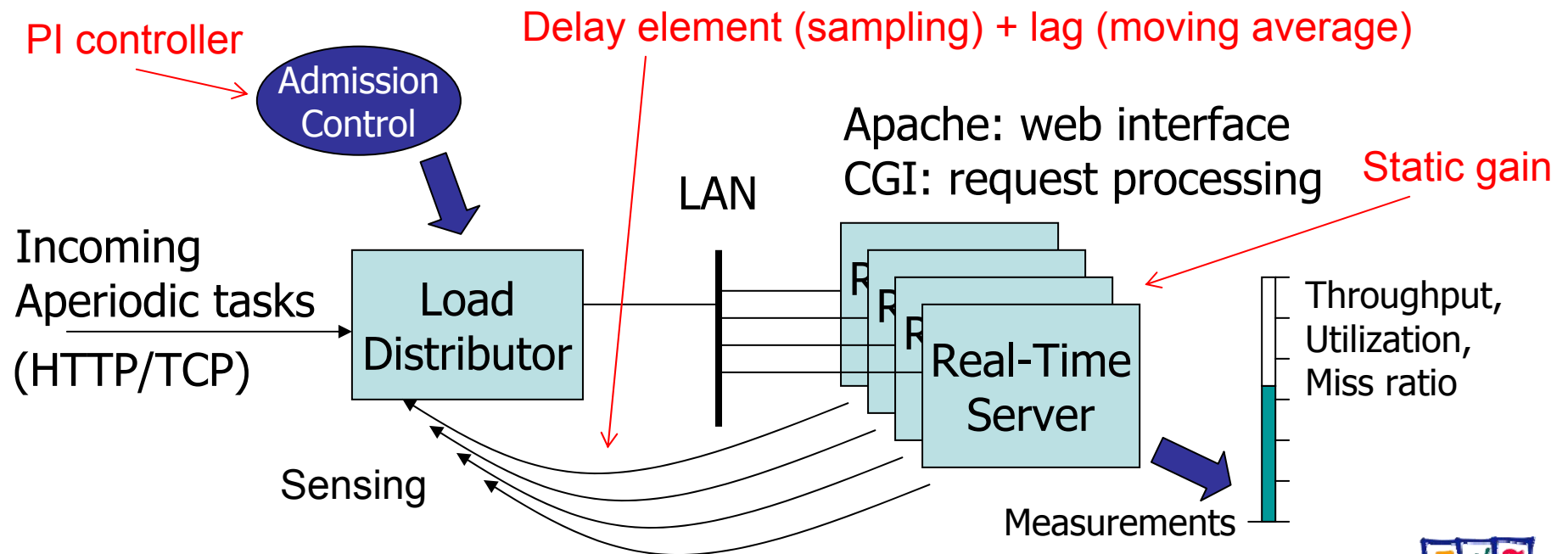
- Case 1: Centralized control, centralized actuation
- Case 2: Centralized control, distributed actuation
- Case 3: Distributed (localized) control and actuation

## Control Examples in Distributed Systems

- **Case 1: Centralized control, centralized actuation**
- Case 2: Centralized control, distributed actuation
- Case 3: Distributed (localized) control and actuation

# Experimental Testbed and Evaluation

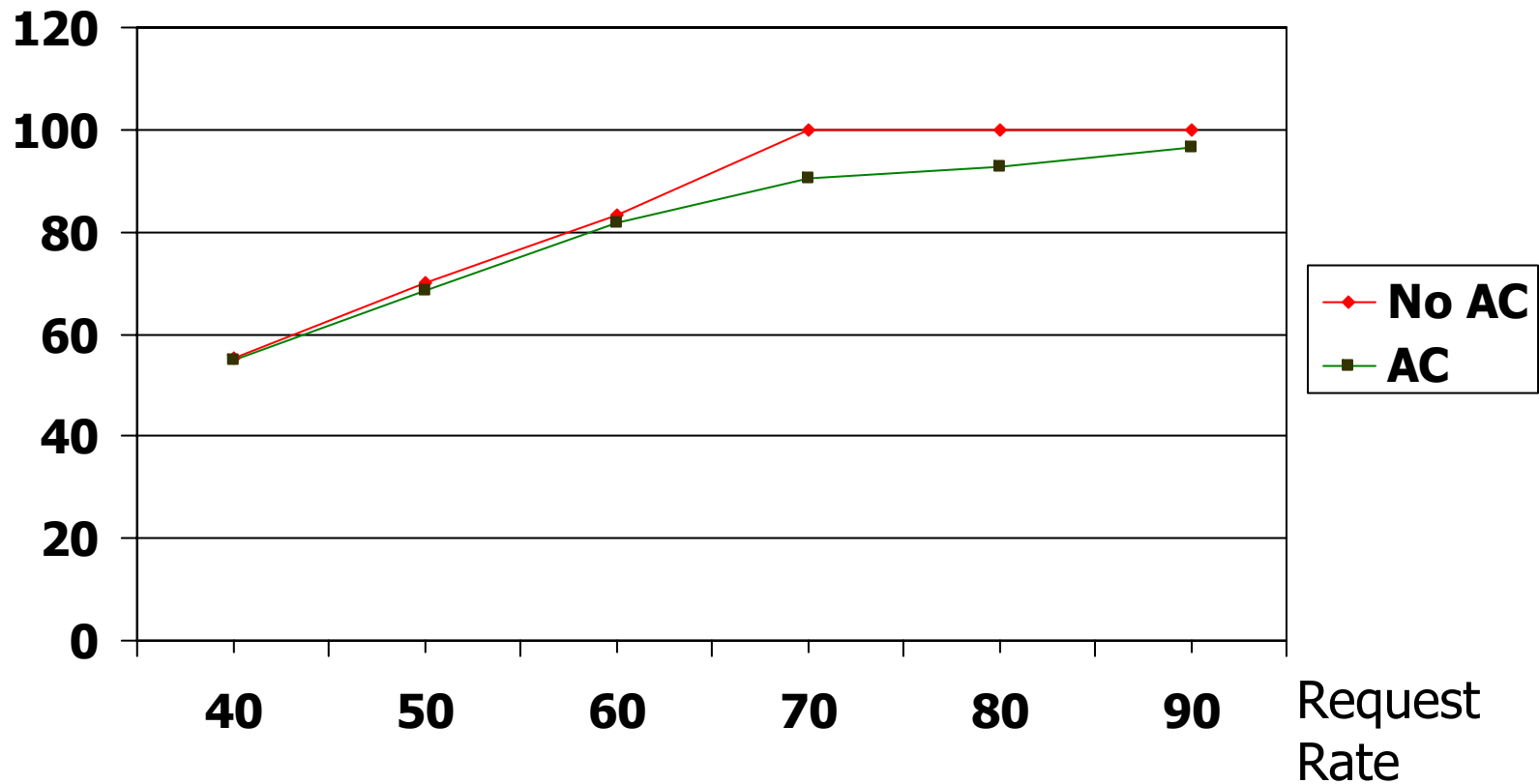
- A real-time computing cluster is developed under Linux
- 4 worker Linux PCs connected by 100Mbps LAN to a front-end load distributor



# Server Utilization

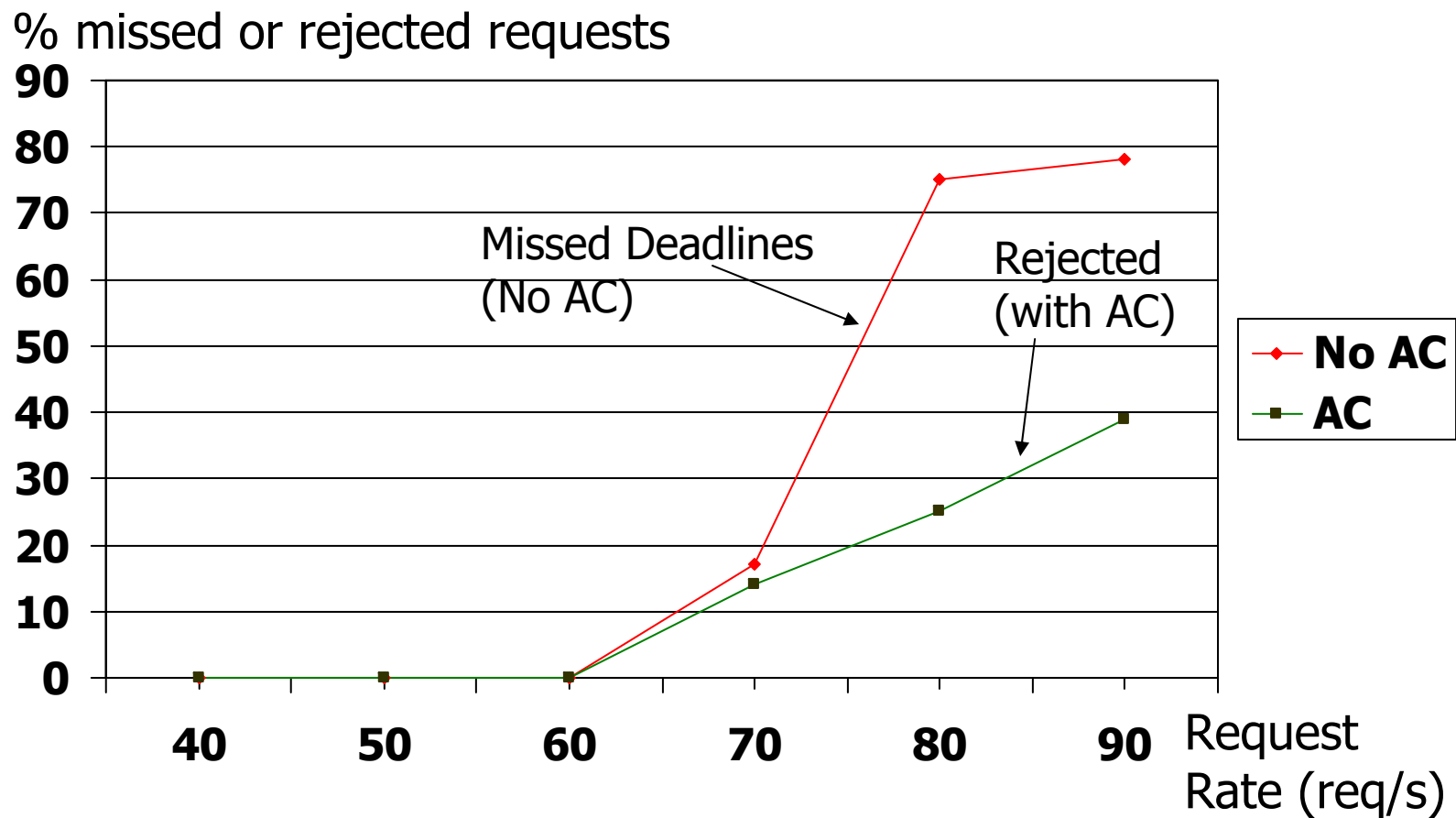
Utilization-based admission control does not underutilize server

Cluster Utilization



# Miss Ratio Profile

Utilization-based admission control improves task success ratio





# Control Examples in Distributed Systems

- Case 1: Centralized control, centralized actuation
- Case 2: Centralized control, distributed actuation
- Case 3: Distributed (localized) control and actuation

## Control Examples in Distributed Systems

- Case 1: Centralized control, centralized actuation
- **Case 2: Centralized control, distributed actuation**
- Case 3: Distributed (localized) control and actuation

# Energy Management in Server Farms

Energy expended on:

- Computing (powering up racks of machines)
  - Sensors: Machine utilization, Delay, Throughput, ...
  - Actuators: DVS, turning machines On/Off
- Cooling
  - Sensors: Temperature, air flow, ...
  - Actuators: Air-conditioning units, fans, ...
- Energy bill is 40-50% of total profit



# Problem Formulation

Step 1: Formulate objective and constraints

$$\min_{x_1, \dots, x_n} f(x_1, \dots, x_n)$$

Control knobs  
┌───────────┐

$$\text{subject to } g_j(x_1, \dots, x_n) \leq 0, \quad j = 1, \dots, m$$

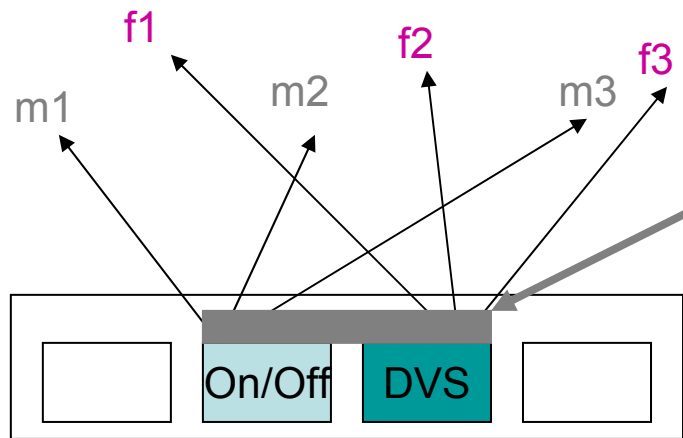
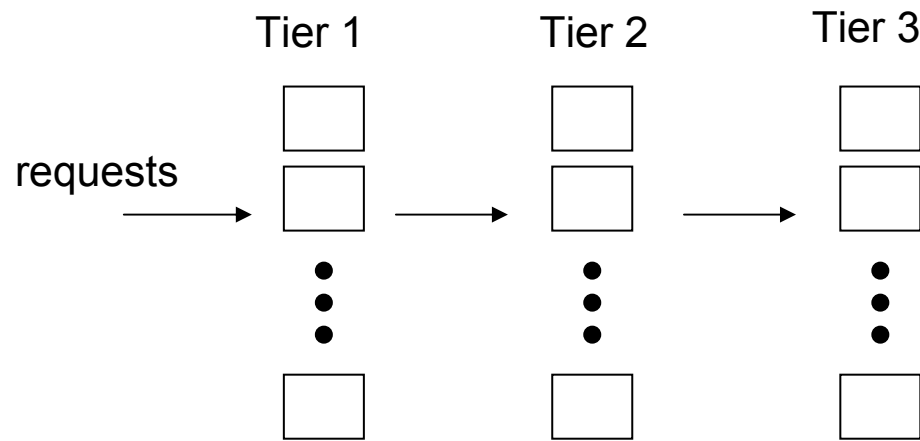
Step 2: Determine optimality conditions (KKT)

$$\Gamma_{x_i} \leftarrow \frac{\partial f(x_1, \dots, x_n)}{\partial x_i} + \sum_{j=1}^m \nu_j \frac{\partial g_j(x_1, \dots, x_n)}{\partial x_i} = 0$$

Step 3: Set up control loops

$$\Gamma_{x_1} = \Gamma_{x_2} = \Gamma_{\text{average}} \leftarrow \text{Set point}$$

# A Server Farm Case study



Composed distributed middleware

Find knob settings, (m1, m2, m3, f1, f2, f3) such that energy consumption is reduced



# Formulation

## Formulate constrained optimization

$$P_i(f_i) = A_i \cdot f_i^p + B_i$$

Power estimation of a machine at tier i

$$U_i = \frac{\lambda}{\mu} = \frac{\lambda_i/m_i}{f_i} = \frac{\lambda_i}{m_i f_i}$$

Queuing equation using number of machines and arrival rate

$$P_i(U_i, m_i) = A_i \cdot \left( \frac{\lambda_i}{U_i m_i} \right)^p + B_i = \frac{A_i \lambda_i^p}{U_i^p m_i^p} + B_i$$

Power estimation function of a machine at tier i

$$\min_{U_i \geq 0, m_i \geq 0} P_{tot}(U_i, m_i) = \sum_{i=1}^3 m_i \left( \frac{A_i \lambda_i^3}{U_i^3 m_i^3} + B_i \right)$$

$$\text{subject to } \sum_{i=1}^3 \frac{m_i}{\lambda_i} \cdot \frac{U_i}{1 - U_i} \leq K,$$

$$\sum_{i=1}^3 m_i \leq M$$

Find best composition of

$(m_1, m_2, m_3, U_1, U_2, U_3)$ ,

hence  $(m_1, m_2, m_3, f_1, f_2, f_3)$

## Optimality Conditions

- Derive necessary condition for optimality
  - Karush-Kuhn-Tucker (KKT) condition

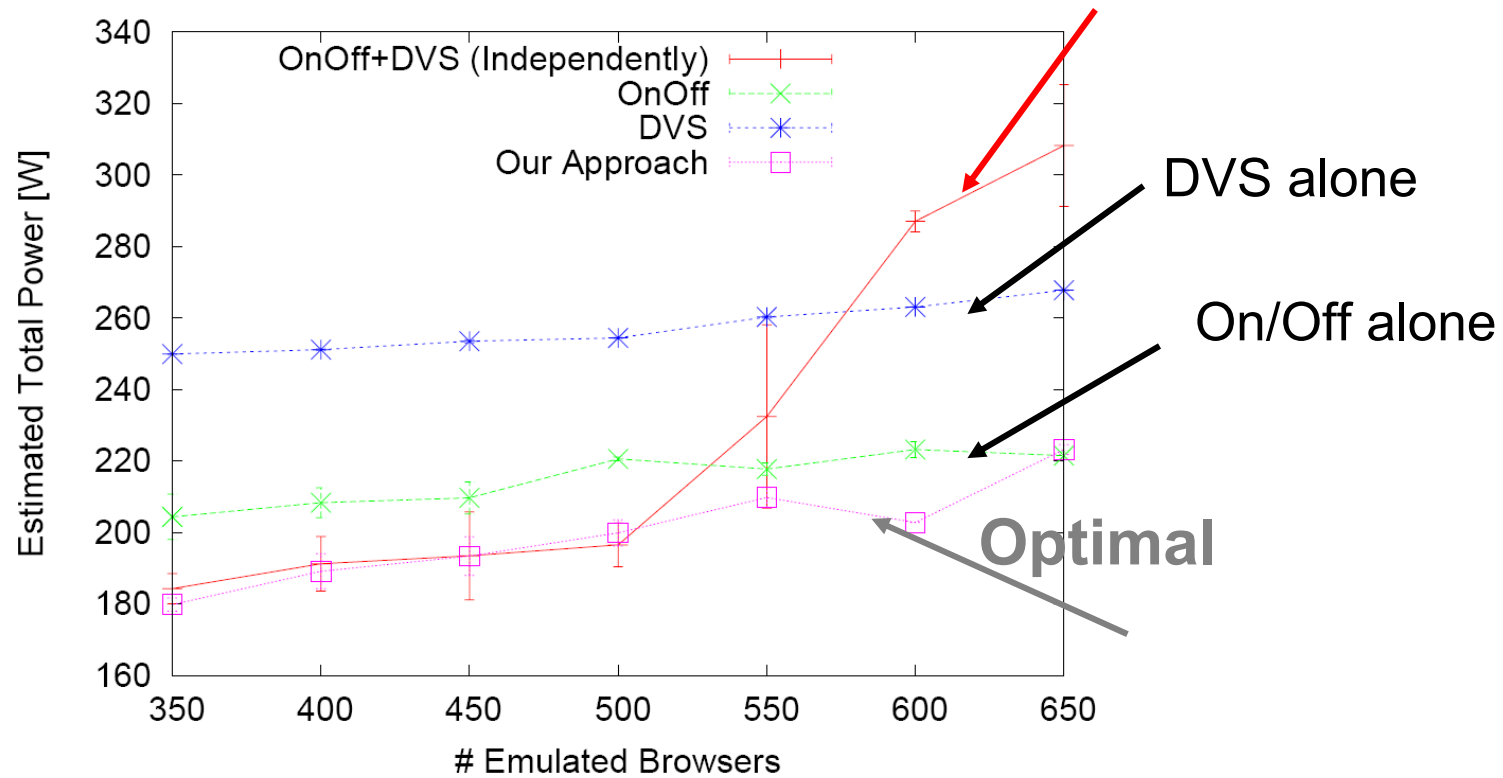
$$\frac{\lambda_1^4(1-U_1)^2}{m_1^3U_1^4} = \frac{\lambda_2^4(1-U_2)^2}{m_2^3U_2^4} = \frac{\lambda_3^4(1-U_3)^2}{m_3^3U_3^4}$$
$$\Gamma(m_1, U_1) = \Gamma(m_2, U_2) = \Gamma(m_3, U_3)$$

- $\text{Error}_i = \Gamma_{\text{avg}} - \Gamma(m_i, U_i)$ , where  $\Gamma_{\text{avg}}$  is average of  $\Gamma(m_i, U_i)$



# Evaluation

DVS + On/Off





# Control Examples in Distributed Systems

- Case 1: Centralized control, centralized actuation
- Case 2: Centralized control, distributed actuation
- Case 3: Distributed (localized) control and actuation

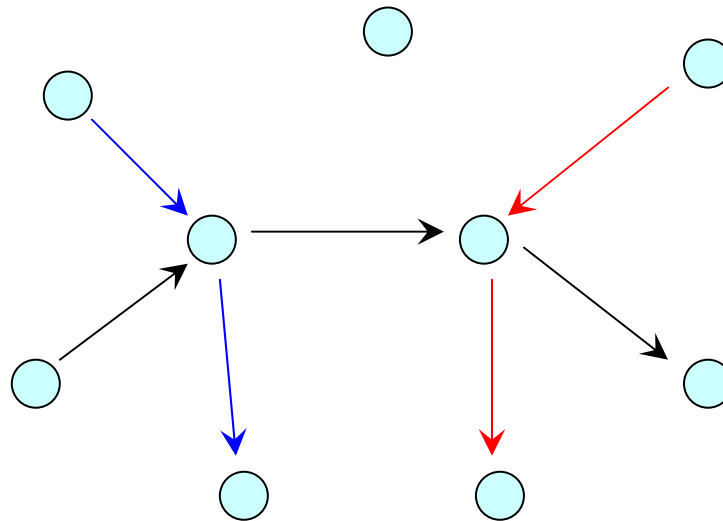


# Control Examples in Distributed Systems

- Case 1: Centralized control, centralized actuation
- Case 2: Centralized control, distributed actuation
- **Case 3: Distributed (localized) control and actuation**



# Example: Flow Rate Control in a Wireless Network subject to Delay Constraints



**Problem:** Allocate rates to elastic flows in a wireless network so as to maximize network utility, while meeting end-to-end delay requirements



# Problem Formulation

Step 1: Formulate objective and constraints

$$\begin{array}{l} \min_{x_1, \dots, x_n} \quad \overbrace{f(x_1, \dots, x_n)}^{\text{Control knobs}} \\ \text{subject to } \quad g_j(x_1, \dots, x_n) \leq 0, \quad j = 1, \dots, m \end{array}$$

Step 2: Decentralize the constraints:  $\sum_{i=1, \dots, n} Term_i \leq 0$

$$\xrightarrow{\text{becomes}} Sum_i = Term_i + Sum_{i-1}, Sum_n \leq 0$$

Step 3: Determine optimality conditions (KKT)

$$\Gamma_{x_i} \leftarrow \frac{\partial f(x_1, \dots, x_n)}{\partial x_i} + \sum_{j=1}^m \nu_j \frac{\partial g_j(x_1, \dots, x_n)}{\partial x_i} = 0$$

Step 4: Set up control loops

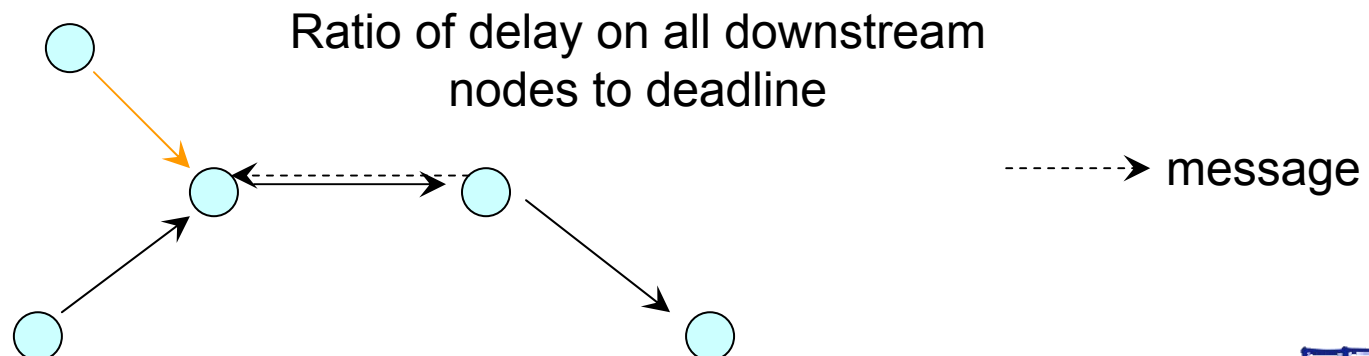
$$x_i(k) = x_i(k-1) + C_i h(\nu_i, \dots, \nu_m)$$

$$\nu_j(k) = \nu_j(k-1) + G_j \psi(x_1, \dots, x_n)$$



## Decentralizing the Delay Constraint

- For each flow  $s$  and hop  $i$ , define additional variables to hold the value of the left hand side of the constraint for hops  $i$  and up to the destination
- Each node only requires value from immediate downstream node
- Source constraint compares end-to-end delay to deadline



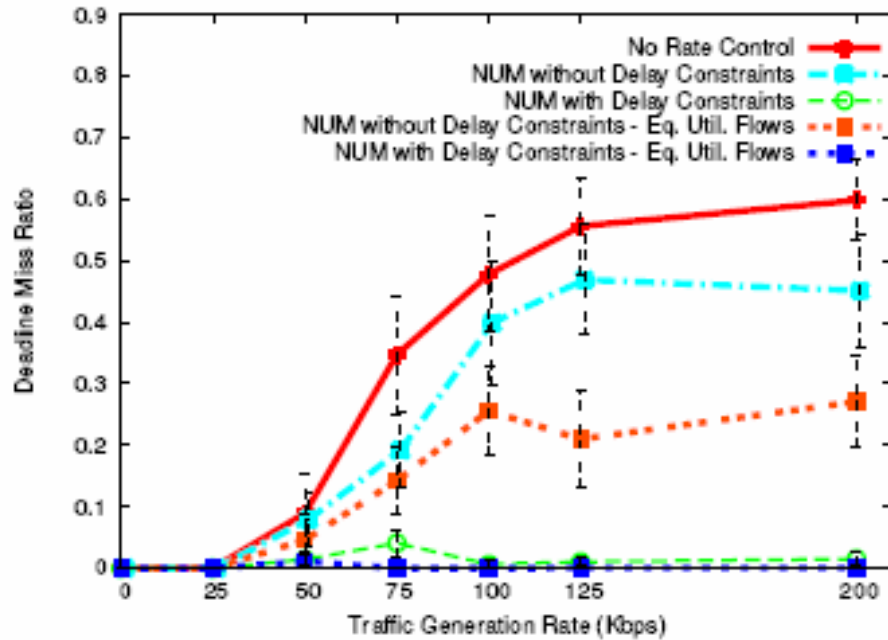


# Simulation Setup

- Implemented on ns2
- 50 nodes placed uniformly at random
- 802.11 with prioritized scheduling; DSDV routing
- 5 elastic flows, 3 priorities with no. of flows in each in the ratio 1:2:4 (for high:medium:low) – end-to-end deadlines 2, 4, 7s
- Algorithms:
  - ❖ No rate control
  - ❖ NUM w/o delay constraints performs only rate control based on capacity constraints
  - ❖ NUM with delay constraints performs rate control based on capacity as well as delay constraints
- Utility function
  - ❖ Importance proportional to urgency
  - ❖ Importance same for all flows regardless of urgency ('Eq. Util. Flows')

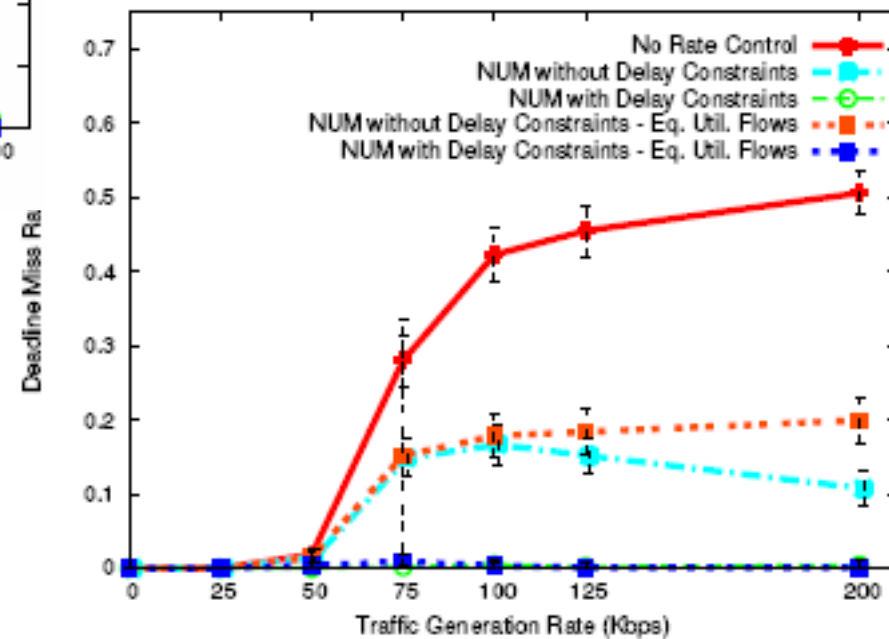


# Simulation Results



High Priority Flows

Low Priority Flows





## Conclusions

- Emerging distributed systems will feature increased scale, interaction with a physical environment, uncertainty, and autonomy
- We need analytic tools for predicting and controlling the temporal behavior of such systems
- Theoretical foundations are needed to bridge the gap between software and feedback control abstractions
- A theory is developed for translating schedulability constraints in a class of distributed systems into constraints on utilization (or virtual queue) metrics
- These constraints can be decentralized leading to localized algorithms that collectively converge to global optima
- The feedback control problem derives from an optimization problem: The distributed system as an iterative optimizer



## A Summary of Challenges

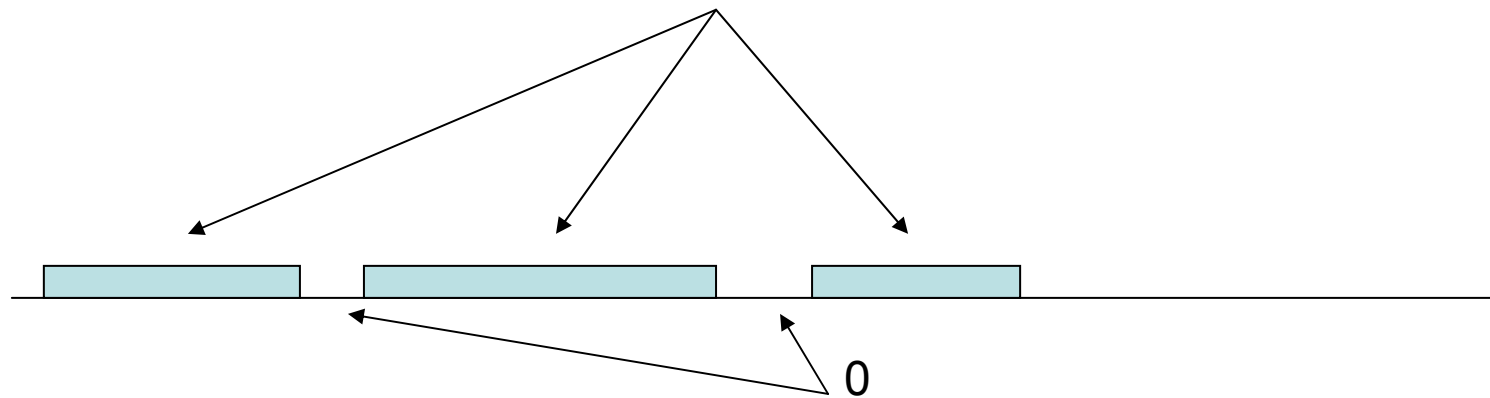
- What are other general categories of fine-grained constraints that can be converted to aggregate state variables amenable to control?
- How to translate desired global properties into localized protocol interactions?
- How to model nonlinearities specific to software?
- How to incorporate complex information extraction algorithms (e.g., data mining) in control loops?
- How to achieve convergence in poorly structured evolving systems? (e.g., mobile ad hoc networks, changing connectivity, non-uniform resource density, etc.)



# Backup: Instantaneous versus Real Utilization

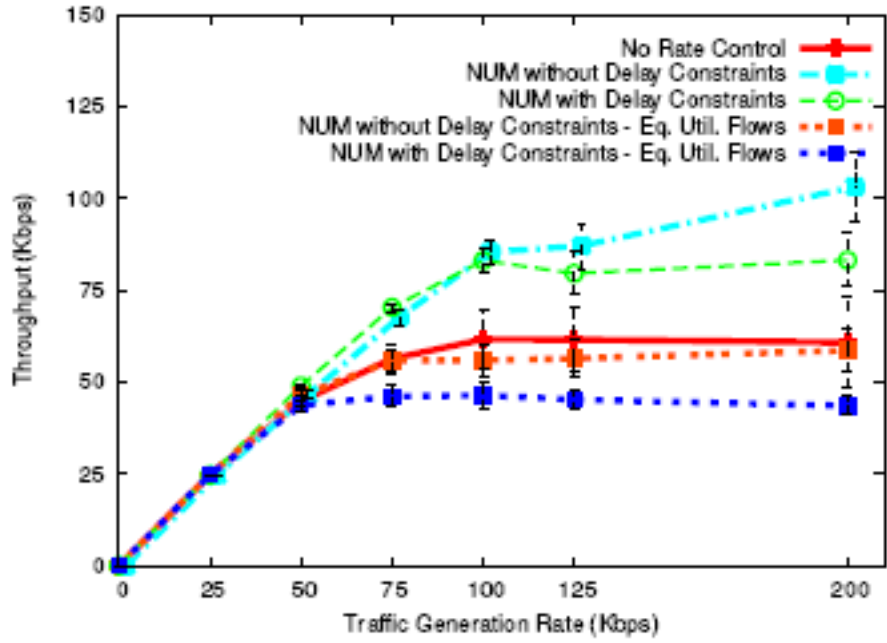
- An important property is that:  
avg. instantaneous utilization < avg. real utilization.  
→ Hence, server is not underutilized!
- Proof:

Real utilization = 1  
Synthetic utilization < 0.58



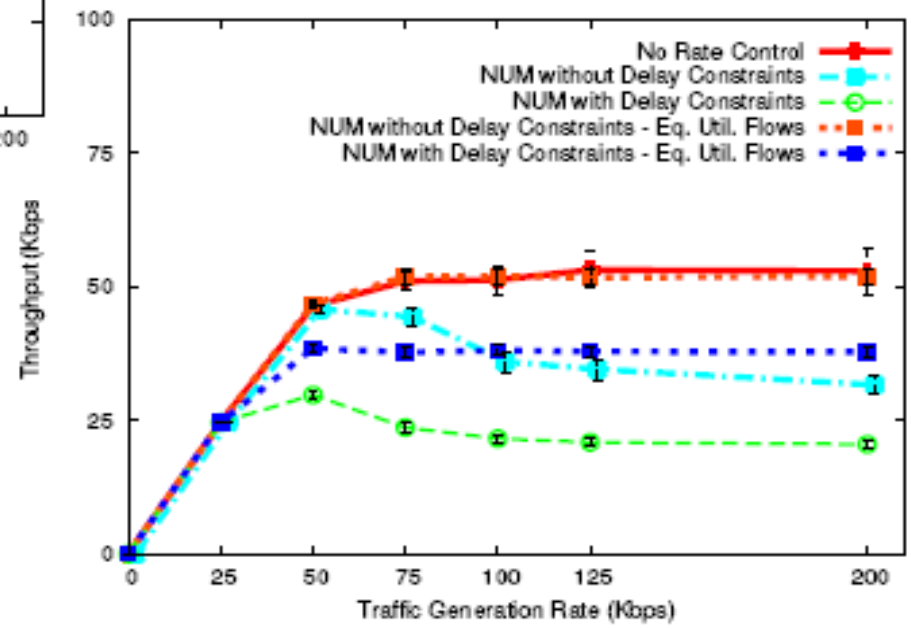


# Simulation Results (2/3)



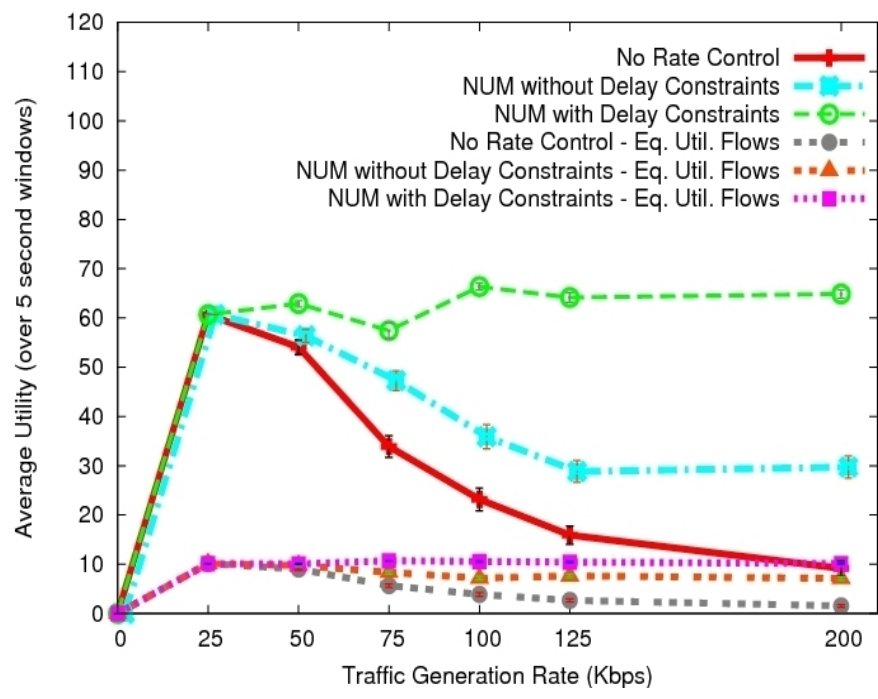
High Priority Flows

Low Priority Flows

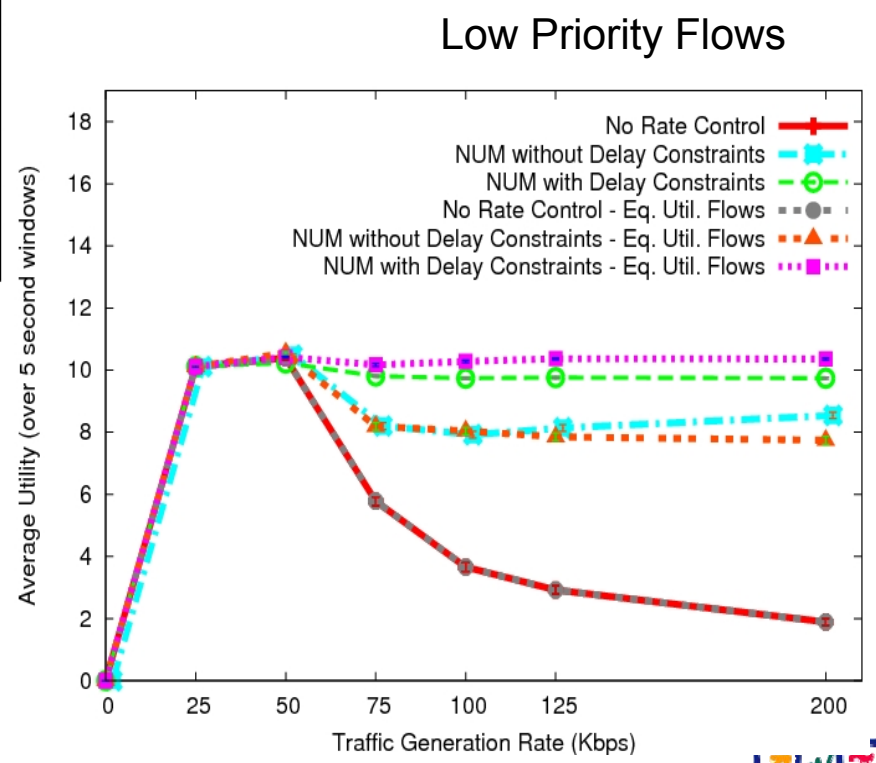




# Simulation Results (3/3)



High Priority Flows



Low Priority Flows