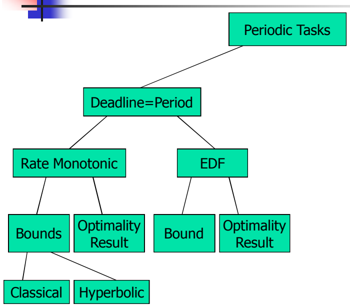
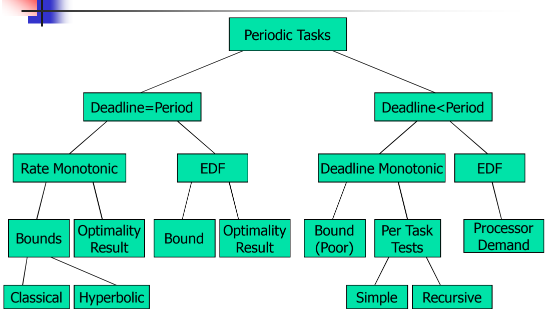
Aperiodic Servers

Fixed Priority

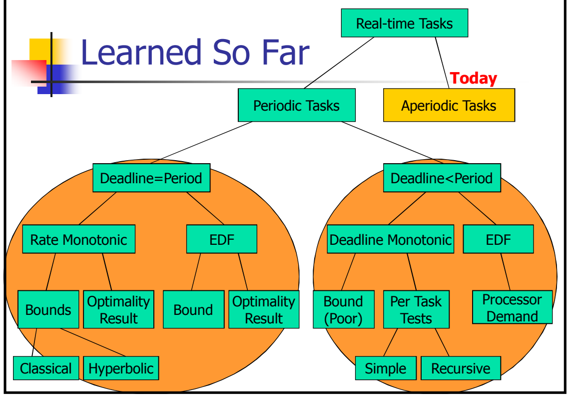
Learned So Far



Learned So Far



Learned So Far



Mixed Periodic and Aperiodic Task Systems

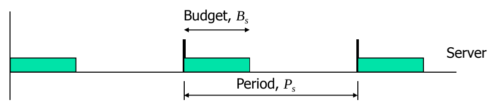
- Question: how to execute aperiodic tasks without violating schedulability guarantees given to periodic tasks?

Mixed Periodic and Aperiodic Task Systems

- Question: how to execute aperiodic tasks without violating schedulability guarantees given to periodic tasks?
- One Answer: Execute aperiodic tasks at lowest priority
  - Problem: Poor performance for aperiodic tasks

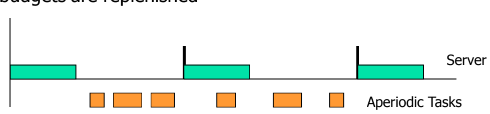
Mixed Periodic and Aperiodic Task Systems

- Idea: aperiodic tasks can be served by periodically invoked servers
- The server can be accounted for in periodic task schedulability analysis
- The server has a period P_s and a budget B_s
- Server can serve aperiodic tasks until budget expires
- Servers have different flavors depending on the details of when they are invoked, what priority they have, and how budgets are replenished



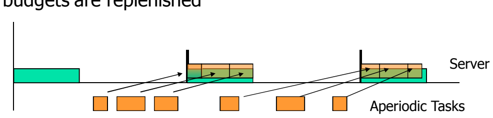
Mixed Periodic and Aperiodic Task Systems

- Idea: aperiodic tasks can be served by periodically invoked servers
- The server can be accounted for in periodic task schedulability analysis
- The server has a period P_s and a budget B_s
- Server can serve aperiodic tasks until budget expires
- Servers have different flavors depending on the details of when they are invoked, what priority they have, and how budgets are replenished



Mixed Periodic and Aperiodic Task Systems

- Idea: aperiodic tasks can be served by periodically invoked servers
- The server can be accounted for in periodic task schedulability analysis
- The server has a period P_s and a budget B_s
- Server can serve aperiodic tasks until budget expires
- Servers have different flavors depending on the details of when they are invoked, what priority they have, and how budgets are replenished

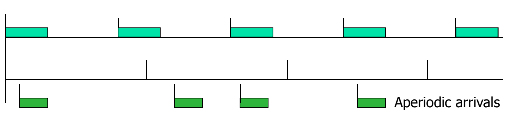


Polling Server

- Runs as a periodic task (priority set according to RM)
- Aperiodic arrivals are queued until the server task is invoked
- When the server is invoked it serves the queue until it is empty or until the budget expires then suspends itself
  - If the queue is empty when the server is invoked it suspends itself immediately.
- Server is treated as a regular periodic task in schedulability analysis

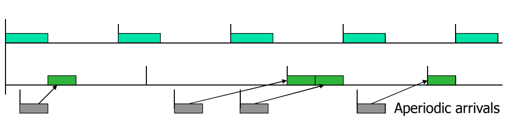
Example of a Polling Server

- Polling server:
  - Period $P_s = 5$
  - Budget $B_s = 2$
- Periodic task
  - $P = 4$
  - $C = 1.5$
- All aperiodic arrivals have C=1



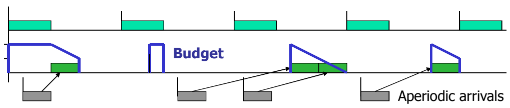
Example of a Polling Server

- Polling server:
  - Period $P_s = 5$
  - Budget $B_s = 2$
- Periodic task
  - $P = 4$
  - $C = 1.5$
- All aperiodic arrivals have C=1



Example of a Polling Server

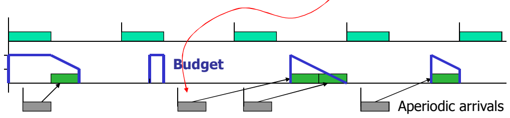
- Polling server:
  - Period $P_s = 5$
  - Budget $B_s = 2$
- Periodic task
  - $P = 4$
  - $C = 1.5$
- All aperiodic arrivals have C=1



Example of a Polling Server

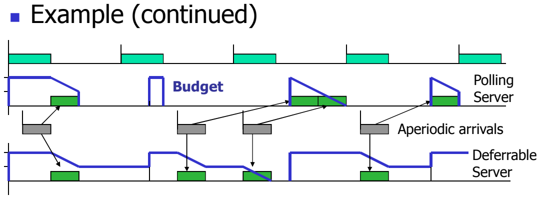
- Polling server:
  - Period $P_s = 5$
  - Budget $B_s = 2$
- Periodic task
  - $P = 4$
  - $C = 1.5$
- All aperiodic arrivals have C=1

Why not execute immediately?

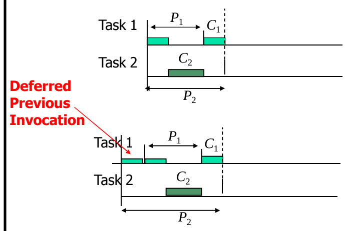


Deferrable Server

- Keeps the balance of the budget until the end of the period
- Example (continued)



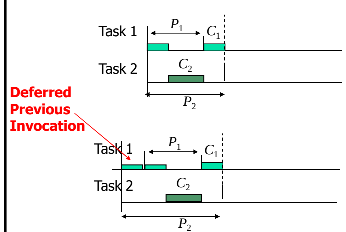
Worst-Case Scenario

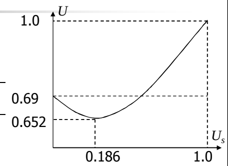


$$U_p \leq \ln\left(\frac{2U_s + 1}{U_s + 1}\right)$$

Exercise: Derive the utilization bound for a deferrable server plus one periodic task

Worst-Case Scenario





$$U_p \leq \ln\left(\frac{2U_s + 1}{U_s + 1}\right)$$

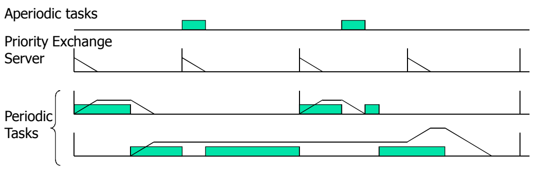
Exercise: Derive the utilization bound for a deferrable server plus one periodic task

Priority Exchange Server

- Like the deferrable server, it keeps the budget until the end of server period
- Unlike the deferrable server the priority slips over time: When not used the priority is exchanged for that of the executing periodic task

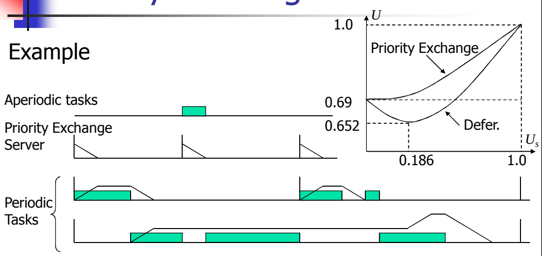
Priority Exchange Server

Example



Priority Exchange Server

Example



$$U_p \leq \ln\left(\frac{2}{U_s + 1}\right)$$

Sporadic Server

- Server is said to be *active* if it is in the *running* or *ready* queue, otherwise it is *idle*.
- When an aperiodic task comes and the budget is not zero, the server becomes active
- Every time the server becomes *active*, say at t_A, it sets replenishment time one period into the future, $t_A + P_s$ (but does not decide on replenishment amount).
- When the server becomes idle, say at t_I, set replenishment amount to capacity consumed in $[t_A, t_I]$

$$U_p \leq \ln\left(\frac{2}{U_s + 1}\right)$$

Slack Stealing Server

- Compute a slack function $A(t_s, t_f)$ that says how much total slack is available
- Admit aperiodic tasks while slack is not exceeded

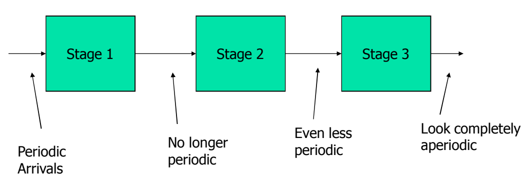
Why An Aperiodic Theory for Real-Time Systems?

Reason #1: Aperiodic tasks are an increasing proportion of workload. They are no longer the "exception"

- Consider a web server serving randomly arriving web requests
- Each request has a desired response time
- Can one invent an aggregate measurable utilization-like metric (we call it synthetic utilization, U), such that all deadlines are met as long as U is below some threshold, U_{max}?
- Feasible region: $0 < U < U_{max}$

Why An Aperiodic Theory for Real-Time Systems?

Reason #2: Even systems where tasks arrive periodically suffer aperiodic artifacts if there are multiple execution stages



What We Know So Far

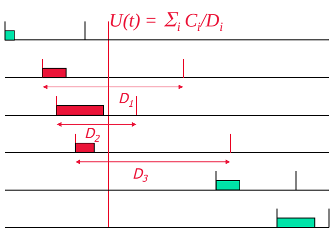
- A utilization bound for periodic tasks (1973):
  - Assume that each task i executes for C_i every period P_i.
  - Processor utilization needed for this task is: $U_i = C_i/P_i$
  - The task set is schedulable by an optimal fixed-priority scheduling policy if $\Sigma_i C_i/P_i < 0.69$
  - Optimal fixed priority policy is rate-monotonic (higher rate = higher priority)

Aperiodic Scheduling Theory Rethinking the Basic Concepts

- The task model
- The notion of utilization
- The classification of scheduling policies
- The sense of optimality of a real-time scheduling policy

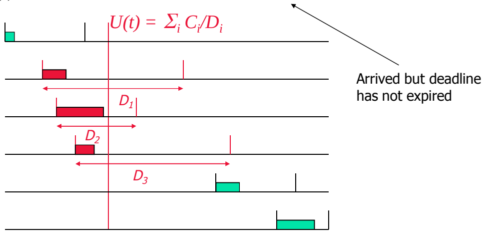
Aperiodic Task and *Instantaneous* Utilization

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



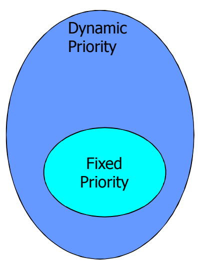
Aperiodic Task and *Instantaneous* Utilization

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



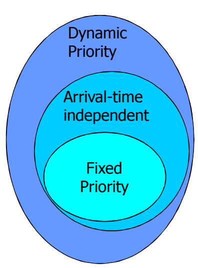
Fixed versus Dynamic Priority Scheduling

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



Why Arrival-Time Independent Scheduling?

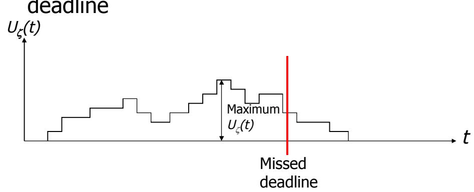
- Easy to implement on current non-real-time operating systems with fixed-priority support (e.g., UNIX, the #1 OS for web servers)
  - Requires a finite number of priority levels
  - Priorities are statically assigned to threads

Generalized Sense of Optimality

- A scheduling policy is optimal in a class if it maximizes the schedulable synthetic 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
- Note: Is it the same as saying optimal can scheduling anything a non-optimal can?

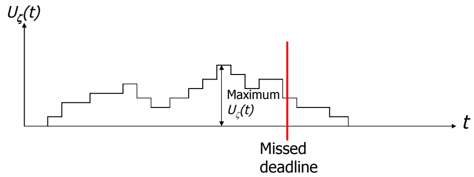
Main Idea of Derivation

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



Main Idea of Derivation

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



Why minimize the *maximum* U ?

Steps of Derivation

- Consider an unschedulable task invocation in a busy period
- Within the busy period minimize the maximum $U(t)$ w.r.t.:
  - Task invocation execution times
  - Task invocation arrival times
  - Task invocation deadlines
- Show that an arbitrary policy is either deadline monotonic or has a higher utilization bound

Main Results

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

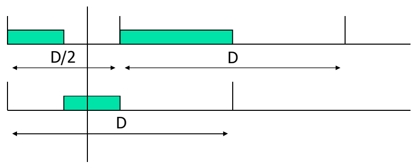
$$U(t) \le \tfrac{1}{2} + \tfrac{1}{2n} \qquad n < 3$$

$$U(t) \le \frac{1}{1+\sqrt{\frac{1}{2}\left(1-\frac{1}{n-1}\right)}} \qquad n \ge 3$$

Deadline-monotonic scheduling is the optimal arrival-time independent policy for acyclic tasks

Example of Tightness

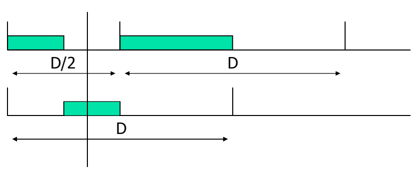
- Consider n=2



$U(t)$ = 0.75

Example of Tightness

- Consider n=2

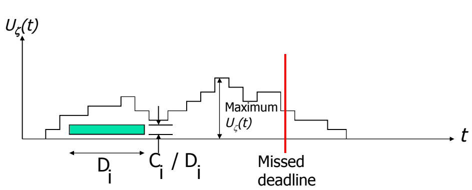


$U(t)$ = 0.75

Why is it lower than the periodic bound?

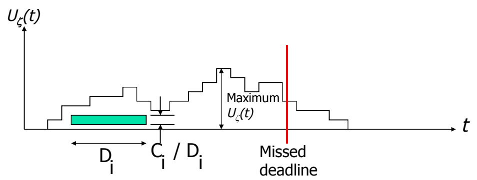
Quick-and-Dirty Derivation

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



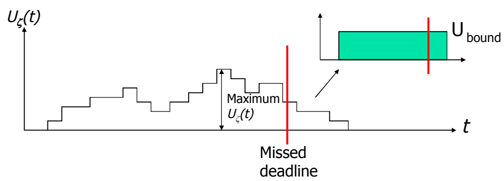
Corollary

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



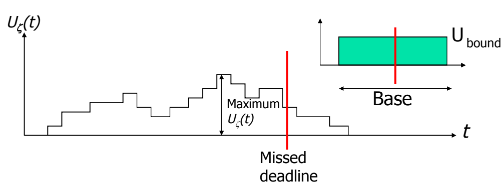
Derivation

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



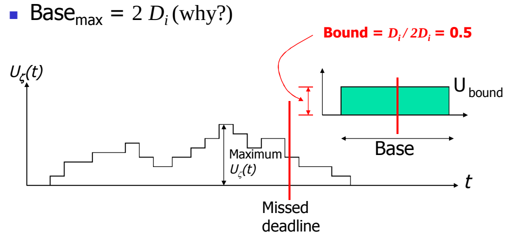
Example: Deadline Monotonic Bound

- $\mathbf{C}_{min} = D_i$ (for task to be critically schedulable)
- $\text{Base}_{max} = 2\, D_i$ (why?)

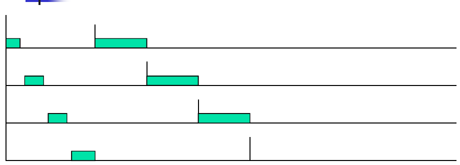


Example: Deadline Monotonic Bound

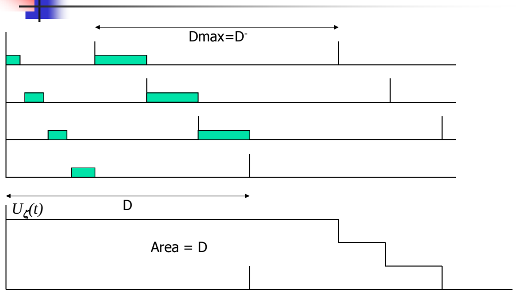
- C_{min} = D_i (for task to be critically schedulable)
- $Base_{max}$ = 2 D_i (why?)



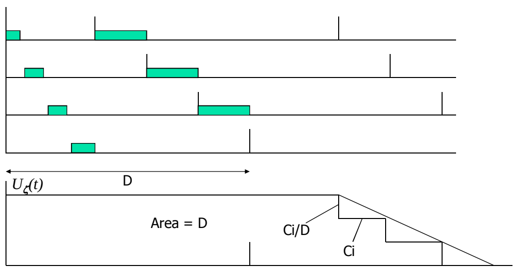
A Realizable Worst-Case Arrival Pattern



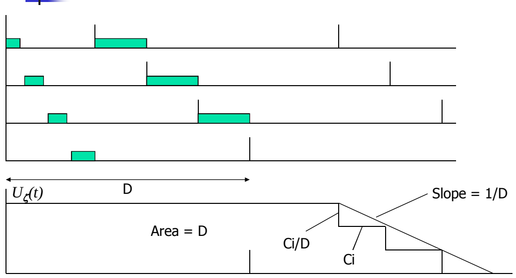
A Realizable Worst-Case Arrival Pattern



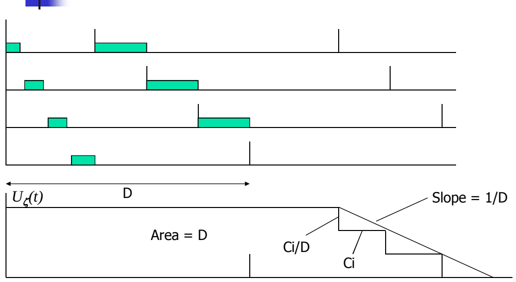
A Realizable Worst-Case Arrival Pattern



A Realizable Worst-Case Arrival Pattern

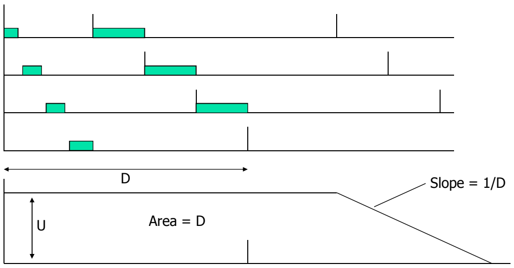


A Realizable Worst-Case Arrival Pattern



How big are the individual tasks when the bound is minimized?

A Realizable Worst-Case Arrival Pattern



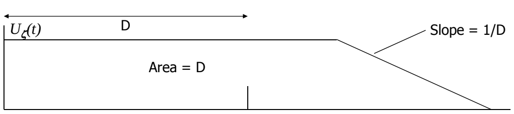
How big are the individual tasks when the bound is minimized?

A Realizable Worst-Case Arrival Pattern

$$U(2D + 2D - UD)/2 = D$$

$$U^2 - 4U + 2 = 0$$

$$U = 2 - \sqrt{2} = \frac{1}{1 + \sqrt{1/2}}$$



How big are the individual tasks when the bound is minimized?

Main Results

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

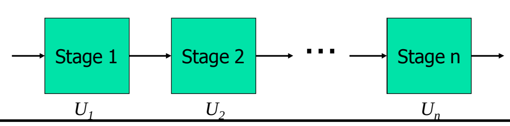
$$U(t) \leq \frac{1}{1 + \sqrt{\frac{1}{2}}}$$

Monitoring Synthetic Utilization

- Aperiodic Task Contracts:
  - On arrival: $U_a = U_a + (C/D)_{contract}$
  - On expiration: $U_a = U_a - (C/D)_{contract}$
- Periodic Task Contracts:
  - On arrival: $U_p = U_p + (C/D)_{contract}$
  - On expiration: $U_p = U_p - (C/D)_{contract}$
- Admission test: $U_a + U_p < U_{bound}$
- On Processor Idle Time: set $U_a = 0$

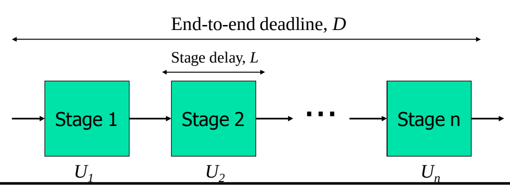
Schedulability in Distributed Systems

- Let $U_1, U_2, \ldots, U_n$ be the (*synthetic*) utilization values of n stages in a pipeline
- We have:
  Stage deadline met if $U_i < Bound$
- Can we derive: end-to-end deadlines met if
  $f(U_1, \ldots, U_n) < Bound$



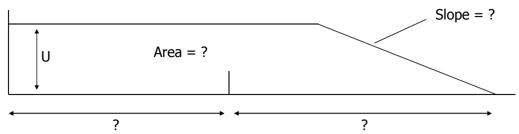
Schedulability in Distributed Systems

- Question: Is there a relation between the fraction of your end-to-end deadline that you spend on one stage and the synthetic utilization of that stage?



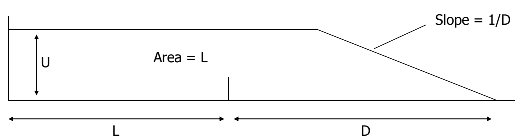
The Stage Delay Theorem

- Consider the worst case pattern:



The Stage Delay Theorem

- Consider the worst case pattern:



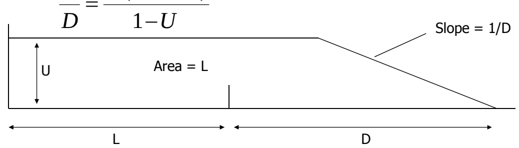
The Stage Delay Theorem

- Consider the worst case pattern:

$$U(2L+2D-UD)/2=L$$

$$DU(1-U/2)=(1-U)L$$

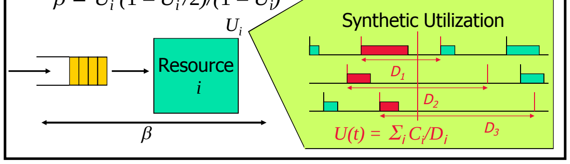
$$\frac{L}{D}=\frac{U(1-U/2)}{1-U}$$



Constructing Feasible Regions: The Stage Delay Theorem

- **The Stage Delay Theorem:** If the synthetic utilization of resource i, does not exceed U_i, then no task is queued on resource i under deadline monotonic scheduling for more than a fraction β of its end-to-end deadline, where:

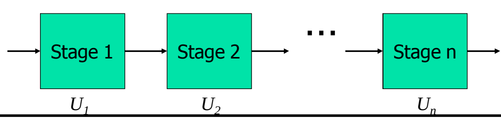
$\beta = U_i\,(1 - U_i/2)/(1 - U_i)$



Main Result: Schedulability of Resource Pipelines

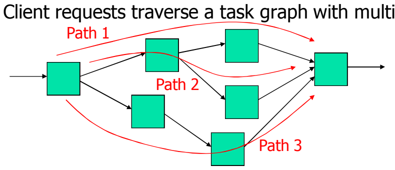
- Let $U_1, U_2, \ldots, U_n$ be the synthetic utilization values of n stages in a pipeline
- All end-to-end deadlines are met if:

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



Schedulability Regions of Arbitrary Task Graphs

- Let $U_1, U_2, \ldots, U_n$ be the synthetic utilization values of n servers $S_1, S_2, \ldots, S_n$ in a distributed system
- Client requests traverse a task graph with multiple flows



- Client requests meet their end-to-end deadlines if:

$$\max_{path} \sum_{S_i \in path} \frac{U_i(1-U_i/2)}{1-U_i} \leq 1$$

Example

- Is every task T schedulable in the pipeline below?

Task	Stage1 Comp. Time	Stage2 Comp. Time	Deadline
Task 1	0.1	0.1	1
Task 2	0.2	0.1	2
Task 3	1	0.125	5
Task 4	2	0.5	20

Example

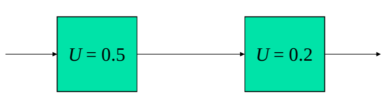
- Is every task T schedulable in the pipeline below?

Task	Stage1 Comp. Time	Stage2 Comp. Time	Deadline
Task 1	0.1	0.1	1
Task 2	0.2	0.1	2
Task 3	1	0.125	5
Task 4	2	0.5	20

Example

- Is every task T schedulable in the pipeline below?

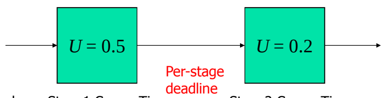
0.5(1-0.25)/0.5 + 0.2(1-0.1)/0.8=0.75+0.225=0.925<1 → OK!



Task	Stage1 Comp. Time	Stage2 Comp. Time	Deadline
Task 1	0.1	0.1	1
Task 2	0.2	0.1	2
Task 3	1	0.125	5
Task 4	2	0.5	20

Example

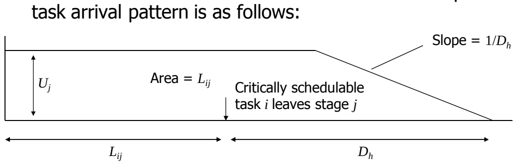
- Note that per-stage analysis will not yield the same results



Task	Stage1 Comp. Time	Per-stage deadline	Stage2 Comp. Time	Deadline
Task 1	0.1	0.5	0.1	1
Task 2	0.2	1	0.1	2
Task 3	1	2.5	0.125	5
Task 4	2	10	0.5	20

Summary: The Worst-Case Arrival Pattern

- Given *any* time-independent priority scheduling policy, the instantaneous utilization curve for the worst case aperiodic task arrival pattern is as follows:



L_{ij}: Maximum delay of task i on stage j
D_i: End-to-end relative deadline of task i
D_h: Largest end-to-end relative deadline of a task of a higher priority than task i (in the case of deadline-monotonic scheduling D_h is bounded by D_i)
U_j: Maximum utilization on stage j

Example: The Stage Delay Theorem

For deadline monotonic scheduling

- Consider the worst case pattern:

$$U(2L + 2D - UD)/2 = L$$

$$DU(1 - U/2) = (1 - U)L$$

$$\frac{L}{D} = \frac{U(1 - U/2)}{1 - U}$$