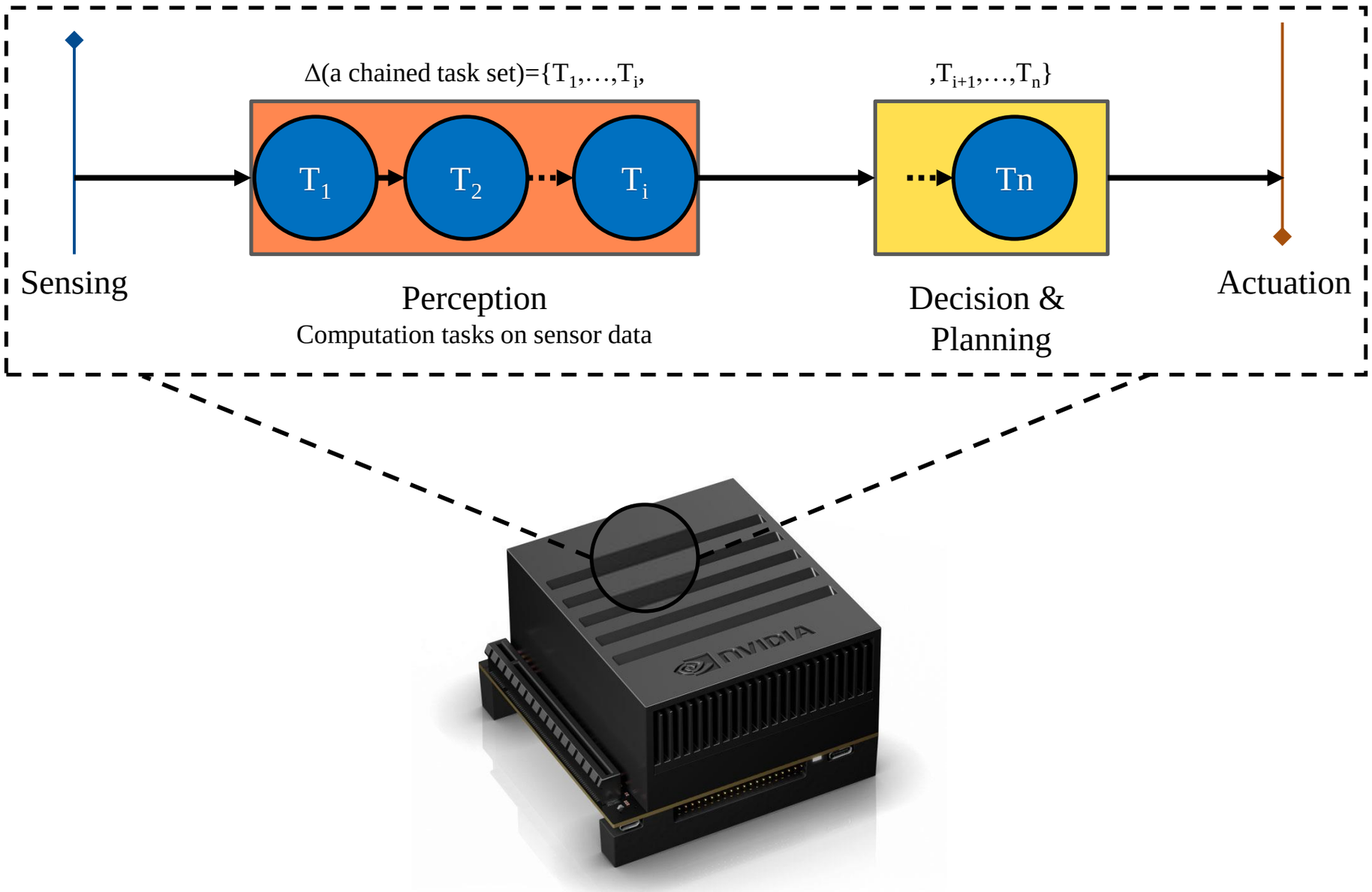
Predictable Data-driven Resource Management: an Implementation using Autoware on Autonomous Platforms

Soroush Bateni and Cong Liu

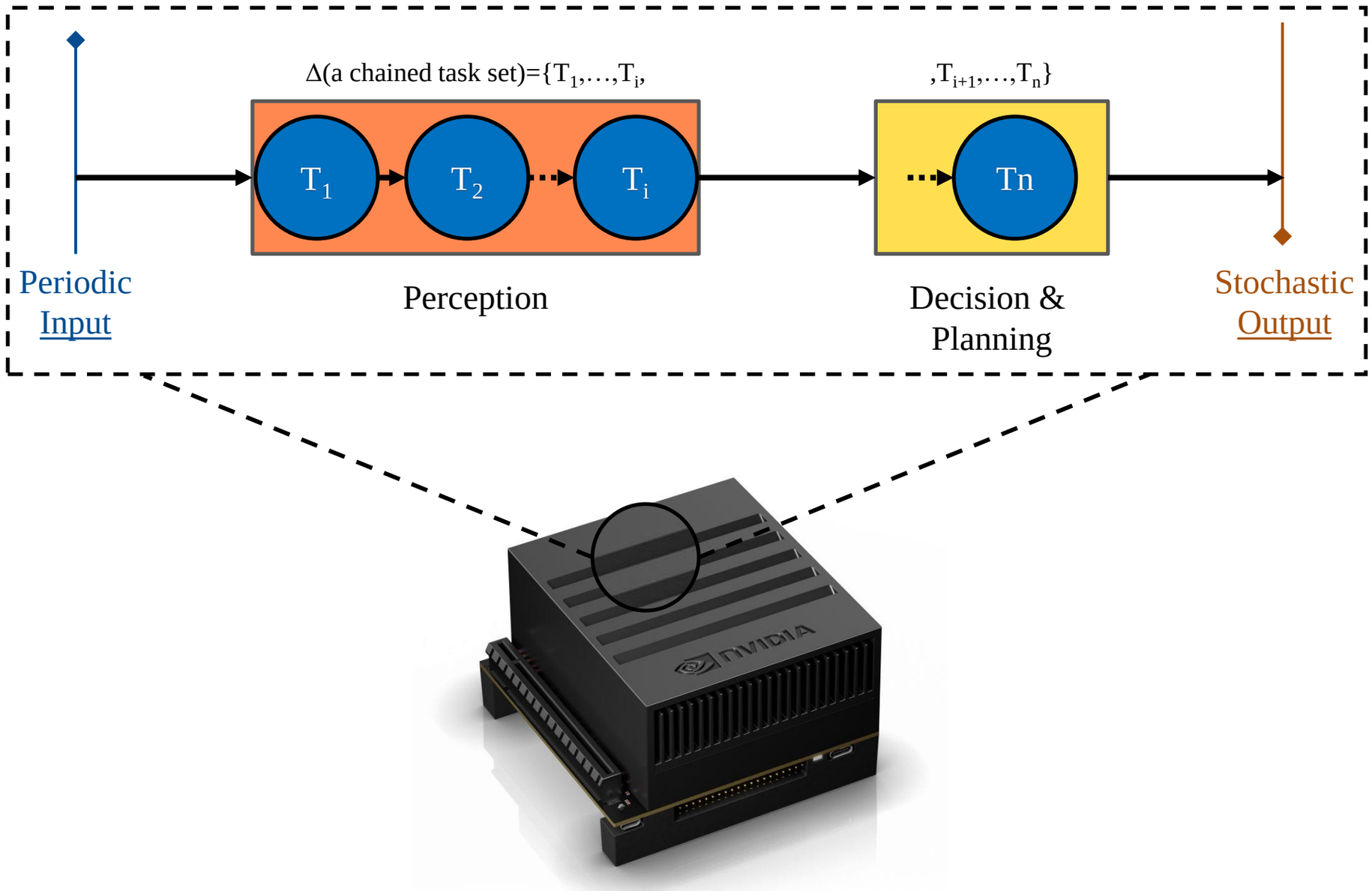
Department of Computer Science
Erik Jonsson School of Engineering & Computer Science
The University of Texas at Dallas

Autonomous Embedded Systems



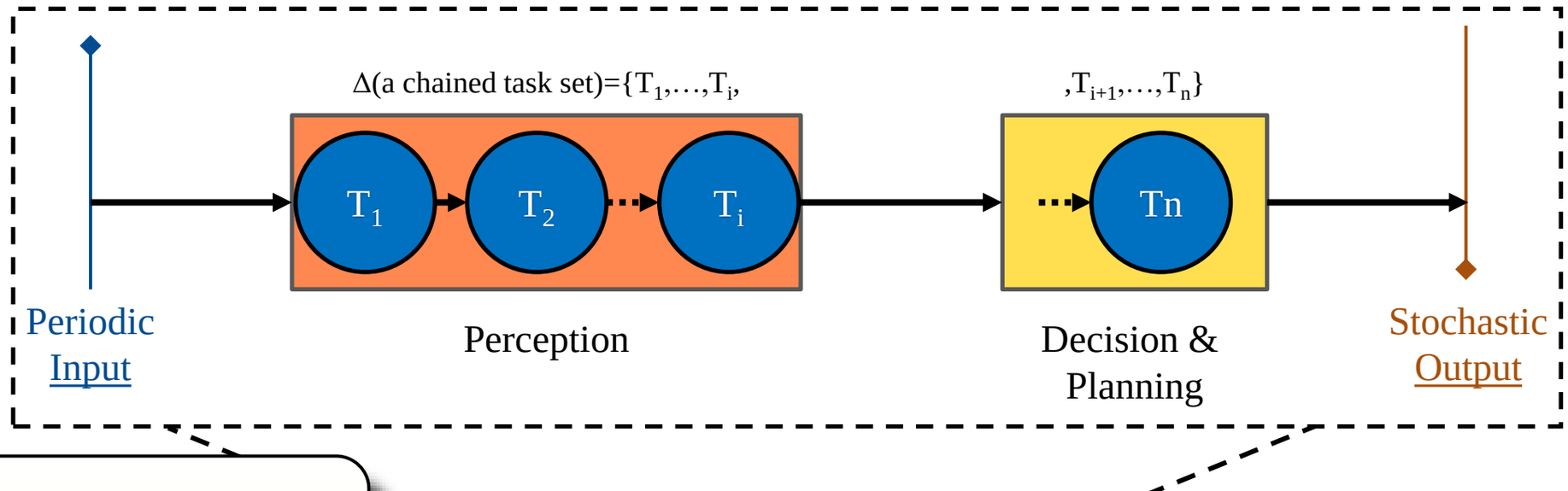
Embedded Platform

Autonomous Embedded Systems



Embedded Platform

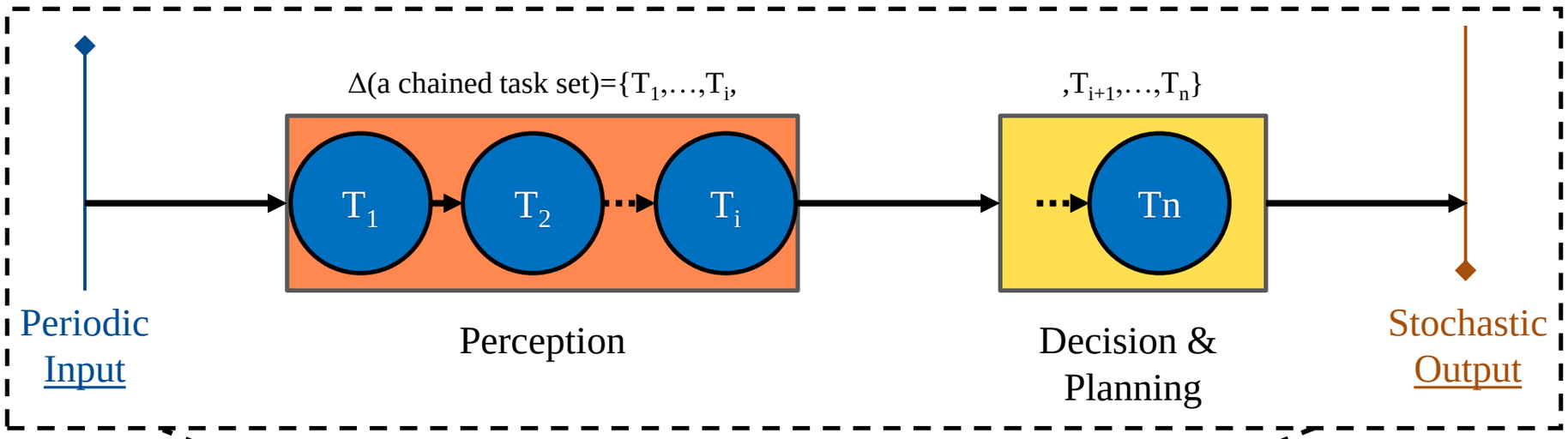
Autonomous Embedded Systems



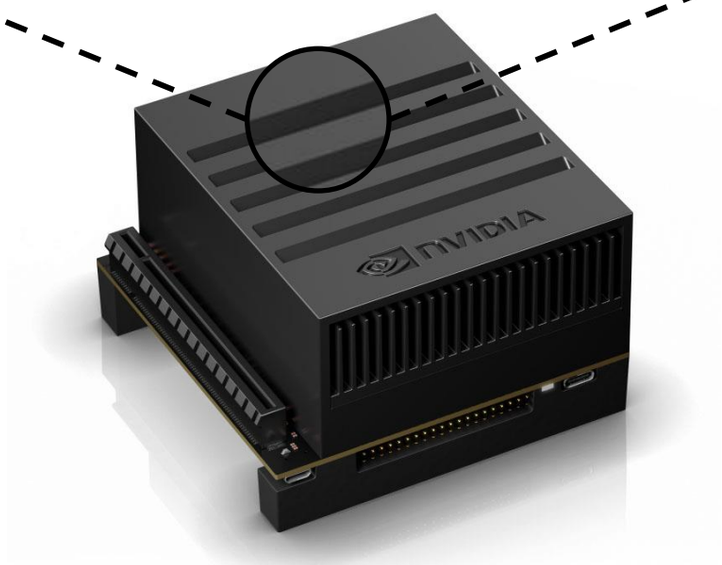
For example, a camera can release images every 33ms.

Embedded Platform

Autonomous Embedded Systems

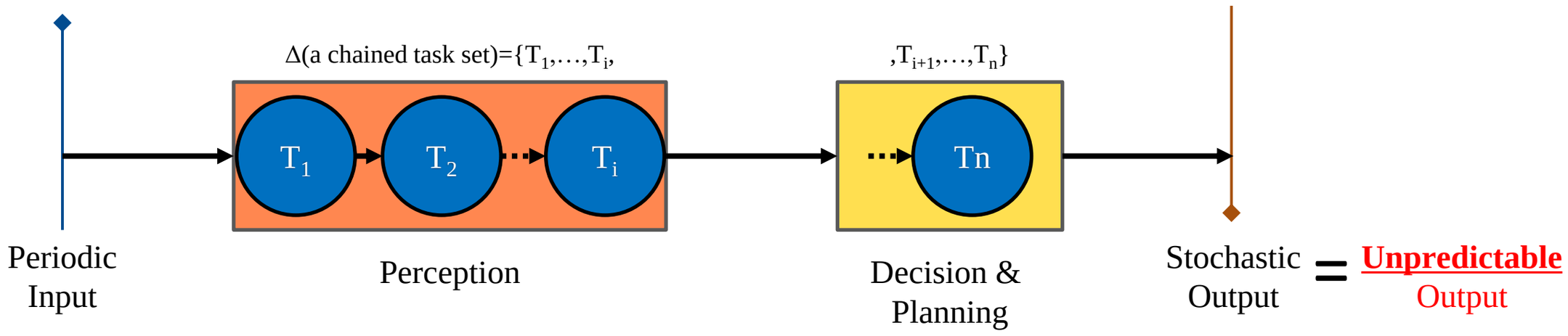


Output depends on when the task set finishes.



Embedded Platform

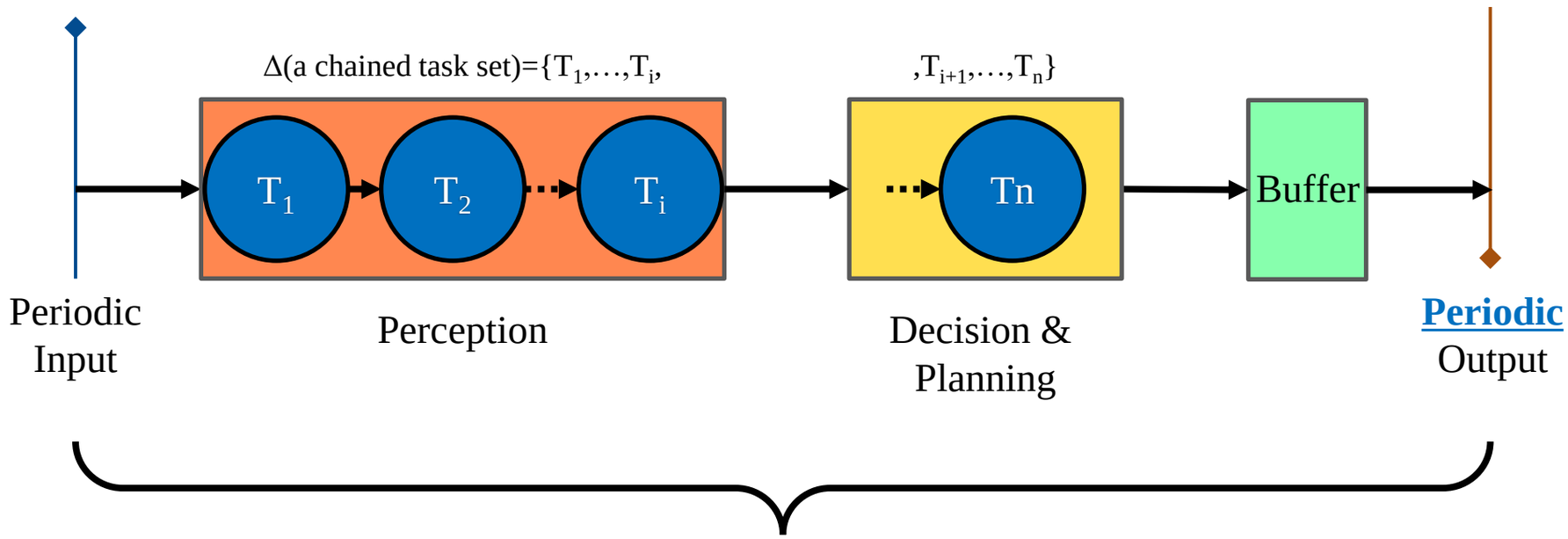
Logical Execution Time: Motivation



- Unpredictable output is undesirable in AES[1]
  - Harder to verify and certify
  - Harder to schedule
  - Can be classified as unexpected behavior pattern, specifically prohibited in most standards
    - A self-driving car sometimes takes 30ms to react and sometimes 25ms

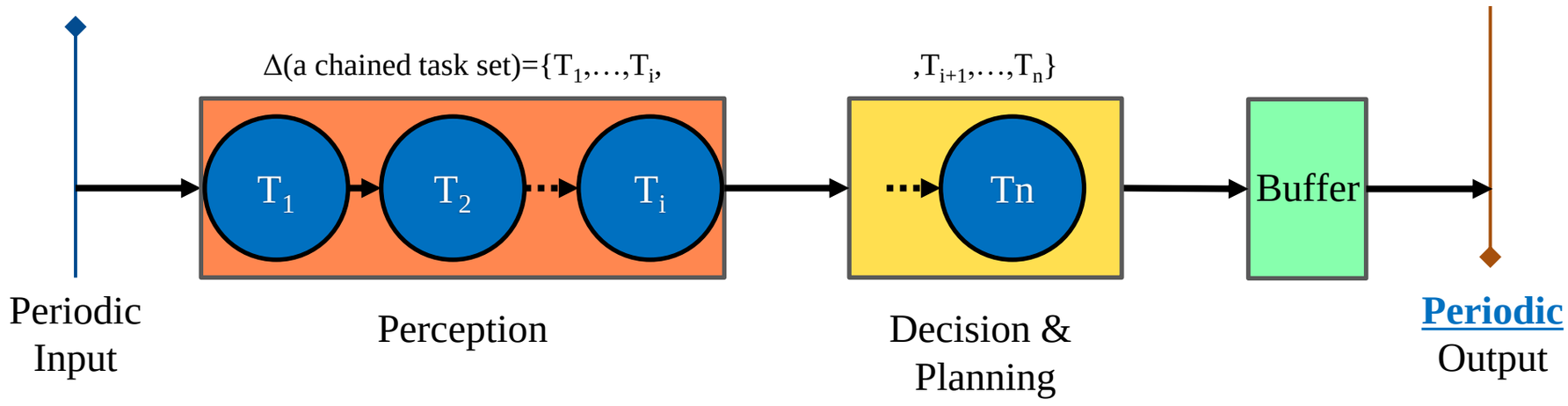
[1] Kirsch, Christoph M., and Ana Sokolova. "The logical execution time paradigm." Advances in Real-Time Systems. Springer, Berlin, Heidelberg, 2012. 103-120.

Logical Execution Time: Intuition



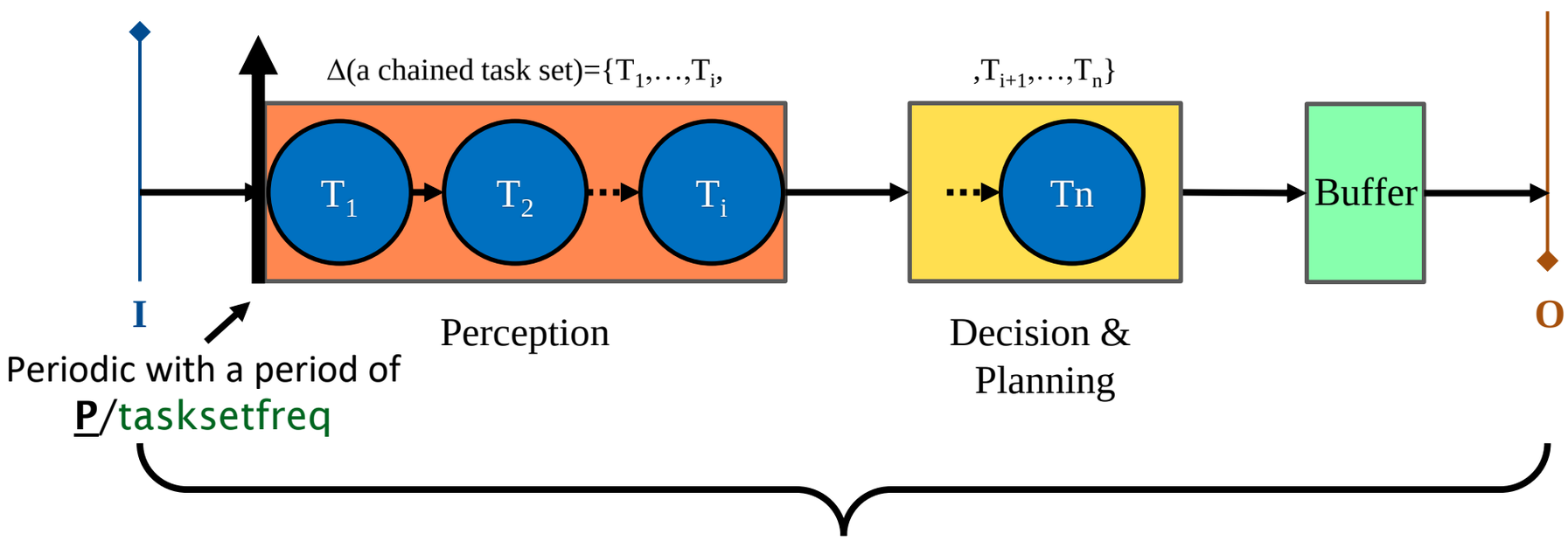
Task system always takes **LET (fixed time)** to execute.

Logical Execution Time: Intuition



The periodic output must be met and not violated.

Logical Execution Time: Example

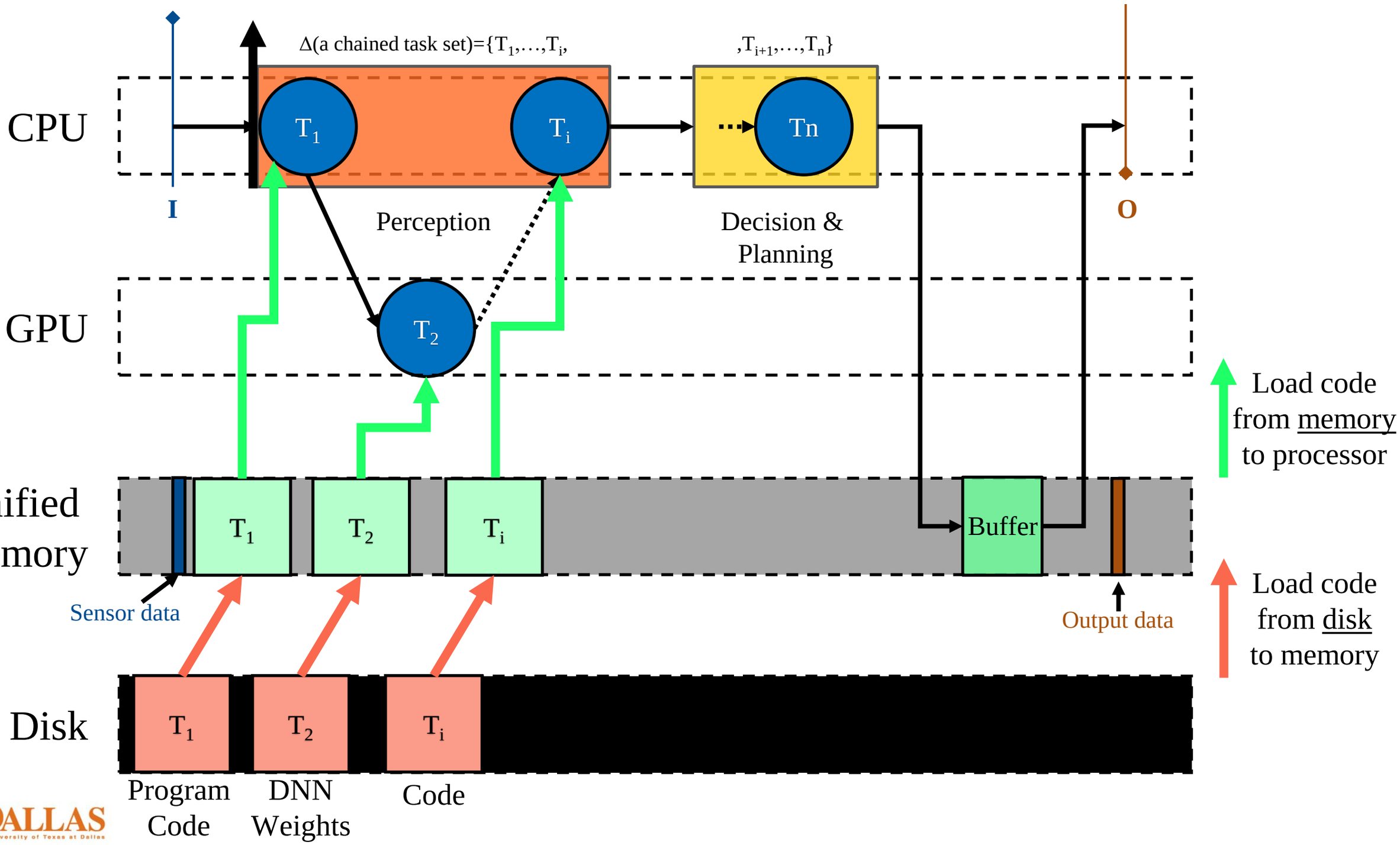


Example Abstract Code

mode m() Period 20 {
inputfreq do $P_I(d_I)$
tasksetfreq do $\Delta(d_\Delta)$
outputfreq do $P_O(d_O)$
}

- P/inputfreq: input interval (period)
- P/tasksetfreq: period of task set Δ
- P/outputfreq: output interval (period)

Integrated CPU/GPU Architecture



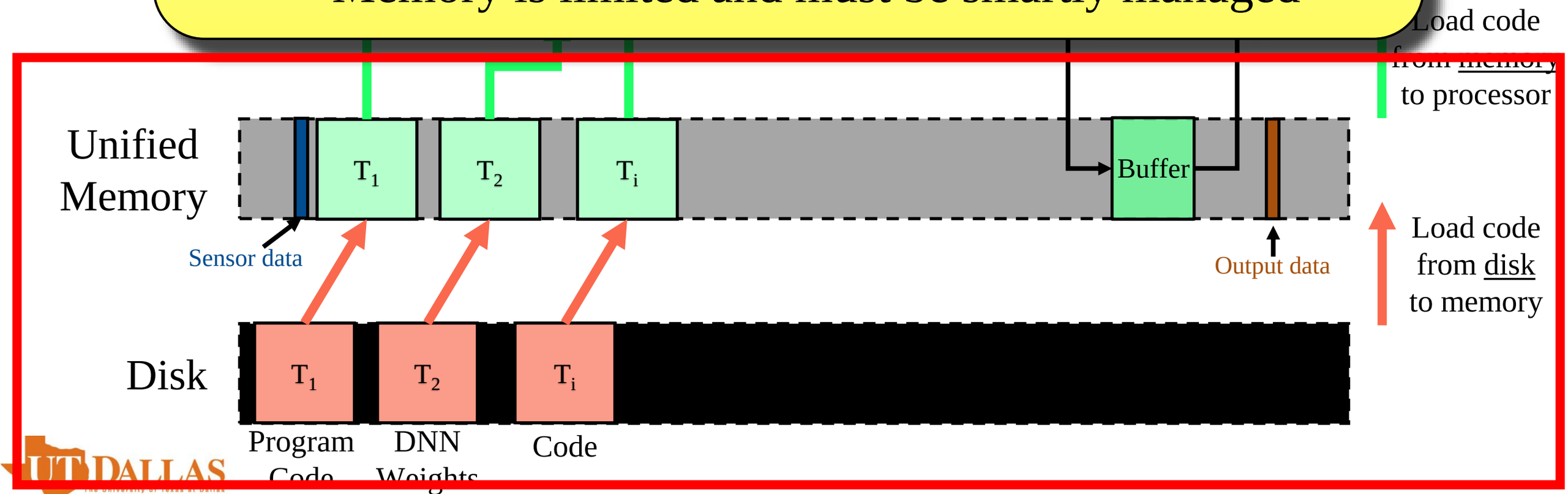
Integrated CPU/GPU Architecture

C

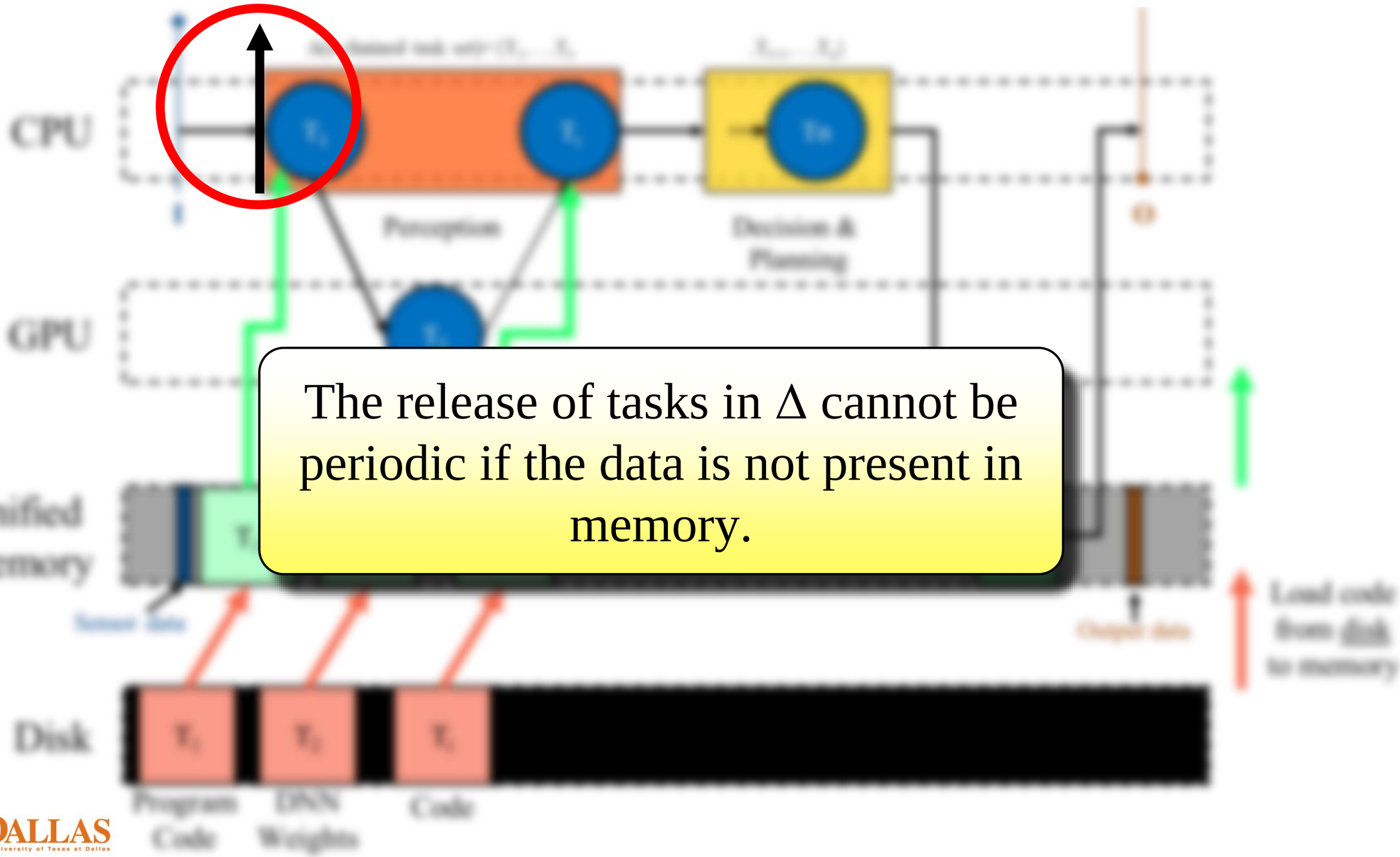
G

Focus

- Sensor data is small compared to program code.
- Loading code from memory to processor cache in a predictable manner has been extensively explored before.
- Memory is limited and must be smartly managed

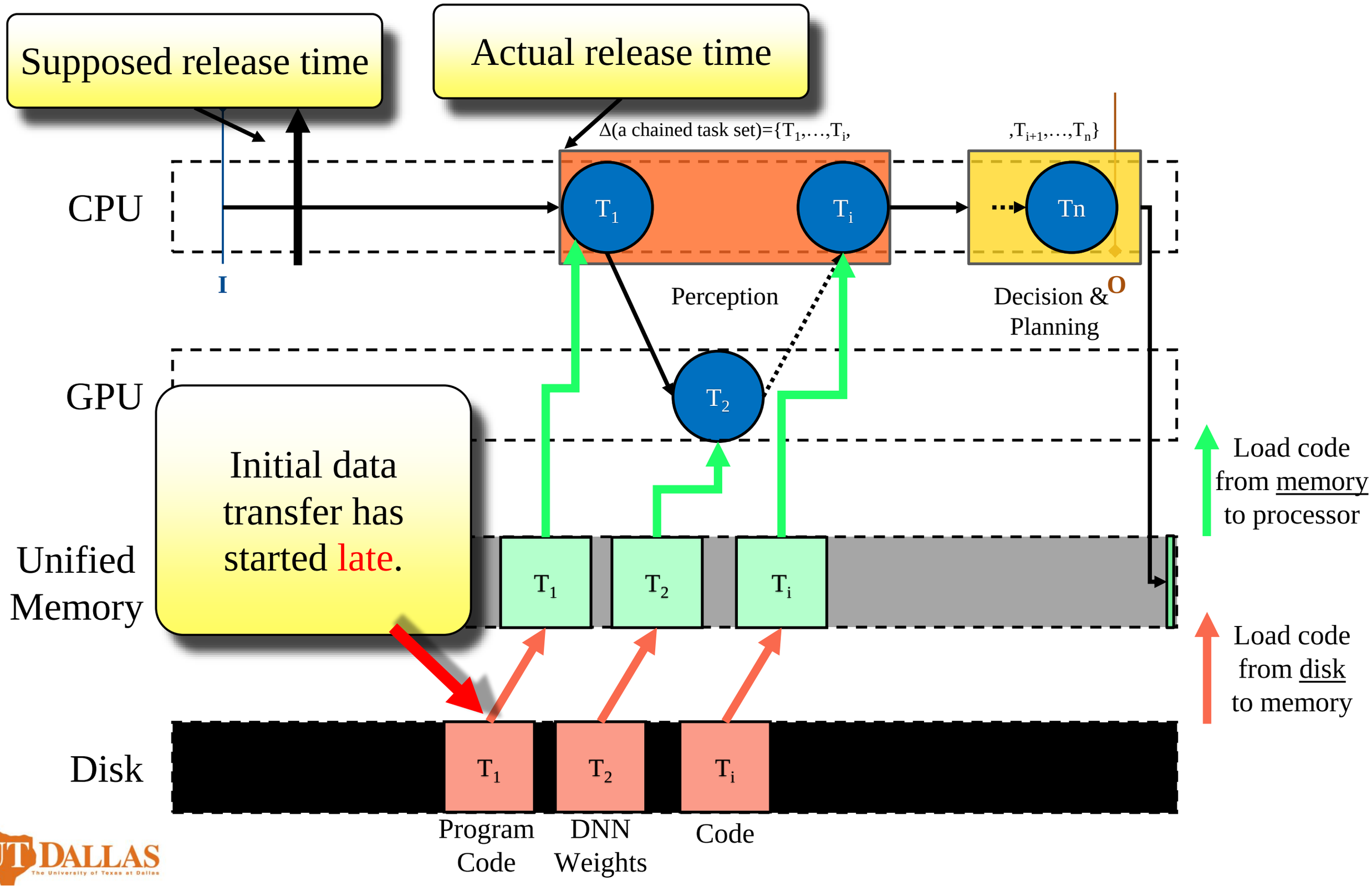


Logical Execution Time: Flaw under Integrated CPU/GPU Architecture

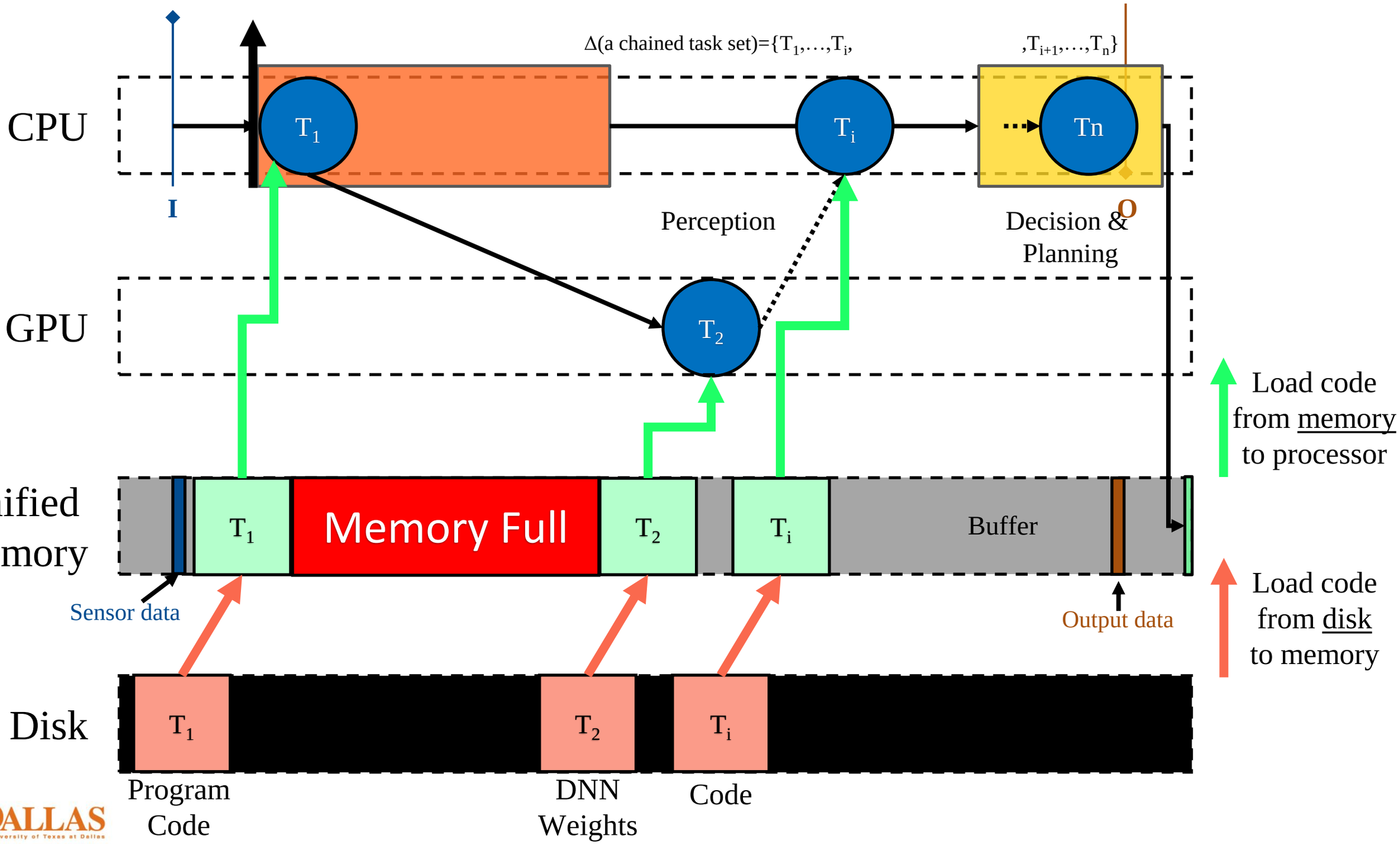


The release of tasks in Δ cannot be periodic if the data is not present in memory.

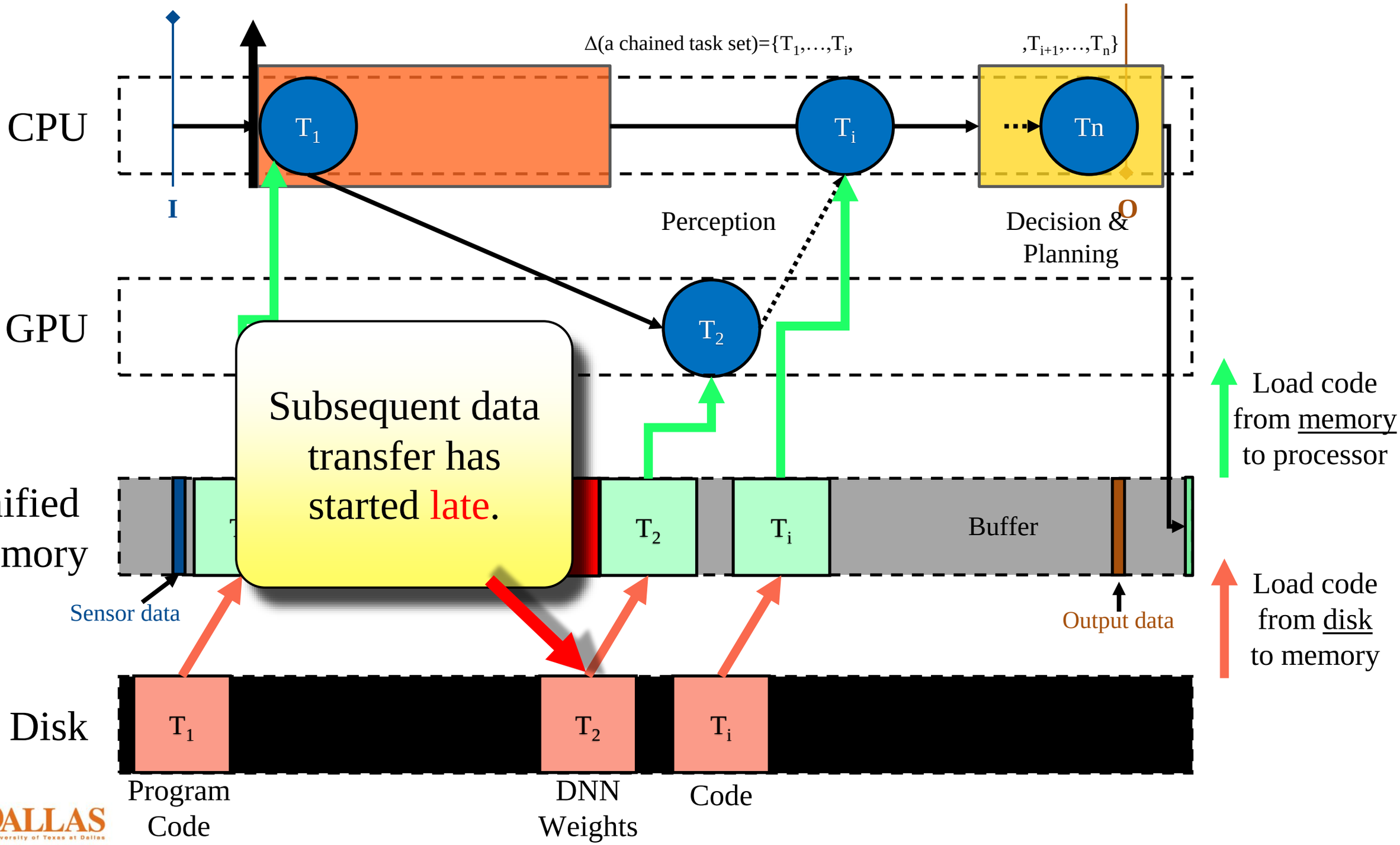
Temporal Data Availability Problem



Spatial Data Availability Problem



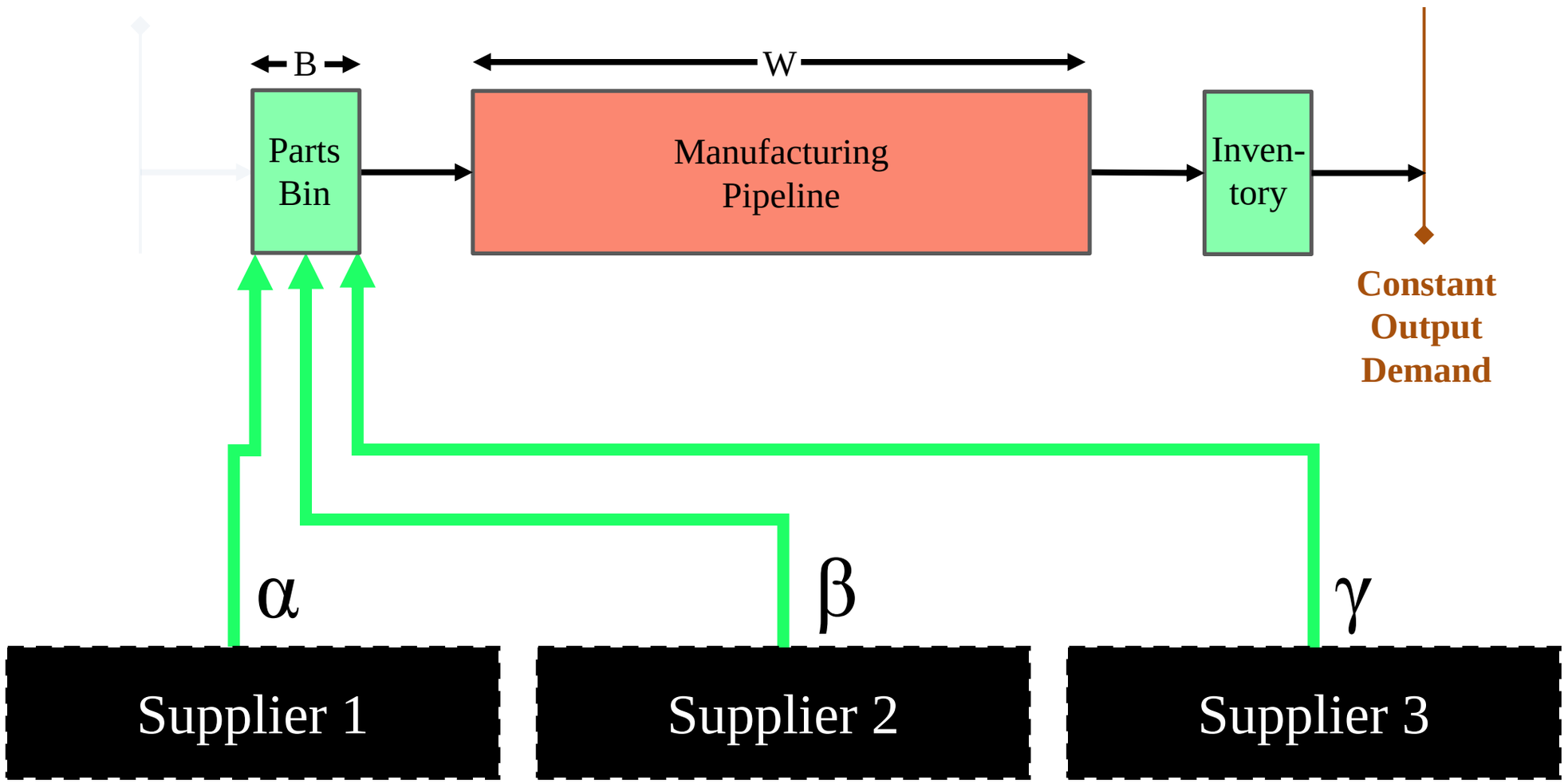
Spatial Data Availability Problem



To Summarize

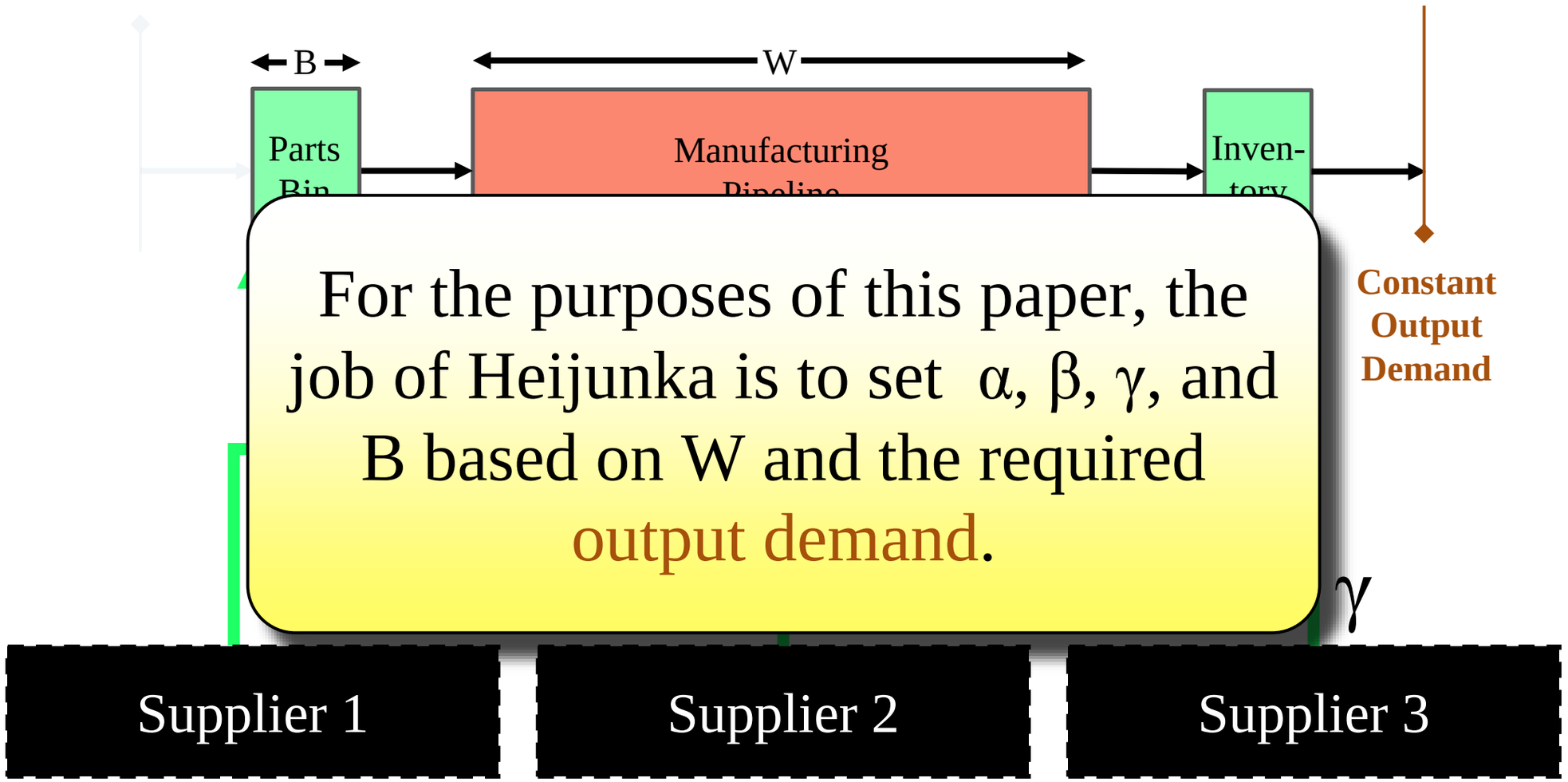
- LET is beneficial for autonomous embedded systems
  - Periodic Input
  - Predictable and verifiable output
- Has a flawed assumption: periodic task release
  - The flaw is due to ignorance to data (when memory is limited)
    - Temporal data availability
    - Spatial data availability
- Existing work
  - Has considered aperiodic input interval on LET (sporadic input events)
  - None to the best of our knowledge has considered aperiodic task release due to data and memory size limitation

Heijunka (Production Leveling)



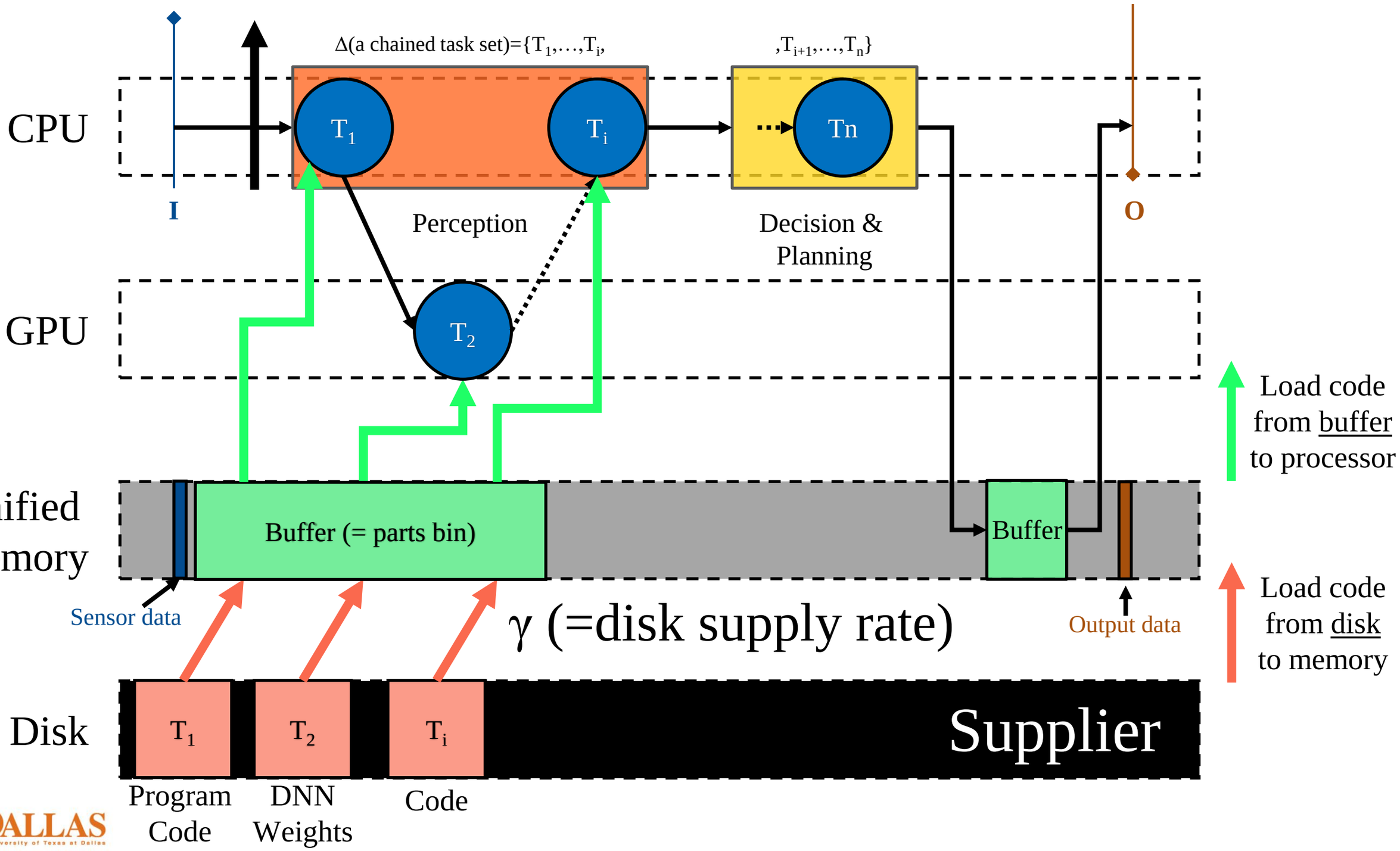
- α, β, and γ are supply rates for each supplier
- B is the size of the parts bin to store parts
- W is the worst-case time it takes to manufacture a car

Heijunka (Production Leveling)



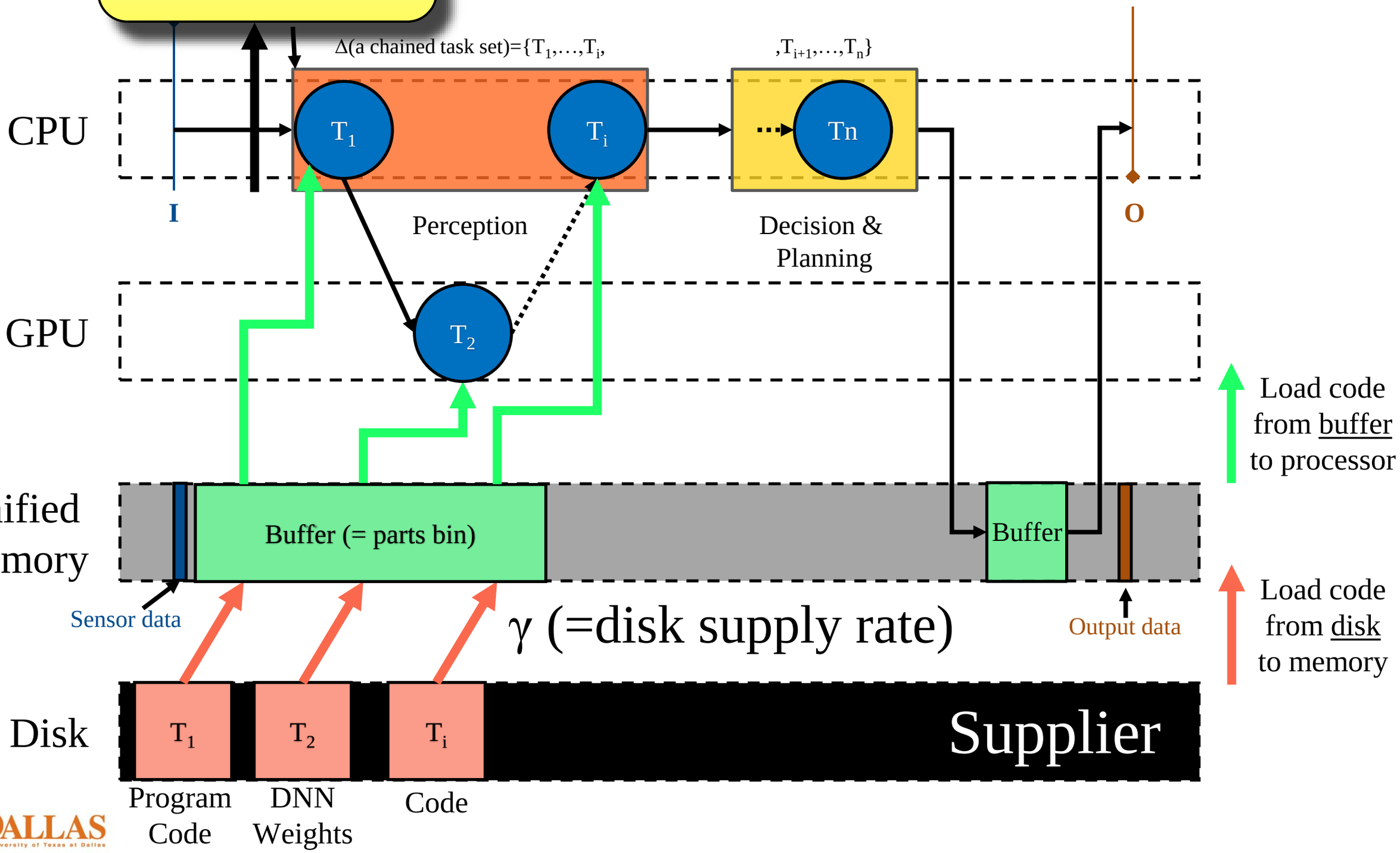
- α, β, and γ are supply rates for each supplier
- B is the size of the parts bin to store parts
- W is the worst-case time it takes to manufacture a car

LET-based System Model with Heijunka Expansion

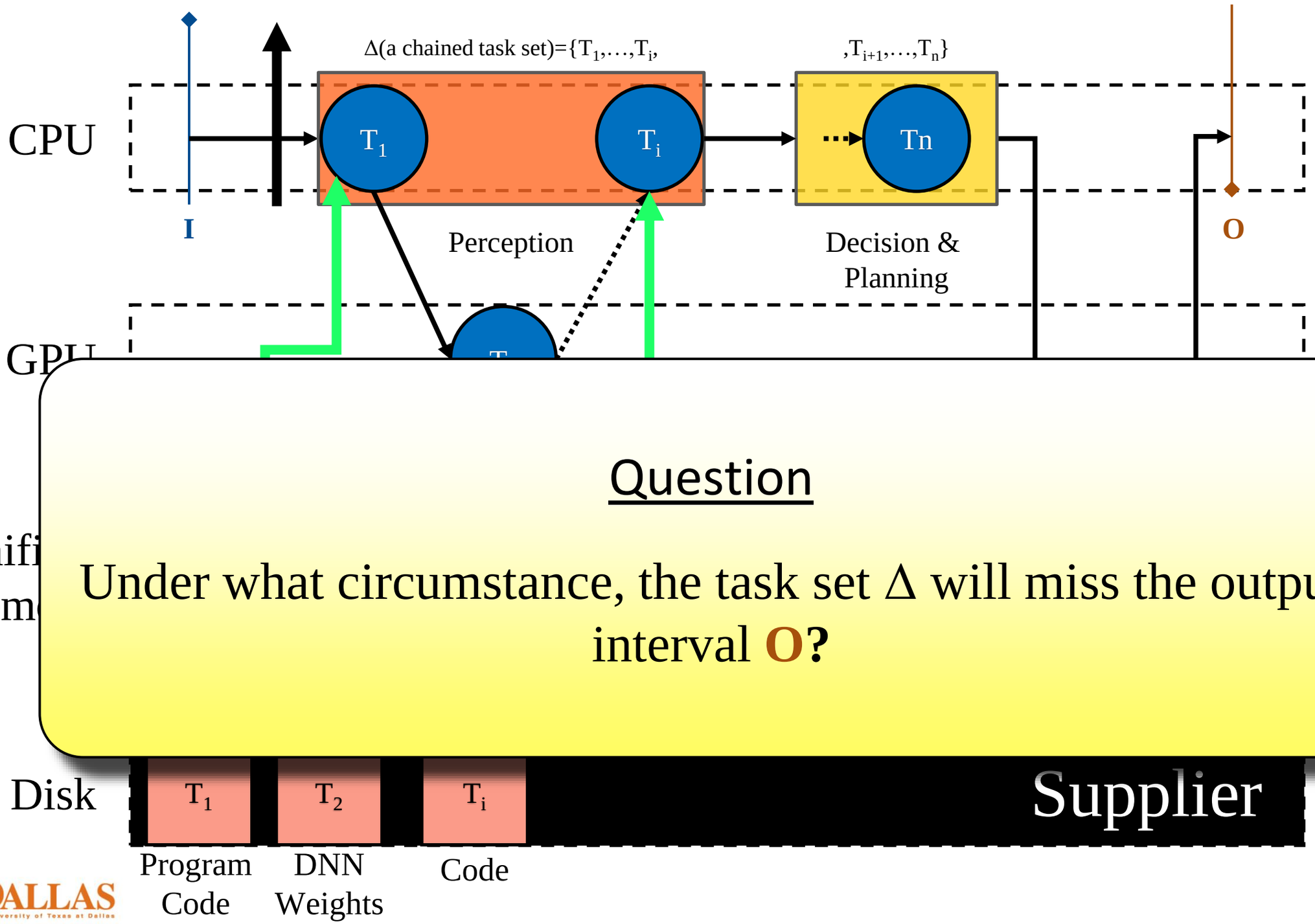


ystem Model with Heijunka Expansion

Task set is still sporadic.



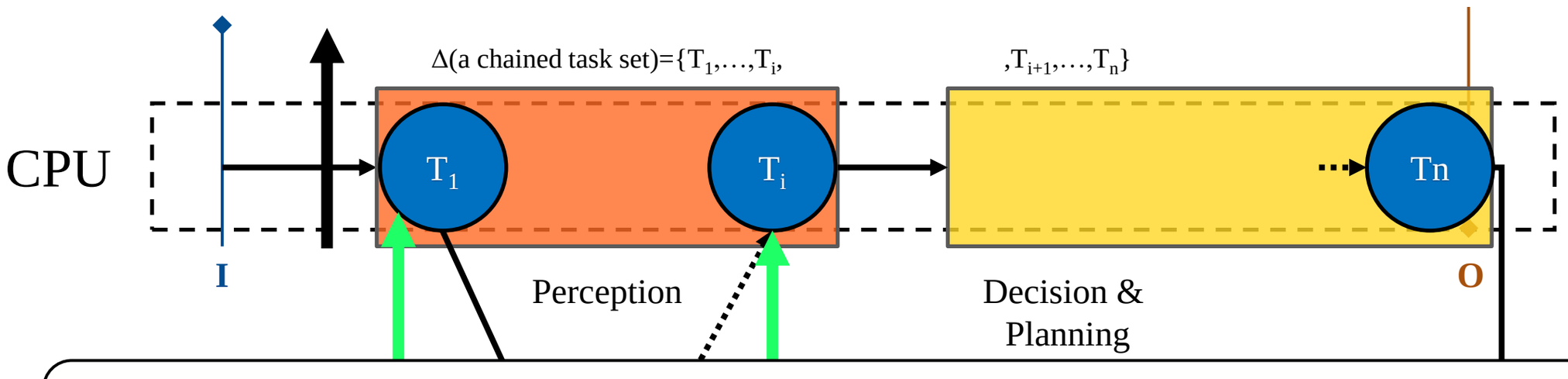
Ensuring Temporal Data Availability



Question

Under what circumstance, the task set Δ will miss the output interval **O**?

Ensuring Temporal Data Availability



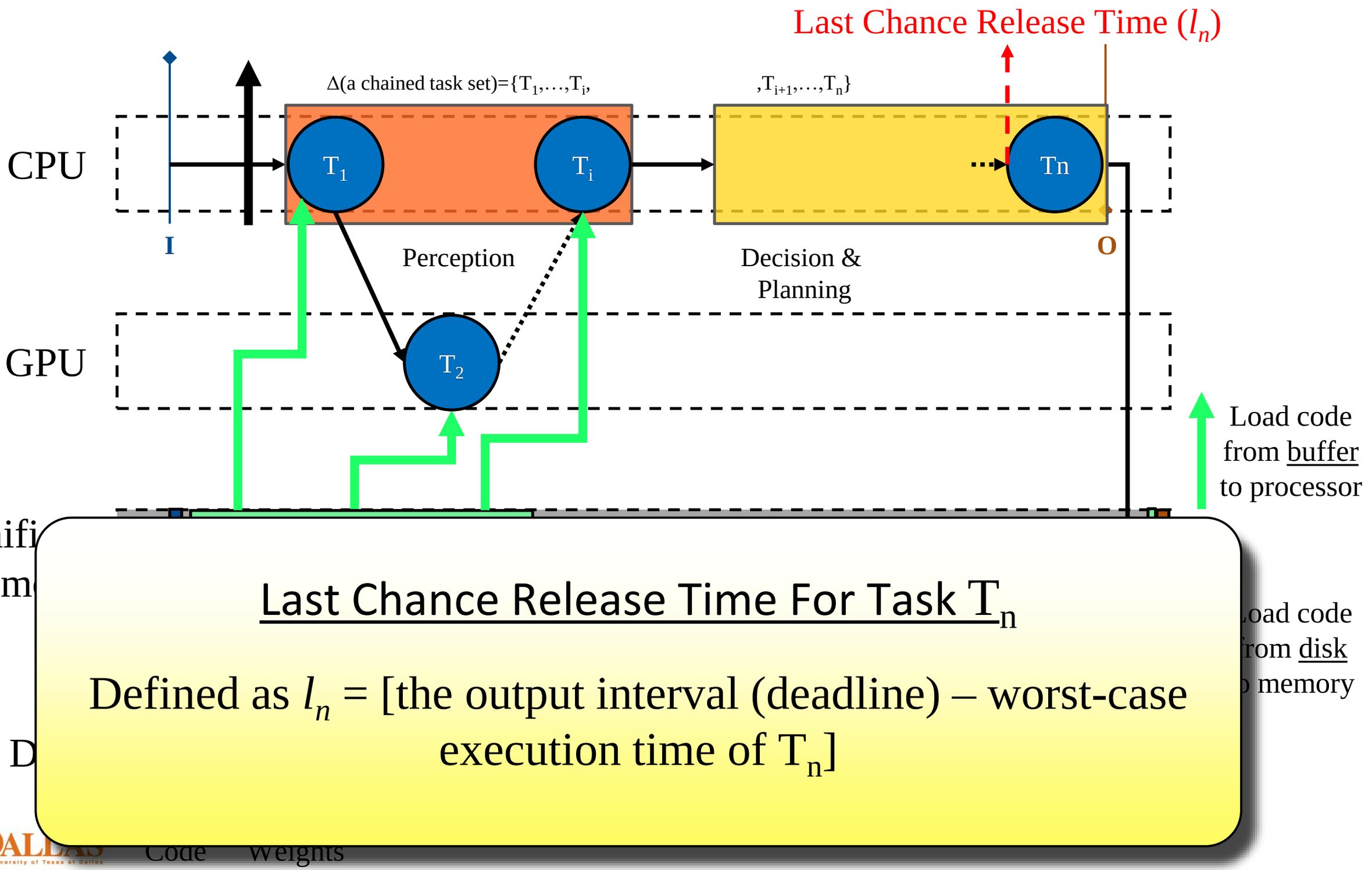
Answer

If T_n is released late.

Question

When is the latest time T_n can be released to not miss the deadline? In other words, what is the latest time instance when the data for T_n should be available?

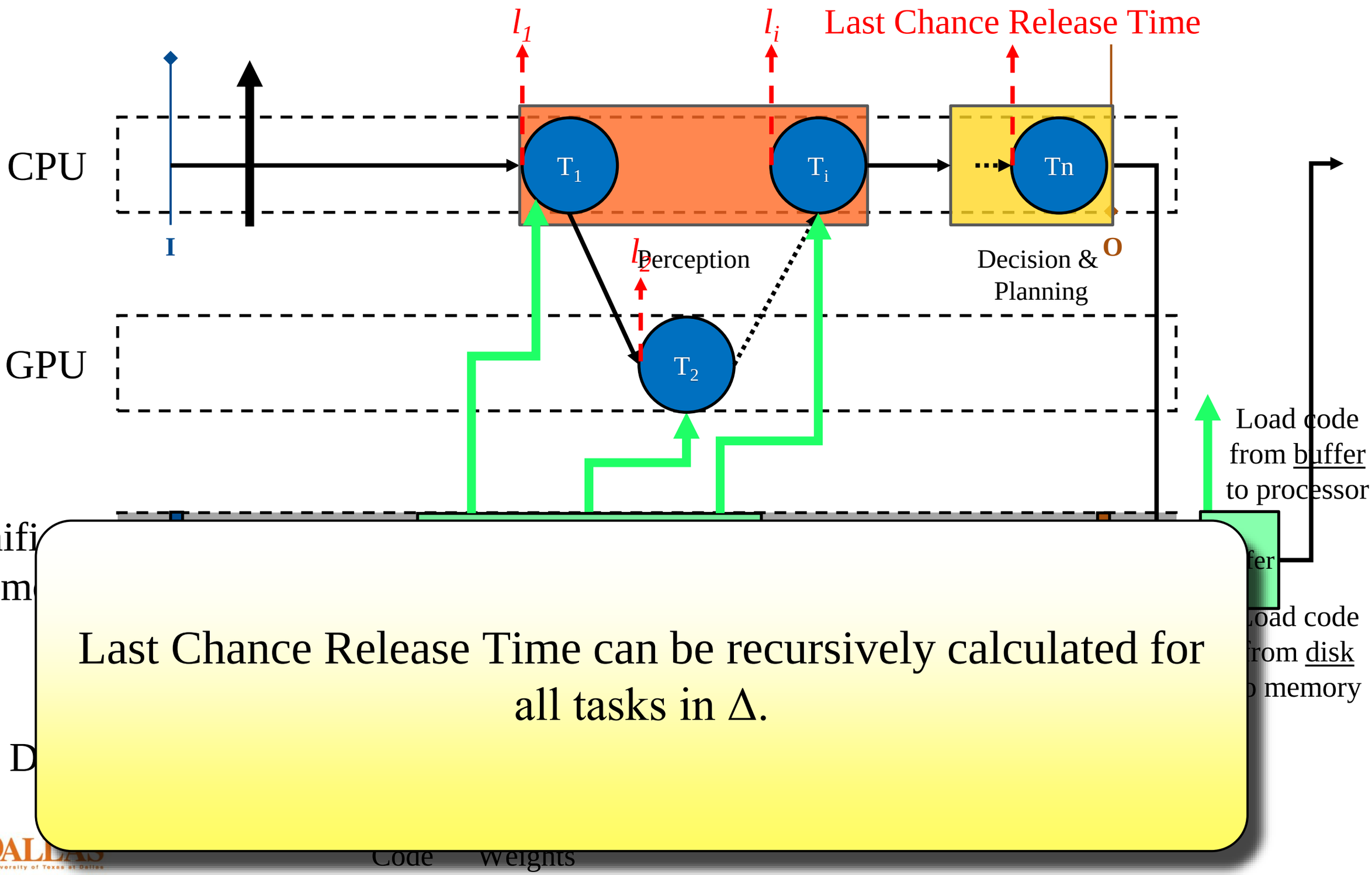
Ensuring Temporal Data Availability



Last Chance Release Time For Task T_n

Defined as l_n = [the output interval (deadline) – worst-case execution time of T_n]

Ensuring Temporal Data Availability



Last Chance Release Time can be recursively calculated for all tasks in Δ.

Ensuring Temporal Data Availability

Question

When should we start copying data from the disk?

C

GPU

T_2

Unified Memory

Buffer (= parts bin)

Sensor data

γ (=disk supply rate)

Load code from buffer to processor

Load code from disk to memory

Disk

T_1 T_2 T_i

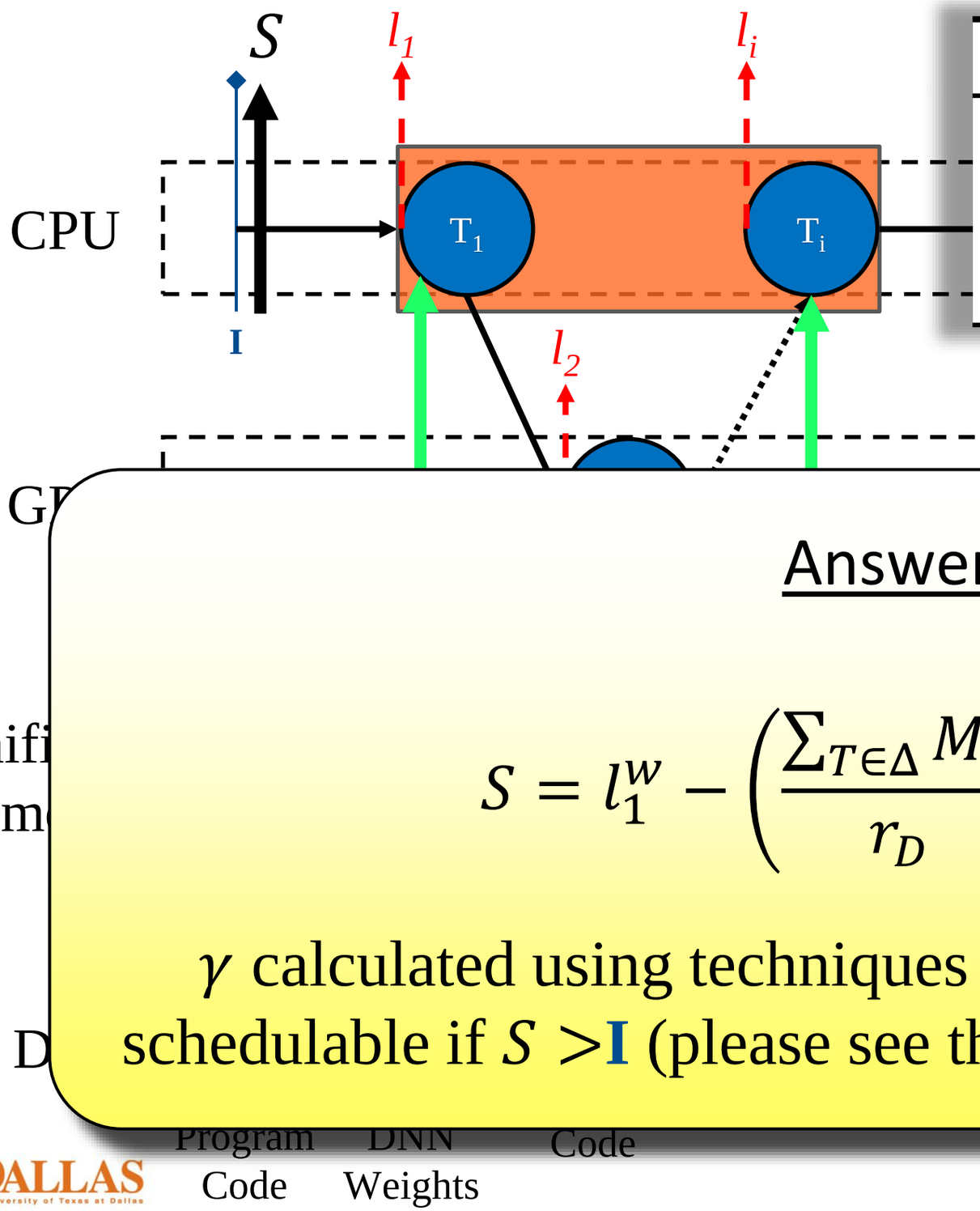
Supplier

Program Code

DNN Weights

Code

Ensuring Temporal Data Availability



Symbol	Description
M	Worst Case Memory Consumption
r_D	Disk transfer rate
γ	Supply rate calculated using Heijunka

Answer

$$S = l_1^w - \left(\frac{\sum_{T \in \Delta} M_T}{r_D}\right) + |\Delta| \cdot \gamma$$

γ calculated using techniques of Heijunka. System is schedulable if $S >$ **I** (please see the paper for more details).

Ensuring Spatial Data Availability

Question

What is the size of the Buffer?

C

Planning

GPU

T_2

Load code from buffer to processor

Unified Memory

Buffer (= parts bin)

Buffer

Load code from disk to memory

Sensor data

Disk

T_1 T_2 T_i

Supplier

Program Code

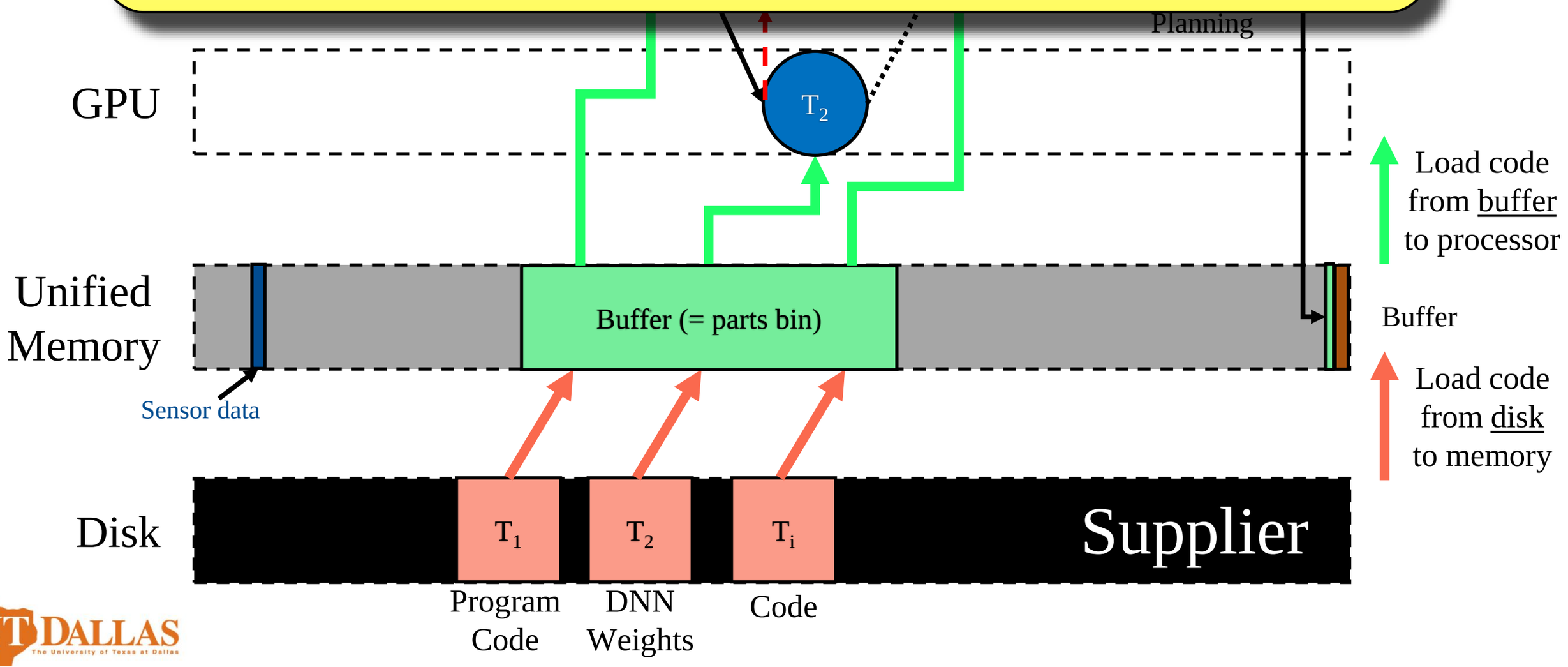
DNN Weights

Code

Ensuring Spatial Data Availability

Answer

$$|\text{Buffer}| = (l_1 - S) \cdot r_D + \sum_{T_i \in \Delta} |l_{i+1} - l_i| \cdot r_D$$



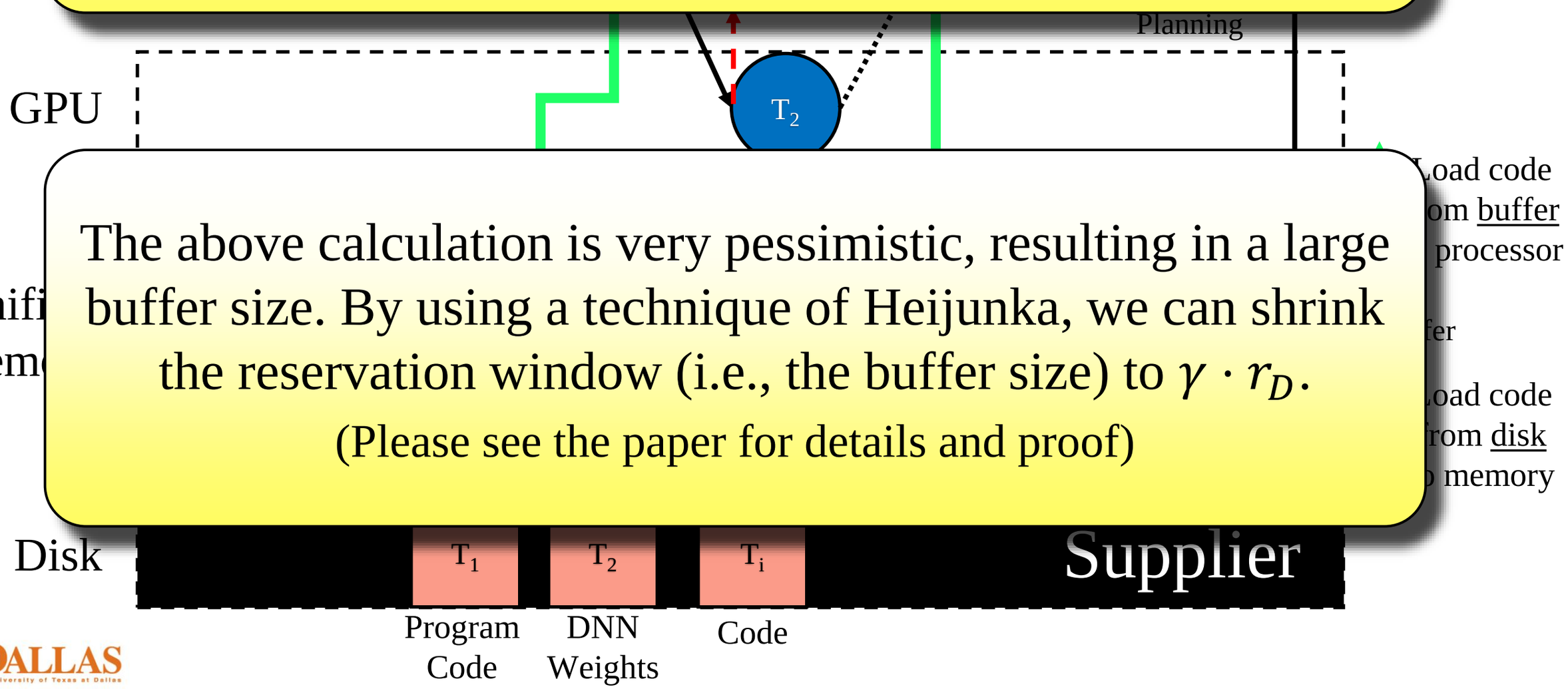
Ensuring Spatial Data Availability

Answer

$$|\text{Buffer}| = (l_1 - S) \cdot r_D + \sum_{T_i \in \Delta} |l_{i+1} - l_i| \cdot r_D$$

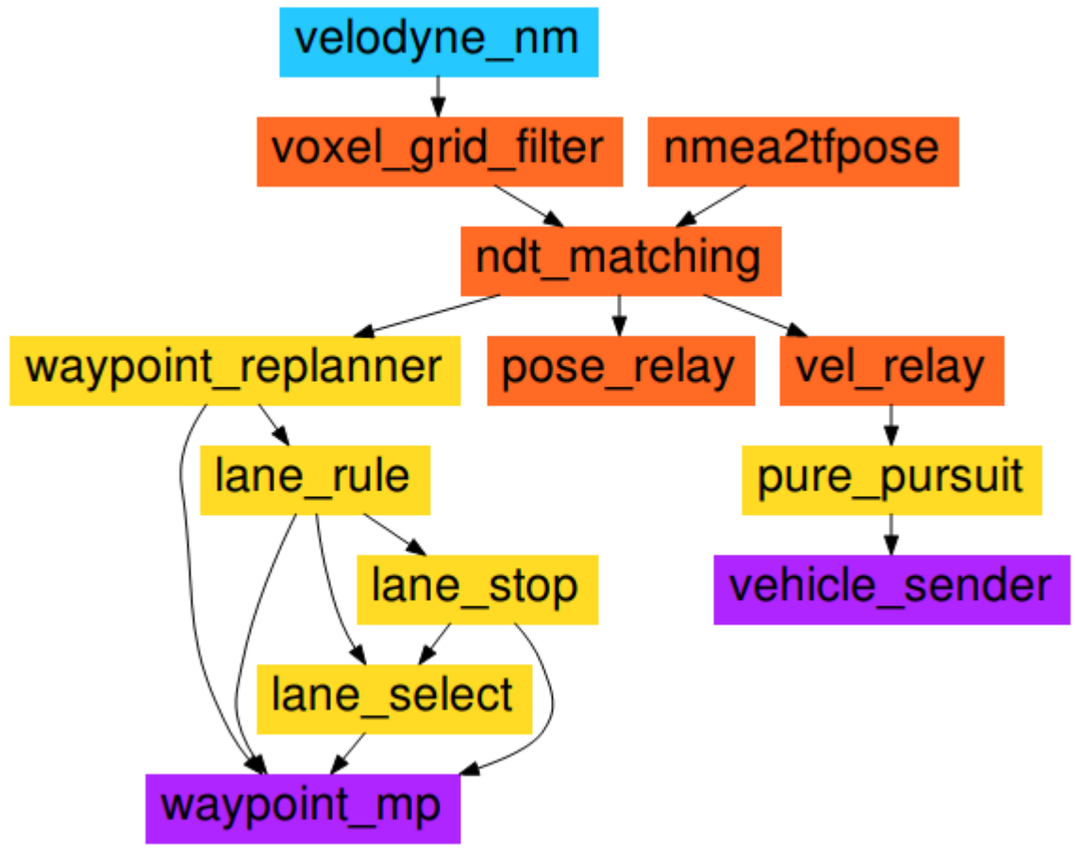
The above calculation is very pessimistic, resulting in a large buffer size. By using a technique of Heijunka, we can shrink the reservation window (i.e., the buffer size) to $\gamma \cdot r_D$.

(Please see the paper for details and proof)



Evaluation

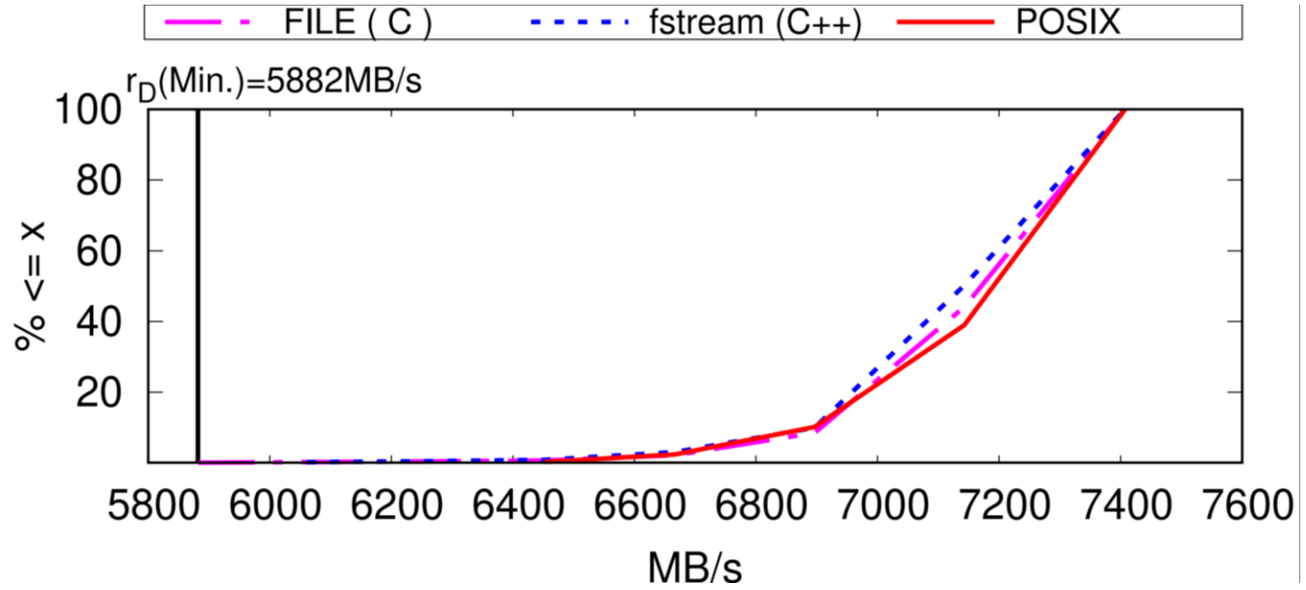
- We implement ResCue, a runtime solution with:
  - A dynamic data scheduler
  - A dynamic memory reservation
- Evaluation Platform
  - Autoware
    - Open-source autonomous driving full-stack software
  - NVIDIA Jetson AGX Xavier
    - 8-core CPU, 512-core Volta-based GPU, 16GB of RAM



Autoware

Results

- First, we calculate r_D for the on-board SSD of AGX Xavier



- $r_D = 5882$, the low 99.9th percentile

Calculated Parameters for Autoware

TABLE I: Components of Autoware sorted in order of execution, and their WCET, WCMC, and the value of l for each component under various input and output periods (LCR value is for the first period). All times are in milliseconds and memory size is in megabytes.

Color	Category	Process	WCET(ms)	WCMC(MB)	LCR(I=42,O=300)	LCR(I=33,O=200)	LCR(I=171,O=171)
	LIDAR (Sensing)	velodyne_nodelet_manager	2.64	56.01	206.100	106.100	77.100
	Downsampling (Perception)	voxel_grid_filter	3.96	52.75	208.740	108.740	79.740
	Localizer (Perception)	ndt_matching	45.37	2063.05 (static)	212.700	112.700	83.700
		nmea2tfpose	20.7	13.32	258.070	158.070	129.070
	Positioning (Perception)	pose_relay (Positioning) vel_relay (Velocity)	12.78	11.98 11.97	278.770	178.770	149.770
	Planning	waypoint_replanner	2.06	9.84	291.550	191.550	162.550
		lane_rule	1.2	28.29	293.610	193.610	164.610
		lane_stop	0.86	11.76	294.810	194.810	165.810
		lane_select	0.84	12.18	295.670	195.670	166.670
		pure_pursuit	1.31	10.09	296.510	196.510	167.510
	Output (rviz)	waypoint_marker_publisher	1.35	275.1	297.820	197.820	168.820
		vehicle_sender	0.83	9.48	299.170	199.170	170.170
Summary		SUM	93.9	483.2			
Parameters		S γ			129.994 0.83	29.994 0.83	0.994 0.83
Memory Reservation					9.76	9.76	9.76
Deadline Miss (% over $1e09$ steps)					0%	0%	0%

Calculated Parameters for Autoware

TABLE I: Components of Autoware sorted in order of execution, and their WCET, WCMC, and the value of λ for each component under various input and output periods (LCR value is for the first period). All times are in m

Different input and output periods

Scenarios: S1 S2 S3

Color	Category	Process	WCET(ms)		LCR(I=42,O=300)	LCR(I=33,O=200)	LCR(I=171,O=171)
	LIDAR (Sensing)	velodyne_nodelet_manager	2.64	56.01	206.100	106.100	77.100
	Downsampling (Perception)	voxel_grid_filter	3.96	52.75	208.740	108.740	79.740
	Localizer (Perception)	ndt_matching	45.37	2063.05 (static)	212.700	112.700	83.700
		nmea2tfpose	20.7	13.32	258.070	158.070	129.070
	Positioning (Perception)	pose_relay (Positioning) vel_relay (Velocity)	12.78	11.98 11.97	278.770	178.770	149.770
	Planning	waypoint_replanner	2.06	9.84	291.550	191.550	162.550
		lane_rule	1.2	28.29	293.610	193.610	164.610
		lane_stop	0.86	11.76	294.810	194.810	165.810
		lane_select	0.84	12.18	295.670	195.670	166.670
		pure_pursuit	1.31	10.09	296.510	196.510	167.510
	Output (rviz)	waypoint_marker_publisher	1.35	275.1	297.820	197.820	168.820
		vehicle_sender	0.83	9.48			
Summary		SUM	93.9	483.2			
Parameters		S			129.994	29.994	0.994
		γ			0.83	0.83	0.83
Memory Reservation					9.76	9.76	9.76
Deadline Miss (% over $1e09$ steps)					0%	0%	0%

Values for S and γ

Calculated Parameters for Autoware

TABLE I: Components of Autoware sorted in order of execution, and their WCET, WCMC, and the value of λ for each component under various input and output periods (LCR value is for the first period). All times are in n

Calculated LCR for each task

Color	Category	Process	WCET(ms)	WCMC(MB)	LCR(I=42,O=300)	LCR(I=33,O=200)	LCR(I=171,O=171)
	LIDAR (Sensing)	velodyne_nodelet_manager	2.64	56.01	206.100	106.100	77.100
	Downsampling (Perception)	voxel_grid_filter	3.96	52.75	208.740	108.740	79.740
	Localizer (Perception)	ndt_matching	45.37	2063.05 (static)	212.700	112.700	83.700
		nmea2tfpose	20.7	13.32	258.070	158.070	129.070
	Positioning (Perception)	pose_relay (Positioning) vel_relay (Velocity)	12.78	11.98 11.97	278.770	178.770	149.770
	Planning	waypoint_replanner	2.06	9.84	291.550	191.550	162.550
		lane_rule	1.2	28.29	293.610	193.610	164.610
		lane_stop	0.86	11.76	294.810	194.810	165.810
		lane_select	0.84	12.18	295.670	195.670	166.670
		pure_pursuit	1.31	10.09	296.510	196.510	167.510
	Output (rviz)	waypoint_marker_publisher	1.35	275.1	297.820	197.820	168.820
		vehicle_sender	0.83	9.48	299.170	199.170	170.170
Summary		SUM	93.9	483.2			
Parameters		S γ			129.994 0.83	29.994 0.83	0.994 0.83
Memory Reservation					9.76	9.76	9.76
Deadline Miss (% over $1e09$ steps)					0%	0%	0%

Response Time Analysis

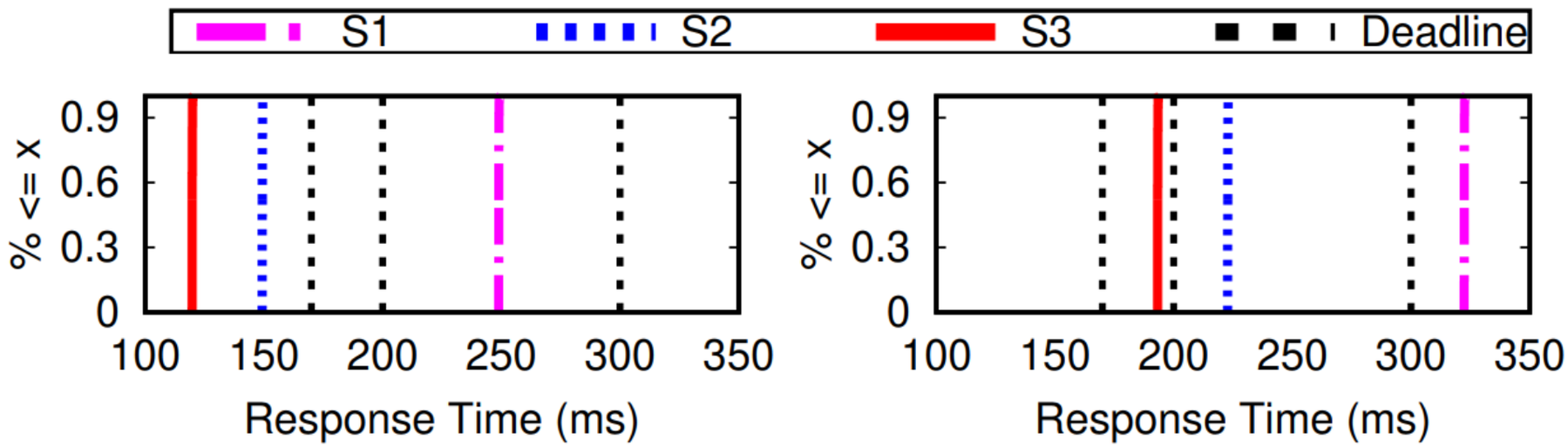
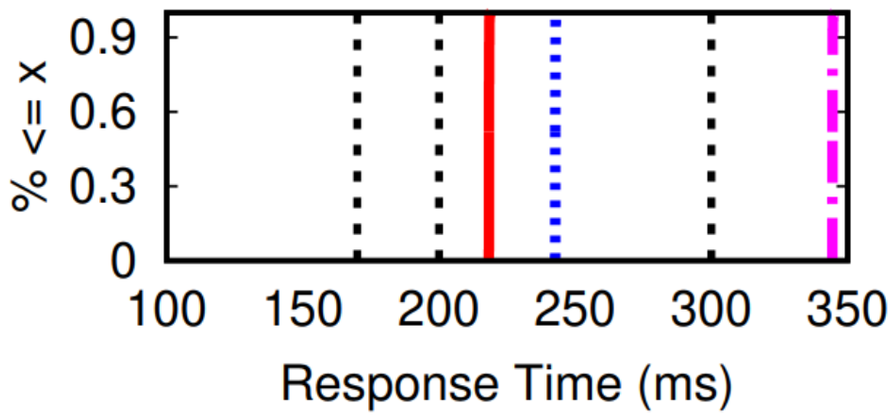


Fig. 9: Response time under ResCue with the three scenarios.

Fig. 10: Response time under ResCue with no data scheduler.



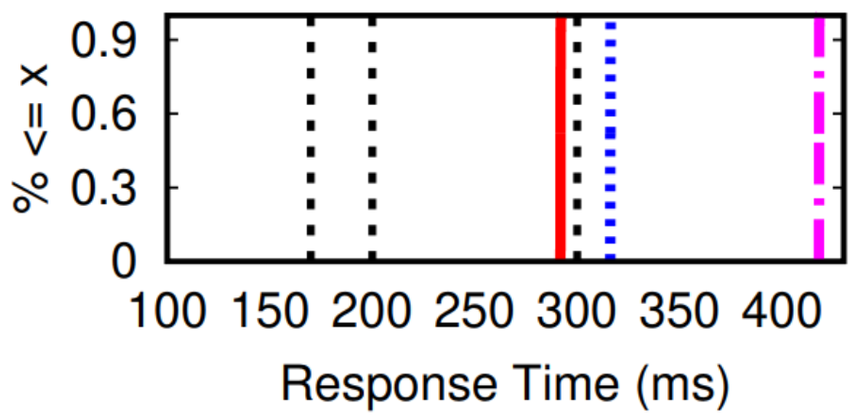
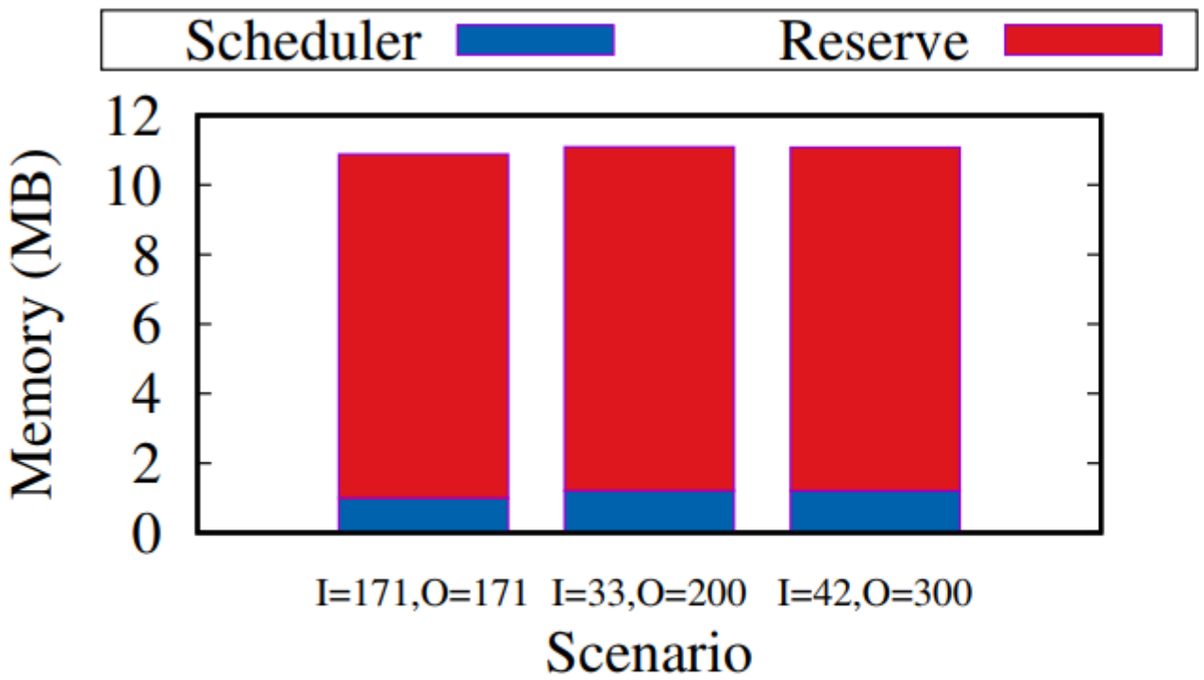


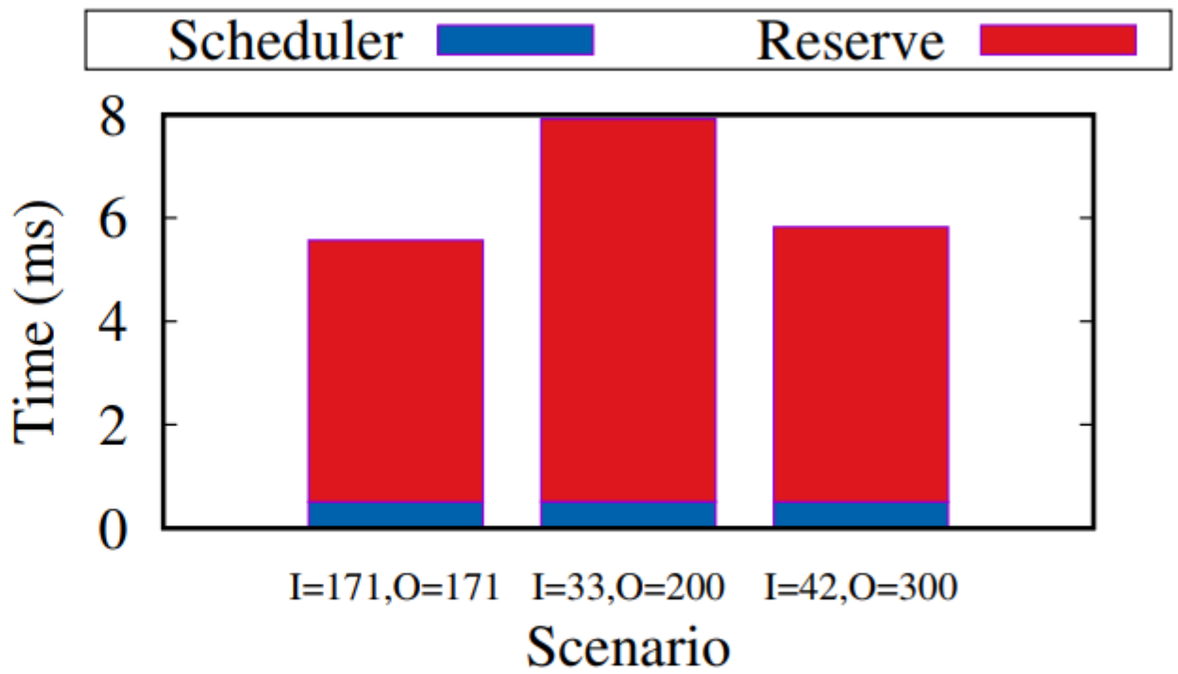
Fig. 11: Response time under ResCue with reservation disabled.

Fig. 12: Response time with all features of ResCue disabled.

Overhead



Memory consumption.



Execution time.

- Overhead is relatively negligible compared to savings

Conclusion

- LET cannot be applied directly to integrated CPU/GPU architectures in Autonomous Embedded Systems
  - Memory is limited
  - Task release times can be sporadic
- We found inspiration from Heijunka, with goals similar to LET (i.e., meeting a constant output) but with a limited supply of parts and storage (e.g., limited disk transfer rate and small unified memory)
- We presented a data-aware expanded system model based on a combination of LET and Heijunka
- Based on our model, we presented ResCue, a runtime solution that provides:
  - Dynamic data scheduling to ensure temporal data availability (e.g., by calculating γ and S)
  - Dynamic memory reservation to ensure spatial data availability (e.g., by calculating the size of the buffer and handling the reservation)