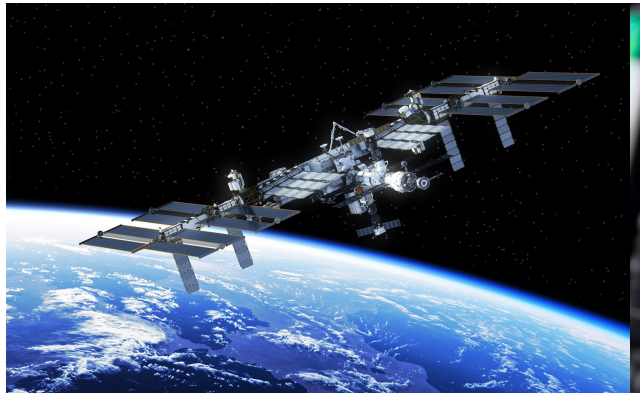
Real-Time Stream Processing in Embedded Systems

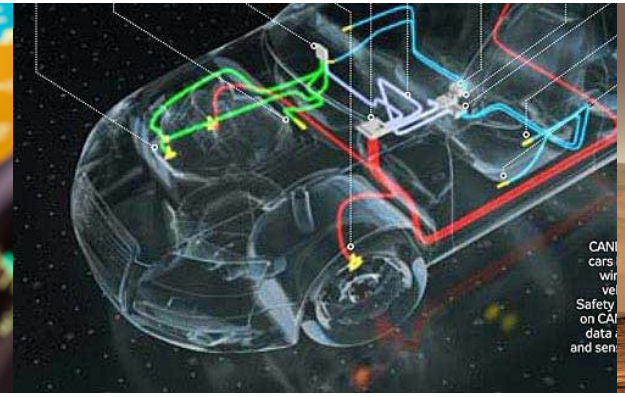
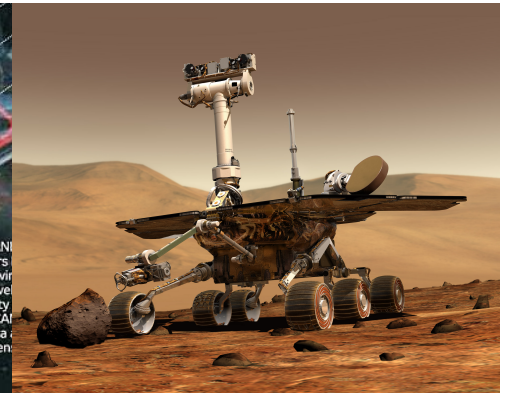
Haitao Mei

Supervisors: Prof. Andy Wellings, Dr. Ian Gray

What is Real-Time Systems

- Any system which has to respond to externally generated input within a finite and specified period
- In hard real-time systems, failures might result in the crush of an airplane



Motivation

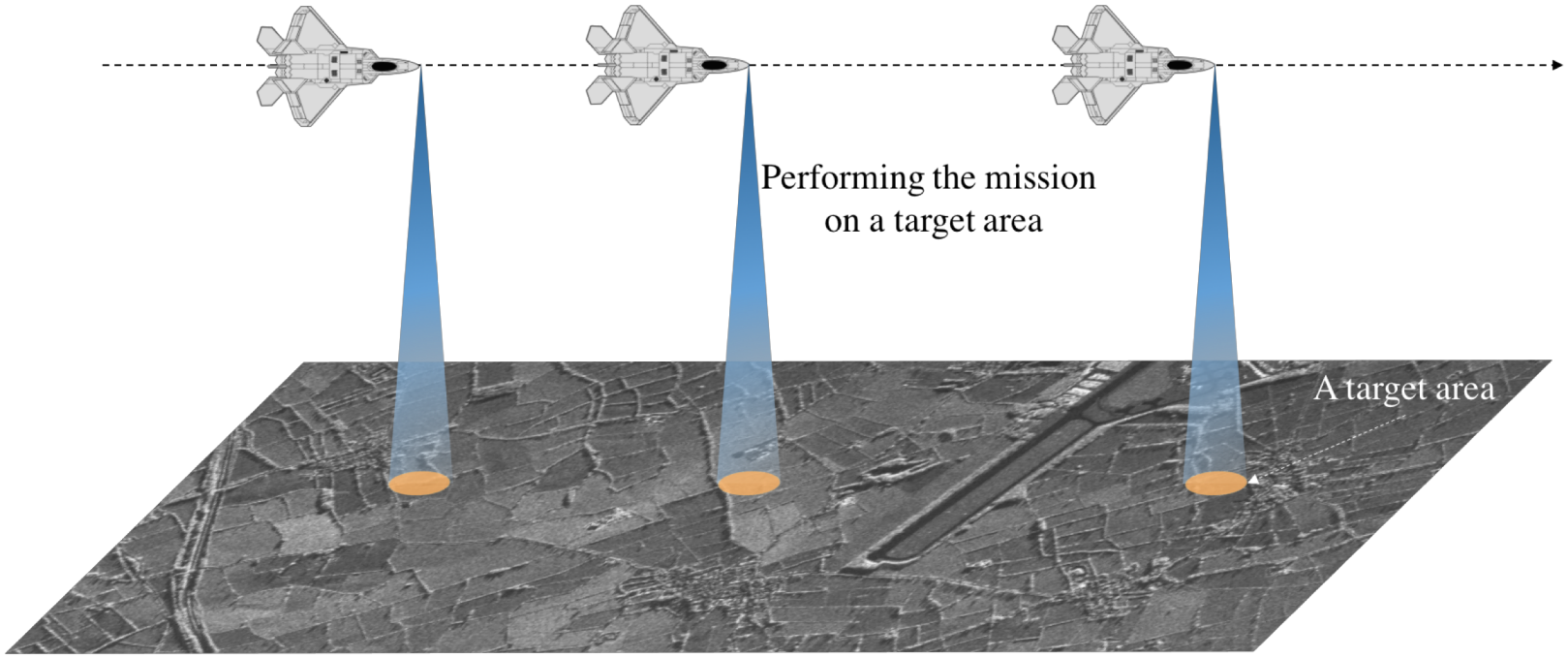
Fast

Guarantees

(i.e., Hard Real-Time)

- Morden embedded/real-time systems uses multiprocessor platforms
- Stream processing programming paradigm allows construction of concurrent data processing programs to exploit the parallelism with concise code, e.g., signal processing, Big Data
- However, existing stream processing techniques are not designed for real-time systems

Real-Time Streaming Use Case



- Real-Time Live Streaming Data Processing
- There are also hard real-time tasks in its defense system

My Research Overview

A RT-Stream Processing Architecture, i.e., System Model

↓

Framework using Real-time Java (or Ada, C with Real-time POSIX)

↓

Providing Integration approach

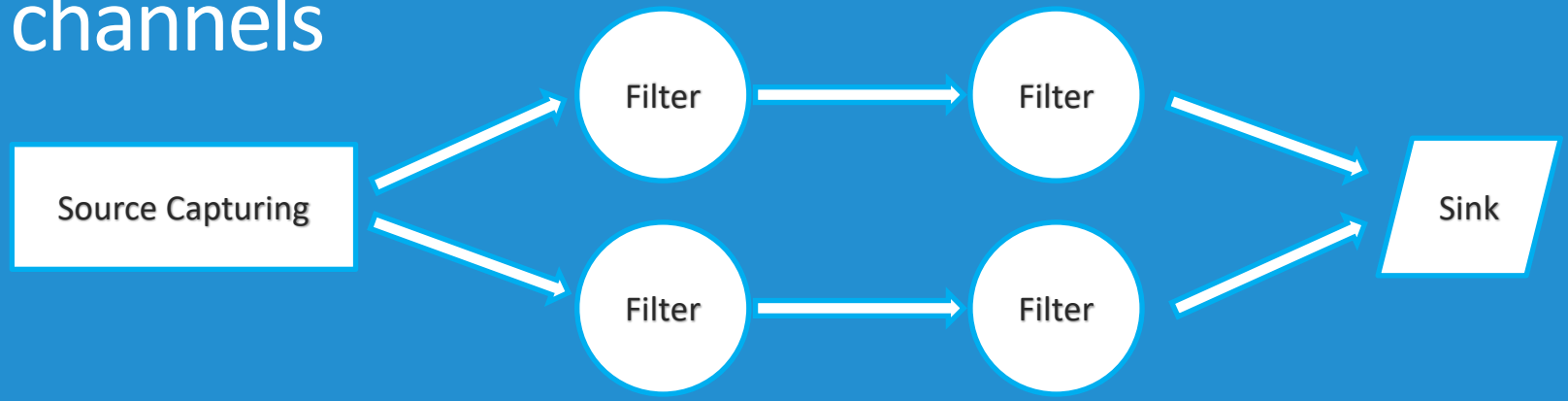
↓

Schedulability Analysis (RTA) – Math Equations to guarantee that No Any Deadline Miss

Contents

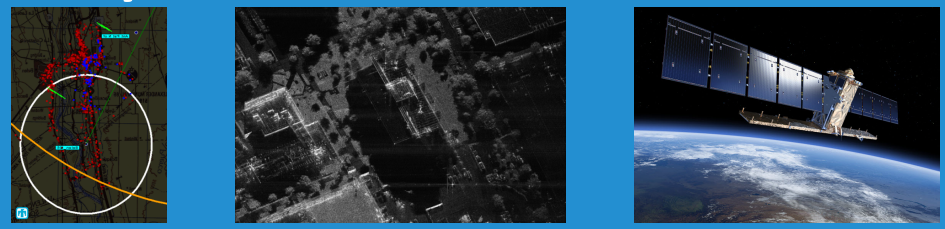
Introduction – RT Stream Processing

A set of modules that compute the input in parallel, and communicate via channels



Real-time Stream Processing

Data flowing through the system from its source to its sink has time constraints



Introduction – Stream Processing Input

Data source can be classified into:

- Batched

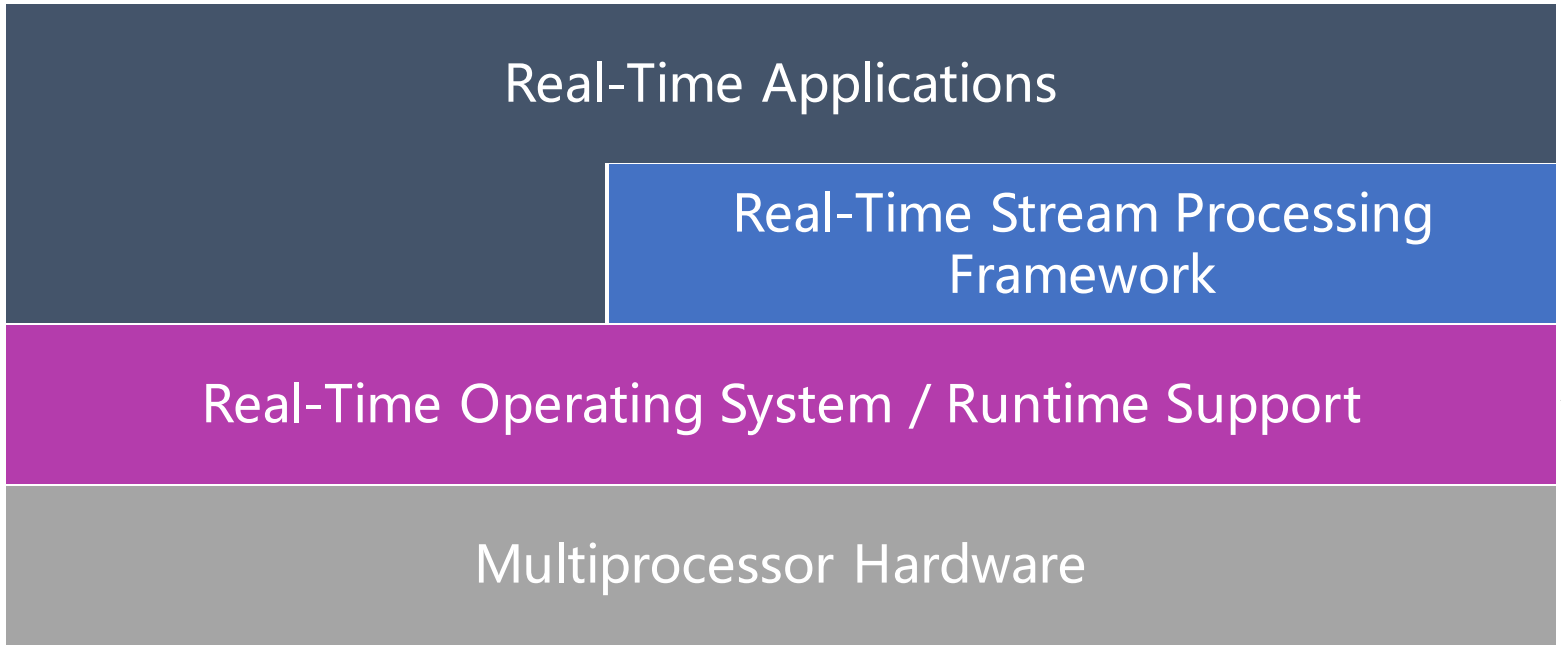
  the data is already present in memory, and its content and size will not change during processing

- Live Streaming

  data that arrives dynamically, its content and size will change with time

System Model

System Overview

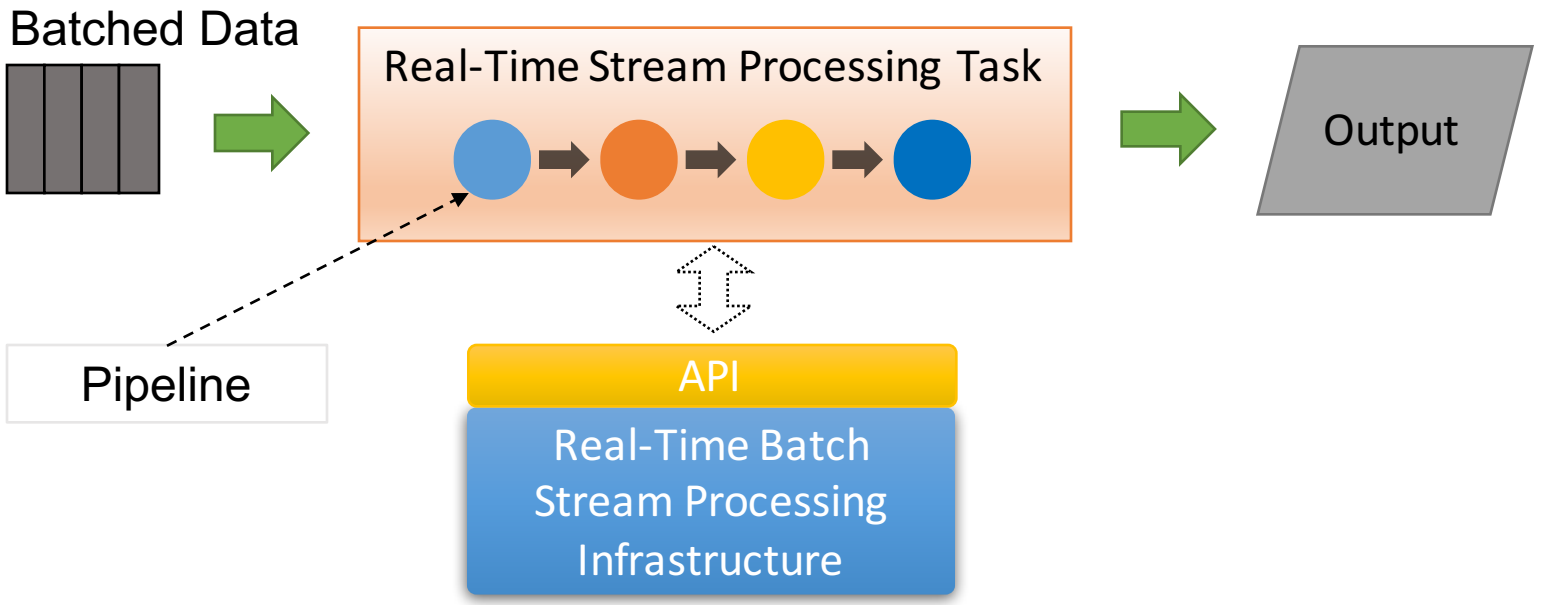


⬅Optional

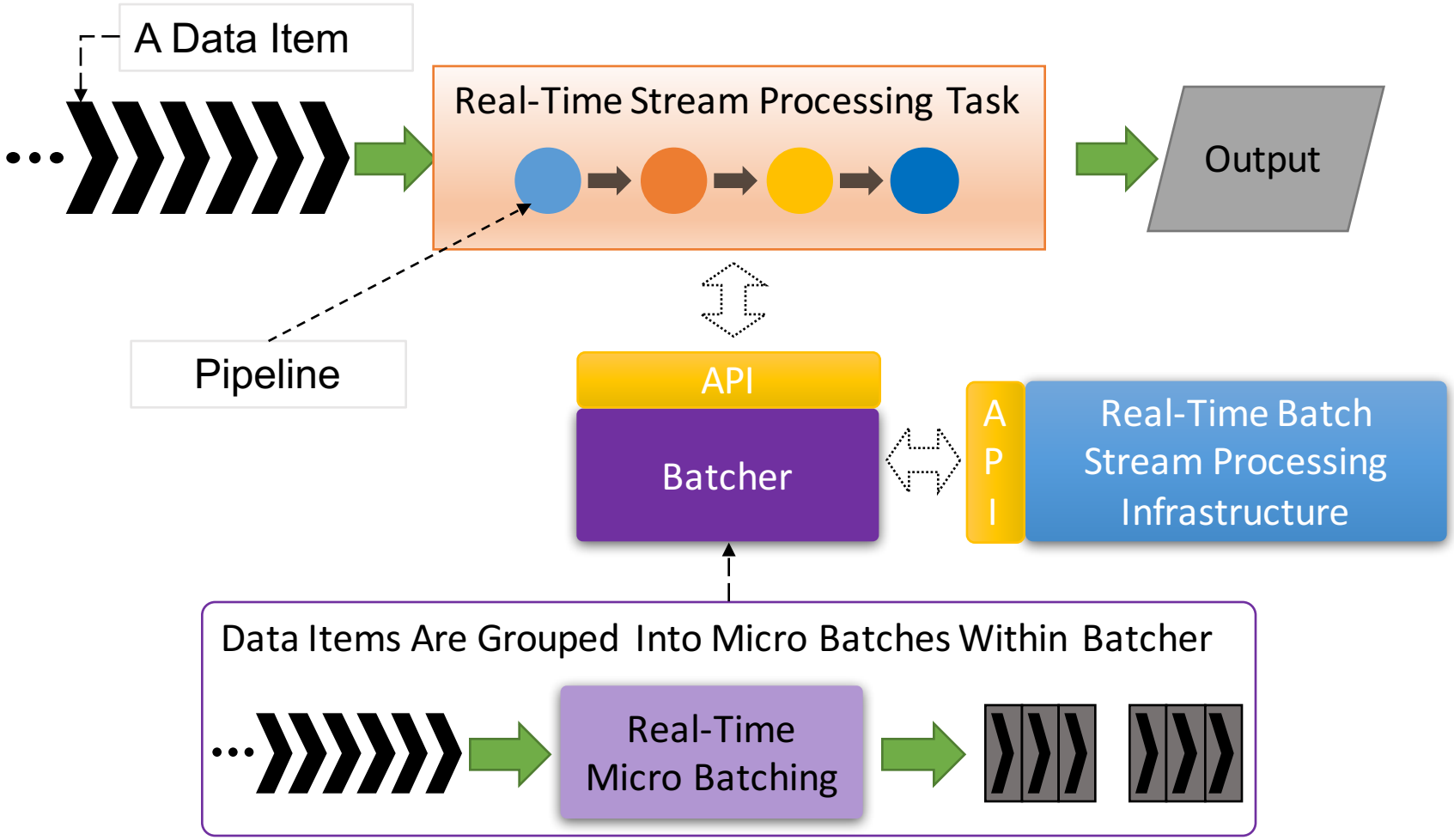
SPRY:

- Supports a stream processing paradigm for both batched and streaming data sources in real-time
- Preserves the guarantees given to existing real-time tasks
- Targets shared memory multiprocessor platforms

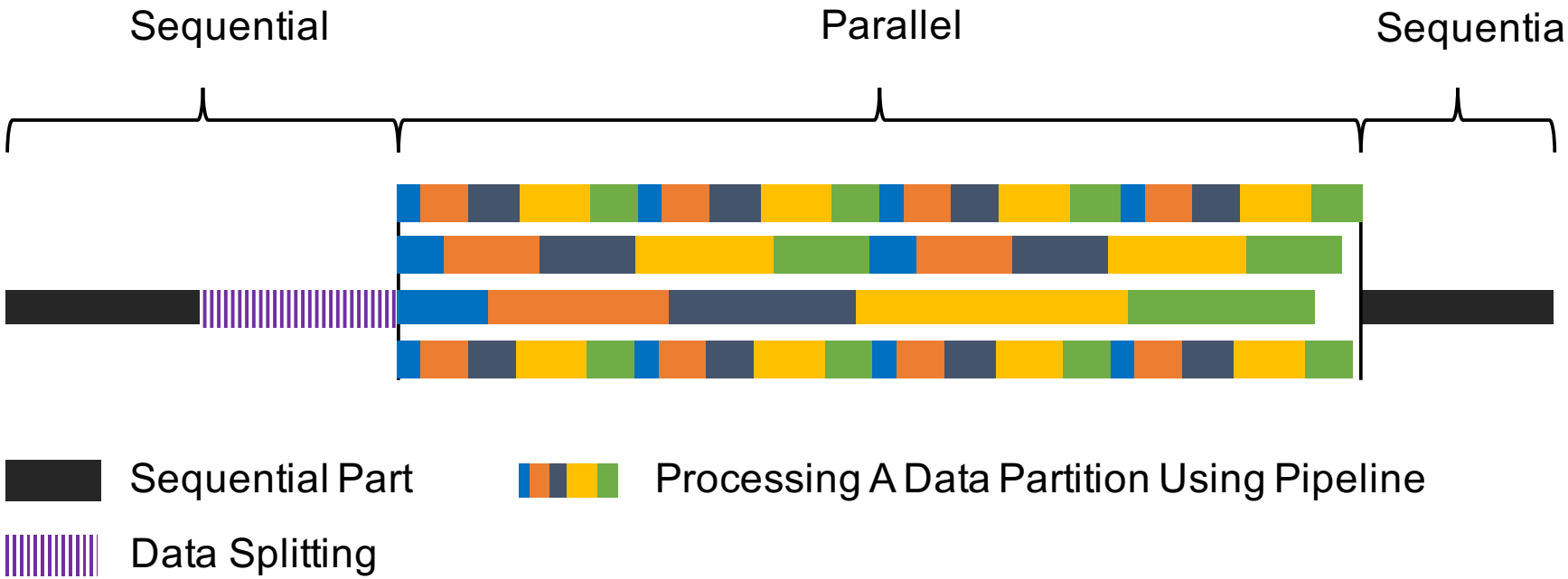
SPRY Use Case – Batched Data



SPRY Use Case – Live Streaming Data



Real-Time Data Parallel Stream Processing Task Model



- Preemptive Fixed Priority
- Fully Partitioned Scheduling
- Sporadic Task Model
- Mixed Hard Real-Time and Soft Real-Time Applications
- Hard and Soft Real-Time Stream Processing Activity
- Sporadic Live Streaming Data
- Multiple Simultaneous Streaming Workloads

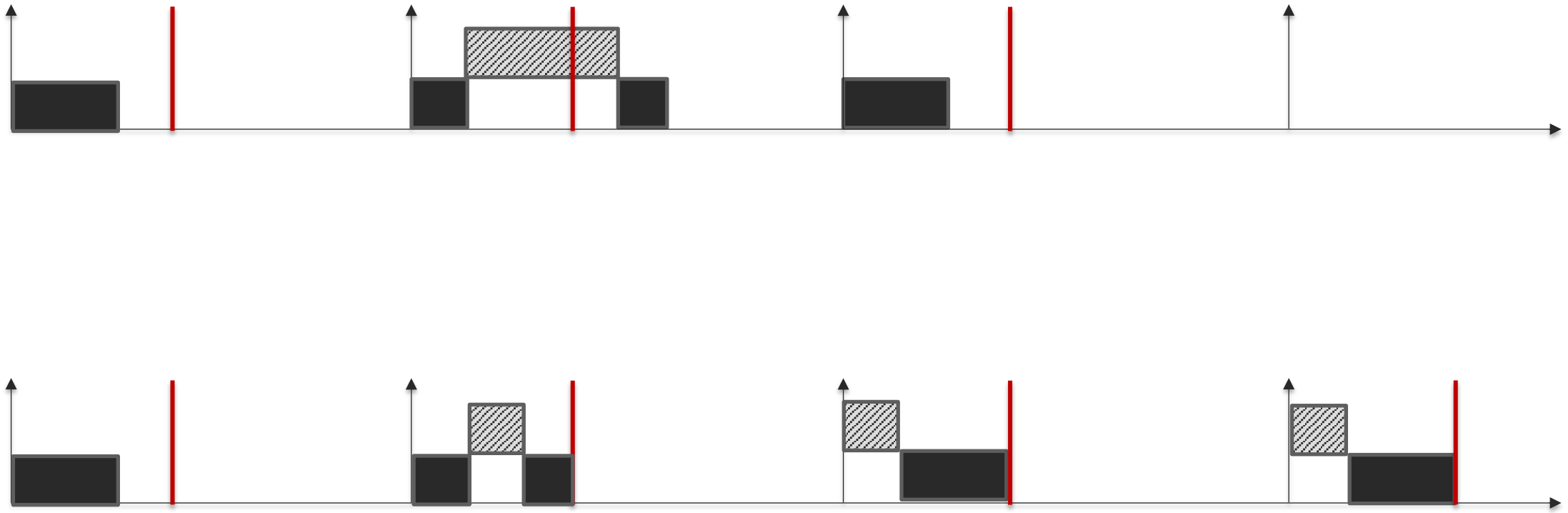
SPRY – Execution-Time Servers & Static Data Allocation

- Typically stream processing is computationally intensive
- Soft streaming might request unbounded CPU times
    - Running it at the lowest priority -> bad response times
    - Running it at too high a priority -> cause critical activities to miss their deadlines

Servers are typically used to support computationally intensive soft real-time tasks to give them good response times but bound their impact on hard real-time tasks

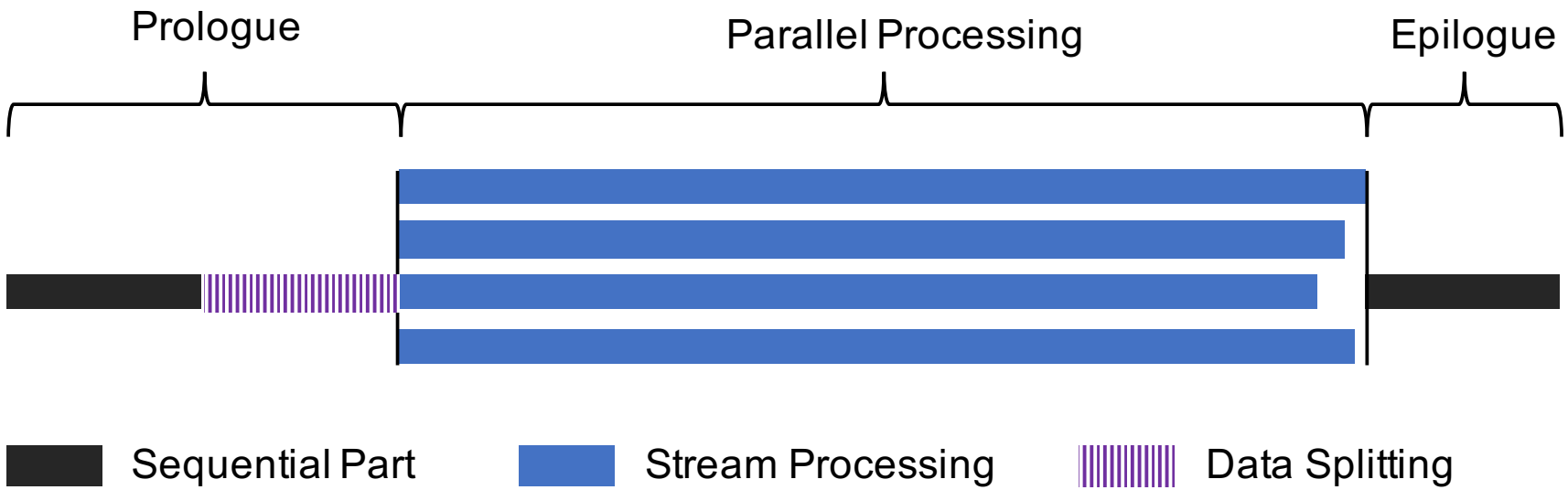
- Work-stealing make the analysis not tractable
    - Static Data Pre-allocation allows user to perform a sufficient worst-case response time analysis (RTA)

Execution Time Server



Schedulability Analysis

Analysis – Parallel Stream Processing Model



Exact Analysis for A Task Executing Under A Server

Bound Tasks

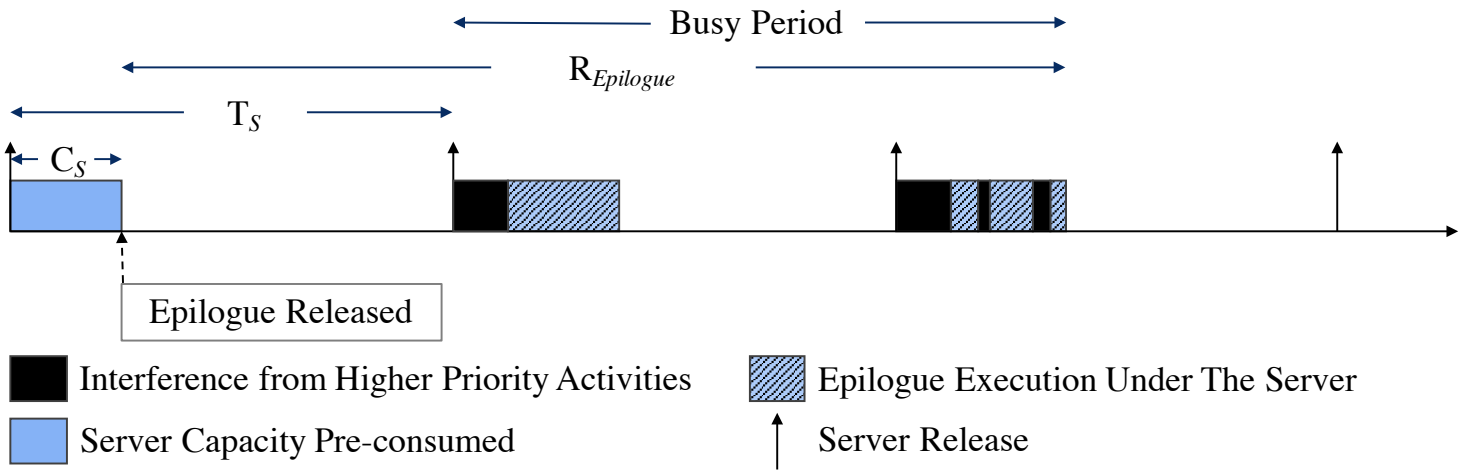
- Two activities are bound means one activity's period is an exact multiple of another one's period, for example, harmonic periods
- If a task is *bound* to a server, then each release of the a bound task coincides with each replenishment of the capacity of the server

We try to make the stream processing on each processor to be the *bound* task, in order to enhance task schedulability, and reduce the server capacity requirements [2]

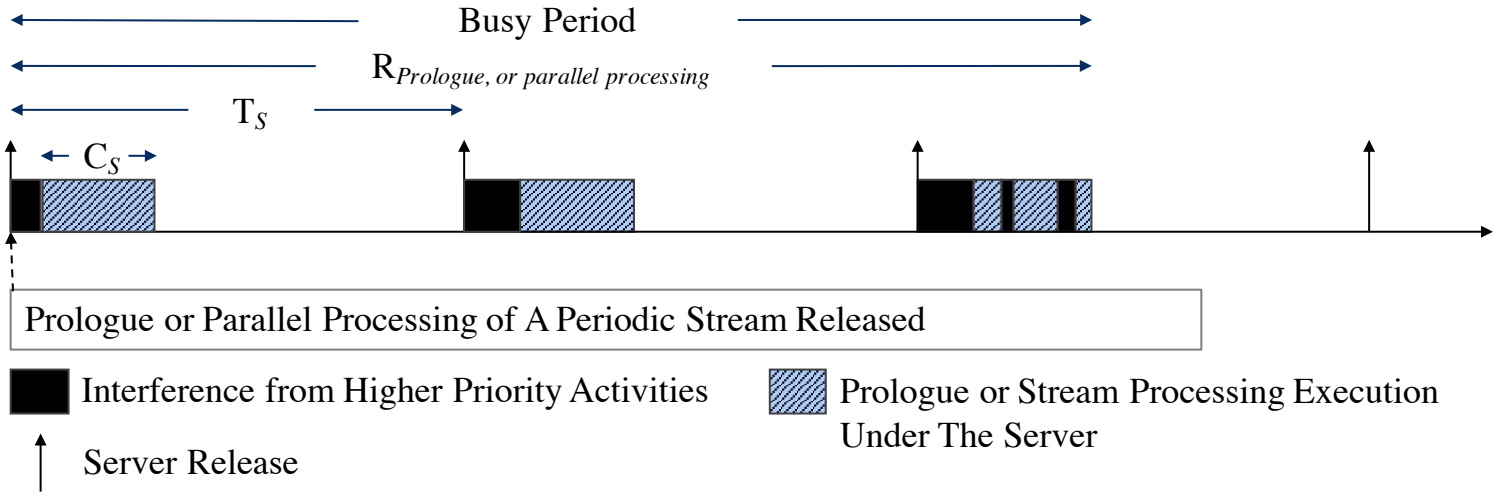
[2] R. Davis and A. Burns. An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In 16th International Conference on Real-Time and Network Systems (RTNS 2008), 2008.

Task i's Critical Instance Under A Server

unbound



bound



Task i's Response Time Under A Server S

Considering it is the only task using the server, therefore, the load of server's busy window is

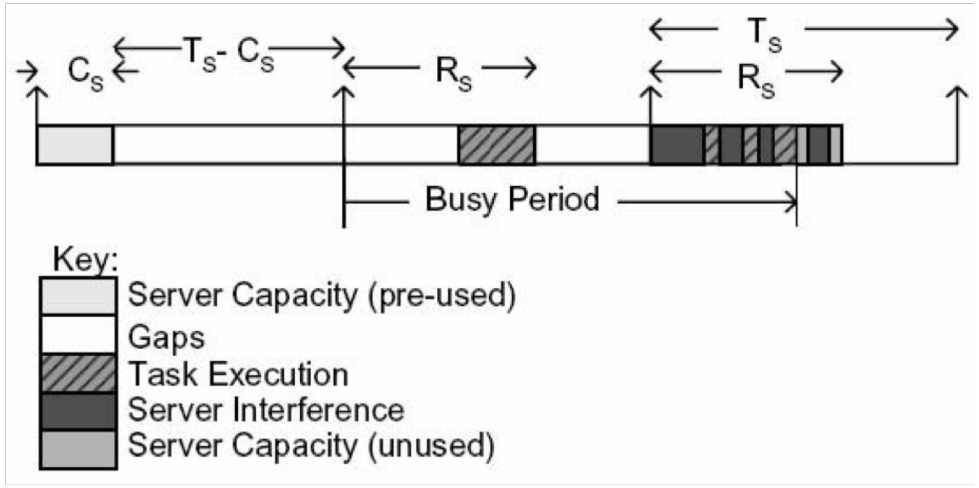
$$C_i$$

The total length of gaps in complete server periods:

$$\left(\left\lceil \frac{C_i}{C_S} \right\rceil - 1\right)(T_S - C_S)$$

The interference received from higher priority activities in the server's final period

$$\sum_{\forall j \in hp(S)} \left\lceil \frac{max\left(0, w_i^n - \left(\left\lceil \frac{C_i}{C_S} \right\rceil - 1\right) T_S\right) + J_j^S}{T_j} \right\rceil C_j$$



Source: R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In 26th IEEE International Real-Time Systems Symposium (RTSS'05), pages 10–pp. IEEE, 2005

For a periodic/sporadic task, or periodic/sporadic server, $J_j^S = 0$. For a deferrable server, $J_j^S = T_j - C_j$, considering the 'back-to-back' hits phenomenon

When the current server is bound relative to the higher priority deferrable server, then $J_j^S = 0$, as double hits can not ever occur, proof in thesis

Task i's Response Time Under A Server S

Using recursive equation to determine the length of the busy period

$$w_i^{n+1} = C_i + \left(\left\lceil \frac{C_i}{C_S} \right\rceil - 1 \right) (T_S - C_S) + \sum_{\forall j \in hp(S)} \left\lceil \frac{max\left(0, w_i^n - \left(\left\lceil \frac{C_i}{C_S} \right\rceil - 1 \right) T_S \right) + J_j^S}{T_j} \right\rceil C_j$$

with the start value of

$$w_i^0 = C_i + \left(\left\lceil \frac{C_i}{C_S} \right\rceil - 1 \right) (T_S - C_S)$$

end when

- $w_{n+1} = w_n$ => $R_i = w_{n+1} + J_i$
- $w_{n+1} > D_i - J_i$ => *Not schedulable*

J_i is the release jitter of the task relative to the release of the server, it is zero for bound task, other wise it is T_s - C_s

Invoking blocking/resource sharing, see thesis for more details

Parallel Stream Worst-Case Response Time Analysis

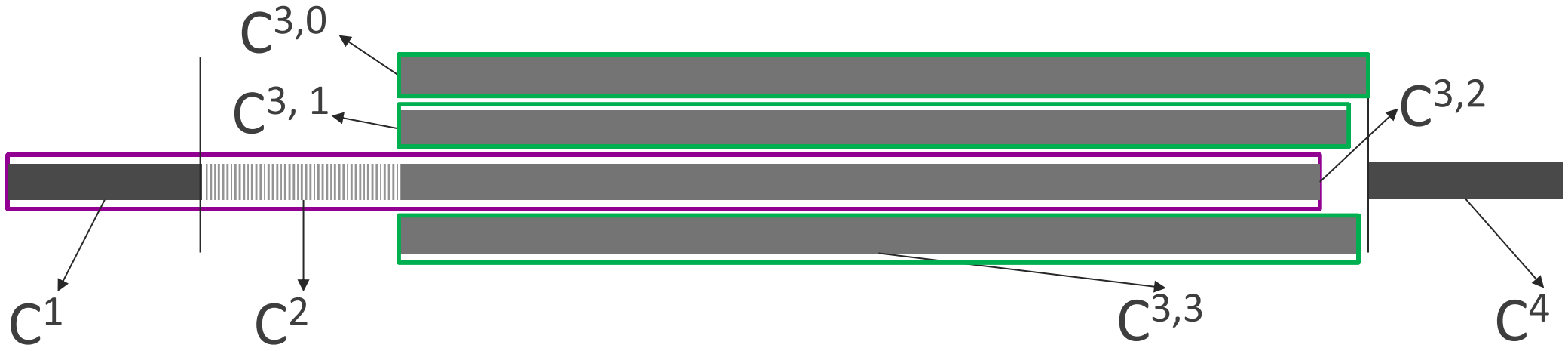
- Stream Processing: Period of *T*
- One server has been generated for each processor (how to generate will be discussed later)

Parallel Stream Worst-Case Response Time Analysis

Step 1 - Worst-case Response Time for the prologue:

- We know the WCET for the sequential before the splitting, and the splitting procedure, i.e., C^1+C^2
- This can be treated as a periodic bound task, executing under the server, therefore, we can use the above analysis techniques to do the analysis
- This can be modelled as a task i with $T_i = T$, $C_i = C^1+C^2$, and $J_i=0$, executing under the server on this processor
- The response time can be calculated, say, R^1

Parallel Stream Worst-Case Response Time Analysis



Step 2 - Worst-case Response Time for the parallel Processing:

- For the processor, where the stream processing is released, it can be modelled as a task i with $T_i = T$, $C_i = C^1+C^2+C^{3,2}$, and $J_i=0$, executing under the corresponding server, => $R^{3,2}$
- For the rest processors, e.g., processor 1, it can be modelled as a task i with $T_i = T$, $C_i = C^{3,1}$, and $J_i=0$, executing under the corresponding server, => $R^{3,1}$
- From release to when the parallel processing finished:

$$R=\text{Max}(R^{3,2},\ C^{3,0}+R^{1},\ C^{3,1}+R^{1},\ C^{3,3}+R^{1})$$

Parallel Stream Worst-Case Response Time Analysis

Step 3 - Worst-case Response Time for the epilogue:

- This can be modelled as a task i with $T_i = T$, $C_i = C^4$, and $J_i = T_S - C_S$, executing under the server S on this processor, because when the epilogue is released, the server's capacity has been potentially used by the prologue and the parallel processing
- The response time can be calculated, say, R^3

Finally, the response time of the whole stream processing is:

$$R_{stream} = R + R^3$$

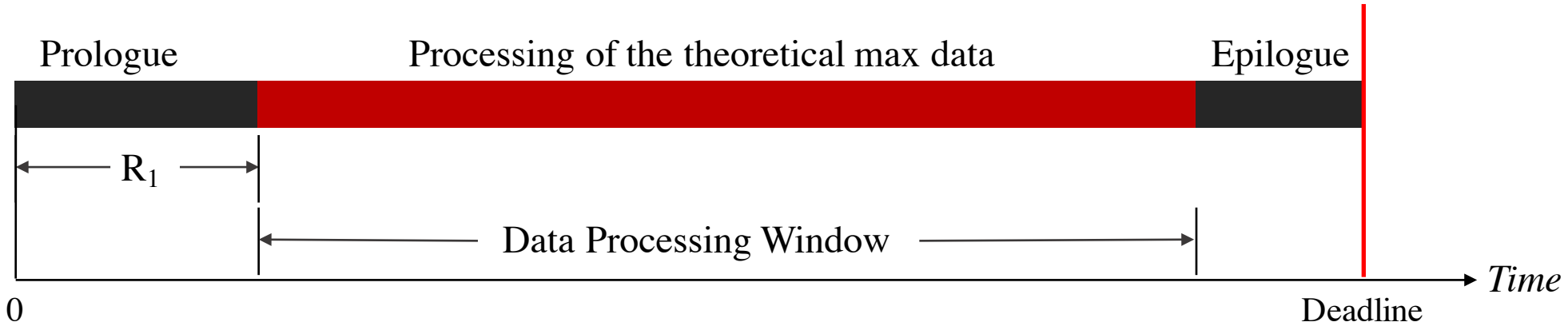
Scheduling and Integration

RT Stream Processing Task for Batched Data

Given a real-time stream processing task for a batched data source, with period T, and deadline D:

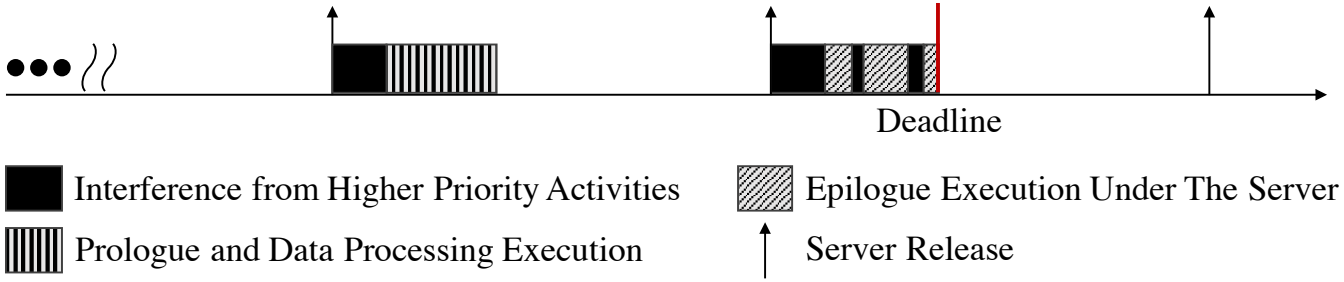
- Generate execution-time servers for each processor
- Perform the data partitioning, and allocate the data partitions to each processor
- Analyse the worst-case response time of this task

Data Processing Window

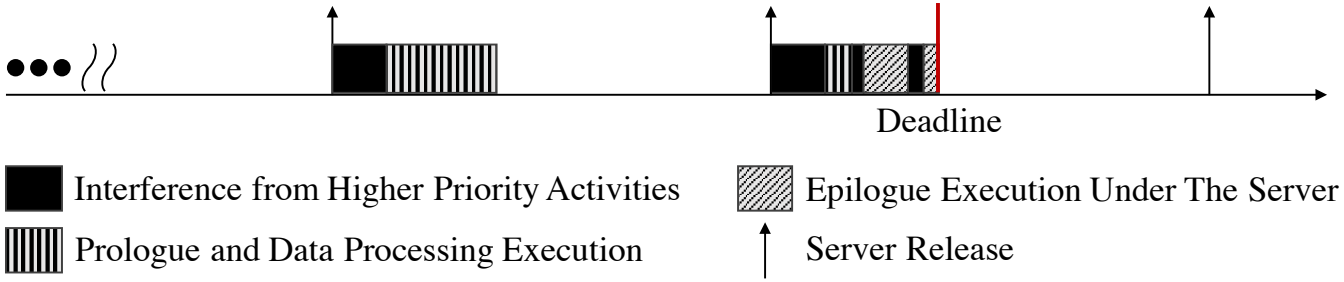


- The latest time when the prologue finishes, i.e., $R_{prologue}$
- The latest time, when the epilogue has to start, in order to finish its execution before the deadline

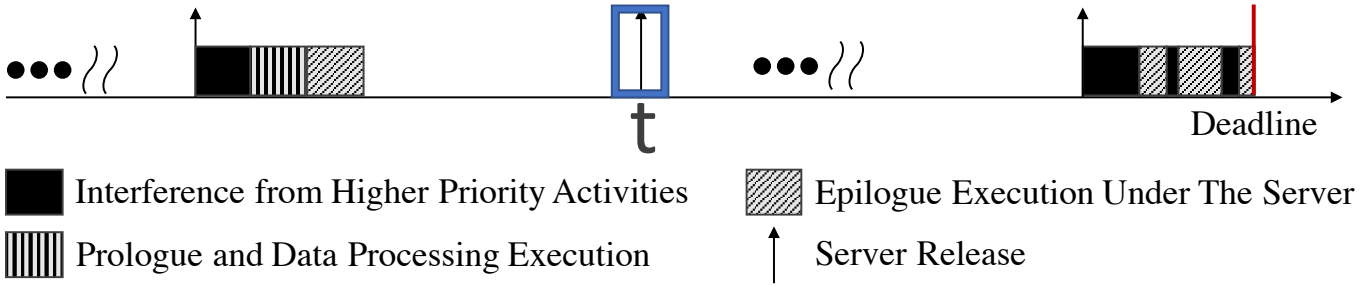
Data Processing Window



→ $D - R_{epilogue}$



→ $D - R_{epilogue}$



→ $t - R_{epilogue'}$

Server Generation

Given a Stream Processing Task i, For each processor:

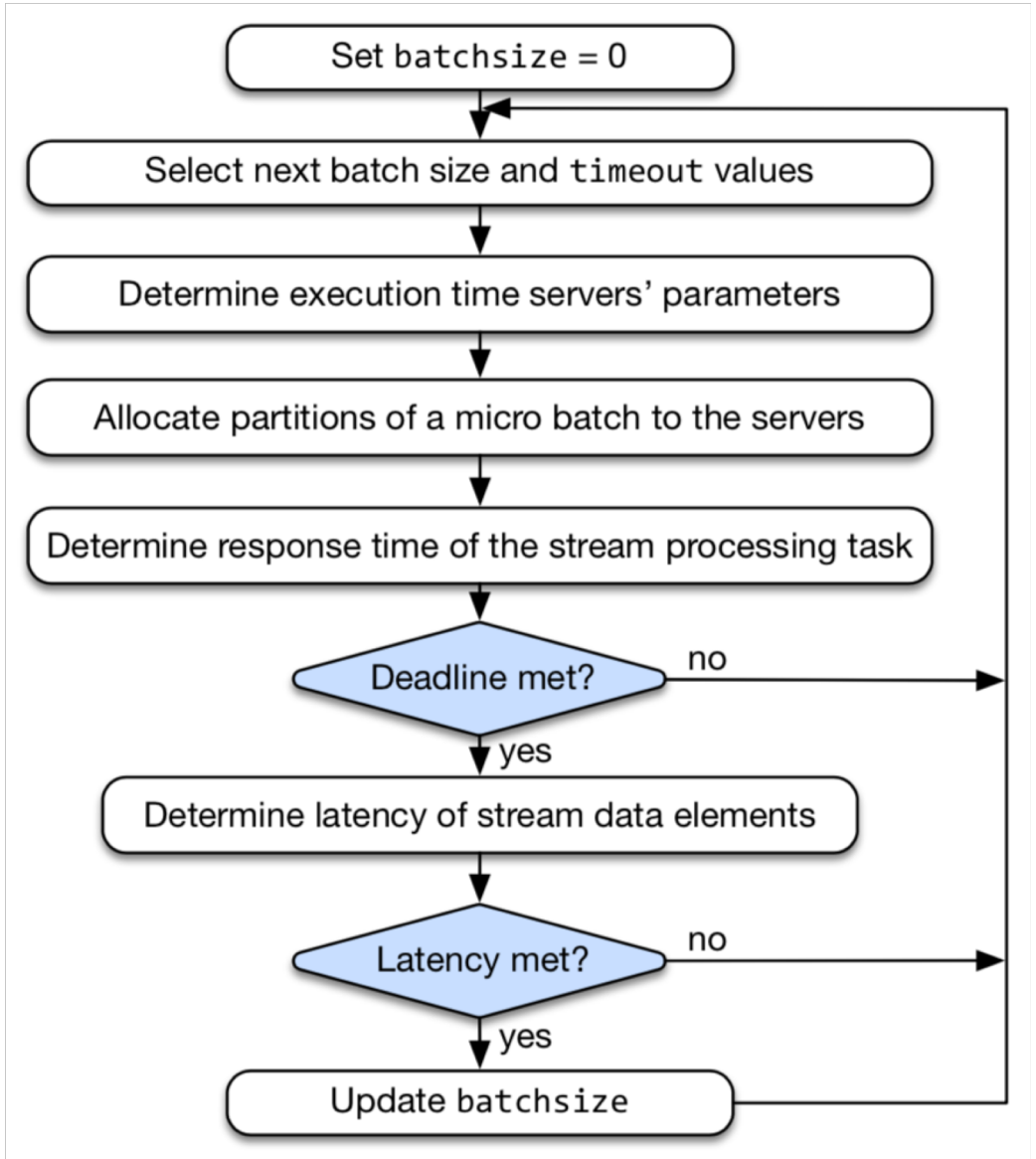
1. Order the hard real-time activities using deadline monotonic priority assignment, and check the schedulability
2. Calculates **exact divisors** of T_i as the potential periods for the server
3. Get one unchecked period from Step 2 as the period T_S, create a prologue server S, with the deadline $D_S = T_S$
4. Find the start priority for the server S using deadline monotonic assignment
5. Using the binary search between 0 and T_S to determine the maximum capacity for S at its priority level
6. Calculate the biggest possible data processing window: *DPW*
7. For the remaining processor, find the server can deliver max capacity within *DPW*
8. Repeat Step 3, then a set of servers, which deliver max capacity for the parallel processing, can be obtained

Static Data Pre-Allocation

After splitting, for each partition:

1. Find the processors, its server has enough capacity left for processing this partition
2. Calculate the time when the partition's processing can be completed in each processor in step 1
3. Allocate it to the processor, which is the earliest one to finish the processing of it

Live Streaming – Determining The Max Batch Size



Worst-case: data arriving continuously

Given the size for the micro batch is N

The period is $(N\text{-}1) \times MIT^{item}$

Live Streaming – Determining The Timeout

Arguably, the data does not always arrive with MIT^{item}, therefore, in this case, the micro batch can not be fully filled up, release the micro batch earlier to avoid any deadline miss

Given the max size for the micro batch is N, the timeout is $(N\text{-}1) \times MIT^{item}$

For the case that a micro batch is released to be processed using timeout, the worst-case is that the 1^{th} to $(N-1)^{th}$ data items arrive with the MIT^{item}, the N^{th} item does not arrive

This is almost identical to the batch with N data times

Live Streaming – Schedulability of The Half Batch

If the batch in the data flow's worst-case, i.e., data arrives with MIT^{item} and the batch size=N, is schedulable, the half batch with the timeout = $N\text{-}1) \times MIT^{item}$ is certainly schedulable

Proof

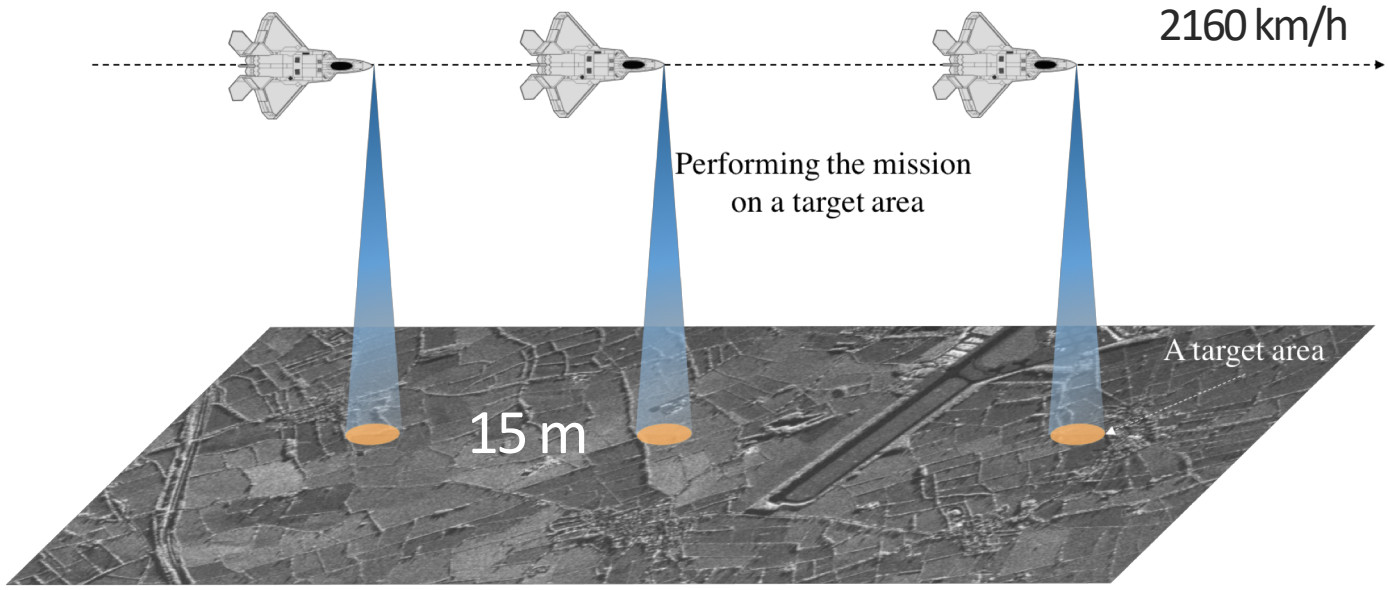
Comparing with a full micro batch:

1. It has at most N items, therefore the response time of the this micro batch is not bigger than the response time of the full micro batch
2. For any i^{th} item, where $1 \leq i \leq N$, the item is allocated to the same processor compared to the full micro batch, → the item has the same response time of processing
3. For any i^{th} item, where $1 \leq i \leq N$, the waiting time of this item is less than or equal to the one in the full micro batch, as the data items do not always arrive with MIT^{item} , → the latency of this item will not be any bigger

Therefore, the half batch with the timeout = $N \times MIT^{item}$ is certainly schedulable

Case Study

Hard Real-Time Streaming



- 4 processors SMP system
- Data arriving interval 15m ÷ 2160km/h = 40 ms
- Each image of a target area must be generated within 480 ms
- WCET of generating an image from the raw echoes is 40 ms using a single core

Defense System

Name	Priority	C	T & D	U	*Proc*
Weapon Release	98	3	200	1.5	0
Rader Tracking Filter	84	2	25	8.0	1
RWR Contact Mgmt	72	5	25	20.0	2
Data Bus Poll Device	68	1	40	2.5	3
Weapon Aiming	64	3	50	6.0	0
Radar Target Update	60	5	50	10.0	3
Nav Update	56	8	59	13.6	0
Display Graphic	40	9	80	11.3	1
Display Hook Update	36	2	80	2.5	3
Tracking Target Update	32	5	100	5	3
Nav Steering Cmds	24	3	200	1.5	1
Display Stores Update	20	1	200	0.5	2
Display Key Set	16	1	200	0.5	3
Display Stat Update	12	3	200	1.5	2
BET E Status Update	8	1	1000	0.1	3
Nav Status	4	1	1000	0.1	3

Hard Real-Time Streaming

Schedulable max batch size = 17

Candidate Prologue Servers

Priority	Max C	T	DPW	Max C in DPW From All Processors
55	314.000	400	360.0	1147.0 (MAX)
55	153.000	200	360.0	1139.0
55	75.000	100	360.0	1133.0
55	55.000	80	360.0	1108.0
99	21.000	50	388.0	1071.7
99	21.000	40	388.0	1113.7
99	14.000	25	388.0	1127.7
99	12.000	20	388.0	1143.7

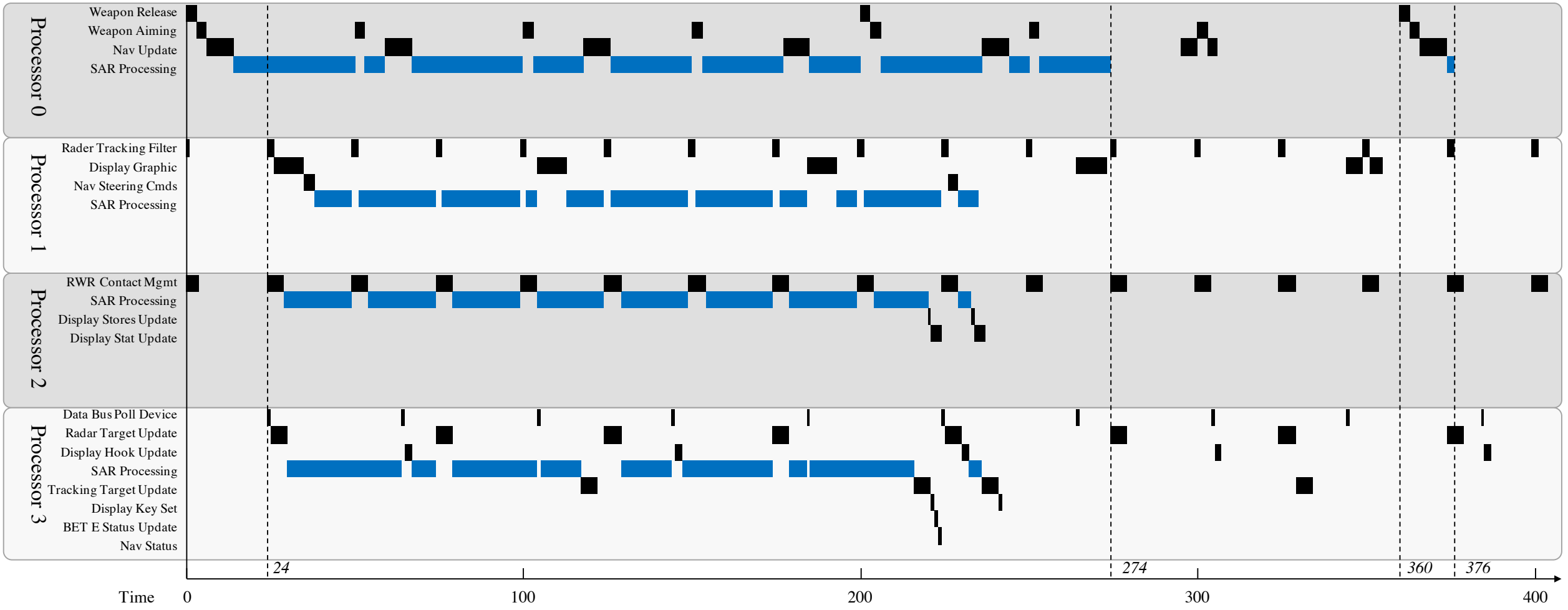
All Generated Servers

Name	Priority	C	T	D	Processor	U
S_0	55	314	400	400	0	0.785
S_1	23	317	400	400	1	0.793
S_2	71	156	200	200	2	0.780
S_3	35	78	100	100	3	0.780

Data Allocation

Processor	Allocated Data Items (Arrival Index)
0	2, 4, 8, 12, 16
1	3, 7, 10, 14
2	1, 5, 9, 13
3	0, 6, 11, 15

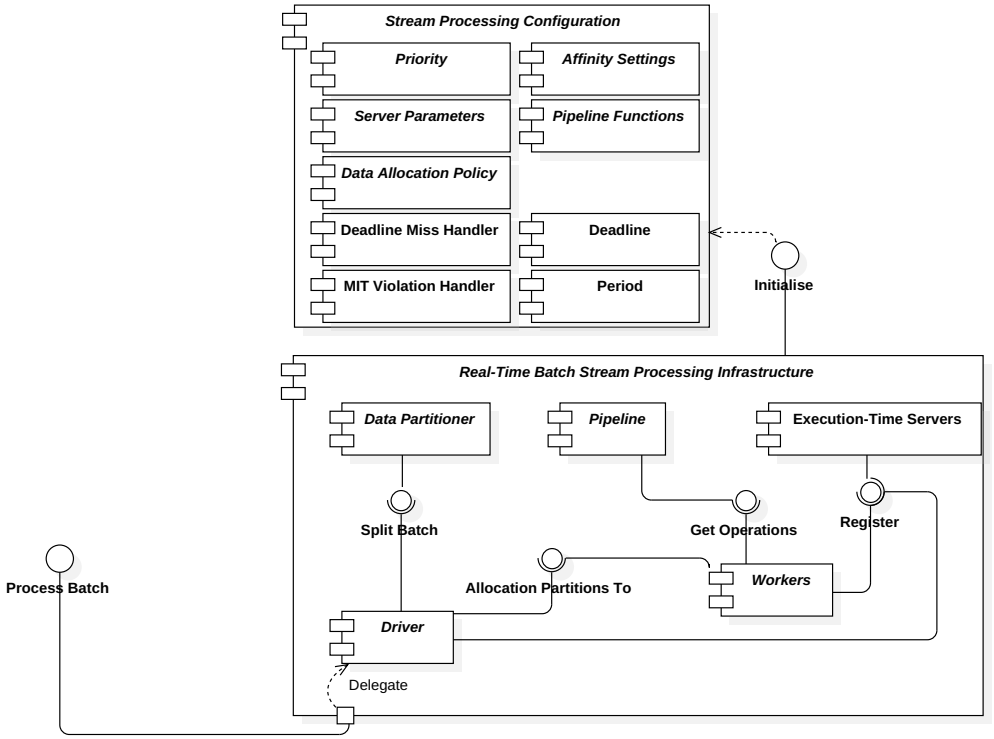
Worst-Case Execution Visualisation

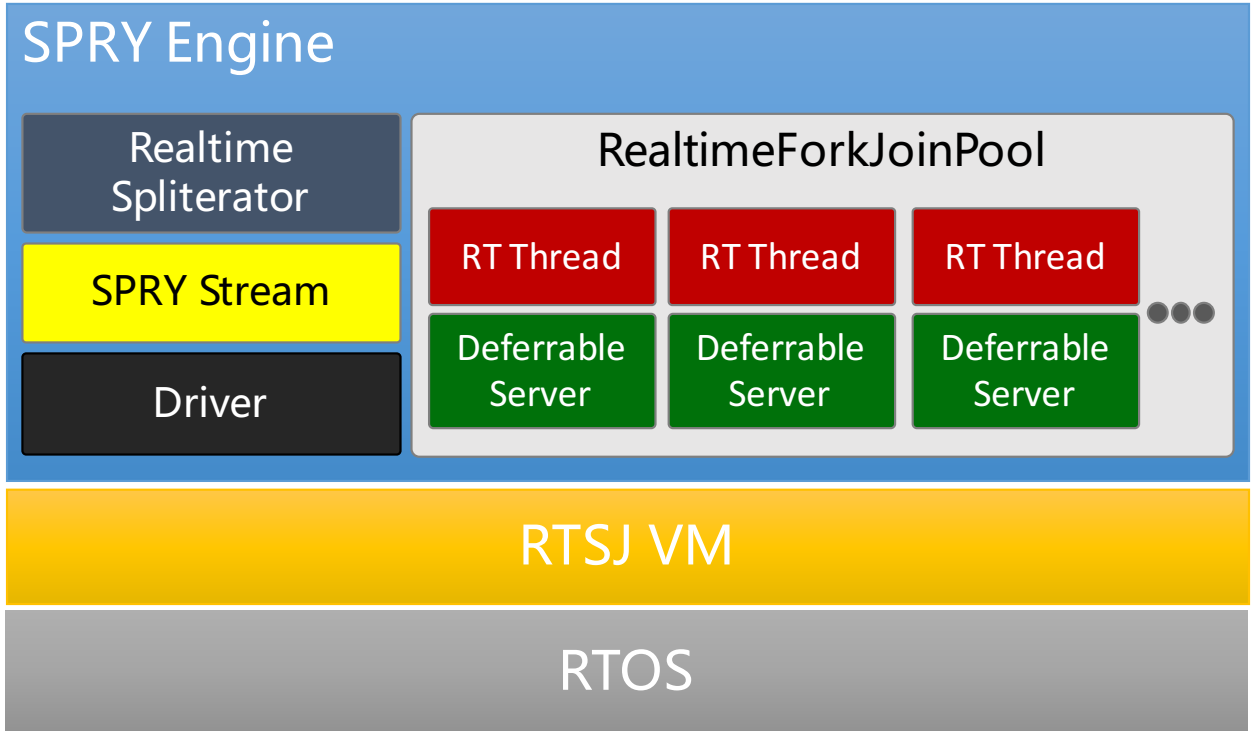


Implementation

Implementation

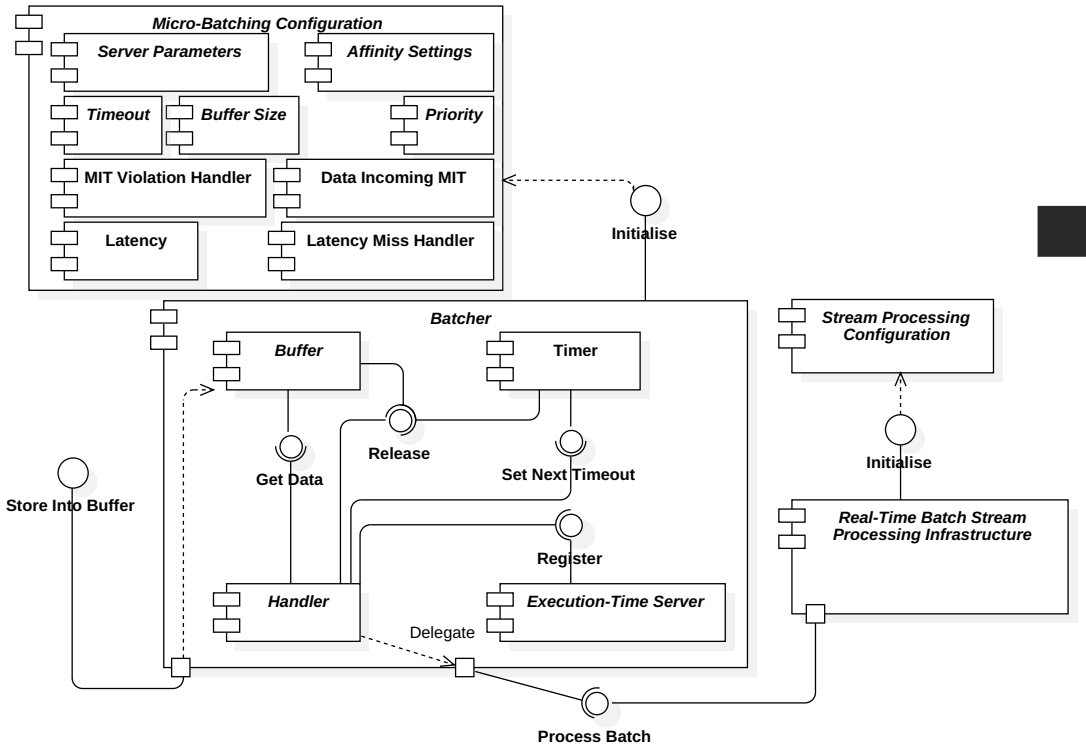
- Java 8 Streams
- The Real-Time Specification For Java (RTSJ)

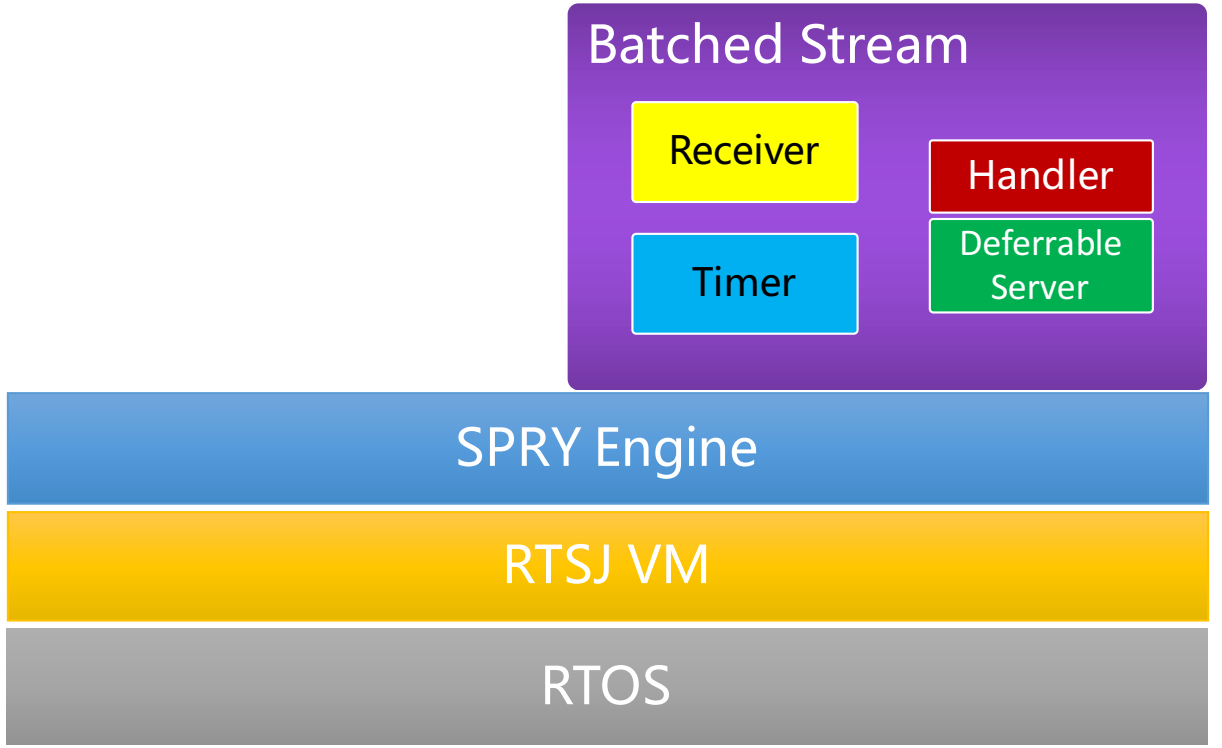




Implementation

- Java 8 Streams
- The Real-Time Specification For Java (RTSJ)





Implementation

1. Making Streams can be processed at a real-time priority, and executed under execution-time servers
    - Real-time ForkJoinPool[1]
    - Implementation of Deferrable Servers
2. Implementation SPRY Pipeline
3. Replace the work-stealing algorithm in ForkJoinPool

Too many details for each step, see thesis

[1]Made new proposal to RTSJ Specification, has been adopted by JSR 282 expert group; The JCP Expert Group has released a new version of the RTSJ (Version 2.0) in early 2016

Evaluation

Experiment Setup

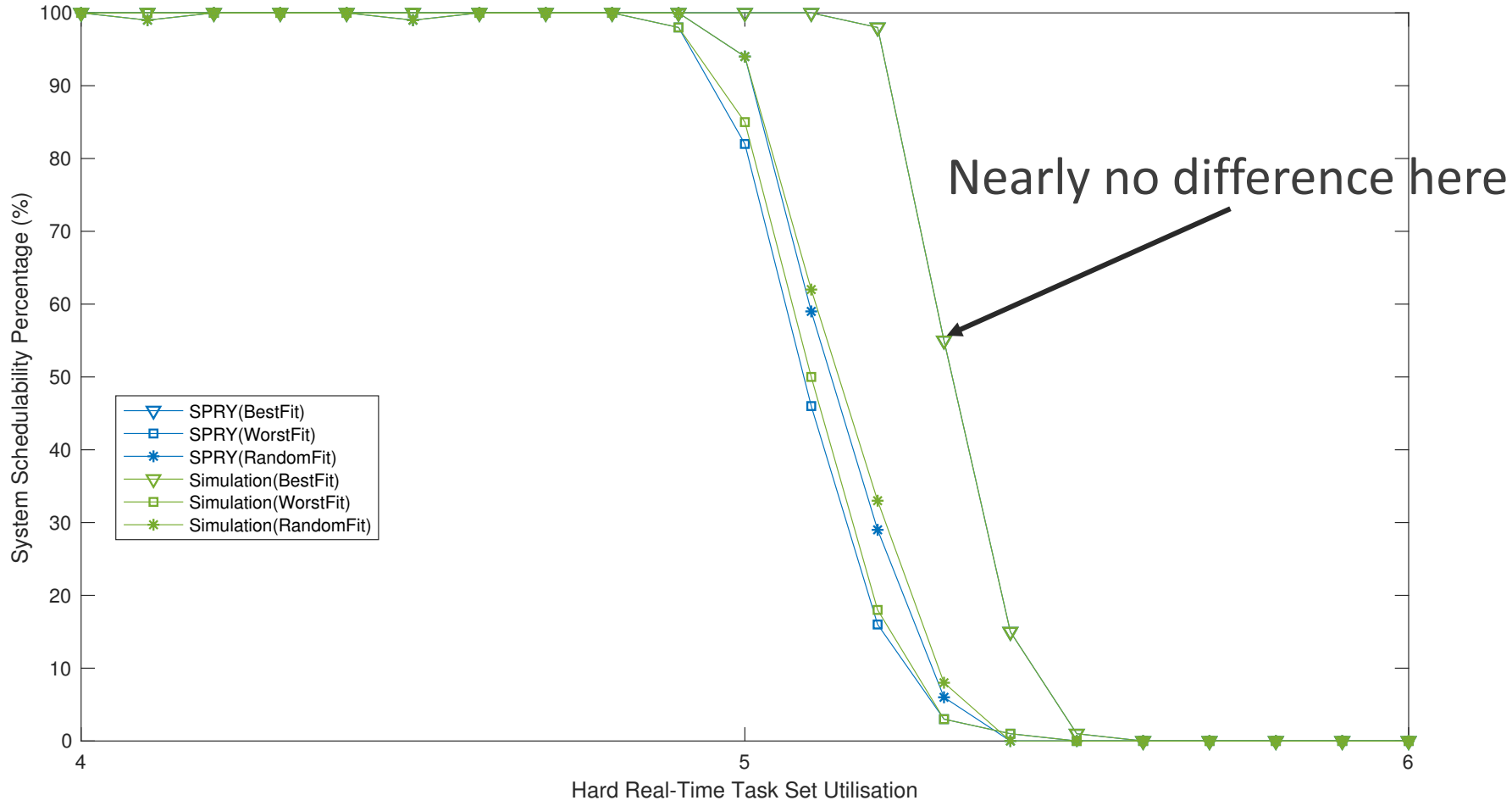
- 128 randomly generated hard real-time tasks
  - Periods: 1~1000 random
  - Normal Utilisation distribution[2]
- 16 cores machine (128 cores also has been investigated)
- Hard real-time stream processing tasks
  - Batched or Live streaming
  - Configurable Period, Deadline, Computation load, etc.

❖ Run experiment 100 times, test system schedulability, calculate how many runs are schedulable

[2] R. I. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. Real- Time Systems, 47(1):1–40, 2011.

Analysis Accuracy

Is our worst-case response time analysis techniques are too pessimism?
Analysis VS. Simulation



Comparing To Traditional Embedded Approach

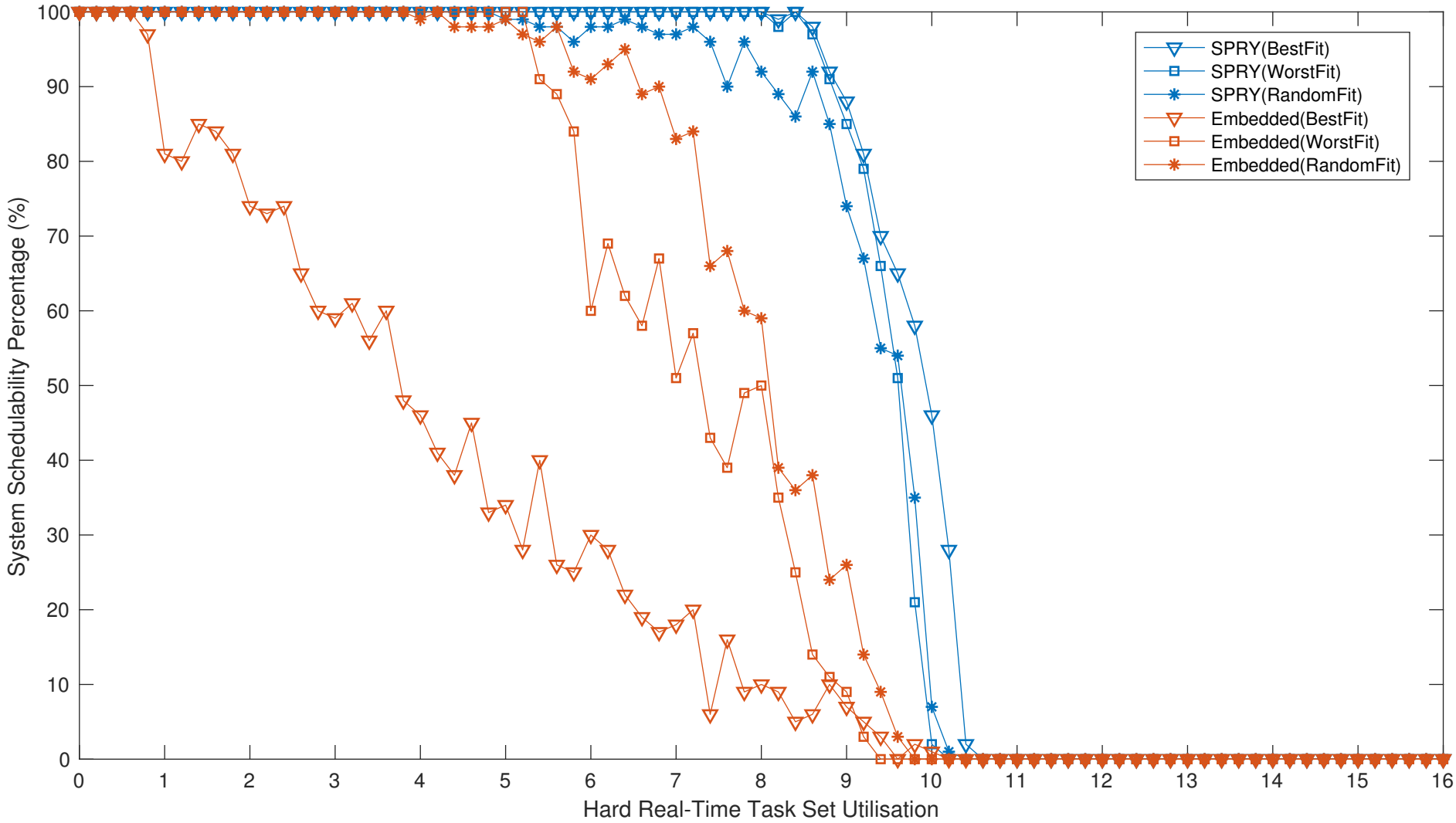
1. Splits the data source into partitions
2. Creates the corresponding prologue task, data processing tasks (one per partition), and the epilogue task
3. The period is equals to the stream processing task's period
4. The priority is determined using deadline monotonic priority assignment
5. When allocating tasks, the generated stream processing sub tasks and all the hard real-time tasks in the task set are considered together, with a first-fit, worst-fit, or random-fit allocation algorithm
6. If multiple generated stream processing sub tasks are allocated to the same core, they are merged into one task. In addition, the epilogue task is merged into the data task that finishes lastly

Comparing To Traditional Embedded Approach

Real-time stream processing task:

- T = 800
- C = 4000
- D = 800
- Prologue = 80
- Epilogue = 80

128 hard tasks

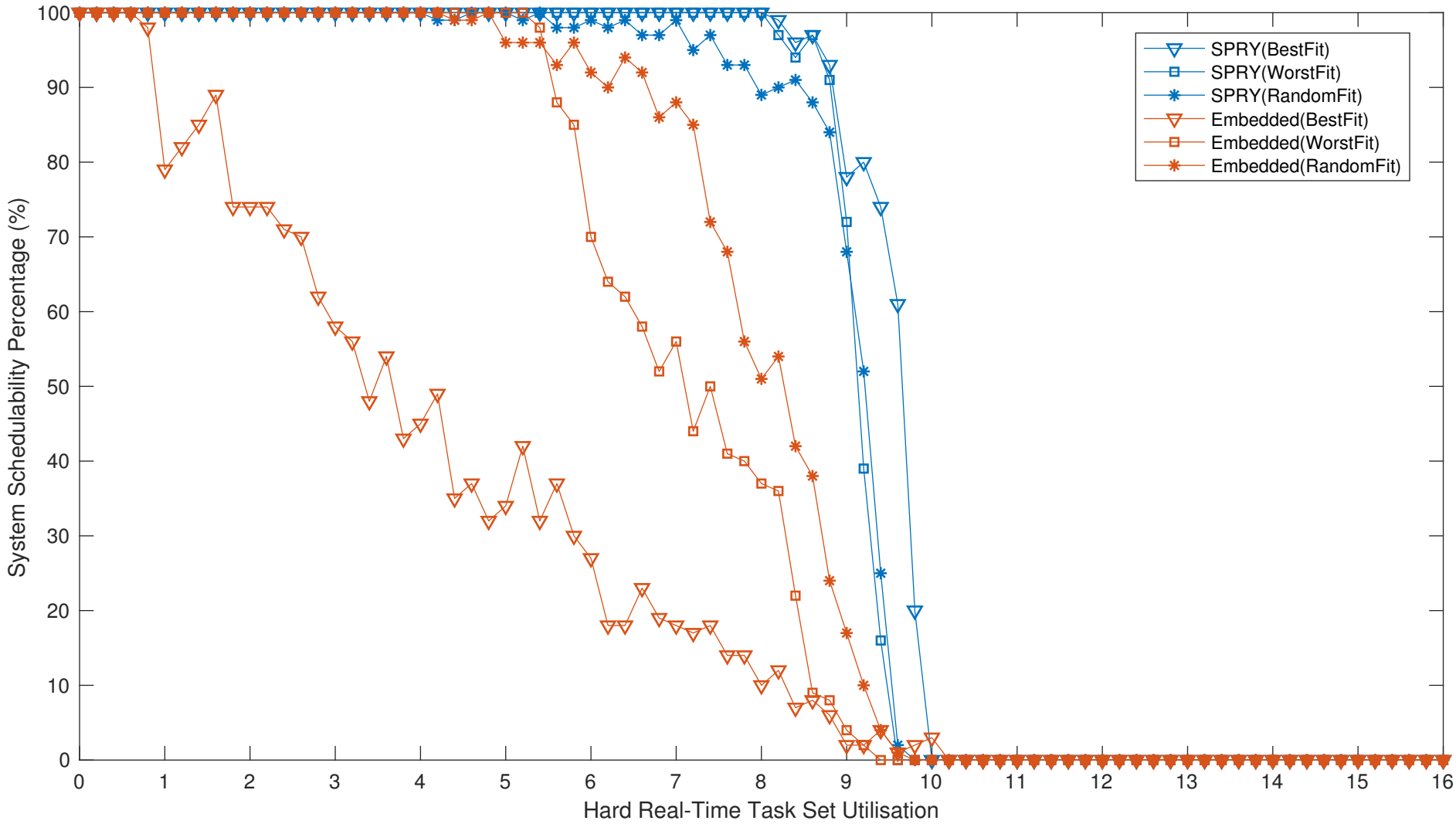


With Server Implementation Overhead

Same configuration

Server overhead:

10% of capacity



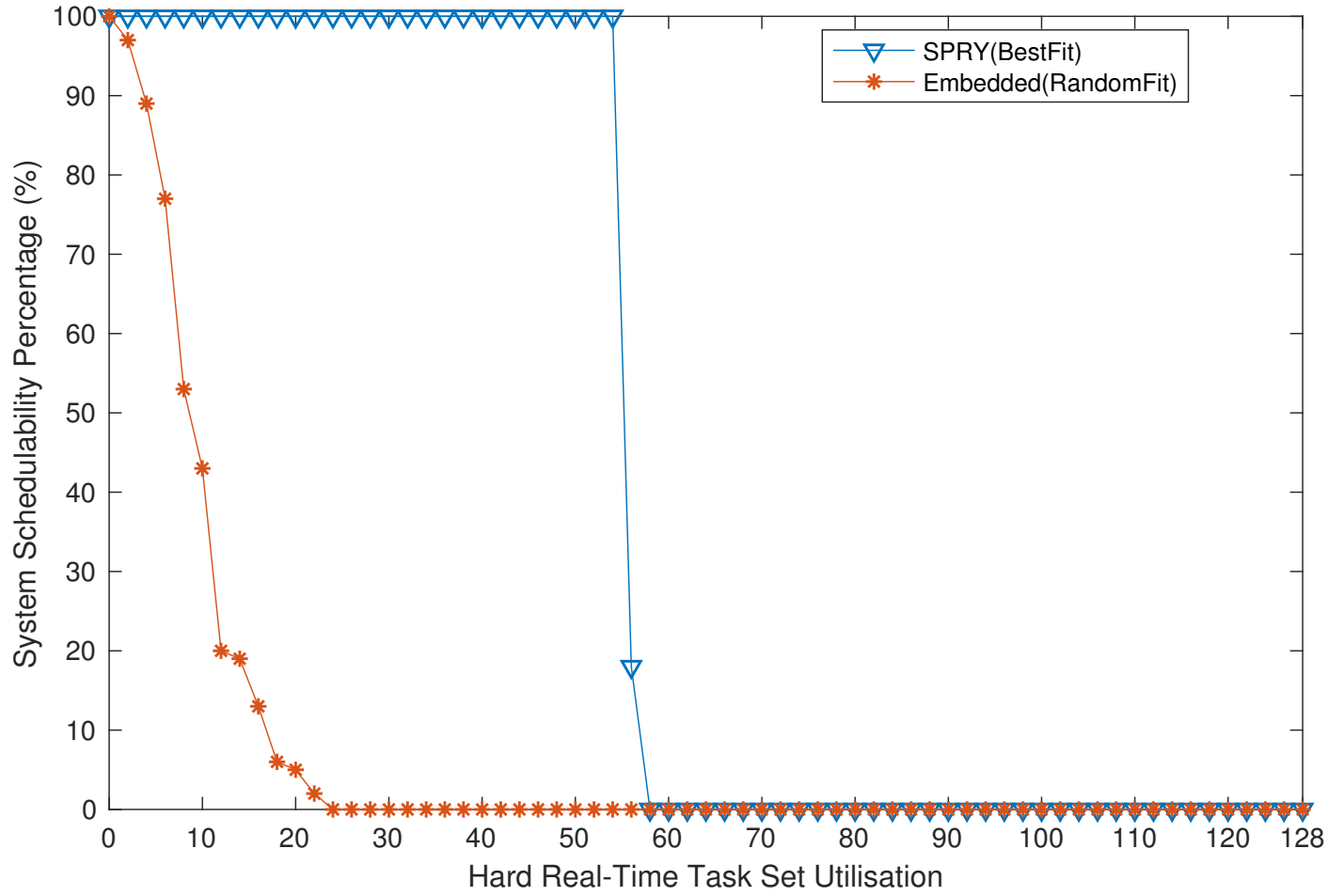
Scalability – 128 cores

1024 hard tasks

Real-time stream processing task:

- T=800
- C=56000 (U = 7000%)
- D=800

Theoretical max schedulable hard task U = 5800%



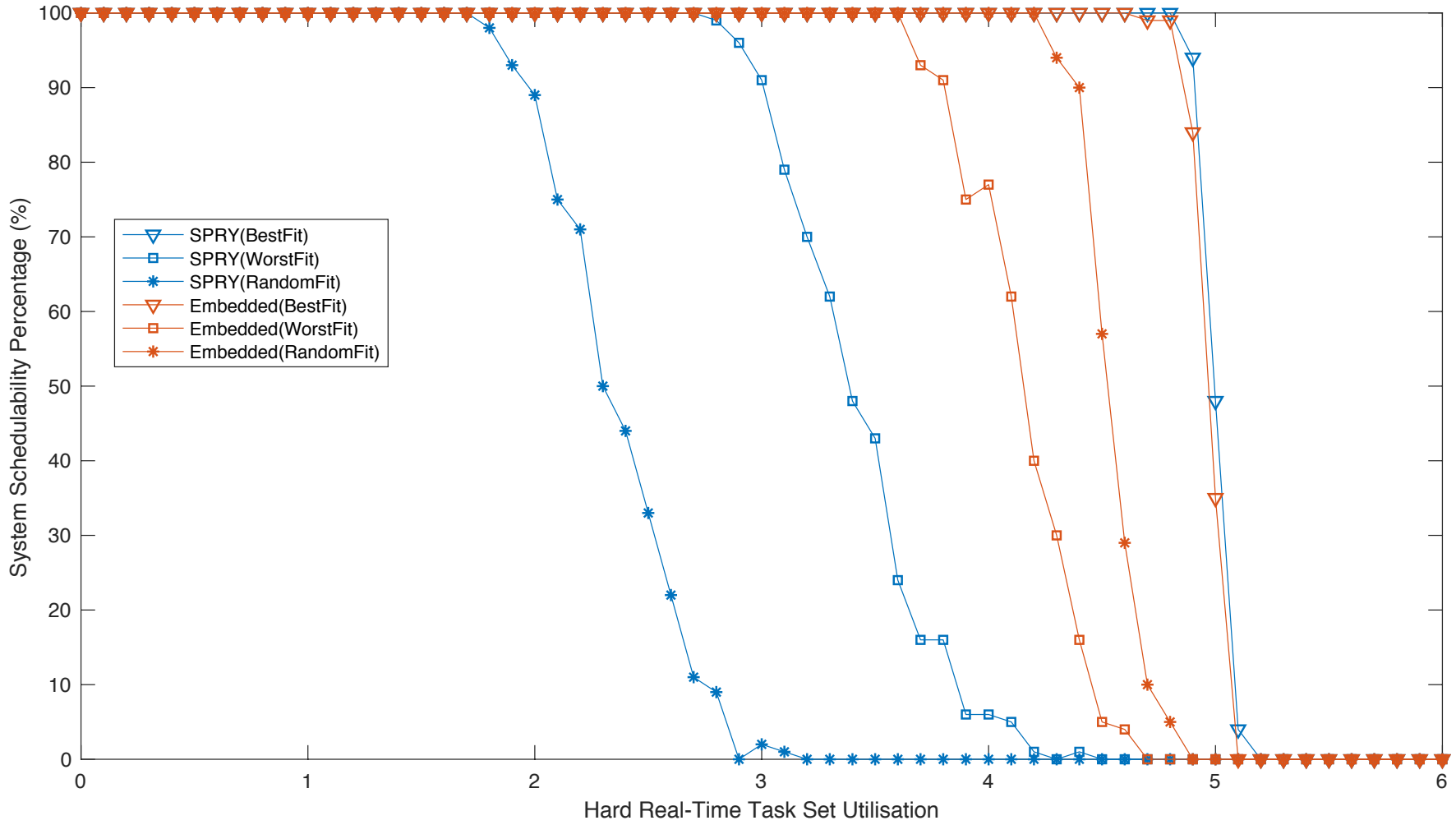
Live Streaming Data

WCET for processing each data item: 10

Min Arrival Interval: 1

Latency requirement: 30

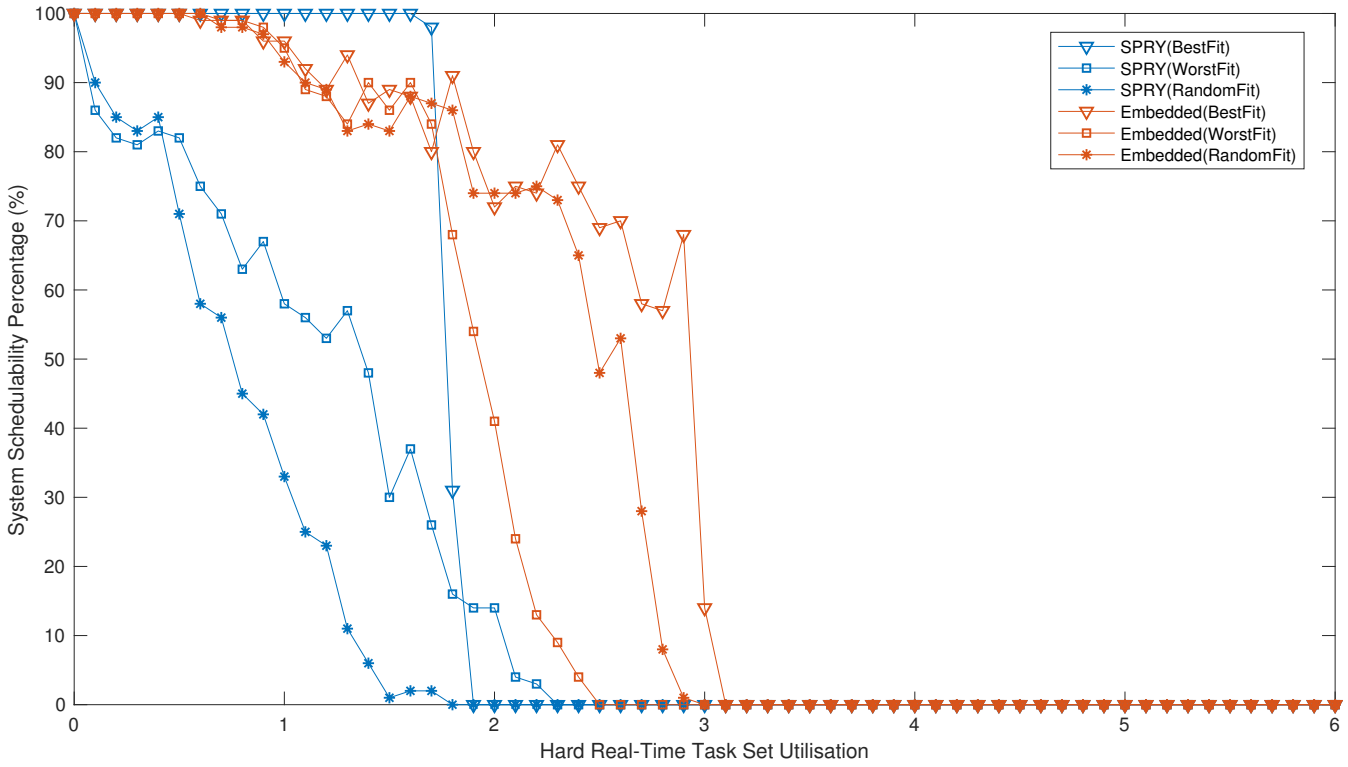
128 hard tasks



Limitation – Task Allocation

Batched data source with a period of 15 time units, 16 data partitions (WCET of each is 10 time units), and the deadline of 10 time units

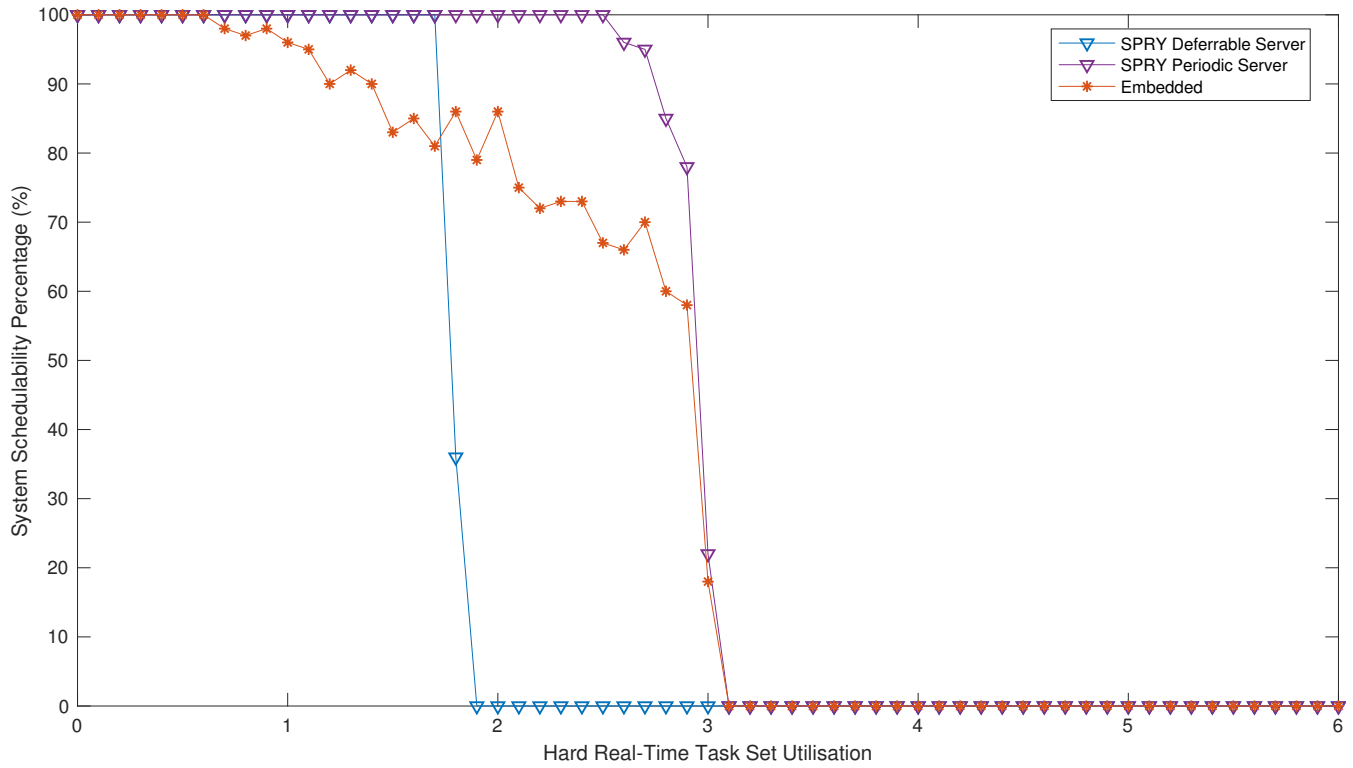
- Requires 16 Servers (T=15, C=10, D=10)



Possible Solution

1. Create 16 Servers (T=15, C=10, D=10)
2. Then allocate hard real-time tasks

- However, optimal tasks allocation order is difficult to be found, as task allocation on fully-partitioned system is NP-Hard



Limitation – Micro-Batching

The real-time micro-batching can not schedule a live streaming data source, in which the latency requirement L is less than 2 times of the WCET of processing each item C^{item}

Proof

- The response time R of processing of each micro-batching (even though the size of which is 1) ≥ C^{item}
- R should be less than or equal to the interval of micro-batching timeout →the waiting time W ≥ R
- For any data item, $R^{item} \geq C^{item}$
- Latency of the first data item $L = W + R^{item} \geq 2 \text{ X } C^{item}$

Possible Solution

Processing an item immediately without batching

Difficulty:

Generating servers for each item results in the period is very small → not practical in real-world

Preliminary Work

- Allocate item according to its arrival window
- Using same server generation
- Extend our current analysis

This work will significantly reduce the latency

Conclusions

- An architecture, which integrates real-time stream processing activities with hard real-time components
- SPRY – RTSJ implemenation (including a contribution to RTSJ 2.0)
- Server Generation Algorithm
- Real-time Stream Processing Task's Schedulability Analysis
- How to configure SPRY:
  - Scheduling the batched data parallel stream processing with a deadline
  - Scheduling the streaming data with a given latency for each item
- All the hard real-time tasks, and stream processing tasks can certainly meet their deadlines

Future Work

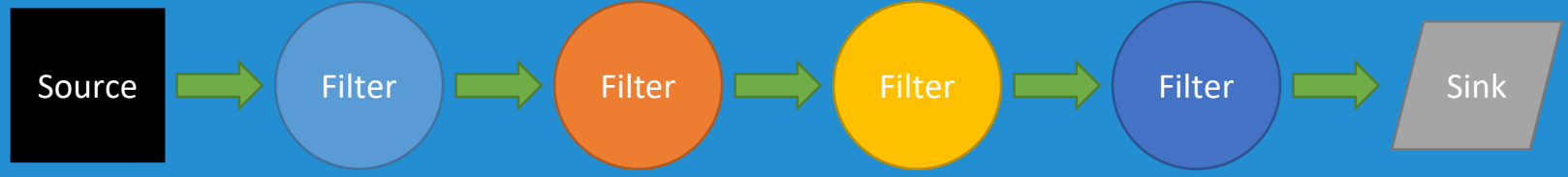
- Multiple Streams
- Supporting Many-Core/Distributed Platforms
- Live Streaming Data Processing without Micro-Batching
- Task Allocation
- Supporting GPU
- Implementation in Other Programming Languages
- Global Scheduling

Thank You

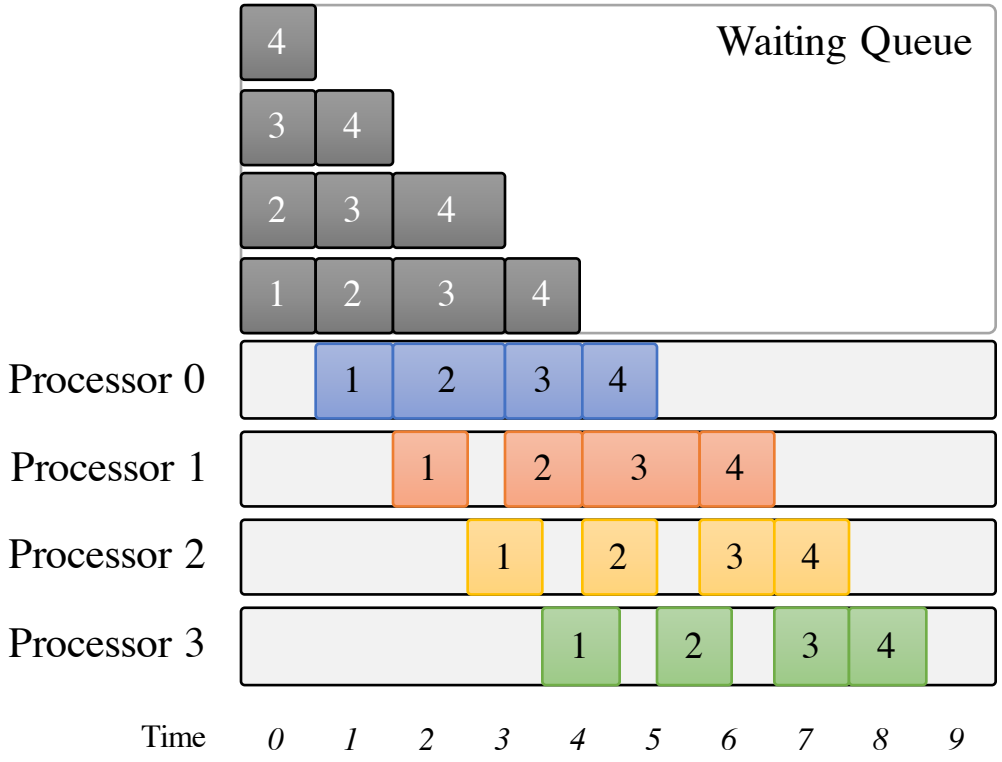
Thesis Hypothesis

- Programming languages or existing frameworks' support for stream processing is insufficient for addressing real-time requirements
- However, a generic architecture of a real-time stream processing infrastructure can be created to support predictable and analysable processing of both batched and live streaming data sources, and can be used in high-integrity real-time embedded systems
- Moreover, the architecture can be implemented as a framework using Java, with the Java Fork/Join framework and the Real-Time Specification for Java (RTSJ)

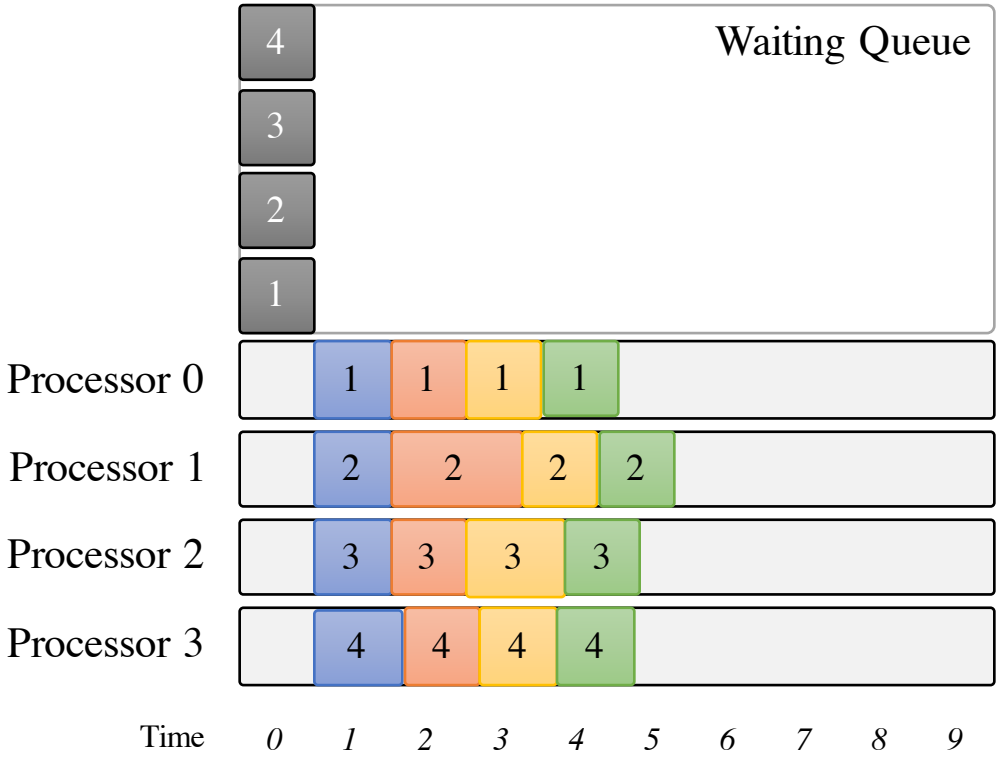
Parallel Model



Control-Parallel VS. Data-Parallel

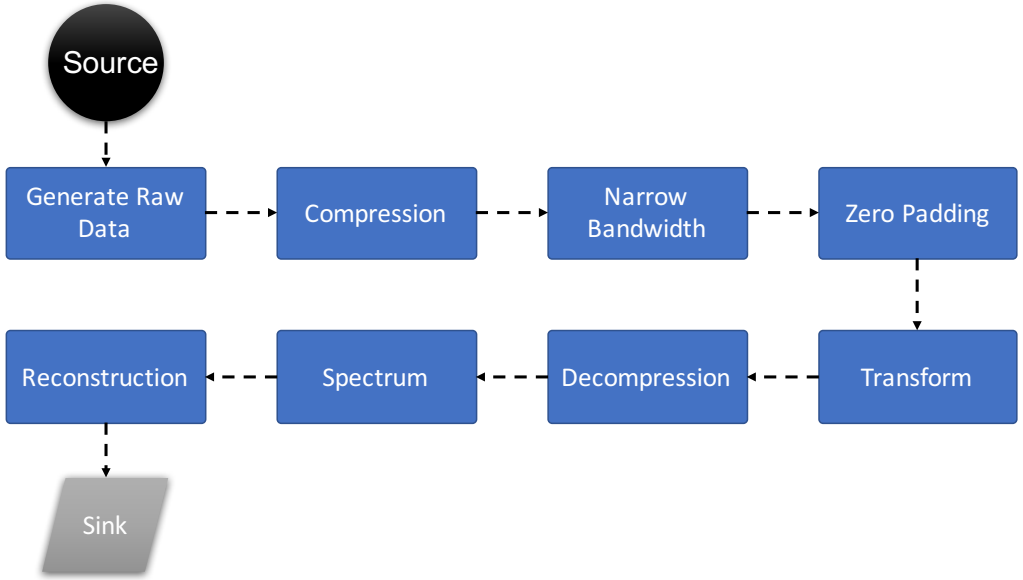


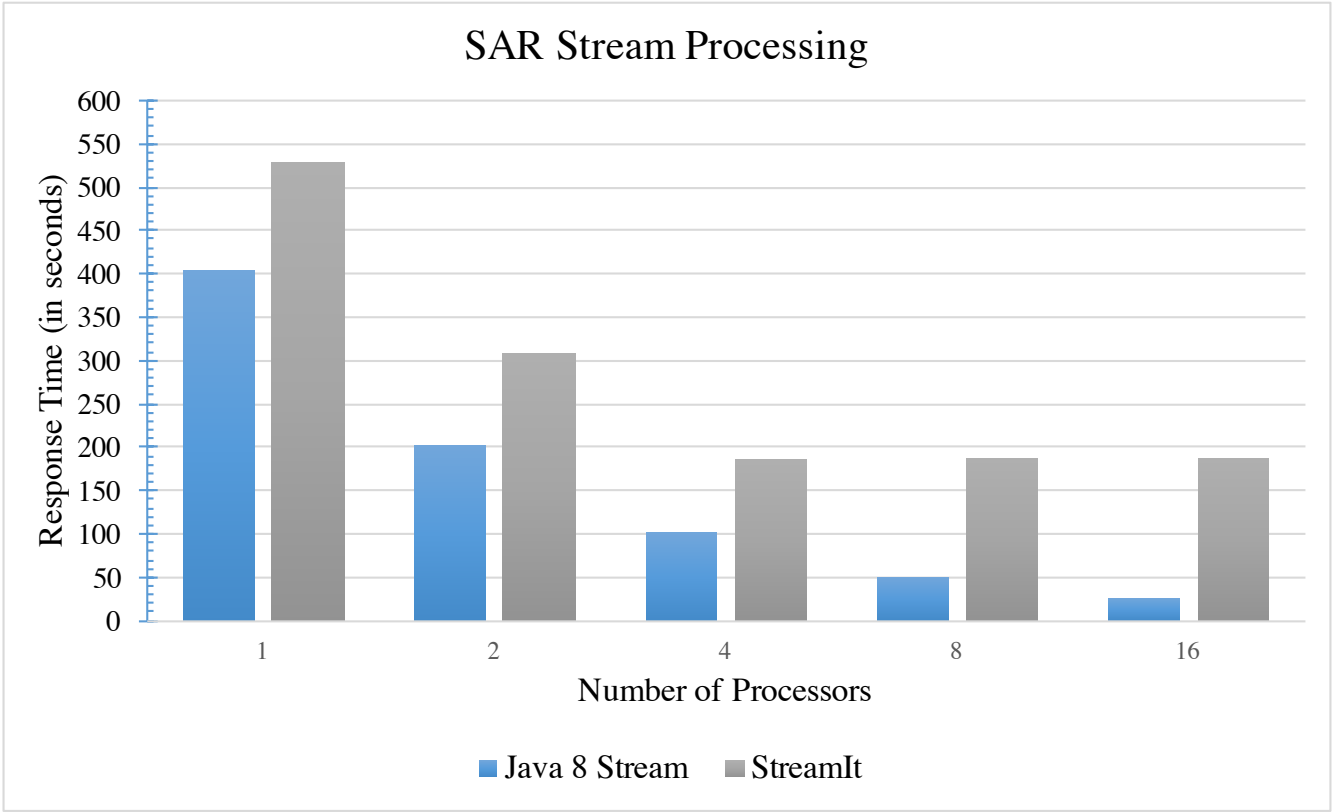
i Filter 1 processing data i
i Filter 2 processing data i
i Filter 3 processing data i
i Filter 4 processing data i



i Filter 1 processing data i
i Filter 2 processing data i
i Filter 3 processing data i
i Filter 4 processing data i

Data Parallel VS. Control Parallel





- Java 8 Streams VS. StreamIt
- Enough inputs (24 hours running)
- Run 30 times, get worst-case
- Up to 16 cores

Stream Response Time Optimisation

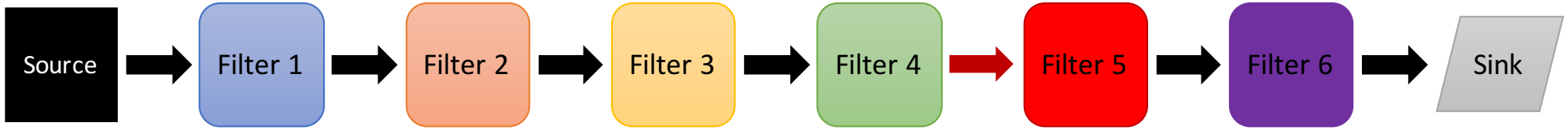
How to minimise the worst-case response time of a stream processing

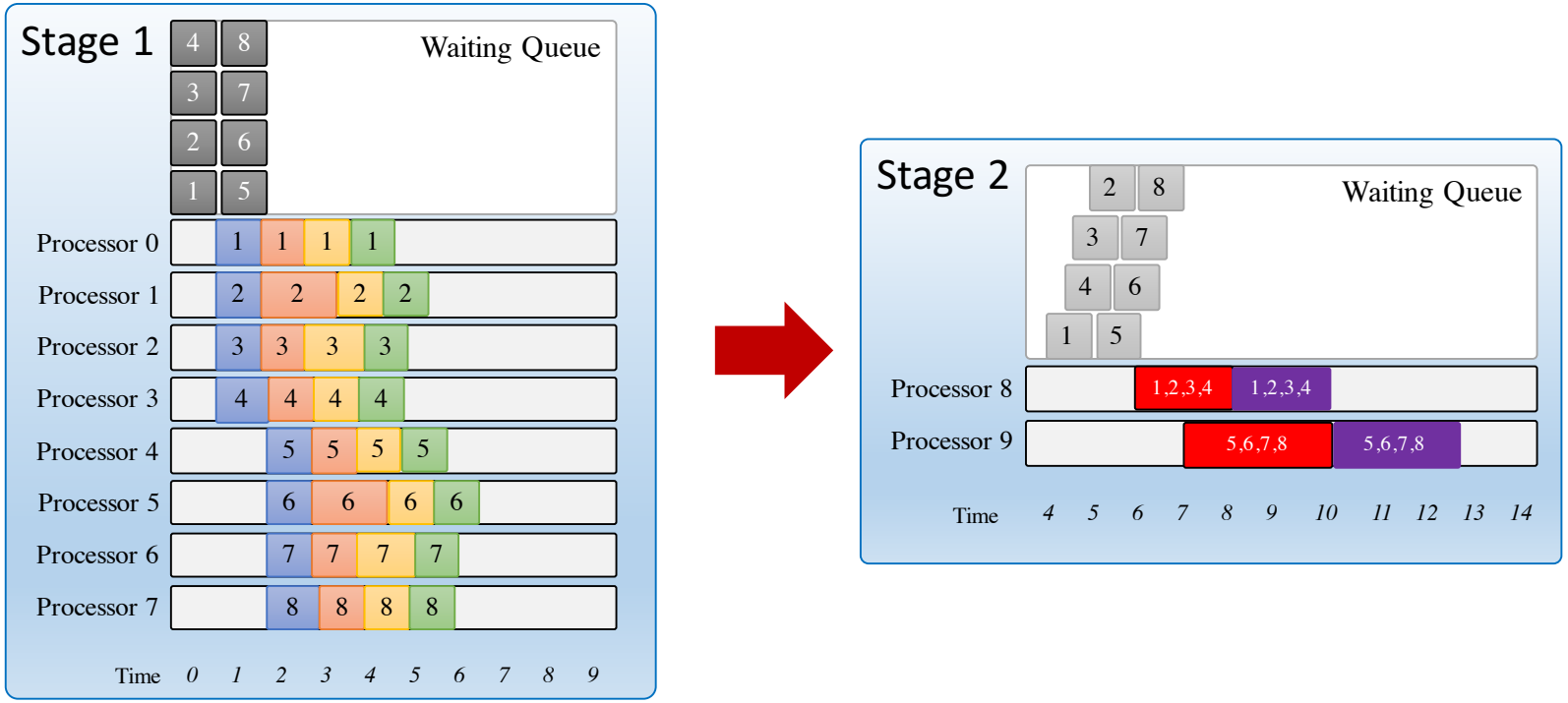
- Introduce an artificial deadline for the task
- Slightly decrease that artificial deadline until the task cannot be scheduled using the above analysis techniques
- The minimum possible worst-case response time or latency can be obtained from the analysis upon the task with the deadline just before the unschedulable artificial deadline

Backup Slides

Parallel Model

Hybrid





i Filter 1 processing data i
i Filter 2 processing data i
Filter 5 processing a data collection
i Filter 3 processing data i
i Filter 4 processing data i
Filter 6 processing a data collection

Real-Time Stream Processing Code Example

```
PriorityParameters priority;      /* The priority */
BitSet affinities;                /* All the allocated processors */
ProcessingGroup[] servers;        /* Execution-Time Servers */
ArrayList<String> data;           /* A batched data source */
DataAllocationPolicy dap;         /* Data allocation policy */
int prologueProcessor;            /* The prologue processor */
long count;
SPRYEngine<String> spry = new SPRYEngine<>(
        priority,                 /* The processing pipeline */
        p -> p.flatMap(line -> Stream.of(line.split("\\W+"))).countDeferred(), dap, affinities,
prologueProcessor, servers);
/* set the deadline and its miss handler */
spry.setDeadlineMissHandler(deadline, deadlineMissHandler);
/* set the period and its violation handler */
spry.setMITViolateHandler(period, MITViolateHandler);
/* set the call back to get the result */
spry.setCallback(r -> count = r);
/* Within a real-time thread, perform the real-time stream process for the data by invoking the
SPRYEngine */
spry.processBatch(data);
```

Research Goals

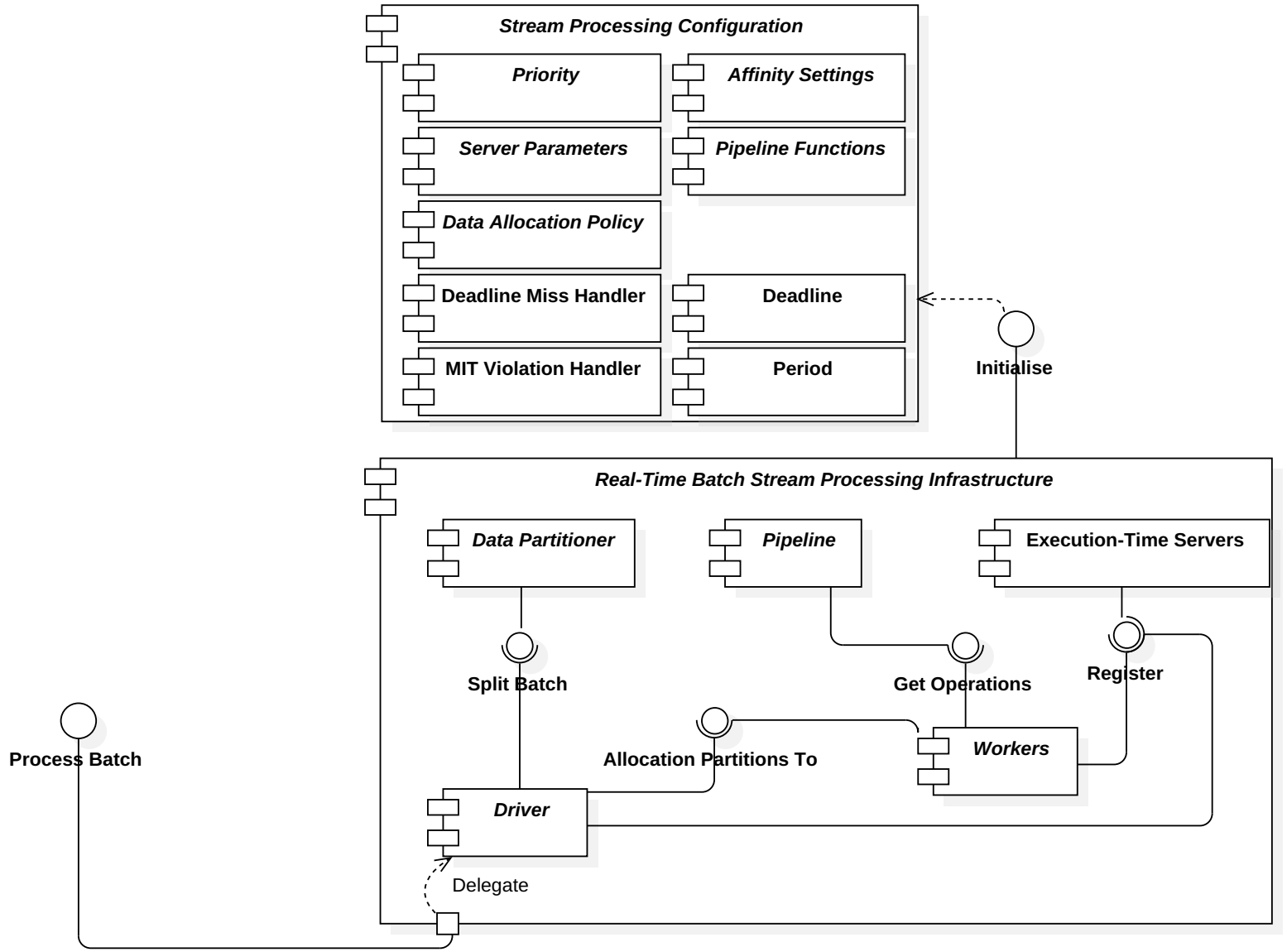
Provide a real-time stream processing framework + schedulability analysis tools, so that

Users can create real-time streaming applications just with the following 3 steps:

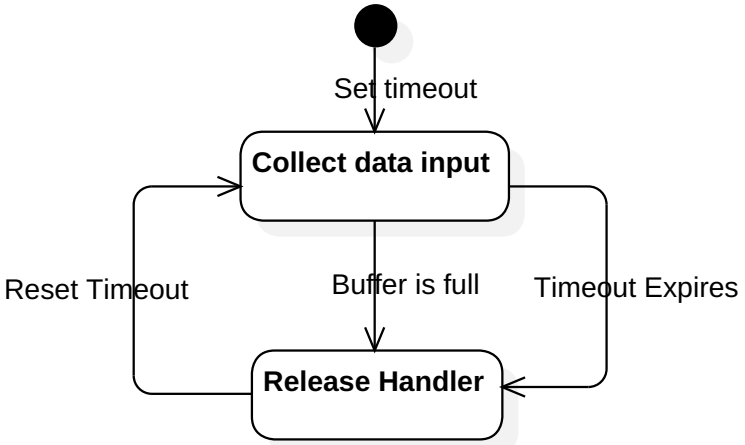
1. Build conceptual model, i.e., streaming task, and other RT tasks
2. Using tools for schedulability test
3. Program the systems using the framework with very concise code

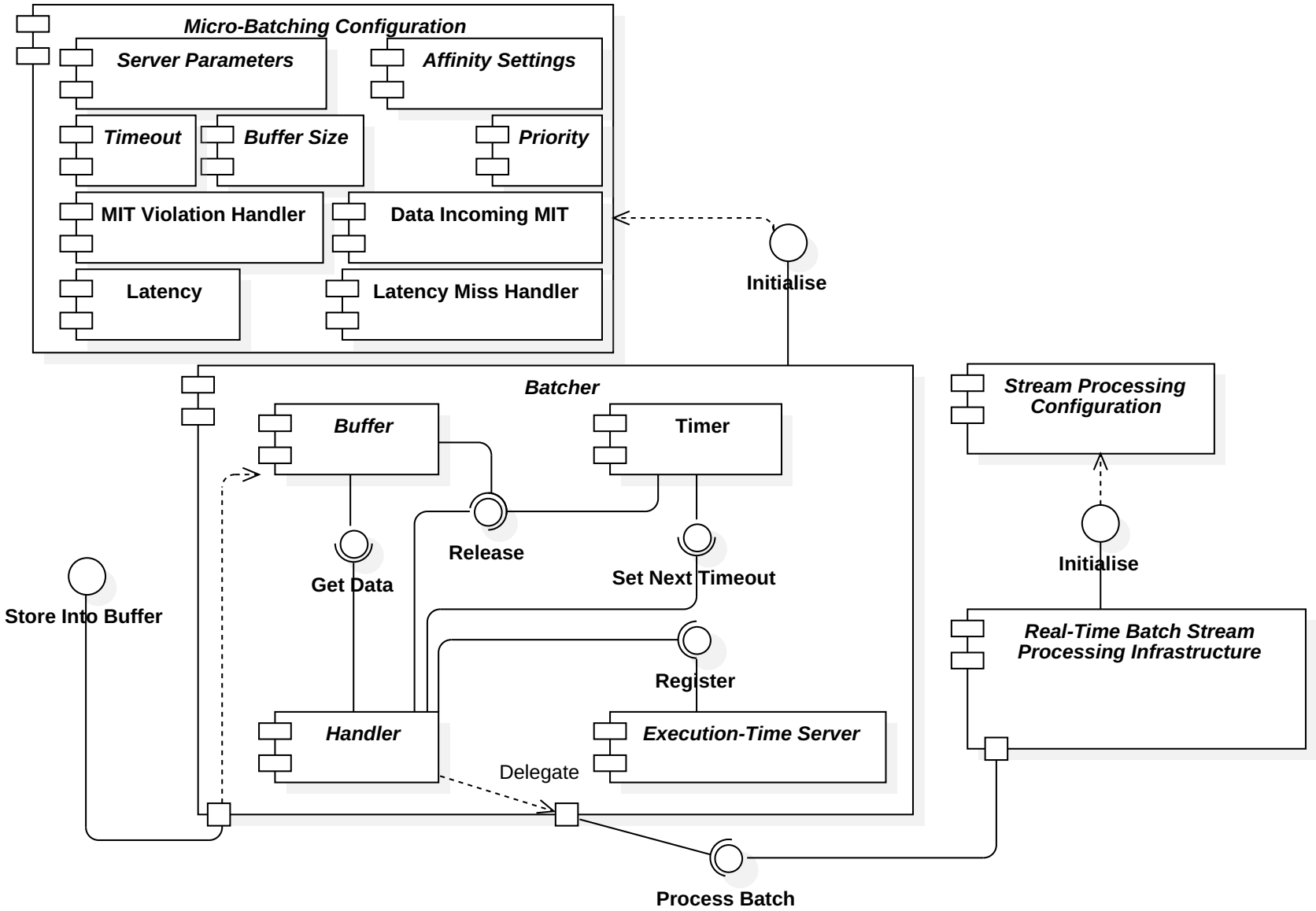
The framework provides tools to generate configurations, e.g., Priorities, Execution-time Server Parameters, Data Partition Policy

RT Streaming Infrastructure Architecture



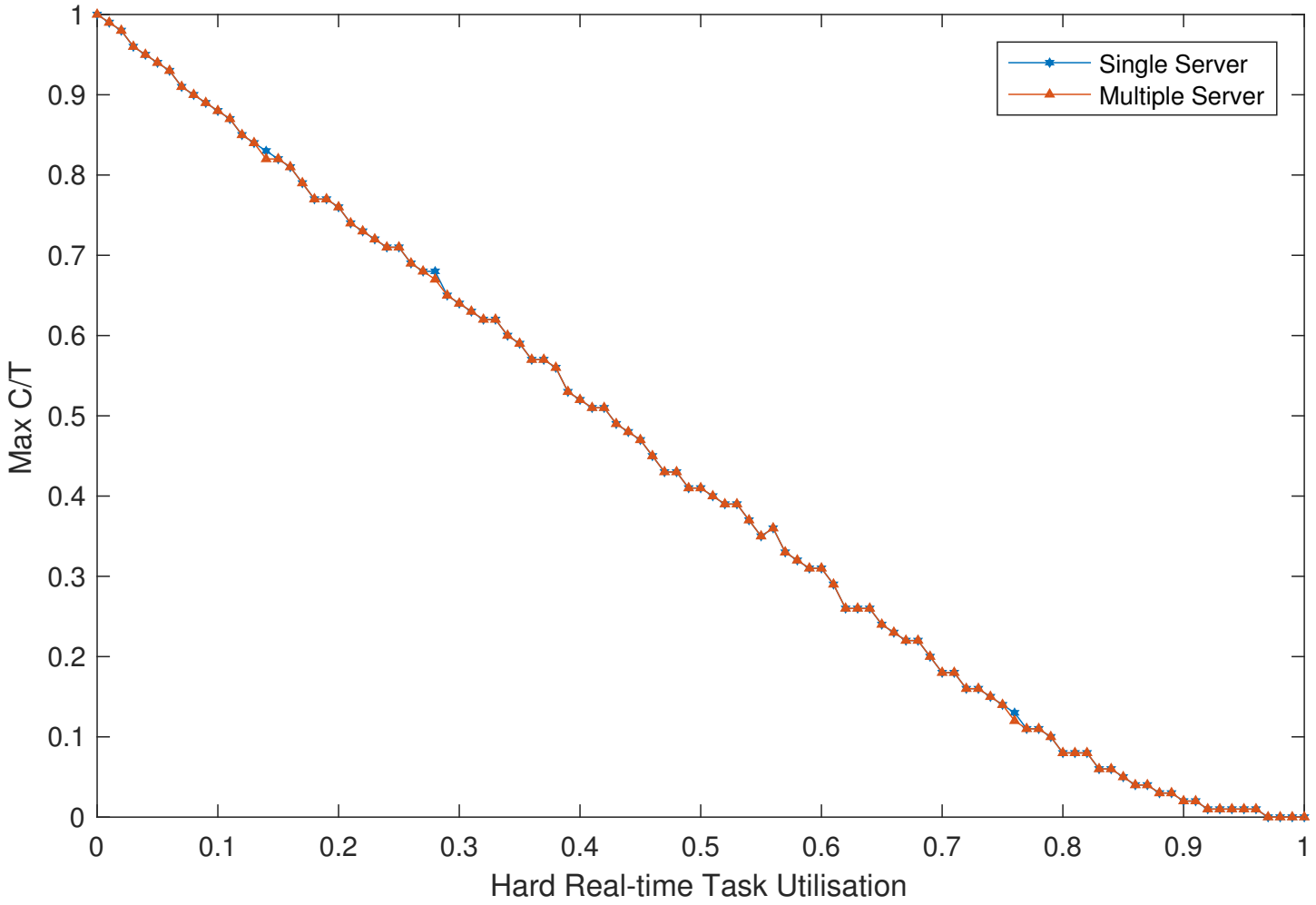
Real-time Micro-Batching Architecture





Single Server VS. Multiple Servers

Will multiple servers claims more computation time?



- Nearly No