PREM-based Optimal Task Segmentation Under Fixed Priority Scheduling

Presented by: Giovani Gracioli

Authors: Muhammad R. Soliman

Rodolfo Pellizzoni

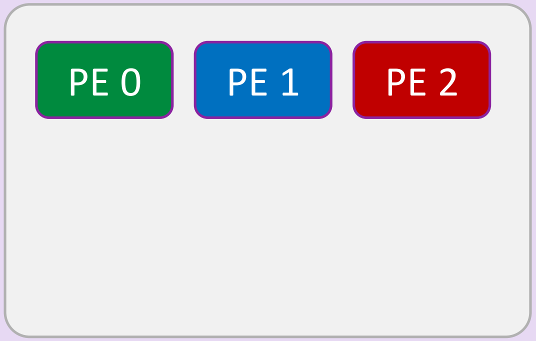
ECRTS
9-12 July
Stuttgart, Germany
2019
31th Euromicro Conference on Real-Time Systems
9-12 July 2019 | Stuttgart, Germany
BOSCH

Outline

- Introduction
- Task Model
- Schedulability Analysis
- Task Set Segmentation
- Program Segmentation
- Evaluation
- Conclusion and Future Work

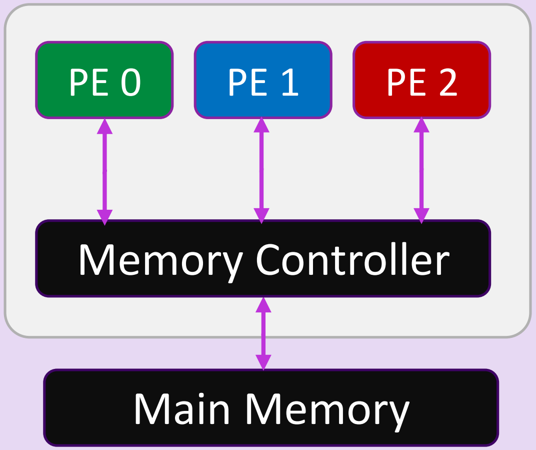
Introduction: MPSoC / PREM

Multi-Processor System-on-Chip

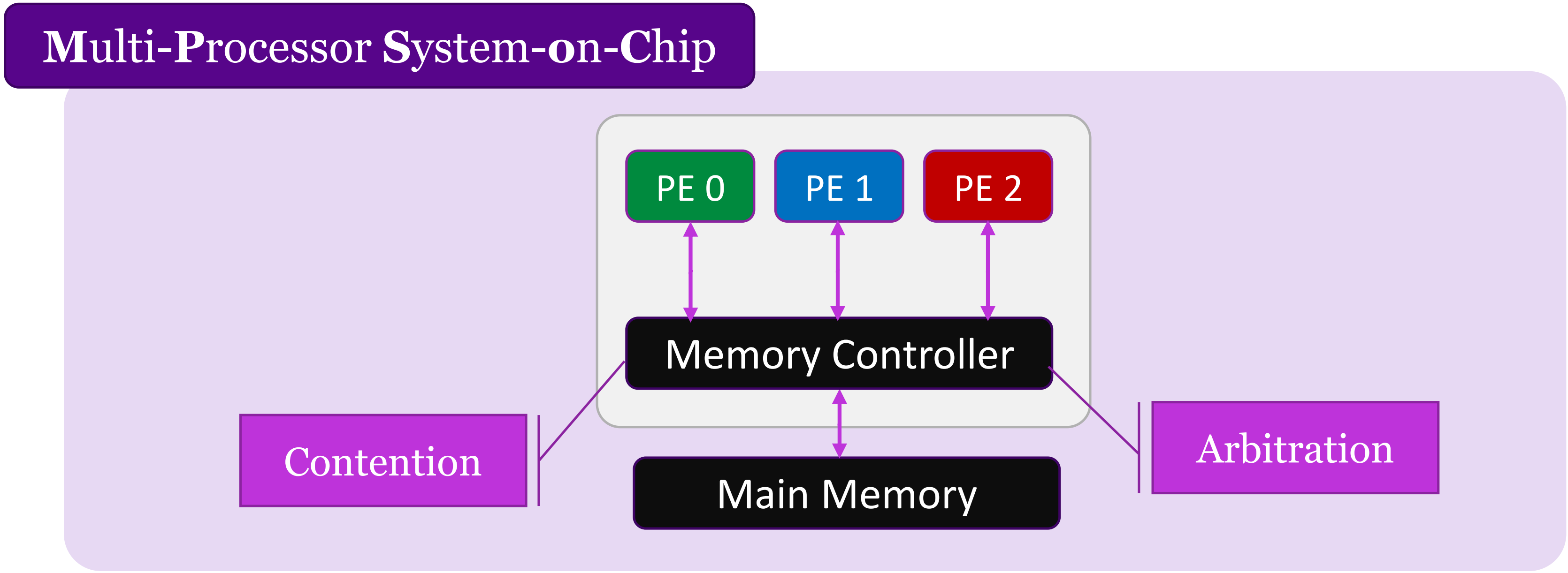


Introduction: MPSoC / PREM

Multi-Processor System-on-Chip

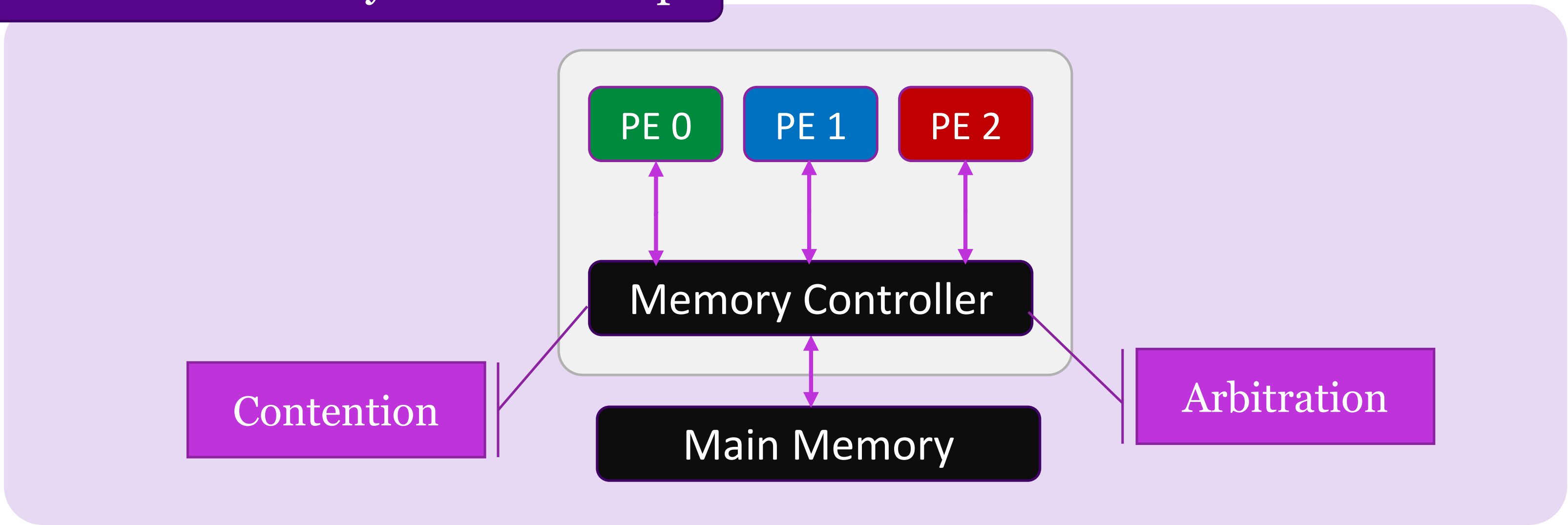


Introduction: MPSoC / PREM



Introduction: MPSoC / PREM

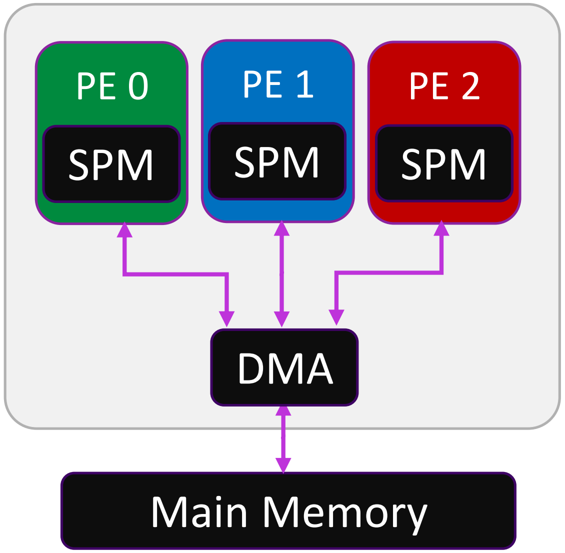
Multi-Processor System-on-Chip



PRedictable Execution Model

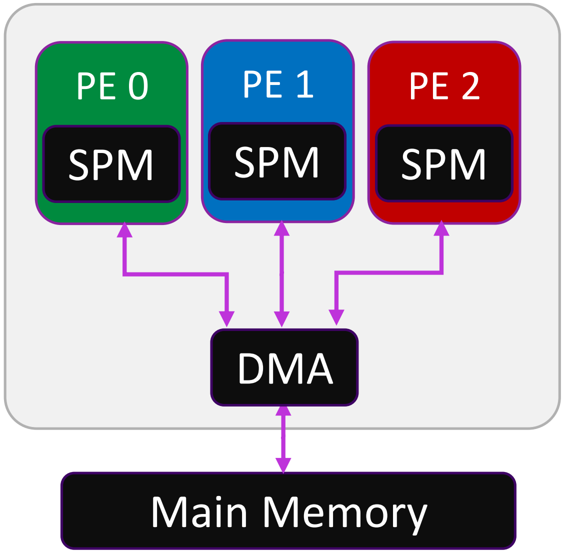
Memory / Computation = Memory + Computation

Introduction: PREM (3-Phase Model)



Introduction: PREM (3-Phase Model)

↑Load Compute ↓Unload

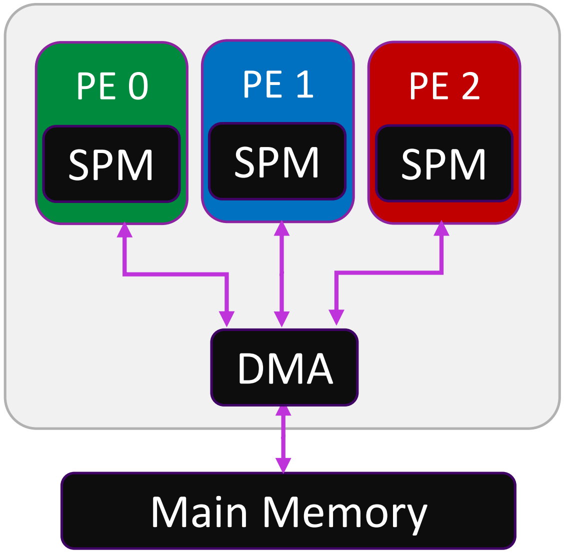


Introduction: PREM (3-Phase Model)

↑Load | Compute | ↓Unload

↑Load

Load the required code and data to SPM

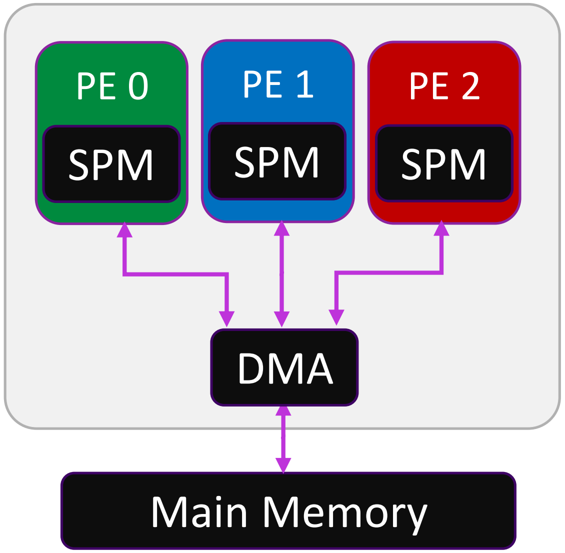


Introduction: PREM (3-Phase Model)

↑Load Compute ↓Unload

↑Load — Load the required code and data to SPM

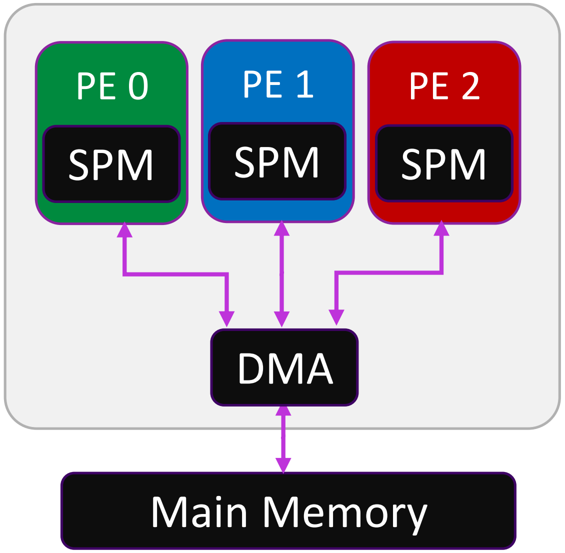
Compute — Execute the task directly from SPM



Introduction: PREM (3-Phase Model)

↑Load Compute ↓Unload

↑Load	Load the required code and data to SPM
Compute	Execute the task directly from SPM
↓Unload	Write-back the modified data



PE 0

SPM

PE 1

SPM

PE 2

SPM

DMA

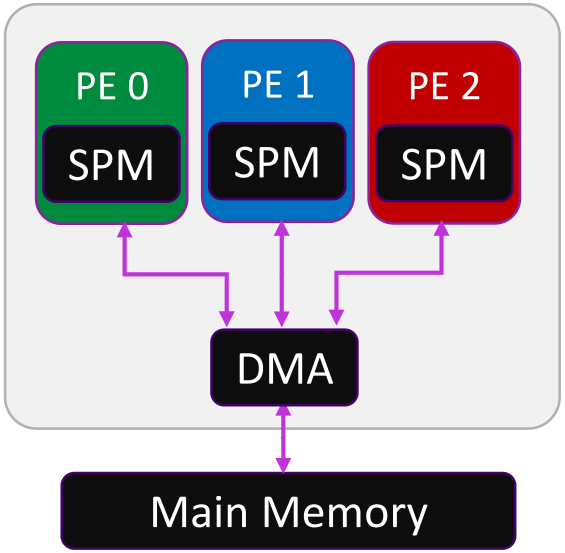
Main Memory

Introduction: PREM (3-Phase Model)

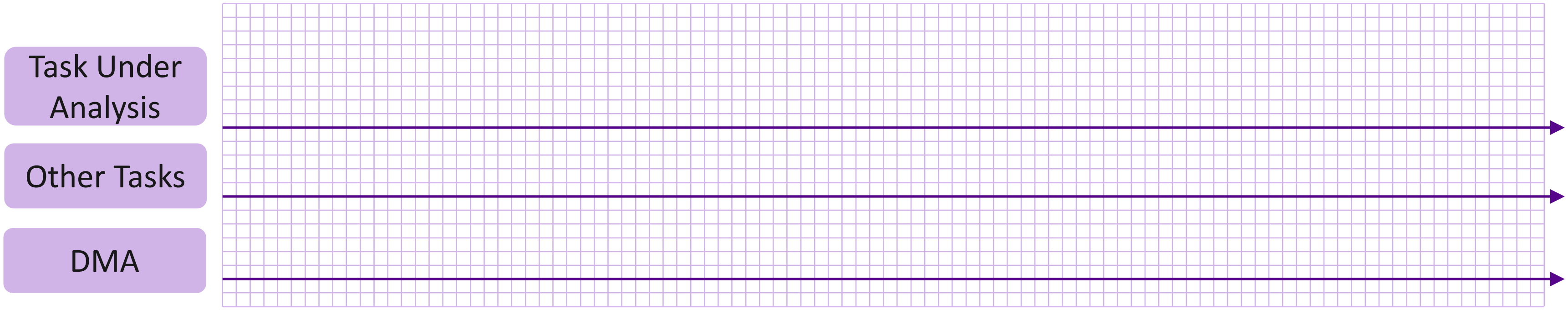
↑Load Compute ↓Unload

↑Load	Load the required code and data to SPM
Compute	Execute the task directly from SPM
↓Unload	Write-back the modified data

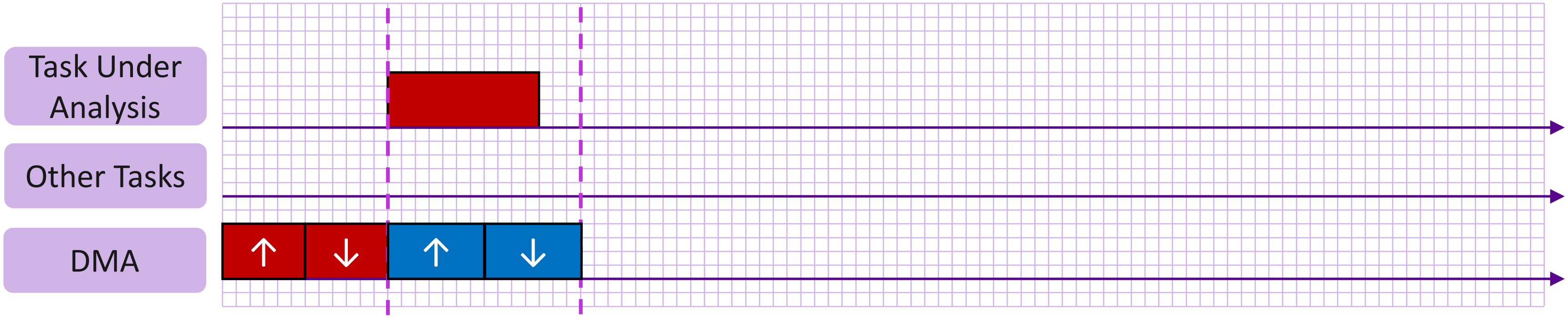
A single memory phase is executed at any one time in the system.



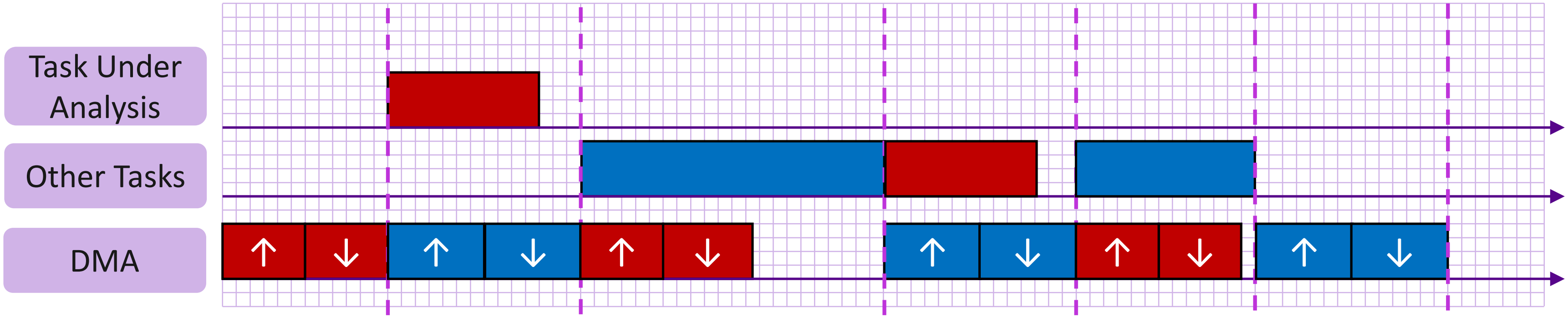
Introduction: PREM (3-Phase Model)

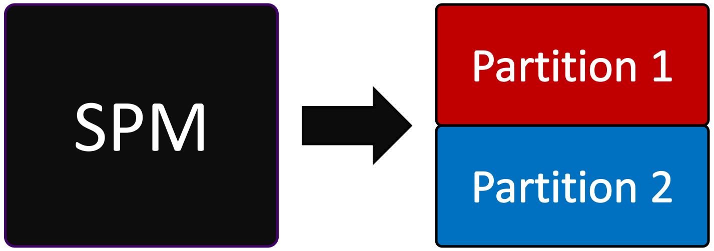


Introduction: PREM (3-Phase Model)

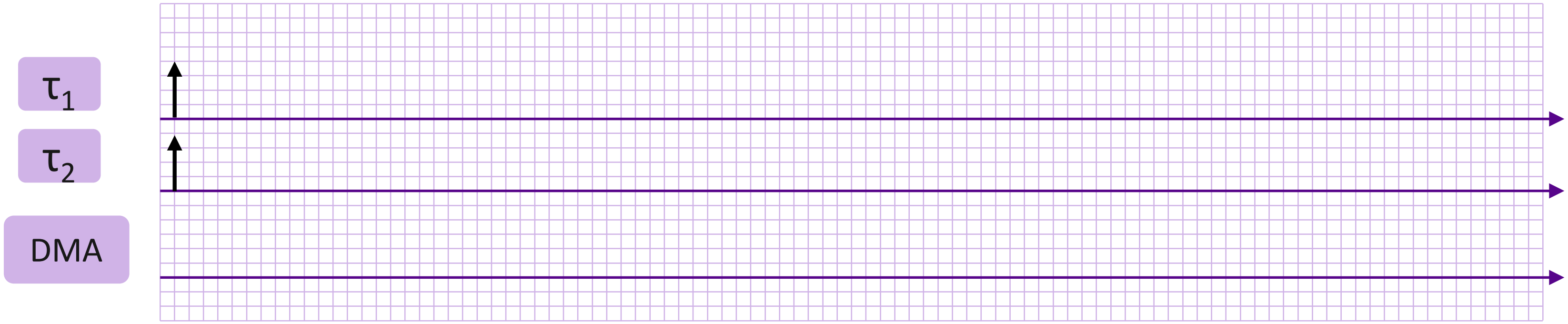


Introduction: PREM (3-Phase Model)

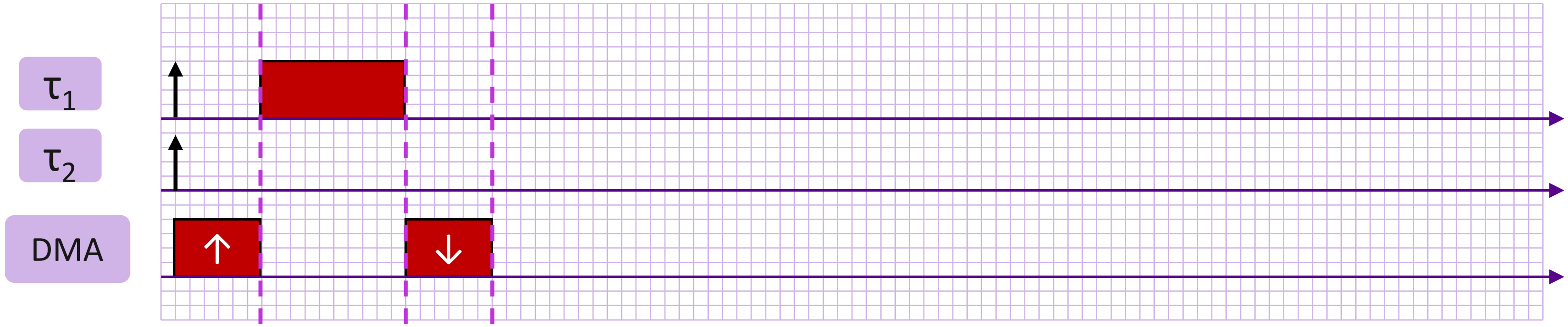




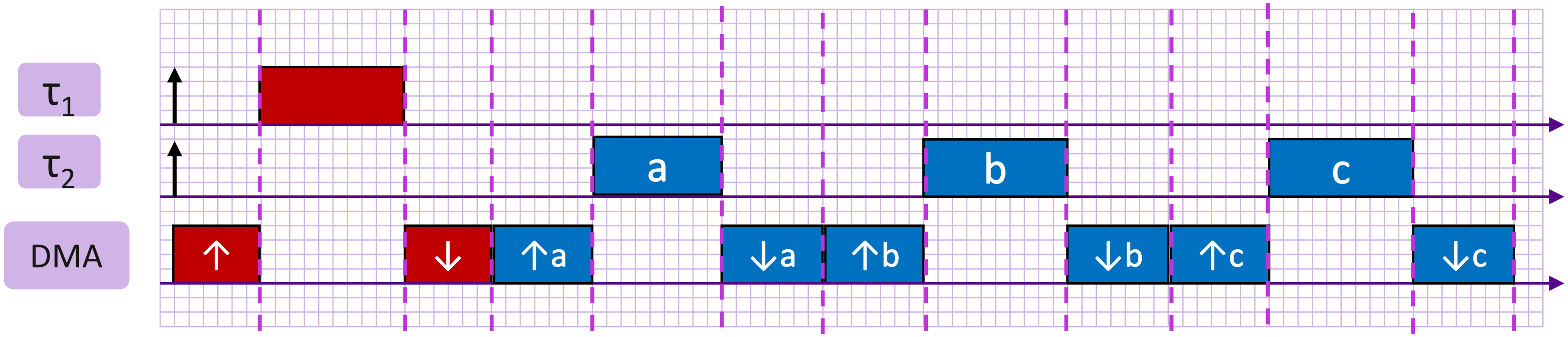
Introduction: PREM (3-Phase Model)



Introduction: PREM (3-Phase Model)

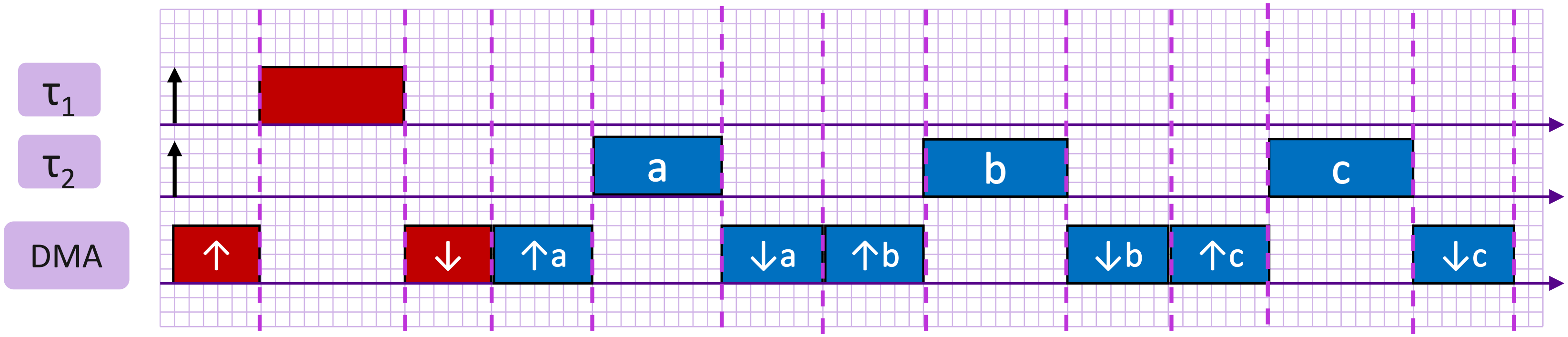


Introduction: PREM (3-Phase Model)



- Segmentation:
  - Large code / data footprint → do not fit in SPM.
  - Data accesses are input-dependent → only known at run-time

Introduction: PREM (3-Phase Model)

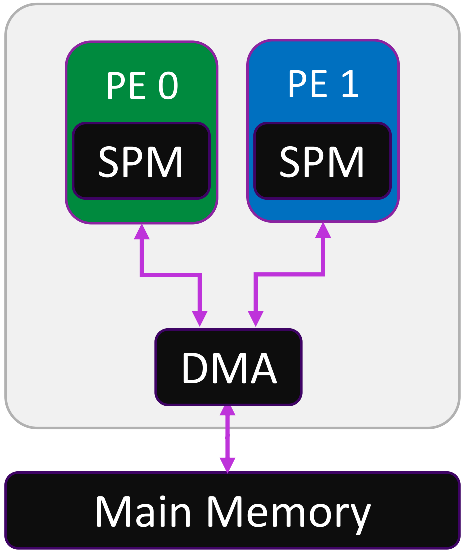


- Segmentation:
  - Large code / data footprint → do not fit in SPM.
  - Data accesses are input-dependent → only known at run-time

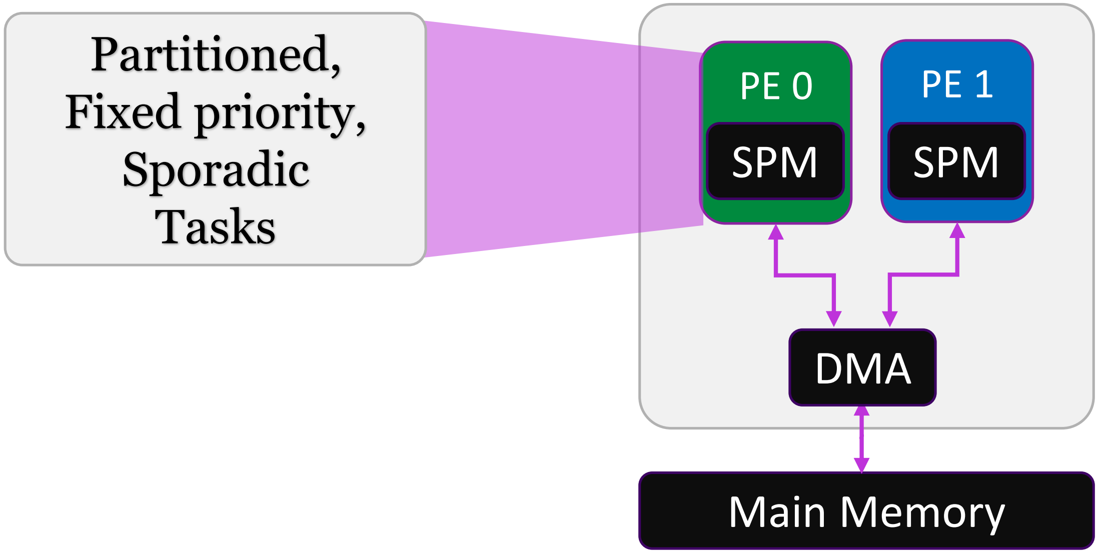
Contribution

How to compile a program based on PREM?

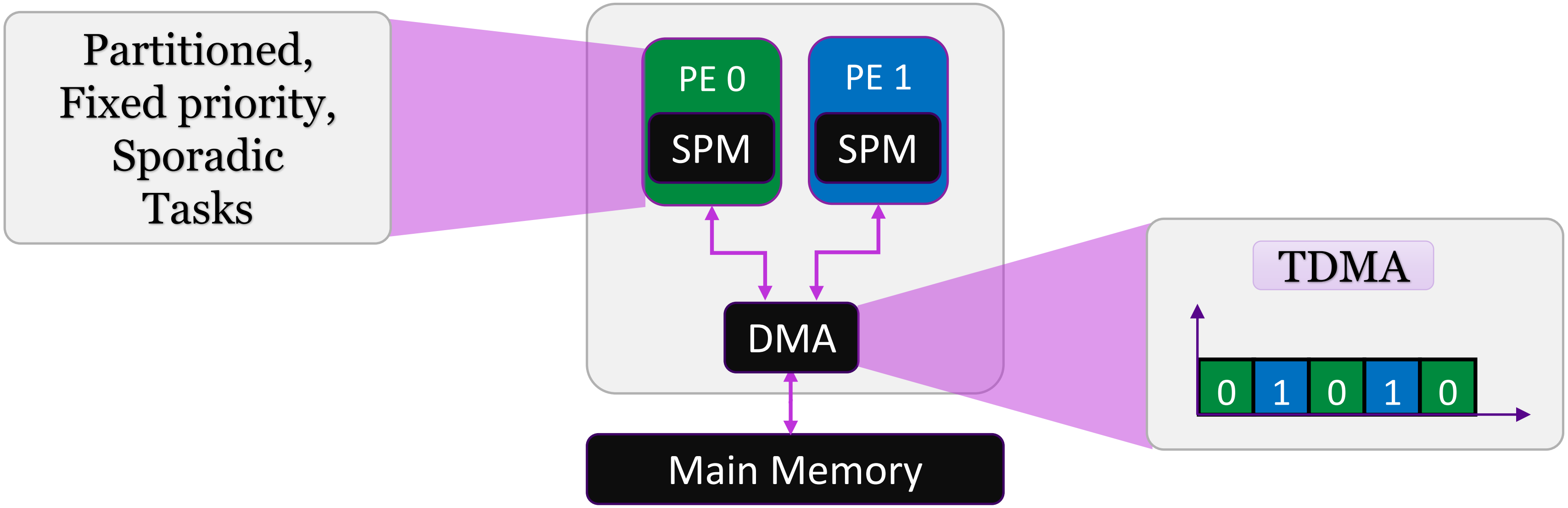
Introduction: Processor / Memory Schedule



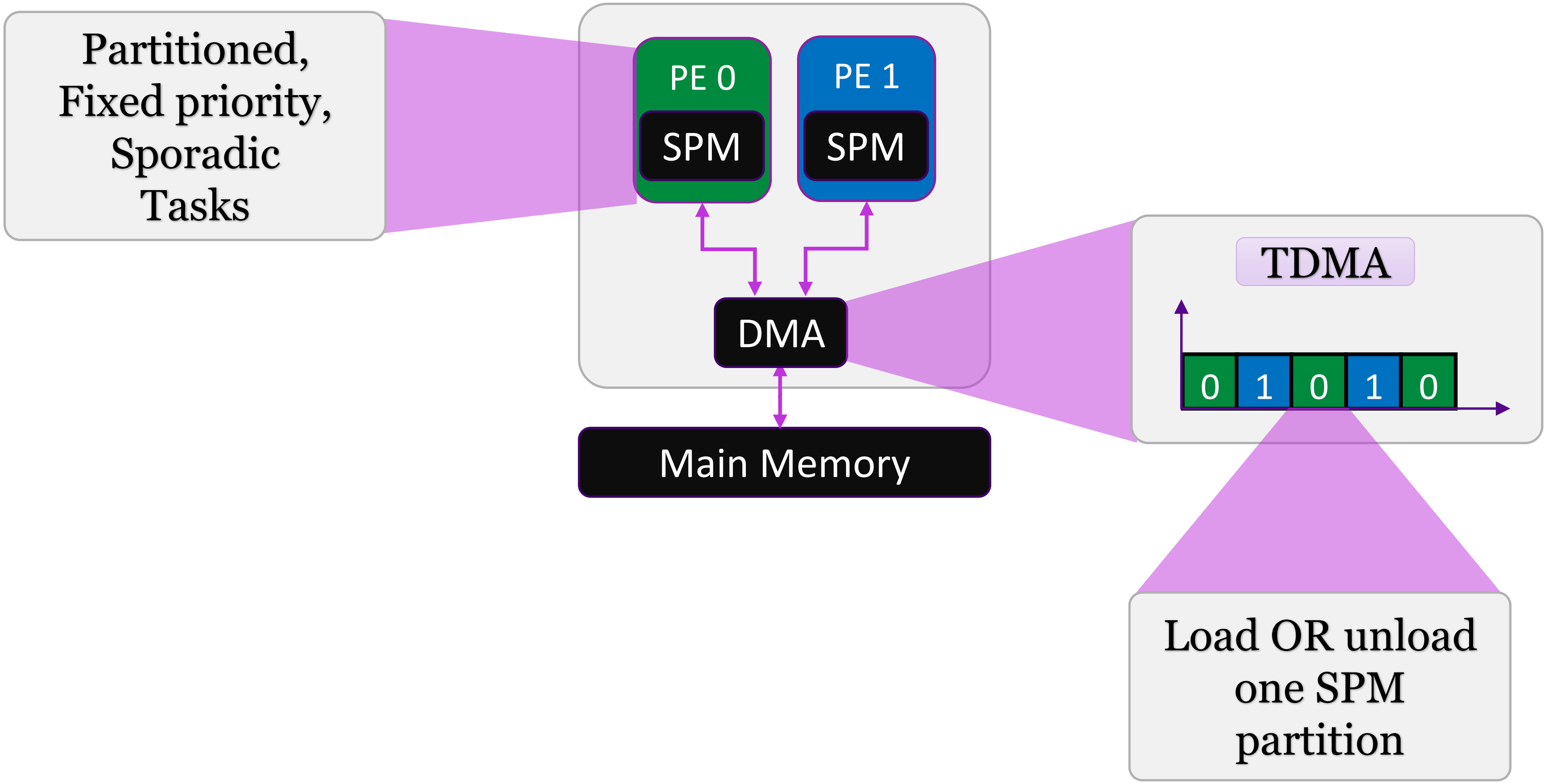
Introduction: Processor / Memory Schedule



Introduction: Processor / Memory Schedule



Introduction: Processor / Memory Schedule



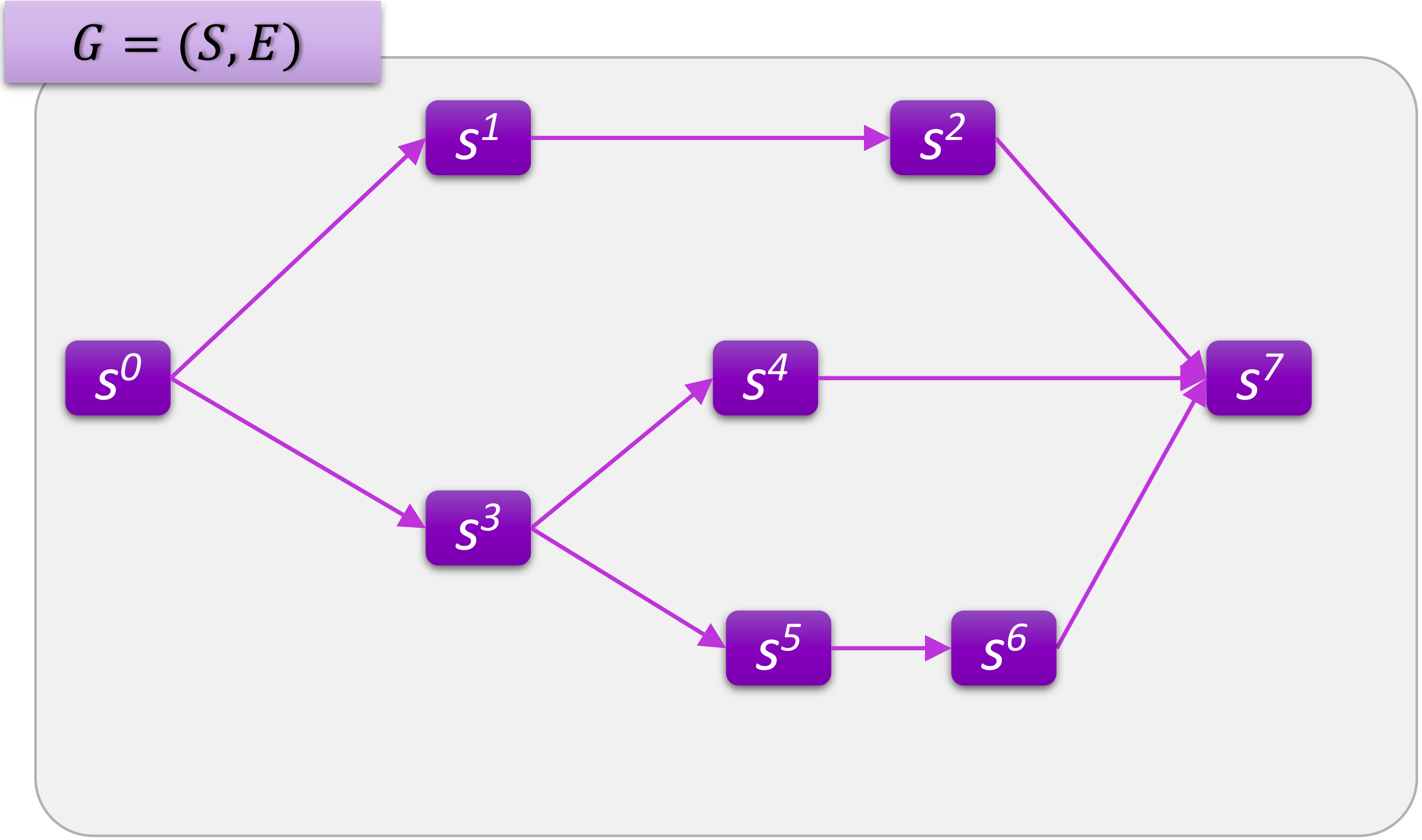
Partitioned, Fixed priority, Sporadic Tasks

Load OR unload one SPM partition

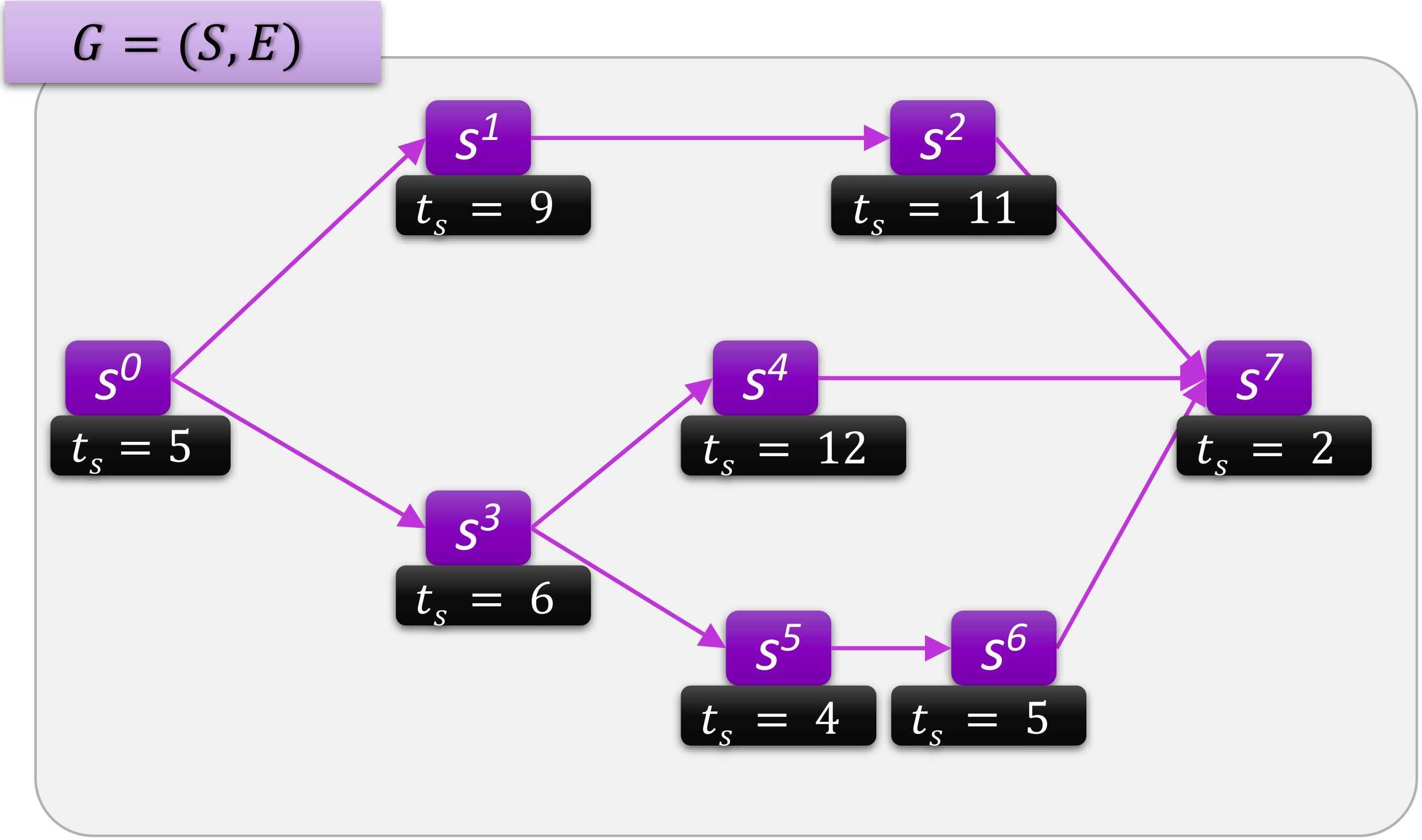
Task Model

- Sequential, conditional PREM tasks
- Non-preemptive segment execution
- Each task has a period T_i and a deadline $D_i <= T_i$
- Fixed memory time Δ to load/unload each segment
  - For a TDMA slot σ and M processors: $\Delta = (M+1)^* \sigma$

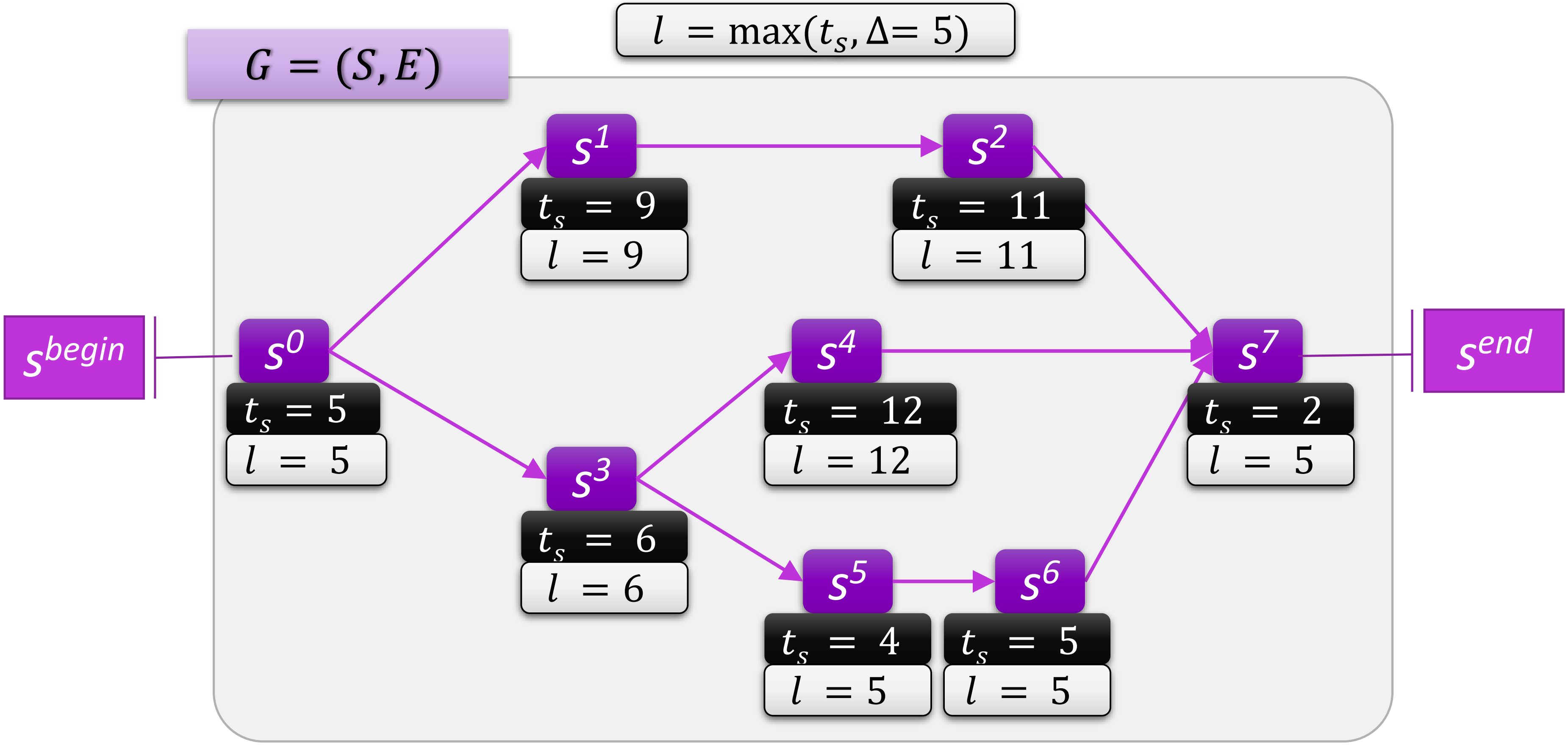
Task Model: DAG Representation



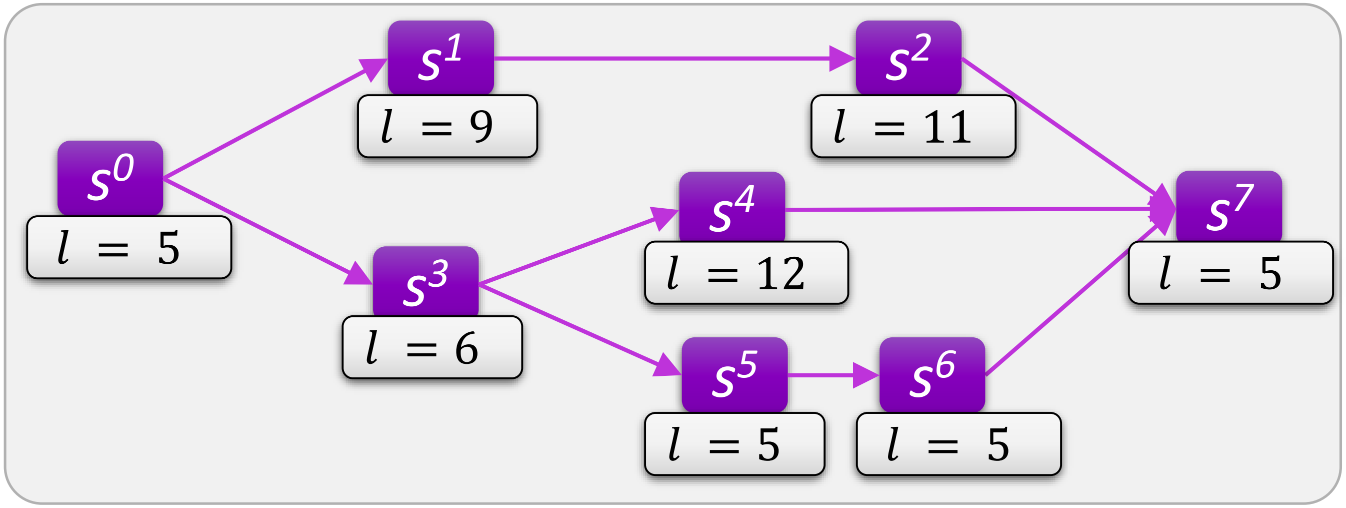
Task Model: DAG Representation



Task Model: DAG Representation

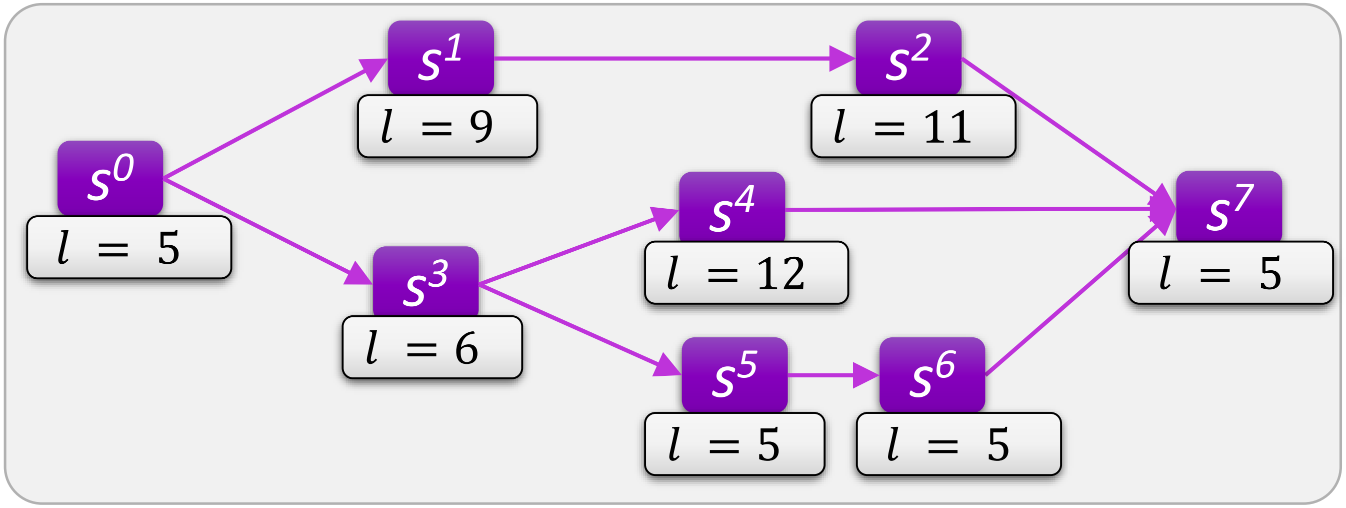


Task Model: Paths



p s^1 s^2 $I = 2$ $L = 20$ $end = 11$

Task Model: Paths



P	s^0	s^1	s^2	s^7		$I = 4$	$L = 30$	$end = 5$
P'	s^0	s^3	s^4	s^7		$I = 4$	$L = 28$	$end = 5$
P''	s^0	s^3	s^5	s^6	s^7	$I = 5$	$L = 26$	$end = 5$

Task Model: Path/DAG Domination

$P' \geq P$ $\quad$ $P'.I \geq P.I$ & $P'.L \geq P.L$ & $P'.end \leq P.end$

- If neither $P' \geq P$ nor $P \geq P'$, P' and P are incomparable.
- A DAG can be characterized by its dominating maximal paths G.C which replaces the concept of WCET for sequential programs.
- If it is possible to choose between two paths, a dominated path is (better) than the dominating path.

Task Model: Path/DAG Domination

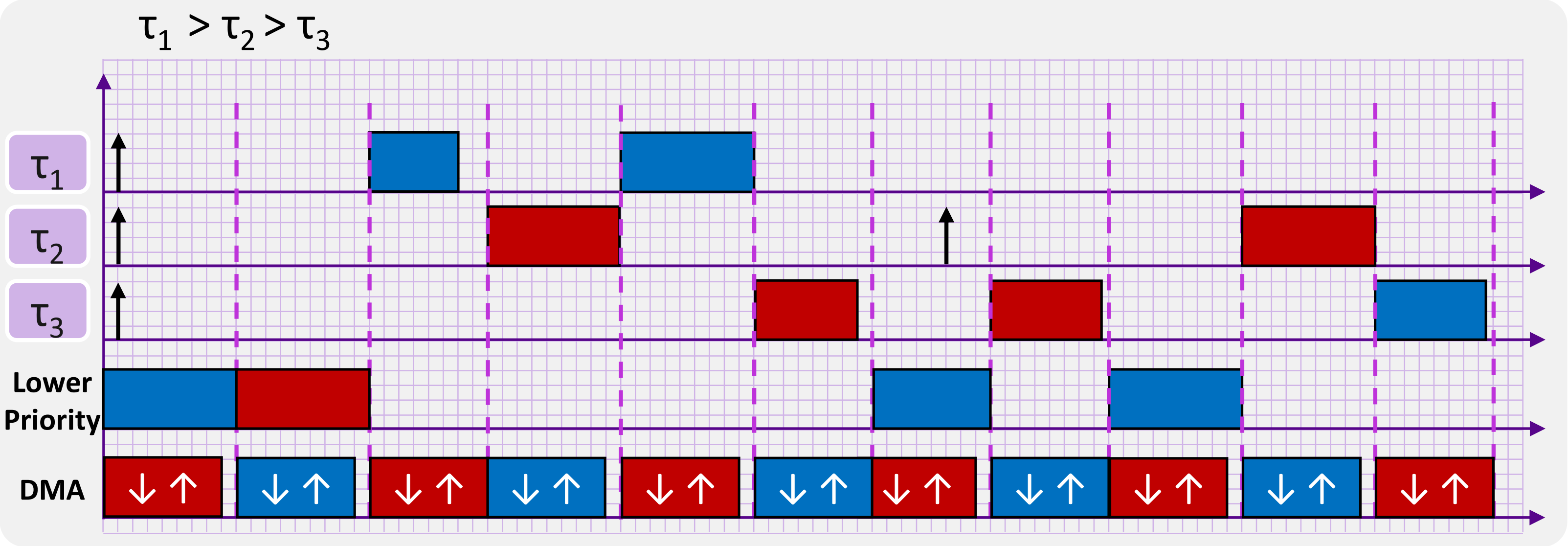
$P' \geq P$ $\quad$ $P'.I \geq P.I$ & $P'.L \geq P.L$ & $P'.end \leq P.end$

- If neither $P' \geq P$ nor $P \geq P'$, P' and P are incomparable.
- A DAG can be characterized by its dominating maximal paths G.C which replaces the concept of WCET for sequential programs.
- If it is possible to choose between two paths, a dominated path is (better) than the dominating path.

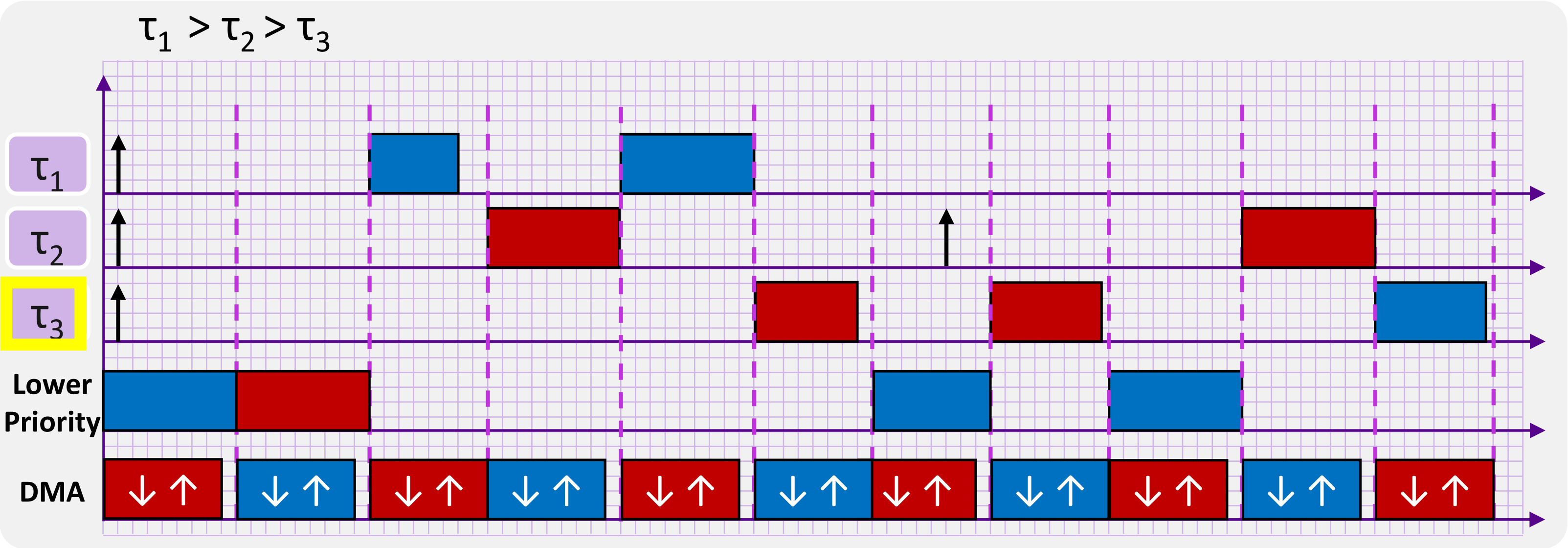
$G' \geq G$ $\quad$ $\forall P \in G, \exists P' \in G':$ $\quad$ $P' \geq P$

- If neither $G' \geq G$ nor $G \geq G'$, G' and G are incomparable.
- If it is possible to choose between two DAGs, a dominated path is (better) than the dominating path.

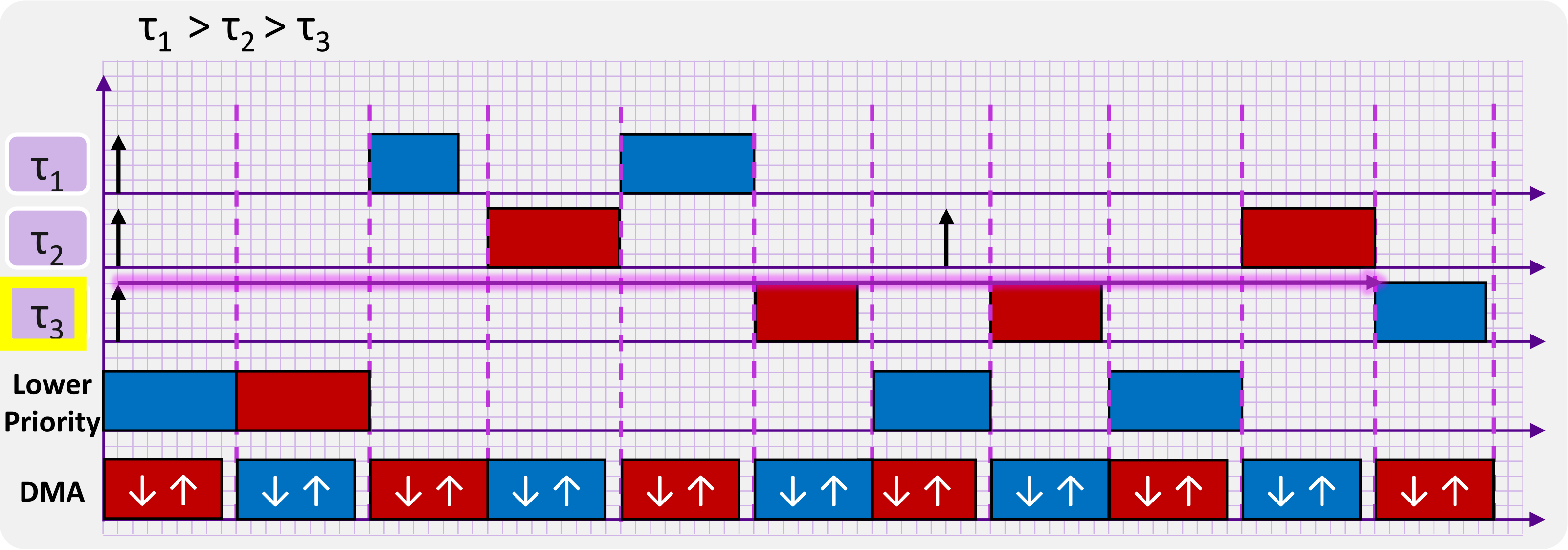
Schedulability Analysis



Schedulability Analysis

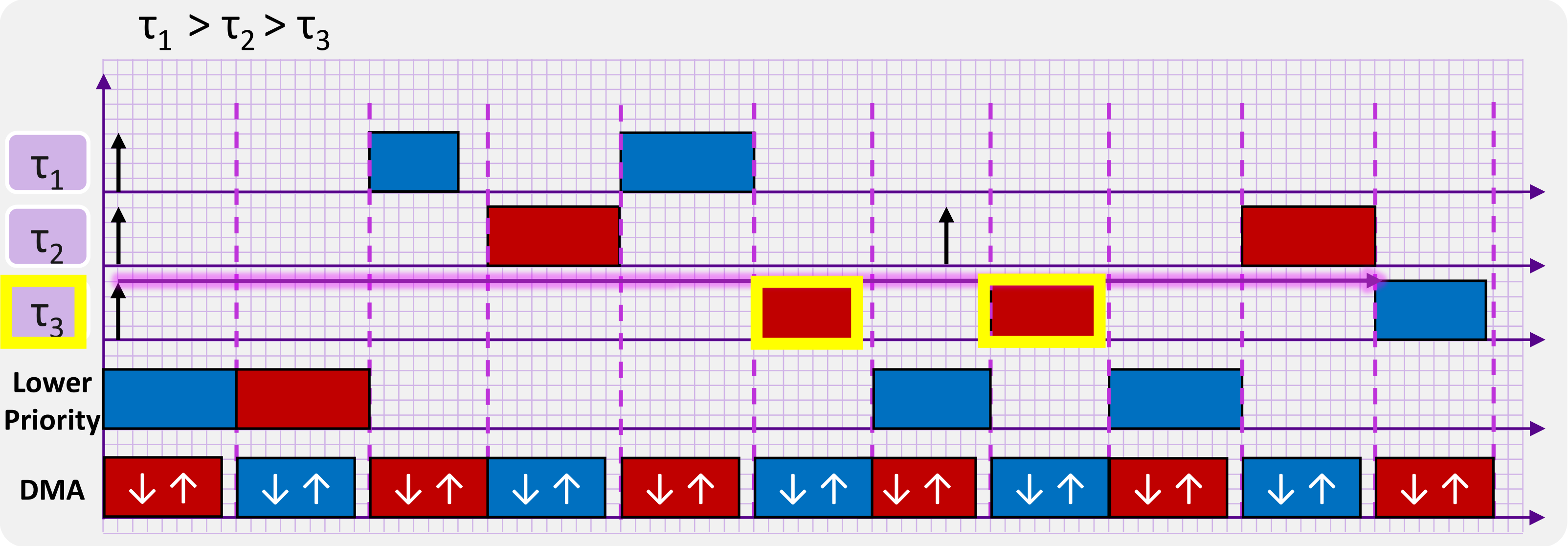


Schedulability Analysis



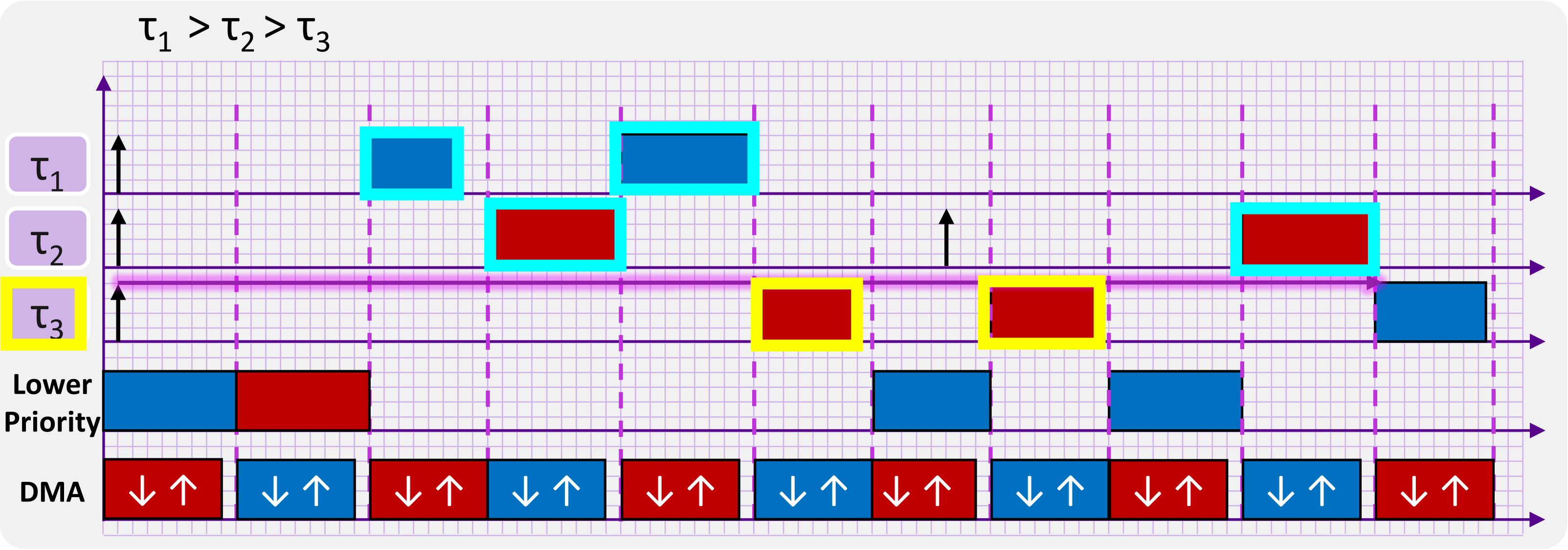
$R_3(P)$

Schedulability Analysis



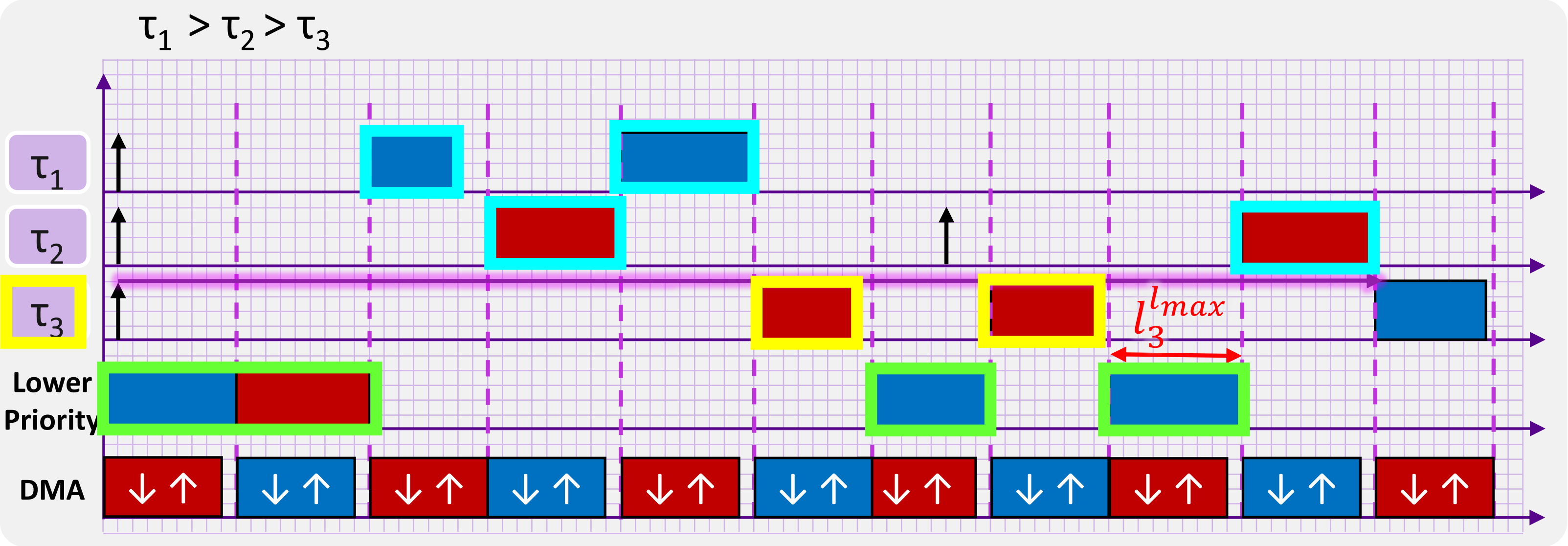
$$R_3(P) = P.L - P.end$$

Schedulability Analysis



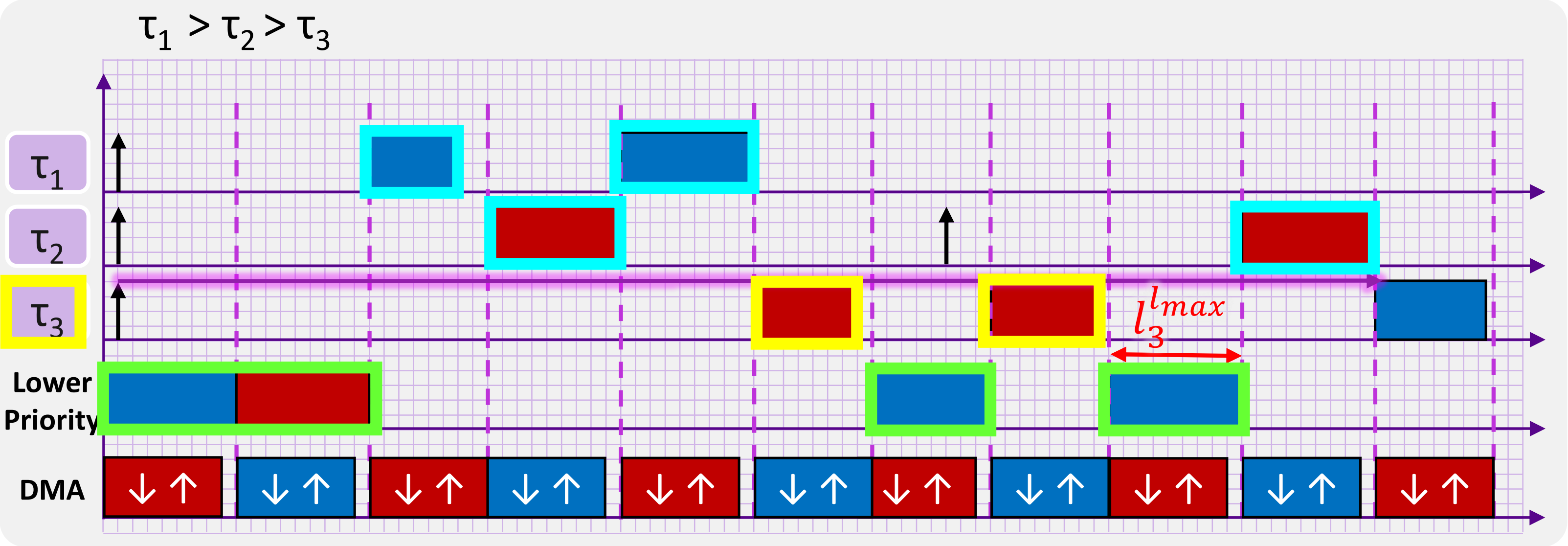
$$\boxed{R_3(P)} = \boxed{P.L - P.end} + \boxed{Inter_3(R_3(P))}$$

Schedulability Analysis



$$R_3(P) = \boxed{P.L - P.end} + \boxed{Inter_3(R_3(P))} + \boxed{(P.I + 1) * l_3^{lmax}}$$

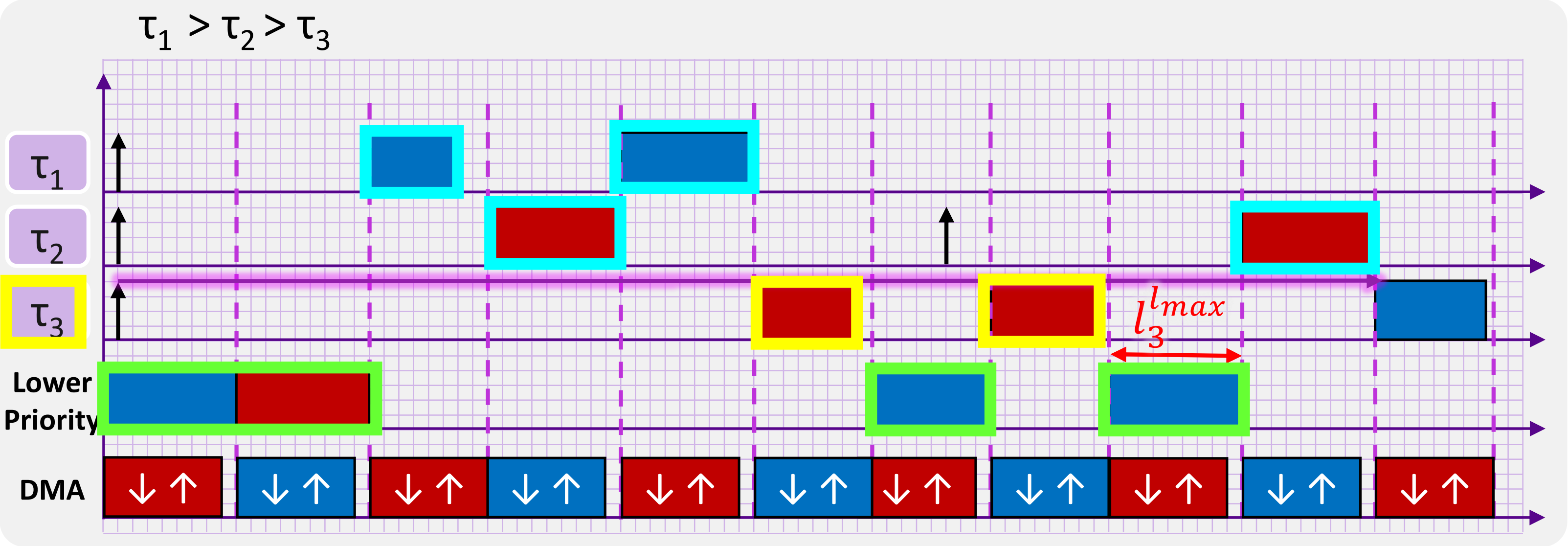
Schedulability Analysis



$$R_3(P) = P.L - P.end + Inter_3(R_3(P)) + (P.I + 1) * l_3^{lmax}$$

$$R_3(P) \le D_3 - P.end$$

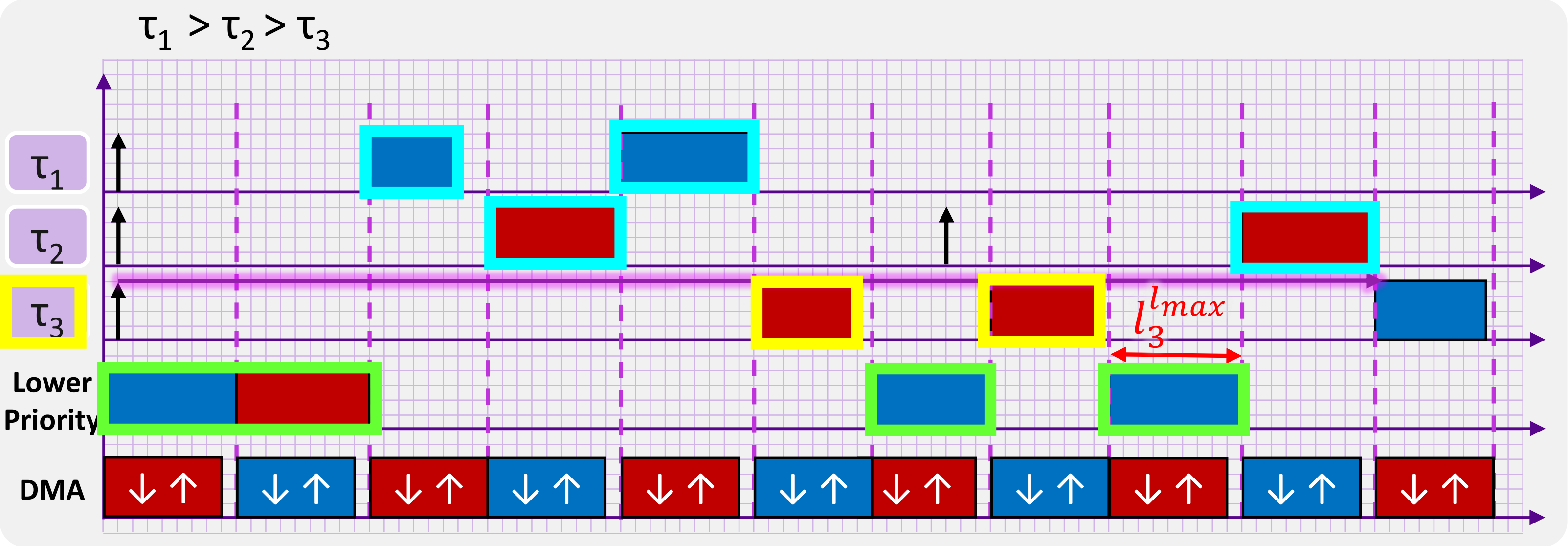
Schedulability Analysis



$$R_3(P) = \boxed{P.L - P.end} + \boxed{Inter_3(R_3(P))} + \boxed{(P.I+1) * l_3^{lmax}}$$

$\forall P \in G_3.C$: $R_3(P) \le D_3 - P.end$

Schedulability Analysis



$$R_3(P) = \boxed{P.L - P.end} + \boxed{Inter_3(R_3(P))} + \boxed{(P.I + 1) * l_3^{lmax}}$$

$\forall P \in G_3.C$: $R_3(P) \leq D_3 - P.end$

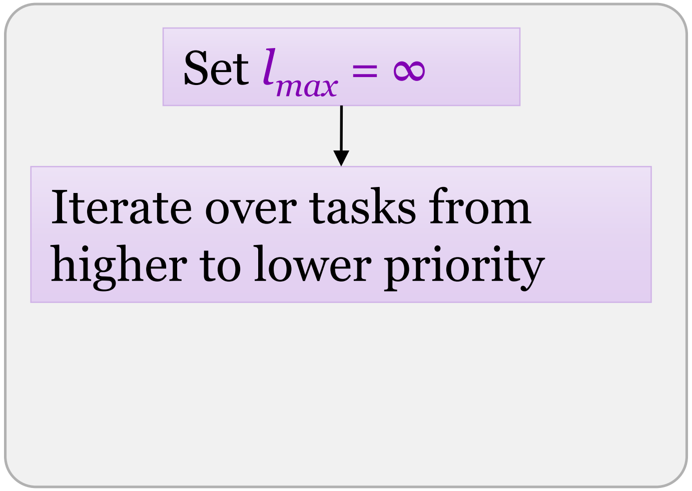
- $R_3(P)$ depends on l_3^{lmax} parameter only from lower priority tasks
- If the higher priority interference is known and the task is segmented, a maximum length l_{max} can be forced on the lower priority tasks to preserve the schedulability of the task.

ECRTS

Task Set Segmentation

Set $l_{max} = \infty$

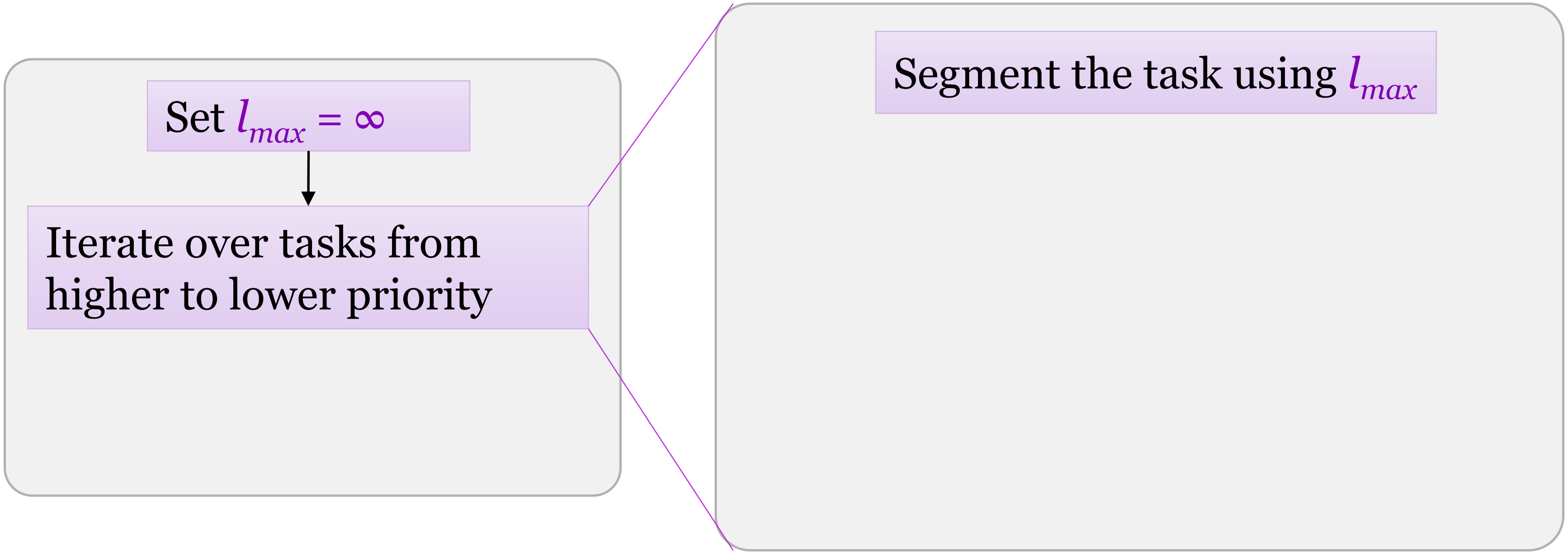
Task Set Segmentation



Set $l_{max} = \infty$

Iterate over tasks from higher to lower priority

Task Set Segmentation

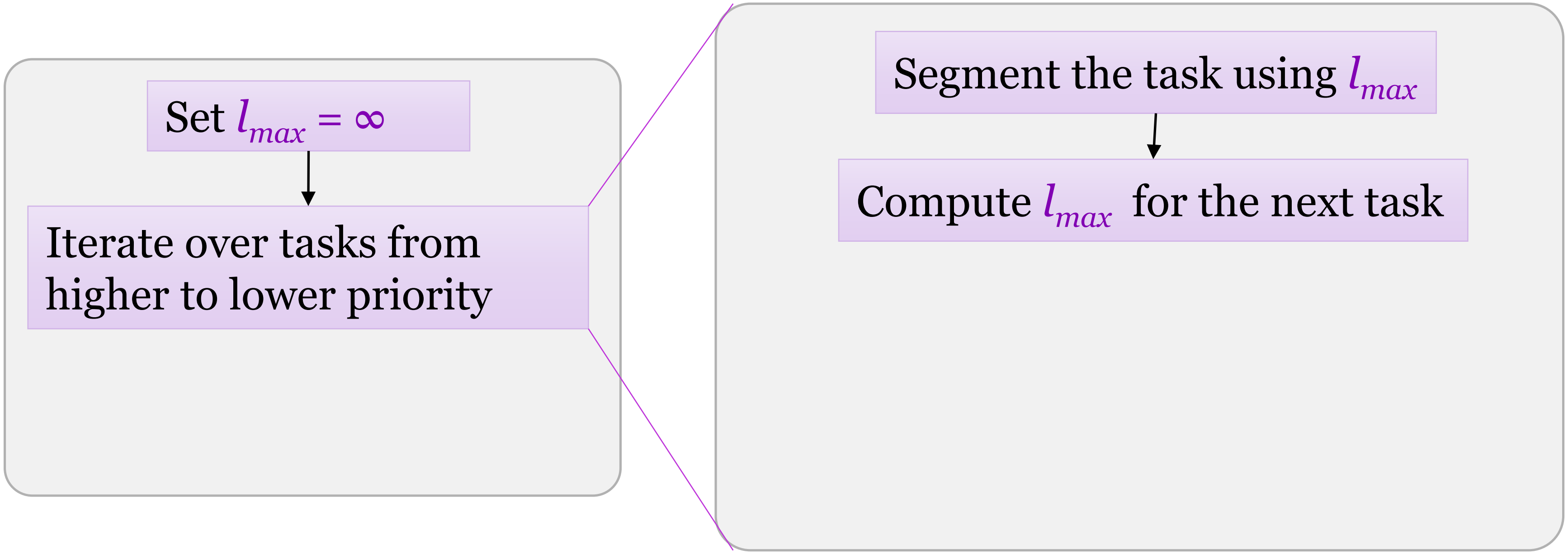


Set $l_{max} = \infty$

Iterate over tasks from higher to lower priority

Segment the task using l_{max}

Task Set Segmentation



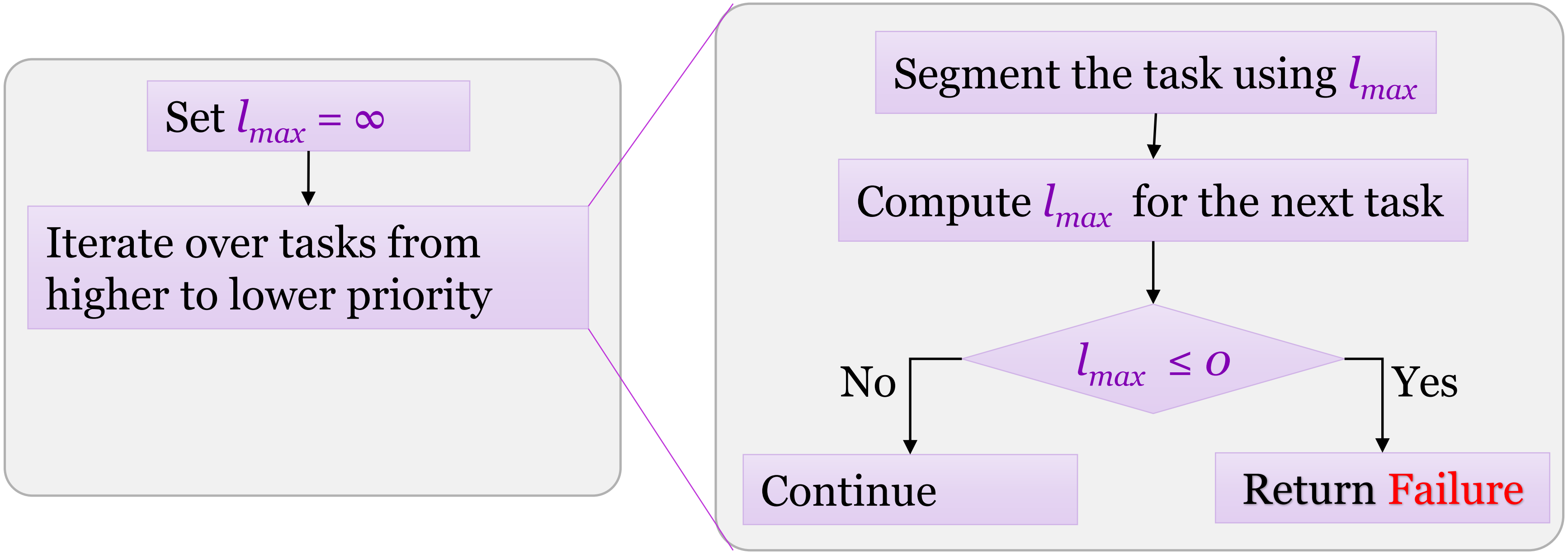
Set $l_{max} = \infty$

Iterate over tasks from higher to lower priority

Segment the task using l_{max}

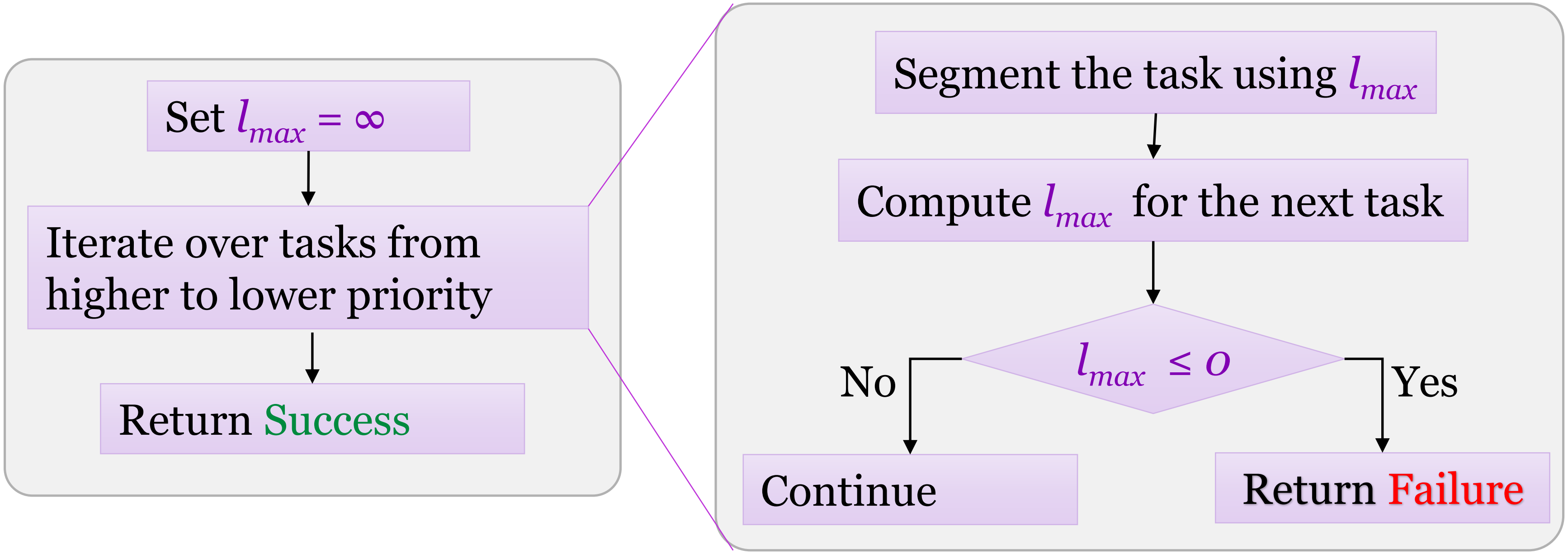
Compute l_{max} for the next task

Task Set Segmentation



Set $l_{max} = \infty$
Iterate over tasks from higher to lower priority
Segment the task using l_{max}
Compute l_{max} for the next task
$l_{max} \leq 0$
No
Yes
Continue
Return Failure

Task Set Segmentation



Set $l_{max} = \infty$

Iterate over tasks from higher to lower priority

Return Success

Segment the task using l_{max}

Compute l_{max} for the next task

$l_{max} \leq 0$

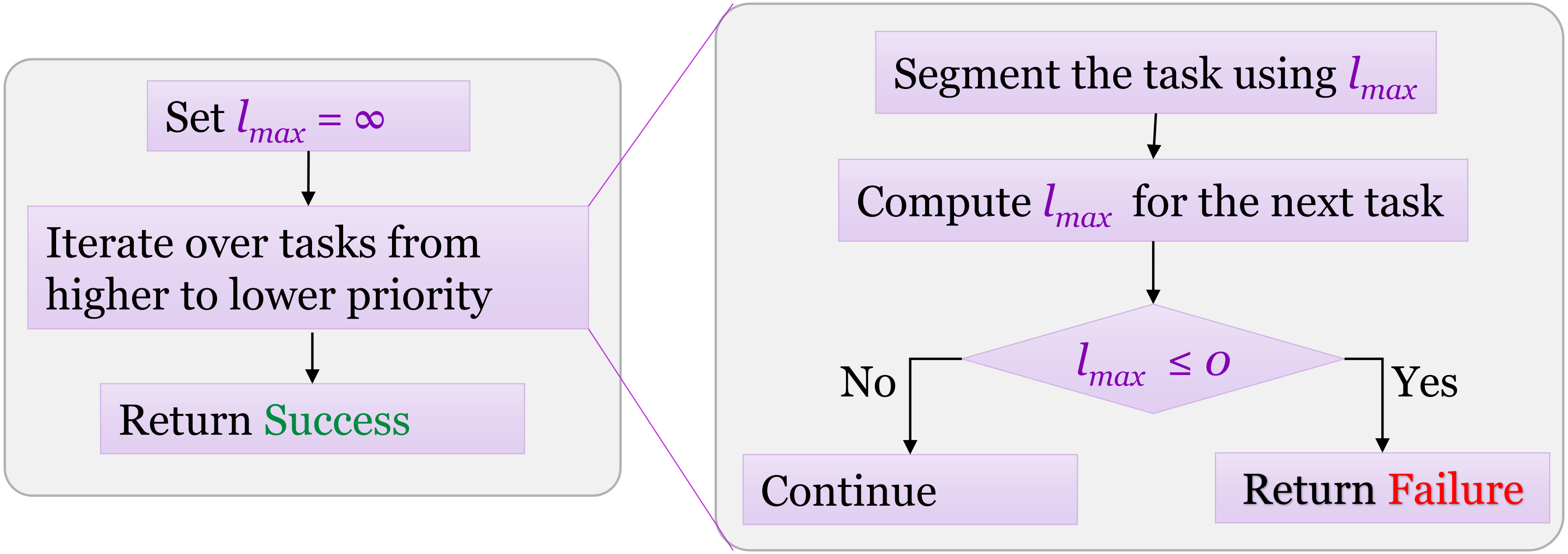
No

Yes

Continue

Return Failure

Task Set Segmentation



Set $l_{max} = \infty$

Iterate over tasks from higher to lower priority

Return Success

Segment the task using l_{max}

Compute l_{max} for the next task

$l_{max} \leq o$

No

Yes

Continue

Return Failure

- The paper proves that this algorithm results in an optimal task set segmentation that optimizes the schedulability.
- The program segmentation algorithm must preserve the optimality of the system by generating a set of DAGs that contains the best (dominated) DAGs from all the possible DAGs of the program.

Program Segmentation: Structure (main)

Region-based tree program structure

Program Segmentation: Structure (main)

Region-based tree program structure

Sub-graph with single entry and single exit

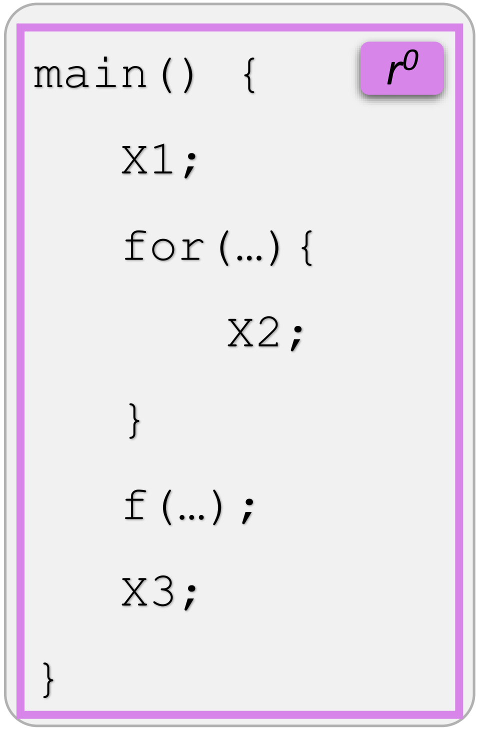
Program Segmentation: Structure (main)

```
main() {
    X1;
    for(…){
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit

Program Segmentation: Structure (main)

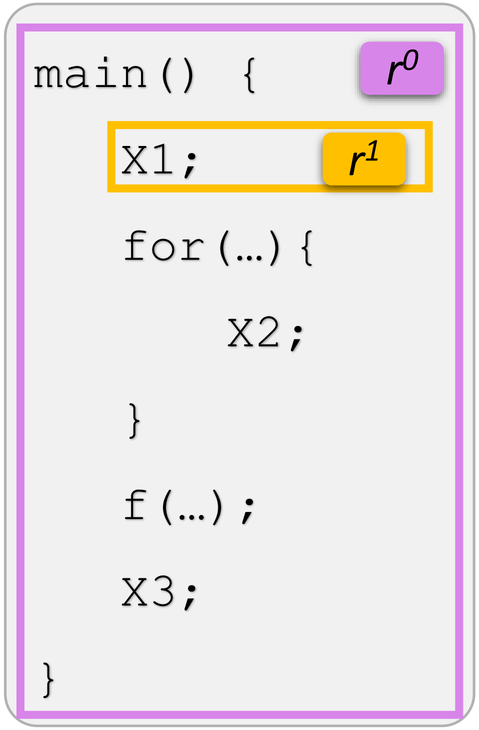


```
main() {
    X1;
    for(…){
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit

Program Segmentation: Structure (main)

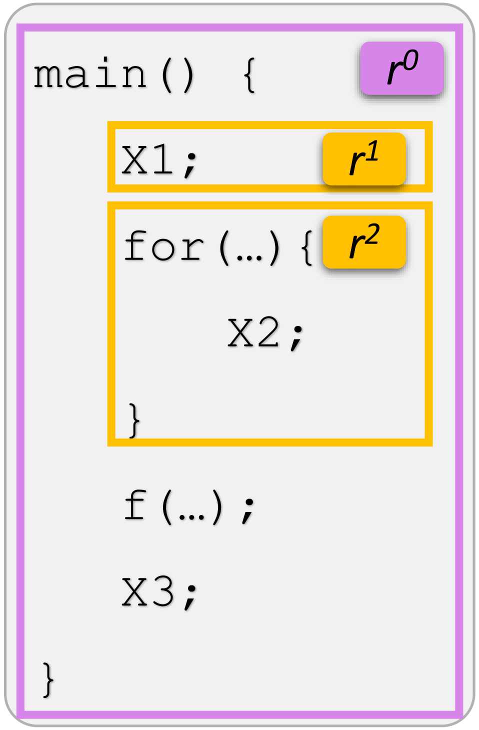


```
main() {
    X1;
    for(…){
          X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit

Program Segmentation: Structure (main)

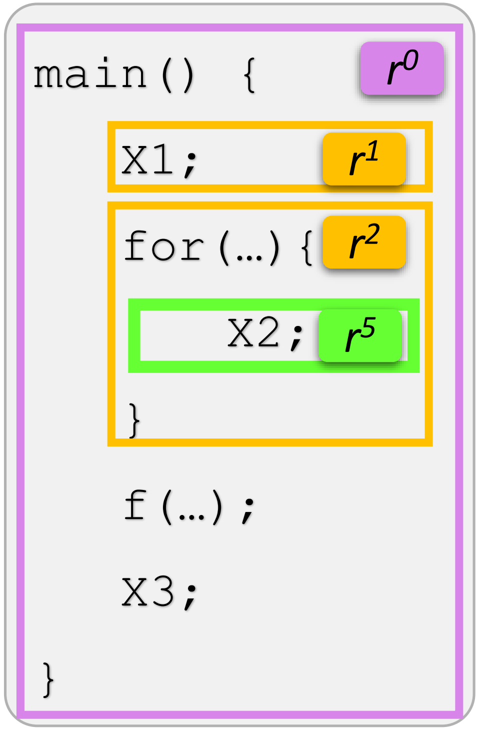


```
main() {
    X1;
    for(…) {
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit

Program Segmentation: Structure (main)

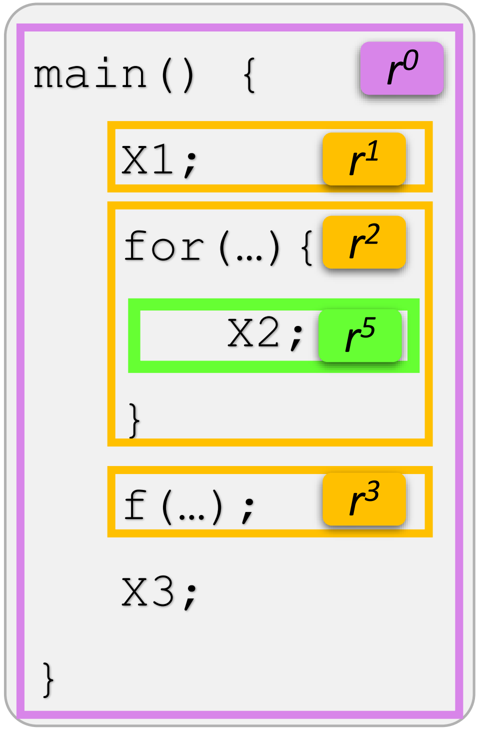


```
main() {
    X1;
    for(…) {
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit

Program Segmentation: Structure (main)

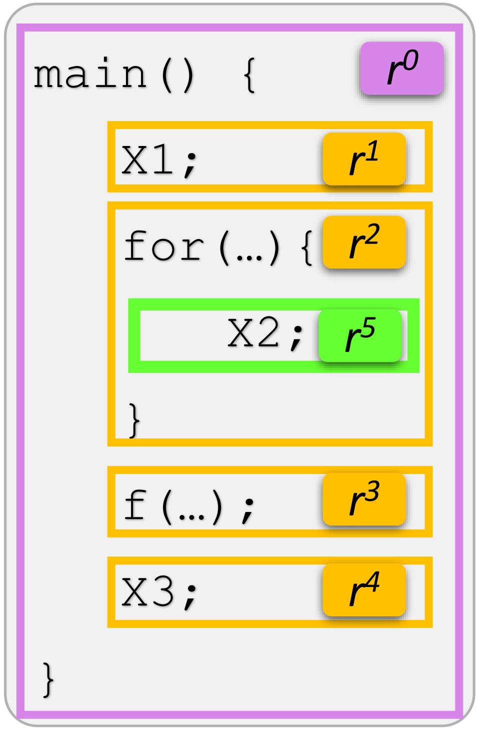


```
main() {
    X1;
    for(…) {
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit

Program Segmentation: Structure (main)

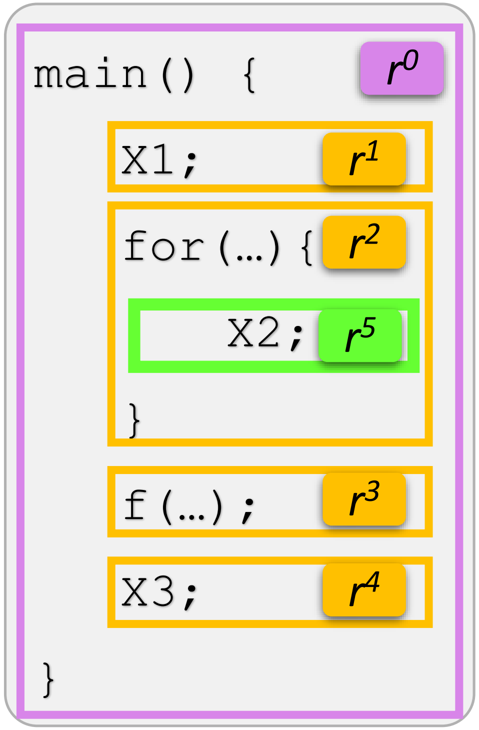


```
main() {
    X1;
    for(…) {
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit

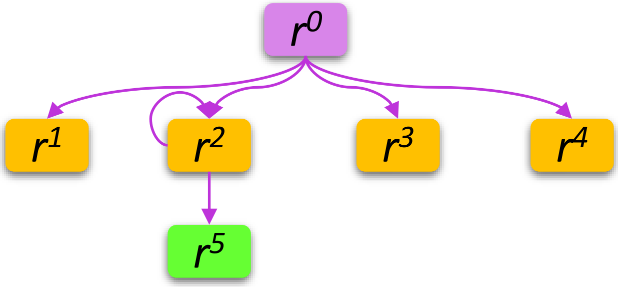
Program Segmentation: Structure (main)



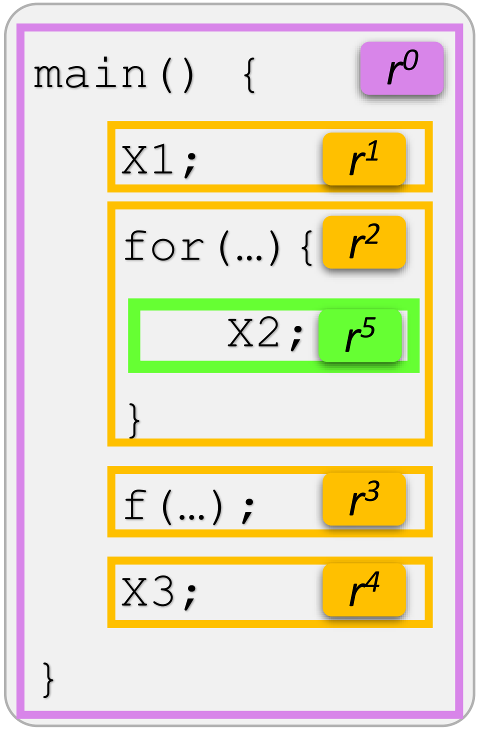
```
main() {
    X1;
    for(…) {
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit



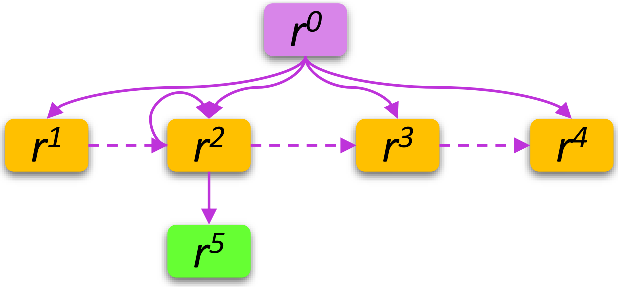
Program Segmentation: Structure (main)



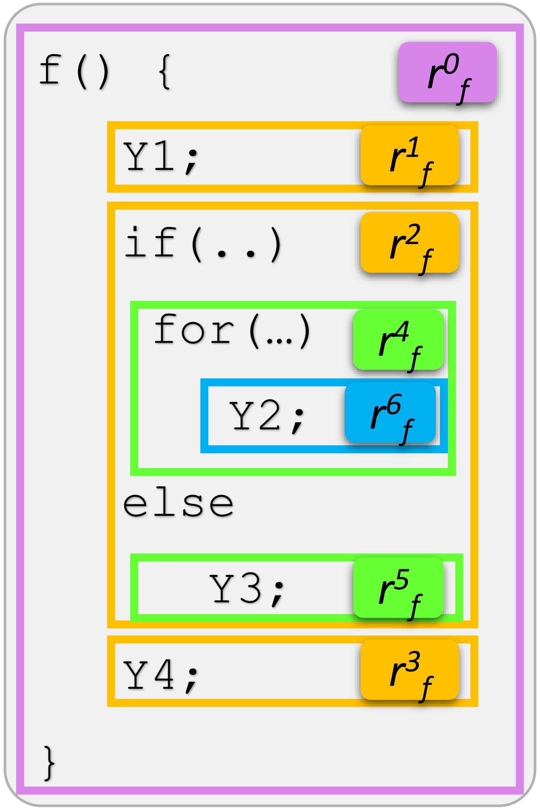
```
main()  {
    X1;
    for(…) {
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

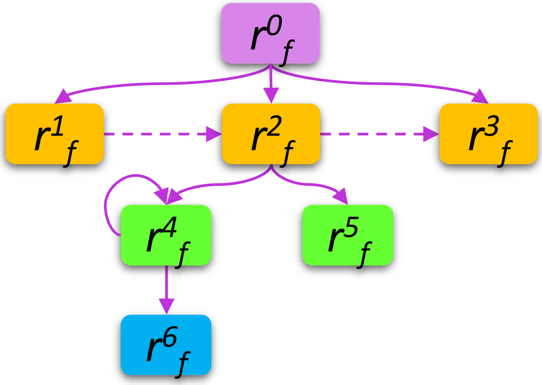
Sub-graph with single entry and single exit



Program Segmentation: Structure (f)



```
f() {
    Y1;
    if(..)
      for(…)
          Y2;
    else
        Y3;
    Y4;
}
```



Program Segmentation: Loop Transformations

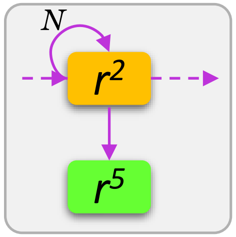
Program Segmentation: Loop Transformations

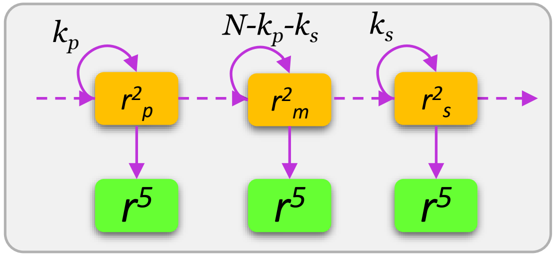
Loop Splitting

Loop Tiling

Program Segmentation: Loop Transformations

Loop Splitting

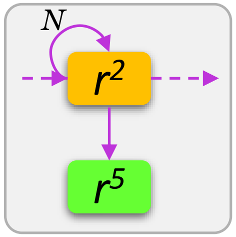


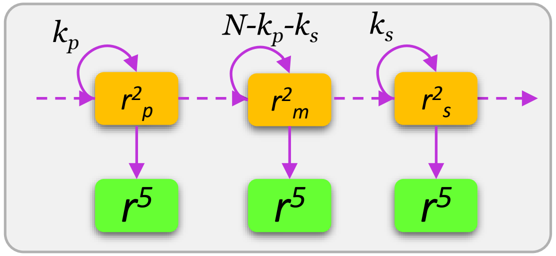


Loop Tiling

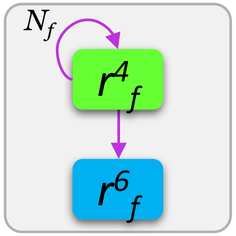
Program Segmentation: Loop Transformations

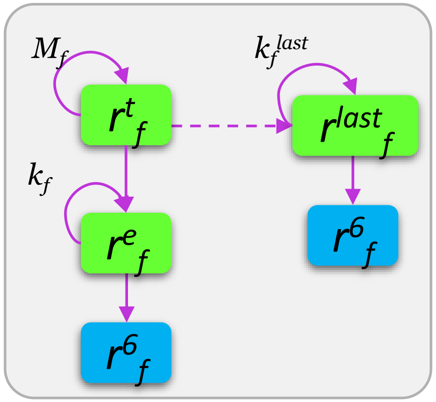
Loop Splitting



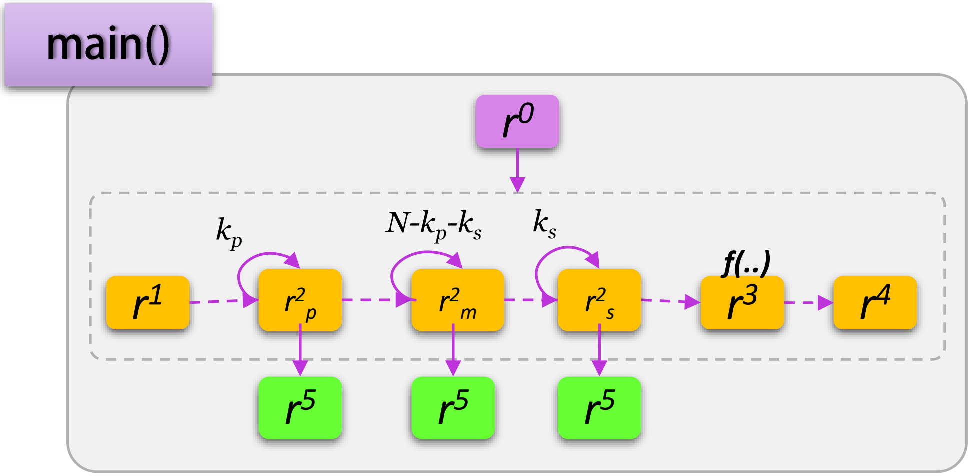


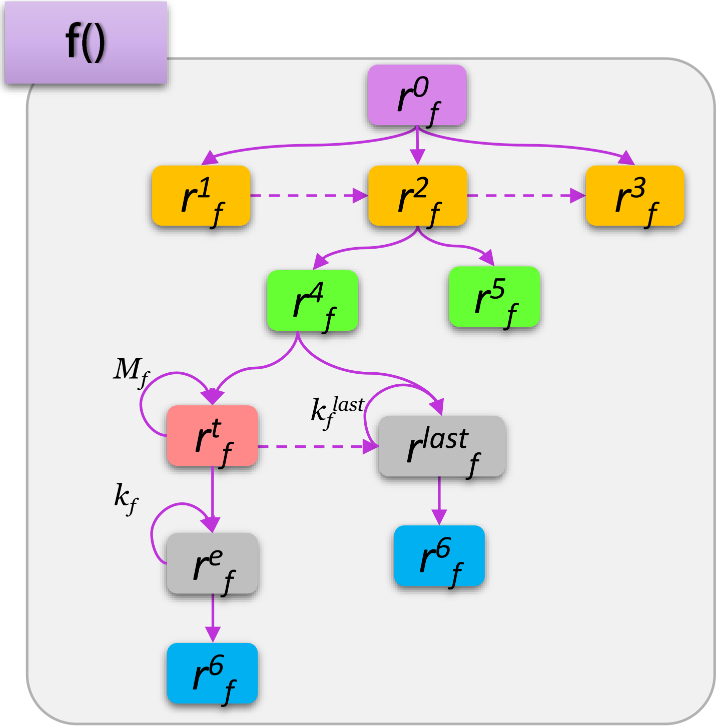
Loop Tiling



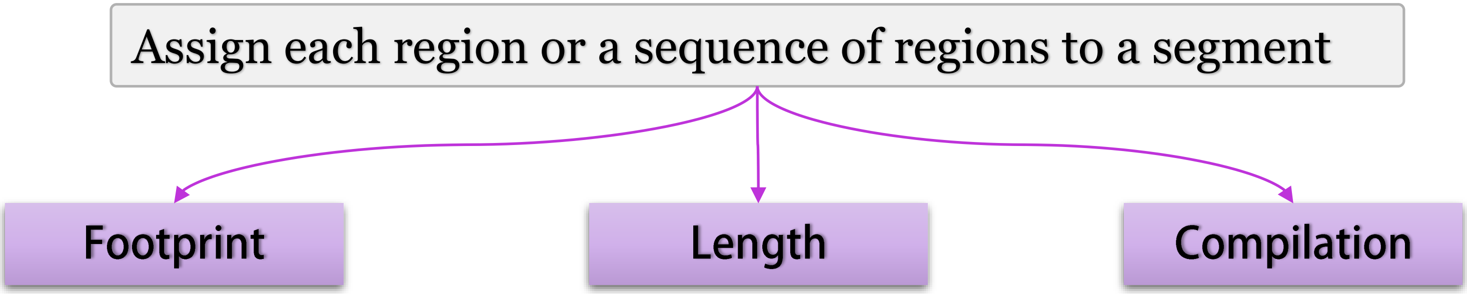


Program Segmentation: Final Trees





Program Segmentation: Valid Segmentation



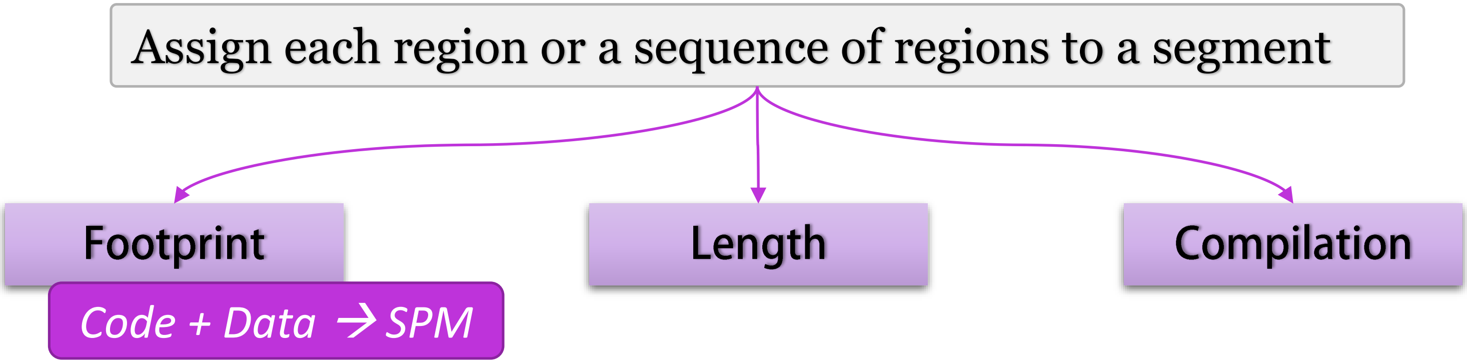
Assign each region or a sequence of regions to a segment

Footprint

Length

Compilation

Program Segmentation: Valid Segmentation



Assign each region or a sequence of regions to a segment

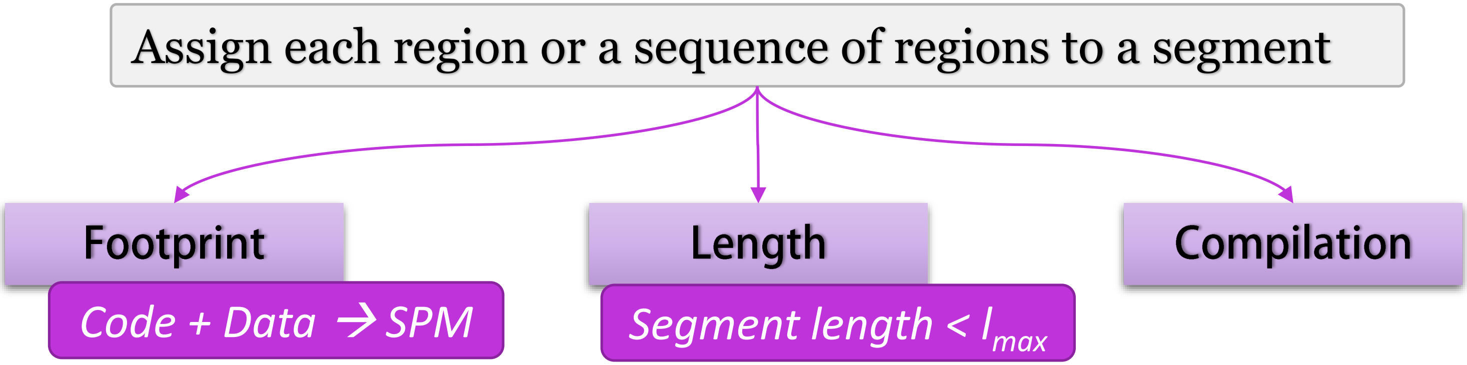
Footprint

Code + Data → SPM

Length

Compilation

Program Segmentation: Valid Segmentation



Assign each region or a sequence of regions to a segment

Footprint

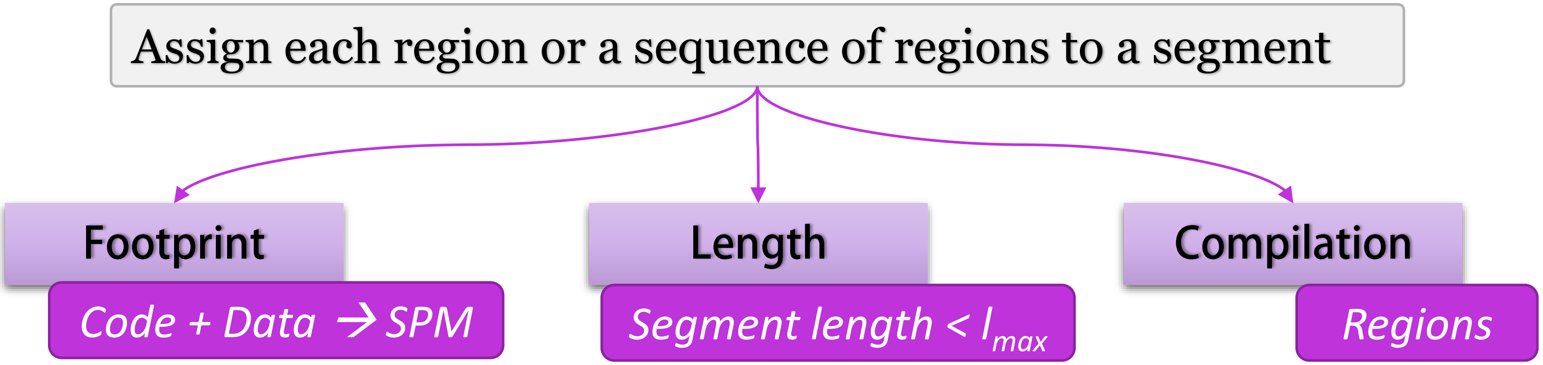
Code + Data → SPM

Length

Segment length < l_{max}

Compilation

Program Segmentation: Valid Segmentation



Assign each region or a sequence of regions to a segment

Footprint

Code + Data → SPM

Length

Segment length < l_{max}

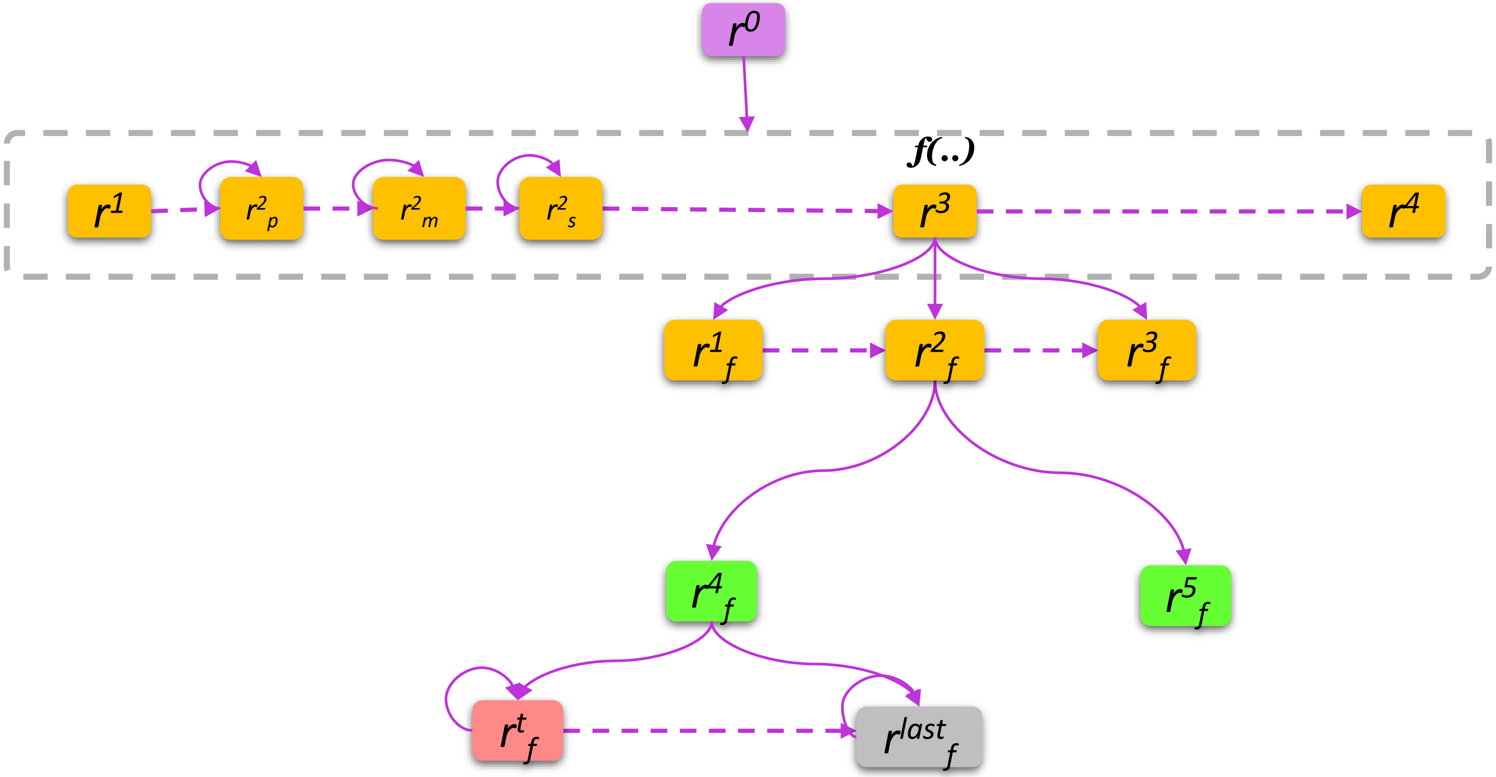
Compilation

Regions

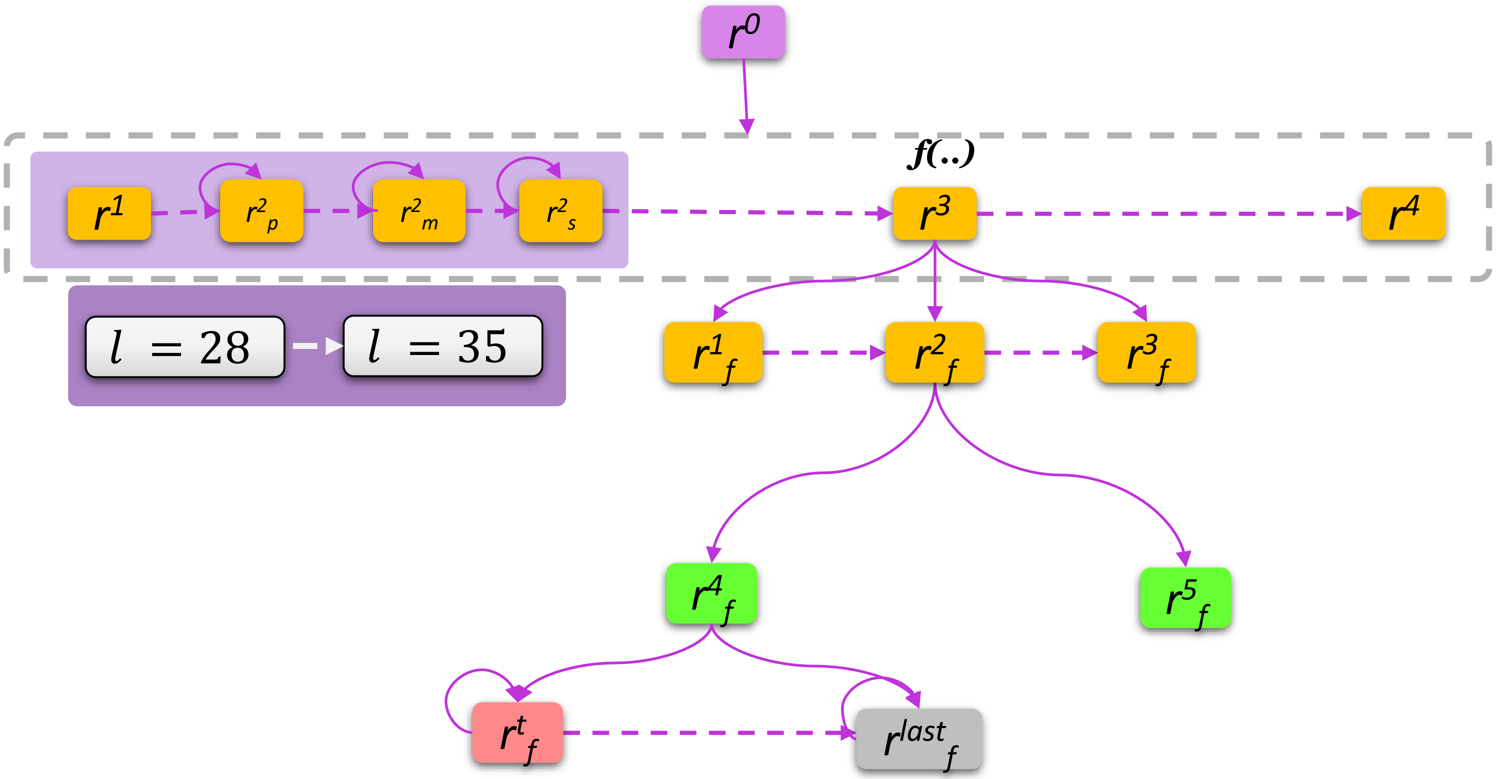
Program Segmentation: Segmented Tree (1)

- A tree where each node is a segment path.
- It is obtained by substituting region sequences with a set of paths.
- A segmented tree generates a set of DAGs where each DAG is constructed by taking one path out of each path set.

Program Segmentation: Segmented Tree (2)

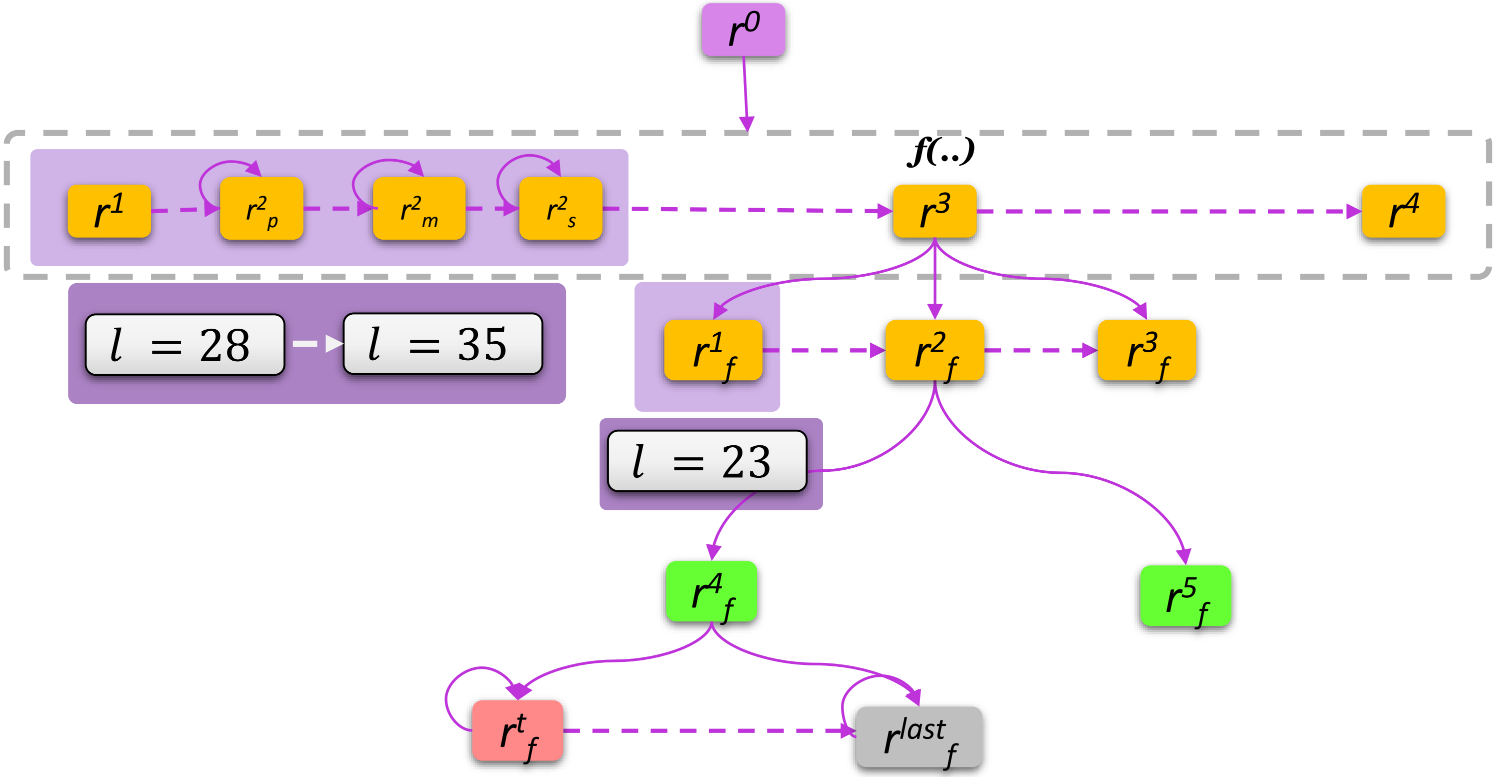


Program Segmentation: Segmented Tree (2)

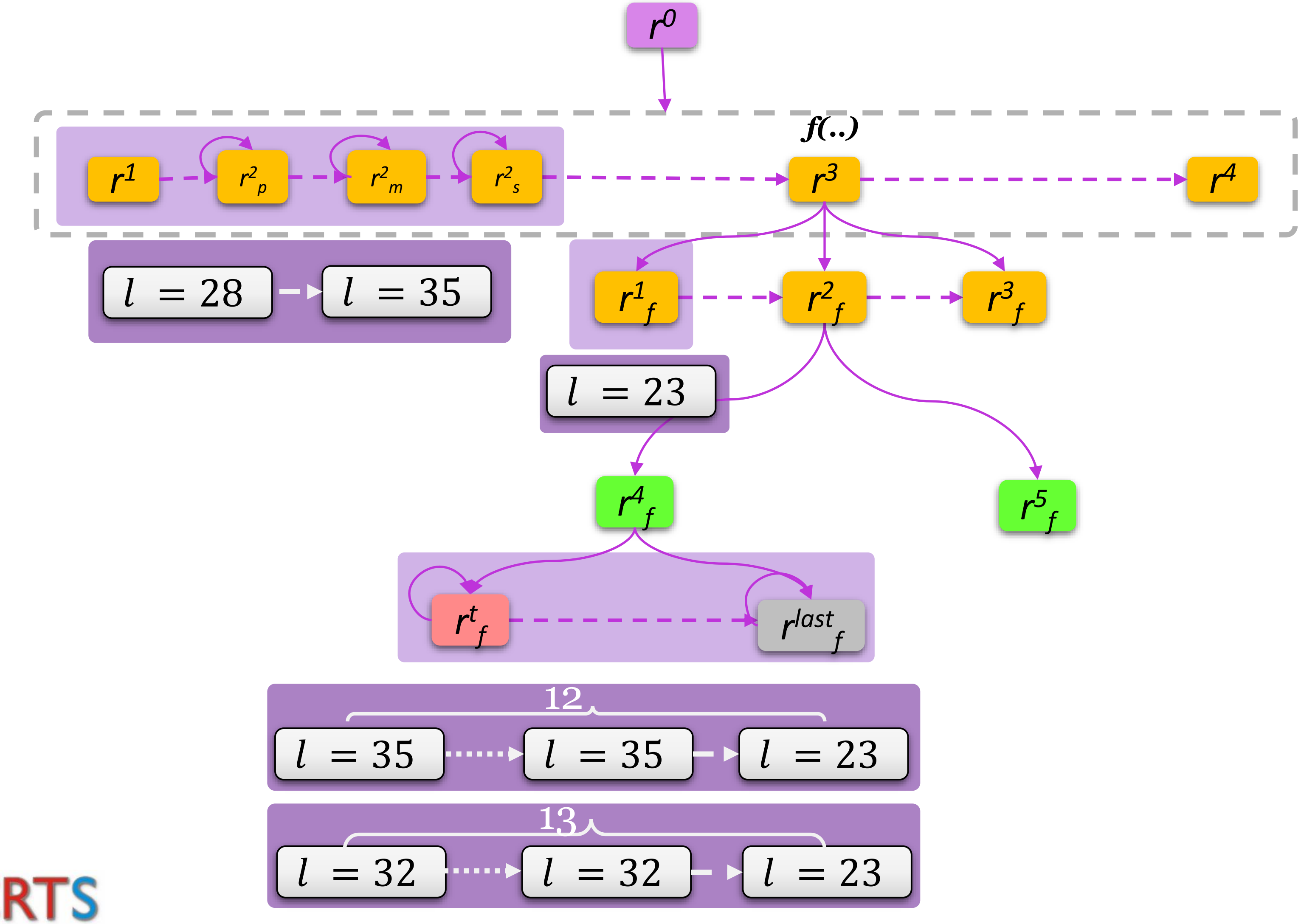


ECRTS

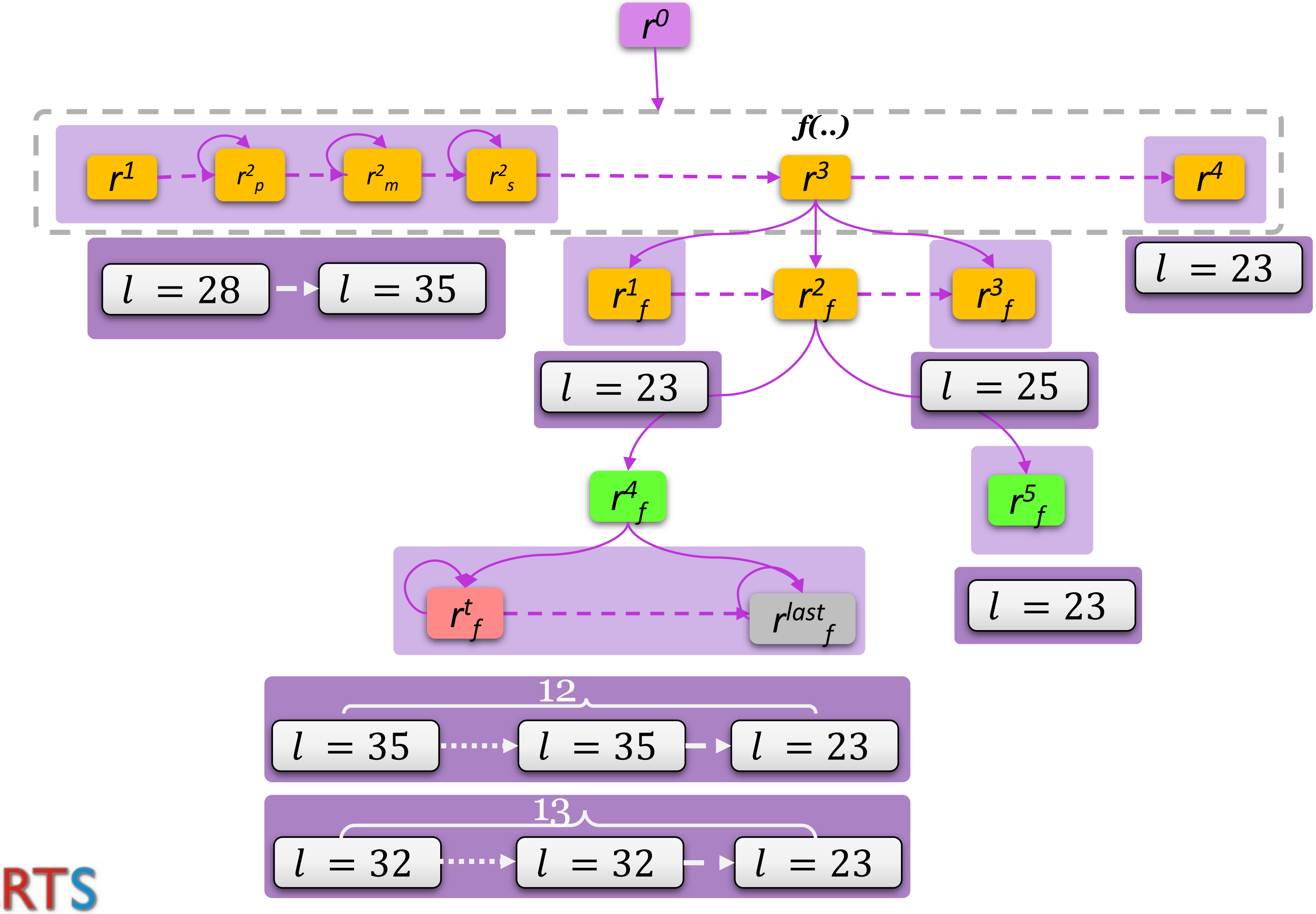
Program Segmentation: Segmented Tree (2)



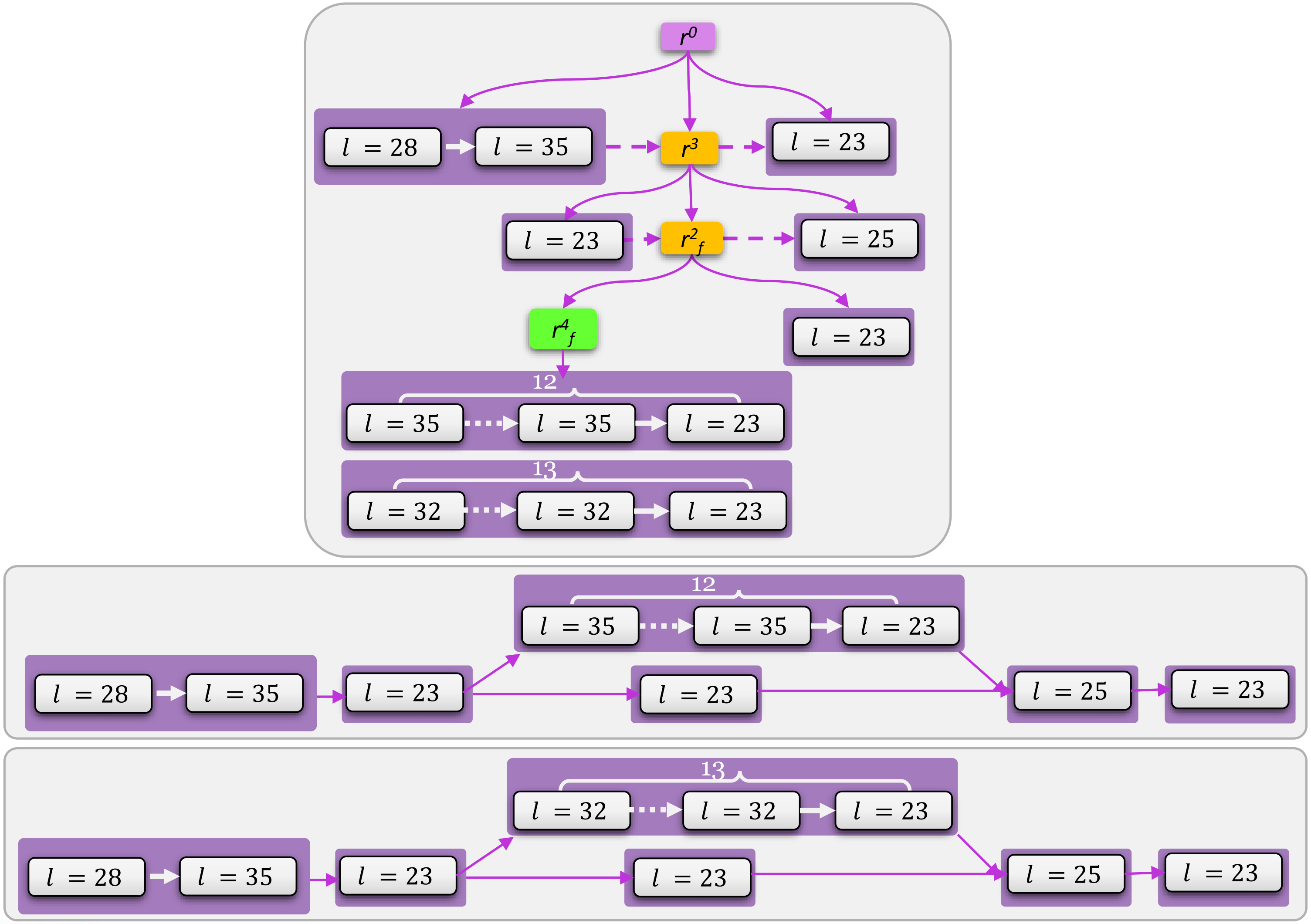
Program Segmentation: Segmented Tree (2)



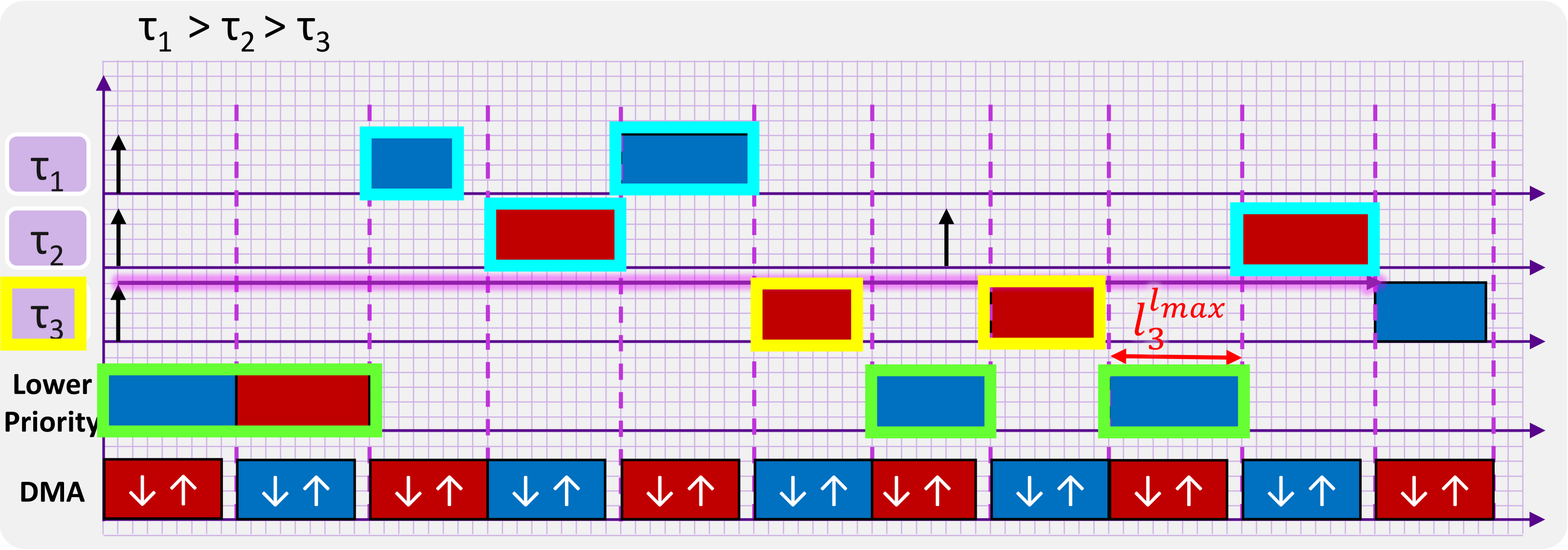
Program Segmentation: Segmented Tree (2)



Program Segmentation: Segmented Tree (3)

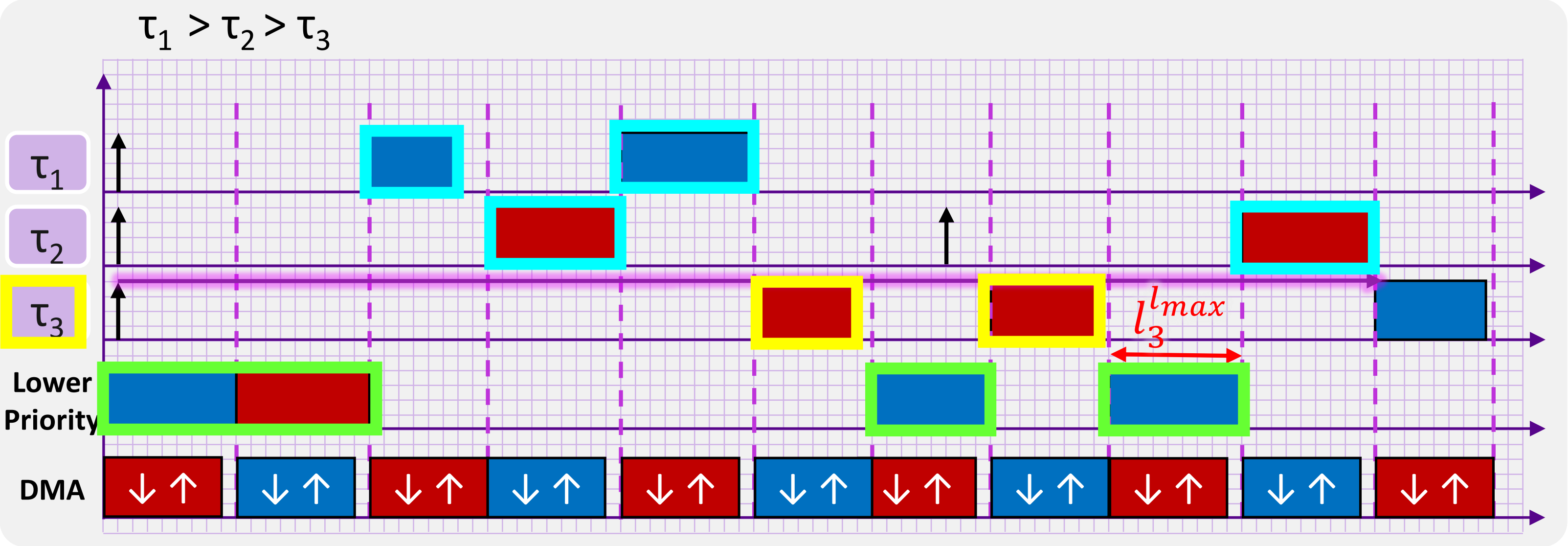


Program Segmentation: Algorithms (1)



$$R_3(P) = P.L - P.end + Inter_3(R_3(P)) + (P.I + 1) * l_3^{lmax}$$

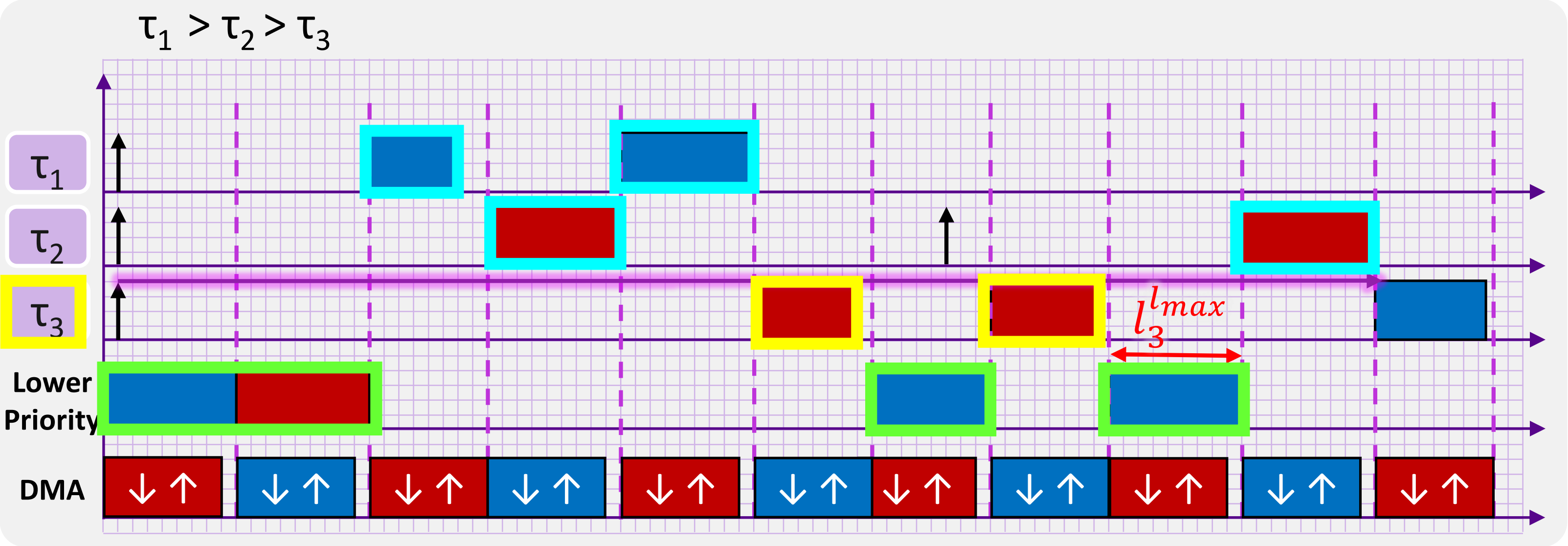
Program Segmentation: Algorithms (1)



$$R_3(P) = P.L - P.end + Inter_3(R_3(P)) + (P.I + 1) * l_3^{lmax}$$

Minimize P.L

Program Segmentation: Algorithms (1)

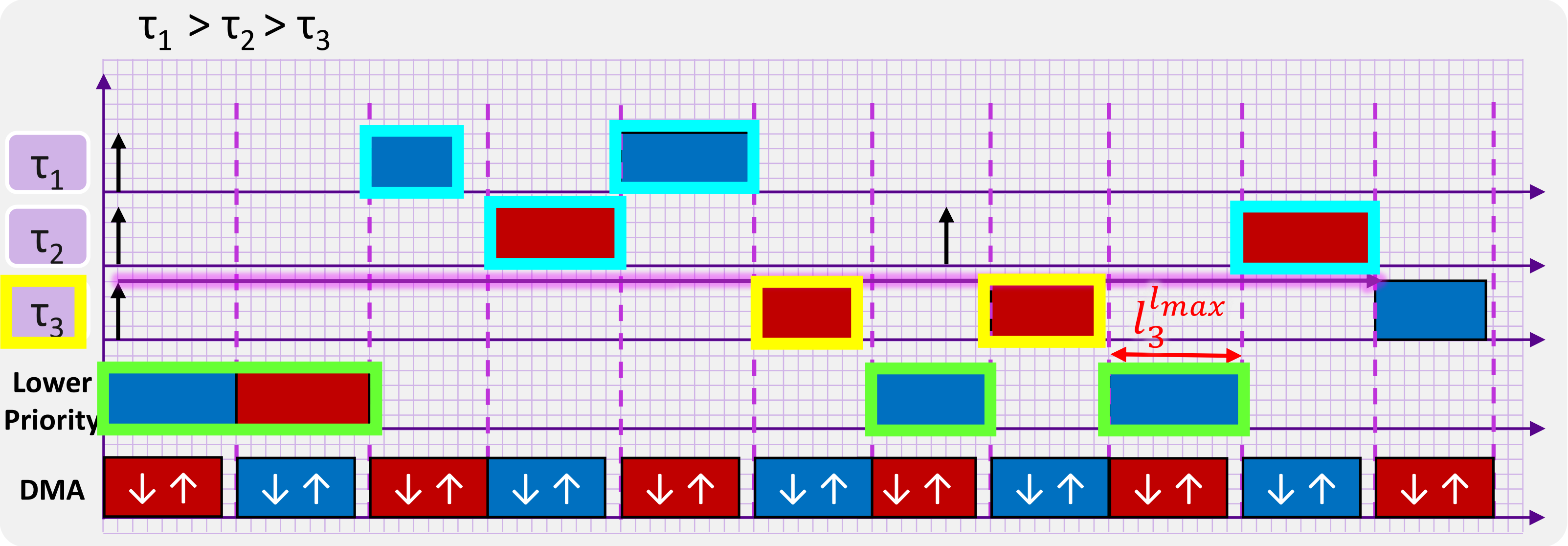


$$R_3(P) = \boxed{P.L - P.end} + \boxed{Inter_3(R_3(P))} + \boxed{(P.I + 1) * l_3^{lmax}}$$

Minimize P.L

Maximize P.end

Program Segmentation: Algorithms (1)



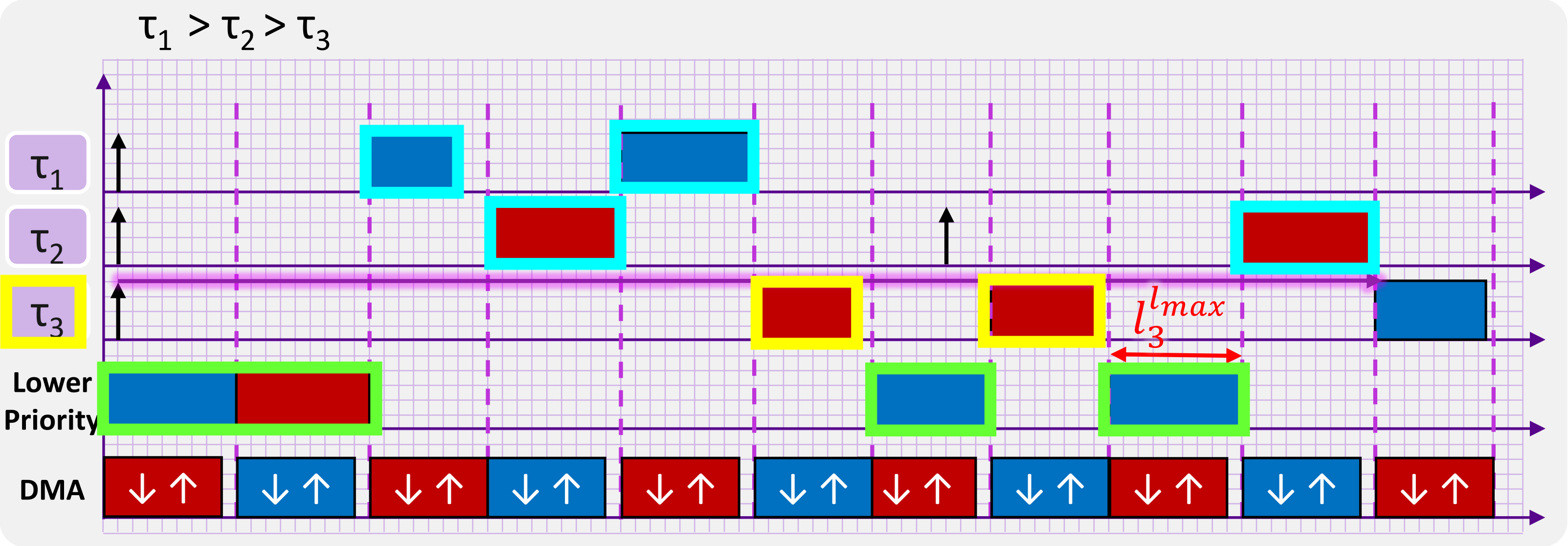
$$R_3(P) = \boxed{P.L - P.end} + \boxed{Inter_3(R_3(P))} + \boxed{(P.I + 1) * l_3^{lmax}}$$

Minimize P.L

Maximize P.end

Minimize P.I

Program Segmentation: Algorithms (1)



$$\boxed{R_3(P)} = \boxed{P.L - P.end} + \boxed{Inter_3(R_3(P))} + \boxed{(P.I + 1) * l_3^{lmax}}$$

Minimize P.L

Maximize P.end

Minimize P.I

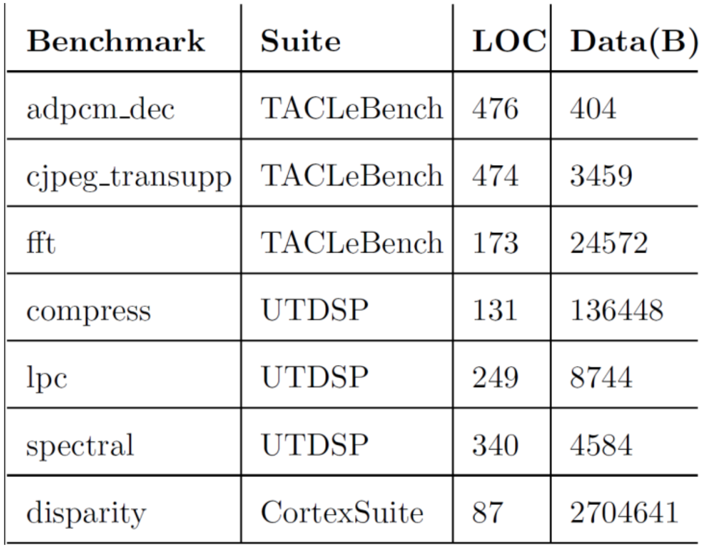
Based on path domination → keep dominated paths

Program Segmentation: Algorithms (2)

- The segmentation algorithms generates the possible paths for the segmented tree based on the constraints.
- The generated paths are filtered using path domination to eliminate the dominating (worse) paths.
- The DAGs generated from the segmented tree are filtered using the DAG domination to keep the dominated (better) DAGs.
- Pruning conditions are used to avoid enumerating all the DAGs which is very time consuming due to the parameterized split/tile transformations.

Evaluation (1)

- The segmentation framework is implemented using LLVM compiler.
- Simple MIPS processor model: 5-stage pipeline, no branch prediction.
- Vary the SPM size between 4 kB to 512 kB exponentially.
- Multiple benchmarks from different suites.
- Test for system utilization between 0.2 – 0.95.
- For each system utilization → 100 task set, 5-15 tasks / task set.
- Results reported in terms of system schedulability.



Benchmark	**Suite**	**LOC**	**Data(B)**
adpcm_dec	TACLeBench	476	404
cjpeg_transupp	TACLeBench	474	3459
fft	TACLeBench	173	24572
compress	UTDSP	131	136448
lpc	UTDSP	249	8744
spectral	UTDSP	340	4584
disparity	CortexSuite	87	2704641

Evaluation (2)

		Length	*Footprint*	*Compilation*
Proposed	Optimal	l_{max}	*SPM size*	*Regions*

Evaluation (2)

		Length	*Footprint*	*Compilation*
	Ideal	l_{max}	*None*	*None*
Proposed	Optimal	l_{max}	*SPM size*	*Regions*

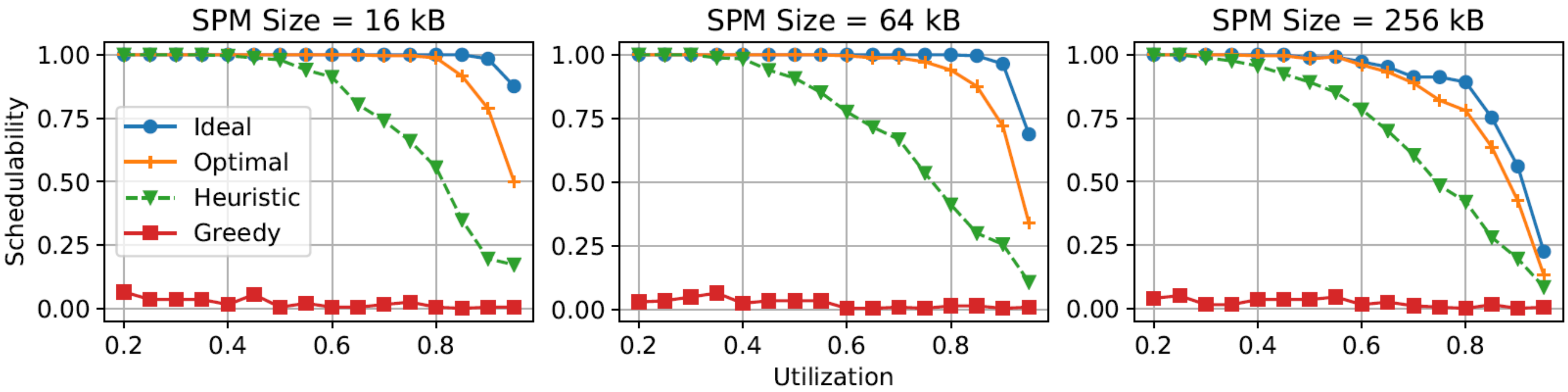
Evaluation (2)

		Length	*Footprint*	*Compilation*
	Ideal	l_{max}	*None*	*None*
Proposed	Optimal	l_{max}	*SPM size*	*Regions*
	Heuristic	*Fixed* l_{max}	*SPM size*	*Regions*

Evaluation (2)

		Length	*Footprint*	*Compilation*
	Ideal	l_{max}	*None*	*None*
Proposed	Optimal	l_{max}	*SPM size*	*Regions*
	Heuristic	*Fixed* l_{max}	*SPM size*	*Regions*
	Greedy	*None*	*SPM size*	*Regions*

Evaluation (3)



Conclusion & Future Work

Conclusion

- The paper proposes a segmentation framework based on LLVM compiler to automatically generate PREM-compatible code for sequential programs running on a general purpose processor.
- An optimal task set segmentation algorithm is derived under fixed-priority scheduling for fixed-size DMA time.
- The evaluation shows that the proposed algorithm outperforms both greedy and heuristic algorithms.

Future Work

- The framework can be extended to other PREM-based scheduling schemes.
- The framework can also consider other task and platform models, especially parallel tasks.

31th Euromicro Conference on Real-Time Systems
9-12 July 2019 | Stuttgart, Germany

For questions, please contact the authors:

mrefaat@uwaterloo.ca
rpellizz@uwaterloo.ca

Thank you