Scheduling and Optimization of Fault-Tolerant Embedded Systems

Petru Eles, Viacheslav Izosimov, Paul Pop*, Zebo Peng

Department of Computer and Information Science (IDA)
Linköping University
http://www.ida.liu.se/~eslab/

***Department of Informatics and Mathematical Modelling**
Technical University of Denmark

We want systems to be fault tolerant!

We want systems to be fault tolerant!

Safety critical

We want systems to be fault tolerant!

QoS should not deteriorate (too much)!

- **Real-Time:**

  Time constraints have to be satisfied even in the presence of faults

- **Permanent faults**
- **Transient/Intermittent faults**

- Permanent faults
- **Transient/Intermittent faults**

☞ Permanent faults

☞ Transient/Intermittent faults

- Increasing susceptibility to transient faults:
  - Shrinking transistor size
  - Increasing frequency
  - Increasing temperature
  - Decreasing supply voltage
  - .....

☞ Permanent faults

☞ Transient/Intermittent faults

- Increasing susceptibility to transient faults:
  - Shrinking transistor size
  - Increasing frequency
  - Increasing temperature
  - Decreasing supply voltage
  - .....
- Sources of transient faults:
  - Electromagnetic interferences
  - Crosstalk
  - Ground bounce
  - Radiation by
    - cosmic particles
    - packaging material
  - Power supply fluctuations
  - .....

- The potential number of transient faults is large!

- It is larger than that of permanent faults!

- The potential number of transient faults is large!
- It is larger than that of permanent faults!
- Specific approaches are needed in order to achieve *cost efficient* fault tolerant systems in the context of such an increased number of transient faults!

☞ In this talk

- System-level perspective
- Fault tolerant systems with real-time constraints
- Transient processor faults
- Fault tolerance techniques considered
  - software replication
  - re-execution
  - checkpointing

☞ In this talk

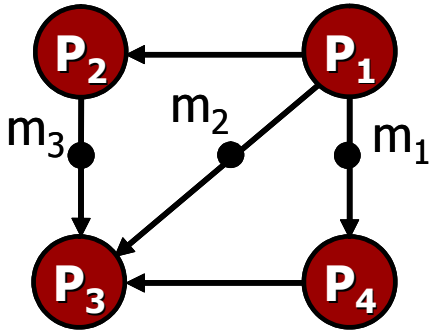
- System-level perspective
- Fault tolerant systems with real-time constraints
- Transient processor faults
- Fault tolerance techniques considered
  - software replication
  - re-execution
  - checkpointing

- Problem focus:
  - System-level optimization and scheduling
    - time constraints are satisfied
    - required fault tolerance is achieved
    - limited amount of resources available

Outline

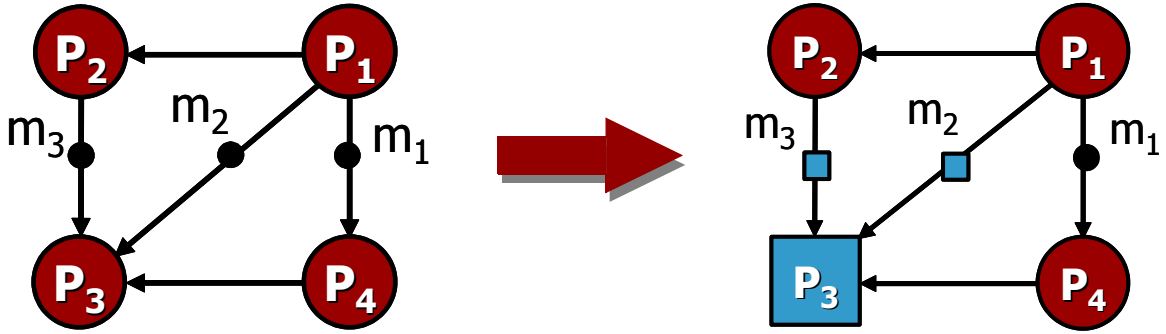
- Overall Flow, Application and System Model
- Fault Tolerance Policy Assignment
- Transparency
- Re-execution and Checkpointing
- Generation of Fault-Tolerant Schedules
- Cross-layer Optimization with Hardening Alternatives
- Conclusions

Overall Flow



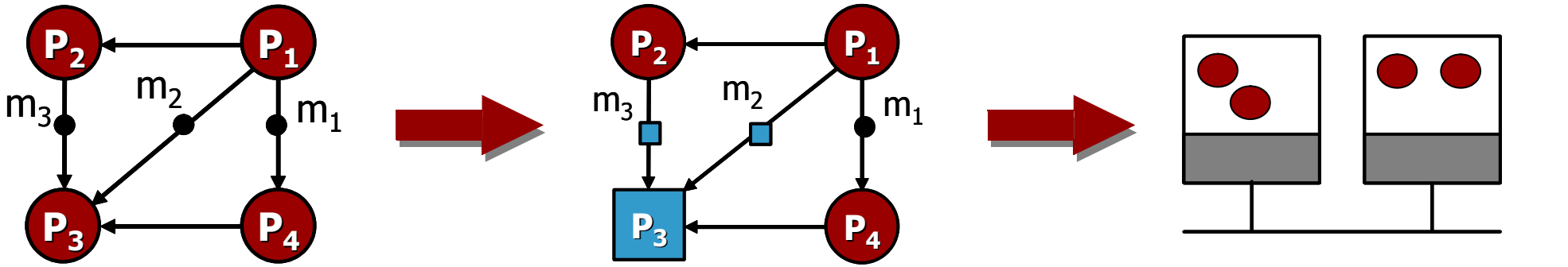
- Process graphs
- Periods, deadlines
- Max. nr. of faults
- Overheads

Overall Flow



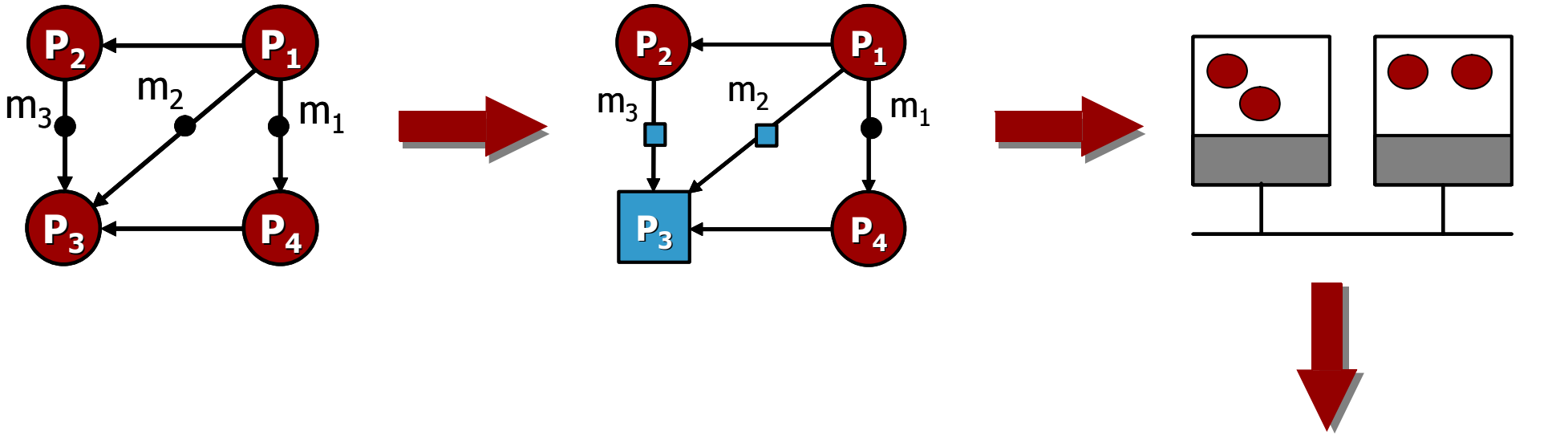
- Fault tolerance policies
- Transparency

Overall Flow

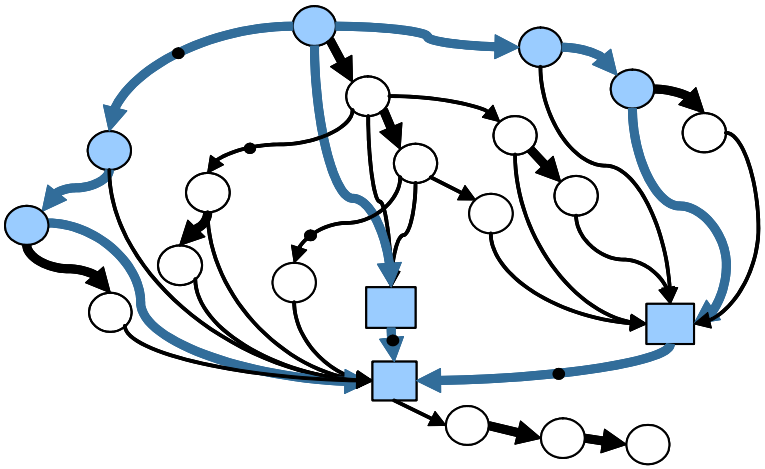


- Mapping of processes and replicas

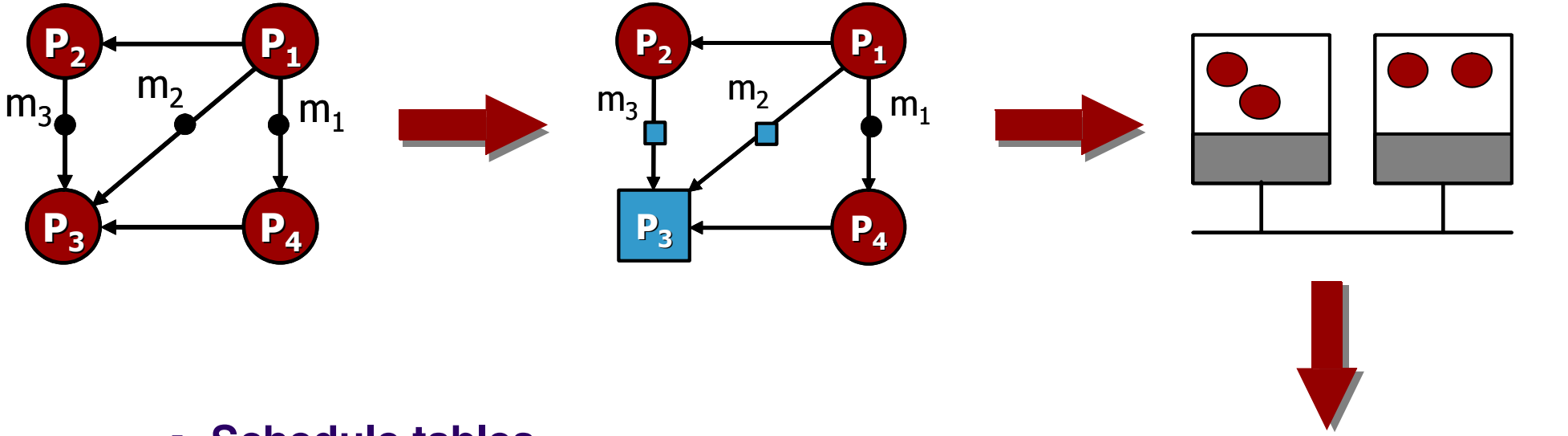
Overall Flow



- Fault-Tolerant Conditional Process Graph

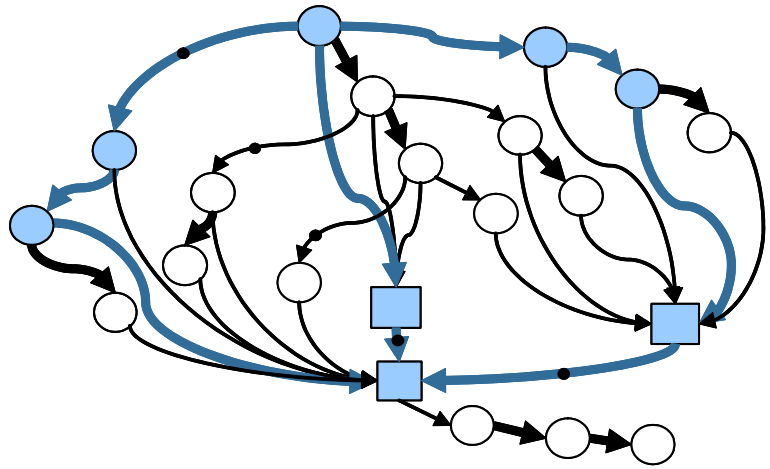


Overall Flow

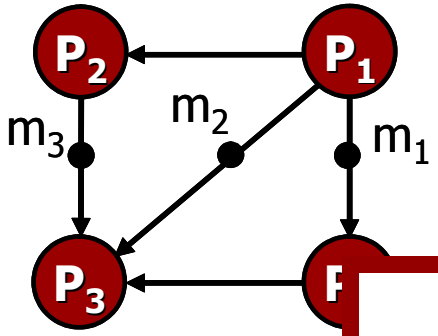


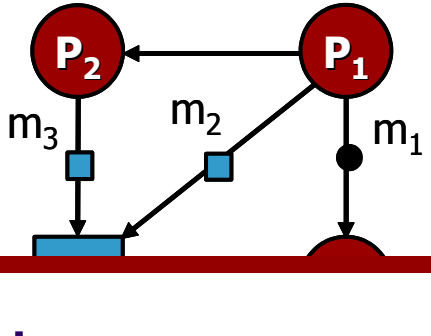
- Schedule tables

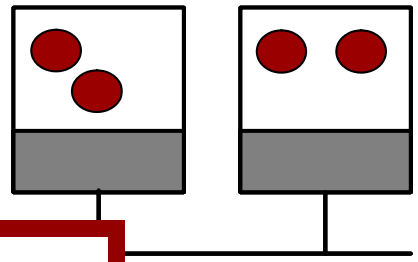
N_1	*true*	F_{P_1}	$\overline{F}_{P_1}$	$F_{P_1} \wedge F_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1} \wedge F_{P_2}$
P_1	0	35		70		
P_2			30	100	65	90
m_1			31	100	66	
m_2			105	105	105	
m_3				120		120



Overall Flow

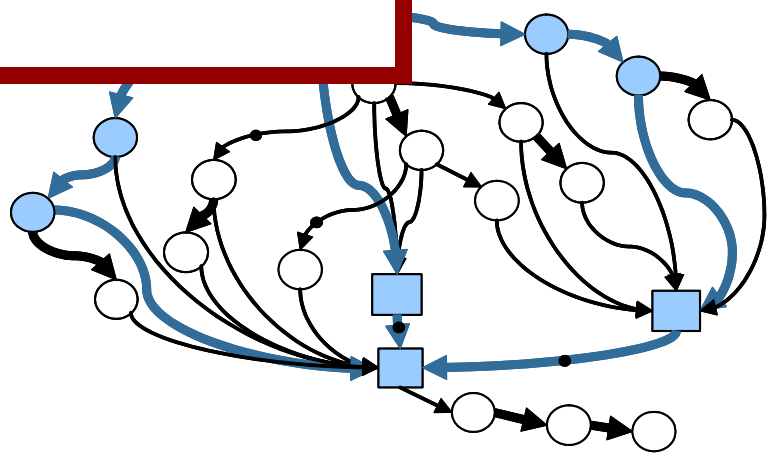






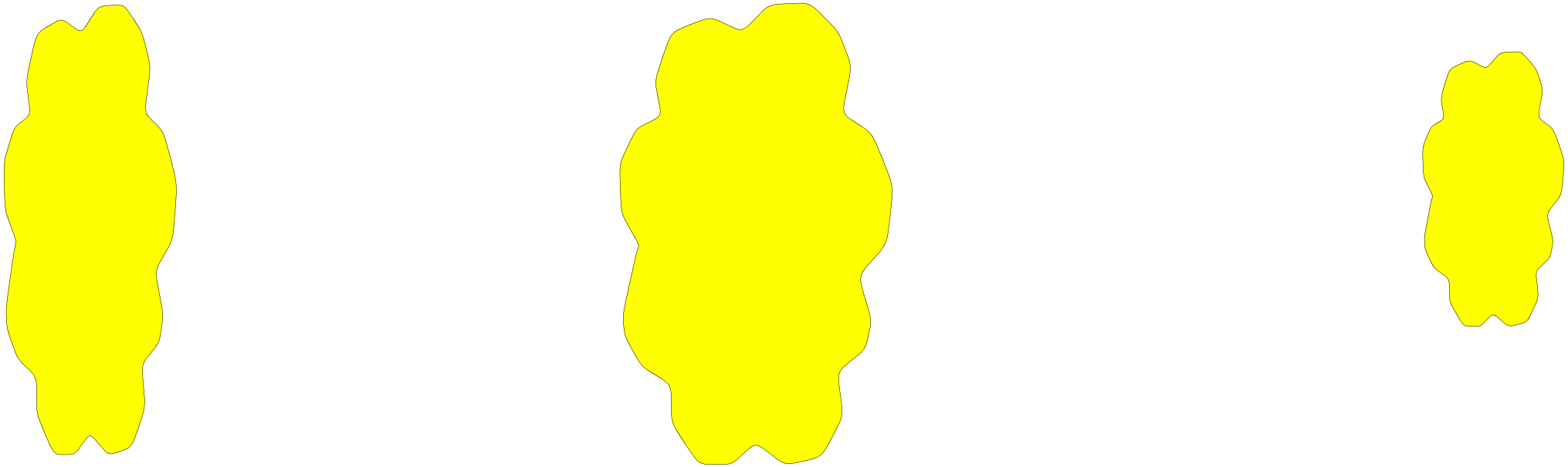
- **Optimized**
  - **Mapping&Schedule**
  - **Fault tolerance policy assignment**
  - **Checkpointing**

N_1	*true*	F_{P_1}	$\overline{F}_{P_1}$	F		
P_1	0	35		70		
P_2			30	100	65	90
m_1			31	100	66	
m_2			105	105	105	
m_3				120		120



Application Model

☞ An application is modelled as a set of process graphs:



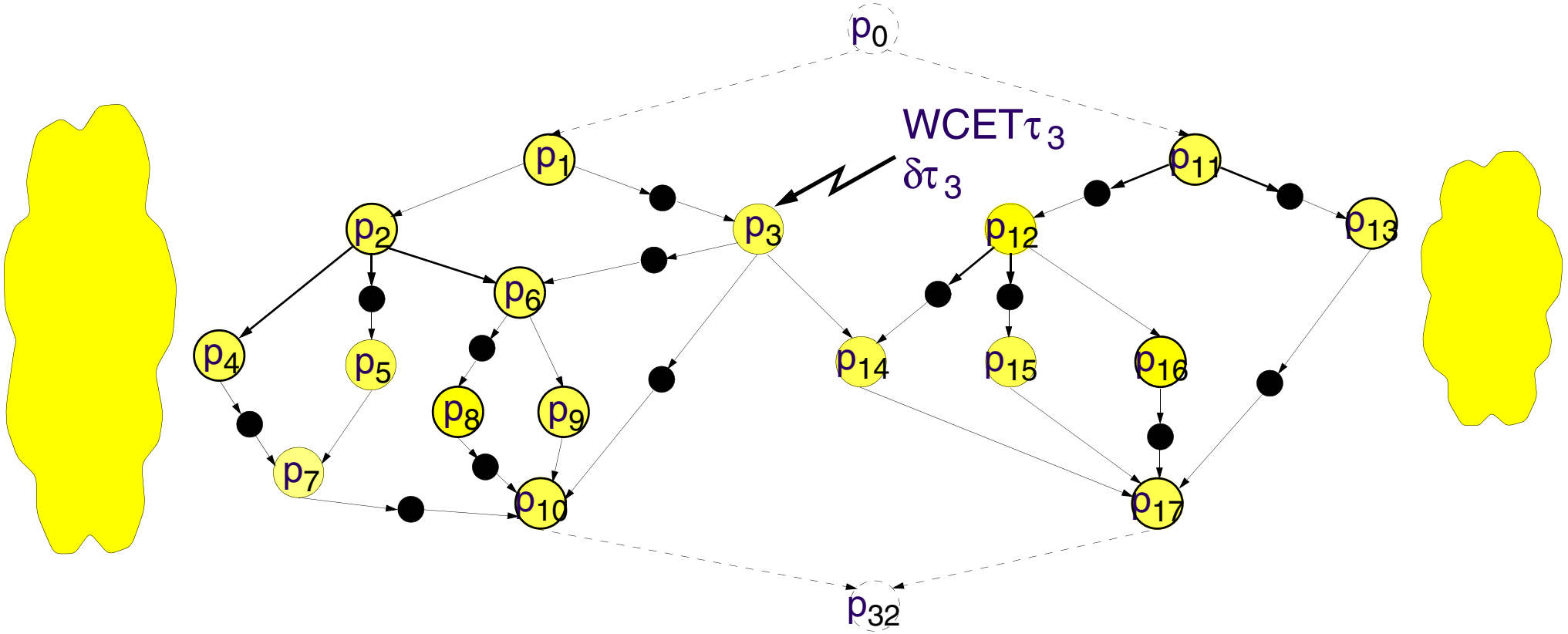
Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

Application Model

☞ An application is modelled as a set of process graphs:

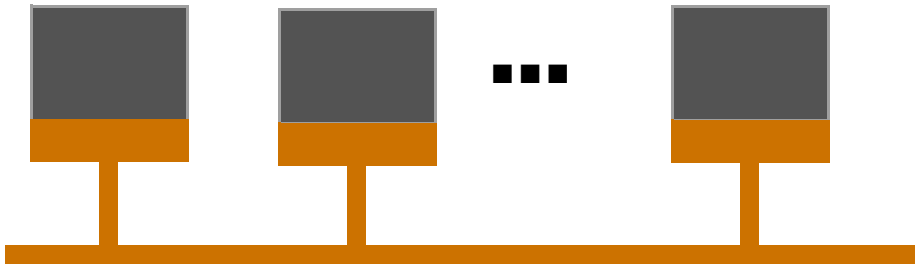


Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

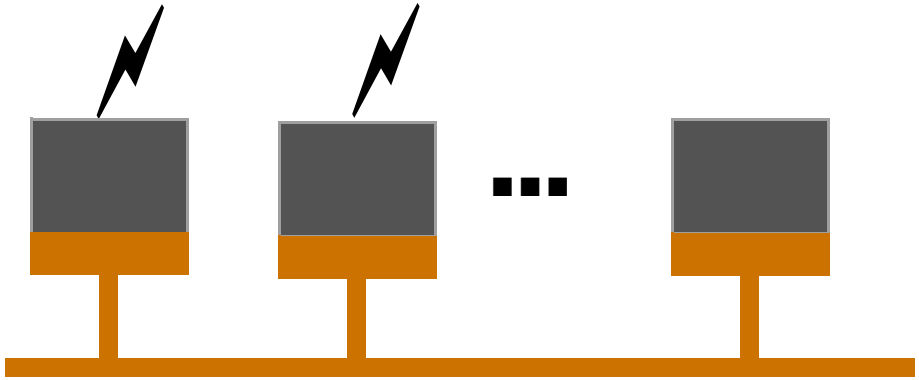
Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

System Model



- Fault tolerant TDMA bus

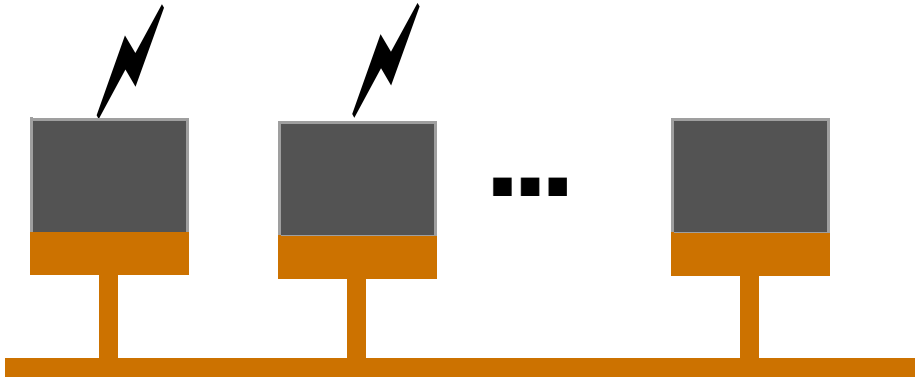
System Model



- Fault tolerant TDMA bus

☞ Max. *k* transient faults can occur during one operation cycle

System Model



- Fault tolerant TDMA bus

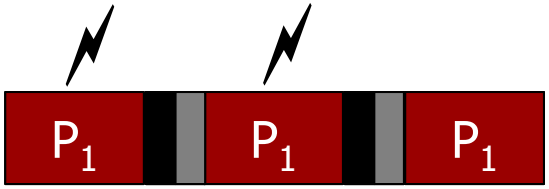
☞ Max. *k* transient faults can occur during one operation cycle

Overheads:
- Error detection overhead
- Recovery overhead
- Checkpointing overhead

Fault Tolerance Policies

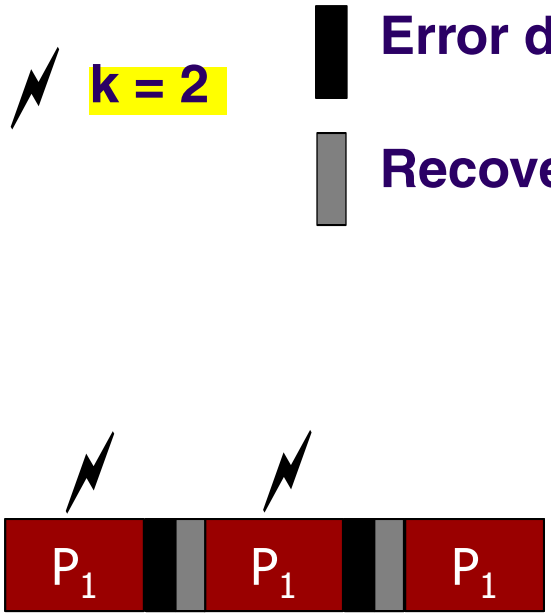
Error detection overhead

Recovery overhead

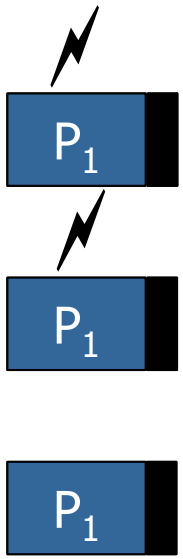


Re-execution

Fault Tolerance Policies



Re-execution



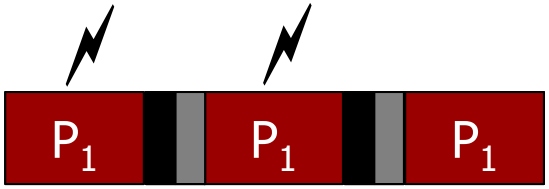
Replication

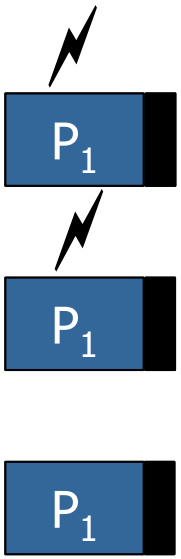
Fault Tolerance Policies

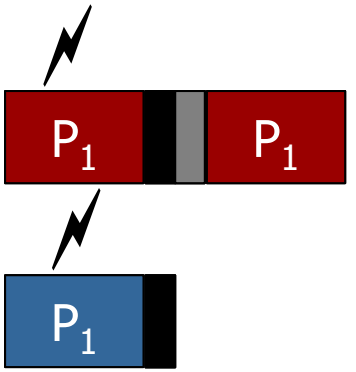
k = 2

Error detection overhead

Recovery overhead







Re-execution

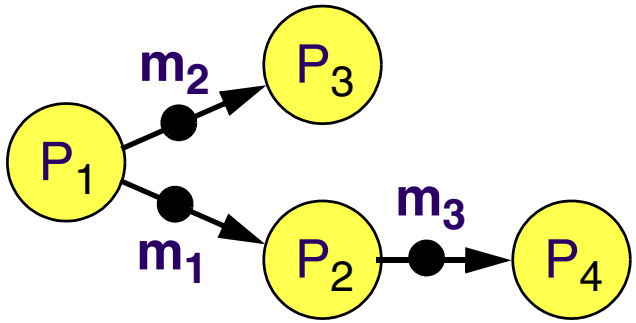
Replication

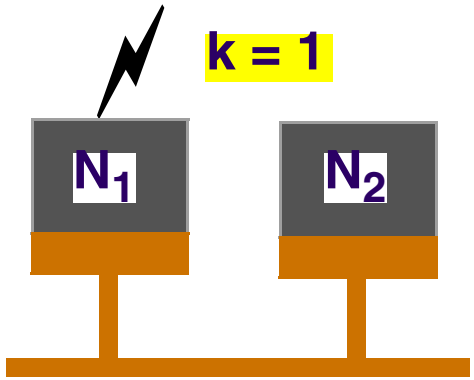
Re-execution&Replication

Outline

- Overall Flow, Application and System Model
- Fault Tolerance Policy Assignment
- Transparency
- Re-execution and Checkpointing
- Generation of Fault-Tolerant Schedules
- Cross-layer Optimization with Hardening Alternatives
- Conclusions

Fault Tolerance Policy Assignment

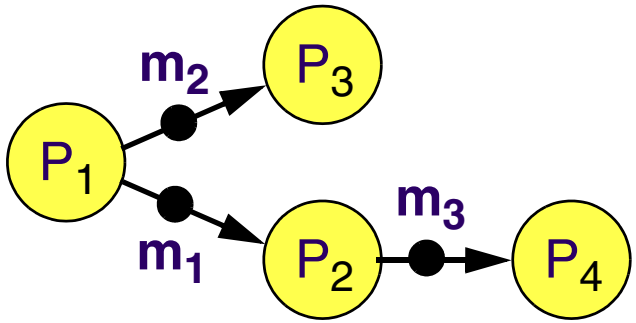


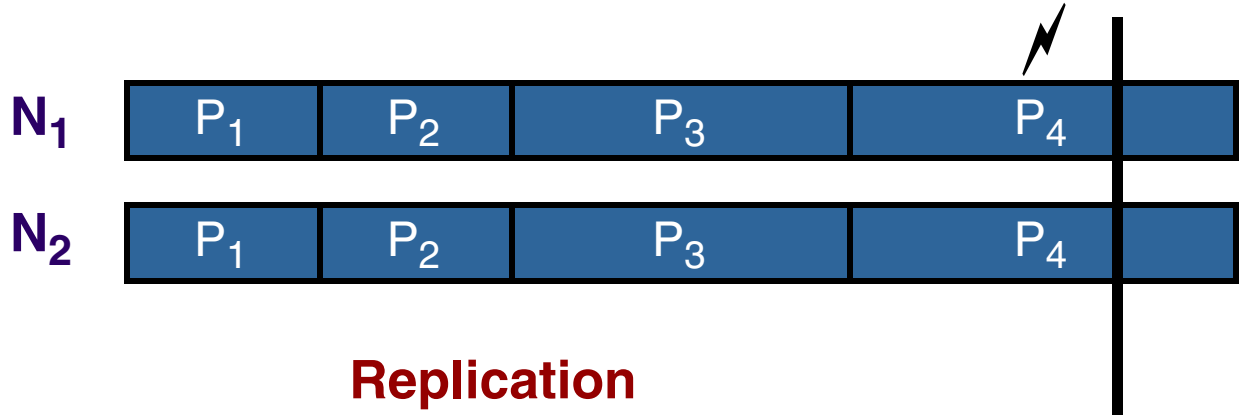


	N_1	N_2
P_1	40	40
P_2	40	40
P_3	70	70
P_4	80	80

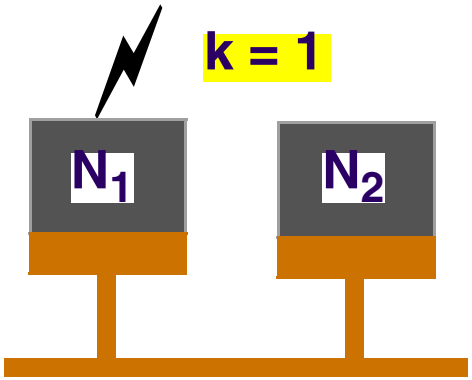
m_1: 10
m_2: 10
m_3: 10
▒ : 5
▉ : in WCET

Fault Tolerance Policy Assignment





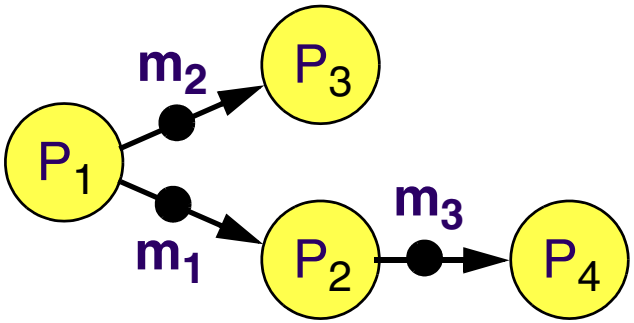
Replication

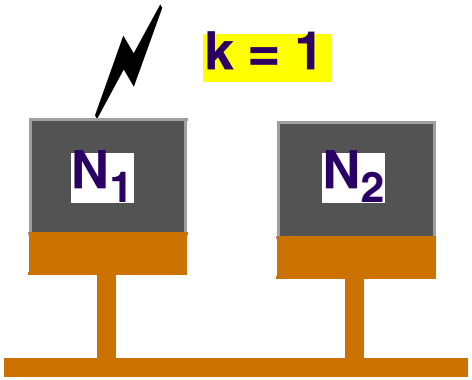


	N_1	N_2
P_1	40	40
P_2	40	40
P_3	70	70
P_4	80	80

m_1: 10
m_2: 10
m_3: 10
▒ : 5
▉ : in WCET

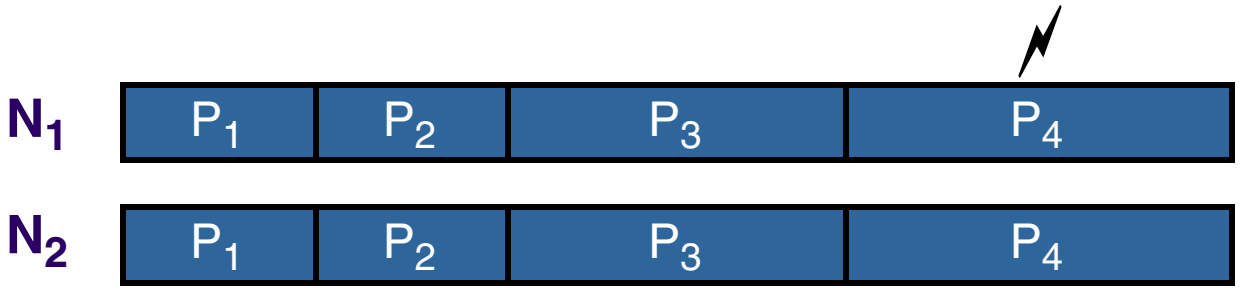
Fault Tolerance Policy Assignment

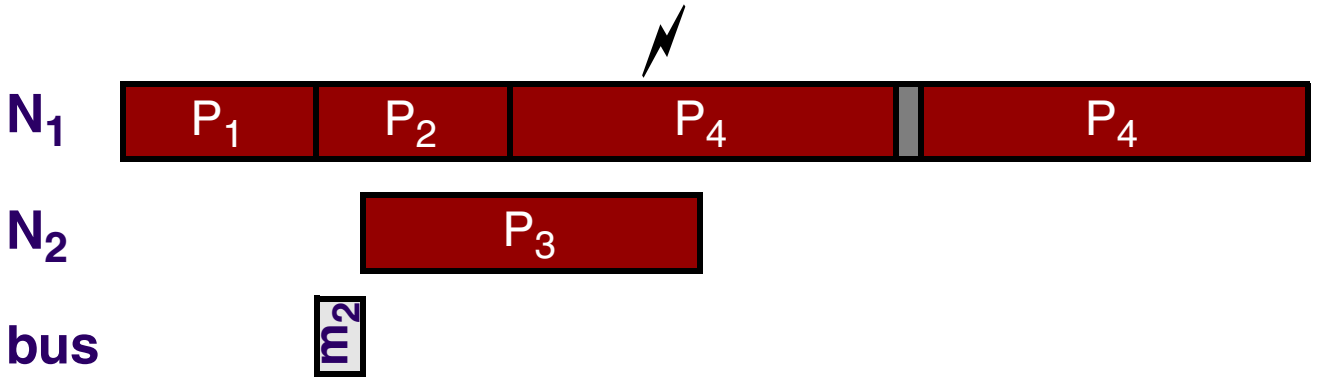




	N_1	N_2
P_1	40	40
P_2	40	40
P_3	70	70
P_4	80	80

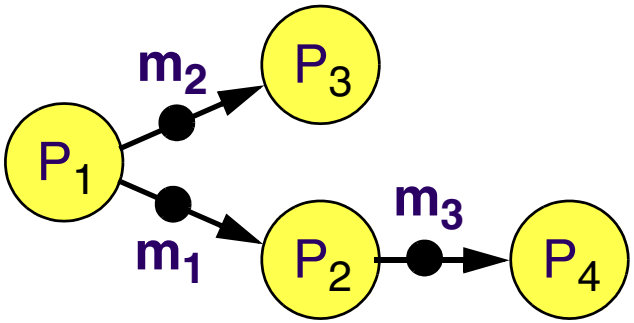
m_1: 10
m_2: 10
m_3: 10
▒ : 5
▉ : in WCET

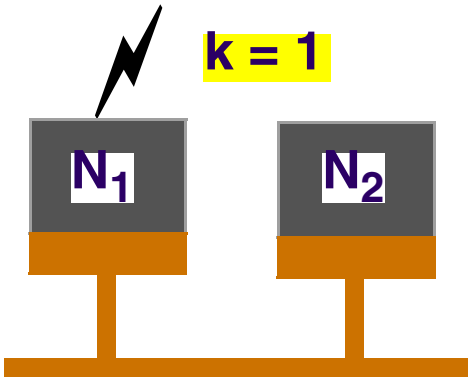




Re-execution

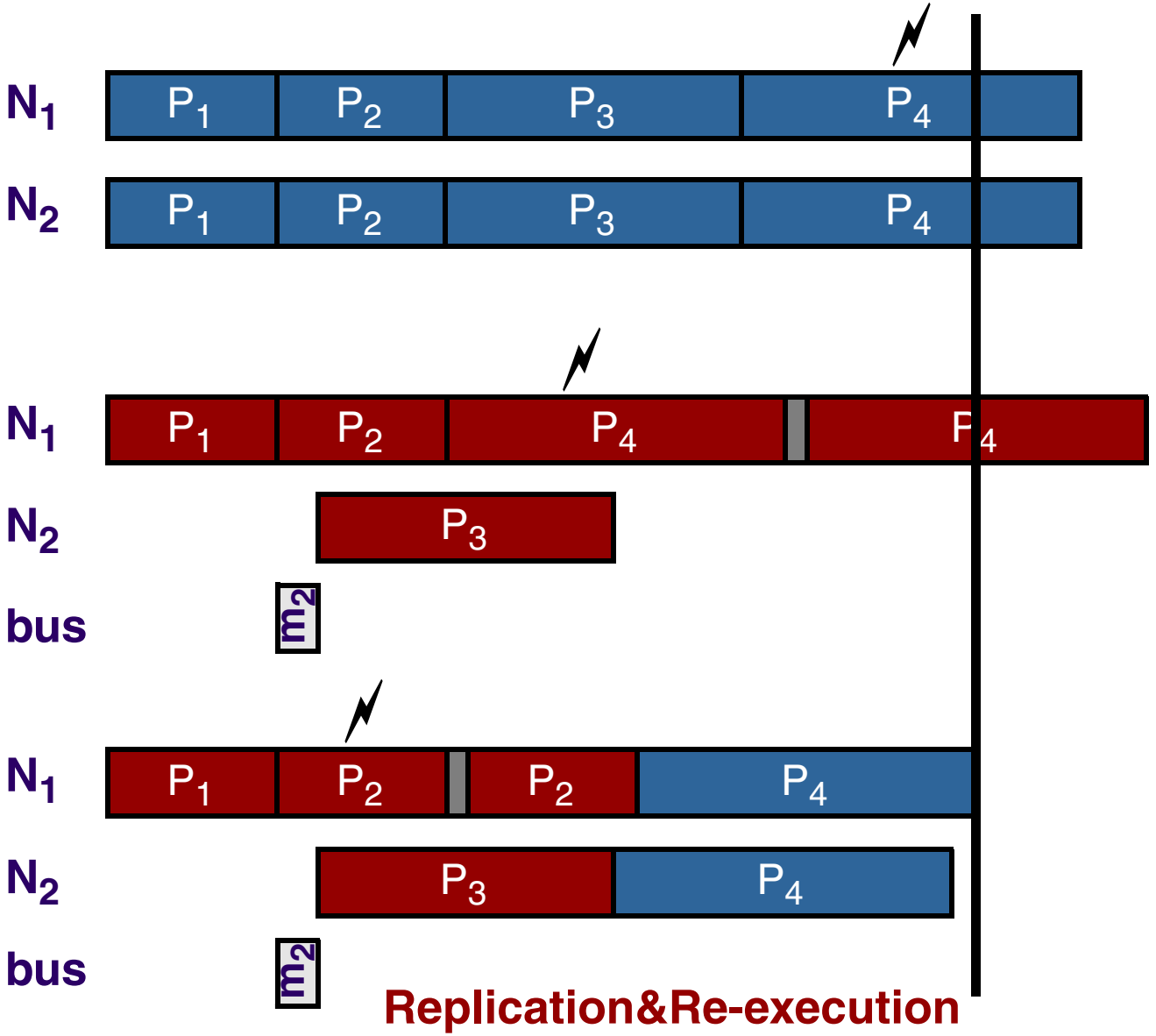
Fault Tolerance Policy Assignment





	N_1	N_2
P_1	40	40
P_2	40	40
P_3	70	70
P_4	80	80

m_1: 10
m_2: 10
m_3: 10
▒ : 5
▮ : in WCET



Fault Tolerance Policy Assignment

The Problem:

- **Find a fault tolerance policy assignment such that, with the available amount of resources, it is possible to generate a schedule which in the worst case satisfies the imposed deadlines.**

Transparency

- To what extent should fault occurrences in a certain part of the application affect the schedule of other parts?

Should a fault occurrence be hidden, or should the system schedules be changed such as to optimally adapt to the new situation?

Transparency

Transparency is good

- Easier to test, debug, verify
- Less complex schedules

Transparency

Transparency is good

- Easier to test, debug, verify
- Less complex schedules

But

It's not for free

- Performance overhead:

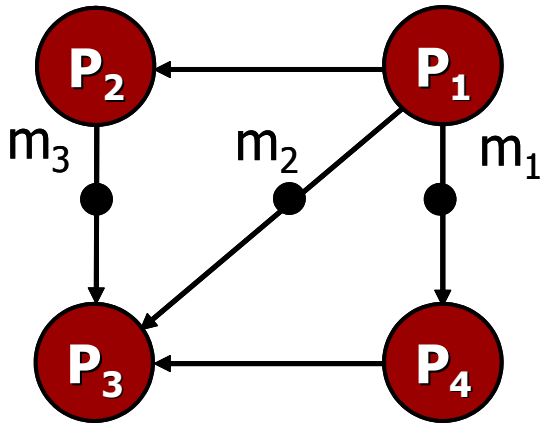
  Longer schedules

Transparency

☞ Transparency is achieved by *frozen* processes/messages: their start time is the same in all alternative schedules.

Transparency

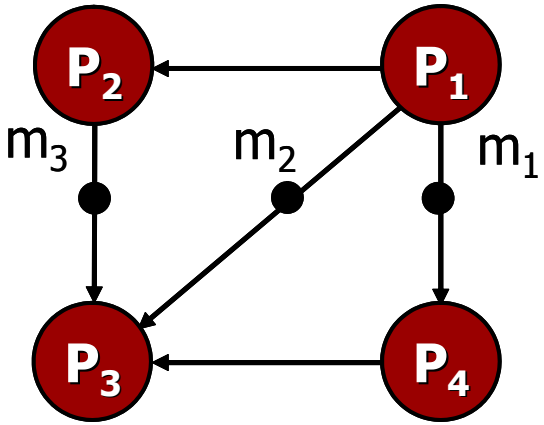
☞ Transparency is achieved by *frozen* processes/messages: their start time is the same in all alternative schedules.



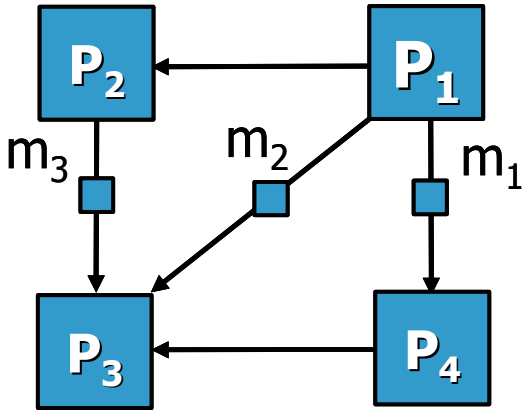
No transparency

Transparency

☞ Transparency is achieved by *frozen* processes/messages: their start time is the same in all alternative schedules.



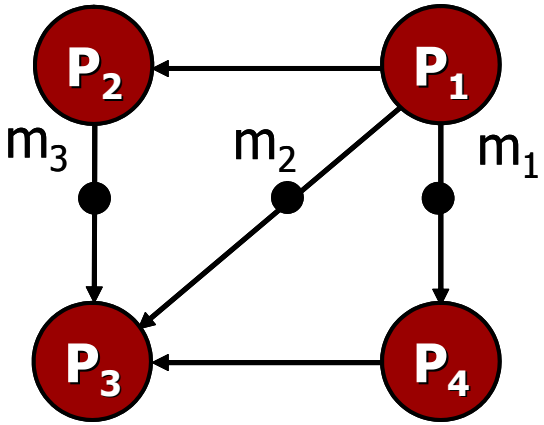
No transparency



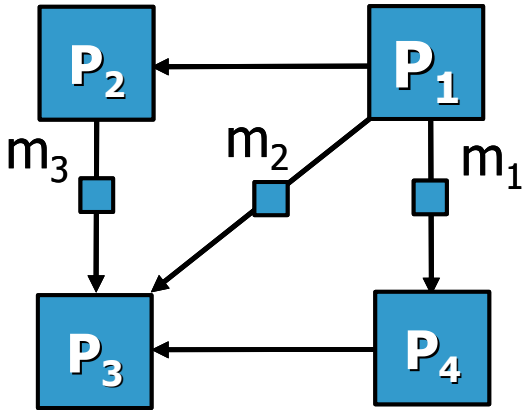
Full transparency

Transparency

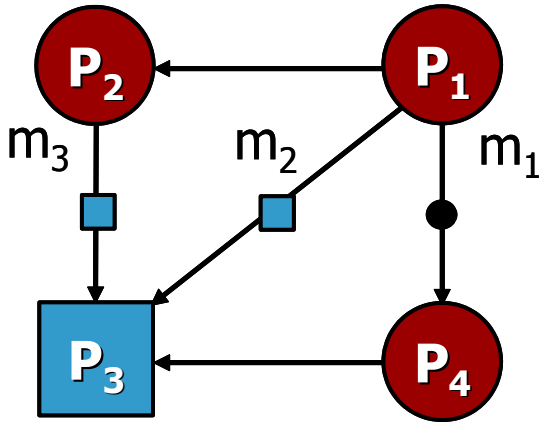
☞ Transparency is achieved by *frozen* processes/messages: their start time is the same in all alternative schedules.



No transparency

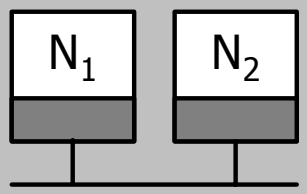


Full transparency



Customised transparency

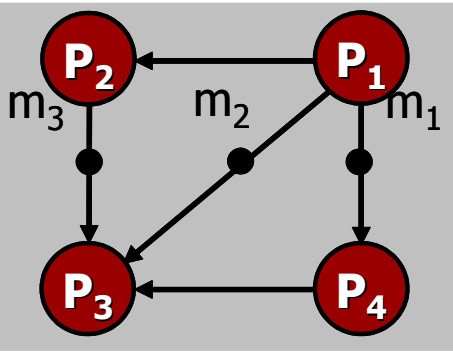
Transparency



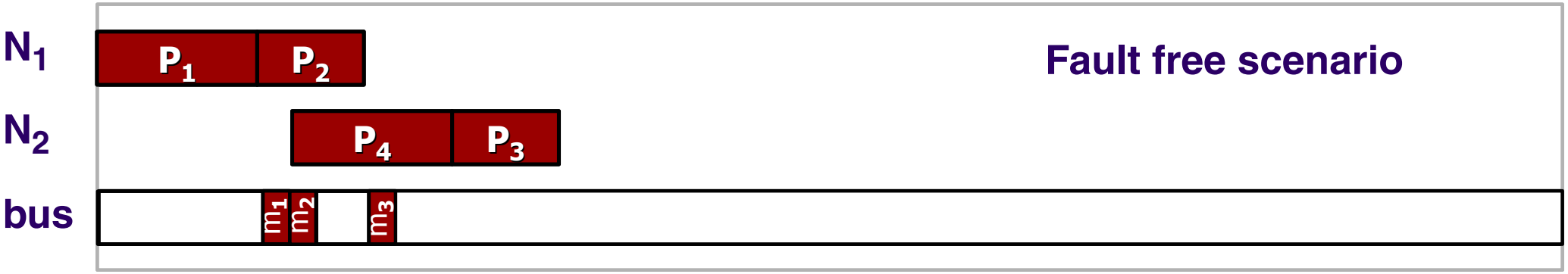
	N_1	N_2
P_1	30	X
P_2	20	X
P_3	X	20
P_4	X	30

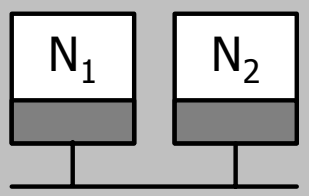
= 5 ms

$k = 2$



Non-Transparent Schedule

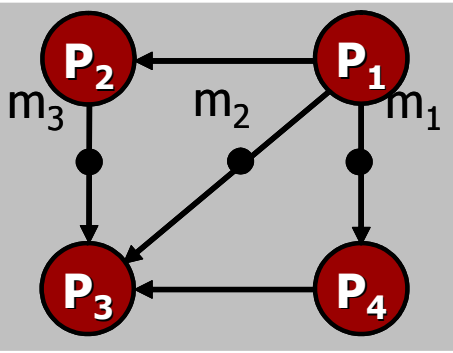




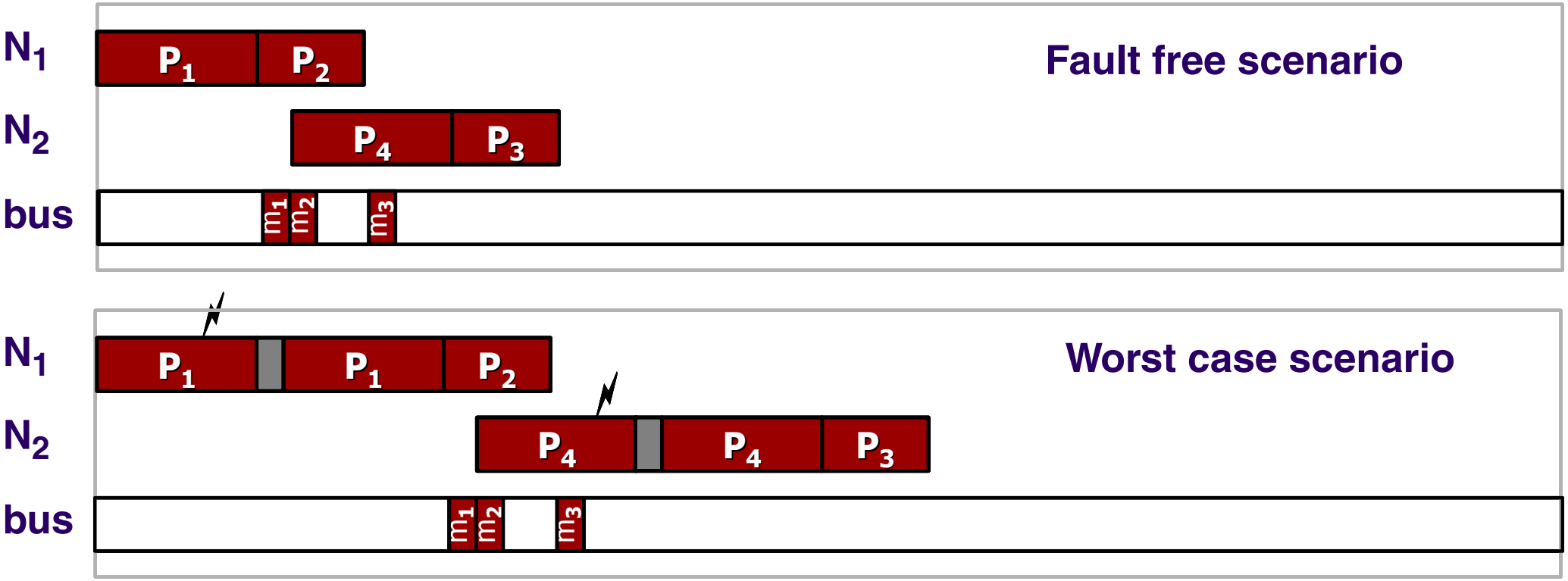
	N_1	N_2
P_1	30	X
P_2	20	X
P_3	X	20
P_4	X	30

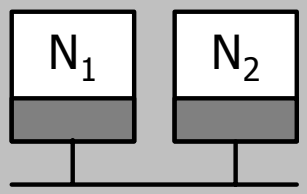
= 5 ms

$k = 2$



Non-Transparent Schedule

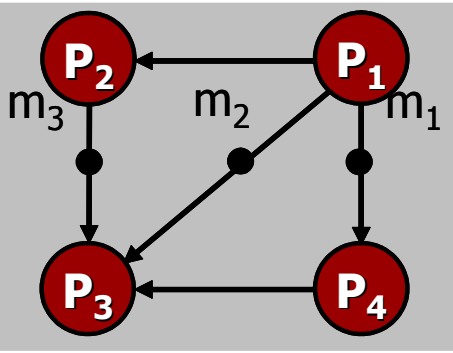




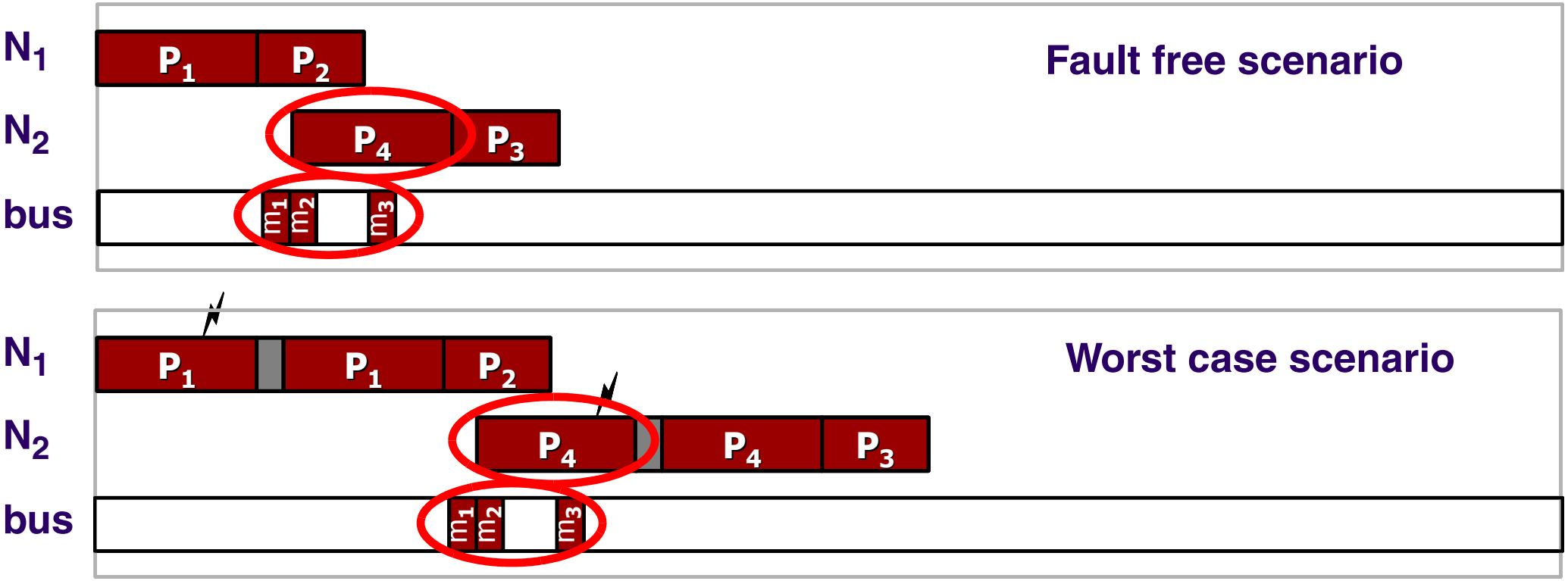
	N_1	N_2
P_1	30	X
P_2	20	X
P_3	X	20
P_4	X	30

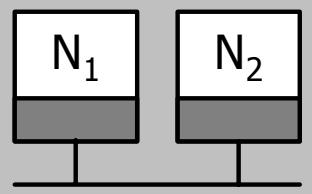
= 5 ms

$k = 2$



Non-Transparent Schedule

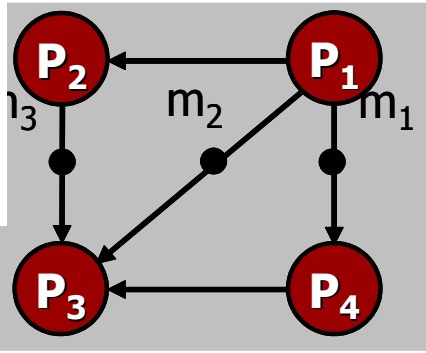




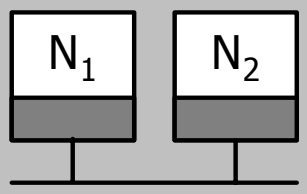
	N_1
P_1	30
P_2	20
P_3	X
P_4	X

Different start times in different scenarios.

$k = 2$



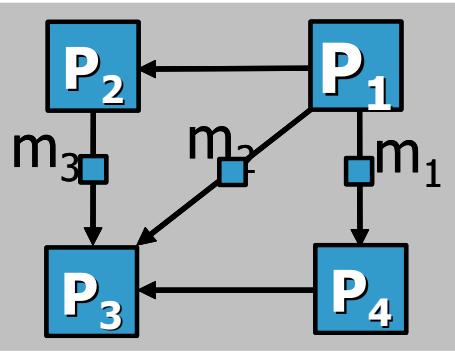
Fully Transparent Schedule



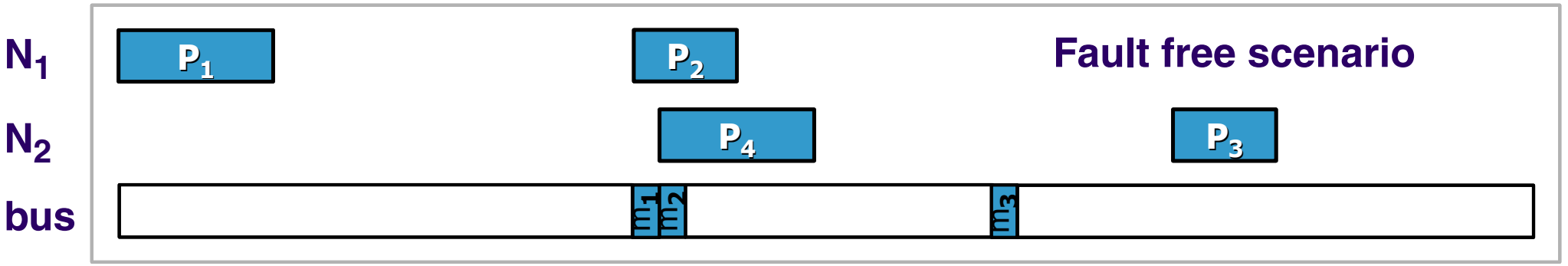
	N_1	N_2
P_1	30	X
P_2	20	X
P_3	X	20
P_4	X	30

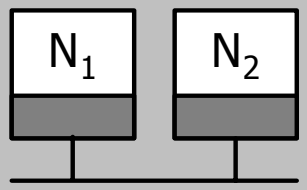
= 5 ms

$k = 2$



Fully Transparent Schedule

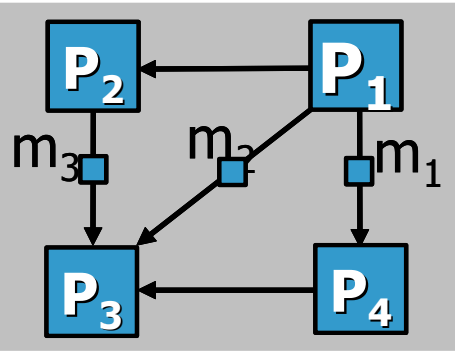




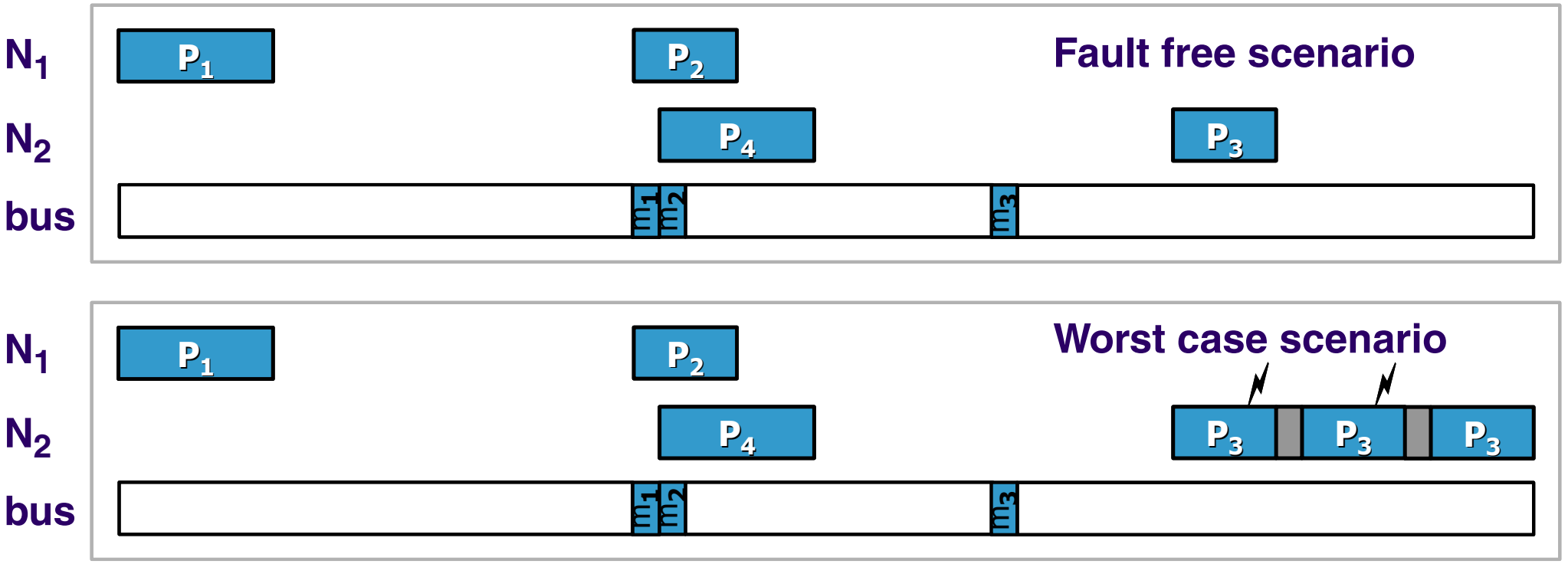
	N_1	N_2
P_1	30	X
P_2	20	X
P_3	X	20
P_4	X	30

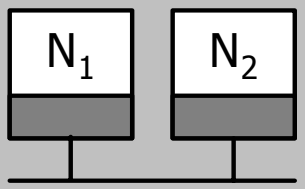
= 5 ms

$k = 2$



Fully Transparent Schedule

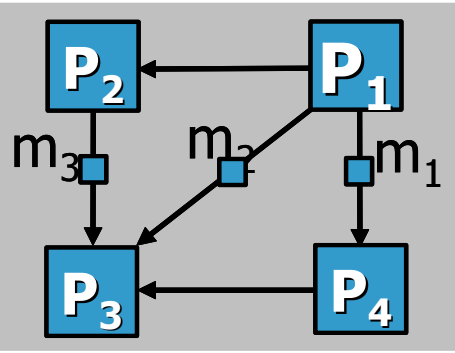




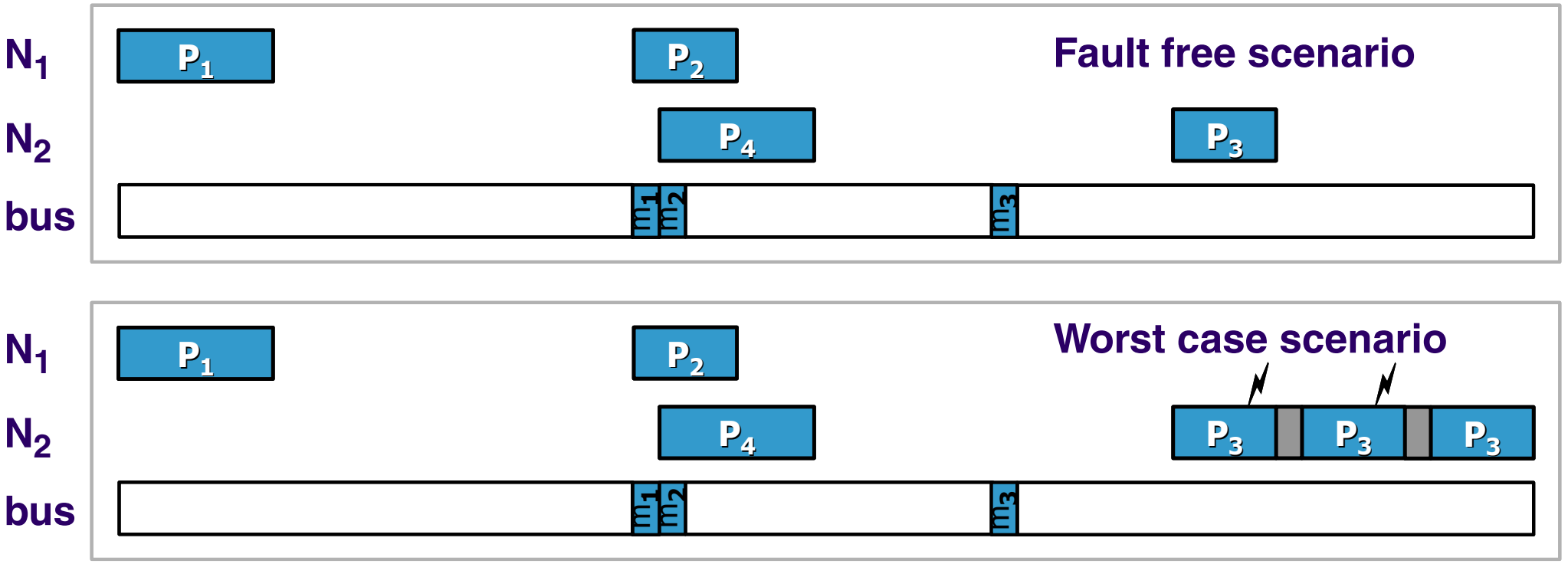
	N_1	N_2
P_1	30	X
P_2	20	X
P_3	X	20
P_4	X	30

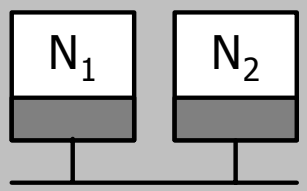
= 5 ms

$k = 2$



Fully Transparent Schedule

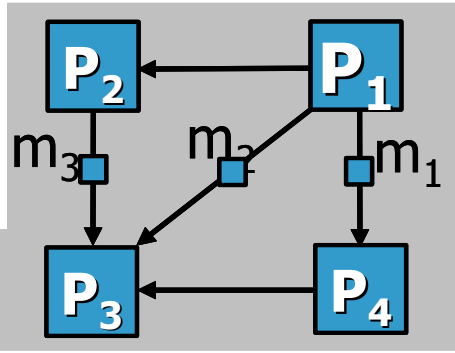




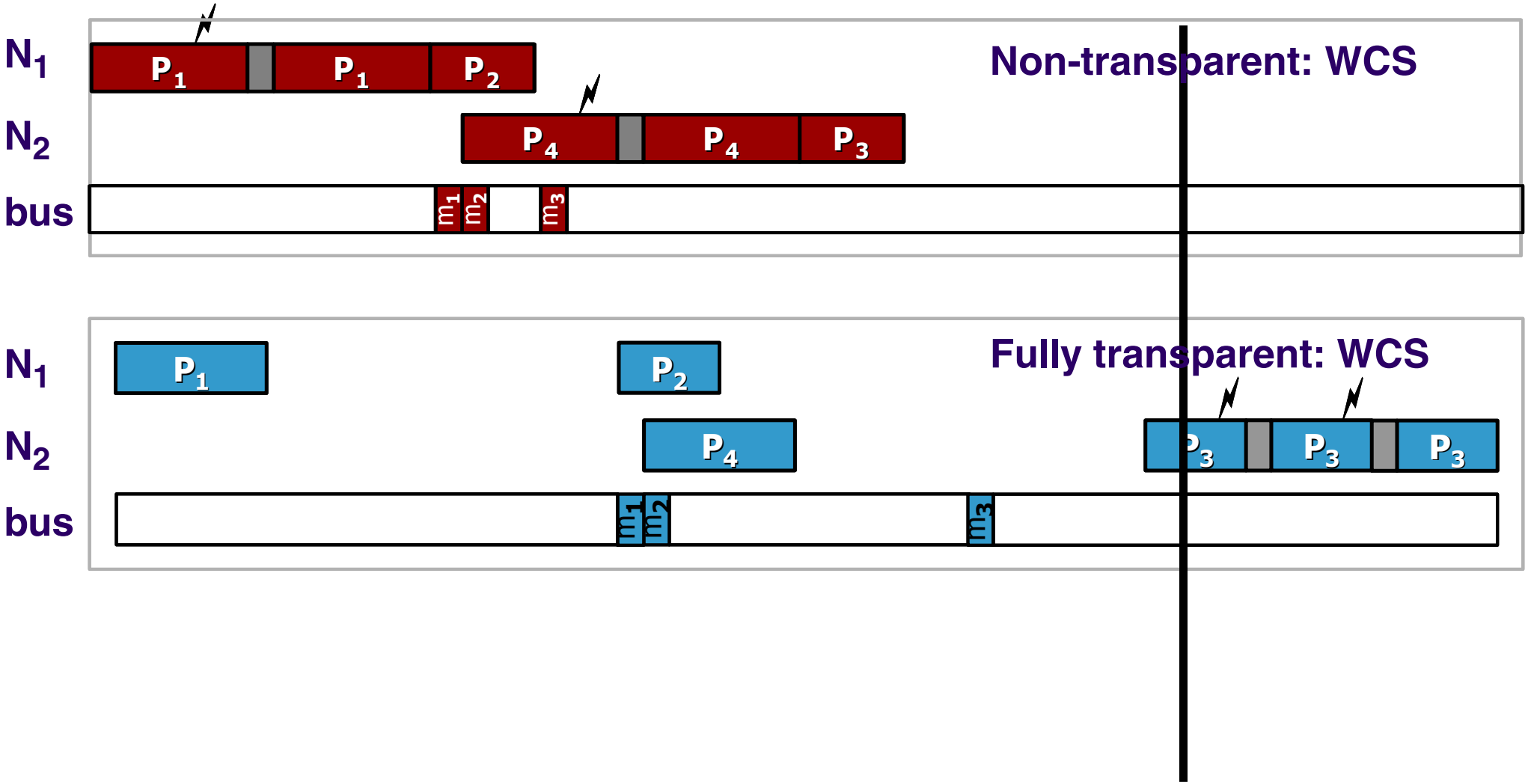
	N_1
P_1	30
P_2	20
P_3	X
P_4	X

Same start times in all scenarios.

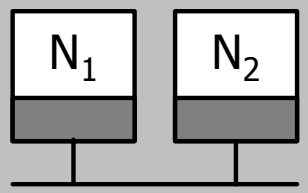
$k = 2$



Fully Transparent vs. Non-Transparent Schedule



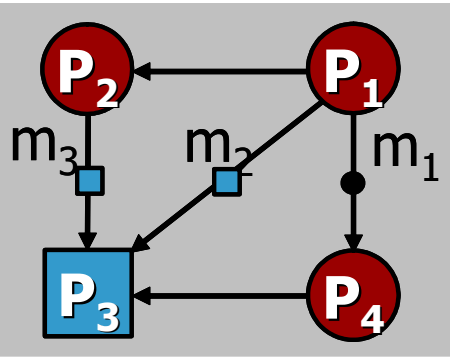
Customized Transparency



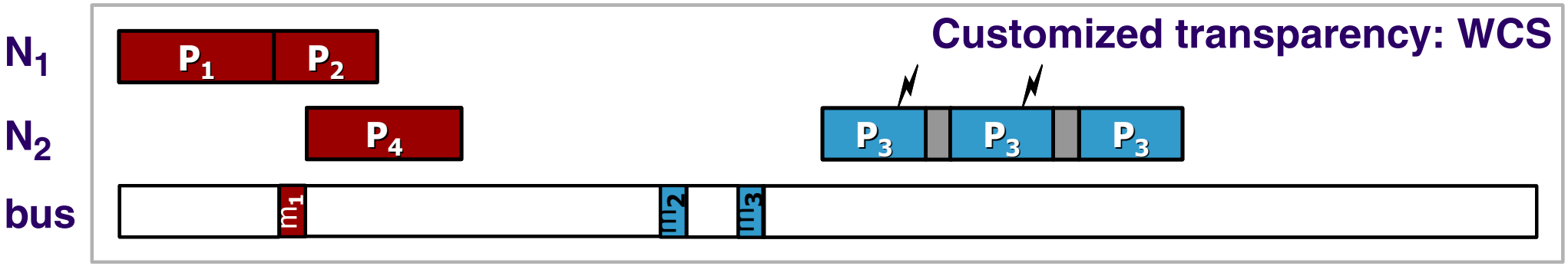
	N_1	N_2
P_1	30	X
P_2	20	X
P_3	X	20
P_4	X	30

= 5 ms

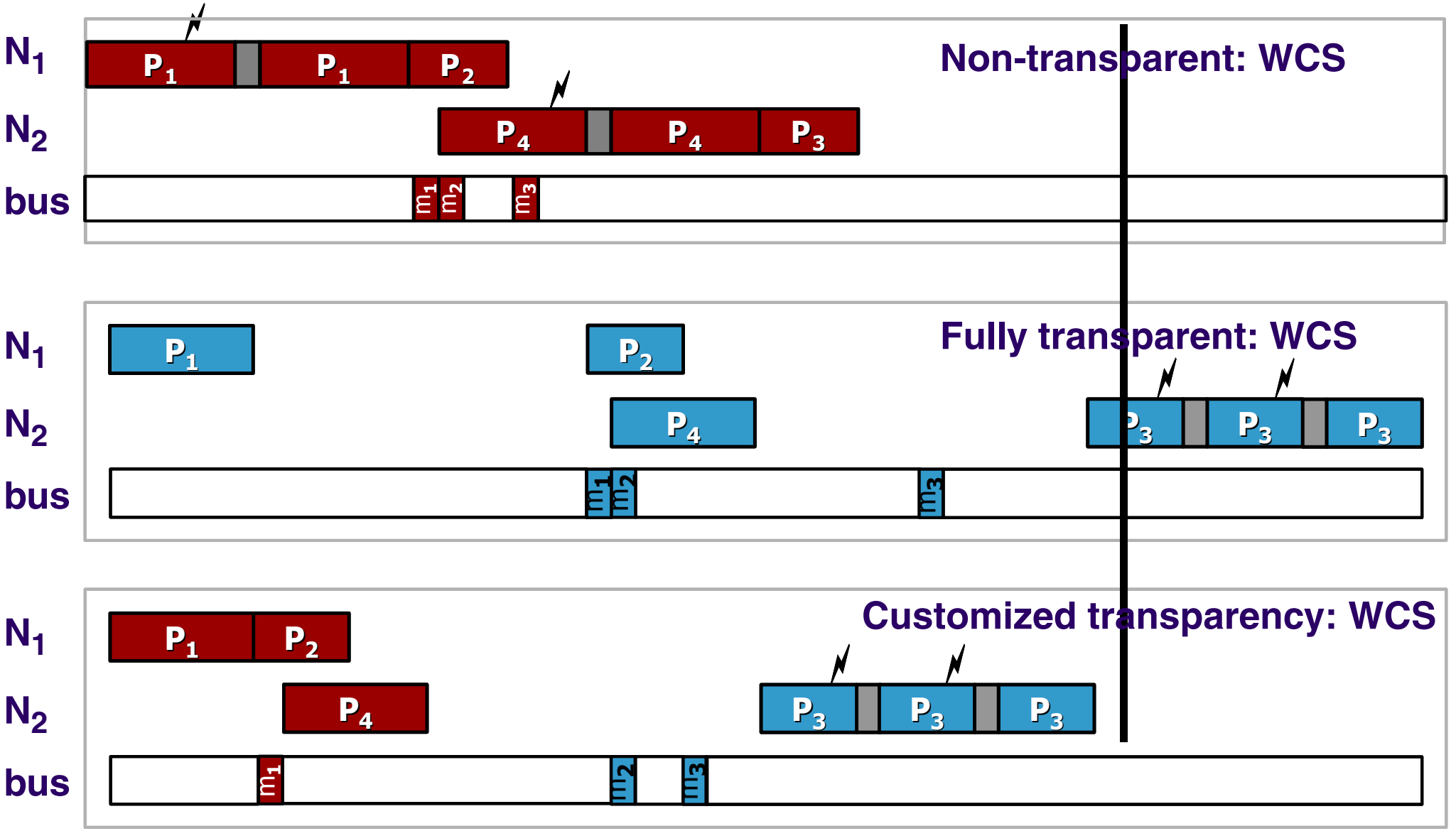
$k = 2$



Customized Transparency



Customized Transparency



Customized Transparency

The Problem:

- **Generate an efficient schedule with a customized degree of transparency, according to the designer's specification.**

For example: Isolate fault occurrences from one processor to another by freezing interprocessor messages.

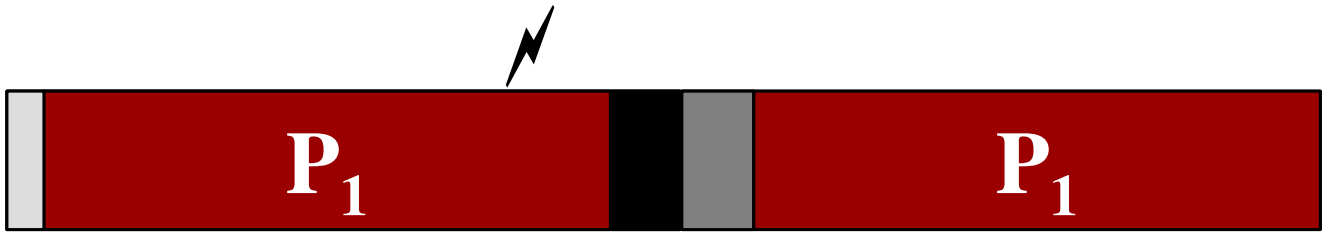
Re-execution with Checkpointing

Error detection overhead

k = 1

Recovery overhead

Checkpointing overhead



Re-execution

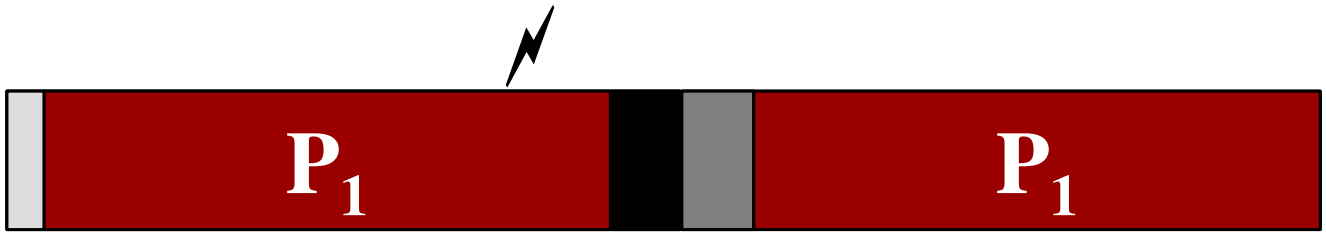
Re-execution with Checkpointing

Error detection overhead

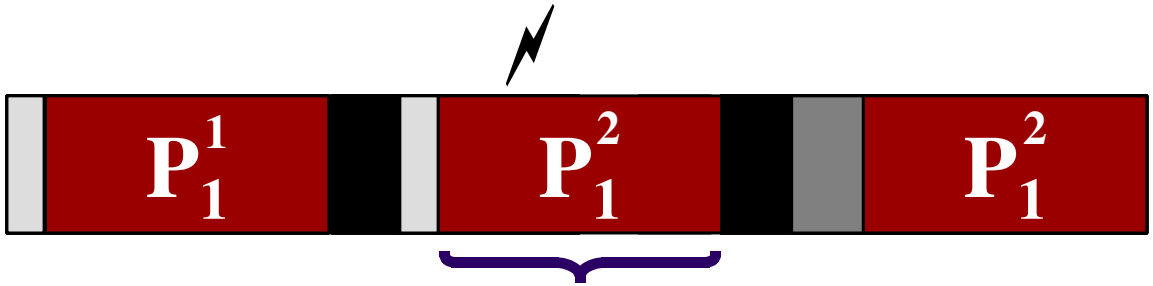
Recovery overhead

Checkpointing overhead

k = 1



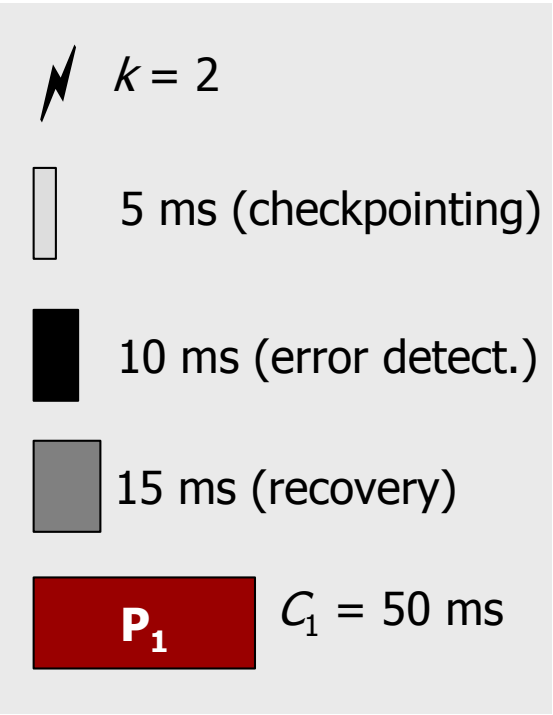
Re-execution



Re-execution with checkpointing

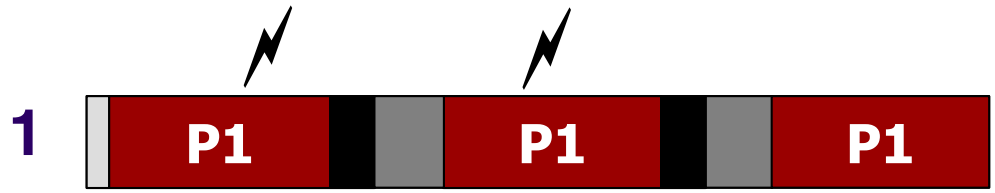
Only this segment
is re-executed

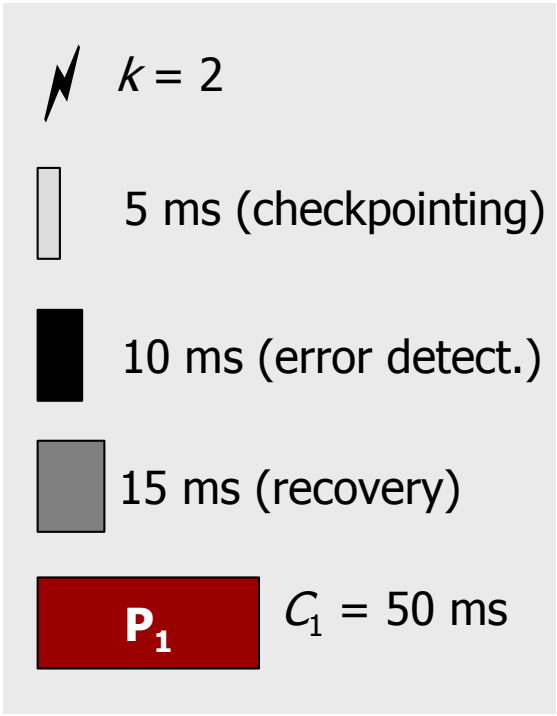
Checkpoint Optimization



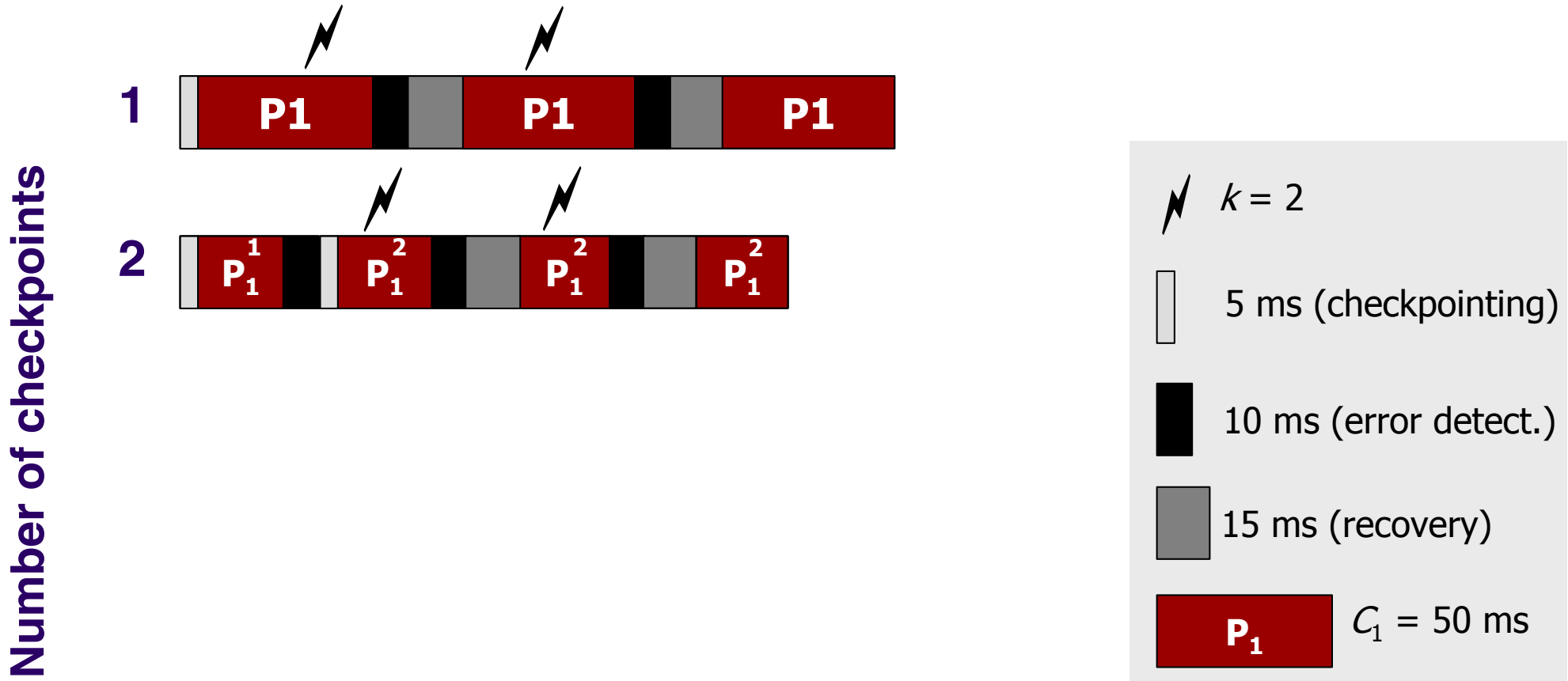
Checkpoint Optimization

Number of checkpoints

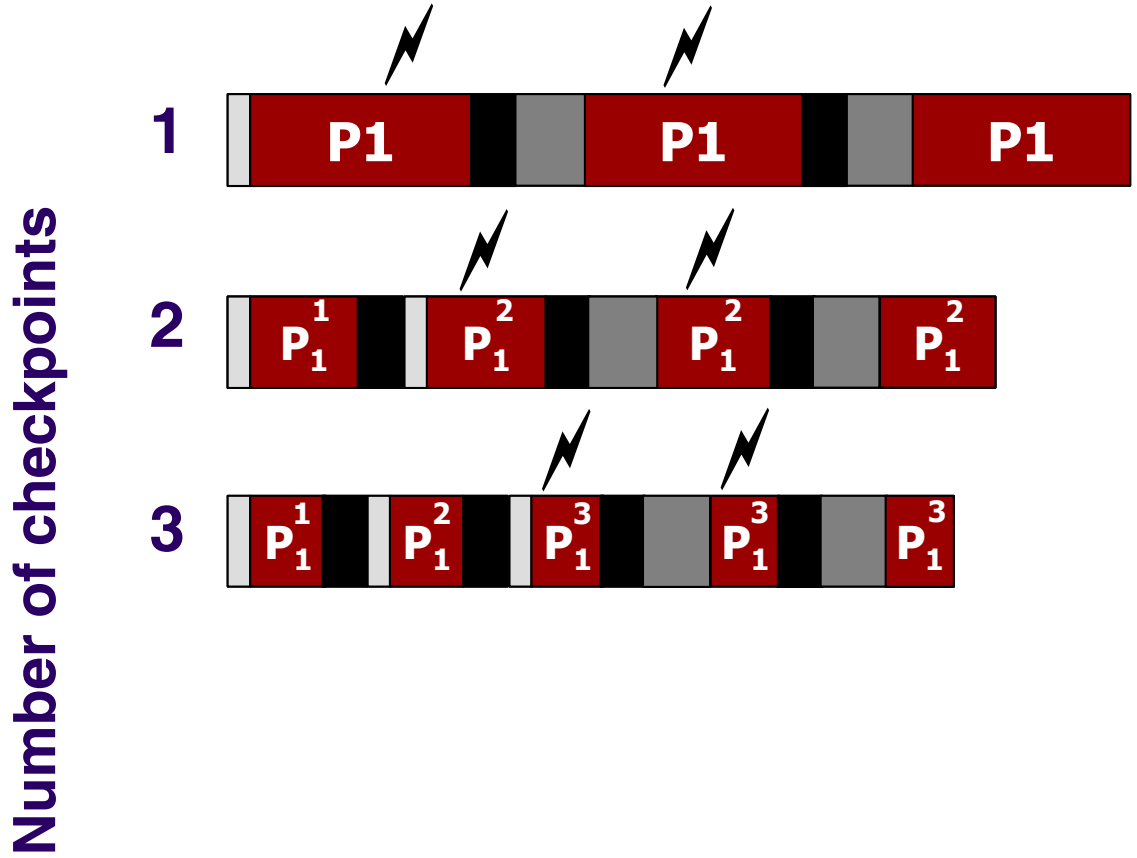




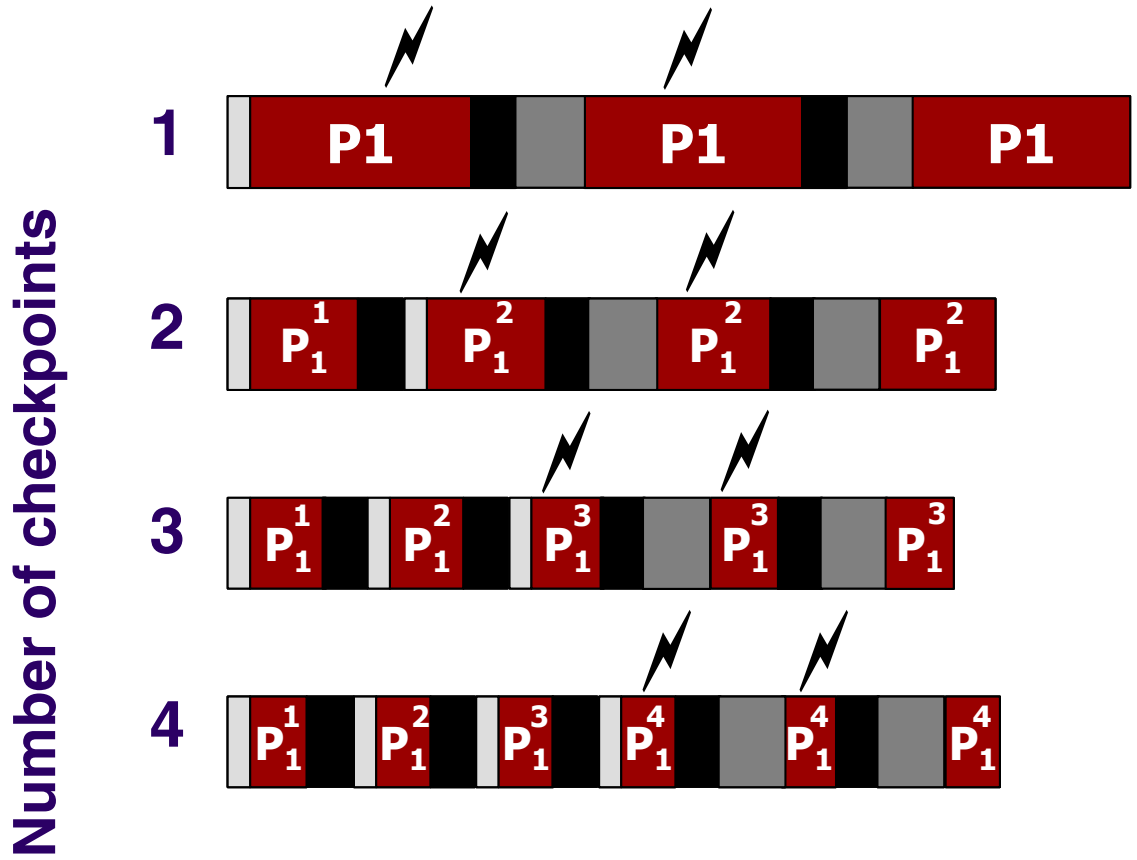
Checkpoint Optimization

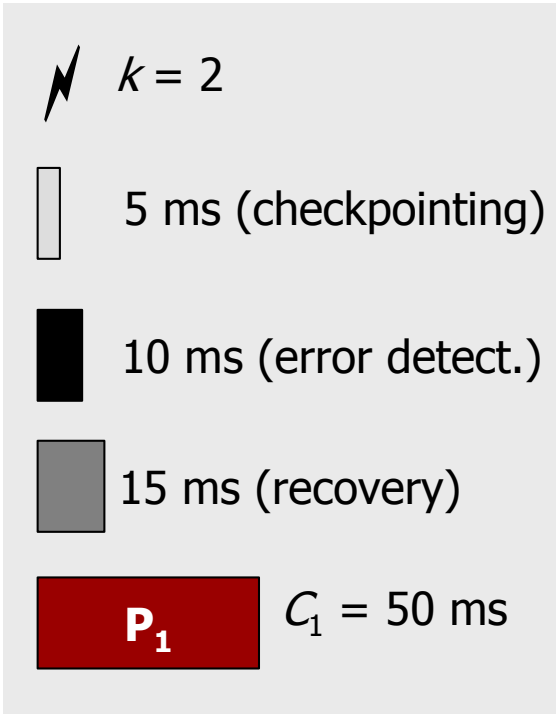


Checkpoint Optimization

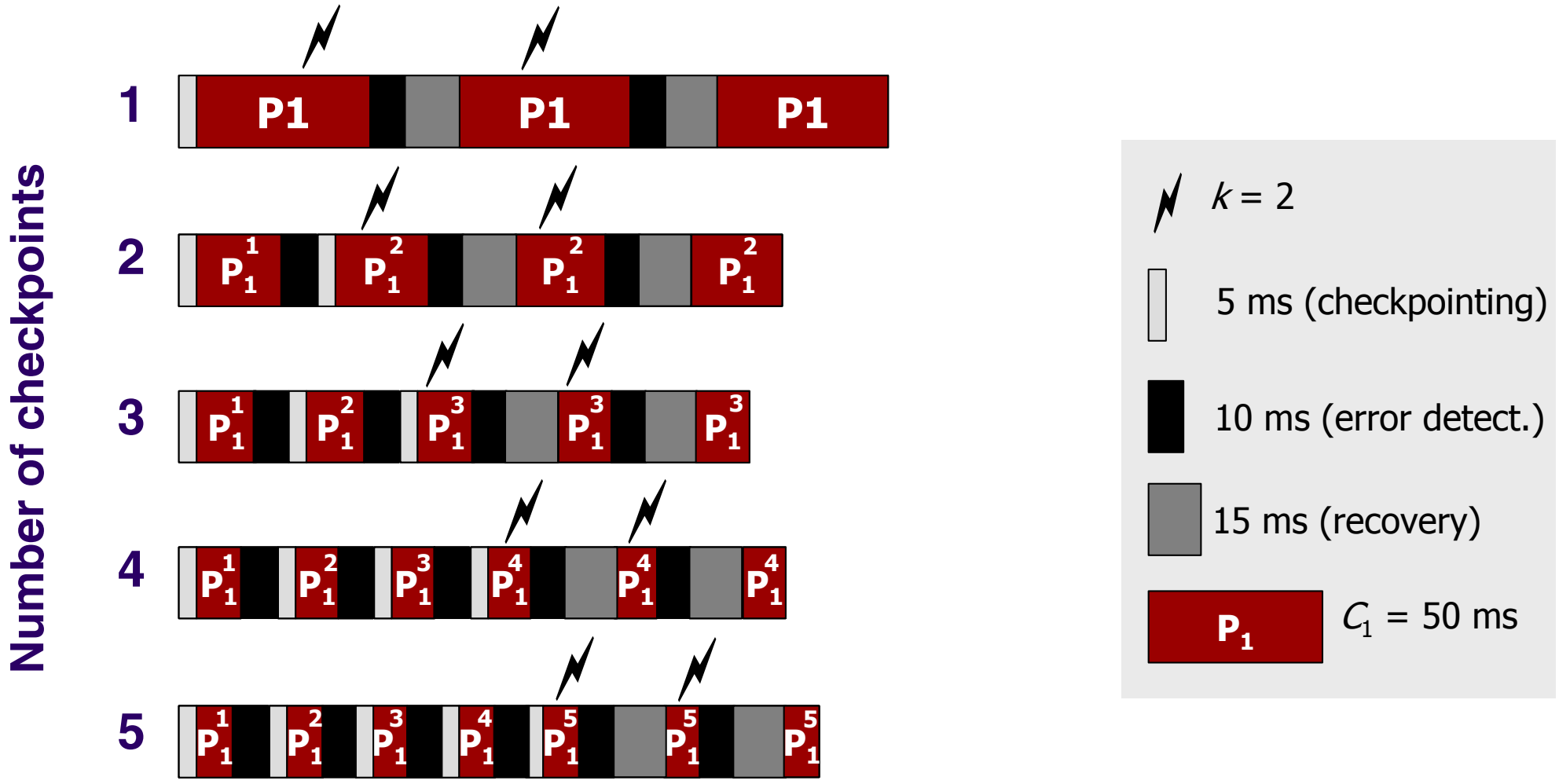


Checkpoint Optimization

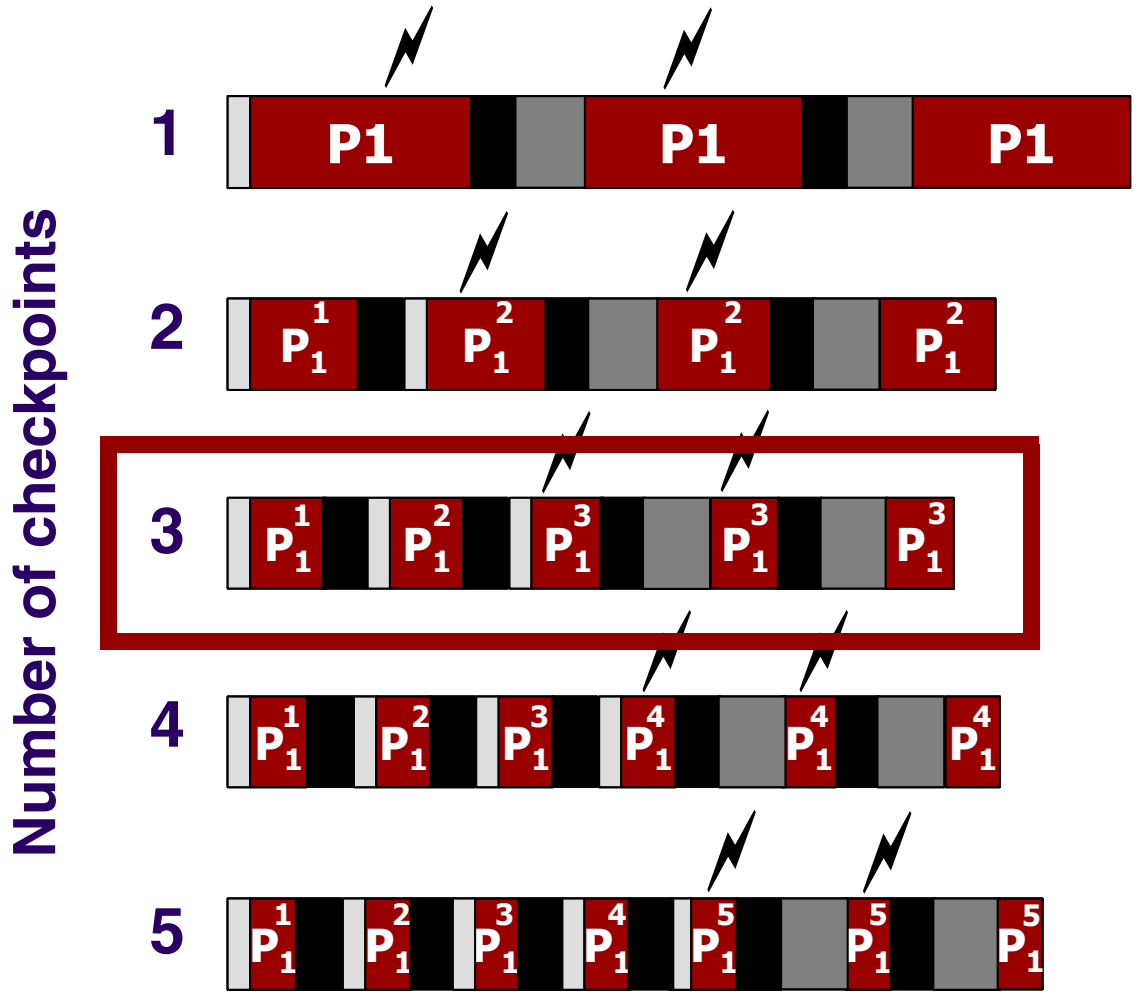




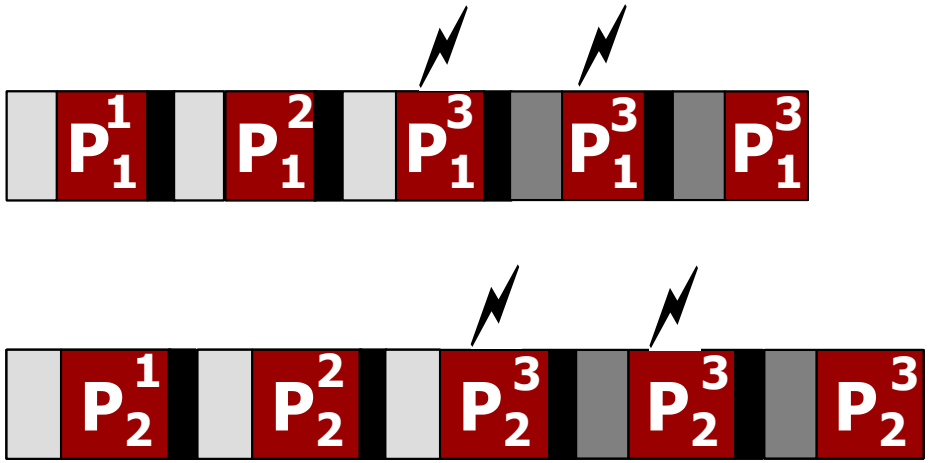
Checkpoint Optimization



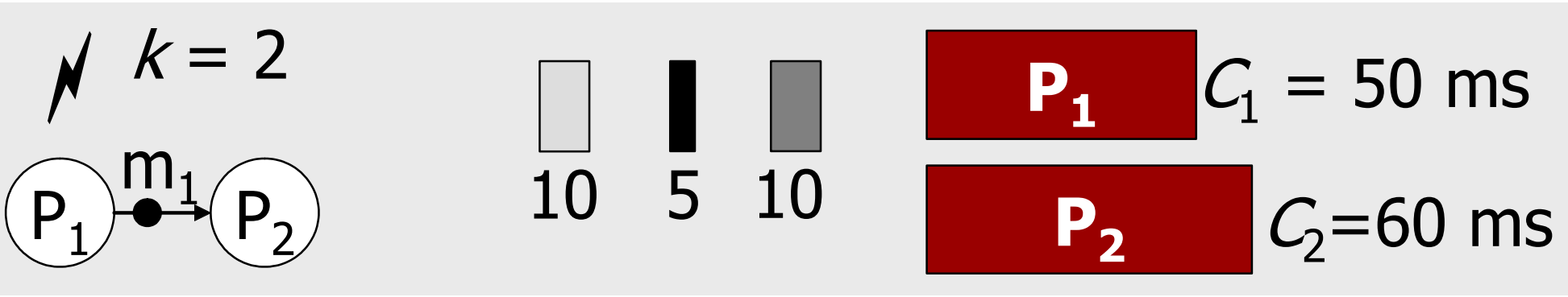
Checkpoint Optimization



Checkpoint Optimization

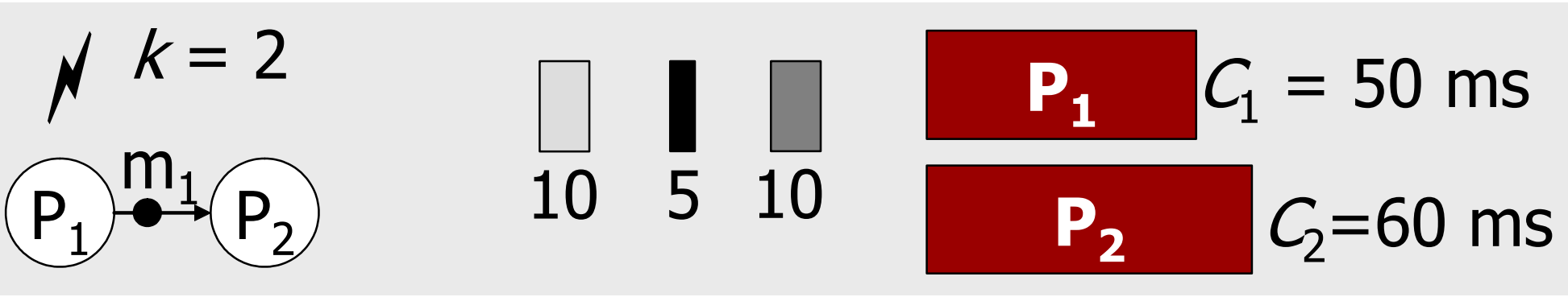


- The optimal number of checkpoints for both P_1 and P_2, if considered in isolation, is *three*!



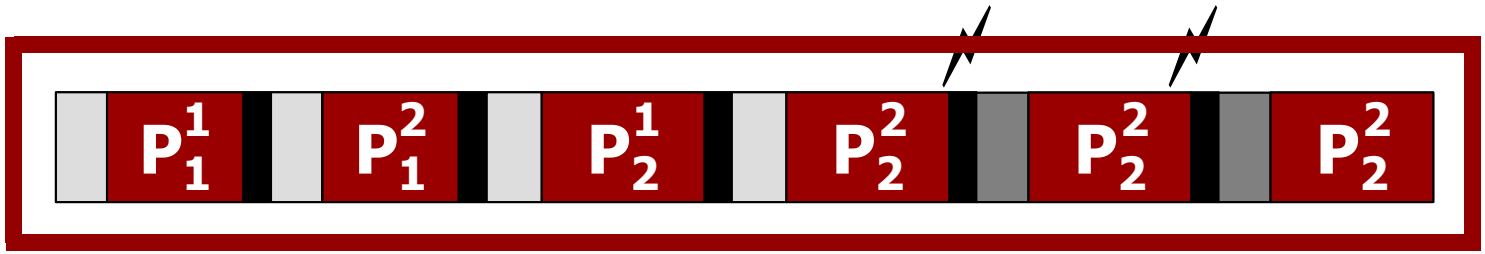
Checkpoint Optimization

3 checkpoints

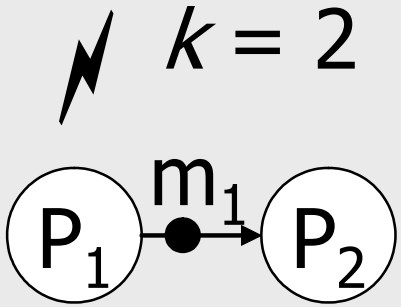


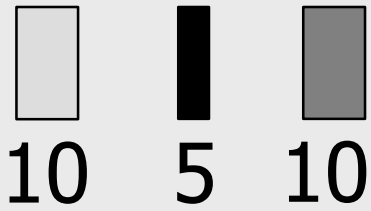
Checkpoint Optimization

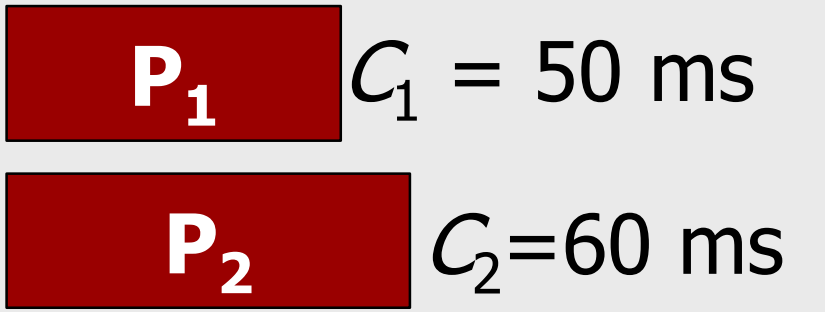
3 checkpoints



2 checkpoints!







Checkpoint Optimization

The Problem:

- **Find the number of checkpoints for each process using re-execution, such that, with the available amount of resources, it is possible to generate a schedule which in the worst case satisfies the imposed deadlines.**

The “Big” Problem

- Given:
  - Application (set of process graphs) + imposed deadlines
  - System architecture (processors, buses)
  - Maximum number of transient faults
  - Overheads (error detection, re-execution, checkpointing)
  - Transparency requirements

The “Big” Problem

- Given:
  - Application (set of process graphs) + imposed deadlines
  - System architecture (processors, buses)
  - Maximum number of transient faults
  - Overheads (error detection, re-execution, checkpointing)
  - Transparency requirements

- Find:
  - Fault tolerance policy assignment for each process
  - Number of checkpoints for processes to be re-executed
  - Mapping of processes and of their replica
  - Schedule

The “Big” Problem

- Given:
  - Application (set of process graphs) + imposed deadlines
  - System architecture (processors, buses)
  - Maximum number of transient faults
  - Overheads (error detection, re-execution, checkpointing)
  - Transparency requirements
- Find:
  - Fault tolerance policy assignment for each process
  - Number of checkpoints for processes to be re-executed
  - Mapping of processes and of their replica
  - Schedule
- Such that:
  - Deadlines are satisfied in the worst case

The “Big” Problem

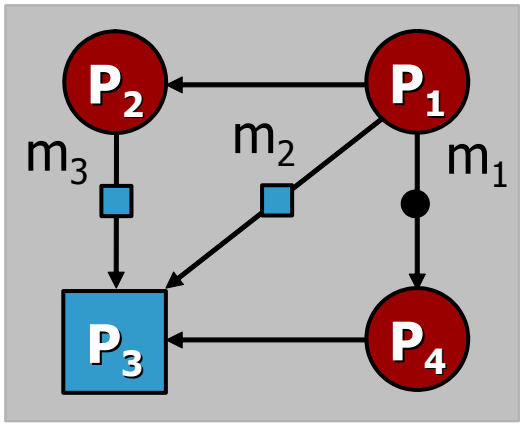
- Given:
  - Application (set of process graphs) + imposed deadlines
  - System architecture (processors, buses)
  - Maximum number of transient faults
  - Ove kpointing)
  - Tran
- Find:
  - Faul cess
  - Number of checkpoints for processes to be re-executed
  - Mapping of processes and of their replica
  - Schedule
- Such that:
  - Deadlines are satisfied in the worst case

A combination of greedy heuristics and tabu search based design space exploration.

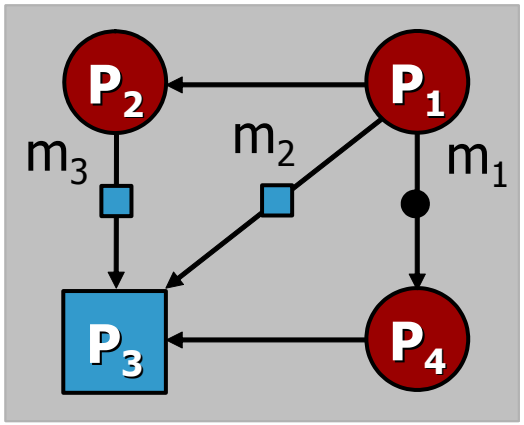
Outline

- Overall Flow, Application and System Model
- Fault Tolerance Policy Assignment
- Transparency
- Re-execution and Checkpointing
- **Generation of Fault-Tolerant Schedules**
- Cross-layer Optimization with Hardening Alternatives
- Conclusions

Generation of Fault-Tolerant Schedules

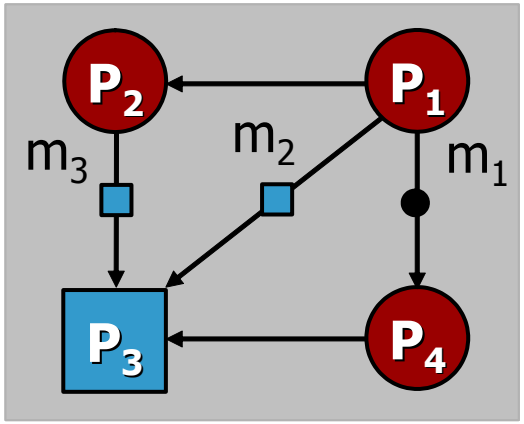


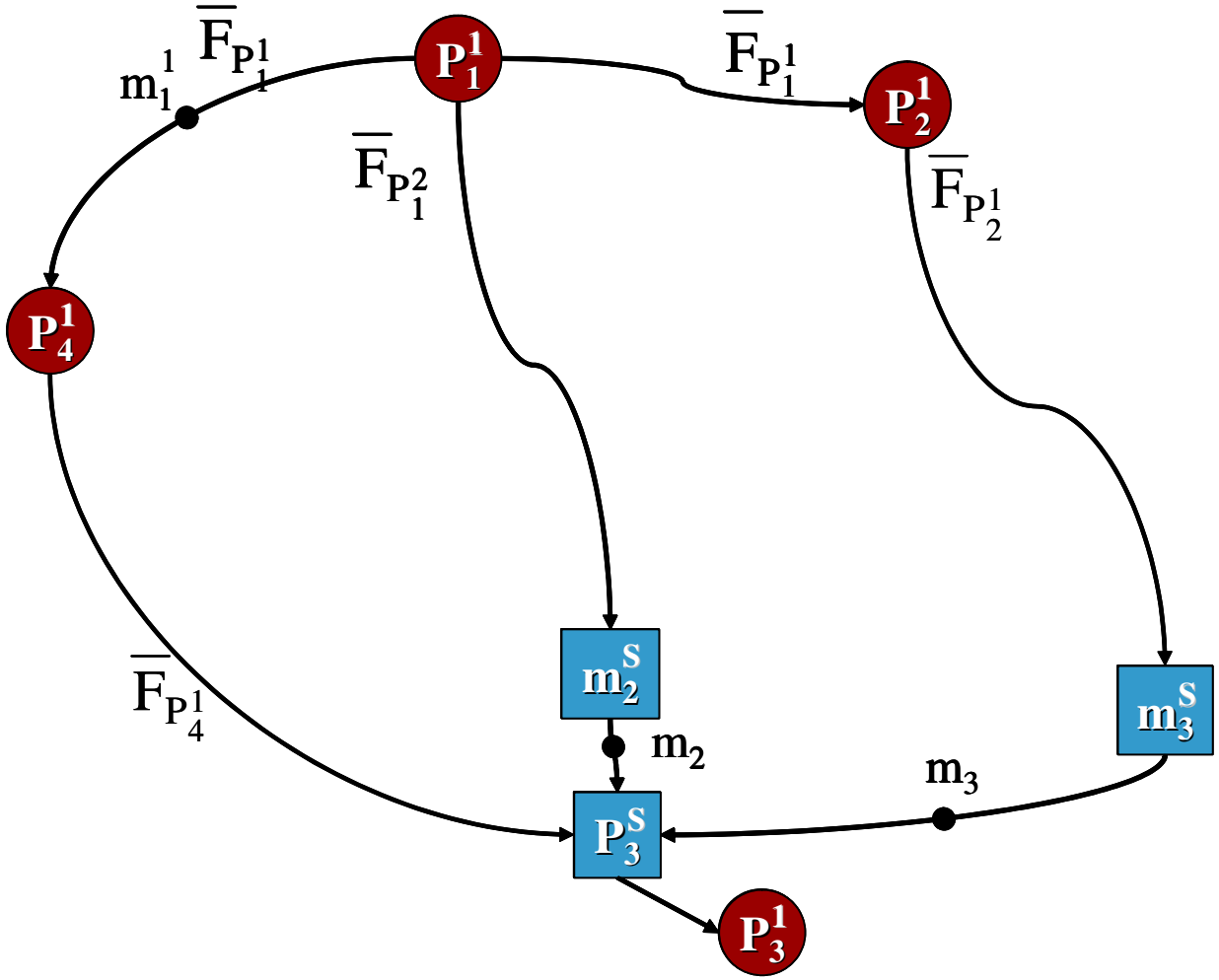
Generation of Fault-Tolerant Schedules



Generation of Fault-Tolerant Schedules

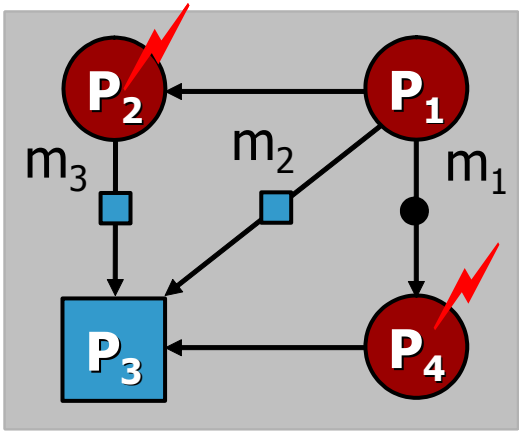
No faults Scenario

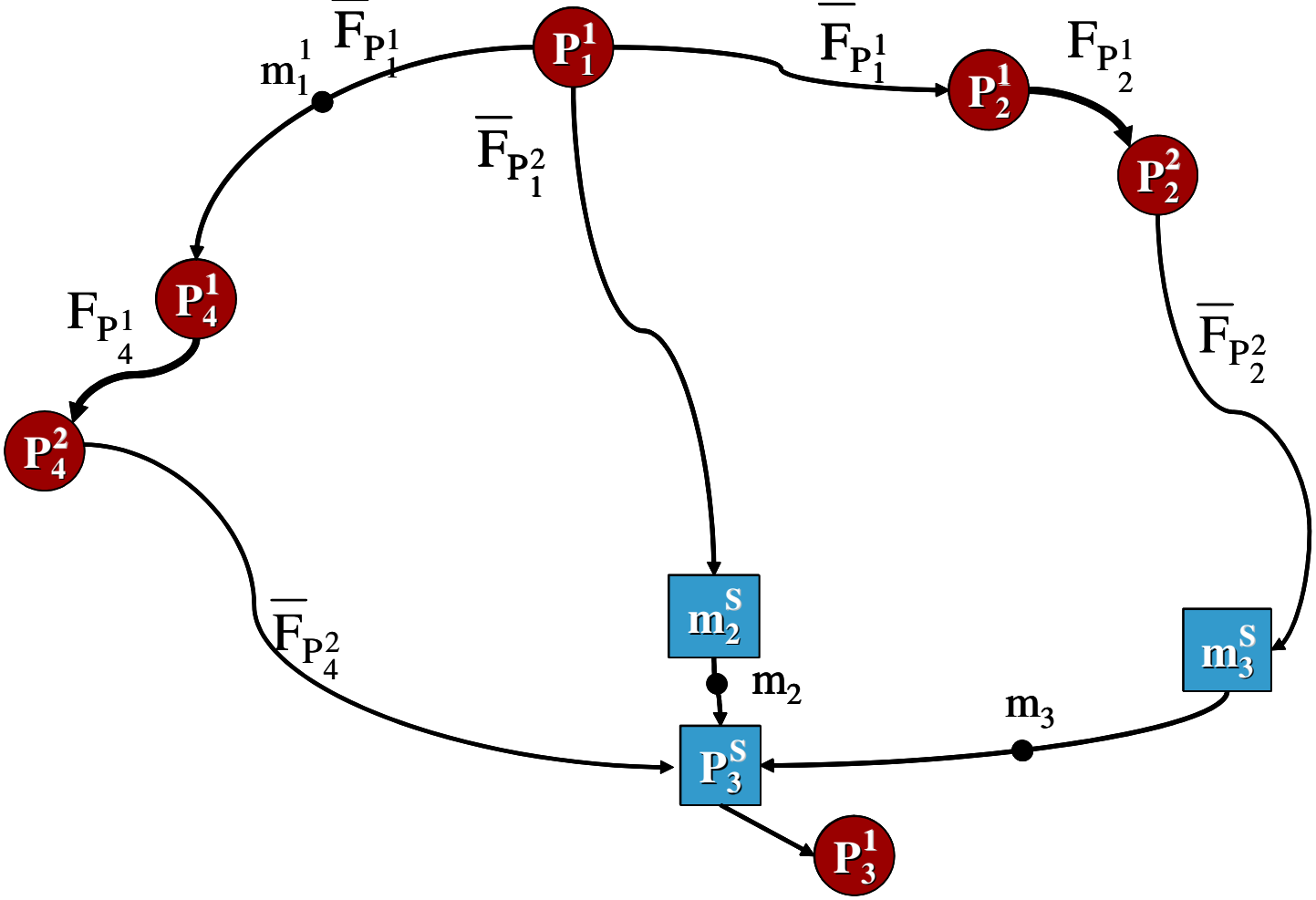




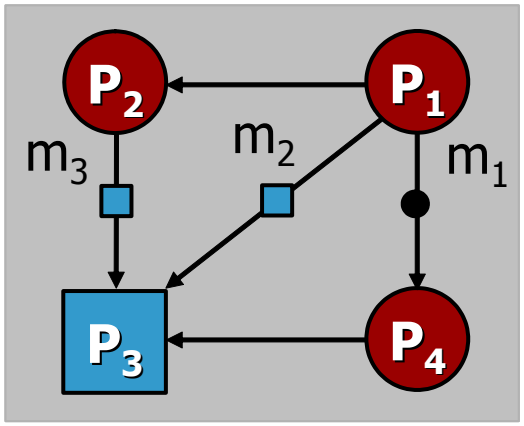
Generation of Fault-Tolerant Schedules

Fault in P_2 and P_4





Generation of Fault-Tolerant Schedules



Schedule Table

N_1	$true$	F_{P_1}	$\overline{F}_{P_1}$	$F_{P_1} \wedge F_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1} \wedge F_{P_2}$
P_1	0	35		70		
P_2			30	100	65	90
m_1			31	100	66	
m_2			105	105	105	
m_3				120		120

Schedule Table

N_1	$true$	F_{P_1}	$\overline{F}_{P_1}$	$F_{P_1} \wedge F_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1} \wedge F_{P_2}$
P_1	0	35		70		
P_2			30	100	65	90
m_1			31	100	66	
m_2			105	105	105	
m_3				120		120

Schedule Table

N_1	$true$	F_{P_1}	$\overline{F}_{P_1}$	$F_{P_1} \wedge F_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1}$	$F_{P_1} \wedge \overline{F}_{P_1} \wedge F_{P_2}$
P_1	0	35		70		
P_2			30	100	65	90
m_1			31	100	66	
m_2			105	105	105	
m_3				120		120

Generation of Fault-Tolerant Schedules

☞ Trade-offs

- Worst case schedule length

  vs

- Schedule tables size

  vs

- Transparency

  vs

- Schedule generation time

Generation of Fault-Tolerant Schedules

	60 processes		
	k=1	k=2	k=3
100%	0.71	2.09	4.35
75%	1.20	4.64	11.55
50%	1.53	7.09	18.28
25%	1.92	10.00	28.31
0%	2.16	11.72	34.62

Max. nr. of faults

Transparency

Size of schedule tables (K)

Increased transparency

Smaller schedule table

Generation of Fault-Tolerant Schedules

Transparency	60 processes		
	k=1	k=2	k=3
100%	32	58	86
75%	28	54	79
50%	19	39	58
25%	13	30	43
0%	12	24	34

Max. nr. of faults

Schedule length overhead (%)

Increased transparency

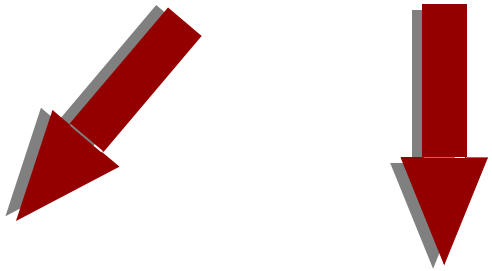
Longer schedule

Generation of Fault-Tolerant Schedules

Maintain the same ordering in all fault scenarios

Generation of Fault-Tolerant Schedules

Maintain the same ordering in all fault scenarios

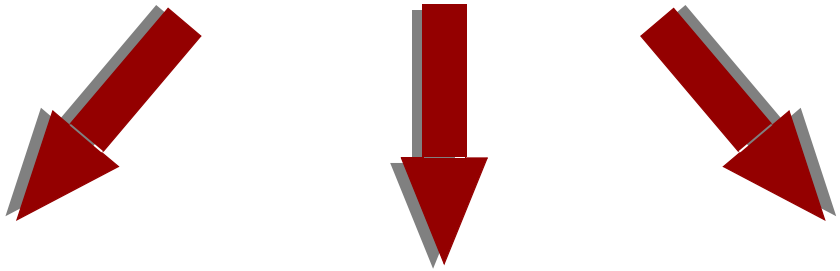


Extremely(!) small size of schedule tables

Very low schedule generation time

Generation of Fault-Tolerant Schedules

Maintain the same ordering in all fault scenarios



Extremely(!) small size of schedule tables

Very low schedule generation time

~15% longer schedules

Outline

- Overall Flow, Application and System Model
- Fault Tolerance Policy Assignment
- Transparency
- Re-execution and Checkpointing
- Generation of Fault-Tolerant Schedules
- Cross-layer Optimization with Hardening Alternatives
- Conclusions

Optimization with Hardening Alternatives

☞ Where comes that mystical *k* from?

Optimization with Hardening Alternatives

☞ **Where comes that mystical *k* from?**

- **γ: maximum probability of system failure in time unit (hour).**

  Reliability goal:

$$\rho = 1 - \gamma$$

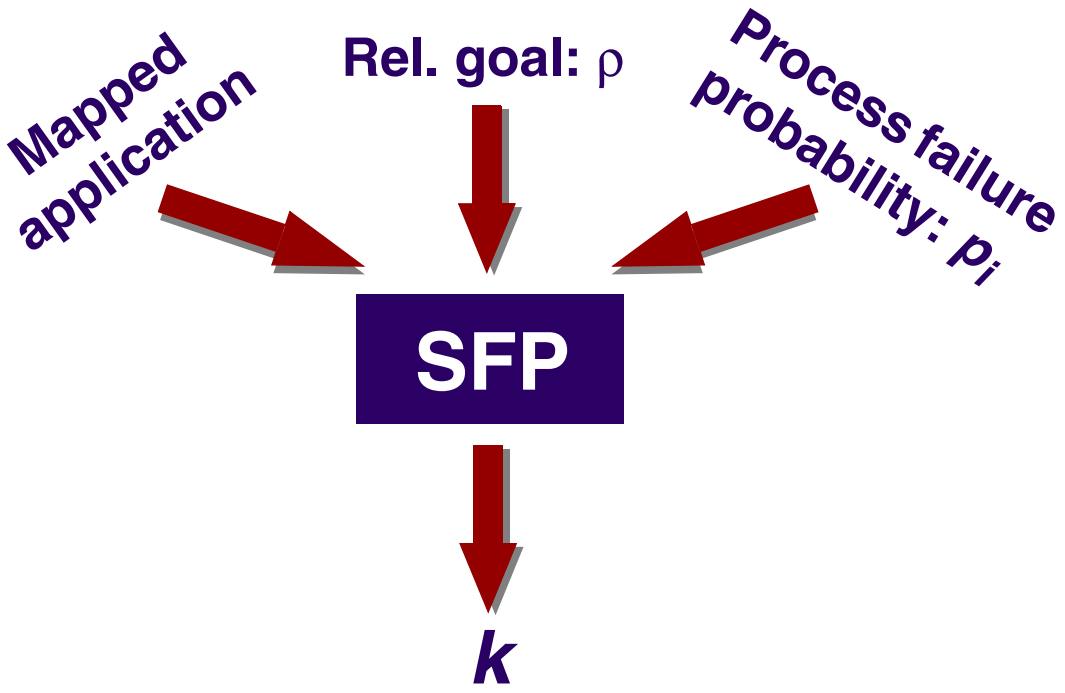
Optimization with Hardening Alternatives

☞ **Where comes that mystical *k* from?**

- **γ: maximum probability of system failure in time unit (hour).**

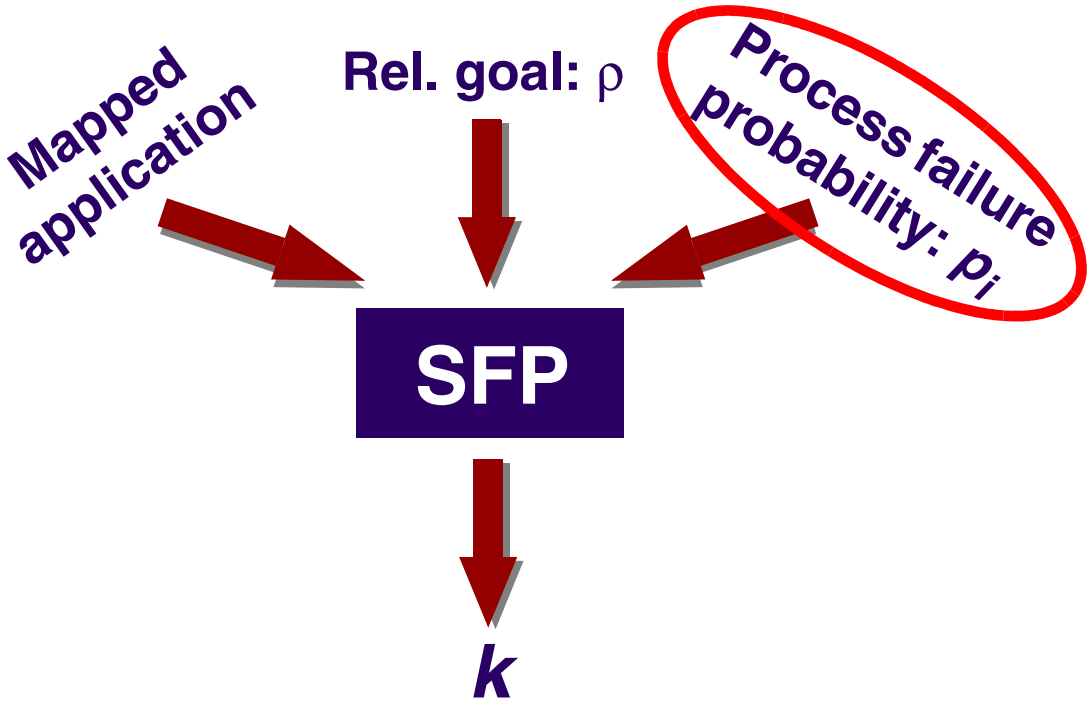
  Reliability goal:

  $\rho = 1 - \gamma$



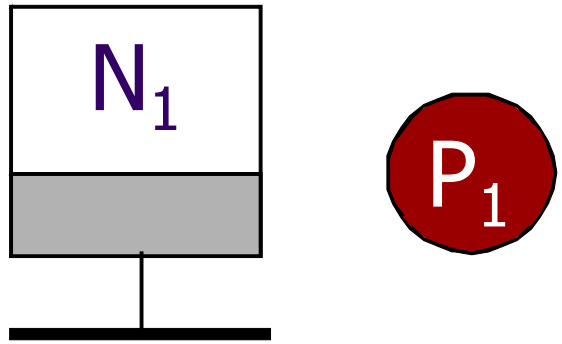
Optimization with Hardening Alternatives

☞ For processors with different *hardening level* the p_i is different.



Optimization with Hardening Alternatives

- Reliability goal: $\rho = 1 - \gamma = 1 - 10^{-5}$
- Period: T = 360 ms



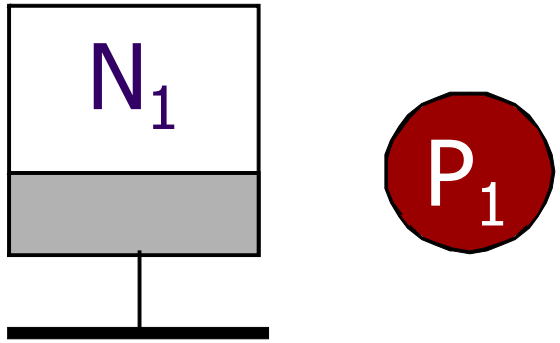
Optimization with Hardening Alternatives

Hardening versions of computation node N_1

N_1	h = 1		h = 2		h = 3	
	t	p	t	p	t	p
P_1	80	$4 \cdot 10^{-2}$	100	$4 \cdot 10^{-4}$	160	$4 \cdot 10^{-6}$
Cost	10		20		40	

Increase in reliability

Decrease in process failure probabilities



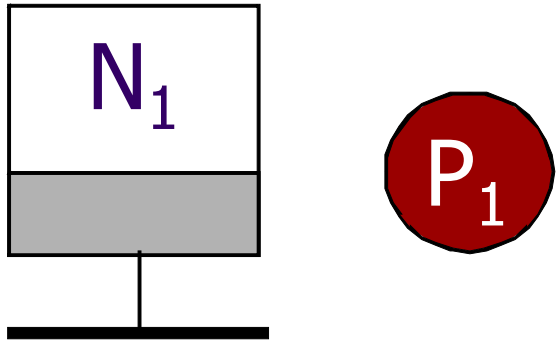
Optimization with Hardening Alternatives

Worst-case execution times are increased

Hardening performance degradation (HPD)

N_1	h = 1		h = 2		h = 3	
	t	p	t	p	t	p
P_1	80	$4 \cdot 10^{-2}$	100	$4 \cdot 10^{-4}$	160	$4 \cdot 10^{-6}$
Cost	10		20		40	

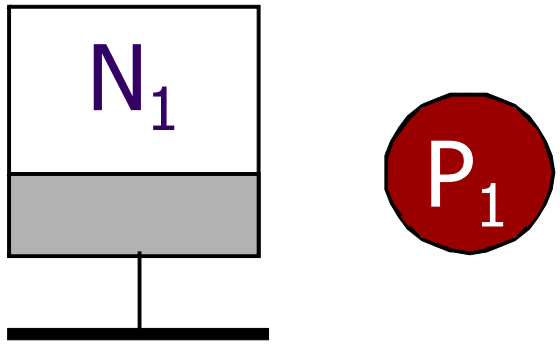
Cost is increased with more hardening!



Optimization with Hardening Alternatives

N_1	h = 1		h = 2		h = 3	
	t	p	t	p	t	p
P_1	80	$4 \cdot 10^{-2}$	100	$4 \cdot 10^{-4}$	160	$4 \cdot 10^{-6}$
Cost	10		20		40	

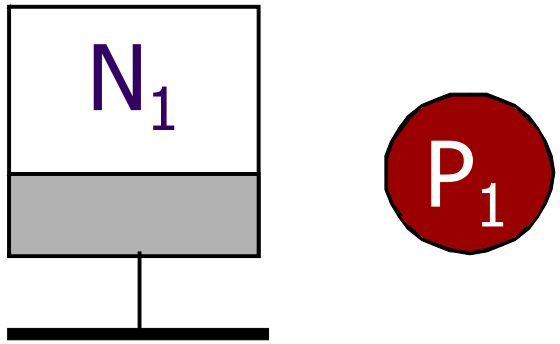
***k*=6** ***k*=2** ***k*=1**

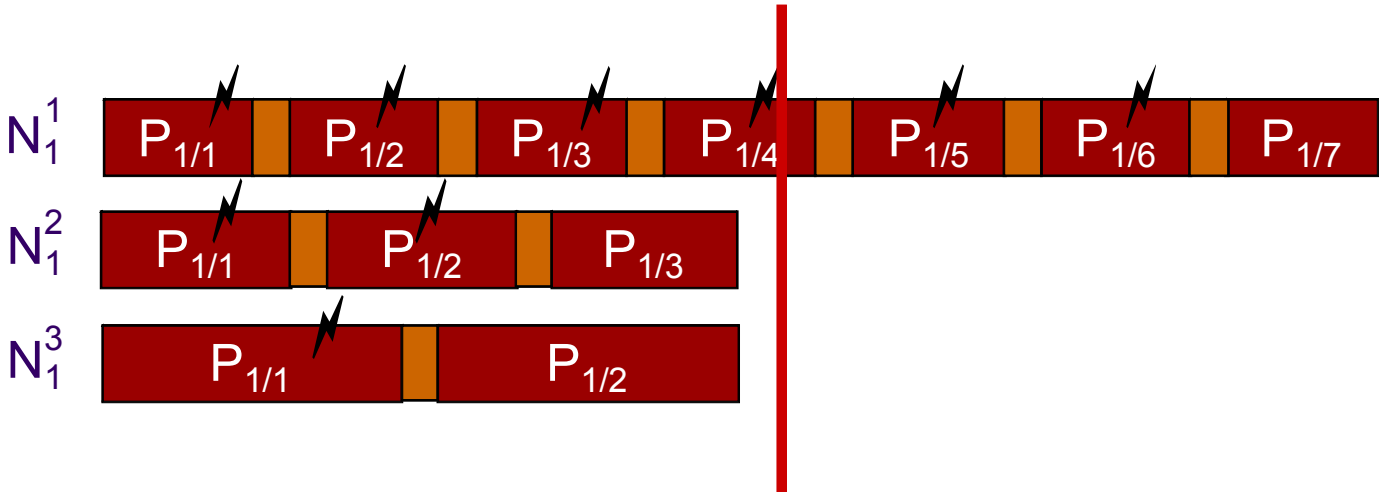


Optimization with Hardening Alternatives

N_1	h = 1		h = 2		h = 3	
	t	p	t	p	t	p
P_1	80	$4 \cdot 10^{-2}$	100	$4 \cdot 10^{-4}$	160	$4 \cdot 10^{-6}$
Cost	10		20		40	

***k*=6** ***k*=2** ***k*=1**

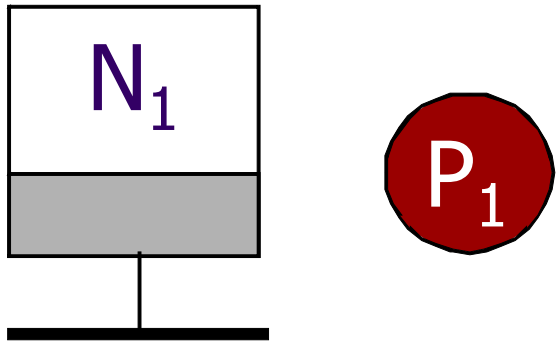


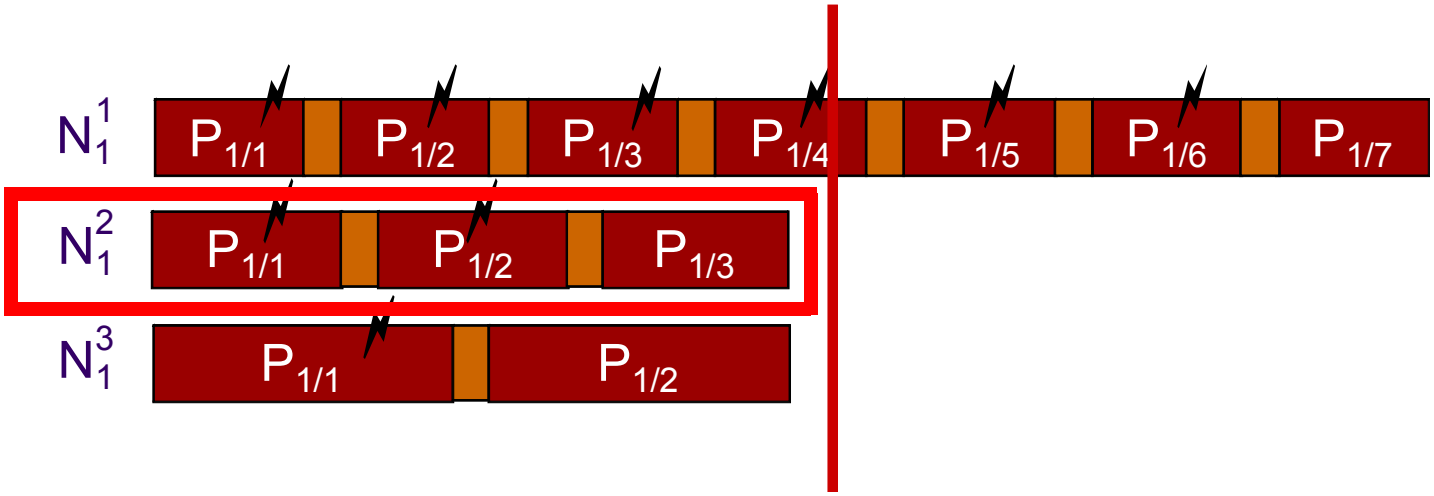


Optimization with Hardening Alternatives

N_1	h = 1		h = 2		h = 3	
	t	p	t	p	t	p
P_1	80	$4 \cdot 10^{-2}$	100	$4 \cdot 10^{-4}$	160	$4 \cdot 10^{-6}$
Cost	10		20		40	

***k*=6** ***k*=2** ***k*=1**



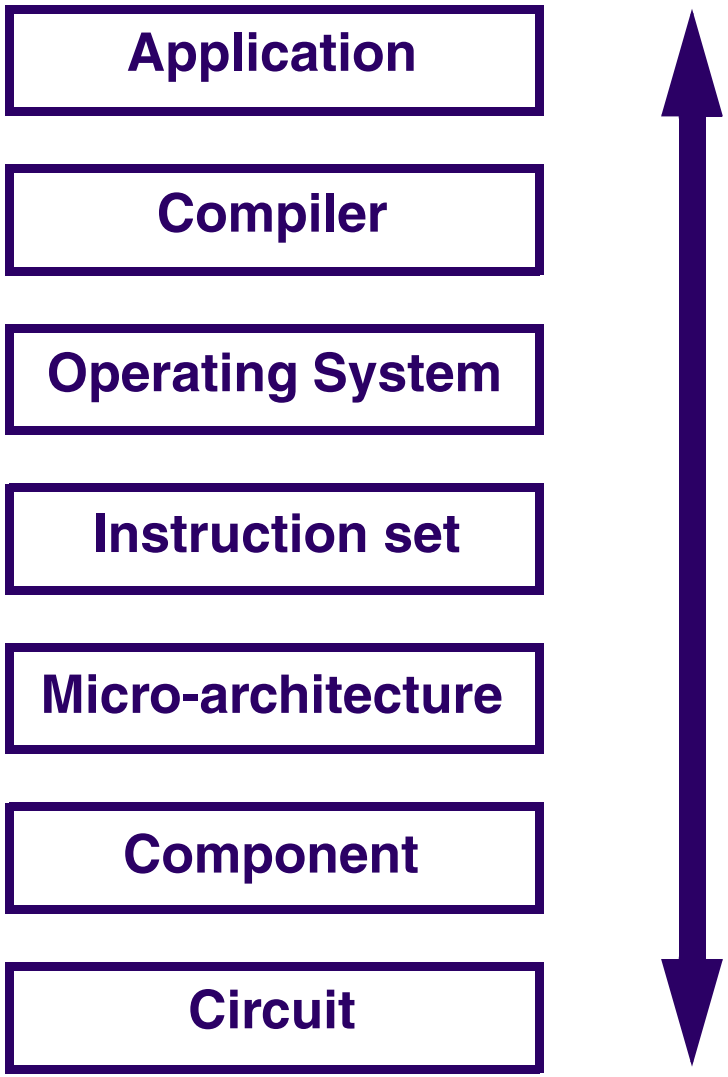


Optimization with Hardening Alternatives

The Problem:

- **Find the *hardening level of each node*, the mapping, fault tolerance policy assignment and number of checkpoints for each process, and generate a system schedule, such that the deadlines, *reliability goal*, and *cost constraint* are satisfied.**

Cross-layer Fault Tolerance Optimization



☞ Efficient fault-tolerance needs a cross-layer approach:

- All layers are involved
- Application specific cross-layer trade-offs

Conclusions

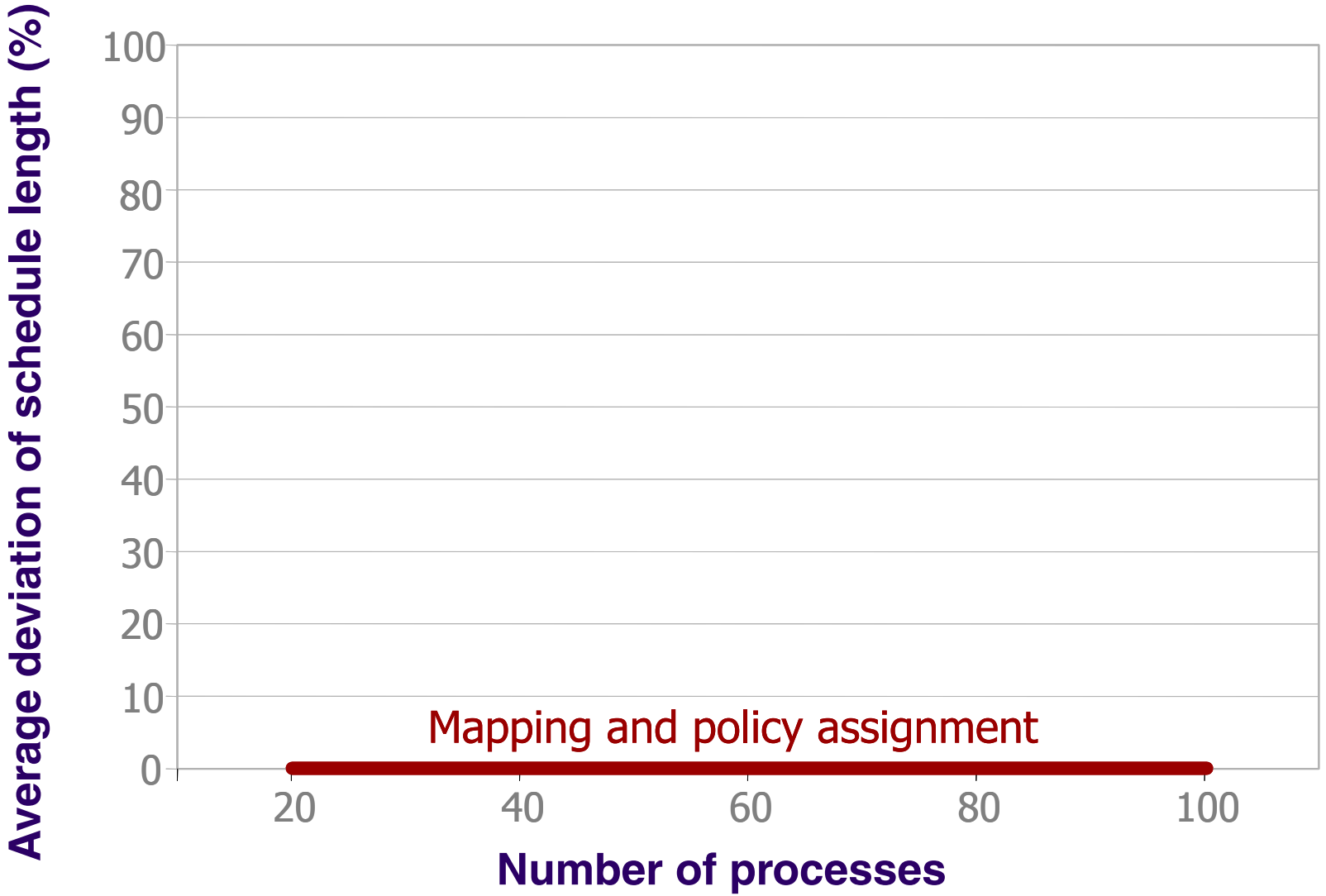
- Design optimization of fault tolerant real-time systems:
  - Fault Tolerance Policy assignment
  - Transparency
  - Checkpoint optimization
  - Schedule generation
  - Optimization with hardening alternatives (cross-layer);

Conclusions

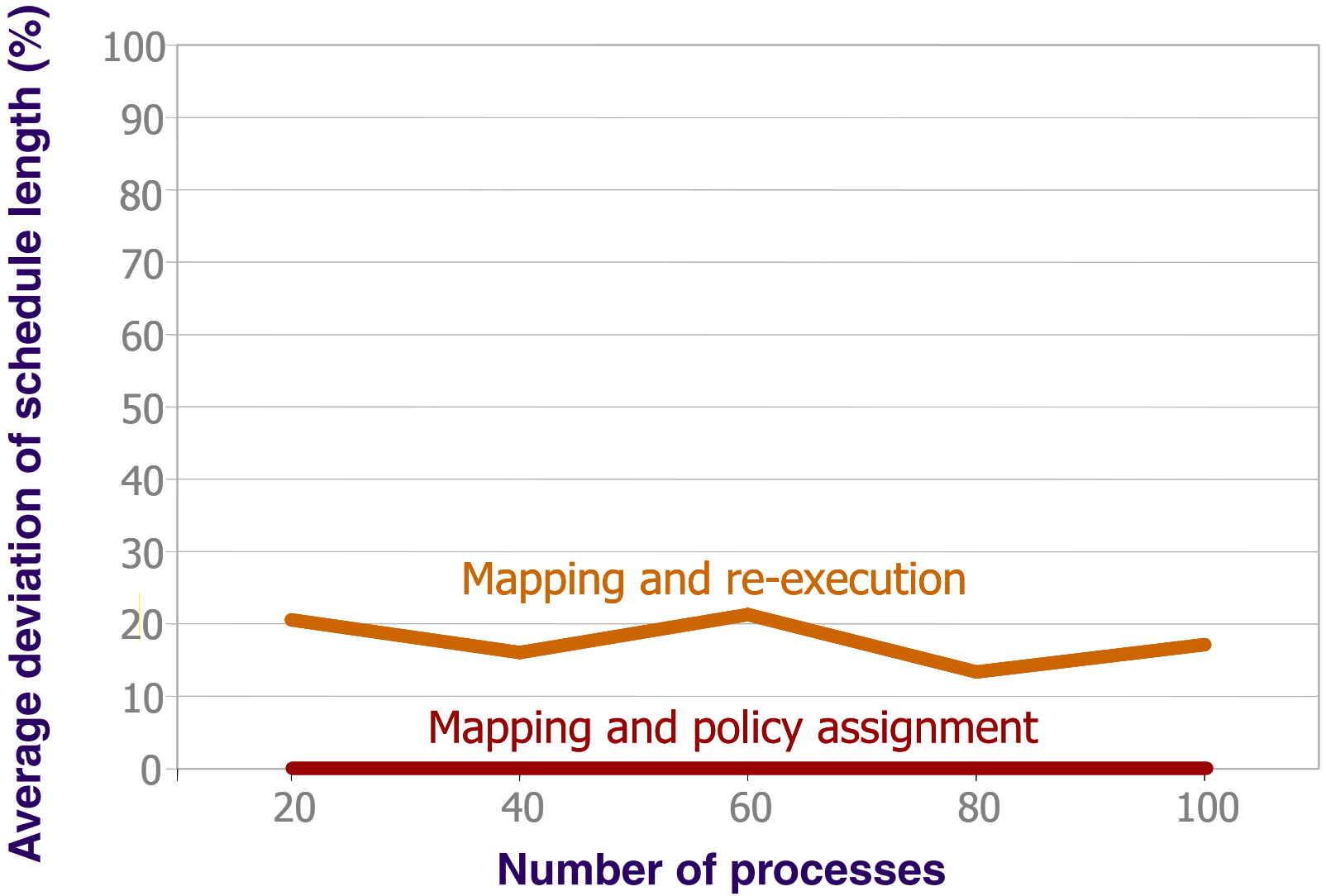
- Design optimization of fault tolerant real-time systems:
  - Fault Tolerance Policy assignment
  - Transparency
  - Checkpoint optimization
  - Schedule generation
  - Optimization with hardening alternatives

Efficient system-level design techniques can help to meet real-time and fault tolerance requirements in the context of limited amount of resources.

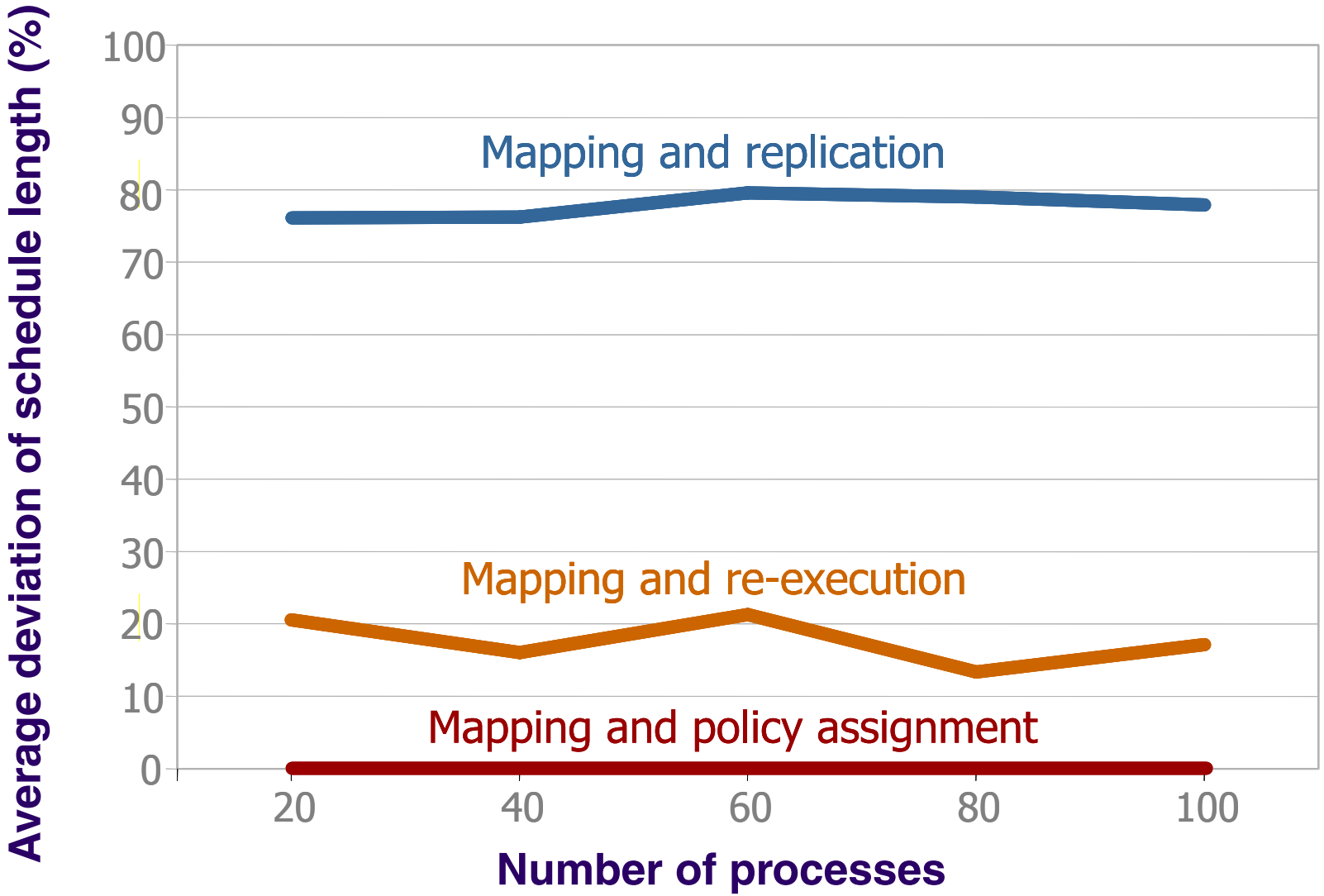
Fault Tolerance Policy Assignment



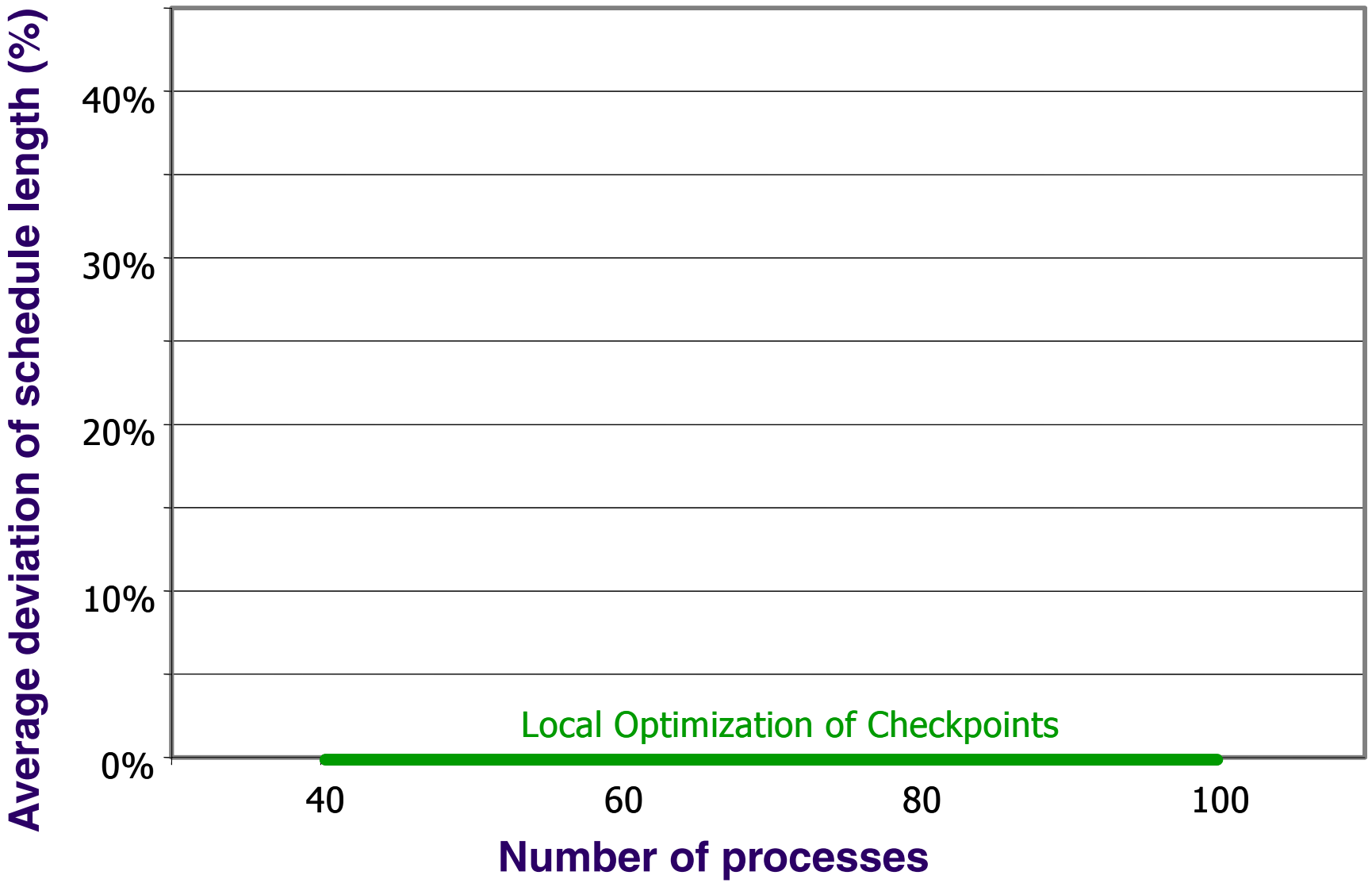
Fault Tolerance Policy Assignment



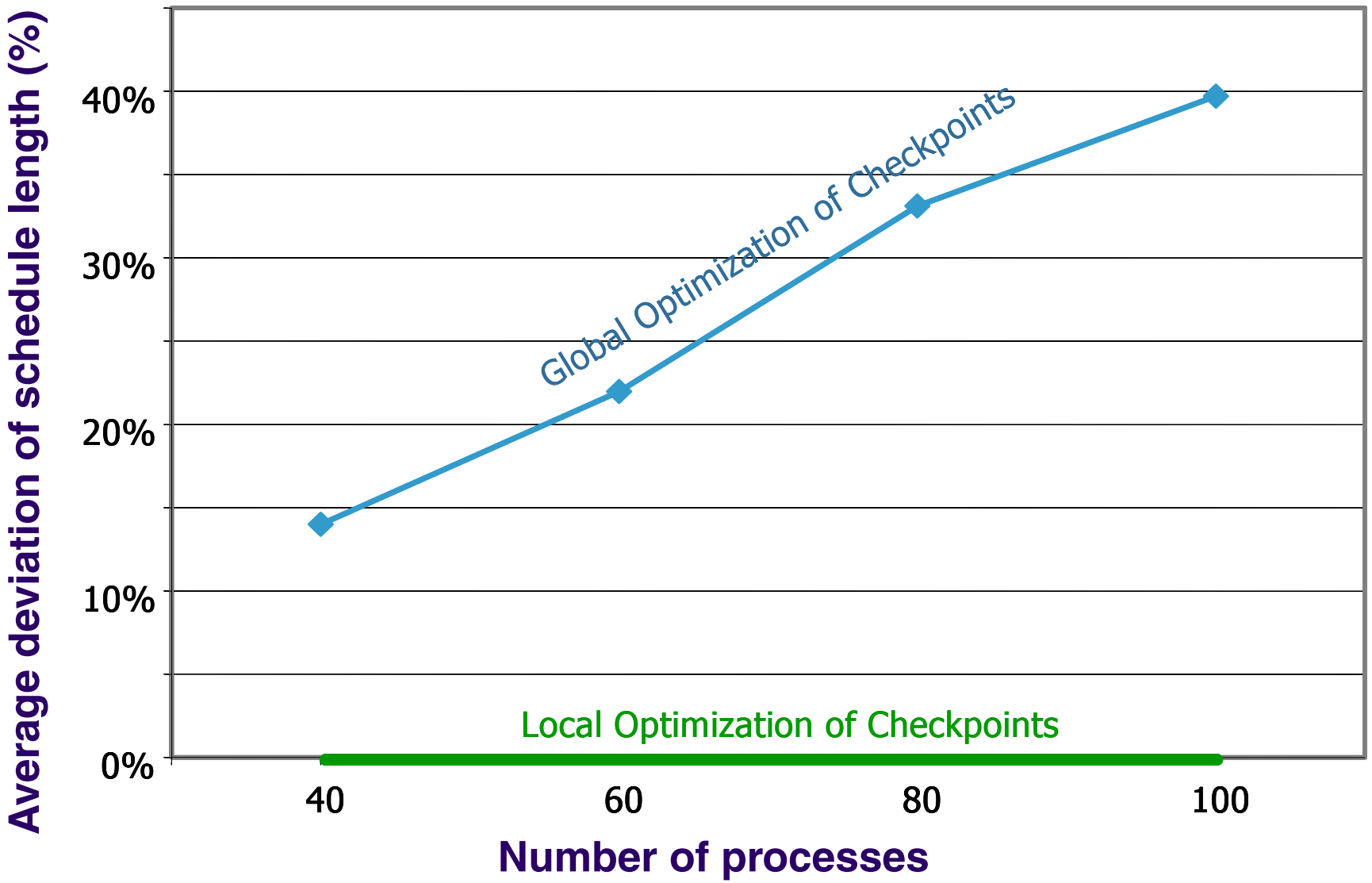
Fault Tolerance Policy Assignment



Checkpoint Optimization



Checkpoint Optimization



Conclusions

- Design optimization of fault tolerant real-time systems.
  - Fault Tolerance Policy assignment
  - Transparency
  - Checkpoint optimization
  - Schedule generation

Conclusions

- Design optimization of fault tolerant real-time systems.
  - Fault Tolerance Policy assignment
  - Transparency
  - Checkpoint optimization
  - Schedule generation

Efficient system-level design techniques can help to meet real-time and fault tolerance requirements in the context of limited amount of resources.