On Resilient Task Allocation and Scheduling with Uncertain Quality Checkers

Qian Zhang, Ting Wang and Qiang Xu

Department of Computer Science & Engineering
The Chinese University of Hong Kong

CUhk REliable Computing Laboratory

Outline

- Background
- Preliminaries
- The proposed methodology
  - Probability of quality satisfaction
  - Online scheduler
- Experiments

Emerging Error-Resilient Applications

Image Processing

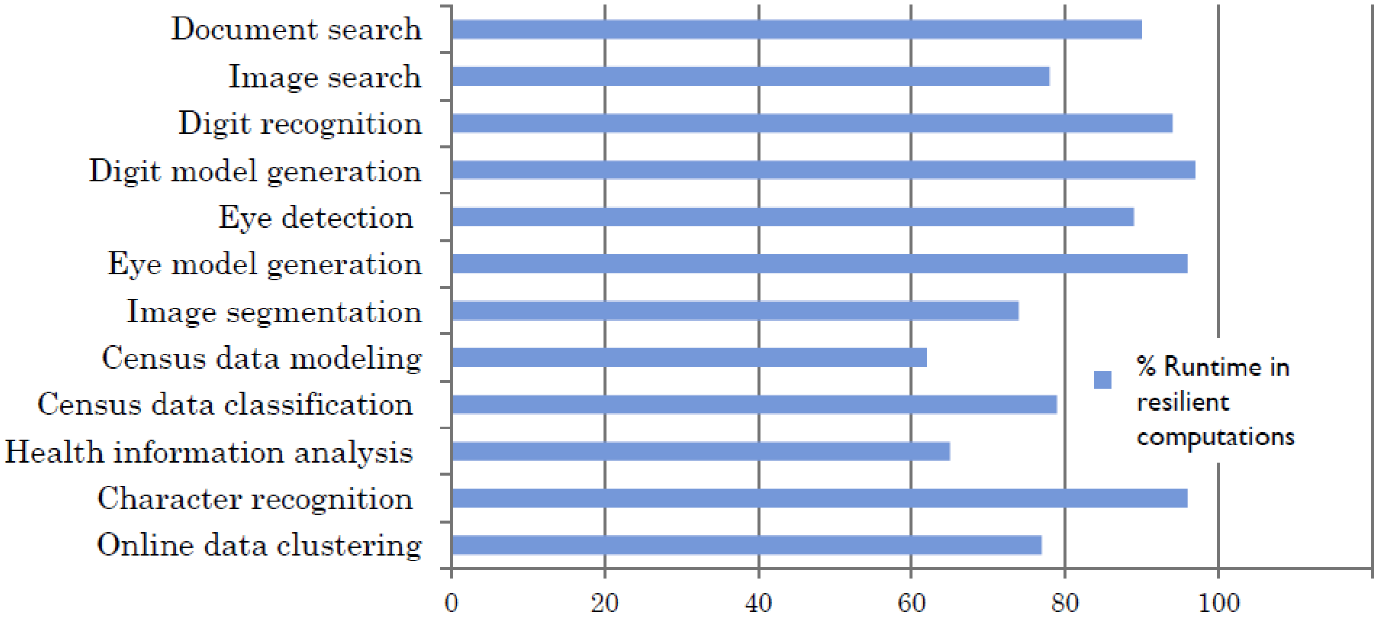
Data Mining

- Noisy input
- Stochastic Processing
- “Acceptable” instead of precise output

Robotics

Multi-Media

Emerging Error-Resilient Applications



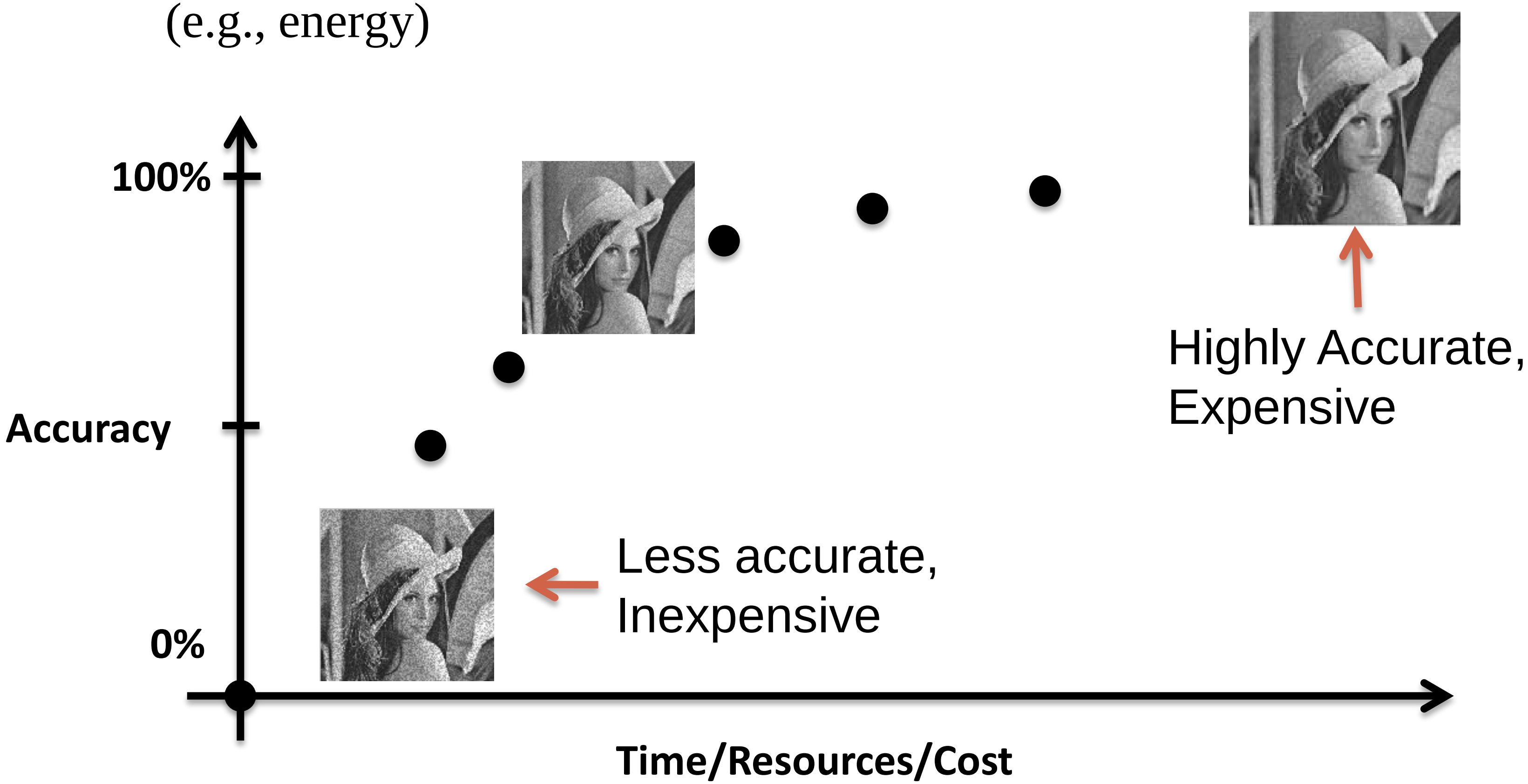
Applications have a mix of resilient and sensitive computations

83% of runtime spent in computations can be approximated

V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," DAC 2013.

What is Approximate Computing?

- Approximate computing
  - A technique to tradeoff computation quality and computational effort (e.g., energy)



Approximate Computing

Key idea: Trade off computation quality and energy consumption (Unreliable hardware units may produce incorrect results with much lower power.)

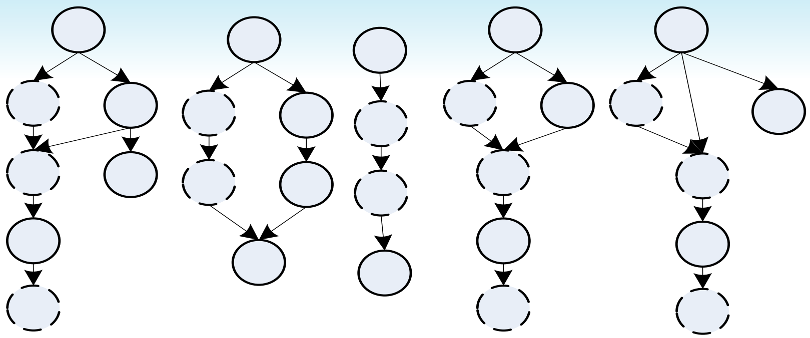
Approximate computing in software	Approximate architecture &system design	Approximate circuit design

- Voltage Over-Scaling
  - Circuits work below the nominal voltage for energy reduction
    - Error vs. Energy
- Resilience-Aware Scheduling
  - Not well explored
  - *ApproxMap*
    - J. Yi et al. “Approxmap: On task allocation and scheduling for resilient applications,” ASPDAC, pp. 1–6, IEEE, 2016.

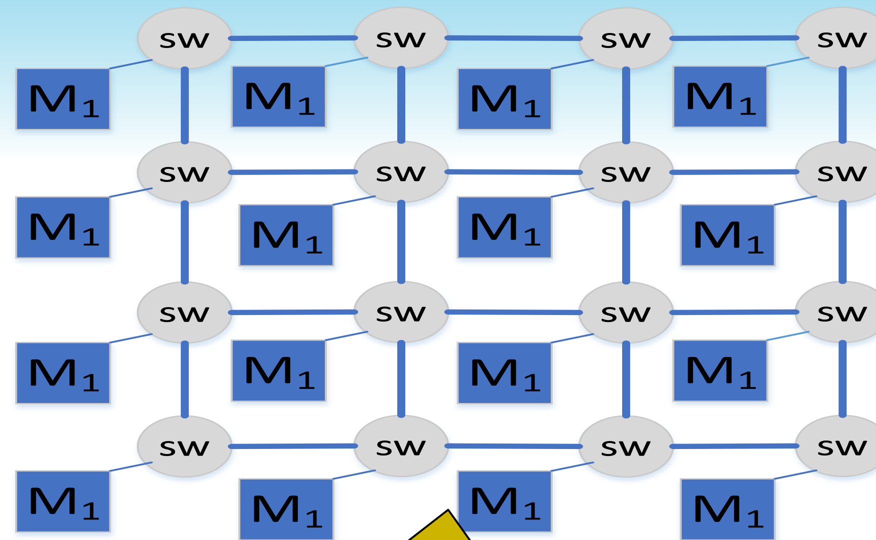
Outline

- Background
- Preliminaries
  - *ApproxMap*: Resilience-aware scheduling
- The proposed methodology
  - Probability of quality satisfaction
  - Online scheduler
- Experiments

ApproxMap: Resilience-Aware Scheduling on Multicore Platforms



Resilient applications



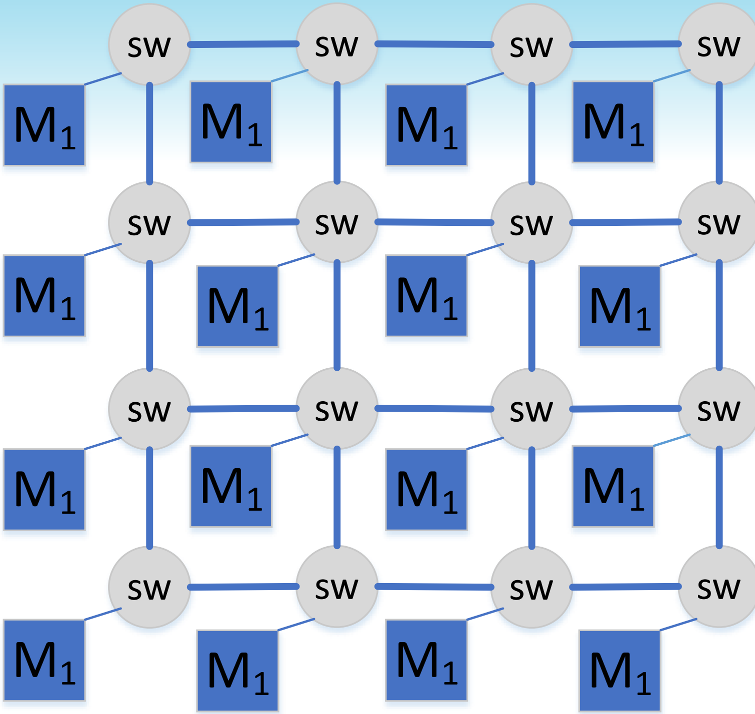
Mapping and scheduling

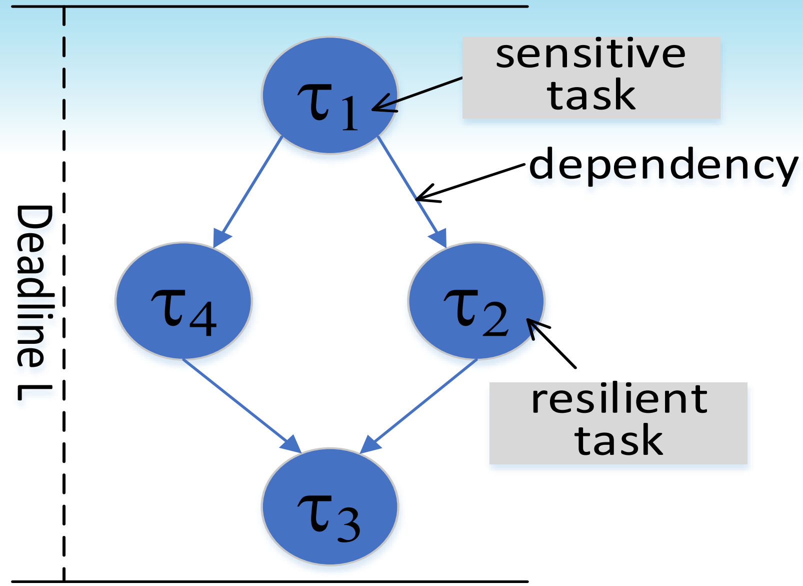
How to treat error-resilient tasks and error-sensitive tasks differently for energy gains ?

How to ensure the target quality requirement, and to meet the application performance requirement ?

Note: Here we assume the processor cores are architecturally identical and the only source of heterogeneity is their operating voltage levels.

Architecture & Application Model



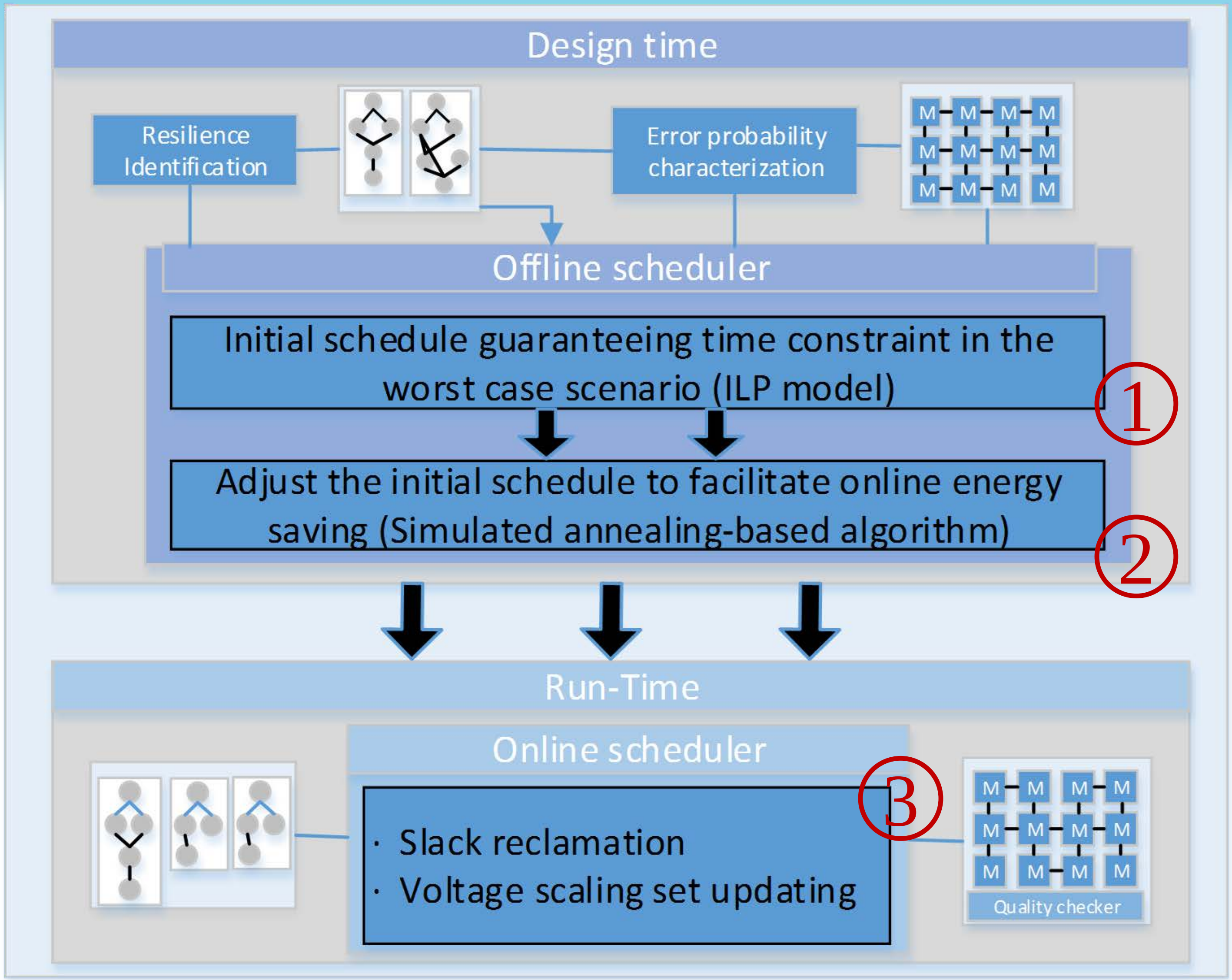


Data flow graph G=<V, E, R>

Operating voltage $V = \{V_1, V_2, \cdots, V_K\}$, where $V_1 < V_2 < \cdots < V_K$.

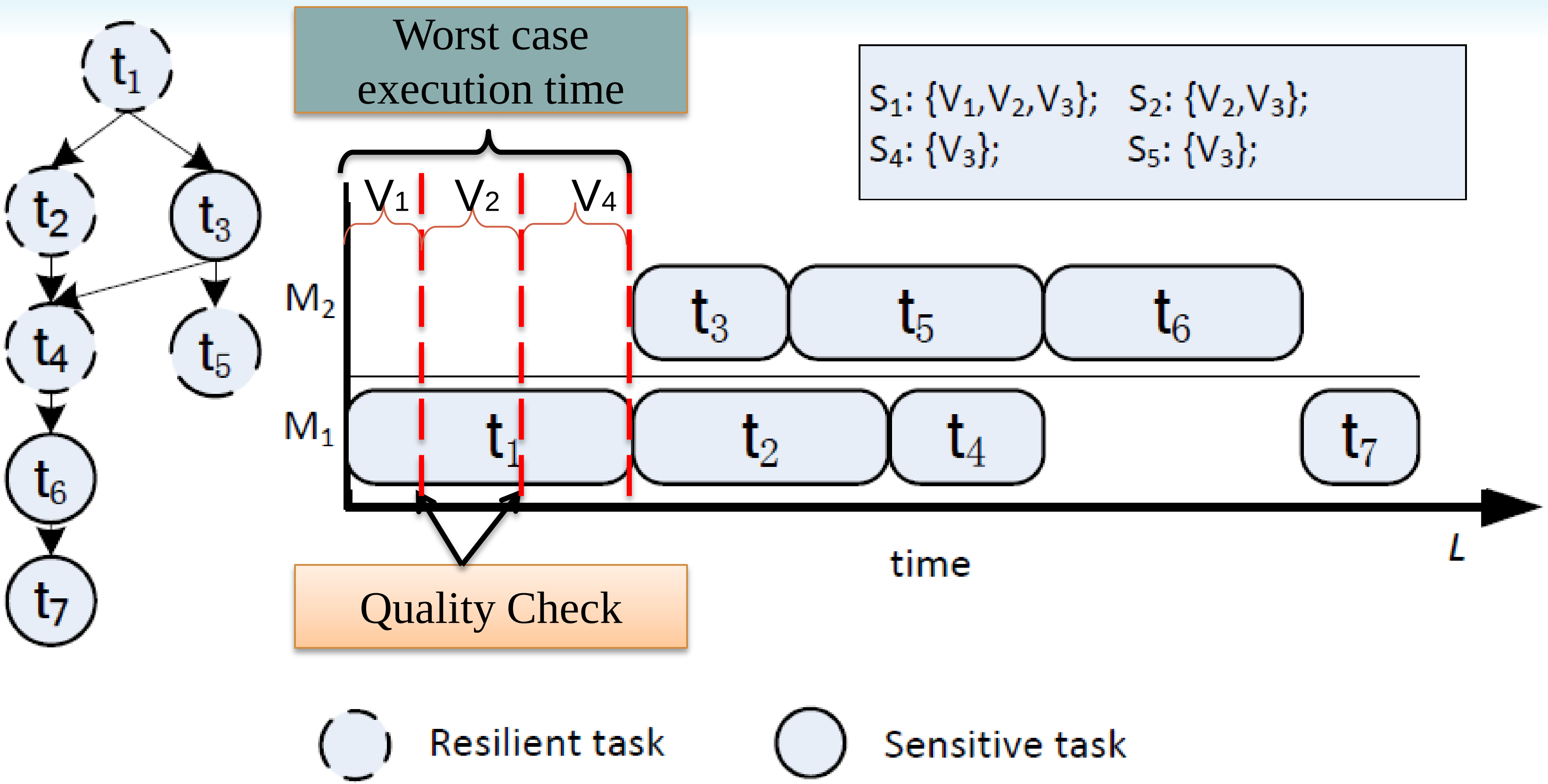
V_K is the nominal voltage, while the other voltage level could potentially impact the correctness of the computation.

ApproxMap



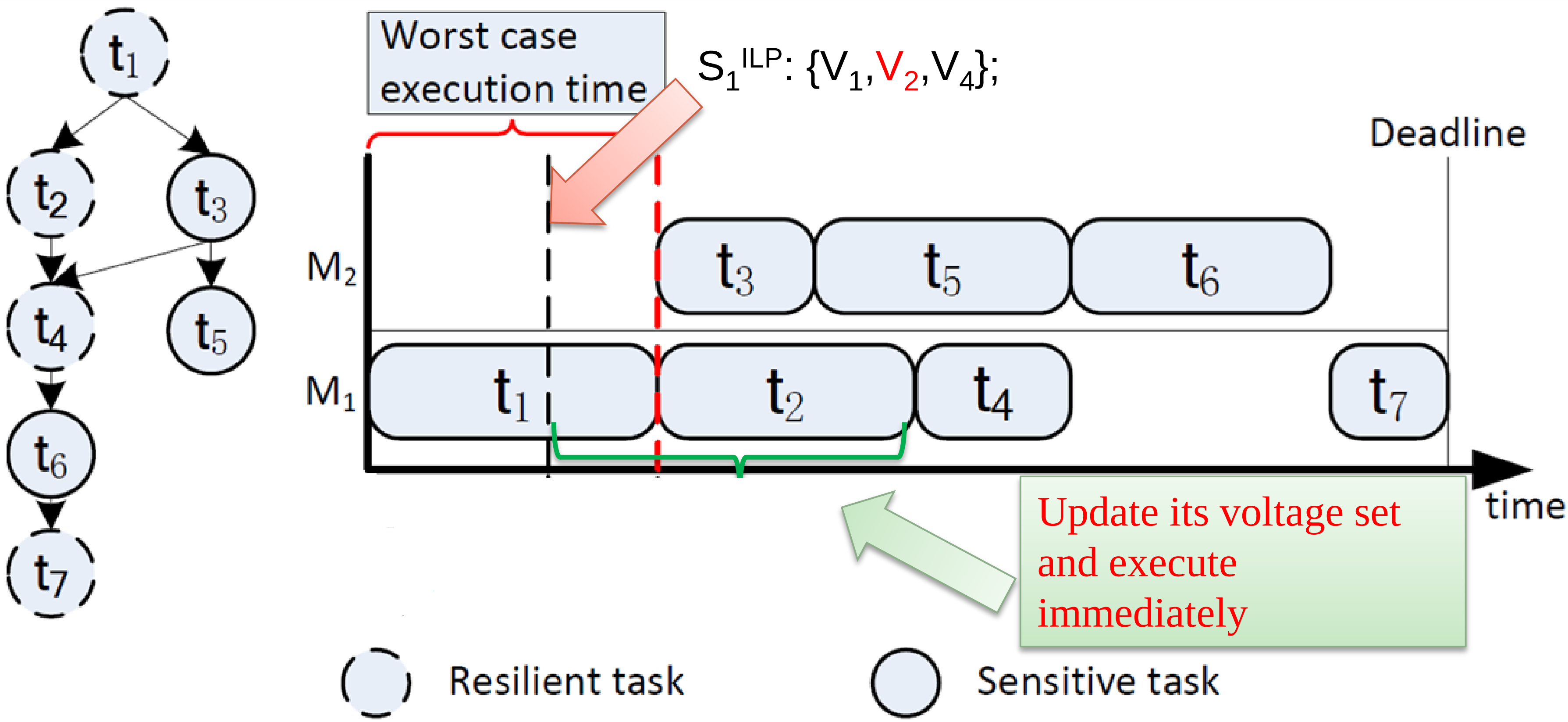
Design time
Resilience Identification
Error probability characterization
M
Offline scheduler
Initial schedule guaranteeing time constraint in the worst case scenario (ILP model)
1
Adjust the initial schedule to facilitate online energy saving (Simulated annealing-based algorithm)
2
Run-Time
Online scheduler
3
· Slack reclamation
· Voltage scaling set updating
Quality checker

ApproxMap: Offline Schedule



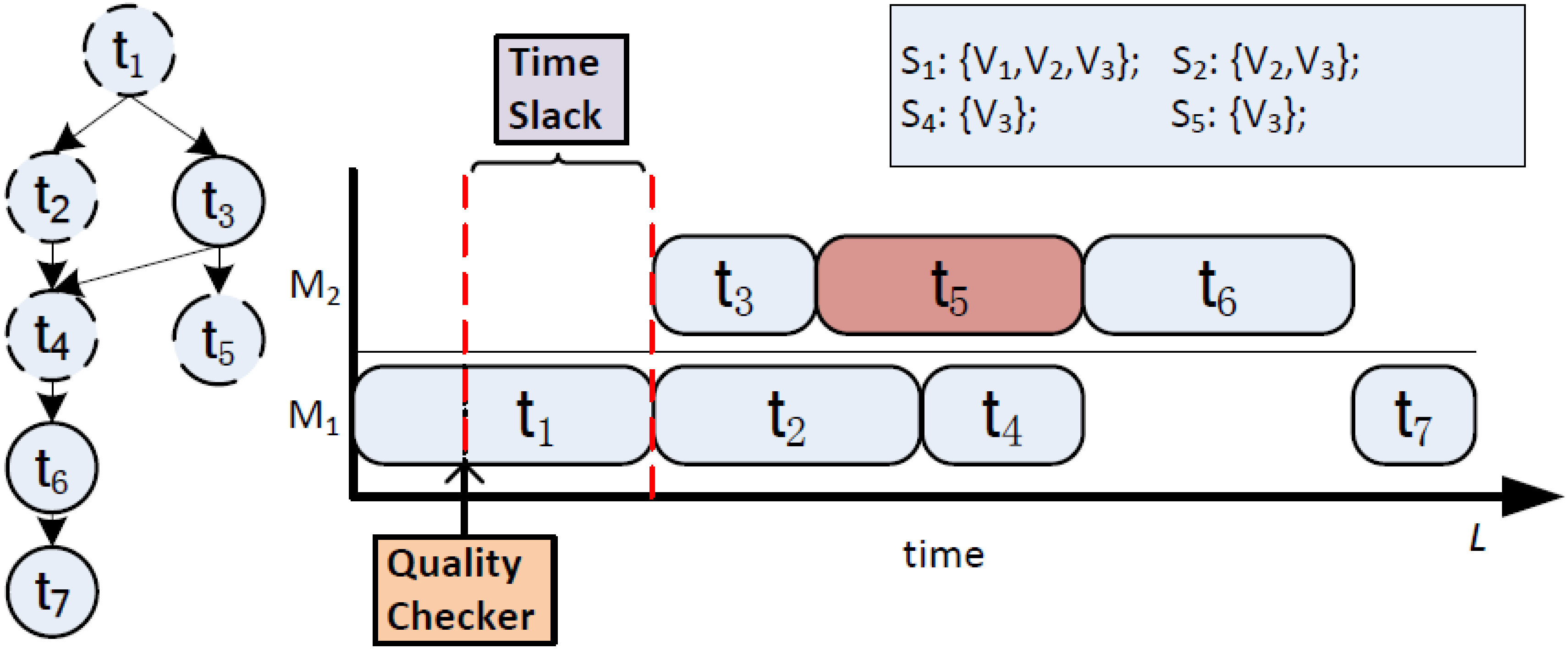
Slack window: resilient task complete before the worst case execution time

ApproxMap: Online Adjustment



ApproxMap: Unsolved Issue I

- Runtime Quality Satisfaction Issue



Quality checker is unreliable!
(it is usually trained by a learning model and predict quality violation with *p*% accuracy.)

Quality Satisfaction

- Runtime Quality Satisfaction Issue
  - Quality satisfaction

$$E < TH$$

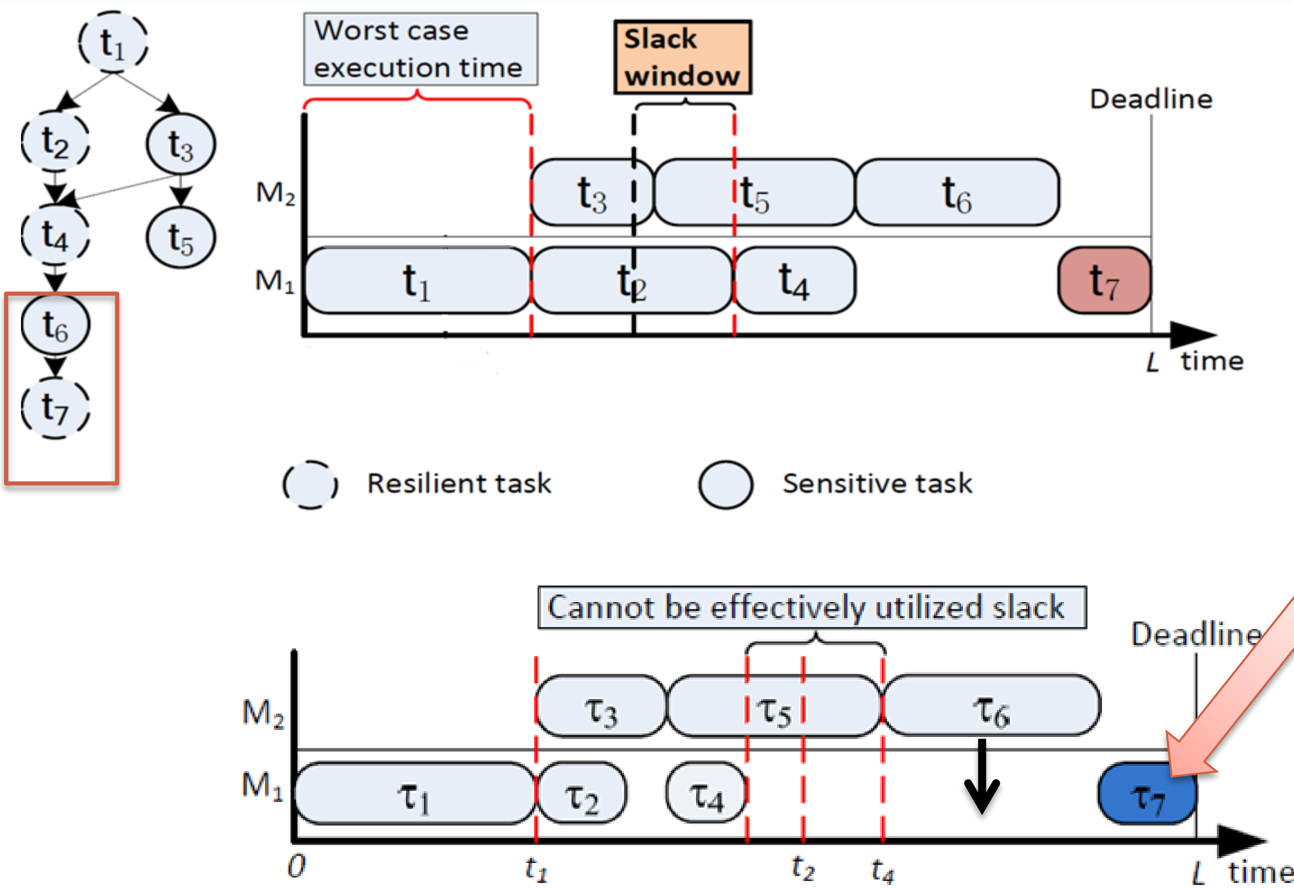
$$\Downarrow$$

  - Probability of quality satisfaction

$$\max \; Prob.(E < TH)$$

ApproxMap: Unsolved Issue II

- Detailed task-core adjustment & voltage set adjustment
  - single-core adjustment may cause the online time slack unusable.



t_7 cannot utilize any slack at runtime, because it has to wait for t_6, which cannot finish earlier on M2.

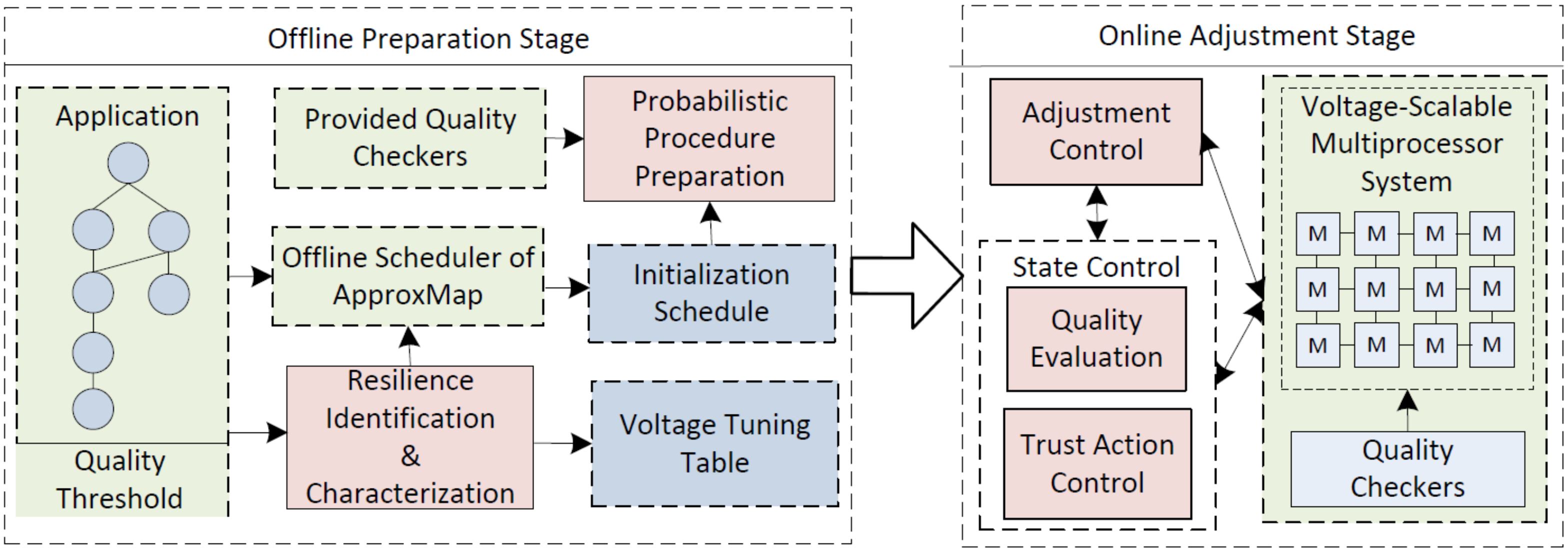
However, if assign task t_6 on M1 instead of M2, this problem can be solved.

Outline

- Background
- Preliminaries
- The proposed methodology
  - Probability of quality satisfaction
  - Online adjustment
- Experiments

The Proposed Methodology

- Selectively trust each intermediate checking result based on a probability procedure and the runtime situation to *maximize the probability of quality satisfaction*;
- Characterize *voltage tuning table* for each resilient task under different voltage levels by jointly considering computation quality and energy consumption;
- Enable *multi-core* resilient task *adjustment*.



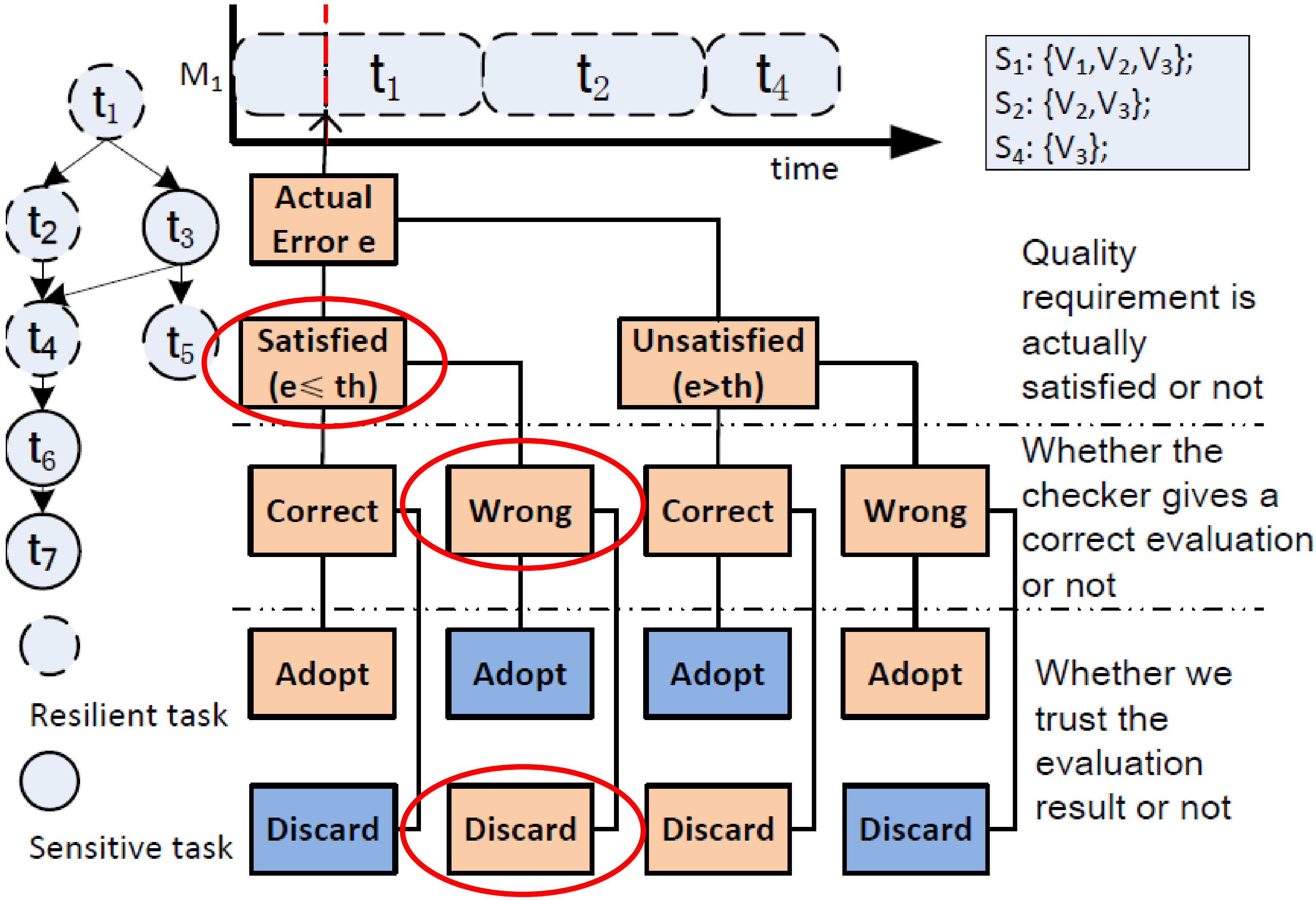
Probability of Quality Satisfaction

- Assume
  - The probability of a quality checker can give a correct evaluation is p
  - The probability of we believing such a evaluation is q
- Problem
  - Find such a value/distribution of q that satisfies

$$\max \ Prob.(E < TH)$$

State Transfer

- Given the initial schedule in design time, we say the system is in different states if it is running different tasks.



State Transfer

- Probability of state transfer
  - i.e., the probability of finishing current resilient task t

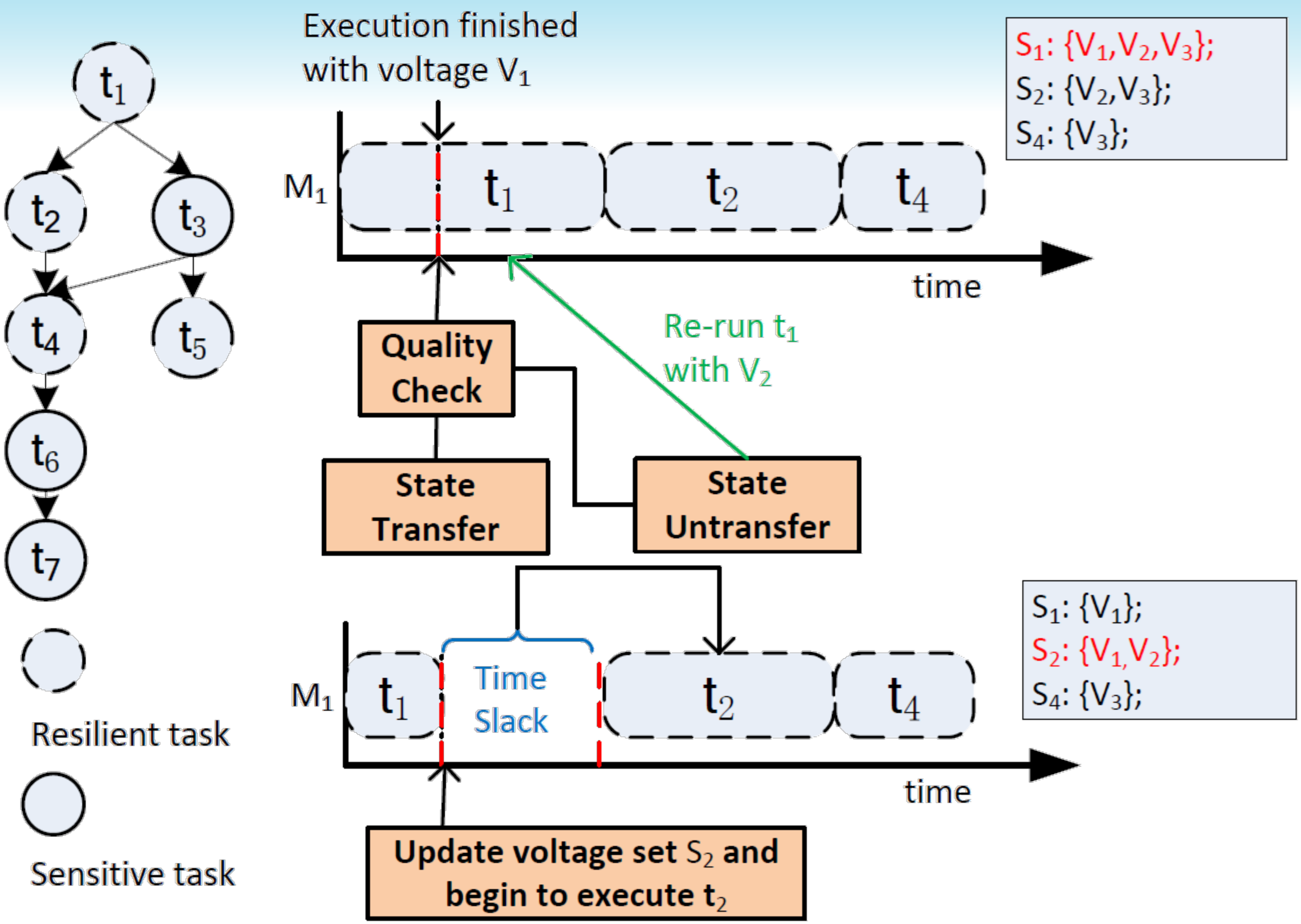
$$
\begin{aligned}
P(Orange) = {} & P(e \le th)P(predict_{correct})P(adopt) + \\
& P\,(e \le th)P\big(predict_{wrong}\big)P(discard) + \\
& P\,(e > th)P(predict_{correct})P(adopt) + \\
& P\,(e > th)P\big(predict_{wrong}\big)P(discard) +
\end{aligned}
$$

- Problem

$$\max P(E < TH)\ \mathrm{w.r.t.\ p, q, th}$$

  - Then we can guarantee the computation quality with maximum probability by selectively believing the checking results based on q and runtime situation.

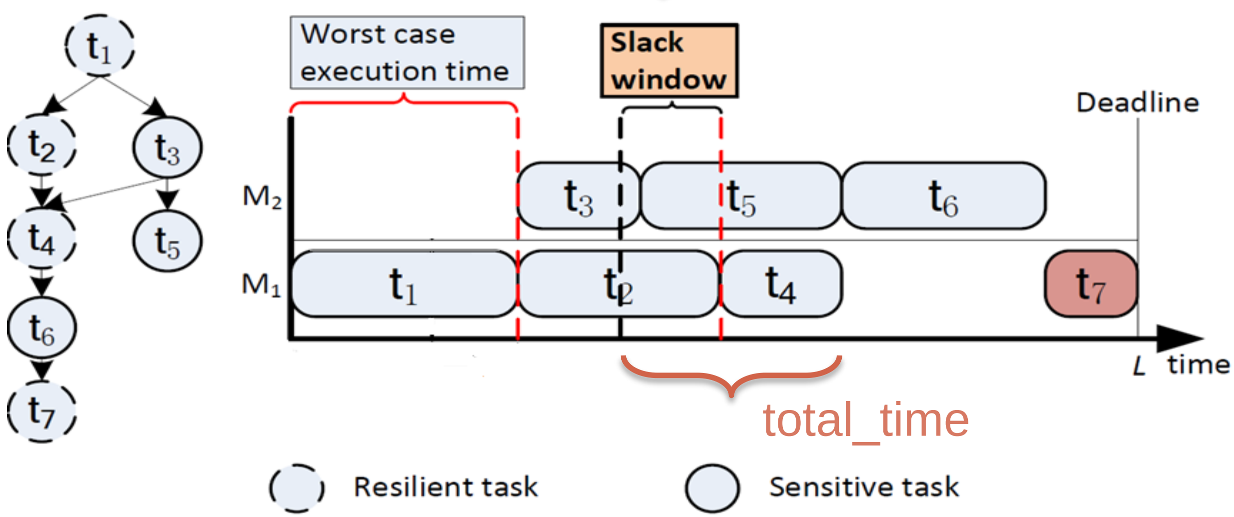
Online Execution



Online scheduler

- Update voltage scaling set for task t_i in PEST (potential energy saving tasks):

$$slack = start_{\mathrm{i}} - time_{\mathrm{current}}$$
$$total\_time = |S_i| + slack$$
$$S_i = updateS(t_i, total\_time)$$



Online scheduler

- Update voltage scaling set for task t_i in PEST (potential energy saving tasks):

$$slack = start_{\mathrm{i}} - time_{\mathrm{current}}$$
$$total\_time = |S_i| + slack$$
$$S_i = updateS(t_i, total\_time)$$

- updateS(task, available_time)
  - Heuristic
    - Sorting the voltage levels by $\frac{potential\ energy\ efficiency}{quality\ degradation}$
    - Update S_i by selecting voltages according to total $available\_time$

V1	Time 1
V2	Time 2
⋮	⋮
Vk	Time k

Outline

- Background
- Preliminaries and problem definition
- The proposed methodology
  - Probability of quality satisfaction
  - Online scheduler
- Experiments

Experimental Setup

- Initial schedule from ApproxMap
  - Gurobi 5.60 with CVX 2.1 in Matlab
- Representative task graphs
  - TGFF 3.5
- Voltage scalable system with 4 processors, and each processor has four operation voltages (1.69 V, 1.46 V, 1.38 V, 1.32 V)
- Variation of datasets
  - Take the mean value over 1000 runs for the same task graph

Comparison with Baseline

- For a given quality requirement, increasing the portion of resilient tasks can bring benefits on energy savings.
- For each case, lowering quality requirement benefits to energy efficiency.

Application	Fraction	Baseline	Ours(TH=5%)		Ours (TH=10%)	
			mean	E.B.%	mean	E.B.%
creds1_0	30%	1.85	1.64	11.24%	1.57	15.08%
	50%	1.85	1.59	14.23%	1.38	25.36%
	70%	1.85	1.22	33.91%	0.99	46.77%
kbasic_task	30%	2.14	1.88	11.98%	1.81	15.42%
	50%	2.14	1.66	22.36%	1.48	31.07%
	70%	2.14	1.36	36.33%	1.12	47.64%
kbasic_parallel_xover	30%	2.42	2.24	7.25%	2.13	11.85%
	50%	2.42	2.14	11.69%	1.90	21.54%
	70%	2.42	2.04	15.68%	1.76	27.17%
kseries_parallel	30%	3.66	3.37	7.97%	3.26	10.92%
	50%	3.66	3.24	11.48%	2.89	20.84%
	70%	3.66	3.16	13.74%	2.64	27.82%

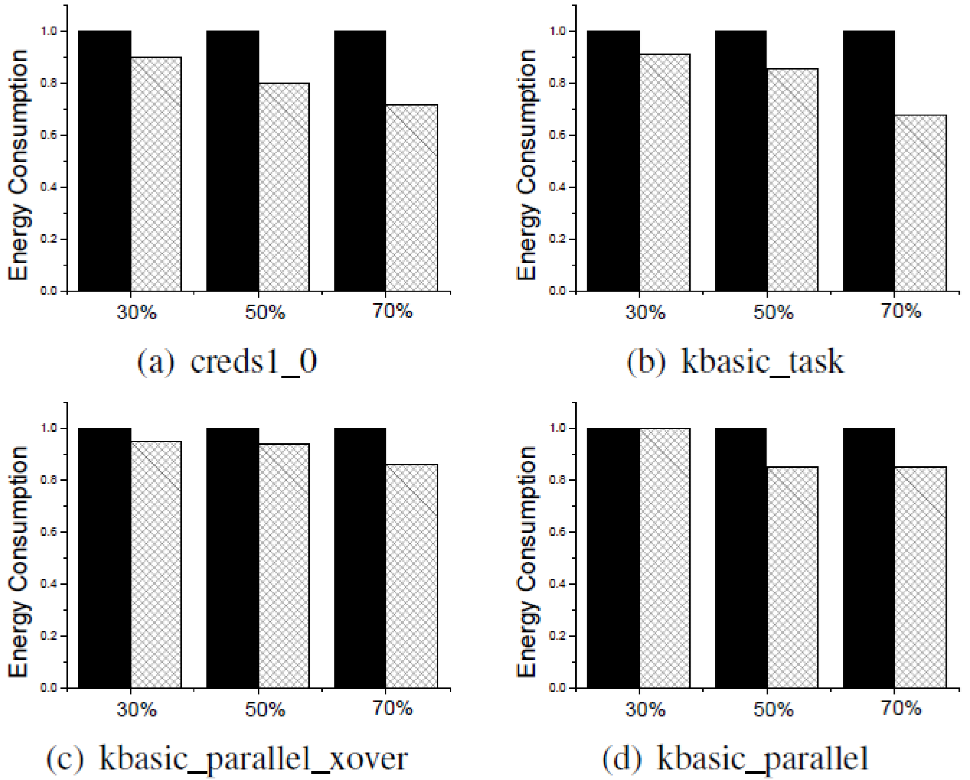
Comparison with ApproxMap

- Probability of Quality Satisfaction (10% quality threshold)
  - Collect the "pass/fail" data over 1000 runs for each application with different resilient portions.

Application	Fraction	ApproxMap		Ours	
		passed	prob.	passed	prob.
creds1_0	30%	934	93.40%	1000	100%
	50%	930	93.00%	986	98.60%
	70%	901	90.10%	951	95.10%
kbasic_task	30%	899	[illegible]	1000	[illegible]
	50%	810	[illegible]	901	[illegible]
	70%	649	64.90%	886	88.60%
kbasic_parallel_xover	30%	719	71.90%	795	79.50%
	50%	583	58.30%	830	83.00%
	70%	798	[illegible]	950	[illegible]
kbasic_parallel	30%	962	[illegible]	992	[illegible]
	50%	970	97.10%	974	97.40%
	70%	969	96.90%	984	98.40%

Comparison with ApproxMap

- Efficacy of Online Adjustment
  - As we use the same offline scheduler of ApproxMap, the evaluation of our online procedure is presented by comparing energy consumptions with *ApproxMap*.
  - In terms of normalized energy consumption, wherein we set *ApproxMap* as 1 and error threshold as 10% .



Thank You for Your Attention!