Slack-based Bus Arbitration Scheme for Soft Real-time Constrained Embedded Systems

Minje Jun

Kwanhu Bang

Hyuck-Jun Lee

Naehyuk Chang

Eui-Young Chung

Contents

- Motivation
- Previous works
- Key Idea
- Introduce proposed scheme in detail
- Experiment setting and results
- Conclusion

Motivation

- ✓ **Rapid growth in multimedia device market**
  - MP3 player, PMP, DTV, Mobile Phone, etc
- ✓ **Convergence of features into a single device**
  - Increases data intensity
  - So, higher performance bus is required
- ✓ **Real-time nature of multimedia applications**
  - Latency is becoming a more critical factor

Previous Works

- To meet the need for higher performance bus,
  - Some improved bandwidth fairness and minimized latencies of certain modules [K.Lahiri et al.]
  - Some proposed new bus architectures both to improve bandwidth and to minimize overall latency [R. Lu et al., K. Sekar et al]

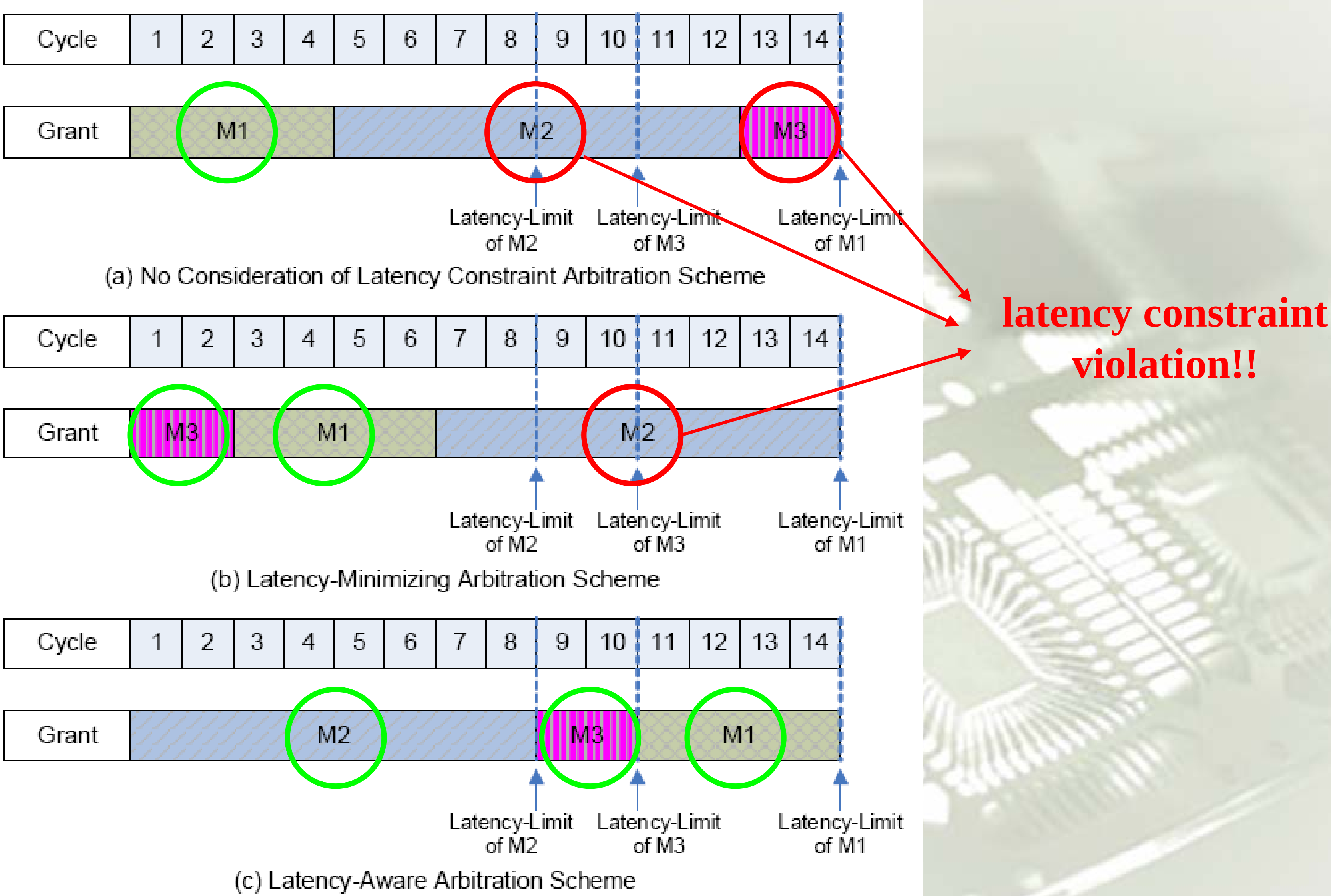
- But there were some insufficiency
  - Latency over-reducing of some modules may cause latency violation of others
  - Large HW overhead for larger scale systems in future

What We Propose Is..

✓ **So, we propose *latency-constraint concerning* bus arbitration scheme,**

- **Without change in IPs**
- **Without change in bus protocol**
- **With negligible change in existing arbiter**
- **With acceptable HW overhead**

Basic Concept of Slack-based Arbitration



Slack Calculation

✓ Use latency slack, which is

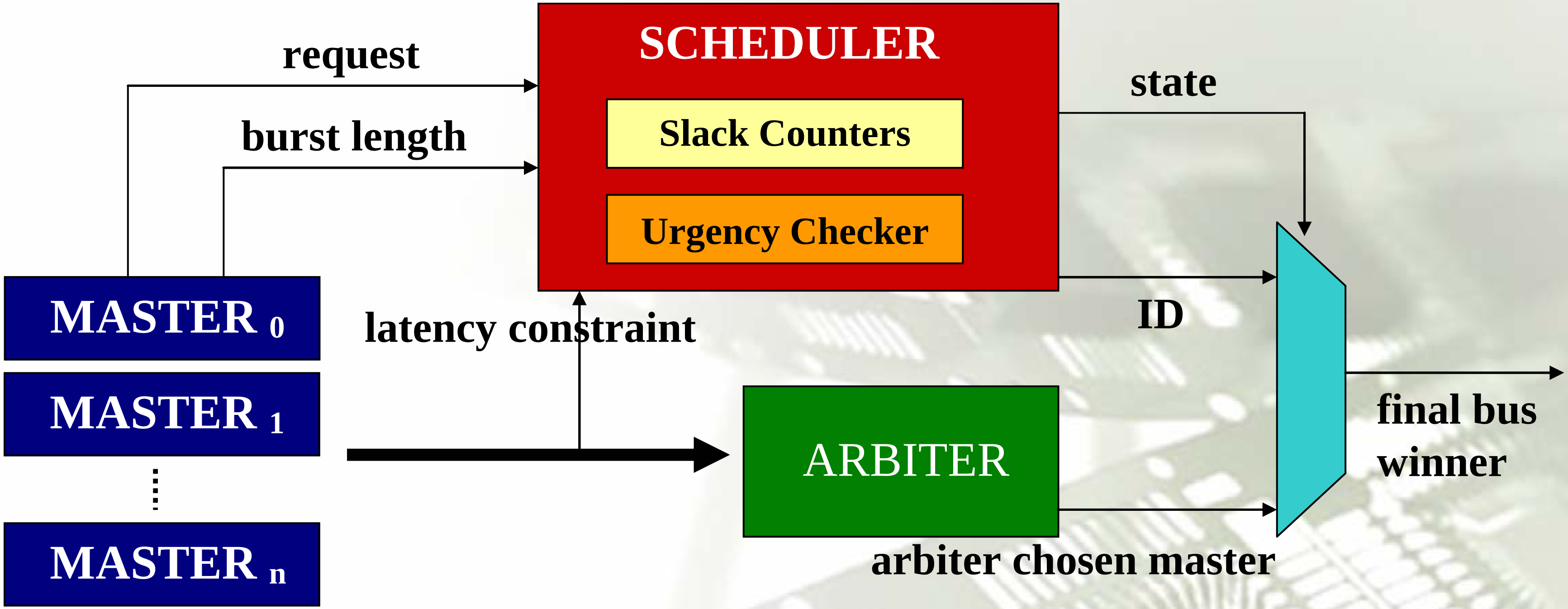
$$Slack_i = L_i - T_i \times B_i - S_j$$

where L_i : the given latency constraint,
T_i : unit burst beat transfer time
B_i : burst length of the transfer of i^{th} master
S_j : latency of target j^{th} slave

✓ Multiplication can be avoided

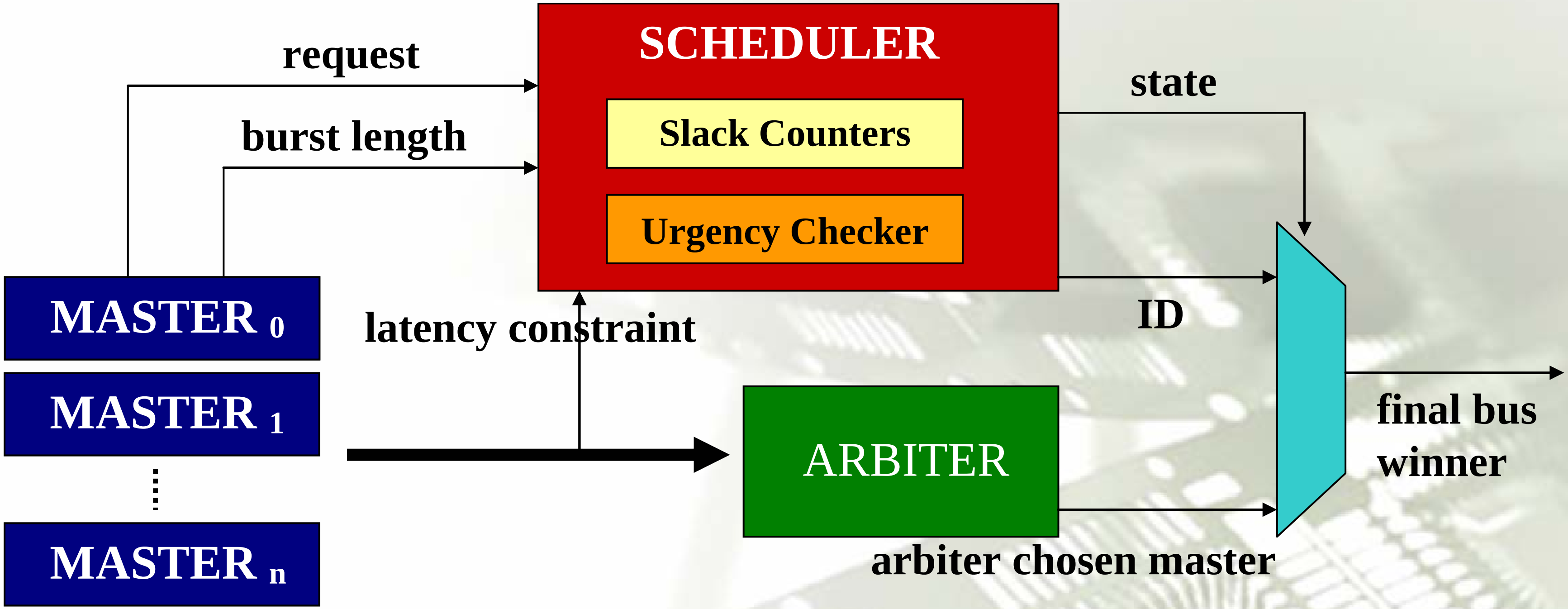
– Since, practically, T_i is one clock cycle

Scheme Overview



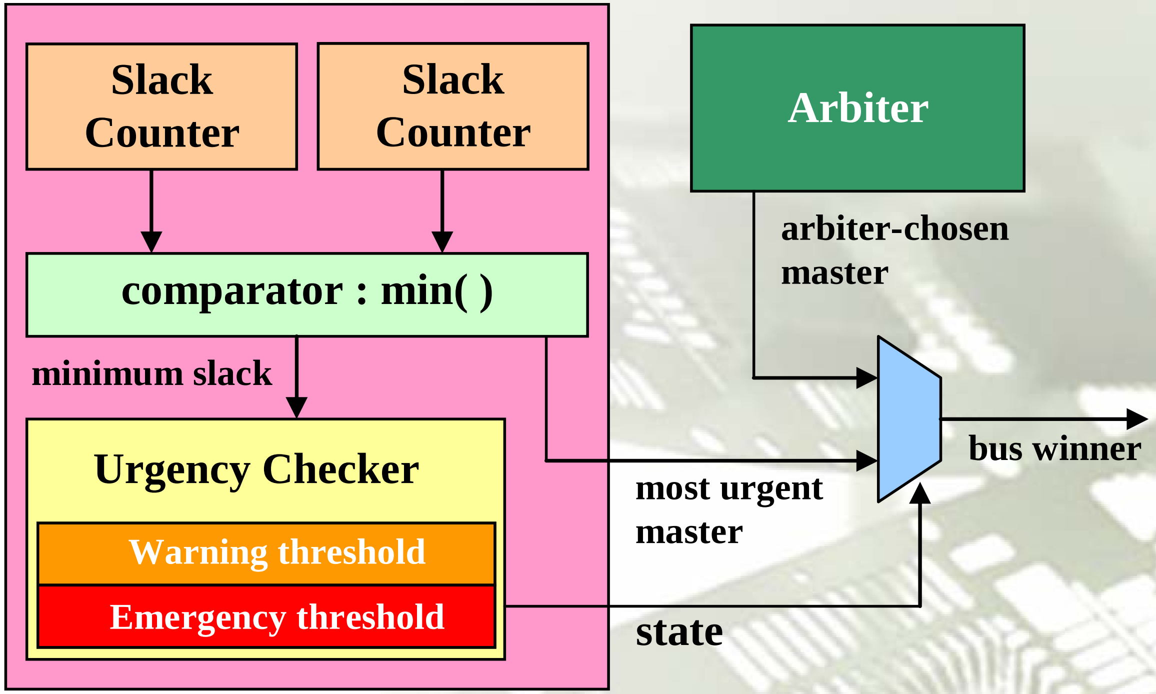
- Masters request signals to scheduler as well as arbiter
- Each master's burst length signals to scheduler
- Master programs its constraint to scheduler through bus

Scheme Overview (cont'd)



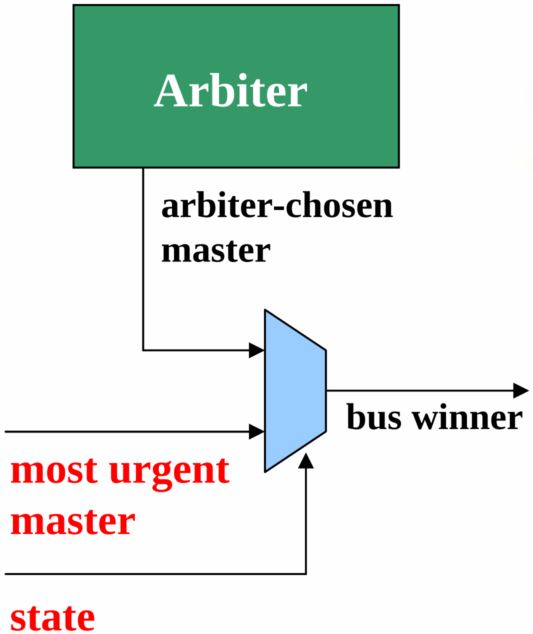
- When a master starts request, scheduler starts counting its slack
- Scheduler outputs *ID* of the most urgent master and *state* of its urgency

Scheduler Detail



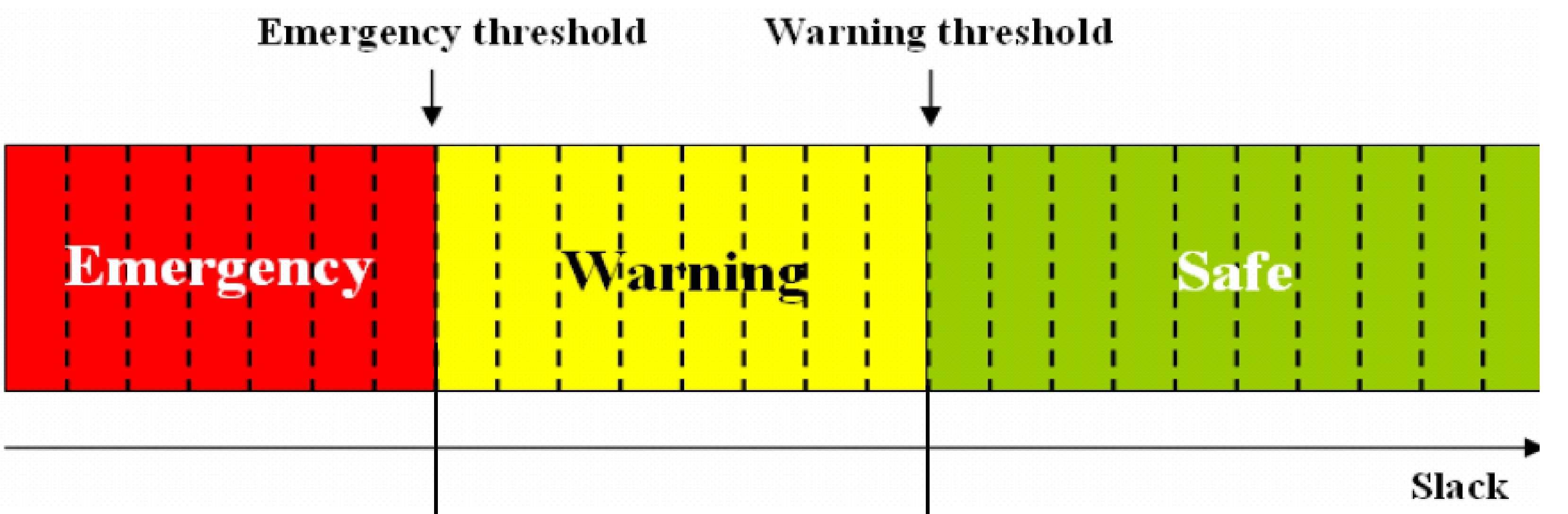
- Scheduler consists of <u>the same number of slack counters</u> as the latency-sensitive masters, <u>a comparator</u> finding the minimum slack, <u>an urgency checker</u> with the warning and the emergency thresholds

Scheduler Detail – Final Stage



- Most urgent master (min. slack) selected by scheduler
- Arbiter selects a master just as its normal operation
- According to the *state* signal, the final bus winner is chosen
- Negligible change needed to existing arbiter

Scheduler Detail – State Policy

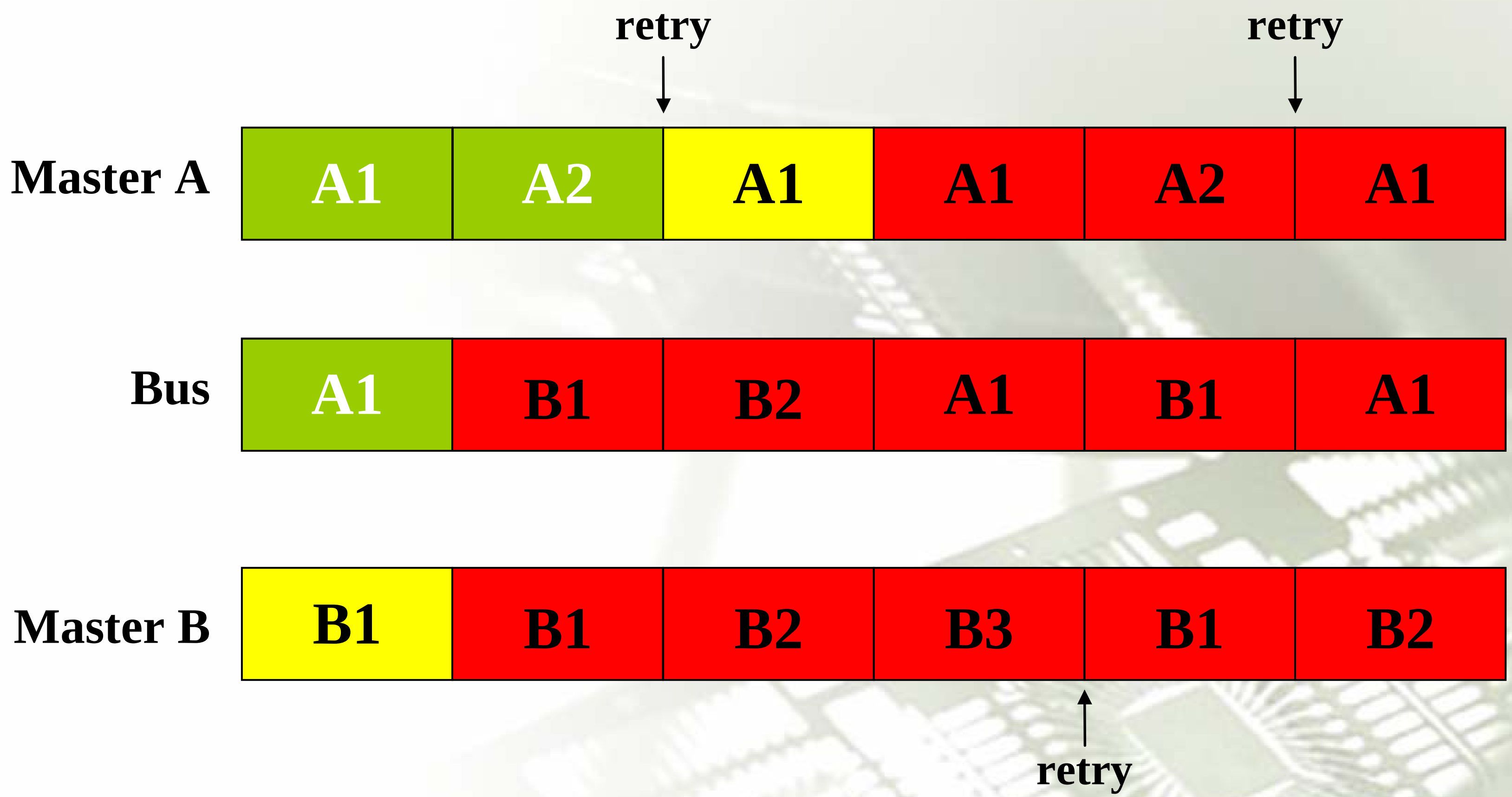


- Scheduler-chosen master get the bus ownership right now
- Unless the current transfer is locked

- Scheduler-chosen master get the highest priority in next bus grant decision
- But cannot interfere the current transfer

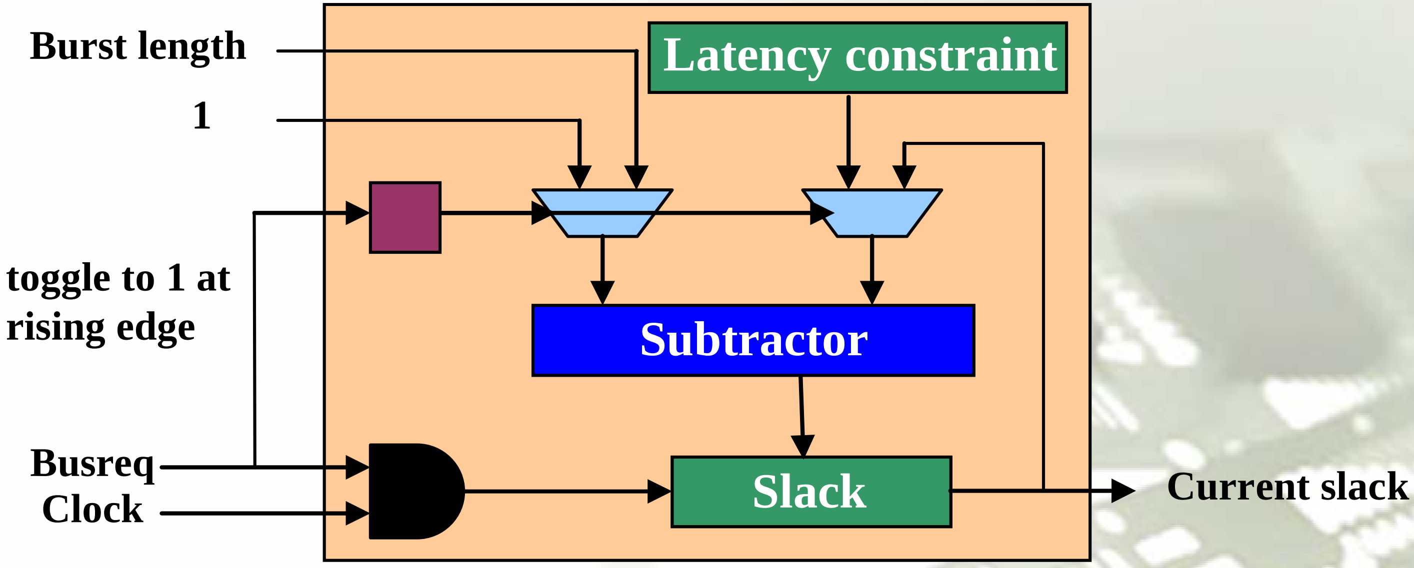
- No scheduler's effect on the arbitration
- Arbitration is solely dependent on the arbiter

Scheduler State Policy – Deadlock



- ✓ To avoid this deadlock, use *lock* feature to the transfer which was once interrupted by another emergency transfer

Slack Counter Detail



- Two registers, Two 2x1 selectors, and One subtractor with a few elementary units.
- When the master starts its request, computes the initial slack
- Then every cycle, it decrements its slack by one

Experiment Settings and Results

Settings

- ✓ **AHB protocol**
- ✓ **4 masters, each requires the same bandwidth as the others.**
  - **Total bandwidth is assumed to be 140% of ideal bandwidth so that it is heavy enough to verify our method.**
  - **Workload summary**

	Bandwidth requirement	Burst length	Latency constraint(cycles)
M1	35%	8	30
M2	35%	8	72
M3	35%	16	30
M4	35%	16	72

Settings (cont'd)

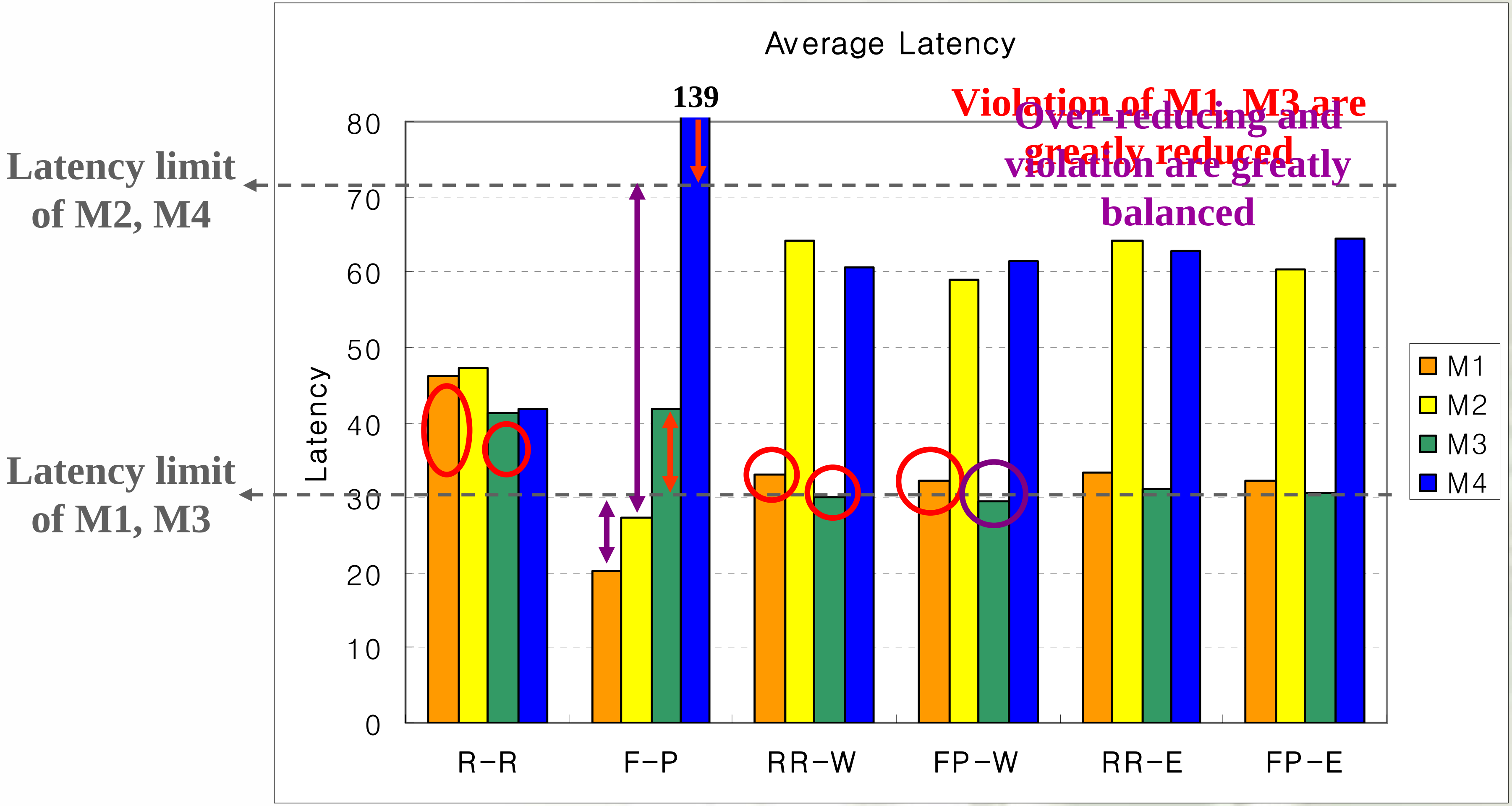
- ✓ 1 slave having delay of 8 cycles
- ✓ Comparison with : round-robin/fix-priority arbiter
  - with / without scheduler augmented and
  - with / without emergency state enabled

<Nomenclatures>

	R-R	RR-W	RR-E	F-P	FP-W	FP-E
Arbiter	round-robin	round-robin	round-robin	fixed-priority	fixed-priority	fixed-priority
Warning	x	o	o	x	o	o
Emergency	x	x	0	x	x	o

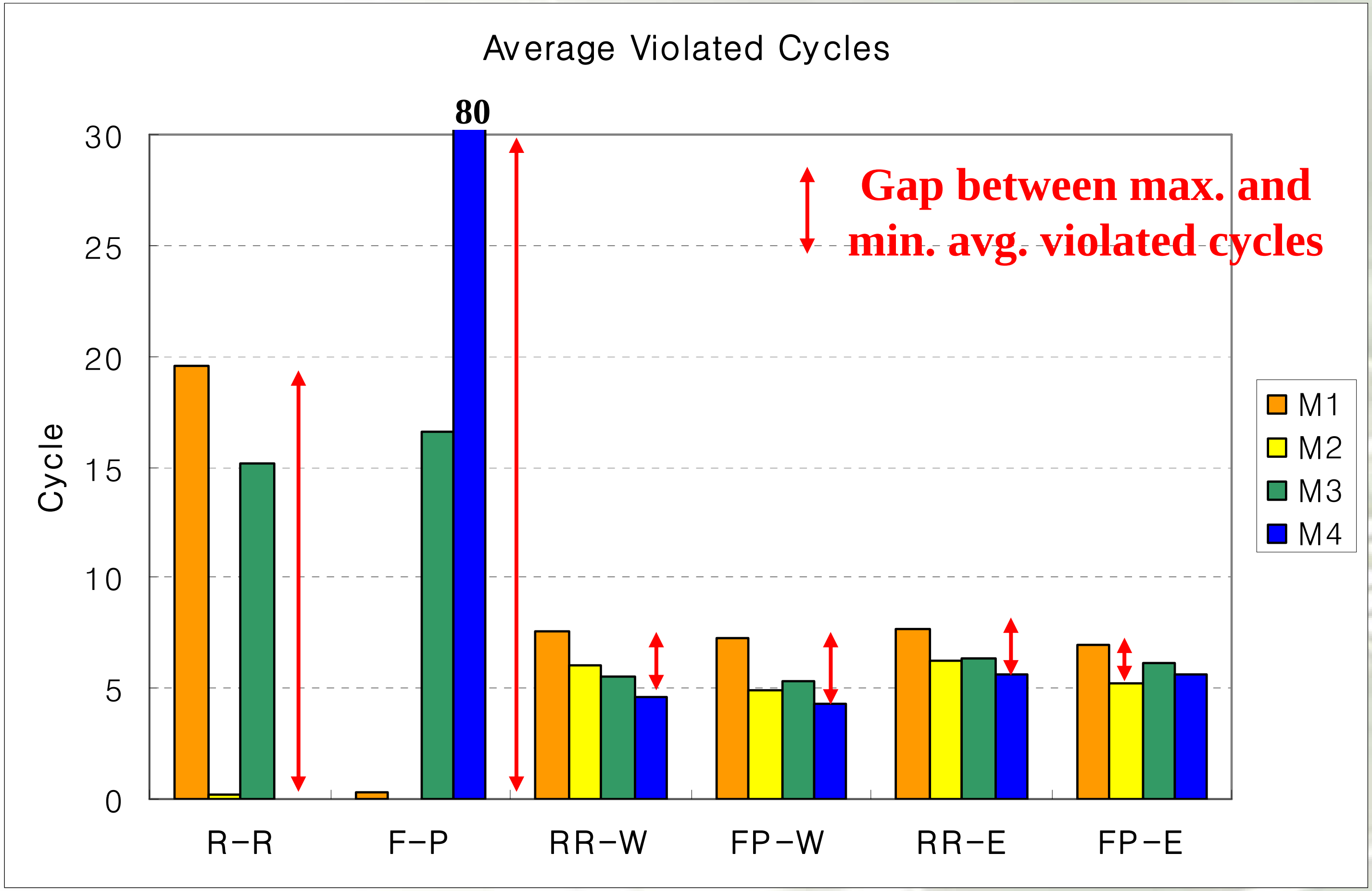
Average Latency

- ✓ **Average latency of each master in different configurations**



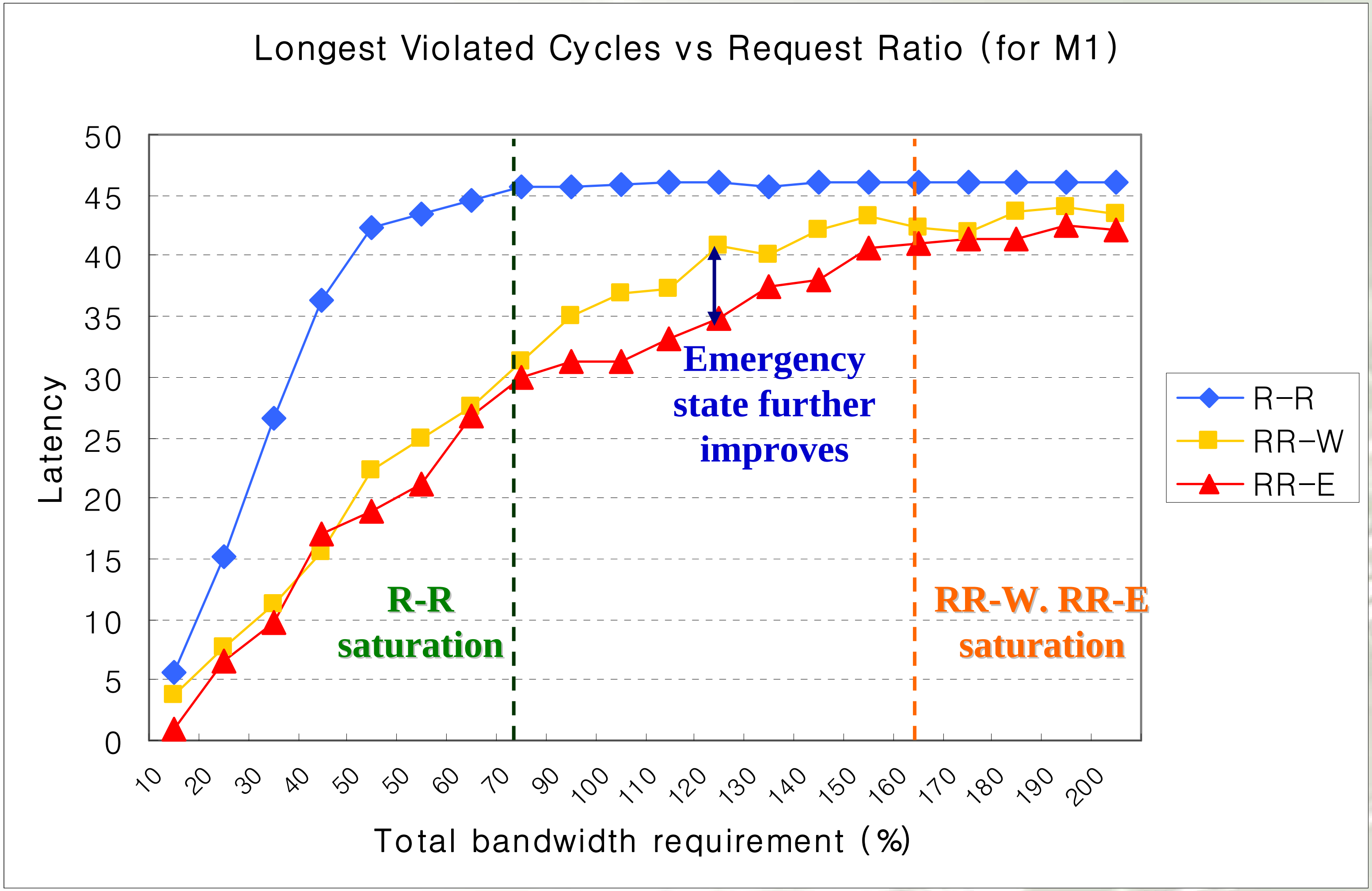
Average Violated Cycles

✓ **Average violated cycles beyond their constraints**



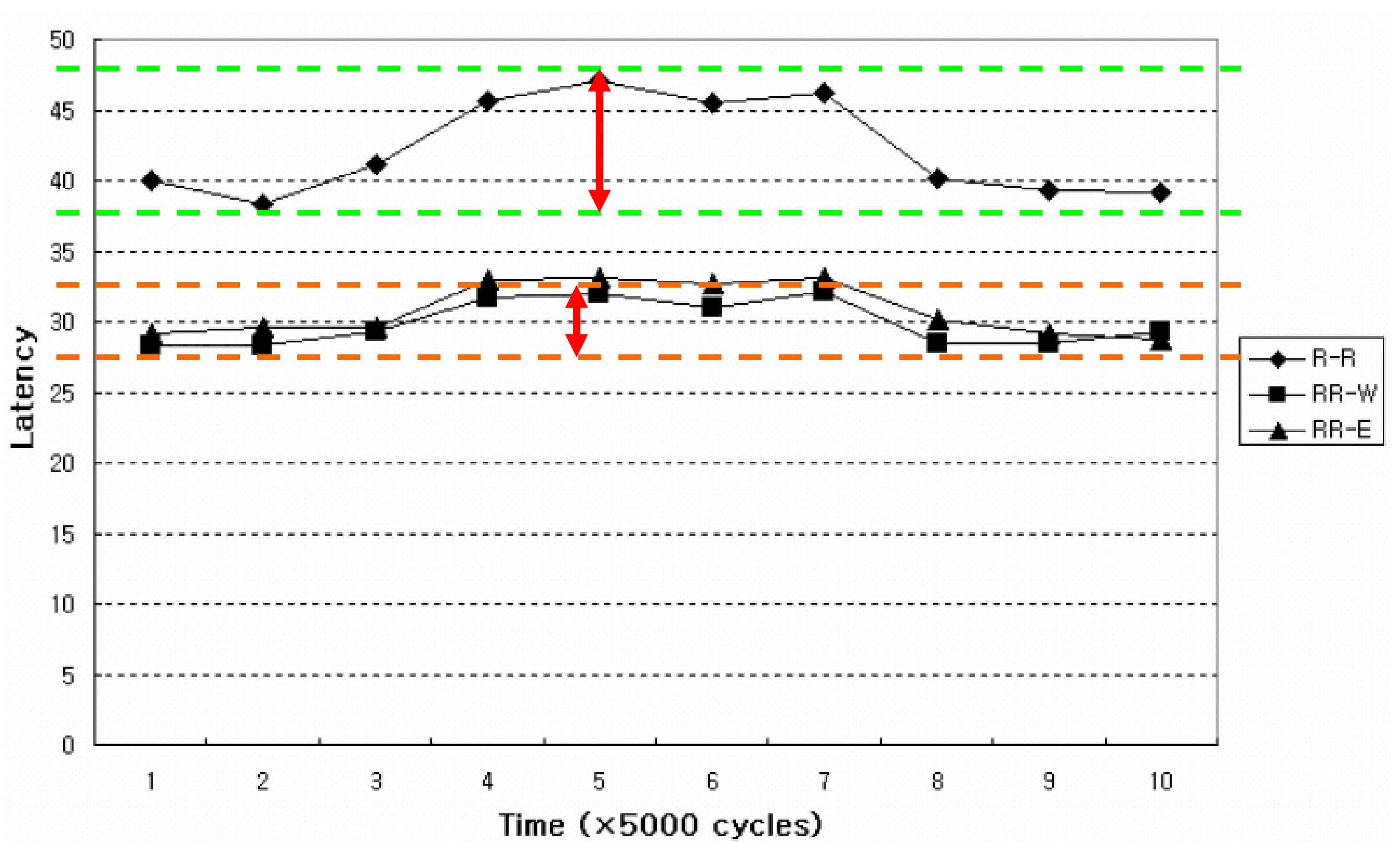
Sensitivity on Traffic Heaviness

✓ **Longest violated cycles against total bandwidth requirement**



Sensitivity on Bursty Traffic

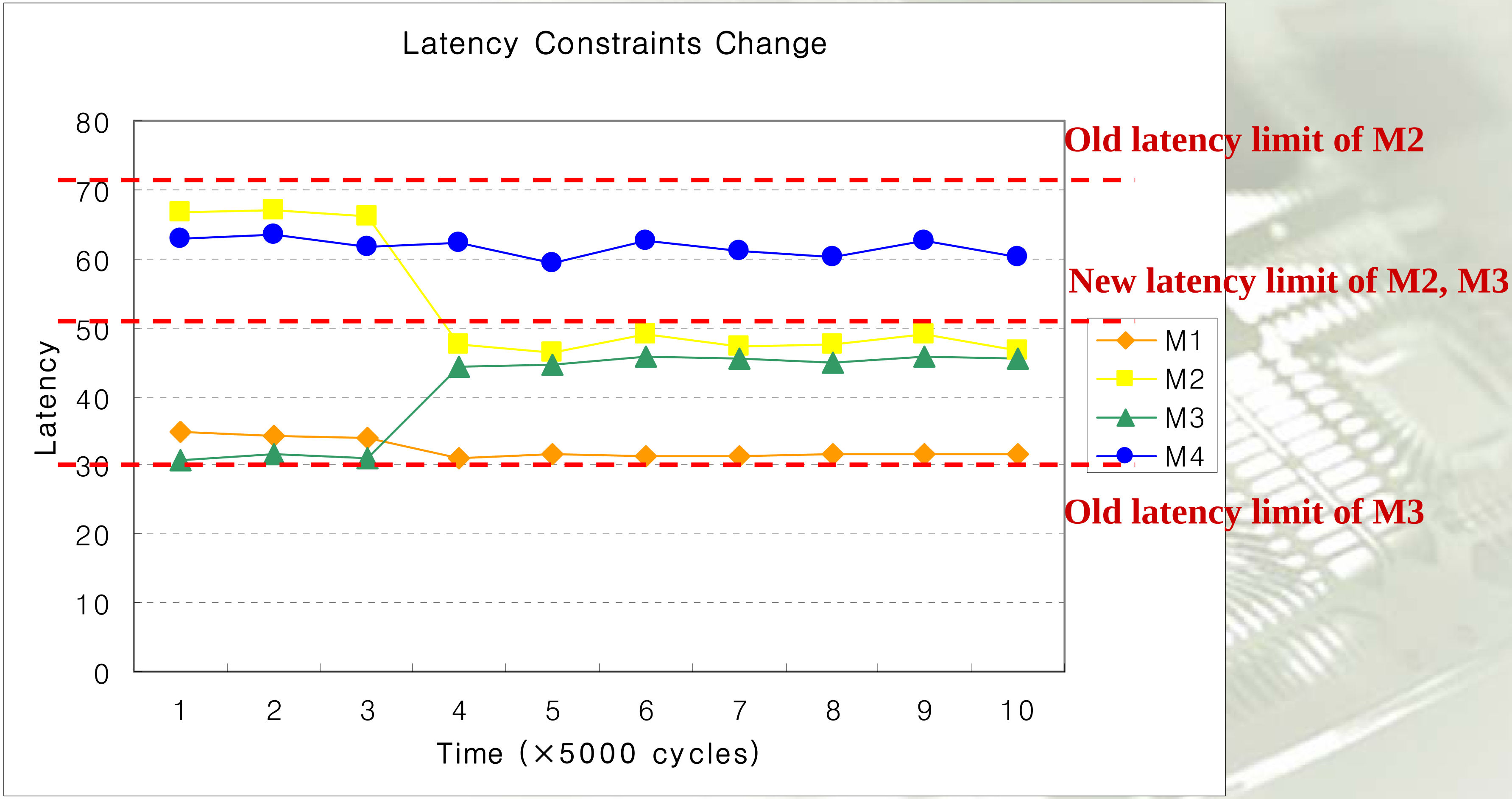
- ✓ **Varying required bandwidth**
  - **M3 requires 15% more bandwidth between 3 and 7 on the x-axis**



- Latency fluctuation due to increase of traffic heaviness is less with scheduler

Sensitivity on Latency Constraint Change

- **Dynamic constraint programming**
  - **M2 decreases its constraint to 51, while M3 increases its to 51.**



Conclusion

- ✓ **With the scheduler, we (with round-robin arbiter)**
  - **reduced average violated cycles by up to 60%**
  - **reduced longest violated cycles by up to 32%**
  - **without change in master/slave IPs,**
  - **and with acceptable additional HW overhead**

- ✓ **The scheduler can be attached to any existing arbiter as a complementary unit to improve latency characteristics for real-time applications**

Thank You Very Much