SafeMC: A System for the Design and Evaluation of Mode Change Protocols

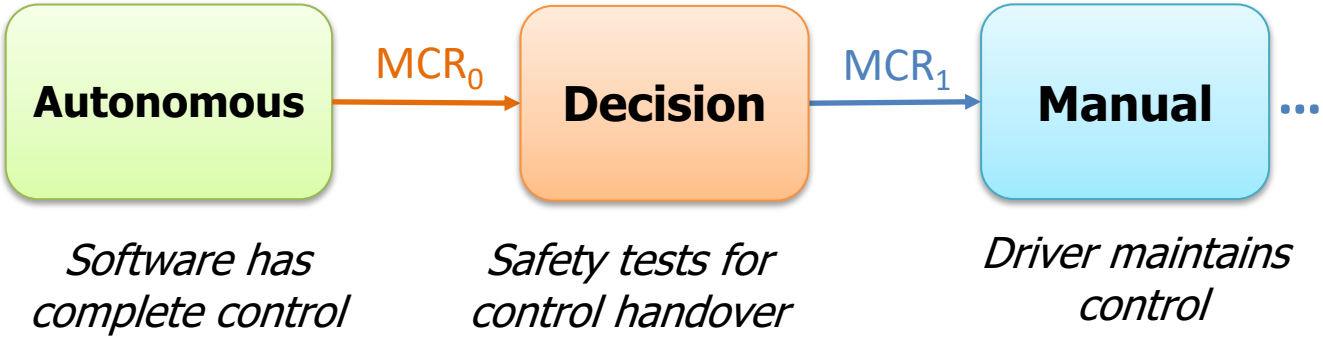
Tianyang Chen **Linh Thi Xuan Phan**

University of Pennsylvania

Penn Engineering

PRECISE
PENN RESEARCH IN EMBEDDED COMPUTING AND INTEGRATED SYSTEMS ENGINEERING

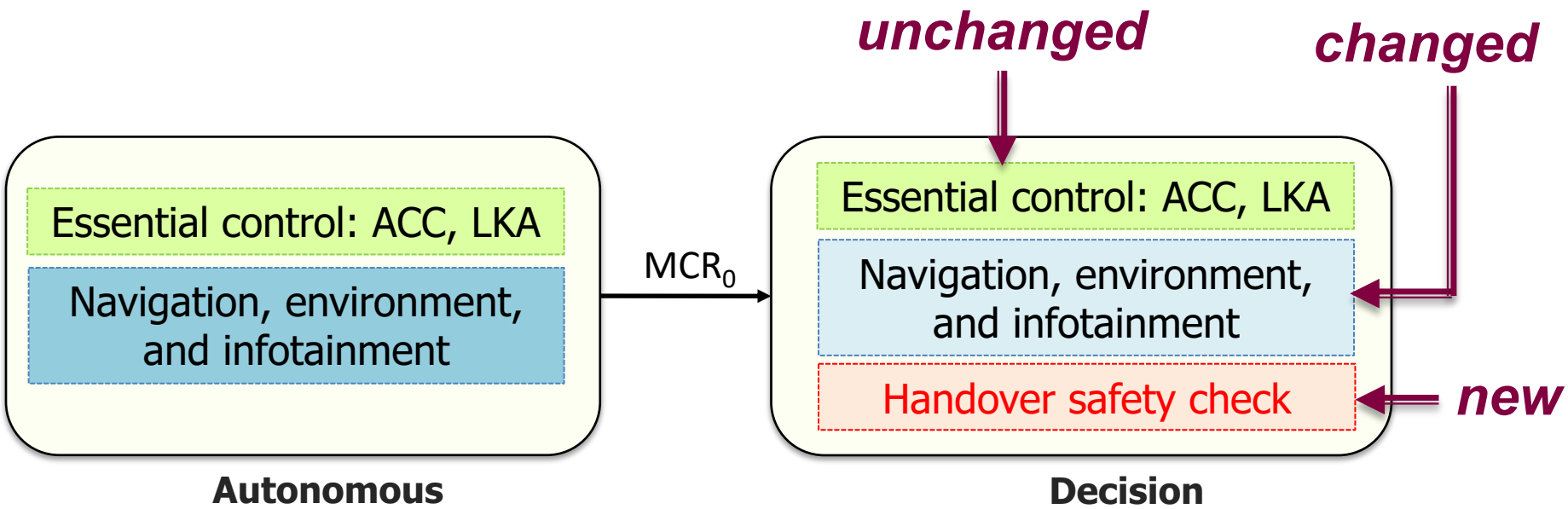
Scenario: Self-driving cars



- **Mode:** a functionality, a system configuration
  - a set of tasks that need to execute
  - e.g., adaptive cruise control, lane keeping assistance, …
- **Mode transition:** a change in the system mode
  - triggered by a mode change request (MCR)
  - e.g., a driver's request to take back control of the vehicle

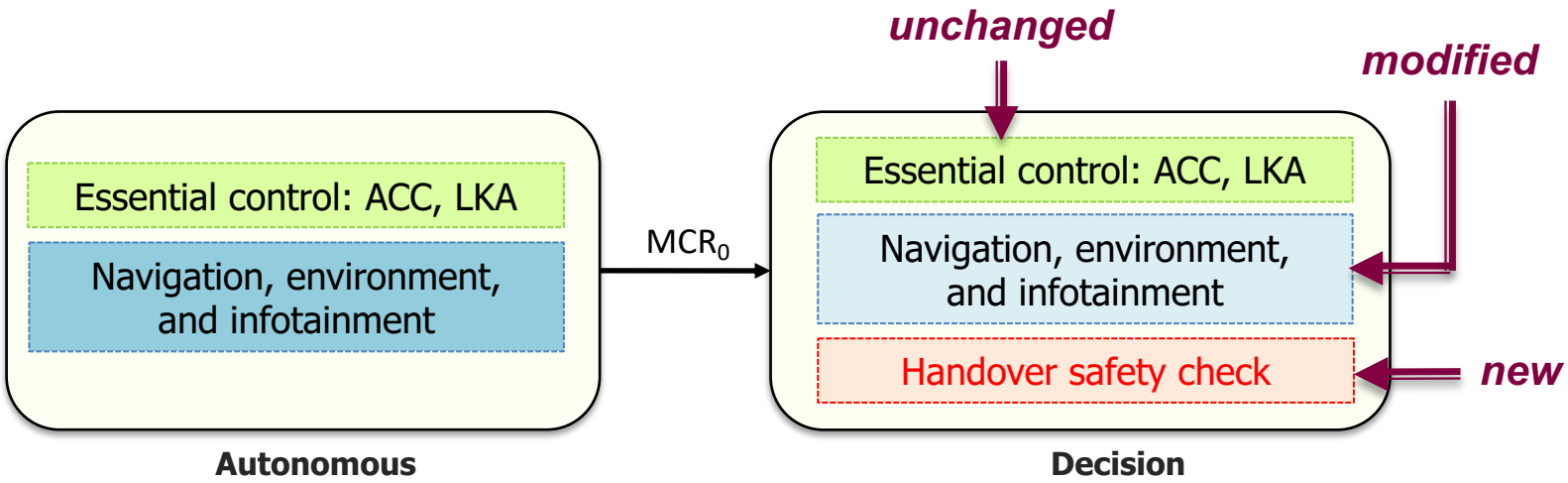
[Gomez et al. Transitioning a mixed-mode vehicle to autonomous mode, Dec 2011. US Patent 8,078,349]

Problems with mode transitions



- Need to execute:
  - Unfinished jobs of the old mode
  - Newly released jobs of the new mode
- Result: Potential overloads and deadline misses

Solution: Mode-change protocol



- Specify execution semantics of a mode transition
  - How to handle unfinished jobs?
  - When to release new jobs?
  - When to apply the modified parameters? etc.
- Goal: Timing guarantees and safety during a transition

Limitations of existing work

- Lack experimental evaluations
  - Timing may be violated in practice due to overheads
- Evaluations performed on different environments
  - Hard to compare between protocols
- Focus on relatively simple protocols
  - Agnostic of mode transitions and tasks' safety criticality
  - Require customizations in practice

How to determine which MCP is the best-fit for a given system?

Contributions

- **An extensible set of mode change primitives**
  - implement a broad spectrum of mode change behaviors
- **SafeMC system design**
  - enables efficient specification and evaluation of MCPs
- **A prototype implementation of SafeMC in Xen**
- **Experimental evaluations**
  - comparisons of existing MCPs
  - case study: design of novel protocols specific for an application

Outline

- Introduction
- **Mode change primitives**
- SafeMC design and implementation
- Evaluation
- Case study

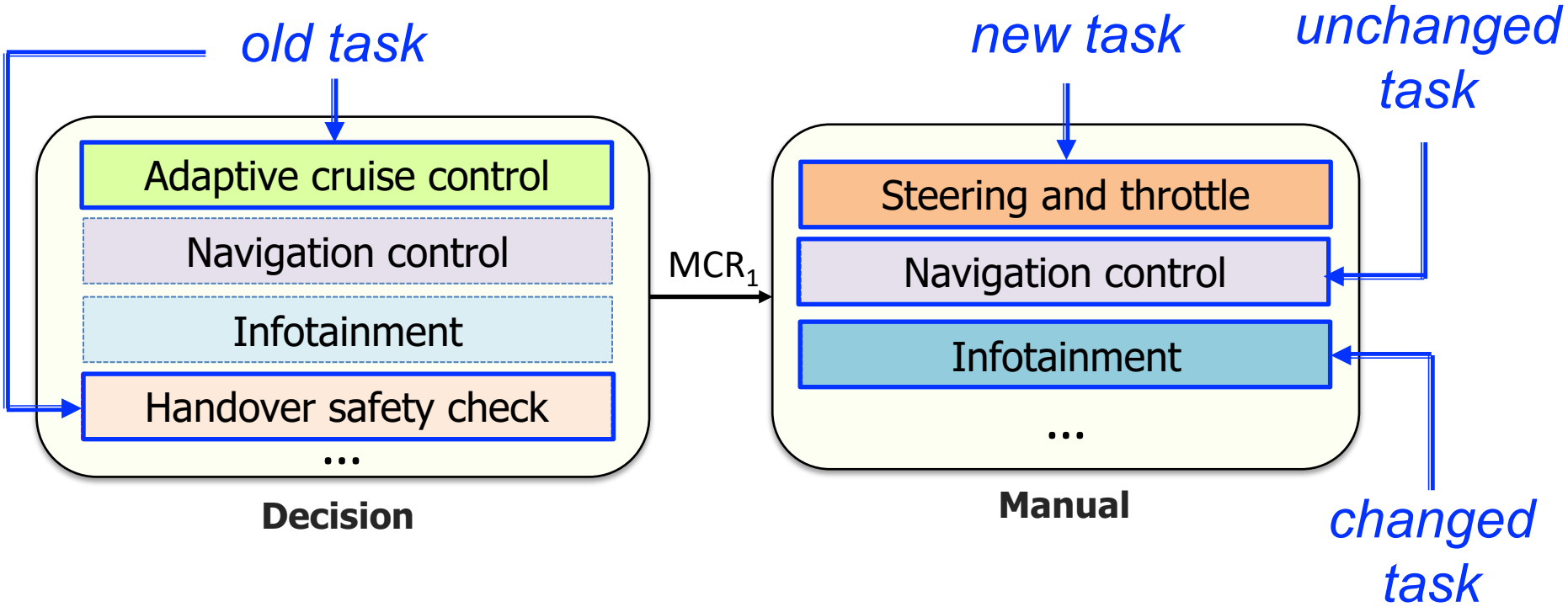
Insight

- MCPs are built on a common set of key primitives
  - Operate on the smallest scheduling entity (a job)
  - Specify the actions that the MCP must perform
  - Specify conditions under which the actions to be invoked
- Primitives are composed at different levels of granularity to form MCPs
  - Job type, task, task type, or transition
- Example: Maximum Period Offset
  - For all transitions: *transition type (global)*

    New jobs of changed tasks: released after an offset

    job type *task type* *action* *condition*

Types of tasks and jobs



- Job type
  - Pending: Unfinished jobs that are not currently executing
  - Executing: Currently executing jobs
  - New: New jobs to be released after the MCR instant

Basic primitives

- Describe actions for jobs of a specific type, for a particular task, or for all tasks of a specific type

Primitive = [Action; Guard]

Action to be applied to the considered jobs (e.g., abort)

Conditions for when the action should be applied (e.g., after a certain offset)

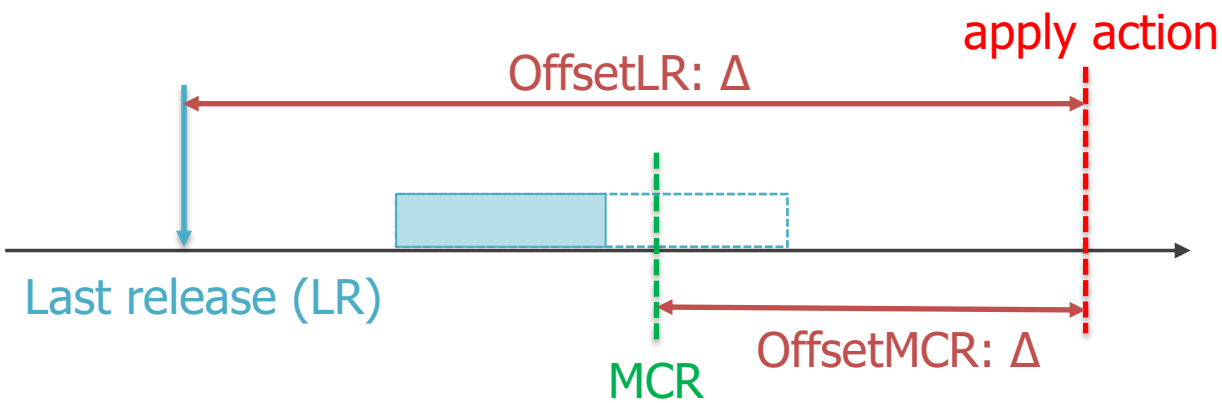
- Compose primitives to form an MCP for the system

Basic primitive: Actions

Action	Description	Use
CONTINUE	The jobs are continued to be scheduled with current timing attributes (WCET, deadline, period, criticality)	*Pending or Executing job types*
UPDATE	The jobs continued to be scheduled but timing attributes are changed to that of the new mode	
ABORT	The jobs are aborted and removed from the system	
ABORT[K]	The oldest K jobs are aborted and removed from the system	
RELEASE	New jobs whose attributes are defined in the new mode will be released when the associated guard becomes true	*New job type*
RELEASE_O	New jobs whose attributes are defined in the old mode will be released as usual while the associated guard is true	

**If an action has no guard, it will be applied immediately*

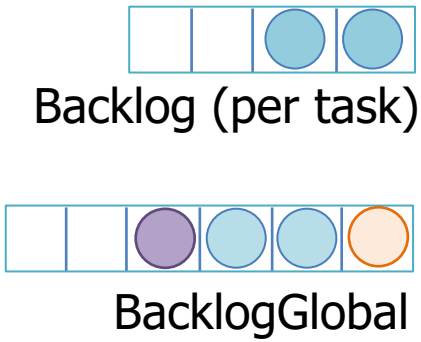
Basic primitive: Offset guards

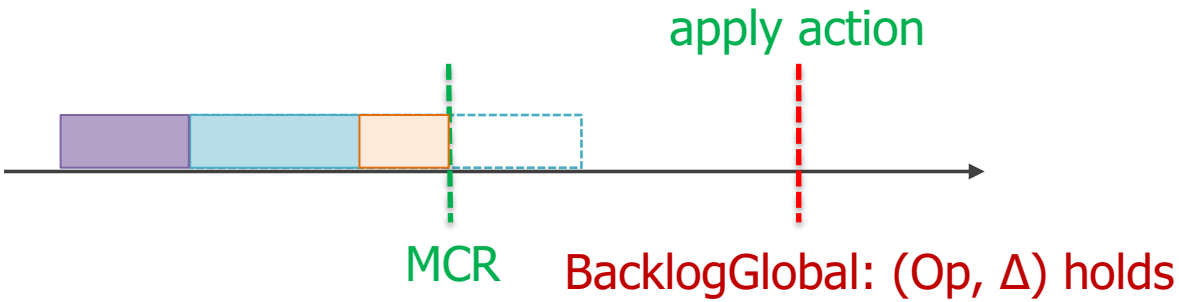


- Apply the action after an offset Δ from MCR or LR instant
  - Δ is a constant threshold, or based on task's parameters

- Example: [RELEASE; OffsetLR: MAX_PERIOD]
  - The first new job is released after a delay equal to the max period of tasks in old and new modes from the last release

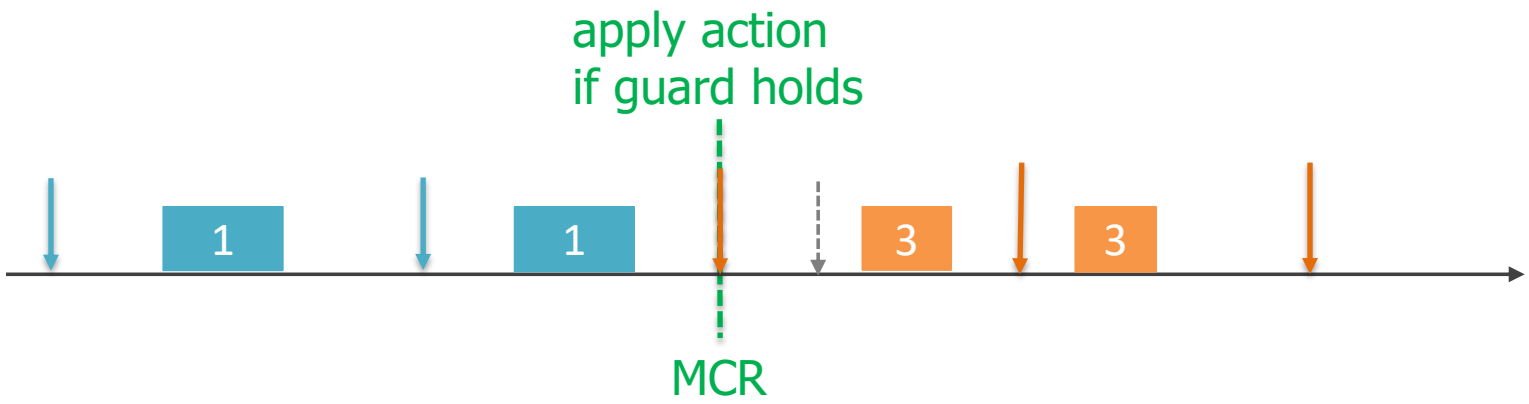
Basic primitive: Backlog guards





- Specify a formula on the backlog of the job queue
  - Op ∈ { ≤, ≥, =, <, >}
  - Δ: a constant, or based on the queue size
- Example: [ABORT; Backlog: (≥, 0.5 · MAX_VAL)]
  - Abort old/pending jobs of the task if its queue is at least half full
- Example: [RELEASE; BacklogGlobal: (=, 0)]
  - Release new jobs when the global queue is empty

Basic primitive: Criticality guards



- Specify a formula on the task's old and new criticality
- Example: [RELEASE; Criticality: (<, OLD_VAL, NEW_VAL)]
  - Release new jobs immediately if the task's criticality is increased
- Example: [ABORT; Criticality: (≤, OLD_VAL, 2)]
  - Abort all existing jobs with criticality less than or equal to 2

More primitives are available in the paper

Examples of MCP specs

<table>
<tr><td>JOB TYPE</td><td colspan="3">New (Action = RELEASE)</td><td>Exec/Pending</td></tr>
<tr><td>TASK TYPE</td><td>UNCHANGED</td><td>CHANGED</td><td>NEW</td><td>ALL (except NEW)</td></tr>
<tr><td>Max Period Offset</td><td>OffsetLR: OLD_PERIOD</td><td colspan="2">OffsetMCR: MAX_PERIOD</td><td rowspan="5">CONTINUE</td></tr>
<tr><td>Min Offset w/ Periodicity</td><td>OffsetLR: OLD_PERIOD</td><td colspan="2">BacklogGlobal: (=, 0)</td></tr>
<tr><td>Min Offset w/o Periodicity</td><td colspan="3">BacklogGlobal: (=, 0)</td></tr>
<tr><td>Asyn. with Periodicity</td><td colspan="2">OffsetLR: OLD_PERIOD</td><td>OffsetMCR: Δ</td></tr>
<tr><td>Asyn. w/o Periodicity</td><td colspan="3">OffsetMCR: Δ_i</td></tr>
</table>

[Real and Crespo. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. RTS 2004.]

Outline

- Introduction
- Mode change primitives
- **SafeMC design and implementation**
- Evaluation
- Case study

SafeMC Design

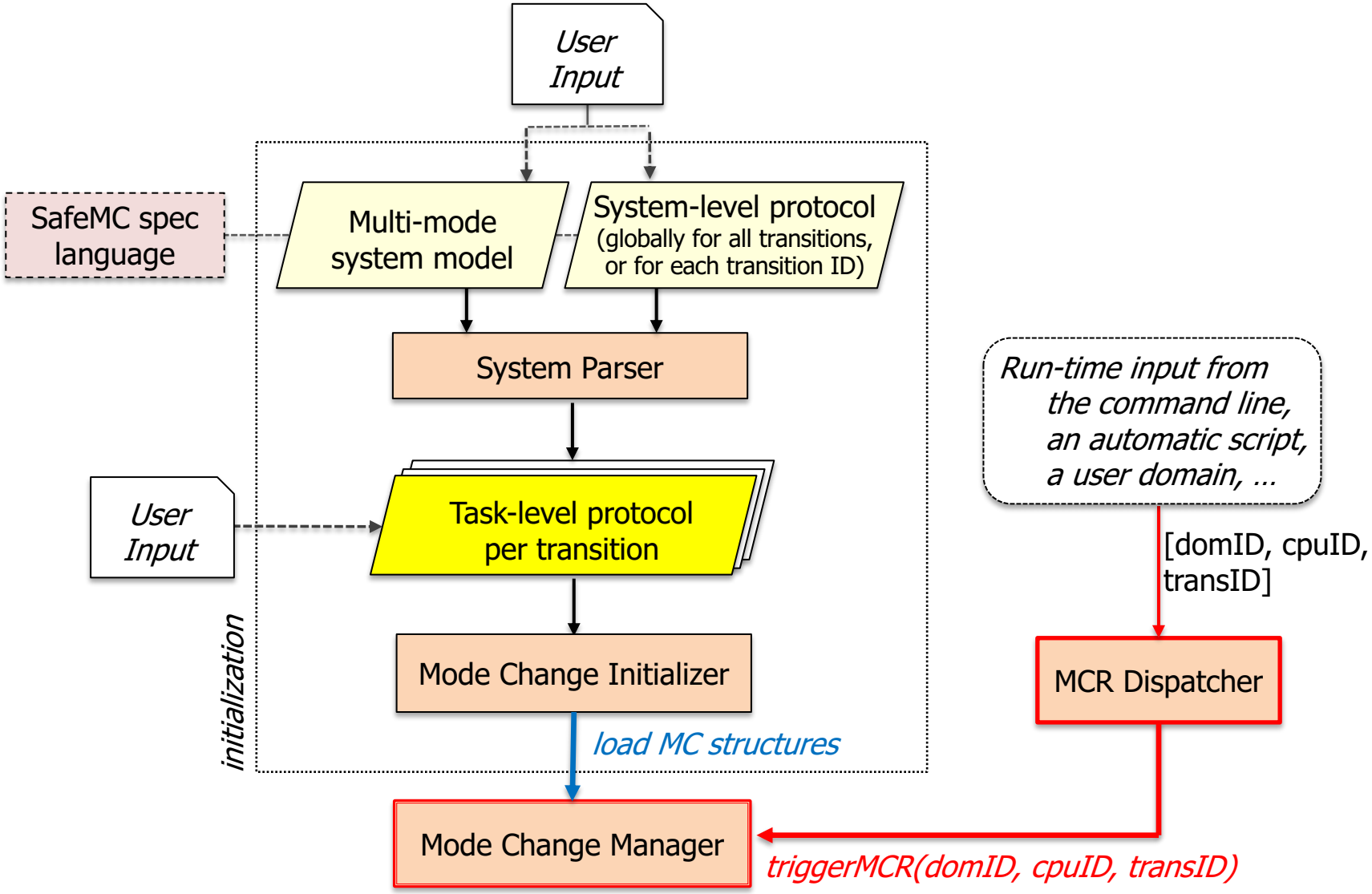
MCP-specific algorithmic-core

- Define mode change primitives
- Easily customized or re-written from scratch

MCP-independent common runtime

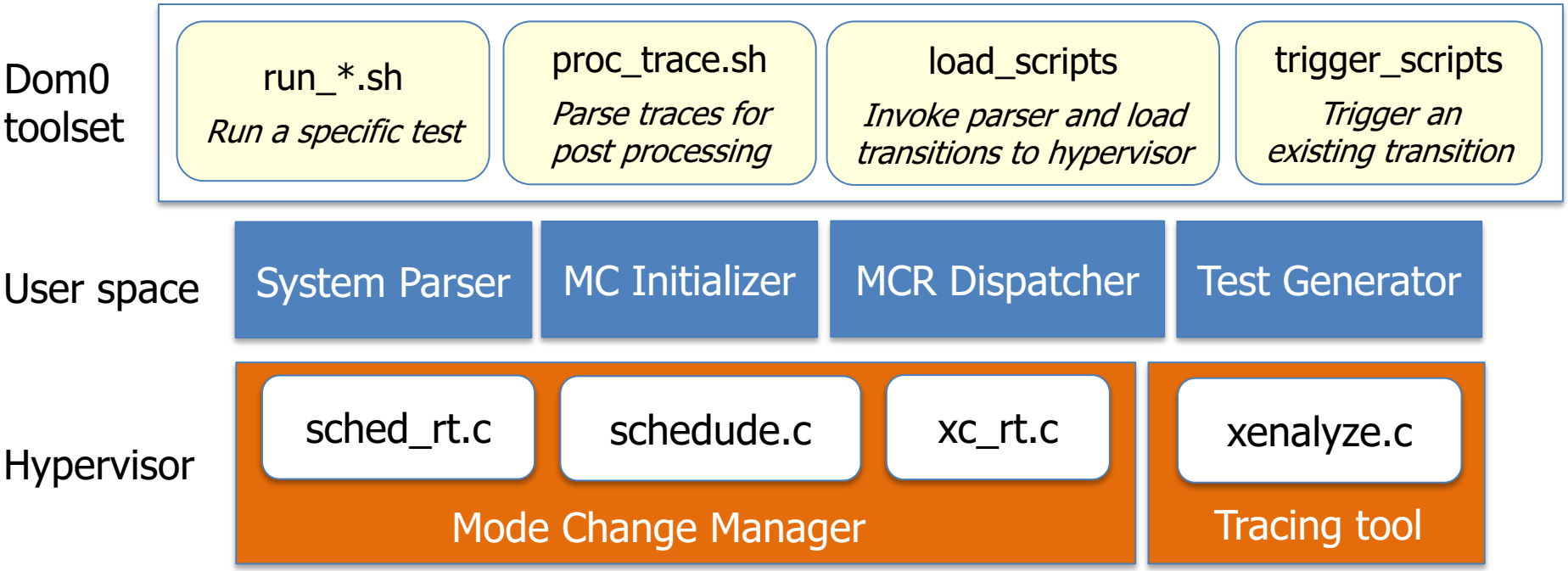
- Implement the algorithm
- Simplify experimentation
- Enable fair comparisons on a real platform

SafeMC Runtime



Prototype

- Xen-based implementation of SafeMC
  - Support mode changes at the hypervisor level
  - Rationale: substantial interests in real-time virtualization, potential applications to beyond traditional RT systems,…



Outline

- Introduction
- Mode change primitives
- SafeMC design and implementation
- **Evaluation**
- **Case study**

Evaluation setup & goals

- Platform: Xen with SafeMC enabled
- Workload: Randomly-generated multi-mode systems
  - Each mode has 4/8/16/32 VCPUs and utilization ≤ 96%
  - VCPU's WCET in [1.5ms,12ms], max period = 50ms
- Goal #1: Run-time overhead of SafeMC
- Goal #2: Protocol comparisons
  - Release-new: How fast the new mode is activated
  - Finish-old: How fast the system completely leaves the old mode
  - Finish-old-new: How fast the system completely enters the new mode
- Goal #3: Develop novel protocols for a given application
  - using a self-driving case study (based on Google's patent)

SafeMC Run-time overhead (ns)

Overhead averaged across 1000 measurements per protocol per multi-mode system

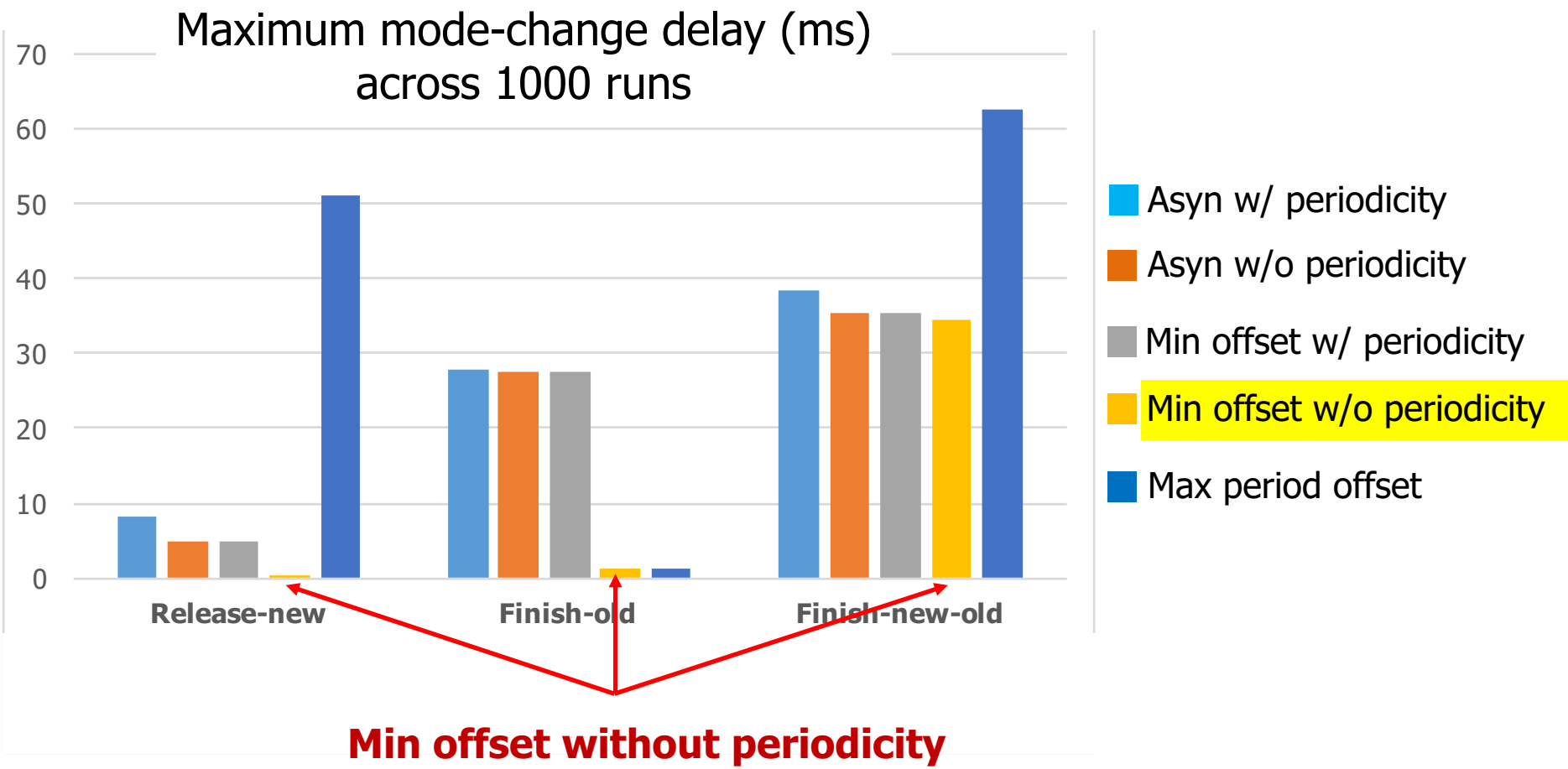
#VCPUs	Schedule	Context switch	Release	Mode change
4	540.5	125.4	1025.0	1986.2
8	478.2	86.8	1465.5	3697.5
16	399.8	44.5	1900.8	7063.2
32	366.2	64.0	3267.0	12753.5

Negligible overhead
(comparable to vanilla Xen's RTDS*)

Reasonable
MC overhead

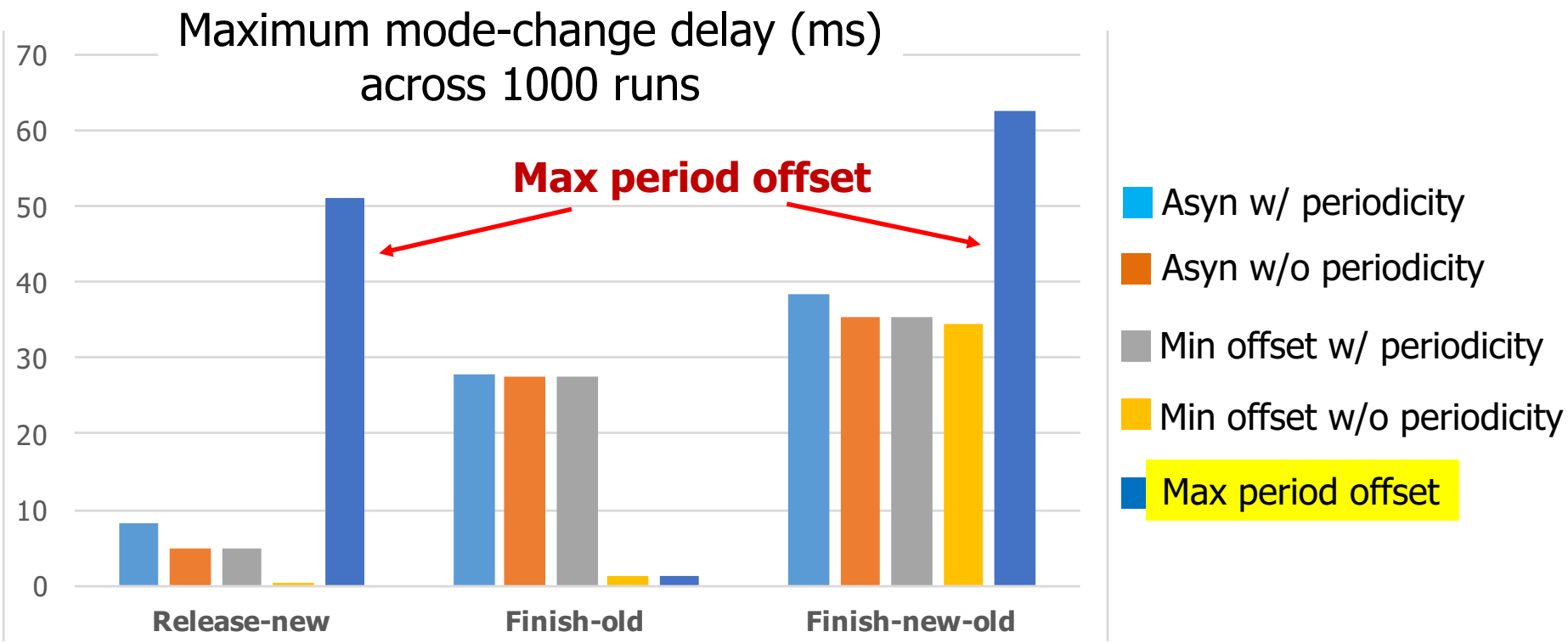
[*] Xi et al. Real-time multi-core virtual machine scheduling in Xen. EMSOFT2014

Result: Protocol comparisons



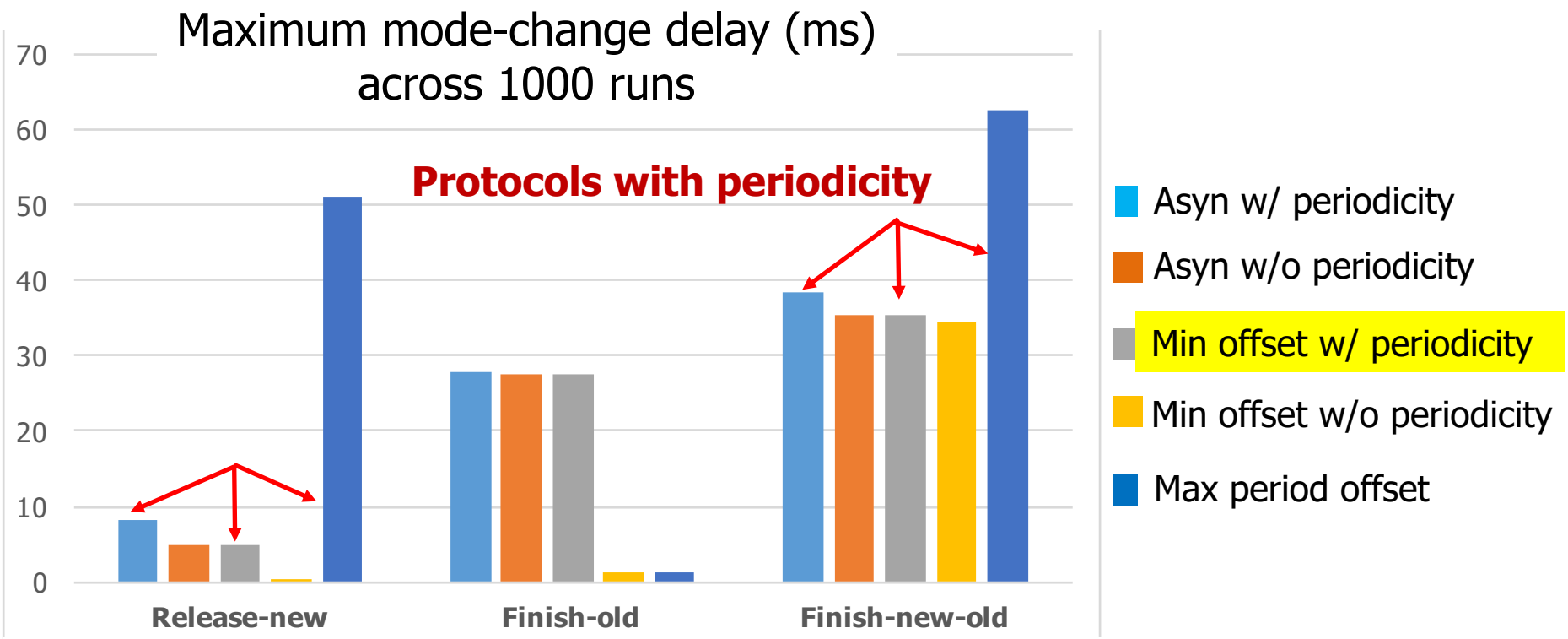
- Delay new releases until existing jobs complete → **smallest MC delay**
- Suitable for applications where periodicity is not critical

Result: Protocol comparisons



- Delay releases of new and changed tasks by an offset = max period
- **Perform poorly in activating/completely entering the new mode**
- Should be avoided if tasks in the new mode are highly critical

Result: Protocol comparisons



- Unchanged tasks are not affected by the MC
- **Min offset w/ periodicity performs well in entering new mode**
- Good choice when periodicity is required and the new mode is critical

Case study: Result

Max mode change delay (ms) across 100 runs

Transition	Protocol	Release-new	Finish-old	Release-new-old
MCR_0	Existing	32.984	2.005	39.927
	Hybrid	3.441	15.223	26.176
MCR_1	Existing	46.312	8.305	51.036
	Hybrid	0.005	1.908	11.947

- MCR_0: Hybrid protocol activates and completely enter the Decision mode much more quickly
- MCR_1: Hybrid protocol outperforms in all three metrics

SafeMC enables the design of novel MCPs specific to the applications

Conclusion

- Choosing the right MCP is critical for safe and efficient design of a multi-mode system
- SafeMC: a system for designing, experimentally evaluating and comparing MCPs
  - MCP-specific algorithmic “core” that implements a broad extensible set of common mode change primitives
  - MCP-independent, shared runtime for executing and evaluating an MCP in a multi-mode system
- Benefits
  - Enable side-by-side experimental comparison
  - Support rapid prototyping and customization