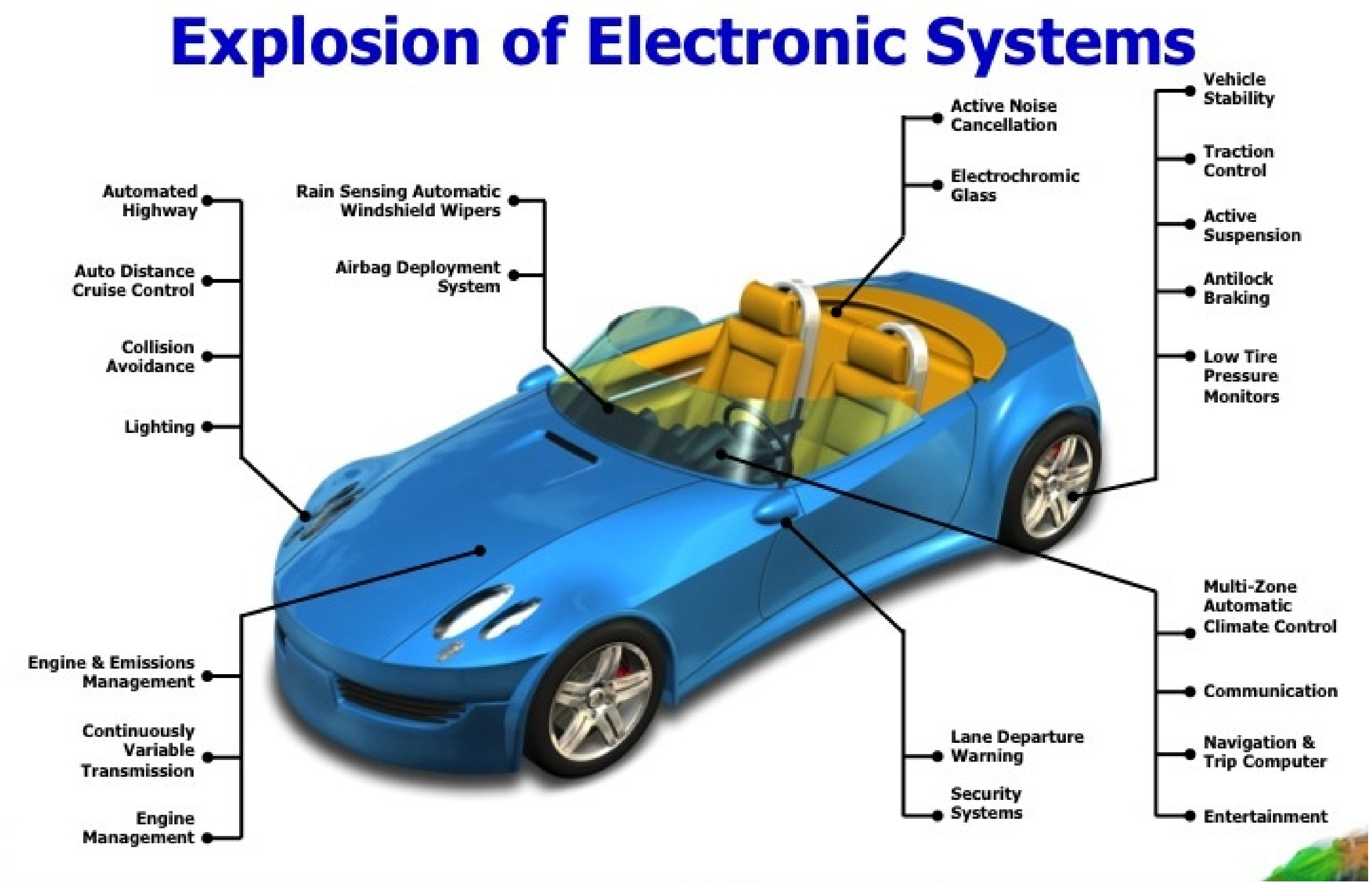
8. Mixed-criticality systems

Where we see how the want of *more-for-less* has entered the high-integrity domain, requiring tasks with different levels of criticality to coexist in a real-time system

Background /1

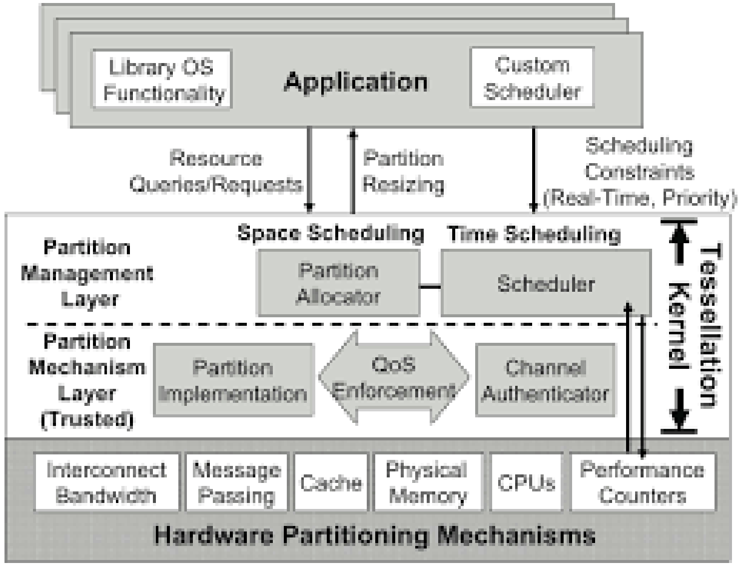
- Critical systems are known as *high-assurance* (high-integrity)
  - System operation must *always* perform as intended, *provably*
  - They used to consist of specialized SW running on dedicated HW
    - Not all components are equally critical, hence not all deserve the high cost of high-assurance development
  - *Isolation* segregates the more trusted from the less trusted
    - Isolation is conservative, prepared to waste resources to warrant integrity
- *Digital transformation* wants greater unitary functional value in critical systems, seeking to reduce waste
  - *Integration* is pragmatic, it wants more value for less resource usage
    - Less trusted components may yield high competitive advantage
- Tension builds between integration and isolation

An example of digital transformation



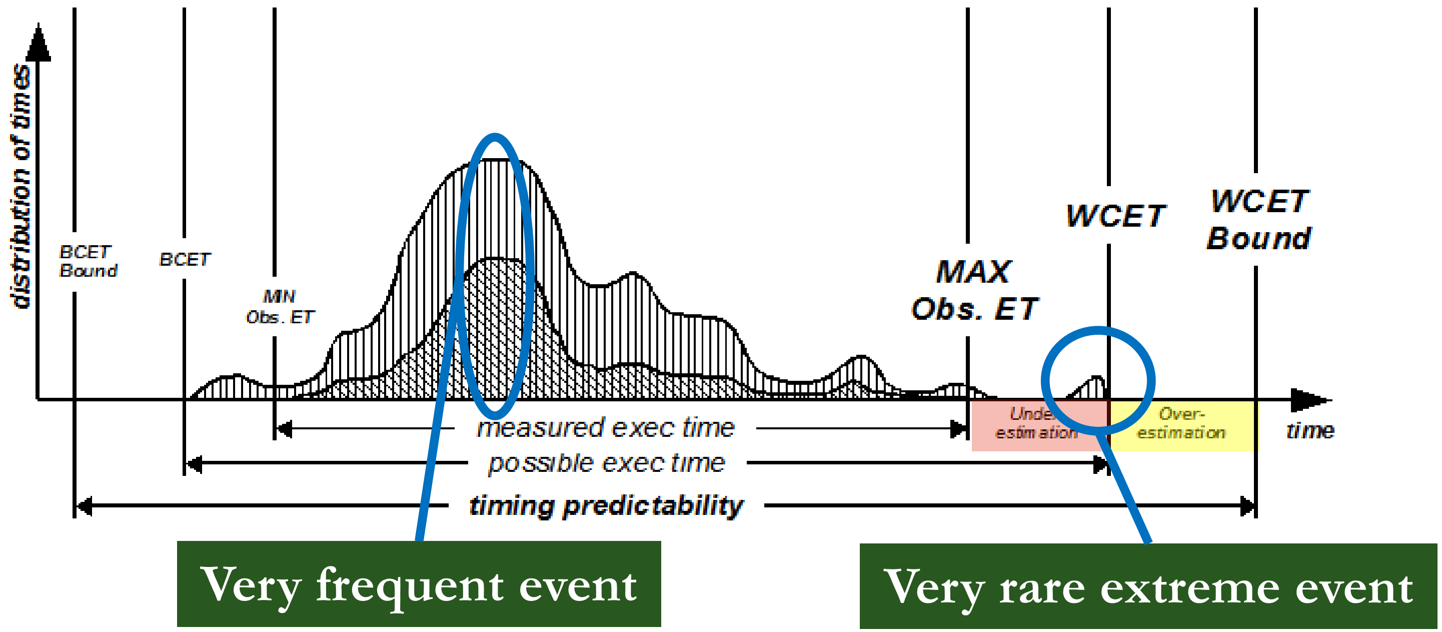
Background /2

- Isolation employs *static allocations*, with *conservative margins* to mitigate the uncertainty of extreme events
  - Conservative margins are wasteful if the worst-case profile has an extreme tail
    - Very far to the right of "normality"



- Baseline approach known as **Time and Space Partitioning**
  - It warrants isolation via a *resource scheduling* hypervisor

The consequence of conservatism



- Budgeting for the (rare) extreme would cost many times *more* than provisioning for the average (frequent) case
- You may not want to budget for the WC statically, but you must be able to sustain it when it happens

Background /3

- Well-behaved integration may reduce waste
  - Tasks with different levels of criticality might be allowed to co-exist under strict *safeguarding* guarantees
  - Main goal is maximum (safe) use of CPU
- Tasks with higher integrity requirements (HI-crit) must be guaranteed up to their WC, but with a *default* allocation that covers only the *high watermark*
  - The central tenet of **Mixed-Criticality Systems** (MCS)
- When a HI-crit job executes above default budget, a *mode-change* event trips, which changes system configuration
  - HI-crit tasks retain their WC guarantees
  - LO-crit tasks are held up until normality returns is restored

Vestal's initial vision of MCS (2007)

- Single-core system, with tasks divided in *criticality-based* groups
  - A *mode* attribute $L_i \in \{LO, \dots, HI\}$ attached to each task τ_i determines its budget allocation
  - HI-crit tasks are given a *high conservative margin* over their *measured* WCET
  - LO-crit tasks have *no* margin
  - *Any* task can use the unclaimed margin, but only HI-crit tasks can claim it
- The RTA for this type of system becomes

$$\boldsymbol{R_i = C_i(L_i) + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_j)}$$

  - Each task τ_i is assumed to contribute its per-criticality (L_i) allocation
  - A feasible system does *not* need a mode change event
- Priority and criticality do *not* coincide
  - We need a *priority assignment scheme* (Audsley's) that serves the MCS intent

Vestal's experimental evidence

$$\boldsymbol{margin} = \frac{allocated - measured}{measured}$$

task	T_i	L_i	measured	allocated	margin
			Workload 1		
P1 40hz	25	B	1.06	1.4	32.1%
P1 20hz	50	B	3.09	3.9	26.2%
P2 20hz	50	B	2.7	2.8	3.7%
P3 20hz	50	B	1.09	1.4	28.4%
P4 40hz	25	A	0.94	1.1	17%
P4 20hz	50	A	1.57	1.8	14.6%
P4 10hz	100	A	1.68	2.0	19%
P4 5hz	200	A	4.5	5.3	17.8%
P5 20hz	50	B	2.94	3.7	25.9%
P5 10hz	100	B	1.41	1.8	27.7%
P5 5hz	200	B	6.75	8.5	25.9%
P6 20hz	50	D	5.4	5.4	0%
P6 5hz	200	D	2.4	2.4	0%
P7 20hz	50	D	0.94	1.3	38.3%
P7 5hz	200	D	1.06	1.5	41.5%
P8 40hz	25	D	2.28	2.3	0.9%
P8 10hz	100	D	4.75	4.8	1.1%
P8 5hz	200	D	12.87	13	1%
P9 10hz	100	D	0.47	0.6	27.7%
PA 20hz	50	C	1.24	1.9	53.2%
PB 20hz	50	D	1.62	2.4	48.1%
utilization			80.4%	93%	21.4%

Non-weighted average

Table 1: Example Multi-Criticalty Workload

method	Workload 1 Δ^*	increase
deadline monotonic priority traditional analysis	1.08	–
deadline monotonic multi-criticality analysis	1.20	11%
multi-criticality Audsley's multi-criticality analysis	1.20	11%
transformed & deadline monotonic multi-criticality analysis	1.20	11%
transformed & multi-criticality Audsley's multi-criticality analysis	1.20	11%

Table 2: Comparative Evaluation Results

- ***Δ**** : **largest simultaneous increase in budget allocation of HI-crit tasks (over *measured* bound) that preserves *overall* feasibility without mode-change events**
- **12% extra margin earned *without* wastage**

Immediate ramifications

- EDF does *not* dominate FPS for systems with criticality levels
  - Feasible systems can be constructed that EDF is unable to schedule
- The MCS model of (constrained-deadline) sporadic tasks may be formalized as $(T^{\rightarrow}, D, C^{\rightarrow}, L)$ where L is a set of criticality levels such that $L_j > L_i \Longrightarrow C(L_j) \geq C(L_i), T(L_j) \leq T(L_i)$
  - *The higher the task's criticality, the larger the guarantee above its default allocation*
  - Most commonly, $L = \{LO, HI\}$ and $T^{\rightarrow} = T$
- **The solution rests on an effective fixed-priority ordering**
  - Apply deadline-monotonic ordering to all HI-crit and LO-crit tasks
  - Test LO-crit tasks from lowest-priority up (Audsley's style)
    - If feasible, it takes that priority
    - Else, try next task; if none is feasible, failure
  - This logic assures best guarantees for HI-crit tasks

Adaptive Mixed Criticality (2012)

- Single-core system, FPS assumed (Baruah, Burns, and Davis)
- To attain higher average utilization, WCET allocation is *not* static
  - When a HI-crit job exceeds its LO-crit budget, a **mode change** alarm trips
  - To safeguard all HI-crit tasks, all LO-crit tasks are temporarily suspended
- Three distinct feasibility conditions

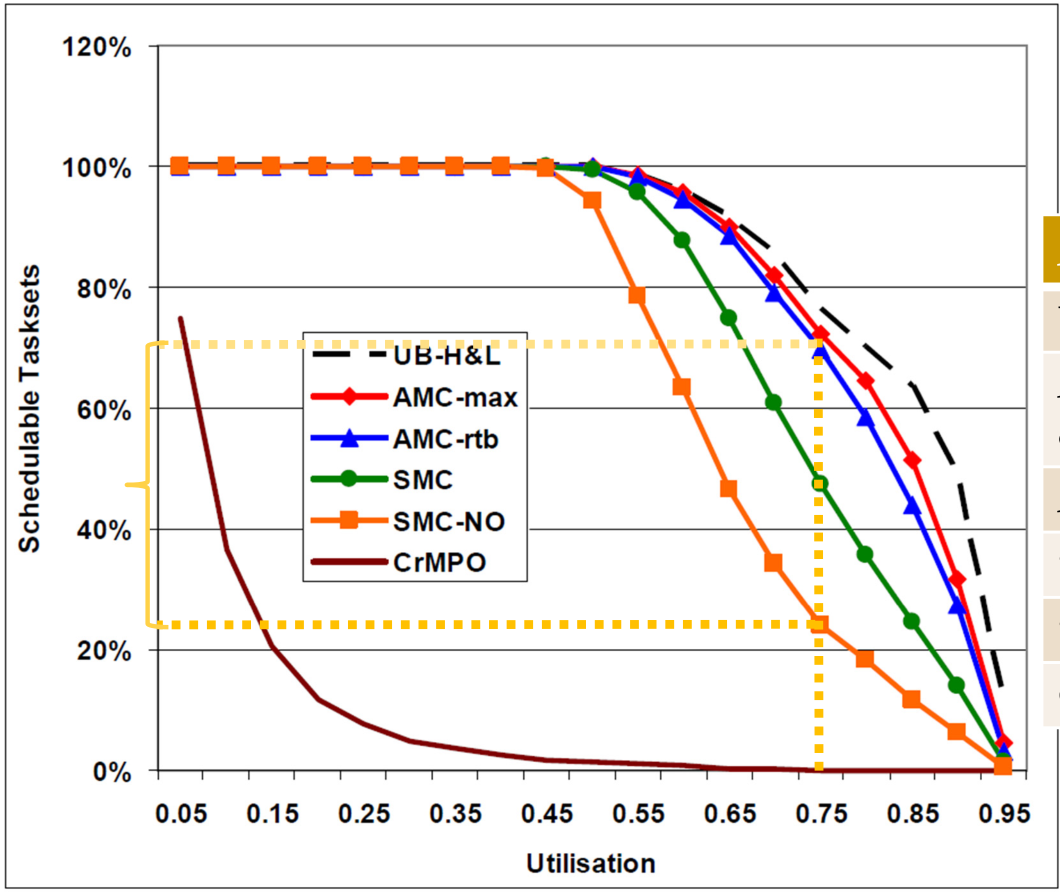
LO-crit mode: $R_i(Lo) = C_i(Lo) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(Lo)}{T_j} \right\rceil C_j(Lo)$

HI-crit mode: $R_i(Hi) = C_i(Hi) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(Hi)}{T_j} \right\rceil C_j(Hi)$

LO-2-HI mode: $R_i^* = C_i(Hi) + \sum_{j \in hpH(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j(Hi) + \sum_{k \in hpL(i)} \left\lceil \frac{R_i(Lo)}{T_k} \right\rceil C_k(Lo)$

  - Pessimistically assuming LO-crit tasks to contribute their maximum interference *before* being suspended

Asserted benefits



Approaches
UB-H&L: theoretical upper bound
AMC-max: adaptive mixed-criticality (minor tweak over base AMC)
AMC-rtb: adaptive mixed-criticality (base method)
SMC: as Vestal, but with mode-change monitoring
SMC-NO: Vestal's original approach
CrMPO: priorities assigned in order of criticality

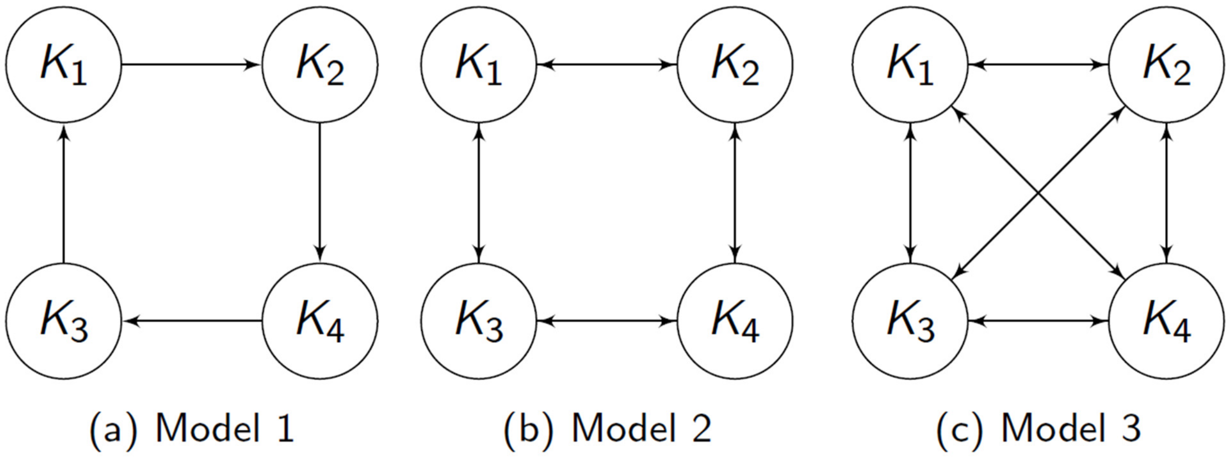
20 tasks per taskset, an average of 50% tasks assumed Hi-crit, $C(Hi) = 2 \times C(Lo)$

What with multicores?

- Higher functional value accrued with LO-crit tasks allowed to migrate instead of being suspended
  - This requires partitioned scheduling and *per-core* criticality mode
  - HI-crit tasks statically assigned to a core
  - LO-crit tasks feasible in (per-core) *HI-crit mode* are statically assigned
  - LO-crit tasks that would be abandoned on one core and could fit feasibly on another core, are allowed to migrate to it
  - Residual LO-crit tasks marked "*expendable*"
- Only a small fraction of cores is assumed to enter HI-crit mode simultaneously
  - The system should be kept feasible *up to that limit*
- Solution in three mutually-dependent parts
  - Partition tasks, determine allowable migrations, assign priorities

Xu & Burns (2019) /1

- 3 models of migration for a quad-core processor



  - **Model 1**: each core has one migration route
  - **Model 2**: each core has two migration routes
  - **Model 3**: each core allows migration to all other cores

Xu & Burns (2019) /2

1. Order tasks by decreasing criticality
2. Use (First-Fit, Best-Fit, Worst-Fit) bin-packing for task-to-core assignment
   - WF empirically proven better
3. Use Audsley's algorithm to assign per-core priorities
   - If HI-crit task not feasible on one core, try it on another core
   - If HI-crit task cannot be feasibly assigned, then **failure**
   - If LO-crit task not feasible on core, pick highest-priority LO-crit task feasible on that core and try a *migration route* for it (method SEMI-2)
   - If that fails, try next LO-crit task down: if any LO-crit task remains unassigned, mark it expendable

- The system needs to be studied *before* and *after* mode change
  - Dependent on how many cores can enter HI-crit mode simultaneously
  - We look at the *1-mode-change* case only: the others can be built analogously

Xu & Burns (2019) /3

- **Before** mode change (*steady mode*), core K_s hosts some HI-crit tasks, some LO-crit tasks, and some LO-crit "*can migrate*" tasks

$$\boldsymbol{R_i(Lo) = C_i(Lo) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(Lo)}{T_j} \right\rceil C_j(Lo)}$$

- **After** mode change ($L_i > Lo$) in core K_s, with migration route to core K_t
  - Core K_s sheds its "*can migrate*" LO-crit tasks ($\boldsymbol{M_{K_s}}$), which contribute their maximum interference before going

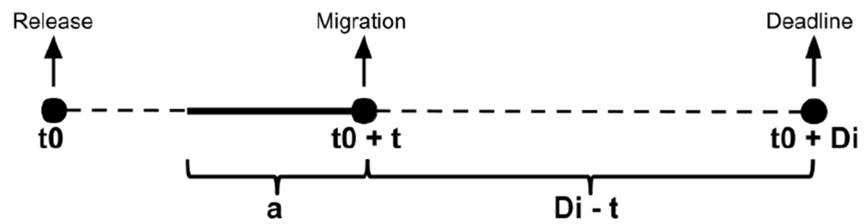
$$\boldsymbol{R_i(L_i) = C_i(L_i) + \sum_{j \in hp(i), K_s} \left\lceil \frac{R_i(L_i)}{T_j} \right\rceil C_j(L_j) + \sum_{\omega \in hp(i), M_{K_s}} \left\lceil \frac{R_i(Lo)}{T_\omega} \right\rceil C_\omega(Lo)}$$

Xu & Burns (2019) /4

- **After** mode change, in core K_t with migration from core K_s
  - Core K_t will have to schedule the incoming LO-crit tasks

$$\boldsymbol{R_i(Lo) = C_i(Lo) + \sum_{j \in hp(i), K_t} \left\lceil \frac{R_i(Lo) + J_j}{T_j} \right\rceil C_j(Lo)}$$

  - Any "*can migrate*" task τ_j will carry residual work $(C_j - a)$ with relative deadline $(D_j - t)$ to core K_t



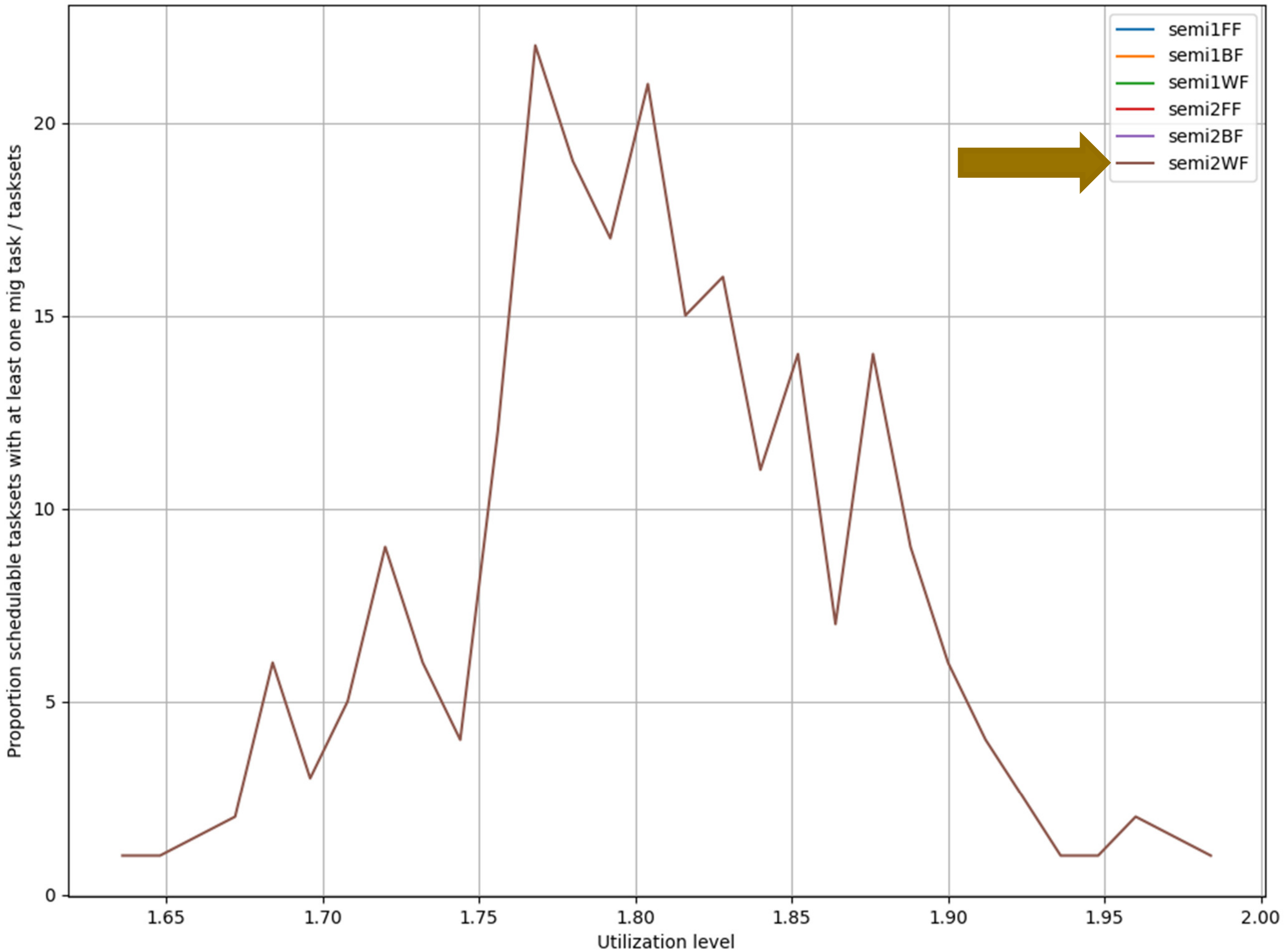
  - In the worst case, any such task τ_j will suffer maximum release jitter

$$J_j \leq R_j(Lo) - C_j(Lo)$$

Performance evaluation /1

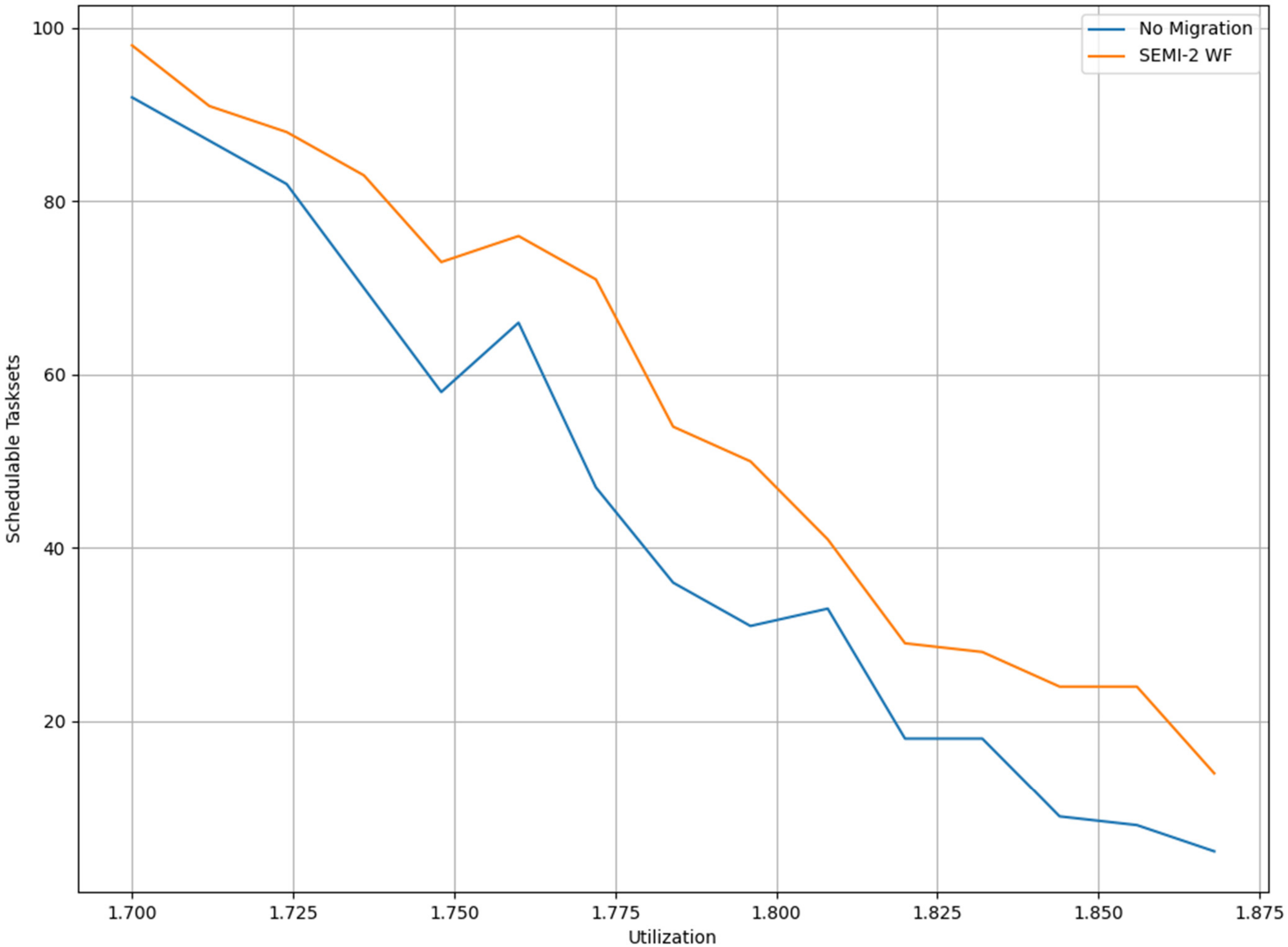
- **Reproducibility** of (Xu & Burns, 2019) RTA-based simulations
  - How far migration (SEMI-2 WF) dominates no migration (AMC) for percentage of feasible tasksets
- **Realism** of proposed solution for a 2-core processor
  - Real execution experiments for RTA-feasible tasksets *with* migration
  - How many of them remain feasible
  - How many runs disrupted by (budget exceeded, deadline missed) events
    - Their occurrence tells system should be made even more sensitive
- Over varying control parameters
  - Log-uniform *period distribution* in the $[10, 1000]$ ms range, within bounded hyperperiod
  - *Taskset cardinality* in the $[20, 35]$ range per core
  - *Task utilization* in the $[0.05, 0.6]$ range generated with the Dirichlet-Rescale algorithm (Griffin, Bate, Davis, 2020)

Performance evaluation /2



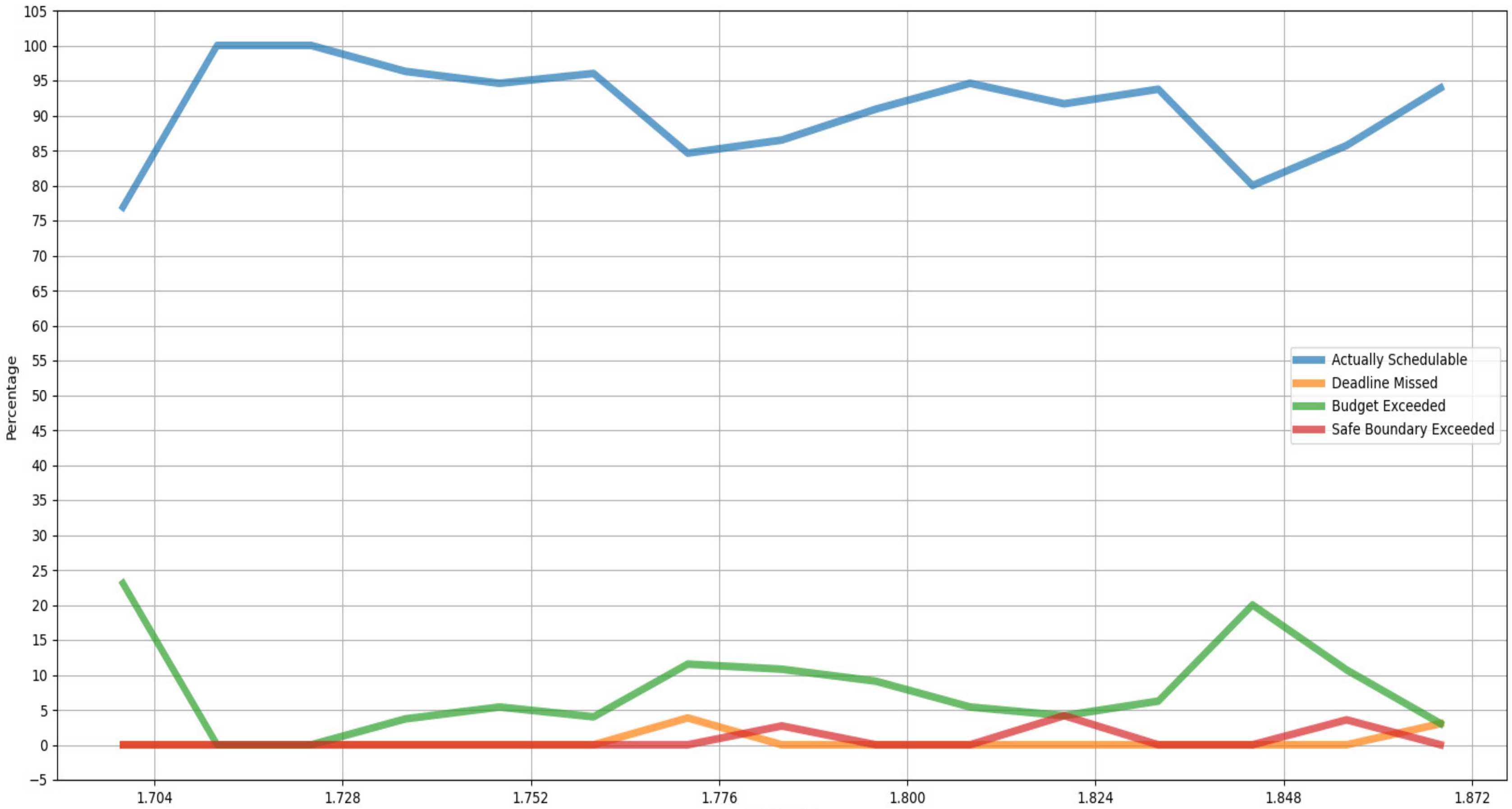
Ratio of LO-crit to HI-crit: 2. Taskset size: 12. Max harmonic: 2. Small periods: $[10, 200]$ ms. Large periods: $[400, 1000]$ ms.

Performance evaluation /3



Ratio of LO-crit to HI-crit: 2. Taskset size: 12. Max harmonic: 2. Small periods: $[10, 200]$ ms. Large periods: $[400, 1000]$ ms.

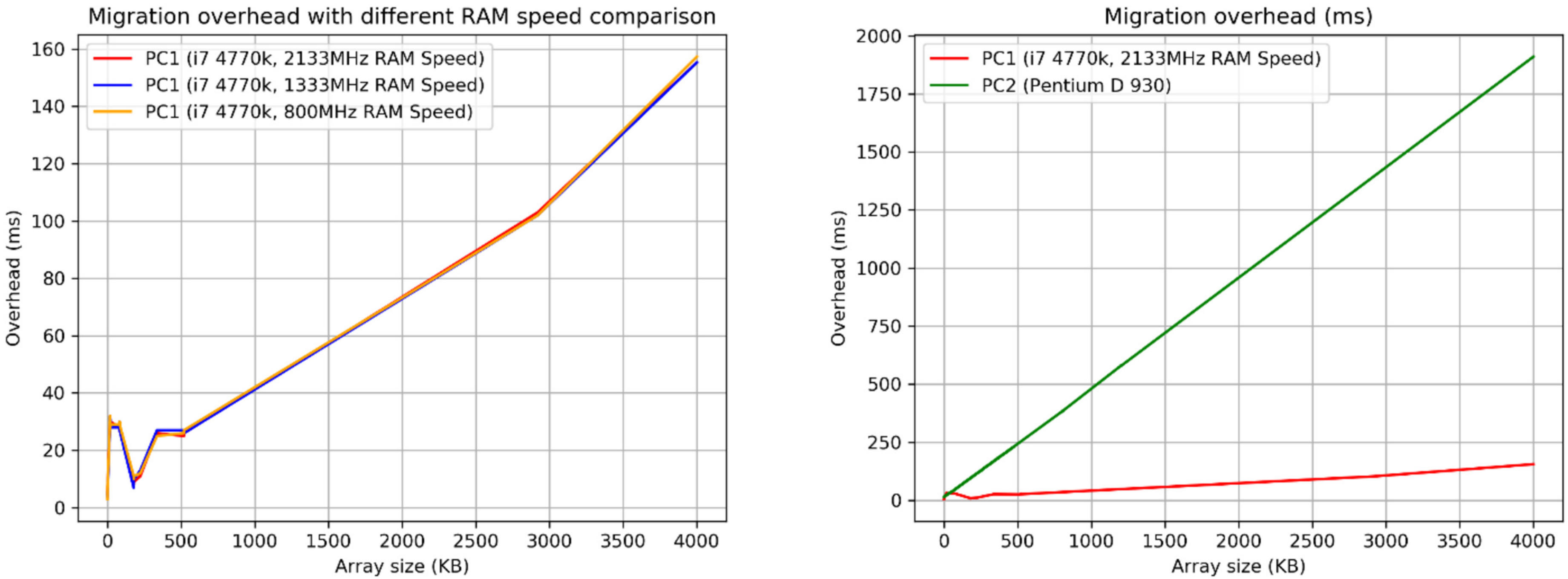
Performance evaluation /4



Ratio of LO-crit to HI-crit: 2. Taskset size: 12. Max harmonic: 2. Small periods: $[10, 200]$ ms. Large periods: $[400, 1000]$ ms.

Migration costs are not negligible

PC1, 4-core i7, 64KB L1 cache, 256KB L2 cache, 8MB L3 shared cache
PC2, con 2-core, 16KB L1 cache, 2MB L2 cache L2



Minimal Linux, 10k R/W random access ops on variable-size array (0.4 kB – 4 MB in 0.4 kB increments), job migration every even iteration

Summary

- Digital transformation wants real-time systems to embed an ever increasing number of value-added software functions
  - Some such functions are of *high criticality* and must be assured
  - Other functions are less critical, but we want to deploy them in the same processor as the other ones to accrue more functional value per unit of computation
- This need has originated *mixed-criticality systems*
  - We have examined approaches that give sufficient assurance of *time isolation* while achieving high schedulable utilization

Selected readings

- S. Vestal (**2007**)
  Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance
  DOI: 10.1109/RTSS.2007.47
- H. Xu, A. Burns (**2019**)
  A semi-partitioned model for mixed criticality systems
  DOI: 10.1016/j.jss.2019.01.015
- M. Bottaro (**2021**)
  Exploring the viability of a MCS multicore runtime demonstrator
  Work in progress