

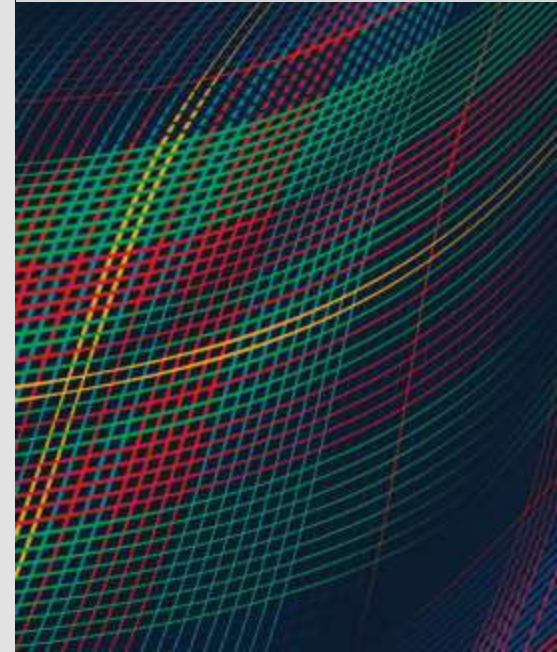
# Methodology of Combining Empirical Stress Testing and Formal-Methods Based Schedulability Analysis for Real- Time Multicore Software

**Carnegie Mellon University:**  
Bjorn Andersson, Dionisio de Niz

**OCTOBER 2023**

© 2023 Carnegie Mellon University

**Distribution A:  
Approved for Public Release.  
Distribution is Unlimited**



Copyright 2023 Carnegie Mellon University, U.S. Army Combat Capabilities Development Command Aviation & Missile Center.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

DM23-0820

# The need for safety (of aircraft)

**Ariane 5 Flight 501:** Catastrophic failure (causing self-destruct) in flight in 1996 due to incorrect software reuse.

**Boeing 737 MAX:** Two crashes (in 2018 and 2019) due to software flaw in the MCAS (Maneuvering Characteristics Augmentation System).

**Airbus A400M:** Crash during a test flight in Spain 2015 due to software flaw.

**Helicopters:** Army Helicopter crashes in April 2023: Apache in Alaska and Black Hawk in Kentucky.

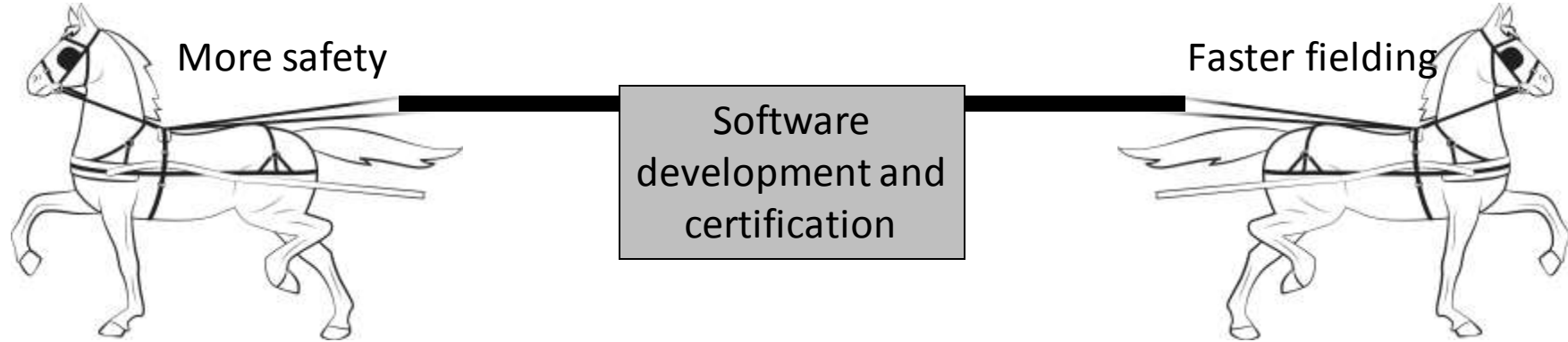
Some crashes are caused by software. The increase complexity of software will make the impact of software on safety more important.



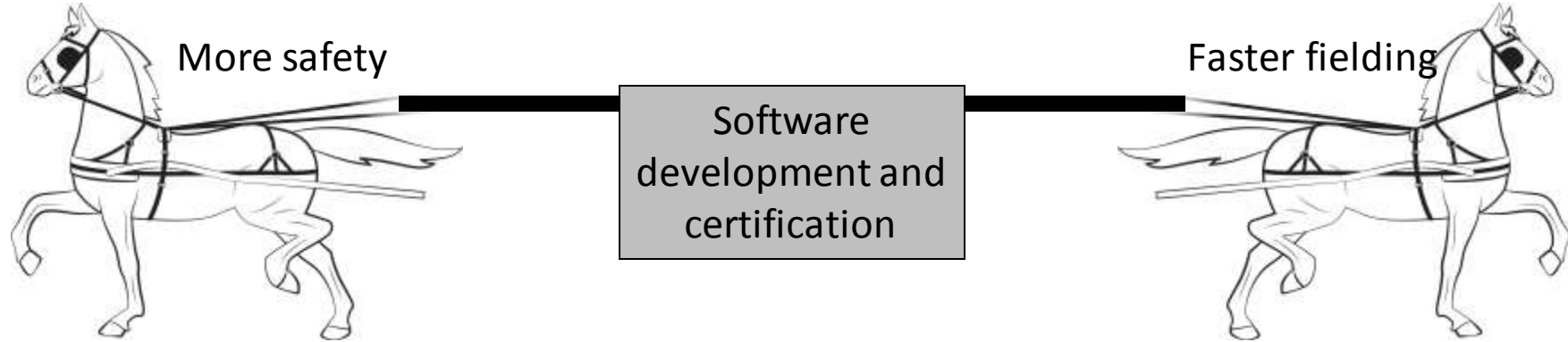
# The need for speed (of software development of aircraft)



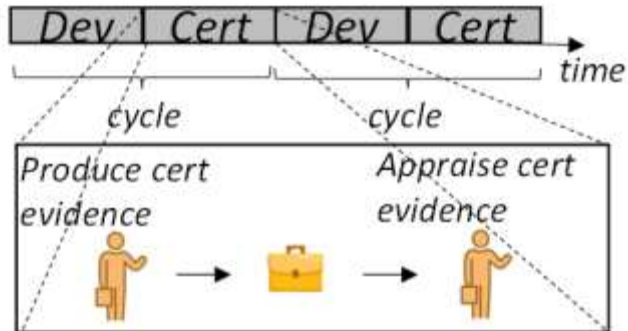
The Department will instead reward rapid experimentation, acquisition, and fielding.

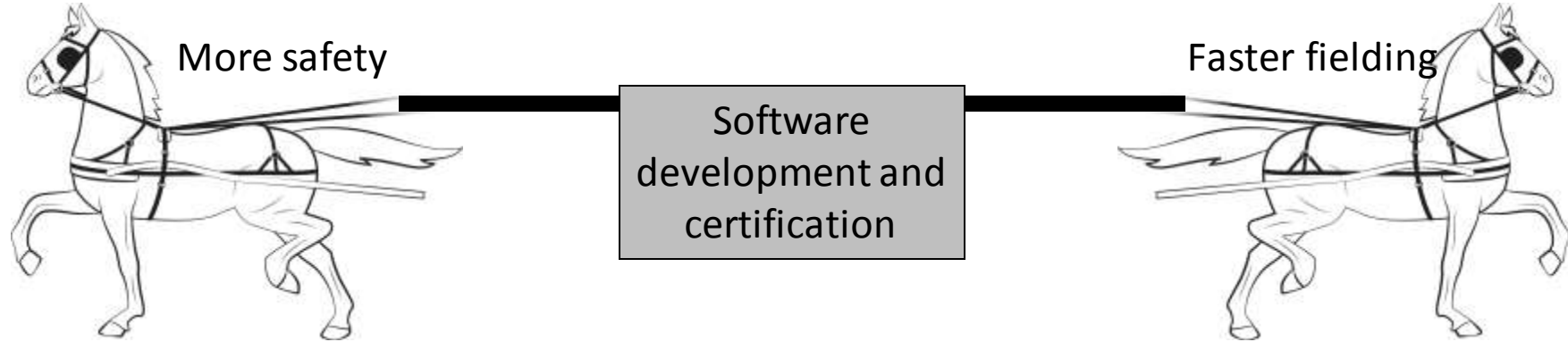


How do we combine the need for safety with the need for speed?

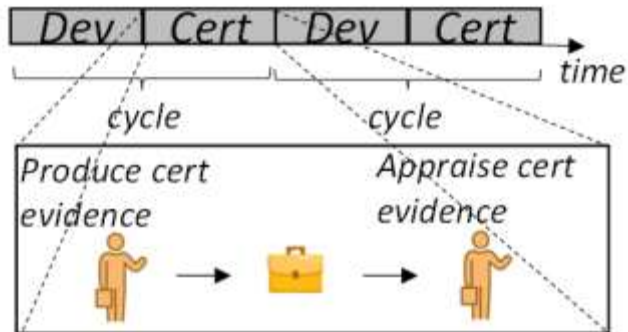


How do we combine the need for safety with the need for speed?

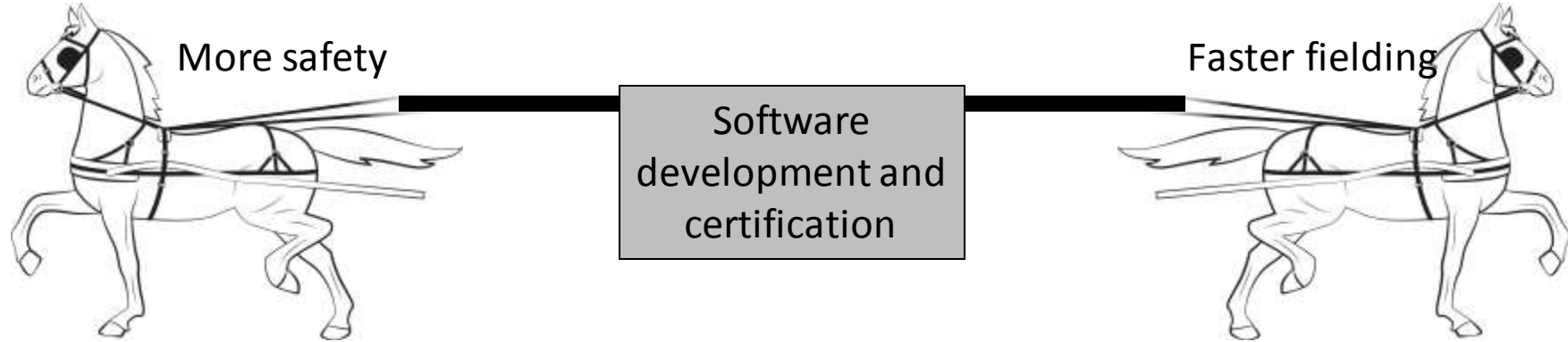




How do we combine the need for safety with the need for speed?



Even if we decrease the development time to zero, then certification time is still there and becomes a bottleneck for our ability to field new systems rapidly.



How do we combine the need for safety with the need for speed?

*Test data as cert evidence*

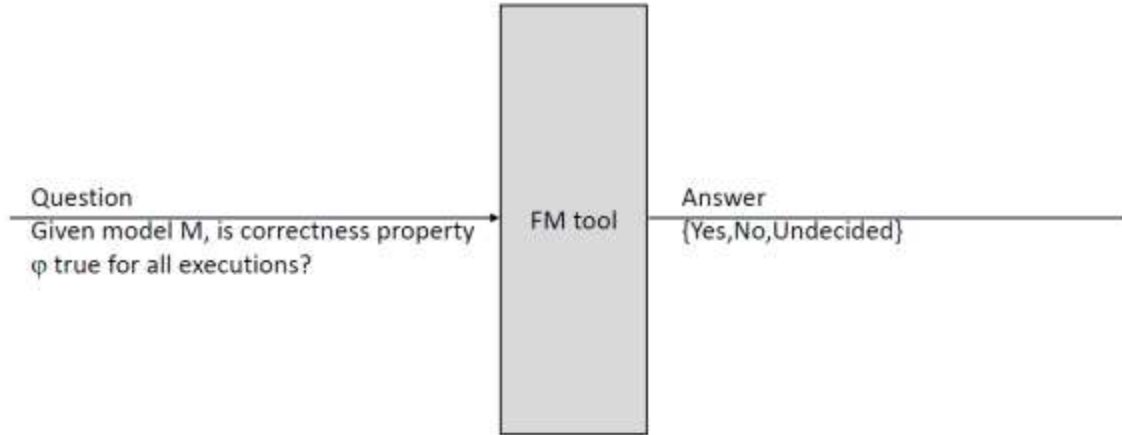
- *Exhaustive takes too long*
- *Non-exhaustive is unsafe*



Formal methods have the potential to avoid these



# Formal methods



Input: (i) a model of a system and (ii) a correctness condition.

Output: True/False/Undecided

# Different correctness properties

Logical correctness:

Compute  $2+3$ .

The result should be 5.

Temporal correctness:

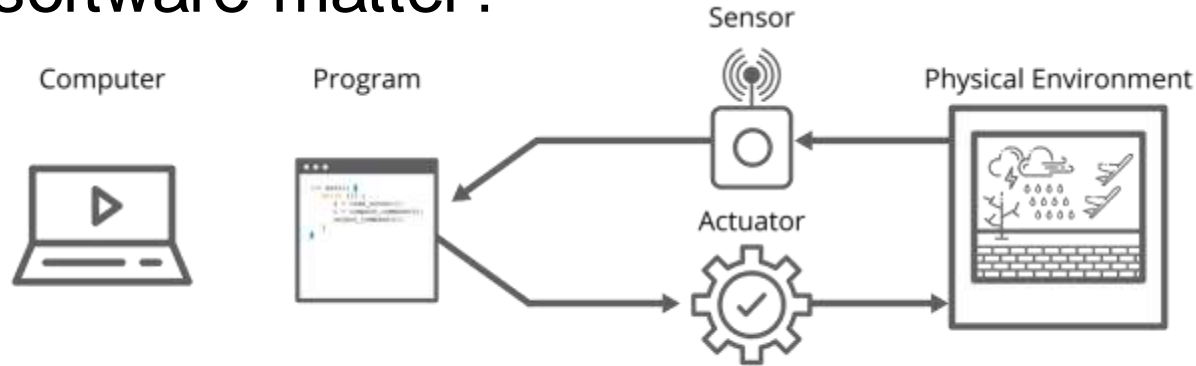
Compute  $2+3$ .

The time from when the computation is requested until its result delivered should be at most 20 milliseconds.



Focus of this talk

# Why timing of software matter?



Interaction of the software with the physical environment imposes timing requirements on software.

“Timing problems are one of the common causes of run-time failures in process-control systems, and timing is often inadequately specified.”

M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl, and B.E. Melhart, "Software requirements analysis for real-time process-control systems," IEEE TSE, 1991.

# Why timing of software matter?

Army avionics interact with physical environment. The world does not stand still.

“The trick there, when you’re processing flight critical information, it has to be a deterministic environment, meaning we know exactly where a piece of data is going to be exactly when we need to — no room for error,” Langhout says. “On a multi-core processor there’s a lot of sharing going on across the cores, so right now we’re not able to do that.”

- Jeff Langhout, Acting Director, U.S. Army Aviation and Missile Research Development and Engineering Center (AMRDEC)

Source: “Army still working on multi-core processor for UH-60V,” May 2017, Available at <https://www.flightglobal.com/news/articles/army-still-working-on-multi-core-processor-for-uh-6-436895/>.

# Why satisfying timing requirements is challenging?

Satisfy for all scenarios

Depends on underlying hardware platform—not just software

Depends on external physical world

Event-driven

Undocumented hardware

Multitasking, Inter-core interference

# Concurrent Programming

```
int thread1() {  
    perform_initialization_thread_1();  
    while (1) {  
        wait_for_event_thread1();  
        s = read_sensor_thread1();  
        o = perform_processing_thread1(s);  
        actuate_command_thread1(o);  
    }  
}
```

# Concurrent Programming

```
int thread1() {  
    perform_initialization_thread_1();  
    while (1) {  
        wait_for_event_thread1();  
        s = read_sensor_thread1();  
        lock_semaphore(sem);  
        d = read_shared_data_thread1();  
        unlock_semaphore(sem);  
        o = perform_processing_thread1(s,d);  
        actuate_command_thread1(o);  
    }  
}
```





# Concurrent Programming

```
int thread1() {
    perform_initialization_thread_1();
    while (1) {
        wait_for_event_thread1();
        s = read_sensor_thread1();
        lock_semaphore(sem);
        d = read_shared_data_thread1();
        unlock_semaphore(sem);
        o = perform_processing_thread1(s,d);
        actuate_command_thread1(o);
    }
}
```

deadline

Many sources of delay:

Thread's own execution

The thread may get preempted

The thread may block on semaphore

Thread may experience inter-core interference (multicore)

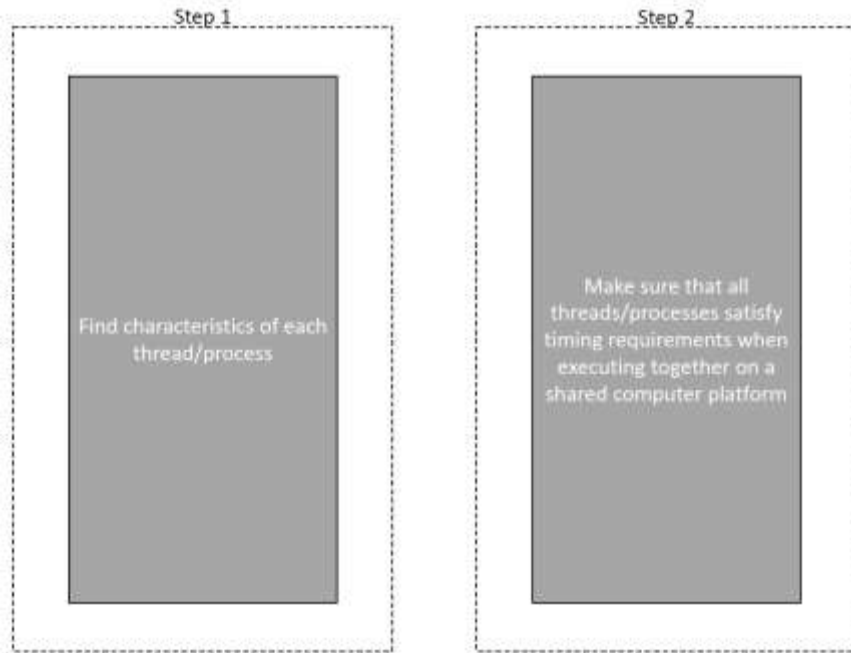
Q: How to verify timing of software executing on multicore?

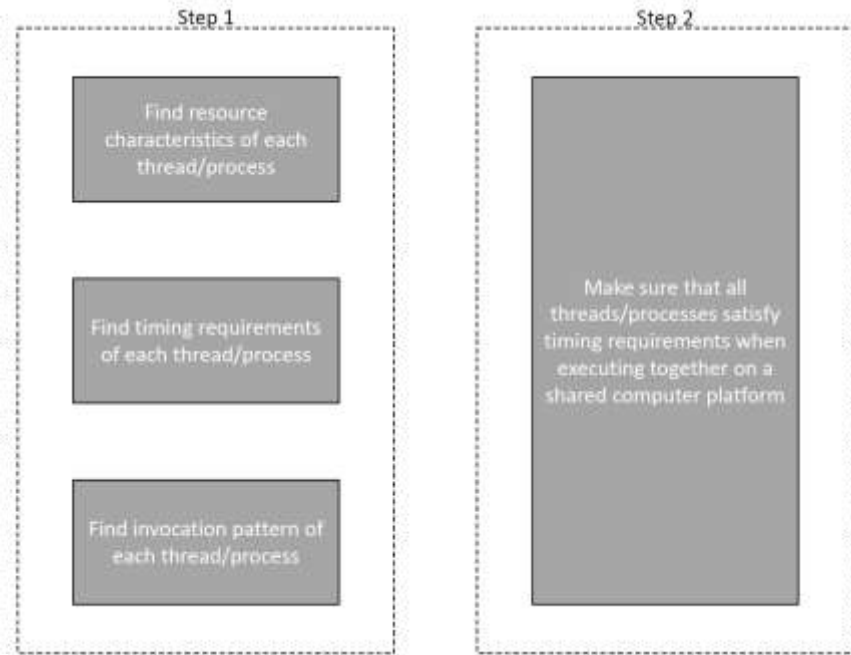
A: Use a two-step framework where some activities may use formal methods and some may not.



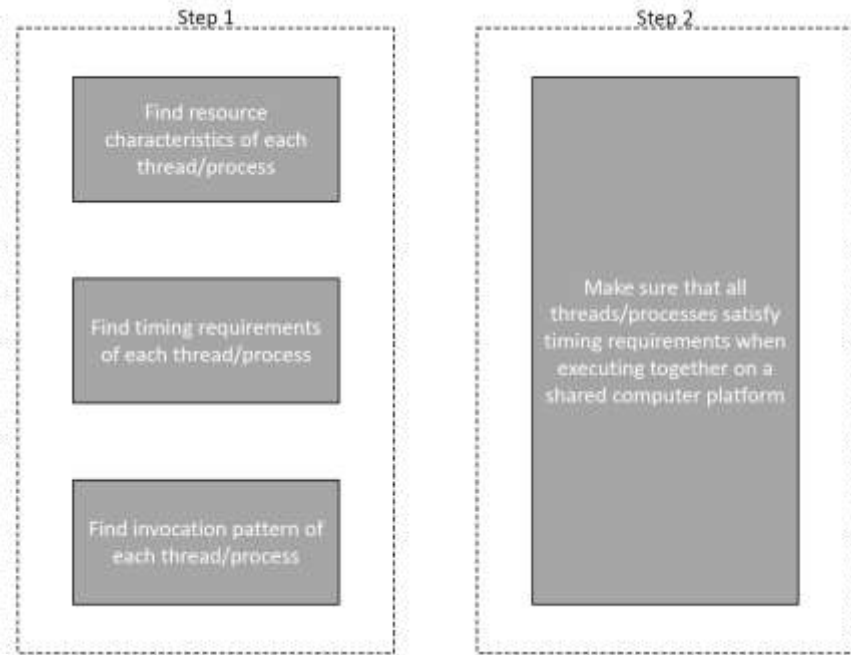
Focus of this talk

Well-known in the research literature

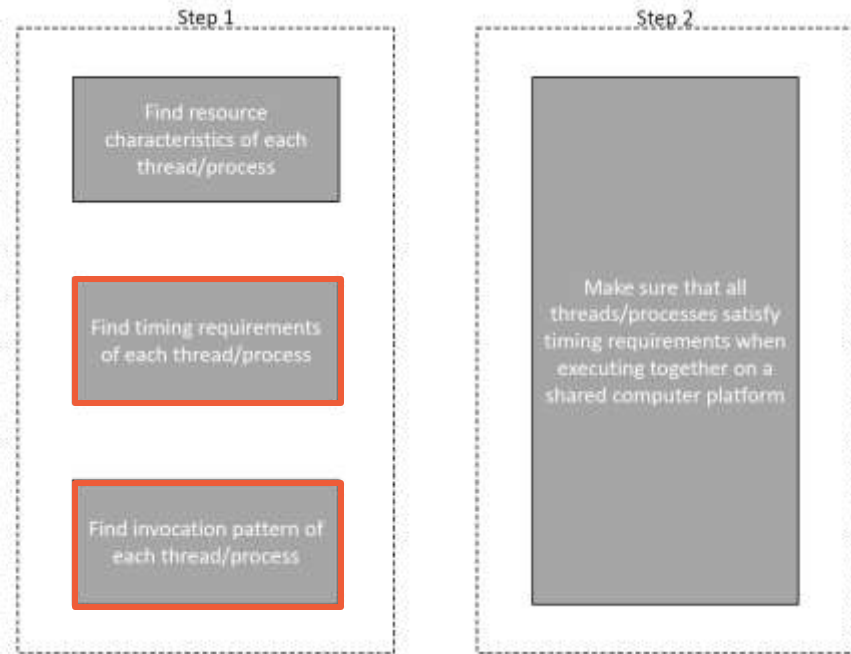




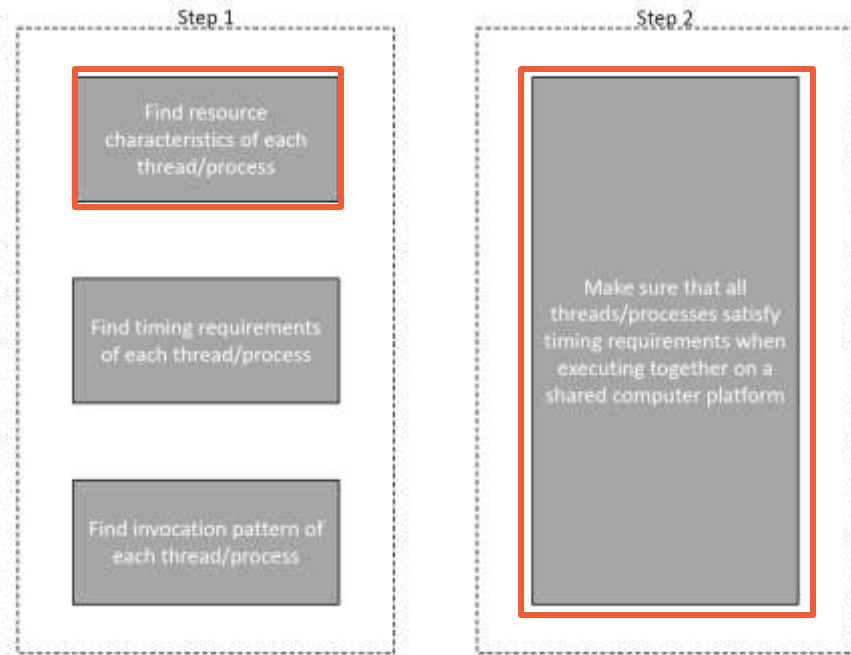
# Are there tools for these activities?



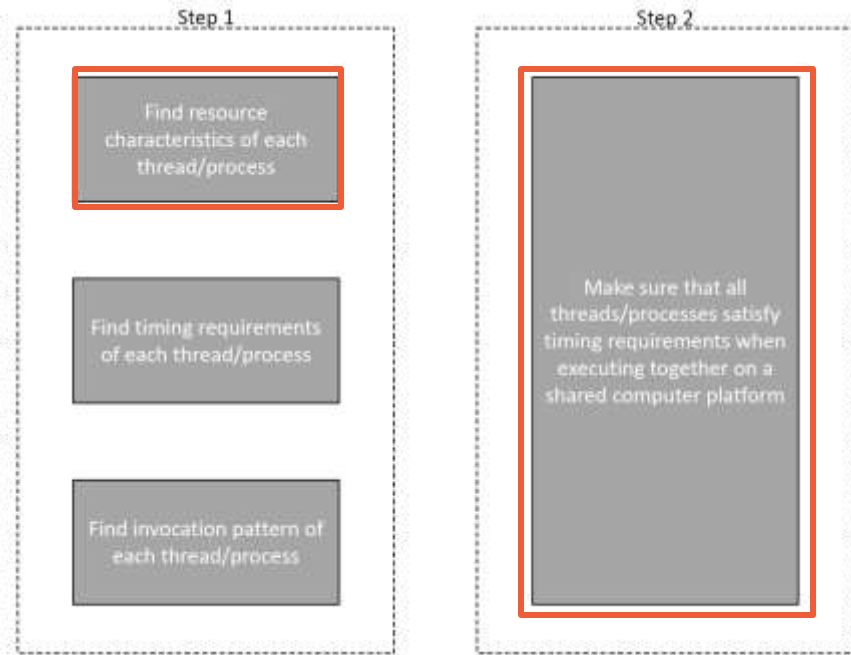
# Read design documents and source code.



# Are there tools for these activities?

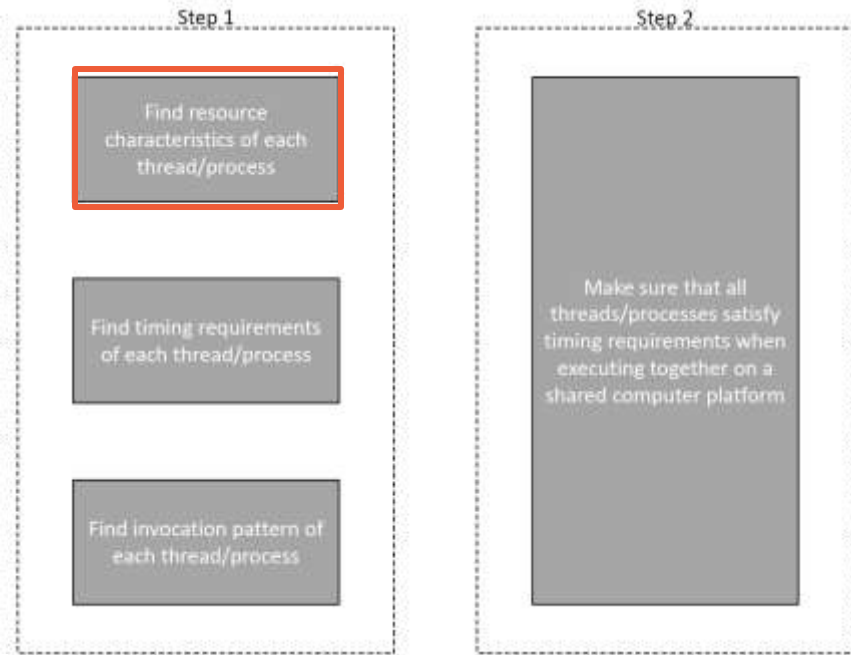


# Rich literature. Let us see some of SEI's and AvMC's tools.

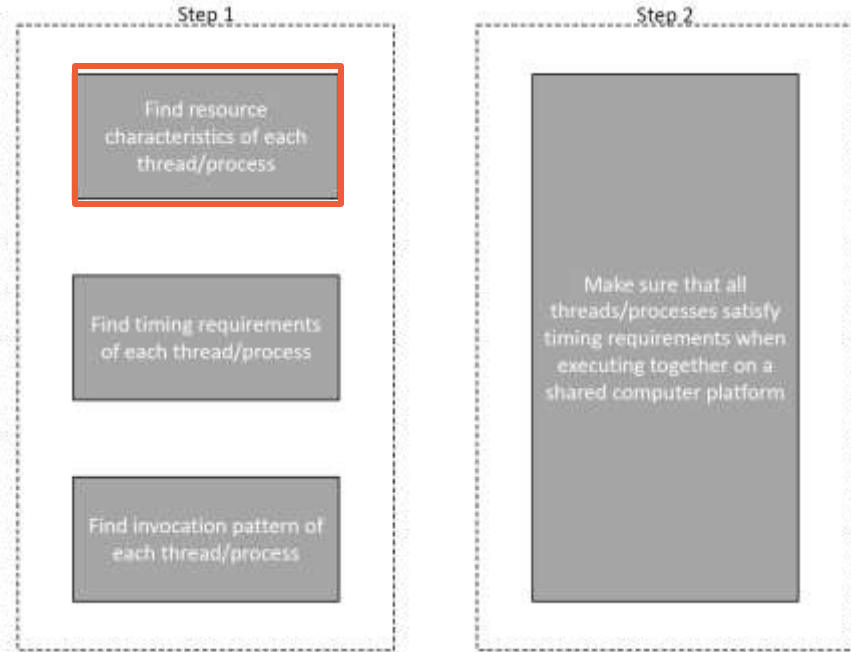




# Single-core. Find Estimate of Worst-Case Execution Time.



# Single-core. Find Estimate of Worst-Case Execution Time.



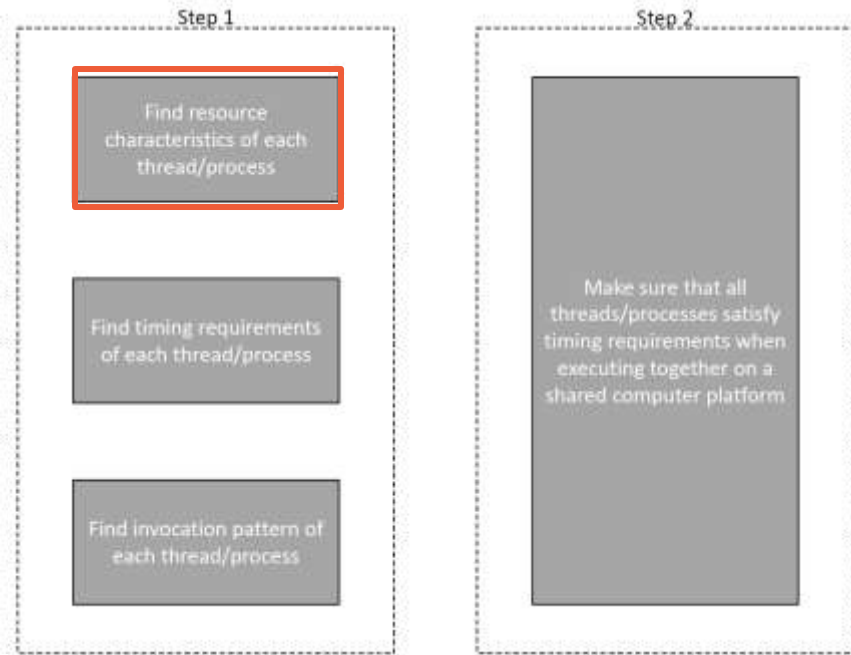
# Worst-Case Execution Time Analysis

```

ba@ba-desktop: ~/ga_find_wcet
File Edit View Search Terminal Help
ba@ba-desktop:~/ga_find_wcet$ more myconf
0
262144
ba@ba-desktop:~/ga_find_wcet$ ./ga_find_wcet myconf ./bubblesort
0.306359
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < ascending_integers.dat
real    0m0.002s
user    0m0.001s
sys     0m0.001s
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < descending_integers.dat
real    0m0.275s
user    0m0.275s
sys     0m0.000s
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < worstcaseinput_GA_inputfile.dat
real    0m0.308s
user    0m0.308s
sys     0m0.000s
ba@ba-desktop:~/ga_find_wcet$
  
```

# Genetic Algorithms. End-to-End Measurements.

# Single-core. Find Estimate of Worst-Case Execution Time.



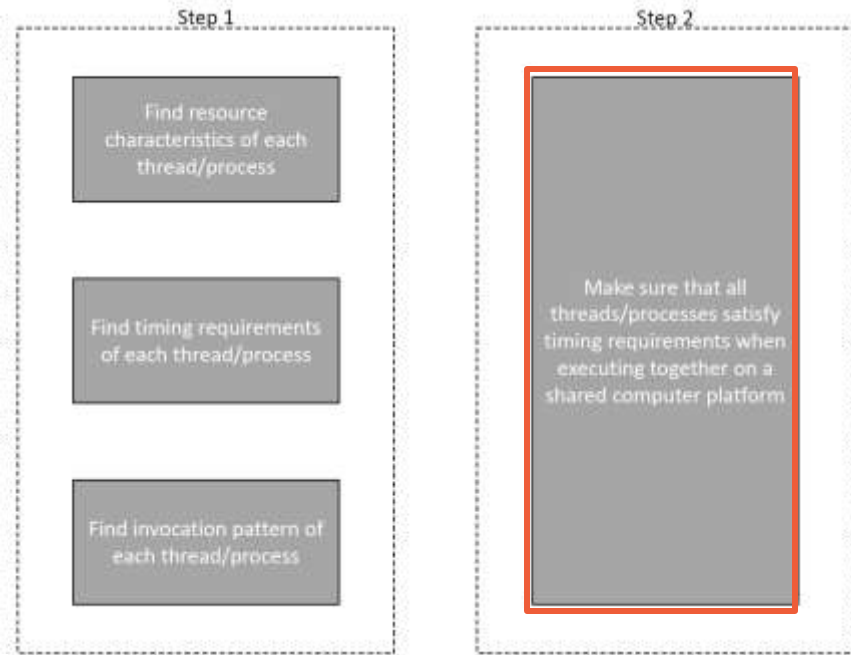
# Worst-Case Execution Time Analysis

```

ba@ba-desktop: ~/ga_find_wcet
File Edit View Search Terminal Help
ba@ba-desktop:~/ga_find_wcet$ more myconf
0
262144
ba@ba-desktop:~/ga_find_wcet$ ./ga_find_wcet myconf ./bubblesort
0.306359
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < ascending_integers.dat
real    0m0.002s
user    0m0.001s
sys     0m0.001s
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < descending_integers.dat
real    0m0.275s
user    0m0.275s
sys     0m0.000s
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < worstcaseinput_GA_inputfile.dat
real    0m0.308s
user    0m0.308s
sys     0m0.000s
ba@ba-desktop:~/ga_find_wcet$
  
```

[https://www.andrew.cmu.edu/user/banderss/software/ga\\_find\\_wcet/ga\\_find\\_wcet.c](https://www.andrew.cmu.edu/user/banderss/software/ga_find_wcet/ga_find_wcet.c)

# Single-core. Response-Time Analysis.

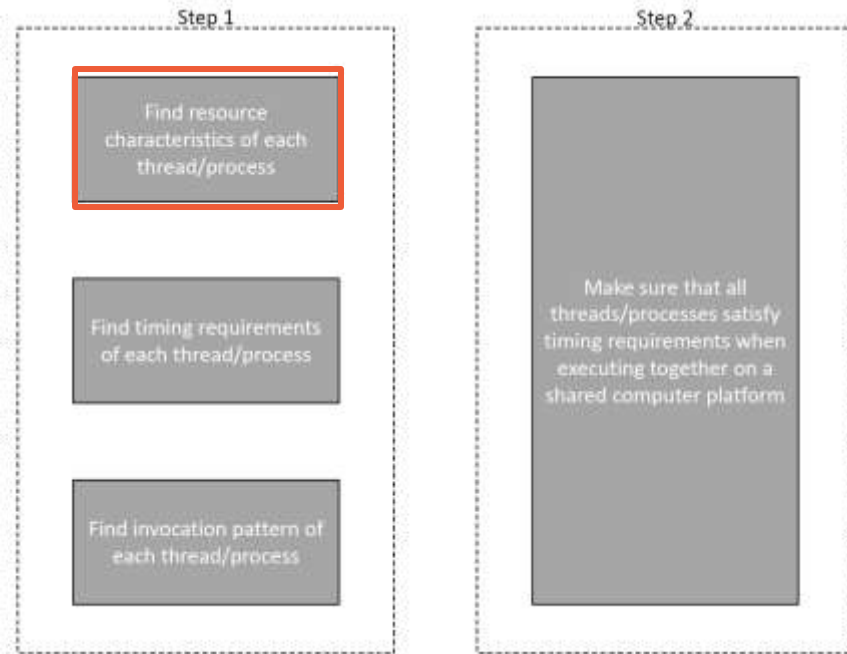


Delay from arrival of thread  $i$  until it finishes  
 = time for its own execution  
 +  
 time for higher-priority threads' execution

Compute upper bound on this.

# Multicore. Find Estimate of Worst-Case Slowdown

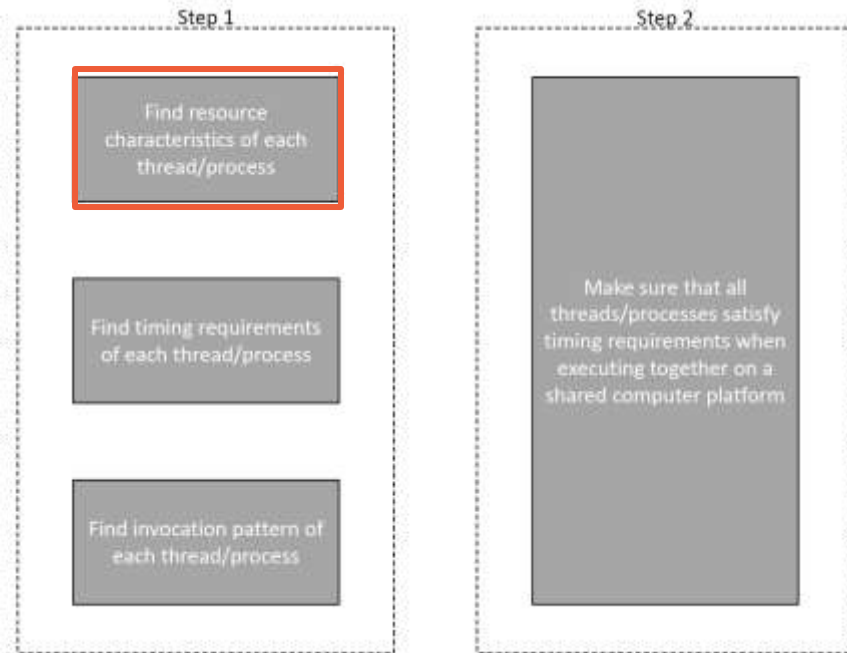
## Tool



[https://www.andrew.cmu.edu/user/banderss/software/pyschedanalysisrunner\\_including\\_ga\\_based\\_parameter\\_extraction/pyschedanalysisrunner\\_including\\_ga\\_based\\_parameter\\_extraction.py](https://www.andrew.cmu.edu/user/banderss/software/pyschedanalysisrunner_including_ga_based_parameter_extraction/pyschedanalysisrunner_including_ga_based_parameter_extraction.py)

# Multicore. Find Estimate of Worst-Case Slowdown

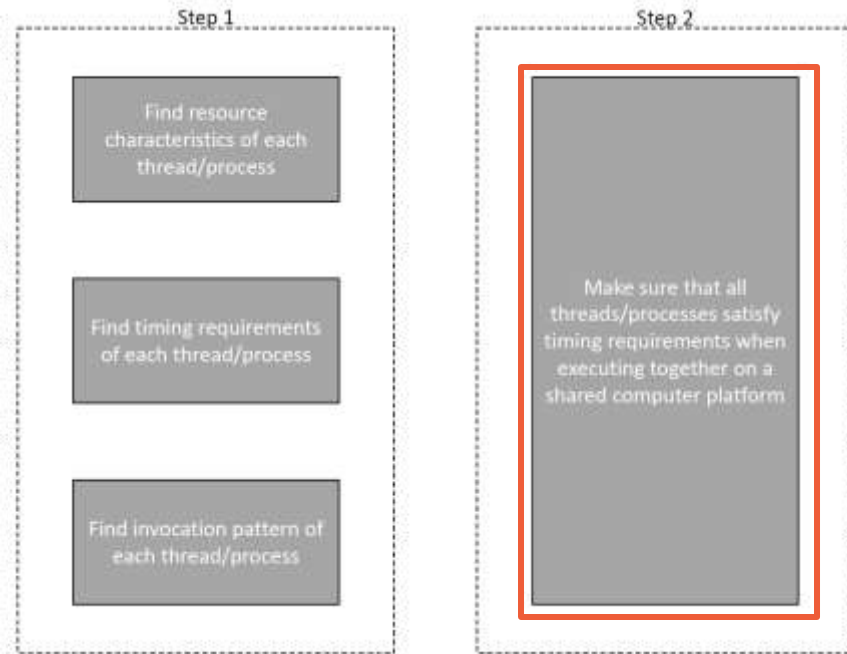
## Tool



Finds slowdown for each thread  $i$ , for each co-runner set of threads  $co$ .

# Multicore. Compute upper bounds on response times

## Tool

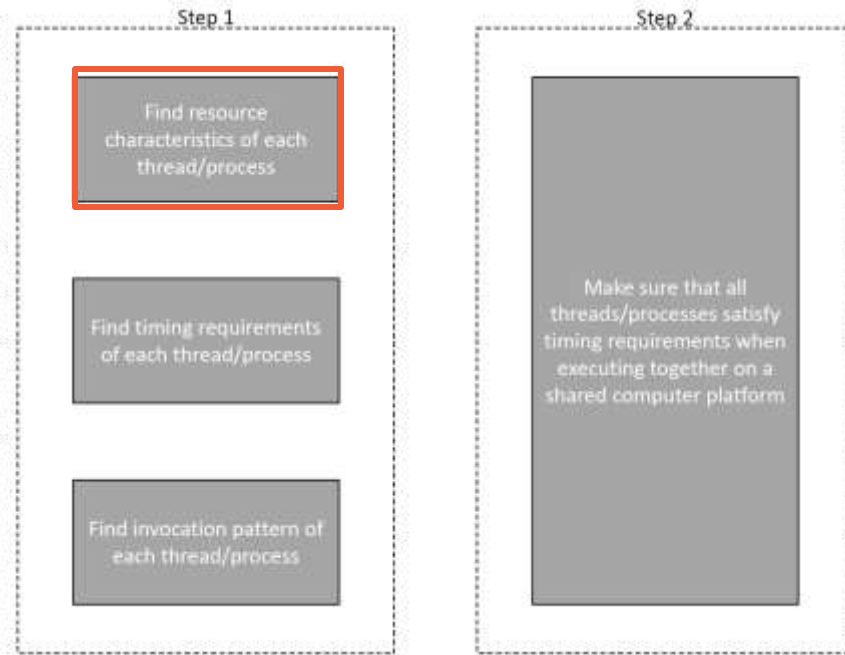


[https://www.andrew.cmu.edu/user/banderss/software/pyschedanalysiscorunner\\_including\\_ga\\_based\\_parameter\\_extraction/pyschedanalysiscorunner\\_including\\_ga\\_based\\_parameter\\_extraction.py](https://www.andrew.cmu.edu/user/banderss/software/pyschedanalysiscorunner_including_ga_based_parameter_extraction/pyschedanalysiscorunner_including_ga_based_parameter_extraction.py)



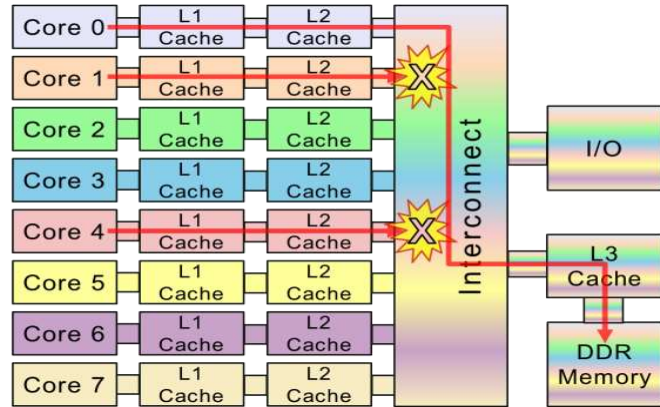
# Multicore. Profiling

# Tool

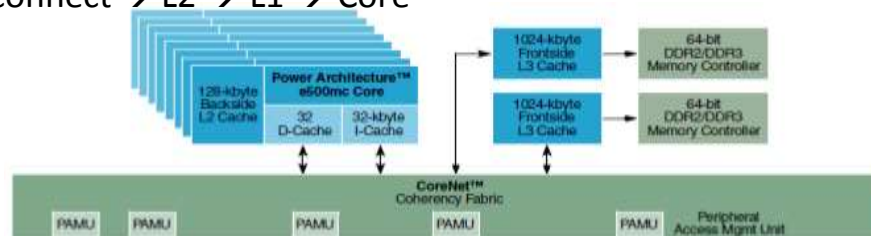
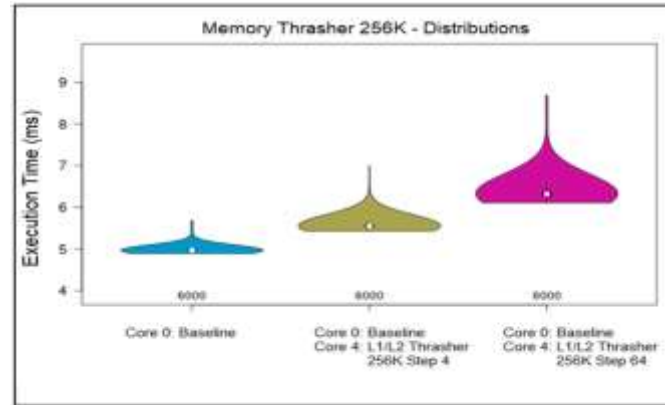




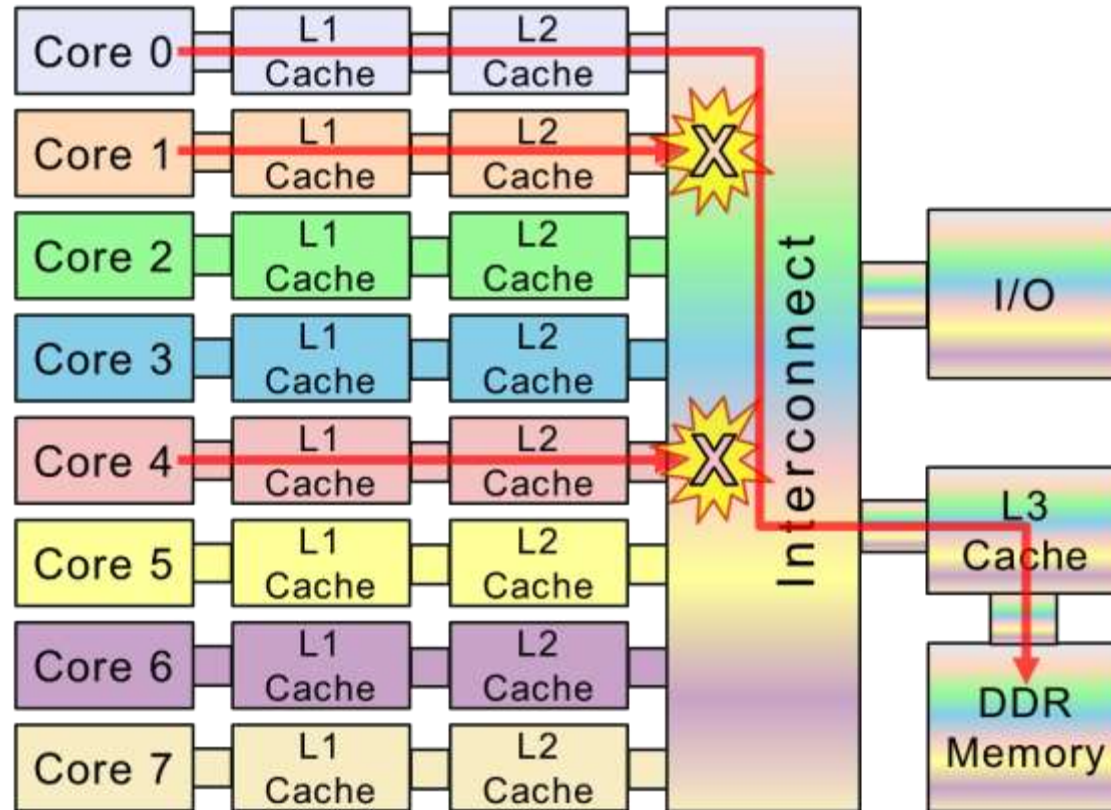
# INTERFERENCES DUE TO CONCURRENT ACCESSSES TO SHARED RESOURCES EXAMPLE



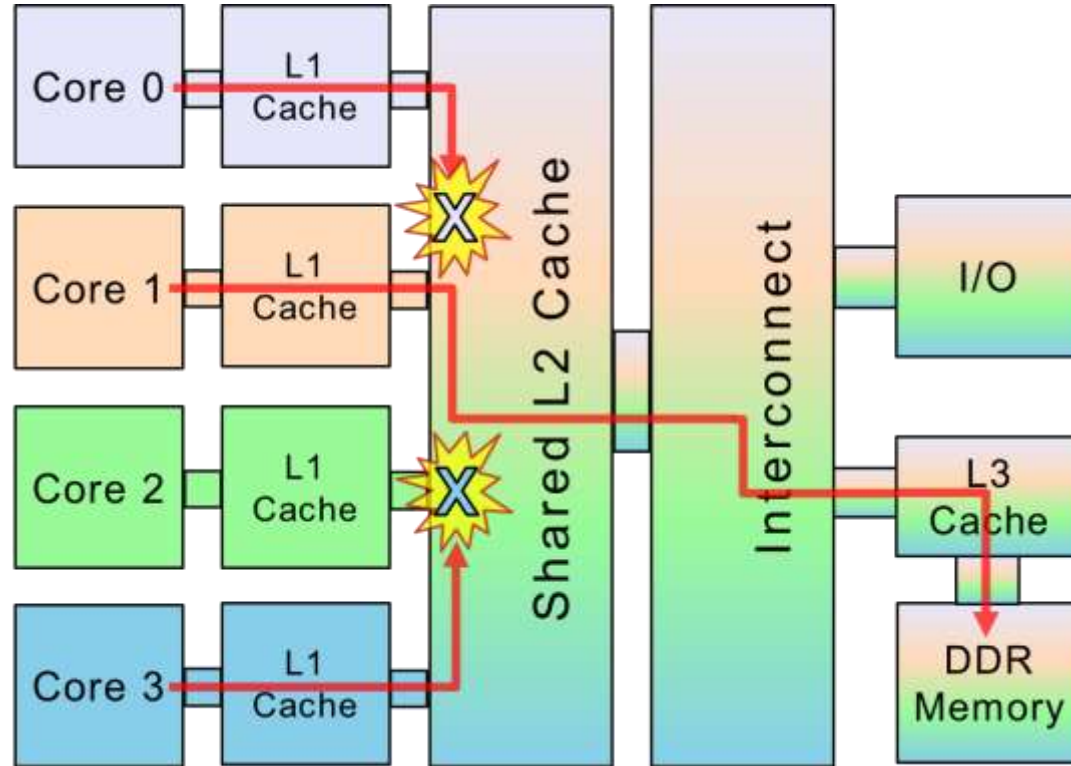
- Core → L1 → L2 → Interconnect → DDR
- DDR → Interconnect → L2 → L1 → Core



## EXAMPLE – NXP P4080



## EXAMPLE – NXP T2080

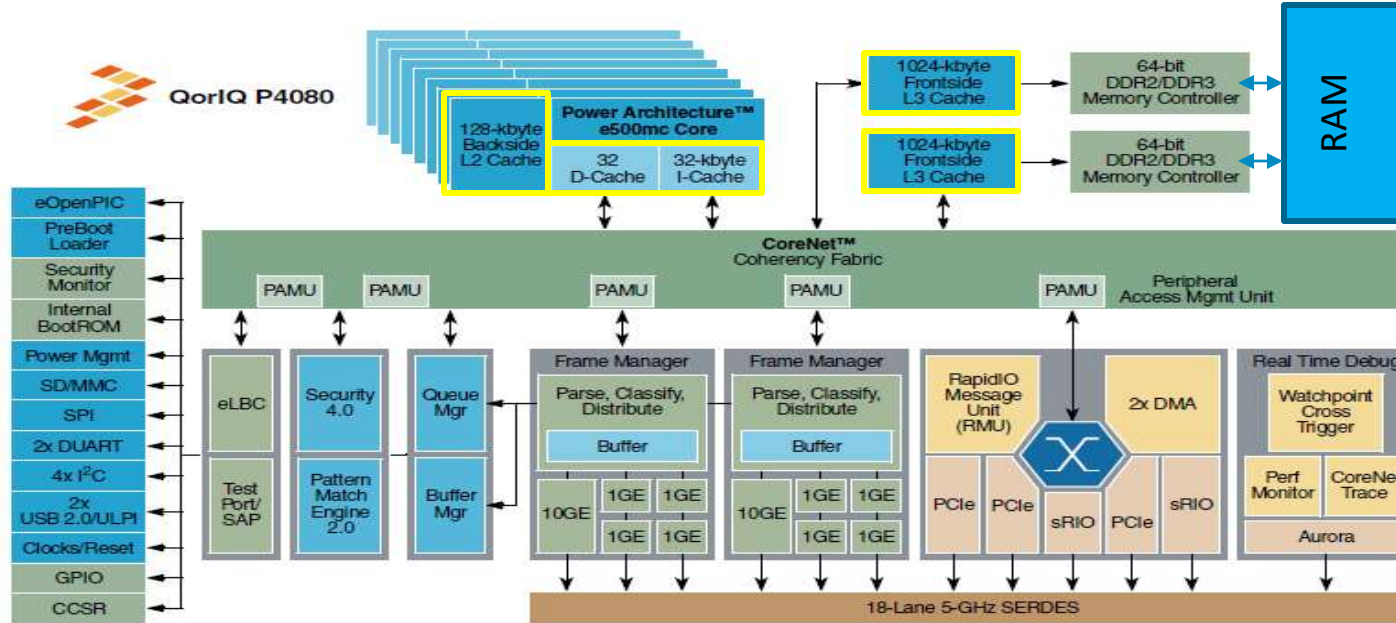


# AvMC MCP INTERFERENCE ANALYSIS

## MCP Interference Matrix

Resource	Issue	Analysis	Mitigation	Verification
<b>Local Cache</b>	<ul style="list-style-type: none"> <li>- Local cache miss</li> <li>- Local cache eviction</li> <li>- Cache coherency</li> </ul>	<ul style="list-style-type: none"> <li>- Performance metrics collection</li> <li>- Cache contents inspection</li> <li>- Use of cache simulators</li> </ul>	<ul style="list-style-type: none"> <li>- Memory allocation adjustment</li> <li>- Multiple memory controllers</li> <li>- Cache inhibition</li> <li>- Cache locking</li> </ul>	<ul style="list-style-type: none"> <li>- Performance metrics collection</li> <li>- Cache contents inspection</li> </ul>
<b>Shared Cache</b>	<ul style="list-style-type: none"> <li>- Cache miss</li> <li>- Cache eviction</li> <li>- Cache coherency</li> </ul>	<ul style="list-style-type: none"> <li>- Performance metrics collection</li> <li>- Analysis of configuration settings</li> <li>- Use of cache simulators</li> </ul>	<ul style="list-style-type: none"> <li>- Memory allocation adjustment</li> <li>- Multiple memory controllers</li> <li>- Cache inhibition</li> <li>- Cache locking</li> <li>- Cache coloring</li> <li>- Cache partitioning</li> <li>- Disabling shared cache</li> </ul>	<ul style="list-style-type: none"> <li>- Off-core performance metrics collection</li> <li>- Analysis of configuration settings</li> </ul>
<b>Main Memory</b>	<ul style="list-style-type: none"> <li>- Access Latency</li> </ul>	<ul style="list-style-type: none"> <li>- Execution timing</li> <li>- Off-core Performance metrics collection</li> </ul>	<ul style="list-style-type: none"> <li>- Memory bank partitioning</li> </ul>	<ul style="list-style-type: none"> <li>- Execution timing</li> <li>- Off-core Performance metrics collection</li> </ul>

# NXP P4080 SYSTEM ON A CHIP (SOC)

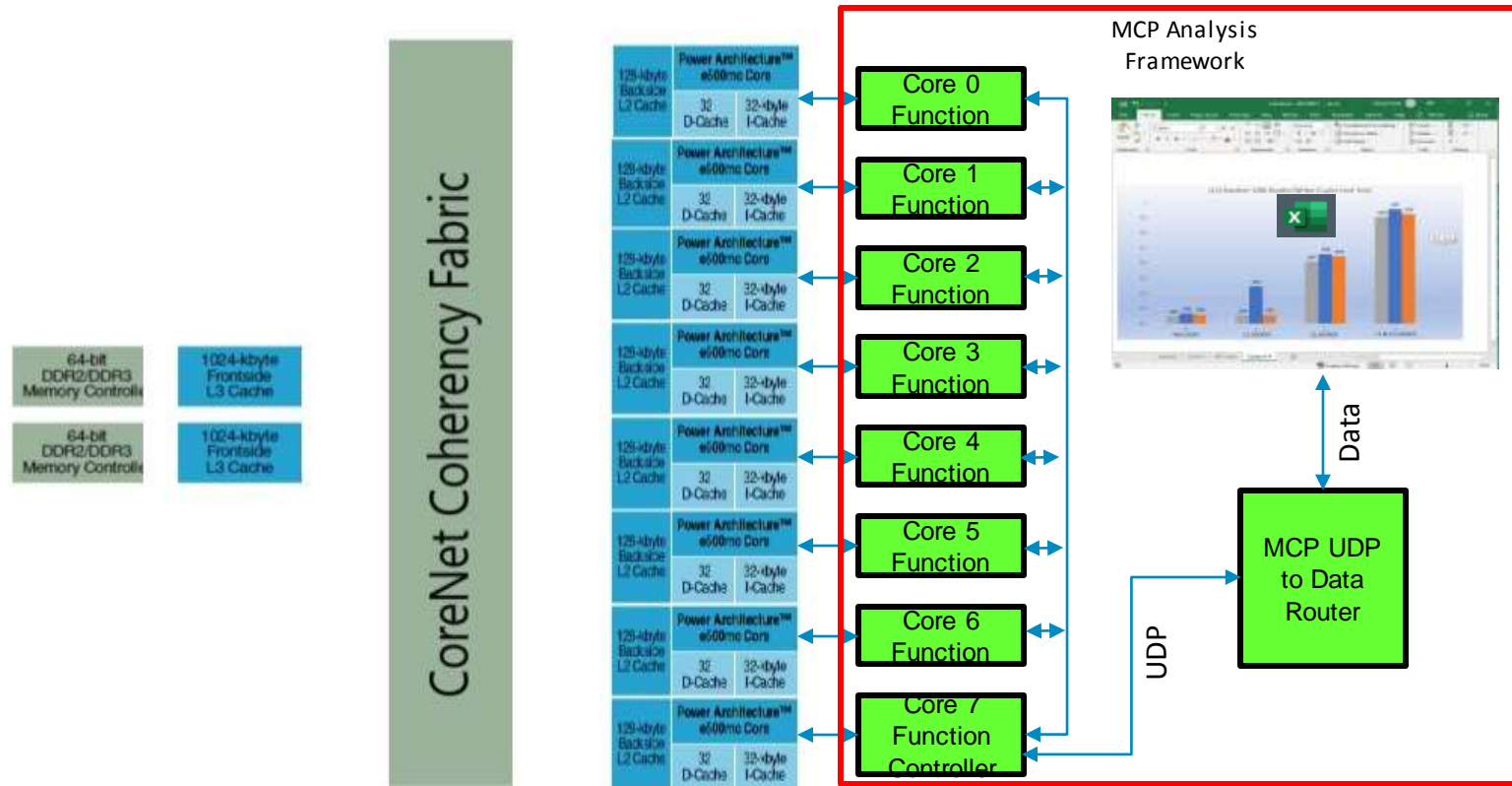


QorIQ™ P4080 Communications Processor Product Brief, Rev. 1

Freescale Semiconductor

SOURCE: <https://www.nxp.com/products/processors-and-microcontrollers/power-architecture-processors/qorIQ-platforms/p-series/qorIQ-p4080-p4040-p4081-multicore-communications-processors:P4080>

## MCP Multi-Core Analysis Framework



# Conclusion

The two-step framework for timing verification has been known for a long time in the academic research literature on real-time systems.

We have presented it to the avionics community.

There are tools that support activities of this two-step framework. Some from SEI and AvMC presented here.

Some tools can be downloaded here:

<https://www.andrew.cmu.edu/user/banderss/projects.html>

# Thanks!