Memory Hierarchies, Pipelines, and Buses for Future Architecture in Time-Critical Embedded Systems

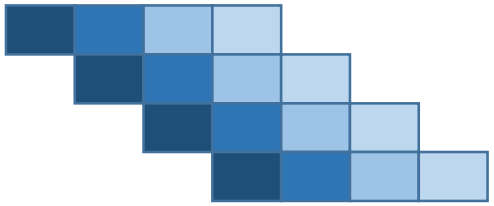
Presenter: Ashutosh Dhekne

PhD student, Computer Science,

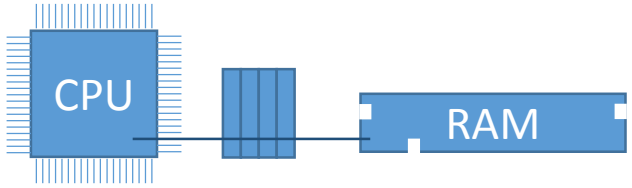
University of Illinois at Urbana Champaign

dhekne2@Illinois.edu

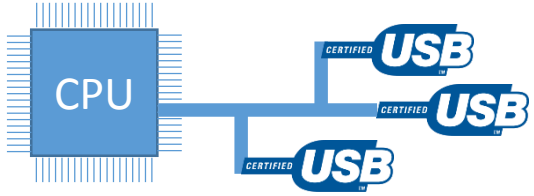
Various Optimization Techniques



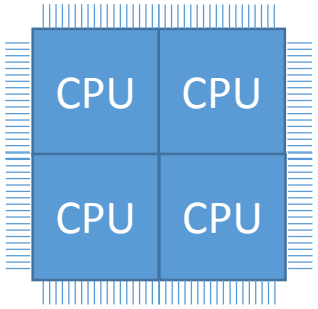
Pipelined Execution



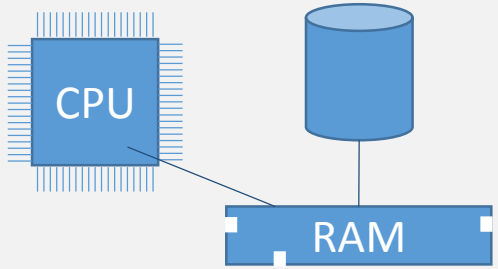
Caching



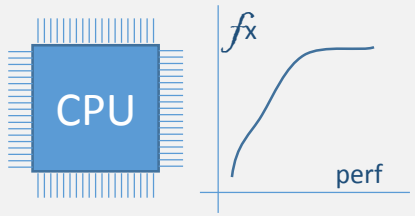
Buses



Multiprocessor Systems



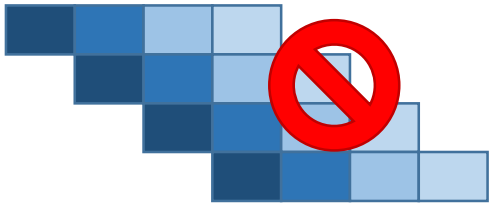
Virtual Memory



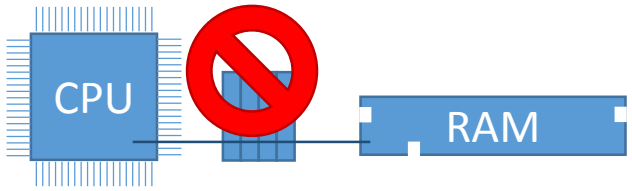
Frequency Scaling

Higher degree of software control

For Timing Analysis

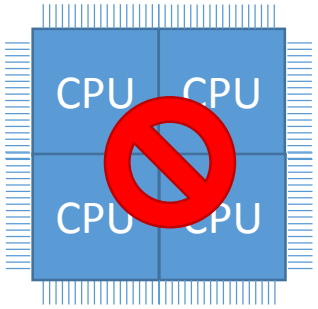


Pipelined Execution

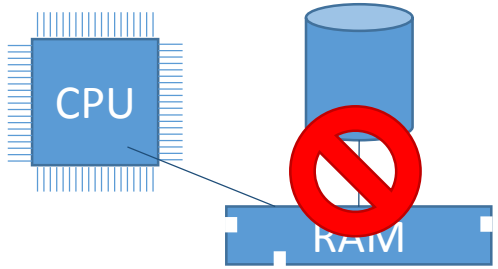


Caching

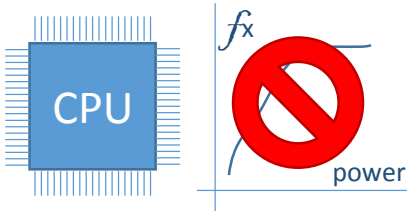
Buses



Multiprocessor Systems



Virtual Memory

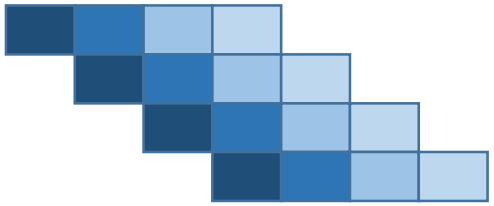


Frequency Scaling

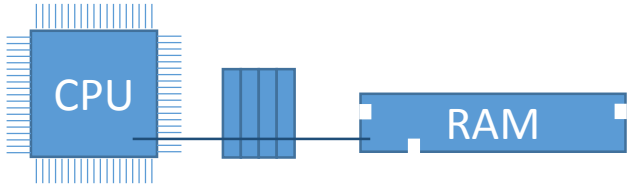
Ability to Analyze

- Is required because …
  - Scheduling algorithms require the upper and lower bounds of tasks
  - Predictability of when a particular task will be completed is an important property of embedded systems
  - Physical interactions require actions enacted within a certain time limit
- Is hard because …
  - User inputs cannot always be predicted
  - Processors optimize average execution time
  - Out of proportion penalties experienced on certain events
  - We want to make use of as many optimizations as possible

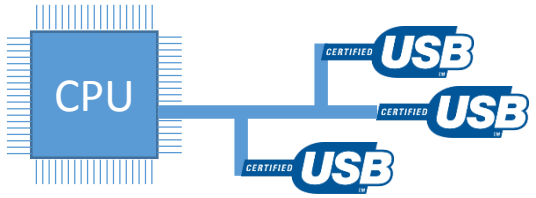
What is recommended



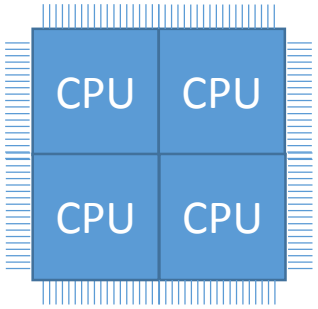
Pipelined Execution
[No Domino Effect]



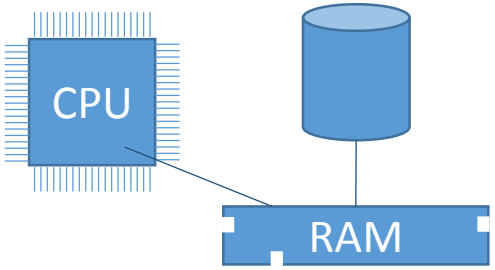
Caching
[Use LRU]



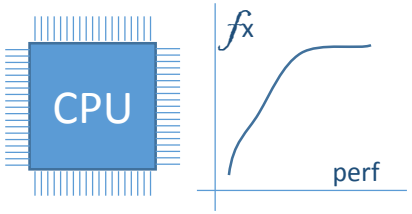
Buses
[Match CPU clock speeds]



Multiprocessor Systems
[Stay away]



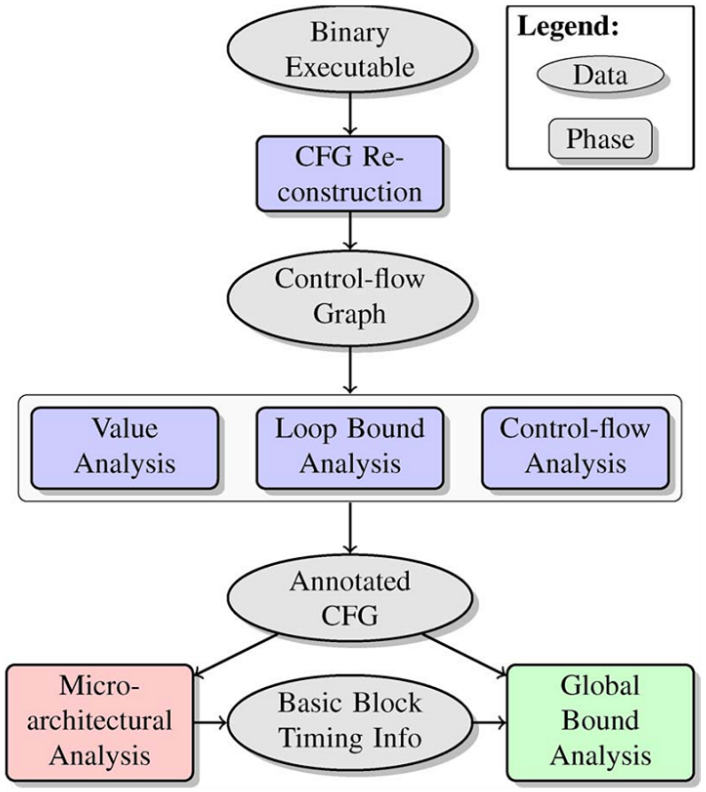
Virtual Memory
[Do not use]



Frequency Scaling
[Not Discussed]

How to Analyze Time

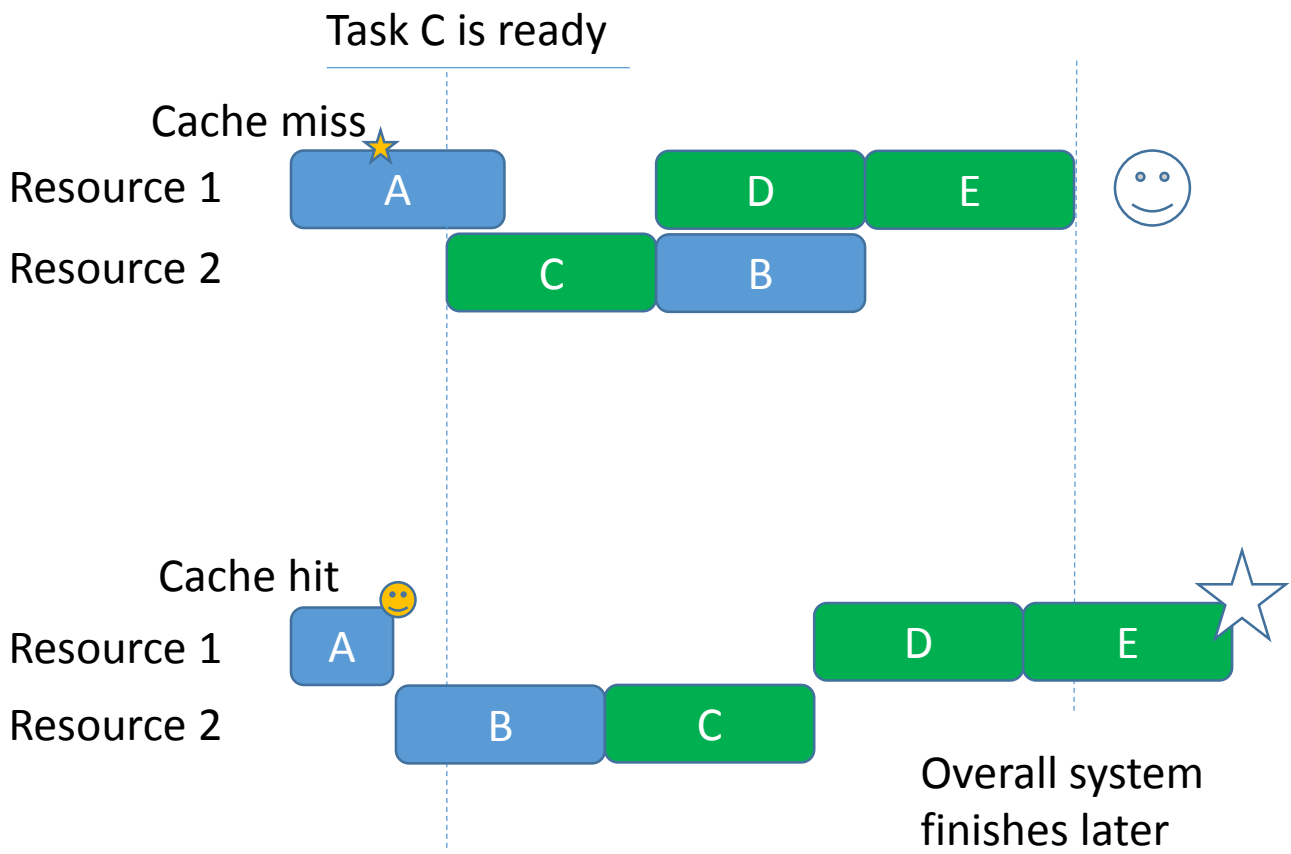
- Control Flow Reconstruction
- Value Analysis
- Loop Bound Analysis
- Control Flow Analysis
- Microarchitectural Analysis
- Global Bound Analysis



[link]

Pipelining

- Allows parallel execution of instructions
- Average speed of execution is greatly enhanced
- But can cause scheduling anomalies.
- In the adjoining diag., A completed faster but still ended up with increasing the overall schedule's schedule.



Domino Effect

- A system exhibits domino effect if there are two hardware states s and t such that the difference in execution time (of the same program string starting in s and t) may be arbitrary high.
- Domino effect is seen in real hardware as well
- An example of domino effect in FIFO policy (similar implications in pipelines as well)

	Initially (..)	Cache Miss?	Initially (CA)	Cache Miss?	Iteration
A	A.	Yes	CA	No	1
B	BA	Yes	BC	Yes	1
C	CB	Yes	BC	No	1
A	AC	Yes	AB	Yes	2
B	BA	Yes	AB	No	2
C	CB	Yes	CA	Yes	2

Example from *Thomas Lundqvist*, "A WCET Analysis Method for Pipelined Microprocessors with Cache Memories"
And adapted by *Christoph Berg*, in "PLRU Cache domino effect"

Classification of Architectures

- Fully Time Compositional Architectures
  - Does not exhibit Domino effect
  - Architecture does not exhibit timing anomaly.
  - Stalls all components of the pipeline on a timing issue.
- Compositional with Constant Bound Time
  - Timing Anomalies exist, but no domino effect
  - Local worst case paths can be eliminated.
- Noncompositional
  - Exhibit both Domino’s effect and timing anomalies

Caches

- Least Recently Used
  - Queue for each cache set.
  - On cache miss, the fetched element is placed at the front of the queue
  - On cache hit, that element is brought to the front
- First In First Out
  - New elements are added at the front of the queue
  - Old elements might choose to replace one of the existing
- PLRU
  - Tree based approximation of the LRU.
  - Pages are at the leaves of the trees

May and Must Analysis

- Starting from a complete unknown state of the cache, can we predict what all the cache contains after running a program that uses cache?

- The **must** cache at a program point is a set of memory blocks that are definitely in each concrete cache at that point.

- The **may** cache is a set of memory blocks that may be in a concrete cache whenever that program execution reaches that program point.

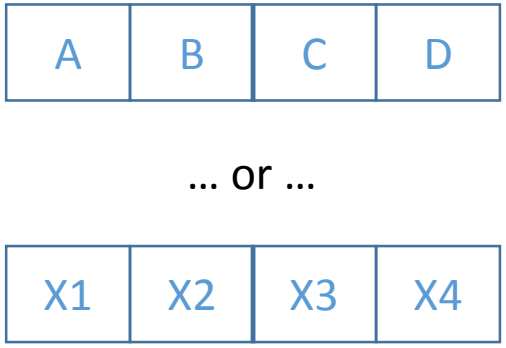
May and Must Analysis

- Initially, no memory blocks can be guaranteed to be in the cache. Hence the **must** set is an empty set.
- Any of the blocks in the system could be in the cache. So the **may** set comprises of all the blocks in memory.

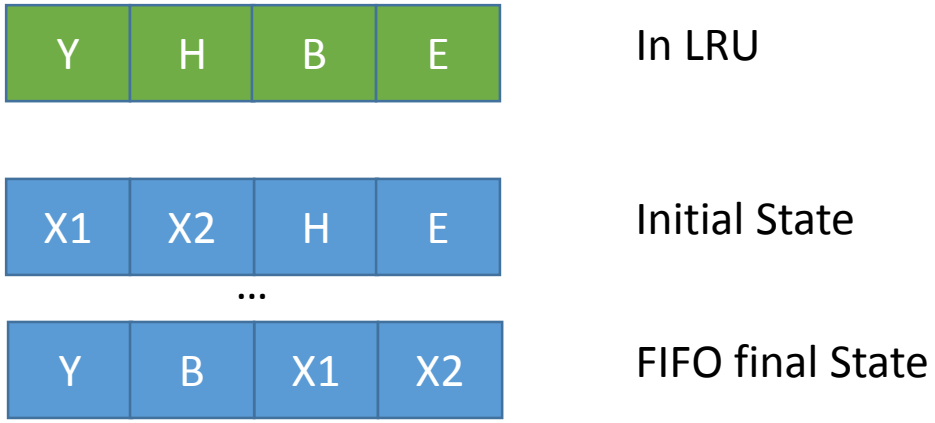
- As we see more and more *distinct* blocks of memory being used, the amount of information we have about the memory blocks in the cache starts to increase
- The rate of this increase in the information is dependent on the replacement method

May and Must Analysis

- Initial State
  - Must set is {}
  - May set is {A, B, …, X1, …, X4}



- We start seeing a sequence of blocks due to program execution {E, B, H, Y}.

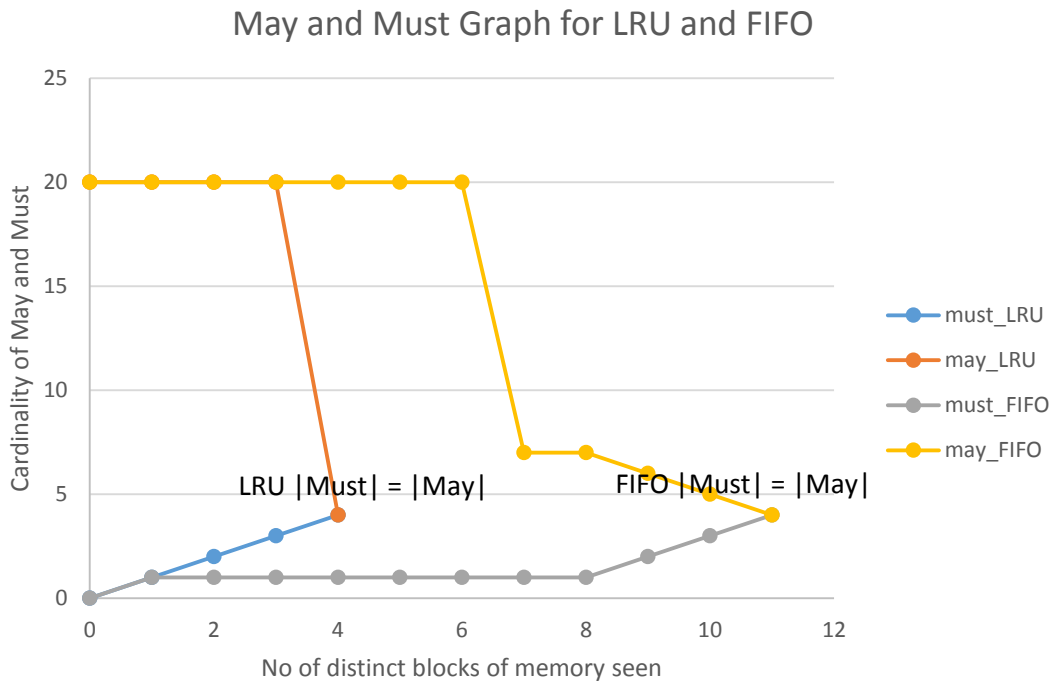


Overcoming Inertia

- How do we eliminate the dependency on the initial state?
- **May(s)** is the set of cache contents that may still be in the cache set after accessing the sequence **s** regardless of the initial state.
- **Must(s)** is the set of cache contents that must be in the cache after accessing the sequence **s** regardless of the initial state.
- As we see more and more data, the cardinality of these two sets converge

Nothing like LRU

Comparison between
LRU and FIFO for a
4-way associative cache



Note that the events indicated by the blobs on the lines are discrete. Hence the blobs are important. The line does help in understanding of the graph and has been left in place just for that reason.

LRU converges much faster than FIFO

Cache Intricacies

- Unlike instruction caches, data cache could get dirty
- Write-through and write back modes affect timing analysis
- Use of various levels of cache also impacts timing analysis
  - L1, L2, L3 caches with varying amount of cache sizes and different replacement policies could make timing analysis an intractable problem
- Do all microprocessors allow turning off of caches?

Buses

- Buses typically have a very low frequency compared to that of the CPU
- The CPU can interface with the bus only periodically
- Bus-clock states is given by fCPU/[gcd(fCPU, fBUS)]
- It is beneficial to have fCPU/fBUS be integral ratio
- Pipelining of bus creates complications
- CSMA/CA or DRAM refresh cycles have to be analyzed based on their amortized cost over time

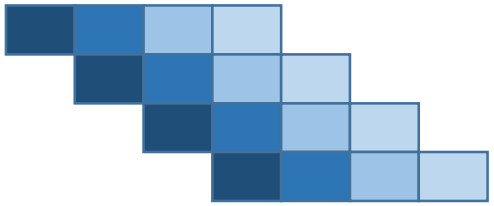
Multi-Core Architecture

- Individual CPU cores having their own L1 and L2 cache with LRU replacement algorithm
- Careful scheduling of tasks is required for ensuring no collision of read and write

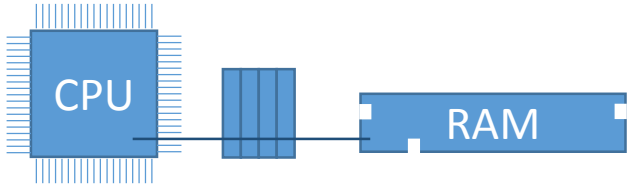
Conclusions

- Memory Hierarchy
  - Caches with LRU replacement policy, and/or scratchpad memories
  - Separate L1 instruction and data cache
  - A flat linear byte-oriented memory layout without paging
  - Nonshared memory
- Pipelines
  - Use compositional pipelines

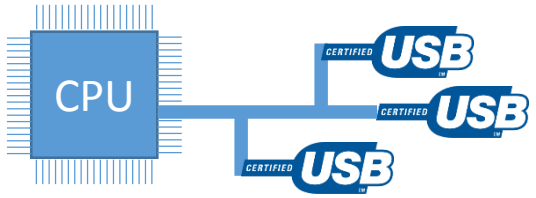
What is recommended (once again)



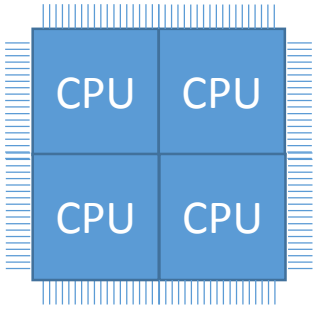
Pipelined Execution
[No Domino Effect]



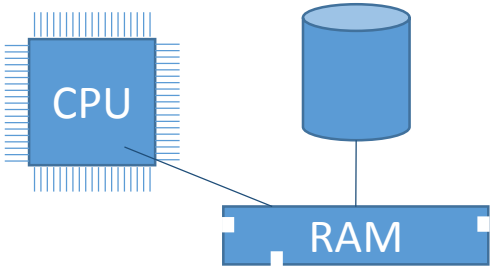
Caching
[Use LRU]



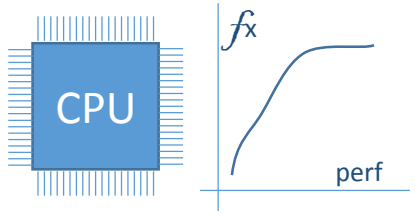
Buses
[Match CPU clock speeds]



Multiprocessor Systems
[Stay away]



Virtual Memory
[Do not use]



Frequency Scaling
[Not Discussed]

Thank you!