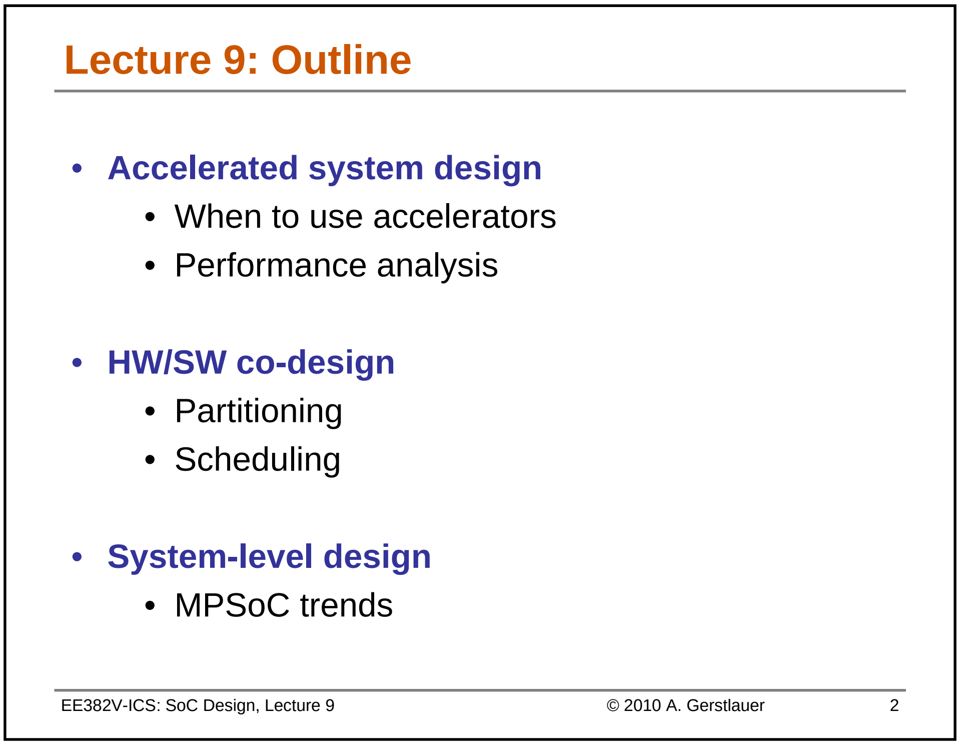
EE382V-ICS: System-on-a-Chip (SoC) Design

Lecture 9 – HW/SW Co-Design

Sources:
Prof. Margarida Jacome, UT Austin
Prof. Lothar Thiele, ETH Zürich

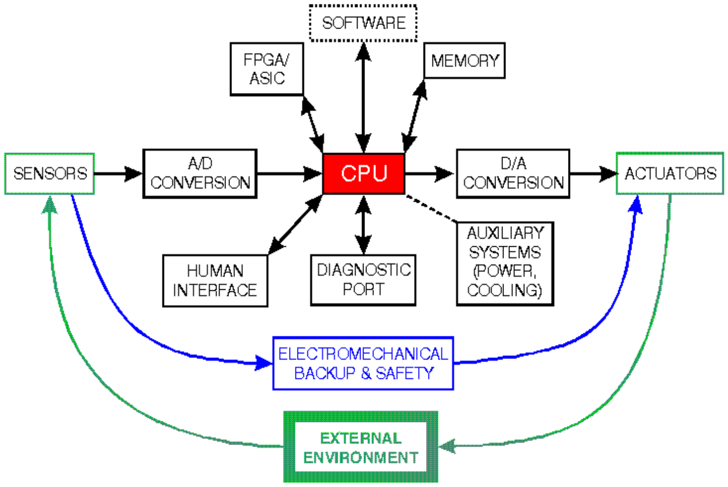
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu



Lecture 9: Outline

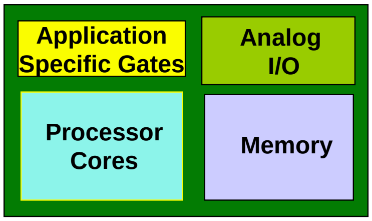
- **Accelerated system design**
  - When to use accelerators
  - Performance analysis
- **HW/SW co-design**
  - Partitioning
  - Scheduling
- **System-level design**
  - MPSoC trends

“Traditional” Embedded Systems



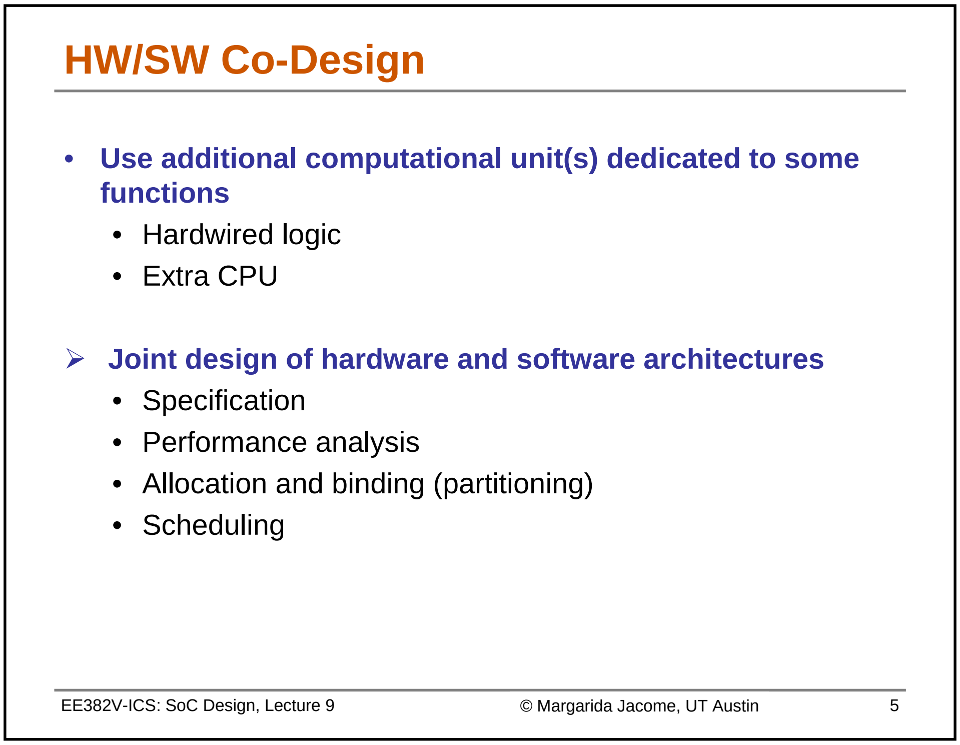
Modern Embedded Systems?

- **Employ a combination of**
  - SW on programmable processors
    - – Flexibility, complexity
  - Application-specific, custom HW
    - – Performance, low power
  - Transducers, sensors, actuators
  - A/D & D/A converters
    - – Interact with analog, continuous-time environment



➢ **Micro-controllers & digital signal processors (DSPs)**

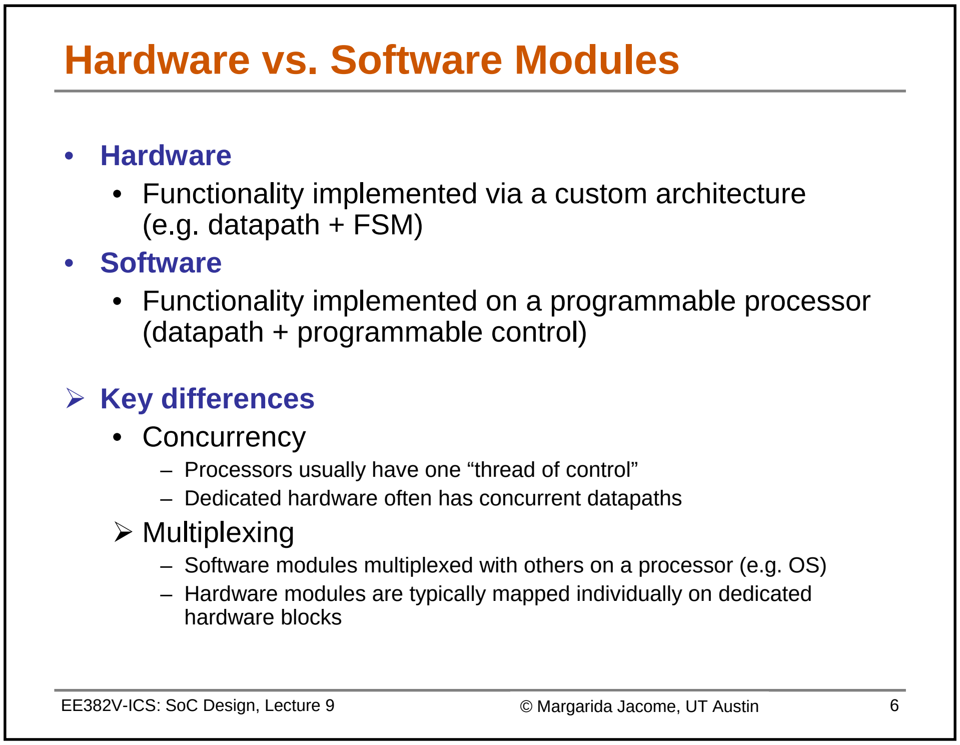
➢ **ASICs & Field programmable gate arrays (FPGAs)**



HW/SW Co-Design

- **Use additional computational unit(s) dedicated to some functions**
  - Hardwired logic
  - Extra CPU

➢ **Joint design of hardware and software architectures**
  - Specification
  - Performance analysis
  - Allocation and binding (partitioning)
  - Scheduling



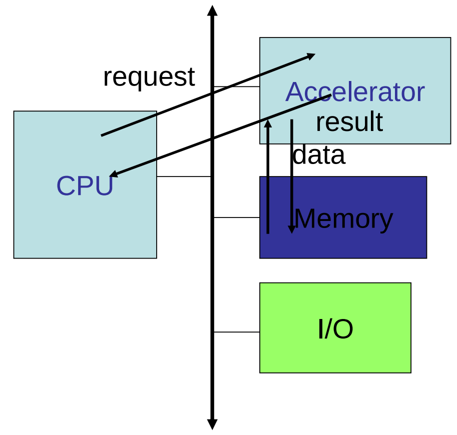
Hardware vs. Software Modules

- **Hardware**
  - Functionality implemented via a custom architecture (e.g. datapath + FSM)
- **Software**
  - Functionality implemented on a programmable processor (datapath + programmable control)

➢ **Key differences**
  - Concurrency
    - Processors usually have one “thread of control”
    - Dedicated hardware often has concurrent datapaths

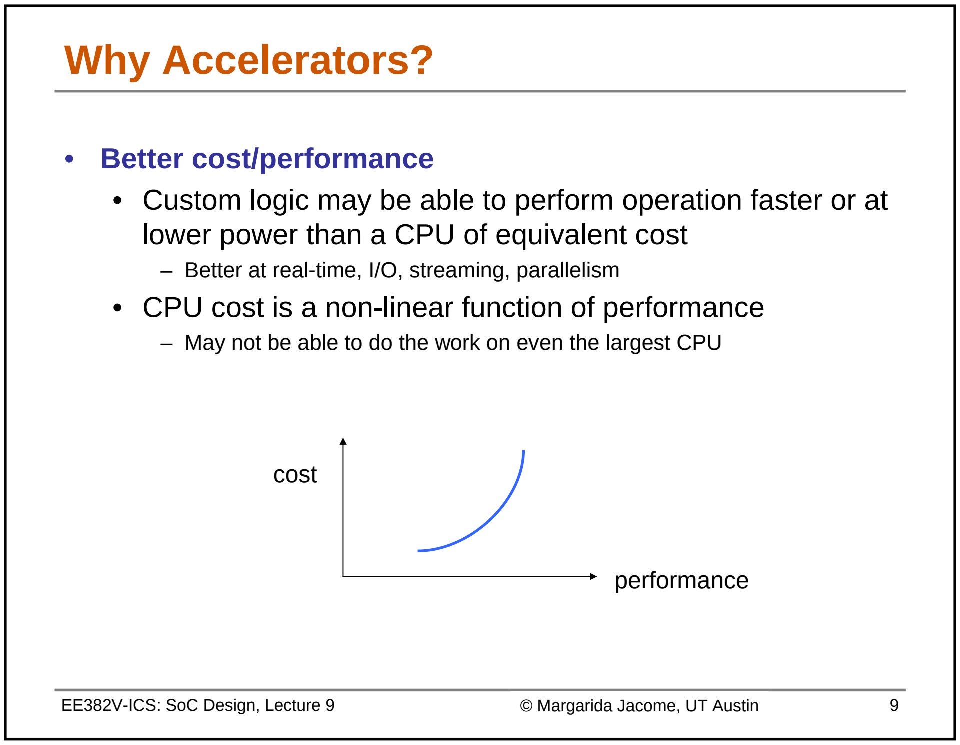
  ➢ Multiplexing
    - Software modules multiplexed with others on a processor (e.g. OS)
    - Hardware modules are typically mapped individually on dedicated hardware blocks

Accelerated System Architecture



Accelerators

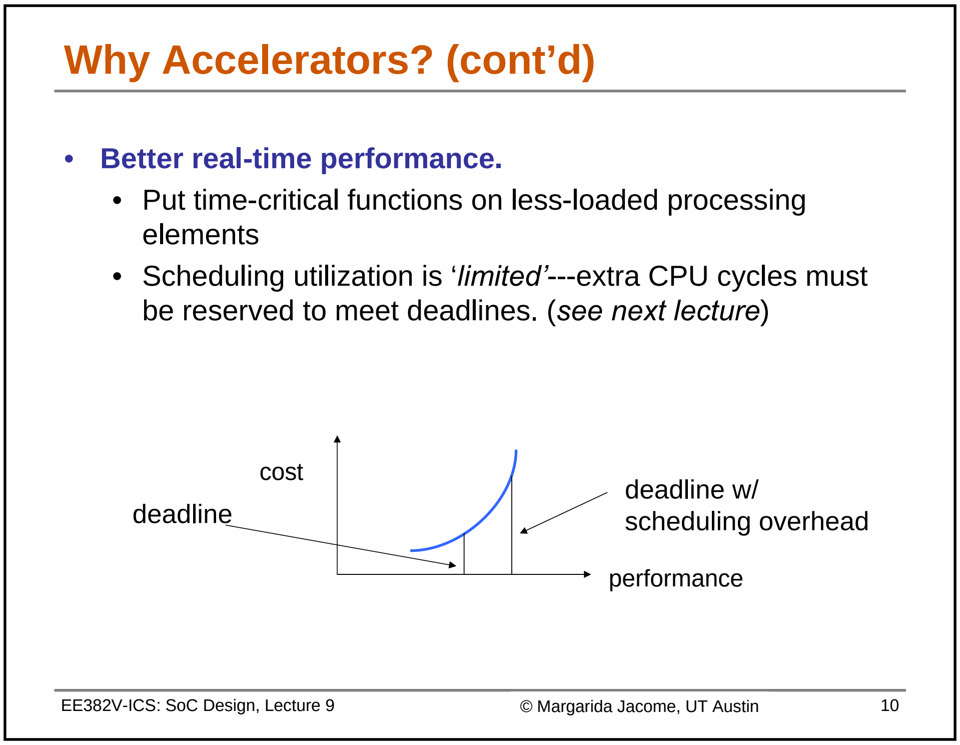
- **Accelerator vs. co-processor**
  - A co-processor executes instructions.
    - Instructions are dispatched by the CPU
  - An accelerator appears as a device on the bus.
    - The accelerator is controlled via registers
- **Accelerator implementations**
  - Application-specific integrated circuit (ASIC)
  - Field-programmable gate array (FPGA).
  - Standard component.
    - Example: graphics processor.
  - SoCs enable multiple accelerators, peripherals, and some memory to be placed with a CPU on a single chip



Why Accelerators?

- **Better cost/performance**
  - Custom logic may be able to perform operation faster or at lower power than a CPU of equivalent cost
    - Better at real-time, I/O, streaming, parallelism
  - CPU cost is a non-linear function of performance
    - May not be able to do the work on even the largest CPU

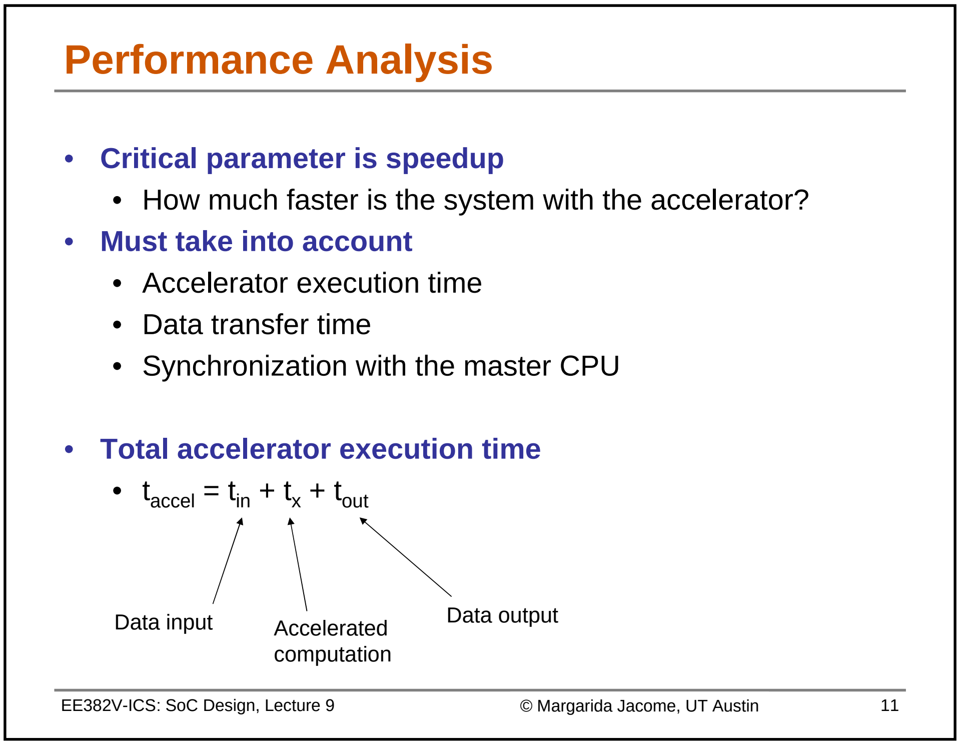
cost
performance



Why Accelerators? (cont'd)

- **Better real-time performance.**
  - Put time-critical functions on less-loaded processing elements
  - Scheduling utilization is '*limited*'---extra CPU cycles must be reserved to meet deadlines. (*see next lecture*)

cost
deadline
deadline w/
scheduling overhead
performance



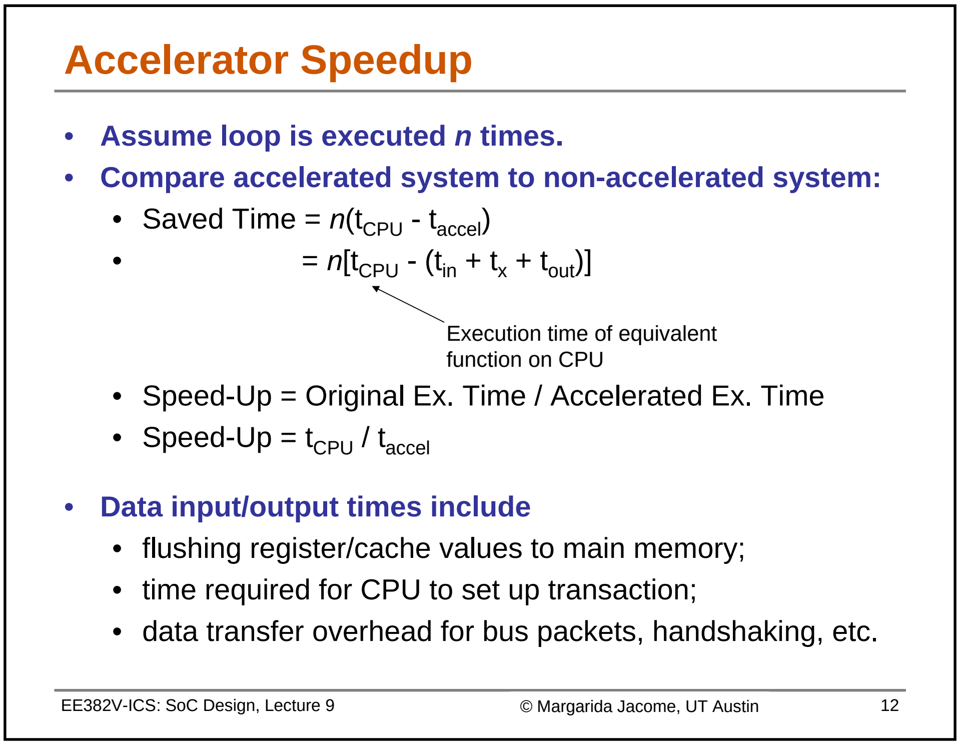
Performance Analysis

- **Critical parameter is speedup**
  - How much faster is the system with the accelerator?
- **Must take into account**
  - Accelerator execution time
  - Data transfer time
  - Synchronization with the master CPU
- **Total accelerator execution time**
  - $t_{accel} = t_{in} + t_x + t_{out}$

Data input

Accelerated computation

Data output

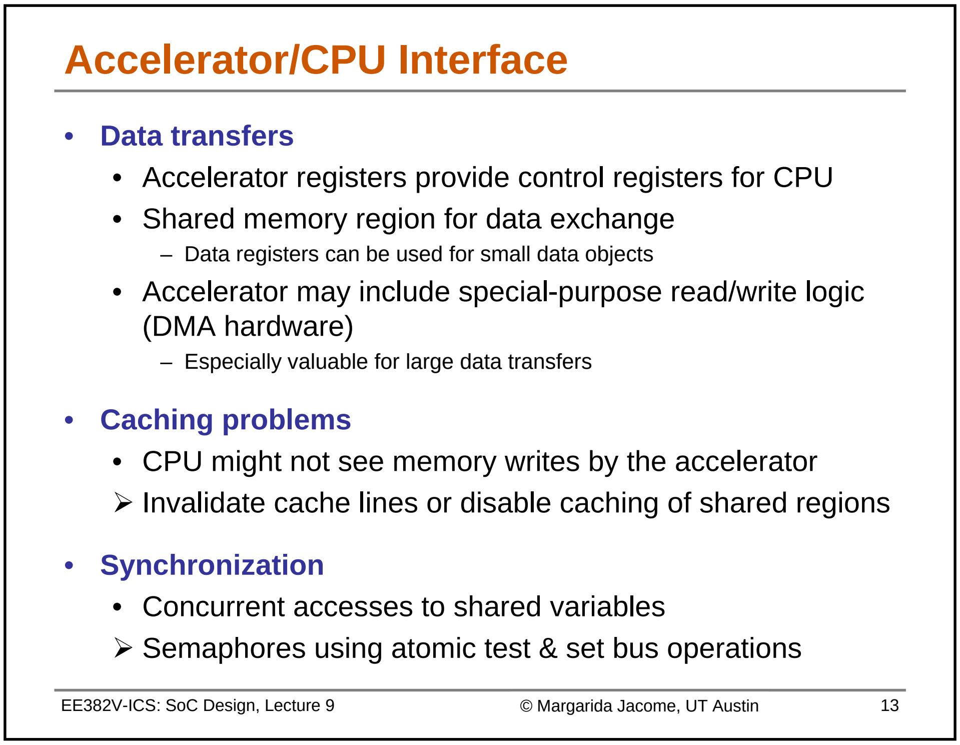


Accelerator Speedup

- **Assume loop is executed *n* times.**
- **Compare accelerated system to non-accelerated system:**
  - Saved Time = $n(t_{CPU} - t_{accel})$
  - $= n[t_{CPU} - (t_{in} + t_x + t_{out})]$

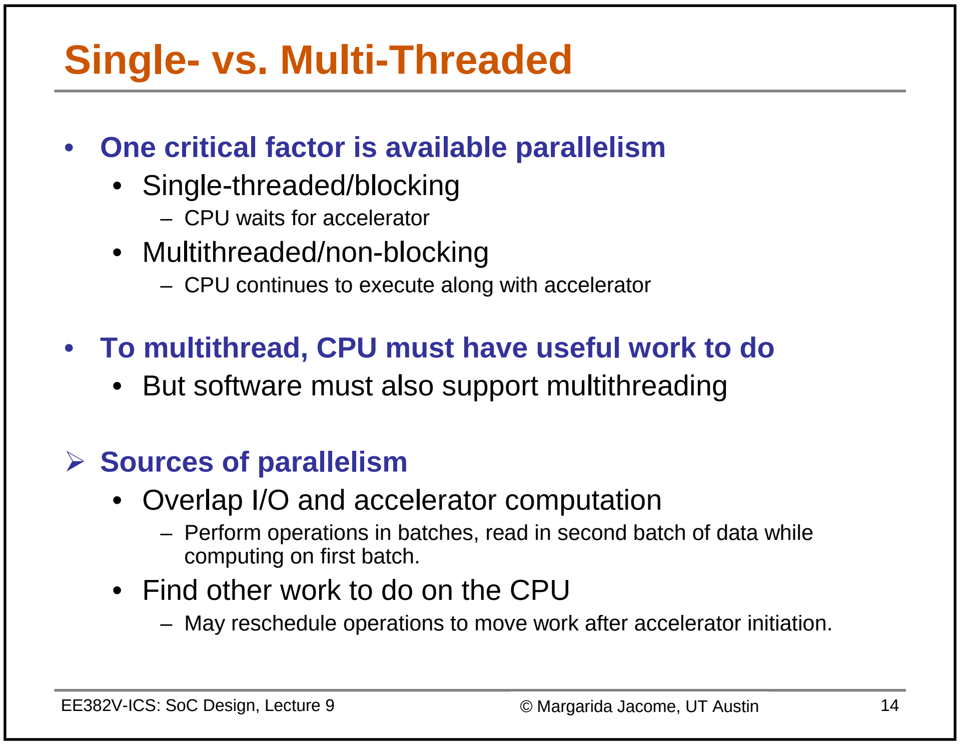
Execution time of equivalent function on CPU

  - Speed-Up = Original Ex. Time / Accelerated Ex. Time
  - Speed-Up = t_{CPU} / t_{accel}
- **Data input/output times include**
  - flushing register/cache values to main memory;
  - time required for CPU to set up transaction;
  - data transfer overhead for bus packets, handshaking, etc.



Accelerator/CPU Interface

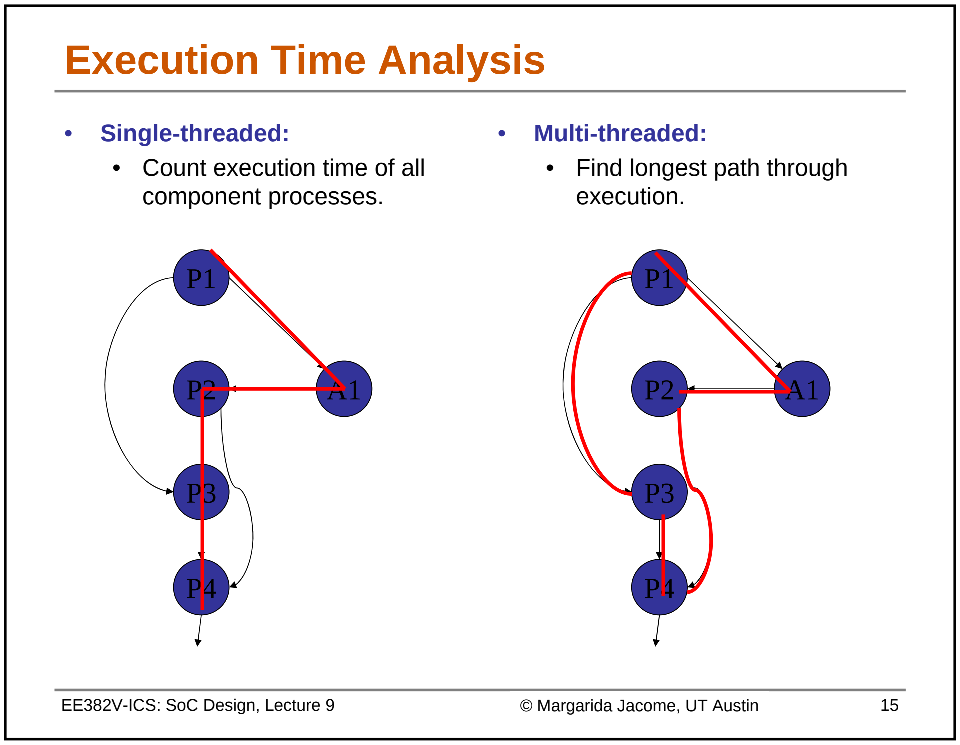
- **Data transfers**
  - Accelerator registers provide control registers for CPU
  - Shared memory region for data exchange
    - Data registers can be used for small data objects
  - Accelerator may include special-purpose read/write logic (DMA hardware)
    - Especially valuable for large data transfers
- **Caching problems**
  - CPU might not see memory writes by the accelerator
  - ➢ Invalidate cache lines or disable caching of shared regions
- **Synchronization**
  - Concurrent accesses to shared variables
  - ➢ Semaphores using atomic test & set bus operations



Single- vs. Multi-Threaded

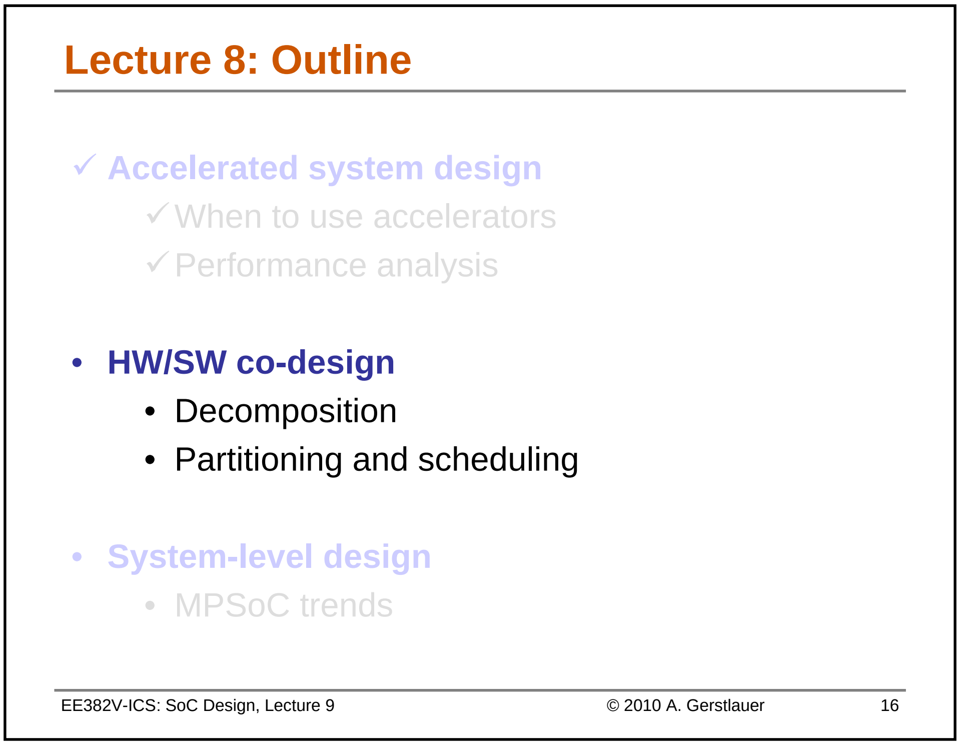
- **One critical factor is available parallelism**
  - Single-threaded/blocking
    - CPU waits for accelerator
  - Multithreaded/non-blocking
    - CPU continues to execute along with accelerator
- **To multithread, CPU must have useful work to do**
  - But software must also support multithreading

➢ **Sources of parallelism**
  - Overlap I/O and accelerator computation
    - Perform operations in batches, read in second batch of data while computing on first batch.
  - Find other work to do on the CPU
    - May reschedule operations to move work after accelerator initiation.



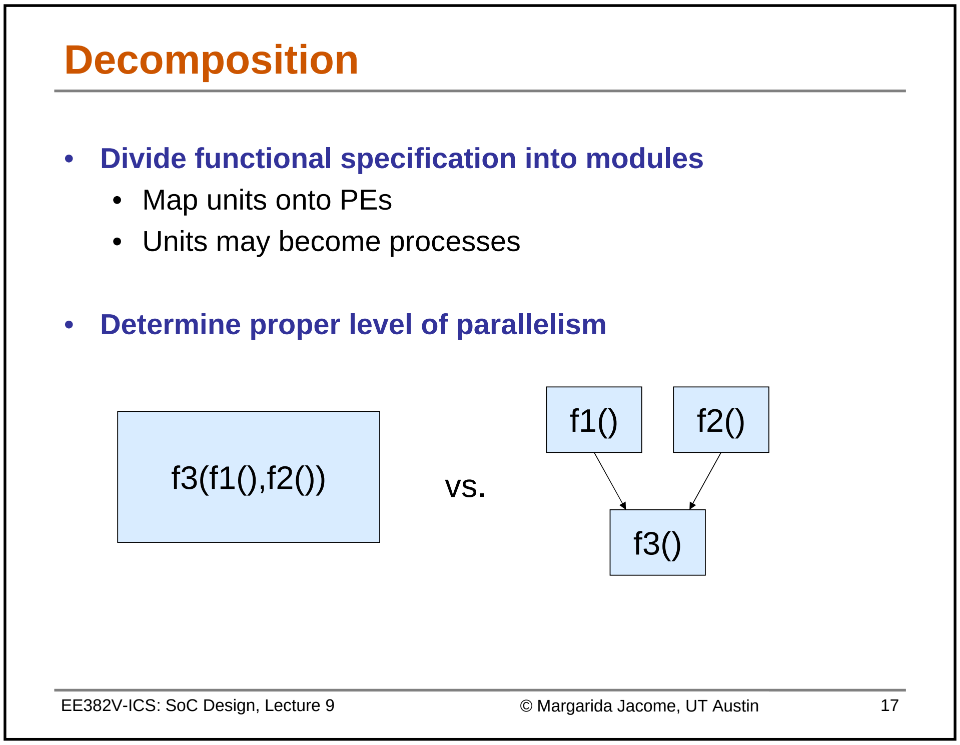
Execution Time Analysis

- **Single-threaded:**
  - Count execution time of all component processes.
- **Multi-threaded:**
  - Find longest path through execution.



Lecture 8: Outline

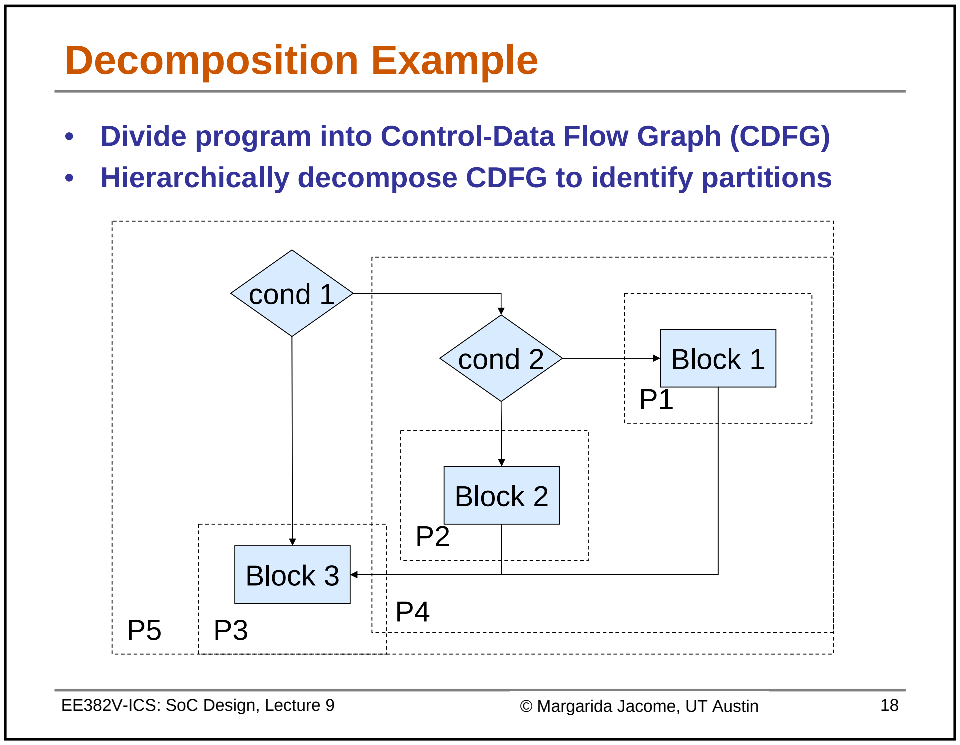
- ✓ Accelerated system design
  - ✓ When to use accelerators
  - ✓ Performance analysis
- **HW/SW co-design**
  - Decomposition
  - Partitioning and scheduling
- System-level design
  - MPSoC trends



Decomposition

- **Divide functional specification into modules**
  - Map units onto PEs
  - Units may become processes
- **Determine proper level of parallelism**

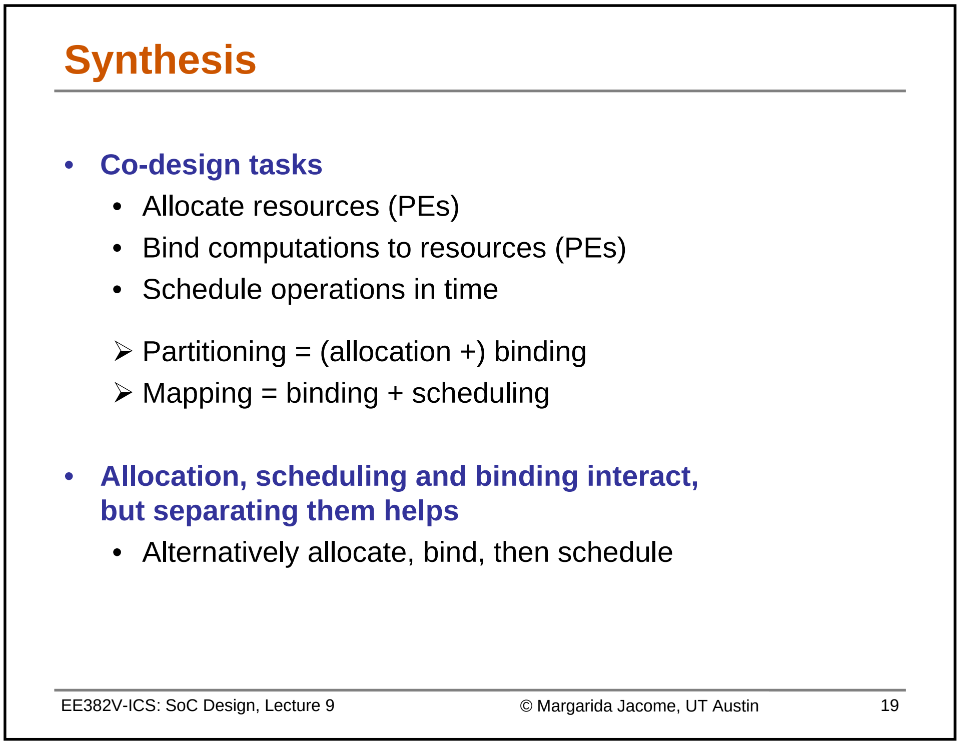
EE382V-ICS: SoC Design, Lecture 9 © Margarida Jacome, UT Austin 17



Decomposition Example

- **Divide program into Control-Data Flow Graph (CDFG)**
- **Hierarchically decompose CDFG to identify partitions**

EE382V-ICS: SoC Design, Lecture 9 © Margarida Jacome, UT Austin 18



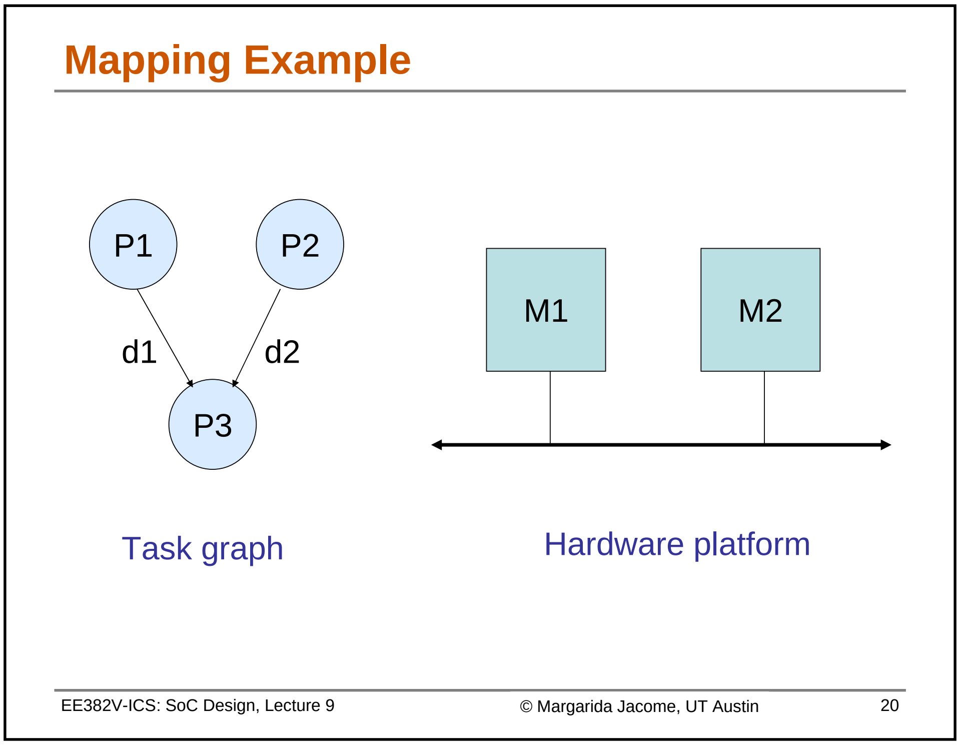
Synthesis

- **Co-design tasks**
  - Allocate resources (PEs)
  - Bind computations to resources (PEs)
  - Schedule operations in time

  ➢ Partitioning = (allocation +) binding

  ➢ Mapping = binding + scheduling

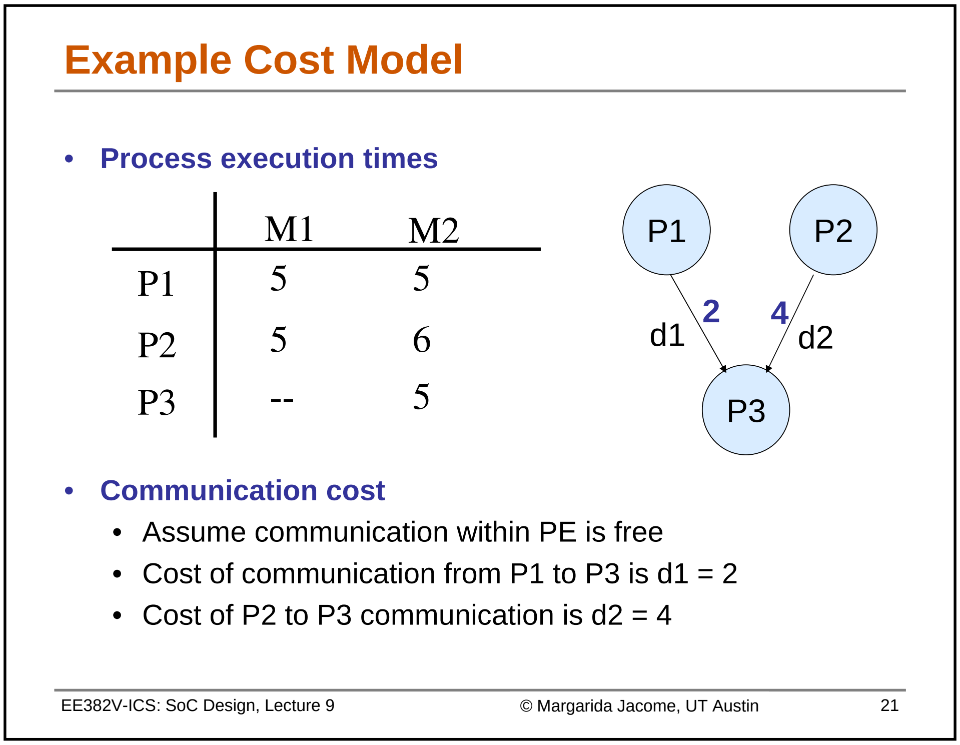
- **Allocation, scheduling and binding interact, but separating them helps**
  - Alternatively allocate, bind, then schedule



Mapping Example

Task graph

Hardware platform



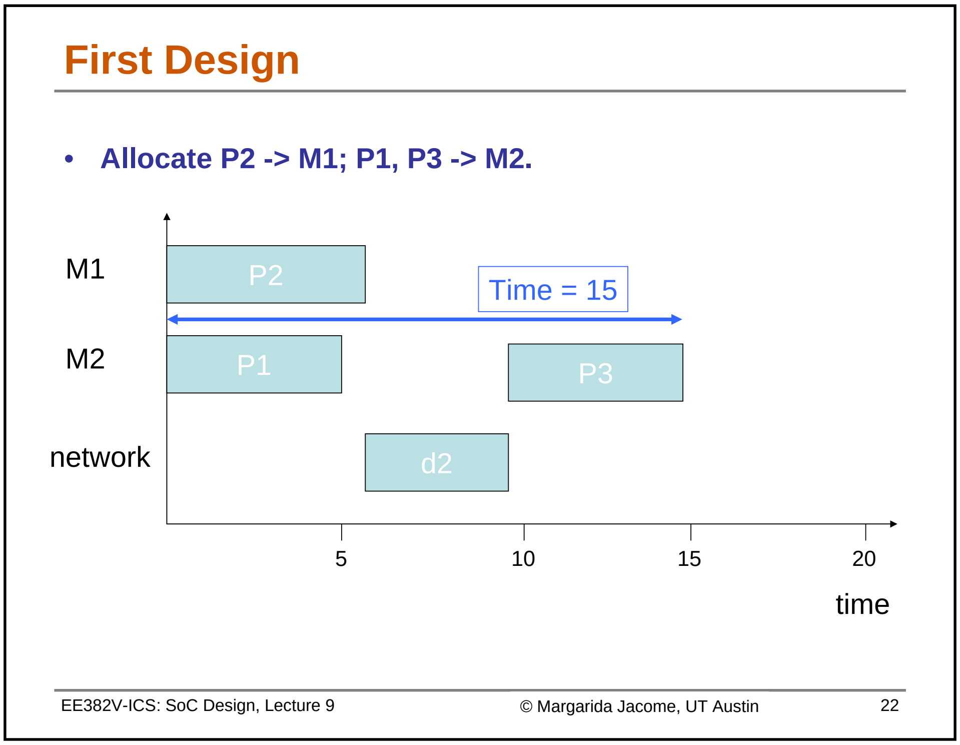
Example Cost Model

- **Process execution times**

	M1	M2
P1	5	5
P2	5	6
P3	--	5

P1
P2
2
4
d1
d2
P3

- **Communication cost**
  - Assume communication within PE is free
  - Cost of communication from P1 to P3 is d1 = 2
  - Cost of P2 to P3 communication is d2 = 4

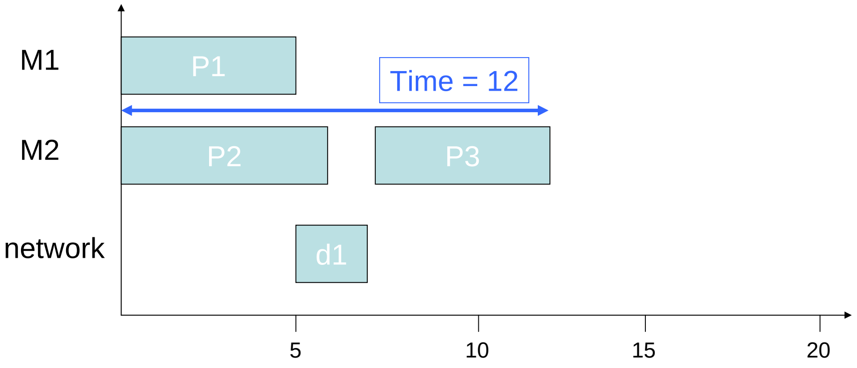


First Design

- **Allocate P2 -> M1; P1, P3 -> M2.**

Second Design

- **Allocate P1 -> M1; P2, P3 -> M2:**



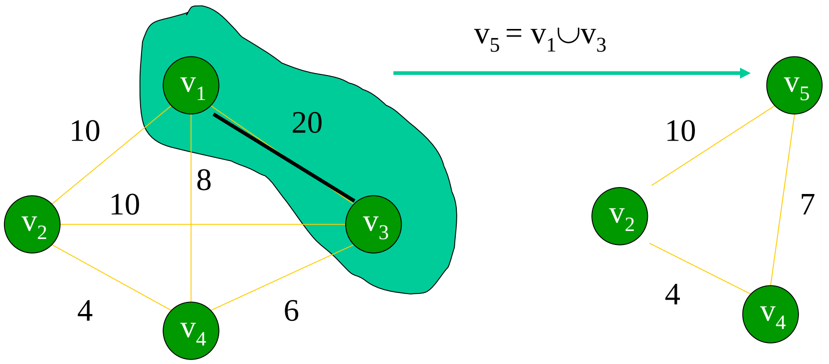
Co-Design Approaches

- **Partitioning**
  - Exact methods
    - Integer linear programming (ILP) formulations
  - Heuristics
    - Constructive: Hierarchical clustering
    - Iterative: Kernighan-Lin
- **Scheduling**
  - Static
    - ILP formulations for combined scheduling & partitioning
    - ➢ Borrowed from high-level synthesis (see later lectures)
  - Dynamic
    - Operating system
    - ➢ Real-time scheduling

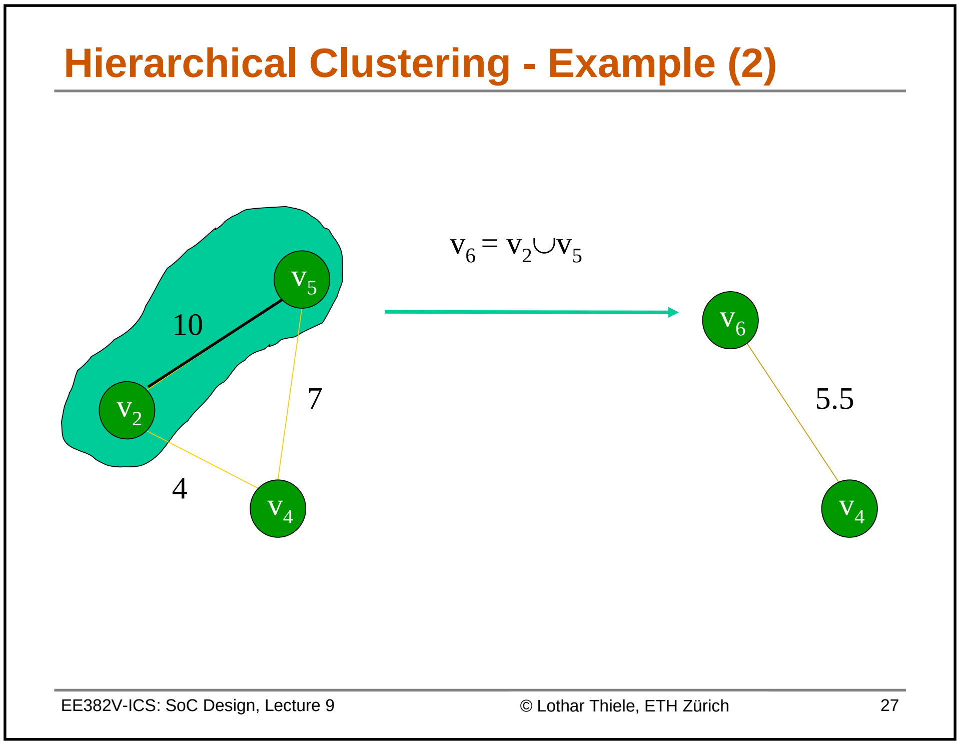
Constructive Methods

- **Random mapping**
  - Each object is assigned to a block randomly
- **Hierarchical clustering**
  - Stepwise grouping of objects
  - Closeness function determines how desirable it is to group two objects
- ➢ **Constructive methods**
  - Often used to generate a starting partition for iterative methods
  - Show the difficulty of finding proper closeness functions

Hierarchical Clustering - Example (1)



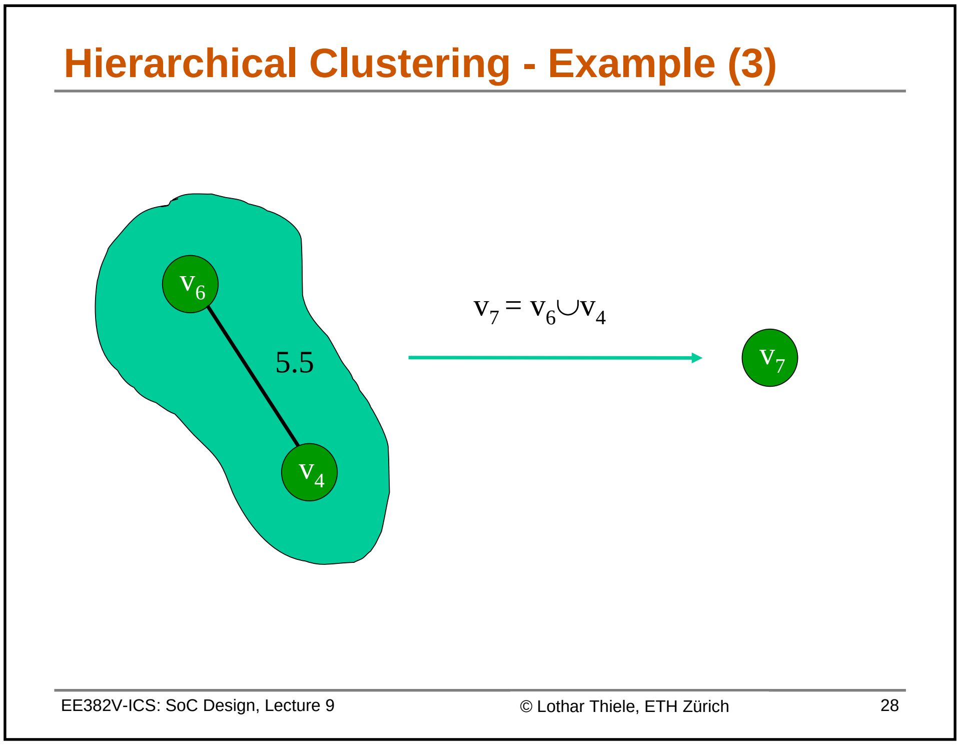
Closeness function: arithmetic mean of weights



Hierarchical Clustering - Example (2)

$v_6 = v_2 \cup v_5$

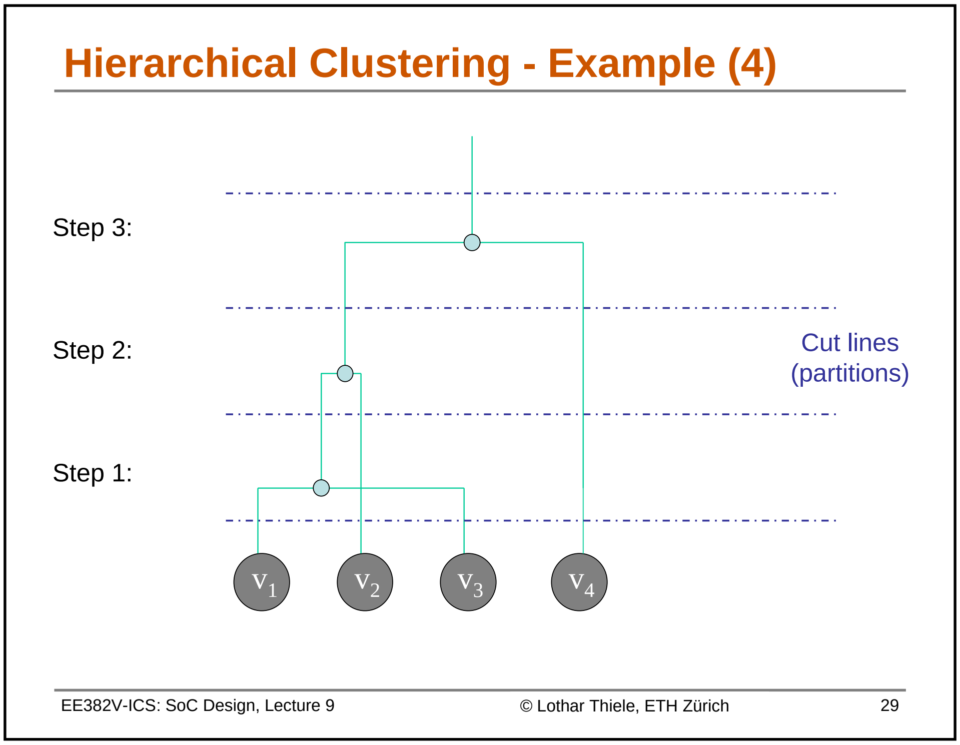
EE382V-ICS: SoC Design, Lecture 9 © Lothar Thiele, ETH Zürich 27



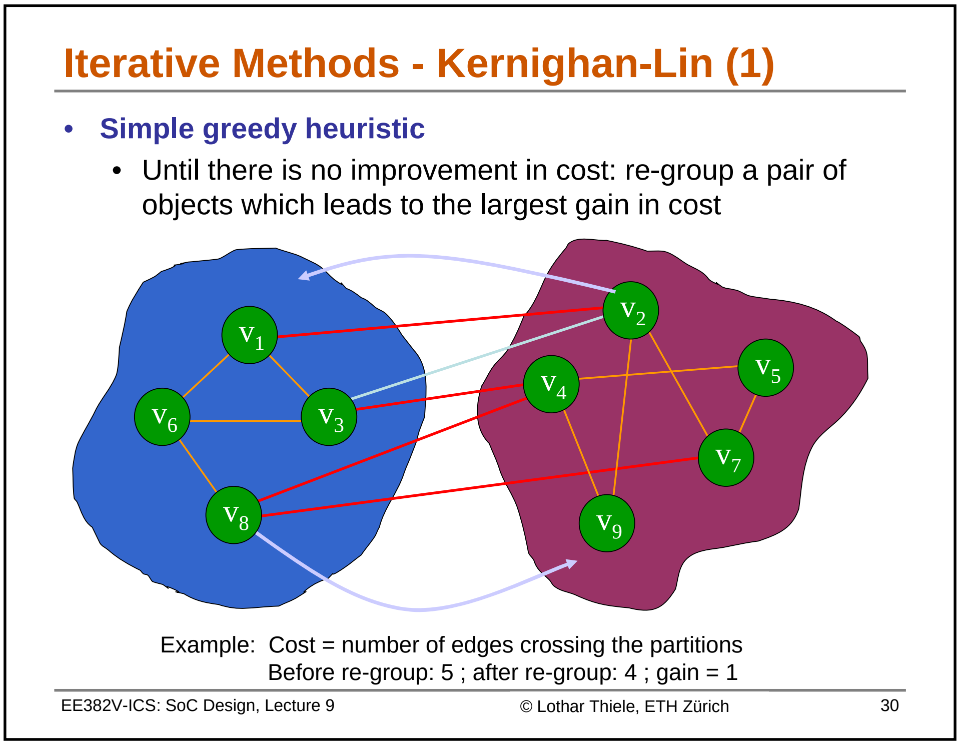
Hierarchical Clustering - Example (3)

$v_7 = v_6 \cup v_4$

EE382V-ICS: SoC Design, Lecture 9 © Lothar Thiele, ETH Zürich 28



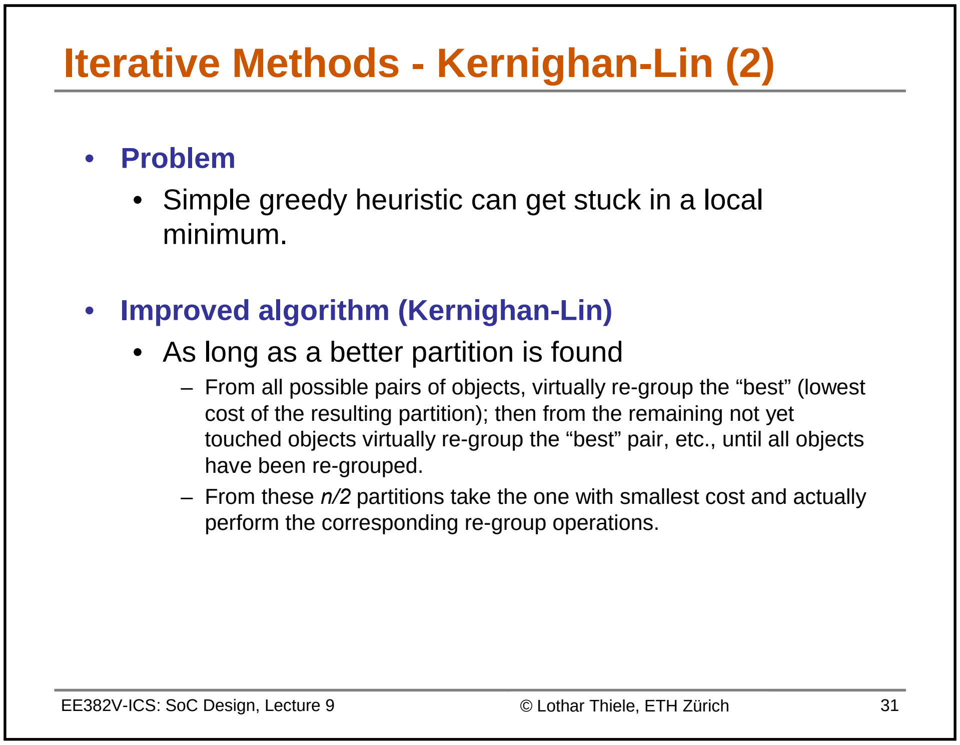
Hierarchical Clustering - Example (4)



Iterative Methods - Kernighan-Lin (1)

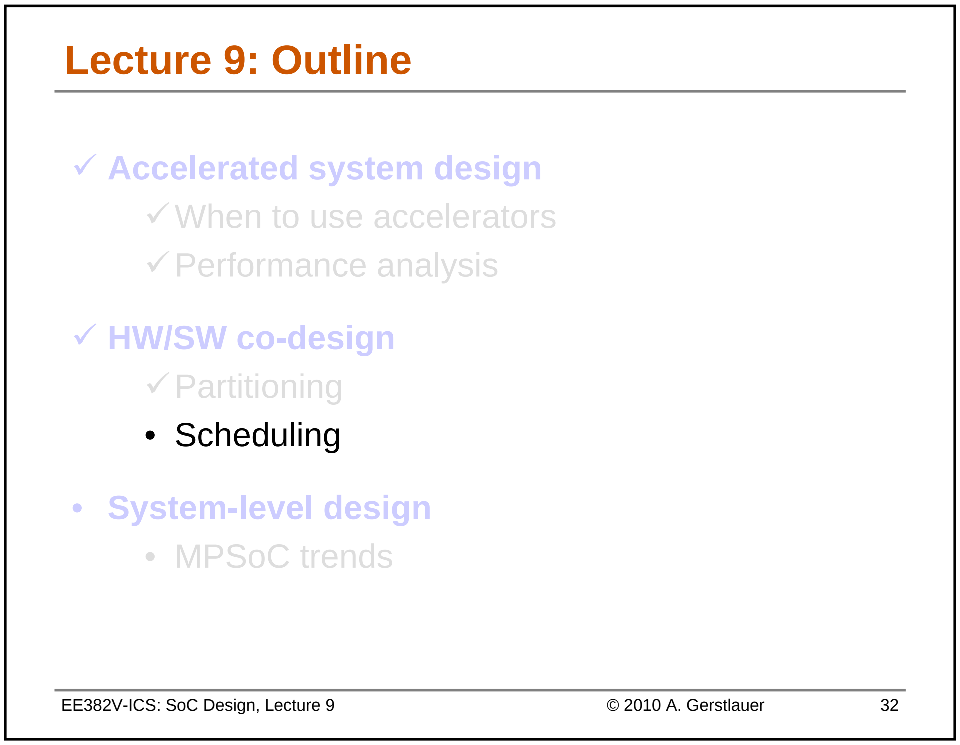
- **Simple greedy heuristic**
  - Until there is no improvement in cost: re-group a pair of objects which leads to the largest gain in cost

Example: Cost = number of edges crossing the partitions
Before re-group: 5 ; after re-group: 4 ; gain = 1



Iterative Methods - Kernighan-Lin (2)

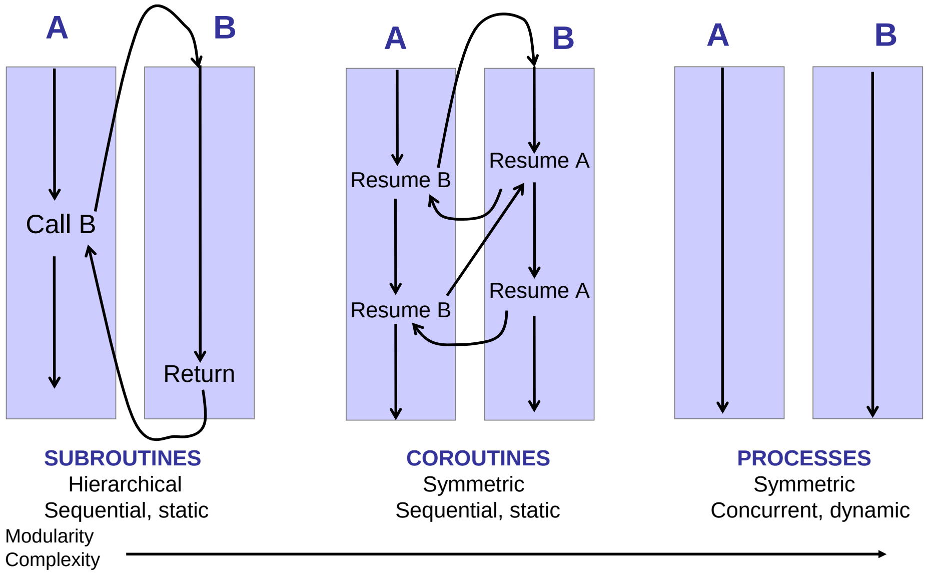
- **Problem**
  - Simple greedy heuristic can get stuck in a local minimum.
- **Improved algorithm (Kernighan-Lin)**
  - As long as a better partition is found
    - From all possible pairs of objects, virtually re-group the “best” (lowest cost of the resulting partition); then from the remaining not yet touched objects virtually re-group the “best” pair, etc., until all objects have been re-grouped.
    - From these *n/2* partitions take the one with smallest cost and actually perform the corresponding re-group operations.



Lecture 9: Outline

- ✓ **Accelerated system design**
  - ✓ When to use accelerators
  - ✓ Performance analysis
- ✓ **HW/SW co-design**
  - ✓ Partitioning
  - Scheduling
- **System-level design**
  - MPSoC trends

Multiplexing Software Modules



SUBROUTINES
Hierarchical
Sequential, static

COROUTINES
Symmetric
Sequential, static

PROCESSES
Symmetric
Concurrent, dynamic

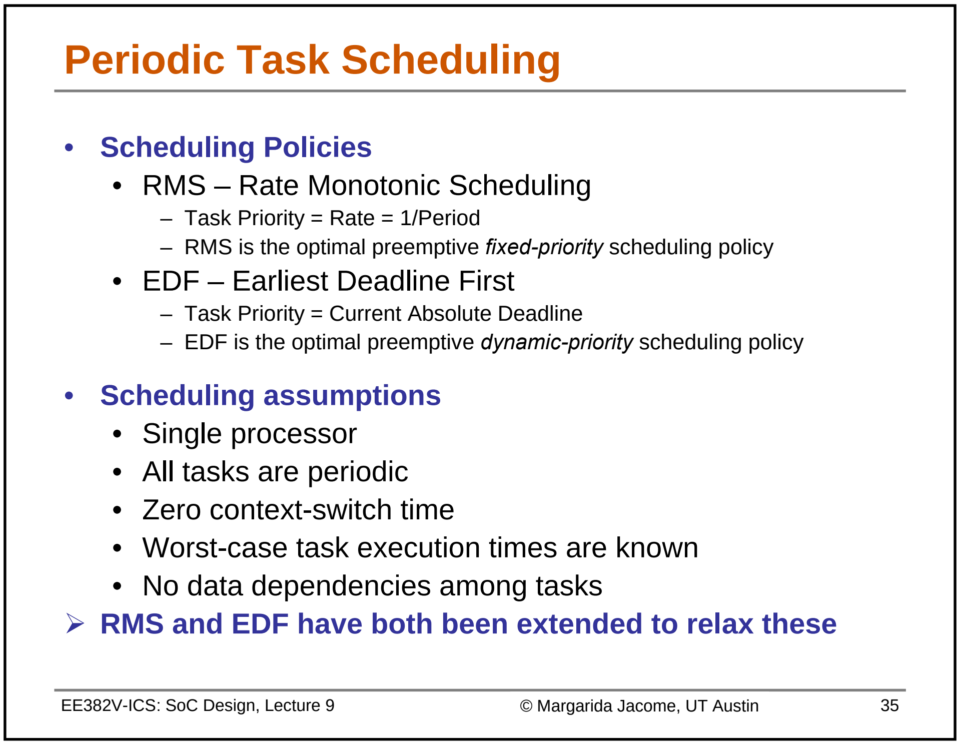
Modularity
Complexity

EE382V-ICS: SoC Design, Lecture 9 © Margarida Jacome, UT Austin 33

Real-Time Scheduling

- **Task types**
  - Periodic
    - Rate Monotonic Scheduling (RMS)
    - Earliest Deadline First (EDF)
  - Aperiodic/sporadic
    - EDF can be applied
- **Task dependencies**
  - Aperiodic tasks with precedence constraints
    - Latest Deadline First (LDF)
    - Modified EDF algorithm, or heuristics
- **Preemptive vs. non-preemptive**
  - Task with higher priority can preempt lower priority one
- **Mono- and multi-processor scheduling**
  - Centralized RTOS, symmetric multi-processing (SMP)
  - Distributed RTOS, asymmetric multi-processing (AMP)

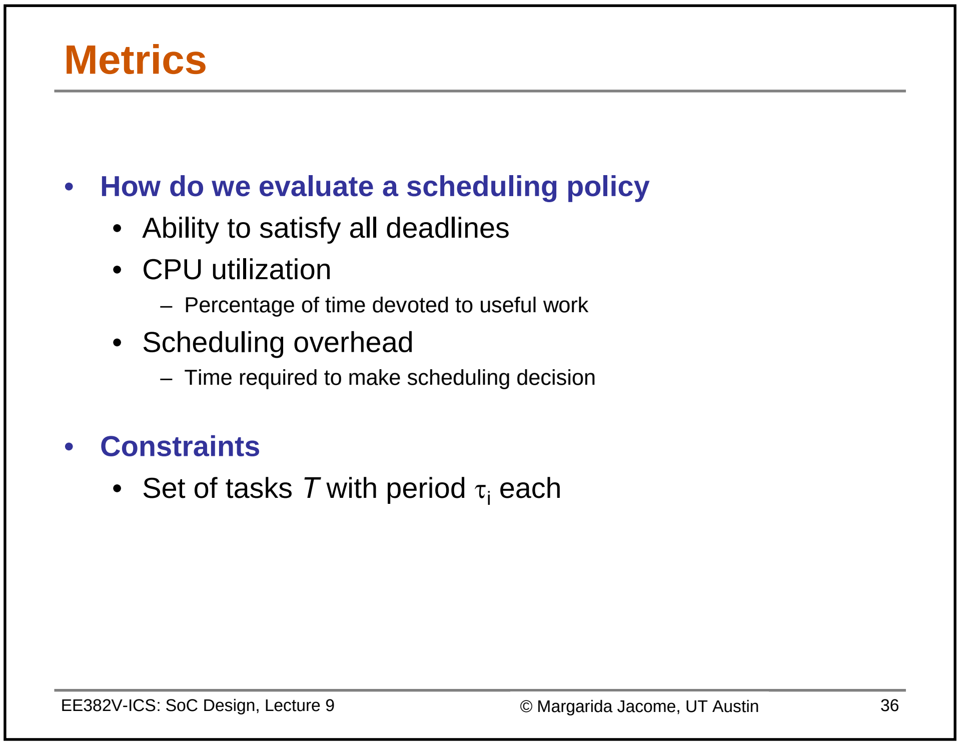
EE382V-ICS: SoC Design, Lecture 9 © 2010 A. Gerstlauer 34



Periodic Task Scheduling

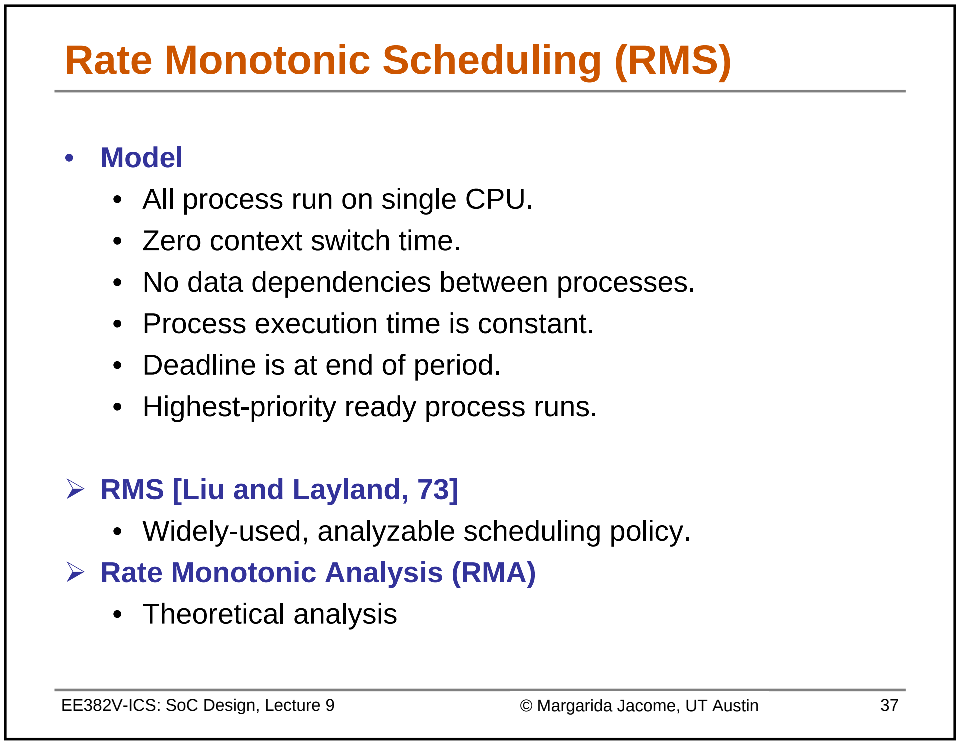
- **Scheduling Policies**
  - RMS – Rate Monotonic Scheduling
    - Task Priority = Rate = 1/Period
    - RMS is the optimal preemptive *fixed-priority* scheduling policy
  - EDF – Earliest Deadline First
    - Task Priority = Current Absolute Deadline
    - EDF is the optimal preemptive *dynamic-priority* scheduling policy
- **Scheduling assumptions**
  - Single processor
  - All tasks are periodic
  - Zero context-switch time
  - Worst-case task execution times are known
  - No data dependencies among tasks

➢ **RMS and EDF have both been extended to relax these**



Metrics

- **How do we evaluate a scheduling policy**
  - Ability to satisfy all deadlines
  - CPU utilization
    - Percentage of time devoted to useful work
  - Scheduling overhead
    - Time required to make scheduling decision
- **Constraints**
  - Set of tasks *T* with period τ_i each



Rate Monotonic Scheduling (RMS)

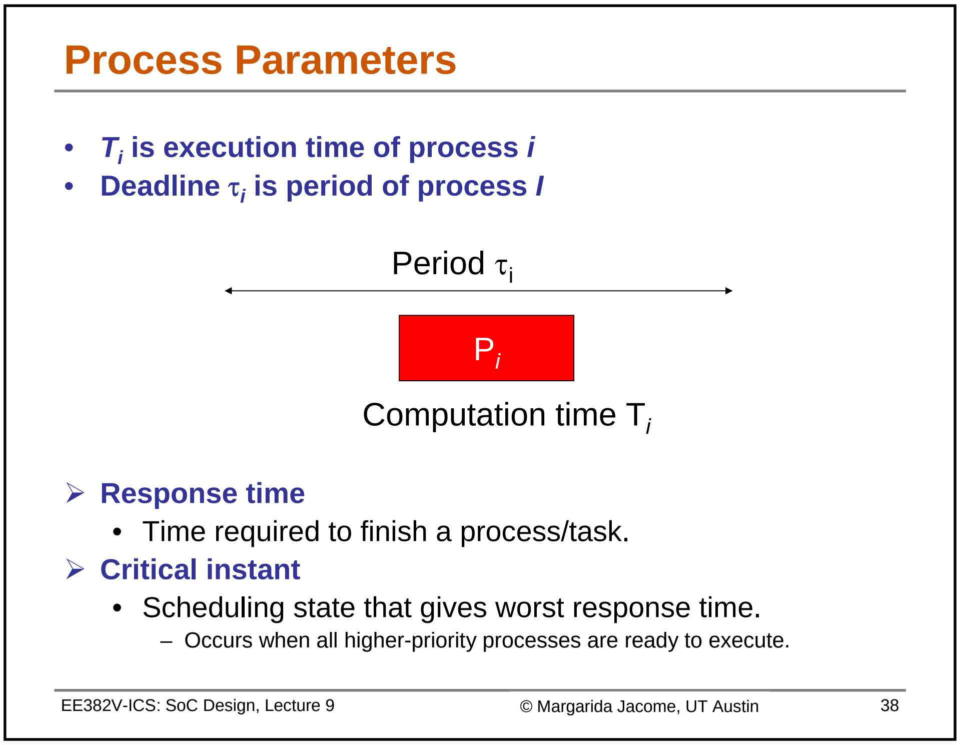
- **Model**
  - All process run on single CPU.
  - Zero context switch time.
  - No data dependencies between processes.
  - Process execution time is constant.
  - Deadline is at end of period.
  - Highest-priority ready process runs.

➢ **RMS [Liu and Layland, 73]**

- Widely-used, analyzable scheduling policy.

➢ **Rate Monotonic Analysis (RMA)**

- Theoretical analysis



Process Parameters

- **T_i is execution time of process *i***
- **Deadline τ_i is period of process *I***

Period τ_i

P_i

Computation time T_i

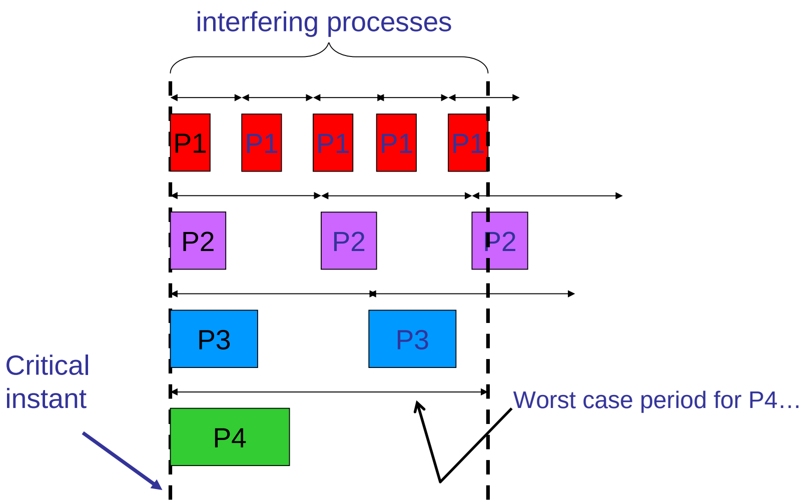
➢ **Response time**

- Time required to finish a process/task.

➢ **Critical instant**

- Scheduling state that gives worst response time.
  - Occurs when all higher-priority processes are ready to execute.

Critical Instant


interfering processes

Critical instant

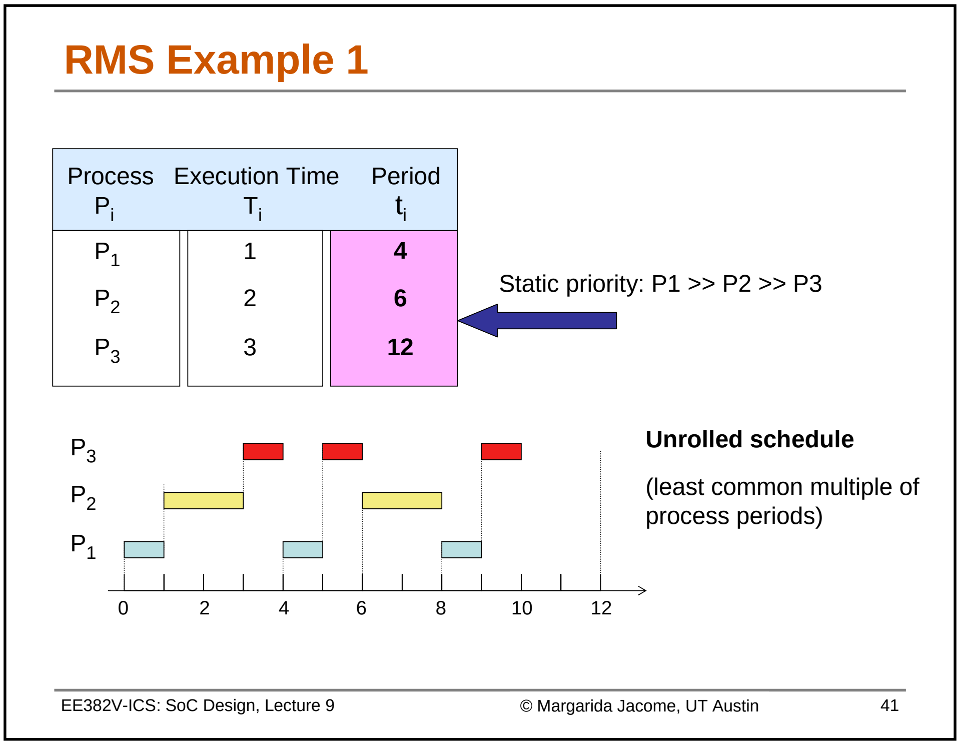
Worst case period for P4…

RMS Priorities

- **Optimal (fixed) priority assignment**
  - Shortest-period process gets highest priority
    - priority based preemption can be used…
  - Priority inversely proportional to period
  - Break ties arbitrarily

➢ **No fixed-priority scheme does better.**

  ➢ *RMS provides the highest worst case CPU utilization while ensuring that all processes meet their deadlines*



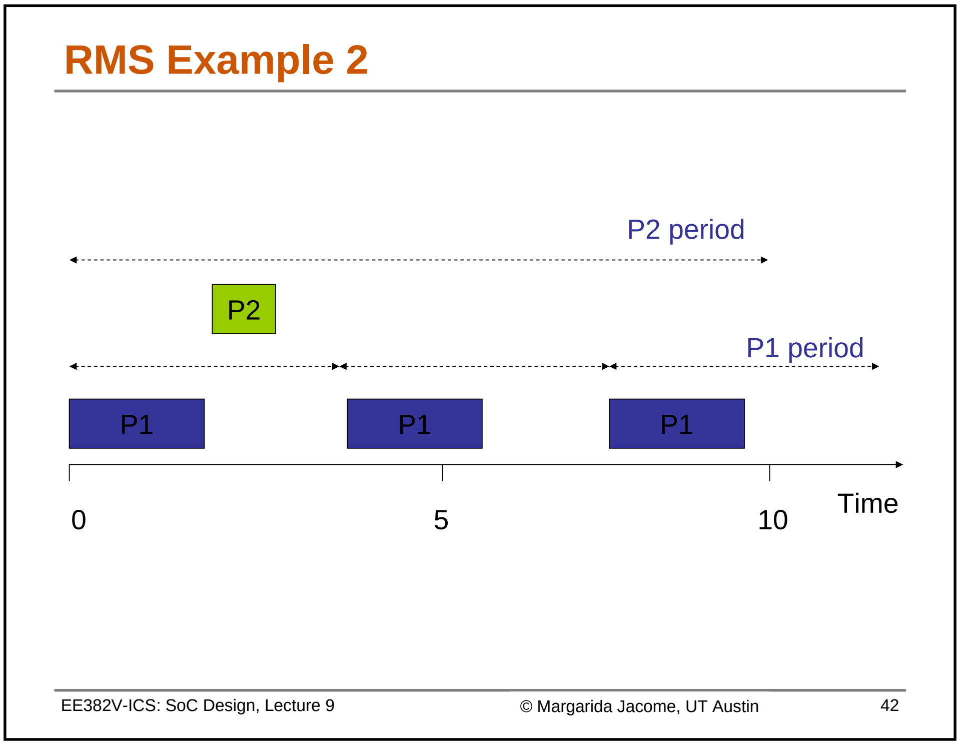
RMS Example 1

Process P_i	Execution Time T_i	Period t_i
P_1	1	**4**
P_2	2	**6**
P_3	3	**12**

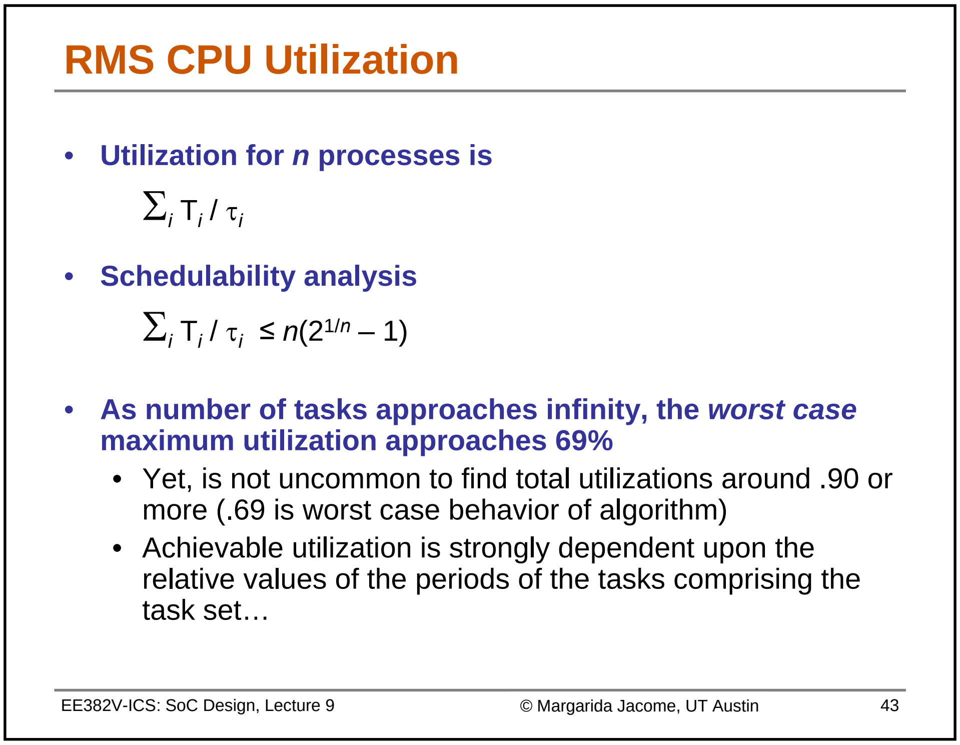
Static priority: P1 >> P2 >> P3

Unrolled schedule

(least common multiple of process periods)



RMS Example 2



RMS CPU Utilization

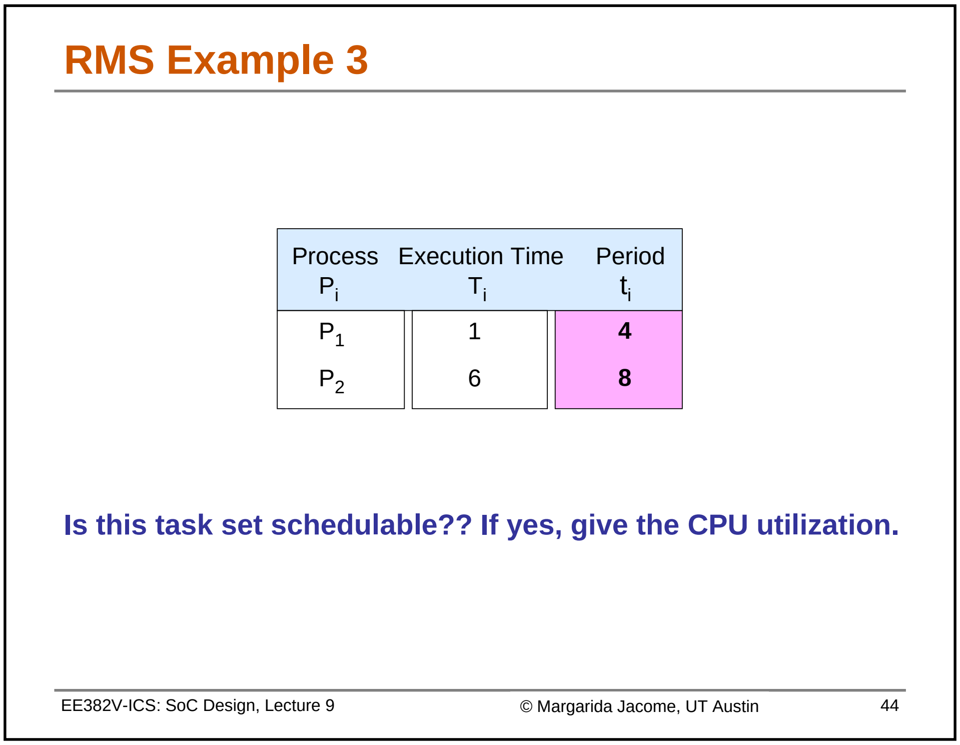
- **Utilization for *n* processes is**

  $\sum_i T_i / \tau_i$

- **Schedulability analysis**

  $\sum_i T_i / \tau_i \leq n(2^{1/n} - 1)$

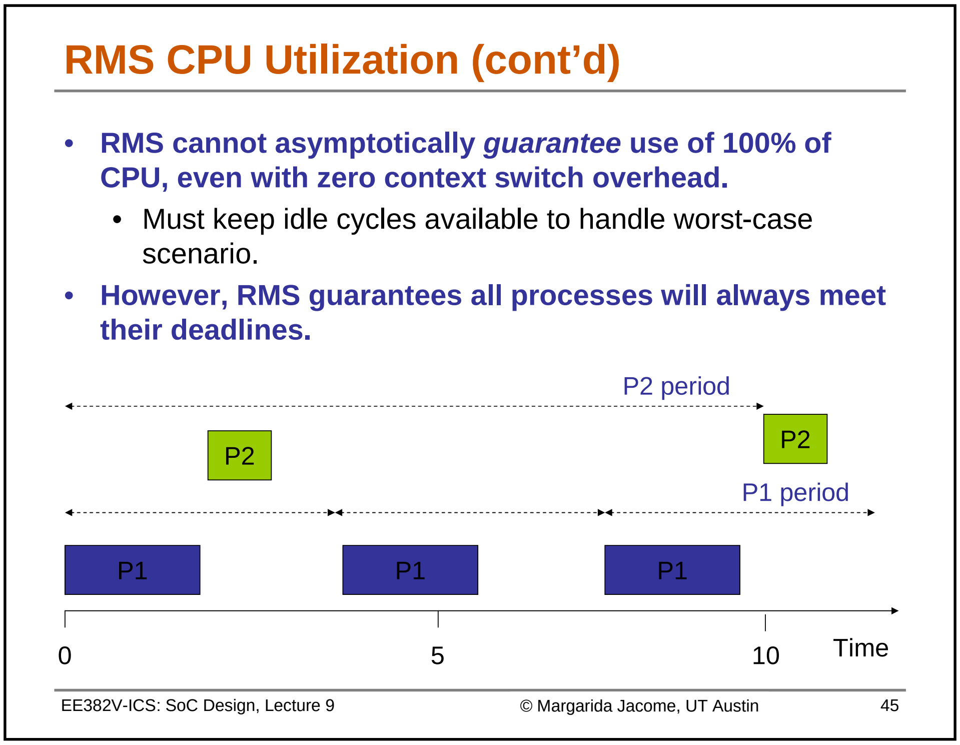
- **As number of tasks approaches infinity, the *worst case* maximum utilization approaches 69%**
  - Yet, is not uncommon to find total utilizations around .90 or more (.69 is worst case behavior of algorithm)
  - Achievable utilization is strongly dependent upon the relative values of the periods of the tasks comprising the task set…



RMS Example 3

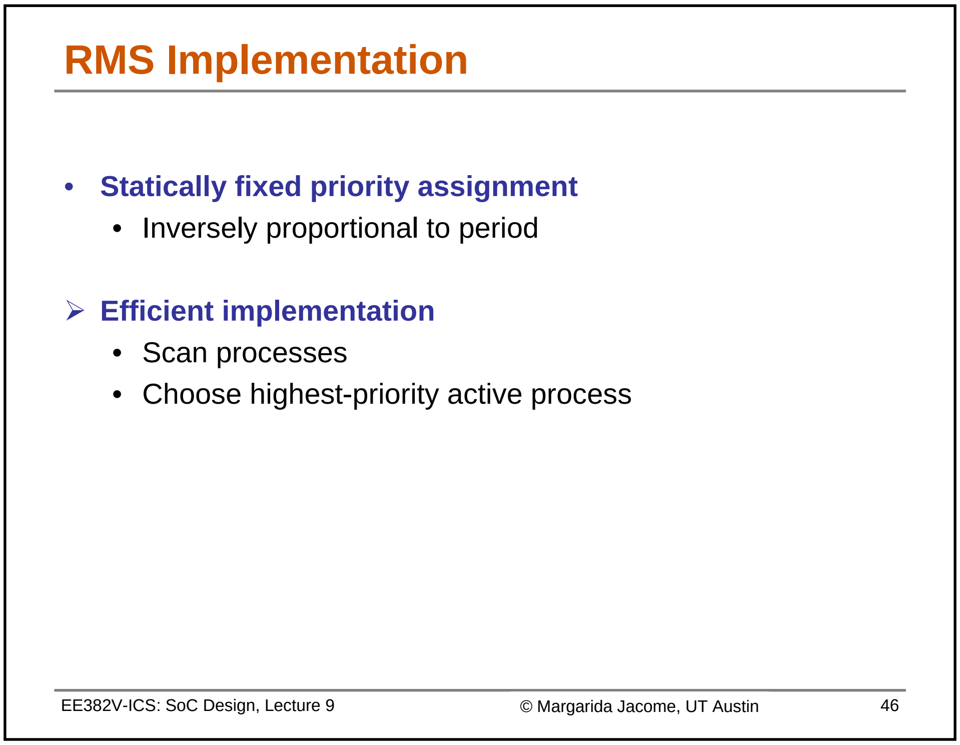
Process P_i	Execution Time T_i	Period t_i
P_1	1	**4**
P_2	6	**8**

Is this task set schedulable?? If yes, give the CPU utilization.



RMS CPU Utilization (cont'd)

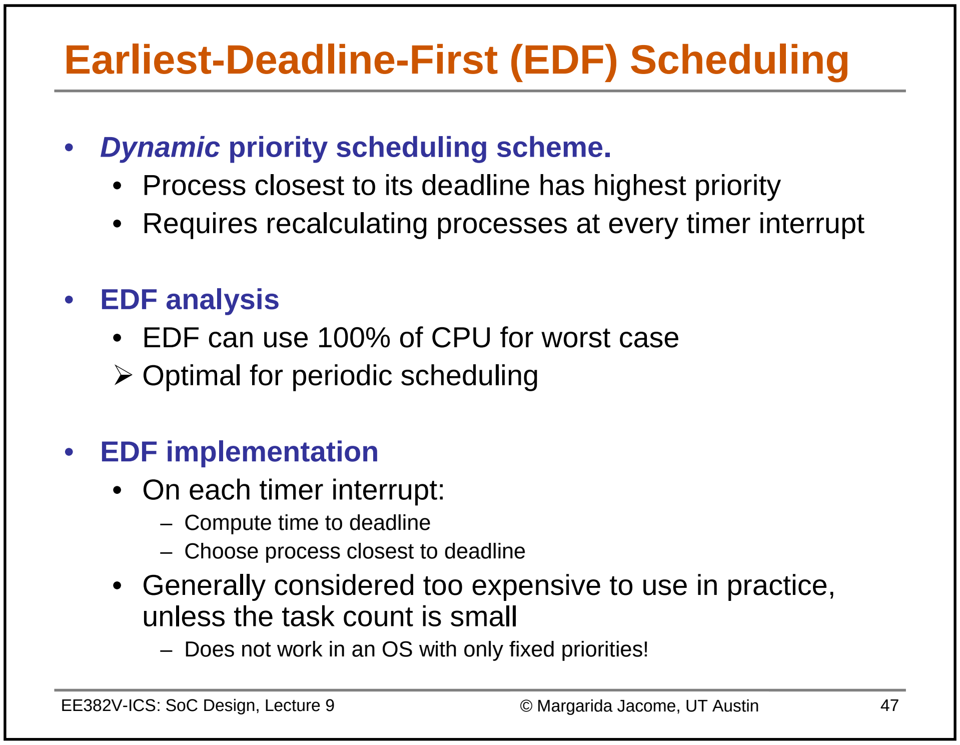
- **RMS cannot asymptotically *guarantee* use of 100% of CPU, even with zero context switch overhead.**
  - Must keep idle cycles available to handle worst-case scenario.
- **However, RMS guarantees all processes will always meet their deadlines.**



RMS Implementation

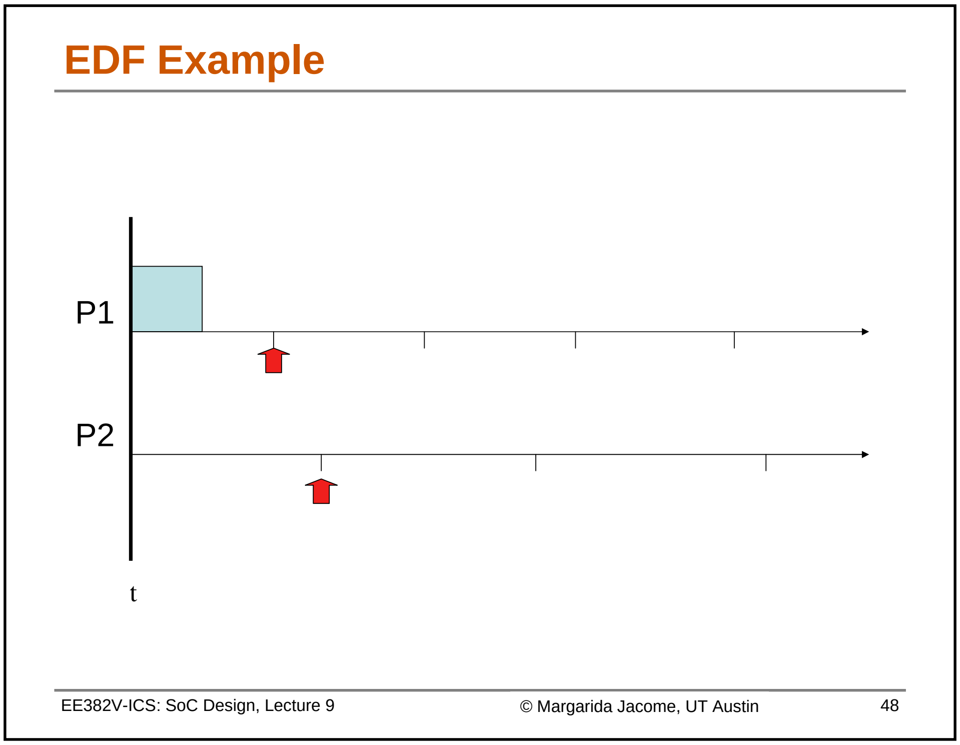
- **Statically fixed priority assignment**
  - Inversely proportional to period

➢ **Efficient implementation**
  - Scan processes
  - Choose highest-priority active process

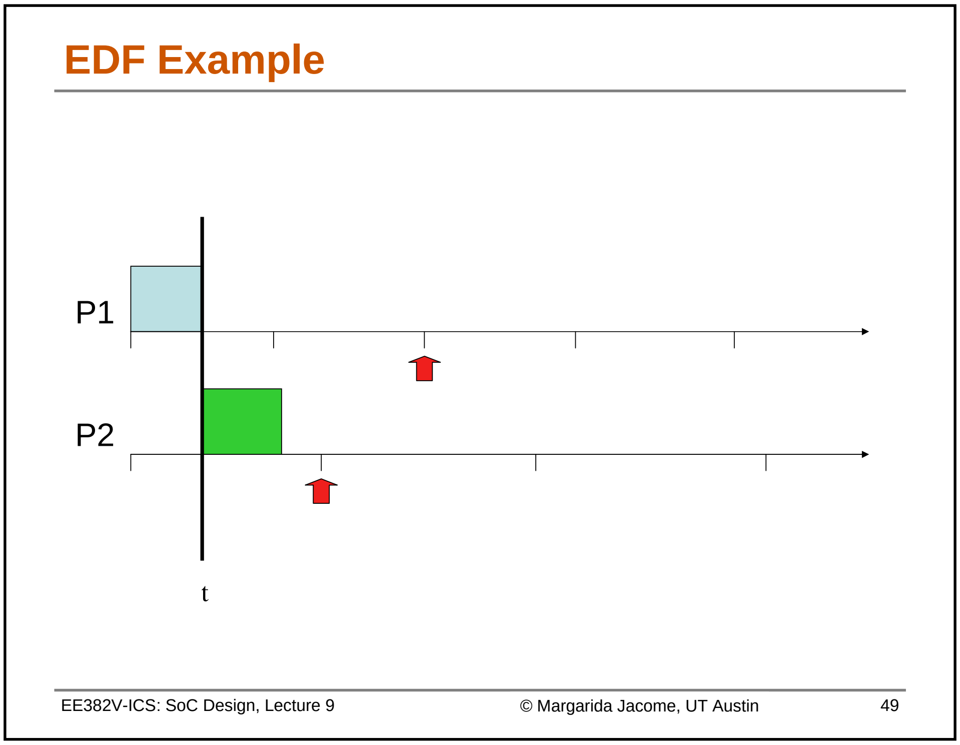


Earliest-Deadline-First (EDF) Scheduling

- ***Dynamic* priority scheduling scheme.**
  - Process closest to its deadline has highest priority
  - Requires recalculating processes at every timer interrupt
- **EDF analysis**
  - EDF can use 100% of CPU for worst case
  - ➢ Optimal for periodic scheduling
- **EDF implementation**
  - On each timer interrupt:
    - Compute time to deadline
    - Choose process closest to deadline
  - Generally considered too expensive to use in practice, unless the task count is small
    - Does not work in an OS with only fixed priorities!

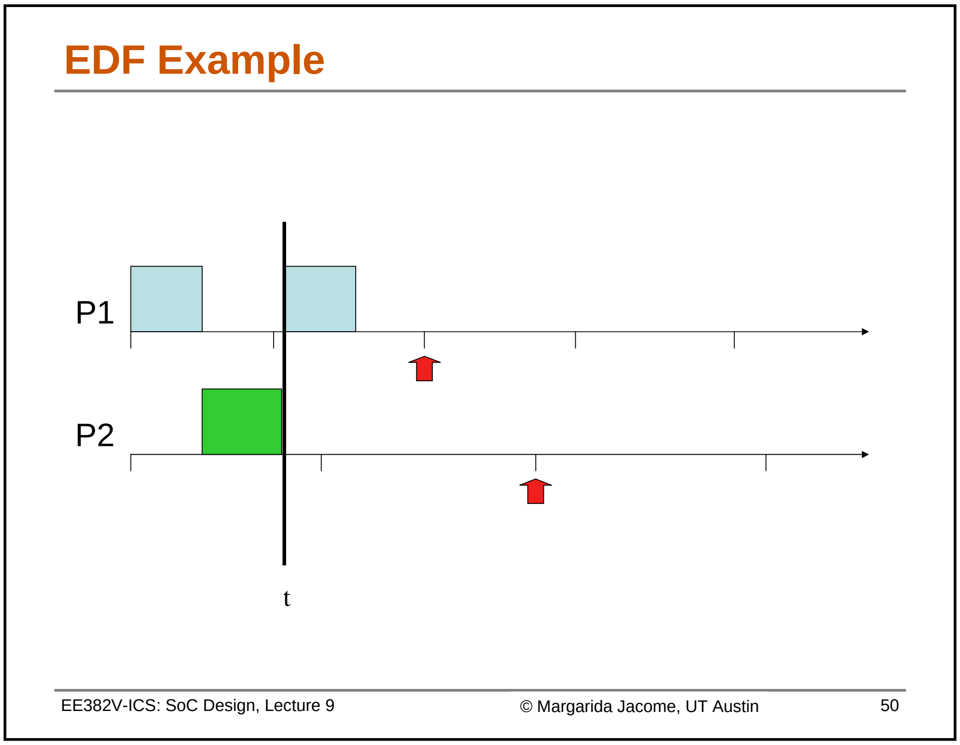


EDF Example



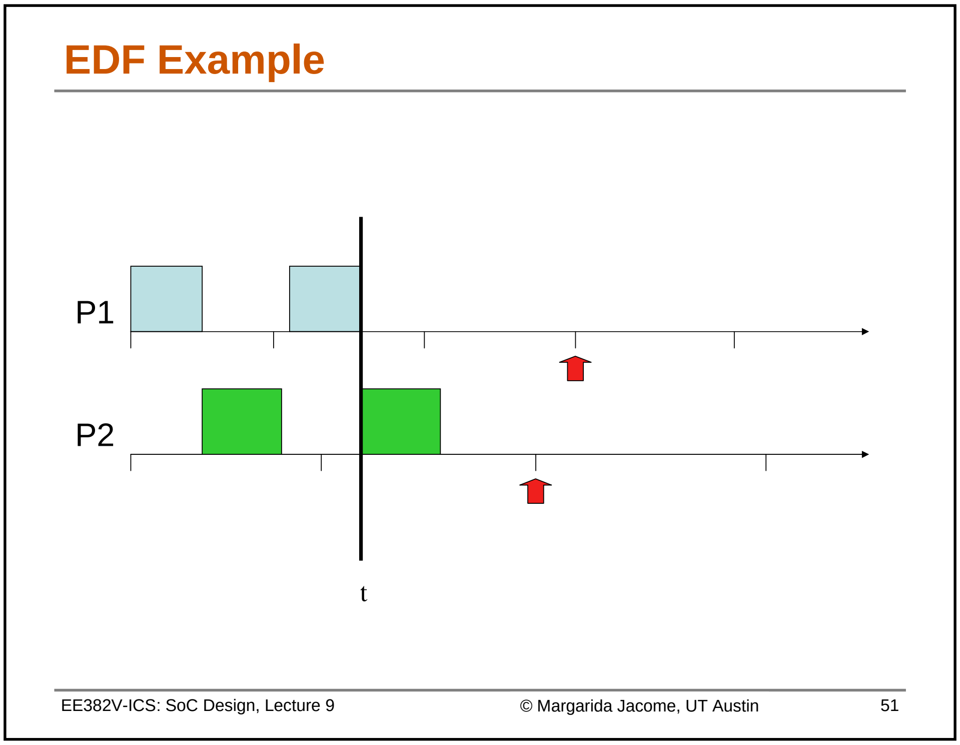
EDF Example

EE382V-ICS: SoC Design, Lecture 9 © Margarida Jacome, UT Austin 49

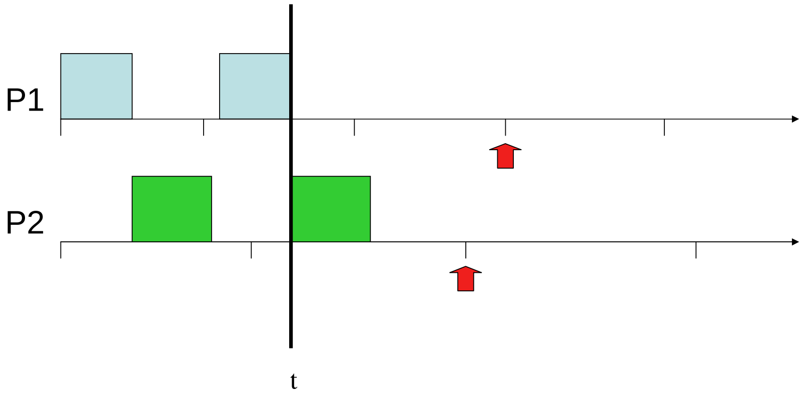


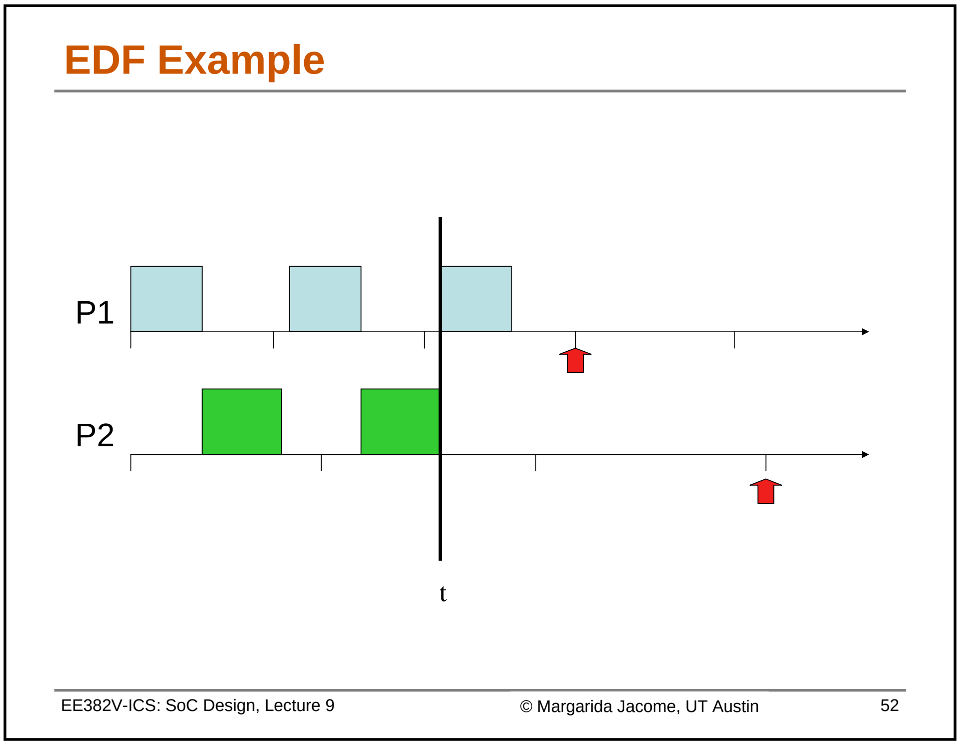
EDF Example

EE382V-ICS: SoC Design, Lecture 9 © Margarida Jacome, UT Austin 50

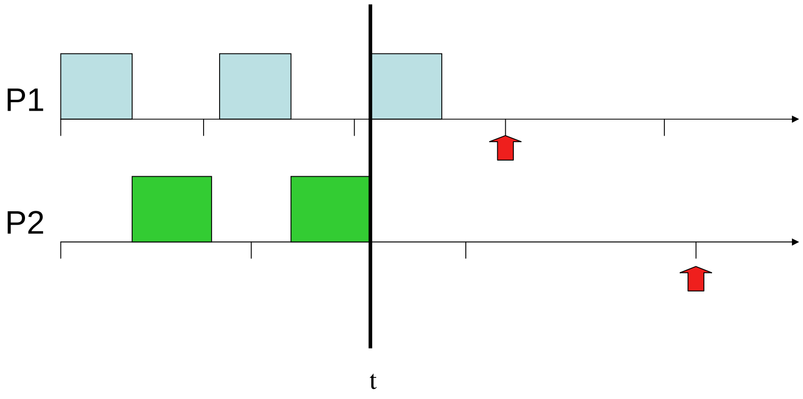


EDF Example

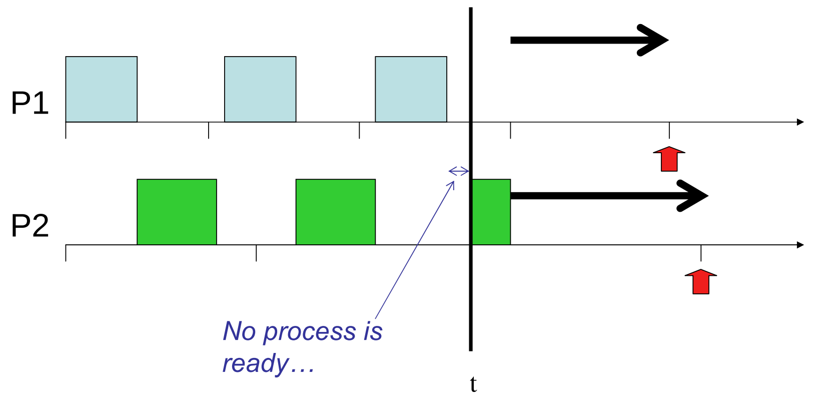




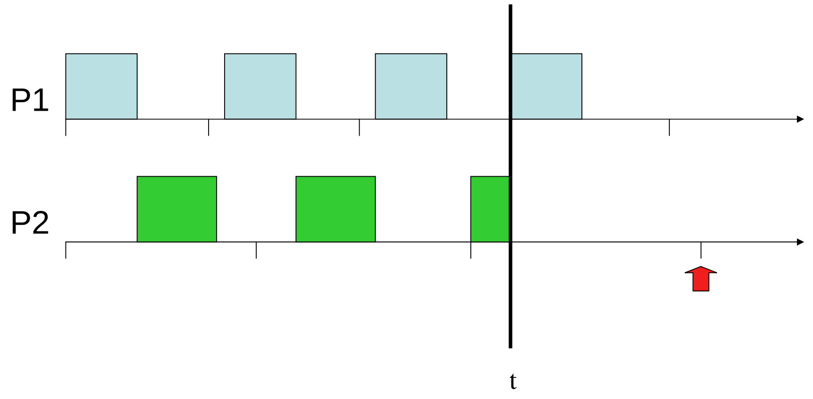
EDF Example



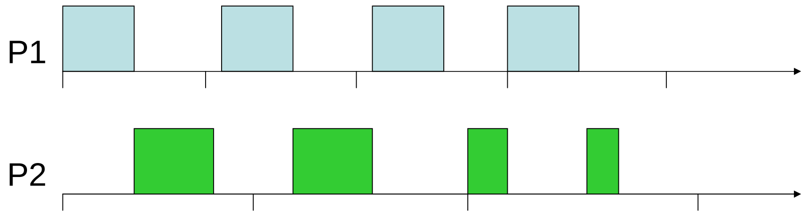
EDF Example



EDF Example



EDF Example



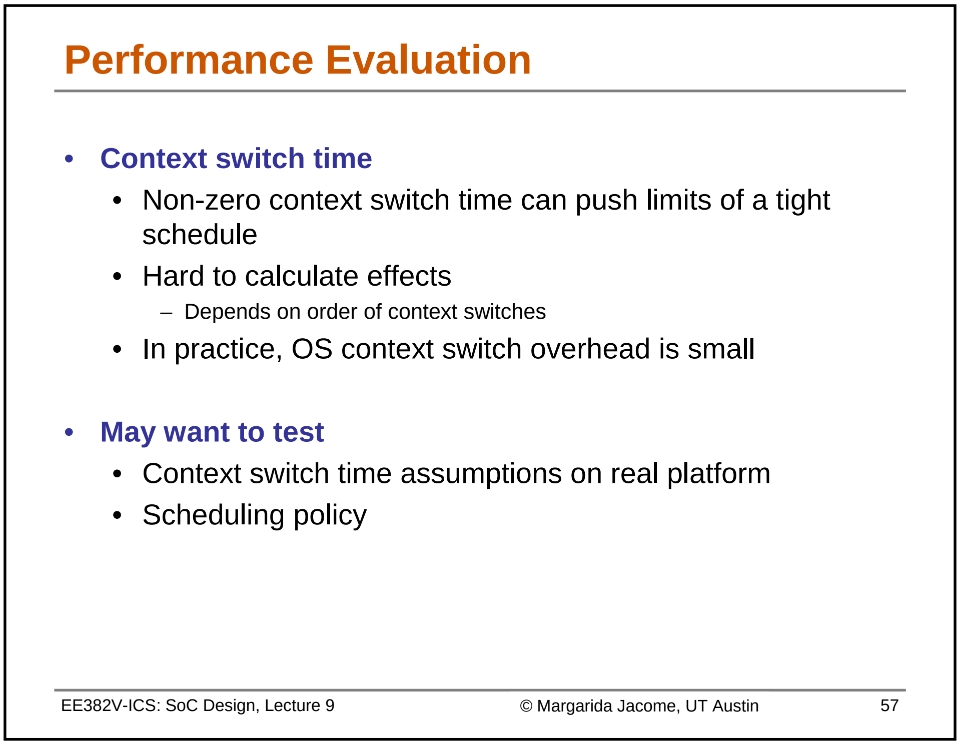
Priority Inversion

- **Low-priority process keeps high-priority process from running.**
  - Improper use of system resources can cause scheduling problems
    - Low-priority process grabs I/O device.
    - High-priority device needs I/O device, but can't get it until low-priority process is done.
  - ➢ Can cause deadlock

➢ **Give priorities to system resources**

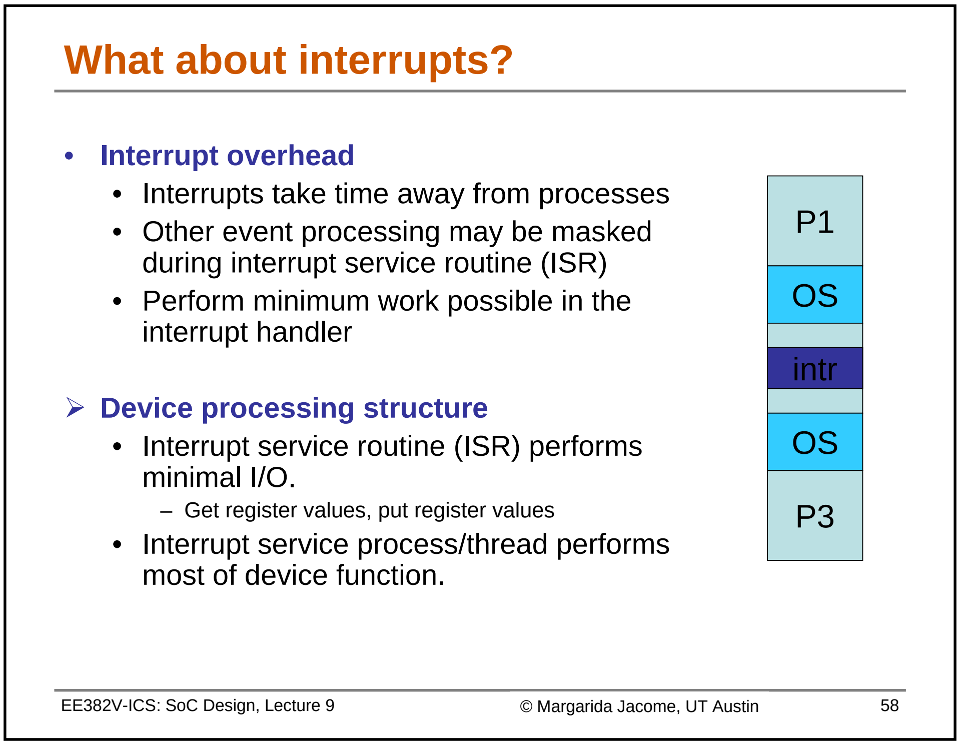
➢ **Have process inherit the priority of a resource that it requests**

- ➢ Low-priority process inherits priority of device if higher



Performance Evaluation

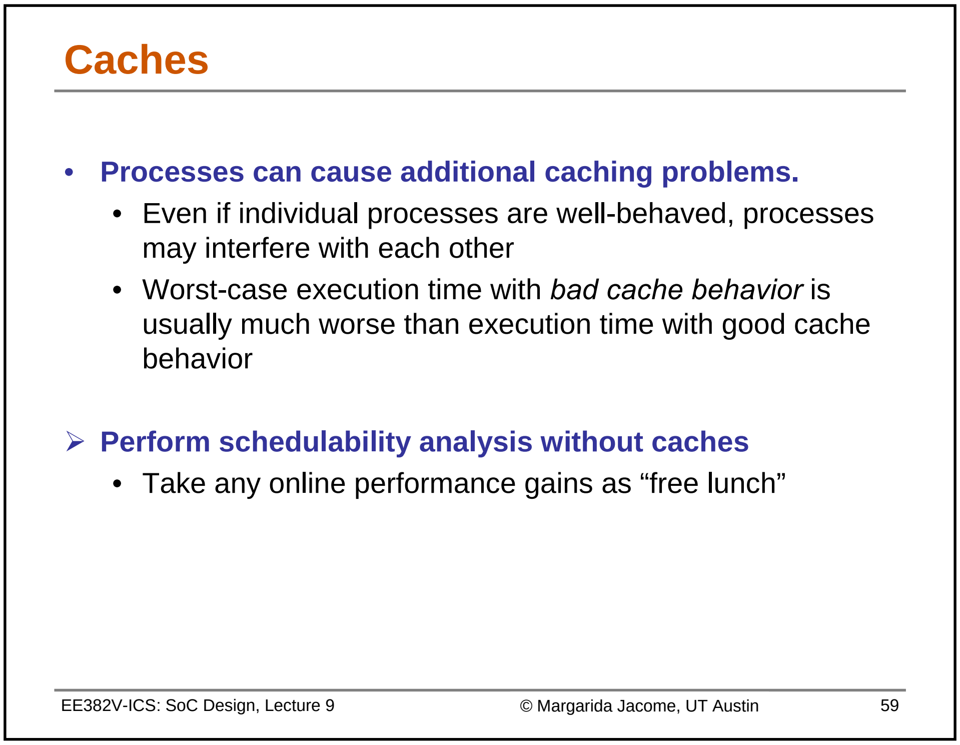
- **Context switch time**
  - Non-zero context switch time can push limits of a tight schedule
  - Hard to calculate effects
    - Depends on order of context switches
  - In practice, OS context switch overhead is small
- **May want to test**
  - Context switch time assumptions on real platform
  - Scheduling policy



What about interrupts?

- **Interrupt overhead**
  - Interrupts take time away from processes
  - Other event processing may be masked during interrupt service routine (ISR)
  - Perform minimum work possible in the interrupt handler
- ➢ **Device processing structure**
  - Interrupt service routine (ISR) performs minimal I/O.
    - Get register values, put register values
  - Interrupt service process/thread performs most of device function.

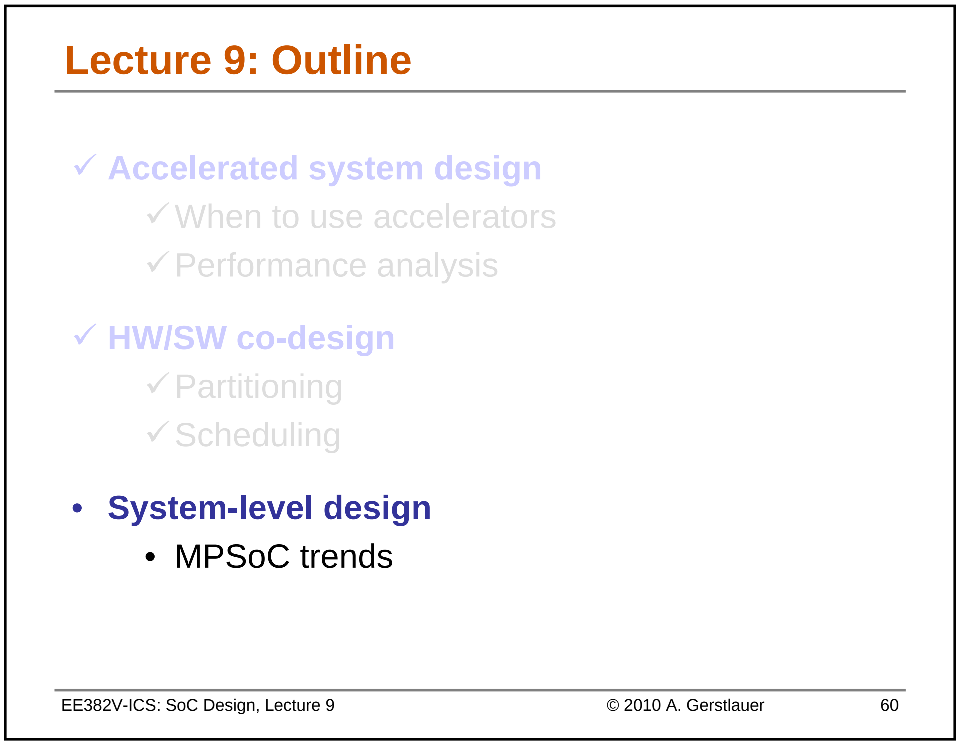
P1
OS
intr
OS
P3



Caches

- **Processes can cause additional caching problems.**
  - Even if individual processes are well-behaved, processes may interfere with each other
  - Worst-case execution time with *bad cache behavior* is usually much worse than execution time with good cache behavior

➢ **Perform schedulability analysis without caches**
  - Take any online performance gains as "free lunch"

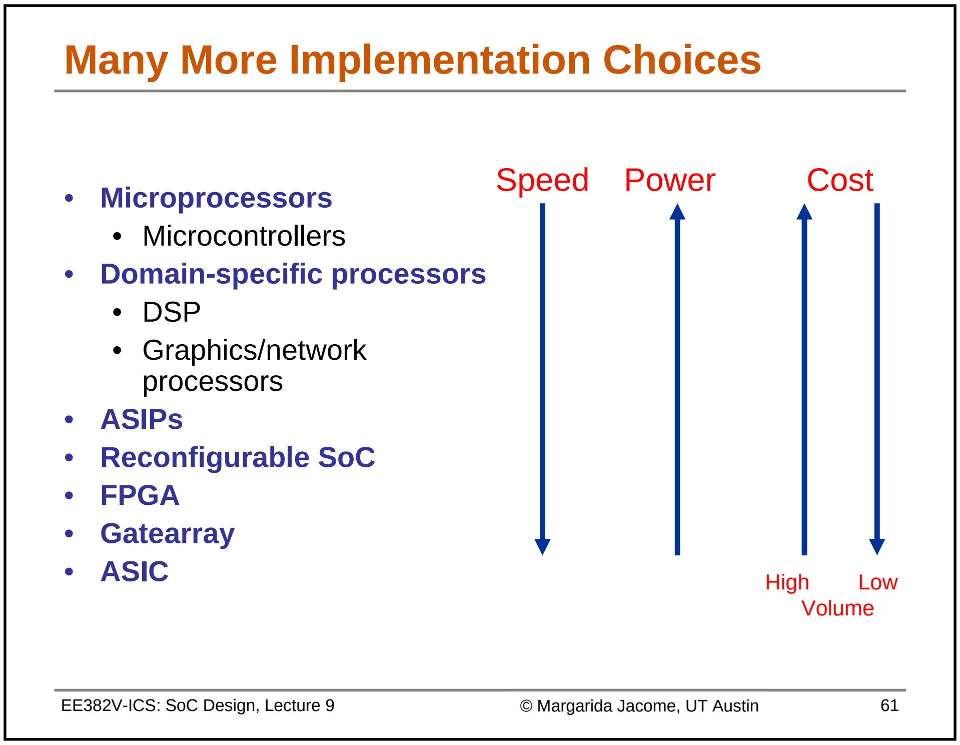


Lecture 9: Outline

✓ **Accelerated system design**
  - ✓ When to use accelerators
  - ✓ Performance analysis

✓ **HW/SW co-design**
  - ✓ Partitioning
  - ✓ Scheduling

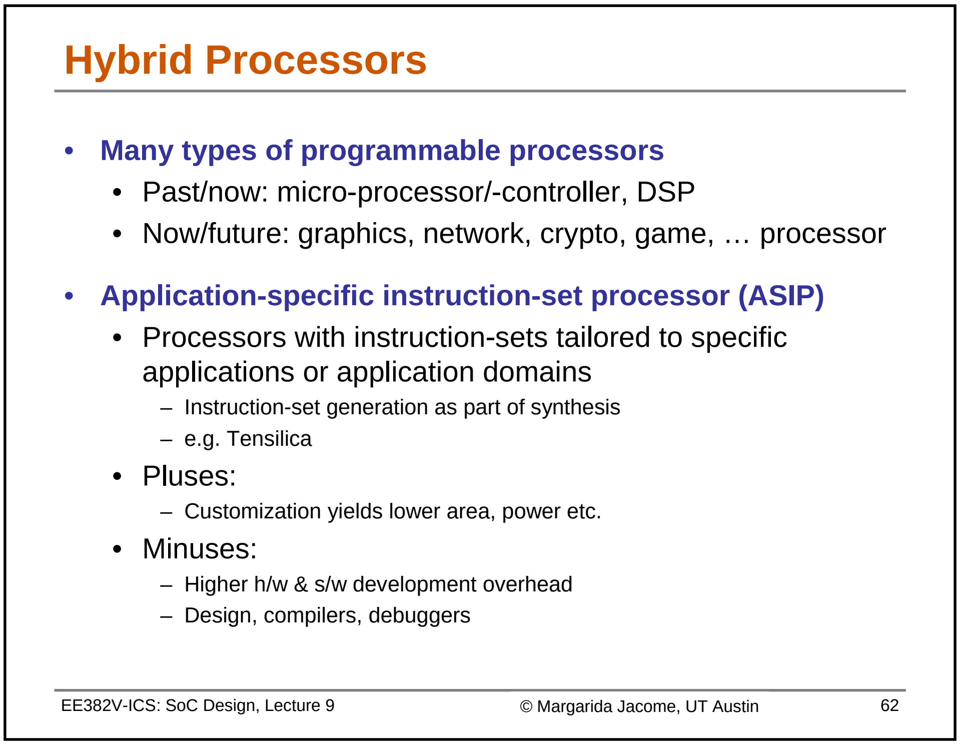
- **System-level design**
  - MPSoC trends



Many More Implementation Choices

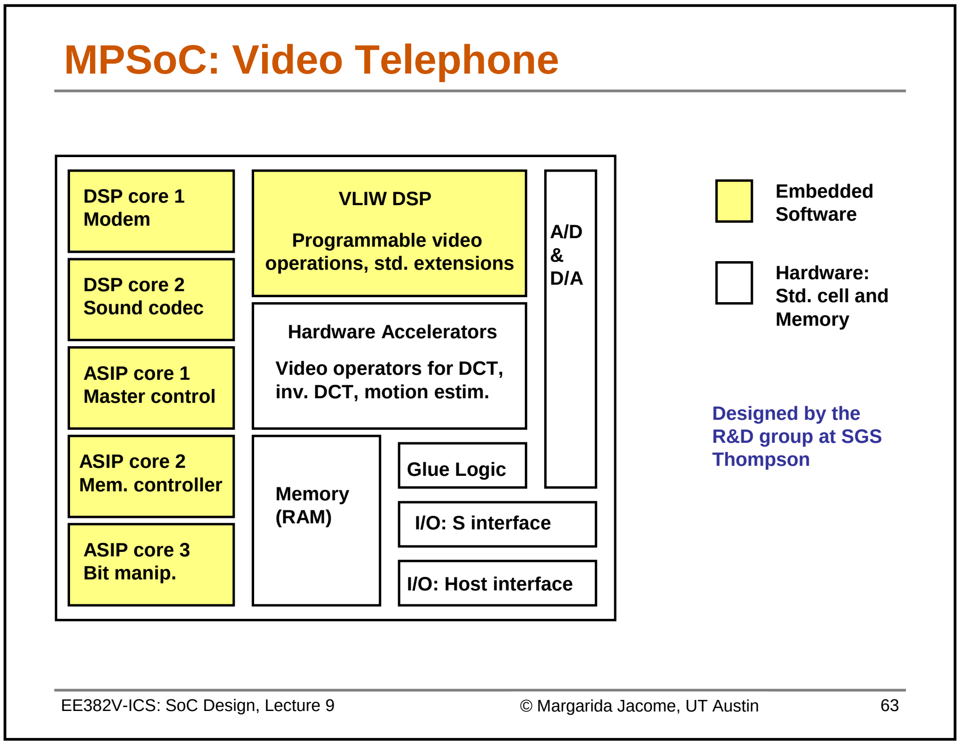
- **Microprocessors**
  - Microcontrollers
- **Domain-specific processors**
  - DSP
  - Graphics/network processors
- **ASIPs**
- **Reconfigurable SoC**
- **FPGA**
- **Gatearray**
- **ASIC**

Speed
Power
Cost
High
Low
Volume



Hybrid Processors

- **Many types of programmable processors**
  - Past/now: micro-processor/-controller, DSP
  - Now/future: graphics, network, crypto, game, … processor
- **Application-specific instruction-set processor (ASIP)**
  - Processors with instruction-sets tailored to specific applications or application domains
    - Instruction-set generation as part of synthesis
    - e.g. Tensilica
  - Pluses:
    - Customization yields lower area, power etc.
  - Minuses:
    - Higher h/w & s/w development overhead
    - Design, compilers, debuggers



MPSoC: Video Telephone

DSP core 1 Modem

DSP core 2 Sound codec

ASIP core 1 Master control

ASIP core 2 Mem. controller

ASIP core 3 Bit manip.

VLIW DSP

Programmable video operations, std. extensions

Hardware Accelerators

Video operators for DCT, inv. DCT, motion estim.

A/D & D/A

Memory (RAM)

Glue Logic

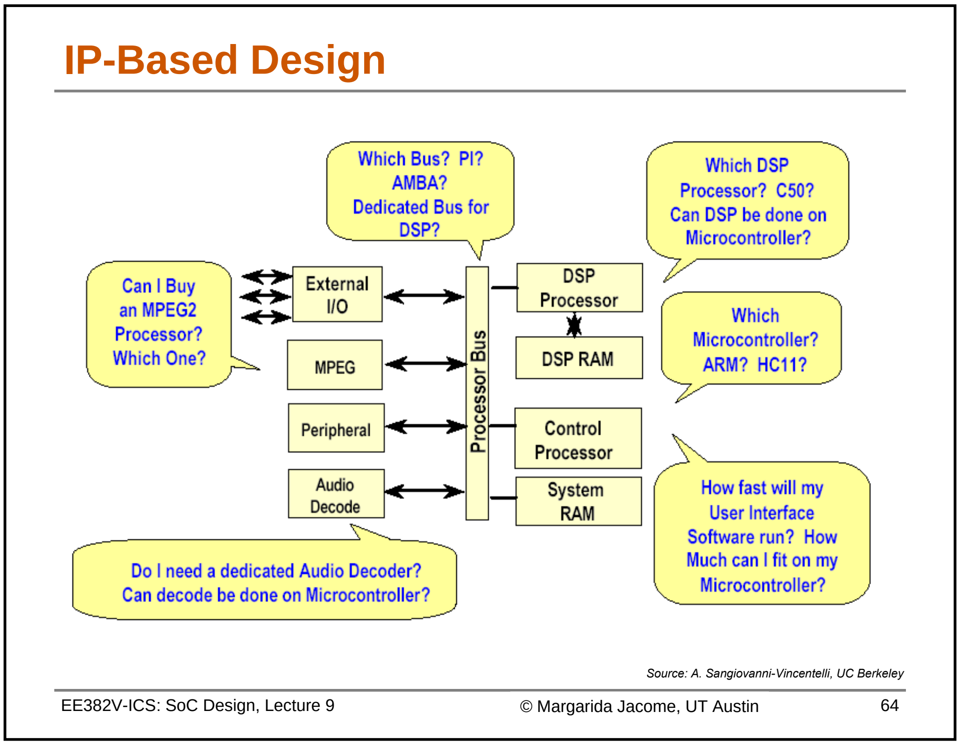
I/O: S interface

I/O: Host interface

🟨 Embedded Software

□ Hardware: Std. cell and Memory

Designed by the R&D group at SGS Thompson



IP-Based Design

Which Bus? PI? AMBA? Dedicated Bus for DSP?

Which DSP Processor? C50? Can DSP be done on Microcontroller?

Can I Buy an MPEG2 Processor? Which One?

External I/O

MPEG

Peripheral

Audio Decode

Processor Bus

DSP Processor

DSP RAM

Control Processor

System RAM

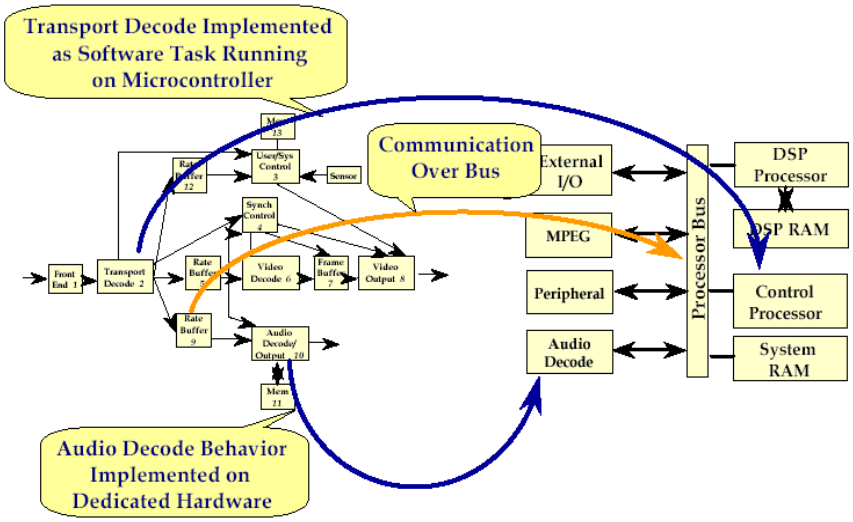
Which Microcontroller? ARM? HC11?

How fast will my User Interface Software run? How Much can I fit on my Microcontroller?

Do I need a dedicated Audio Decoder? Can decode be done on Microcontroller?

Source: A. Sangiovanni-Vincentelli, UC Berkeley

Platform Mapping

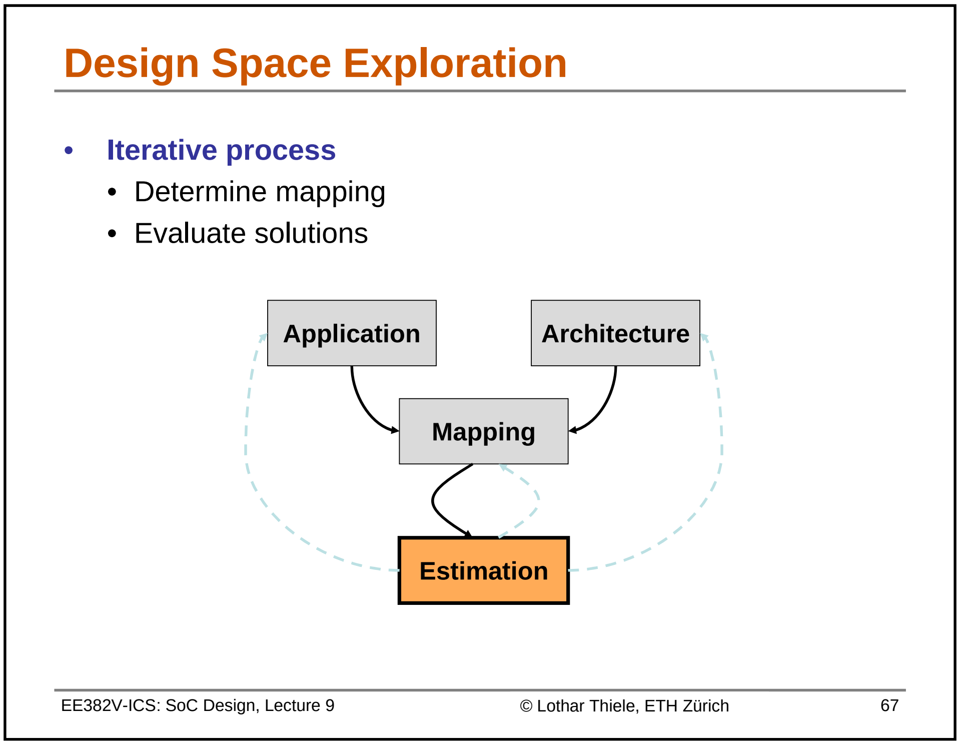


Source: A. Sangiovanni-Vincentelli, UC Berkeley

MPSoC Synthesis Approaches

- **Design space exploration**
  - Multi-objective
  - Pareto optimality
  - ➢ Traditional HW/SW co-design approaches not sufficient

➢ **Set-based approaches**
  - “Intelligent”, randomized search
  - ➢ Evolutionary Algorithms (EA),
    Particle Swarm Optimization (PSO),
    Ant Colony Optimization (ACO)



Design Space Exploration

- **Iterative process**
  - Determine mapping
  - Evaluate solutions

Application

Architecture

Mapping

Estimation

EE382V-ICS: SoC Design, Lecture 9 © Lothar Thiele, ETH Zürich 67