Other OS functions

- Date/time.
- File system.
- Networking.
- Security.

6.3 Priority based scheduling

- Scheduling policies: static or dynamic
- RMS: rate monotonic scheduling
  - static
- EDF: earliest deadline first scheduling
  - dynamic
- Scheduling modeling assumptions exist.

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.

Rate monotonic scheduling

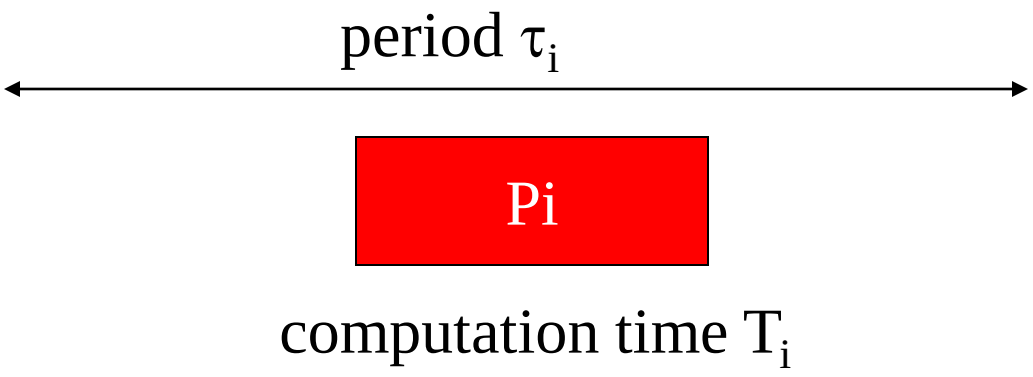
- RMS (Liu and Layland): widely-used, analyzable scheduling policy.
- Analysis is known as Rate Monotonic Analysis (RMA).

RMA model

- All process run on single CPU.
- Assume that 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.

Process parameters

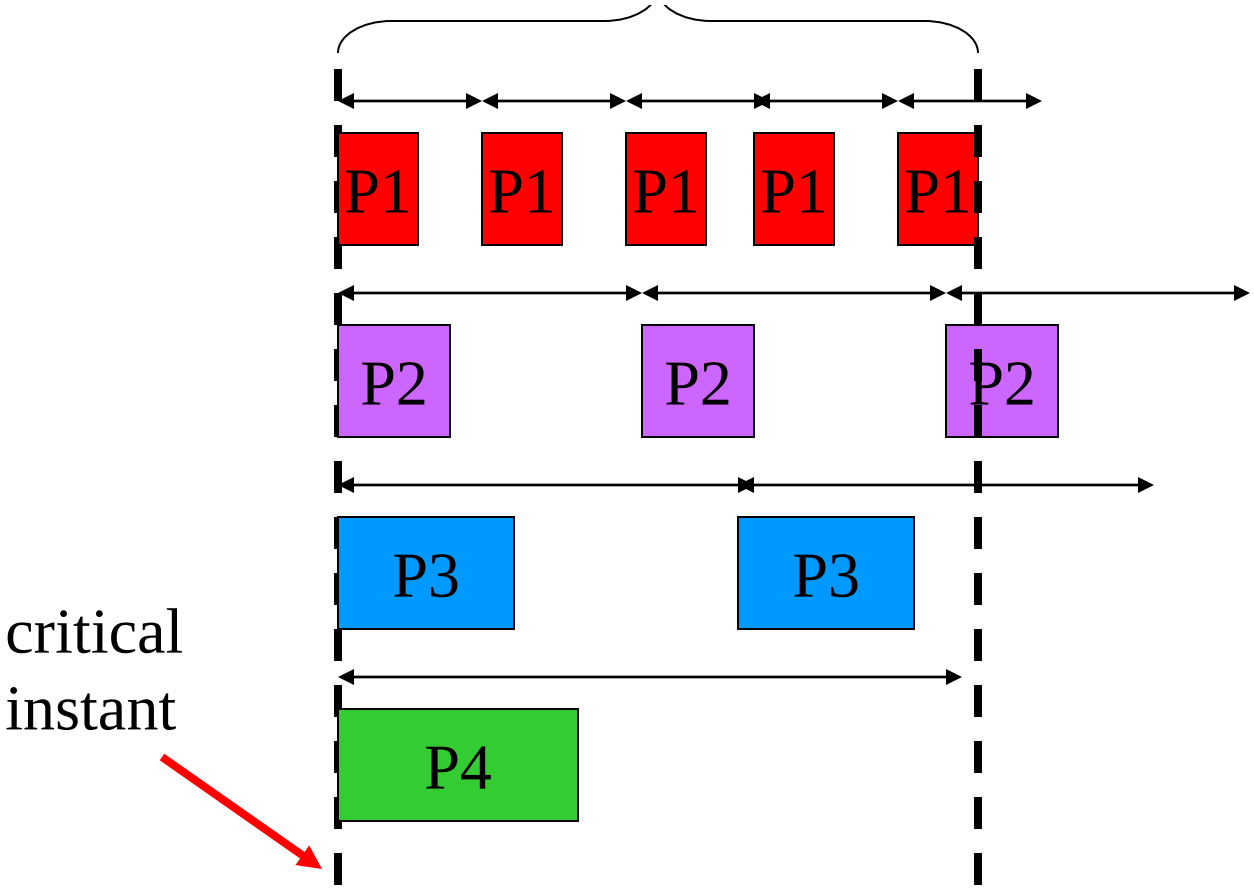
- T_i is computation time of process i; τ_i is period of process i.



Rate-monotonic analysis

- Response time: time required to finish process.
- Critical instant: scheduling state that gives worst response time.
- Critical instant occurs when all higher-priority processes are ready to execute.

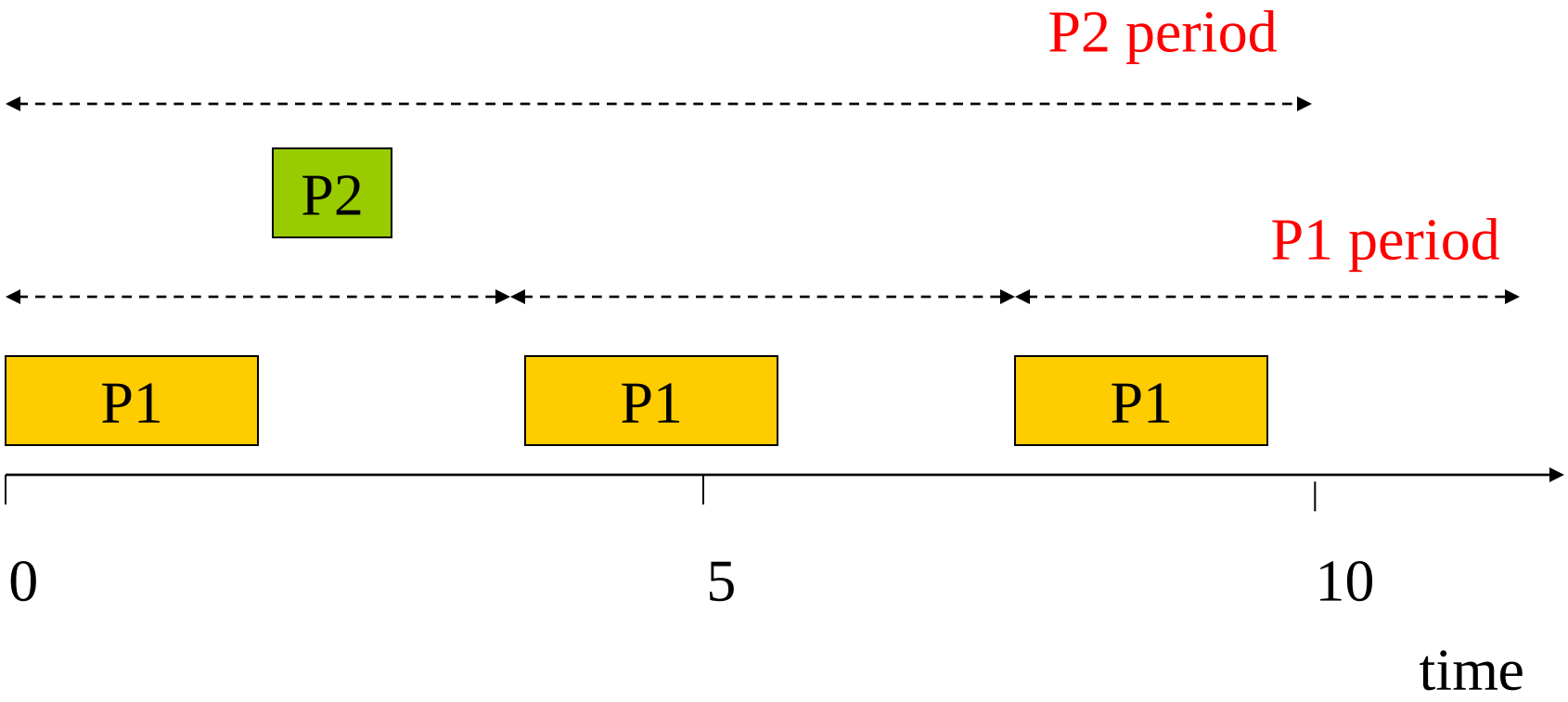
Critical instant



RMS priorities

- Optimal (fixed) priority assignment:
  - shortest-period process gets highest priority;
  - priority inversely proportional to period;
  - break ties arbitrarily.
- No fixed-priority scheme does better than RMS.

RMS example



RMS CPU utilization

- Utilization for n processes is
  - $\sum_i T_i / \tau_i$
- As number of tasks approaches infinity, maximum utilization approaches 69%.

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1) \leq \ln 2 \cong 0.6931$$

RMS CPU utilization, cont'd.

- RMS cannot use 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

- Efficient implementation:
  - scan processes;
  - choose highest-priority active process.

Earliest-deadline-first (EDF)

- EDF: 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.
- But EDF may fail to miss a deadline.

EDF implementation

- On each timer interrupt:
  - compute time to deadline;
  - choose process closest to deadline.
- Generally considered too expensive to use in practice.

Fixing scheduling problems

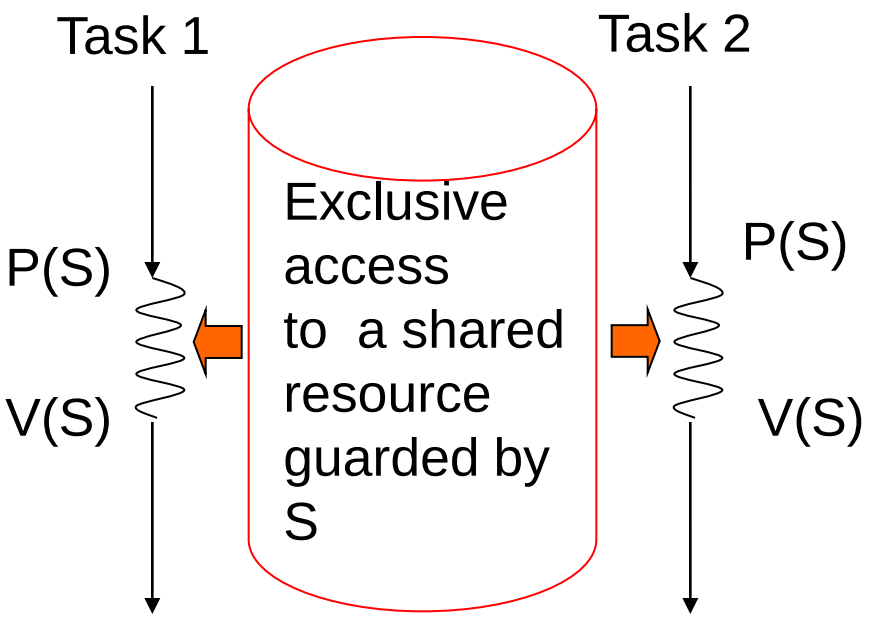
- What if your set of processes is unschedulable?
  - Change deadlines in requirements.
  - Reduce execution times of processes.
  - Get a faster CPU.

Priority inversion

- 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.

Resource access protocols

- **Critical sections:** sections of code at which exclusive access to some resource must be guaranteed.
- Can be guaranteed with semaphores S.

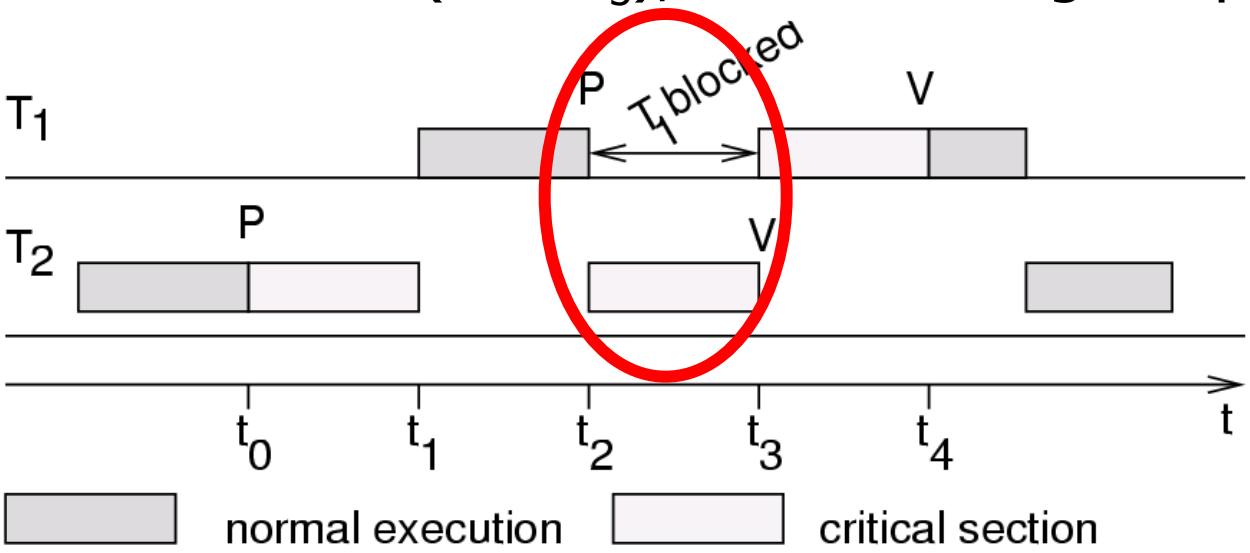


P(S) checks semaphore to see if resource is available and if yes, sets S to "in use". Uninterruptable operations! If no, calling task has to wait.

V(S): sets S to "available", which can be used by another waiting task (if any).

Priority inversion

- Priority T_1 assumed to be higher than priority of T_2.
- If T_2 requests exclusive access first (at t_0), T_1 has to wait until T_2 releases the resource (time t_3), thus inverting the priority:



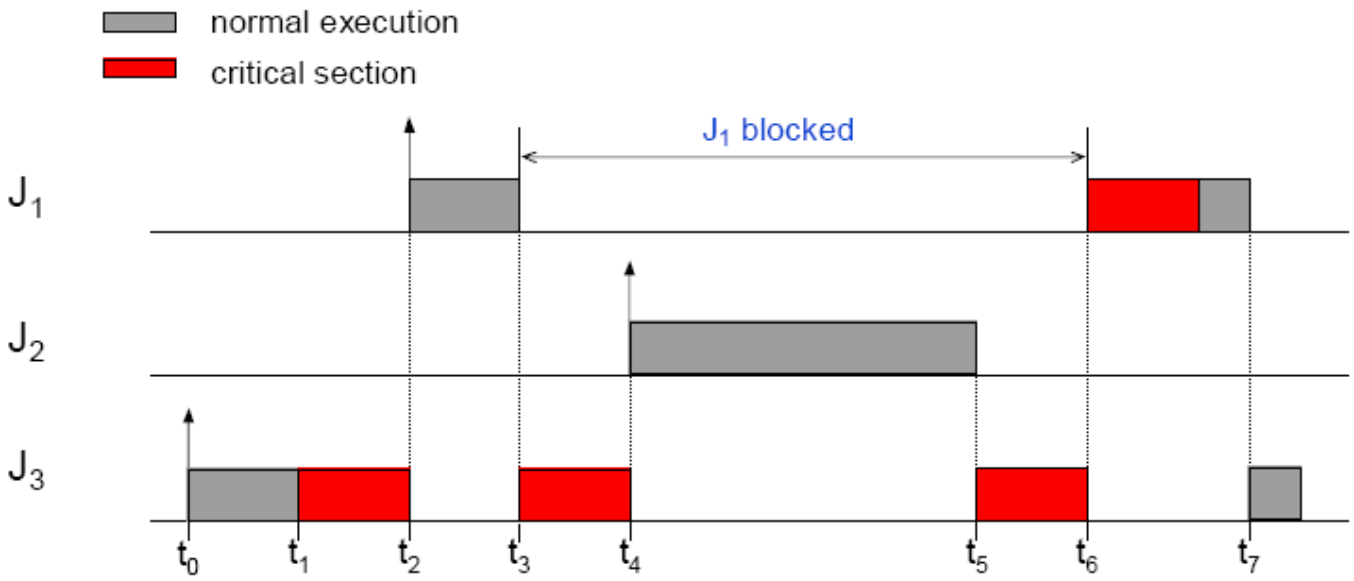
In this example:
duration of inversion bounded by length of critical section of T_2.

Duration of priority inversion with >2 tasks can exceed the length of any critical section

maximum blocking time of J_1 = duration of J_2 in critical section

→ unavoidable due to semantics of critical section

However: blocking time may be unbounded if there are tasks with intermediate priority:



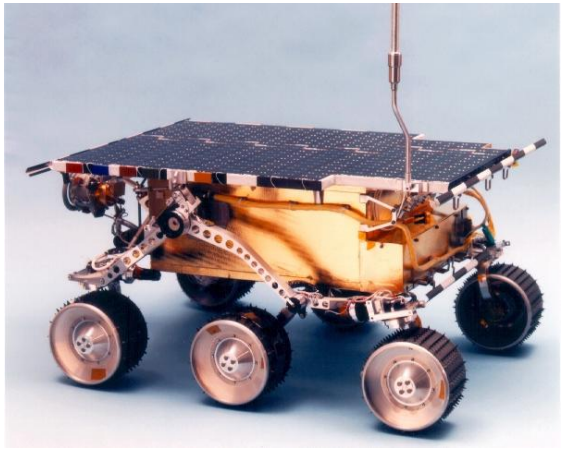
Priority inversion occurs in interval $[t_3, t_6]$

Solving priority inversion

- 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.

MARS Pathfinder problem (1)

⌘“But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".” …



MARS Pathfinder problem (2)

- "VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities."
- "Pathfinder contained an "information bus", a shared memory area for passing information between different components of the spacecraft."
- A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes)."

MARS Pathfinder problem (3)

- The meteorological data gathering task ran as an infrequent, low priority thread, … When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. ..
- The spacecraft also contained a communications task that ran with medium priority."

High priority:	retrieval of data from shared memory
Medium priority:	communications task
Low priority:	thread collecting meteorological data

MARS Pathfinder problem (4)

- Most of the time this combination worked fine.

- However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread.

MARS Pathfinder problem (5)

- In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running.
- After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset. This scenario is a classic case of priority inversion."

Priority inheritance protocol

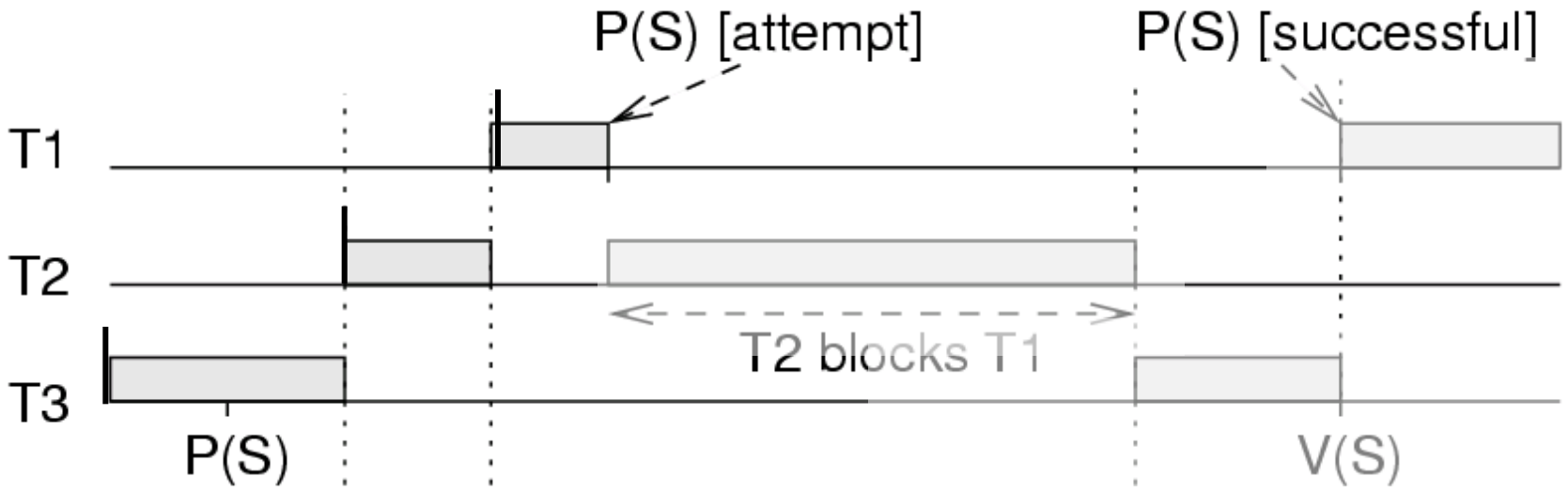
- A task is scheduled according to its active priority. Tasks with the same priorities are scheduled FCFS.
- A task inherits the highest priority from the tasks it blocks.
- If task T1 executes P(S) but its exclusive access was granted to T2, then T1 will be blocked.
- If priority(T2) < priority(T1), then T2 inherits the priority of T1 so that T2 can release the shared resource earlier by preventing medium-priority tasks from preempting T2 and prolonging the blocking period.

Priority inheritance protocol

- When T2 executes V(S), its original priority at the point of entry of the critical section as restored.
- Priority inheritance is transitive
- Assuming that priority(T1) > priority(T2) > priority(T3)
- If T3 blocks T2 and T2 blocks T1, then T3 inherits the priority of T1.

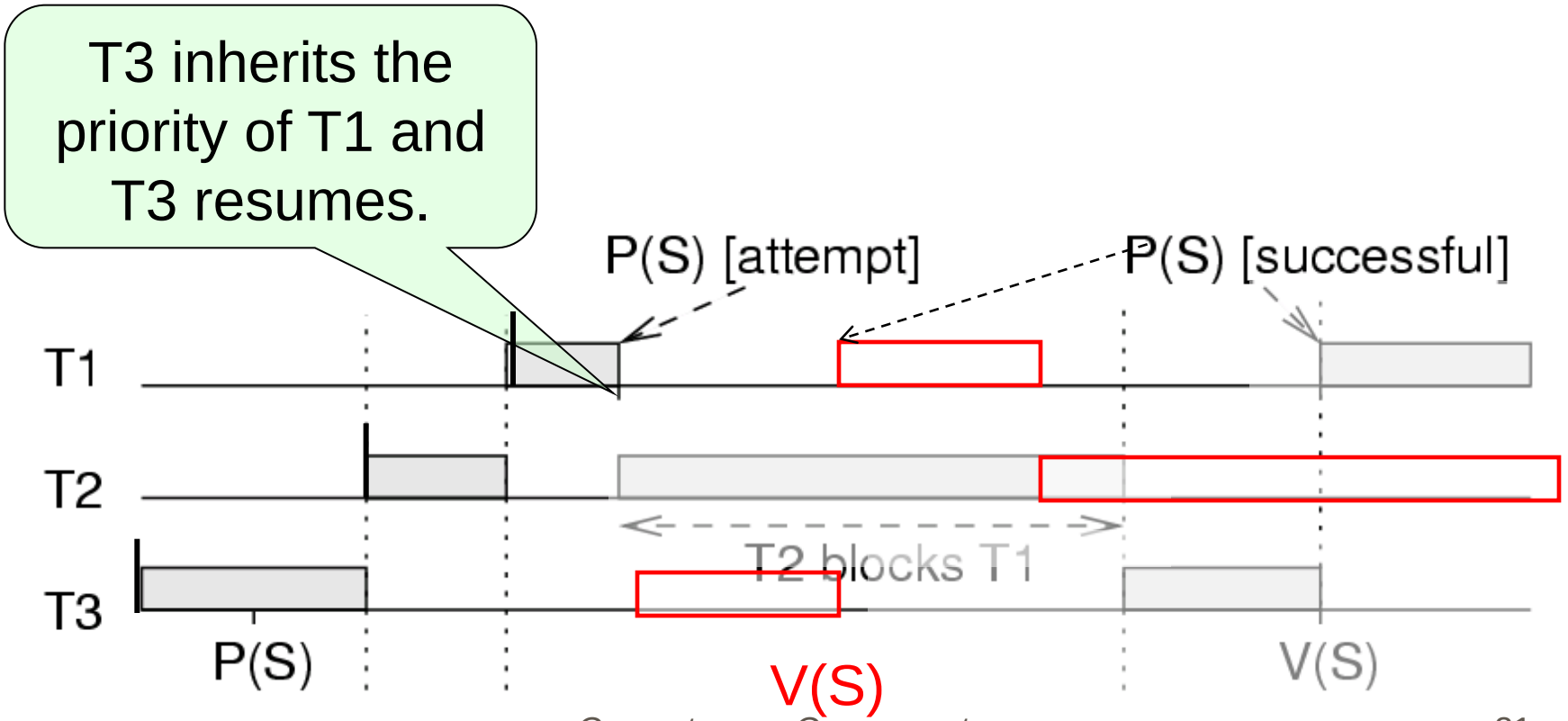
PIP: Example (1)

- without priority inheritance

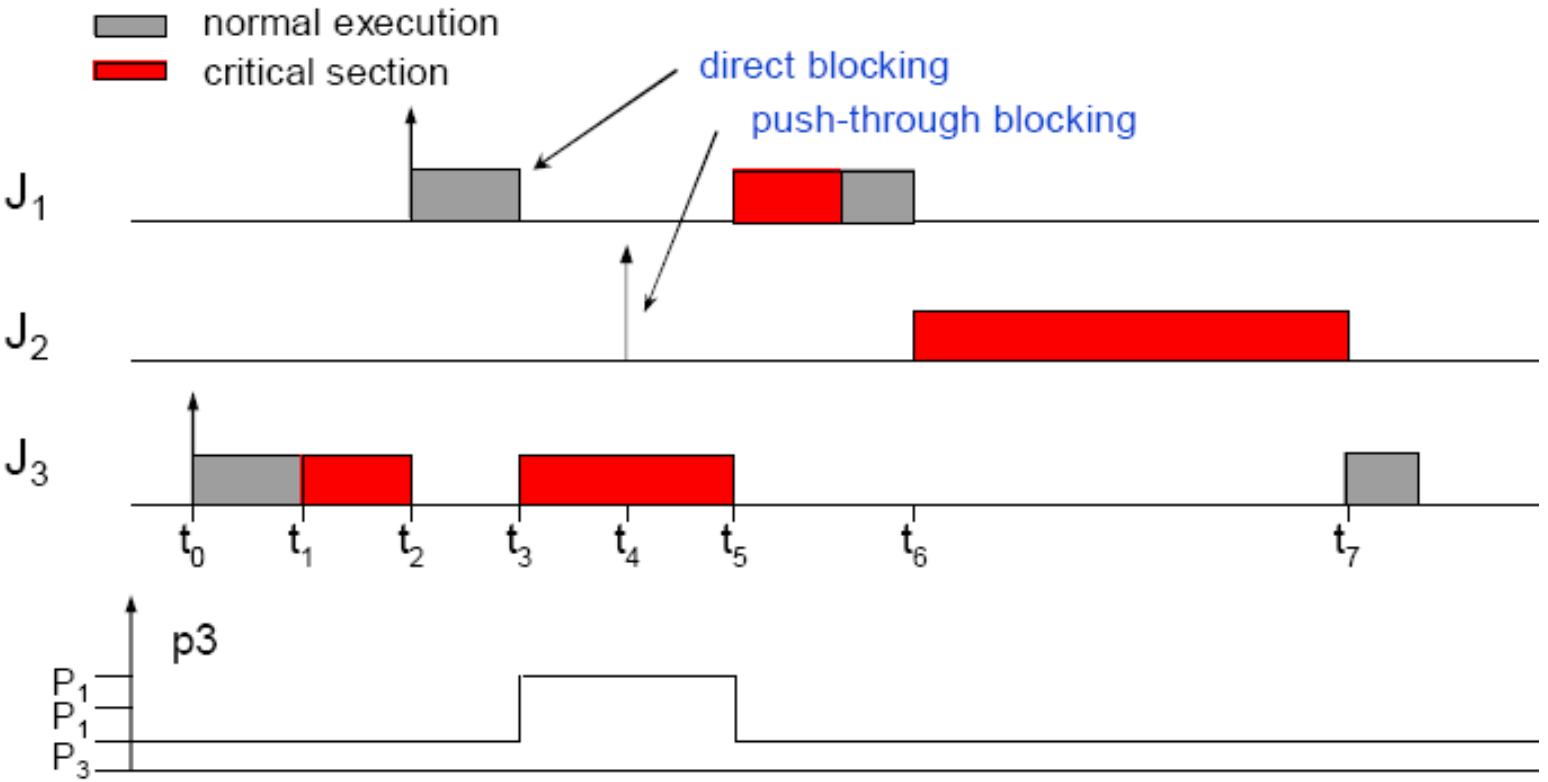


PIP: Example (2)

With priority inheritance



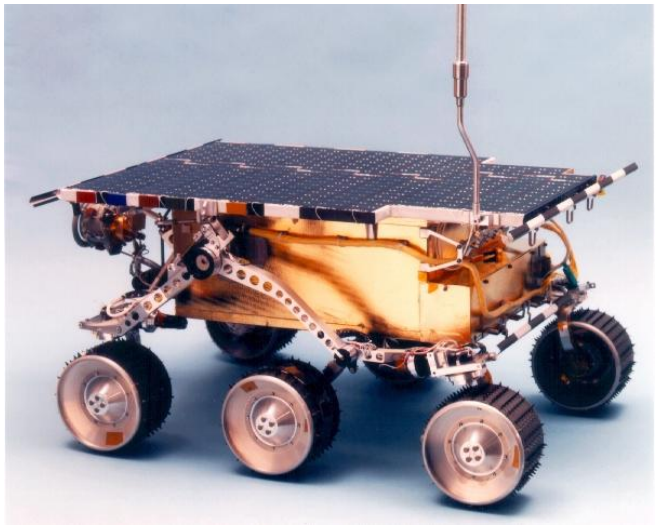
PIP: Example (3)



Priority inversion on Mars

- Priority inheritance also solved the Mars Pathfinder problem: the VxWorks operating system used in the pathfinder implements a flag for the calls to mutex primitives. This flag allows priority inheritance to be set to “on”. When the software was shipped, it was set to “off”.

The problem on Mars was corrected by using the debugging facilities of VxWorks to change the flag to “on”, while the Pathfinder was already on the Mars [Jones, 1997].

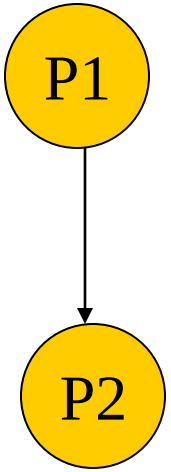


Summary

- **Periodic scheduling**
  - Rate monotonic scheduling
  - EDF
- **Resource access protocols**
  - Priority inversion
  - The Mars pathfinder example
  - Priority inheritance

Data dependencies

- Data dependencies allow us to improve utilization.
  - Restrict combination of processes that can run simultaneously.
- P1 and P2 can't run simultaneously.



Context-switching 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 (hundreds of clock cycles) relative to many common task periods (ms – μs).

7. Multiprocessors

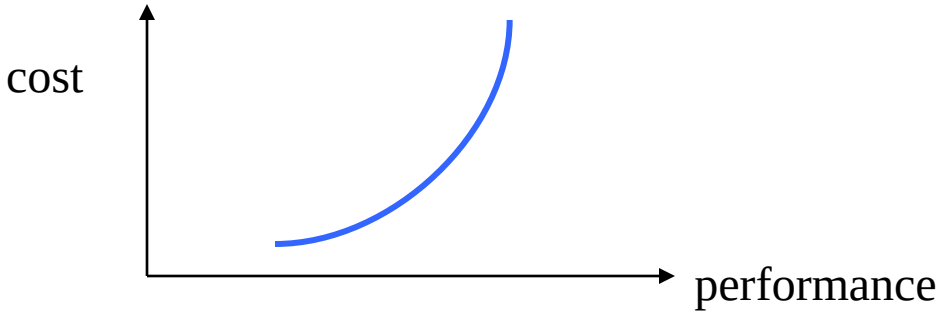
- Why multiprocessors?
- CPUs and accelerators.
- Multiprocessor performance analysis.

Why multiprocessors?

- Programming a single CPU is hard enough.
- Why make life more difficult by adding more processors?
- PE: processing element for computation
  - Whether it is programmable or not.
- Multiprocessors tend to have regular architectures
  - Several identical processors that can access a uniform memory space

Why multiprocessors?

- There are a variety of different multiprocessor architectures
- Better cost/performance.
  - Match each CPU to its tasks or use custom logic (smaller, cheaper).
  - CPU cost is a non-linear function of performance.

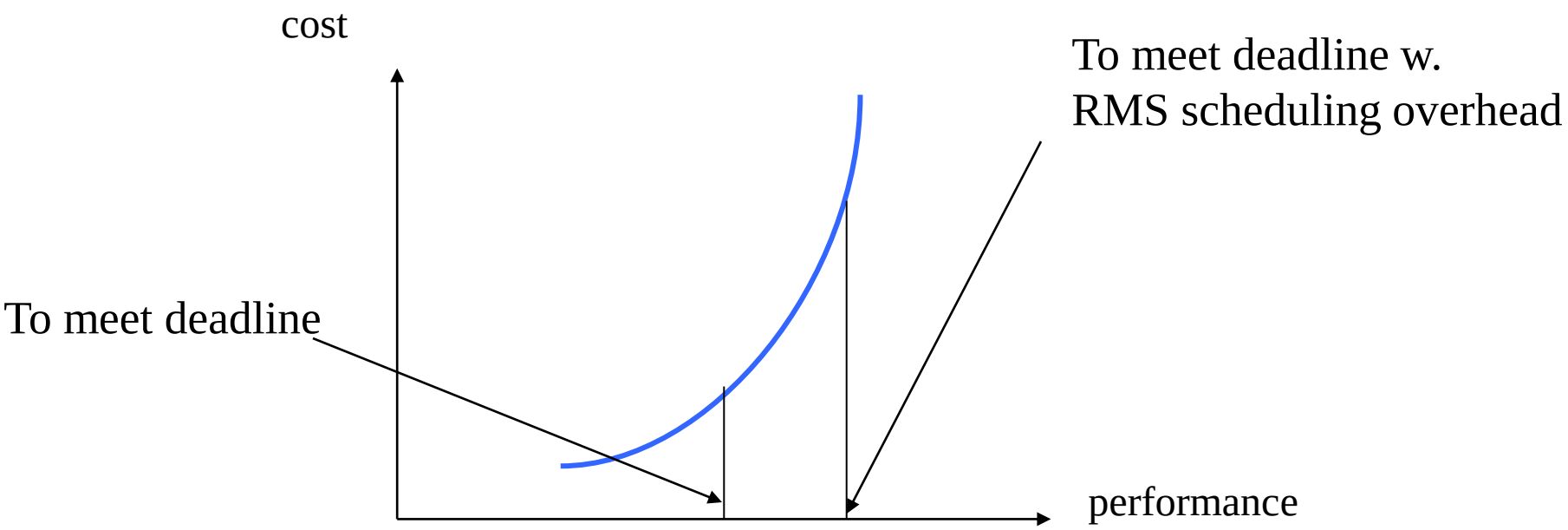


Power
Performance
Cost (Area)

Why multiprocessors?

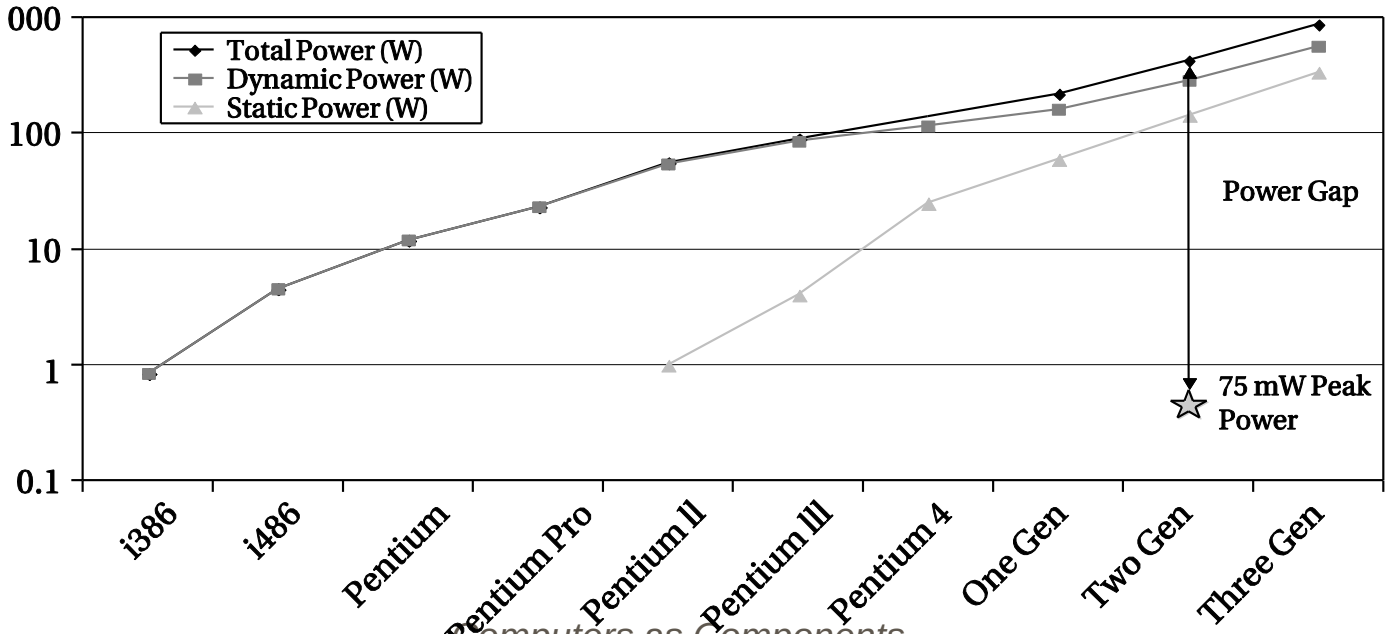
- Splitting the application across multiple processors entails higher engineering cost and lead times.
- Better real-time performance.
  - Put time-critical functions on less-loaded processing elements.
  - Remember RMS utilization---extra CPU cycles must be reserved to meet deadlines.

Why multiprocessors?



Why multiprocessors?

- Using specialized processors or custom logic saves power.
- Desktop processors are not power-efficient enough for battery-powered applications.



Watt-hour

- **Wh=Rated Capacity(Ah) x voltage (V)**
- Specific Power = power to weight ratio
  - W/kg
- Specific Energy = energy capacity to weight ratio
  - Wh/kg
- Power Density = power to volume ratio
  - W/l
- Energy Density = energy to capacity to volume ratio
  - Wh/l

Nameplate Capacity

- Name plate capacity should be determined based on a standard set of requirements:
  - Discharge at C/10 (if the manufacturer recommended rate is different from this, it should be specified when nameplate capacity is provided)
  - Ambient temp (20 +/- 2 deg C)
  - Charge at C/10 (or the manufacturer recommended rate)

Battery Capacity

Type	Capacity (mAh)	Density (Wh/kg)
Alkaline AA	2850	124
Rechargeable	1600	80
NiCd AA	750	41
NiMH AA	1100	51
Lithium ion	1200	100
Lead acid	2000	30

Discharge Rates

Type	Voltage	Peak Drain	Optimal Drain
Alkaline	1.5	0.5C	< 0.2C
NiCd	1.25	20C	1C
Nickel metal	1.25	5C	< 0.5C
Lead acid	2	5C	0.2C
Lithium ion	3.6	2C	< 1C

Comparison of Battery Performance

Item	Conventional Battery (14430G6)	Nexelion (14430W1)
Anode material	Graphite (Carbon)	Tin based Amorphous material
Cathode material	Lithium Cobalt oxide	Multi-stage composite cathode (Mixture of cobalt, manganese, nickel oxides and Lithium Cobalt oxide)
Electrolyte	Hybrid electrolyte	Newly developed hybrid eletrolyte
Size	Diameter 14mm x Height 43mm	Diameter 14mm x Height 43mm
Capacity(0.2CmA)Size	700mAh, 2.6Wh	900mAh, 3.1Wh diameter
Standard charging voltage	4.2V - 3V	4.2V - 2.5V
Energy Density	395Wh/l, 144 Wh/kg	478Wh/l, 158 Wh/kg
Weight	18 g	20 g

14430 is cylindrical with 14 mm dia. and 43 mm high

A123 2.2 Ah high-power Lithium-Ion Cells

Charge Rate Characterization Testing
Summary of Results at - 10°C

Cell AJ 235			A123 Baseline Electrolyte						
Temperature	Discharge Rate	Discharge Current (A)	Charge Capacity (Ah)	Charge Time (Hours)	Percent C/10 Capacity	Percent C/10 Capacity at 20°C	Charge Watt-Hr (Wh)	Discharge Watt-Hr (Wh)	Watt Hour Efficiency (%)
-10°C	C/10	0.220	2.3071	10.5323	100.00	95.71	7.7373	7.4100	95.77
	C/5	0.440	2.2759	5.4073	98.65	94.42	7.7094	7.3225	94.98
	C/2	1.100	2.2609	2.5739	98.00	93.79	7.8137	7.2749	93.10
	C	2.200	2.2518	1.7134	97.60	93.42	7.9254	7.2380	91.33
	1.5C	3.300	2.2455	1.4711	97.33	93.16	7.9795	7.2131	90.40
	2C	4.400	2.2298	1.3958	96.65	92.51	7.9690	7.1615	89.87

Why multiprocessors?

- May consume less energy.
- May be better at streaming data.
- May not be able to do all the work on even the largest single CPU.

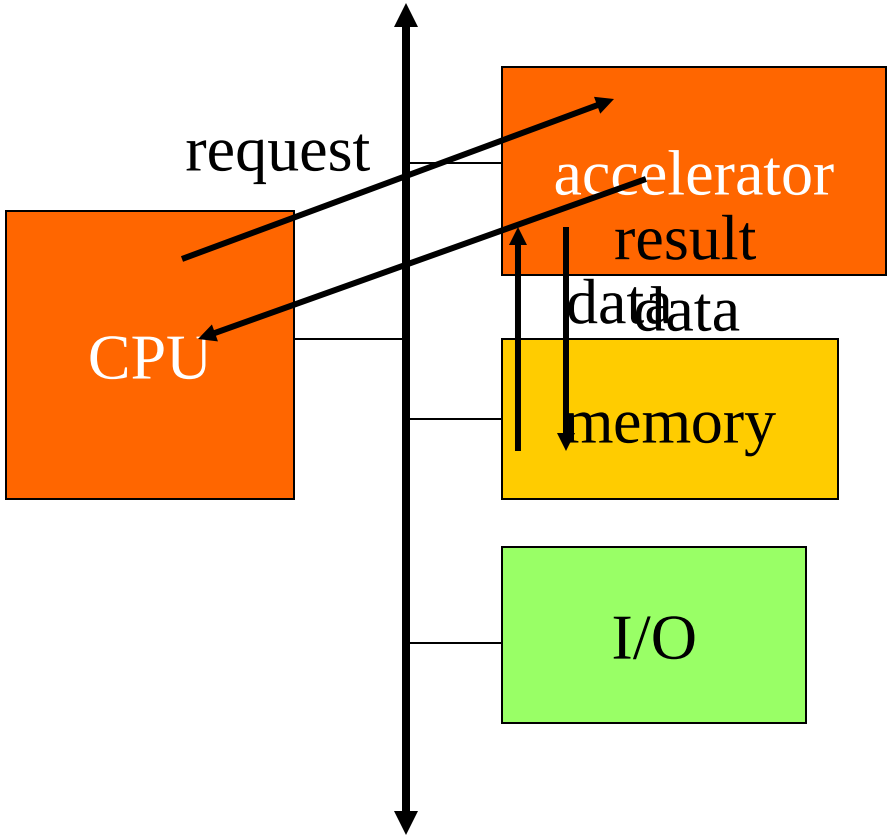
Why multiprocessors?

- Good for processing I/O in real-time.
- May consume less energy.
- May be better at streaming data.
- May not be able to do all the work on even the largest single CPU.
- A thread per processor
  - no context switching

Accelerated systems

- Use additional computational unit dedicated to some functions?
  - Hardwired logic.
  - Extra CPU.
- Hardware/software co-design: joint design of hardware and software architectures.

Accelerated system architecture



Accelerator vs. co-processor

- A co-processor executes instructions with op-code.
  - Instructions are dispatched by the CPU.
- An accelerator appears as a device on the bus.
  - Its programming model interface is functionally equivalent to an I/O device although it does not perform input or output
  - is controlled by it registers.
  - Doe not execute intructions

System design tasks

- Design a heterogeneous multiprocessor architecture.
  - Processing element (PE): CPU, coprocessor, accelerator, etc.
- Program the system.

Accelerated system design

- First, determine that the system really needs to be accelerated.
  - How much faster is the accelerator on the core function?
  - How much data transfer overhead?
- Design the accelerator itself.
- Design CPU interface to accelerator.

Accelerator implementations

- Application-specific integrated circuit.
- Field-programmable gate array (FPGA).
- Standard component.
  - Example: graphics processor.

Accelerated system platforms

- Several off-the-shelf boards are available for acceleration in PCs:
  - FPGA-based core;
  - PC bus interface.

Accelerator/CPU interface

- Accelerator registers provide control registers for CPU.
- Data registers (buffers) can be used for small data objects.
- Accelerator may include special-purpose read/write logic.
  - Especially valuable for large data transfers.
  - DMA to transfer a large volume of data without intervention of CPU

Accelerator/CPU interface

- Design CPU-side interface
  - Application software need to talk to the accelerator (data, instruction)
  - Synchronization between CPU and accelerator
    - The accelerator should know when it has the required data
    - The CPU should know when it has received the designed reuslts.

System integration/debugging

- Try to debug the CPU/accelerator interface separately from the accelerator core.
- Build scaffolding to test the accelerator.
- Hardware/software co-simulation can be useful.

Caching problems

- Main memory provides the primary data transfer mechanism to the accelerator.
- Programs must ensure that caching does not invalidate main memory data.
  - CPU reads location S.
  - Accelerator writes location S.
  - CPU writes location S.

Synchronization

- As with cache, main memory writes to shared memory may cause invalidation:
  - CPU reads S.
  - Accelerator writes S.
  - CPU reads S.

Mobile Phone Trends

TABLE I
MOBILE PHONE TRENDS IN 5-YEAR INTERVALS.

year	1995	2000	2005	2010	2015
cellular generation	2G	2.5-3G	3.5G	pre-4G	4G
cellular standards	GSM	GPRS UMTS	HSPA	HSPA LTE	LTE LTE-A
downlink bitrate [Mb/s]	0.01	0.1	1	10	100
display pixels [×1000]	4	16	64	256	1024
battery energy [Wh]	1	2	3	4	5
CMOS [ITRS, nm]	350	180	90	50	25
PC CPU clock[MHz]	100	1000	3000	6000	8500
PC CPU power [W]	5	20	100	200	200
PC CPU MHz/W	20	50	30	30	42
phone CPU clock[MHz]	20	100	200	500	1000
phone CPU power [W]	0.05	0.05	0.1	0.2	0.3
phone CPU MHz/W	400	2000	2000	2500	3000
workload [GOPS]	0.1	1	10	100	1000
software [MB]	0.1	1	10	100	1000
#programmable cores	1	2	4	8	16

Power and Battery Capacity

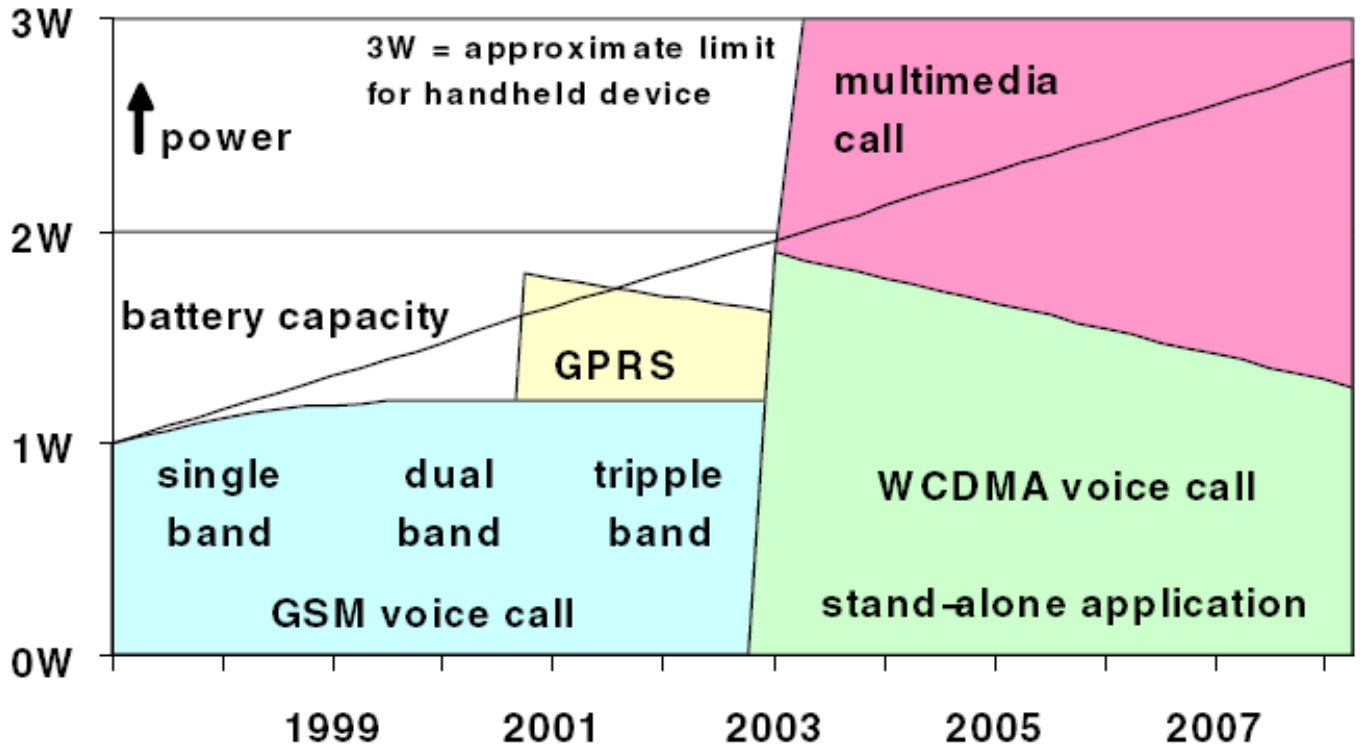
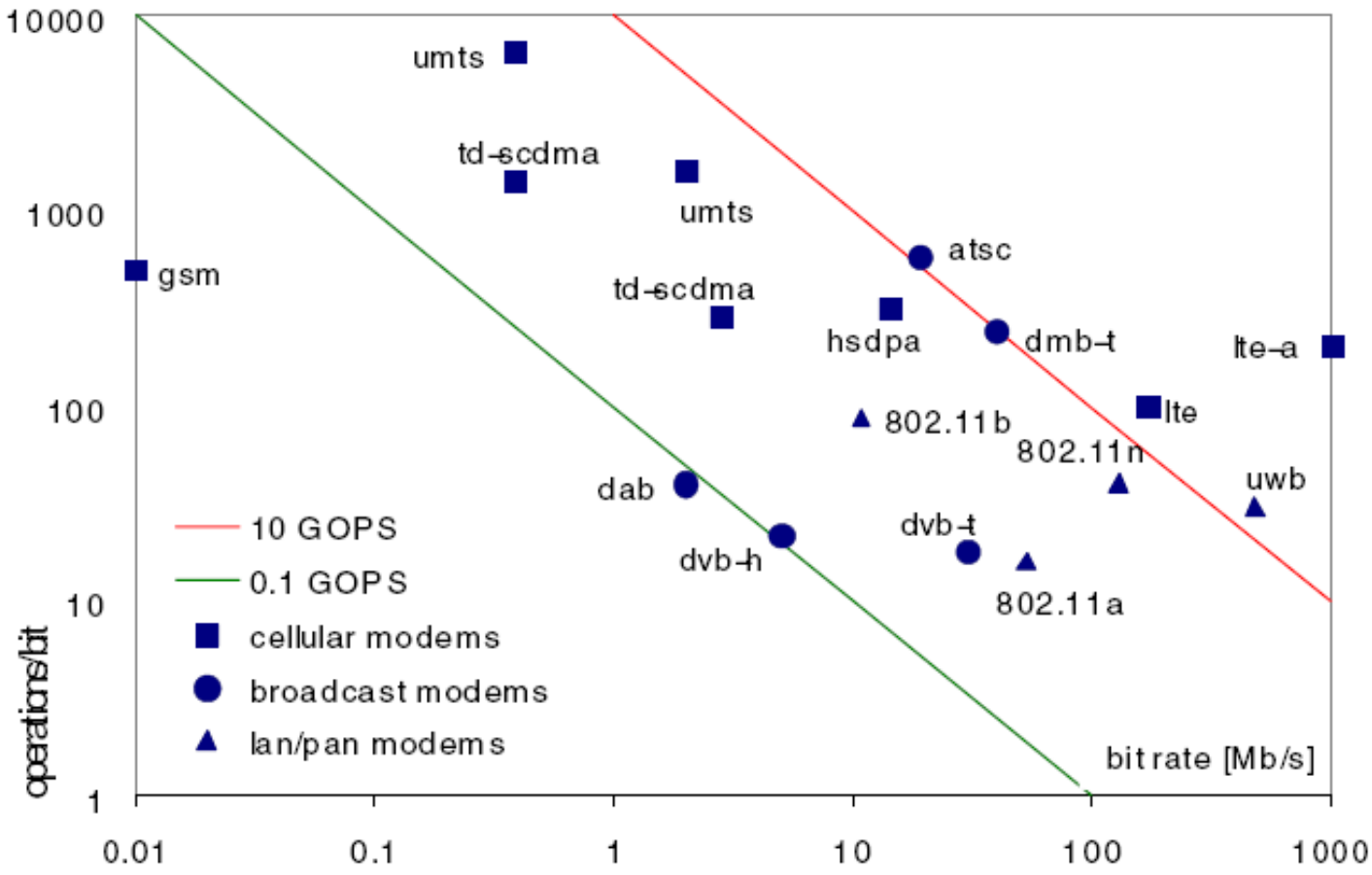
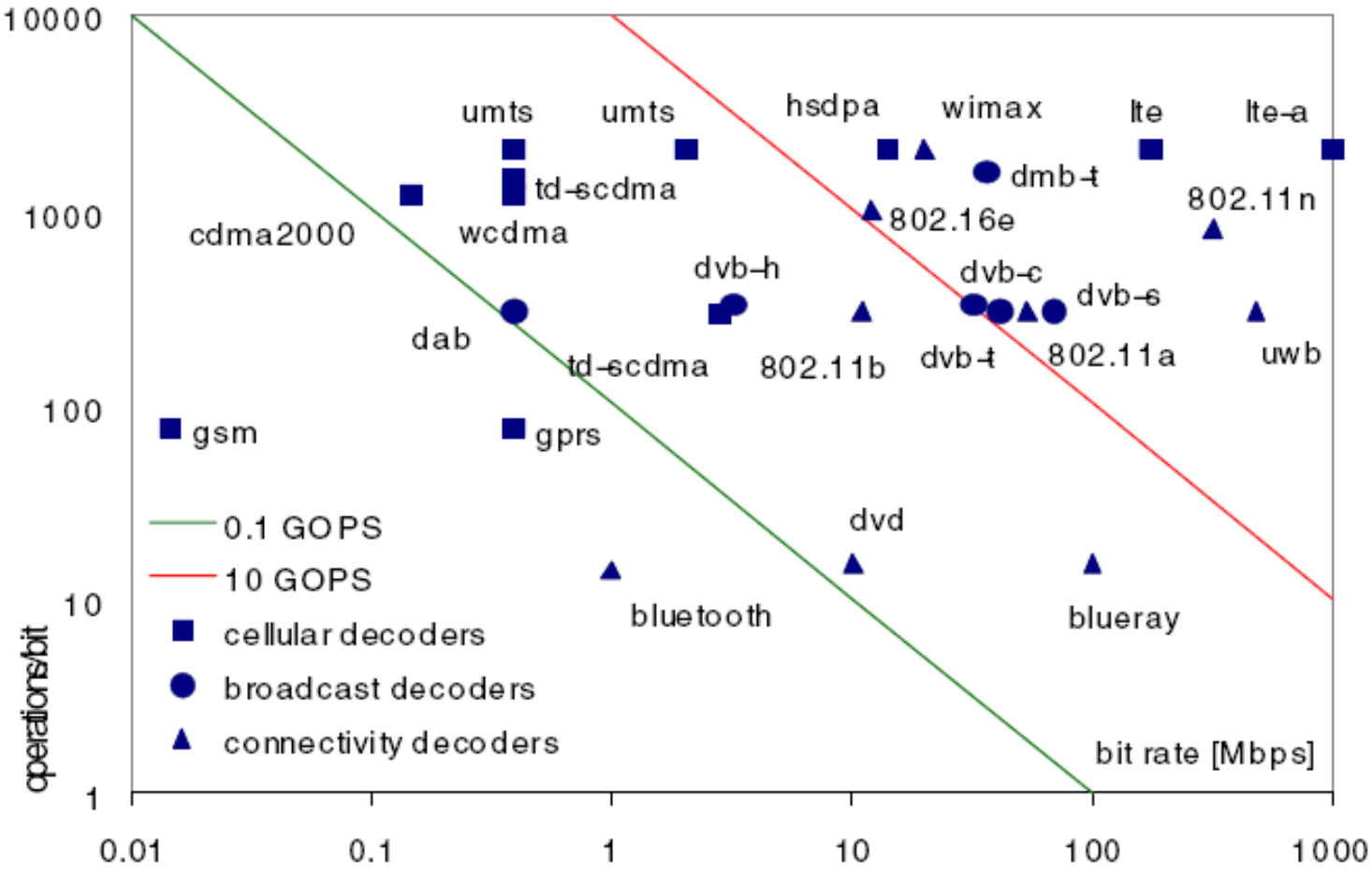


Fig. 1. Battery capacity and power consumption at maximum output power level in cellular transmitters, adapted from [3].

Radio Demodulation Workload



Radio Decoding Workload



3.5G Workload

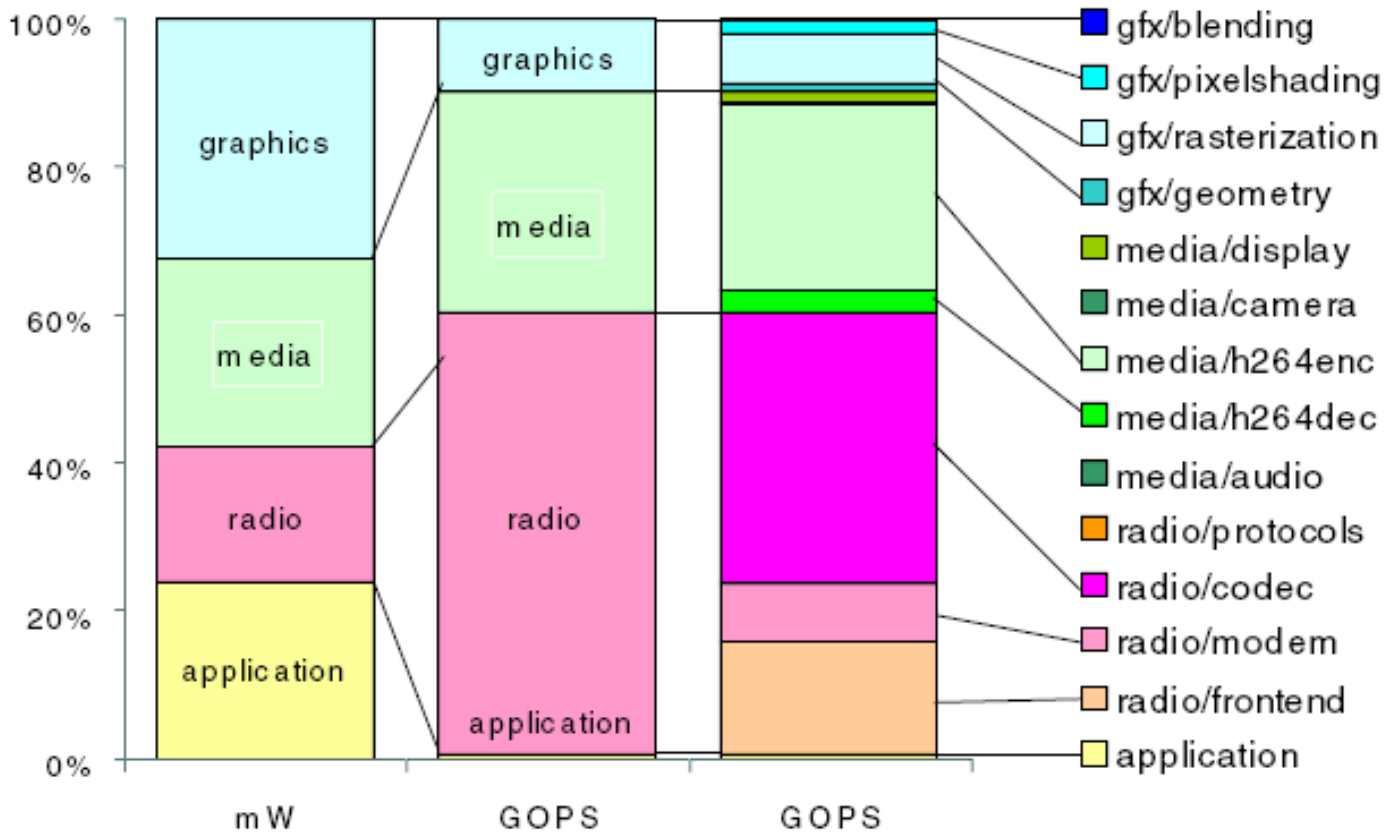


Fig. 3. 3.5G workload (power consumption) as fractions of 100GOPS (1W).

Workload vs Energy/operation

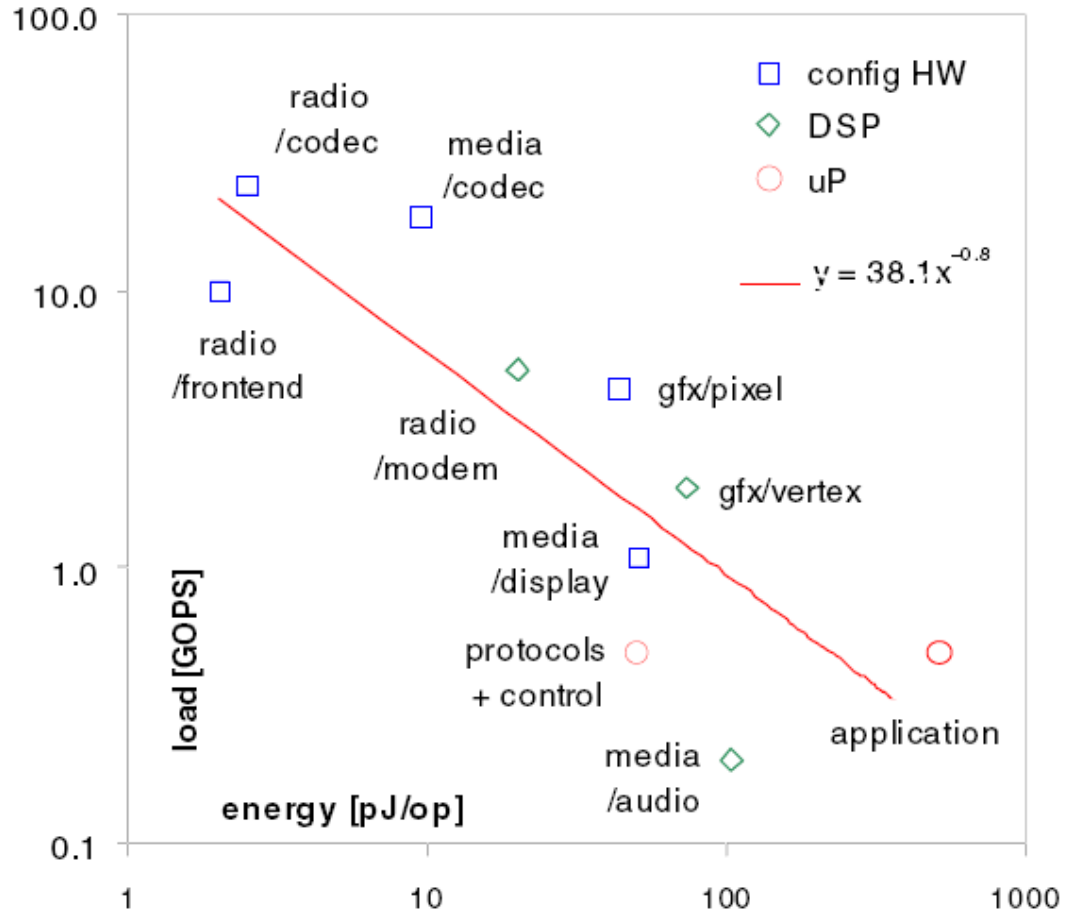


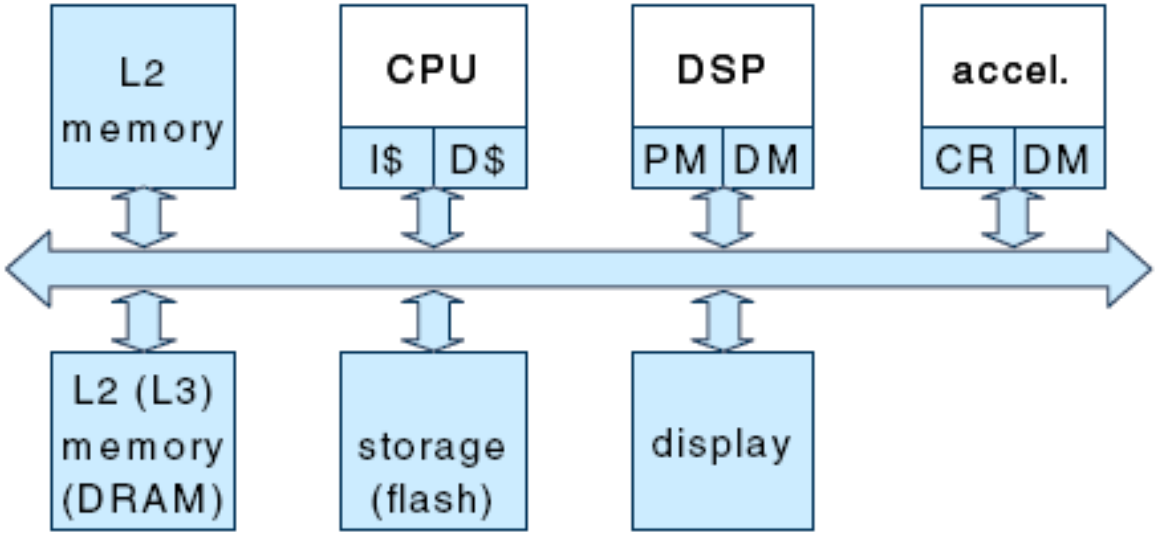
Fig. 4. Workloads [GOPS] versus energy/operation [pJ].

Value of Programming

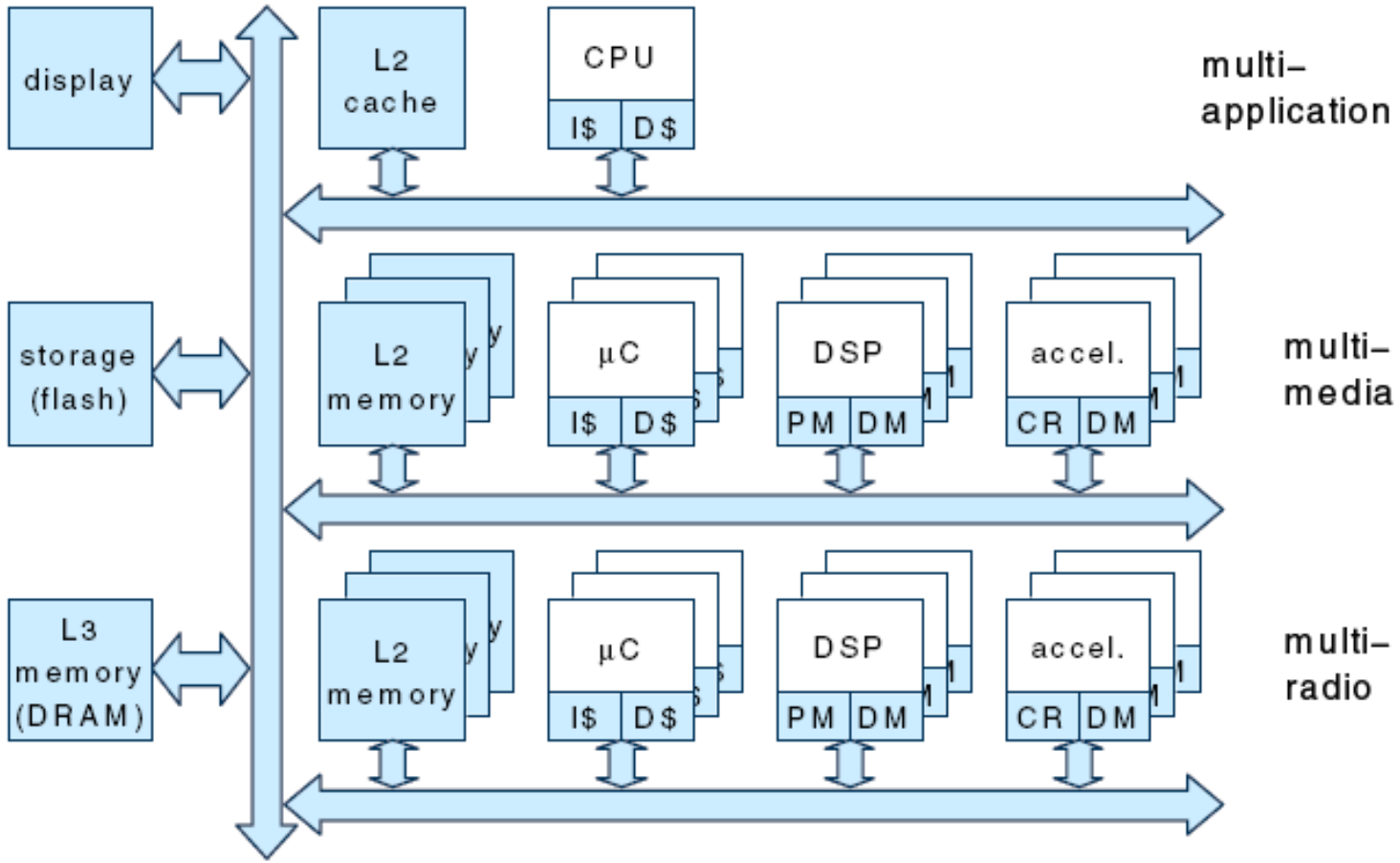
TABLE II
VALUE-AFFORDABILITY OF PROGRAMMABLE SOLUTIONS (EXAMPLES).

	radio	video	3D graphics
very high	protocol stacks		geometry proces.
high	channel estimation		pixel shading
medium	demodulation	motion estimation	
low	turbo decoder (i)fft	entropy (de)coding deblocking	
very low	filters	filters scaling	rasterization pixel blending

2/2.5G Dual-core Architecture



3/3.5G Multi-core Architecture



Cell-phone Chips

TABLE III
PUBLISHED INDUSTRIAL CELL-PHONE CHIPS ("..." DENOTES A SHARED RADIO-MEDIA CORE).

year	ref	source	cmos	total	application		radio		media	
			nm	# cores	core(s)	MHz	core(s)	MHz	core(s)	MHz
1992	[1]	Philips	1000	1	n.a.		KISS-16-V2	...	...	20
2000	[2]	Infineon	250	2	CPU	78	Oak	...	...	78
2001	[9]	Samsung	180	2	ARM9		Teaklite		n.a.	
2003	[6]	Toshiba	130	3	n.a.		n.a.		3x RISC	125
2004	[3]	Nokia	130	2	CPU	50	DSP	...	...	160
2004	[10]	Qualcomm	130	3	ARM9	180	DSP	95	DSP	95
2004	[11]	Renesas	130	2	CPU	216	n.a.		DSP	216
2005	[12]	NEC	130	4	ARM9	200	n.a.		2xARM9 + DSP	200
2006	[13]	ST	130	2	ARM8	156	ST122 DSP	...	...	156
2007	[14]	Infineon	90	2	ARM9	380	TEAKlite	...	...	104
2008	[15]	Renesas et al	65	4	ARM11	500	ARM9	166	ARM11 + SHX2	500
2008	[16]	TI	45	5	ARM11	840	?		C55 + ?	480
2008	[17]	NEC	65	3	ARM11	500	ARM11	250	DSP	500
2009	[18]	Panasonic	45	4	ARM11	486	ARM11	245	2xDSP	216
2009	[19]	Renesas	65	2	CPU	500	n.a.		SHX2	500

Power Management Knobs

TABLE IV

POWER MANAGEMENT KNOBS: f_C DENOTES CLOCK FREQUENCY, V_{DD} AND V_t DENOTE SUPPLY AND THRESHOLD VOLTAGE, AND P_D AND P_S DENOTE DYNAMIC AND STATIC POWER CONSUMPTION.

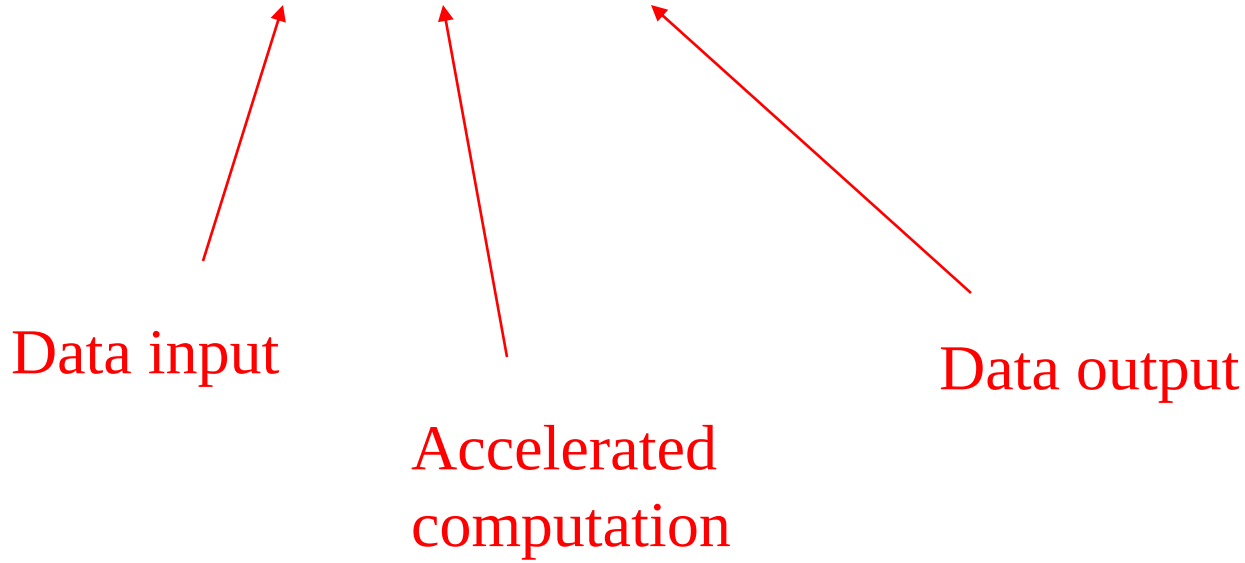
	knob		throughput	power
stop the clock:	$f_C \downarrow 0$	$\Rightarrow$	$\downarrow 0$	$P_D \downarrow 0$
frequency scaling (FS)	$f_C \downarrow$	$\Rightarrow$	$\downarrow$	$P_D \downarrow$
voltage scaling (VS)	$V_{DD} \downarrow$	$\Rightarrow$	$\downarrow$	$P_D \downarrow$
power down	$V_{DD} \downarrow 0$	$\Rightarrow$	$\downarrow 0$	$P_S \downarrow 0$
forward body bias (FBB)	$V_t \downarrow$	$\Rightarrow$	$\uparrow$	$P_S \uparrow$
reverse body bias (RBB)	$V_t \uparrow$	$\Rightarrow$	$\downarrow$	$P_S \downarrow$

Accelerator speedup

- 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.

Accelerator execution time

- Total accelerator execution time:
  - $t_{accel} = t_{in} + t_x + t_{out}$



  - $t_{accel} = \max \{t_{in}, t_x, t_{out}\}$ if pipelined

Accelerator speedup

- Assume loop is executed n times.
- If the software loop is replaced with the accelerator, compare accelerated system to non-accelerated system:
  - $S = n(t_{CPU} - t_{accel})$
    $= n[t_{CPU} - (t_{in} + t_x + t_{out})]$

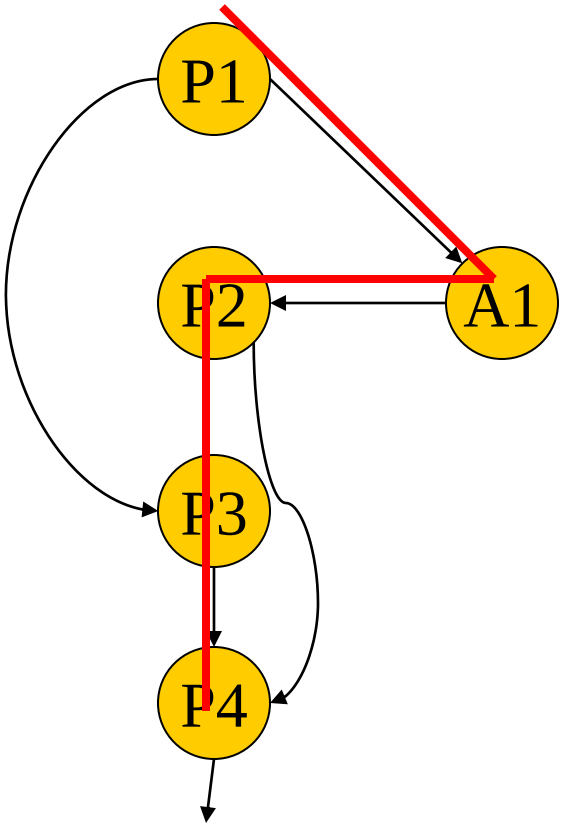
Execution time on CPU

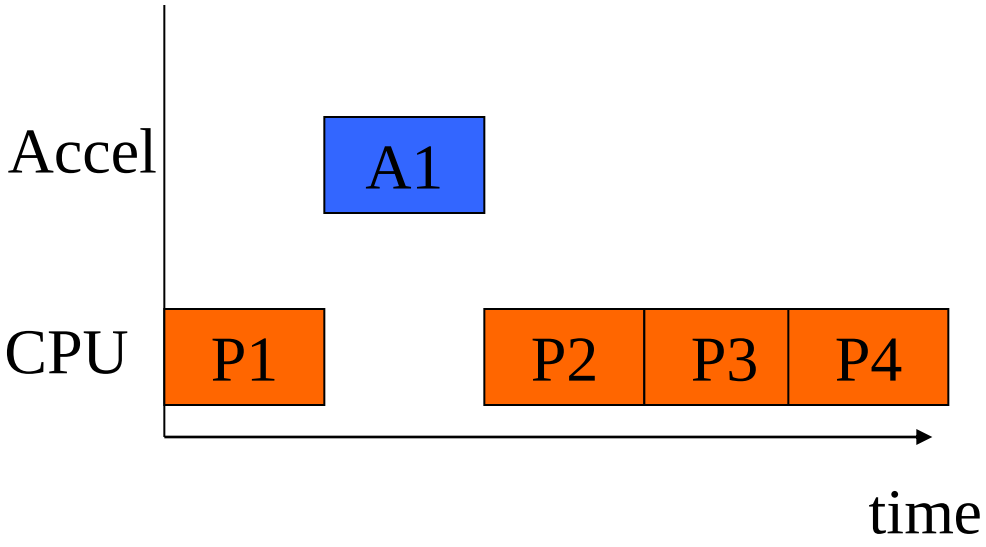
Single- vs. multi-threaded

- 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.
  - Synchronization: software must also support multithreading.

Total execution time

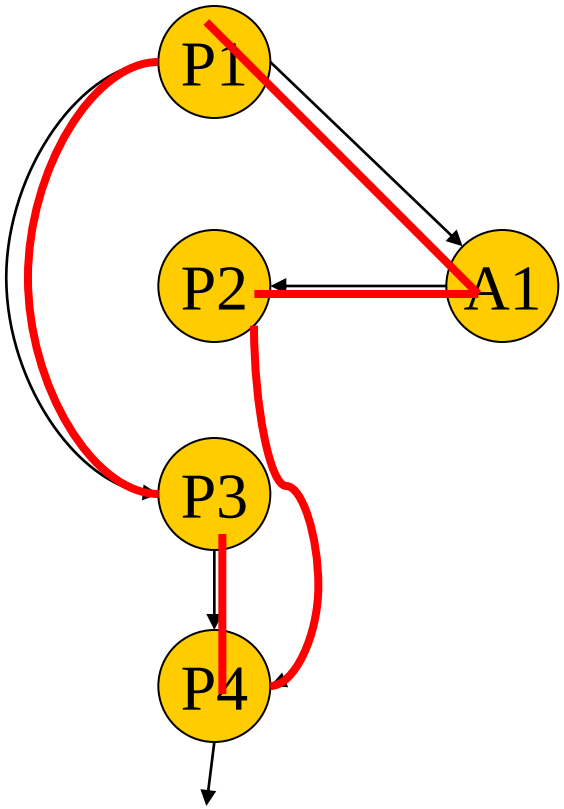
- Single-threaded:

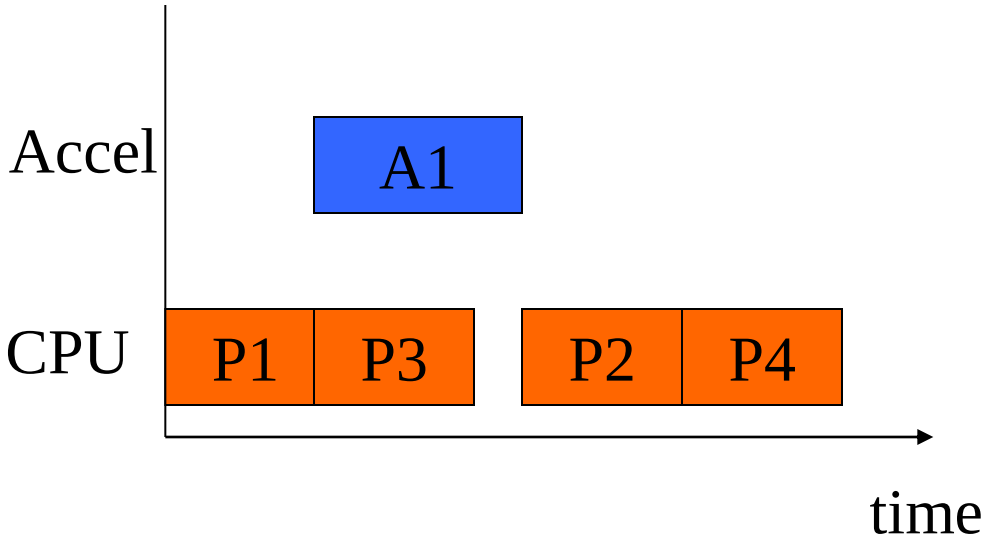




Total execution time

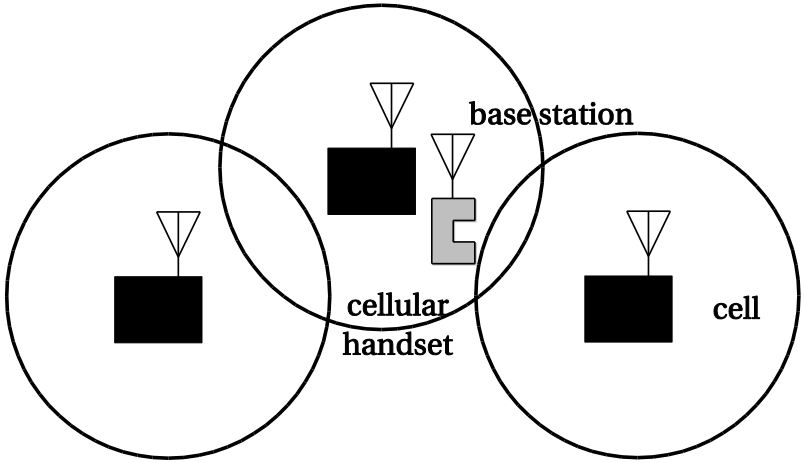
- Multi-threaded:





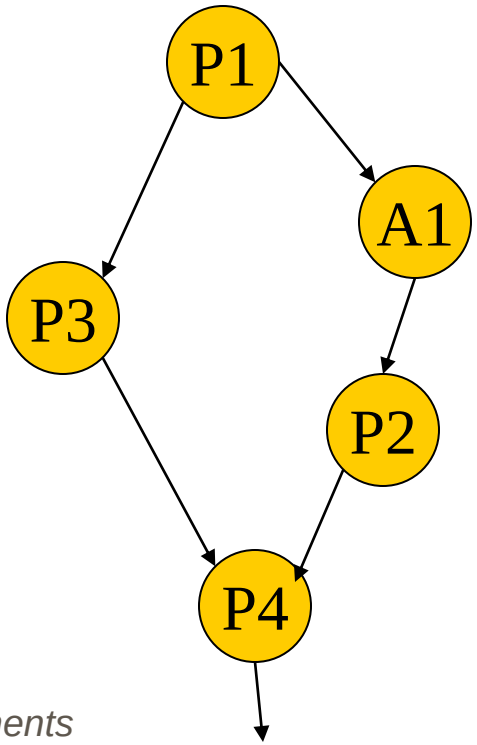
Cell phones

- Most popular CE device in history; most widely used computing device.
  - 1 billion sold per year.
- Handset talks to cell.
- Cells hand off handset as it moves.



Execution time analysis

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



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.

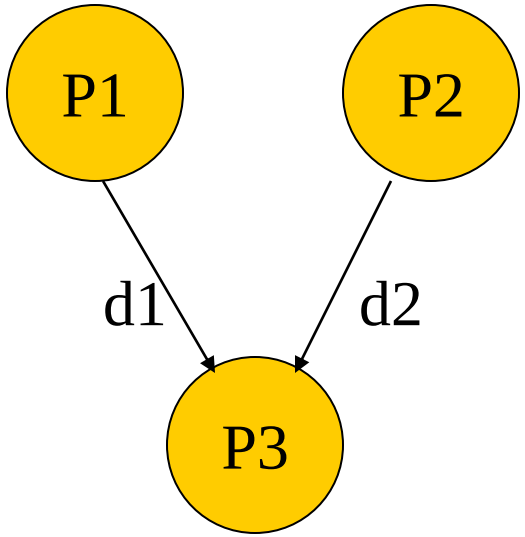
Data input/output times

- Bus transactions include:
  - flushing register/cache values to main memory if necessary;
  - time required for CPU to set up transaction;
  - overhead of data transfers by bus packets, handshaking, etc.

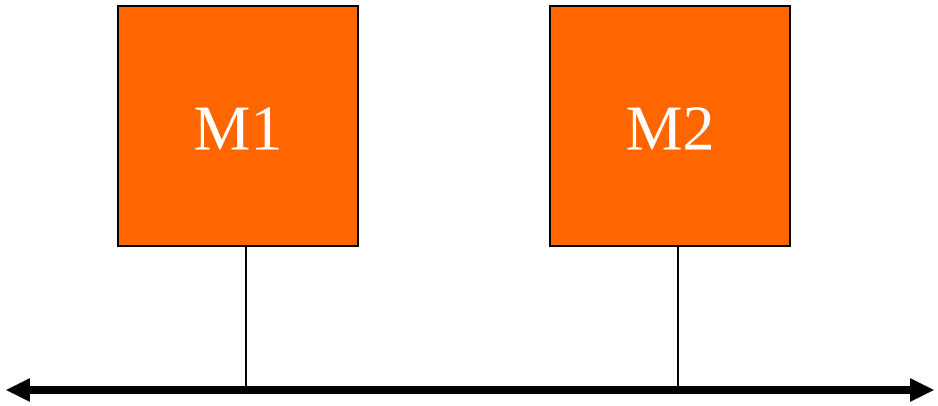
Scheduling and allocation

- Must:
  - schedule operations in time;
  - allocate computations to processing elements.
- Scheduling and allocation interact, but separating them helps.
  - (Alternatively) allocate, then schedule.

Example: scheduling and allocation



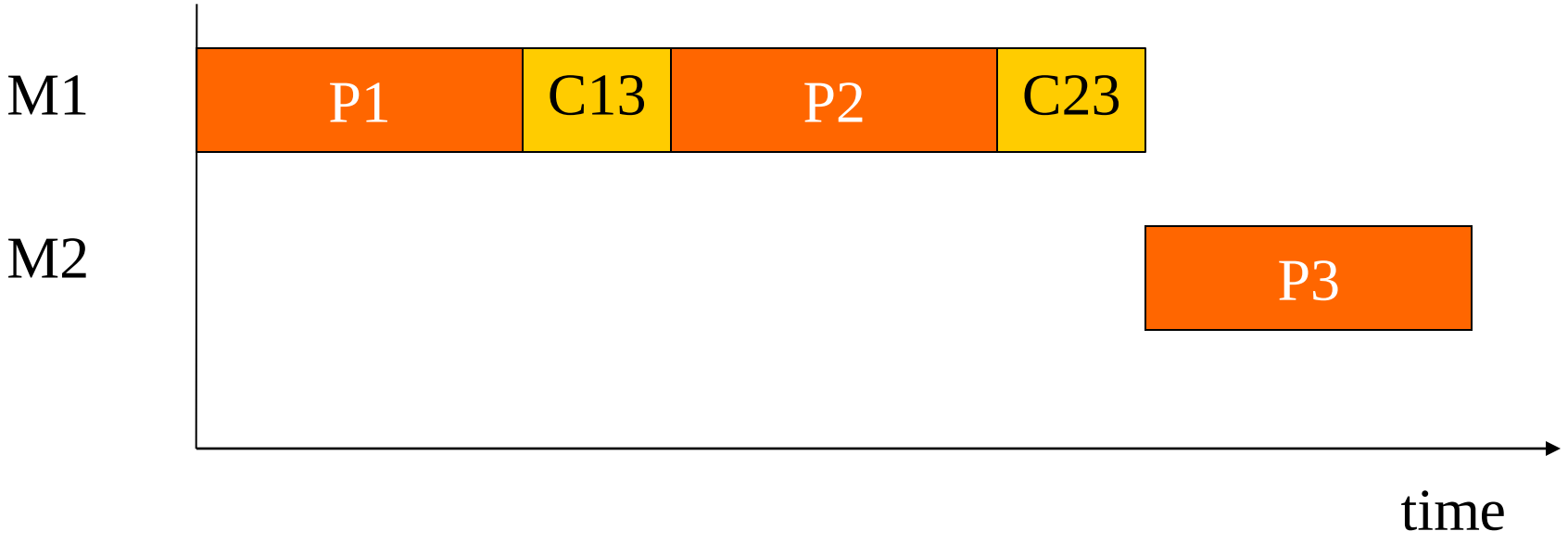
Task graph



Hardware platform

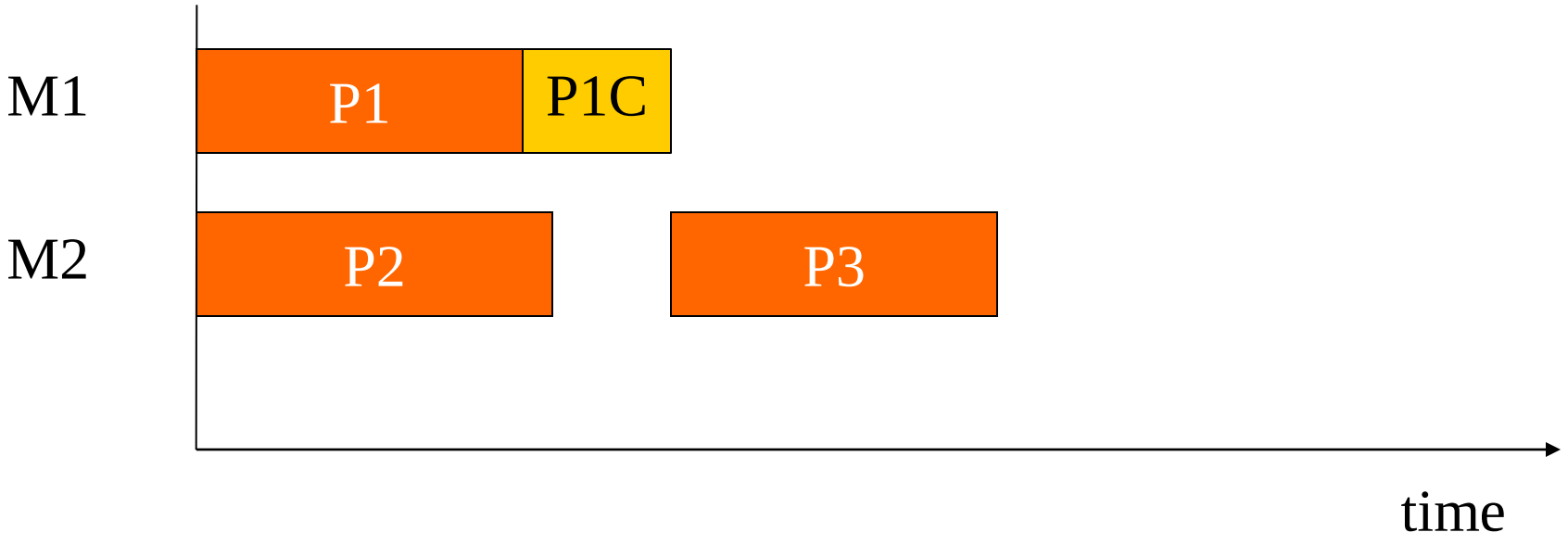
First design

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



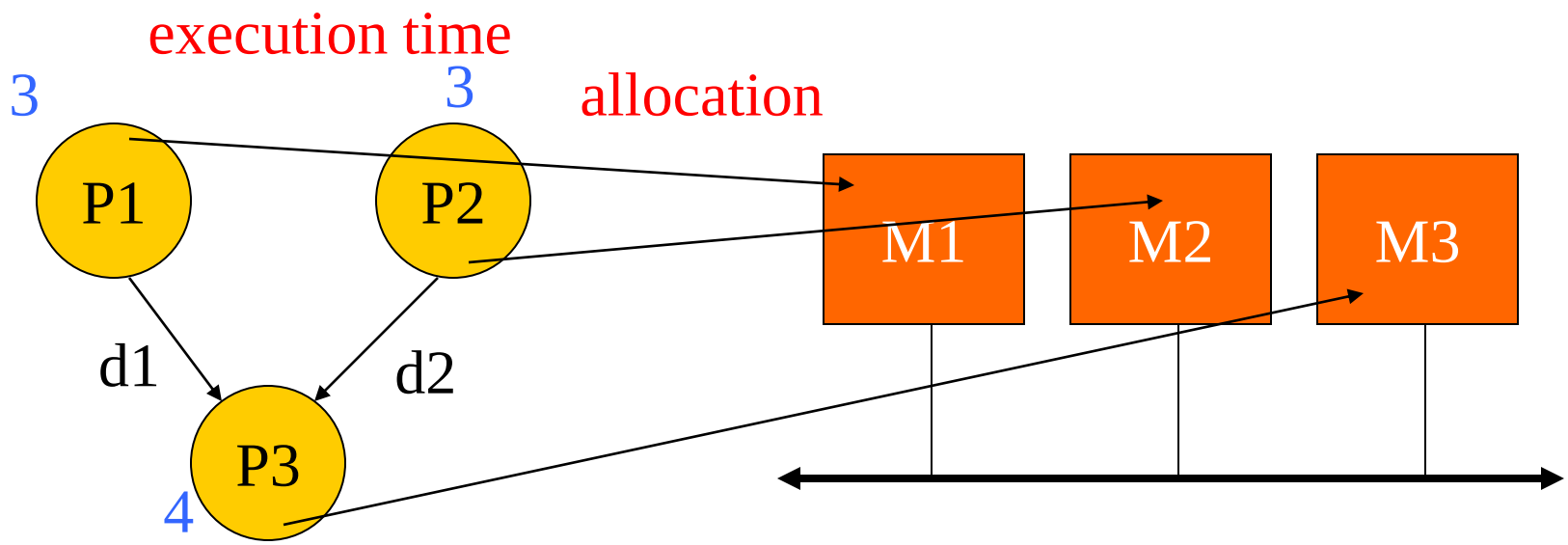
Second design

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



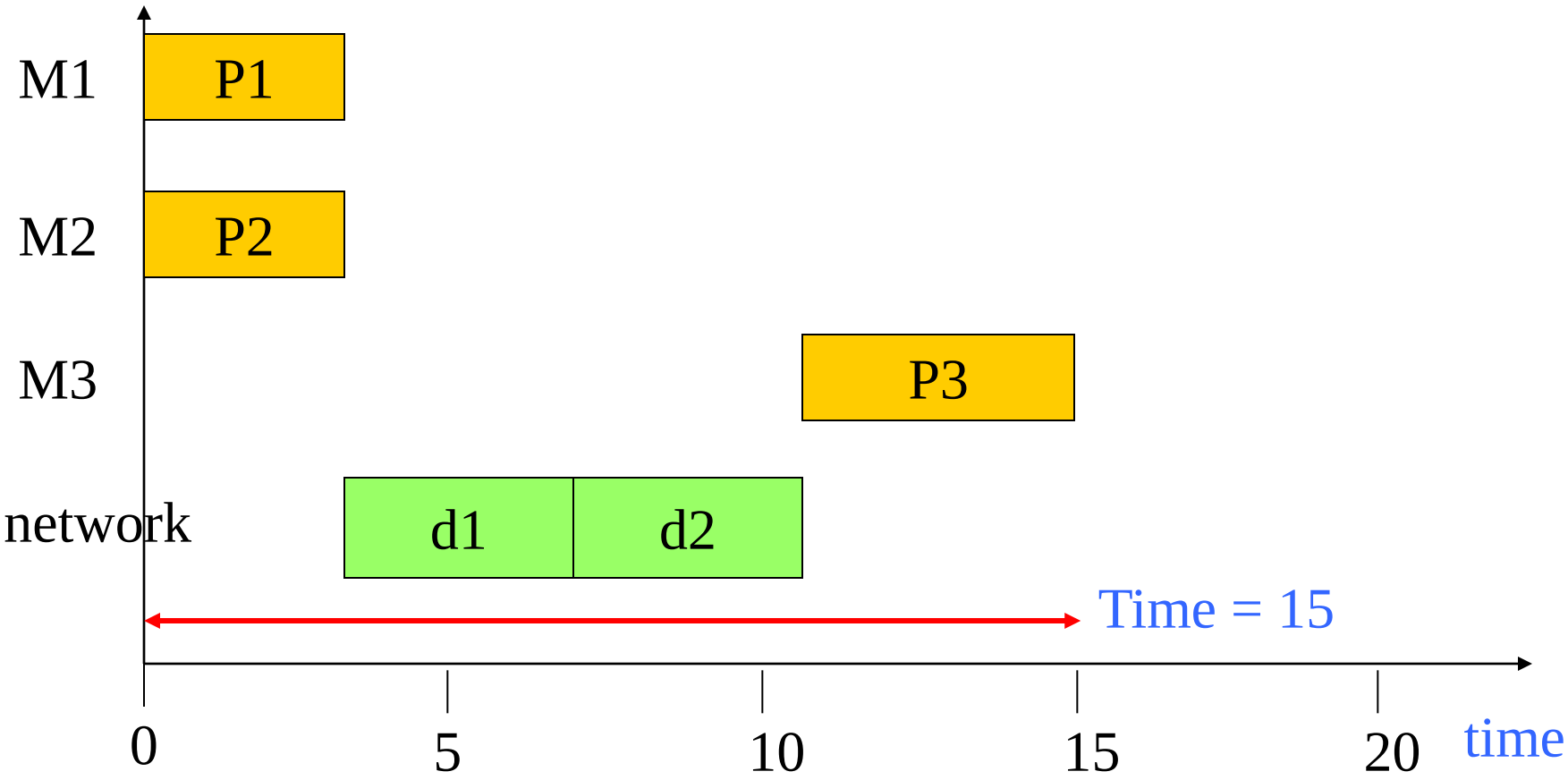
Example: adjusting messages to reduce delay

- Task graph:
- Network:



Transmission time = Td1=Td2=4

Initial schedule



New design

- Modify P3:
  - reads one packet of d1, one packet of d2
  - computes partial result
  - continues to next packet

New schedule