ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

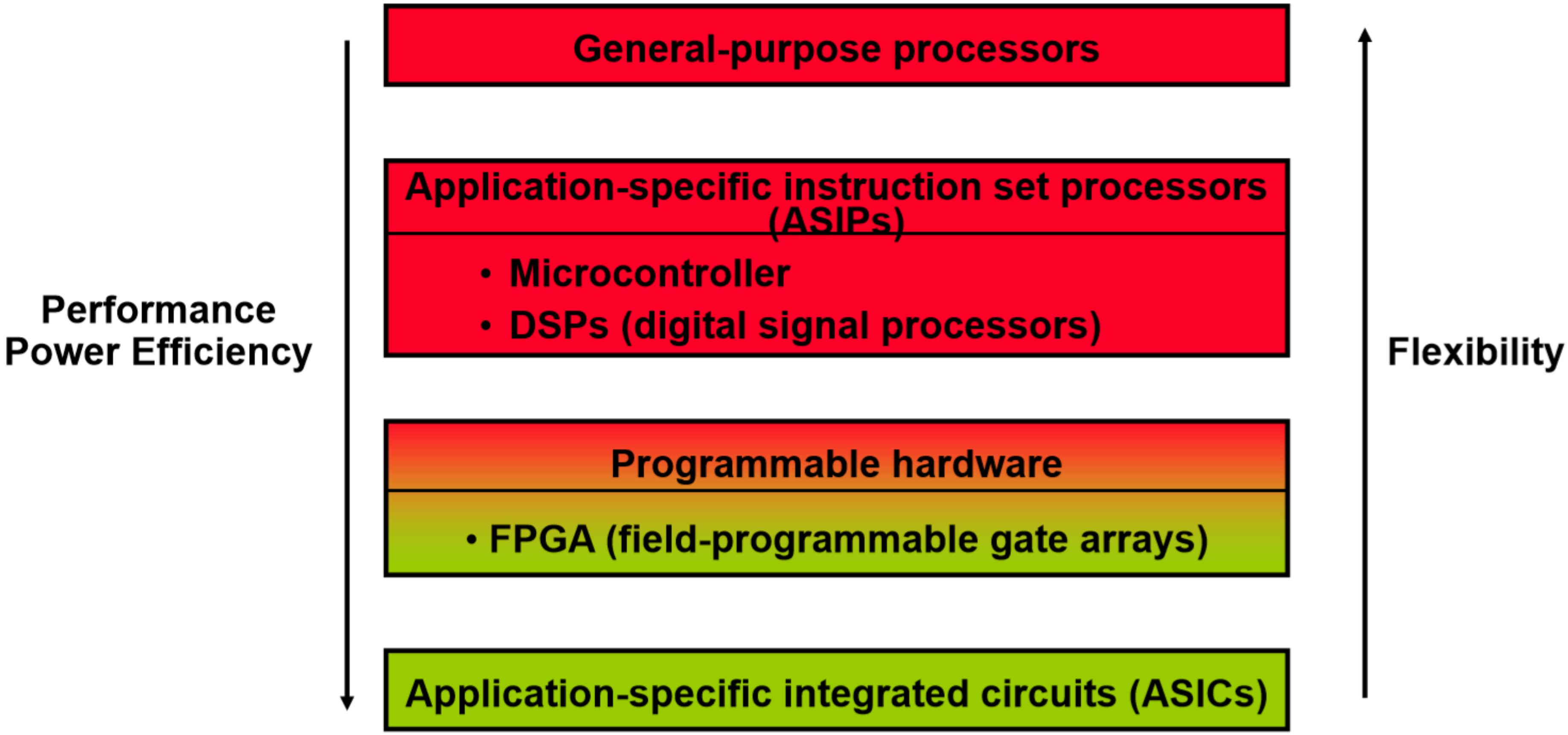
Лекция 6:
Процессоры ИУС РВ

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

Ограничения на процессоры ИУС РВ

- Технологические ограничения:
  - вынужденное применение «грубого» технологического процесса
  - жёсткие ограничения по энергопотреблению и тепловыделению
- Откуда берутся ограничения
  - требование к устойчивости к внешним воздействиям (излучение и т.п.)
  - неразвитость технологических процессов производства микросхем
  - общий лимит на энергопотребление ИУС РВ
  - ограниченные возможности теплоотвода

Implementation Alternatives



General-purpose processors

Application-specific instruction set processors (ASIPs)

- Microcontroller
- DSPs (digital signal processors)

Programmable hardware

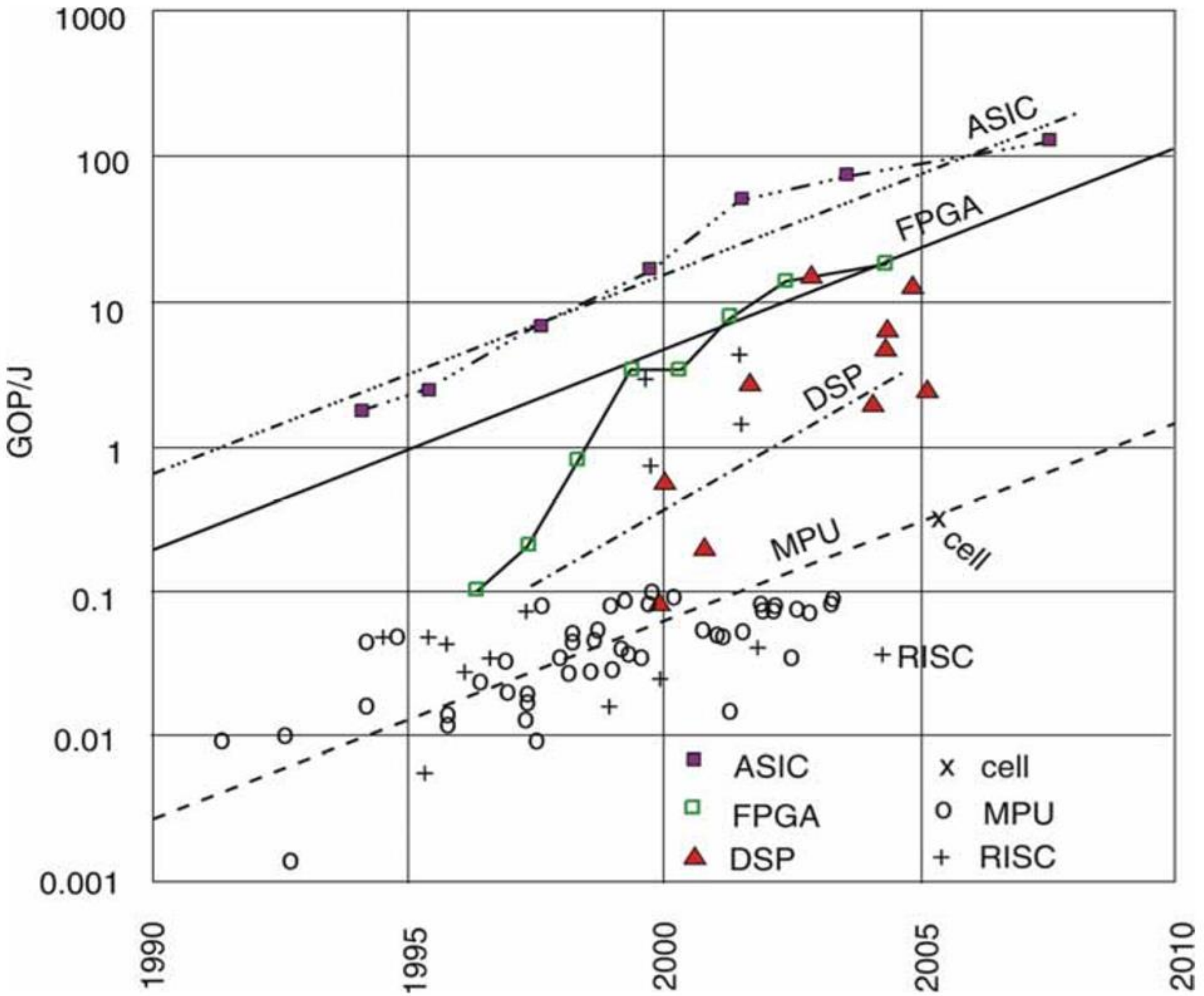
- FPGA (field-programmable gate arrays)

Application-specific integrated circuits (ASICs)

Performance
Power Efficiency

Flexibility

Energy Efficiency

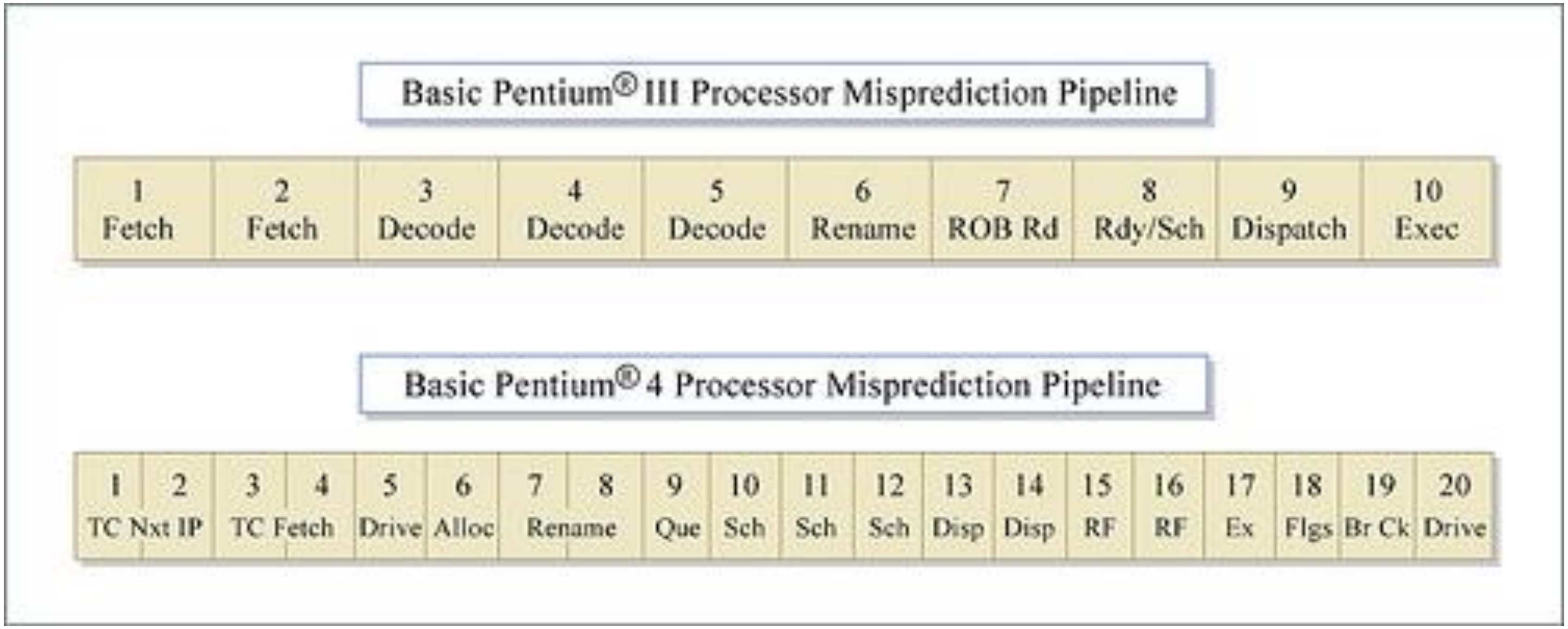


© Hugo De Man,
IMEC, Philips, 2007

General purpose processors
+
specialization

General-purpose Processors

- ***High performance***
  - Highly optimized circuits and technology
  - Use of parallelism
    - superscalar: dynamic scheduling of instructions
    - super-pipelining: instruction pipelining, branch prediction, speculation
  - complex memory hierarchy
- ***Not suited for real-time applications***
  - Execution times are highly unpredictable because of intensive resource sharing and dynamic decisions
- ***Properties***
  - Good average performance for large application mix
  - High power consumption



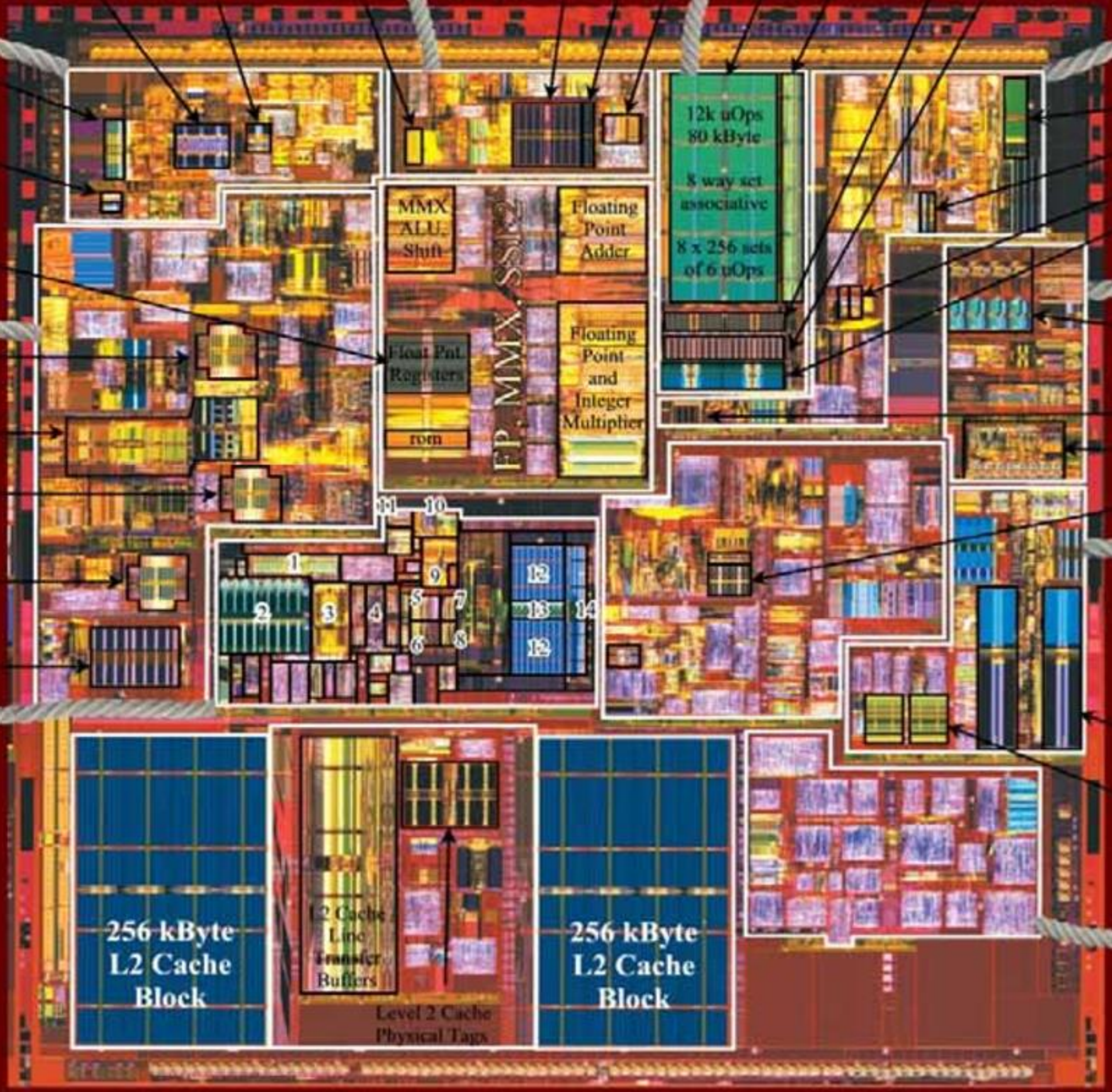
Basic Pentium® III Processor Misprediction Pipeline

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium® 4 Processor Misprediction Pipeline

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive

Intel Pentium 4 Northwood



Buffer Allocation & Register Rename

Instruction Queue (for less critical fields of the uOps)

General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)

Floating Point, MMX, SSE2 Renamed Register File 128 entries of 128 bit.

uOp Schedulers

FP Move Scheduler: (8x8 dependency matrix)

Parallel (Matrix) Scheduler for the two double pumped ALU's

General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)

Load / Store uOp Scheduler: (8x8 dependency matrix)

Load / Store Linear Address Collision History Table

Integer Execution Core

(1) uOp Dispatch unit & Replay Buffer
Dispatches up to 6 uOps / cycle

(2) Integer Renamed Register File
128 entries of 32 bit + 6 status flags
12 read ports and six write ports

(3) Databus switch & Bypasses to and from the Integer Register File.

(4) Flags, Write Back

(5) Double Pumped ALU 0

(6) Double Pumped ALU 1

(7) Load Address Generator Unit

(8) Store Address Generator Unit

(9) Load Buffer (48 entries)

(10) Store Buffer (24 entries)

Execution Pipeline Start

Register Alias History Tables (2x126)

Register Alias Tables

uOp Queue

Micro code Sequencer

Micro code ROM & Flash

Instruction Trace Cache

Trace Cache Fill Buffers

Distributed Tag comparators 24 bit virtual Tags

Trace Cache Access, next Address Predict

Trace Cache Branch Prediction Table (BTB), 512 entries.

Return Stacks (2x16 entries)

Trace Cache next IP's (2x)

Miscellaneous Tag Data

Instruction Decoder

Up to 4 decoded uOps/cycle out. (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer

Trace Cache LRU bits

Raw Instruction Bytes in

Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Instruction Fetch from L2 cache and Branch Prediction

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total

Instruction TLB's 2x64 entry, fully associative for 4k and 4M pages. In: Virtual address [31:12] Out: Physical address [35:12] + 2 page level bits

Front Side Bus Interface, 400..800 MHz

(11) ROB Reorder Buffer 3x42 entries

(12) 8 kByte Level 1 Data cache
four way set associative. 1R/1W

(13) Summed Address Index decode and Way Predict

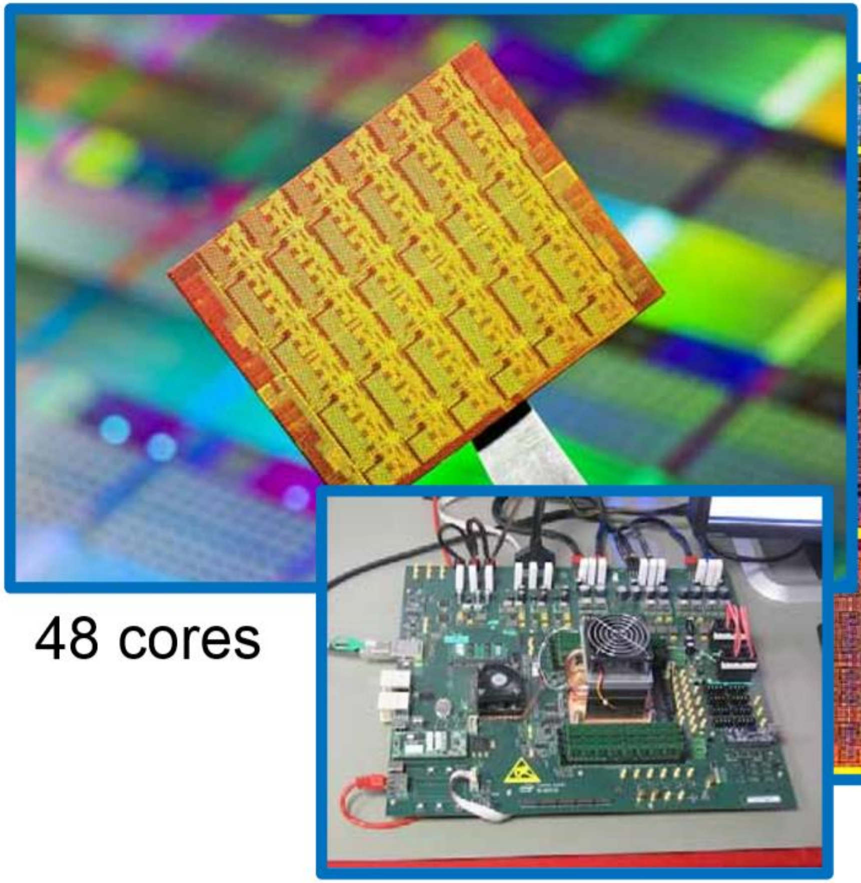
(14) Cache Line Read / Write Transferbuffers and
256 bit wide bus to and from L2 cache

April 19, 2003 www.chip-architect.com

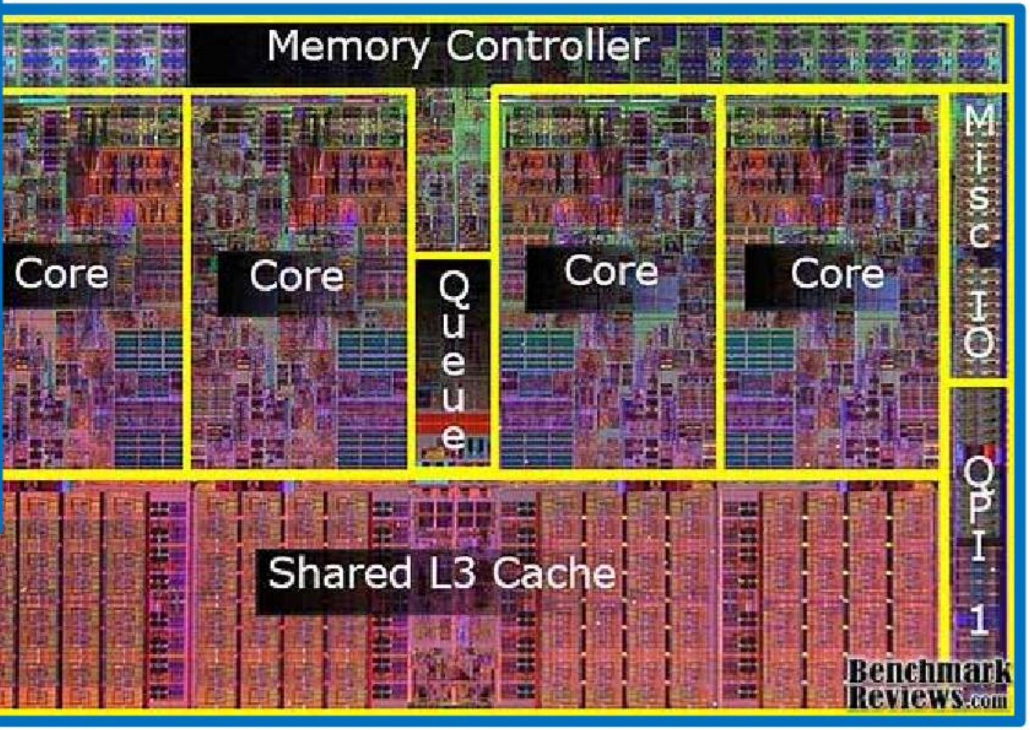
General-purpose Processors

- ***Multicore Processors***
  - Potential of providing higher execution performance by exploiting parallelism
  - Especially useful in high-performance embedded systems, e.g. autonomous driving
  - Disadvantages and problems for embedded systems:
    - Increased interference on shared resources such as buses and shared caches
    - Increased timing uncertainty
    - Often, there is limited parallelism in embedded applications

Multicore Examples

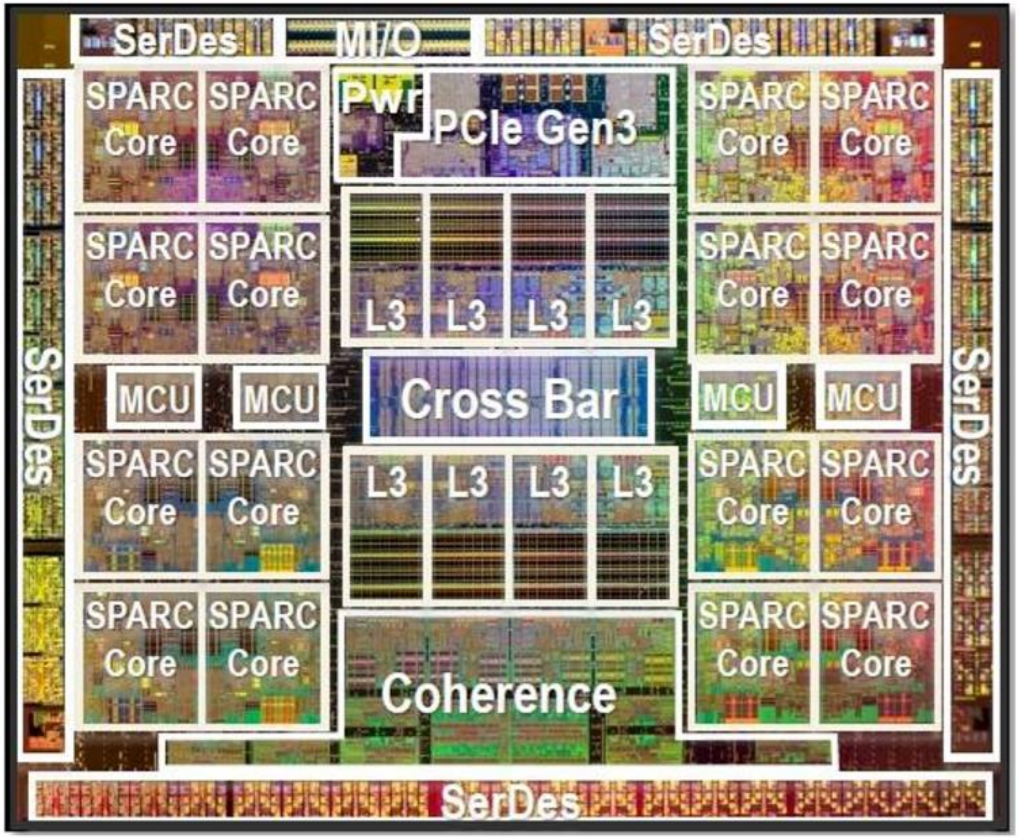


48 cores



4 cores

Multicore Examples



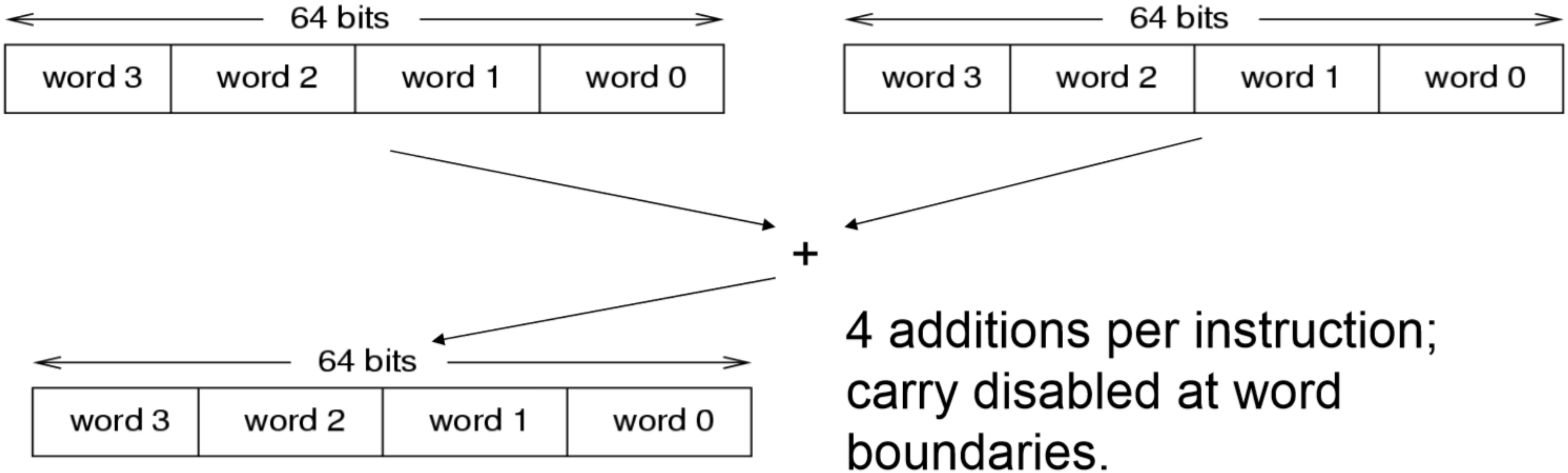
Oracle Sparc T5

System Specialization

- The main difference between general purpose highest volume microprocessors and embedded systems is ***specialization.***
- ***Specialization should respect flexibility***
  - application domain specific systems shall cover a class of applications
  - some flexibility is required to account for late changes, debugging
- ***System analysis required***
  - identification of application properties which can be used for specialization
  - quantification of individual specialization effects

Example: Multimedia-Instructions

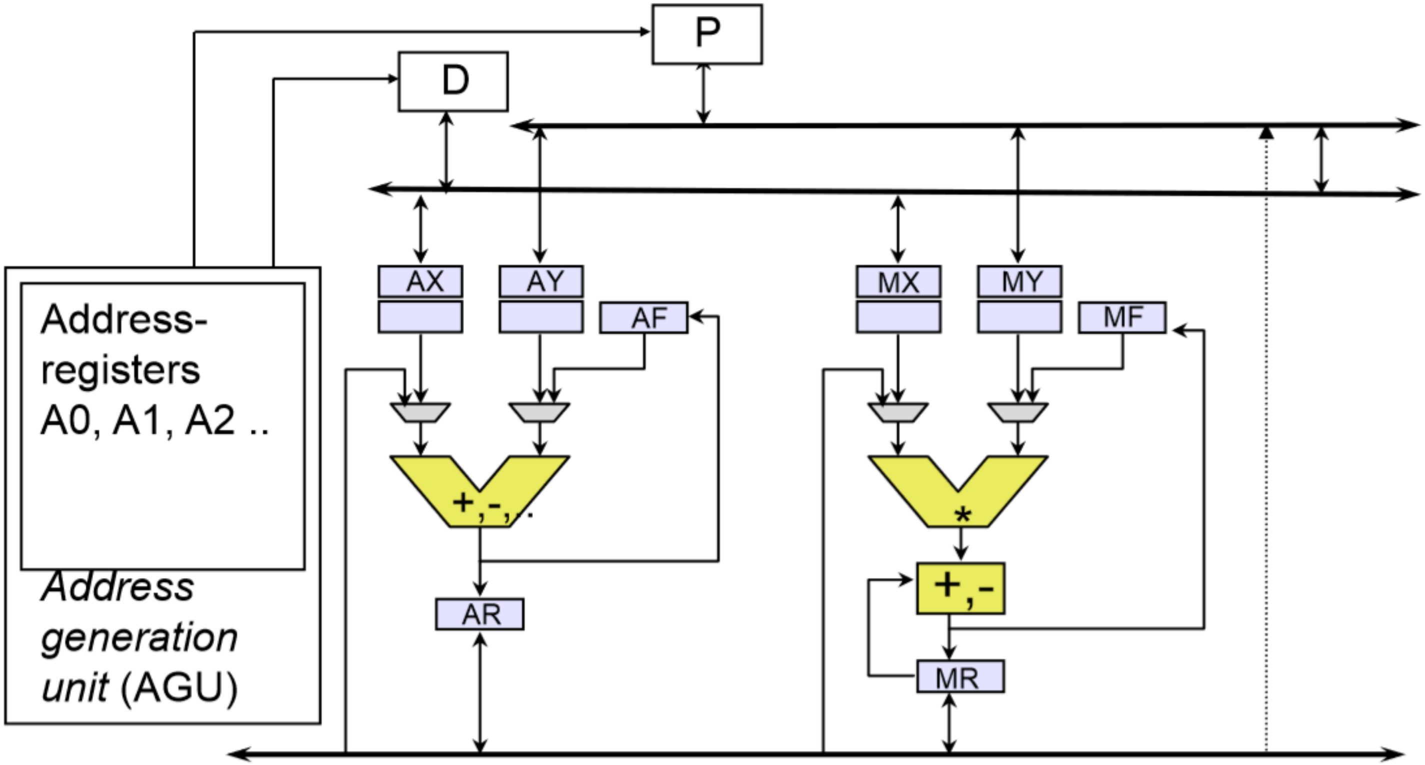
Multimedia instructions exploit that many registers, adders etc are quite wide (32/64 bit), whereas most multimedia data types are narrow (e.g. 8 bit per color, 16 bit per audio sample per channel)
☞ 2-8 values can be stored per register and added.



4 additions per instruction; carry disabled at word boundaries.

Example: Heterogeneous registers

Example (ADSP 210x):

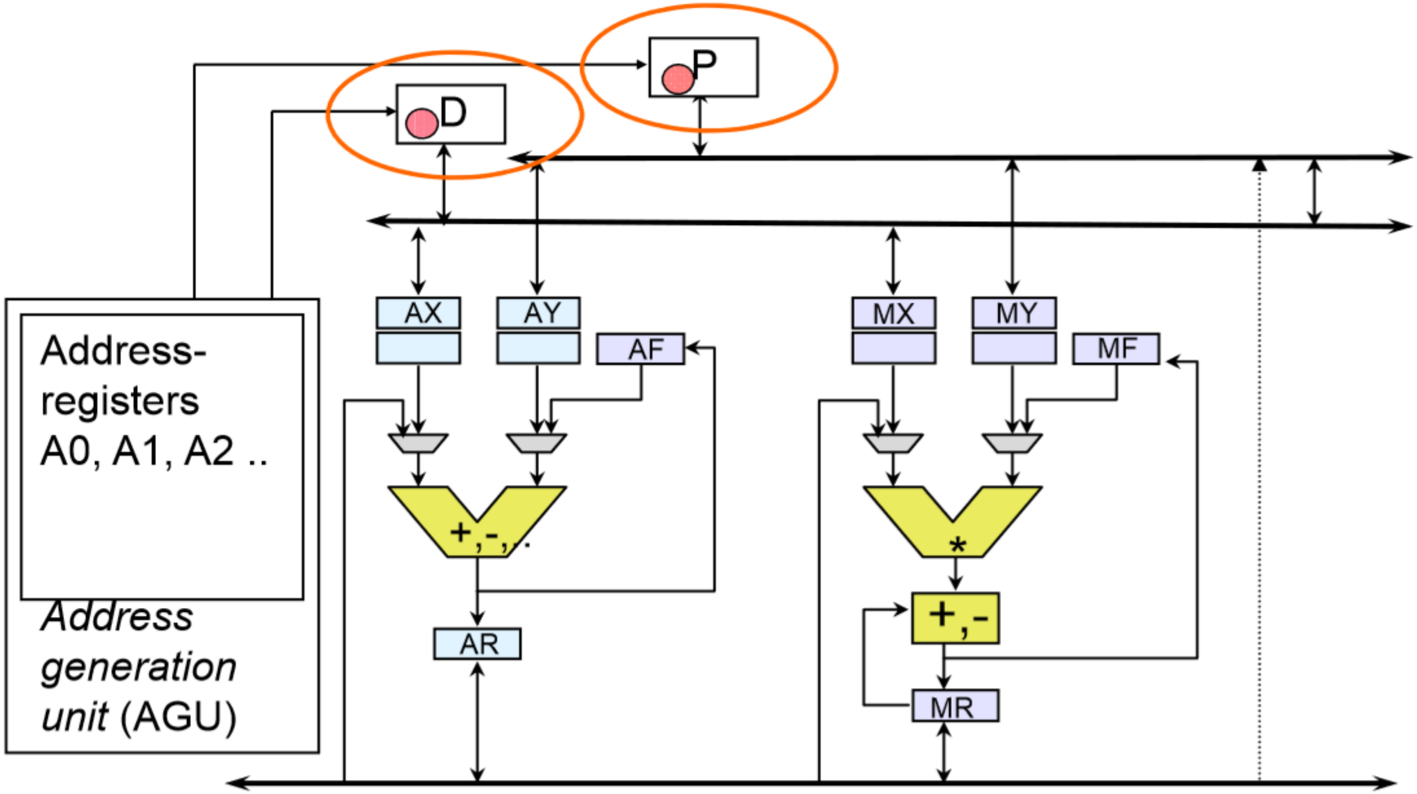


Address-registers A0, A1, A2 ..

Address generation unit (AGU)

Different functionality of registers AR, AX, AY, AF,MX, MY, MF, MR

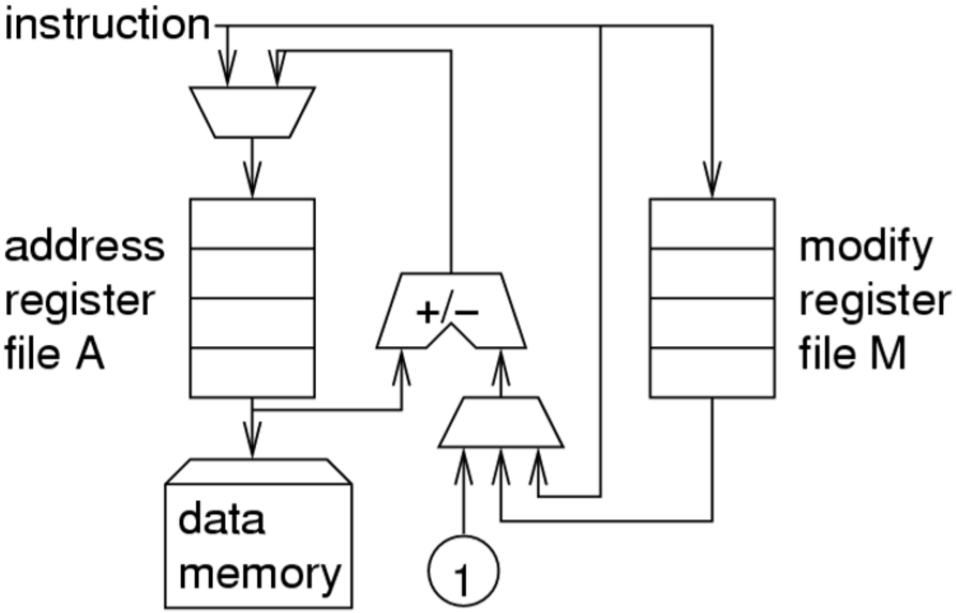
Example: Multiple memory banks or memories



Simplifies parallel fetches

Example: Address generation units

Example (ADSP 210x):



- Data memory can only be fetched with address contained in register file A, but its update can be done in parallel with operation in main data path (takes effectively 0 time).
- Register file A contains several precomputed addresses A[i].
- There is another register file M that contains modification values M[j].

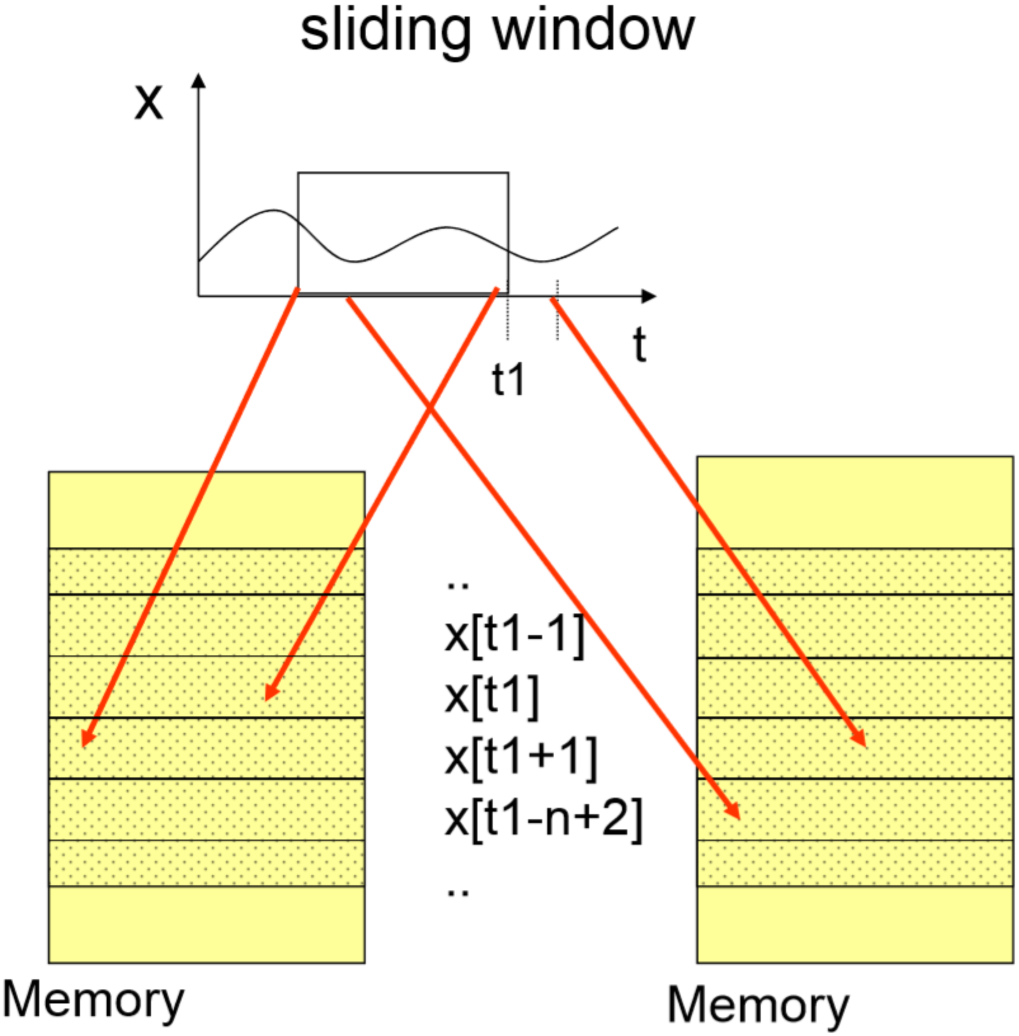
- Possible updates:
  M[j] := ‘immediate’
  A[i] := A[i] ± M[j]
  A[i] := A[i] ± 1
  A[i] := A[i] ± ‘immediate’
  A[i] := ‘immediate’

Example: Modulo addressing

Modulo addressing:
Am++ ≡ Am:=(Am+1) **mod n**
(implements ring or circular buffer in memory)

x[t]: value accessed at time t

..
x[t1-1]
x[t1]
x[t1-n+1]
x[t1-n+2]
..

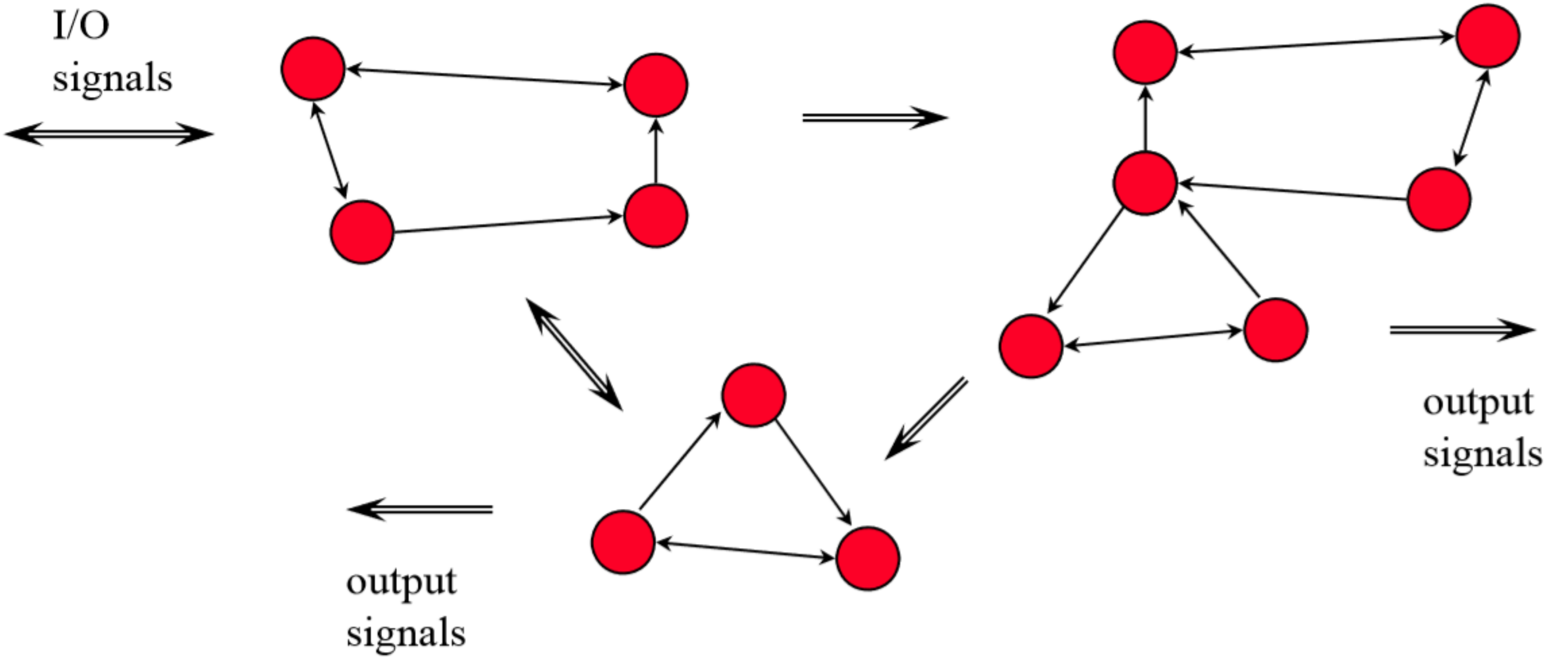


..
x[t1-1]
x[t1]
x[t1+1]
x[t1-n+2]
..

Application specific processors: *microcontrollers*

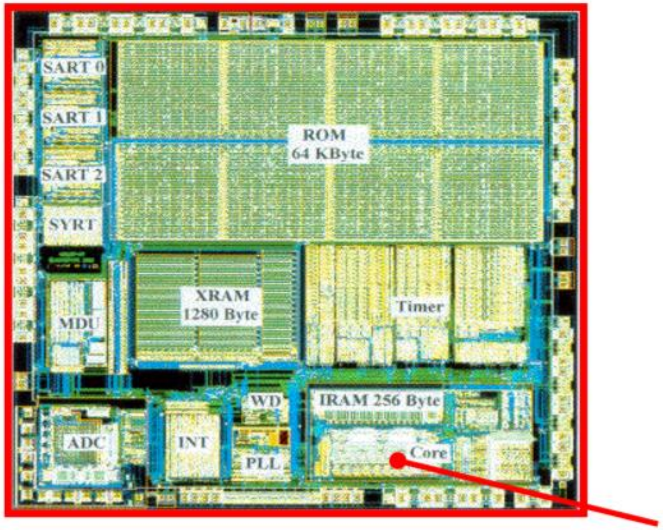
Control Dominated Systems

- Reactive systems with ***event driven behavior***
- Underlying semantics of system description ("input model of computation") typically (coupled) Finite State Machines or Petri Nets



Microcontroller

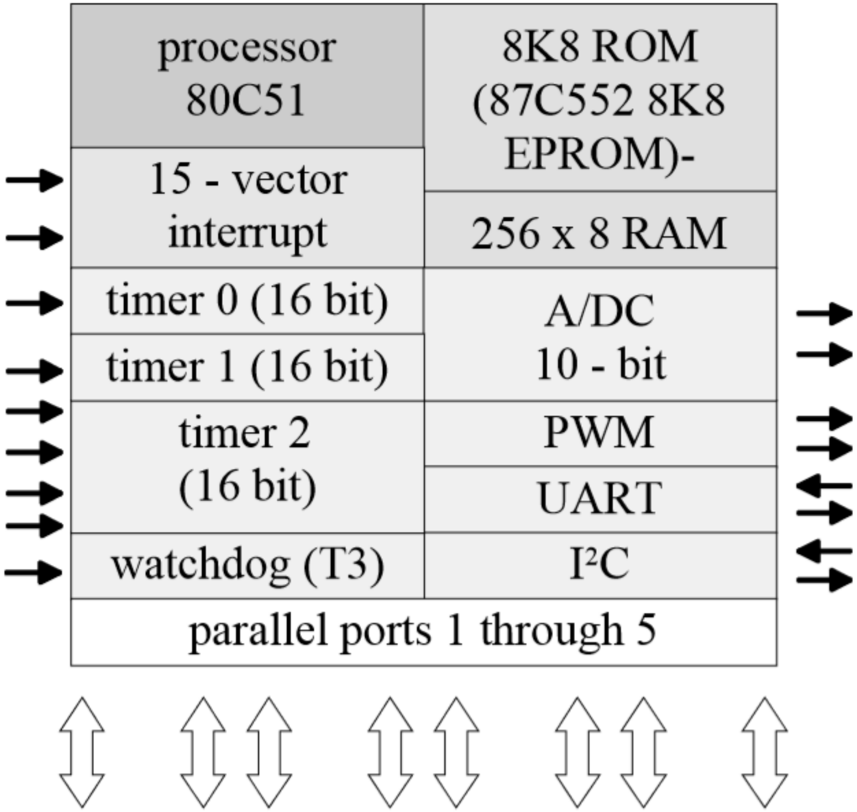
- control-dominant applications
  - supports process scheduling and synchronization
  - preemption (interrupt), context switch
  - short latency times
- low power consumption
- peripheral units often integrated
- suited for real-time applications



8051 core

SIECO51 (Siemens)

Microcontroller as a System-on-Chip



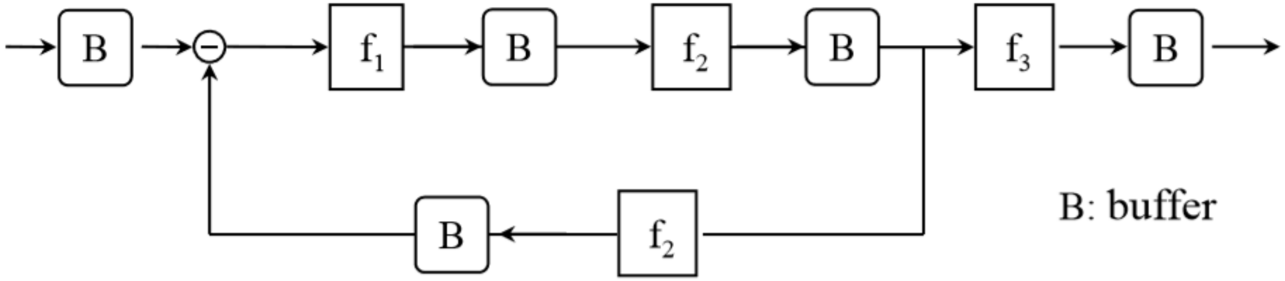
- complete system
- timers
- I²C-bus and par./ser. interfaces for communication
- A/D converter
- watchdog (SW activity timeout): safety
- on-chip memory
- interrupt controller

Philips 83 C552: 8 bit-8051 based microcontroller

Application specific processors: *DSP & VLIW*

Data Dominated Systems

- ***Streaming oriented systems*** with mostly periodic behavior
- Underlying semantics of input description e.g. ***flow graphs*** ("input model of computation")



B: buffer

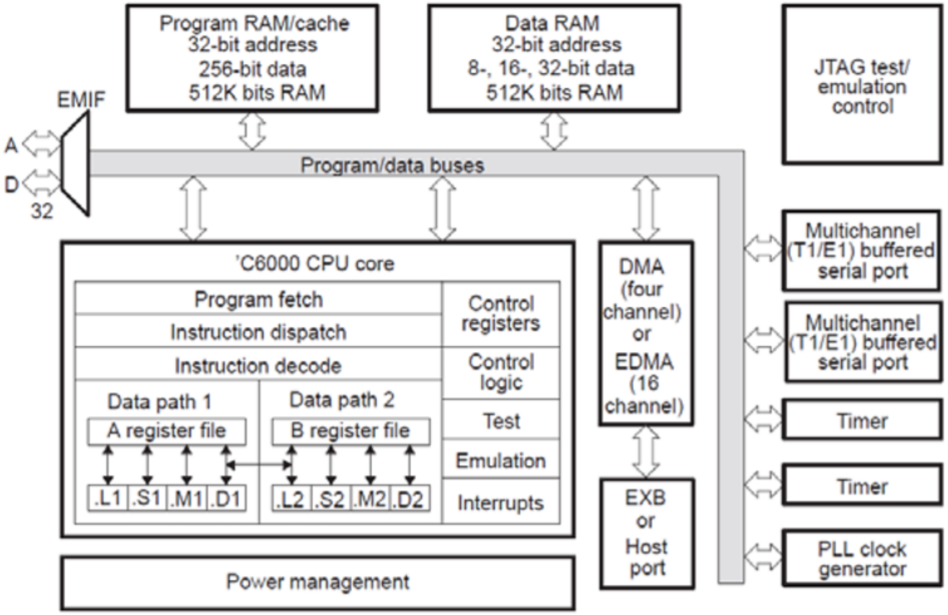
- ***Application examples***: signal processing, control engineering

Digital Signal Processor

- optimized for data-flow applications
- suited for simple control flow
- parallel hardware units (VLIW)
- specialized instruction set
- high data throughput
- zero-overhead loops
- specialized memory

- suited for real-time applications

Figure 2–1. TMS320C62x/C67x Block Diagram



MAC (multiply & accumulate)

```
sum = 0.0;
for (i=0; i<N; i++)
  sum = sum + a[i]*b[i];
```

zero-overhead loop
(repeat next instruction N times)

MAC - Instruktion

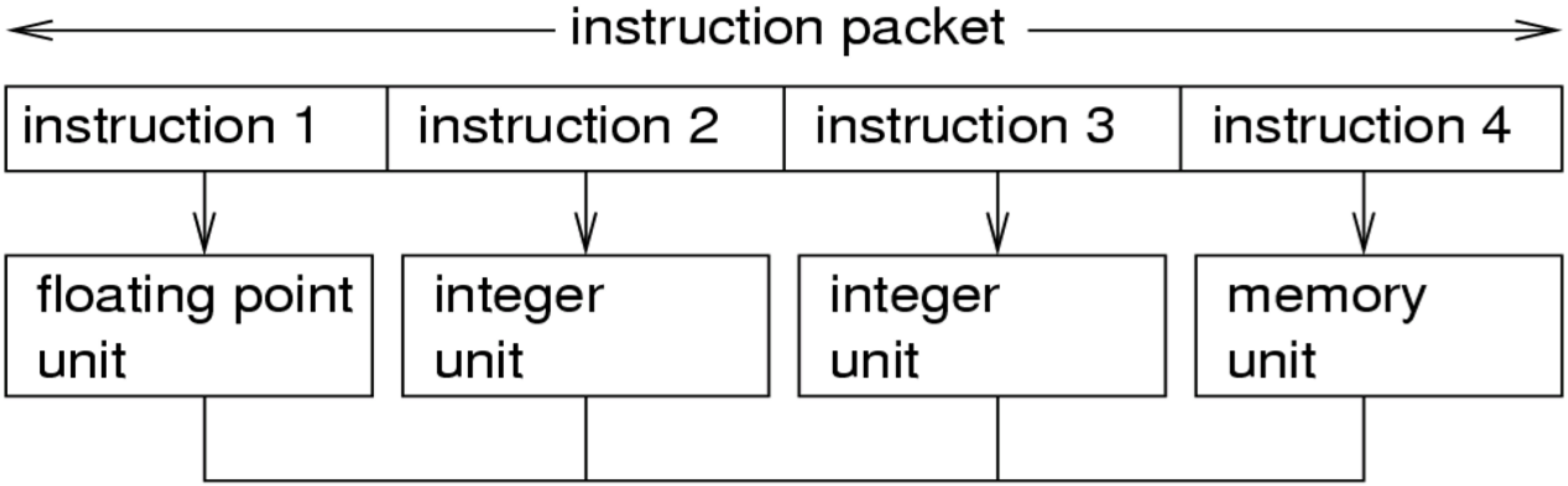
```
    LDF     0, R0
    LDF     0, R1
    RPTS    N
    MPYF3   *(AR0)++, *(AR1)++, R0
||  ADDF3   R0, R1, R1
```

TMS320C3x Assembler
(Texas Instruments)

Very Long Instruction Word (VLIW)

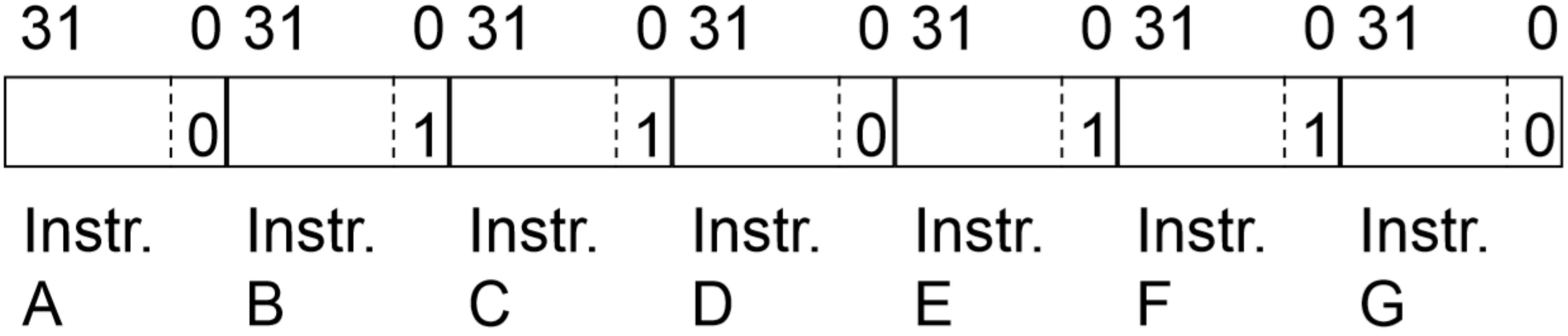
Key idea: detection of possible parallelism to be done by compiler, not by hardware at run-time (inefficient).

VLIW: parallel operations (instructions) encoded in one long word (instruction packet), each instruction controlling one functional unit. E.g.:



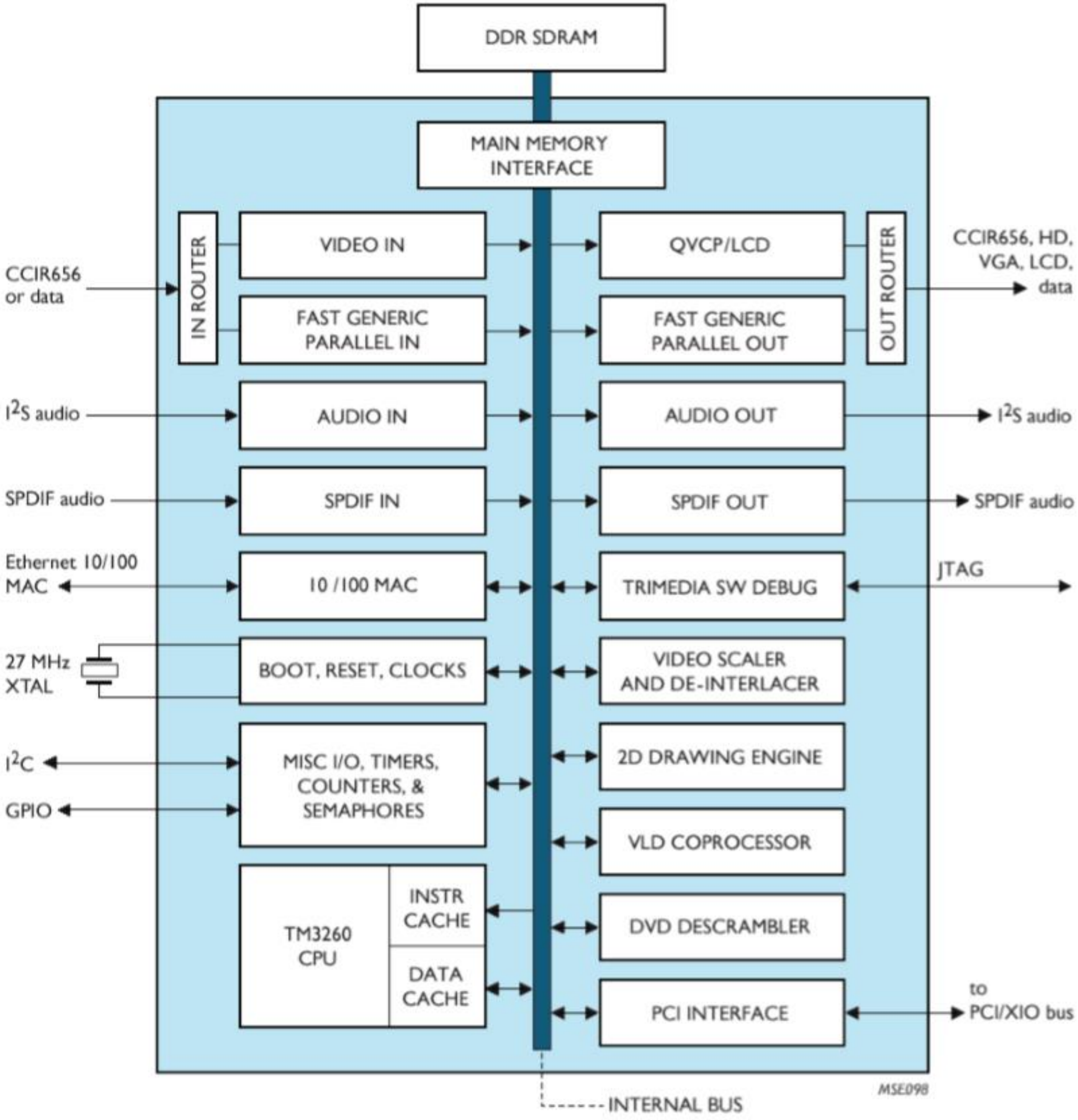
Explicit Parallelism Instruction Computers

The TMS320C62xx VLIW Processor as an example of EPIC:



Cycle	Instruction		
1	A		
2	B	C	D
3	E	F	G

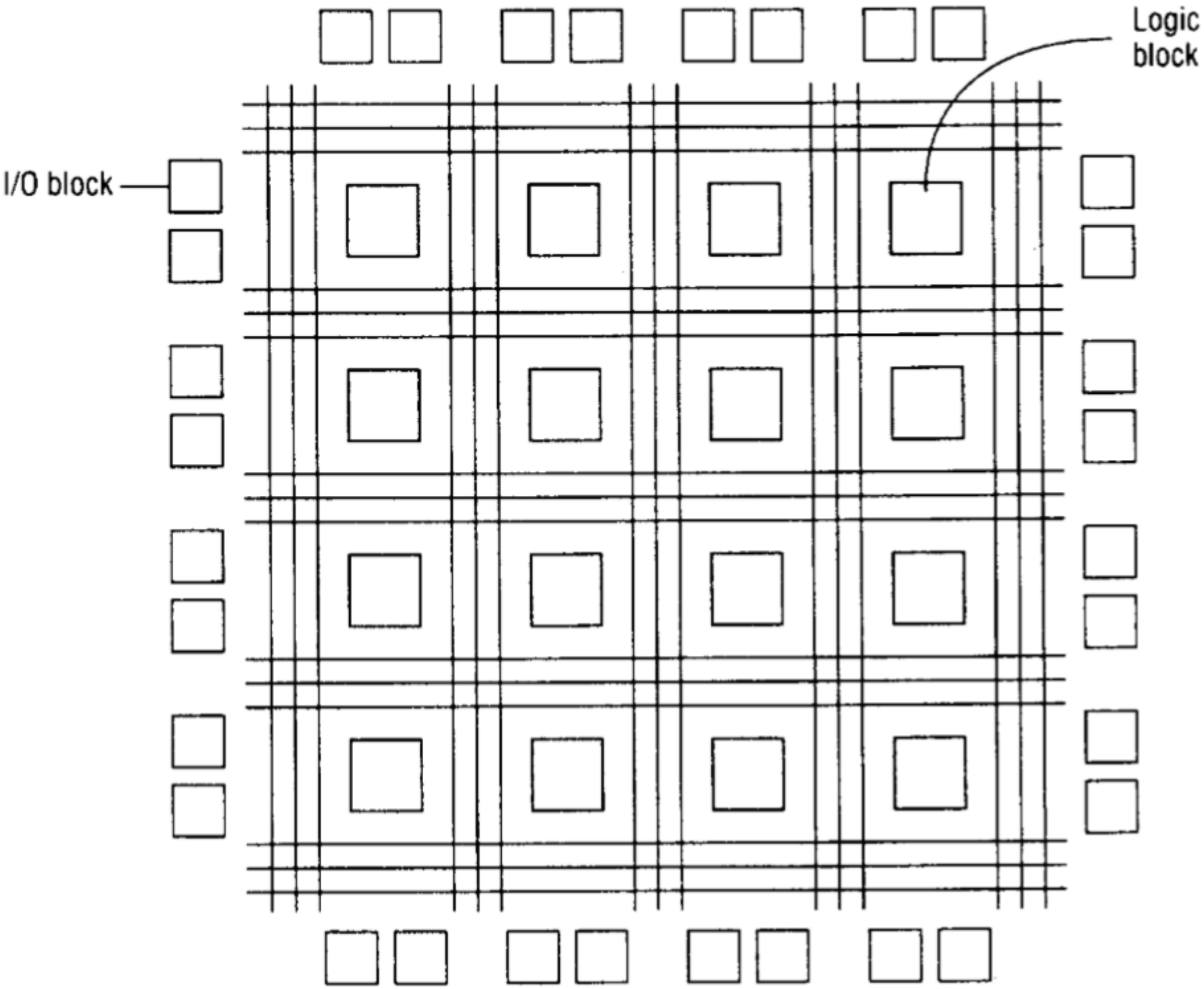
Example: NXP TriMedia TM1000



Programmable hardware: FPGA

FPGA – Basic Strucutre

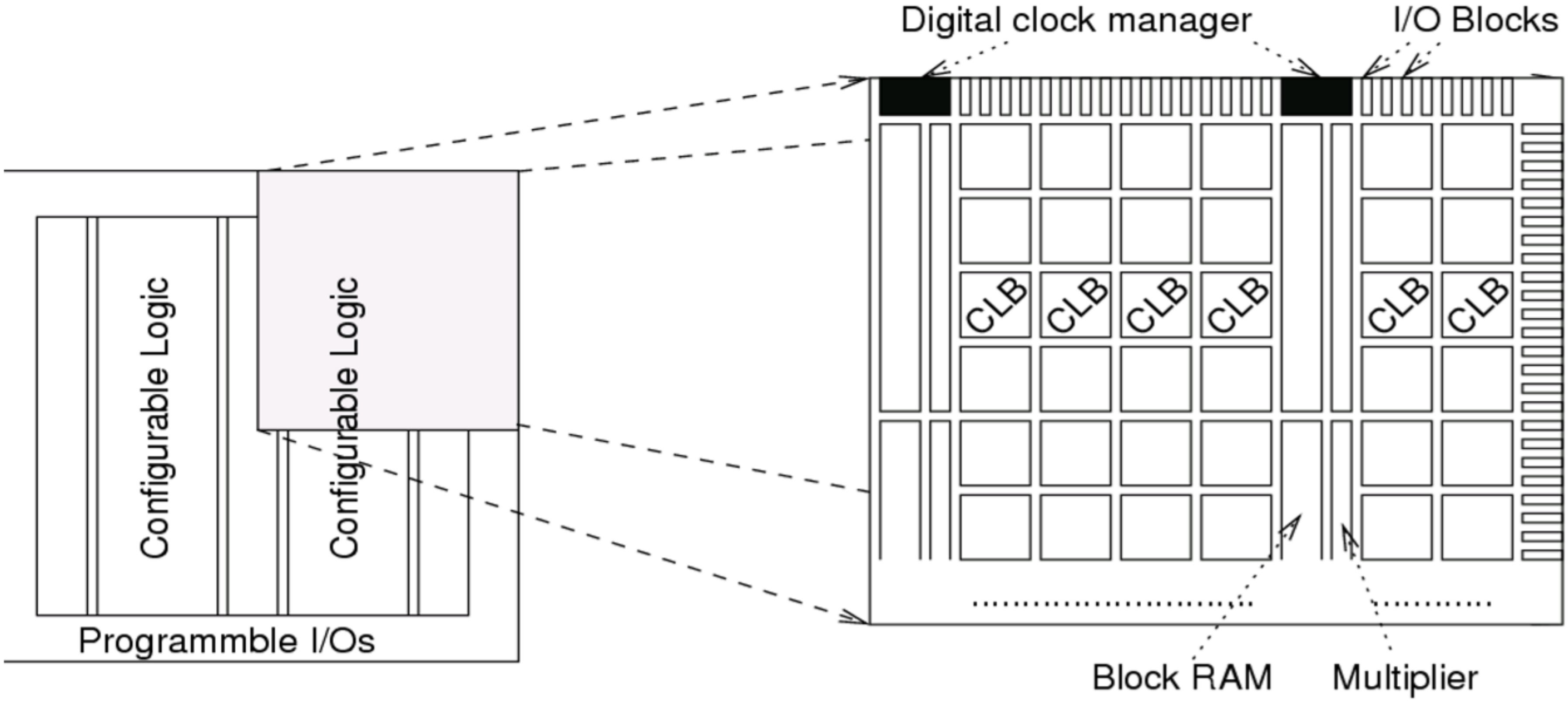
- Logic Units
- I/O Units
- Connections



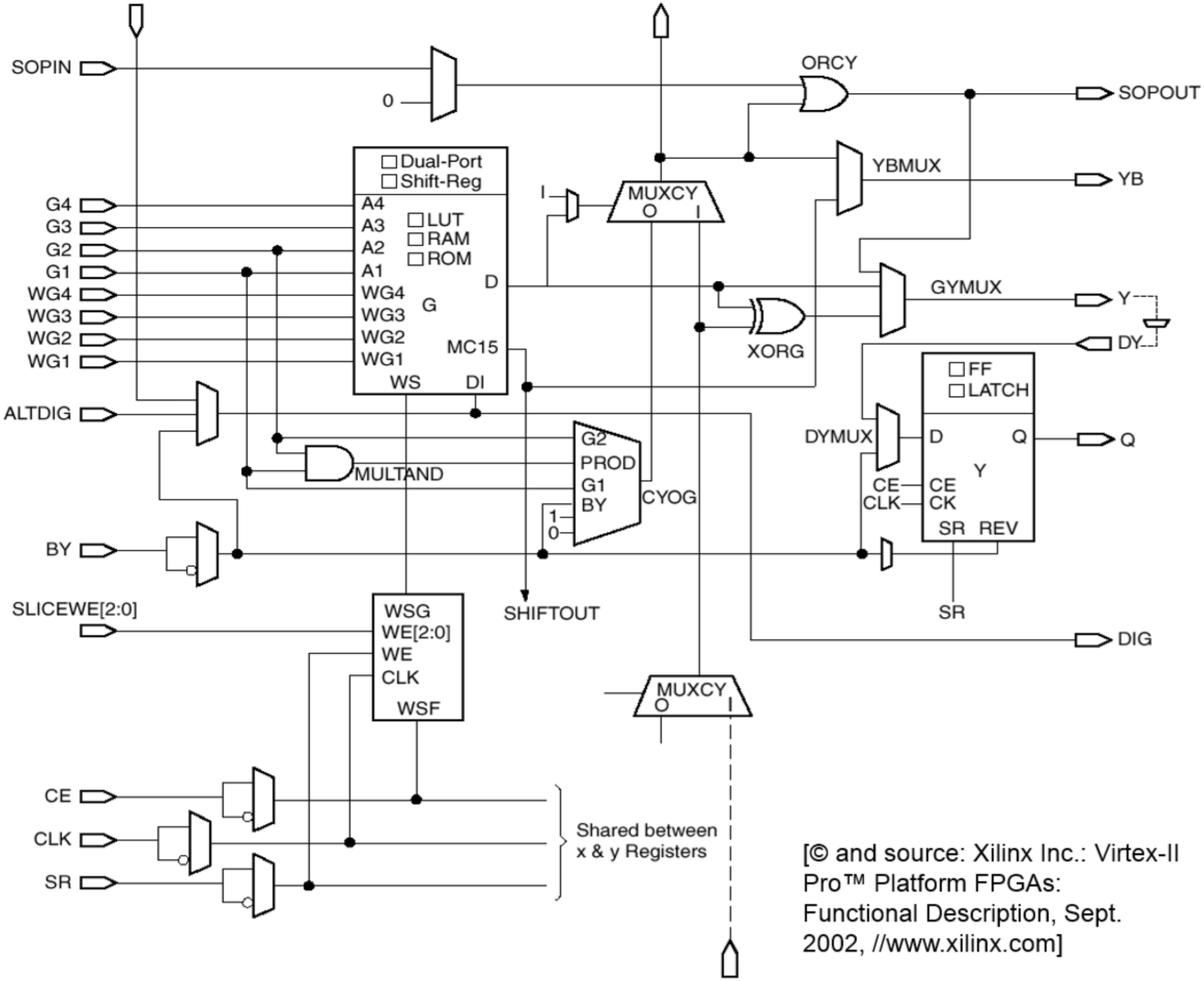
FPGA - Classification

- ***Granularity of logic units***:
  - Gate, tables, memory, functional blocks (ALU, control, data path, processor)
- ***Communication network***:
  - Crossbar, hierarchical mesh, tree
- ***Reconfiguration***:
  - fixed at production time, once at design time, dynamic during run-time

Floor-plan of VIRTEX II FPGAs



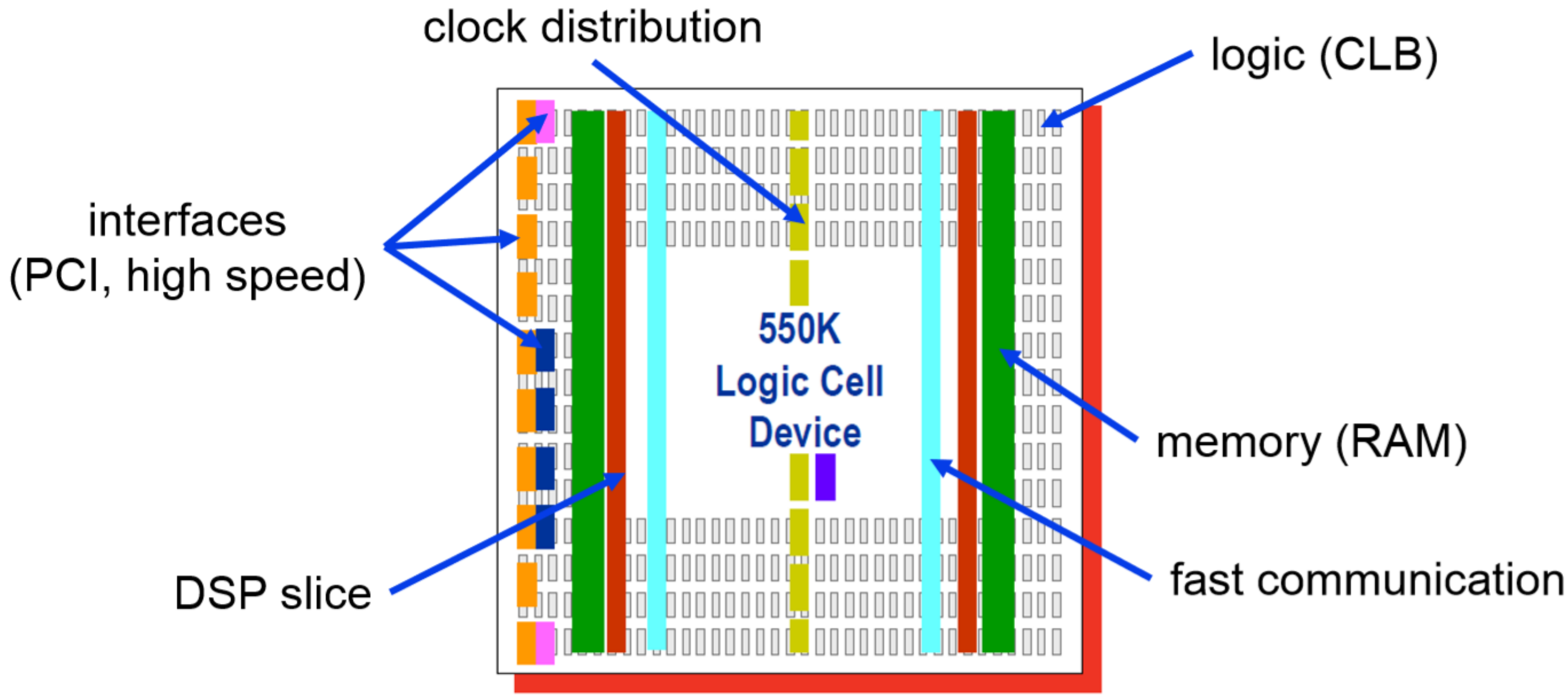
Virtex Logic Cell



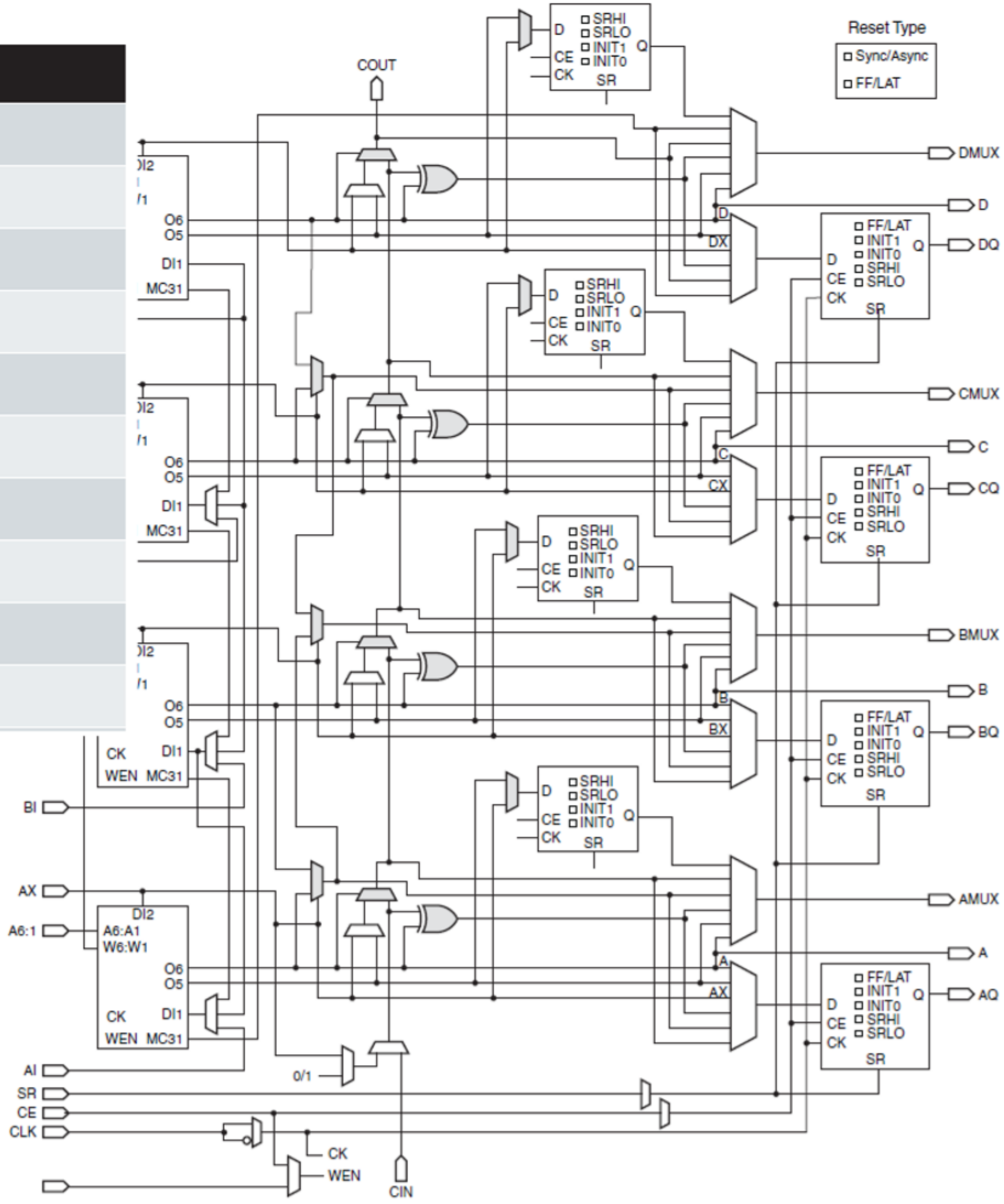
[© and source: Xilinx Inc.: Virtex-II Pro™ Platform FPGAs: Functional Description, Sept. 2002, //www.xilinx.com]

Example Virtex-6

- Combination of flexibility (CLB's), Integration and performance (heterogeneity of hard-IP Blocks)



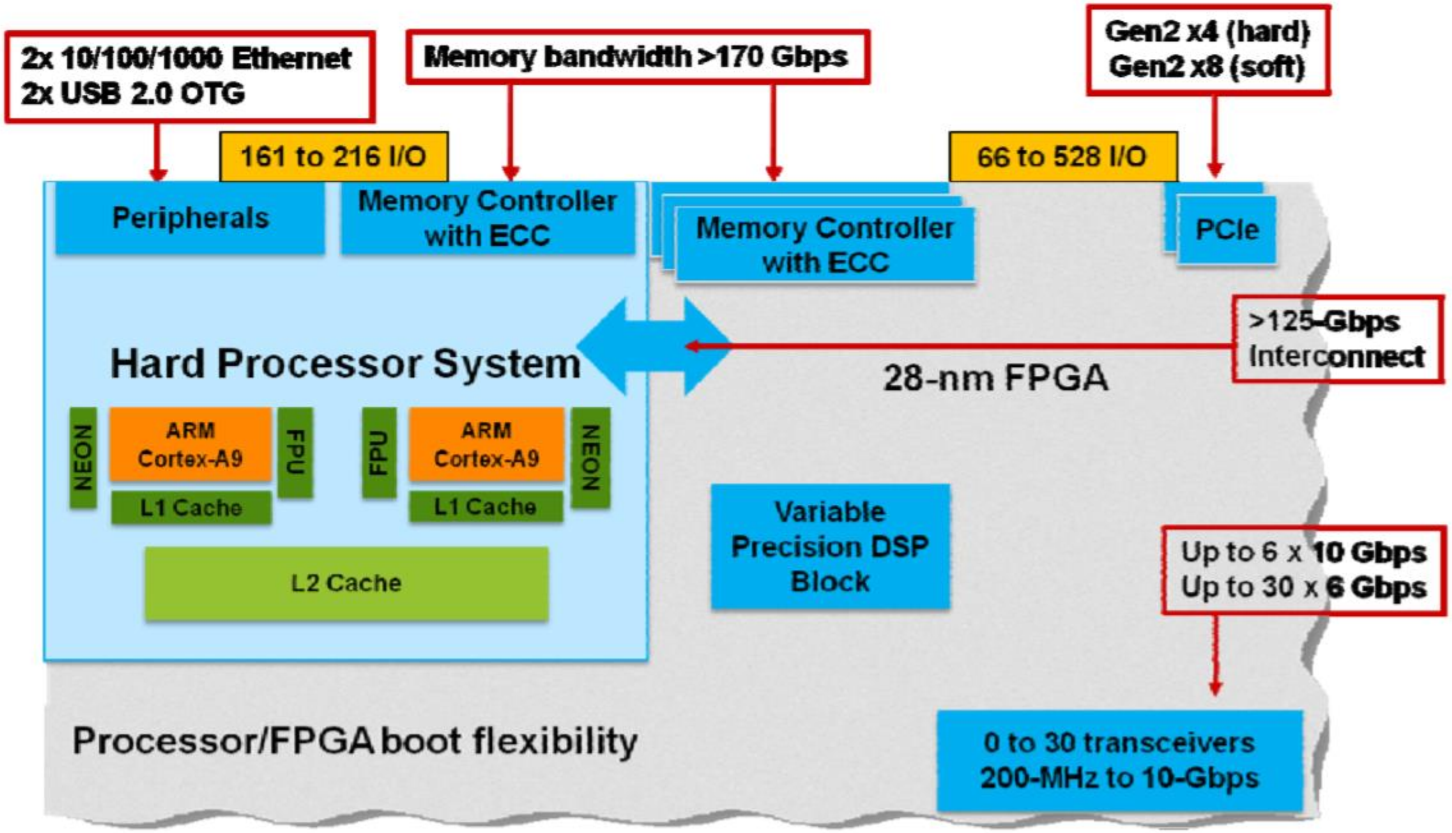
MAXIMUM CAPABILITY	VIRTEX-7 FPGAS
Logic Cells	1,955K
Block RAM	85Mb
DSP Slices	5,280
Peak DSP Performance (symmetric FIR)	6,737 GMACS
Transceiver Count	96
Peak Transceiver Speed	28.05Gbps
Peak Serial Bandwidth (full duplex)	2,784Gbps
PCI Express® Interface	Gen3 x8*
Memory Interface	1,866Mbps
I/O Pins	1,200



Virtex-6 CLB Slice

Configurable System-On-Chip

Example:
Altera's SoC FPGA integrates a dual-core ARM Cortex-A9 processor system with a low power FPGA fabrics



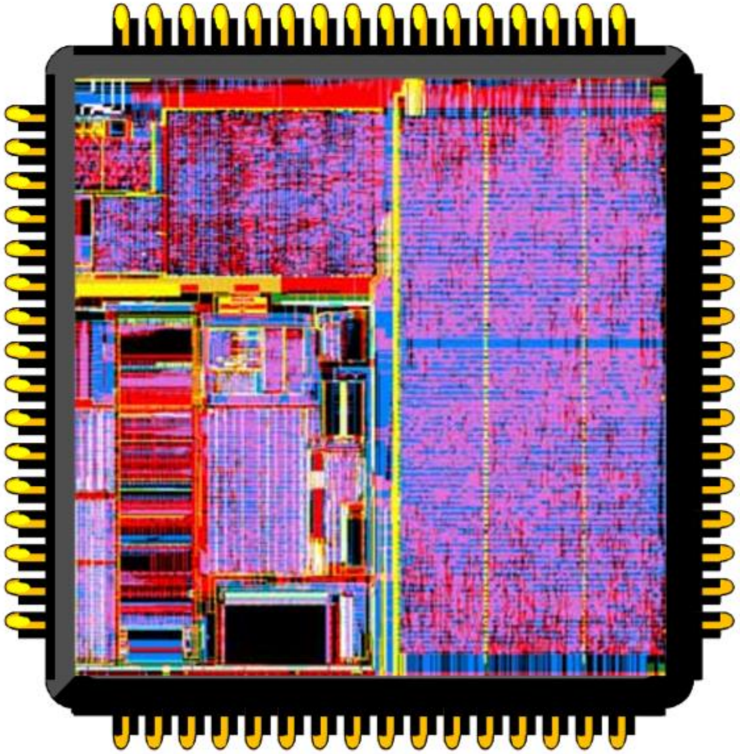
Application Specific Circuits (ASICS)

Custom-designed circuits necessary

- if ultimate speed or
- energy efficiency is the goal and
- large numbers can be sold.

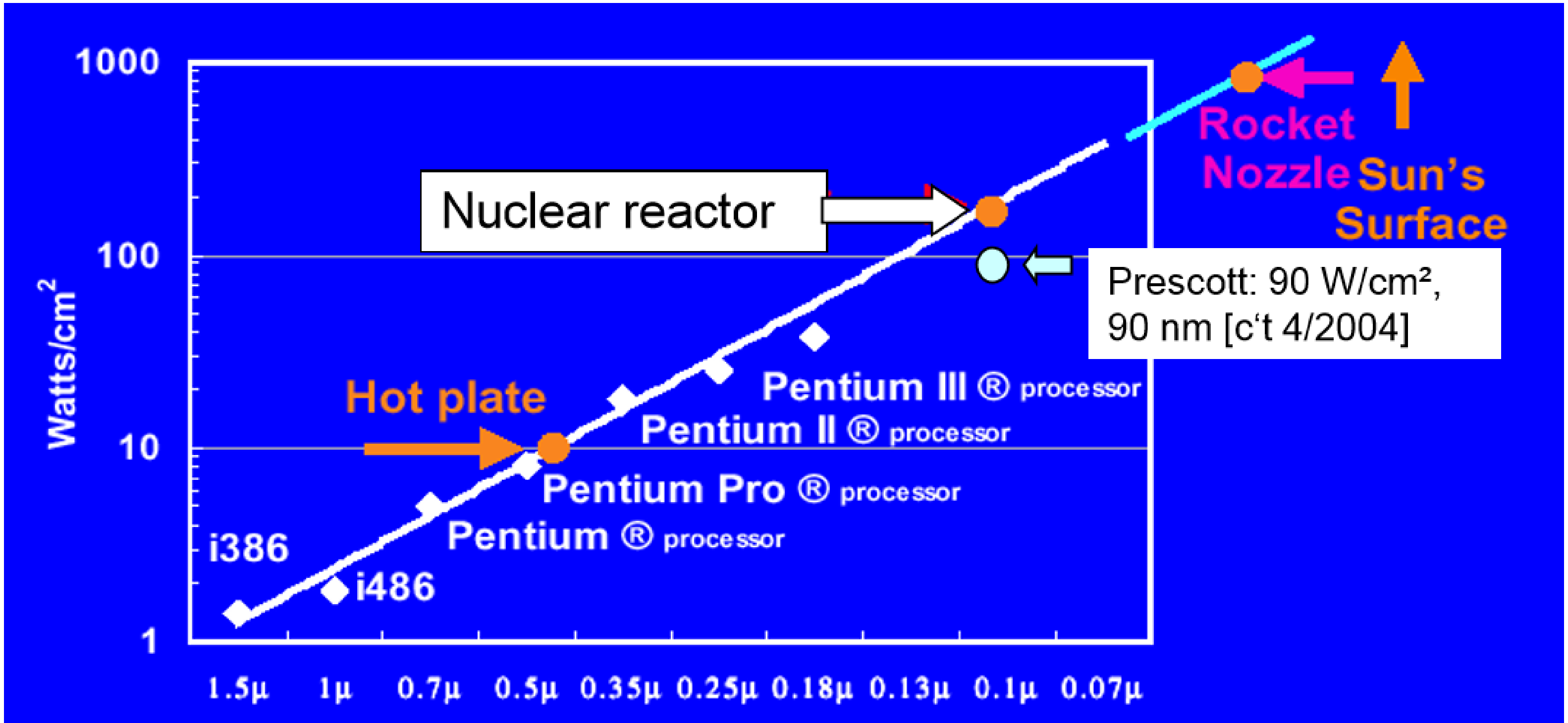
Approach suffers from

- long design times,
- lack of flexibility (changing standards) and
- high costs (e.g. Mill. $ mask costs).



Power aware design & scheduling

PCs: Problem: Power density increasing



Prescott: 90 W/cm², 90 nm [c't 4/2004]

Surpassed hot-plate power density in 0.5µ

Not too long to reach nuclear reactor

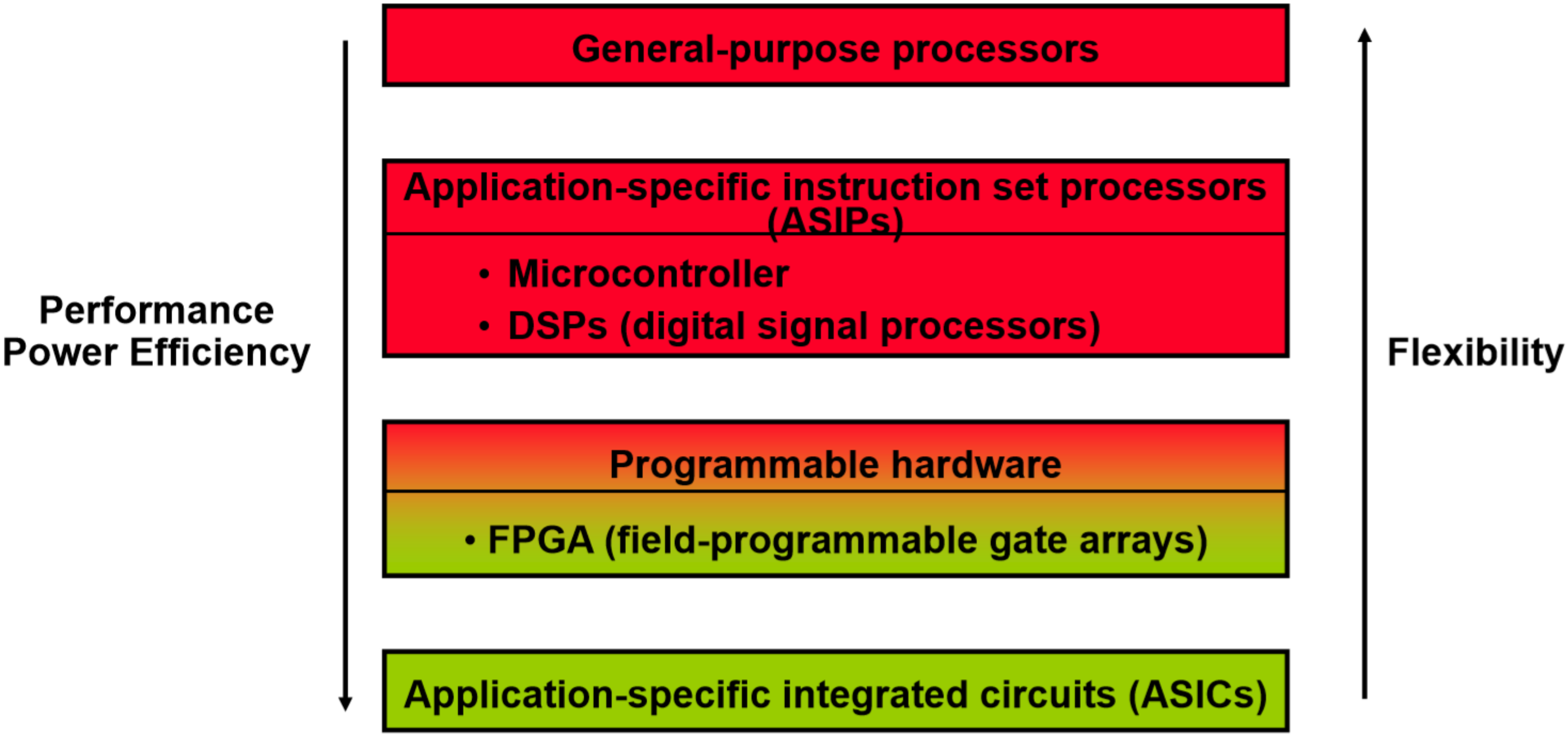
© Intel
M. Pollack,
Micro-32

PCs: Surpassed hot (kitchen) plate …? Why not use it?

Strictly speaking, energy is not “consumed”, but converted from electrical energy into heat energy

http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html

Implementation Alternatives



Performance
Power Efficiency

General-purpose processors

Application-specific instruction set processors (ASIPs)

- Microcontroller
- DSPs (digital signal processors)

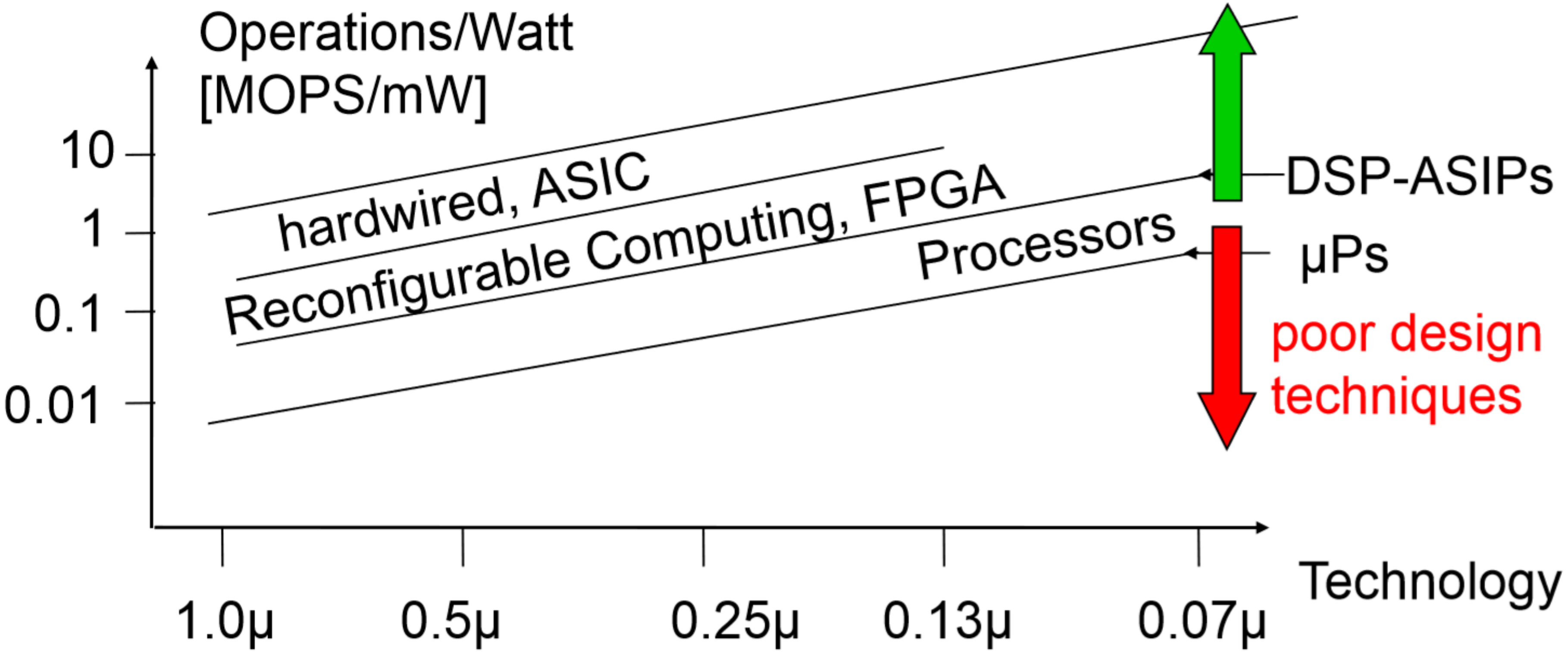
Programmable hardware

- FPGA (field-programmable gate arrays)

Application-specific integrated circuits (ASICs)

Flexibility

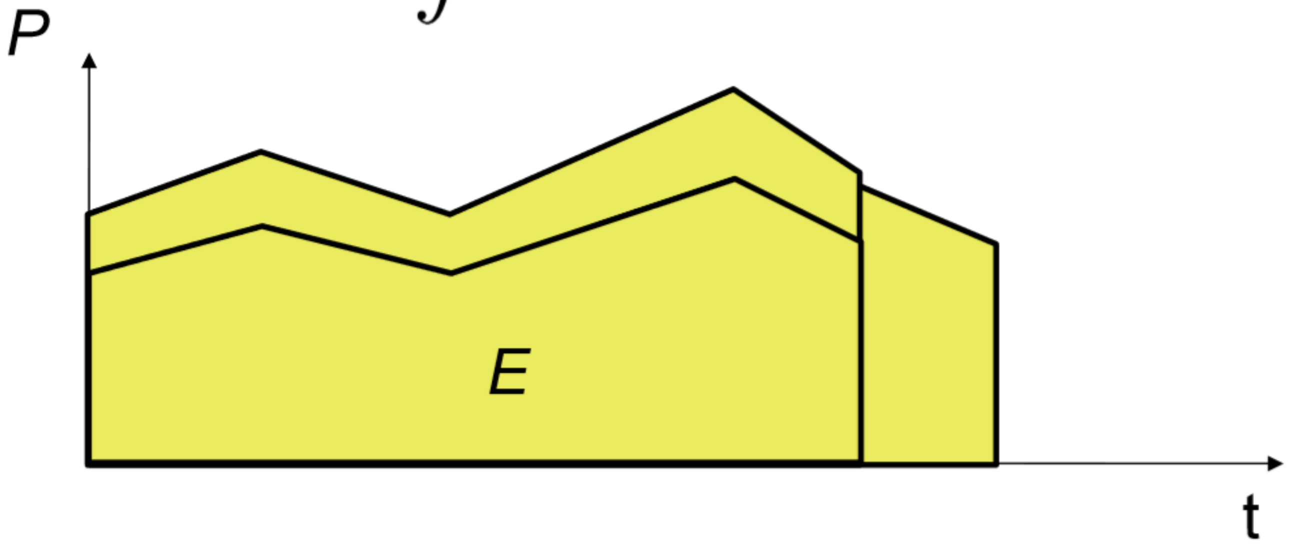
The Power/Flexibility Conflict



Necessary to ***optimize HW and SW.***
Use ***heterogeneous architectures***.
Apply ***specialization techniques***.

Power and Energy are Related

$$E = \int P(t)dt$$



In many cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow faster execution.

Low Power vs. Low Energy

- Minimizing the ***power consumption*** is important for
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - cooling (short term cooling)
    - high cost (estimated to be rising at $1 to $3 per Watt for heat dissipation [Skadron et al. ISCA 2003])
    - limited space
- Minimizing the ***energy consumption*** is important due to
  - restricted availability of energy (mobile systems)
  - limited battery capacities (only slowly improving)
  - very high costs of energy (solar panels, in space)
  - long lifetimes, low temperatures

Dynamic Voltage Scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

$$P \sim \alpha C_L \boxed{V_{dd}^2 f}$$

V_{dd} : supply voltage

α : switching activity

C_L : load capacity

f : clock frequency

Delay for CMOS circuits:

$$\tau \sim C_L \frac{V_{dd}}{(V_{dd} - V_T)^2}$$

V_{dd} : supply voltage

V_T : threshold voltage

$V_T \ll V_{dd}$

Decreasing V_{dd} reduces P quadratically (f constant).

The gate delay increases reciprocally with decreasing V_{dd} .

Maximal frequency f_{max} decreases linearly with decreasing V_{dd} .

Potential for Energy Optimization: DVS

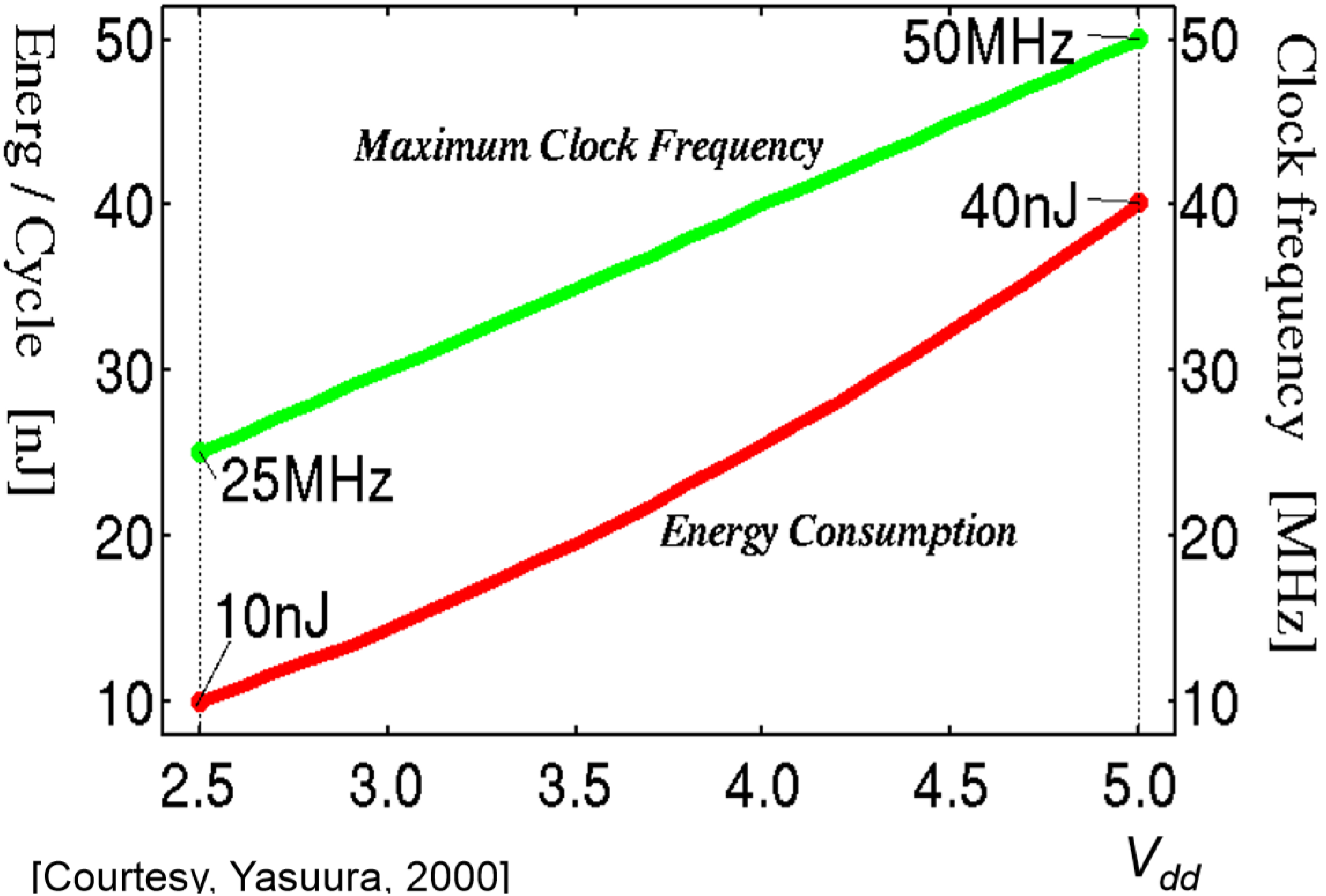
$$P \sim \alpha C_L V_{dd}^2 f$$

$$E \sim \alpha C_L V_{dd}^2 f t = \alpha C_L V_{dd}^2 \ (\#\text{cycles})$$

Saving energy for a given task:

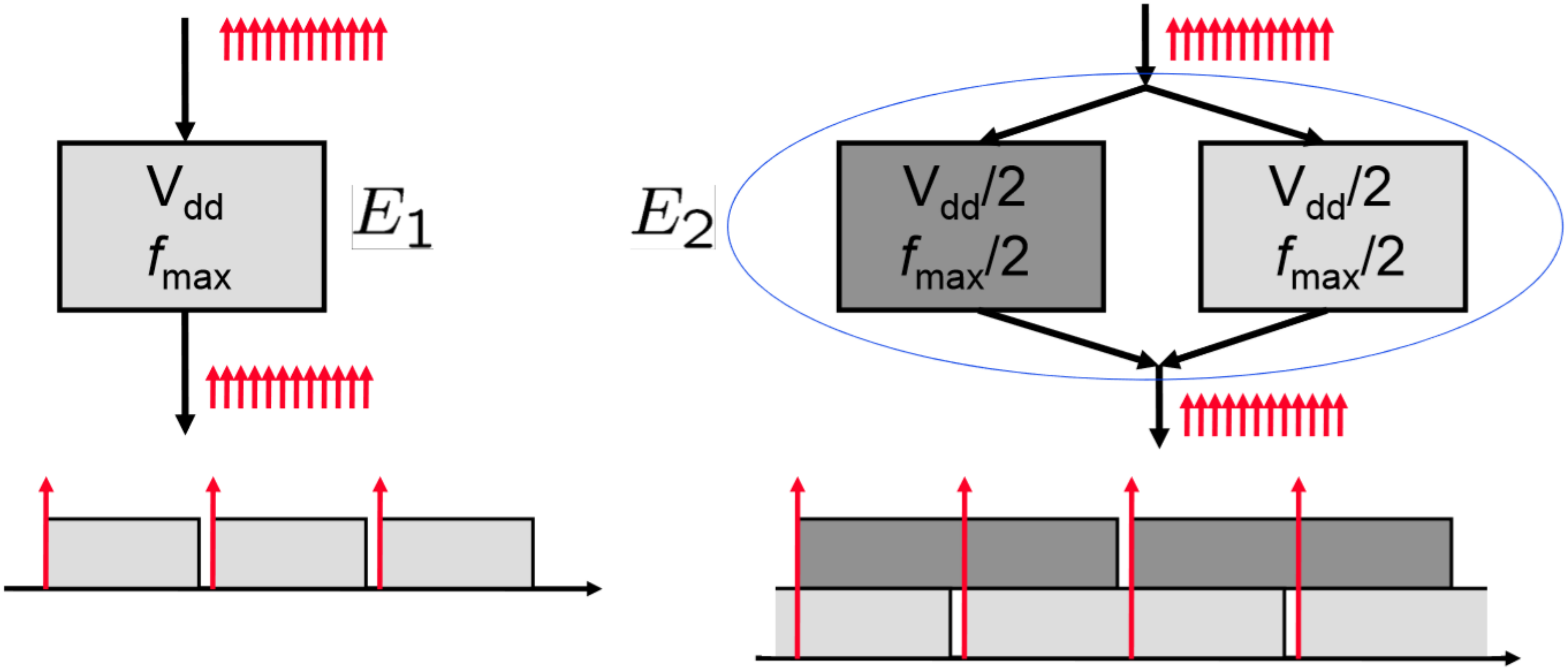
- – Reduce the supply voltage V_{dd}
- – Reduce the number of cycles *#cycles*

Example: Voltage Scaling



[Courtesy, Yasuura, 2000]

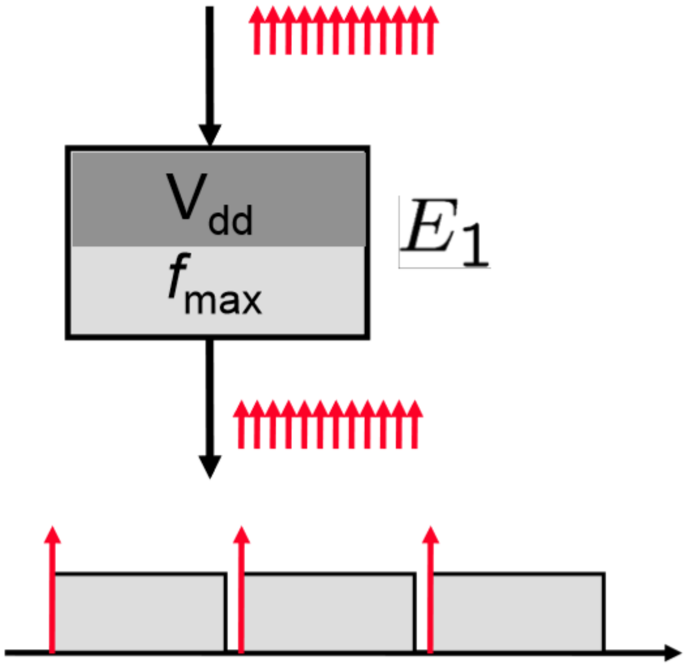
Use of Parallelism

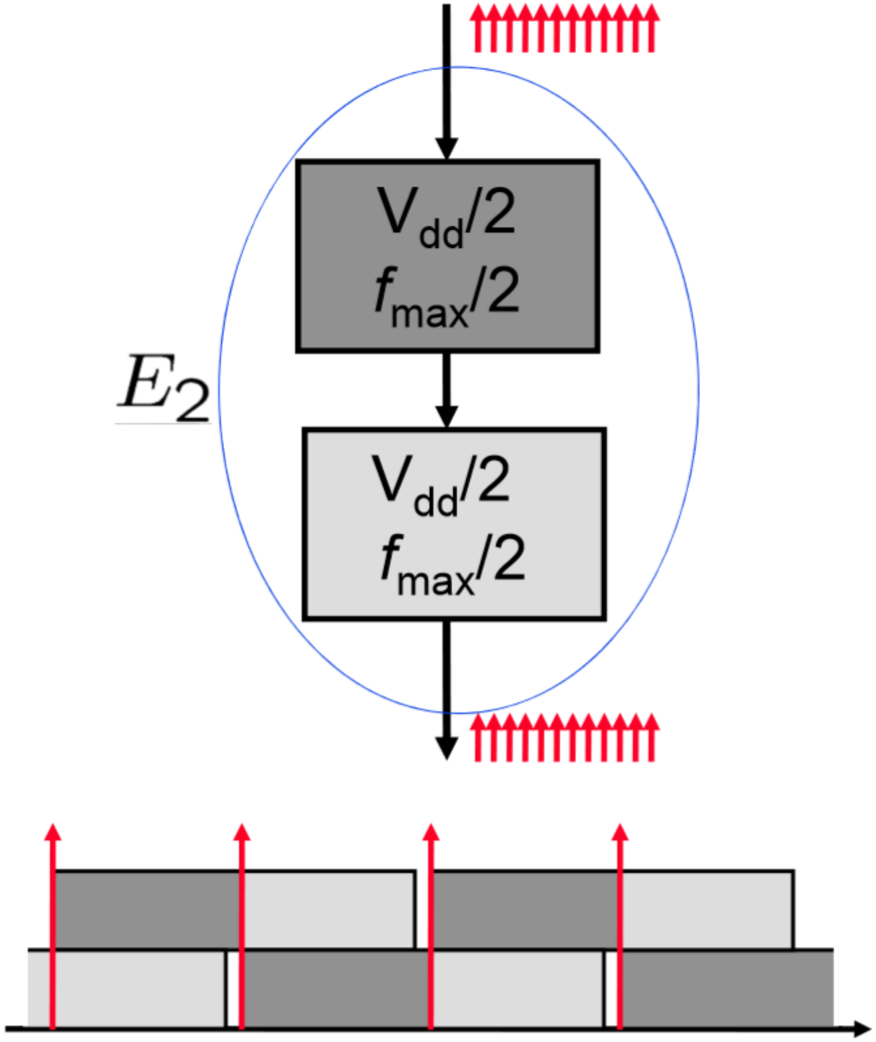


$$E \sim V_{dd}^2\ (\#\text{cycles})$$

$$E_2 = \frac{1}{4}E_1$$

Use of Pipelining



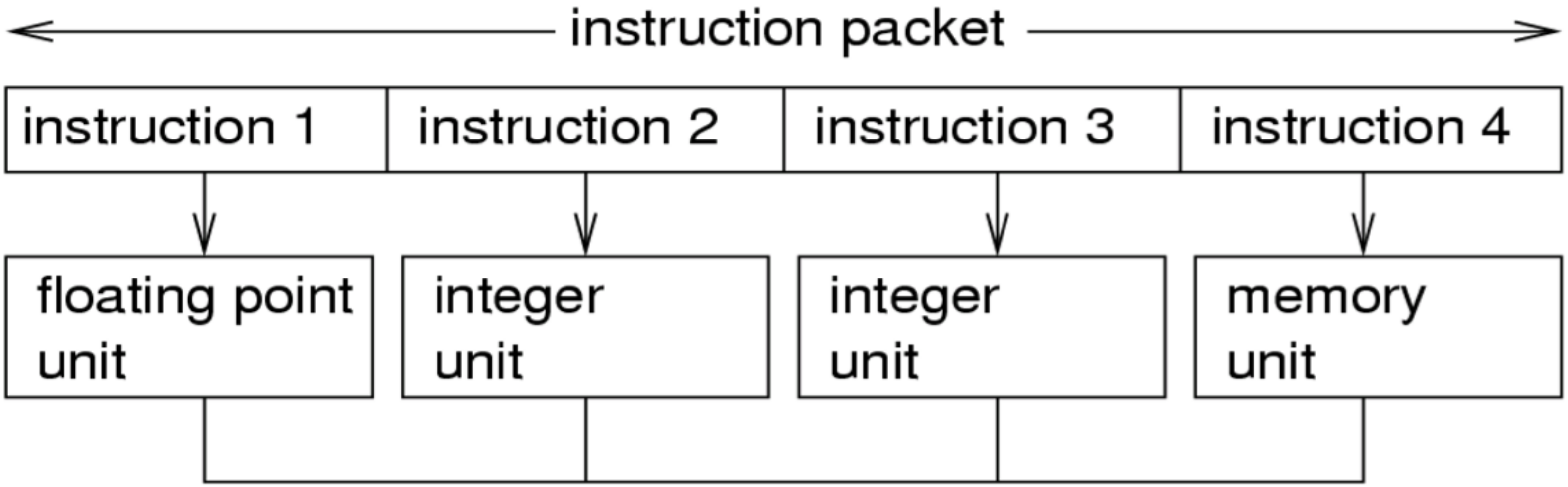


$$E \sim V_{dd}^2 \, (\#\text{cycles})$$

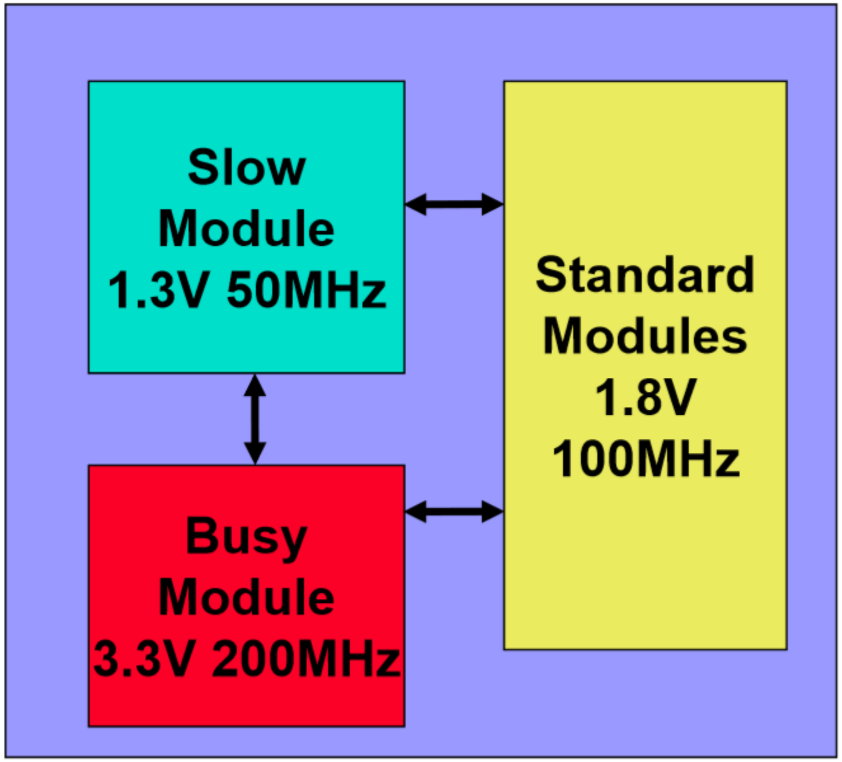
$$E_2 = \frac{1}{4} E_1$$

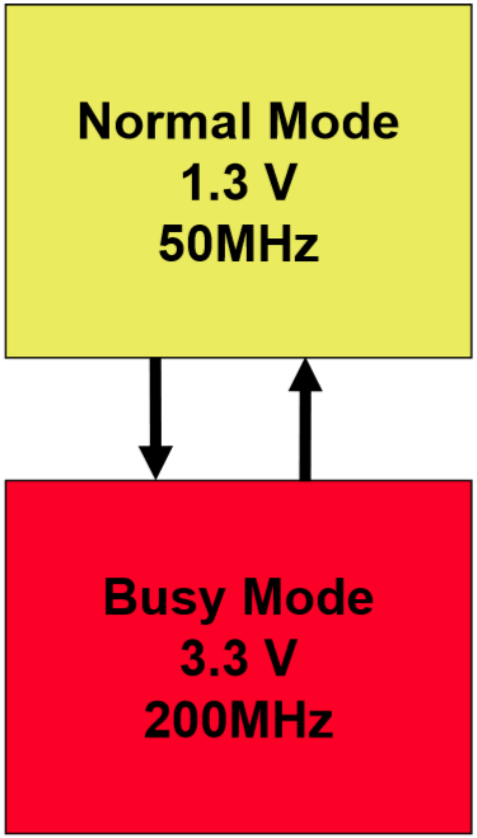
VLIW Architectures

- ***Large degree of parallelism***
  - many computational units, (deeply) pipelined
- ***Simple hardware architecture***
  - explicit parallelism (parallel instruction set)
  - parallelization is done offline (compiler)



Spatial vs. Dynamic Voltage Management





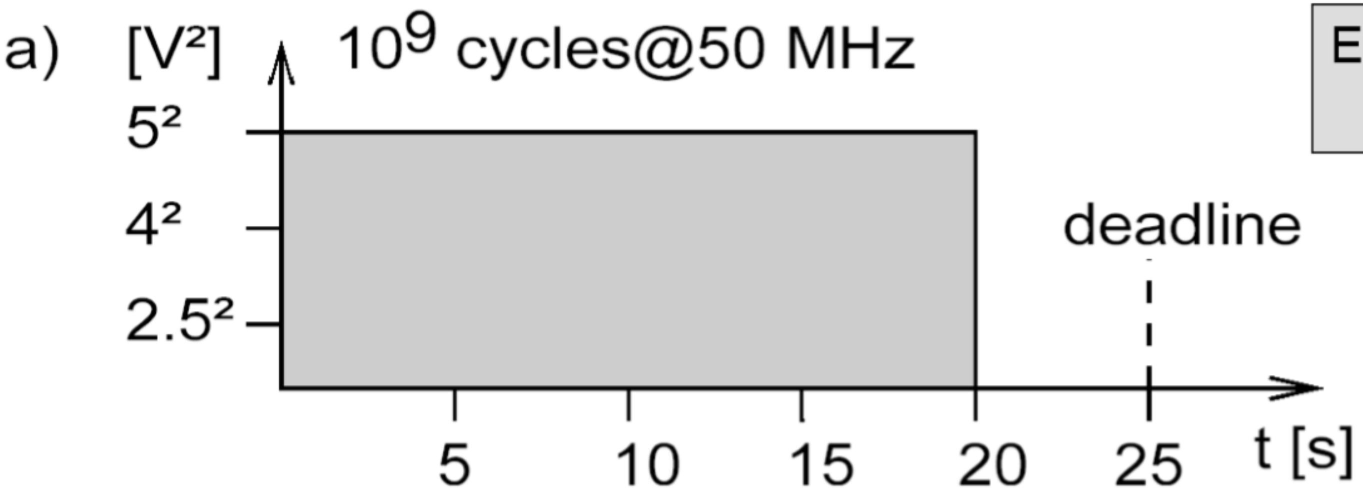
Not all components require same performance.

Required performance may change over time

DVS Example: a) Complete task ASAP

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

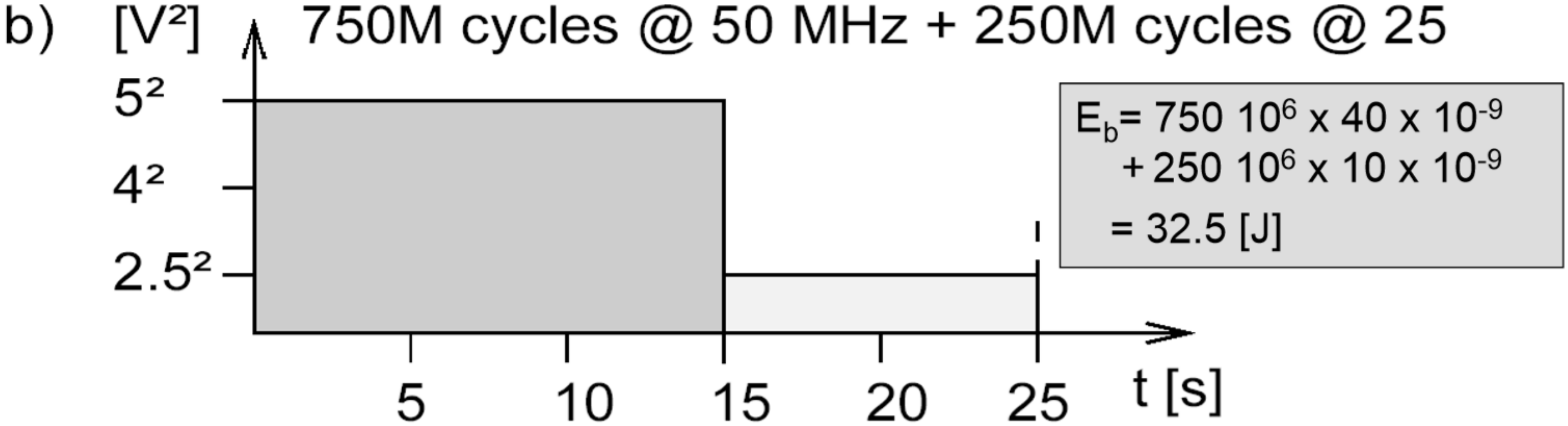
Task that needs to execute 10^9 cycles within 25 seconds.



$E_a = 10^9 \times 40 \times 10^{-9}$
$= 40$ [J]

DVS Example: b) Two voltages

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

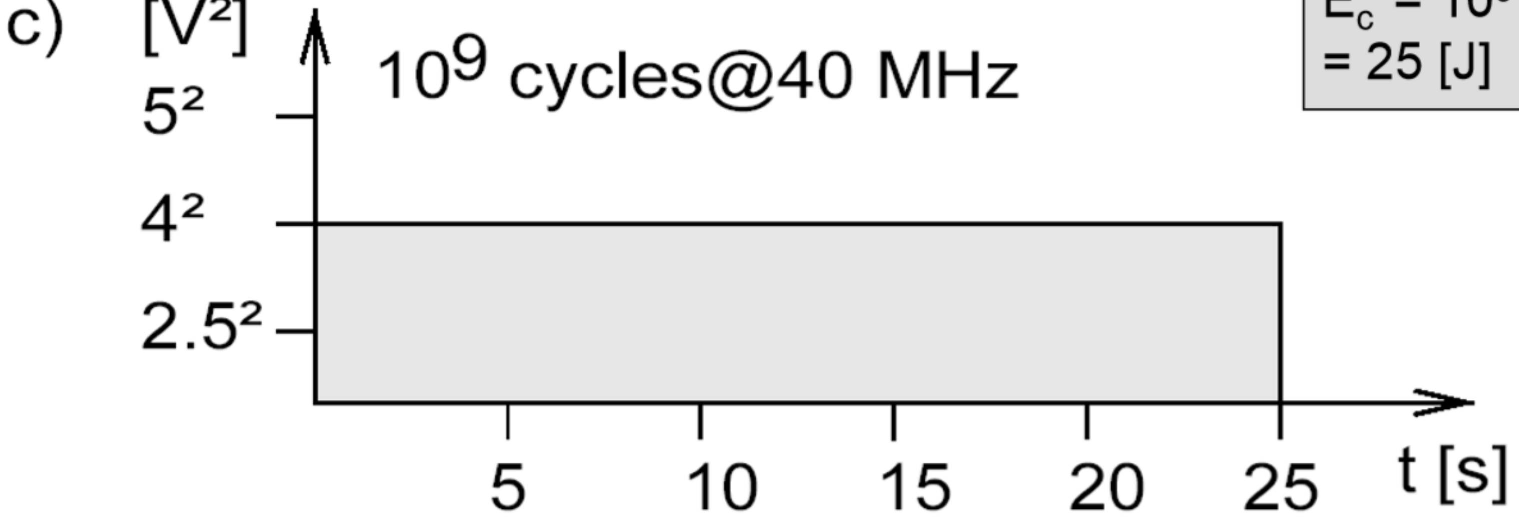


b) 750M cycles @ 50 MHz + 250M cycles @ 25

$E_b = 750\ 10^6 \times 40 \times 10^{-9}$
$+ 250\ 10^6 \times 10 \times 10^{-9}$
$= 32.5$ [J]

DVS Example: c) Optimal Voltage

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40



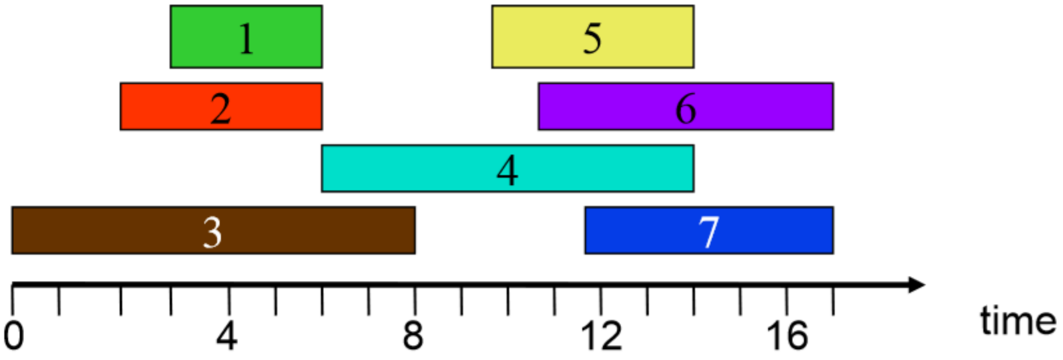
$E_c = 10^9 \times 25 \times 10^{-9}$
$= 25$ [J]

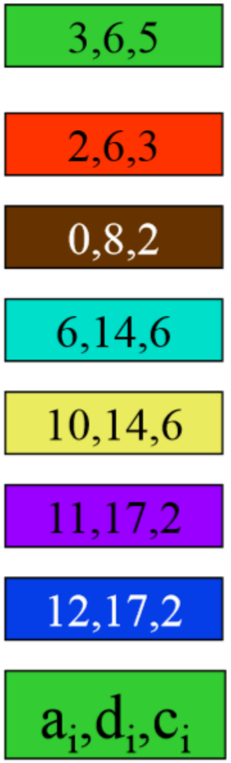
DVS: Offline Scheduling on One Processor

- Let us model a set of independent tasks as follows:
  - We suppose that a task $v_i \in V$
    - requires c_i computation time at normalized processor frequency 1
    - arrives at time a_i
    - has (absolute) deadline constraint d_i
- How do we schedule these tasks such that all these tasks can be finished ***no later than their deadlines*** and the energy consumption is ***minimized***?
  - YDS Algorithm from "A Scheduling Model for Reduce CPU Energy", Frances Yao, Alan Demers, and Scott Shenker, FOCS 1995."

If possible, running at a constant frequency (voltage) minimizes the energy consumption for dynamic voltage scaling.

YDS Algorithm for Offline Scheduling





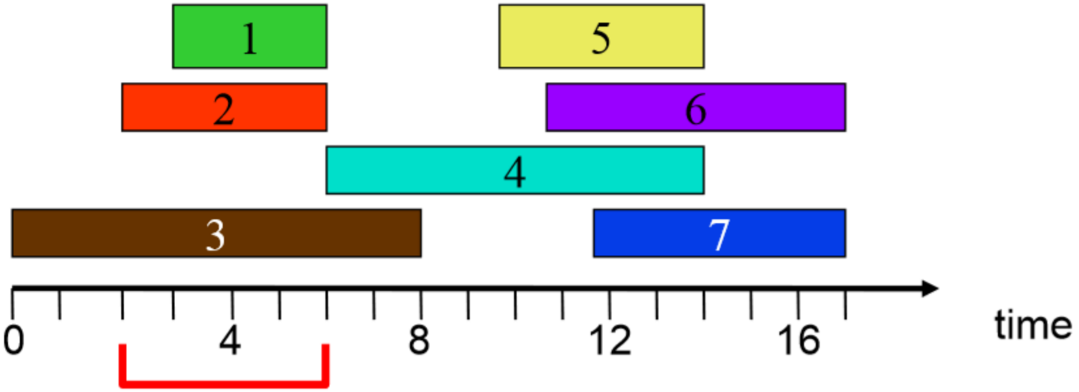
- Define ***intensity*** G([*z*, *z*‘]) in some time interval [*z*, *z*‘]:
  - average accumulated execution time of all tasks that have arrival and deadline in [*z*, *z*‘] relative to the length of the interval *z*‘-*z*

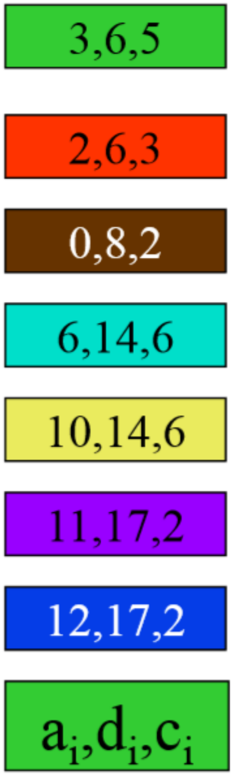
$$V'([z,z']) = \{v_i \in V \ : \ z \leq a_i < d_i \leq z'\}$$

$$G([z,z']) = \sum_{v_i \in V'([z,z'])} c_i/(z'-z)$$

YDS Algorithm for Offline Scheduling

- ***Step 1:*** Execute jobs in the interval with the highest intensity by using the earliest-deadline first schedule and running at the intensity as the frequency.





G([0,6]) = (5+3)/6=8/6, G([0,8]) = (5+3+2)/ (8-0) = 10/8,

G([0,14]) = (5+3+2+6+6)/14=11/7, G([0,17]) = (5+3+2+6+6+2+2)/17=26/17

G([2, 6]) = (5+3)/(6-2)=2, G([2,14]) = (5+3+6+6) / (14-2) = 5/3,

G([2,17]) = (5+3+6+6+2+2)/15=24/15

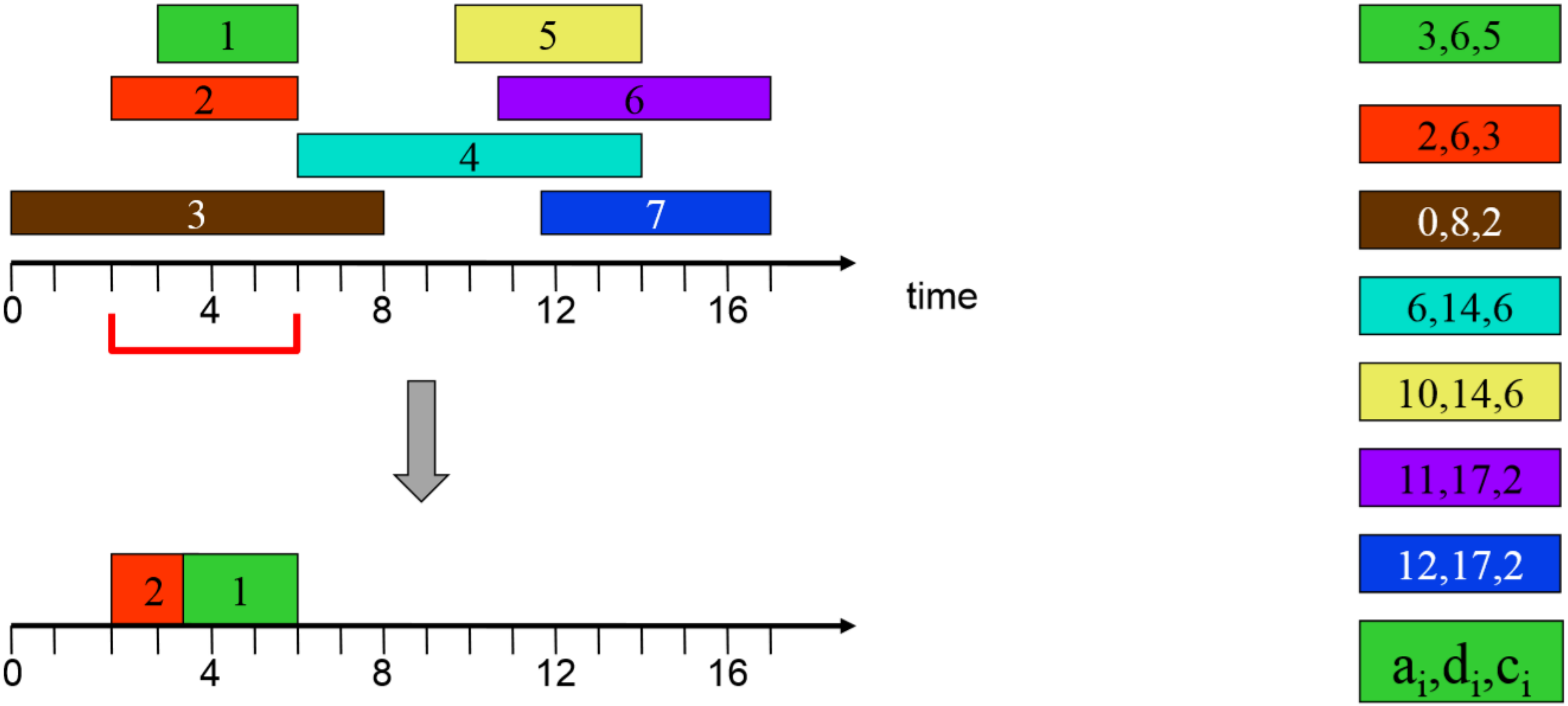
G([3,6]) =5/3, G([3,14]) = (5+6+6)/(14-3) = 17/11, G([3,17])=(5+6+6+2+2)/14=21/14

G([6,14]) = 12/(14-6)=12/8, G([6,17]) = (6+6+2+2)/(17-6)=16/11

G([10,14]) = 6/4, G([10,17]) = 10/7, G([11,17]) = 4/6, G([12,17]) = 2/5

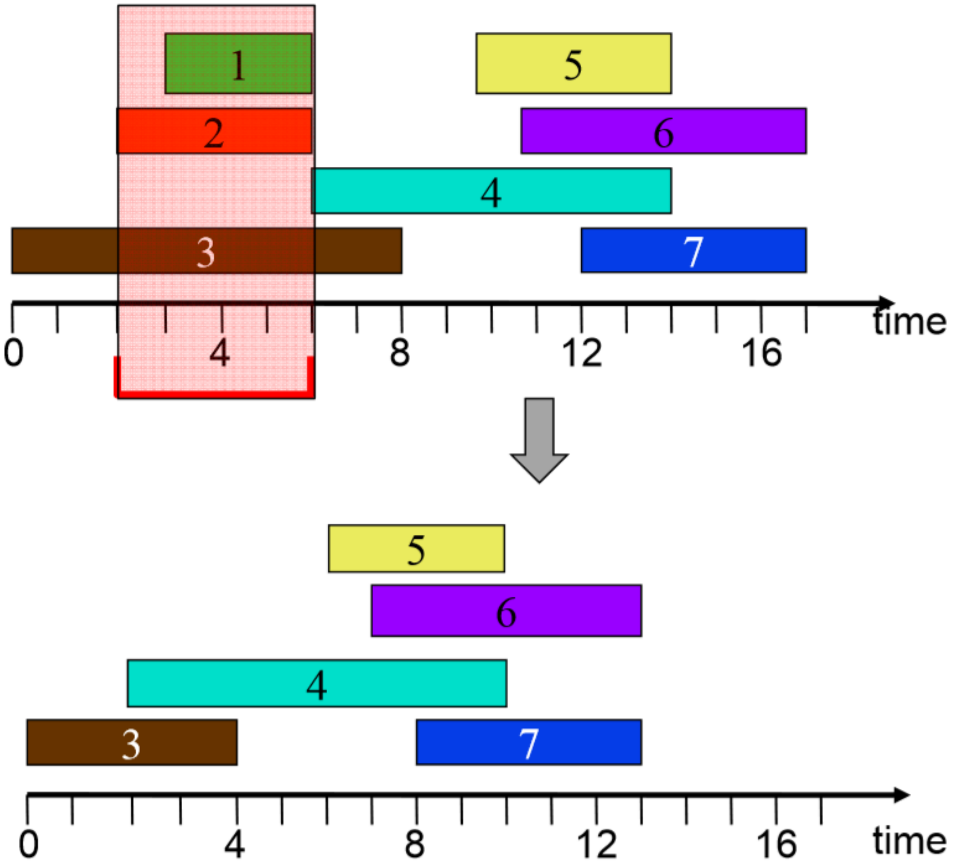
YDS Algorithm for Offline Scheduling

- ***Step 1:*** Execute jobs in the interval with the highest intensity by using the earliest-deadline first schedule and running at the intensity as the frequency.



YDS Algorithm for Offline Scheduling

- ***Step 2:*** Adjust the arrival times and deadlines by excluding the possibility to execute at the previous critical intervals.

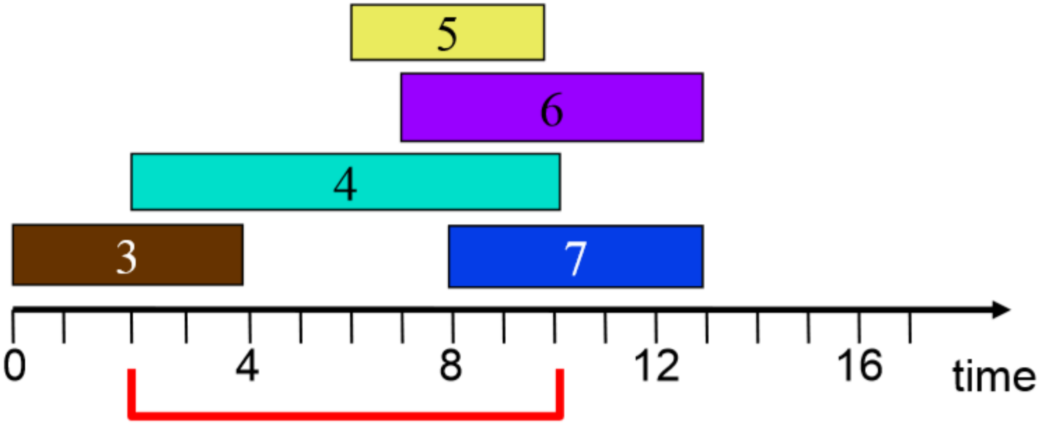


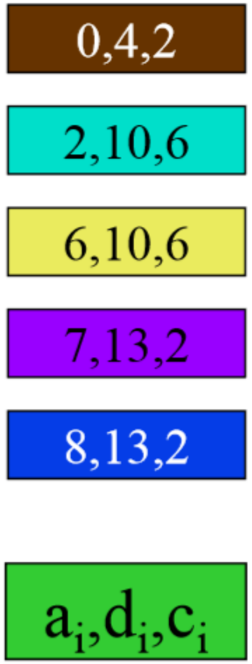
0,8,2	→	0,4,2
6,14,6		2,10,6
10,14,6		6,10,6
11,17,2		7,13,2
12,17,2		8,13,2

a_i,d_i,c_i

YDS Algorithm for Offline Scheduling

- ***Step 3:*** Run the algorithm for the revised input again

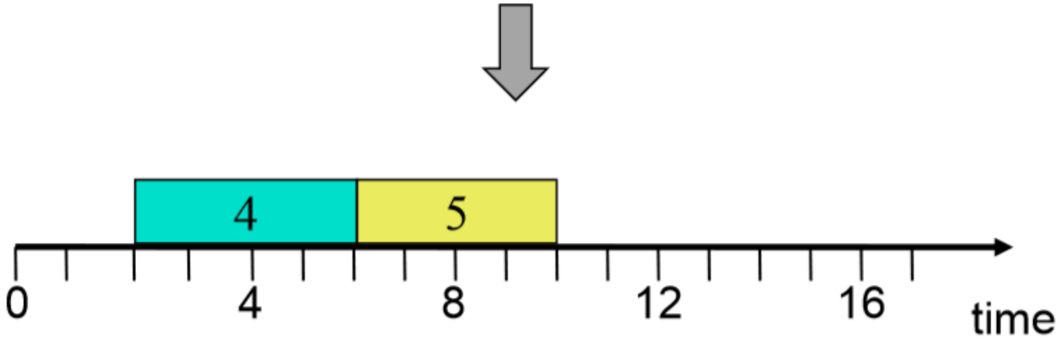




G([0,4])=2/4, G([0,10]) = 14/10, G([0,13])=18/13

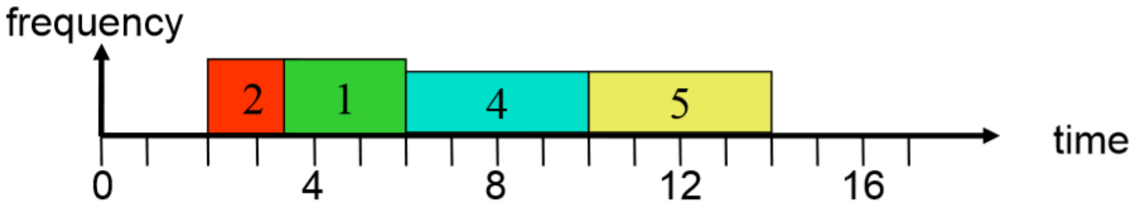
G([2,10])=12/8, G([2,13]) = 16/11, G([6,10])=6/4

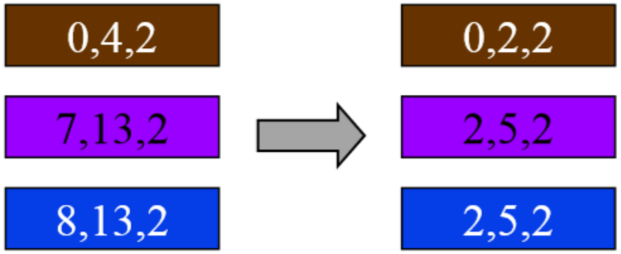
G([6,13])=10/7, G([7,13])=4/6, G([8,13])=4/5



YDS Algorithm for Offline Scheduling

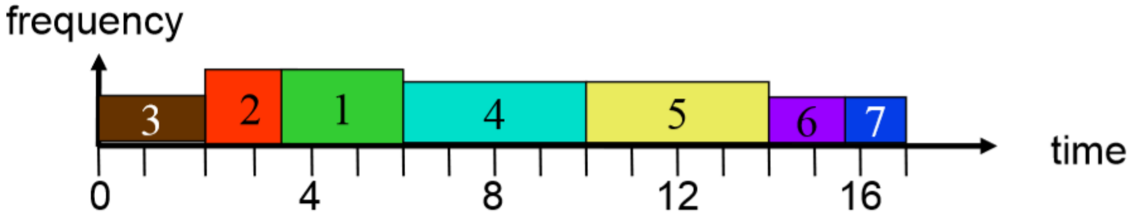
- ***Step 3:*** Run the algorithm for the revised input again
- ***Step 4:*** Put pieces together

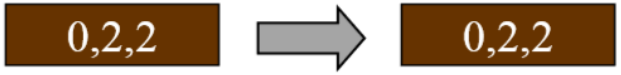



0,4,2 → 0,2,2

7,13,2 → 2,5,2

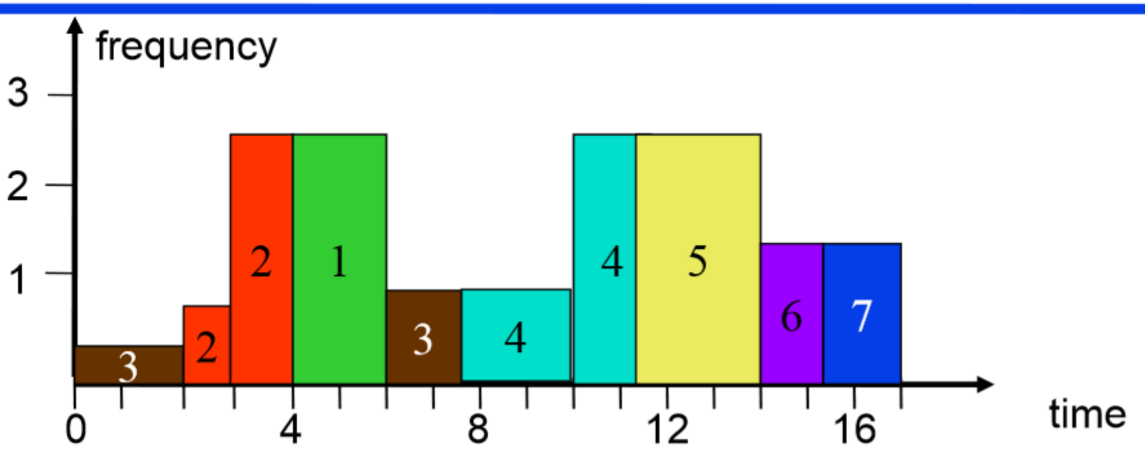
8,13,2 → 2,5,2




0,2,2 → 0,2,2

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
frequency	2	2	1	1.5	1.5	4/3	4/3

DVS: Online Scheduling on One Processor



3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

11,17,2

12,17,2

a_i,d_i,c_i

- **Continuously update to the best schedule for all arrived tasks**
  - Time 0: task v_3 is executed at 2/8
  - Time 2: task v_2 arrives
    - G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4   => execute v_2 at ¾
  - Time 3: task v_1 arrives
    - G([3,6]) = (5+3-3/4)/3=29/12, G([3,8]) < G([3,6]) => execute v_2 and v_1 at 29/12
  - Time 6: task v_4 arrives
    - G([6,8]) = 1.5/2, G([6,14]) = 7.5/8 => execute v_3 and v_4 at 15/16
  - Time 10: task v_5 arrives
    - G([10,14]) = 39/16 => execute v_4 and v_5 at 39/16
  - Time 11 and Time 12
    - The arrival of v_6 and v_7 does not change the critical interval
  - Time 14:
    - G([14,17]) = 4/3 => execute v_6 and v_7 at 4/3

Remarks on YDS Algorithm

- ***Offline***
  - The algorithm guarantees the minimal energy consumption while satisfying the timing constraints
  - The time complexity is O(N^3), where N is the number of tasks in V
    - Finding the critical interval can be done in O(N^2)
    - The number of iterations is at most N
  - Exercise:
    - For periodic real-time tasks with deadline=period, running at ***constant speed with 100% utilization*** under EDF has minimum energy consumption while satisfying the timing constraints.
- ***Online***
  - Compared to the optimal offline solution, the on-line schedule uses at most 27 times of the minimal energy consumption.

Dynamic Power Management (DPM)

Dynamic Power management tries to assign optimal **power saving states**

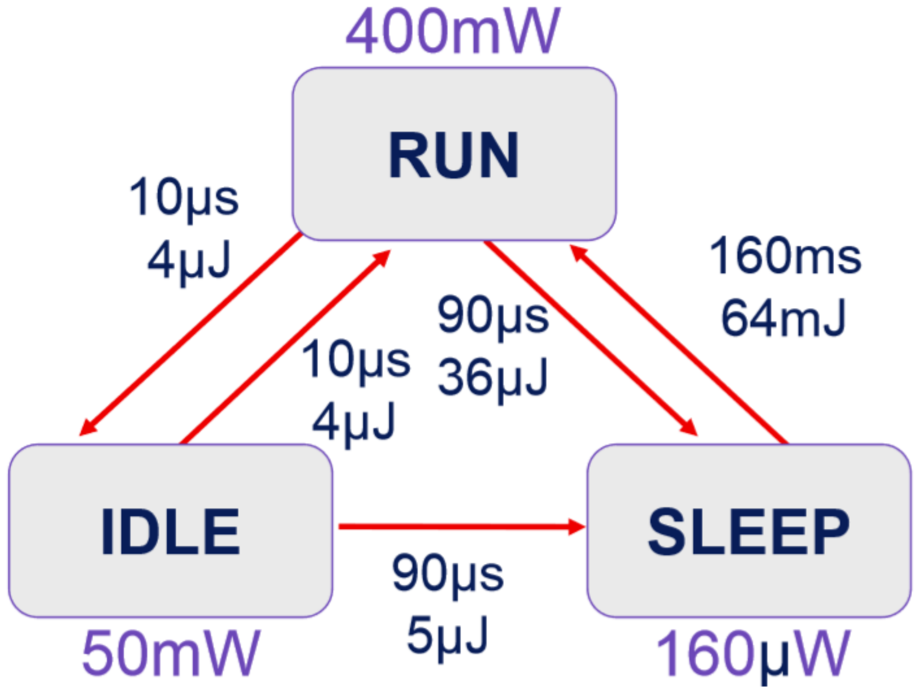
Requires Hardware Support

Example: StrongARM SA1100

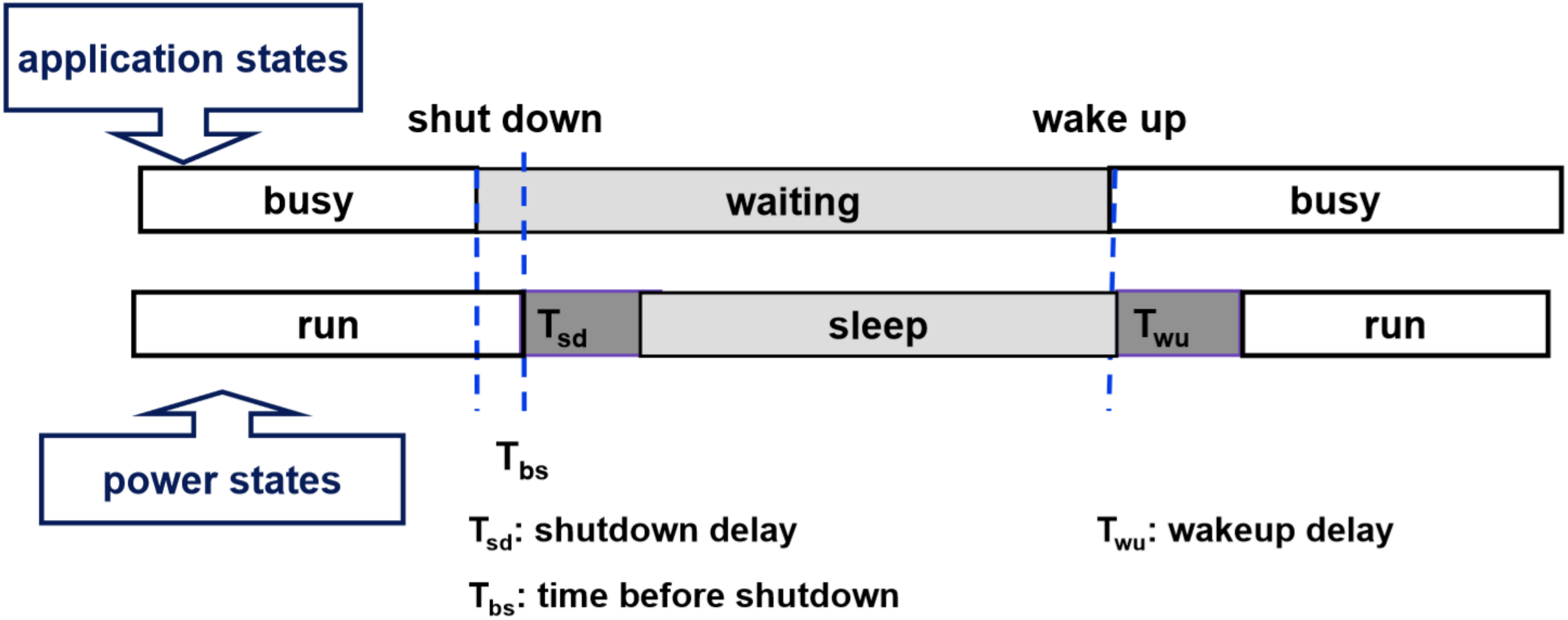
RUN: operational

IDLE: a SW routine may stop the CPU when not in use, while monitoring interrupts

SLEEP: Shutdown of on-chip activity



Reduce Power According to Workload



T_{sd}: shutdown delay

T_{wu}: wakeup delay

T_{bs}: time before shutdown

Desired: Shutdown only during long idle times

→ Tradeoff between savings and overhead

Спасибо за внимание!