Liquid Computing: The Delft Reconfigurable VLIW Processor

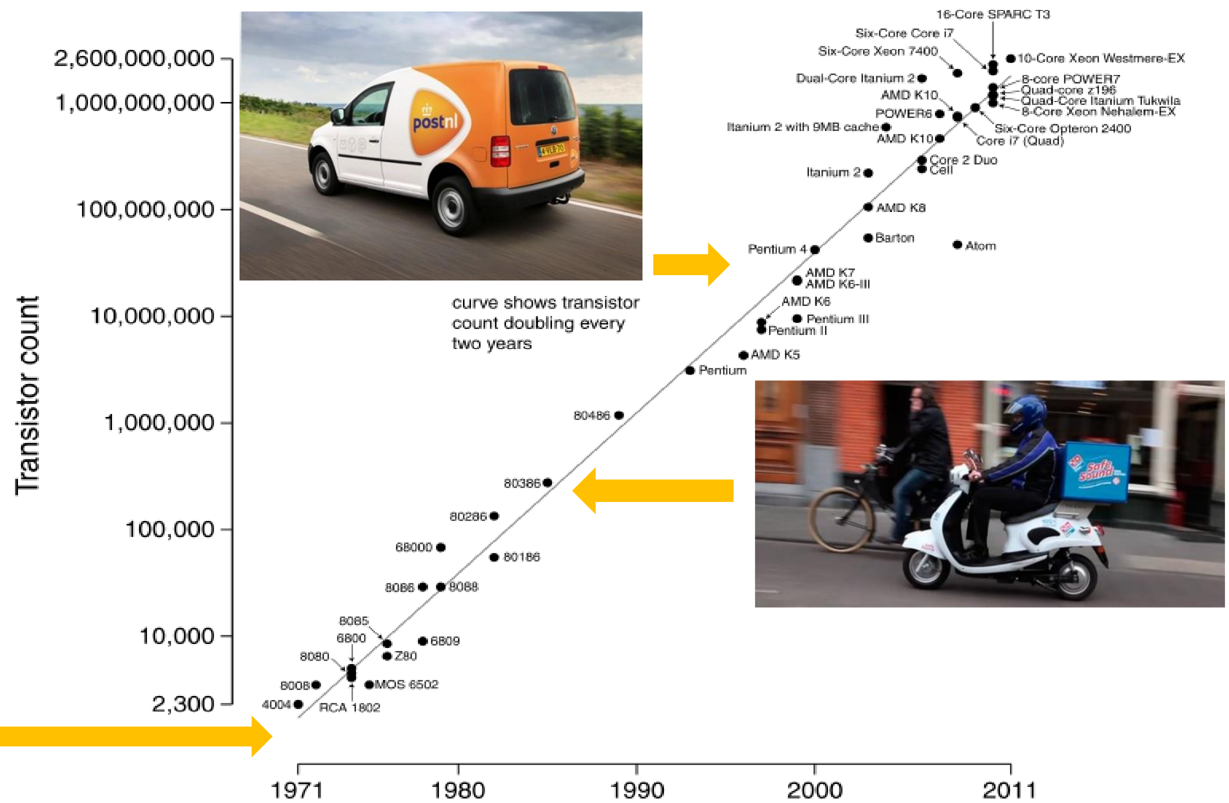
Stephan Wong, Fakhar Anjam, Anthony Brandon,
Joost Hoozemans, Roël Seedorf, and Jeroen van Straten

Computer Engineering Lab
Department of Computer Science and Engineering
Faculty of EEMCS
Delft University of Technology
The Netherlands

Analogy: Package Delivery vs. Processor Design

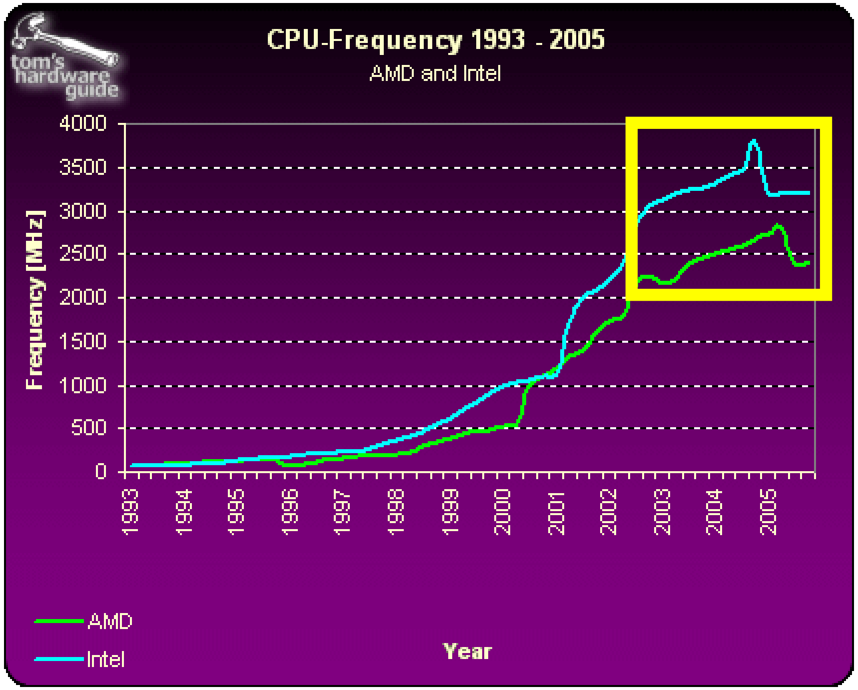
Use the increasing amount of "transistors" to build better package delivery systems:

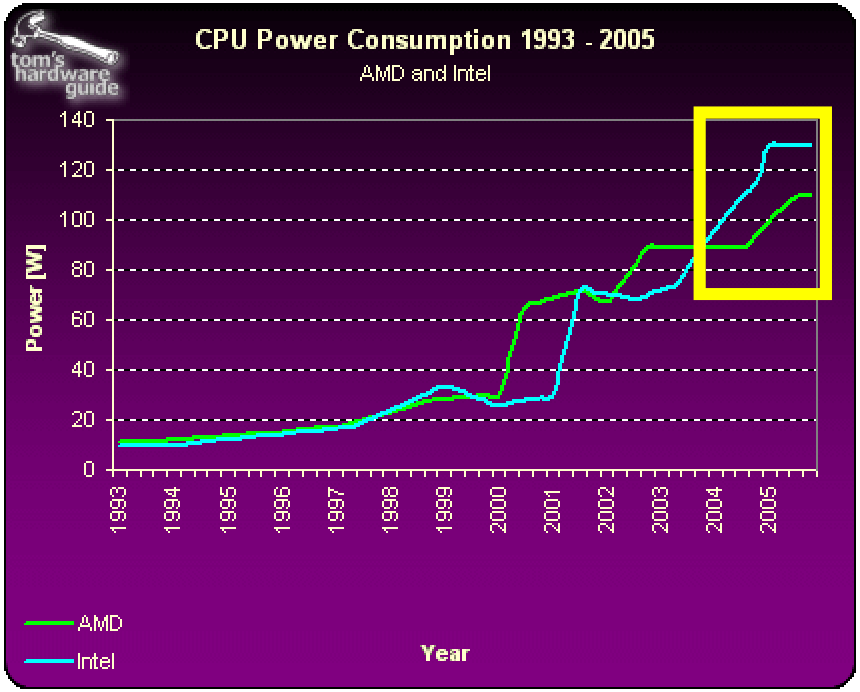
- *Larger packages*
- *Faster delivery*
- *More energy-efficient*



From: "Transistor Count and Moore's Law - 2011" by Wgsimon

Around 2005: Frequency & Power leveling off





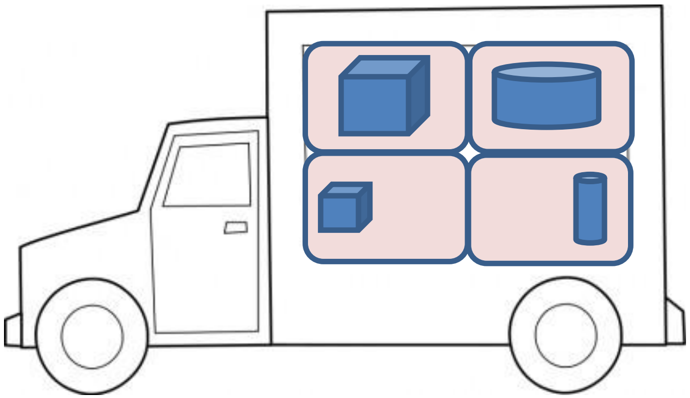
From: www.tomshardware.com

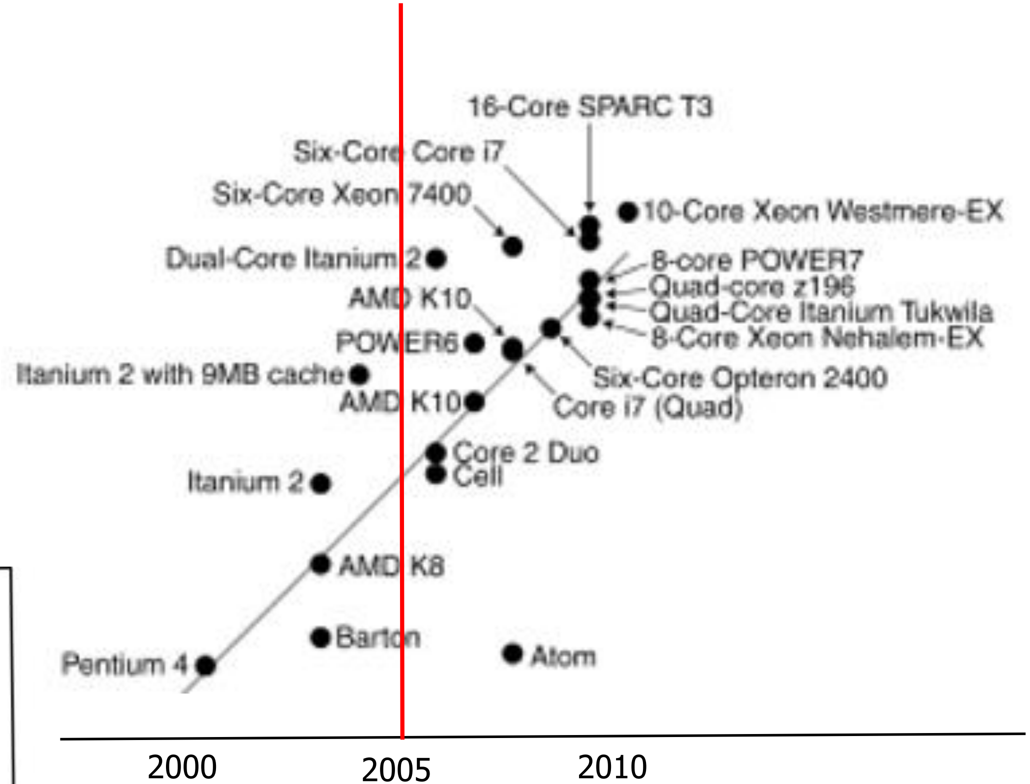
- Dennard Scaling (power density remains constant) ended 2005-2007
- However, Moore's Law (#transistors doubles every ~2 yrs) continued
- What was the effect??

Homogeneous Computing

Terminology (after 2005):

- Dual-core
- Quad-core
- Six-core
- 8-core
- 10-core
- 16-core
- "Just" more the same core

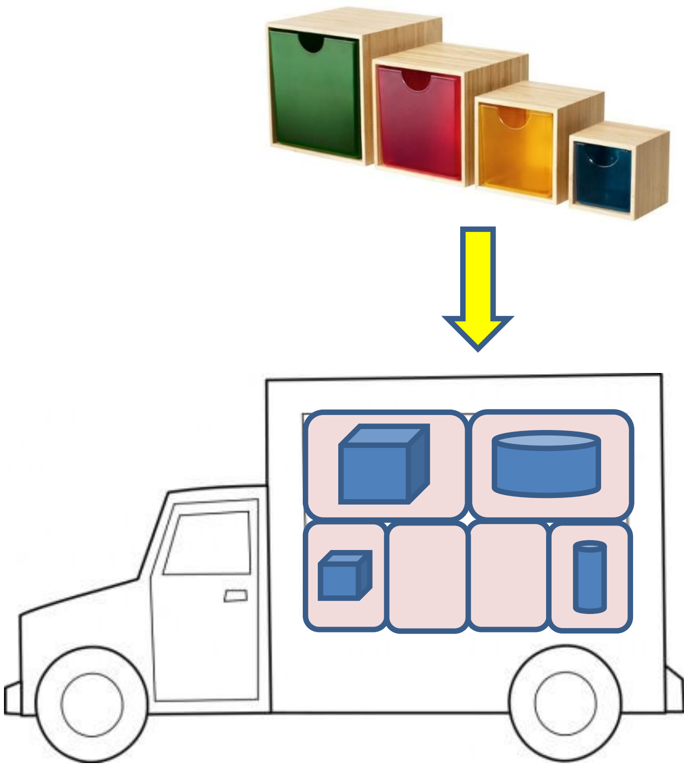


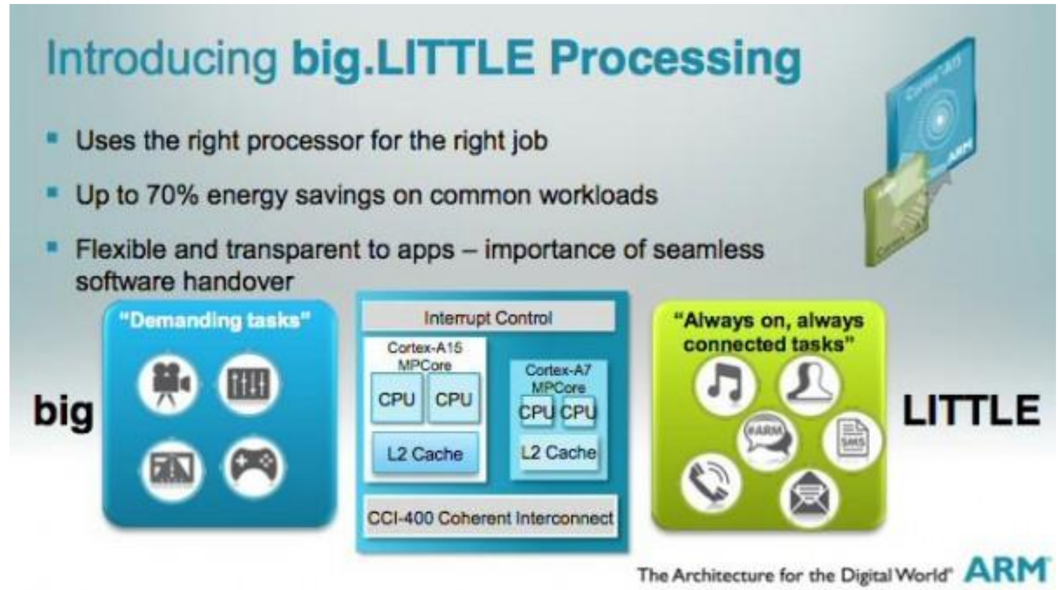


Analogy: Use the <u>same box (=processor)</u> to transport various sizes of packages (=applications)

Heterogeneous Computing

- Example: in 2011 ARM Introduces big.LITTLE





From: www.arm.com

Analogy: Use the different boxes (=heterogeneous processors) to transport various sizes of packages (=applications)

TU Delft

Now What?

- How can we even more efficiently use the transistors, minimize waste, and reduce energy consumption?

→ Liquid Computing

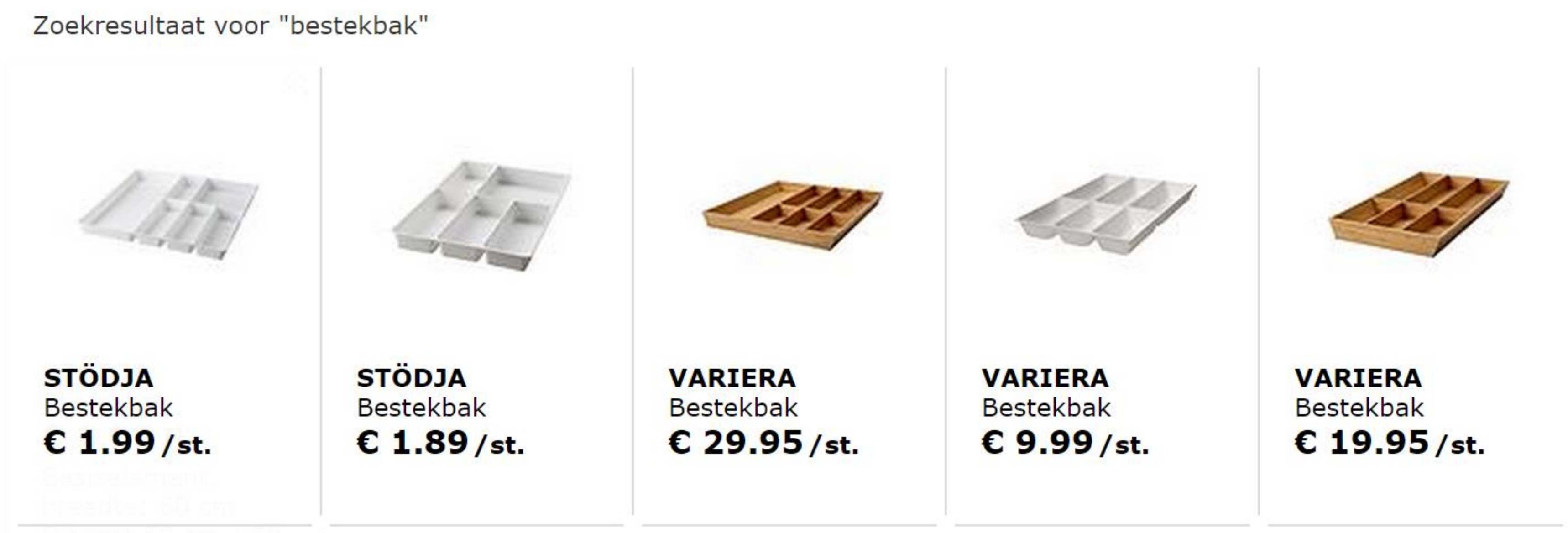
- First two intermezzo's before I give definition

Intermezzo: 3D Printing

From: http://www.makerbot.com/uses/for-professionals

- Continuing analogy → build the *best container* for each package
- This means: build the *best processor* for your application
- → Field-Programmable Gate Arrays (FPGAs) is now best candidate

Intermezzo: IKEA



From: www.ikea.com

- Continuing analogy: reconfigure a kitchen drawer (=reconfigurable processor) for different kitchen utensils (=applications)
- → Parameterized reconfigurable processors
- *(NOTE: One single design and not necessarily using FPGAs)*

Liquid Computing: A definition

- Run-time adaptivity of computing systems (processors, memories, network-on-chips) to meet changing requirements of applications being executed in different environments
- Analogy: Versatile and flexible package delivery system that can cope with any type and size of packages to be transported in all (weather) conditions at any time

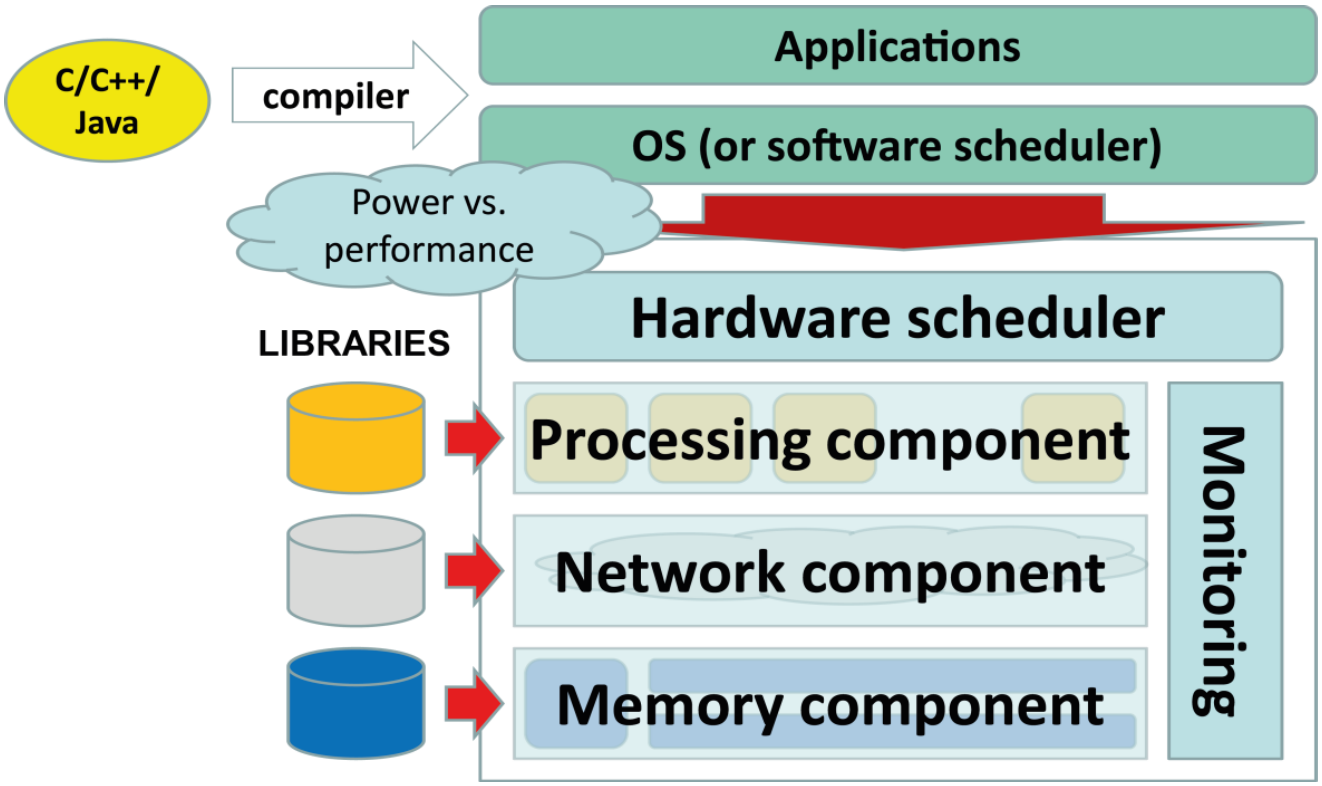
- The predecessor to LC was the ERA project

Embedded Reconfigurable Architectures

Dynamic adaptation to *software requirements & operating environment*

Dynamic adaptation in *performance, power / energy, and resources*

Dynamic reconfiguration of *processor cores, caches, and NoCs*



Partners

Technische Universiteit Delft
(TUD) – NL (Coordinator)

Industrial Systems Institute
(ISI) – GR

Universita' degli Studi di Siena
(UNISI) – IT

Chalmers University
(CHALMERS) – SE

University of Edinburgh
(UEDIN) – UK

Evidence
(EVI) – IT

STMICRO
(STMICRO) – IT

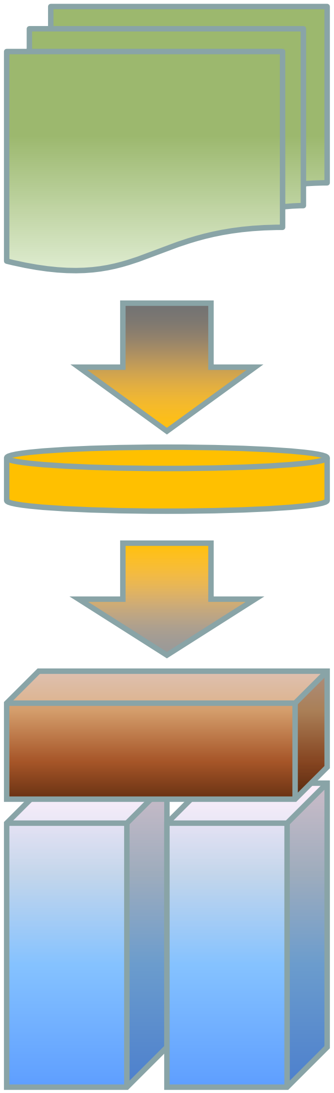
IBM
(IBM) – IL

Universidade do Rio Grande do Sul
(UFRGS) – BR

Uppsala University
(UU) – SE

Contract: **INFSO-ICT-249059**
EU Funding: **2.8 MEuro**
Start - End: **01-2010 – 03-2013**

Mainstream Processors



Programs:

- General-purpose programs (think: desktop, office)
- Domain-specific programs (think: embedded)
- Different characteristics (e.g., parallelism)

Will programs run efficiently on the processor in most cases?

- **for general-purpose computing: YES**
- **for domain-specific computing: NO**

Why not?

- fixed nature of processor - not tuned for applications

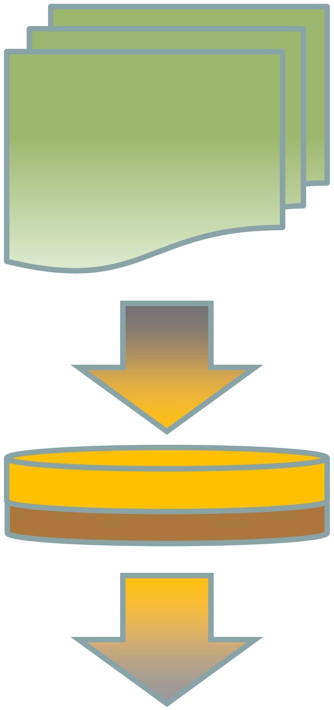
Compiler:

- Targets fixed processor
- Match characteristics with processor capabilities

Single processor:

- General-purpose
- Fixed (parallel) functionality
- Complex hardware to fully utilize parallel hardware (= power hungry)

Embedded Reconfigurable Architectures



Programs:

- General-purpose programs (think: desktop, office)
- Domain-specific programs (think: embedded)
- Different characteristics (e.g., parallelism)

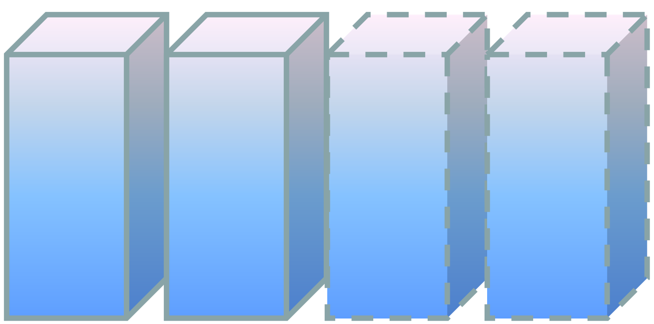
What do we do in the ERA project?

- Parameterization of processor designs
- <u>Match processor designs to the applications (through parameters)</u>
- Perform switching of processor cores dynamically (at run-time)
- Self-optimize based on available resources and power budget

Compiler:

- Targets reconfigurable & parameterized processor
- Match characteristics with processor capabilities

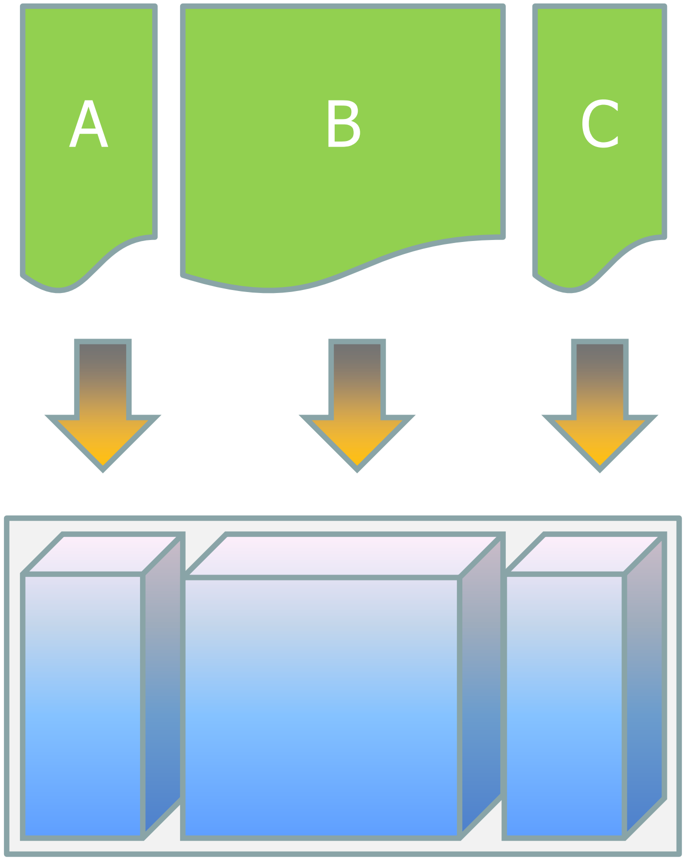
*: moved the complex instruction scheduling to the compiler (= VLIW processor concept)



Many datapaths:

- Can be combined to form different processors
- Reconfigurable & parameterized processor(s)
- Adaptive functionality
- Adaptive behavior based on resources, power budget, and target performance (self-optimization)

An Example (1/3)



Program A wants to run on the ERA platform

Instantiate a core capable of running program A

Run program A on the new core

Program B wants to run on the ERA platform

Instantiate a core capable of running program B

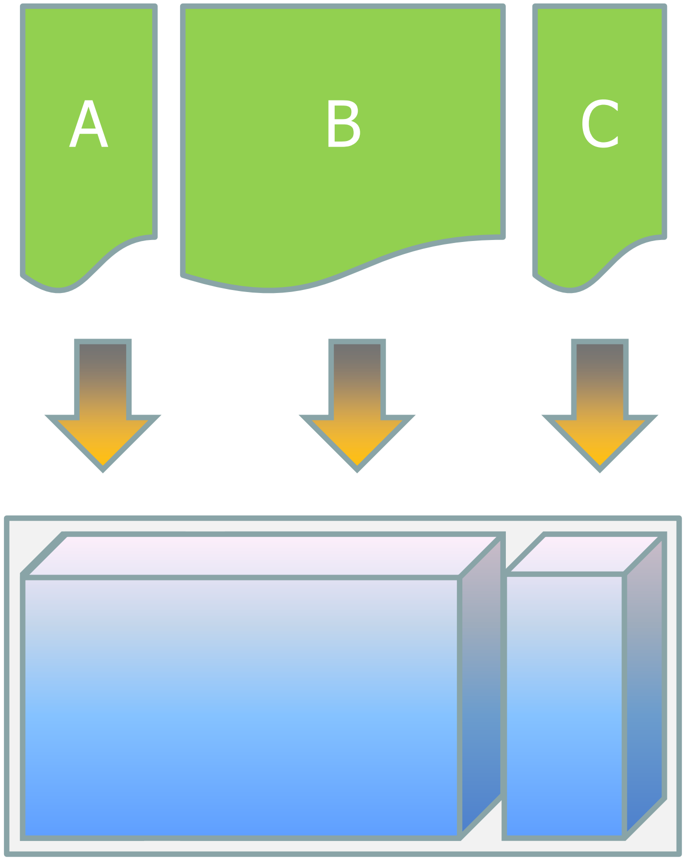
Run program B on the new core

Program C wants to run on the ERA platform

Instantiate a core capable of running program C

Run program C on the new core

An Example (2/3)



Program A finishes

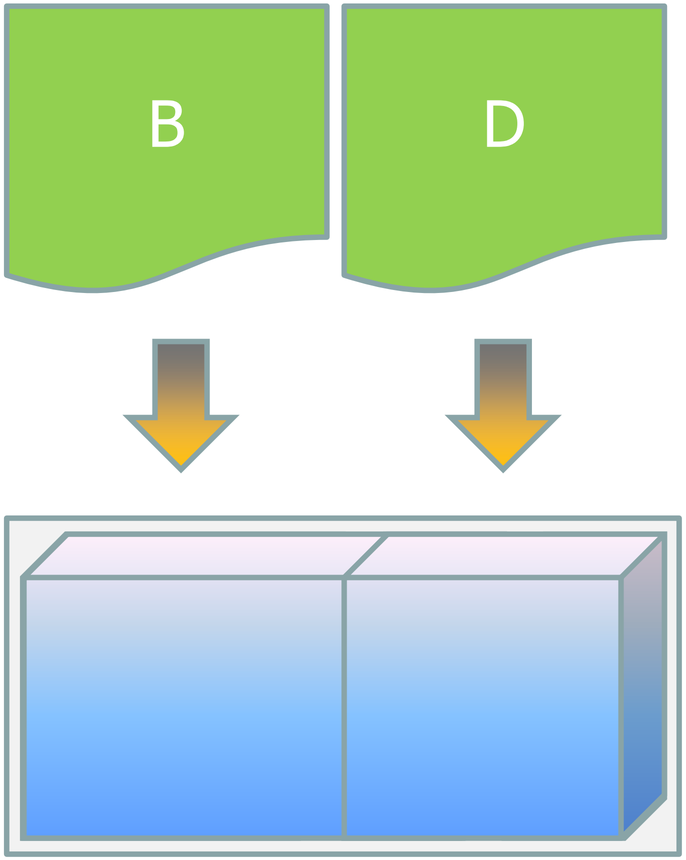
The related core is gated off to save power

Program B utilizes more resources to improve performance

Program C finishes

The related core is gated off to save power

An Example (3/3)



Program D wants to run on the ERA platform

Program D's preferred core size is not available

Instantiate a non-preferred core on the remaining resources and execute program D on that core; OR

Reduce the core size executing program B

Instantiate a core capable of running program D

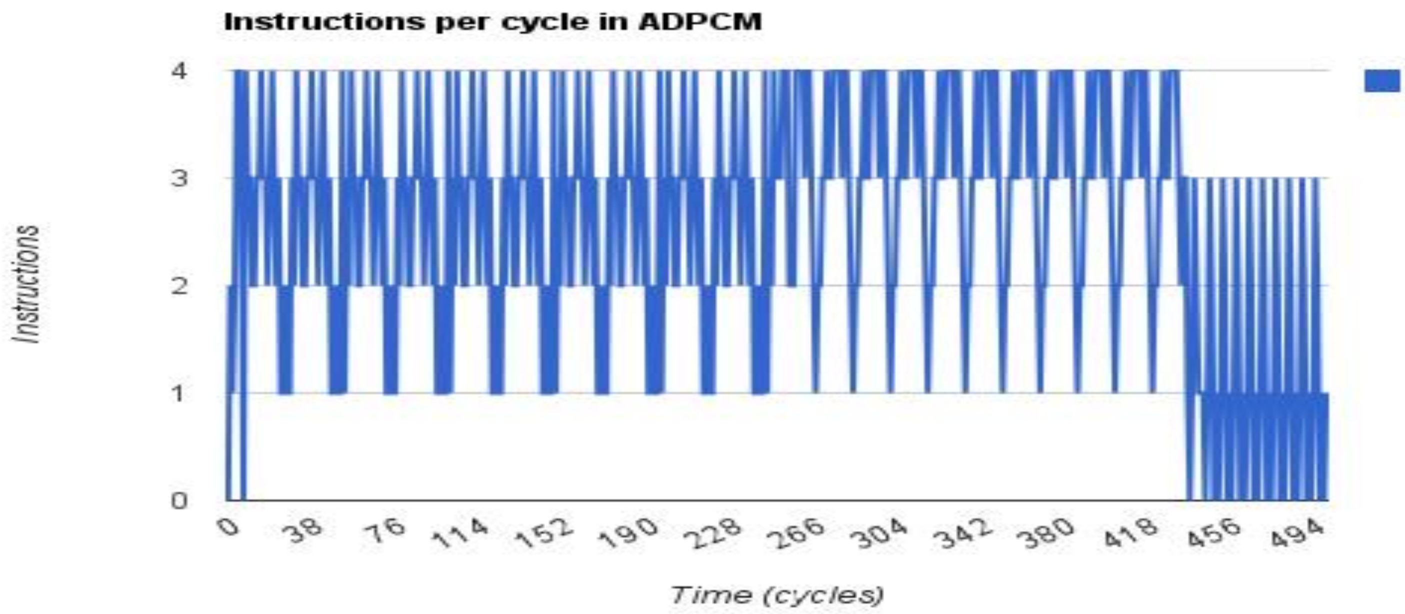
Run program D on the new core

How about the network-on-chip (NoC) and memory hierarchy?

We apply the same concepts illustrated by the processor example to the NoC and memories!!

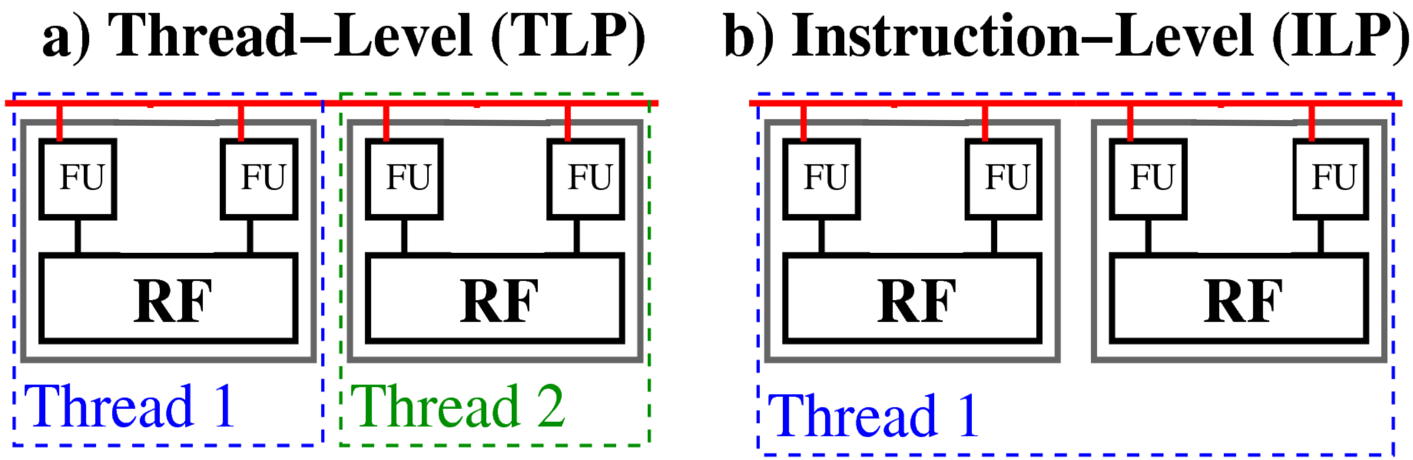
From multiple programs to a single program

- Applications have different phases in which different processor organizations are more suited → Dynamically adapt the hardware to suit these phases
- We already have the tools and hardware support (e.g., “generic” binary, interrupt, reconfigurable issue width cores) to make this a reality
- Now: a real-life example



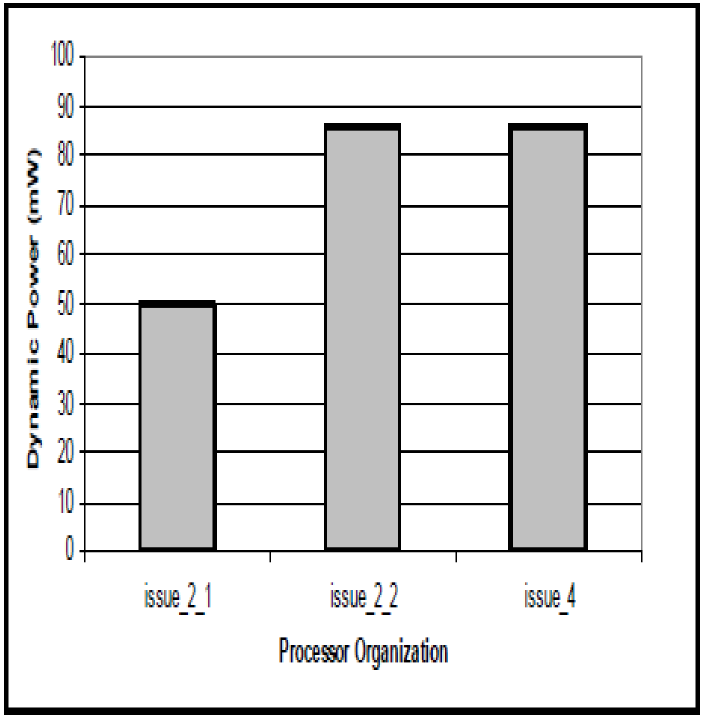
TLP vs. ILP

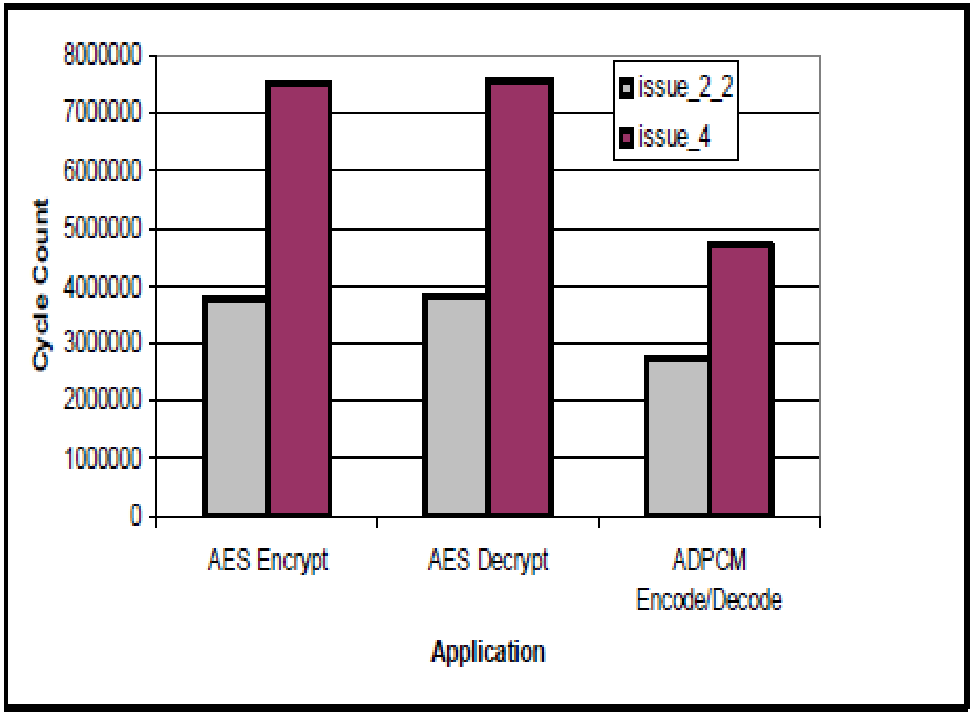
- Leverage reconfigurable multi-core to adapt resources from TLP to ILP or fault-tolerance
- Key ideas:
  - Add direct pair-wise fine-grain communication support to interconnect and ISA
  - Compiler manages ILP through advanced clustering techniques



Energy/Performance Trade-offs

- Higher ILP applications favor wider issue cores
- Higher TLP favors "more & smaller" cores

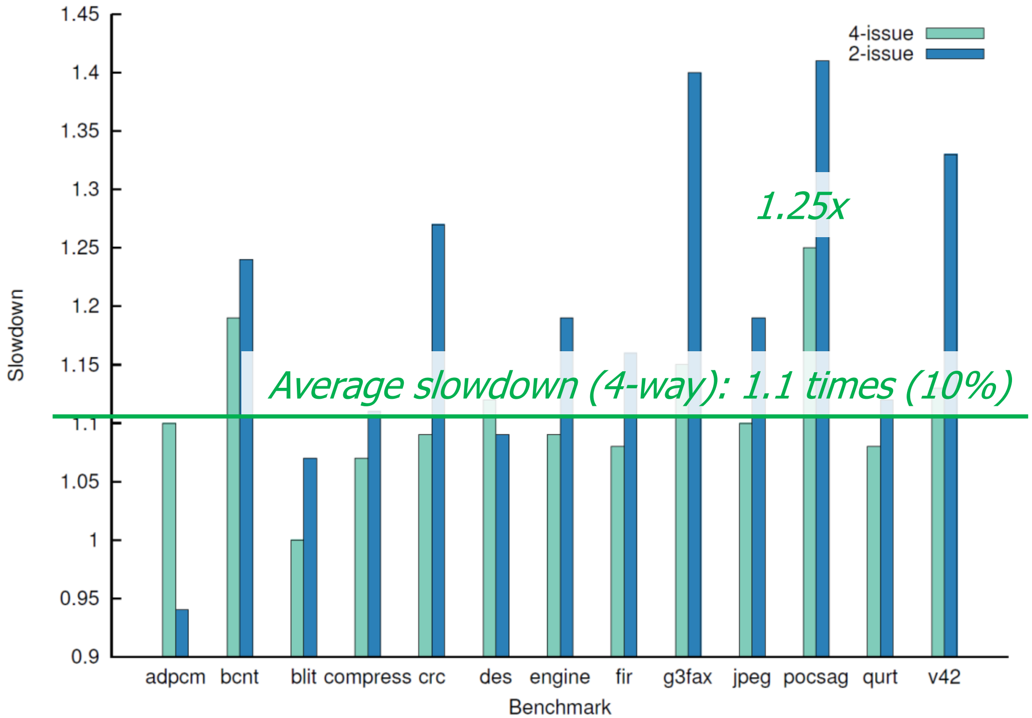




Recent Past Developments for ρ-VEX

Binary compatibility for dynamic issue-width adaptivity:

- Definition of the "generic binary" [*presented at DATE 2013*]
- Approach:
  - Compile for 8-issue and address them as 2-issue bundles
  - Fix false dependencies, skip NOPs-only bundles
  - Simple hardware change in "update PC" & "skip NOPs"
- Advantages (over code versioning):
  - Interruptability
  - Dynamic switching of issue width (controlled by application designer, compiler, hardware scheduler, and/or OS scheduler)
- Disadvantage → performance loss (measured: avg. 30%, projected: 10%)
- NEW RESULTS: avg. 5%



Past Developments for ρ-VEX

ρVEX V1.0 [*presented at FPT 2008*]

Dynamically Reconfigurable Register File for ρ-VEX [*presented at DATE 2010*]

Multi-ported register file design using BRAMs [*presented at FPT 2010*]

ρVEX V2.0 & extensions: [*presented at WRC 2012 (2 papers)*]

- Paper 1: Pipelined, forwarding logic, Paper 2: Support for traps (interrupts, exceptions)

Run-time task migration [*presented at ARC 2012*]

Dynamic issue-width reconfiguration: [*presented at FPT 2010, DATE 2011*]

- Dynamic adaptation of issue slots

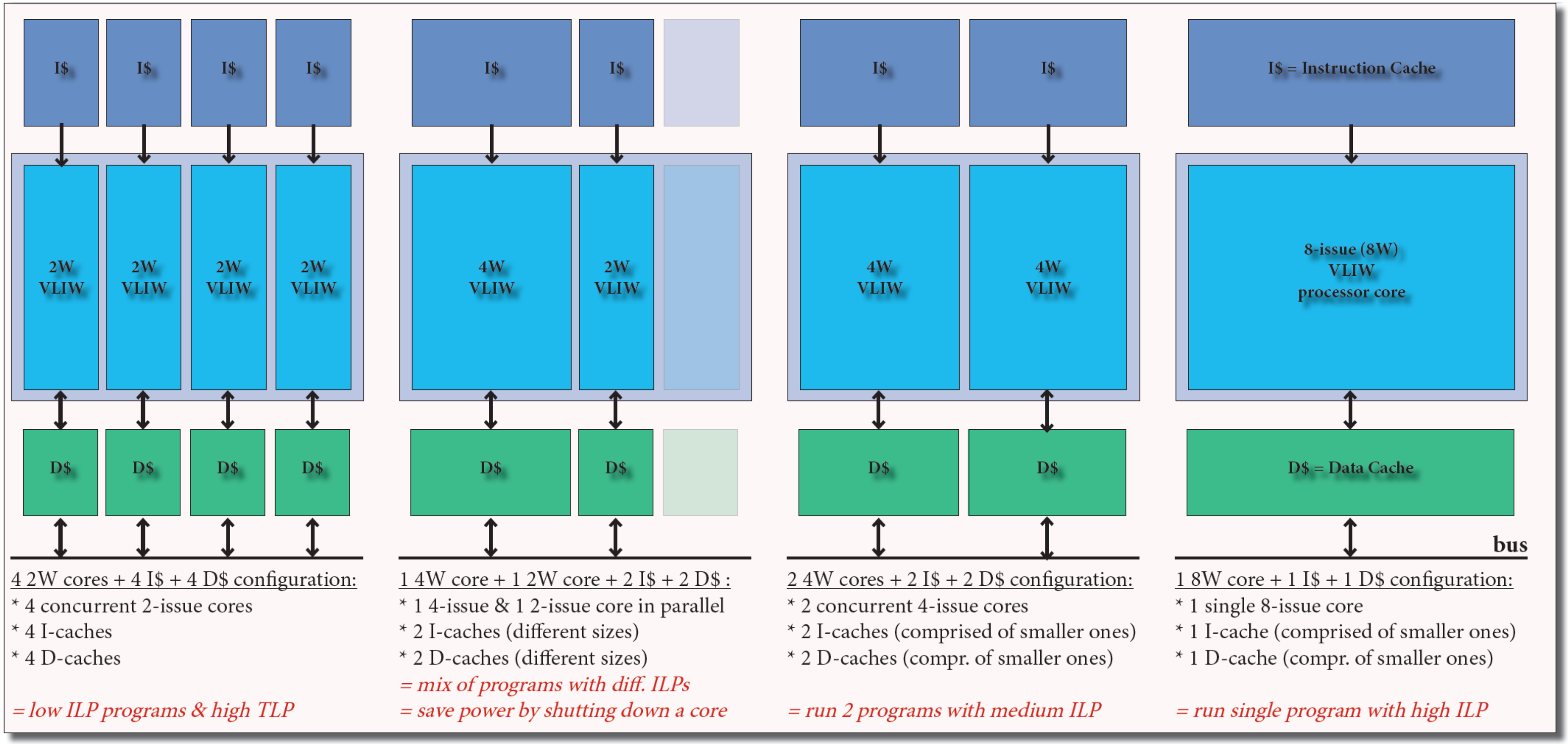
Dynamic issue-width and 1st level I-cache reconfiguration: [*pres. at SAMOS 2012*]

- Simultaneous reconfiguration of core issue width and I-cache parameters

Binary compatibility for dynamic issue-width adaptivity [*presented at DATE 2013*]

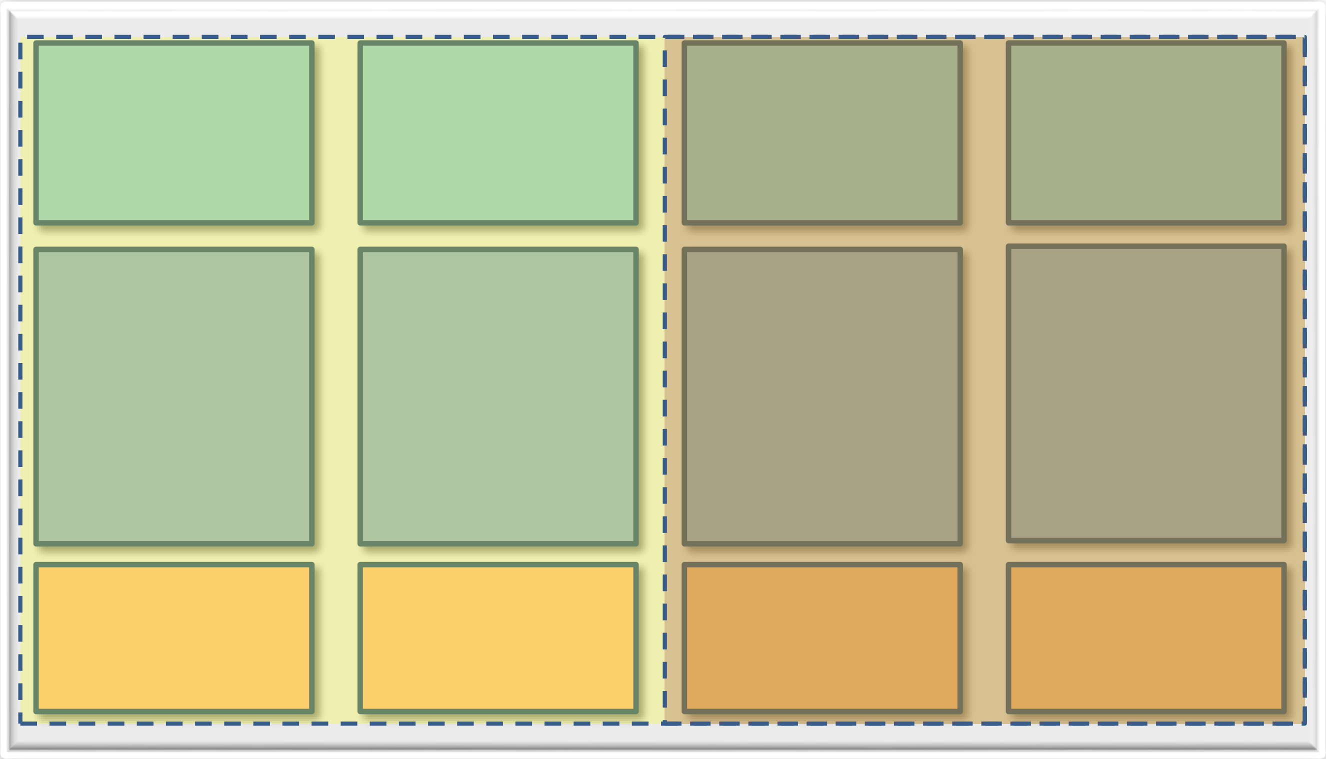
Dynamic support for fault-tolerance [*presented at ARC 2013*]

Redesigned Dynamic ρ-VEX core (2015)



- Dynamic reconfiguration (5 cycles), multiple context support, cache resizing, snooping cache
- Precise interrupts, configuration via memory-mapped registers, dynamic trace unit, support for breakpoints and single stepping through application codes, gdb support, Linux (2.0) running.

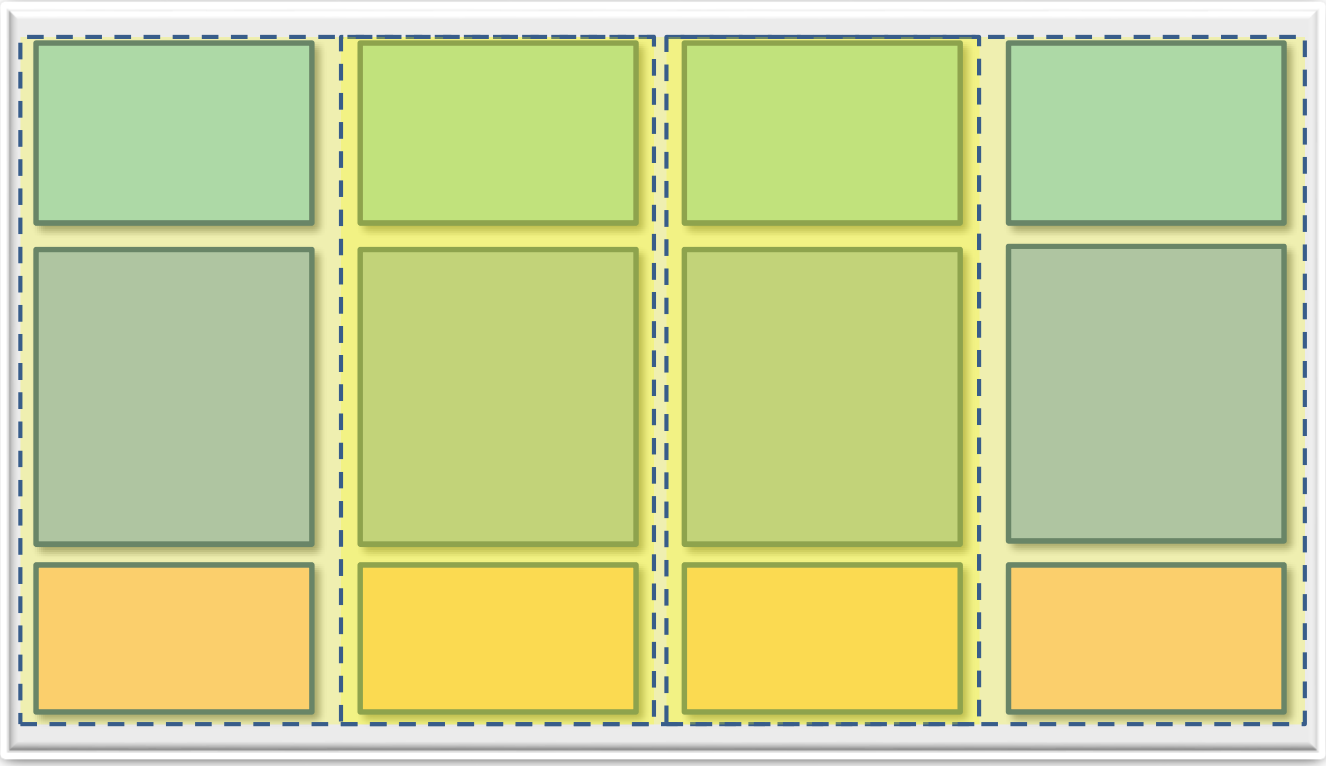
Scenario 1 (responsiveness)



Program A (yellow) is executing in the 8-way mode
Program B (red) needs some execution time
→ Resources for program A are scaled down for a while (to 4-way)
Program A continues without interruption from program B
Bottomline: Program A was always executing and responsive

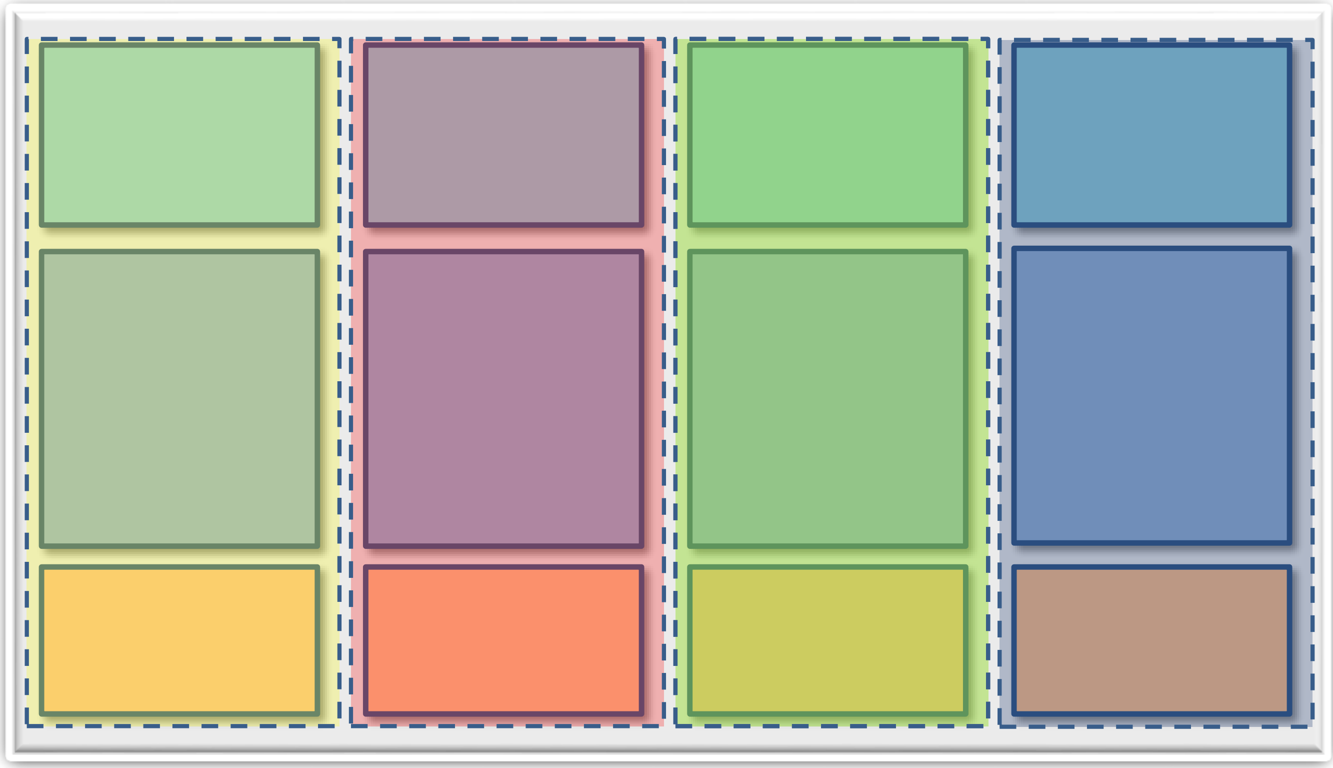
TU Delft

Scenario 2 (fault-tolerance)



Program A (yellow) is executing in the 8-way mode
Program A encounters a critical code section
→ Code is being triplicated on run (slower) on multiple smaller cores
→ Moving code from 2-way core to another is completely transparent
Program A continues as usual in 8-way mode
Bottomline: Program A utilizes redundancy to increase fault coverage

Scenario 3 (context switching)



4 programs running in parallel (4 contexts are present)
1 program can be given more resources by halting other program(s)
→ Contexts of other programs remain inside core (i.e., no memory transfers)
→ Restarting of other programs do not require expensive context switching
4 programs continue running in parallel (albeit at different cores)
<u>**Bottomline: Switching modes and execution cores required zero context switches**</u>

Some more recent developments

- Adding a benchmark takes 15-30 minutes --> running on actual hardware
- Powerstone, MiBench, SPECint 2006 running
- Tracing takes 15 minutes + 30 minutes post-processing resulting in ˜4 GB
- MMU being finalized --> Porting recent Linux
- Student project to work on robots
- Many (technical) details skipped
- Invitation to drop by our lab and see our demos, which includes a playable version of DOOM™

Analogy cont'd: An Army of Delivery Drones

Liquid Computing:

- *Run-time adaptivity*
- *Efficient computing*
- *Fault-tolerance*
- *Efficient HW utilization*
- *Energy-efficient computing*
- *Many exciting ideas to explore!!*

Like having an army of delivery drones of all sizes flowing through our campus!

Liquid Computing in Space

Delfi-C3 from TU Delft

- Harsh environment requires run-time adaptability, e.g., fault-tolerance
- Certain control systems need responsiveness

We are working to bring Liquid Computing into SPACE!

Thank you!

Questions
???

Contact information:
Stephan Wong
J.S.S.M.Wong@tudelft.nl
http://www.ce.ewi.tudelft.nl/wong/