Modern (Embedded) Processor Systems

Prof. Dr. Akash Kumar

Chair for Processor Design

(Ack: my past and current students/PostDocs)

(Some slides adapted from Koren, Krishna, Anand)

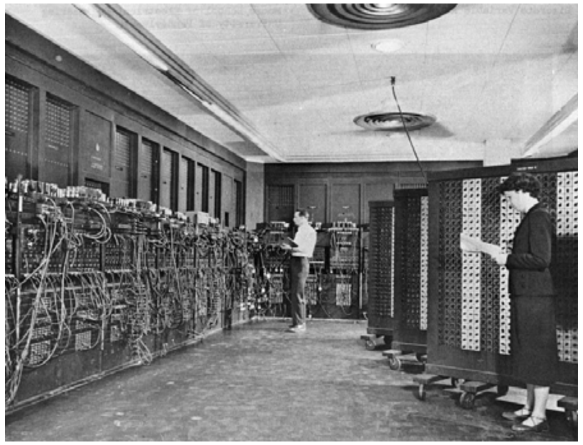
Outline

- History of computer systems
- Trends in modern computer systems
- Design flow and considerations
- Modern challenges and solutions(??)

History of Hardware / VLSI

- Vacuum tube

(Lee De Forest, 1906)



- ENIAC

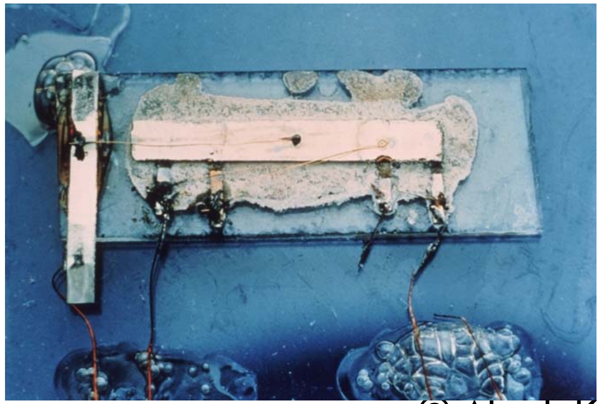
(1946, UPenn)

- Transistor

(1947, Bardeen, Brattain, Shockley)

- Integrated circuit

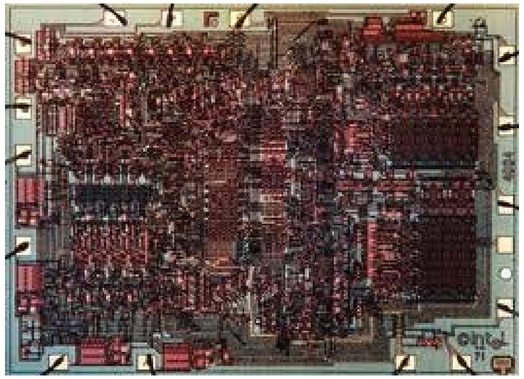
(1958, Jack Kilby)



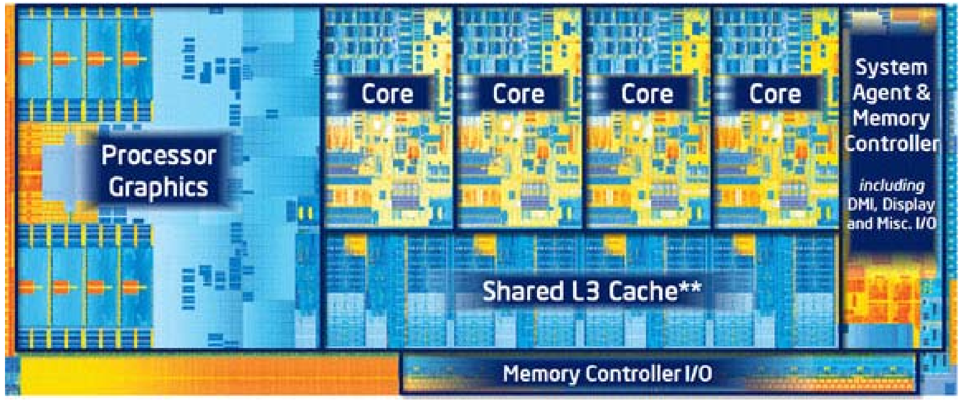
History of Hardware / VLSI

- Intel 4004

  (1971, 1400 transistors)



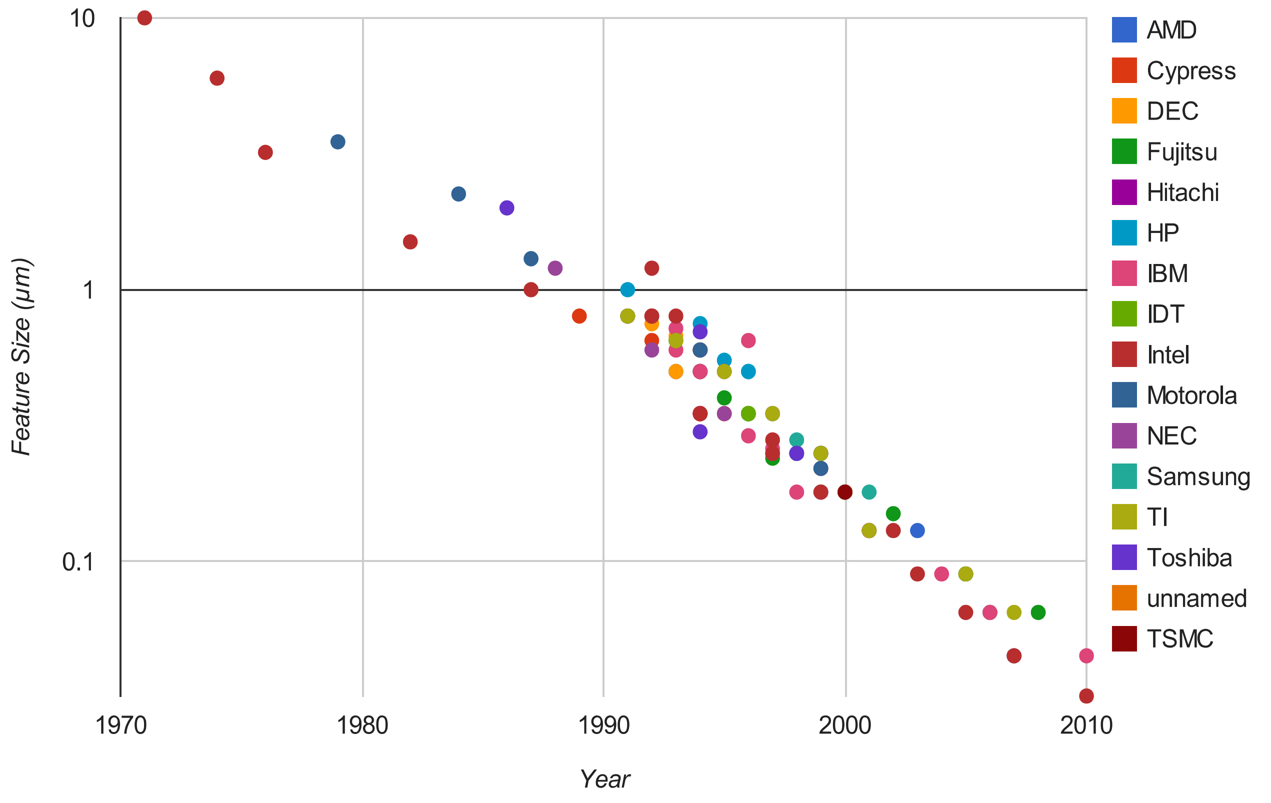
- Intel Core i7 - Ivy Bridge (2012, >1.4 Billion transistors)



- Very Large Scale Integration (VLSI) – originally defined for chips having transistors in the order of 100,000. Other terms such as ULSI came along, but the usage VLSI remains dominant

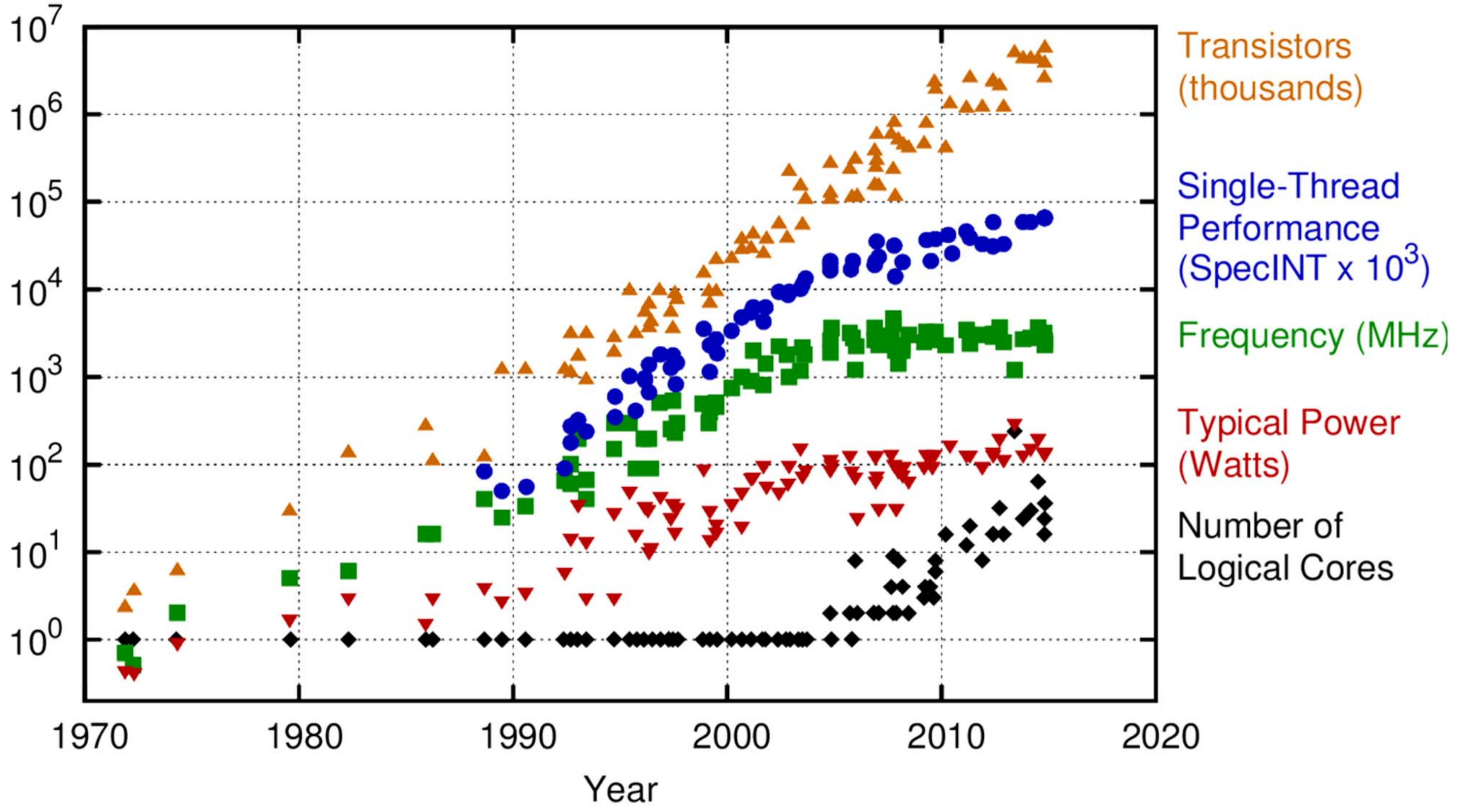
Moore's Law

- In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years



http://cpudb.stanford.edu/

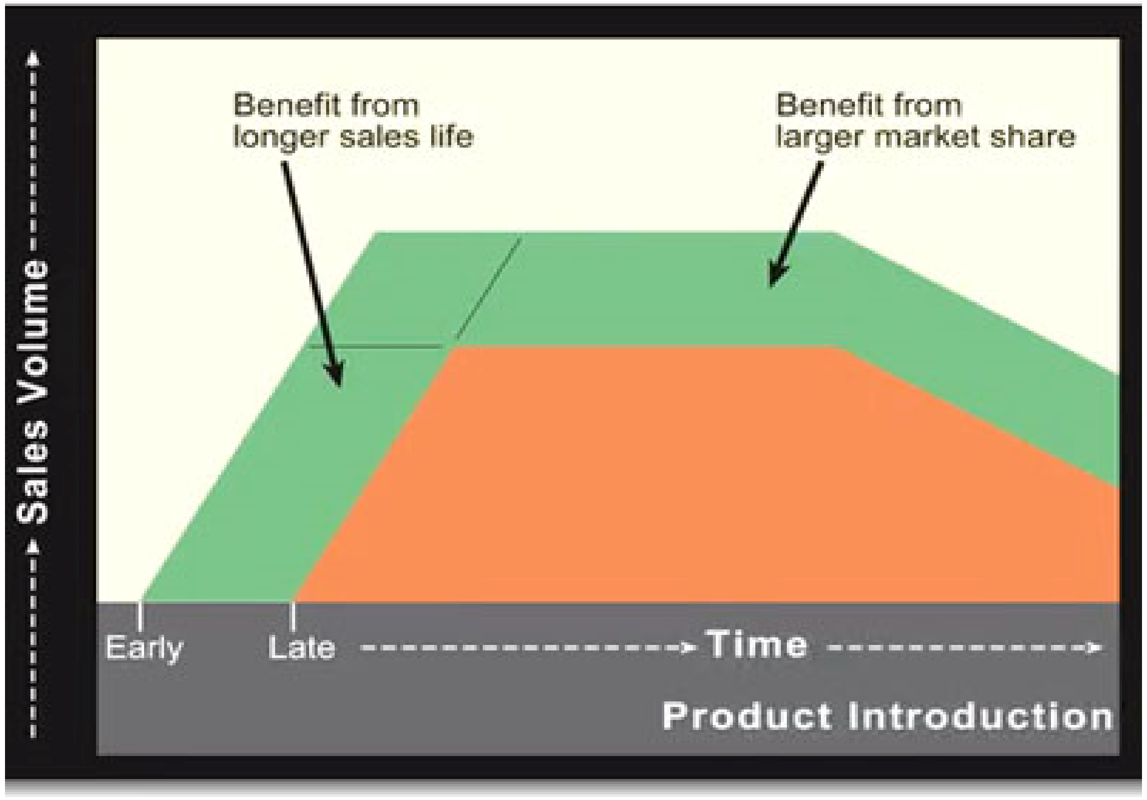
40 Years of microprocessor trend data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Design Productivity Gap

- Increasing number of transistors makes it harder to design the system
  - Late launch of products directly hurts profits



System Design Considerations

- System : sensor -> processor -> actuator
- Considerations
  - Technology
  - Performance
  - Power consumption
  - Volume of production
  - Upgradability / ease of maintenance
  - Reliability
  - Testability
  - Availability of CAD and software tools, IP's, hardware and software libraries
  - Cost, chip area
  - Legal and certification requirements, client specifications
  - …..

Digital Hardware Market Segments

- Processor, GPU

- DRAM, Flash memories

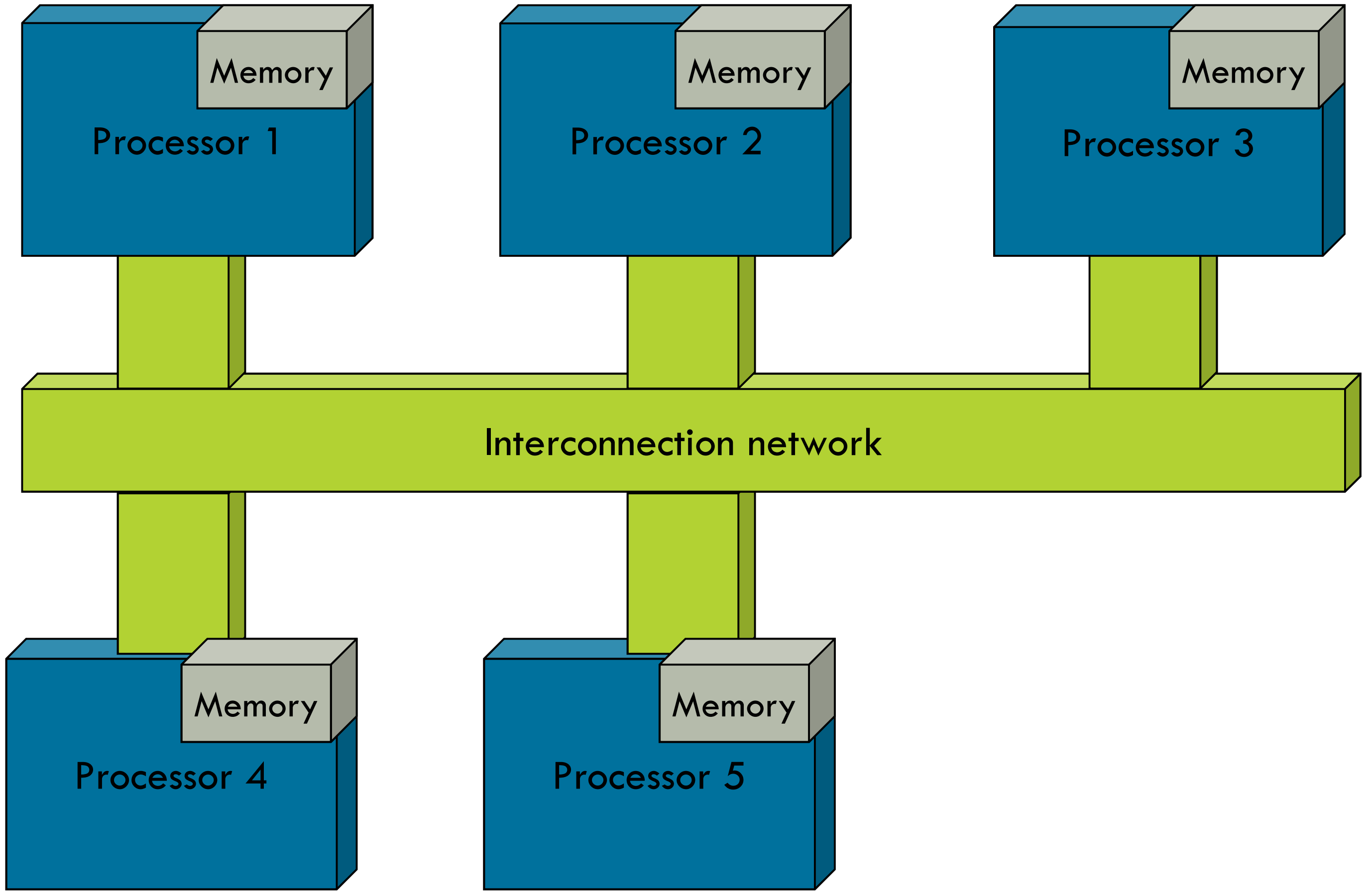
- (Co-)Processor alternatives
  - ASIC (application specific integrated circuit)
  - ASSP (application specific standard product)
  - FPGA (field programmable gate array)

- Convergence as System on Chip (SoC), which may also contain analog, mixed-signal, and radio-frequency functions

Embedded systems architecture

- Trend towards Multi-Processor Systems-on-chip (MPSoC)
- Homogeneous vs heterogeneous systems
- Different memory models
- Different network architectures
  - Network-on-chip
  - Buses

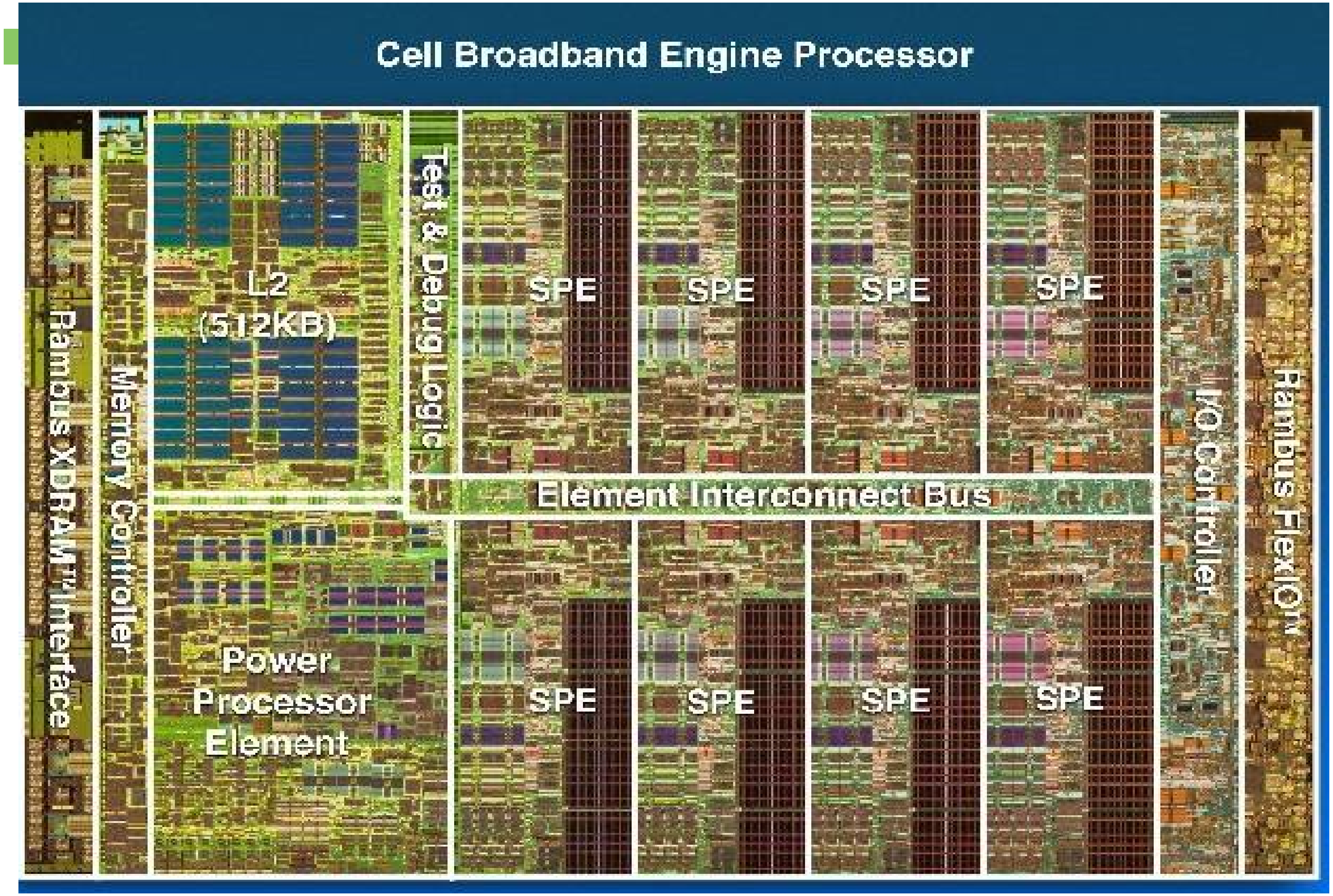
Homogeneous vs heterogeneous



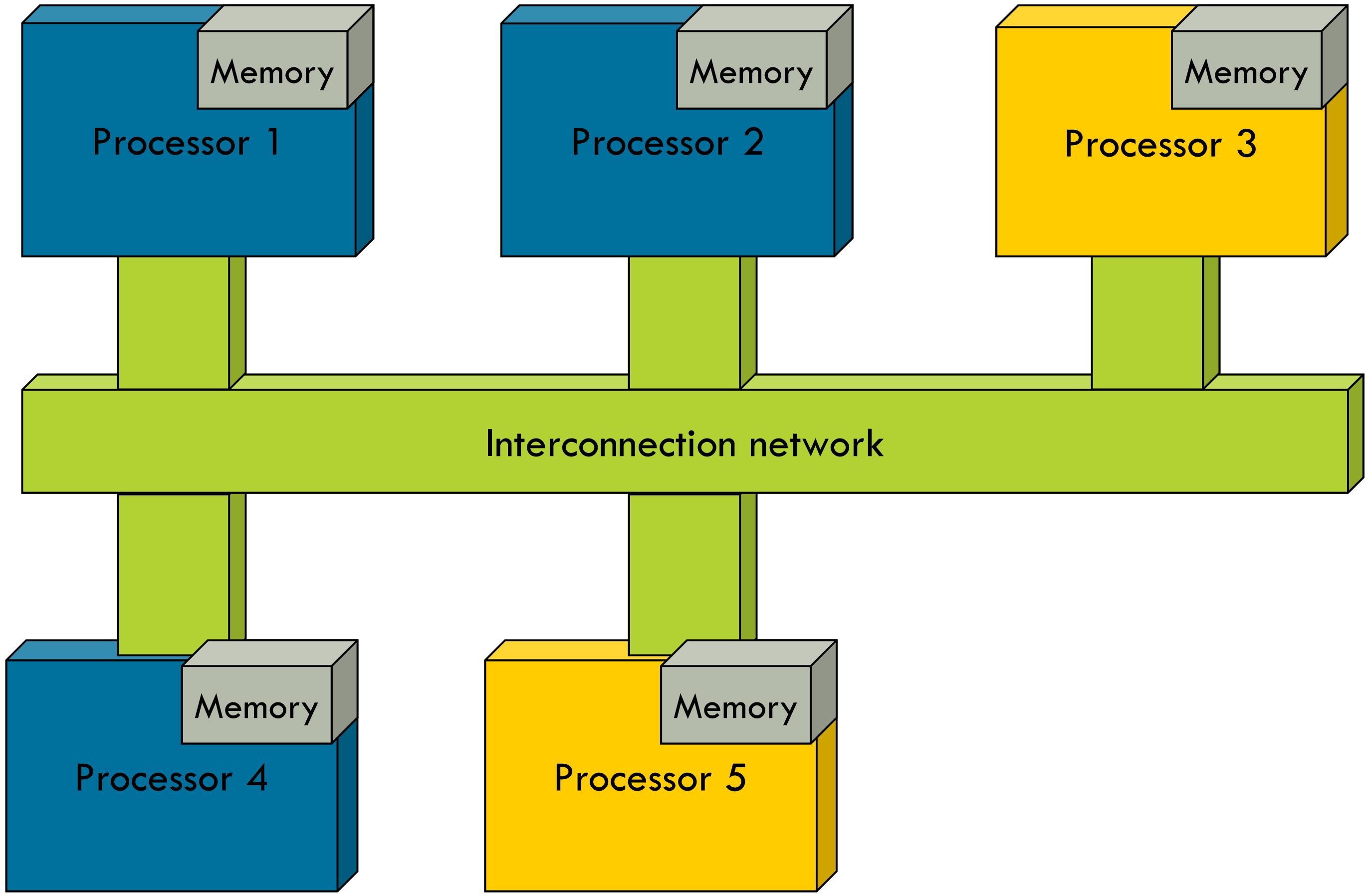
Homogeneous vs heterogeneous

- Heterogeneity is increasing
  - Different levels of parallelism in application
  - uProc – better for control-flow
  - DSP – better for signal processing
  - Dedicated hardware blocks needed for certain parts
  - Improves efficiency and saves power
- Homogeneous systems
  - Better for fault-tolerance
  - Only one compiled version of any application needed
  - Easier to design and replicate
  - Easy to support task migration

Memory usage



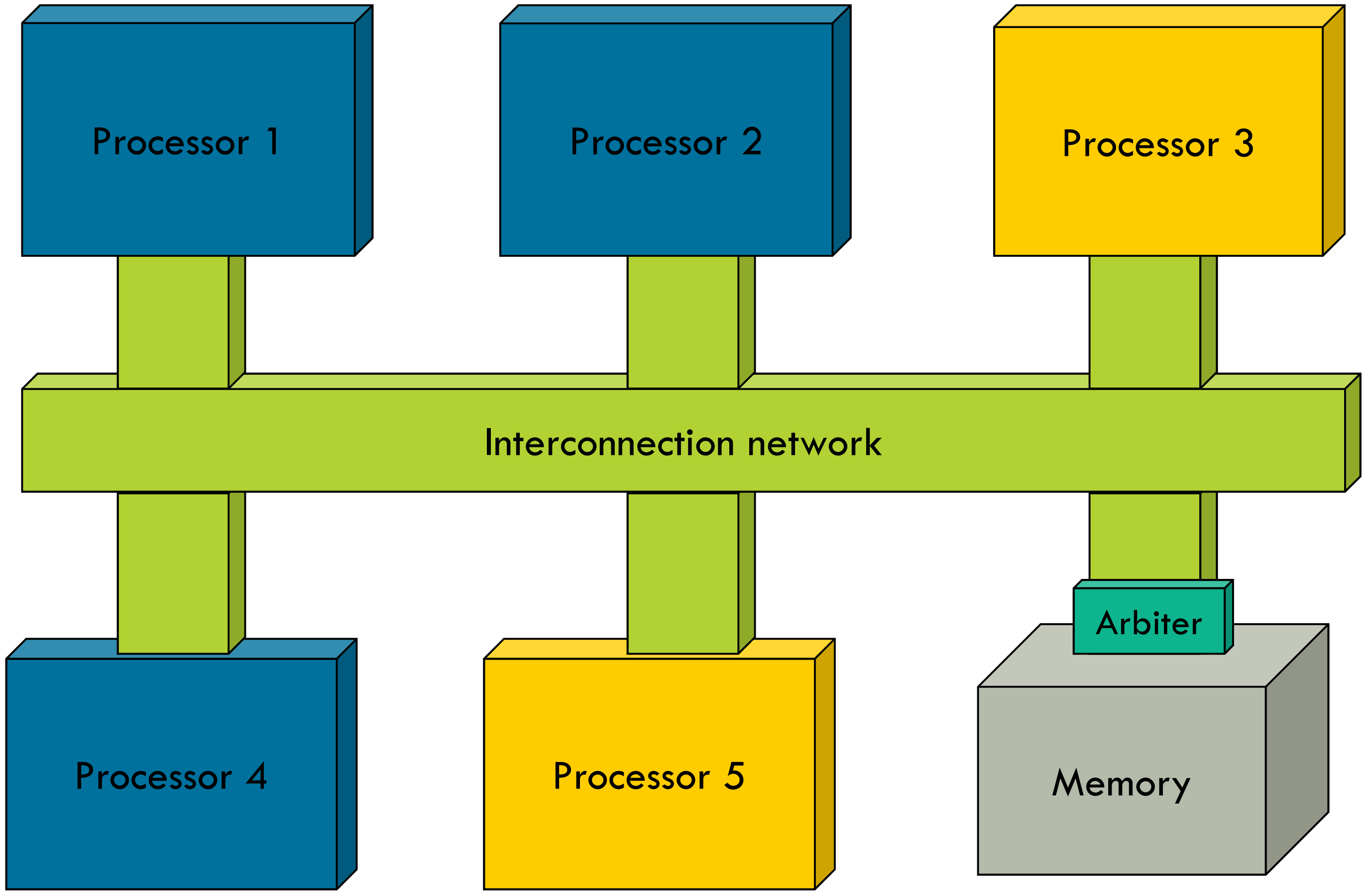
Embedded systems – local memory



Local memory is better for more predictability

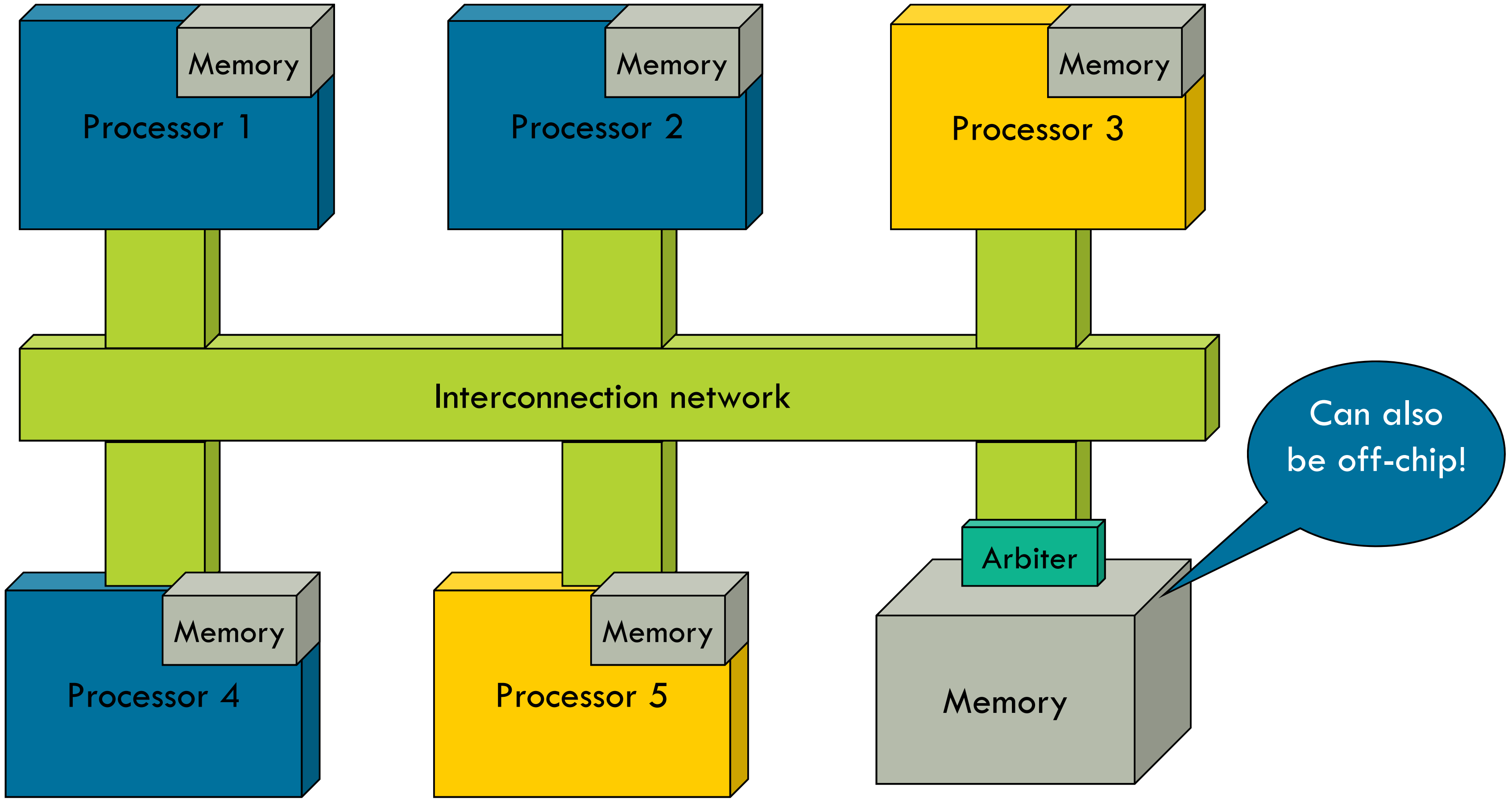
Network/ bus delay may be unpredictable

Embedded systems – global memory



Global memory may be better for shared data

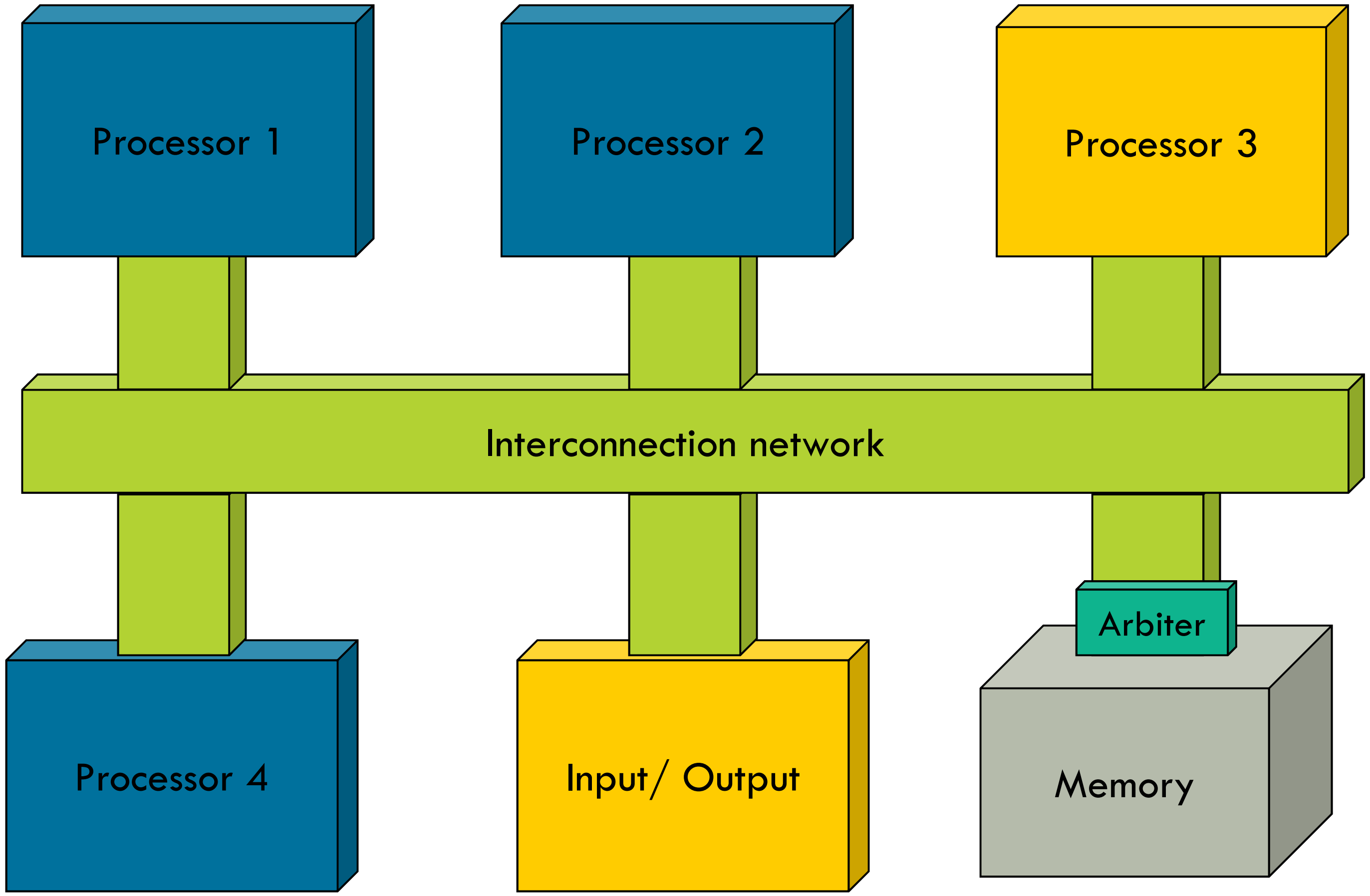
Embedded systems – combination



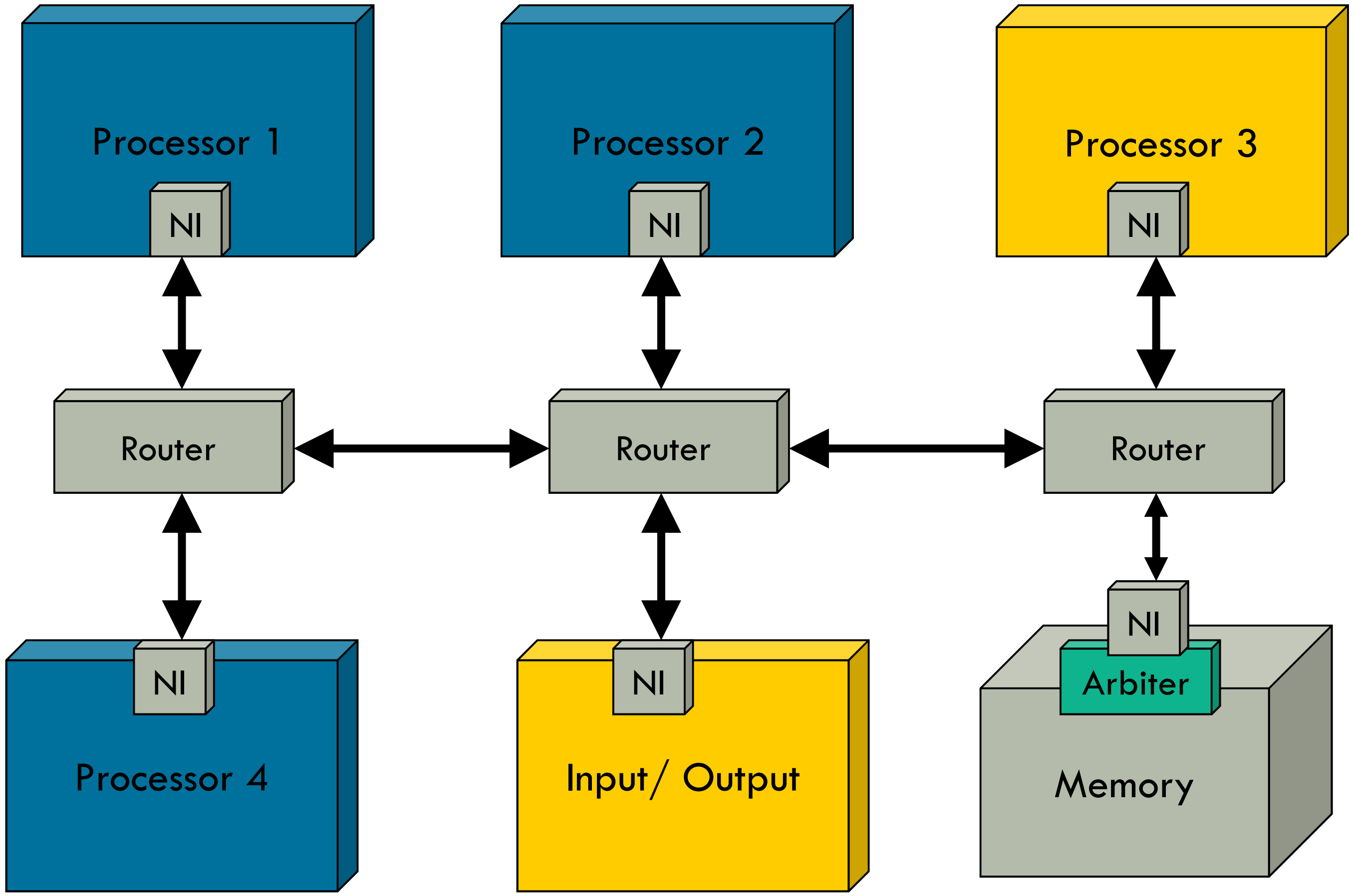
Communication pattern also determines which architecture is better

Message passing OR Shared memory

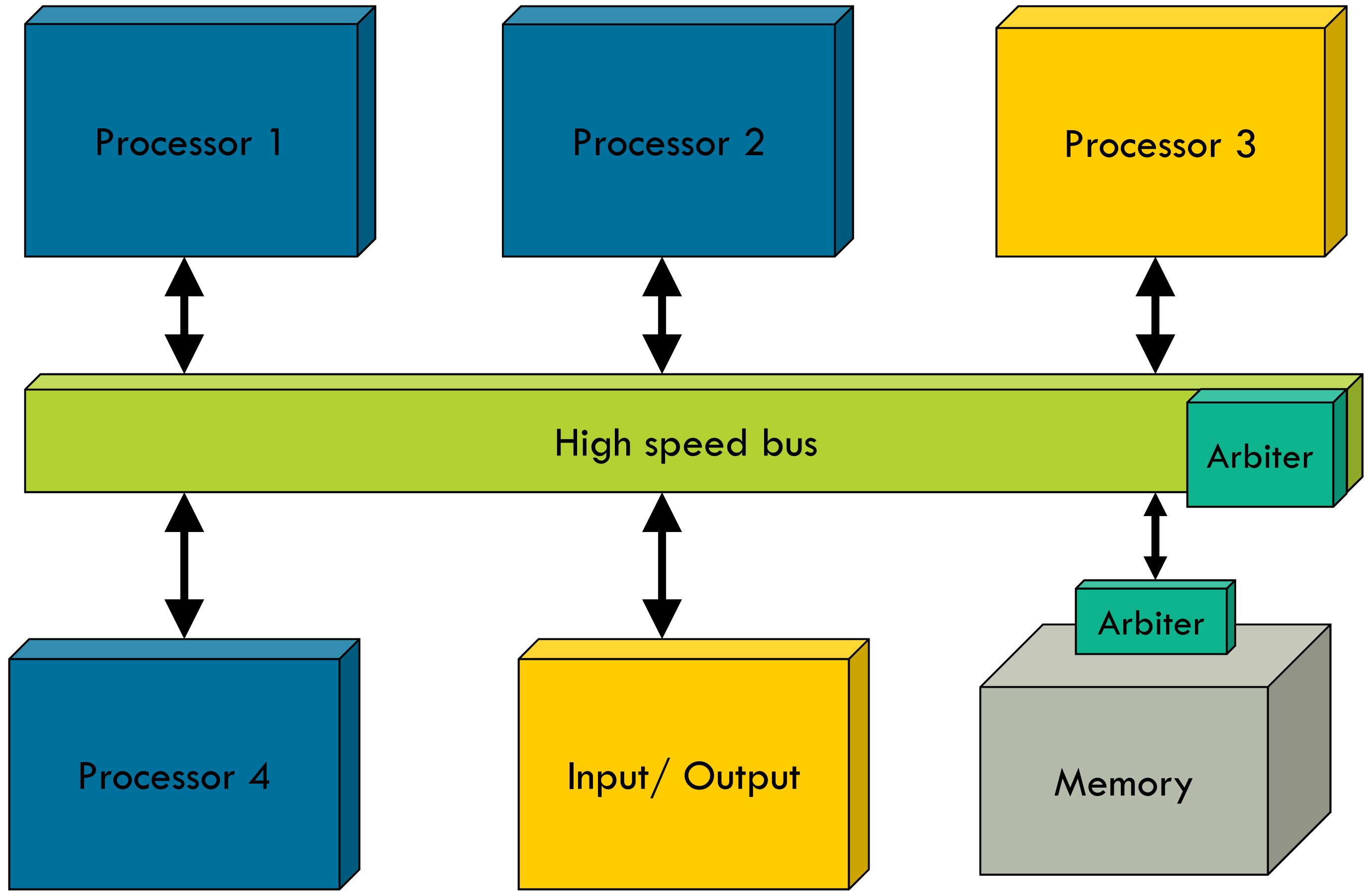
Embedded systems – network



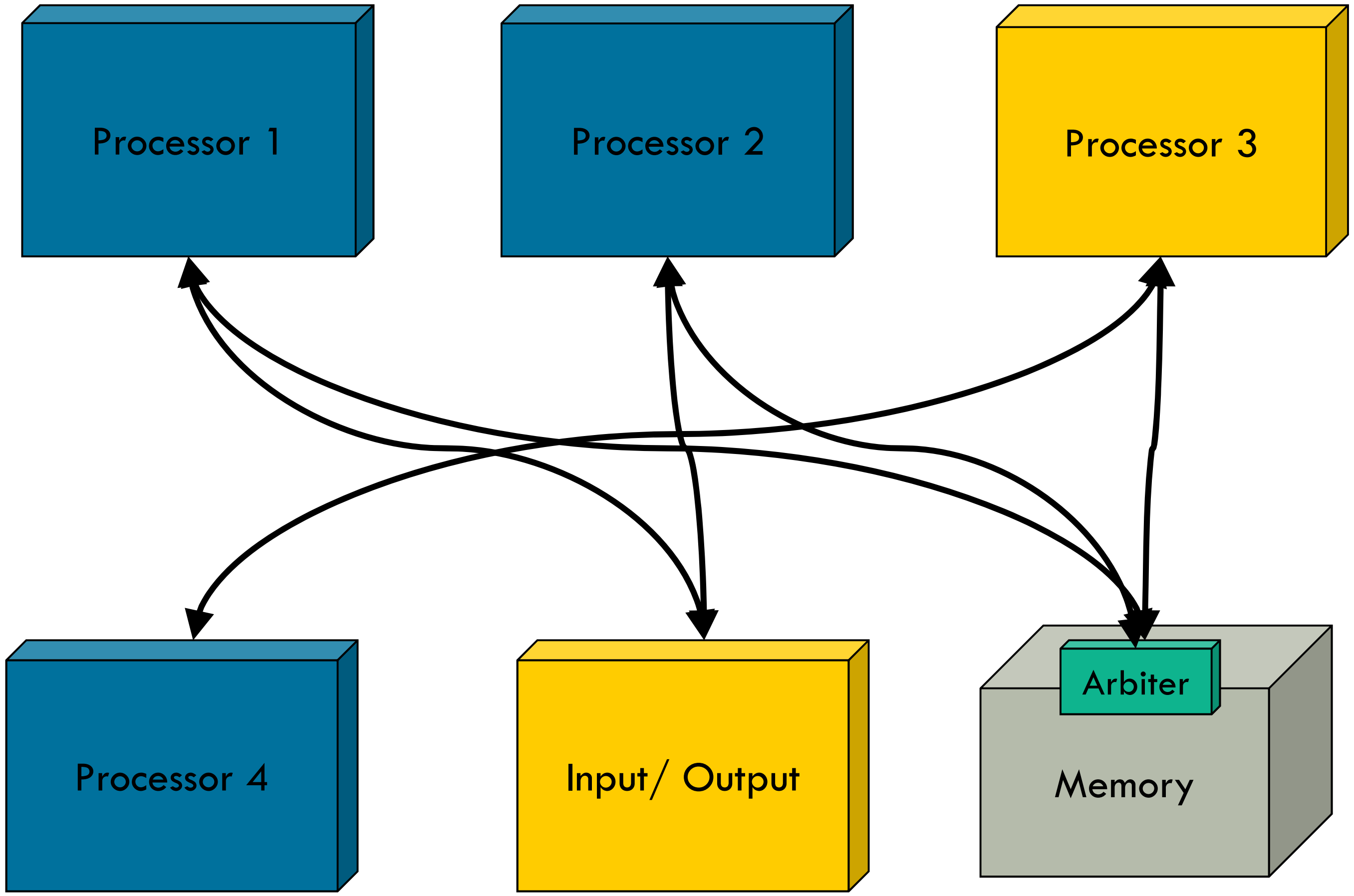
Interconnection network-on-chip



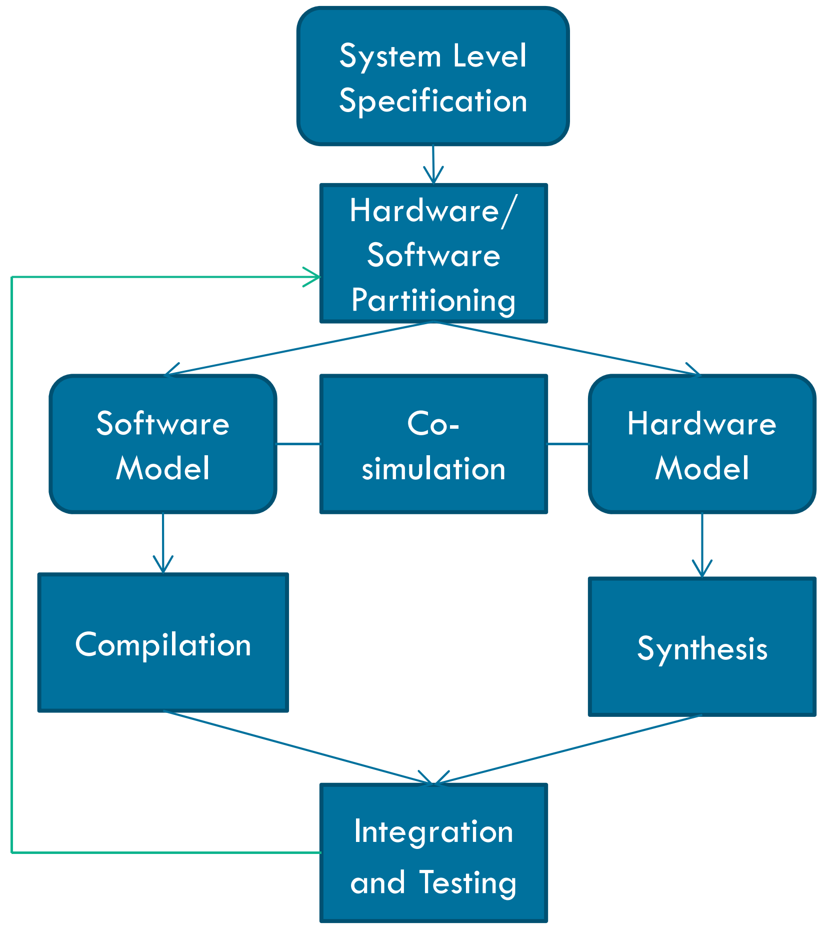
Interconnection network – bus



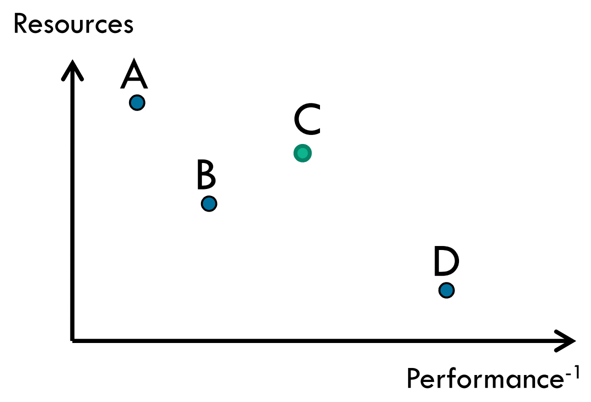
Point-to-point networks



System Design – Hw/Sw Codesign



- Take decisions on whether to implement in hardware or software
  - Consider the advantages vs costs
- If hardware, whether to use commercial off the shelf (COTS) components or custom components



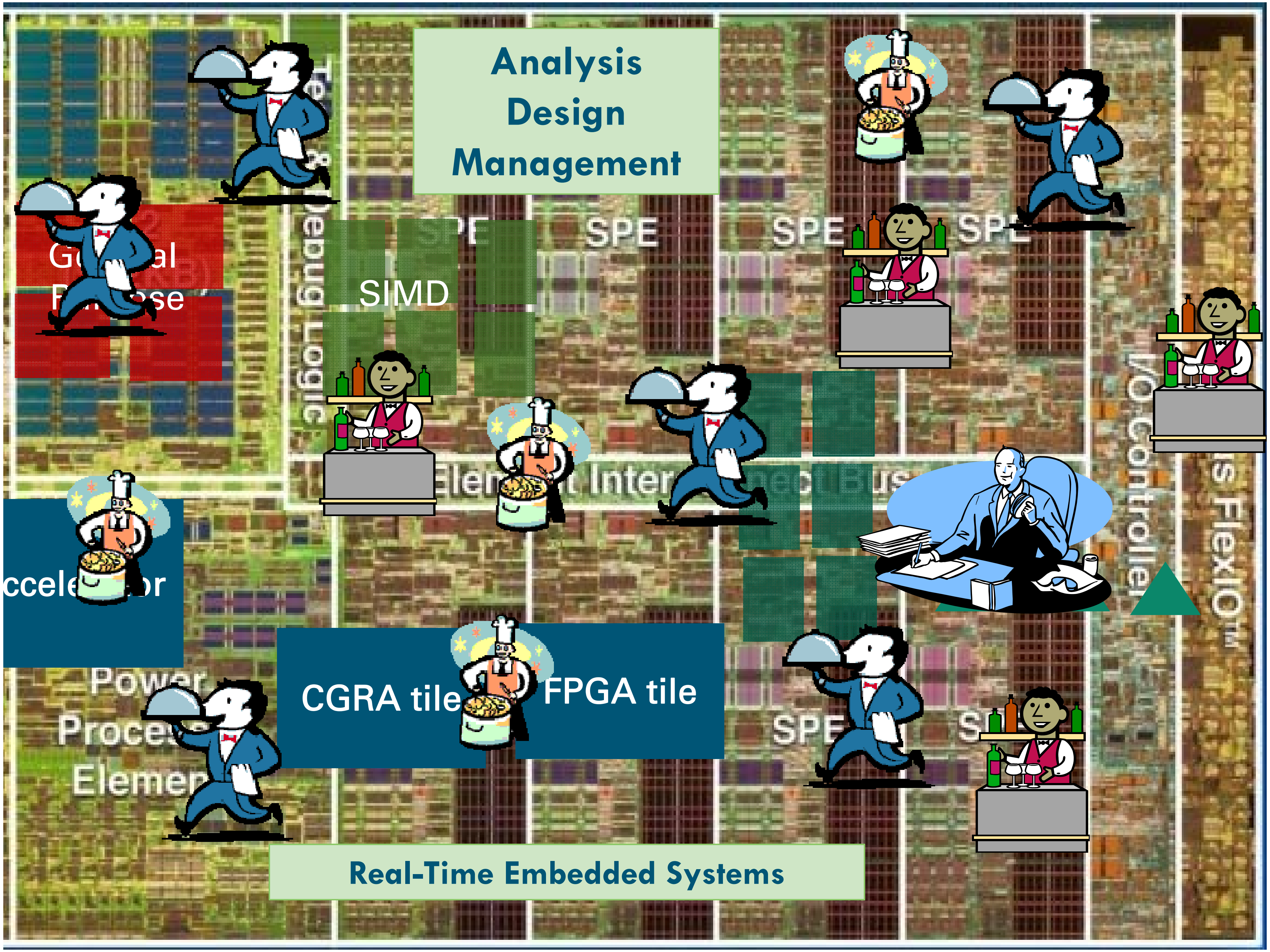
Pareto Curve

Modern Multimedia Embedded Systems

Run-time addition of appl
Large number of use-cases
Guarantee performance
Minimize power consumption
SONY
Heavy-Duty

Predictable Design Flow

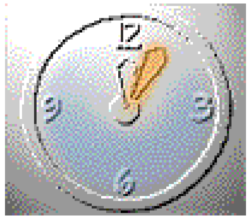
Analysis
Design
Management



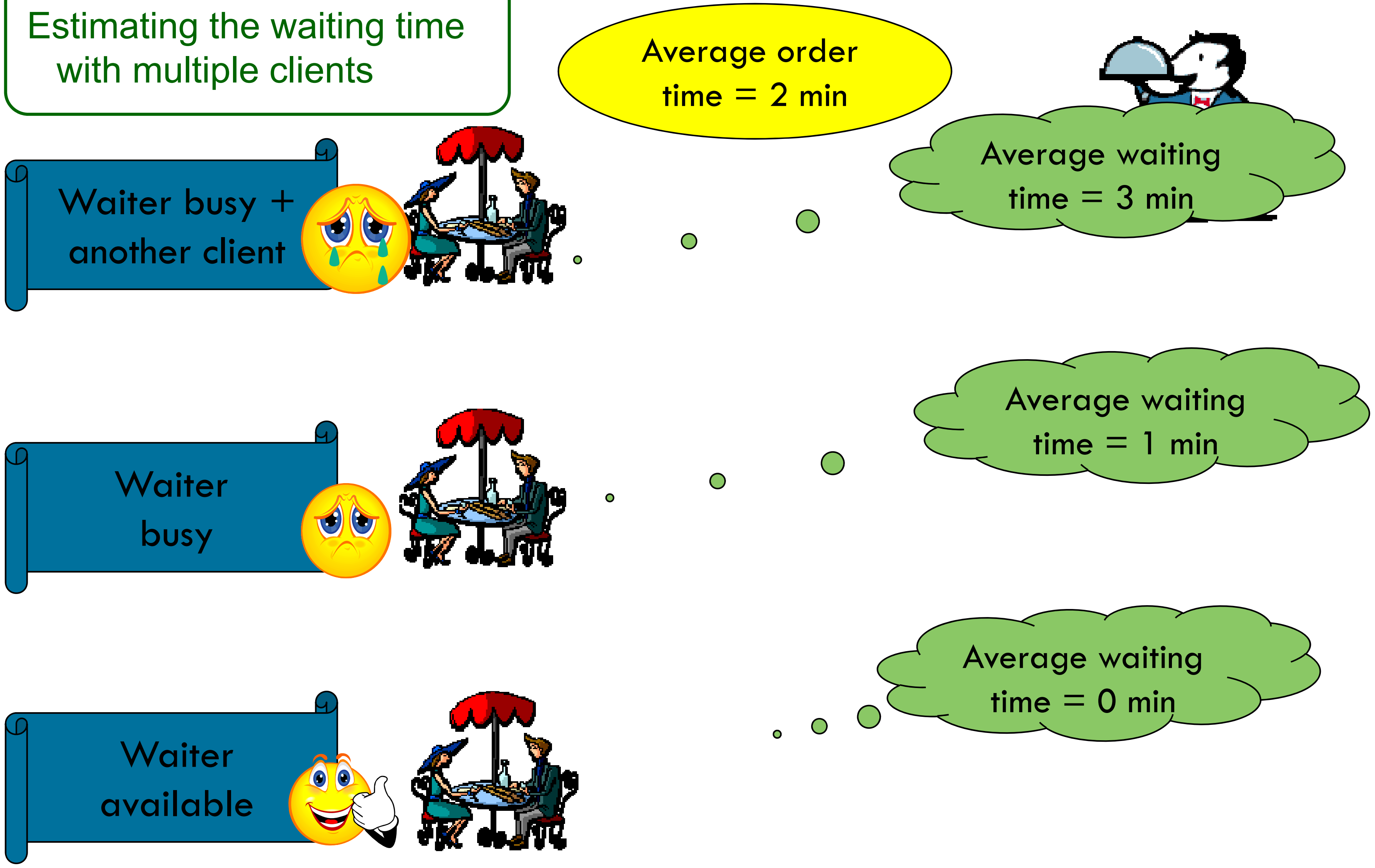
Real-Time Embedded Systems

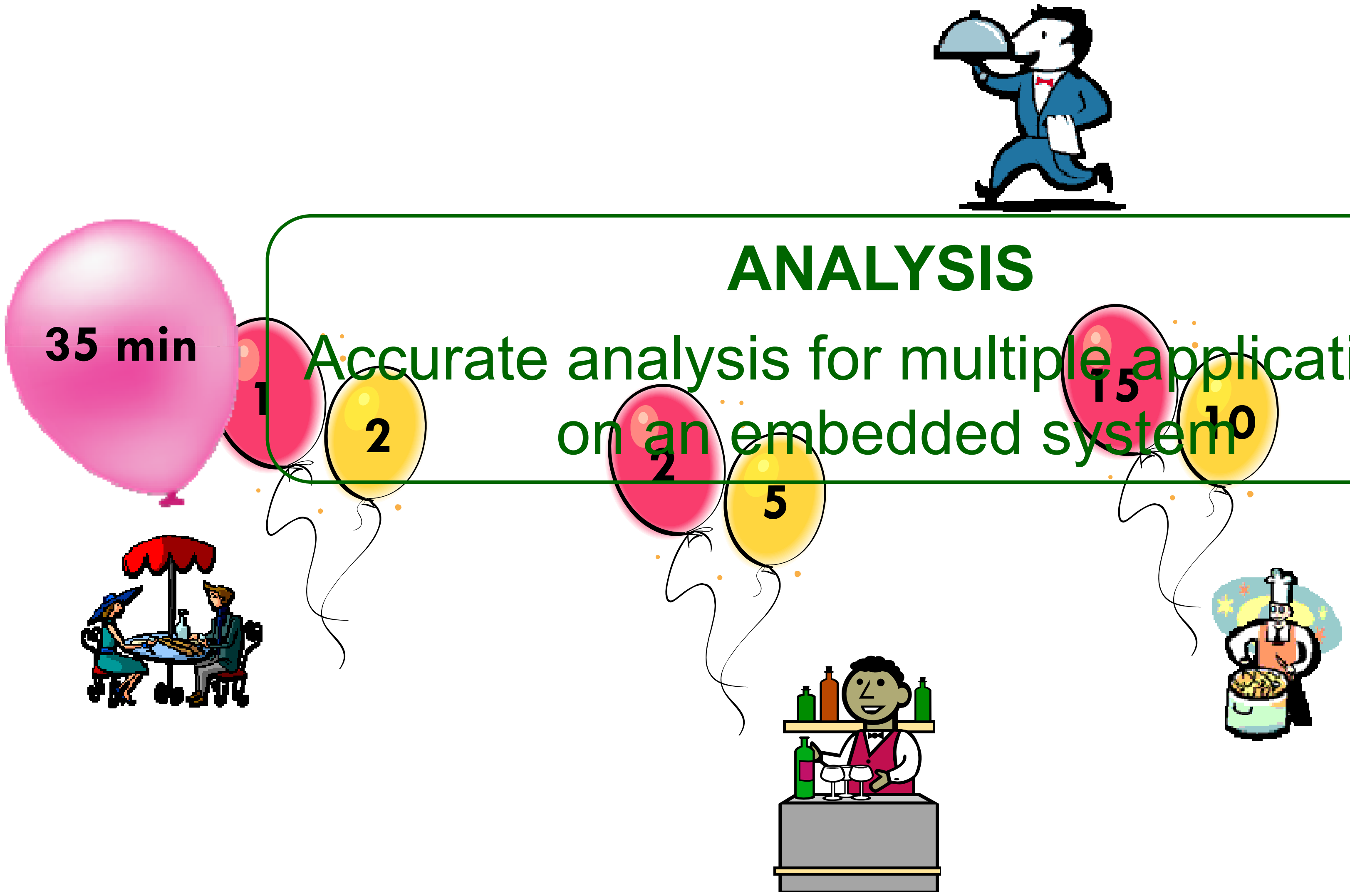
ANALYSIS: Time Spent in a Restaurant

Restaurant



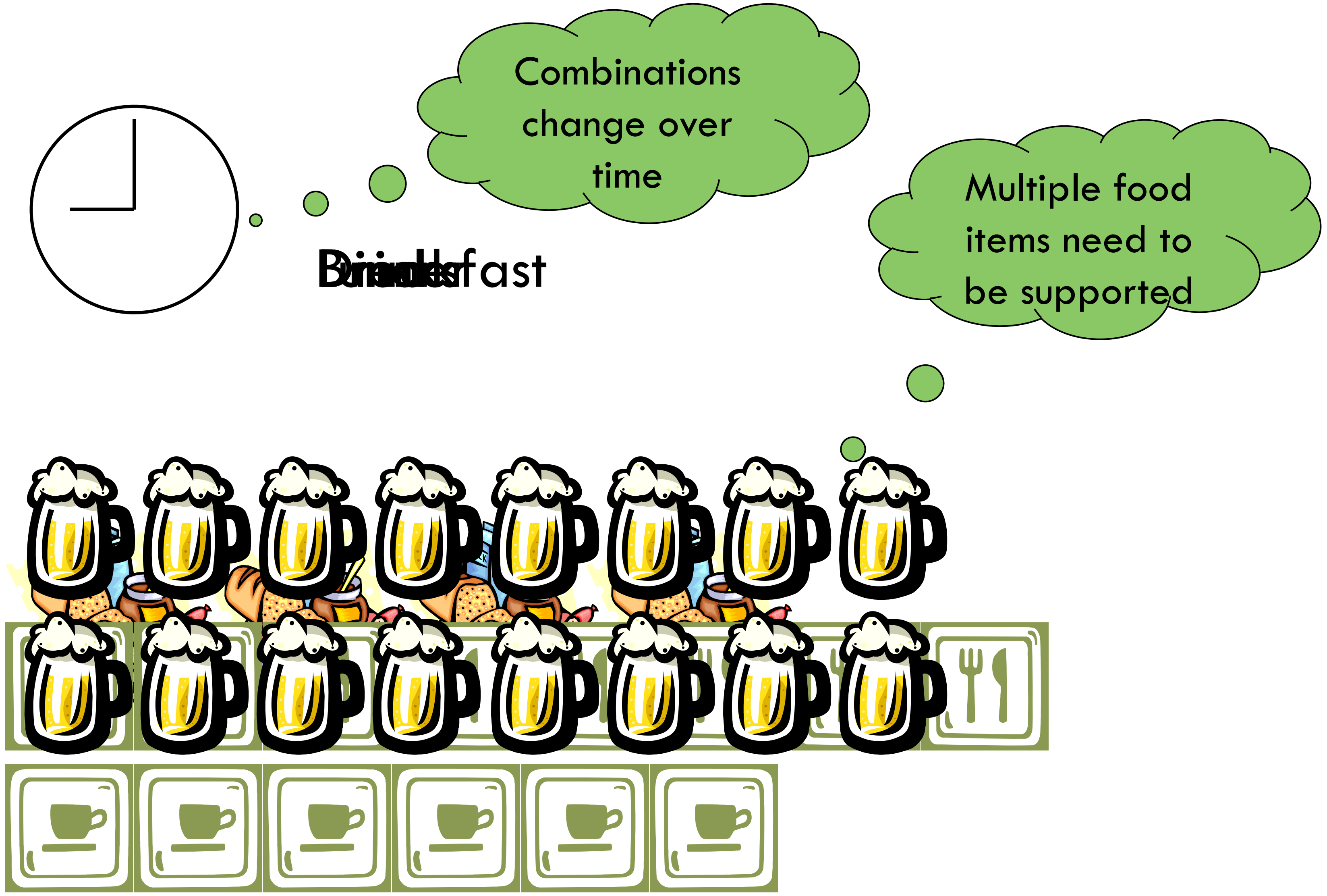
Estimating the waiting time with multiple clients

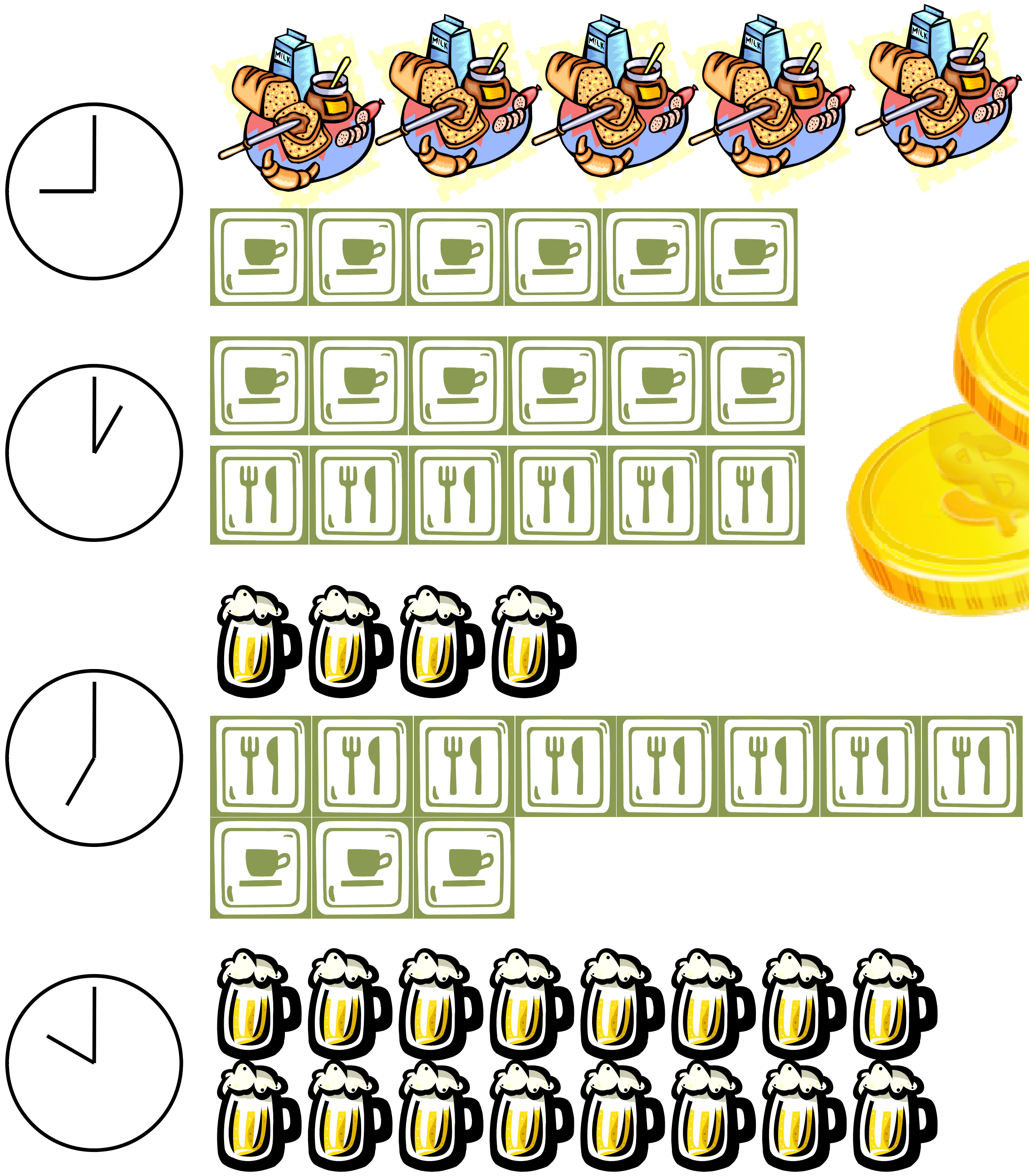

Average order time = 2 min
Waiter busy + another client
Average waiting time = 3 min
Waiter busy
Average waiting time = 1 min
Waiter available
Average waiting time = 0 min



ANALYSIS

Accurate analysis for multiple applications on an embedded system


Combinations change over time
Breakfast
Multiple food items need to be supported



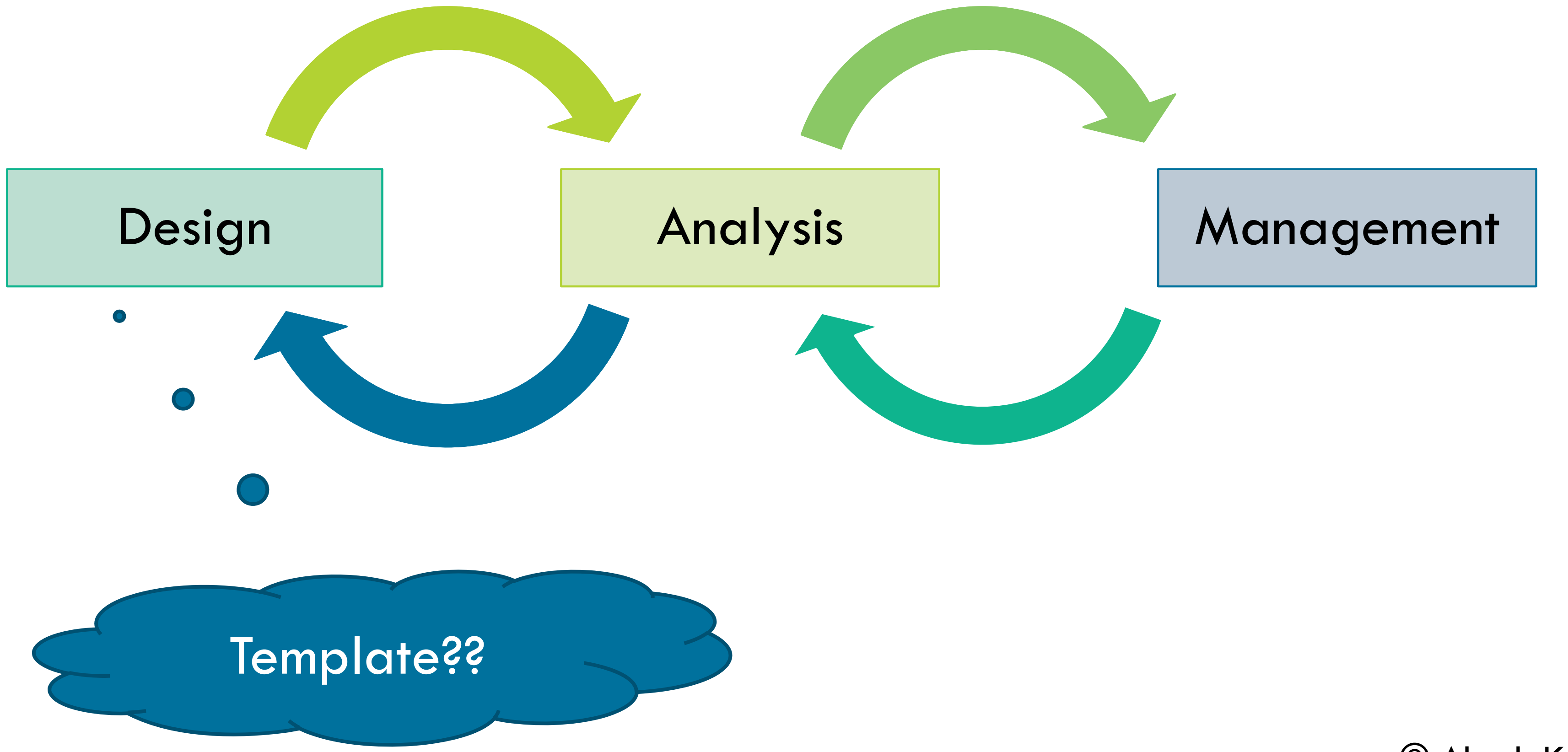
DESIGN

Automated design technique for multiple combinations of applications

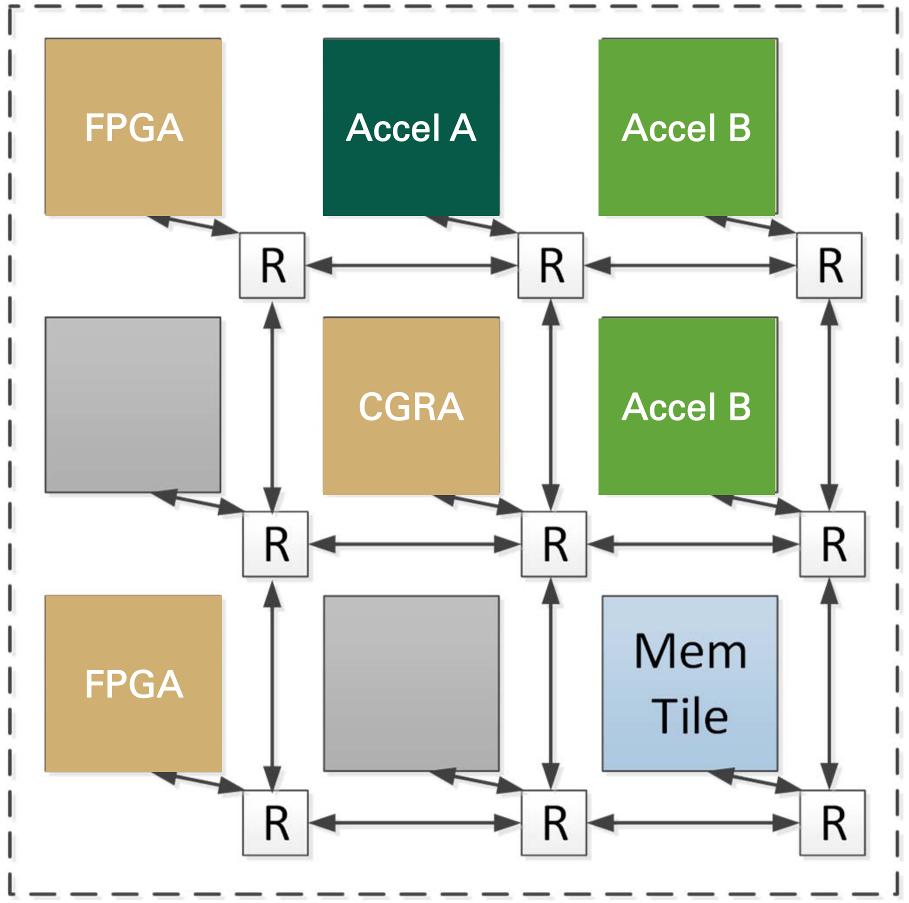
MANAGEMENT

Resource manager for heterogeneous systems running multiple applications

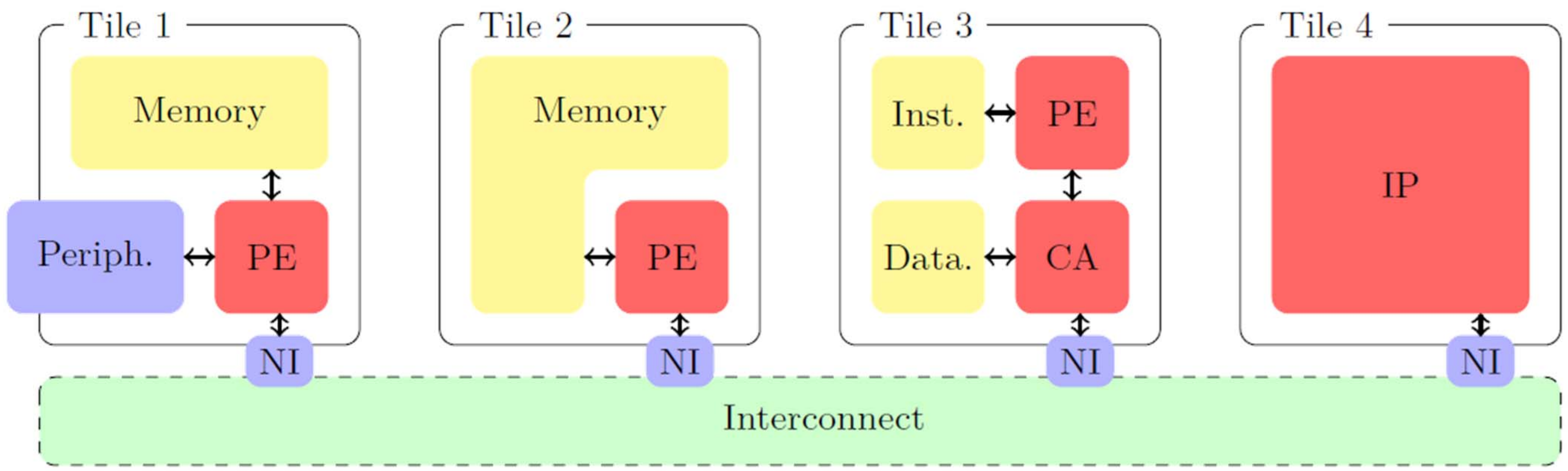
Design- and Run-time Flow



Design Template

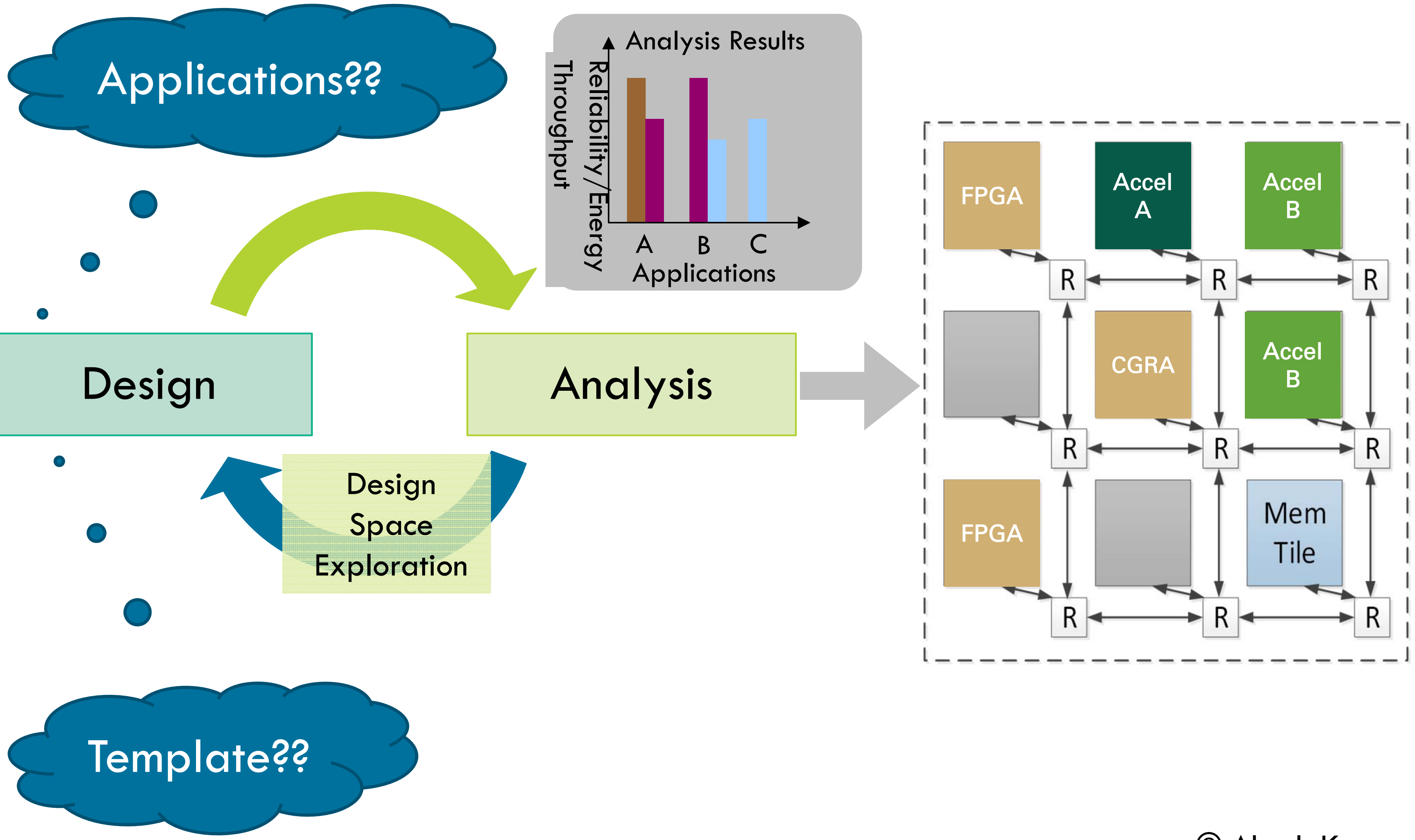


Design Template



CA: Communication Assist (DMA like)

Design- and Run-time Flow

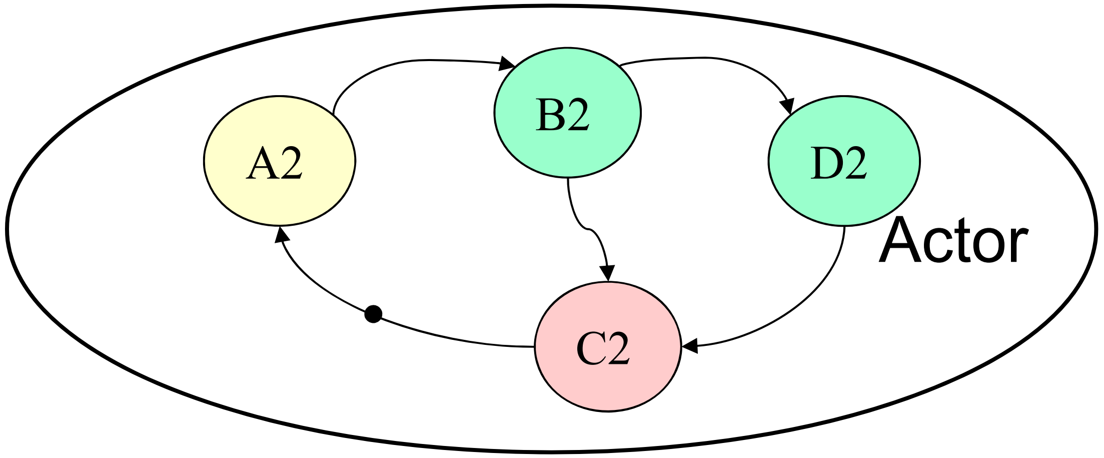


Design- and Run-time Flow

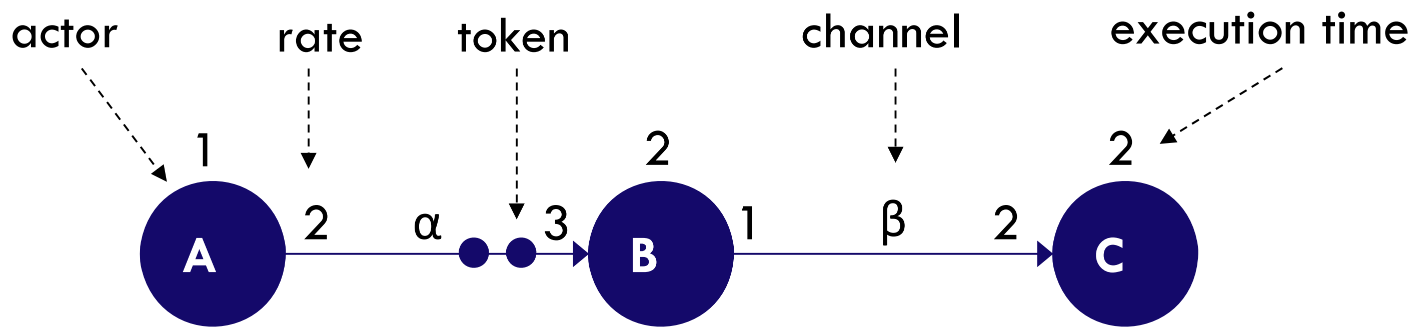
- Applications are known?
- Can multiple applications run simultaneously?
- Application models are available?
- Application domain(s) is known?
- Use representative applications…

Analysis – SDF Graph

- First proposed in 1987 by Edward Lee
- SDF Graphs used extensively
  - SDFG: Synchronous Data Flow Graphs
  - DSP applications
  - Multimedia applications
- Similar to task graphs with dependencies



Synchronous Dataflow Graphs



Actors

- Execution time per processor
- Memory requirement per processor

Channels

- Buffer constraints
- Token size
- Bandwidth requirements

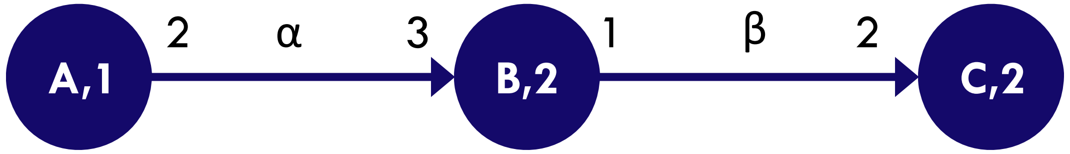
Graph

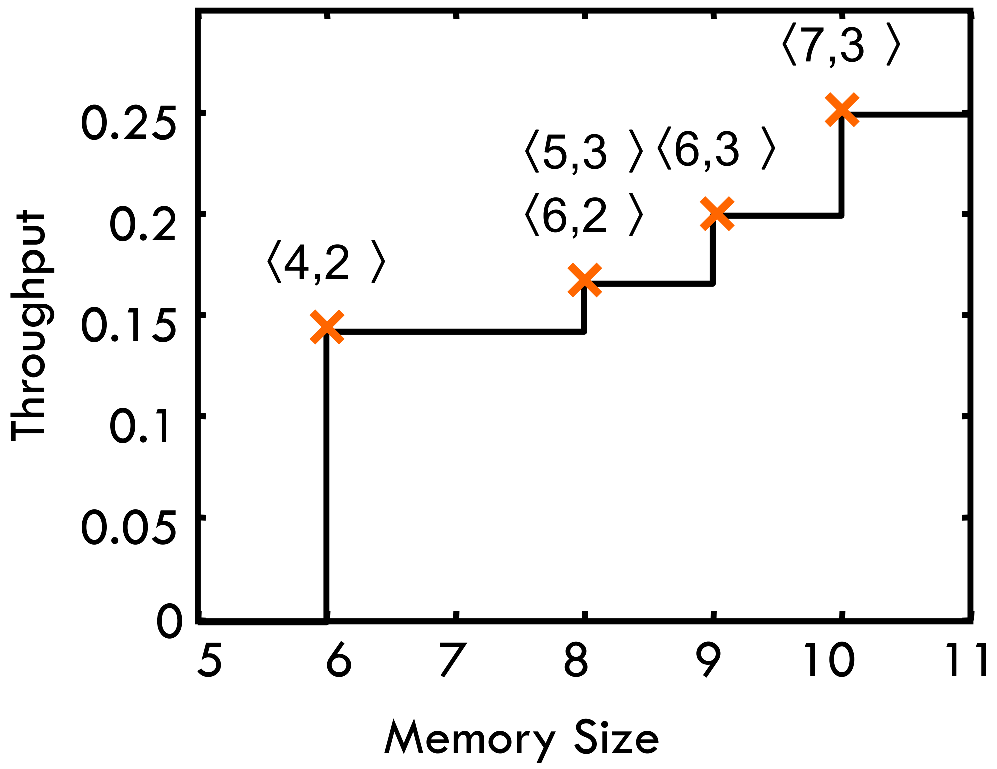
- Throughput constraint

Analysis – SDF Graph

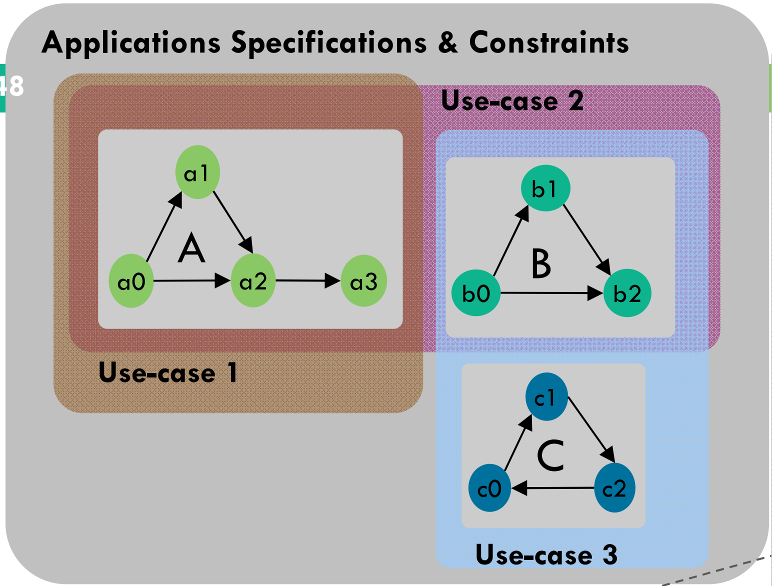
- Analyze deadlocks
- Check for consistency
- Compute throughput
- Model mapping of tasks on processors
- Model scheduling – depends on the algorithm
- Model communication bandwidth
- Model buffers – local memory and network interface
- Evaluate throughput-buffer trade-offs

Throughput-buffer trade-offs



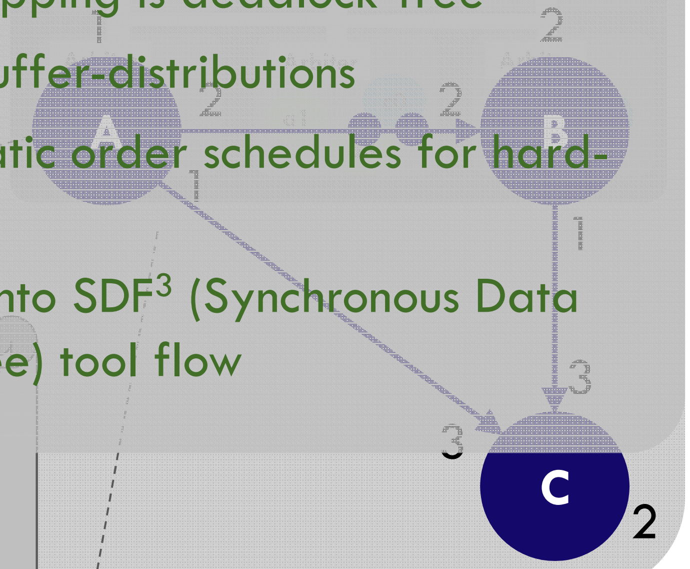


Predictable Design Flow



Mapping applications to the architecture

- Model all aspects, leading to a predictable system
- Verify if mapping is deadlock-free
- Calculate buffer-distributions
- Compute static order schedules for hard-RT apps
- Integrated into SDF^3 (Synchronous Data Flow For Free) tool flow



Architecture

1

```
static int local_variable_A;
void actor_A (TypeB *toB , TypeC *toC){
// calculate something
// and write the output tokens
toB[0] = calculate_valueB1() ;
toB[1] = calculate_valueB2() ;
*toC = calculate_valueC(local_variable_A);
}
```

xploration

Applications

Predictable Design Flow

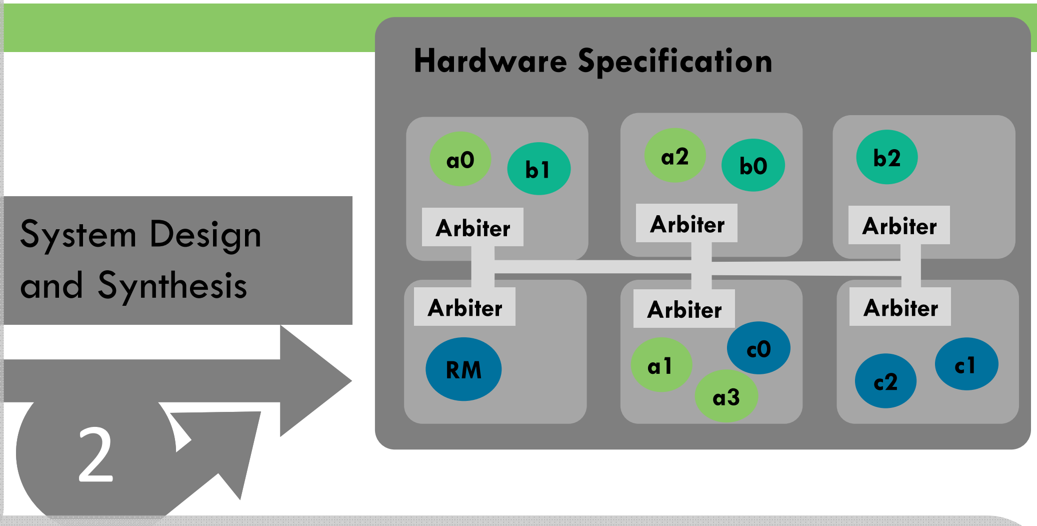
Multi-Application Multi-Processor Synthesis

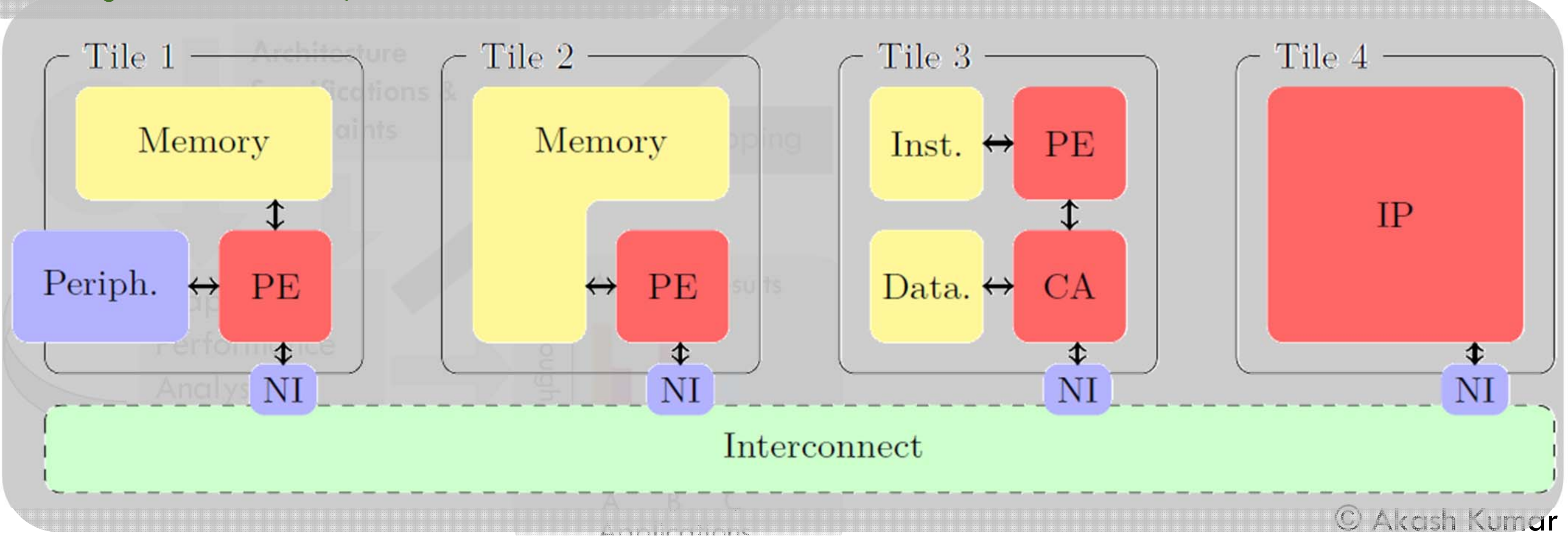
Hardware

- Instantiate processing components
- Instantiate interconnect components
- Route connections, generate VHDL code

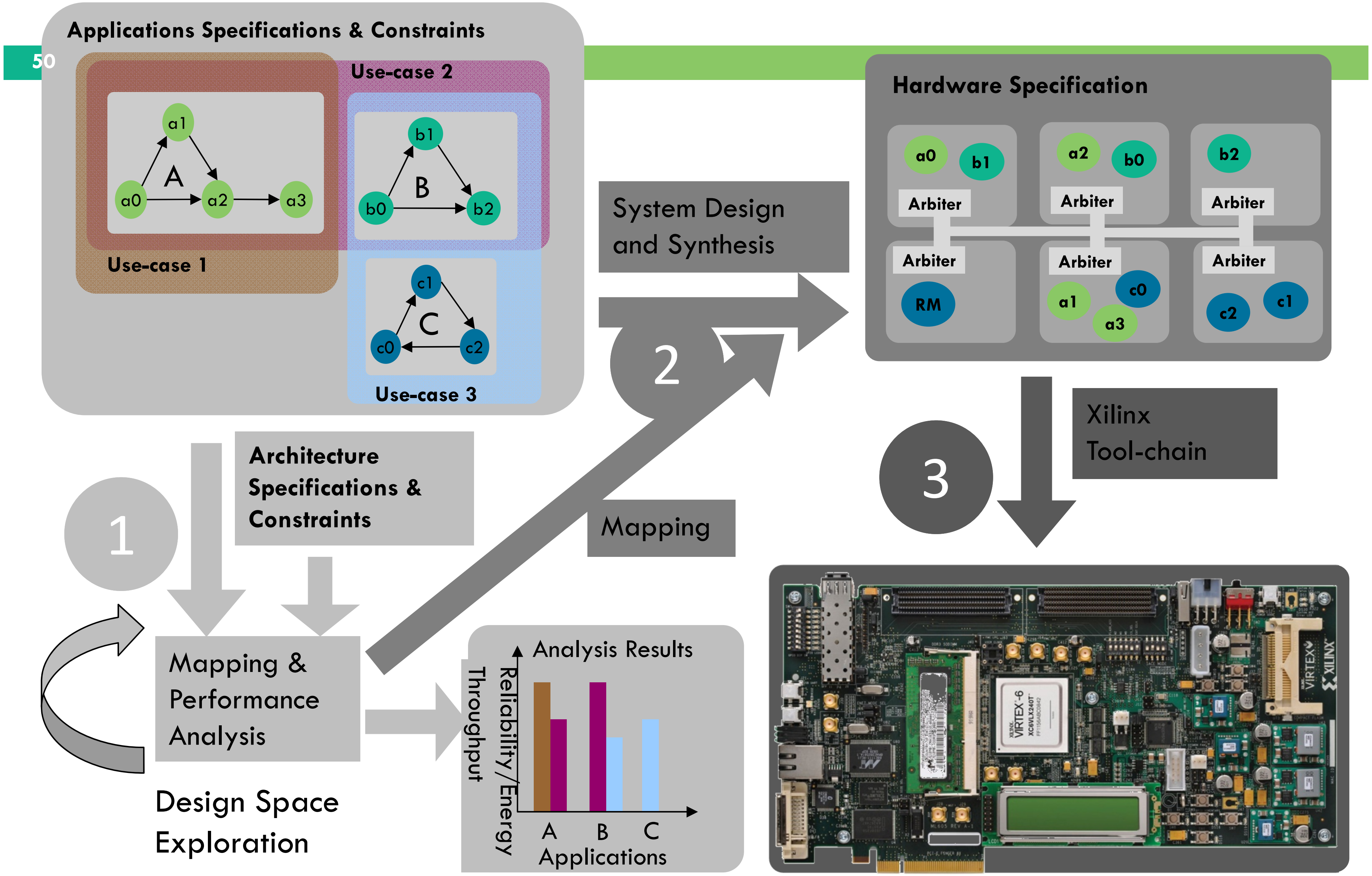
Software

- Generate wrapper code for each actor
- Reserve memory for communication
- Program connections, if needed





Predictable Design Flow



Predictable Design Flow

Design synthesized using TCL scripts

- Script ensures compatibility with different Xilinx software versions
- Carry out design space exploration

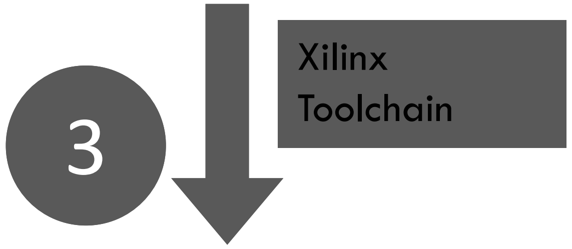
Tool-flow (MAMPS) targeted towards Xilinx FPGAs

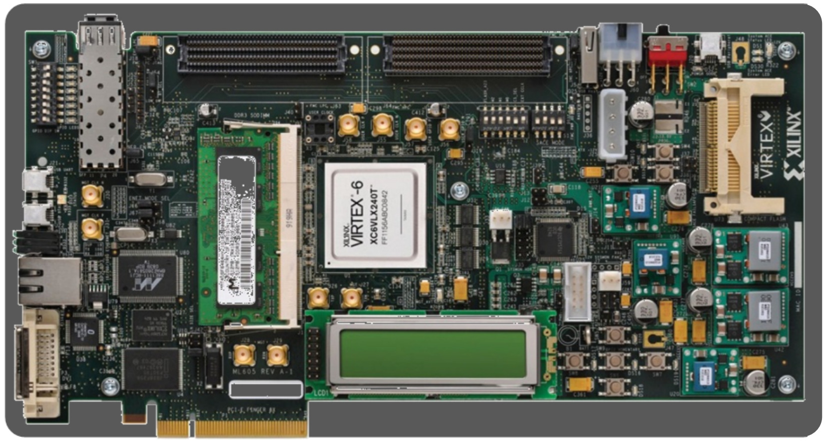
- Virtex 6 – Xilinx ML605 board
- Supports run-time reconfiguration

Tool available online for use

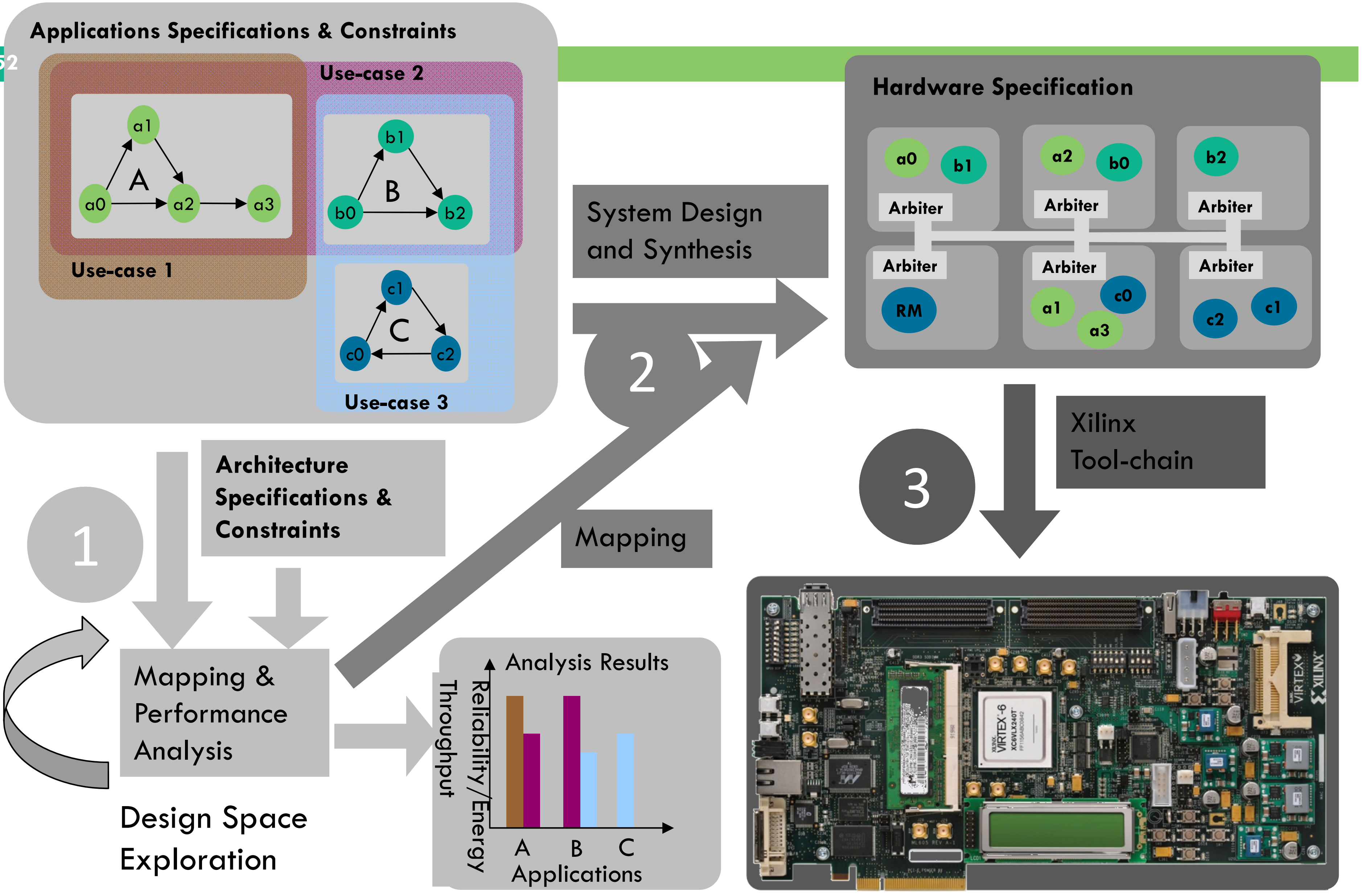
Generated a design with 100 Microblazes!!

Currently used by 20 research groups worldwide

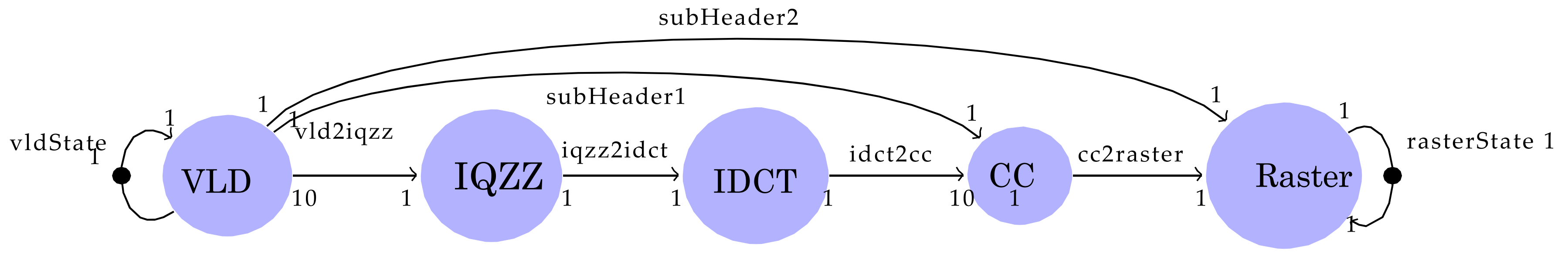




Predictable Design Flow



MJPEG Case Study

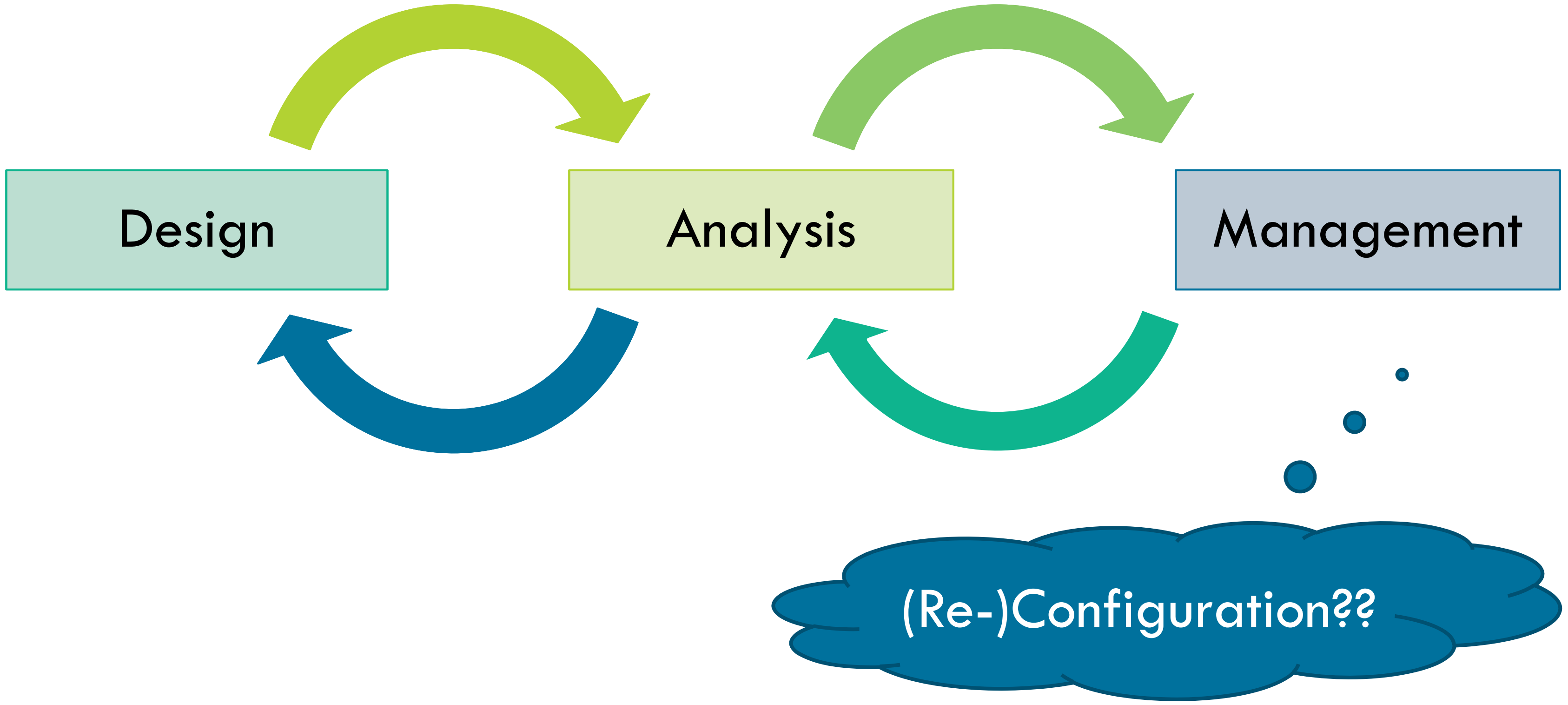


- One iteration decodes a single MCU (minimal coded unit)
- Each MCU consists of up to 10 blocks of frequency values
- WCET determined through measurement and scenario detection techniques

Designer Effort

Step	Time spent
Parallelizing the MJPEG code	< 3 days
Creating the SDF graph	5 minutes
Gathering required actor metrics	1 day
Creating application model	1 hour
Generating architecture model	1 second
Mapping the design (SDF3)	1 minute
Generating Xilinx project (MAMPS)	16 seconds
Synthesis of the system	17 minutes
Total time spent	**~ 4 days**

Design- and Run-time Flow



(Re-)Configuration??

- Determine which resource to use when

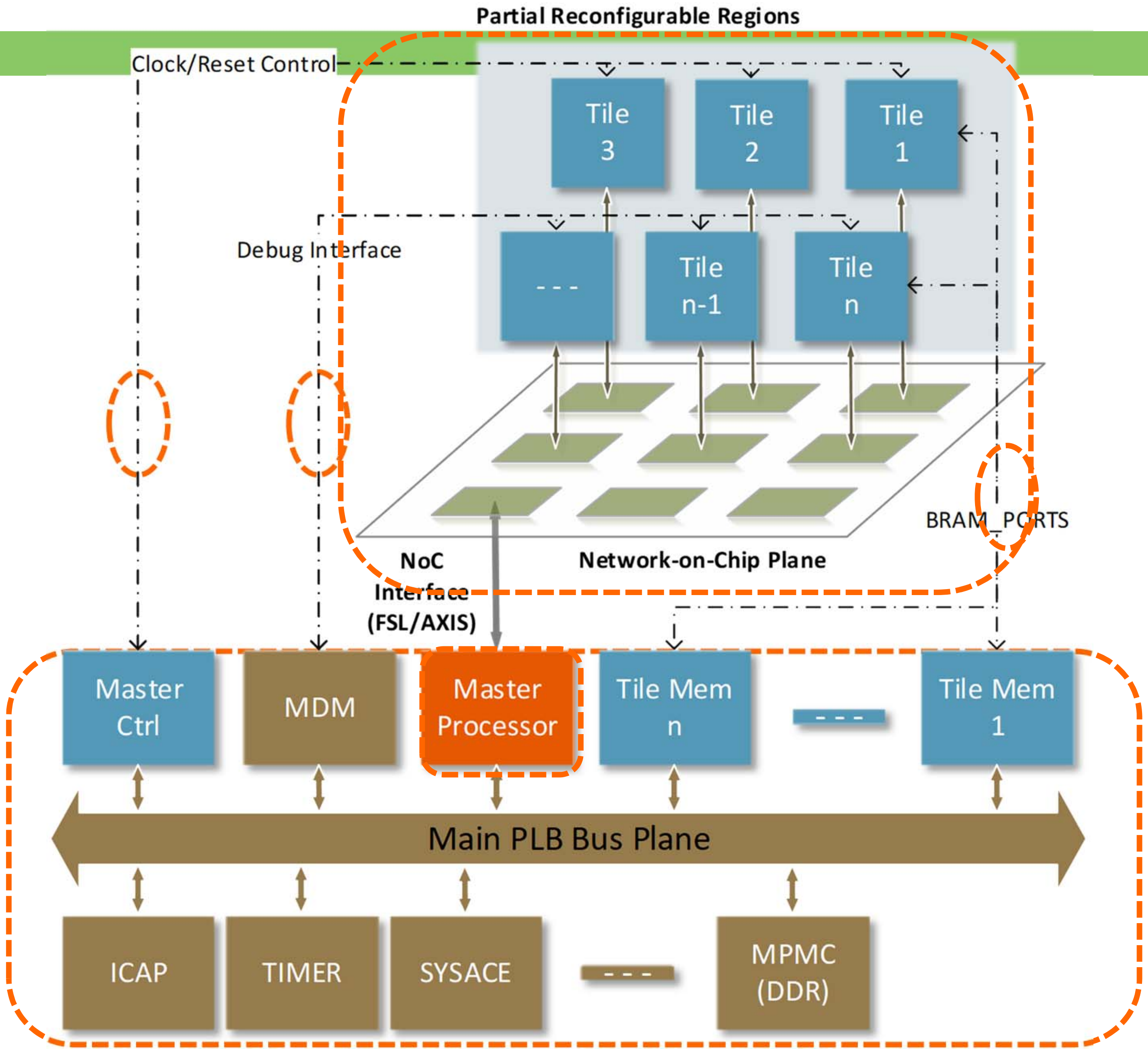
- Change the device types?
- Change the device functionality?
- Change the communication?
- Change the mapping
- Change the schedule

Reconfigurable Heterogeneous MPSoC

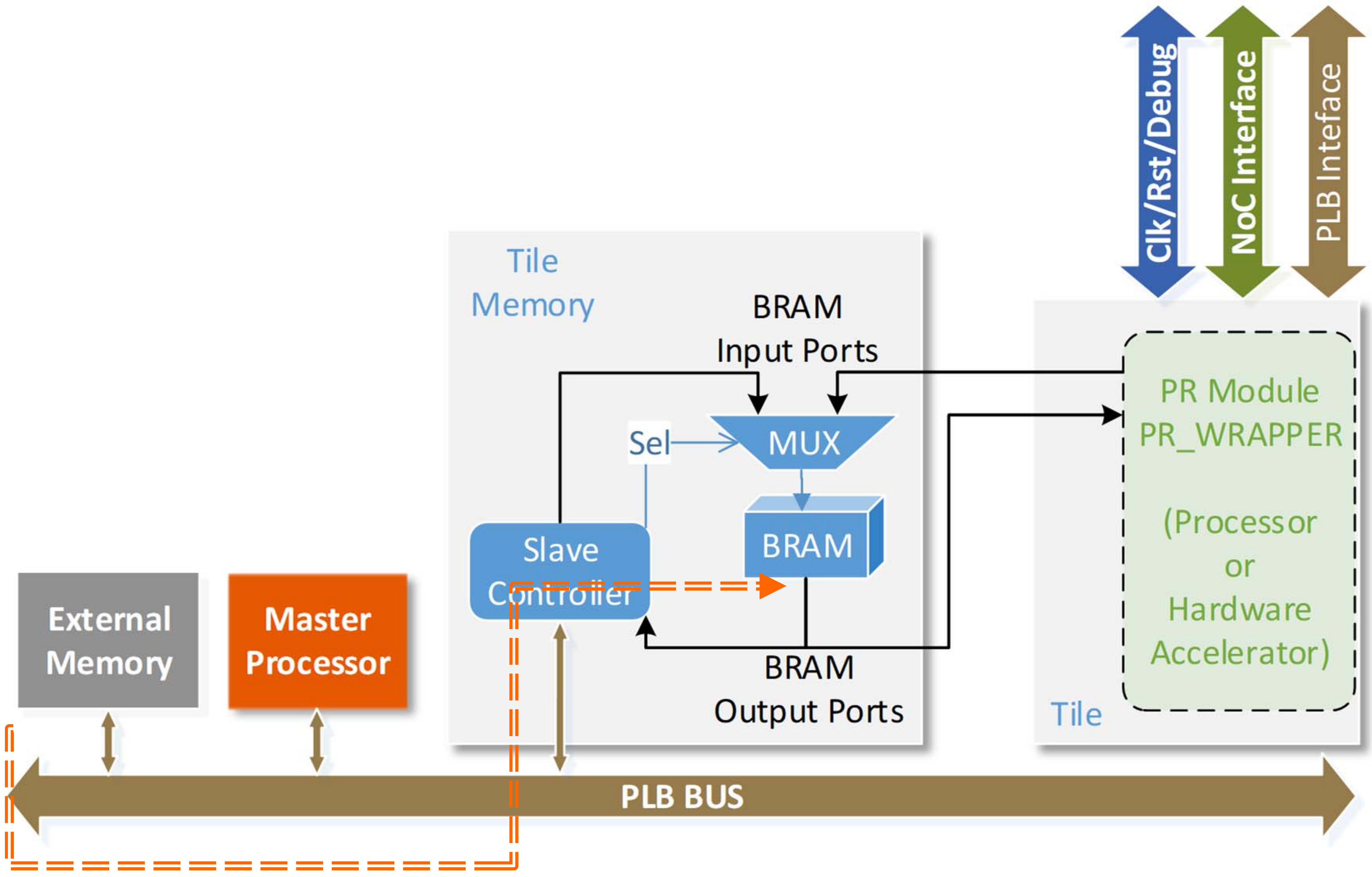
- Customizable at run-time depending upon the application requirements
- The tasks taking a long time in software can be accelerated by configuring the programmable tiles appropriately

- The reconfigurable tiles can be configured to achieve fault-tolerance as well
- Size and cost reduction by time-multiplexing the reconfigurable hardware

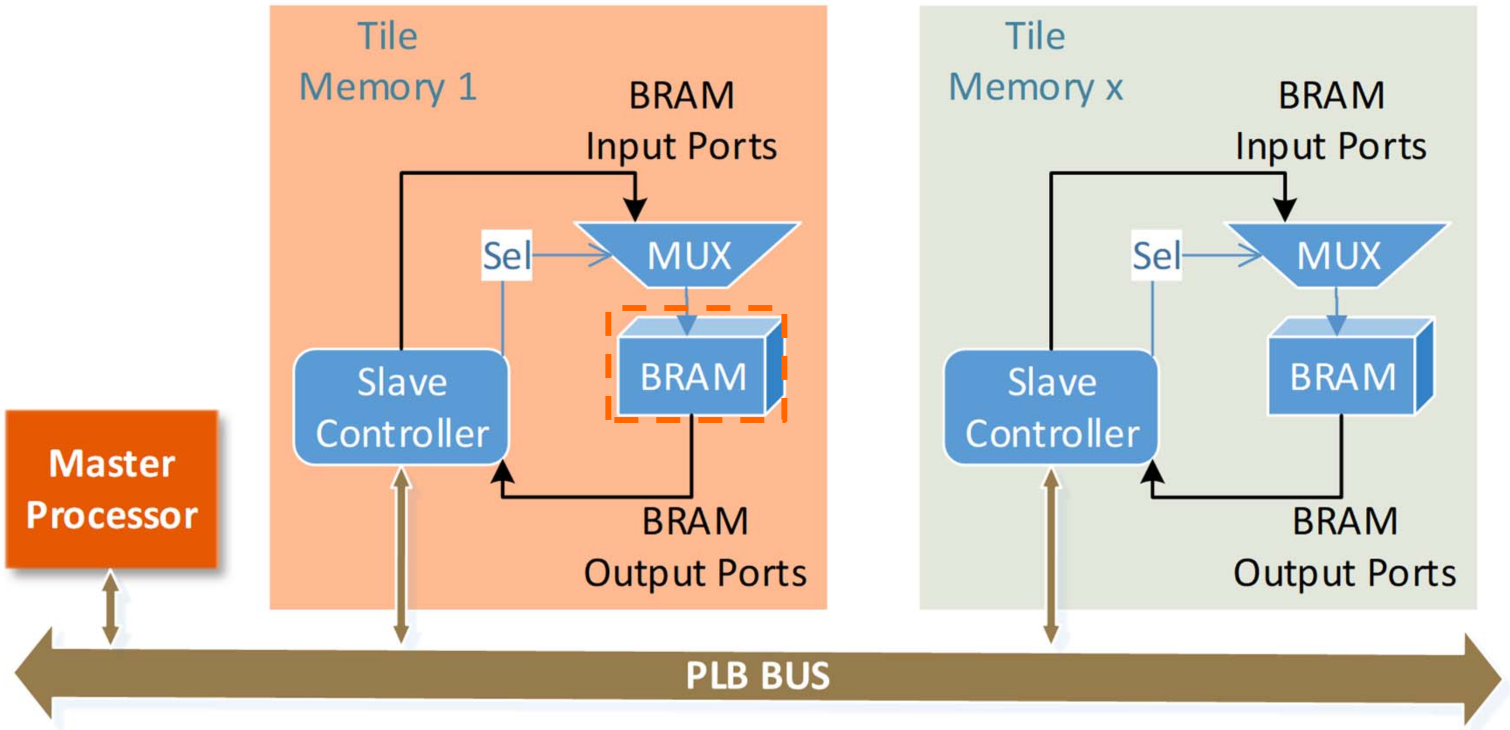
Partially Reconfigurable MPSoC



Loading Processor Executable Code at Run-time



Migrating Tasks

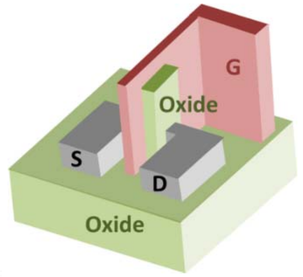


Modern Challenges

Issues and Modern Trends

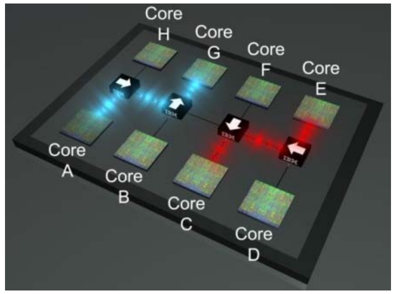
- The communication bottleneck
  - 3D Chips
  - Optical interconnects

- Leakage current limiting size reduction
  - Multi-gate or gate-all-around transistors (Intel 22nm uses 3D/tri-gate transistors)
  - Channel strain engineering, silicon-on-insulator-based technologies, and high-k/metal gate materials



- One may not fit all
  - Hardware/Software Co-design
  - Fault-tolerant / reconfigurable computing

- Power issues
  - Multi-core and heterogeneous architectures



Technology Scaling

- Dennard scaling principles [1]

Device Parameters	Scaling Factor
Device dimension	$1/k$
Doping concentration	$1/k$
Voltage	$1/k$
Current	$1/k$
Capacitance	$1/k$
Delay time per circuit	$1/k$
Power dissipation	$1/k^2$
Area	$1/k^2$
Power density	1

[1] R. Dennard et al. "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," IEEE Journal of Solid-State Circuits, 1974.

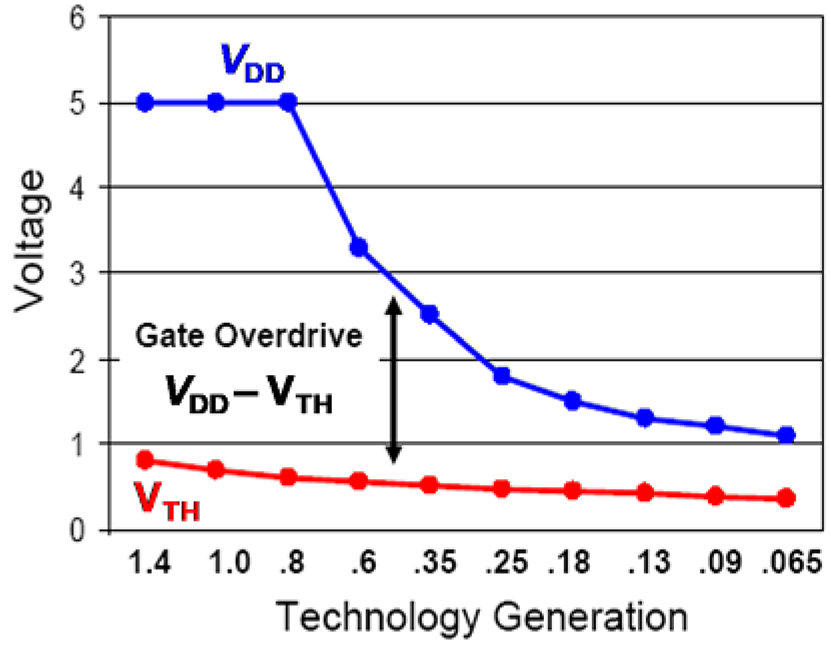
Technology Scaling

- Digression from Dennard’s scaling beyond 65nm
  - Non-ideal voltage scaling: limit on threshold voltage scaling
  - Non-ideal gate oxide scaling
  - Sub-threshold leakage power

- Power dissipation increases with technology scaling
  - Heat localization (hot spots)
  - Higher temperature => device wear-out

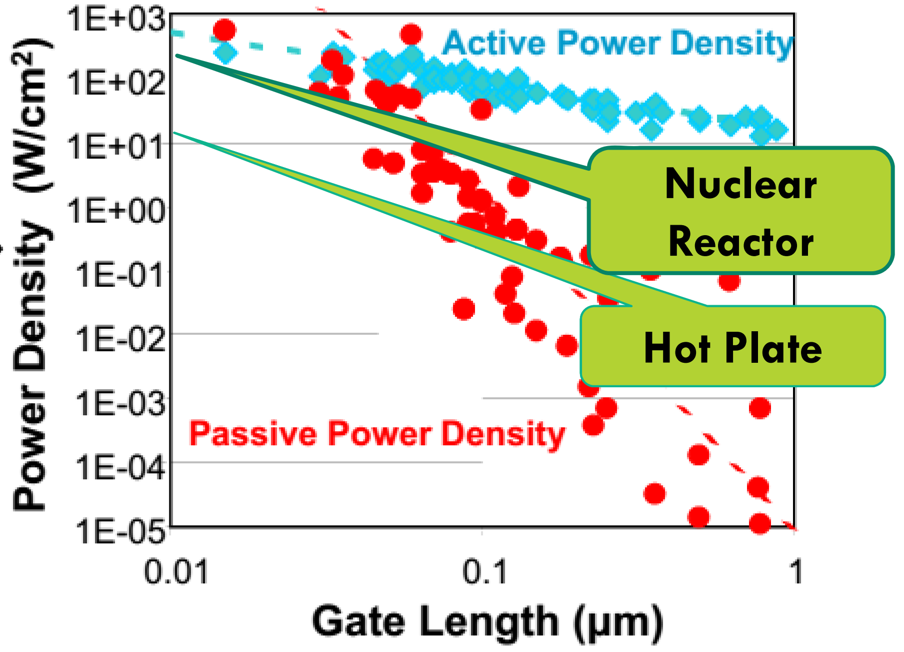
Technology Scaling and Power Density

CMOS Voltage Scaling



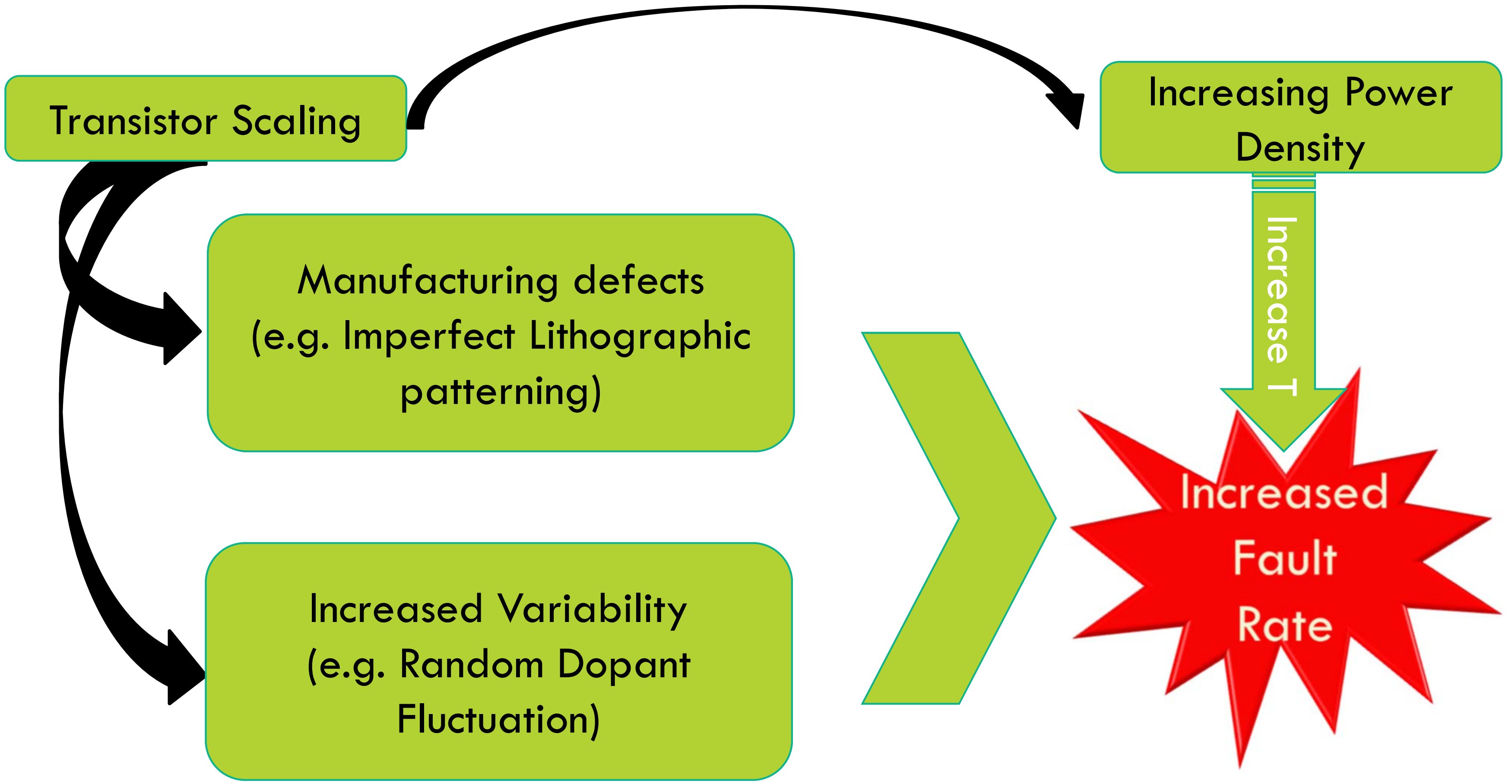
Source: P. Packan (Intel), 2007 IEDM Short Course

Power Density vs. Gate Length

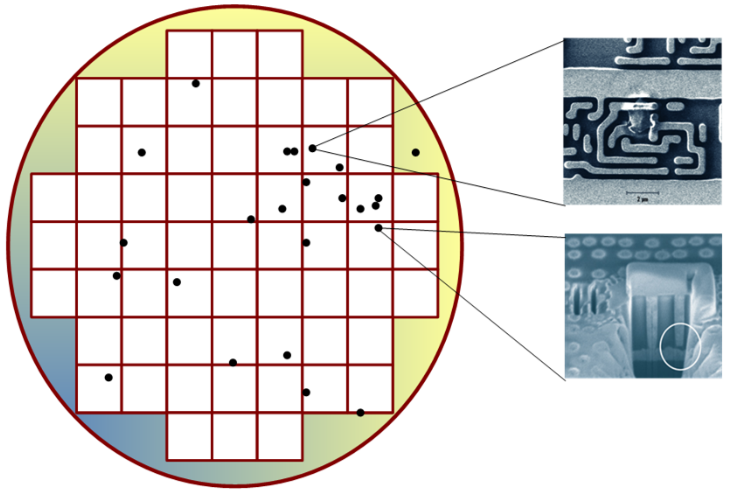


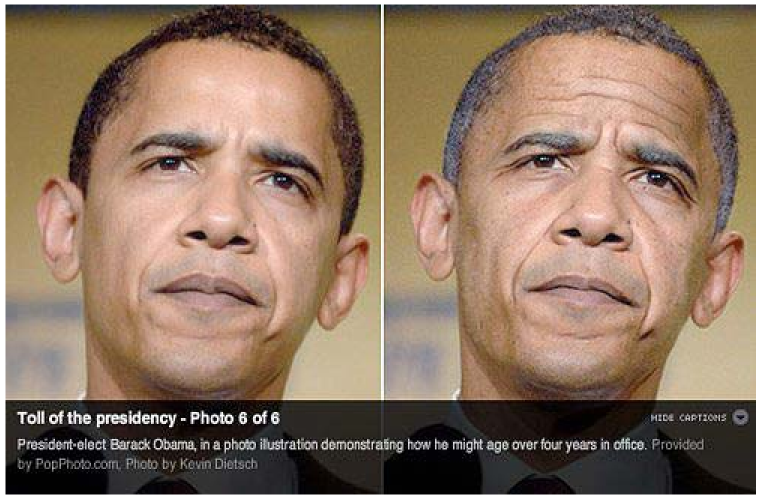
Source: B. Meyerson (IBM) Semico Conf., January 2004

Technology Scaling and Power Density


Transistor Scaling
Increasing Power Density
Increase T
Manufacturing defects (e.g. Imperfect Lithographic patterning)
Increased Variability (e.g. Random Dopant Fluctuation)
Increased Fault Rate

What cause Faults?





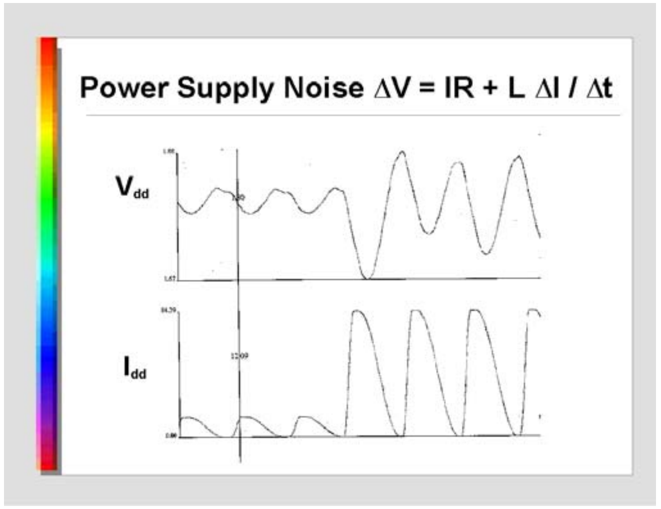
Toll of the presidency - Photo 6 of 6

President-elect Barack Obama, in a photo illustration demonstrating how he might age over four years in office. Provided by PopPhoto.com. Photo by Kevin Dietsch

Manufacturing Defects

Aging
(a.k.a., Circuit Wearout)

What causes Faults?

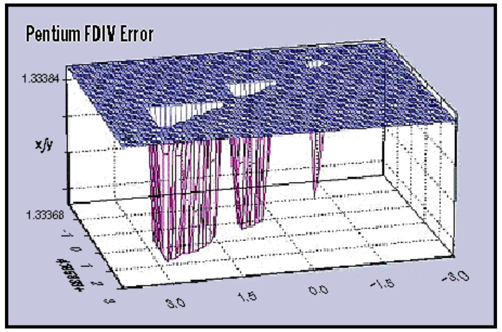


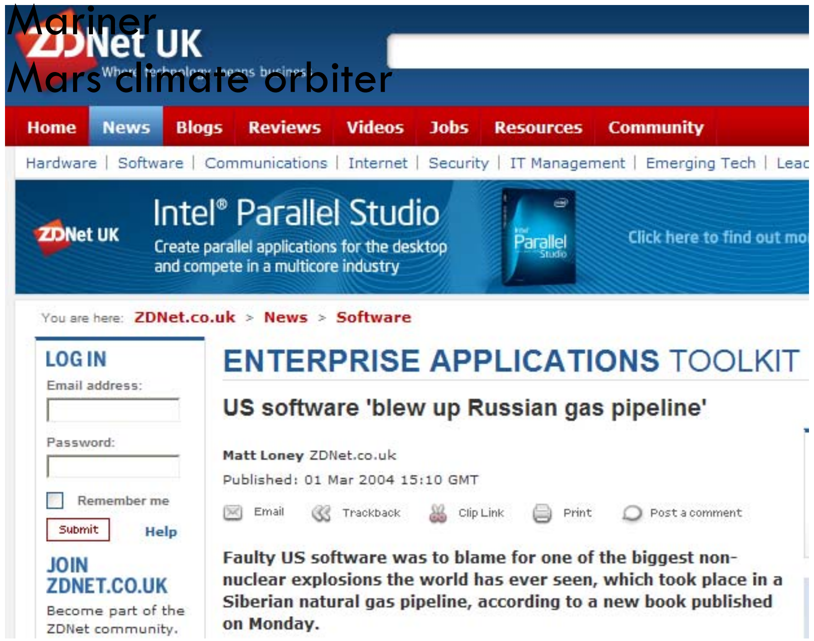
Internal Electronic Noise

Electromagnetic Interference

What cause Faults?

1962: Mariner
1998: Mars climate orbiter




ZDNet UK
Where technology means business

Home | News | Blogs | Reviews | Videos | Jobs | Resources | Community

Hardware | Software | Communications | Internet | Security | IT Management | Emerging Tech | Lead

ZDNet UK Intel® Parallel Studio Create parallel applications for the desktop and compete in a multicore industry Click here to find out mo

You are here: ZDNet.co.uk > News > Software

LOG IN
Email address:
Password:
Remember me
Submit Help

JOIN ZDNET.CO.UK
Become part of the ZDNet community.

ENTERPRISE APPLICATIONS TOOLKIT

US software 'blew up Russian gas pipeline'

Matt Loney ZDNet.co.uk
Published: 01 Mar 2004 15:10 GMT

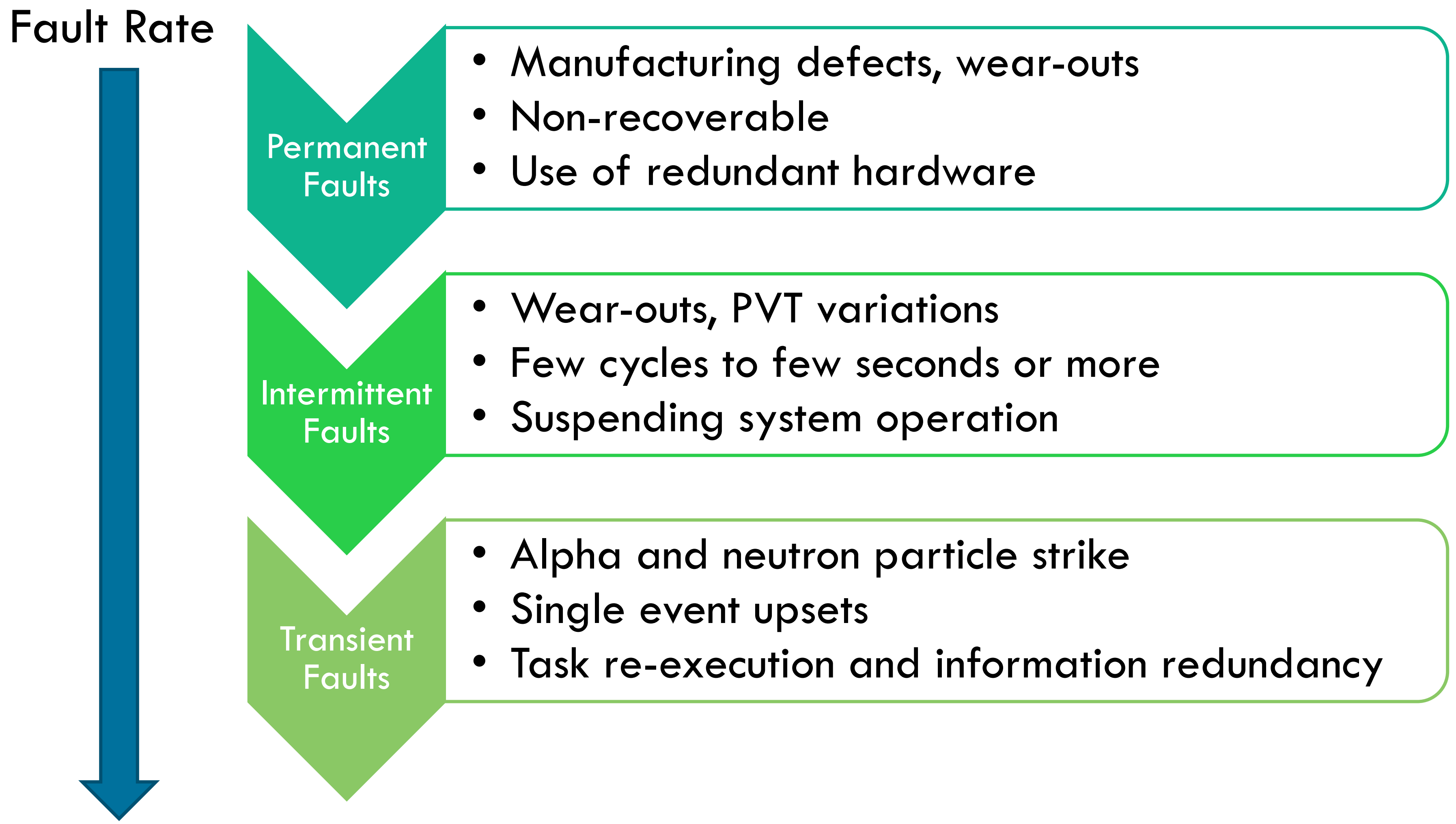
Email Trackback Clip Link Print Post a comment

Faulty US software was to blame for one of the biggest non-nuclear explosions the world has ever seen, which took place in a Siberian natural gas pipeline, according to a new book published on Monday.

Bugs

Malicious attack

Fault Classification

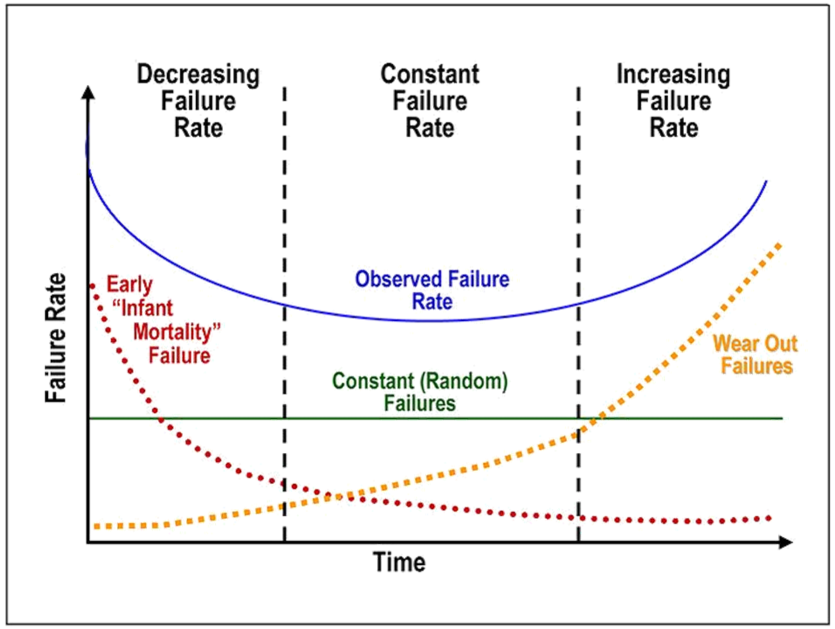


- Manufacturing defects, wear-outs
- Non-recoverable
- Use of redundant hardware

- Wear-outs, PVT variations
- Few cycles to few seconds or more
- Suspending system operation

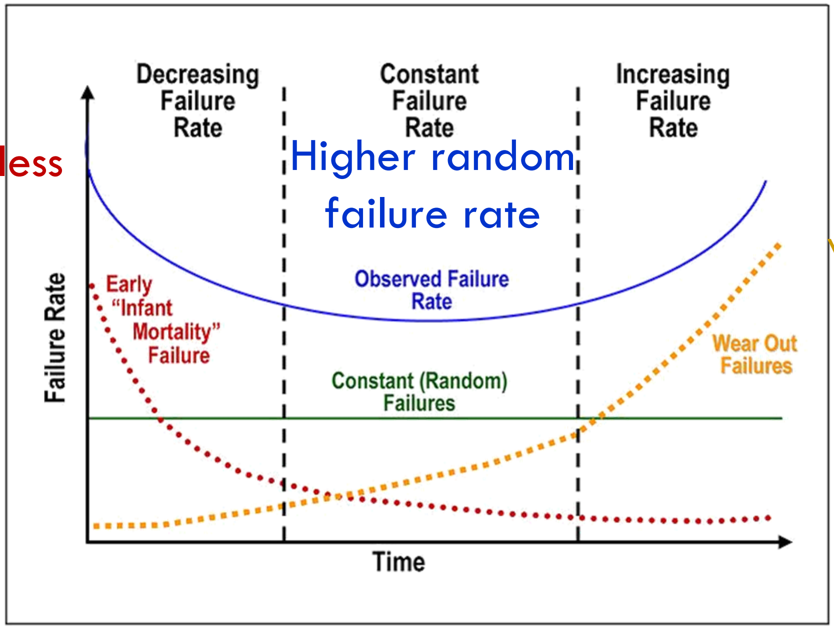
- Alpha and neutron particle strike
- Single event upsets
- Task re-execution and information redundancy

Failures during Lifetime



- Three phases of system lifetime
  - Infant mortality (imperfect test, weak components)
  - Normal lifetime (transient/intermittent faults)
  - Wear-out period (circuit aging)

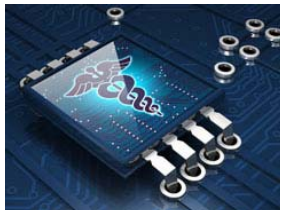
The Impact of Technology Scaling



- More leakage
- More process variability
- Smaller critical charges
  - Trends show soft-error rates incr. exp., 8% per tech generation
- Weaker transistors and wires

Effect on Embedded systems

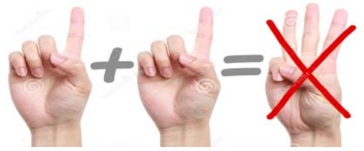
- Decreased Lifetime:
  - Mission failures
  - Reduced safety in critical systems
    - Power plants, transportation, medical etc.
  - Reduced product lifetime

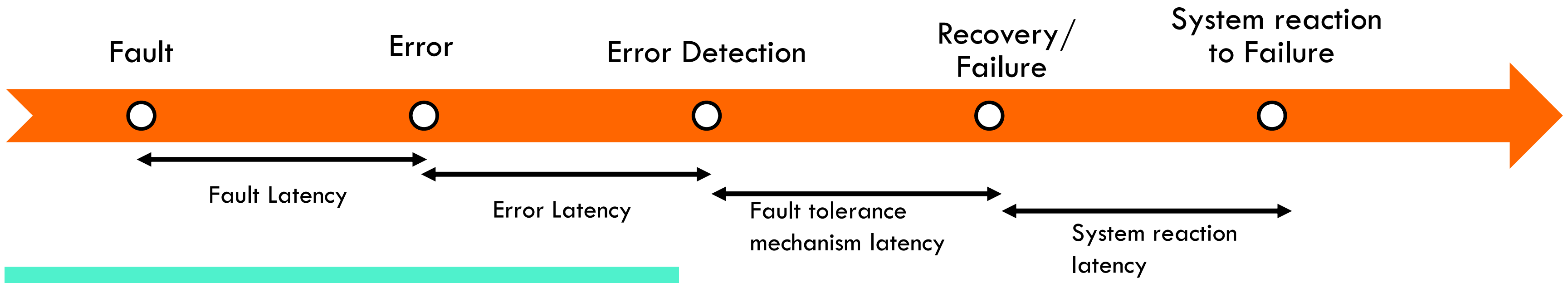


Effect on Embedded systems

- Soft errors:
  - Direct effect on reliability
    - Functional reliability
    - Timing reliability
  - Indirect effect
    - Mitigation methods lead to faster aging

Data Corruption


Computation errors



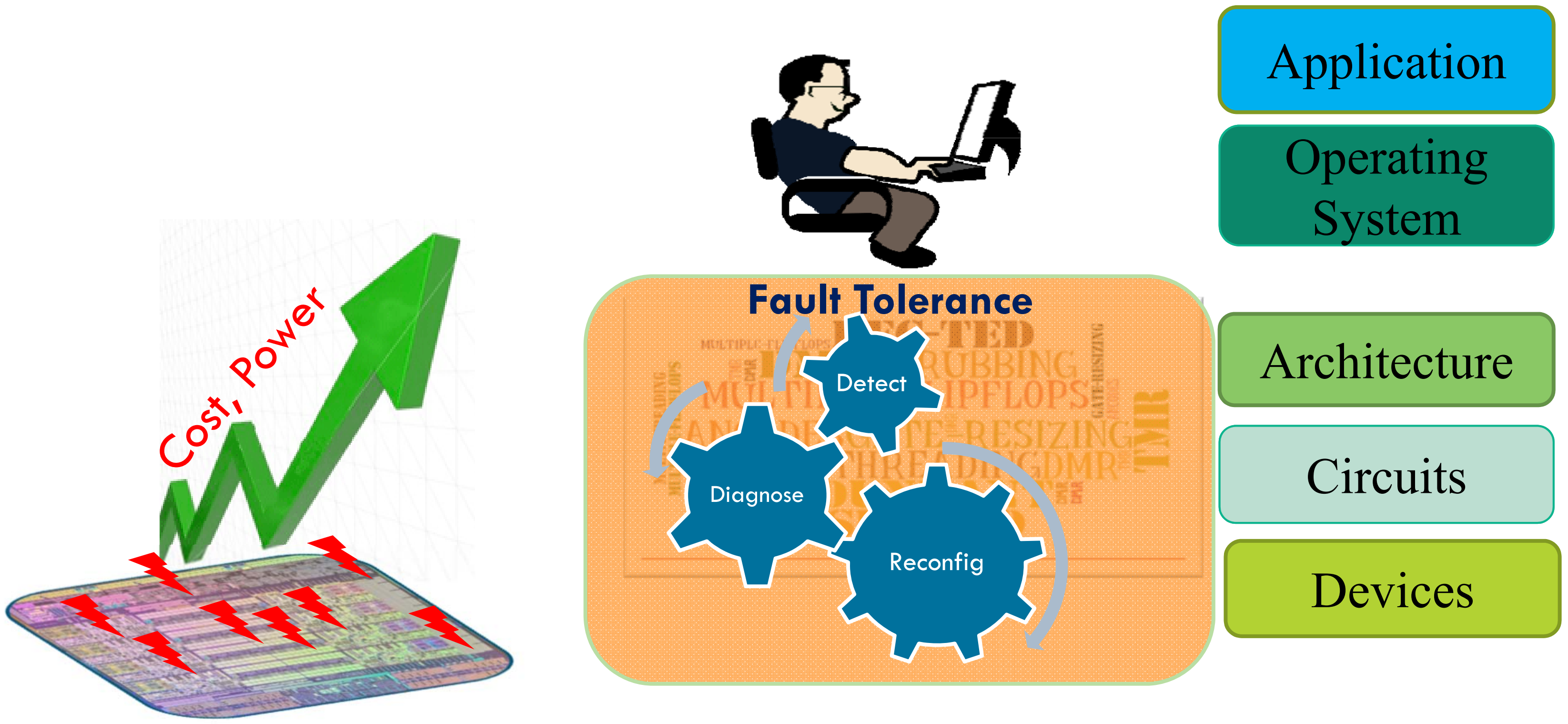
Fault Tolerance Timing Overheads

Fault-Aware System Design

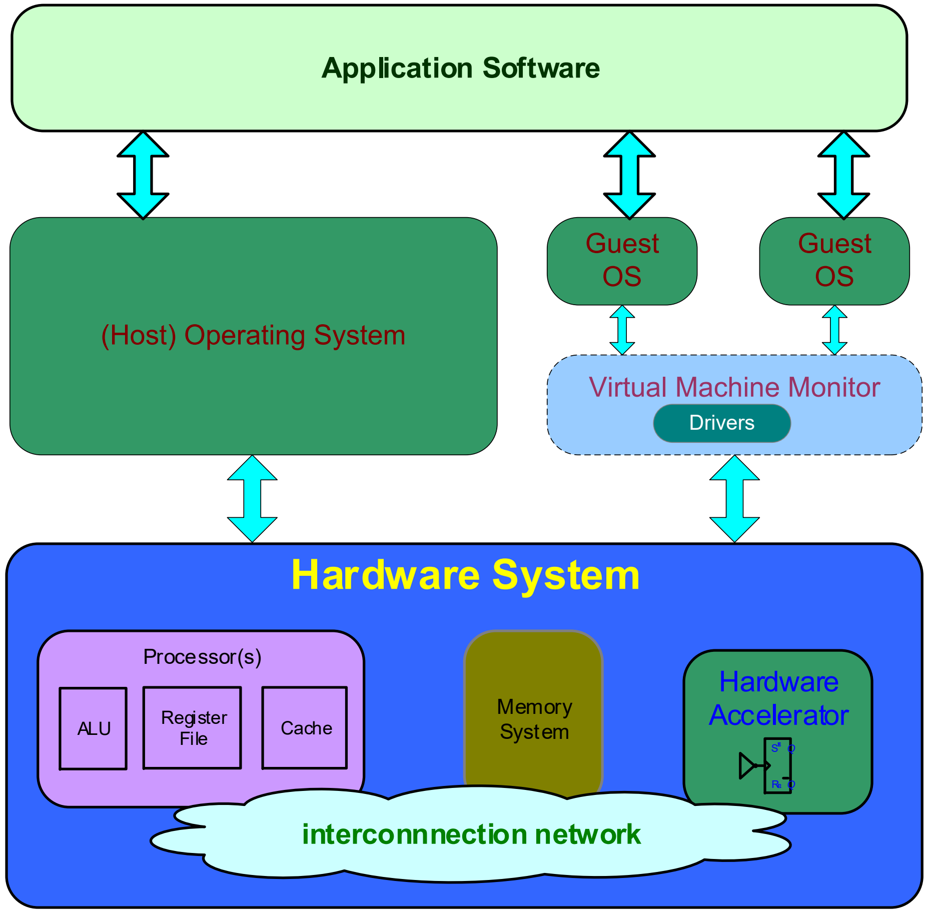
- Faults are inevitable…..learn to live with faults !!!
- How to address them??
  - Fault prevention
  - Fault tolerance
  - Fault removal
  - Fault forecasting

Single-layer Fault tolerance

- The usual "phenomenon-based" approach
- Provide a "perfect" hardware to upper layers



Levels of Fault Tolerance



Software redundancy

Virtualization
Task migration
Redundant multithreading
Fault-tolerant scheduling

Core-level redundancy
TMR/ DWC
Dynamic verification & correction
Block-level redundancy
ECC for memory
Circuit hardening

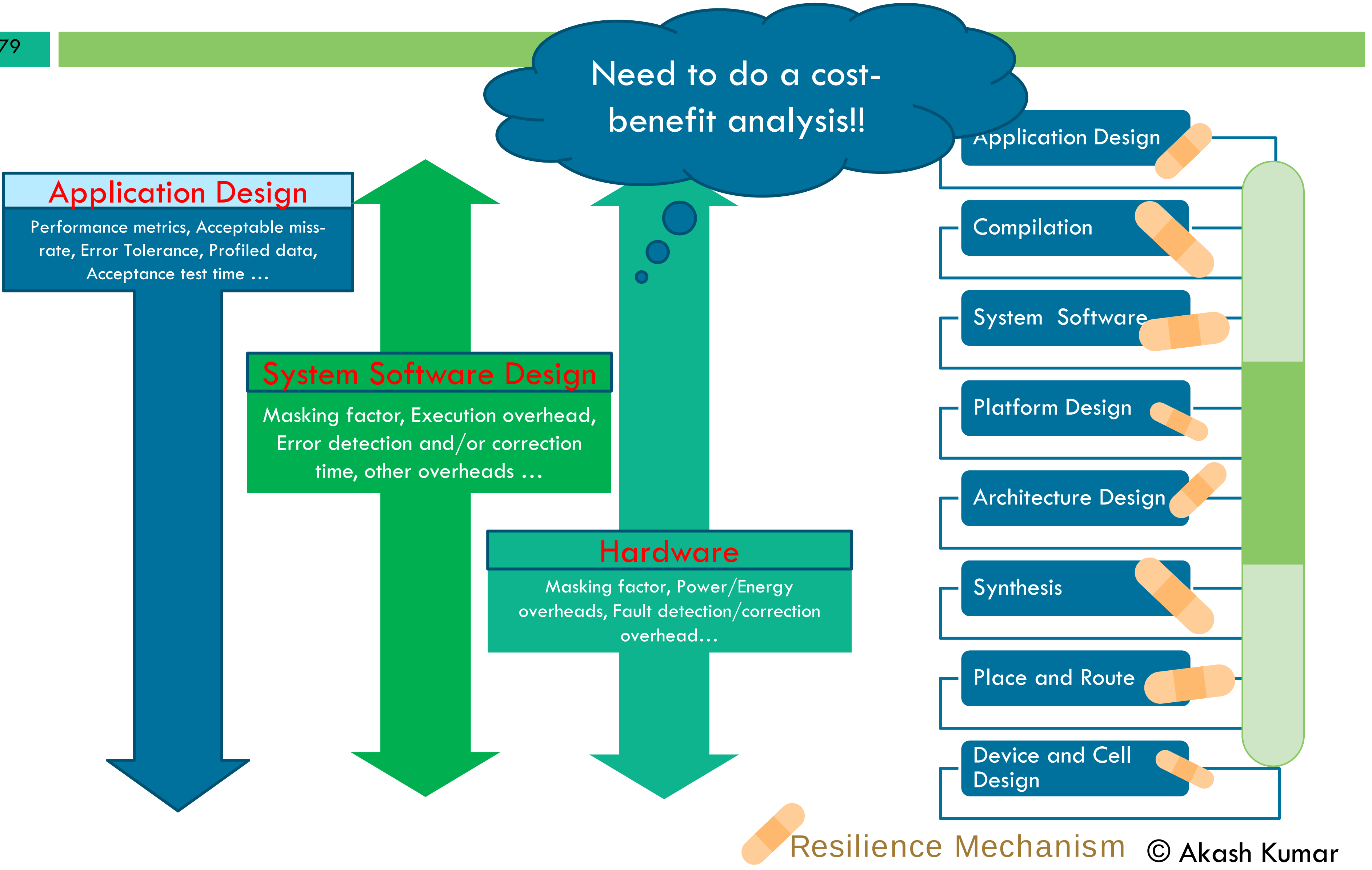
Application areas and requirements

- Variation

Not all applications require the same level of reliability

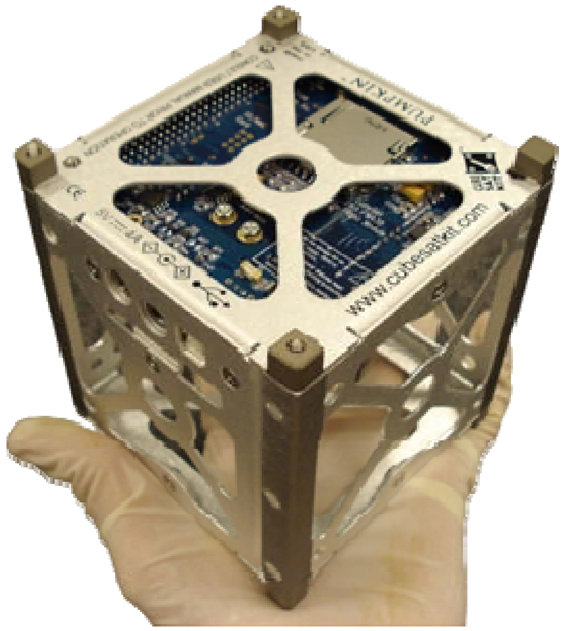
Application Area	Priority of reliability requirements		Other relevant metrics
	Functional Reliability	Timing Reliability	
Banking	High	Medium	
Multimedia	Medium	High	Throughput
Portable multimedia			Throughput, Energy
Health monitoring	High	Medium ~ High	Energy, Lifetime
Satellites / Space Missions	Medium	Medium ~ High	Lifetime

Cross-layer Approach


Need to do a cost-benefit analysis!!
Application Design
Performance metrics, Acceptable miss-rate, Error Tolerance, Profiled data, Acceptance test time …
System Software Design
Masking factor, Execution overhead, Error detection and/or correction time, other overheads …
Hardware
Masking factor, Power/Energy overheads, Fault detection/correction overhead…
Application Design
Compilation
System Software
Platform Design
Architecture Design
Synthesis
Place and Route
Device and Cell Design
Resilience Mechanism

Case-Study – Nanosatellites

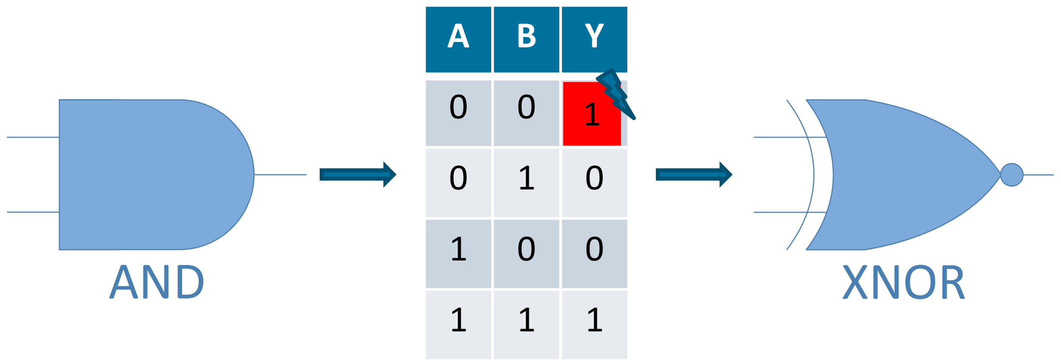
- Light-weight: Wet mass of 1-10kg
- Small satellites: Notion of cube-sats, 1U=10x10x10
- Increasingly being used as they are cheaper to design and launch
  - 2004-2013: 75 launches in total
  - 2014 Q1: 94 launces
- Typically low earth orbit
- Satellite swarms are also used



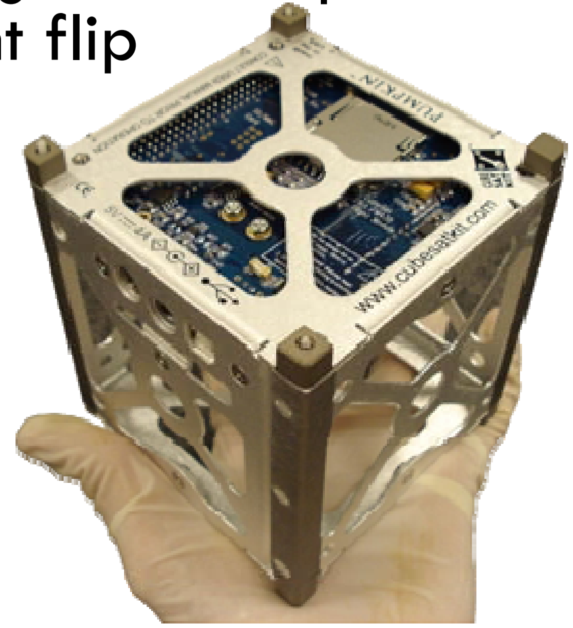
CubeSat – University of Liege

Case-Study – Nanosatellites

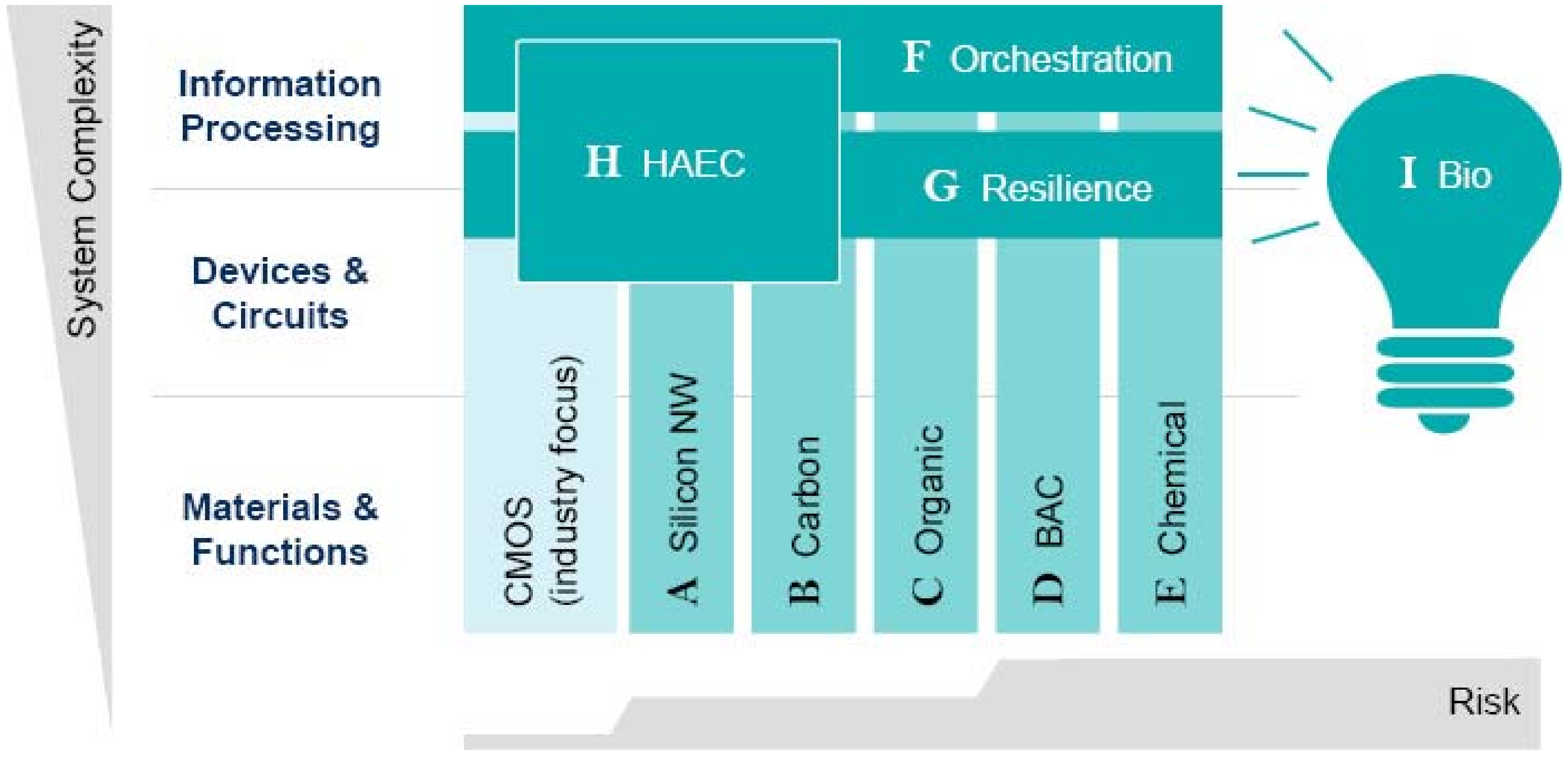
- FPGA use increasing in nanosats – lower price, faster development
- Nanosats affected by high energy particles in space leading to glitches
- Most common error in FPGAs– Single Event Upset (SEU) – a transient error that might flip configuration bits



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

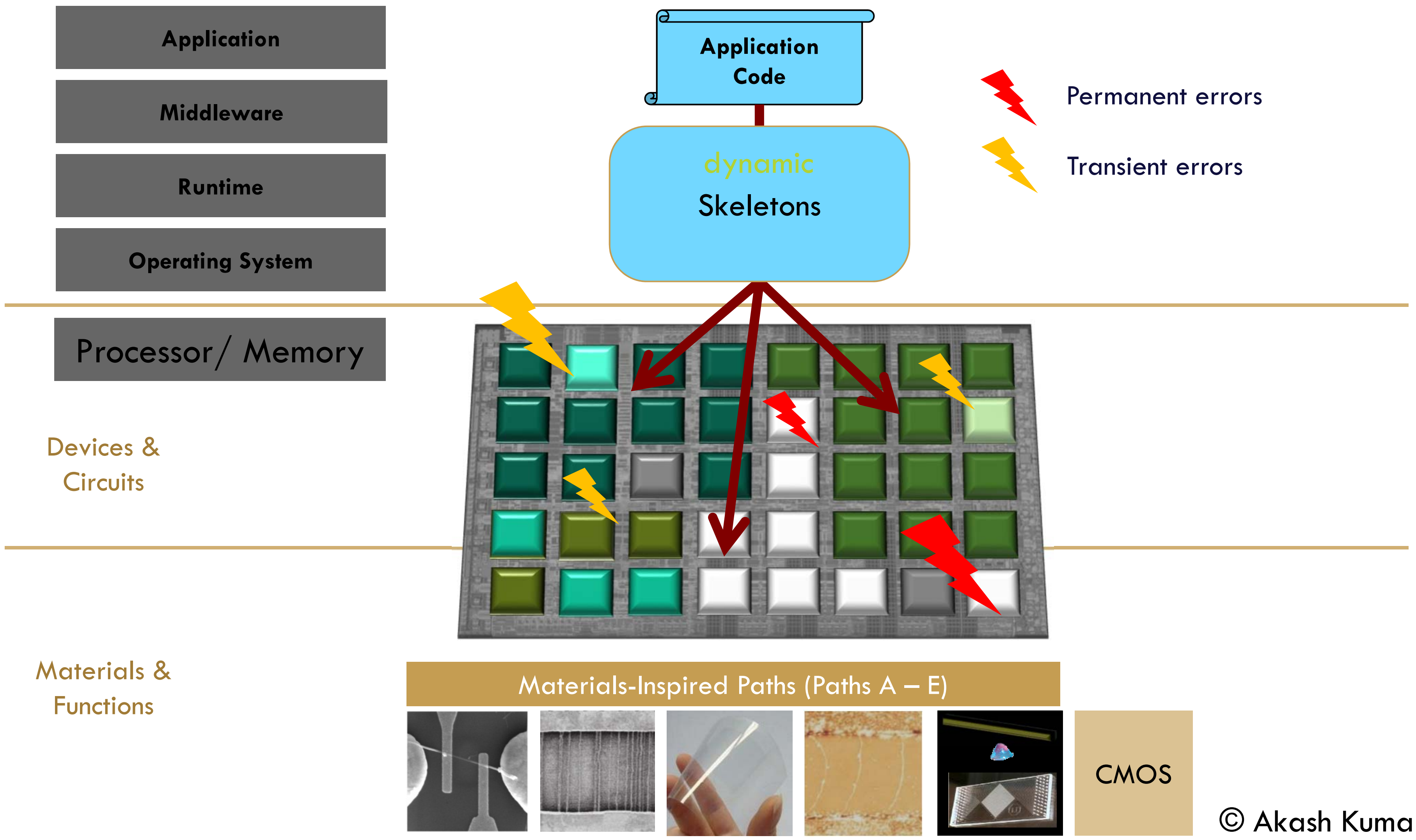

CubeSat – University of Liege

CFAED Paths

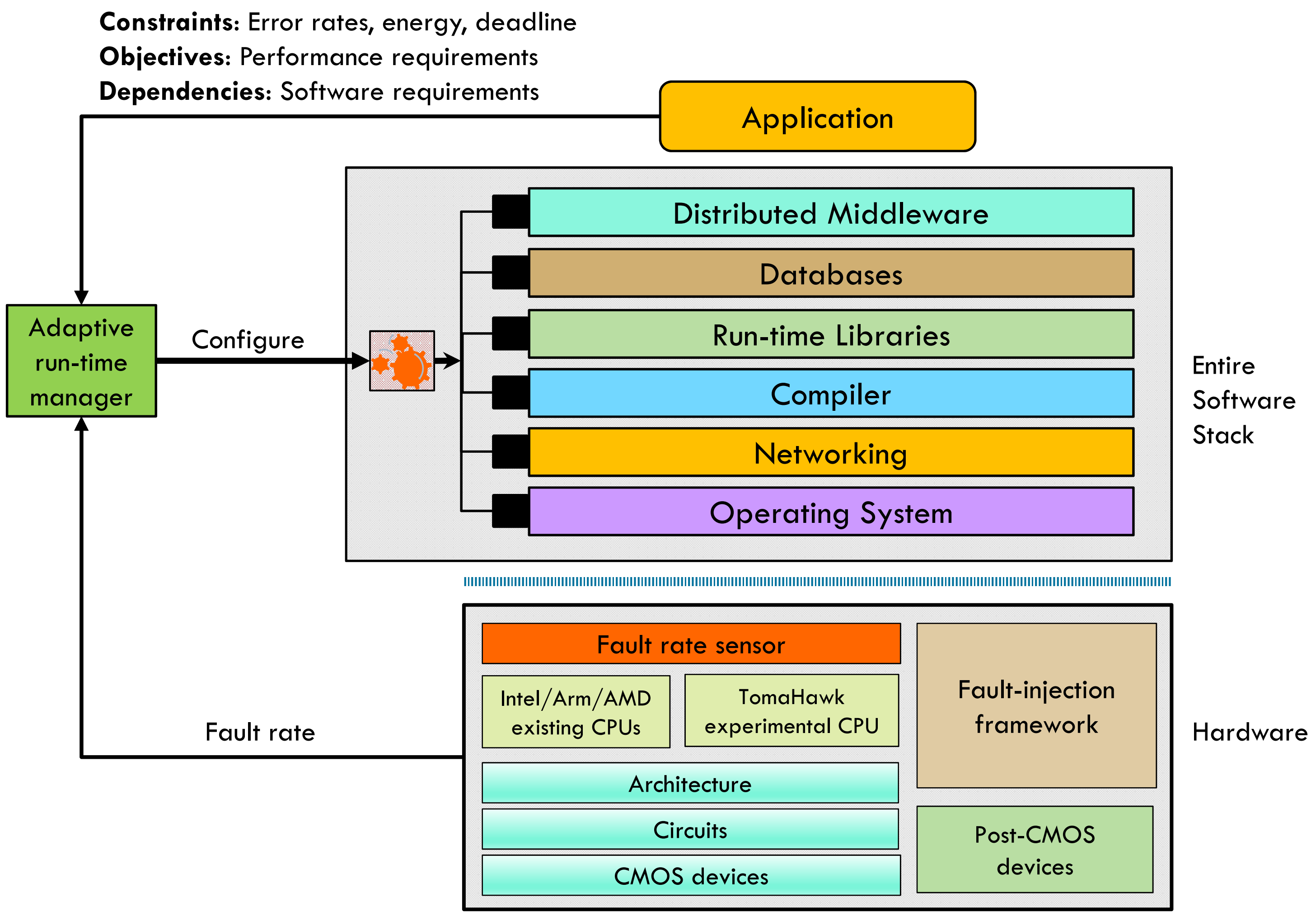

System Complexity
Information Processing
Devices & Circuits
Materials & Functions
H HAEC
F Orchestration
G Resilience
CMOS (industry focus)
A Silicon NW
B Carbon
C Organic
D BAC
E Chemical
I Bio
Risk

Path G: Resilience

- New technologies will have higher failure rates



Overview – Resilience at TU Dresden


Constraints: Error rates, energy, deadline
Objectives: Performance requirements
Dependencies: Software requirements
Application
Adaptive run-time manager
Configure
Distributed Middleware
Databases
Run-time Libraries
Compiler
Networking
Operating System
Entire Software Stack
Fault rate sensor
Intel/Arm/AMD existing CPUs
TomaHawk experimental CPU
Fault-injection framework
Architecture
Circuits
CMOS devices
Post-CMOS devices
Hardware
Fault rate

Approximate Computing

The Computational Efficiency Gap

~200000 W

20 W

20 W

IBM Watson playing Jeopardy, 2011

Humans Approximate

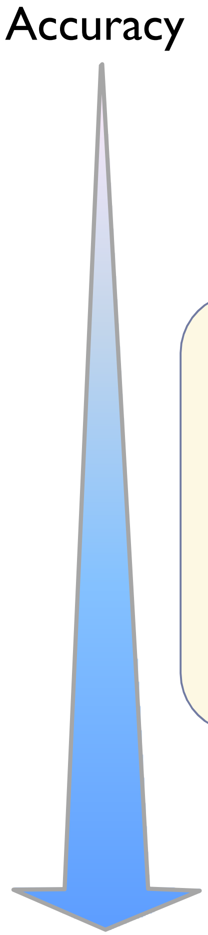
Task:
Division

$$is\ \frac{923}{21} > 1.75?$$

$$is\ \frac{923}{21} > 45?$$

```
21) 923 (43 ---
     84
   ------
      83
      63
      ¦
```

$$\frac{923}{21} = --.--?$$



~1Petaflop/W

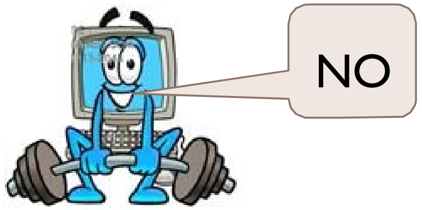
Application context dictates required accuracy of results

Effort expended increases with required accuracy

But Computers DO NOT

$$\frac{923}{21} > 45 \rightarrow$$

```
float x = 923;
float y = 21;
cout << (x/y > 45.0) ?
"YES":"NO";
```



$$\frac{923}{21} > 1.75 \rightarrow$$

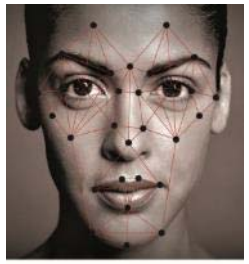
```
float x = 923;
float y = 21;
cout << (x/y > 1.75) ?
"YES":"NO";
```

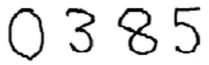
- Overkill (for many applications)
- Leads to inefficiency
- Can computers be more efficient by producing "just good enough" results?

Its an Approximate World … At the Top

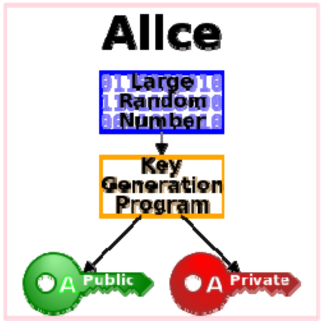
- No golden answer (multiple answers are equally acceptable)
  - Web search, recommendation systems

- Even the best algorithm cannot produce correct results all the time
  - Most recognition / machine learning problems





- Too expensive to produce fully correct or optimal results
  - Heuristic and probabilistic algorithms, relaxed consistency models, …



Miller-Rabin primality test

Eventual consistency

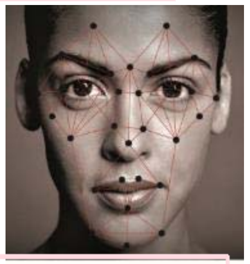
© Akash Kumar

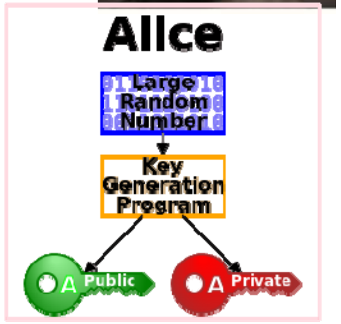
Its an Approximate World … At the Top

No golden answer

Perfect/correct answers not always possible

Too expensive to produce perfect/correct answers





Miller-Rabin primality test

Eventual consistency

Approximate Computing Throughout the Stack

No golden answer

≈

Perfect/correct answers not always possible

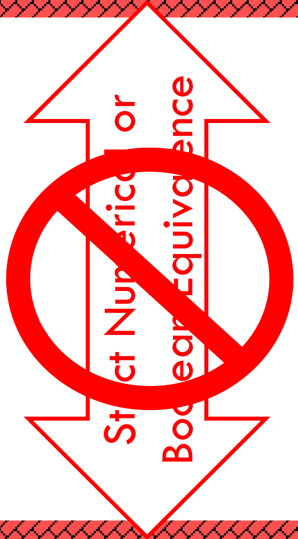
≈

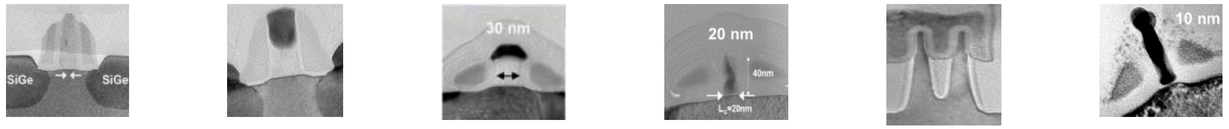
Too expensive to produce perfect/correct answers

Alice
Large Random Number
Key Generation Program
Public
Private

≈

Programming Languages, Compilers, Runtimes
Architecture
Logic
Circuits

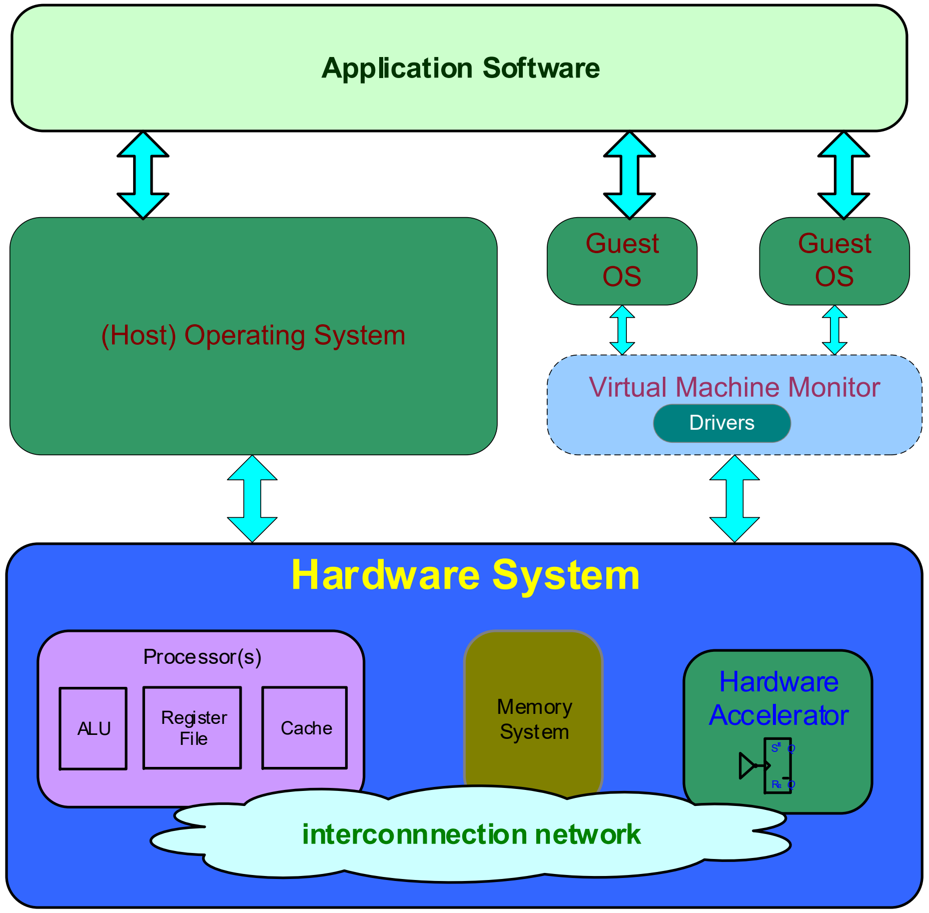




Approximation in System Design

- Arising from the application level
  - Inherent lack of notion or ability for a single ‘correct’ answer
  - ‘Noisy’ or redundant real-world data
  - Perceptual limitations
- Arising from the transistor level
  - Increasing fault-rates
  - Increased effort/resource to achieve fault-tolerance

Approximation in System Design



Application Approximation

Program Analysis for variable approximation

Approximate computing systems/architectures

Approximate computing processors

Reconfigurable approximate modules

Approximate circuit design

Conclusions

- Transistor scaling leading to increased faults
- Designing systems to tolerate faults inevitable
- Need to handle faults at all levels of critical systems

- Applications often lack notion of a ‘correct’ result
- Immense need/potential to trade-off performance and energy consumed

Ongoing Research Activities

Reliability/Energy Optimization

- Reconfigurable approximate computing at run-time
- Optimize energy and reliability
- Minimize thermal cycling and peak temperature
- Task remapping and scheduling for dealing with faults

Processing Architecture Design

- Determine and design appropriate system architecture
- Design predictable components – network and communication assist
- Partially reconfigurable tile-based heterogeneous multiprocessor systems
- Task-migration module in hardware for predictable delay

Low-Power and Fault-Tolerant FPGA Designs

- Improving fault-tolerance of FPGA through LUT content manipulation
- Novel error-correction mechanisms for FPGAs
- Leakage-aware resource management techniques
- Electronic Design Automation – Place and Route for FPGAs

Chair for Processor Design

Questions and Answers

Email: *akash.kumar@tu-dresden.de*