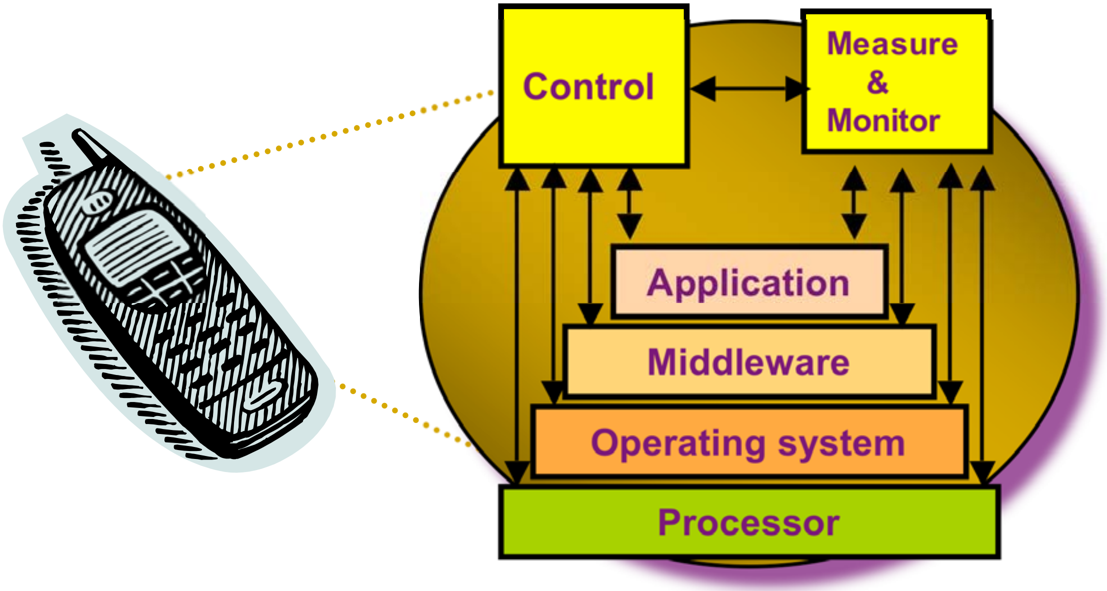
Adaptive Application-Aware Runtime Checking

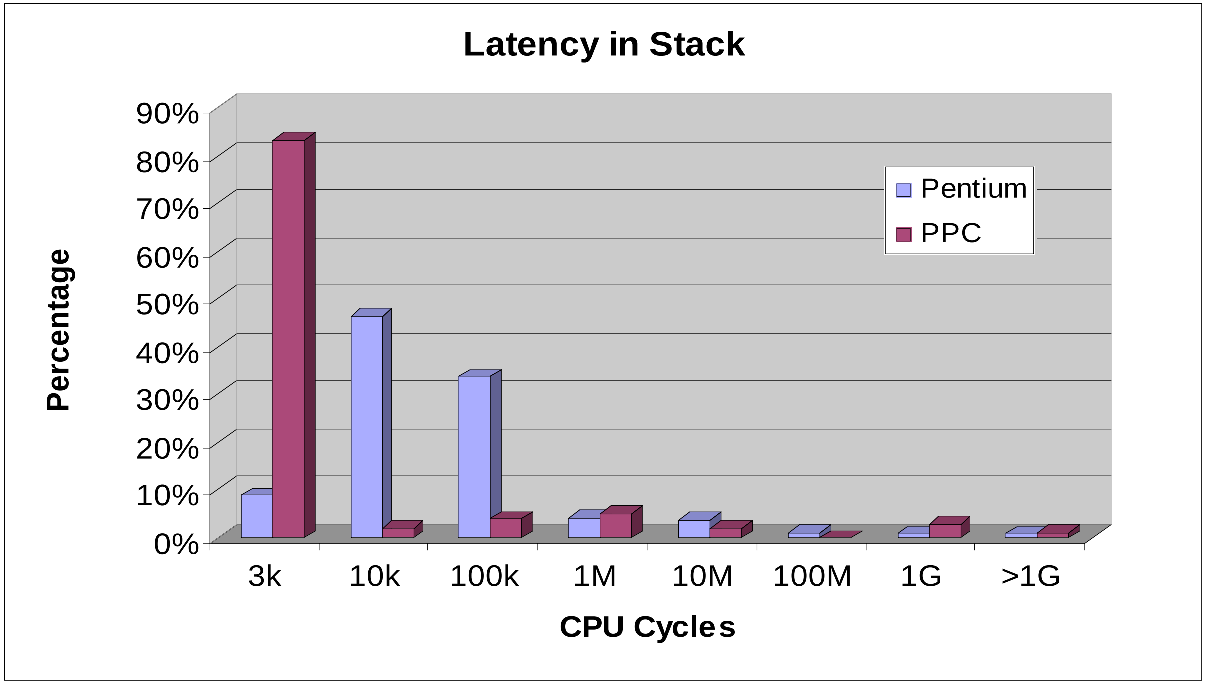
Ravi Iyer, Z. Kalbarczyk, N. Nakka, L. Wang, N. Breems et. al
Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
www.crhc.uiuc.edu/DEPEND

The Embedded Environment: Cell Phones



- Modular design of processes lends itself well to small footprint solutions.
- Specialized Applications optimized for memory/performance requirements.
- Specialized/Customized kernels

Crash Latency
Stack Injection *(Linux on Pentium and PowerPC)*



Early detection of kernel stack overflow on PPC major contributor to reduced crash latency

What is Needed?

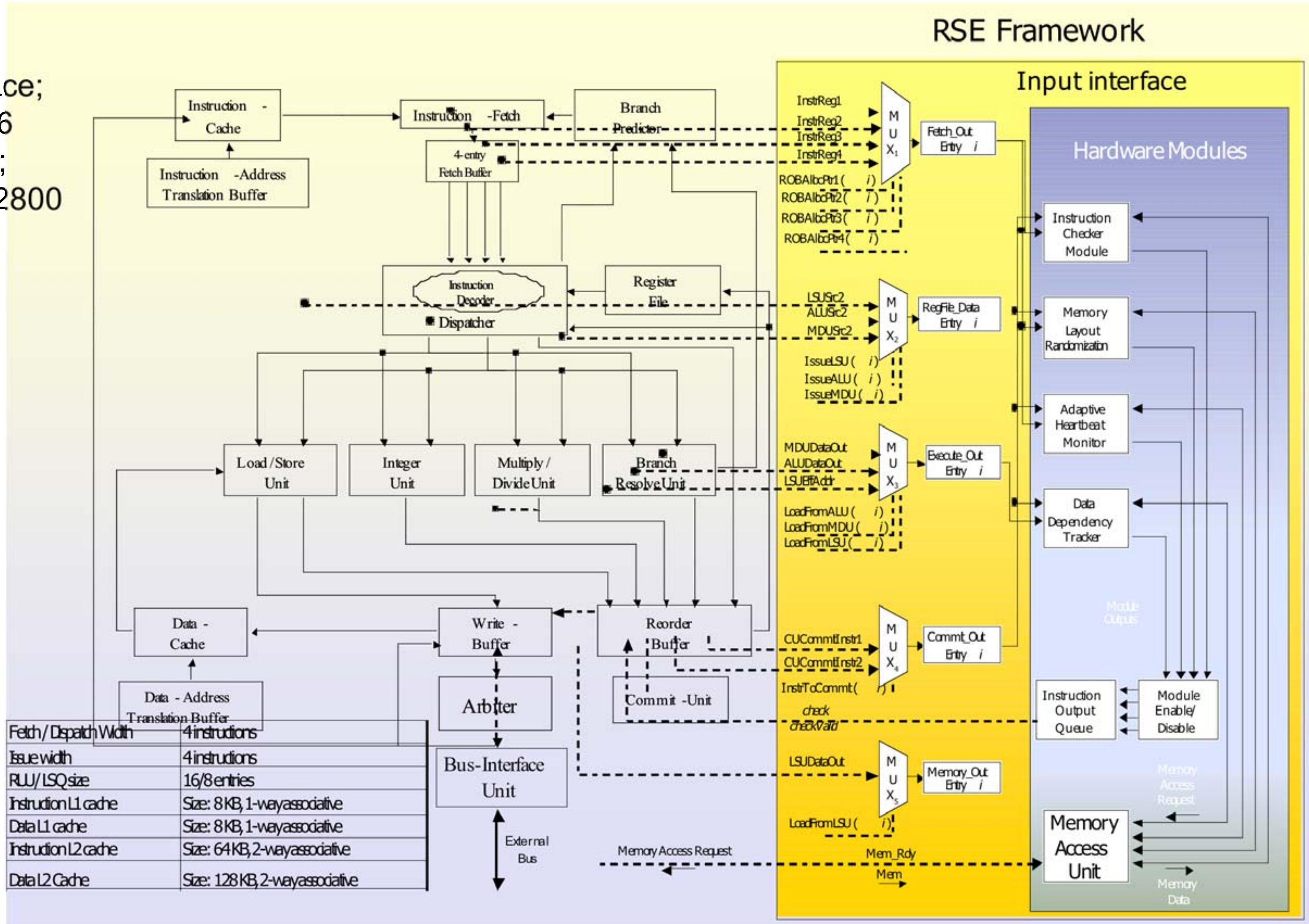
- A hardware/software framework that adapts dynamically to application needs
- Extracting application properties that can be used as an indicator of correct behavior and to drive synthesis of application-aware checks
- Instantiating the optimal hardware/software for runtime application checking
- Embed the devised checks into the custom hardware or software middleware or operating system

Adaptive Application Aware Checking in Hardware: Basics

- Static source-code analysis and profiling provides
  - Which checkers to be used and at what points of application execution
  - Checkers are adapted to application
- Hardware modeling using HDL
- Synthesize modules into reconfigurable hardware framework
- Modules themselves are runtime reconfigurable

Adaptive Application Aware Checking in Hardware: Reliability and Security Engine

For Input Interface;
Queue Size = 16
32-bit regs = 80;
Gate Count = 12800



Fetch/Dispatch Width	4 instructions
Issue width	4 instructions
RUU/LSQ size	16/8 entries
Instruction L1 cache	Size: 8KB, 1-way associative
Data L1 cache	Size: 8KB, 1-way associative
Instruction L2 cache	Size: 64KB, 2-way associative
Data L2 Cache	Size: 128KB, 2-way associative

N. Nakka, J. Xu, Z. Kalbarczyk, R. K. Iyer, "An Architectural Framework for Providing Reliability and Security Support", DSN2004.

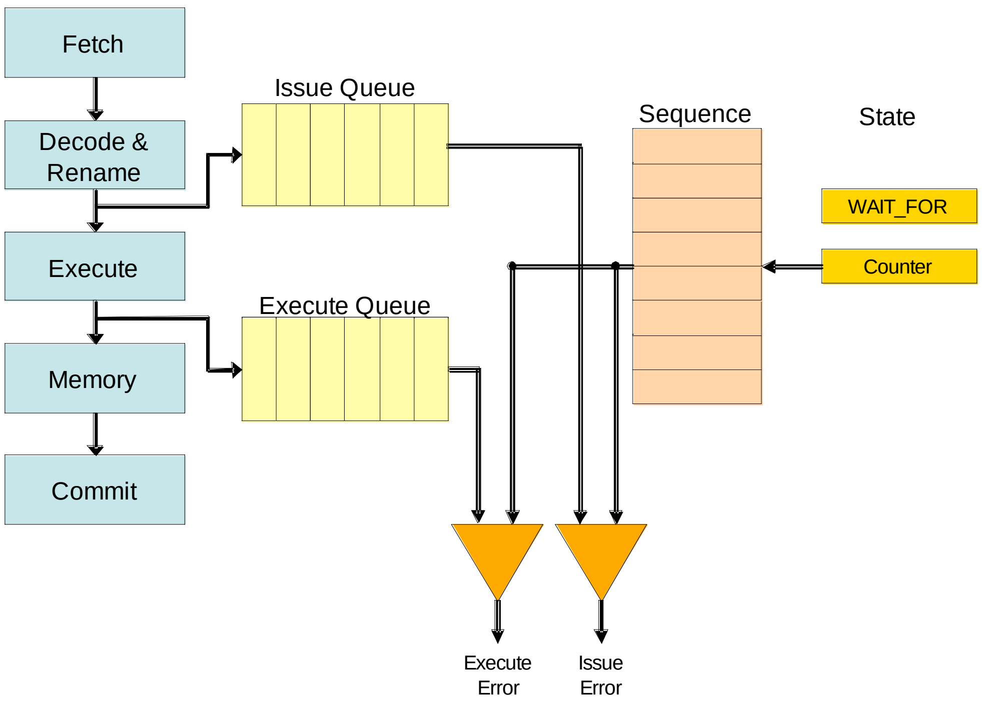
The Processor-Level Framework

- Implemented as an integral part of the processor on the same die
- Embeds hardware modules for reliability, security and recovery that execute in parallel with the instruction execution in the main pipeline
- Provides a generic interface to external processor system through which modules access runtime information for checking
- Application interfaces to framework through CHECK instructions
  - Extension of the ISA
  - Used by the application to invoke specific modules

Detection of Instruction Dependency Violations

- RAW dependency imposes sequential order on execution of instructions
- Errors in processor control logic, binary of instruction can lead to a violation
- Sequence Checker Module (SCM), detects such violations
  - monitors issue and execute events in pipeline
- Representative instruction sequences extracted using static analysis
- CHECKs used to dynamically reconfigure the module with sequences

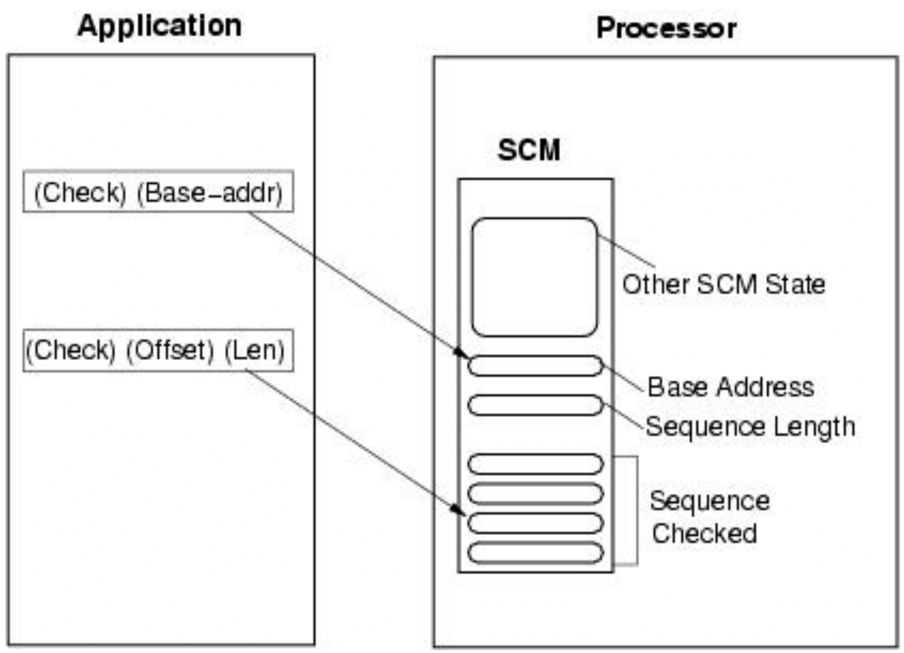
SCM Detection Mechanism



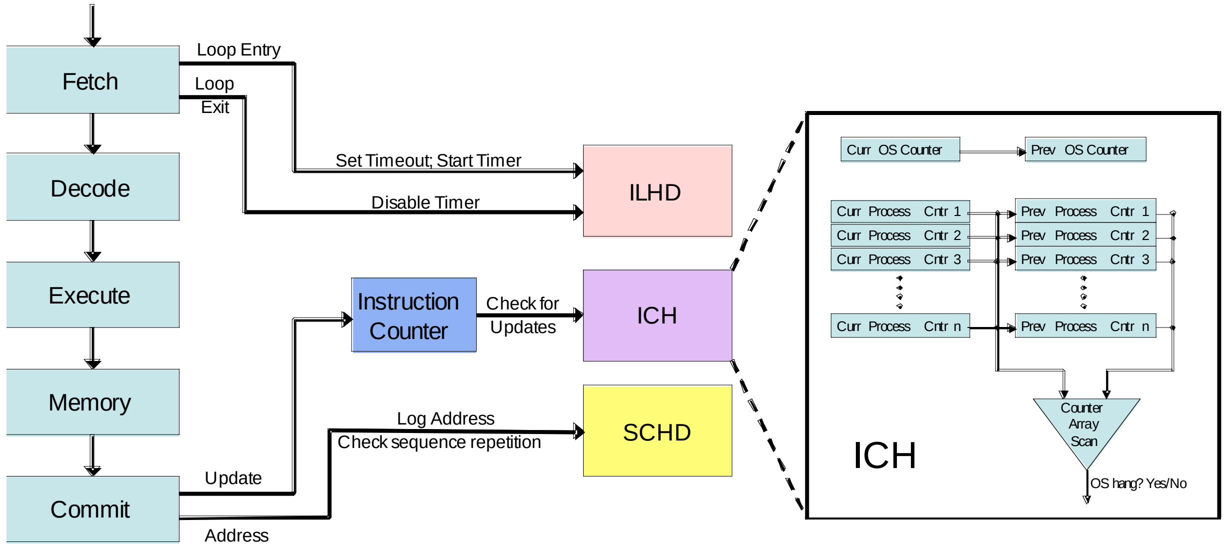
- SCM state for sequence – (*i*, *e*)
  - *i* : instruction on which event is awaited
  - *e* : event (issue/execute)
- Property – at any instance of time, *at most one* instruction of a dependent sequence can be issued or executed
- Instructions in issue and execute queues matched against instructions of sequence
- *at most* one instruction from the queue should match the correct state
- Error Detected when there is :
  - more than one match
  - a match other than expected state

SCM Reconfiguration Architecture

- Achieved with help of CHECK instructions
- Extracted sequences loaded as part of program image
- At runtime SCM loads sequences into set of registers
- Each sequence has additional registers
  - length, state



Process Crash/Hang Detection (1)



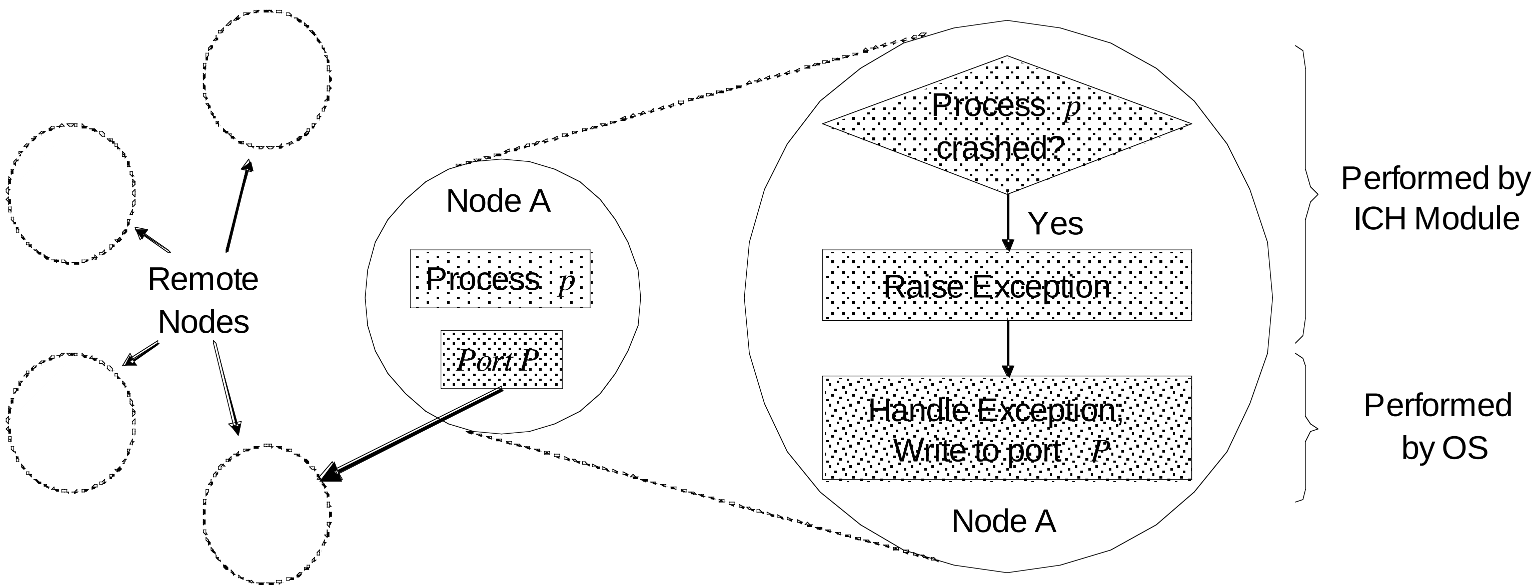
- ◆ Infinite Loop Hang Detection (ILHD) by tracking loop entry and exit points
- ◆ Sequential Code Hang Detection (SCHD) detects illegal repetition of sequence of instructions
- ◆ Instruction Count Heartbeat (ICH) leverages processor performance registers to detect process/OS crashes/hangs

Process/Crash Hang Detection (2)

- Process hang in legal loops
  - Infinite loop Hang Detector (ILHD)
  - Profile-based analysis of application to estimate loop execution time
  - Module reconfigured with timeout for loop as it is entered – CHECK Loop Entry and Loop Exit
- Process hang in illegal loops
  - Sequential code hang detector (SCHD)
  - Parameterize module with length of loop
  - Any loop shorter than given length indicates control error

Process Crash/Hang Detection

- Crash detection
  - Instruction Count Heartbeat (ICH)
  - Uses processor performance counters to detect process and OS crashes
  - Can be extended to support failure detection in distributed systems



Adaptive Application Aware Checking in Software: Runtime Executive (RTE) – Middleware

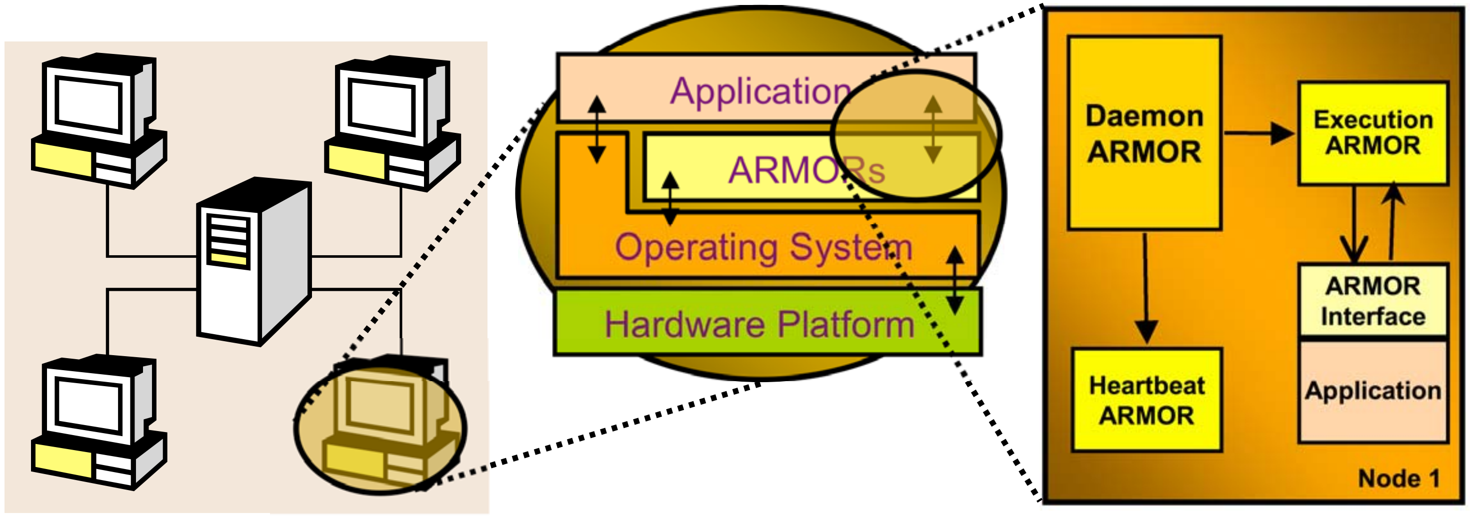
- Reconfigurable statically and dynamically to provide range of customizable error checks to operating system and applications, e.g.,
  - Heartbeats – (i) *adaptive* - the timeout value adapts to changes in the network traffic or node load and (ii) *smart* - the monitored entity excites a set of checks before sending the heartbeat) .
  - Data-Flow Signatures – a pattern of reads and writes to variables in a code block (program object, thread, function, basic block or instruction)
- Self-checking (self-healing)
- Example – reconfigurable ARMOR architecture
  - K. Whisnant, Z. Kalbarczyk, R. Iyer, "A System Model For Dynamically Reconfigurable Software," IBM Systems Journal, Special Issue on Autonomic Computing, March 2003

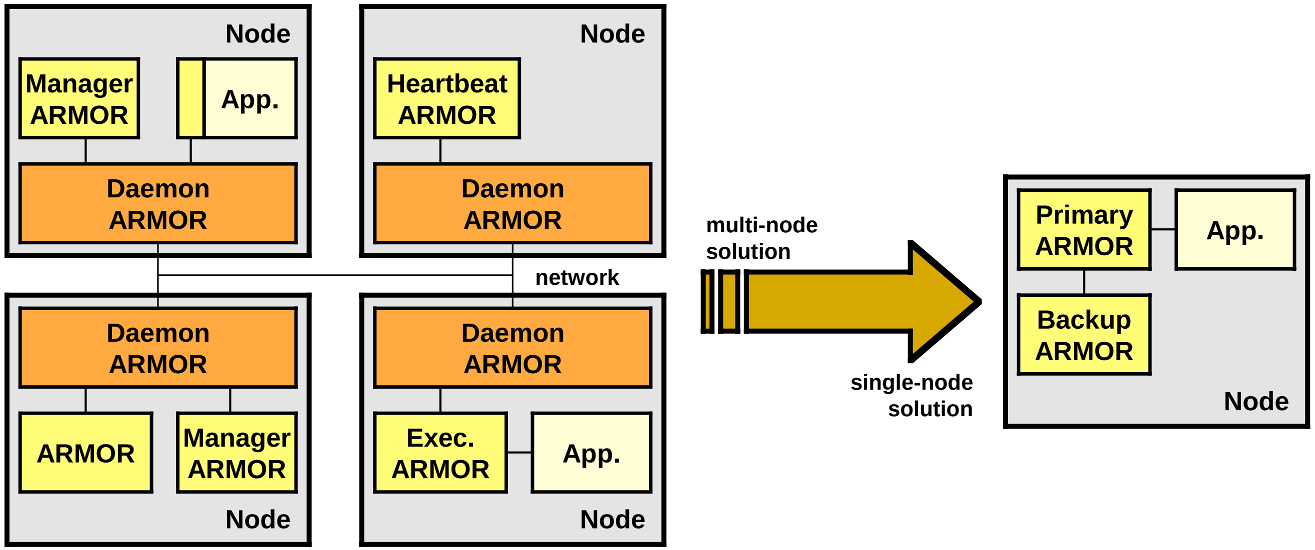
Runtime Executive (RTE): ARMOR Approach

- Adaptive Reconfigurable Mobile Objects of Reliability:
  - Multithreaded processes composed of replaceable building blocks called elements
  - Elements provide error detection and recovery services to user applications or operating system.
- Hierarchy of ARMOR processes form runtime environment:
  - ARMOR runtime environment is itself self checking
- ARMOR properties
  - designed to be reconfigurable
  - resilient to errors by leveraging multiple detection and recovery mechanisms
  - internal self-checking mechanisms to prevent failures from occurring and to limit error propagation.
  - state protected through checkpointing.

Runtime Executive (RTE): ARMOR Approach "Total Solution"

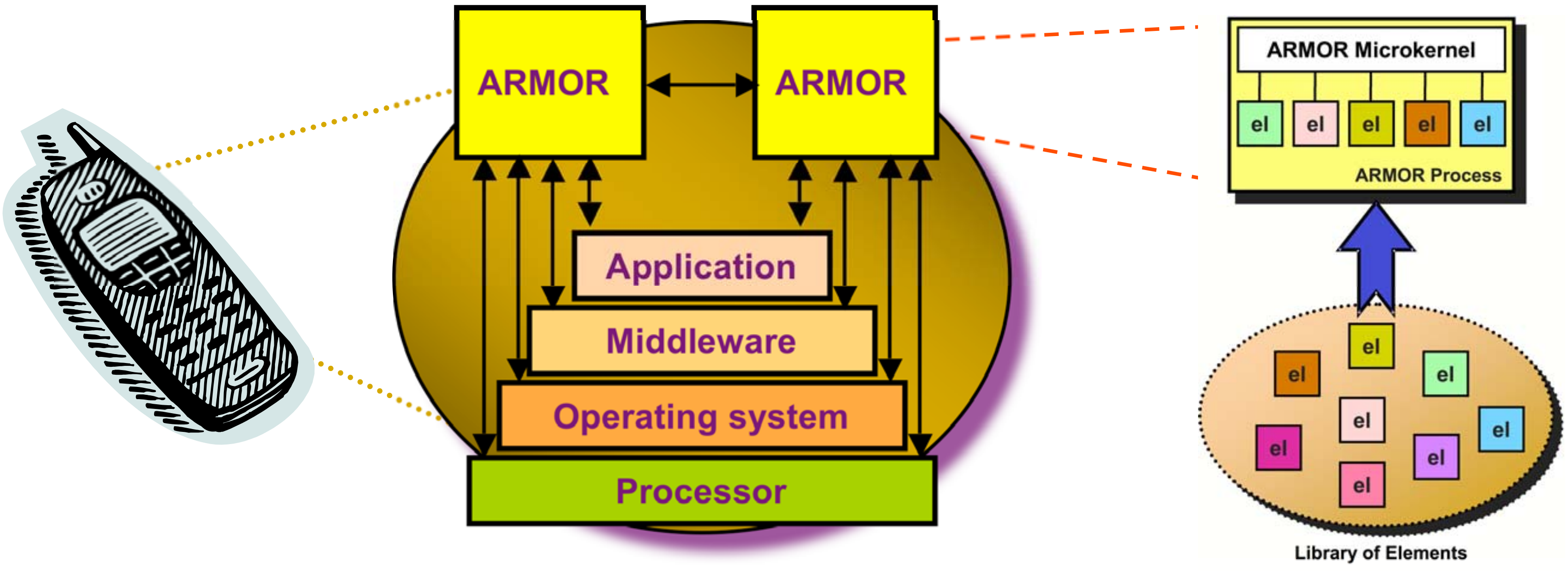
Basic Configuration





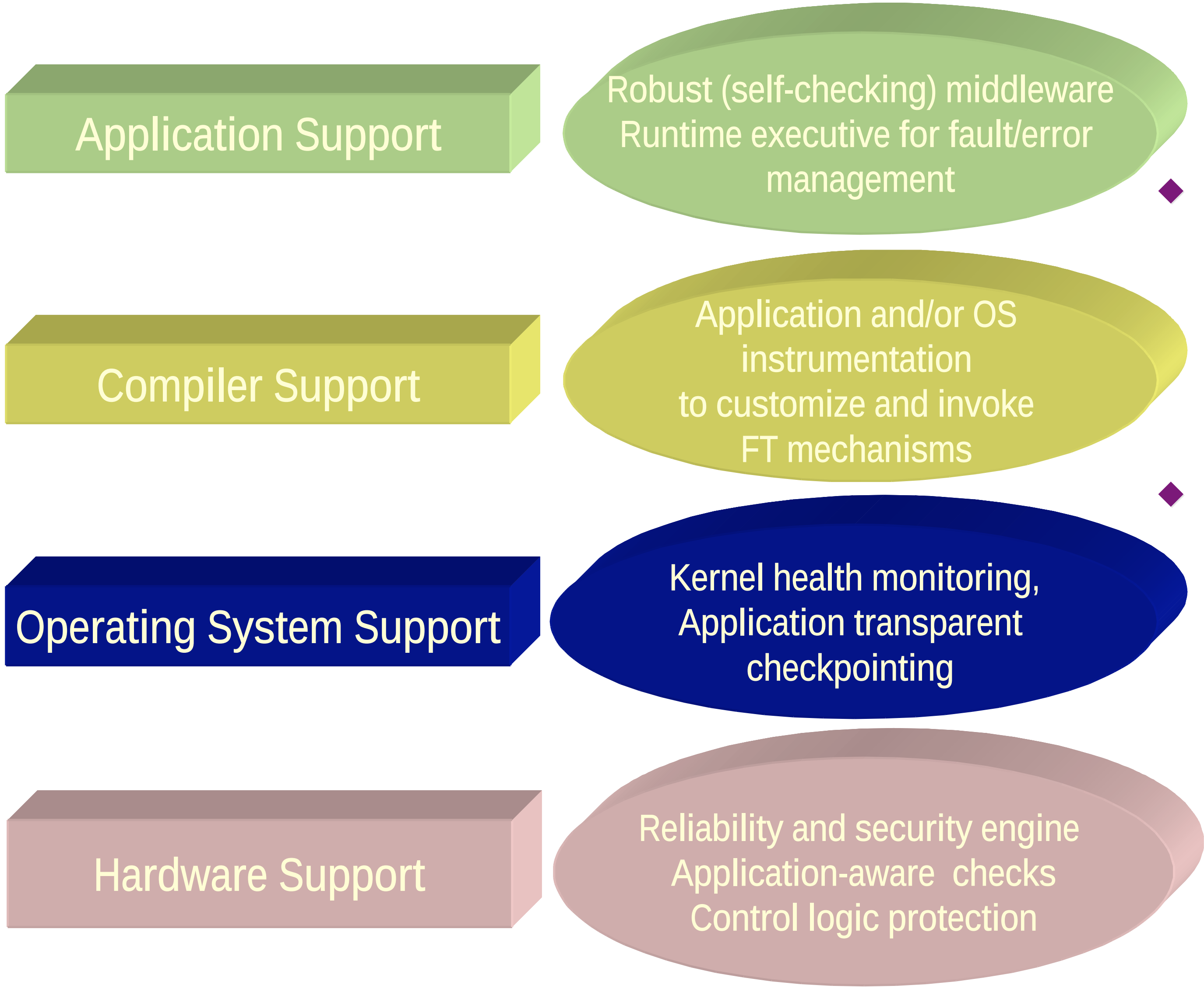
Scaling ARMOR Runtime Environment

Runtime Executive (RTE): ARMOR Approach "Embedded Solution"



- Modular design of processes lends itself well to small footprint solutions.
- Special elements optimized for memory/performance requirements.
- Specialized microkernel:
  - Remove support for inter-ARMOR communication through regular messaging
  - Static configuration of elements; no need to dynamically change elements

Support for Adaptation of Error Detection Across System Hierarchy



- Hardware –
  - a common processor-level framework exploiting features (e.g., debug and performance registers) of current processors
- Software
  - robust, self-checking runtime executive for fault management