

MP-SOC Workshop

From a distributed embedded RTOS to a pragmatic framework for multi-core SoC.

The Future of RTOS for SoC

Eric.Verhulst@eonic.com

www.eonic.com

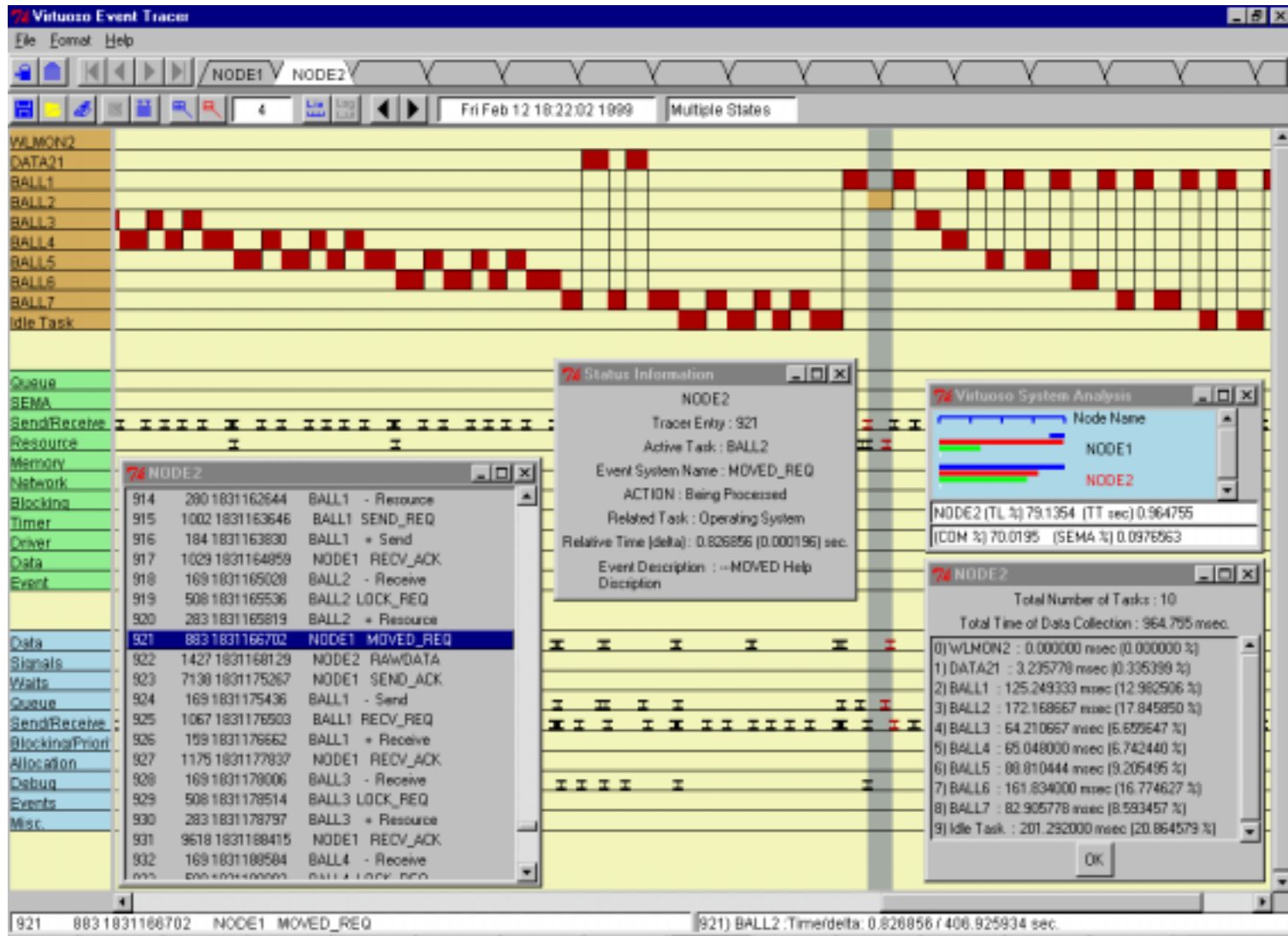
RTOS for embedded DSP : the basics

- Real-Time & Multi-Tasking :
 - **scheduling of multiple tasks** on single CPU
 - **priority** used to meet real-time requirements
- Operating System :
 - **isolates** processor hardware from application
 - **kernel services** allow tasks to synchronize and communicate
 - **interfaces** tasks with I/O through drivers
- Often neglected :
 - protection : no MMU
 - dynamic memory allocation : can be catastrophic
 - parallel DSP : communication should be RTOS service
 - heterogeneous systems : more often the reality

Types of scheduling

- Superloop :
 - loops into single endless loop : [test, call function]
- Round-robin, FIFO :
 - first in, first served
 - cooperative multi-tasking, run till descheduling point
- Priority based, pre-emptive
 - tasks run when executable in order of priority
 - can be descheduled at any moment
- Earliest deadline first :
 - most dynamic form : deadlines are time based
 - but complex to use and implement
 - granularity is an issue

Real-time trace : viewing the scheduling



Multi-tasking

- Origin :
 - a software solution to a hardware limitation
 - von Neuman processors are sequential, the real-world is “parallel” by nature and software is just modeling
- How to ?
 - A function is a [callable] sequential stream of instructions
 - uses resources [mainly registers] => defines “context”
 - non-sequential processing =
 - switching between ownership of processor(s)
 - reducing overhead by using idle time or avoid active wait :
 - each function has its own workspace
 - a task = function with proper context and workspace

A task is more

- Tasks need to interact
 - synchronize
 - pass data = communicate
 - share resources
- a task = a virtual single processor
- a task = unit of abstraction
- a multi-tasking system emulates a real system
- developed out of embedded industrial needs
- theoretical model :
 - CSP : Communicating Sequential Processes
 - C.A.R. Hoare
 - formal, but doesn't match complexity of real world

Control flow, dataflow and time-triggered

- Three dominant real-time paradigms :
 - control flow :
 - event driven - asynchronous : latency is the issue
 - traverse the state machine
 - uncovered states generate complexity
 - data-flow :
 - data-driven : throughput is the issue
 - multi-rate processing generates complexity
 - time-triggered :
 - play safe : allocate timeslots beforehand
 - reliable if system is predictable
 - REAL SYSTEMS : combination of above

DSP : the basics

- Any processor can execute a DSP algorithm
- But :
 - DSP is real-time data processing at high rates :
 - requires adequate concurrent I/O
 - requires fast interrupt handling
 - control code is barely an overhead on a DSP
 - DSP often requires tight inner loop code
 - DSP applications have other constraints as well :
 - Power consumption : Ops/Watt
 - Heat dissipation : convection cooled
 - Performance density : Ops/mm²
 - Cost density : Ops/\$
 - DSP must scale I/O and processing
 - DSP is diverse because of the data types :
 - fixed point 16/24/32 bit, floating point 32/64bit, composite

DSP : the future

- The challenge :
 - stretching the von Neuman machine :
 - throughput processing : VLIW, multiple ALU, pipelining
 - multimedia : mixed datatypes
 - memory speed < CPU speed
 - I/O : DMA
 - boundary conditions :
 - size : as small as a stamp
 - power consumption : runs on an AA battery for days
 - heat : so cool you can touch it
 - processing power : multiples of 10 Gops/sec
 - I/O bandwidth : multiples of 1 Gbit/sec

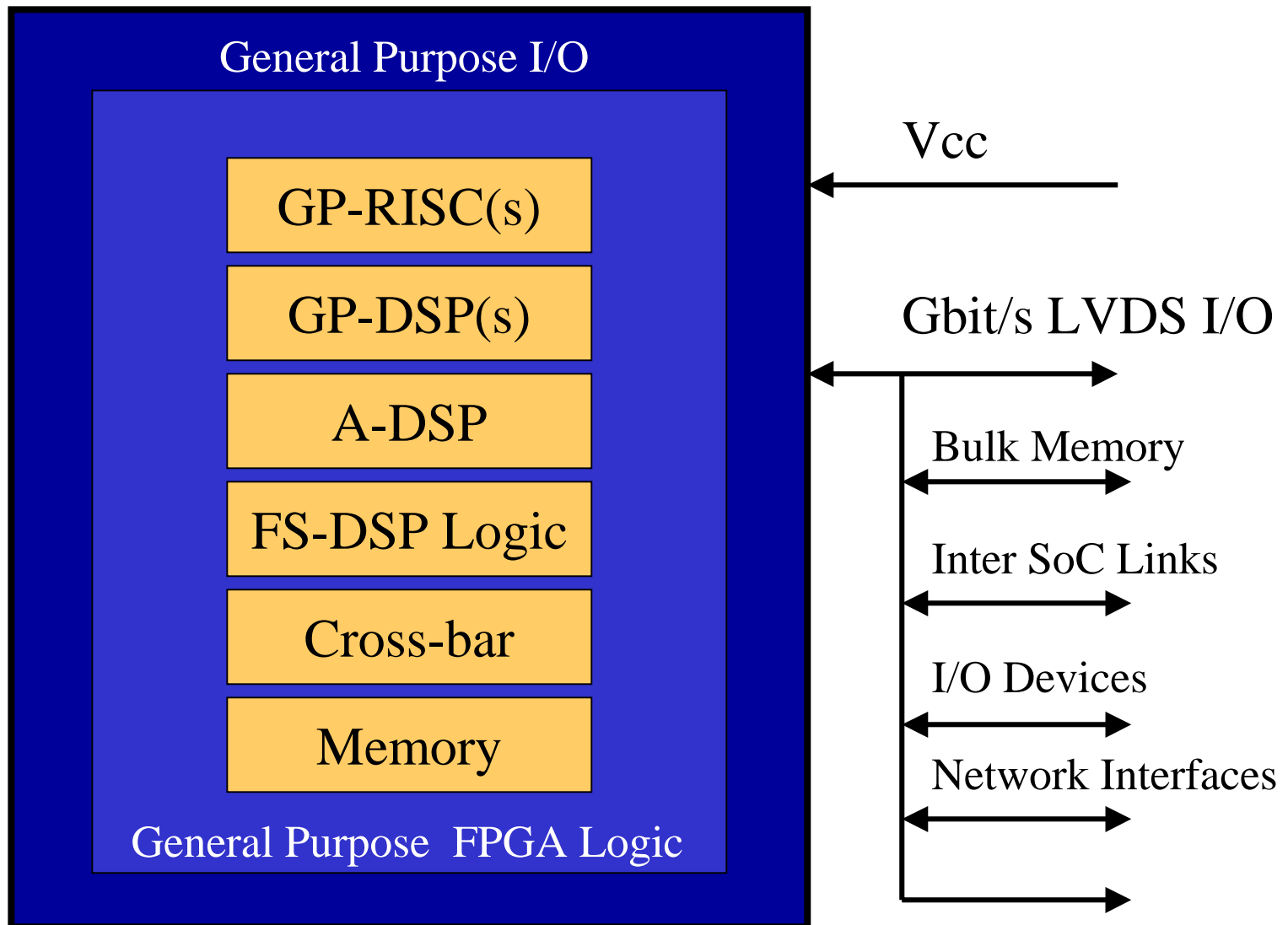
How will it look like (1) ?

- Technology enabled :
 - shrinking line widths (0.13 um and less)
 - copper interconnects
 - lower supply voltage
 - dynamic frequency and voltage control
- Challenges :
 - NRE cost doubles, while line width shrinks
 - frequencies are in the RF, even radar range
 - packaging and I/O (incl.. Memory) are the real bottlenecks
- Facts :
 - single clock 1Bn gates on-a-chip is not trivial
 - power consumption = $F(\text{Hz}, V_{cc})$




How will it look like (2) ?

- Conclusion :
 - multi-core, coarse grain asynchronous SoC design
 - cores as proven components -> well defined interfaces
 - keep critical circuits inside
 - simplify I/O :
 - high speed serial links
 - NRE dictates high volume -> more reprogrammability
 - system is now a component
 - below minimum thresholds of power and cost, it becomes cheap to “burn” gates
 - software becomes the differentiating factor

The next generation SoC with DSP

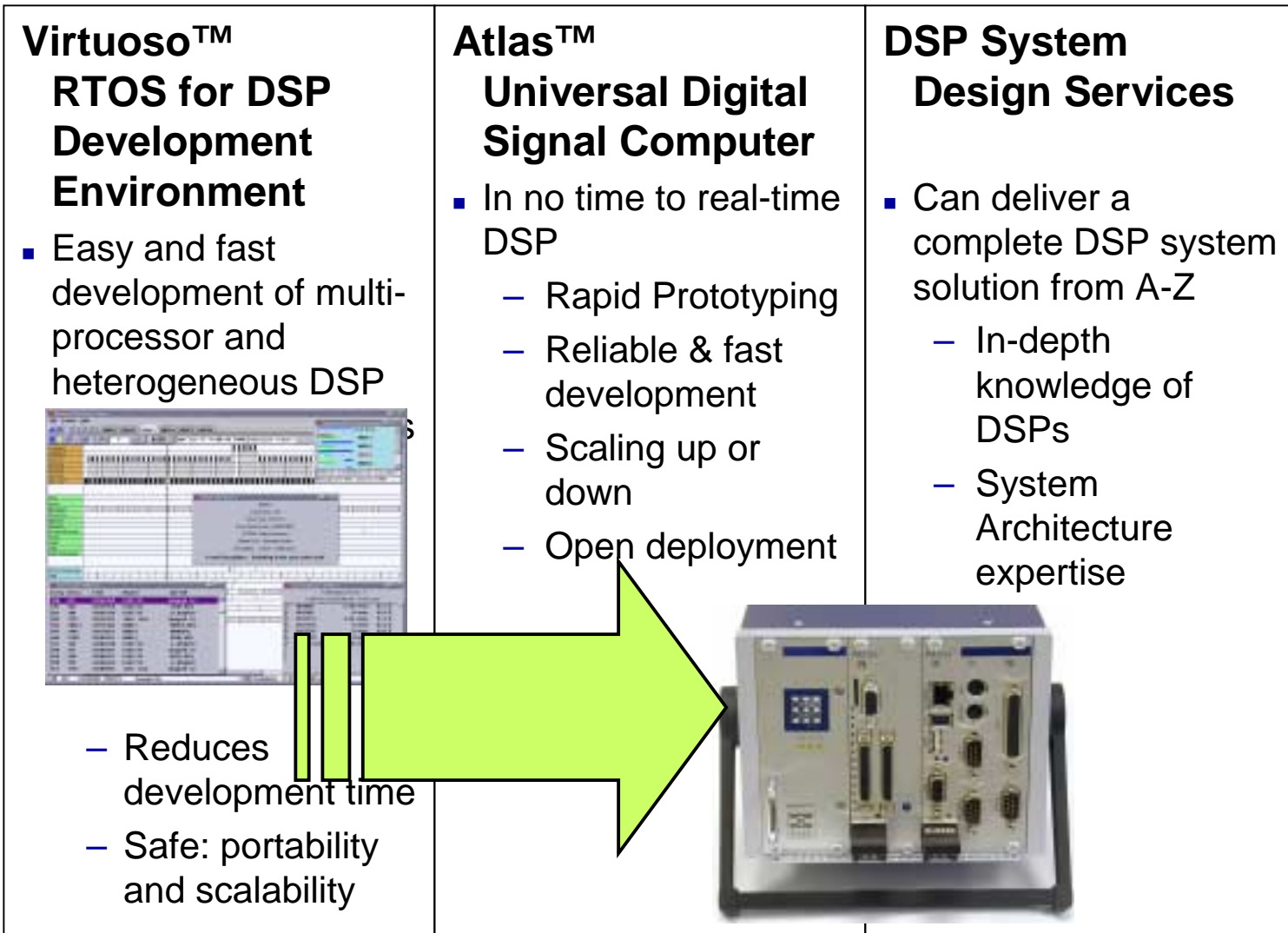


Complexity is the driving force behind the software dominance in applications

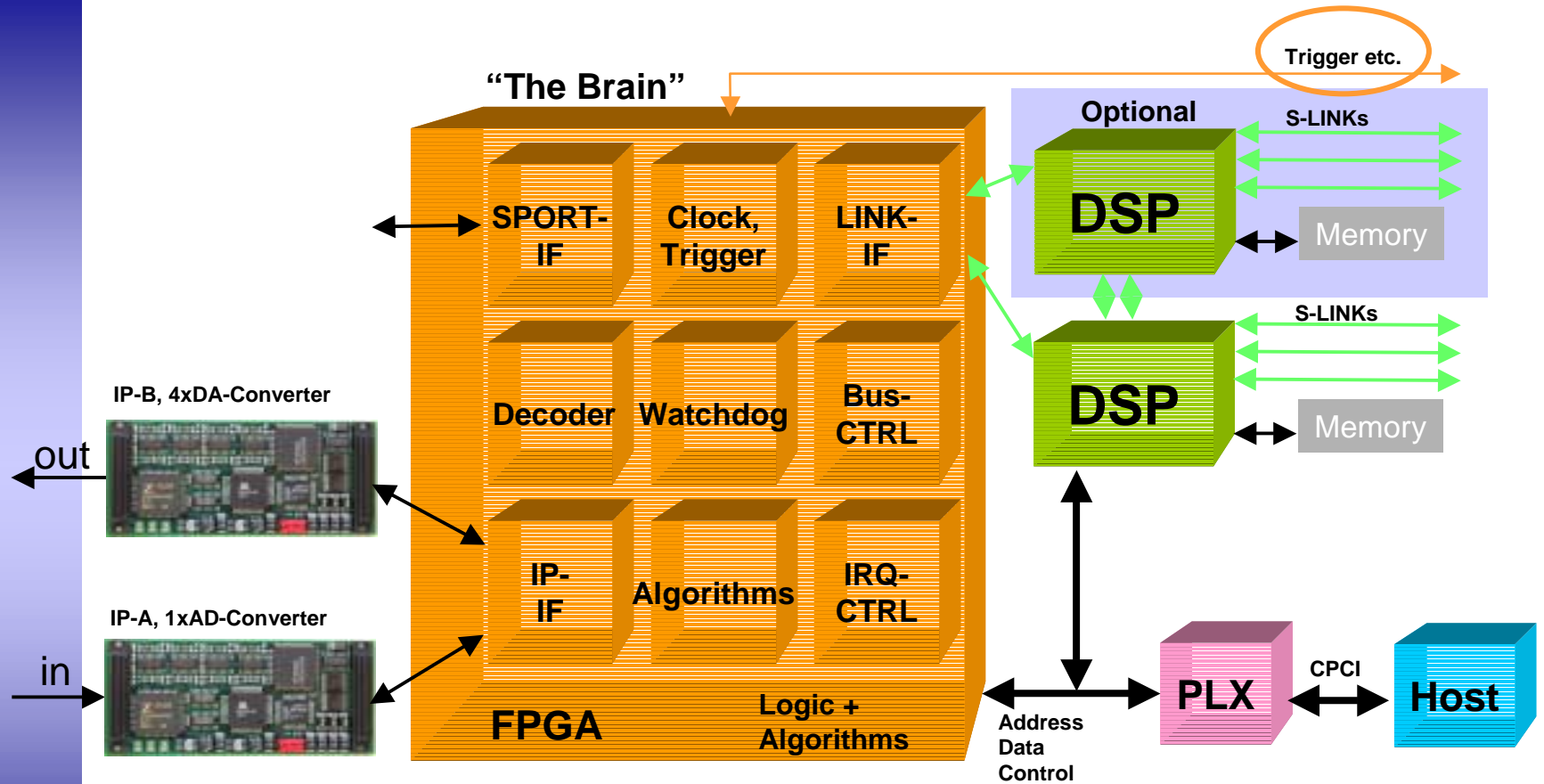
- Processor performance doubles every year 
 - Complex requirements 
 - Shorter time-to-market 
- Software driven design
concept
- because
- software dominates the
application

In a hardware driven world, this is almost like a revolution. Software focused approaches lead it.

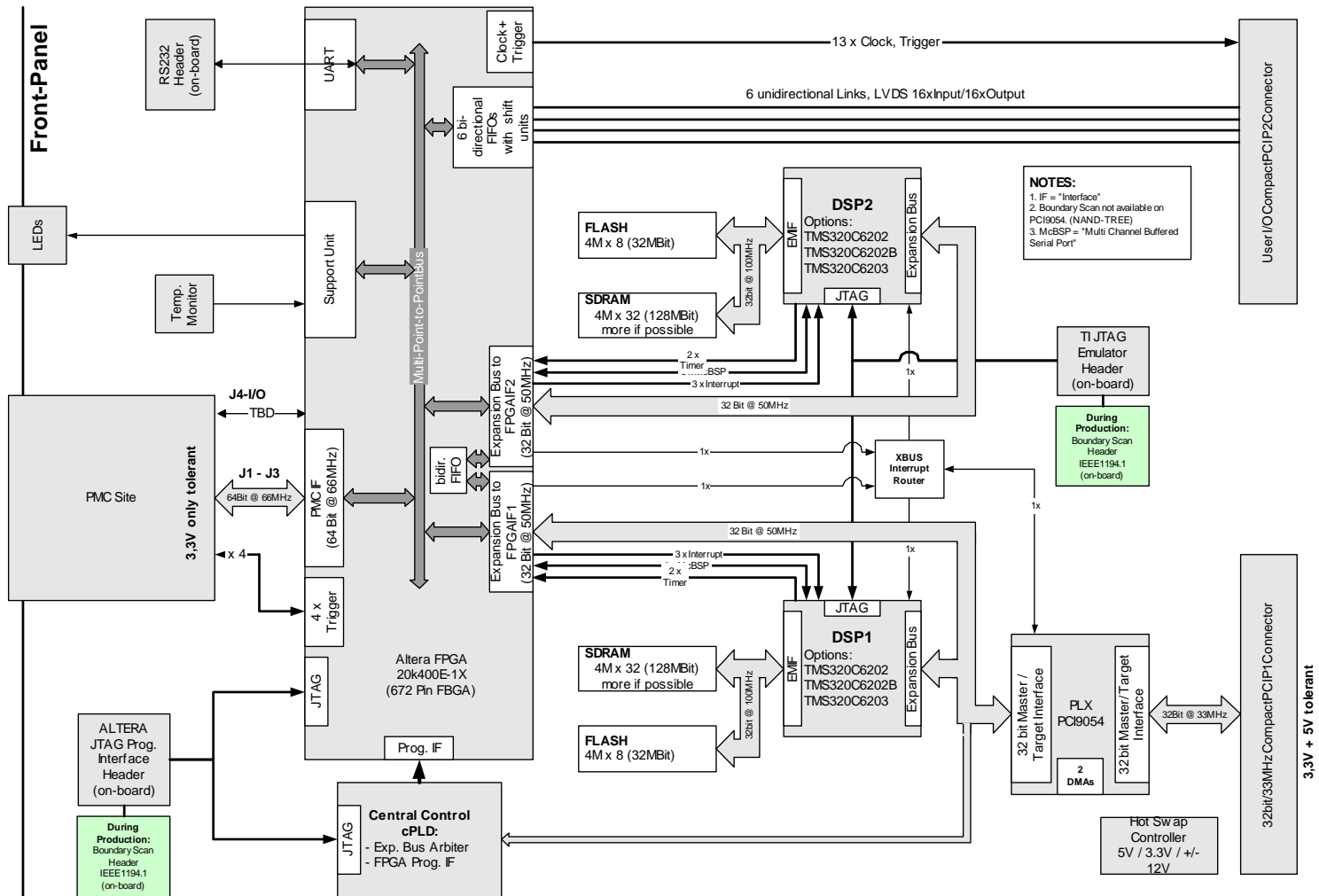
Example : the Virtuoso software RTOS tool as design approach for the Atlas DSP system



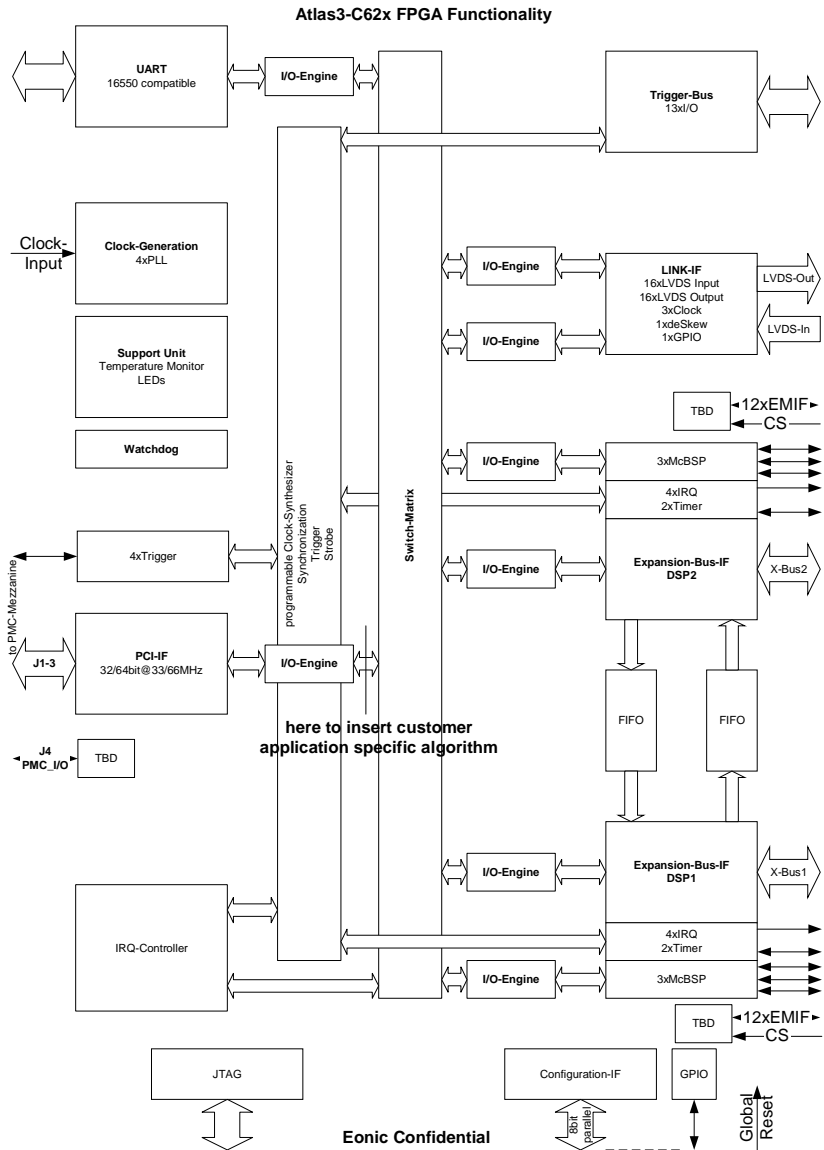
Hardware becomes software at board level



Atlas 3-C6202/3 block diagram



FPGA content as IP : a general purpose I/O and communication engine

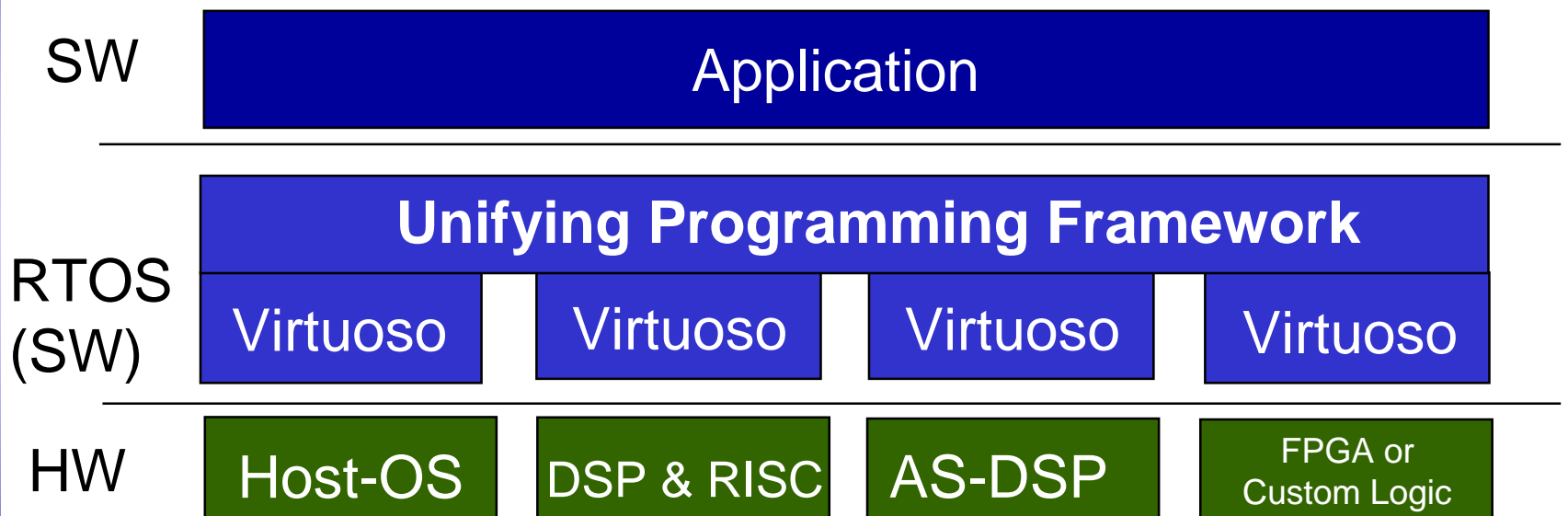


- FPGA implements 90 % of all glue logic on board
- Inter-processor communication using DMA and high speed LVDS
- Intelligent I/O to offload processor (IO-engine)
- Creates processor independent communication and I/O block
- Remaining gates for processing in I/O stream
- Fully supported by Virtuoso's VSP model



What is Virtuoso ?

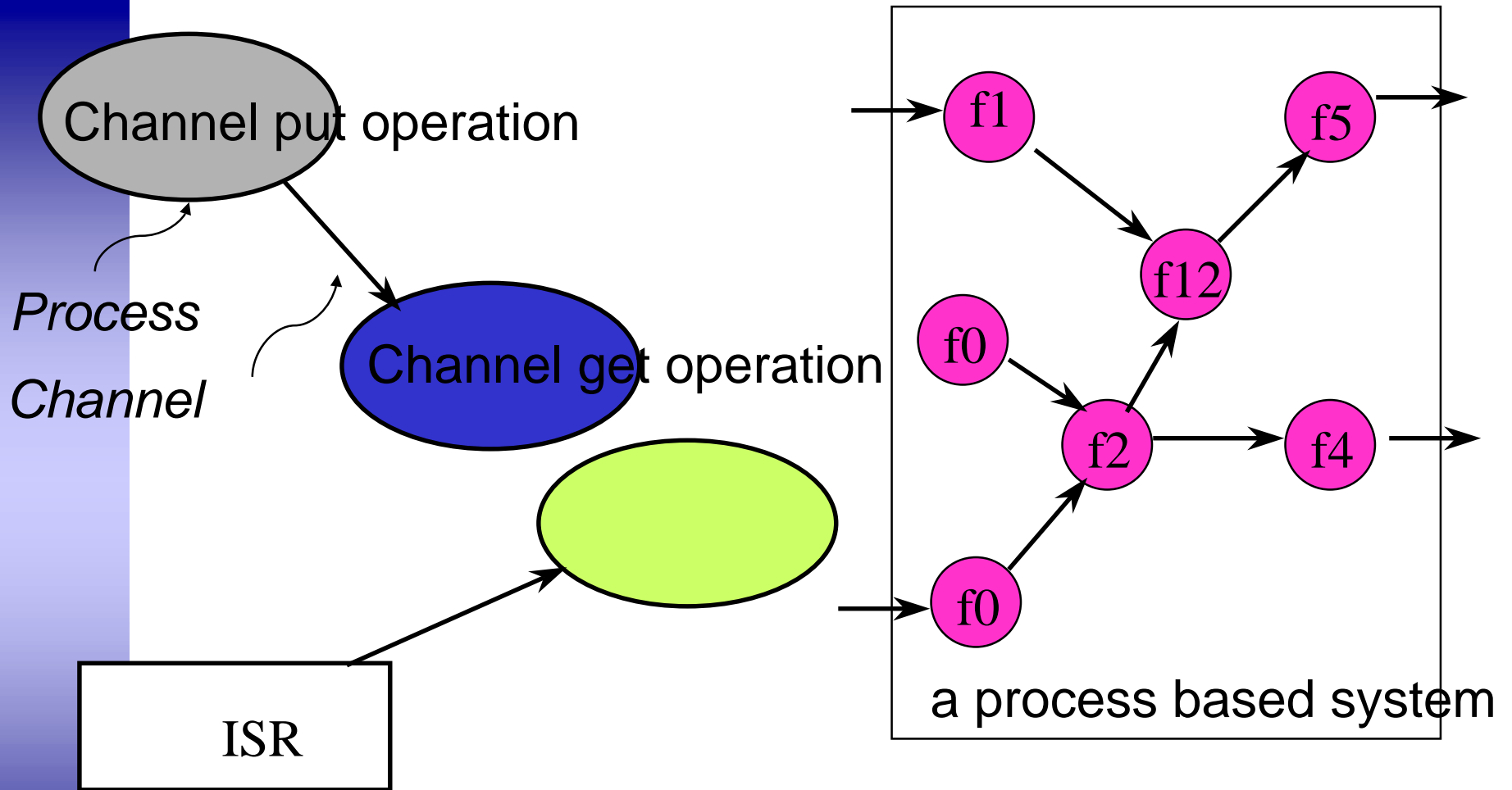
- A Real Time Operating System (RTOS) at the core
- Creates independence of the HW from the SW
 - “Virtual Single Processor” model
- Created By Eonic Systems, now owned by Wind River



Beyond the RTOS

- Multi-tasking = Process Oriented Programming
- A Task =
 - Unit of execution
 - Encapsulated functional behavior
 - Modular programming
- High Level [Programming] Language :
 - common specification :
 - for SW
 - compile to asm
 - for HW
 - compile to VHDL or Verilog
 - enabler for SoC “co-design”

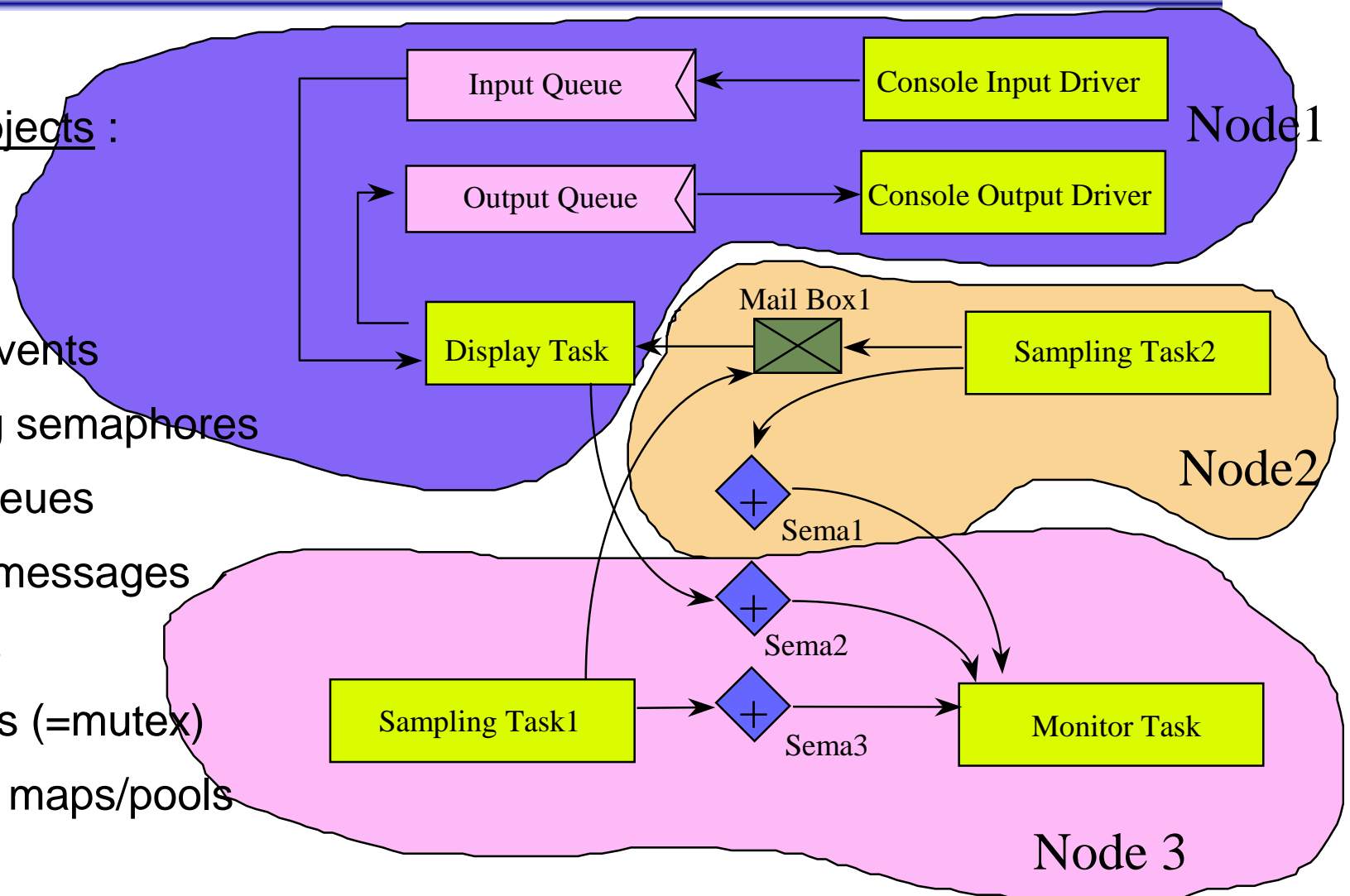
System level design inspired by CSP : processes are abstract building blocks



Virtuoso's Virtual Single Processor : a pragmatic CSP : distributed semantics

RTOS Objects :

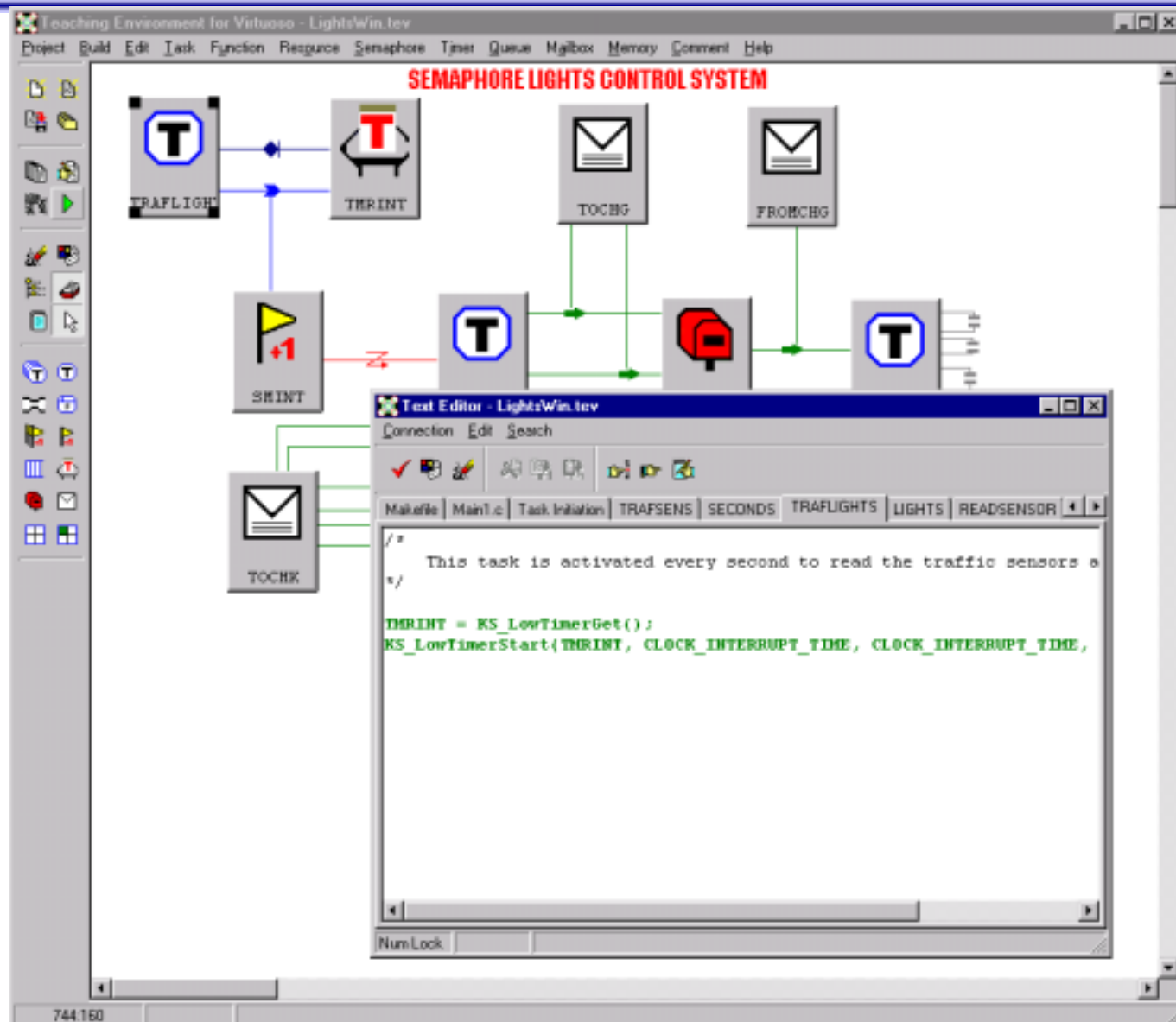
- tasks
- drivers
- binary events
- counting semaphores
- FIFO queues
- mailbox/messages
- channels
- resources (=mutex)
- memory maps/pools



Kernel services : an orthogonal set

- Events : binary data, one to one, local -> interface to HW
- [counting] semaphore : events with a memory, many to many, distributed
- FIFO queue : simple and static datacomm between tasks, many to many, distributed
- Mailbox : variable size datacomm between tasks, rendezvous, one/many to one/many, distributed
- Channels : variable datasize, between tasks and/or host service, asynchronous on send and receive, distributed
- Resources : ownership protection, priority inheritance
- Memory maps/pools
- semantic issues : distributed, group operators, blocking, non-blocking, time-out

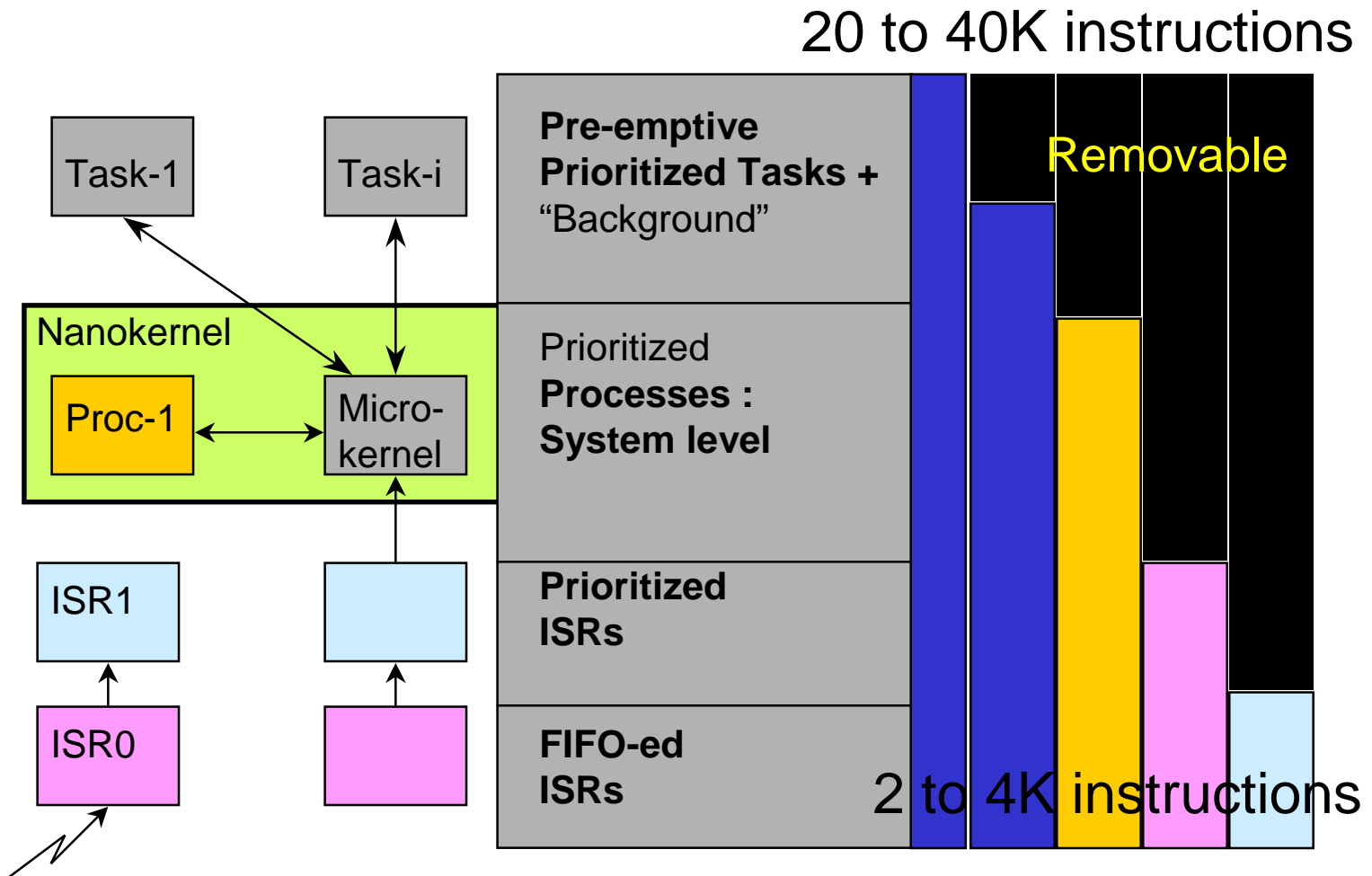
Teaching Environment for Virtuoso



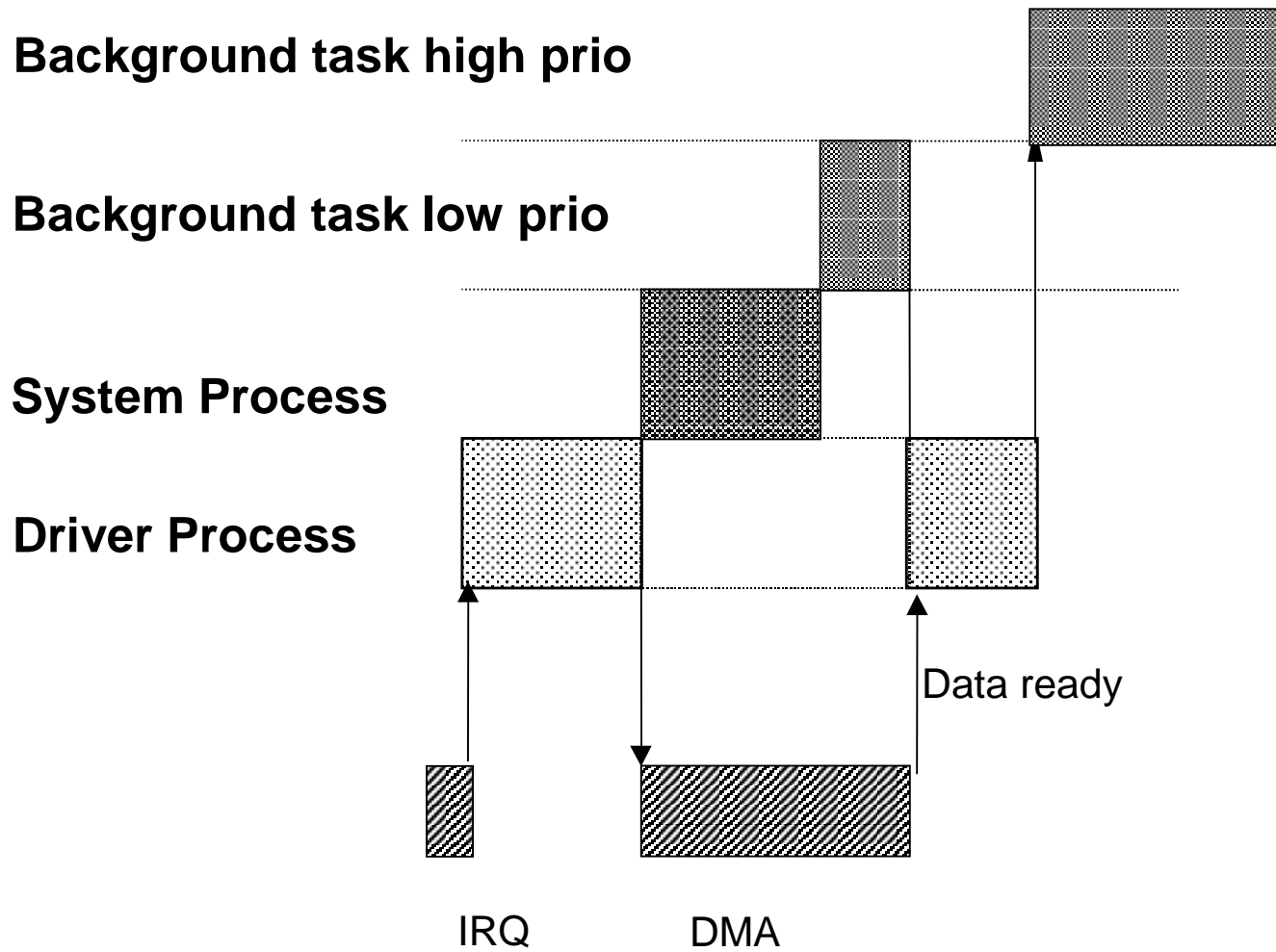
The unique Virtual Single Processor (VSP) model

- Transparent parallel programming
 - Cross development on any platform + portability
 - Scalability, even on heterogeneous targets
 - Distributed semantics
 - Program logic neutral to topology and object mapping
 - Clean API provides for less programming errors
 - Prioritized packet switching communication layer
 - Based on “CSP” (C.A.R. Hoare): Communicating Sequential Processes: VSP is pragmatic superset
-
- **Multitasking and message passing**
 - **Process oriented programming**
 - **Interfacing using communication protocols**

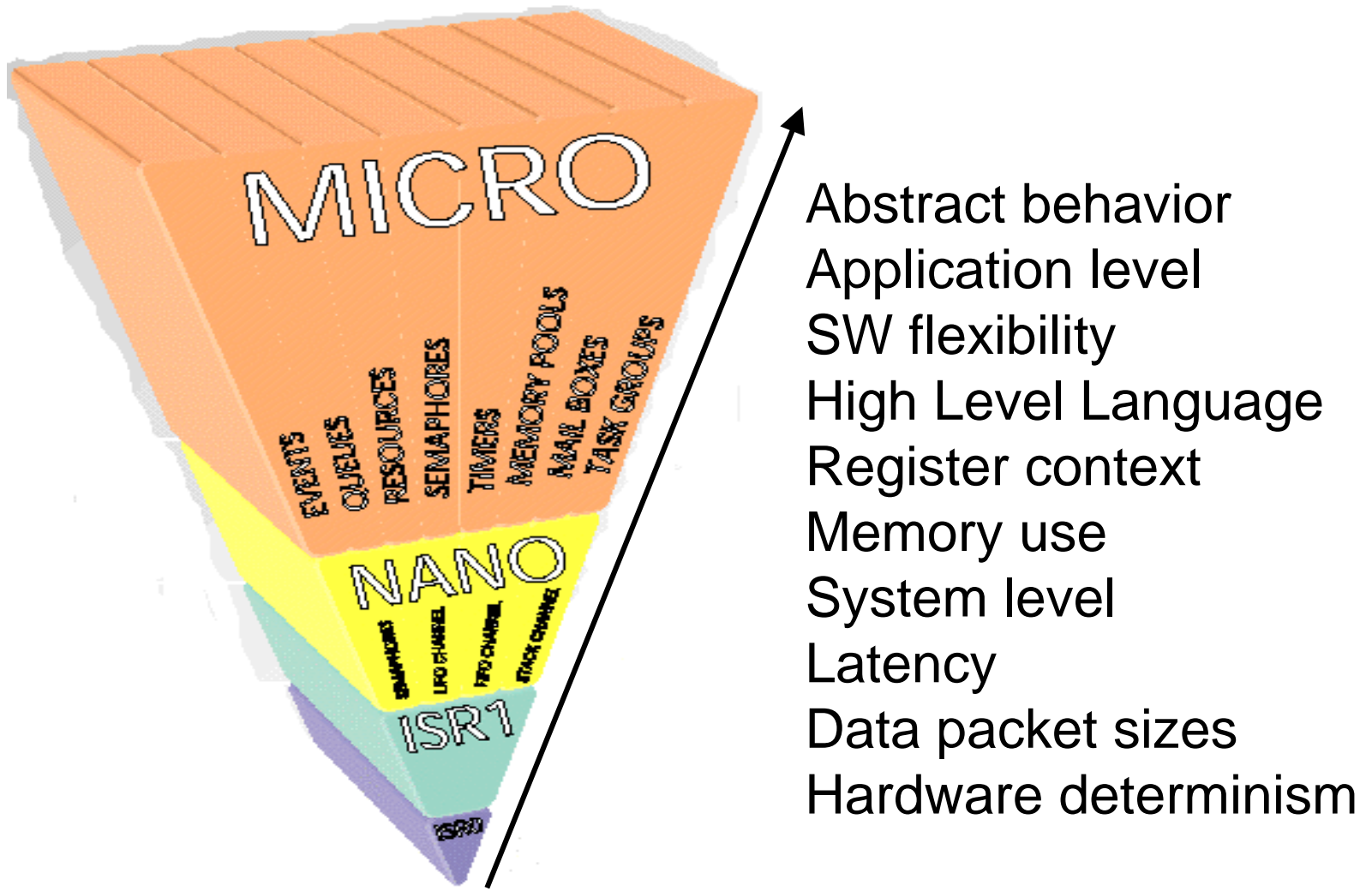
Layered model : separate system from application



Hierarchy = prioritization



Hierarchy and HW and time resources

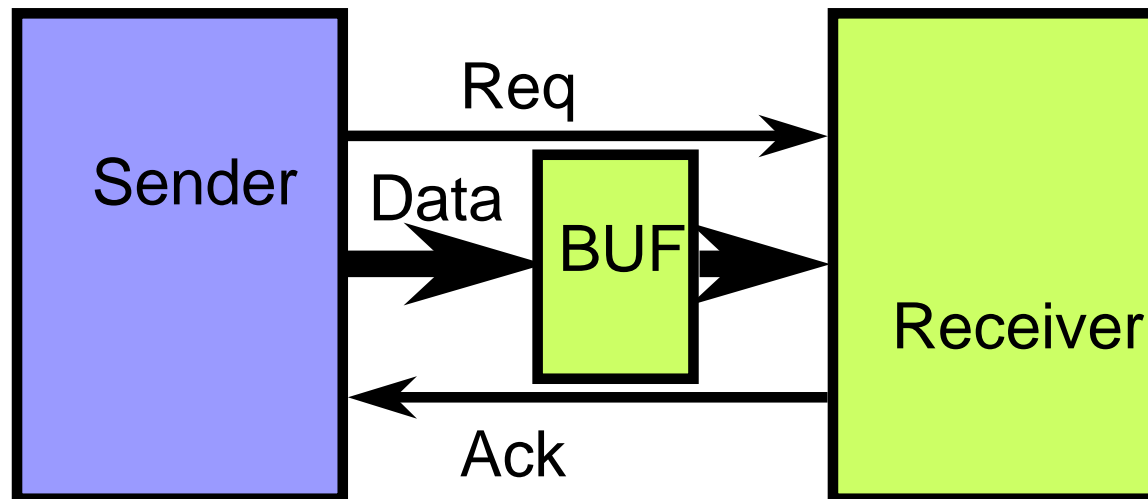


Implementation of VSP model

- Application tasks preemptively scheduled by local microkernel, full context
- Local nanokernel “processes” :
 - Reduced context (some assembly required)
 - Prioritized FIFO scheduler
 - Used for microkernel process and drivers
- Remote services :
 - Detected by objectID of invoked kernel object
 - Microkernel passes command packets to communication layer
 - Communication layer (on “netlinks”) :
 - prioritized packet switching
 - Parallel paths possible
 - Local buffers for throughput routing
 - Deadlock free

CSP at the HW level

- Request/Ack protocol assures correct data transfer between async units, even at the register level
- Is like the mailbox mechanism



RTOS objects : mapping onto HW

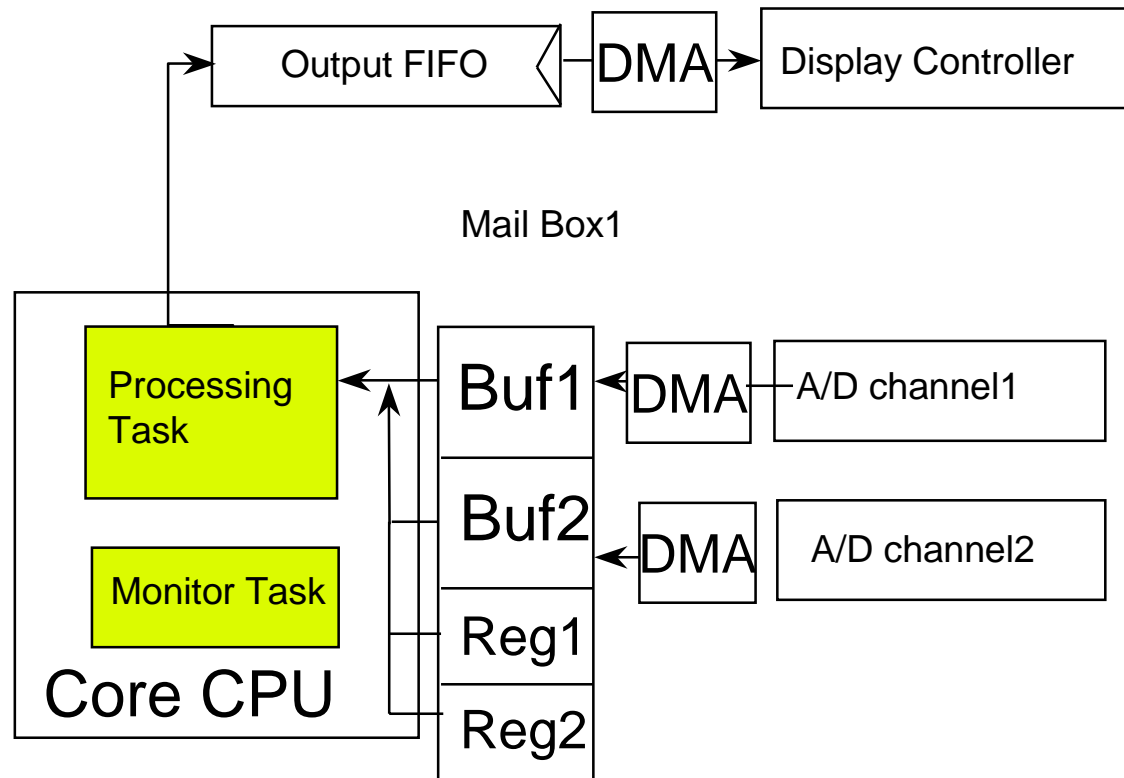
	<u>Software</u>	<u>Hardware</u>
	Task - Process	Logic State Machine
	KS_FifoPutW	FIFO memory
	KS_MsgPutW	shared memory + dma
	KS_SemaSignal	status register + counter

RTOS objects can be used for SW+HW system specification, simulation and implementation

A SW-HW implementation (see slide 21)

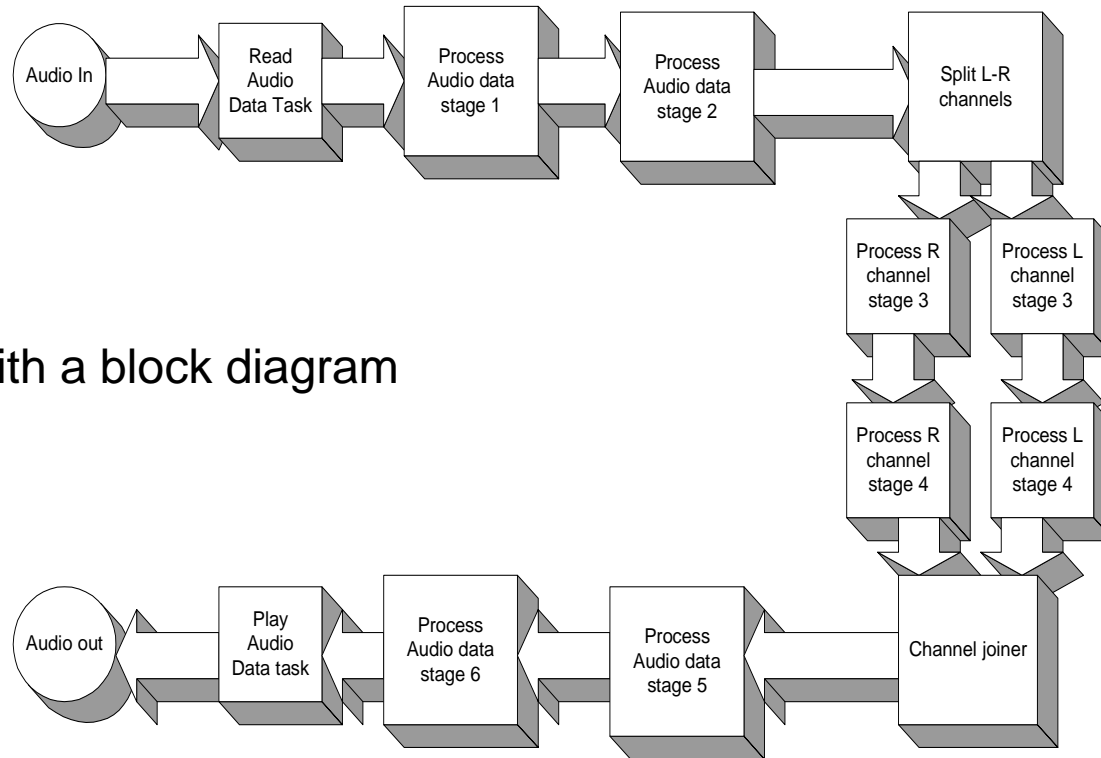
Steps :

1. Algorithm using MATLAB/SDT, Pegasus, ...
2. Simulate logic model with RTOS simulator on host OS like NT
3. Run RTOS program on target CPU
4. Map parts onto SW (C to ASM - binary)
map parts onto HW (C to VHDL or RTL)



How to get there ? Virtuoso VSP

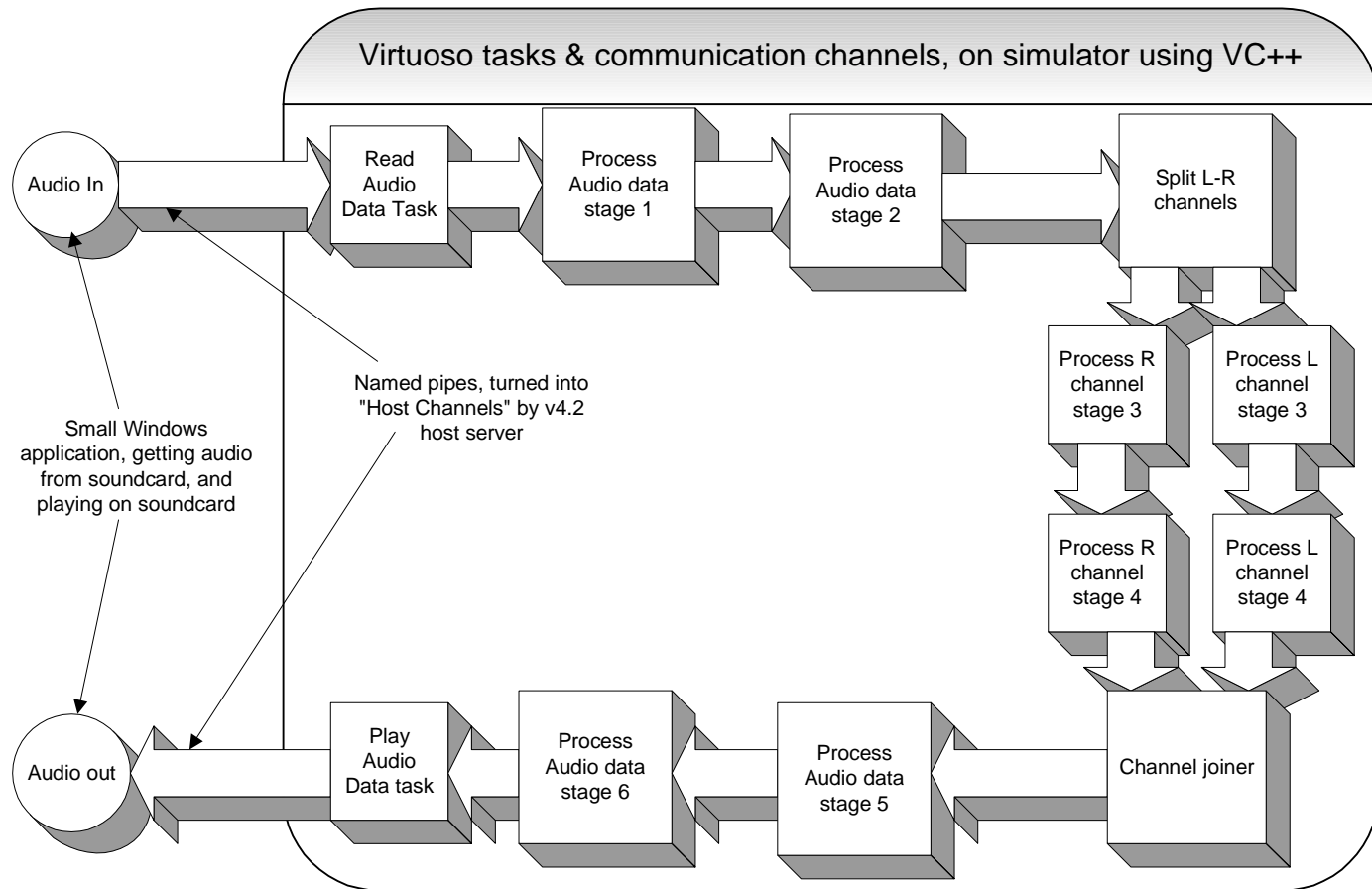
- Example: audio application



- Start with a block diagram

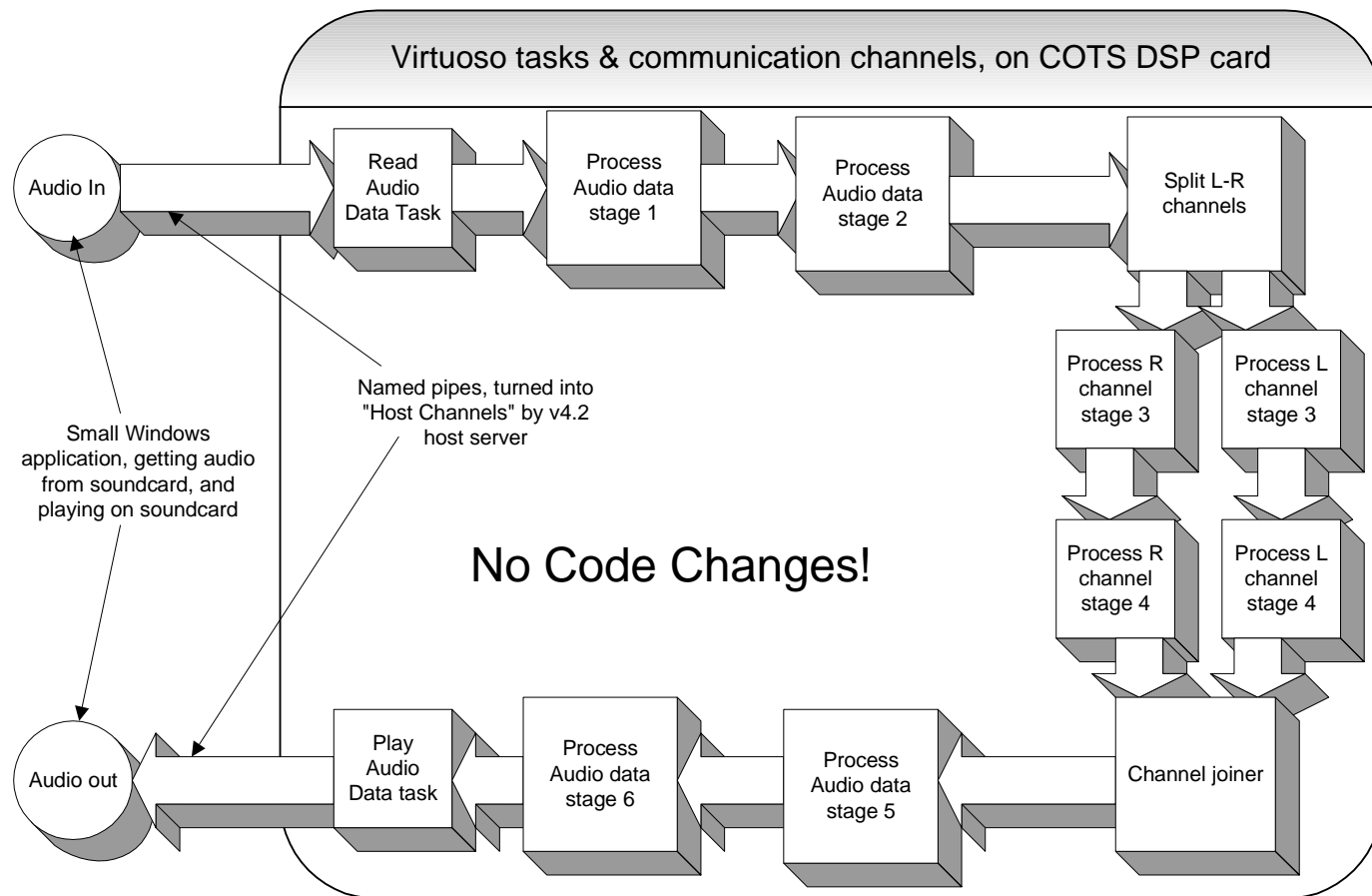
Early development

- Prototype development: Virtuoso simulator on workstation



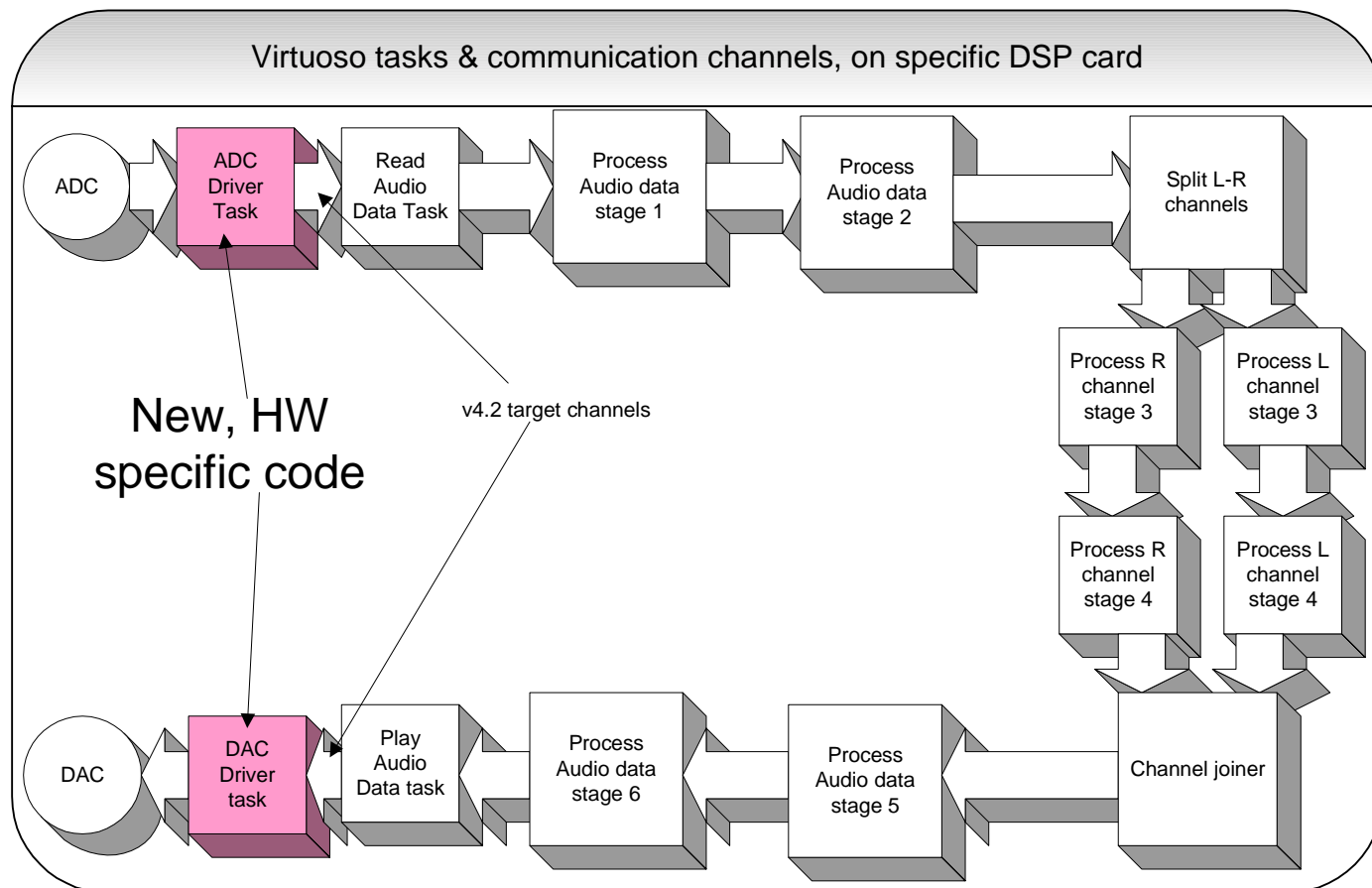
Early development

- Profiling/benchmarking: COTS DSP HW



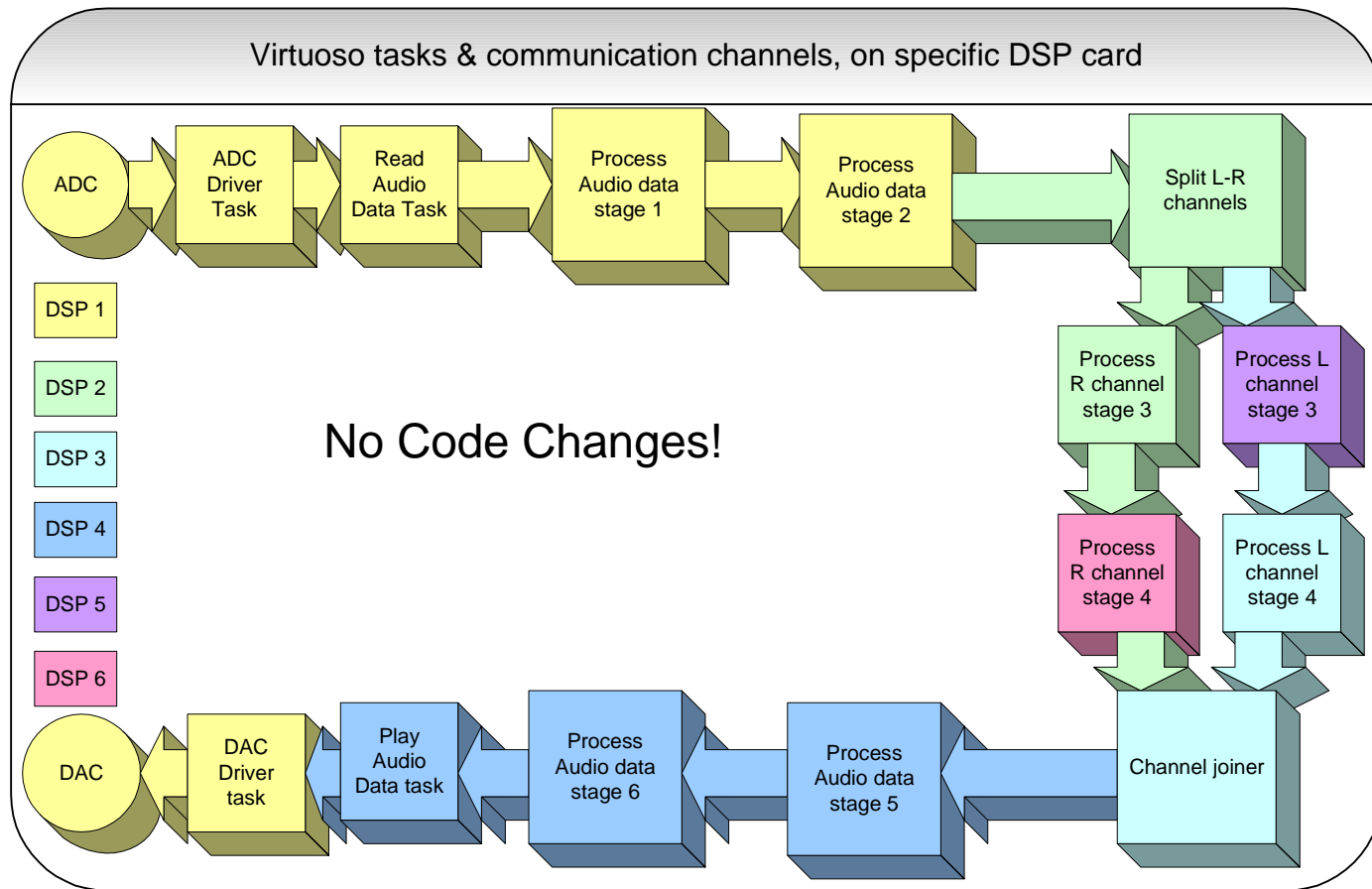
Early development

- Deployment: specific DSP HW, with on-board I/O



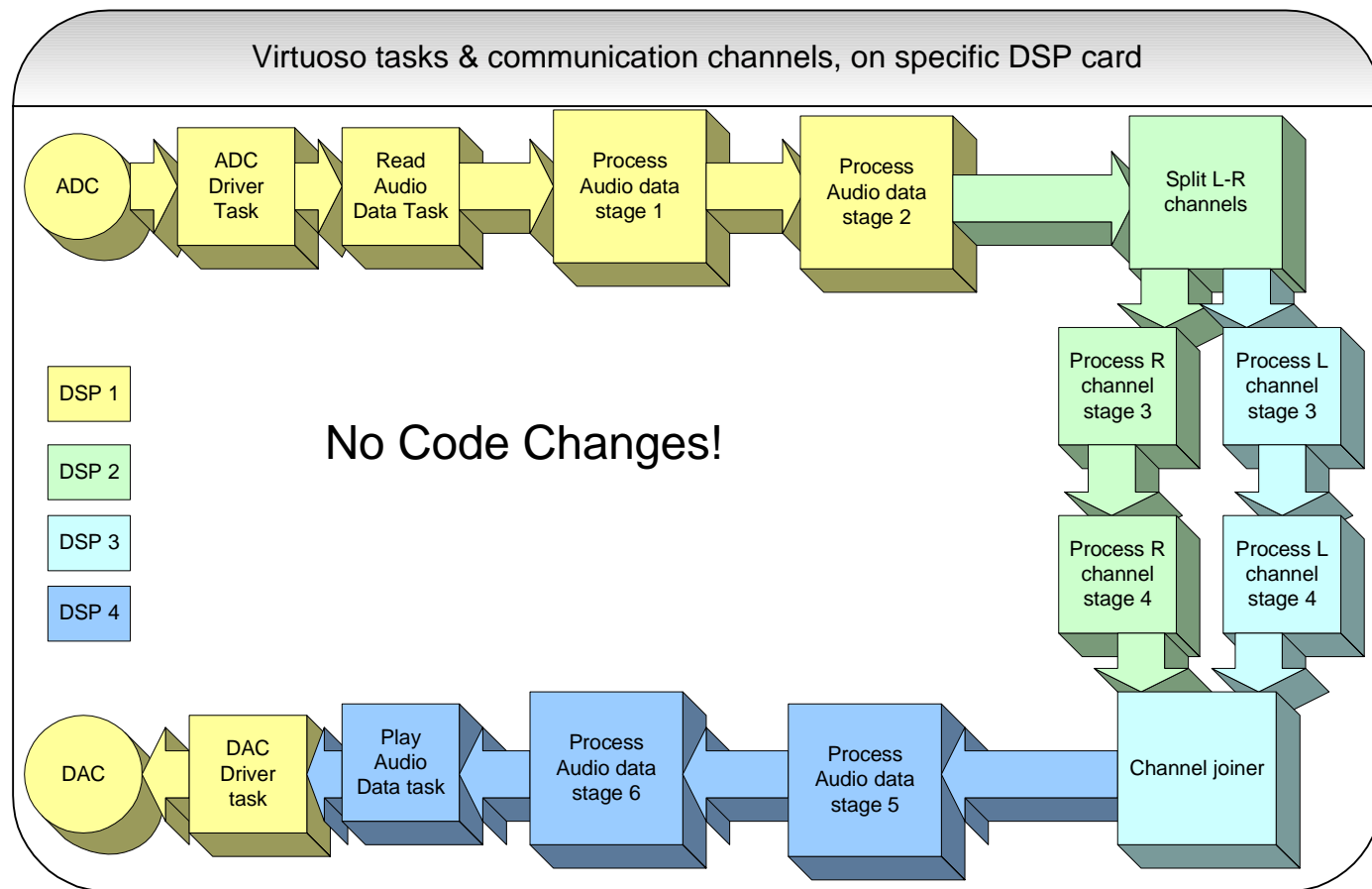
Distribute processing

- distribute tasks over different DSP processors



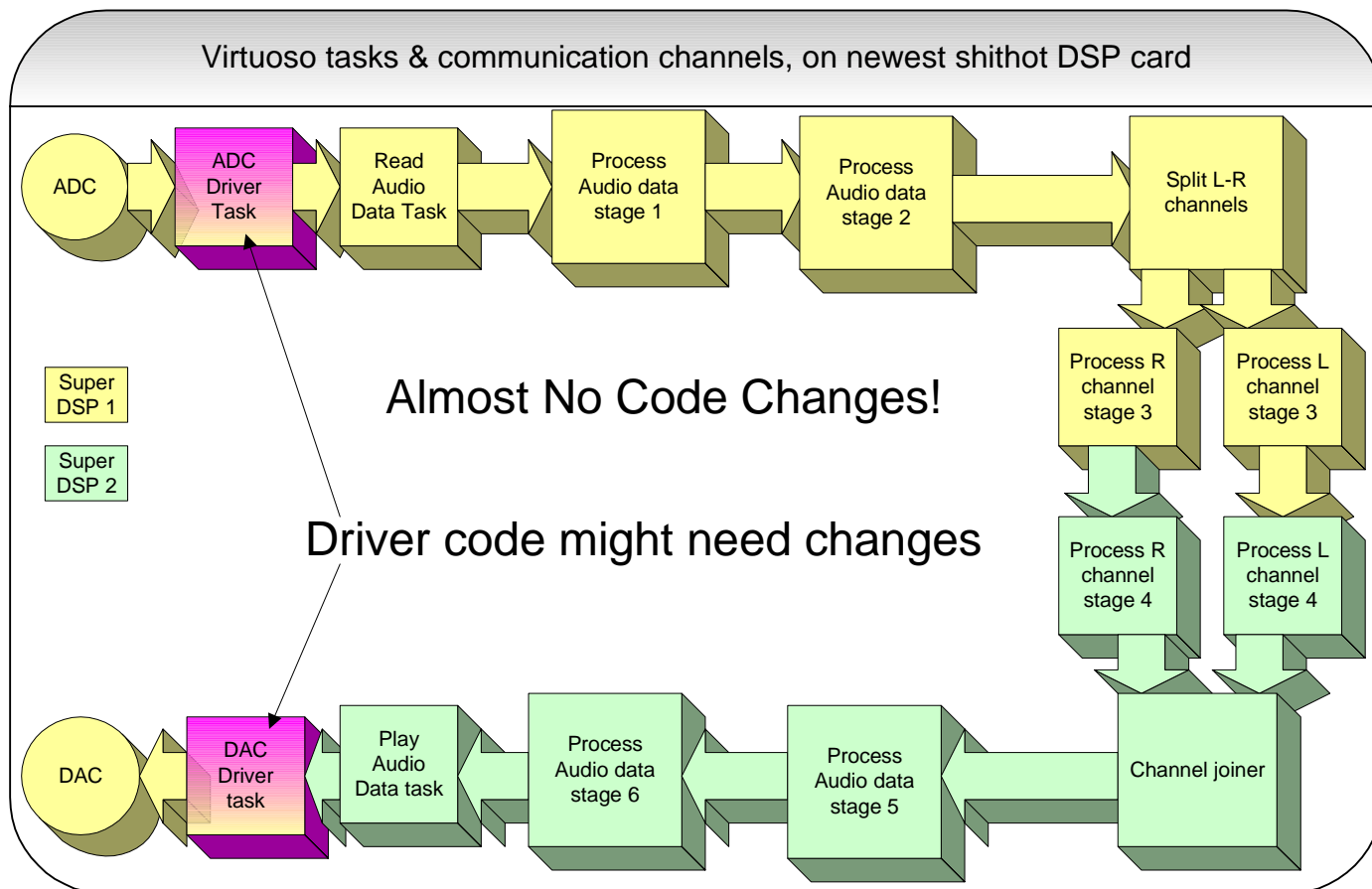
Tune processing

- Introduce hand-optimized algorithms \Rightarrow less processing power



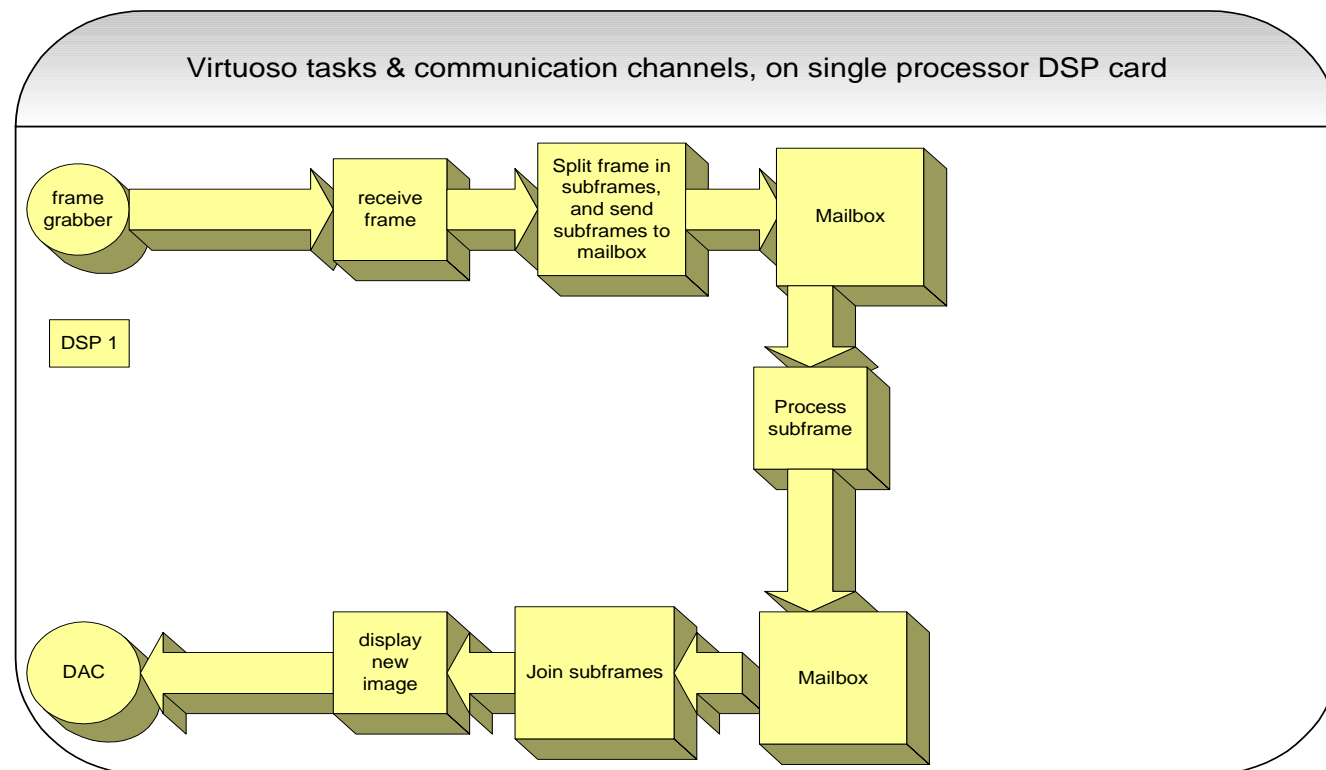
Portable development

- Move on to the newest superduper DSP (when it finally gets there...)



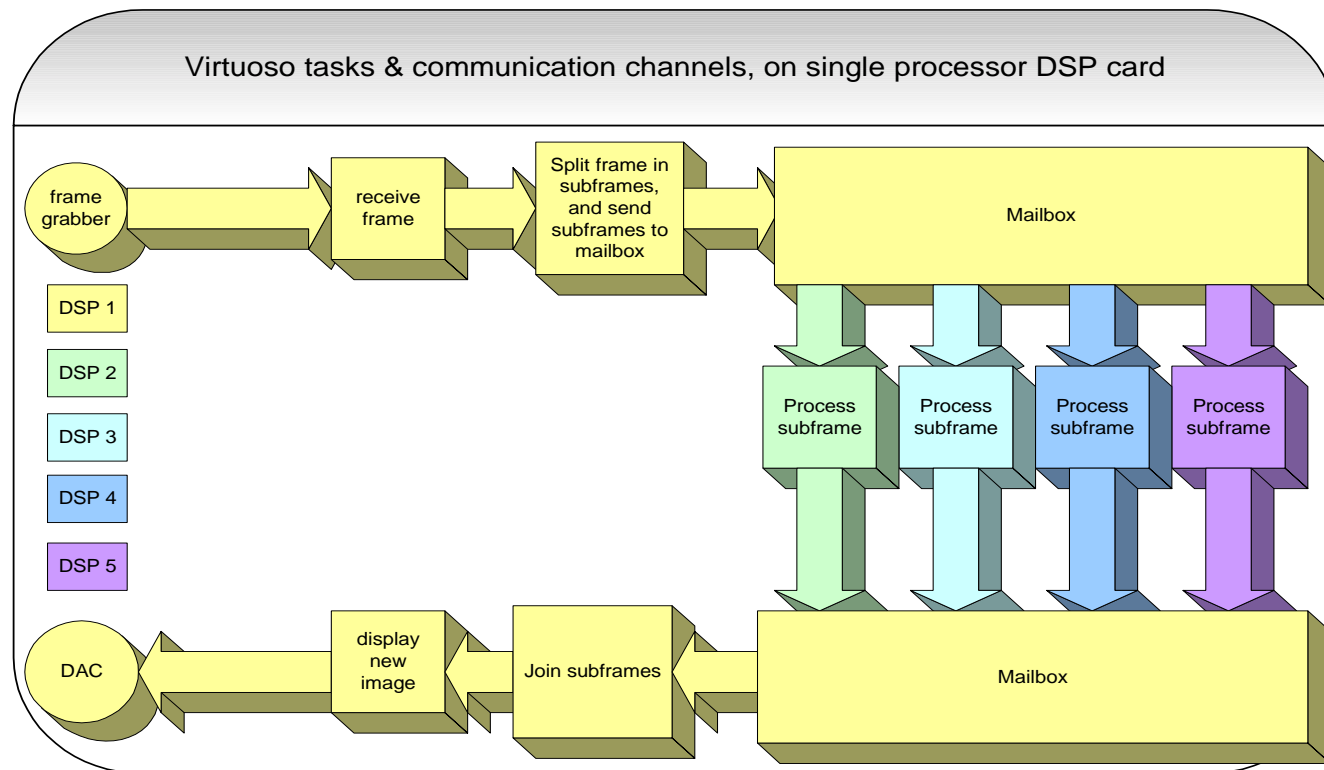
Scalable systems

- Add processing power where needed
 - example: image processing



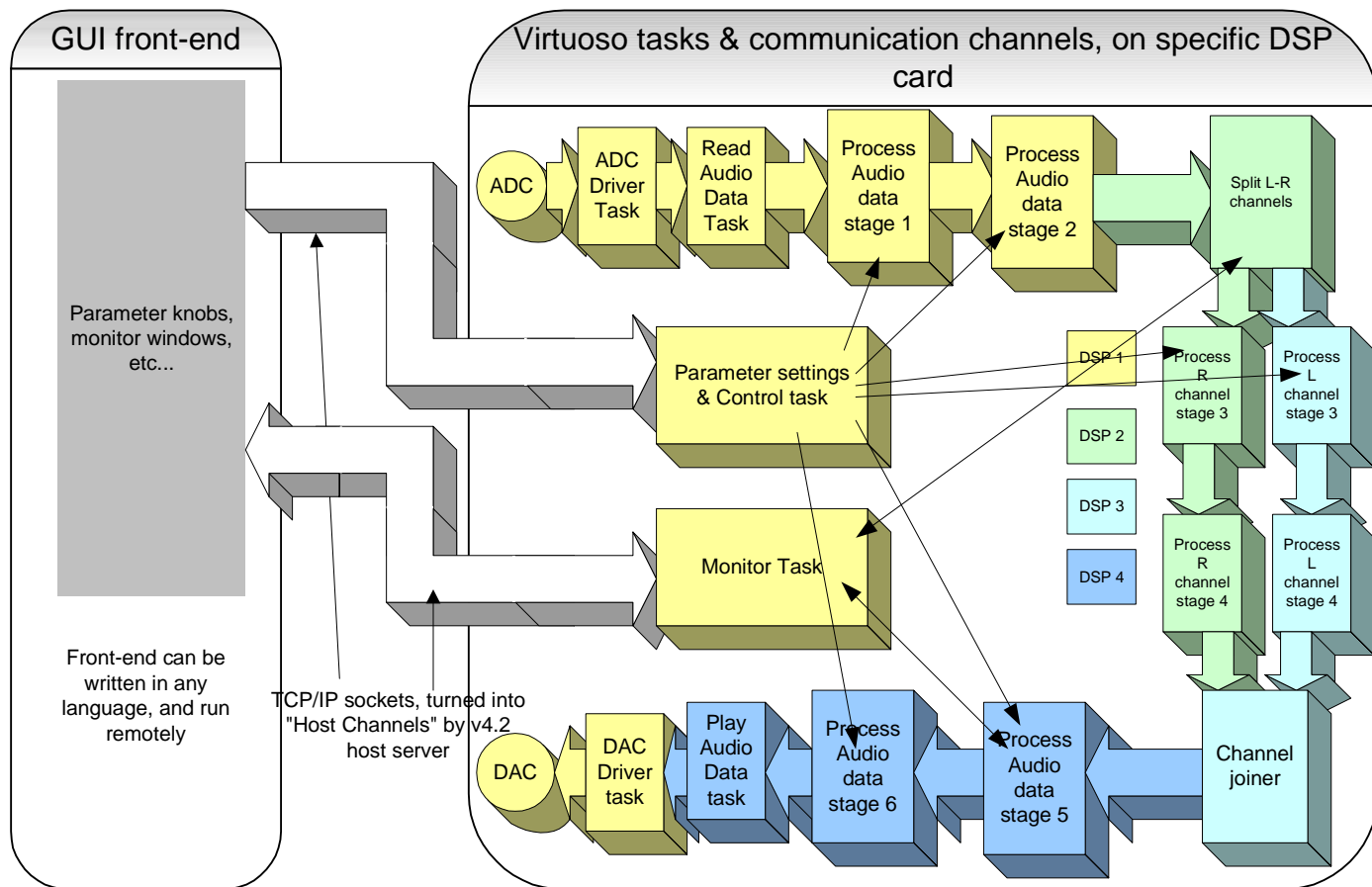
Scalable systems

- Add processing power where needed:
 - Possible by Virtuoso's non-connection oriented API



Full application : include GUI

- Embedded DSP app with GUI front-end

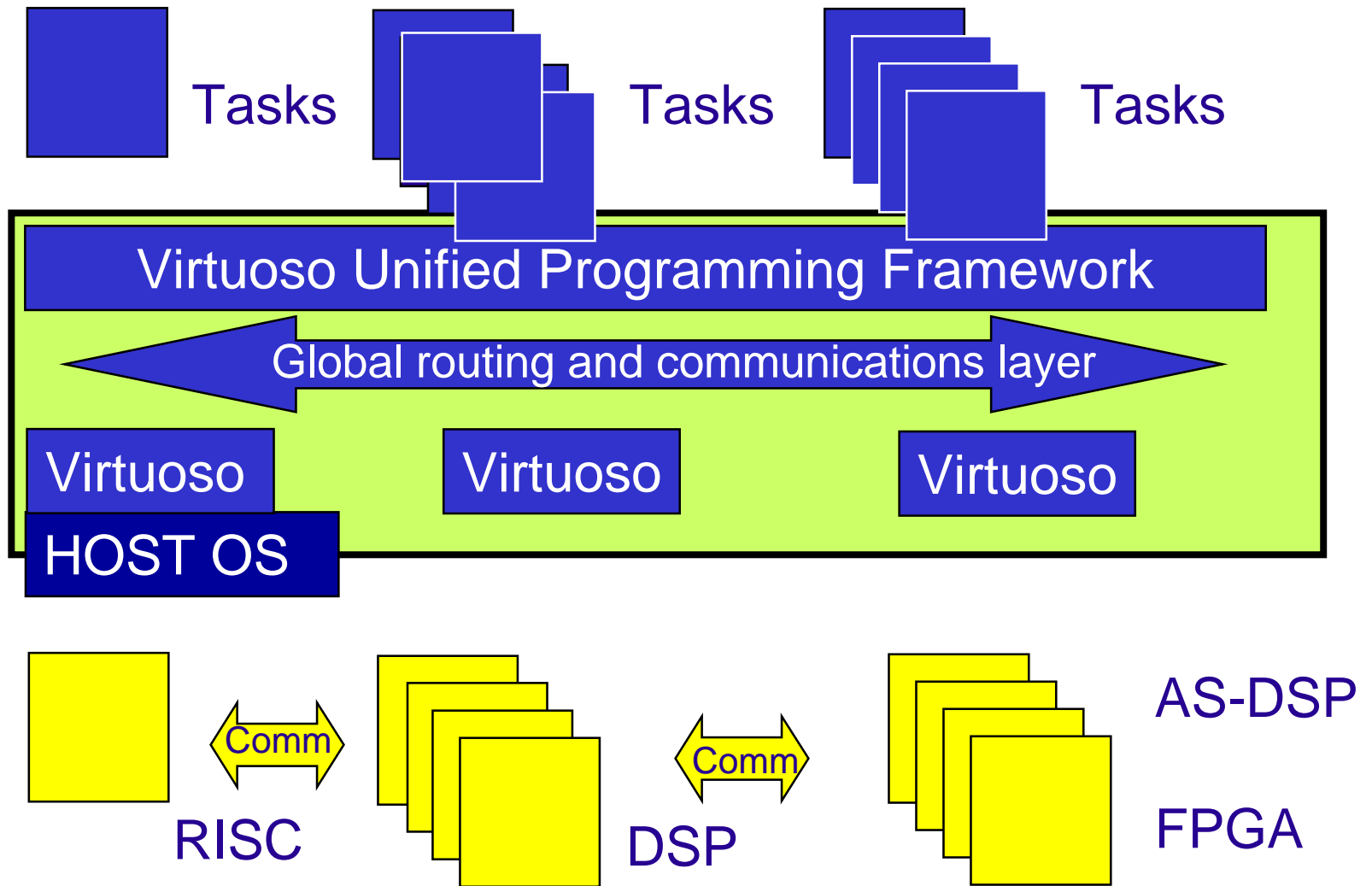




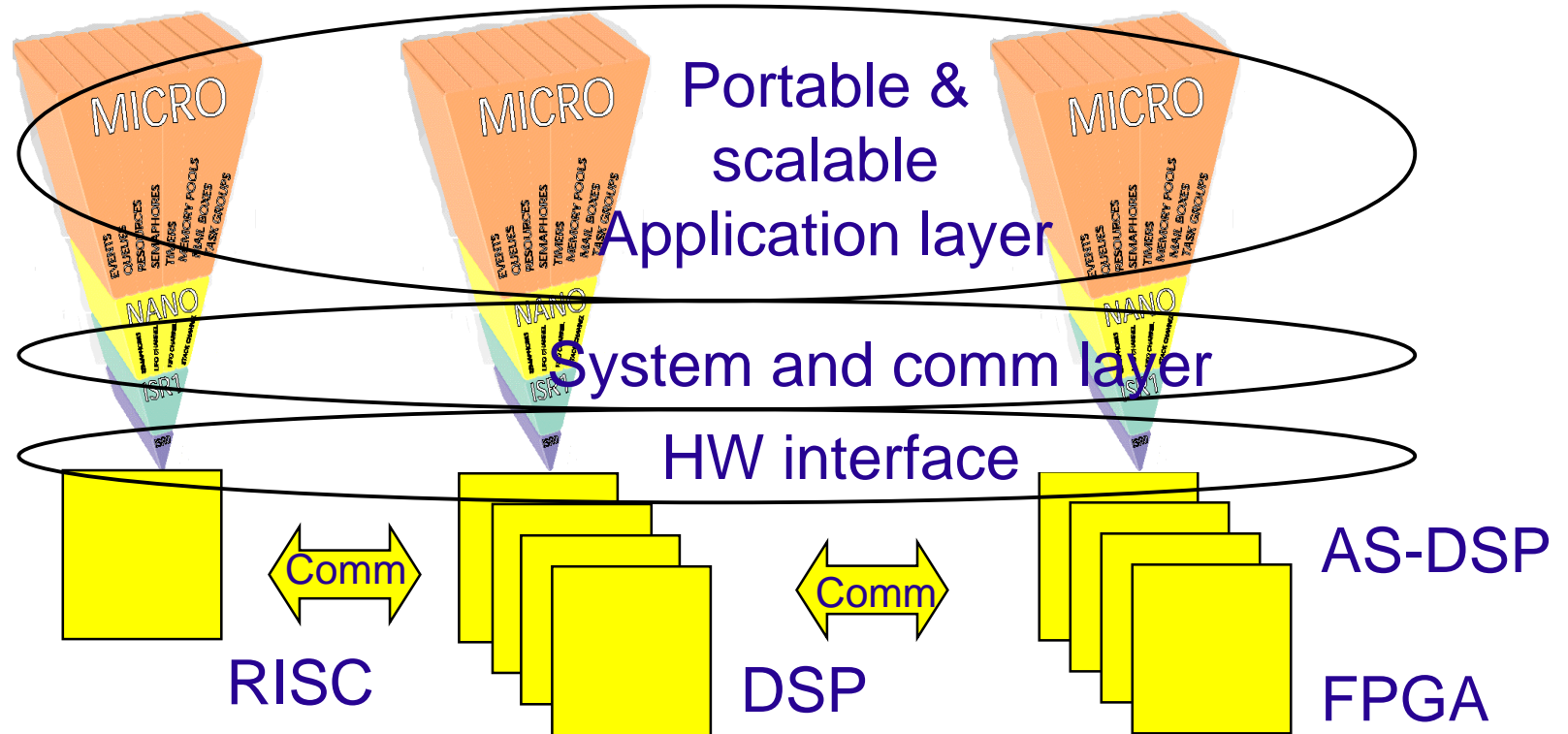
The solution for SoC: Virtuoso Multicore

- Multicore = VSP with a number of additions
 - heterogeneous: mix different processors together
 - co-operative: bolt VSP on top of other OSeS
 - extensions to AS-DSP
 - Generate RTOS primitives from spec
 - Add HW development part

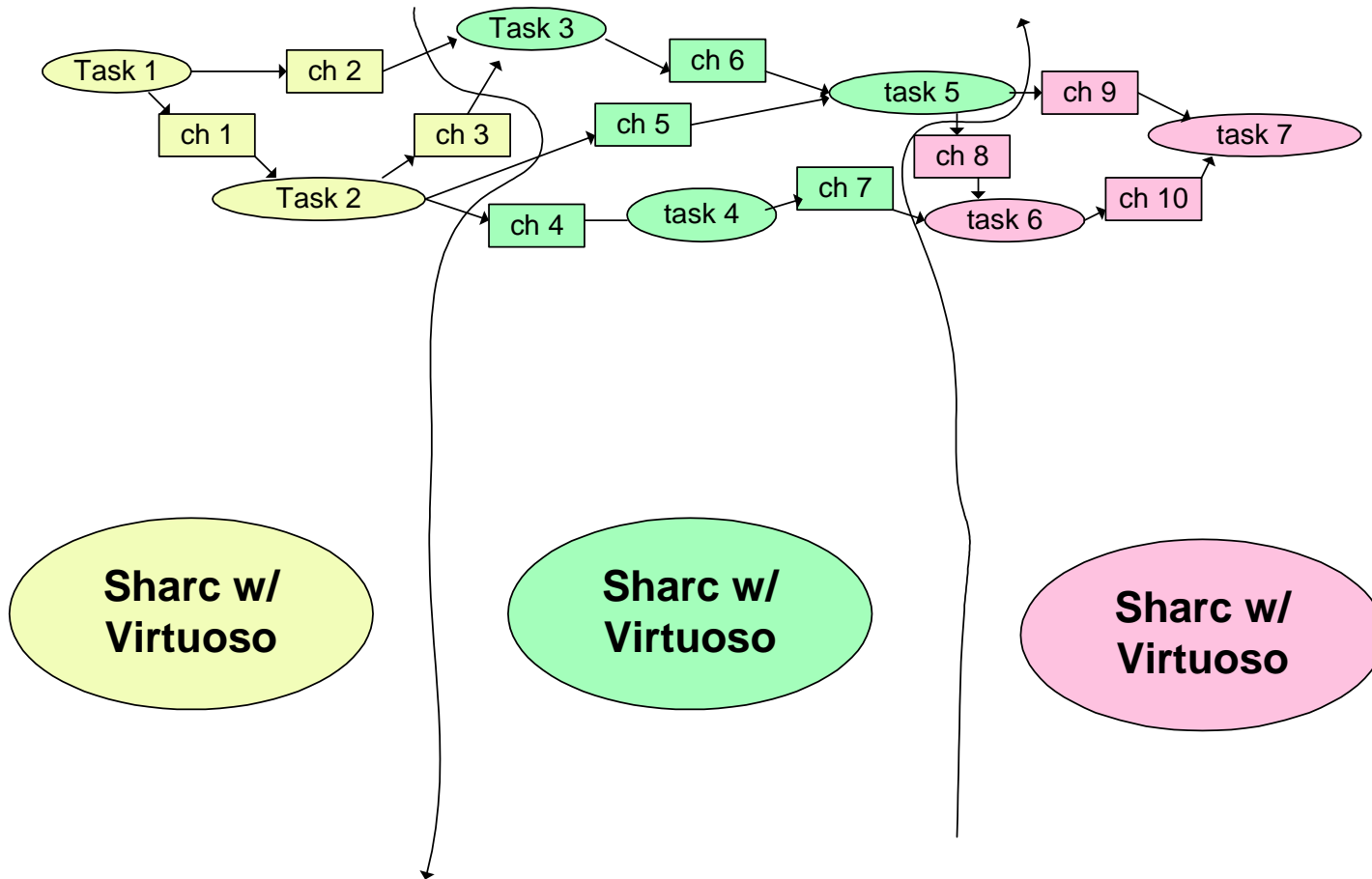
Virtuoso VSP is the only available framework for heterogeneous multicore ASIC / SoC



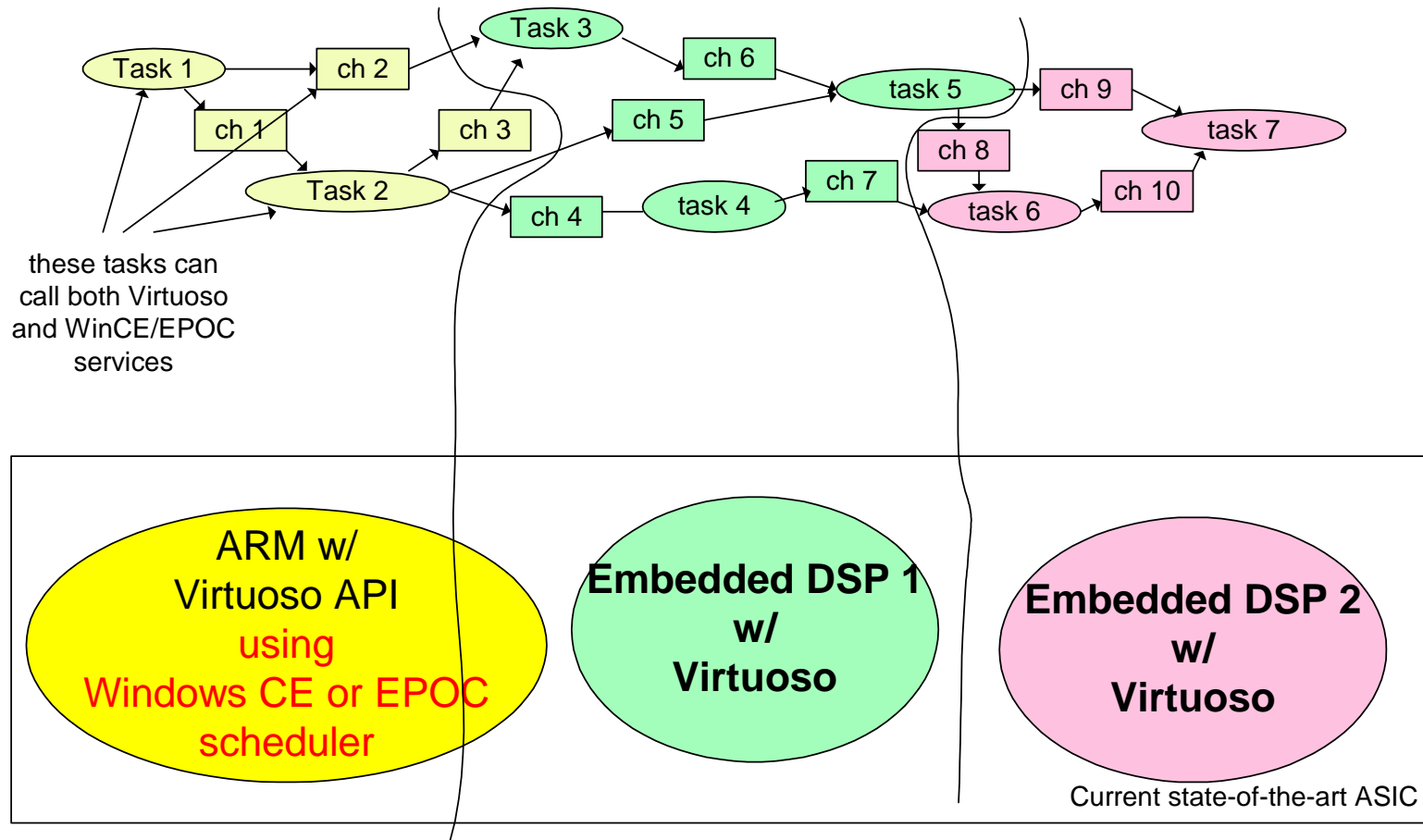
The MP-SoC-RTOS architecture



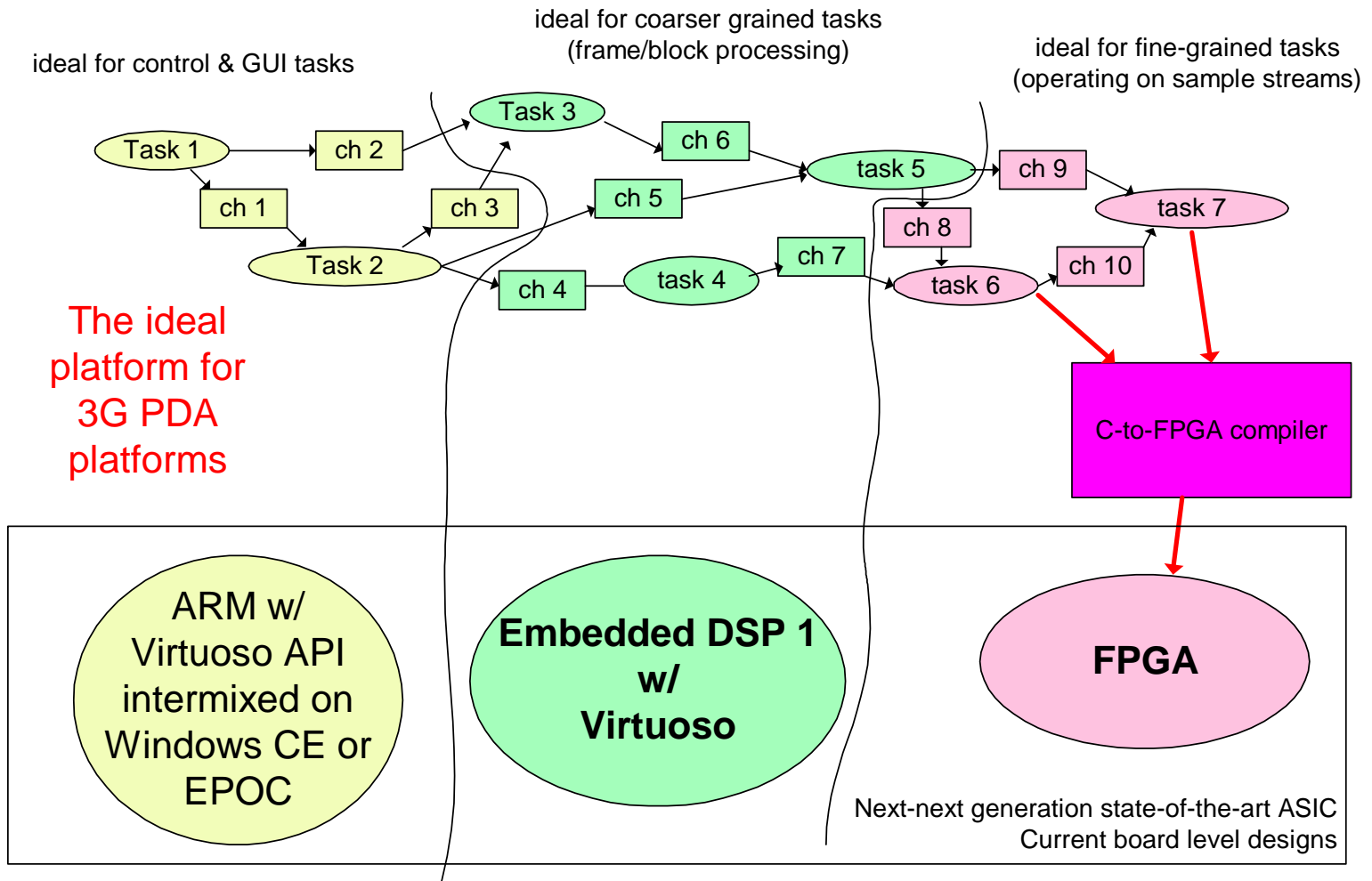
Today : Virtuoso VSP off-the-shelf



Today : Heterogeneous VSP with host OS



Tomorrow : SOC or next generation DSP Heterogeneous VSP with reprogrammable HW



MP-SoC today : lack of on-chip comm

- MP requires low latency, comm network
- Status today :
 - AMBA (ARM and other) : designed for 1 RISC + peripherals
 - CoreConnect (IBM) : idem
 - VSIA : idem
 - Others : SuperHyway (Hitachi), FISP (Mentor), PI-bus (OMI), FPI (Tri-Core), FPGA (do-what-you-want)
 - Exception : SONICS (“SiliconBackplane”) : interface and access timings synthesized at design time
- Needed in future :
 - local bus + on-chip point to point network
 - crossbar for very large multicore
 - arbitration in order of priority
 - benefits : link synchronuous blocks asynchronously

The needs for an on-chip standard communication backbone

- Best example until now SiliconBackplane from Sonics
- Maybe even better : GeodeLink (courtesy Kees Vissers)
- Or IEEE1355 (SpaceWire) : also fault-tolerant
- Hardware level :
 - standard at electrical level (FIFO, DMA)
 - generates of “stub” interface
- Software level :
 - standard control and data interface
 - minimum functionality
 - FIFO buffers
 - allows also template drivers to interface with hardware stubs

Another hurdle to overcome...

- HW centric thinking of SoC and DSP developers
 - simulate/prove everything at the cycle level
 - System level C language proposals are really for hardware design using C...
- A truly SW centric approach is necessary to efficiently program next generation SoC systems
- Also SW engineers need to learn to think parallel
- Major benefit :

keep same source code from [specification], simulation, development to implementation, re-mapping, upgrading and re-using blocks

Conclusion

- RTOS is much more than real-time
- DSP systems are heterogeneous
- So will future SoC component
- Hide complexity inside chip for hardware (in SoC chip)
- Hide complexity inside task for software (with RTOS)
- RTOS comes from industrial experience
- CSP provides unified theoretical base for hardware and software, RTOS makes it pragmatic for real world :
 - “DESIGN PARALLEL, OPTIMIZE SEQUENTIALLY”
- Software meets hardware with same development paradigm
- FPGA with macro-blocks is pre-cursor of next generation SW defined SoC : needs SW development paradigm
- Time for asynchronous HW design ?