Altreonic NV

www.altreonic.com

Eric.Verhulst @ altreonic.com

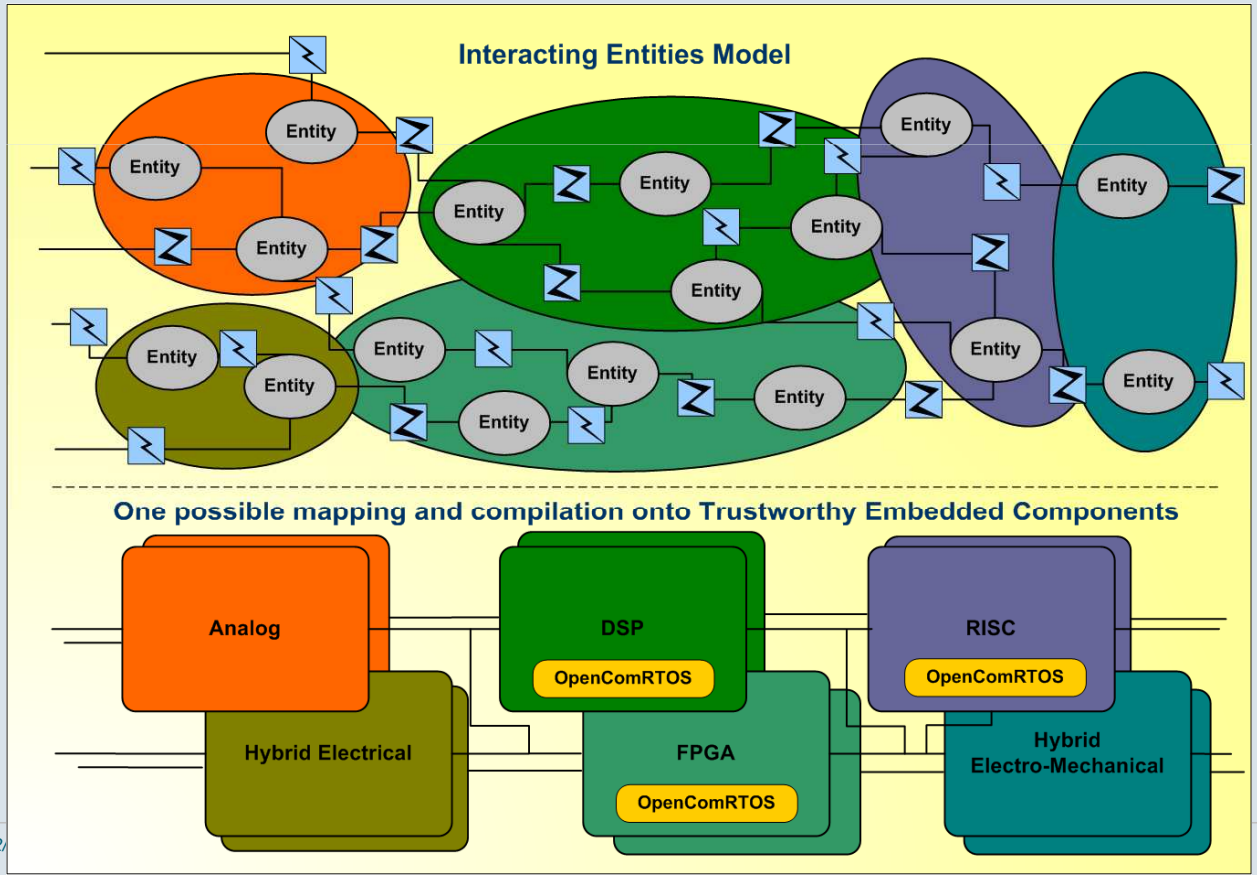
Present and future challenges in developing a manycore RTOS

Altreonic history

- History goes back to Eonic Systems NV
  - *Background in CSP and transputers*
  - *Developed parallel DSP Virtuoso RTOS*
  - *Acquired by Wind River Systems in 2001*
- Open License Society (R&D) 2004
  - *Developing a formalized systems engineering methodology*

    Unified semantics + interacting entities
  - *Formally developed network-centric OpenComRTOS.*
- Altreonic is a spin-off of OLS

The OpenComRTOS goal: program once, run anywhere

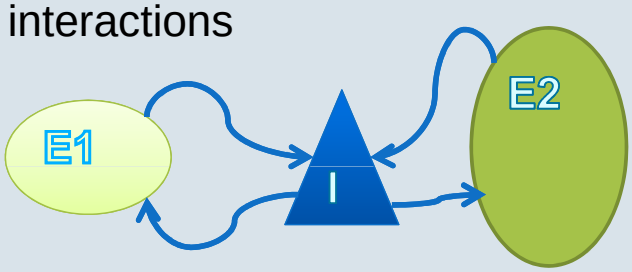


Von Neuman meets Moore

- **Moore's law**:
  - Functionality potentially increases with geometry shrinking
  - We can now easily put milllions of gates on mm2
  - Smaller delays => faster clocking
- BUT:
  - Works best for pipelined processing (= data flow driven)
  - Most applications also have control flow (= multiple flows)
    - Meet von Neuman!
    - Reuses logic elements on chip
    - Stores flow sequence in memory (PC := PC + 1)
    - Flow can be dynamically changed (PC' := PC + X)
- BUT:
  - Programming at that level is tedious
  - Hence: Higher Level programming Languages to add abstraction
  - BUT HLL reflect sequential nature of underlying hardware
- Meet the **von Neuman syndrome**:
    - Programmers think sequentially because of HLL
    - Whereas programs are models of a concurrent (real) world
    - How to program manycore chips with 2, 4, n (heterogeneous) cores?
    - How to program networked systems?

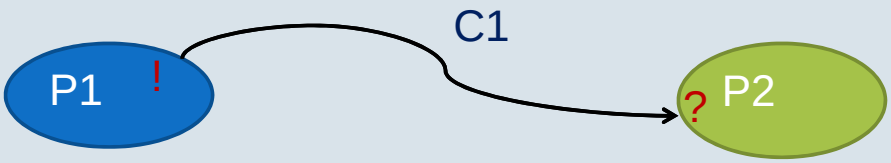
Software doesn't execute hardware

- If programs are models, we need hardware that has the same semantics
- Sequences, concurrency, communication, ….
- Question: which is the common model?
  ⇒**Interacting Entities** as metamodel
  – Entities:
    - can regroup other entities + interactions
    - basic entity is sequential
    - behaviour is internal
  – Interactions:
    - synchronisation points
    - behavior as a consequence
    - pass (state) information between entities
    - can be considered as a subtype of entity



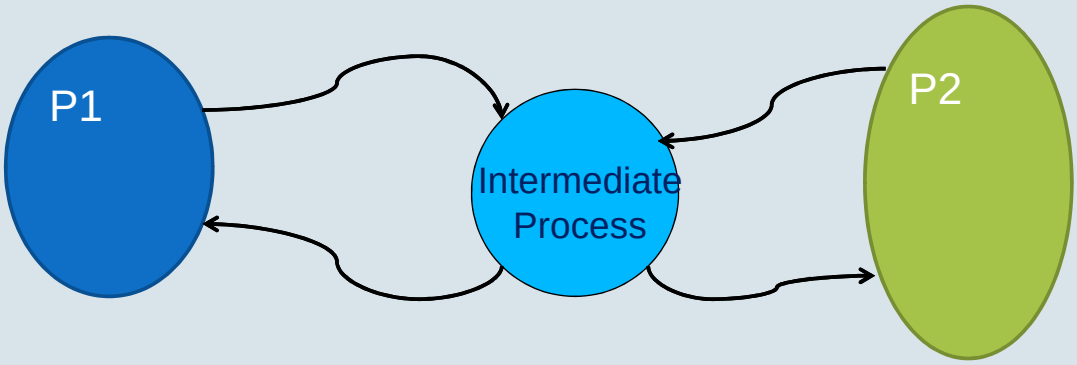
Which formalism?

- Von Neuman machines can be formally modeled by state graphs
- Useful formal model is CSP process algebra
  – **C**ommunicating **S**equential **P**rocesses
  – Variations exist (CCS, …)
  – = Processes + channels
    - Process: sequential segments, connected by channels
    - Channels:
      – Synchronisation events
      – Data passed as side-effect



Synchronising channels in CSP

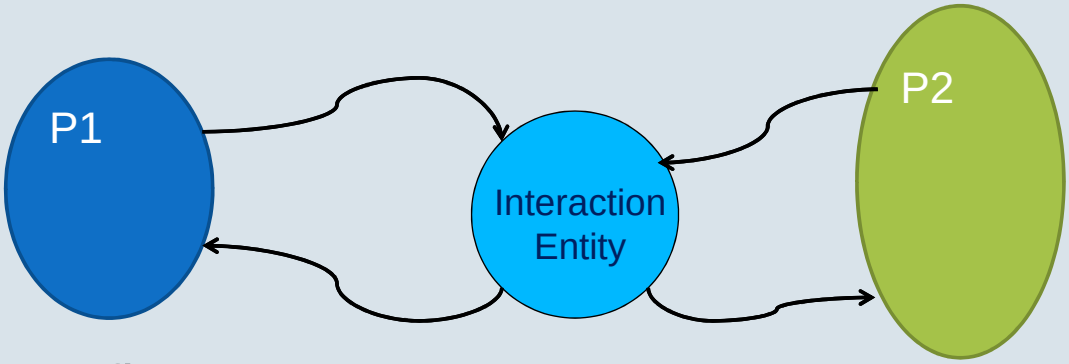
- CSP semantics only express basic semantics



- What about n-n interactions?
- What about composeability?
  - Can be expressed by adding extra processes
  - Tedious and had to maintain
- => more abstraction needed

Beyond CSP channels

- Decouple interaction from process behaviour



- Benefits:
  - scalability, orthogonality, location independence
- Question: what are good semantics?

Common interaction semantics

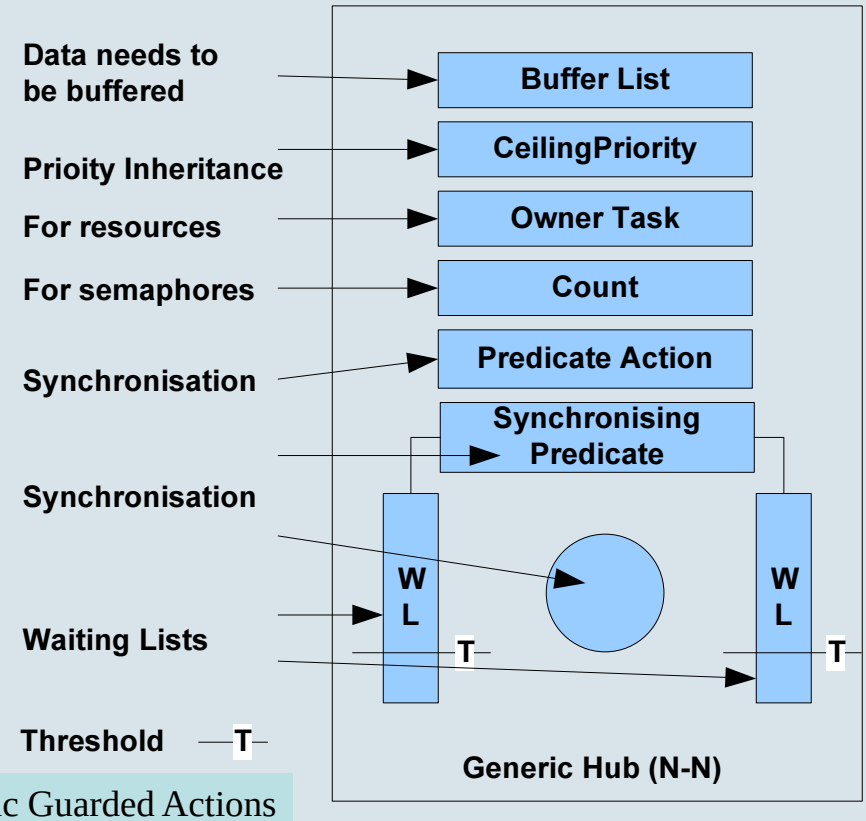
- Most OS have vaguely common types of services: events, semaphores, mutex, pipes, …
- Is there a better way to define them?
  ⇒ Formalisation

⇒ Result:
  – analysis shows that all such services share a common functionality
  – differences are small, often based on semantics

The OpencomRTOS “HUB”

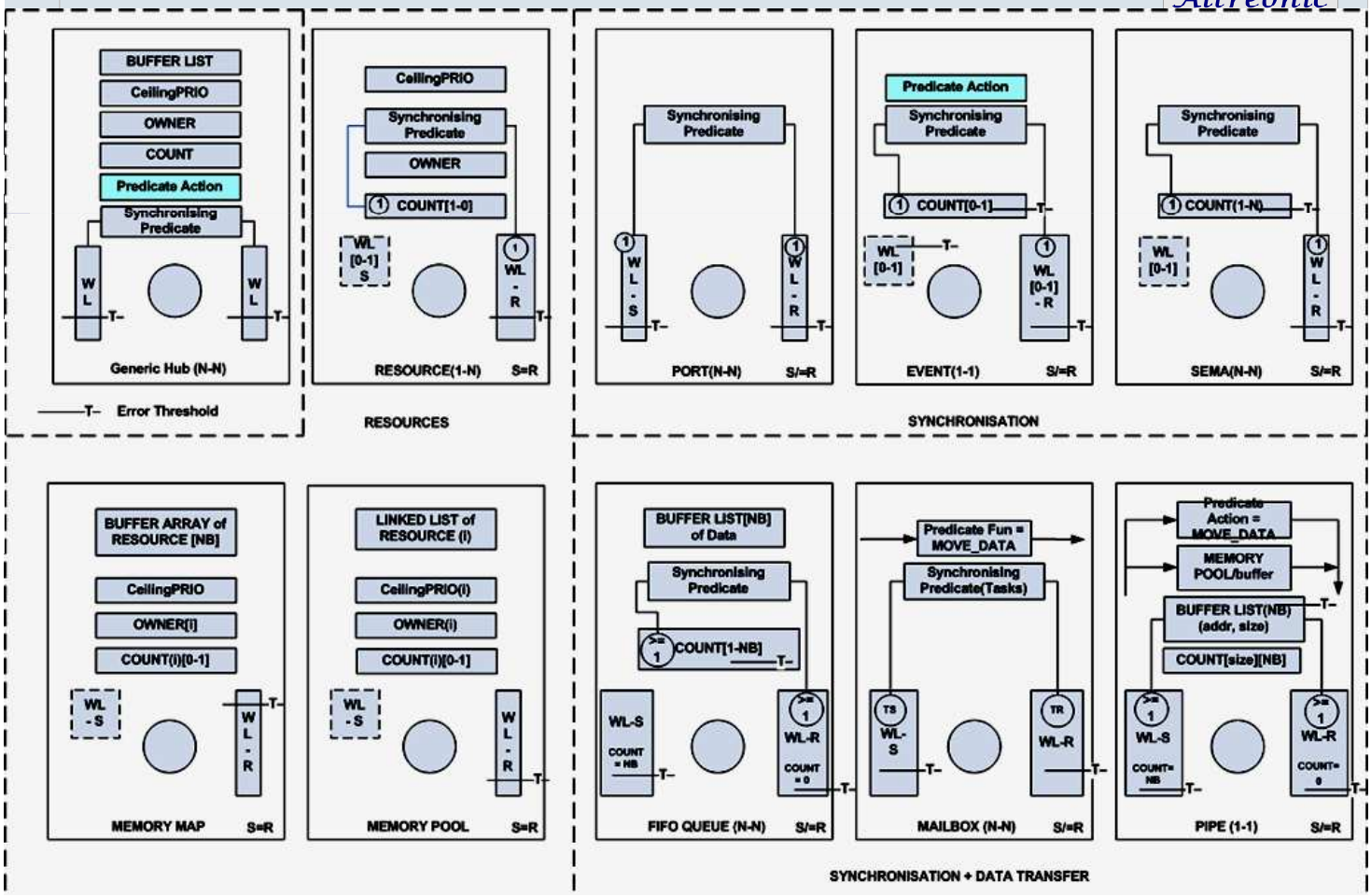
- Result of formal modeling (TLA+)
- Events, semaphores, FIFOs, Ports, resources, mailbox, memory pools, etc. are all variants of a generic HUB
- A HUB has 4 functional parts:
  – Synchronisation point between Tasks
  – Stores task’s waiting state if needed
  – Predicate function: defines synchronisation conditions and lifts waiting state of tasks
  – Synchronisation function: functional behavior after synchronisation: can be anything, including passing data
- All HUBs operate system-wide, but transparently:
  - Virtual Single Processor programming model
- Possibility to create application specific hubs & services!
  - => a new concurrent programming model

The generic hub as metamodel

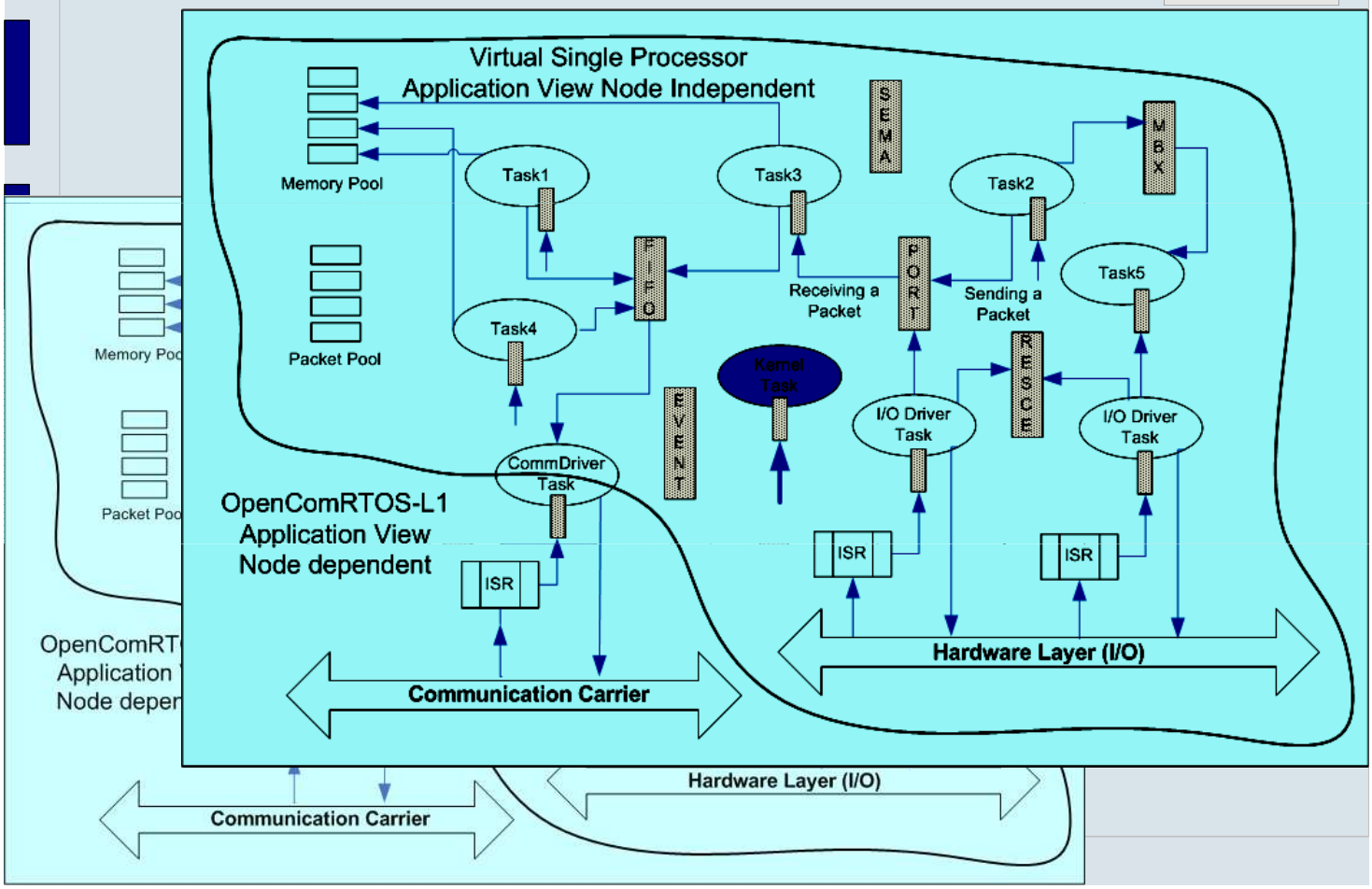


Similar to Atomic Guarded Actions
Or
A pragmatic superset of CSP

All RTOS entities are "HUBs"



Resulting programming model



What else is needed?

- **Scheduling of processes:**
  - RTOS => real-time scheduling
    - Priority based => priority inheritance needed
    - Deadline based: lack of HW support (no cycle counters)
  - Implies context switch
  - Implies critical sections
    - Context switch
    - Updating system datastructures
    - I/O and status bits
- **Communication:**
  - To pass data and control between task's context
  - To pass data and control between processor nodes

  ⇒ Latencies, delays and bandwidth

  ⇒ DMA, drivers, buffering

  ⇒ **Packet based architecture**

Critical resources management

- Dynamic applications have resource limits
  - => QoS based scheduling
- CPU as resource:
  - Priority based scheduling
  - But how to avoid trashing?
  - But how to avoid monopolistic blocking?
    - Priority inheritance is a must
    - Task specific cycle counters needed
- Memory as resource
  - Fragmentation and protection issues
- Bandwidth as resource
  - Arbitration issues
- Energy as a resource (V and F scaling)
  - Dependencies with timing properties

Hub as resources

- Is a hub a resource?
  - Semantics allow for “waiting” to synchronise
  - => waiting in order or priority
  - ? Should we apply priority inheritance?
- Correlate:
  - Hub can be used to model resource arbitration
  - Trade-off between software overhead and best real-time performance
  - => granularity issue
  - => smaller grain size depends on hardware support

Safety and security properties

- Use cases :=
  - Normal cases (see previous slides)
  - Test cases
  - Fault cases
    - Safety cases
    - Security case
- Test cases:
  - Remains task in predefined resource useage?
    - Stack space boundary violation
    - Memory access / boundary violations
    - Numerical exceptions

Safety and security properties

- Fault cases
  - Safety cases
    - Rule number one: even faulty software is predictable
    - Rule number two: software faults are really hardware faults
    - How to catch the faults before they become errors?
      - Localisation
      - Containment
      - Correction
      - Recover and restart
  - Security case
    - Maliciously induced faults
    - More complex as application related

$\Rightarrow$ requires a lot of software
$\Rightarrow$ little support in hardware

Performance goals

- Desktop, games:
  - Driven by peak performance
  - Throughput driven applications
  - Often soft real time
  - Cache speed vs. bulk external memory
  - Memory and power secondary issue (220 VAC)
- Mobility:
  - Serious constraints in
    - Memory, I/O, display, POWER (battery powered)
- Limiting factor is not silicon technology
  - But architecture
  - Computation/communication ratio (best =1)
    - Local: access to memory
    - Global: point-to-point latency and bandwidth

Unique technology

- Formalized, straightforward approach:
  - Makes safety engineering more affordable
  - Full integration of tools
- **OpenComRTOS** unique features :
  - Network-centric RTOS:
    - Transparent distributed processing
    - From many-core to WLAN systems
  - Formally developed and verified
  - Scalable yet very small and complete
    - 5 to 10 Kbytes/node
  - Real-time communication support
  - Heterogeneous target support

***OpenComRTOS** was one of the three final nominees for the:*

embedded AWARD 2009

embedded world 2009 Exhibition&Conference

- Open license model: better than open source

Codesize Figures

Service	MLX-16	MicroBlaze	Leon3	ARM	XMOS
L1 Hub shared	400	4756	4904	2192	4854
L1 Port	4	8	8	4	4
L1 Event	70	88	72	36	54
L1 Semaphore	54	92	96	40	64
L1 Resource	104	96	76	40	50
L1 FIFO	232	356	332	140	222
L1 PacketPool	NA	296	268	120	166
Total L1 Services	1048	5692	5756	2572	5414

Code size figures (in Bytes) obtained for our different ports, compiled with Optimisation Os

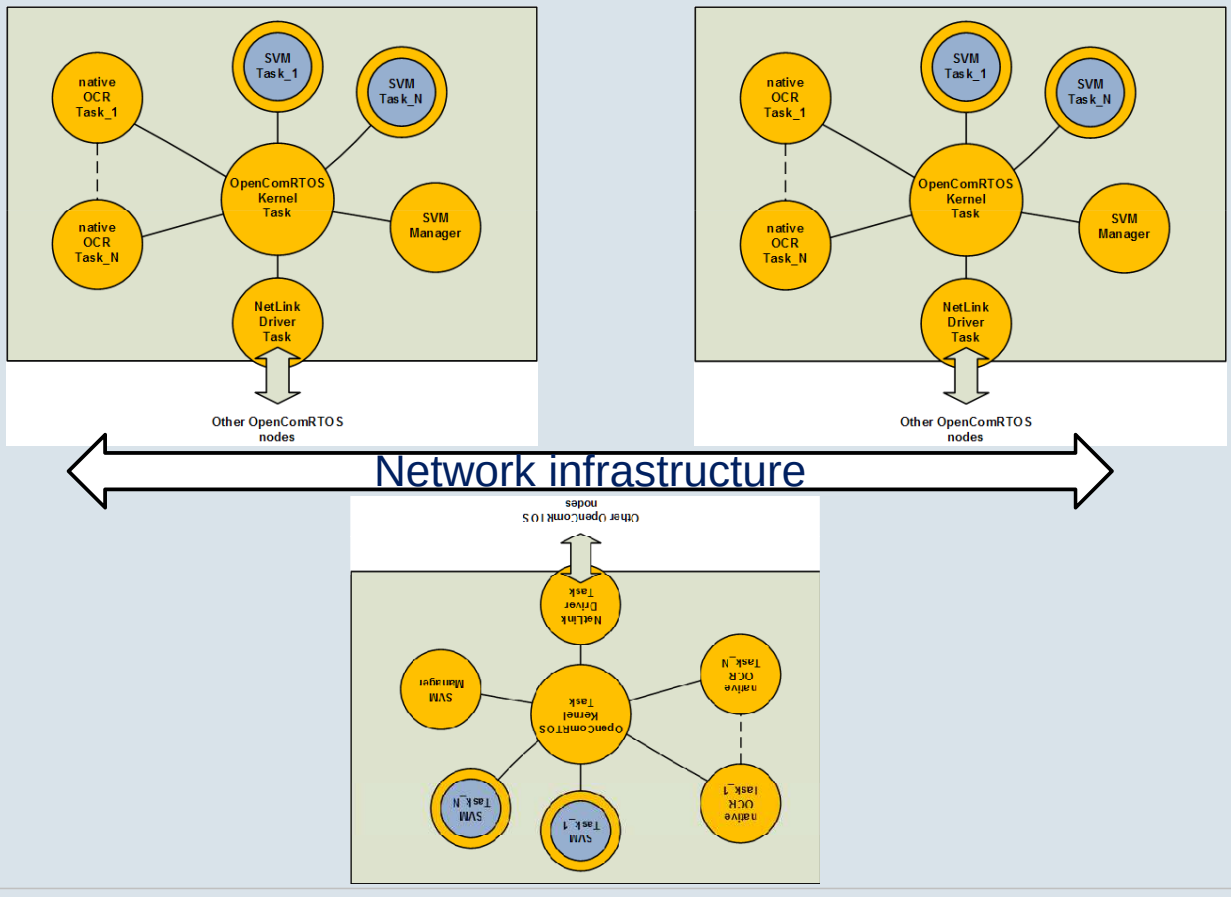
Program once, run anywhere

- Ultra low power:
  - CoolFlux DSP core (24bit, Harvard)
  - Code size full kernel: 2000w PM + 750w data
  - Interrupt latency:
    - IRQ to ISR: < 112 cycles
    - IRQ to task: < 877 cycles
  - Multicore capable
- Single chip multicore
  - Intel SCC 48core “super computer on chip + NoC switch” (in development)
- Heterogeneous networked targets:
  - Win32+Linux+ARM+MicroBlaze+XMOS+LEON3+ … demo programmed as single target

A Safe Virtual Machine for C

- Goal:
  - CPU independent programming
  - Low memory needs (embedded!)
  - Mobile, dynamic code
- Results:
  - Selected ARM Thumb2 instruction set of VM target
    - Compactness
    - Widely used CPU
    - **3.8 Kbytes of code for VM**
    - Executes binary compiled code
    - Capable of native execution on ARM targets
    - VM enhanced with safety support:
      - Memory violations
      - Stack violations
      - Numerical exceptions

Safe VM set-up



Conclusion

- We need hardware that executes software
- Even if dominating paradigm is the other way around (at least for embedded)
- Silicon gates are almost free, so why not use them?
- Nevertheless:

  OpenComRTOS project has proven that a universal concurrent programming paradigm works:
  - Very small code size, yet very scalable
  - Heterogeneous for CPU and communication media
  - Greatly due to formal(ised) development

www.altreonic.com

Eric.Verhulst @ altreonic.com