Towards Using OpenMP in Embedded Systems

OpenMPCon 2015

RWTH Aachen University, Germany

Eric Stotzer

Texas Instruments

Introduction

- Software for embedded systems is increasing in complexity.
- Can OpenMP be used as a programming model that can cope with this complexity?
- Embedded systems have constraints such as real-time deadlines and limited memory resources.
- Embedded Systems can be broadly classified as:
  - Event-driven
  - Compute and Data intensive
- Can the OpenMP tasking model be extended to support an event-driven programming model?
- Embedded Multi-Processor System on Chips are integrating increasing numbers of heterogeneous processors.
- Can the OpenMP accelerator model become a generalized MPSoC programming model?

References and Acknowledgements

- Dr. Barbara Chapman's High Performance Computing and Tools group at the University of Houston and their work with TI and the multicore association.
- W. Wolf, *Computers as Components*, 2nd Ed., 2008.
- E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, 2011.
- R. Oshana, *DSP Software Development Techniques for Embedded and Real-Time Systems*, 2006.

Agenda

- Background on Embedded Systems
- OpenMP in Embedded Systems
- Event-driven model
- Multi-Processor System-on-Chip (MPSoC) model
- Summary and Conclusion

Towards OpenMP in Embedded Systems

Characteristics of Embedded Systems

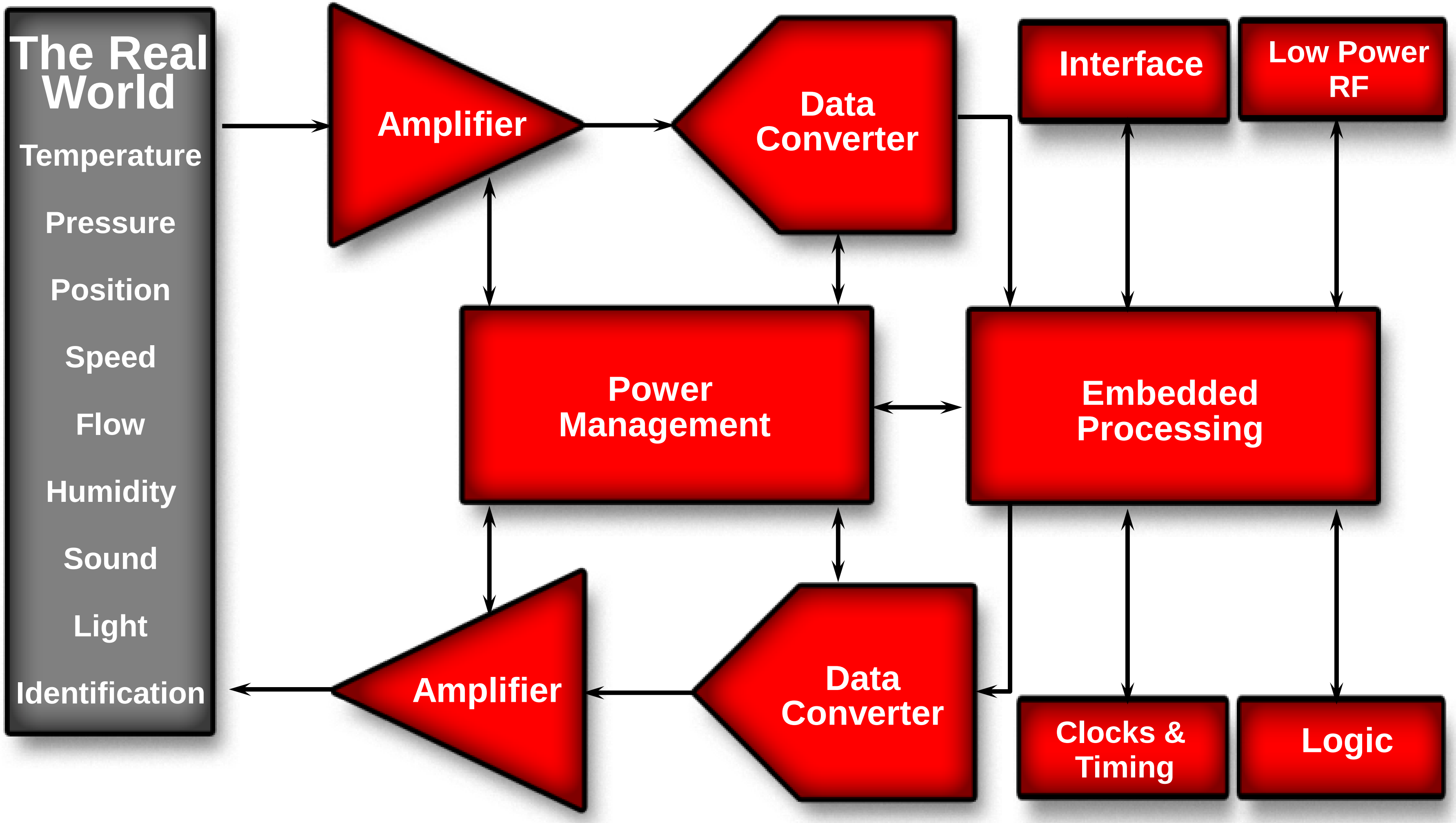
Embedded Processing is all around you

From digital communications and entertainment to medical services, automotive systems and wide-ranging applications in between.

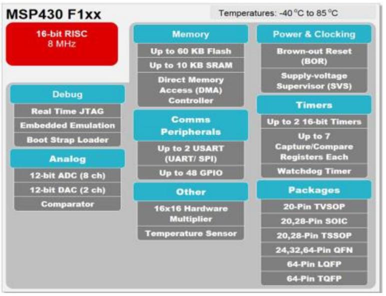
Characteristics of Embedded Systems

- Computers whose job is not primarily information processing, but rather is interacting with physical processes. [Lee and Seshia]
- An embedded computing system is any device that includes a programmable computer but is not itself a general-purpose computer. [Wolf]
- Take advantage of application characteristics to optimize the design. (don't need all the general-purpose bells and whistles). [Wolf]
- Real-time systems: processing must keep up with the rate of I/O.
  - Hard real time: missing deadline causes failure.
  - Soft real time: missing deadline results in degraded performance.
  - Multi-Rate: events occurring at varying rates
  - Performance is about meeting deadlines (finishing ahead of a deadline might not help)
- Operating environment constraints:
  - Power, Temperature, Size, etc…
  - Programs run forever

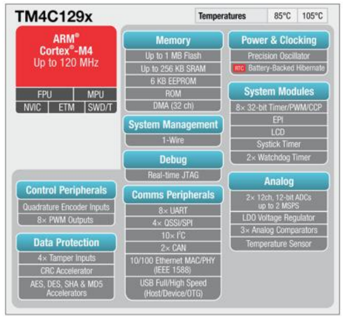
Embedded Systems Respond to Inputs from the Real World

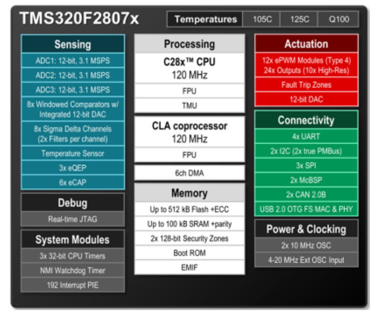


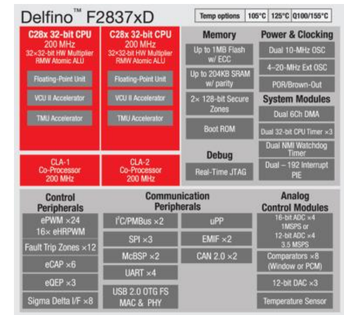
Embedded Platforms are Diverse

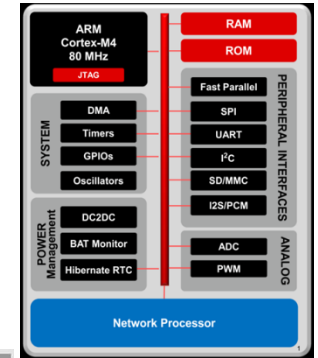


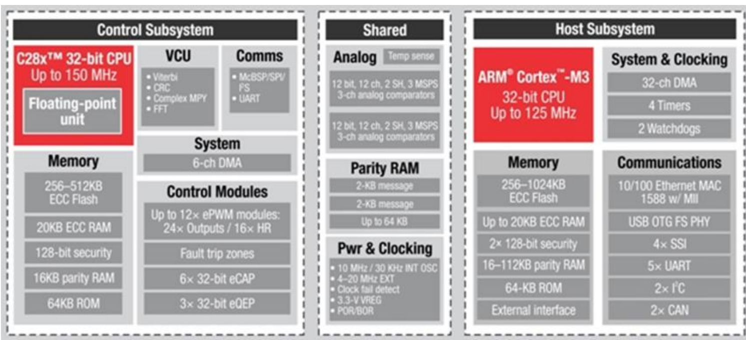
MSP430F1x Ultra-low-power Microcontrollers

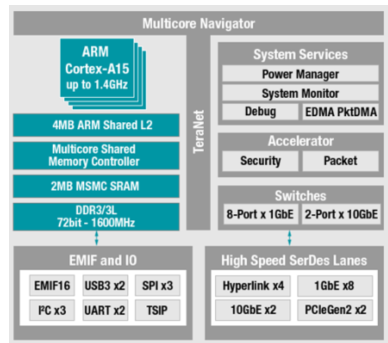


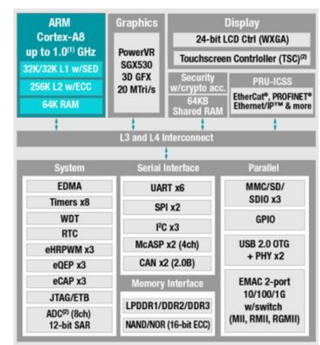


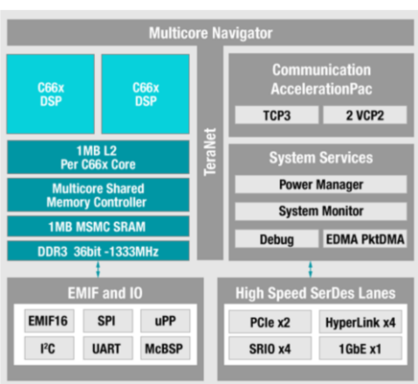








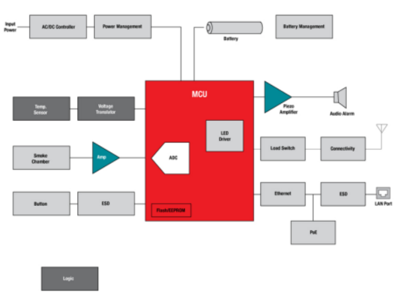


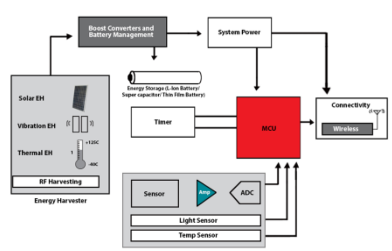


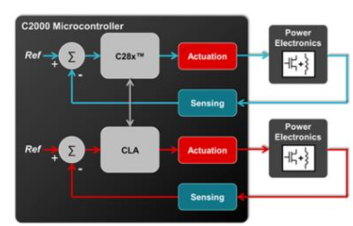
- Ultra-low power microcontrollers (MCUs)
- Mutliple Heterogeneous Cores Integrated onto a single Chip
- Arm processors capable of running SMP Linux
- Acceleration via DSPs, GPUs and hard accelerators
- I/O and peripherals targeted at specific application areas
- Processors dedicated for Real-Time control

Programming Embedded Systems

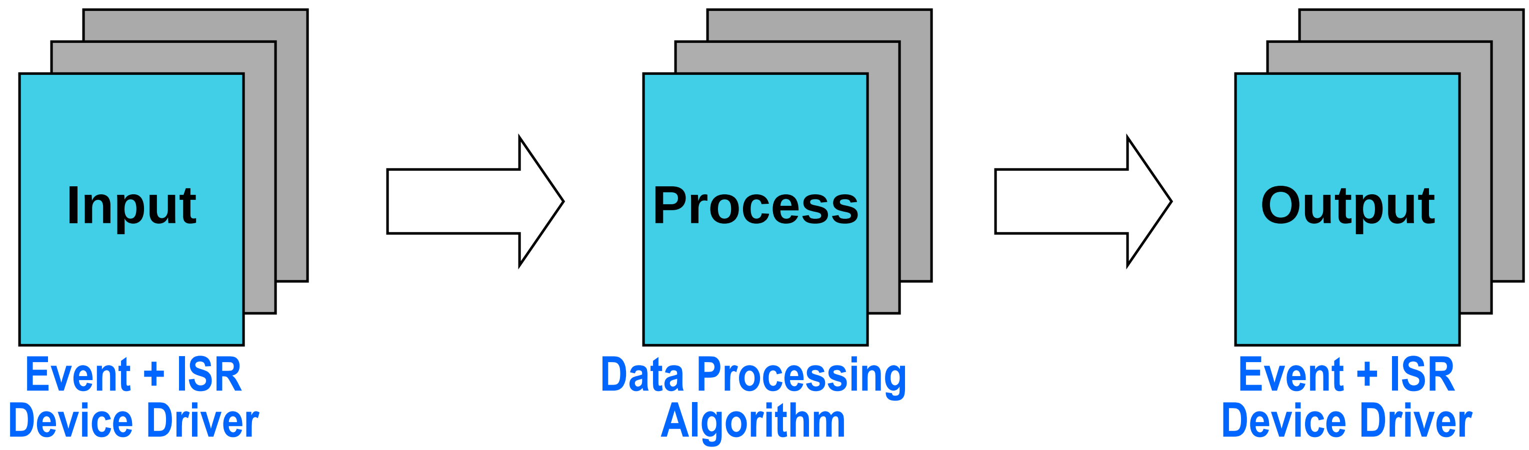
- Concurrency is intrinsic and not always about exploiting Parallelism
- Interaction with I/O peripherals and sensors
- Real-Time
- Timers and Interrupts
- Heterogeneous Memory Architecture (RAM, ROM, Flash, etc…)
- C Programming and Assembly Language
- All code in a new system is often re-compiled.
- Microkernels and Real Time Operating Systems (RTOS)







Embedded Processing Paradigm

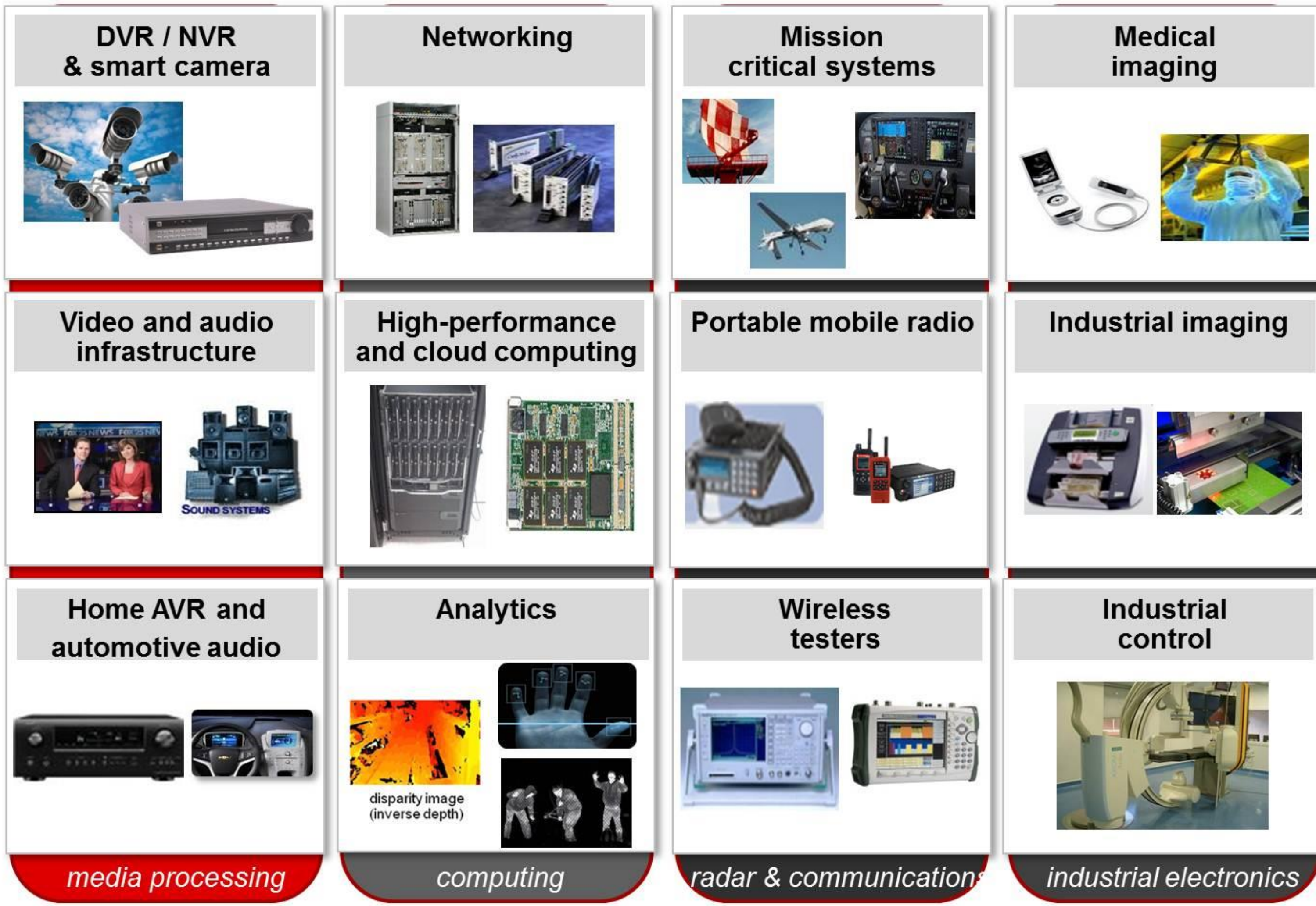


- Simple system: single I-P-O is easy to manage
- As system complexity increases (multiple threads) Needs an RTOS:
  - Can they all meet real time ?
  - Priorities of threads/algos ?
  - Synchronization of events?
  - Data sharing/passing ?

Towards OpenMP in Embedded Systems

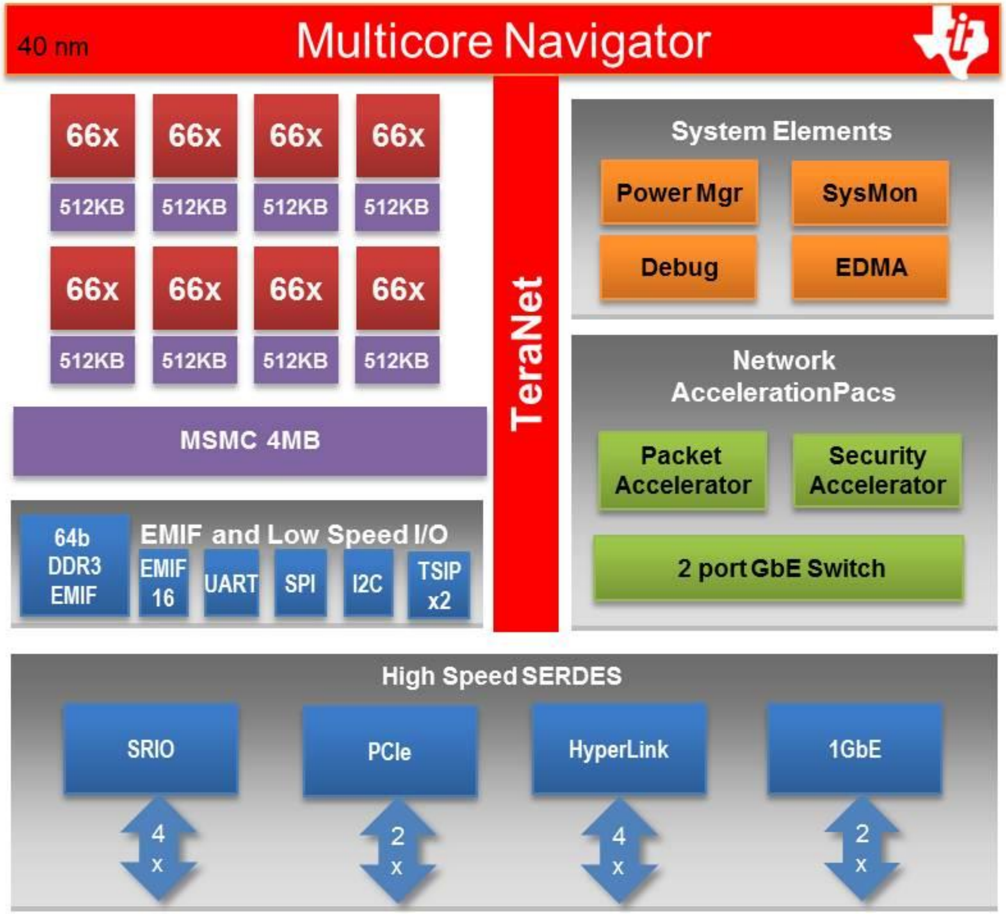
OpenMP in Embedded Systems

High Performance Embedded Computing



Keystone I: C6678 SoC

- Eight 8 C66x cores
- Each with 32k L1P, 32k L1D, 512k L2
- 1 to 1.25 GHz
- 320 GMACS
- 160 SP GFLOPS
- 512 KB/Core of local L2
- 4MB Multicore Shared Memory (MSMC)
- Multicore Navigator (8k HW queues) and TeraNet
- Serial-RapidIO, PCIe-II, Ethernet, 1xHyperlink



24mm x 24mm package

Why OpenMP?

- Traditional approaches:
  - Manually partition workloads to individual cores
  - Optimize partitioned regions for the core
  - This offers high entitlement

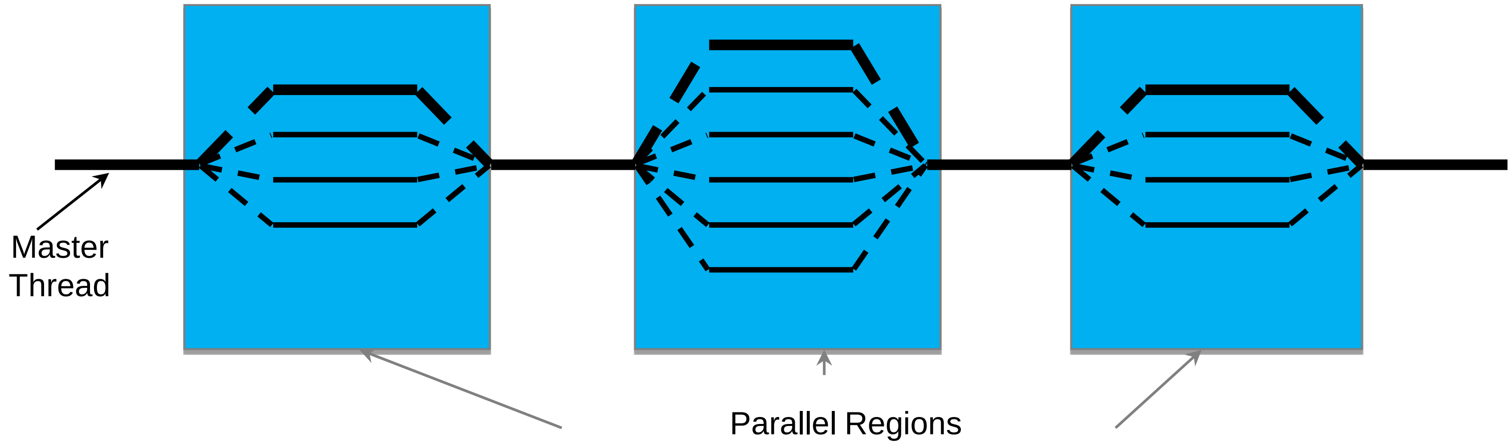
    BUT
  - Partition must be redone for each system configuration
  - Not portable
  - Developer needs detailed knowledge of SoC architecture
    - Increased time to market

- What OpenMP offers:
  - Modify code with pragmas and directives
  - Parallelization and load balancing are abstracted from the user
  - Easy and incremental
  - This offers high performance

    AND
  - Standard tools are portable to many architectures
  - SoC architecture details are abstracted from the developer
  - Data parallelization, task parallelization, accelerator offload, and more are all possible

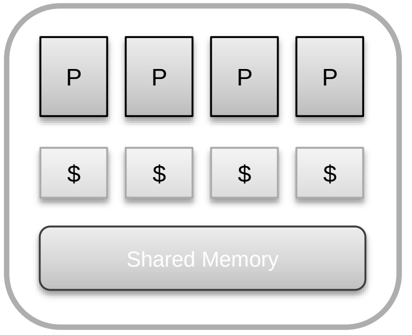
OpenMP Execution Model

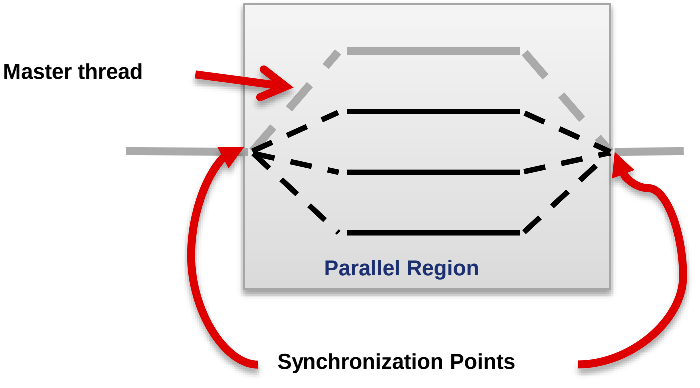
- Fork-join – master thread creates a team of threads on encountering a parallel region
- **Data Parallel** Work sharing constructs are used to distribute work among the team (e.g. loop iterations)
- **Task parallel** Task construct used to generate tasks which are executed by one of the threads on the team



OpenMP Memory Model

- Threads have access to *shared* memory
  - Each thread can have a temporary view of the shared memory (e.g. registers, cache)
  - Temporary view made consistent with shared view of memory at synchronization points
- Threads have *private* memory
  - For data local to each thread

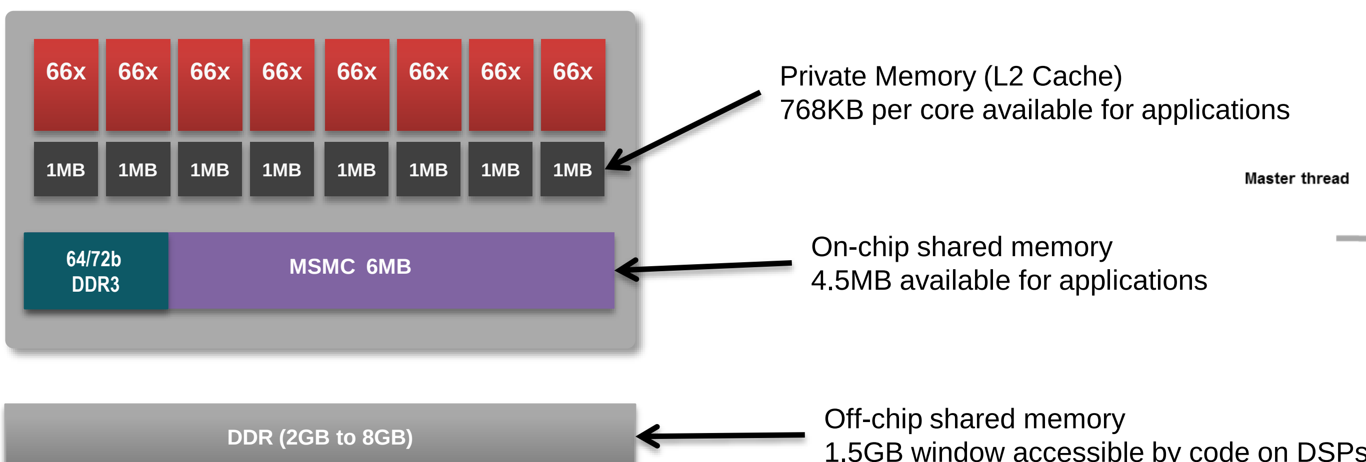


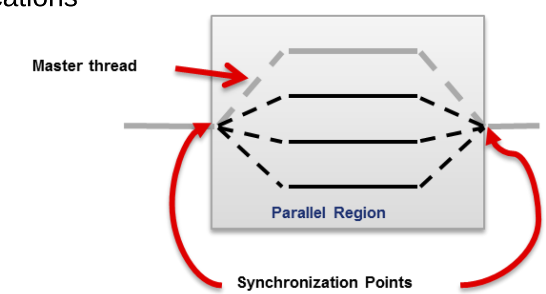


OpenMP on DSPs – Execution and MModel

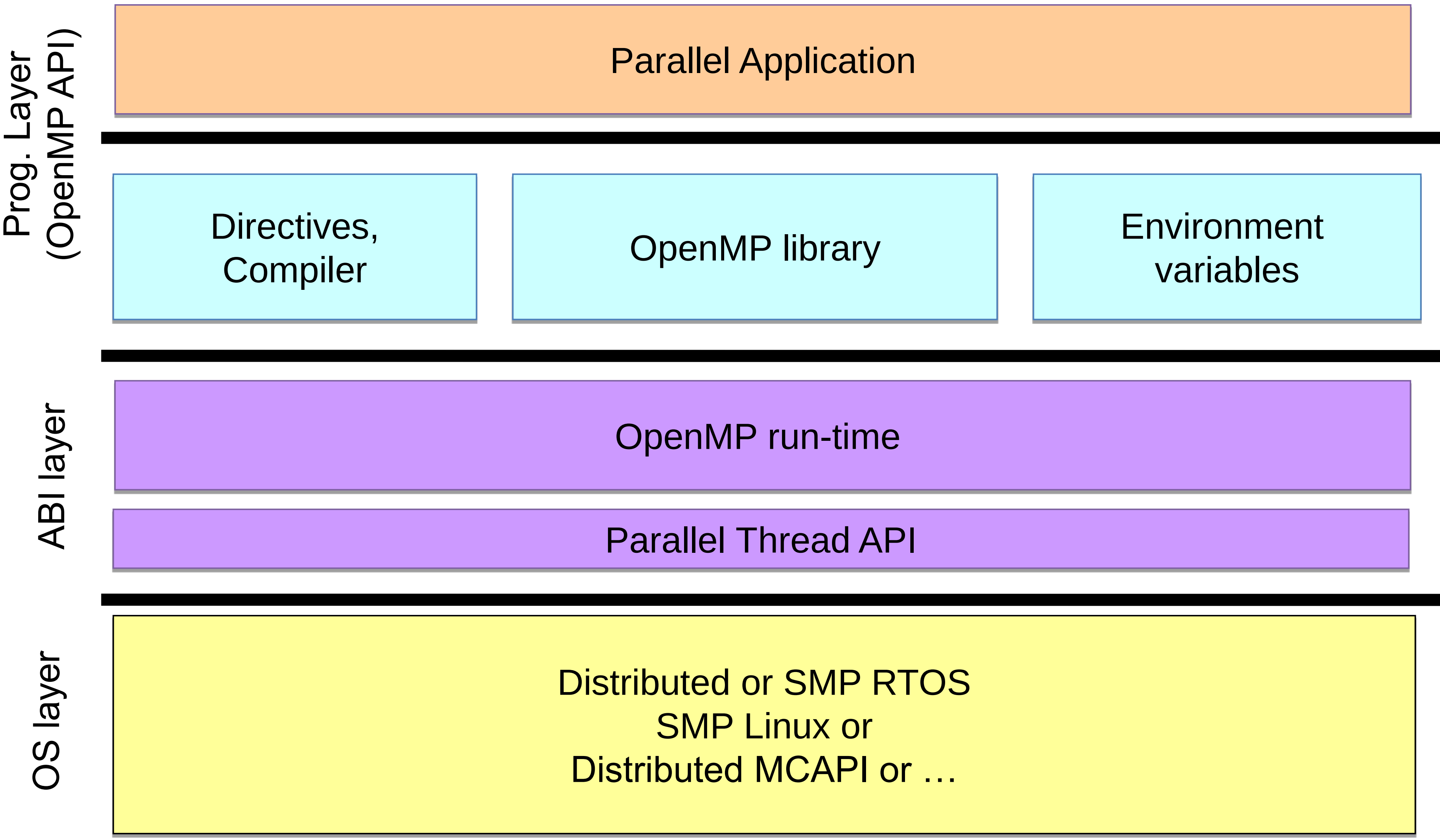
Execution Model:

- 8 C66x DSP cores, one thread per core
- Master thread begins execution on DSP core 0
- DSP cores 1-7 are worker cores, participate in executing the parallel region
- Runtime supports a maximum of 8 threads
- Nested parallel regions are executed by the encountering thread, no additional threads spawned
- No hardware cache coherency across DSP cores
- OpenMP runtime makes a thread's view of memory consistent with shared view by performing cache operations at synchronization points





OpenMP Solution Stack



OpenMP in Embedded Systems

- OpenMP can execute on an embedded RTOS or perhaps even “bare-metal”
- Shared memory:
  - precise hardware cache coherency is not required
  - Exploit weak consistency: implement hybrid software/hardware cache systems
- OpenMP can be successful in embedded systems:
  - Just like other high level languages have been adapted to embedded systems
- OpenMP is useful in embedded systems for the compute intensive parts of an application.
  - But what about the other parts of the program?

Towards OpenMP in Embedded Systems

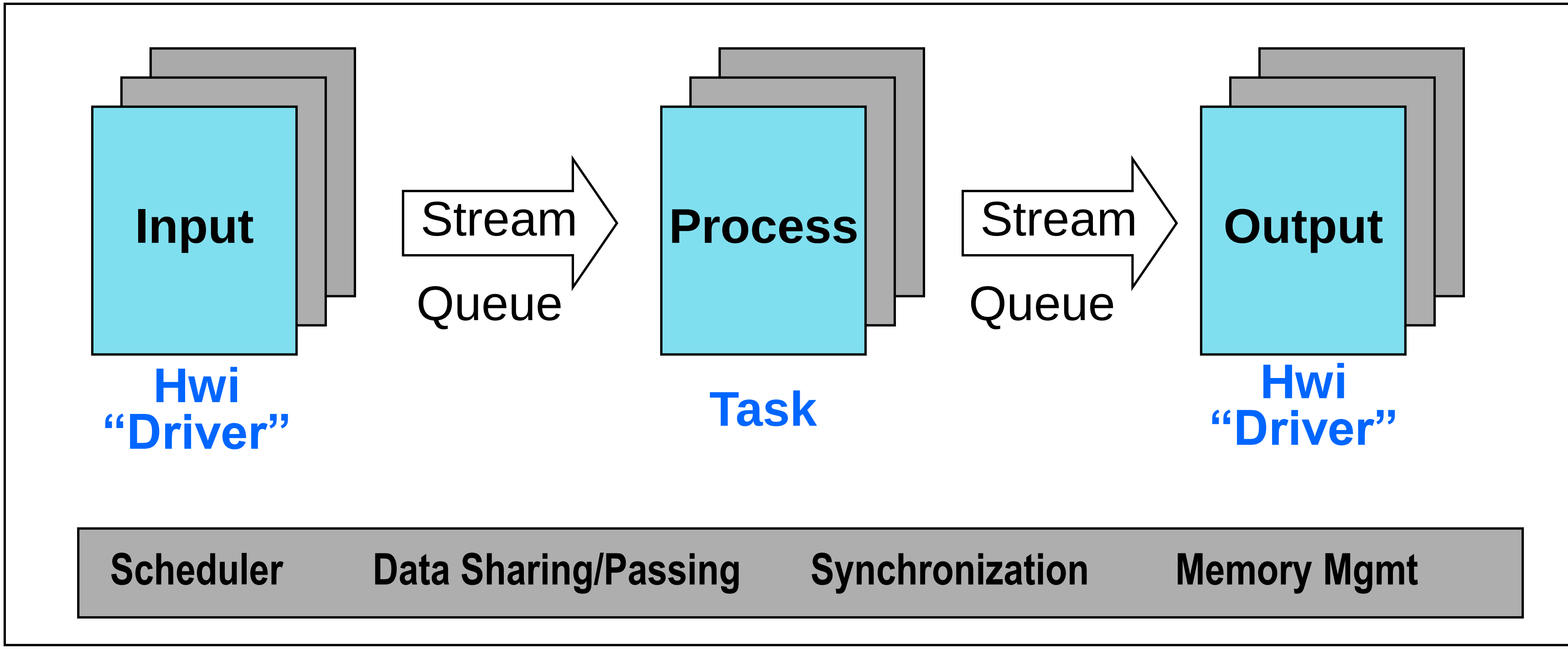
Event-Driven Models

Event Loop

- Embedded Systems respond to events.
- Events are typically inputs from external sensors or other actors in the system.
- The system must stay responsive while events are processed.
- Similar to the model used in GUI programming where an event is a mouse-click
  - See “Pyjama: OpenMP-like implementation for Java, with GUI Extensions”. [Vikas, Giacaman, Sinnen. PMAM 2013]

```
while (1)
{
    event = get_event();
    switch(event)
    {
        case EVENT1:
            process_event1();
            break;
        case EVENT2:
            process_event2();
            break;
        case EVENT3:
            process_event3();
            break;
    }
}
```

Event Driven running on a Real-Time O/S (RTOS)

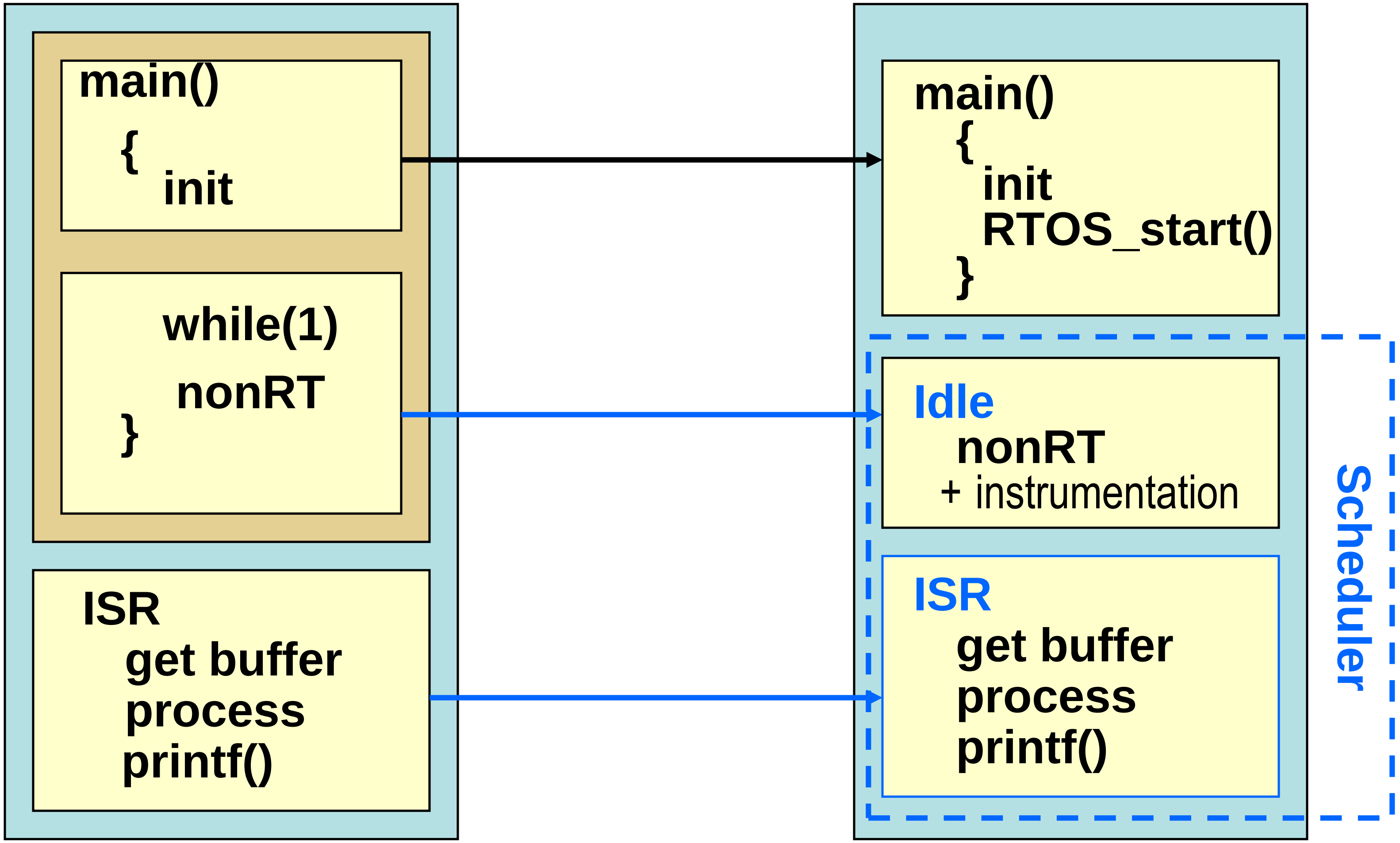


- **Pre-emptive Scheduler to design system to meet real-time (including sync/priorities)**

RTOS vs GP/OS

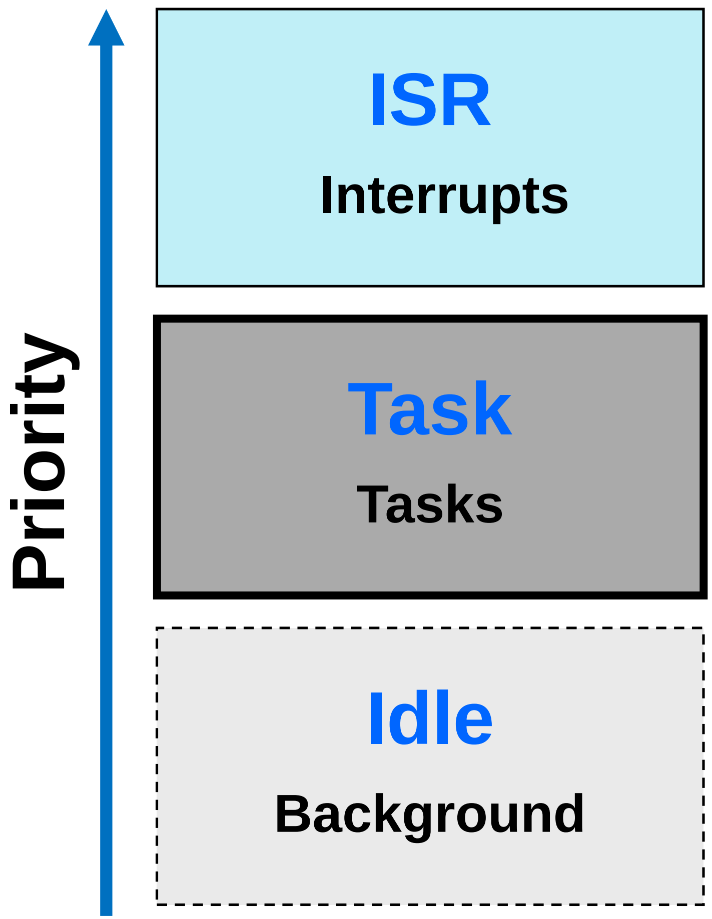
	GP/OS (e.g. Linux)	RTOS (e.g. SYS/BIOS)
Scope	General	Specific
Size	Large: 5M-50M	Small: 5K-50K
Event response	1ms to .1ms	100 – 10 ns
File management	FAT, etc	FatFS
Dynamic Memory	Yes	Yes
Threads	Processes, pThreads, Ints	ISR, Task, Idle
Scheduler	Time Slicing	Preemption
Host Processor	ARM, x86, Power PC	ARM, MSP430, M3, C28x, DSP

Events are often triggered by interrupts



Texas Instruments

RTOS Thread Types

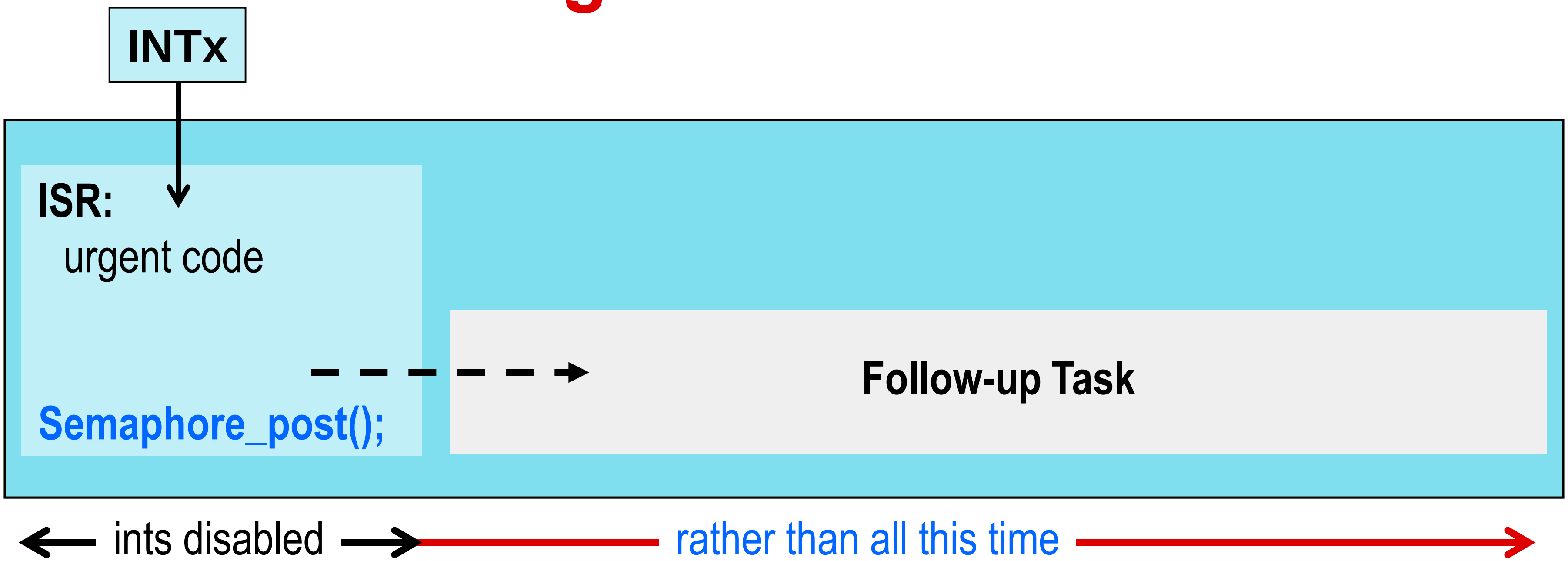


- Implements 'urgent' part of real-time event
- Hardware interrupt triggers ISRs to run
- Priorities set by hardware

- Runs programs concurrently under separate contexts
- Usually enabled to run by posting a '*semaphore*' (a task signaling mechanism)
- Multiple priority levels

- Runs as an infinite loop (like traditional *while(1)* loop)
- Single priority level

ISR's handle urgent activities



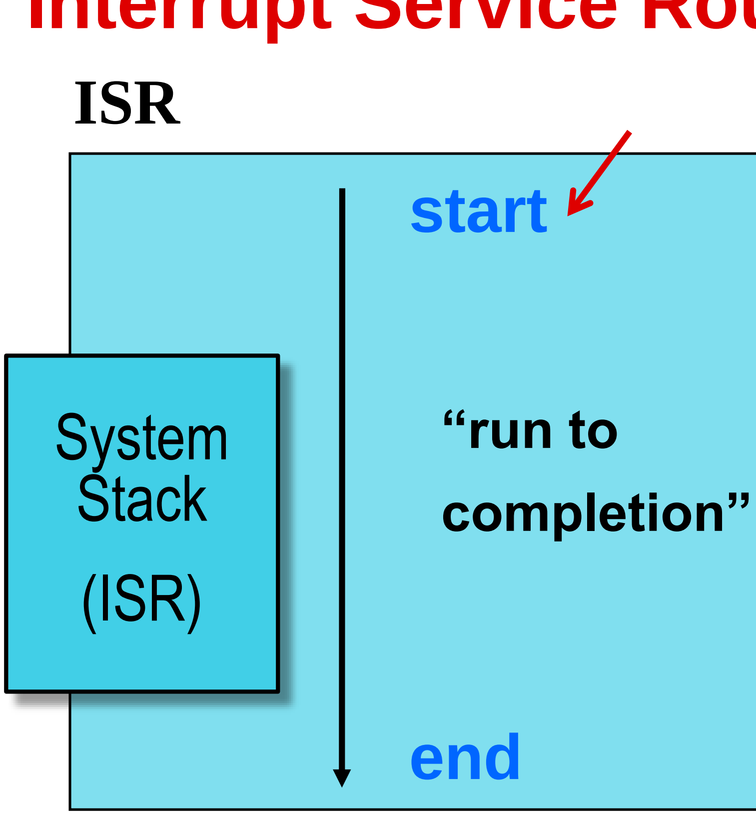
ISR

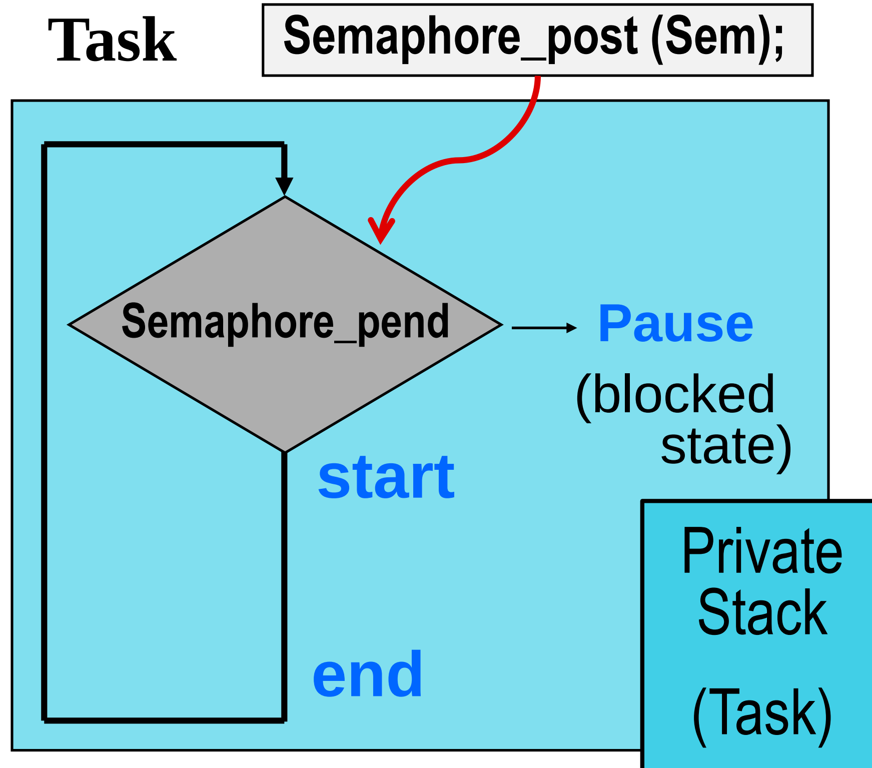
- Fast response to interrupts
- Minimal context switching
- High priority only
- Can post a Task
- Use for urgent code only – then post follow up activity

Task

- Latency in response time
- Context switch performed
- Selectable priority levels
- Can post other Tasks
- Execution managed by scheduler

Interrupt Service Routines (ISRs) and Tasks

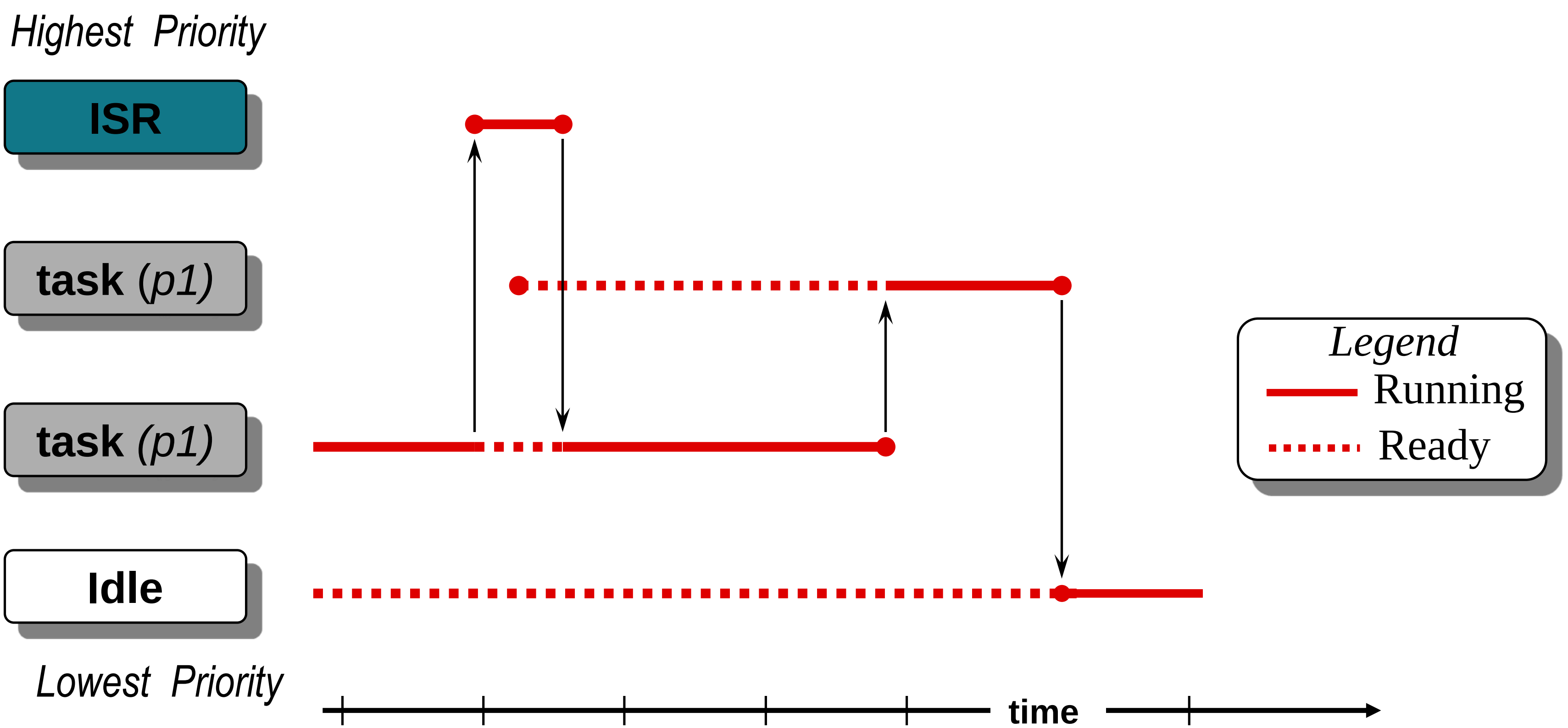




- Process hardware interrupt
- All TSR's share system software stack

- Unblocking triggers execution
- Each Task has its own stack, which allows them to pause (i.e. block)
- Topology: prologue, loop, epilogue…

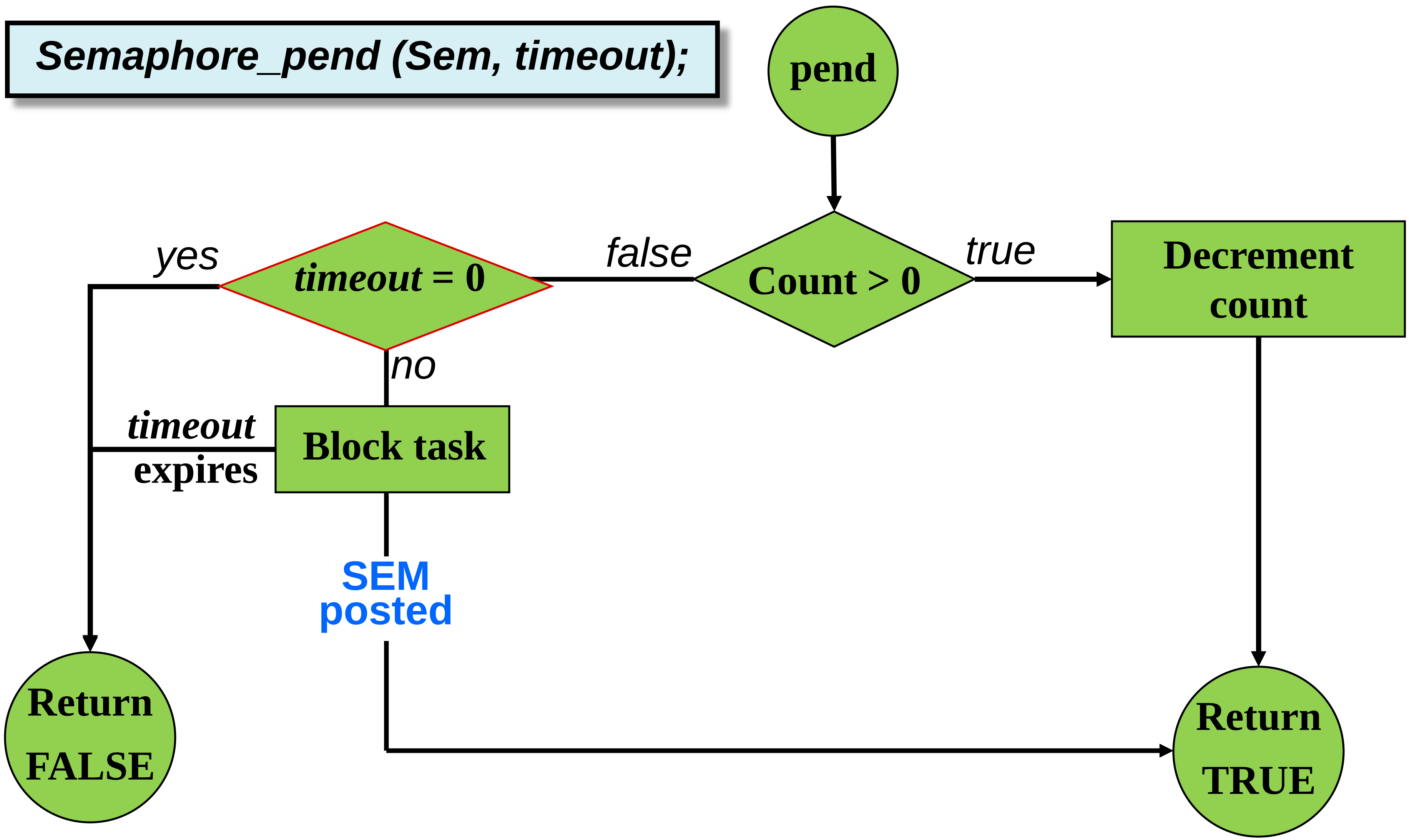
Scheduling Rules on a Single Thread



- Processes of same priority are scheduled first-in first-out (FIFO)

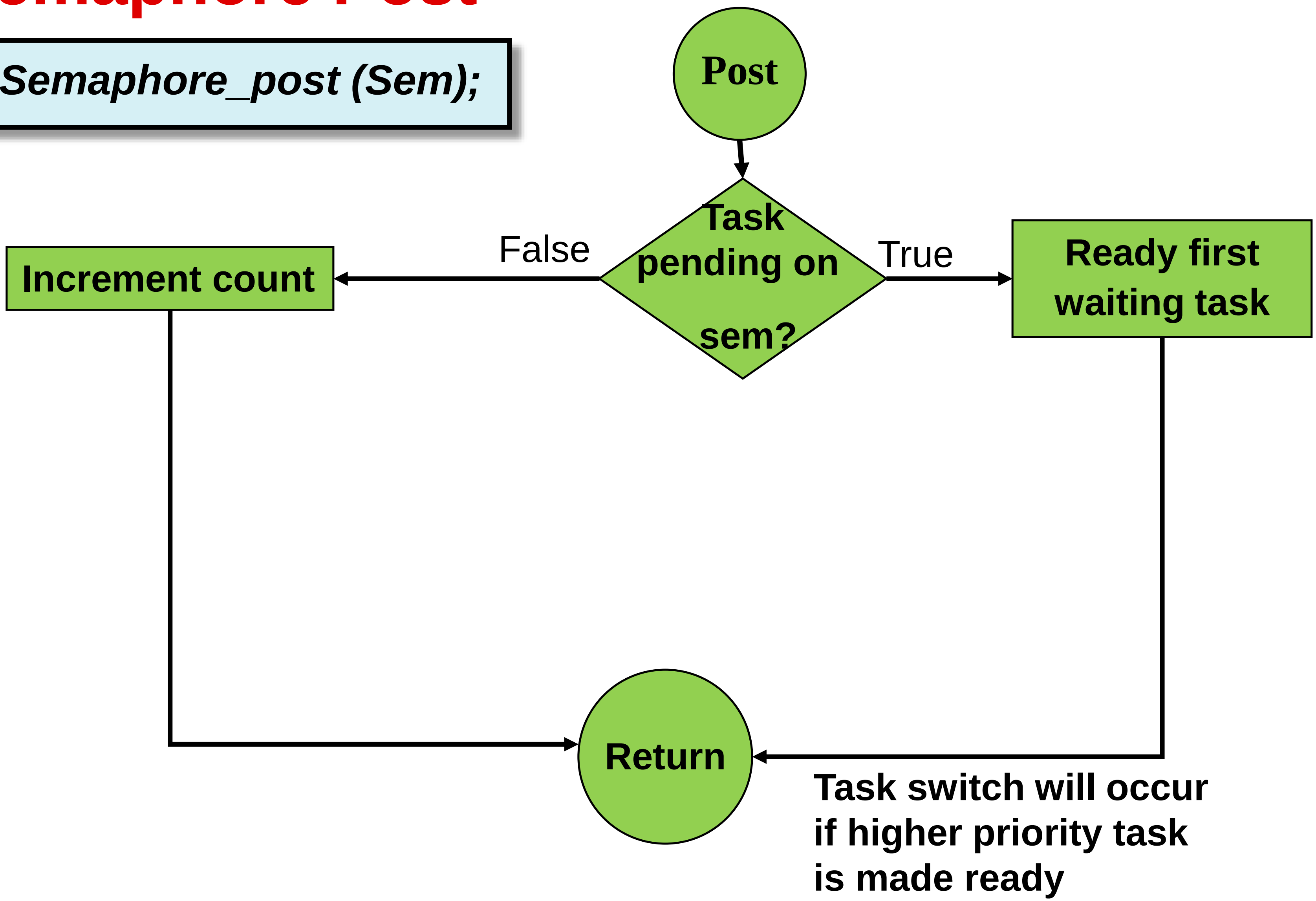
Semaphore Pend

Semaphore_pend (Sem, timeout);



Semaphore Post

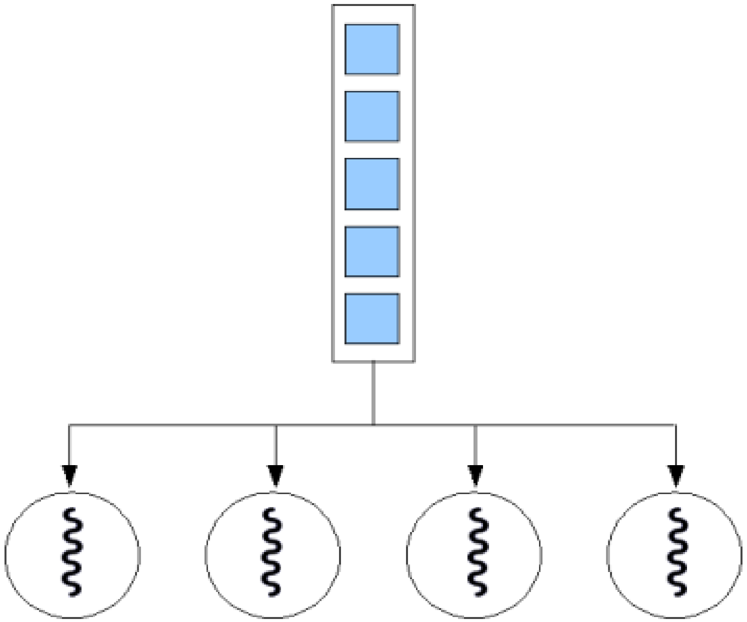
Semaphore_post (Sem);



OpenMP: Task Construct

- Task model supports irregular data dependent parallelism
- Conceptually tasks are assigned to a queue
- Threads execute tasks that they remove from a task queue

```
#pragma omp parallel
{
  #pragma omp single
   {
    p = listhead;
    while (p) {
       #pragma omp task
       process(p);
       p=next(p);
     }
   }
}
```



Event Driven Task Model

```
main()
{
   #pragma omp task isr(1)
   ISR_hwi1();

   #pragma omp task priority(1)
   process_buffer();

   #pragma omp task priority(2)
   idle_task();

   #pragma omp taskwait
}
```

```
ISR_hw1()
{
*buf++ = *XBUF;
cnt++;
if (cnt >= BLKSZ) {
   omp_sem_post(swiFir);
   count = 0;
   pingPong ^= 1;
}
```

```
Process_buffer()
{
  while (1)
  {
   omp_sem_pend(swiFir);
   Filter(buf);
  }
}
```

Event Driven Task Model 2

```
main()
{
#pragma omp task isr(1)
 ISR_hwi1();

#pragma omp task priority(2)
 idle_task();

#pragma omp taskwait
}
```

```
ISR_hw1()
{
   *buf++ = *XBUF;
    cnt++;
    if (cnt >= BLKSZ) {
#pragma omp task priority(1)
       filter_buffer();
       count = 0;
       pingPong ^= 1;
}
```

```
filter_buffer()
{
#pragma omp parallel for
  for (i=0; i<BLKSZ: i++)
    outp[i] = F(buf[i]);
}
```

Event-Driven Tasking Model Summary

- We want to improve the productivity of embedded programmers with higher level models.
- Embedded Systems are very often event driven
- Can the OpenMP tasking model be extended to implement an event driven model?
- Can ISR's be special tasks?
- Is the new task priority clause coming in 4.1 sufficient or …
- Would the task scheduling algorithm need to change or at least be adaptable (like the loop schedule clause)?
- Are persistent tasks that communicate using point-to-point communication (see the previous semaphore examples) more efficient than launching new tasks each time an event occurs?

Texas Instruments

Towards OpenMP in Embedded Systems

MPSoC Model

Texas Instruments

Trends in multicore heterogeneous SoCs

- Market demand for increased processing performance, reduced power, and efficient use of board area
- Demand satisfied by **adding cores**
  - Mix of general purpose CPUs, DSPs
- Challenges:
  - How to efficiently segment tasks between compute engines
  - How to effectively and quickly program multiple cores of different types

Single Core (C66x)

Multicore (C6678)

Heterogeneous Multicore (66AK2H) CPU + Accelerator

Network of Heterogeneous Multicore (HP Proliant m800)

HP Moonshot chassis with m800s

Algorithm implementation must scale to fit available computing power

Keystone II: 66AK2H12/06 SoC

C66x Fixed or Floating Point DSP

- 4x/8x 66x DSP cores up to 1.4GHz
- 2x/4x Cotex ARM A15
- 1MB of local L2 cache RAM per C66 DSP core
- 4MB shared across all ARM

Large on chip and off chip memory

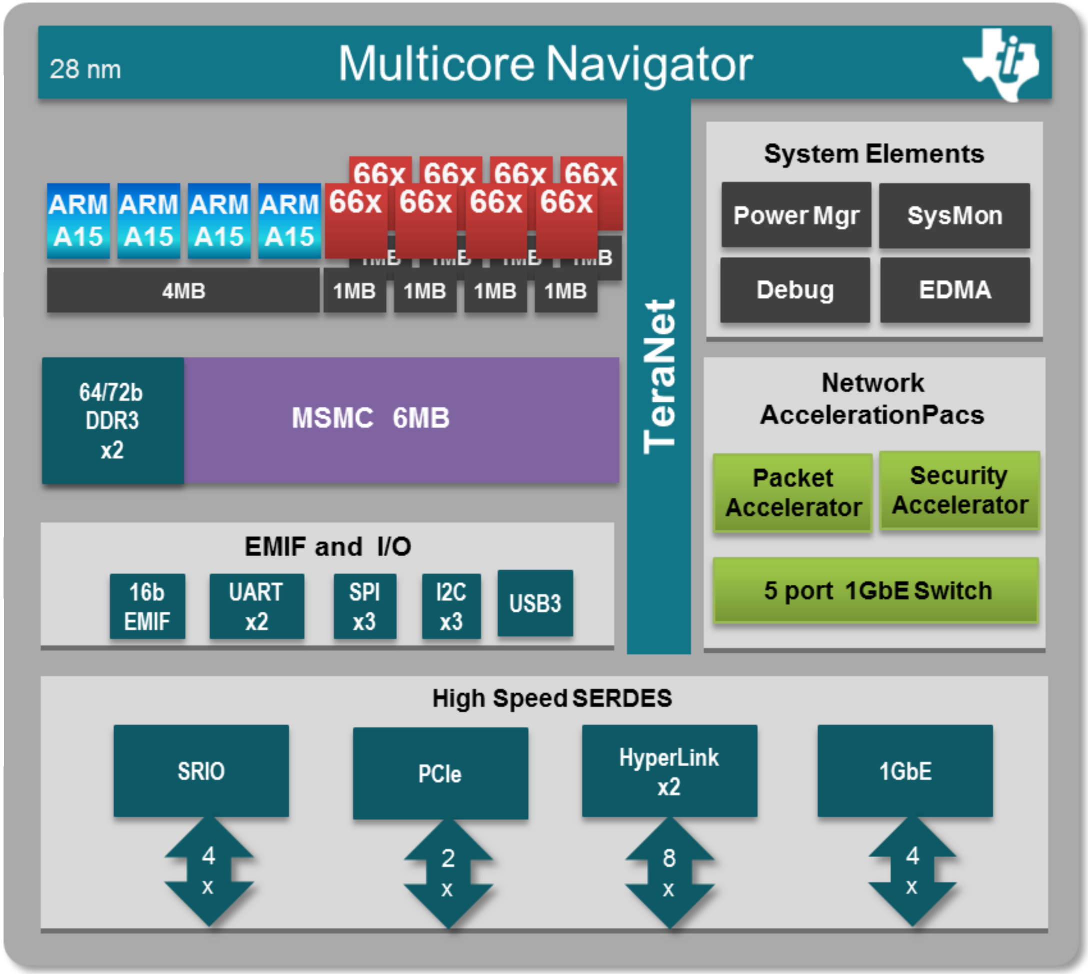
- Multicore Shared Memory Controller provides low latency & high bandwidth memory access
- 6MB Shared L2 on-chip
- 2 x 72 bit DDR3, 72-bit (with ECC), 10GB total addressable, DIMM support (4 ranks total)

KeyStone multicore architecture and acceleration

- Multicore Navigator, TeraNet, HyperLink
- 1GbE Network coprocessor (IPv4/IPv6)
- Crypto Engine (IPSec, SRTP)

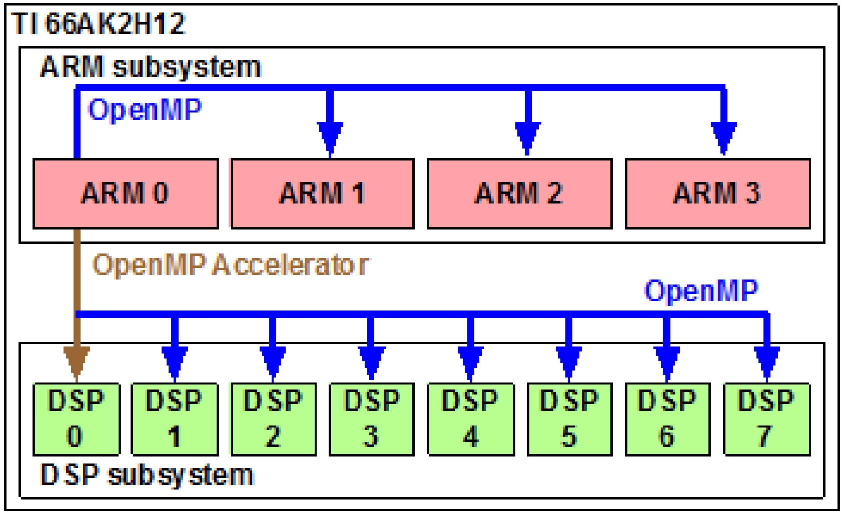
Peripherals

- 4 Port 1G Layer 2 Ethernet Switch
- 2x PCIe, 1x4 SRIO 2.1, EMIF16, USB 3.0 UARTx2, SPI, I^{2}C
- 15-25W depending upon DSP cores, speed, temp & other factors



40mm x 40mm package

OpenMP 4.0 Accelerator Model




```c
void add_openmp(const float *a, const float *b,
                float *c, int size)
{
     #pragma omp target map(to:a[0:size],b[0:size],size) \
                  map(from: c[0:size])
    {
        int i;
        #pragma omp parallel for
        for (i = 0; i < size; i++)
            c[i] = a[i] + b[i];
    }
}
```

Dispatch Model (target regions)

- Notion of host device and target device
- Use 'target' constructs to offload regions of code from host to target device
- Target regions can contain parallel regions

Execution Model

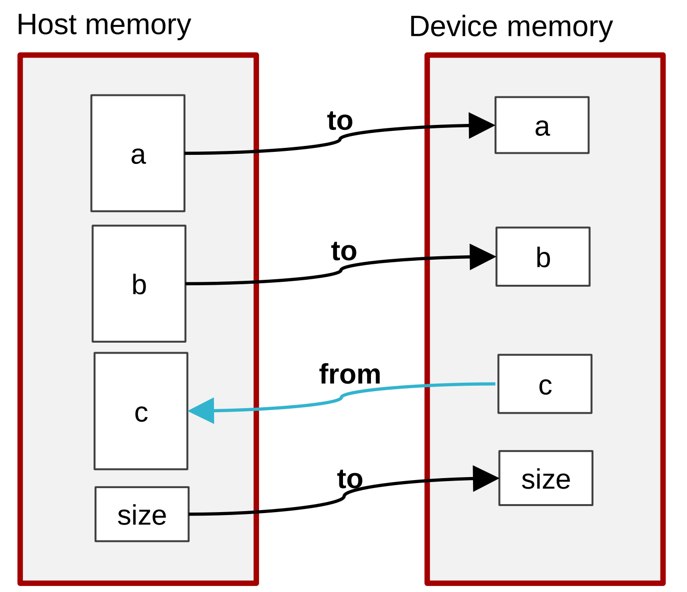
- Each device has it's own threads
- No migration of threads across devices

Memory Model

- Each device has an initial data environment
- Data mapping clauses determine how variables are mapped from the host device data environment to that of the target device
- Variables in different data environments may share storage

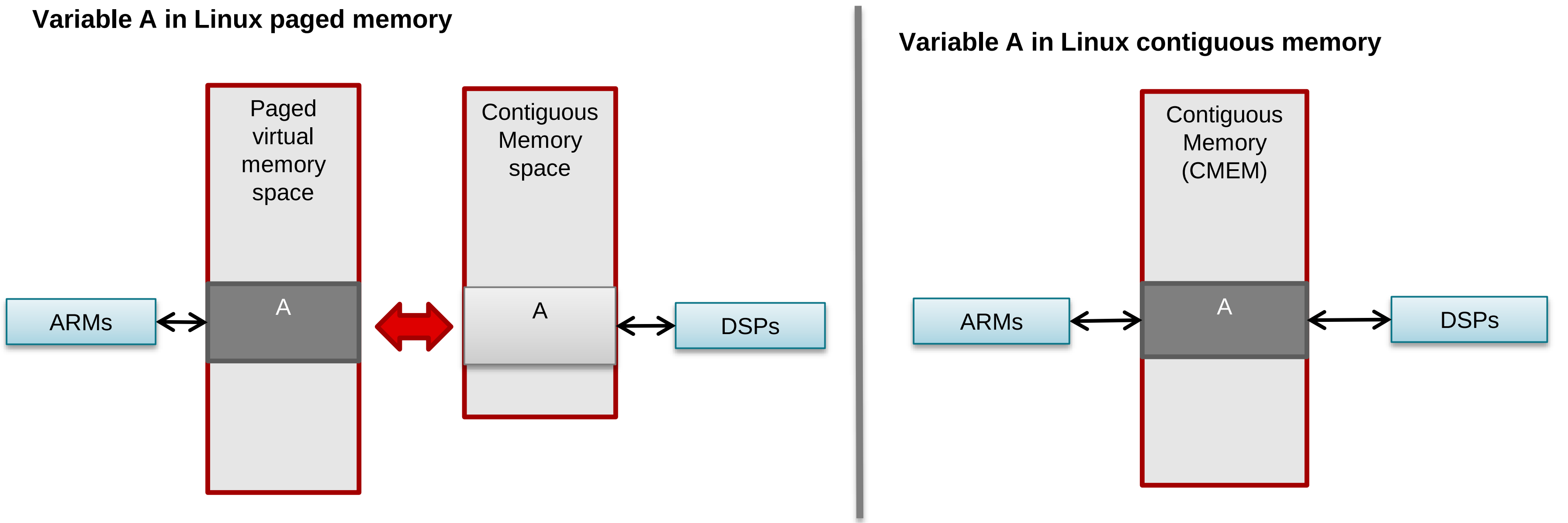
target construct

```
void vadd_openmp(float *a, float *b, float *c, int size)
{
    #pragma omp target map(to:a[0:size],b[0:size],size) \
                       map(from: c[0:size])
    {
        int i;
        #pragma omp parallel for
        for (i = 0; i < size; i++)
            c[i] = a[i] + b[i];
    }
}
```



- Variables a, b, c and size initially reside in host memory
- On encountering a `target` construct:
  - Space is **allocated** in device memory for variables a[0:size], b[0:size], c[0:size] and size
  - Any variables annotated 'to' are **mapped** from host memory to device memory
  - The target region is executed on the device
  - Any variables annotated 'from' are **mapped** from device memory to host memory

Accelerator Memory Model (Logical View)



- DDR/MSMC "physically" shared by ARM(s) and DSP(s)
- However, DSPs do not have a memory management unit (MMU)
  - => DSPs must operate out of contiguous memory
- 2 logical views depending on location of variable in Linux memory
  - Paged virtual memory vs.
  - Contiguous virtual memory
- Variable in paged memory => map clauses translate to copy operations
- Variable in contiguous memory => map clauses translate to ARM-side cache operations

Contiguous Memory management API

- **__malloc_ddr/msmc** Allocate a buffer in contiguous memory (DDR/MSMC SRAM) with given size and return a host pointer to it
- **__free_ddr/msmc** Free device memory with the given host pointer

```
float* a     = (float*) __malloc_ddr(bufsize); // 128 MB

for (int i=0; i < NumElements; ++i)
    a[i] = 1.0;

#pragma omp target map(to:a[0:size],size) map(from: a[0:size])
{
    int i;
    #pragma omp parallel for
    for (i = 0; i < size; i++)
        a[i] *= 2.0;
}

__free_ddr(a);
```

Allocate buffer in device memory

Initialize on host

Map to is a cache write-back operation on host. No copy is performed

Map from is a cache invalidate operation on host. No copy performed

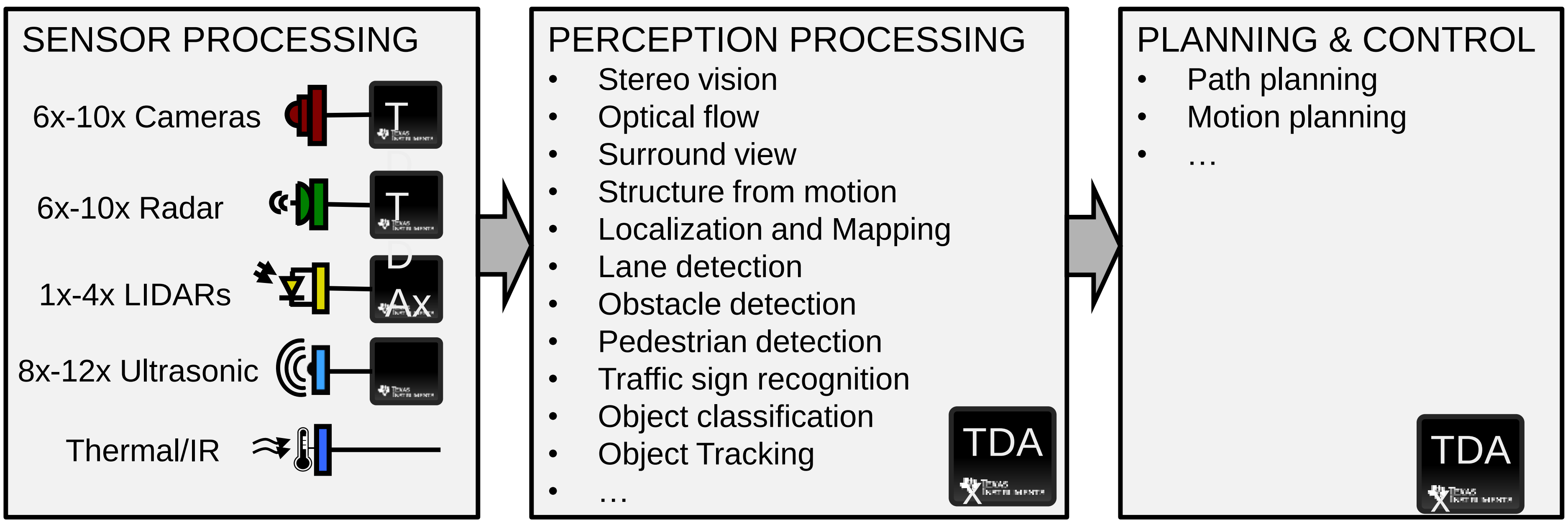
Free buffer

'local' map type

- TI has added a ***local*** map type - maps a variable to the L2 scratchpad memory.
- Such variables are "private" to the target region
  - They have an undefined initial value on entry to the target region
  - Any updates to the variable in the target region cannot be reflected back to the host.
- Mapping host variables to target scratchpad memory provides significant performance improvements.
- In the default configuration, on each DSP core, 768K is available via the local map type.

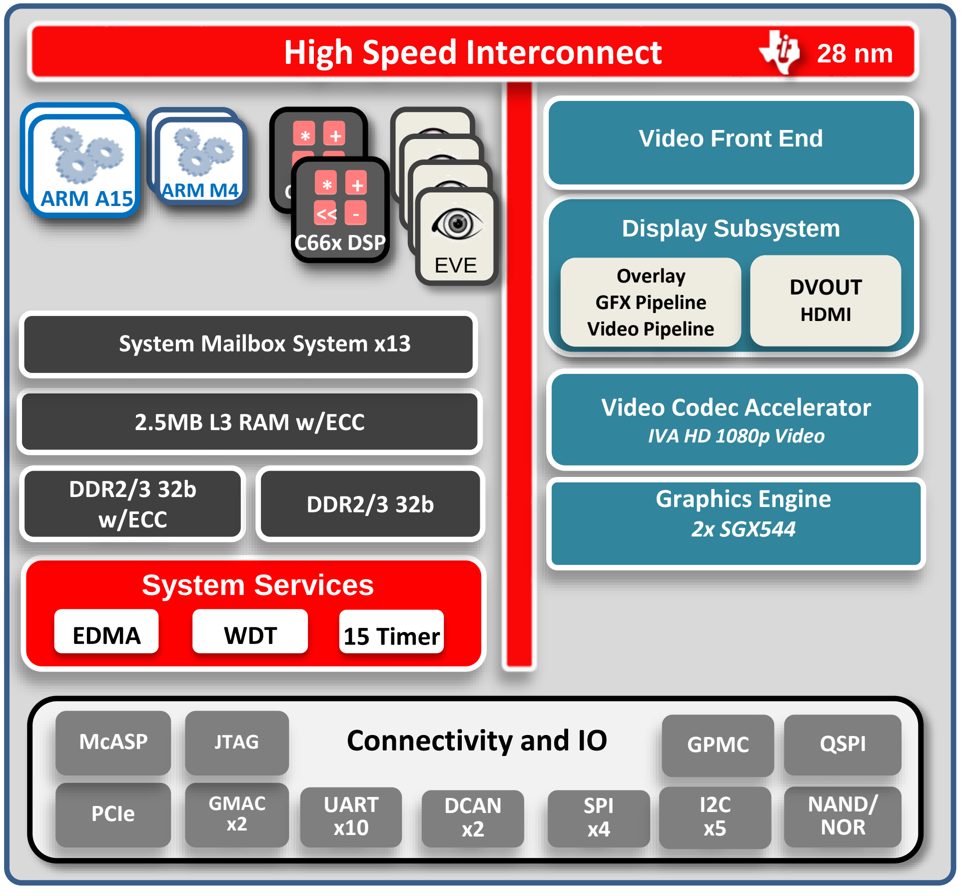
Texas Instruments

Autonomous Vehicle (AV) and Advanced Driver Assistance Systems (ADAS)



MPSoC Example: TDA2x

- Two Next Generation DSP Cores: C66x™
  - Up to 650 MHz
  - Floating Point Extension
- Dual ARM Cortex™ A15 Cores
  - Up to 1000MHz
  - NEON Vector Floating point
- Dual ARM Cortex™ M4 Cores
  - 200 MHz
- Four Vision Accelerator Cores: EVE
  - Upto 650 MHz (8bit or 16bit
- Video Codec Accelerator
  - IVA-HD core running at up to 532MHz
- Graphics Engine
  - Two SGX544 cores delivering capability to render 170Mpoly/s / 5000MPixel/s / 34GFLOPs at 500Mhz
- Internal Memory
  - DSPs: each w/ 32 KB L1D, 32 KB L1P, unified 256 KB L2 Cache
  - ARM : 32 KB L1D, 32 KB L1P, combined 2 MB L2 Cache
  - On Chip L3 RAM: 2.5MB with ECC
- Peripherals Highlights (1.8/ 3.3V IOs)
  - Video Inputs: Six 16 bit ports
  - Display system Digital Video Output
  - Two EMIFs: 2x 32bit wide DDR2/3/3L @ 532MHz, one with ECC
  - GPMC: general purpose memory controller
  - Support for NOR Flash
  - PCIe, 2x Gbit EMAC with AVB support
  - 2x DCAN (High end CAN controller)
  - 10x UART, 5x I²C, 4x McSPI, Quad SPI, McASP, 15x Timers, WDT, GPIO



- Package
  - 23x23mm BGA (ABC), 0.8mm ball pitch
  - 17x17mm BGA (AAS), 0.65mm ball pitch
- Power (~1.0V Core, 1.8/ 3.3V IOs)
  - Target @ 125C Tj ~4-5W, depending on use case

ADAS Applications

Core Applications

Front Camera

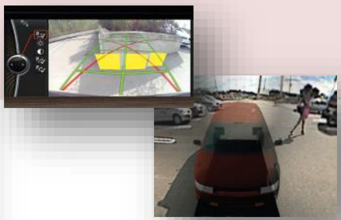
Scalable Performance
Low Power
Safety

Surround View Park Assist

Integrate 3D Graphics
Scalable Analytics
Security

Rear Camera

Low Power
Small Footprint
Scalable Analytics



Radar

Scalable performance
MCU Integration
Safety

Emerging Applications

Sensor Fusion

Performance
Safety
Security

Driver Monitoring

Small Footprint
ISP Integration
Scalable Analytics

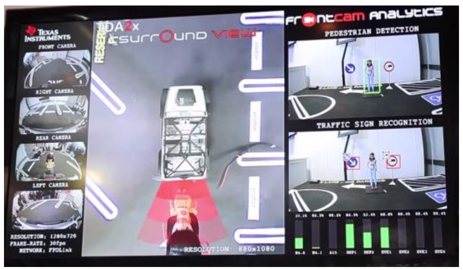
Mirror Replacement

Performance
ISP Integration
Scalable Analytics

One HW and SW architecture allowing for scalability from premium to entry-level vehicles.

Surround View, Ultrasonic and Front Camera

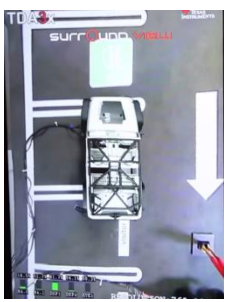
Front Camera



Surround View, Ultrasonic Sensor, PD, TSR

TDA2x ADAS Processor
Texas Instruments

PD, TSR, Lane Detection, Sparse Optical Flow, Stereo Disparity



Surround View

TDA3x ADAS Processor
Texas Instruments

PD, TSR, Lane Detection

Watch CES2015 [Videos]

Can OpenMP become a complete embedded MPSoC programming model?

- We can see how OpenMP can be used to exploit parallelism in compute-intensive parts of the algorithm.
- We can see how OpenMP could be used to offload accelerated algorithms from the 'host' processor domain to an accelerator.
- Can OpenMP provide an embedded event-driven MPSoC (heterogeneous) model where a device can launch code on any other device.
  - ARM M4 cores running an RTOS respond to real-time events and dispatch processing to the other cores in the system.
  - DSP cores are assigned specific real-time events that they process locally.
  - ARM A15 processors running SMP Linux manage the user Interface and then dispatch processing (graphics) to other cores (GPUs)
- A combination of the event-driven model and the MPSoC model.

MPSoC Event Driven Task Model

```
main()
{
   #pragma omp task device(M4) isr(1)
   ISR_hwi1();

   #pragma omp task device(DSP) isr(2)
   ISR_hwi2();

   #pragma omp task device(DSP) priority(2)
   process_driver_fitness();

   #pragma omp task device(DSP) priority(3)
   process_vision_frame();

   #pragma omp task device(A15) priority(1)
   user_interface();

   #pragma omp task device(M4,A15,DSP) priority(1
   idle_task();

   #pragma omp taskwait
}
```

```
ISR_hw1()
{
*frame++ = *XBUF;
cnt++;
if (cnt >= BLKSZ) {
#pragma omp target update\
            device(DSP)\
          to(frame[:BLKSZ])
   omp_sem_post(VisFrame);
   count = 0;
   pingPong ^= 1;
}
```

```
Process_vision_frame()
{
  while (1)
  {
   omp_sem_pend(VisFrame);
   CNNetwork(frame);
  }
}
```


Texas Instruments

Towards OpenMP in Embedded Systems

Summary and Conclusions

TEXAS INSTRUMENTS

Other Topics

- Expressing constraints (balance performance and energy consumption)
  - See IWOMP 2015 papers(s)
- Heterogeneous memory
  - Place objects in specific memory areas
  - RAM, ROM, SRAM, off-chip and on-chip
- Hierarchical memory systems
  - Fast but limited scratch pad memory
  - Data streaming via asynchronous DMA engines
- Resiliency
  - Embedded systems run forever
  - A mechanism to respond and recover from unexpected behavior
  - Is there something in the omp cancel construct?
- Specialization
  - OpenMP is getting bigger.
  - Rebuild OpenMP run-time at program build time
  - Indicate number of threads on a device at program build time

Summary

- OpenMP is the industry standard for directive based parallel programming
- OpenMP can express the parallelism in the compute intensive parts of an embedded program
- Embedded systems are often event-driven and programmers must write custom code to implement this model.
- Extend OpenMP tasking to support to the event-driven model (or create a new concept – the process?)
- OpenMP 4.0 added an accelerator host+device model
- Generalize the OpenMP accelerator model to a heterogeneous MPSoC model
- Vision: Embedded programmers using OpenMP to implement event-driven systems for complex MPSoCs