Scalable Multicore Audio Solutions

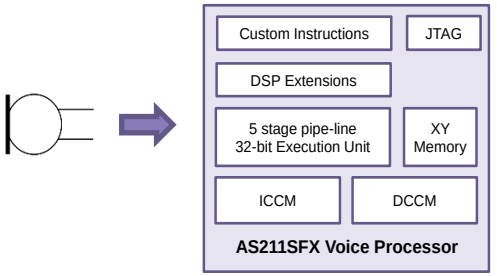
Pieter van der Wolf

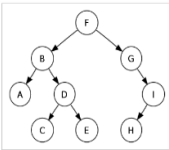
MPSoC 2013

July 15 – 19, 2013

Audio processing requirements

- Audio processing requirements vary widely
  - For different devices
  - For different use cases of a device
- Increasing performance demands
  - HD audio, multi-channel, higher sample rates
  - Wideband voice, advanced pre-/post-processing
  - > 500MHz today, likely to double in coming years
- Low power requirements
  - Mobile devices with “always on” features
  - Voice trigger, voice control
  - ~10MHz command recognition





Need for scalable audio solutions

- Scalability in performance
  - With low power consumption across performance range

- To support different products
  - Products targeted at different market segments
  - Products developed over time (future proof)
  - SoC integrator wants to work with single supplier

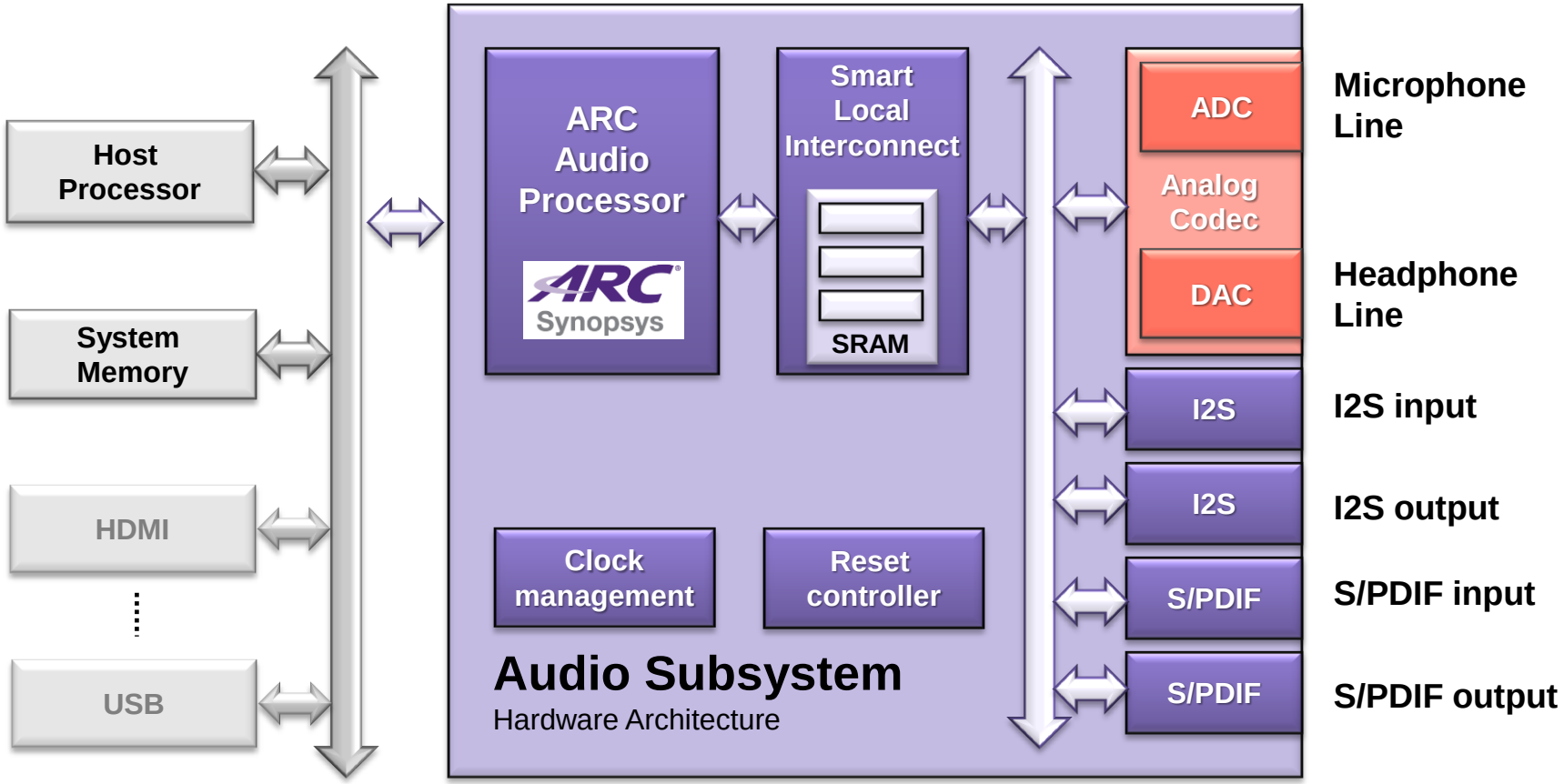
Design time scalability

- To support different use cases of product
  - Different use cases in multi-functional products
  - From “always on” voice trigger to high-end audio / voice features

Run time scalability

SoundWave Audio Subsystem

Hardware architecture

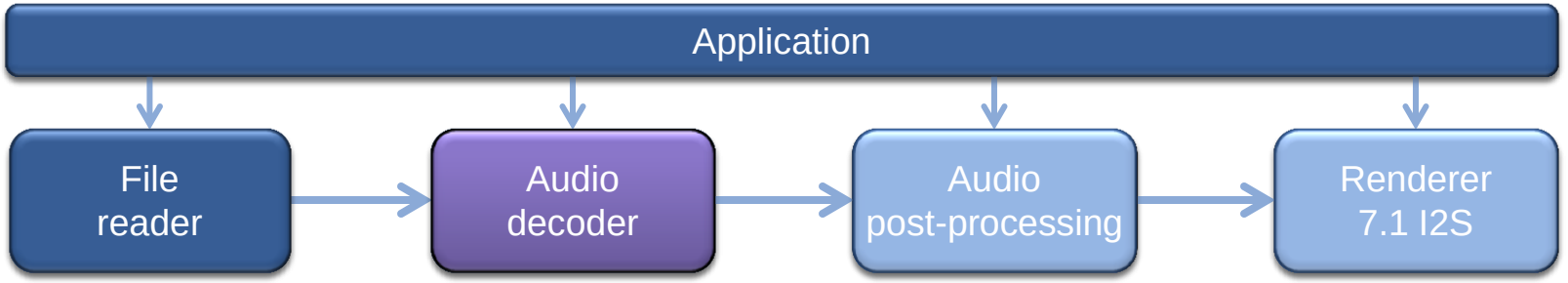


Multi-core audio processor

With configurable core

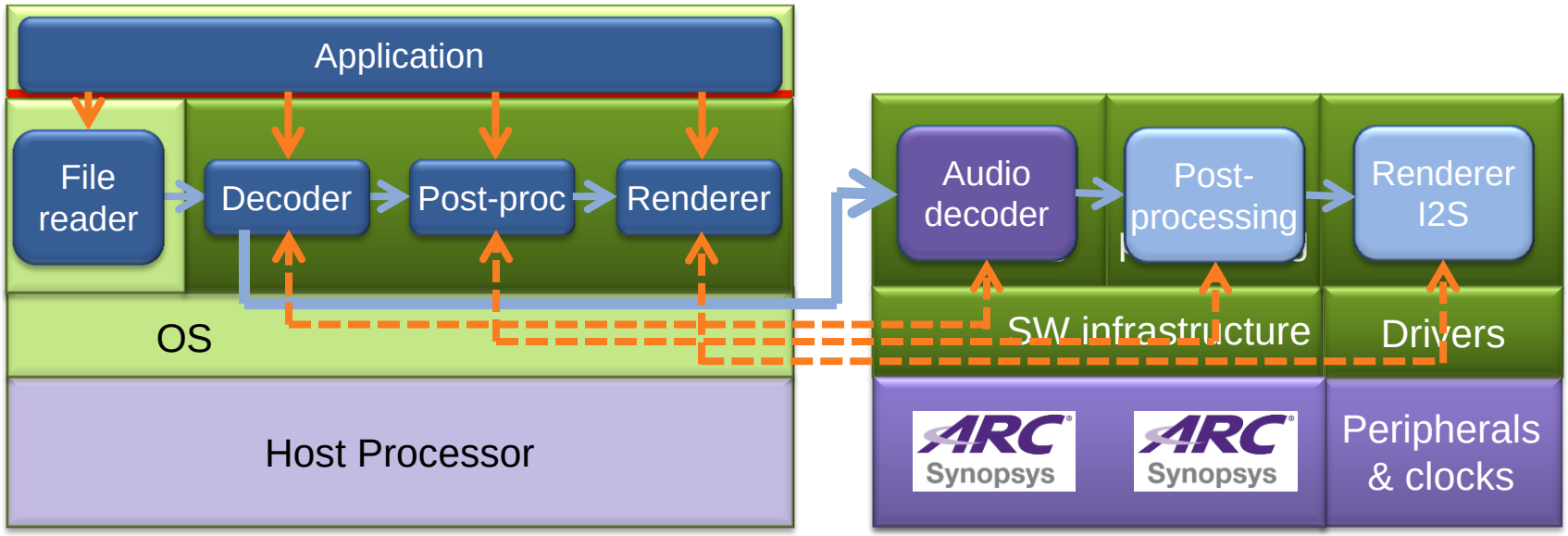
- Multi-core
  - Design-time scalability via number of cores
  - Run-time scalability via power-up/down of cores
  - Power consumption (mW/MHz) equal to single core
- Configurability and extensibility
  - Design-time scalability through specialization
  - Reduce performance requirements (MHz) of audio tasks
- Processor family
  - Design-time scalability through processor selection
  - Run-time scalability via DVFS techniques
  - Performance limited by fastest processor in family
  - High frequency makes processor sensitive to memory latency

Multi-core programming



- Mapping of audio processing functions
  - Function can execute on host processor or on audio processor core
  - Each function fits on single core → lower bound on core performance
  - Broad portfolio of available functions
- Build and execute use cases from host processor
  - Transparent off-loading of audio processing
  - Transparent core crossings
- Efficient communication
  - Keep data / control local, avoid copying overhead

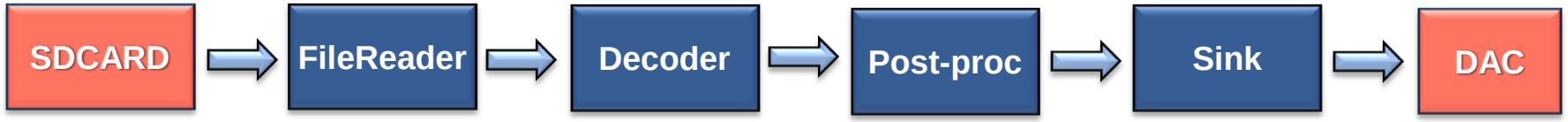
Audio Subsystem HW/SW Architecture



- High-level API for building and executing use cases
- Plug-in makes core crossings transparent
  - Audio functions off-loaded to ARC multi-core audio processor
- SW infrastructure takes care of data streaming
  - Between host and audio processor and among cores of audio processor

File player code snippet

GStreamer example




```
pipeline = gst_pipeline_new ("my-pipeline");

source = gst_element_factory_make ("filereader", "filereader");
g_object_set (G_OBJECT (source), "location", filename, NULL);
g_object_set (G_OBJECT (source), "track", track, NULL);
g_object_get (G_OBJECT (source), "decodertype", &decodertype, NULL);

decoder = gst_element_factory_make ("decoder", "decoder");
g_object_set (G_OBJECT (decoder), "decodertype", decodertype, NULL);
g_object_set (G_OBJECT (decoder), "dspid", 2, NULL);

post-proc = gst_element_factory_make ("post-proc", "post-processing");

sink = gst_element_factory_make ("sink", "renderer I2S");
g_object_set (G_OBJECT (sink), "sinktype", I2S, NULL);

gst_bin_add_many (GST_BIN (pipeline), source, decoder, post-proc, sink, NULL);
gst_element_link_many (source, decoder, post-proc, sink, NULL);

gst_element_set_state (pipeline, GST_STATE_PLAYING);
```

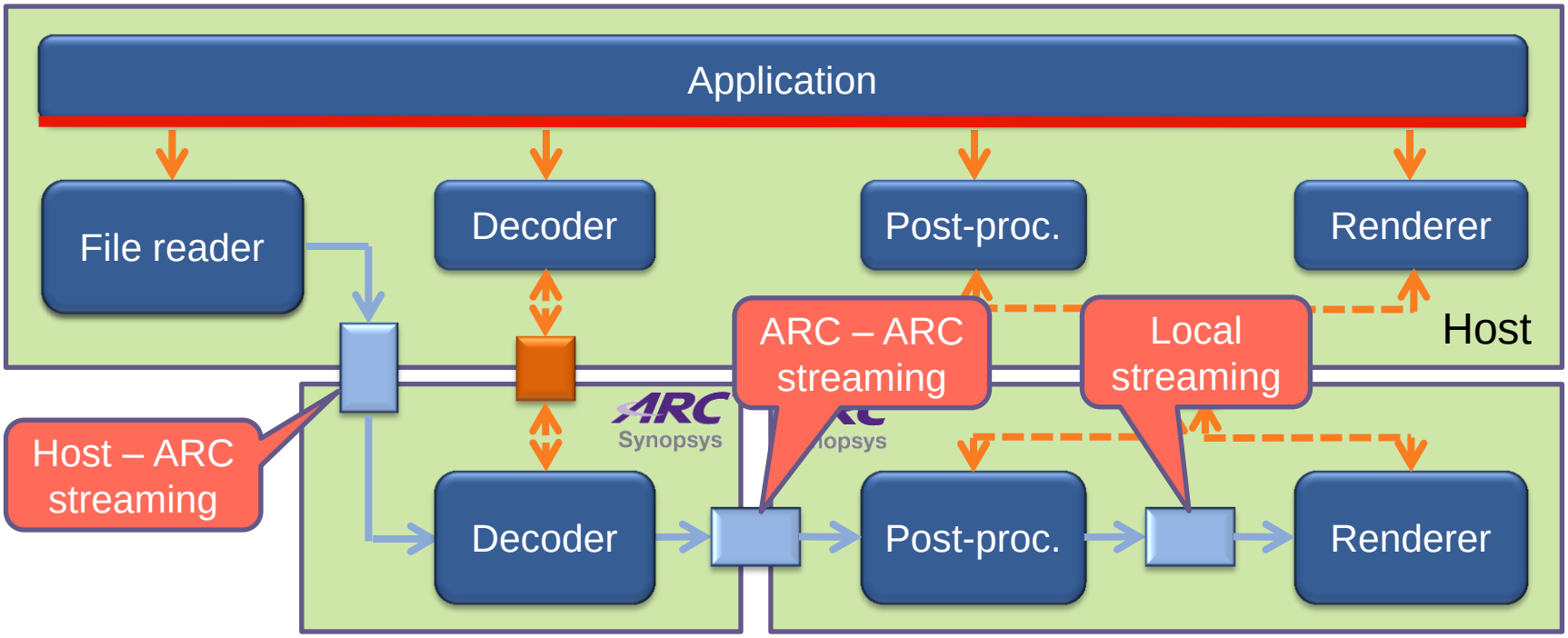
Host

Core 2

Core 1

Core 1

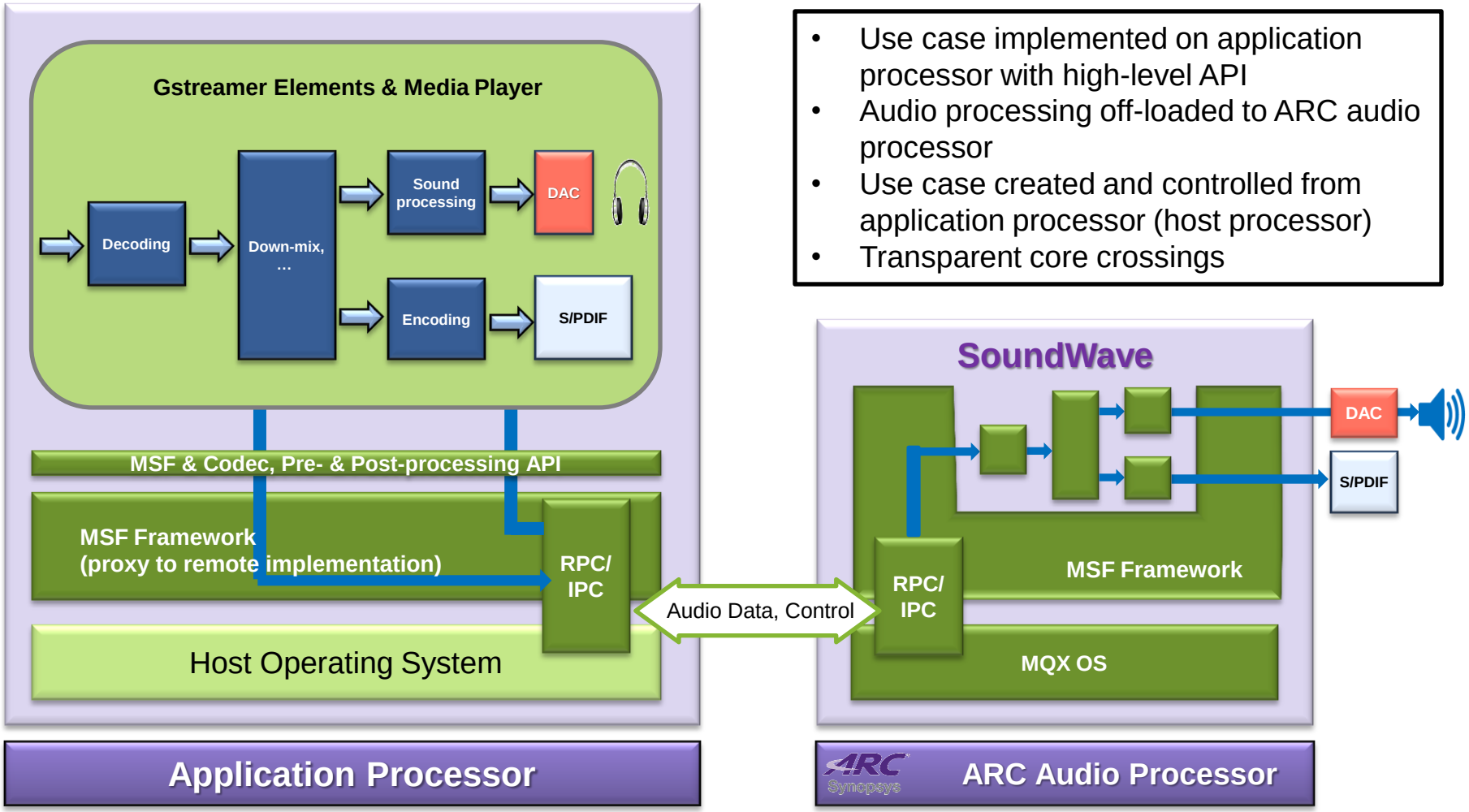
Media Streaming Framework



- Media Streaming Framework for audio processing
- Employs IPC layer for heterogeneous multi-core / multi-OS
- Transparent core crossings
- Efficient (zero-copy data transfer)
- → Building and executing use cases from host processor

Integration Example

Application processor and ARC audio processor



- Use case implemented on application processor with high-level API
- Audio processing off-loaded to ARC audio processor
- Use case created and controlled from application processor (host processor)
- Transparent core crossings

Conclusions

- Need for scalable audio solution
  - To support different products & use cases
- Multicore audio processor offers scalability
  - Design-time and run-time scalability
- Streaming framework supports multicore programming
  - Make core crossings transparent
  - Same audio software for single- and multicore
  - With audio plug-in on host with high-level API
  - Build and execute use cases from host processor
- Scalable multicore audio solution
  - Makes audio off-load easy


Thank You
SYNOPSYS®
Accelerating Innovation