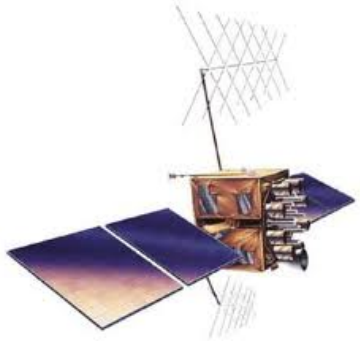
Data Flow Modeling of Radio Applications

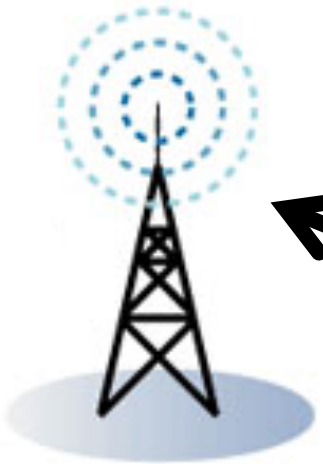
Orlando Moreira

Principal DSP Systems Engineer

Multi-Radio Devices

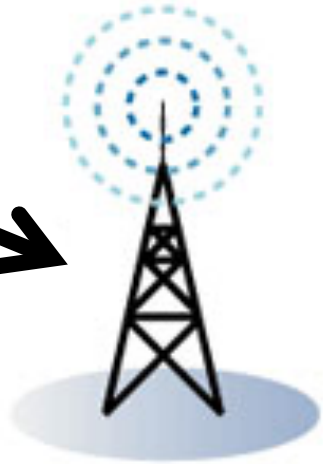


GPS Satellite



3/4G Base station
HSPA

MULTI-RADIO
USER
EQUIPMENT



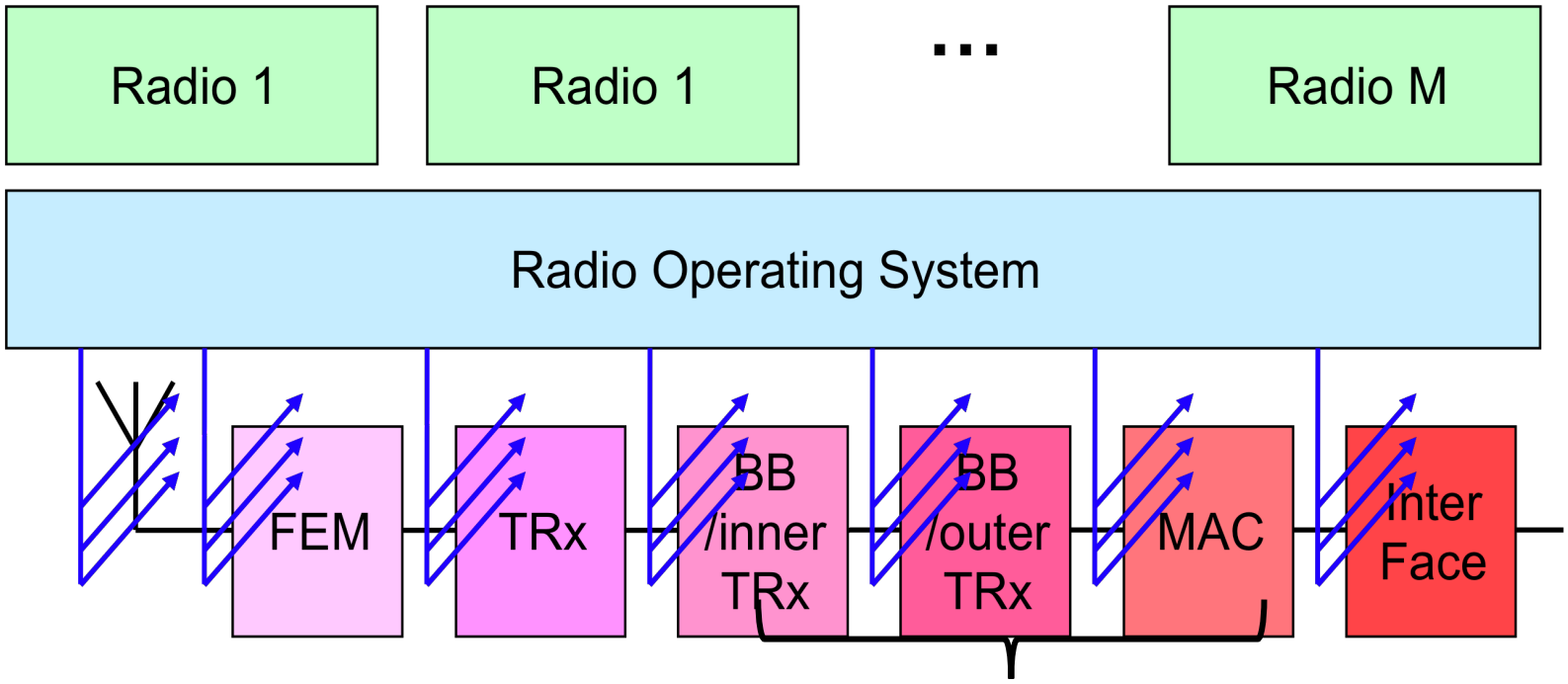
2G Base station
GSM

WLAN Base station

ST ERICSSON

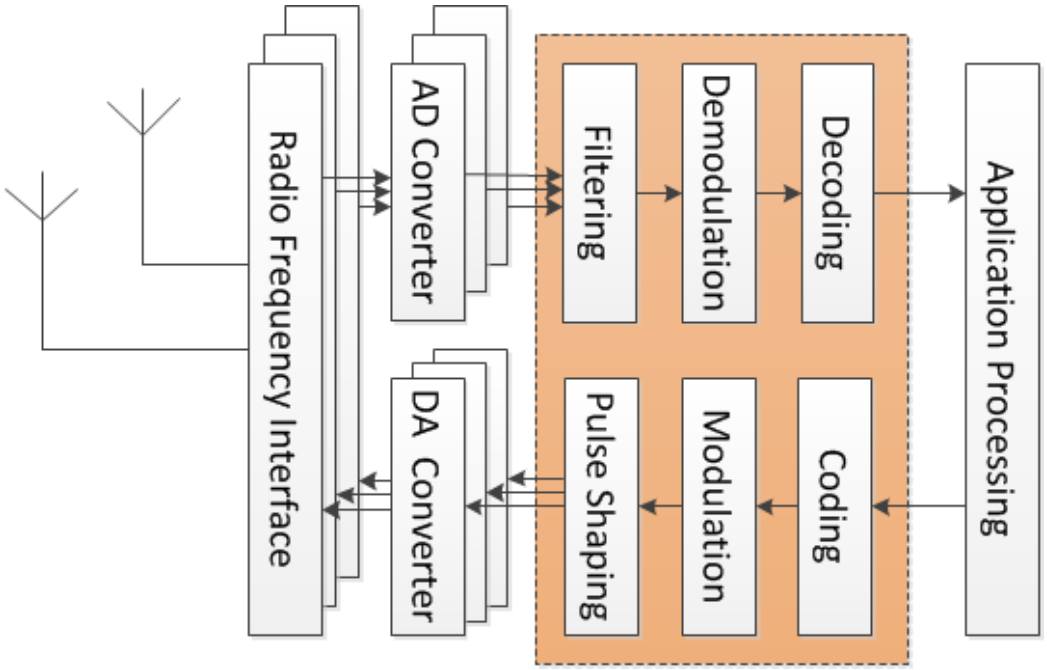
SDR Vision: The Radio Computer

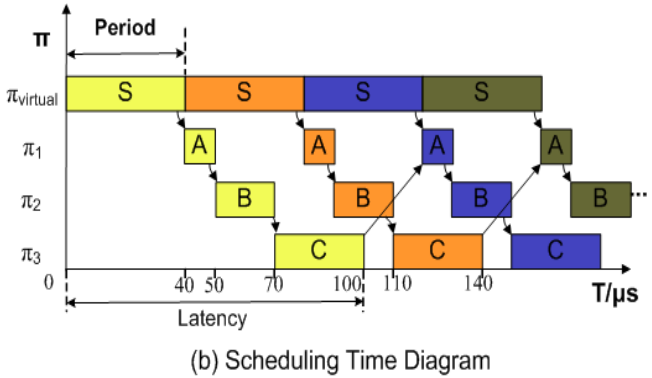
- Hardware virtualization and Runtime Resource Management
- RATs are developped in isolation, delivered independently
- RATs run simultaneously



ST ERICSSON

Application: Radio PHY features



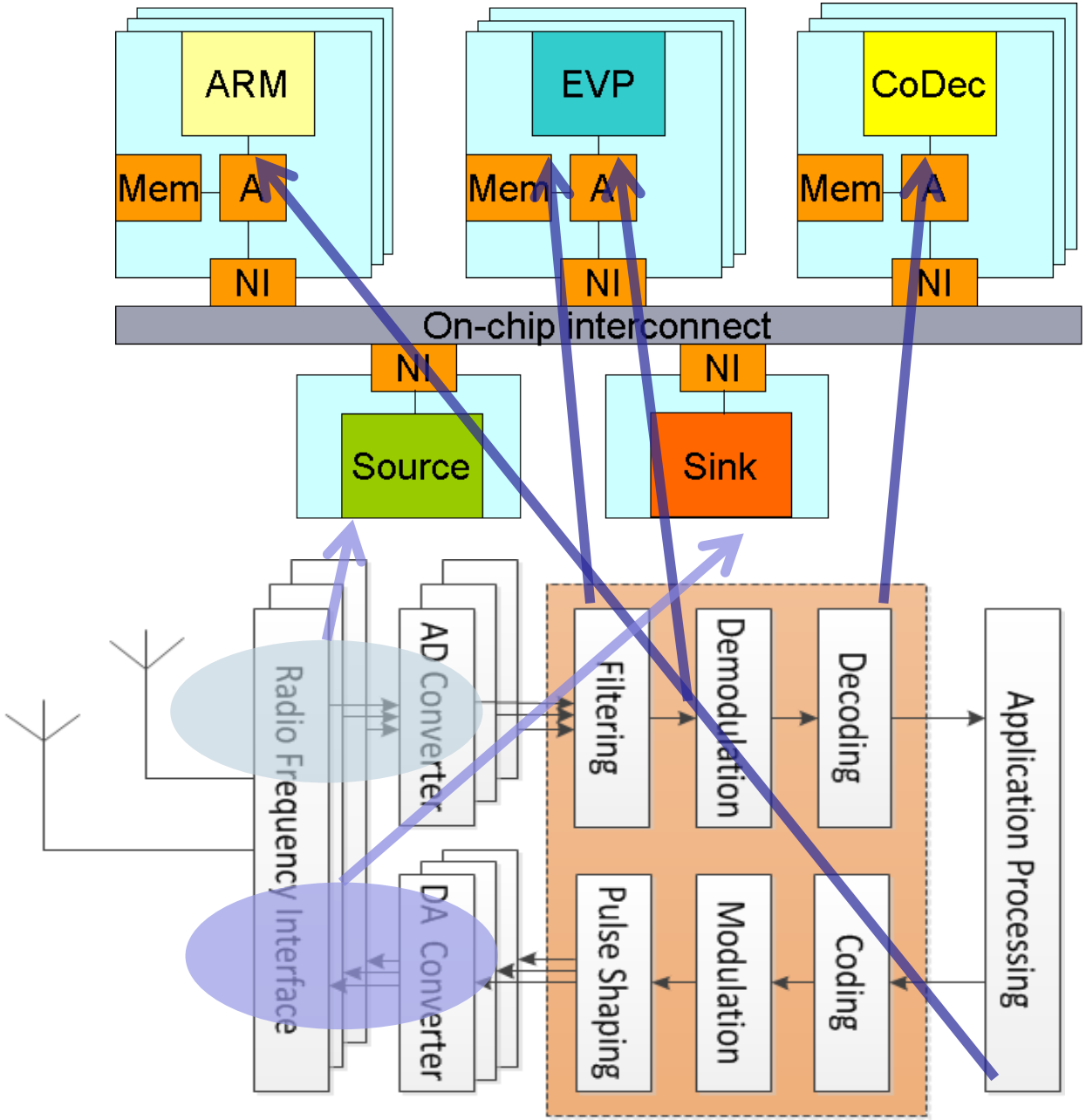


(b) Scheduling Time Diagram

- Hard-real time: throughput, end-to-end latencies
  - must conform to standards, pass certification tests
- Iterative schedules with overlapped execution
- Inter-iteration dependencies
- Periodic or sporadic external sources

ST ERICSSON

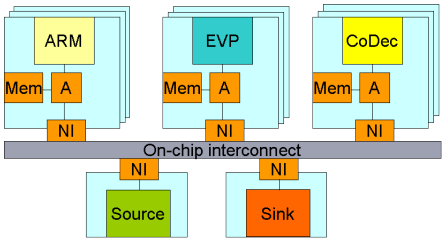
Hardware: Modem Platform

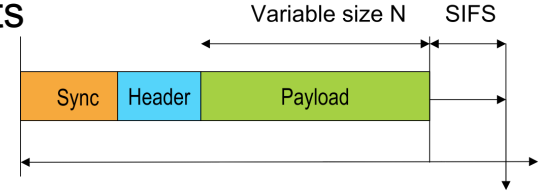


ST ERICSSON

SDR Software Architecture: The Problem

- Platform: Heterogeneous Multiprocessor
- Application: Multiple RATs simultaneously active
  - different rates of operation
  - unpredictable start/stop times
- Requirement: provide Real-Time guarantees
  - RATs must meet end-to-end latency, throughput requirements
- Software Architecture must
  - provide power-efficient, real-time schedules
  - support widest variety of RAT/mode combinations and transitions
  - allow post-design updates and add-ons
  - simplify the RAT design process





ST ERICSSON

Our requirements at a glance

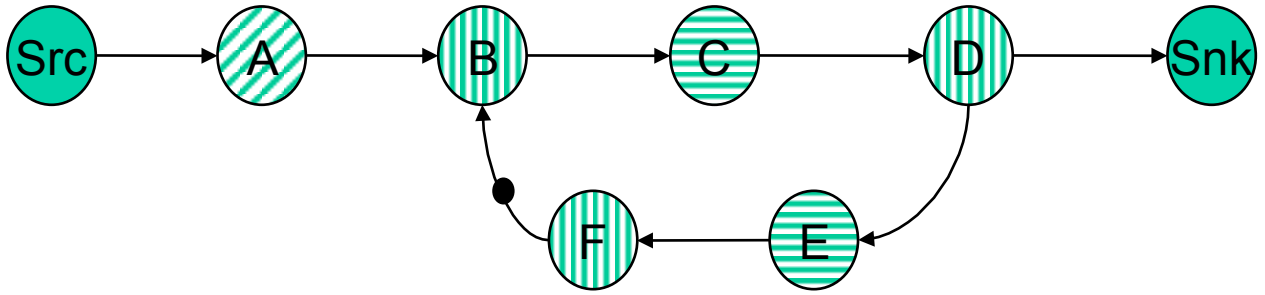
- Real-time:
  - Automated analysis: throughput, end-to-end latencies
  - Hard real-time guarantees, no scheduling anomalies, no deadlocks, no buffer overruns.
- Ease of Programming/Testing/Debugging:
  - Correct-by-contruction concurrent behavior
  - Automated generation of code for communication, synchronization, task schedule
- Efficiency:
  - (Quasi) Static Order scheduling per RAT/Processor.
  - Optimized static determination of buffer sizes
  - Distributed Runtime Synchronization (data triggered)
- The “Magic Bullet”:
  - Data Flow app modeling + Data Flow execution model+ Budget schedulers
  - using a homegrown flavour of data flow customized for radio

Where does unpredictable behavior come from?

- **Application:**
  - Non-deterministic functional behavior (eg: thread model)
  - Dynamism (data-dependent behavior)
  - Results in: undetectable deadlocks, unbounded memory requirements, halting problem
- Solution: Restrict programming model
  - Suiting the domain: must find the right trade-off between expressivity and analyzability
- **Platform:**
  - Resource contention
  - Unbound time to service from resource or high discrepancy between average and worst case
  - Results in: Starvation, Resource Locking, Scheduling anomalies
- Solution: Budgeted arbitration
  - schedulers with service guarantees in all share points

Data Flow formalism

Actors: computing stations with well-defined data-driven activation rules
Arcs: FIFO channels
Tokens: Initial data items in Arcs
Allow the expression of inter-iteration dependencies
For Software Engineers: a concurrent **Component Model** with strong formal properties (as opposed to UML)



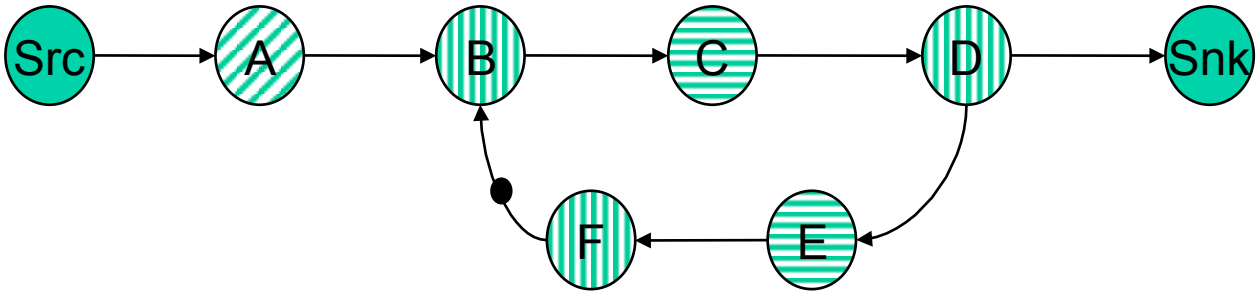
Temporal Analysis (for Static variants: SDF, CSDF):
Longest cycle bounds maximum rate.
Execution in bounded buffer space.
There is always a static periodic schedule that achieves maximum rate
Self timed execution is always upper bounded by static periodic schedule
Monotonic, Linear in timing: No scheduling anomalies.

ST ERICSSON

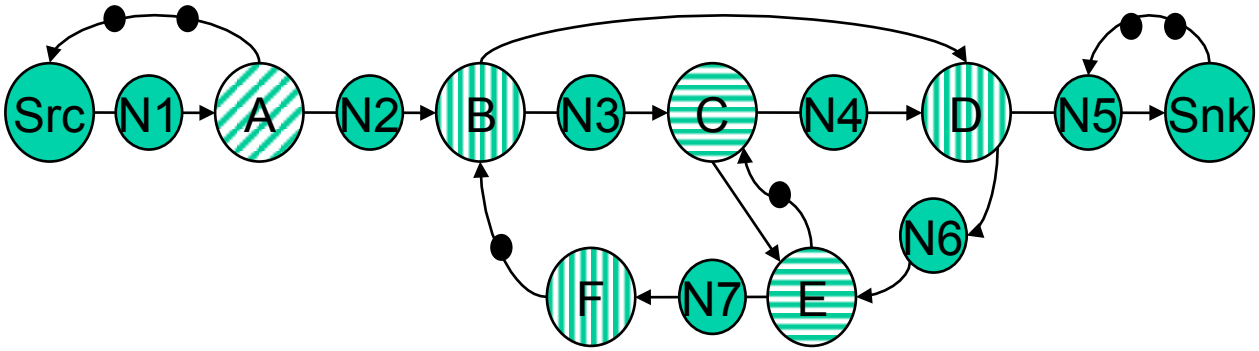
Data Flow formalism: function + mapping+ analysis

Data flow **unifies** concurrent application specification & timing analysis of mapping

Programming Model: Specifying Functionality per RAT



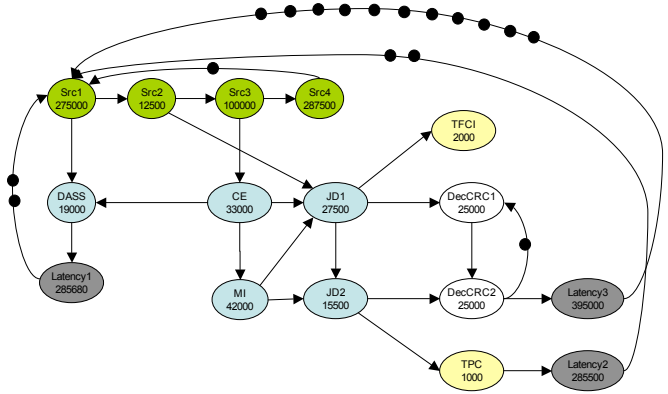
Analysis Model: Modeling Mapping Decisions



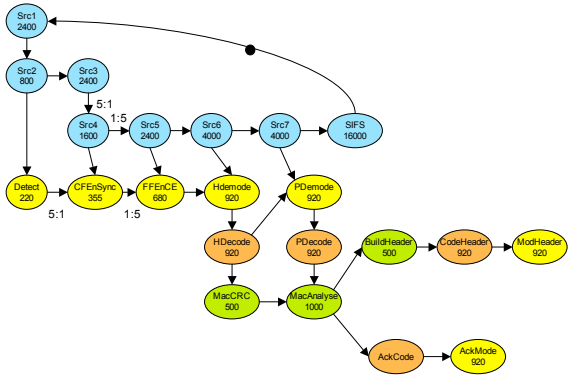
Modeling of buffer sizes, static ordering, communication overhead.

ST ERICSSON

Scheduling Policy – intra vs inter graph



TDS-CDMA Receiver



WLAN Receiver

Intra-graph

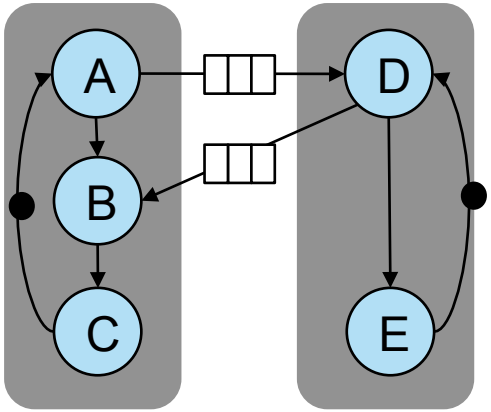
- Dependencies known
- Dependencies are static or quasi -static
- Related Rates of Execution between tasks
- Shared temporal Requirements

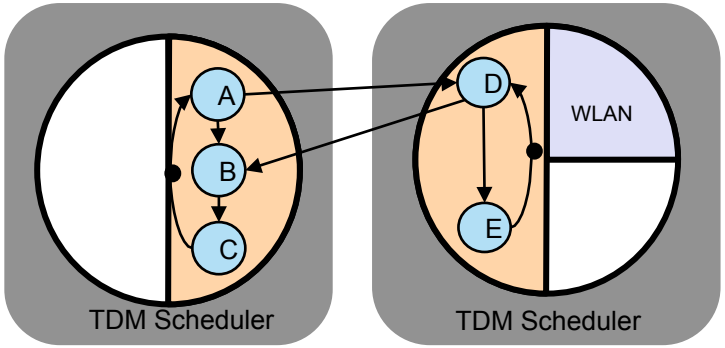
Inter-graph

- No dependencies
- Independent start, stop, loading
- Independent Rates
- Independent Temporal Requirement
- Independent worst-case temporal behavior

ST ERICSSON

Scheduling Policy – intra vs inter graph





Intra-graph

- Inter-processor synchronization:
  Self-timed & data-driven
- Intra-processor:
  Quasi-static order
  - Dependencies known; little dynamism
  - Determined at compile time
  - No context scheduler overhead

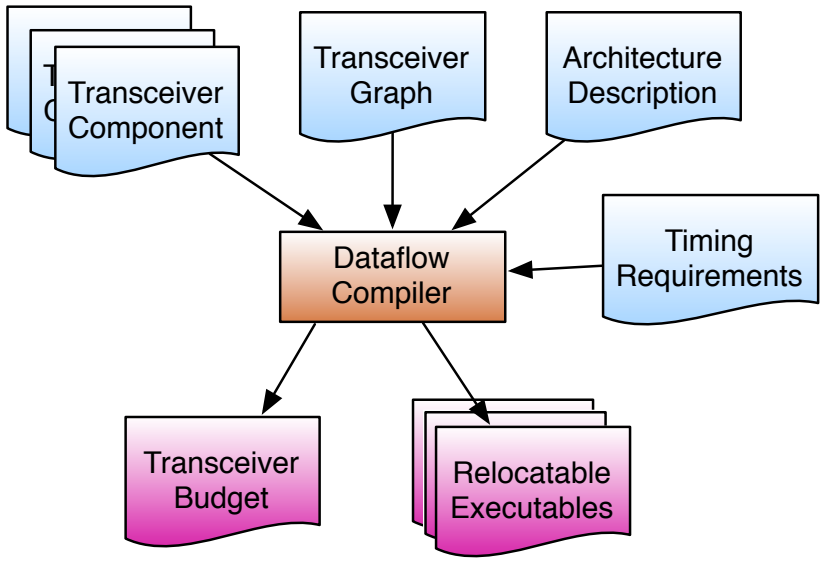
Inter-graph

- Per processor: **Budget scheduler**
  - Guarantees service time per reservation, isolating graph from interferences
  - TDM, NPNBRR, PBS, CCSP(?)
- **Global Resource Manager**:
  - Reservation of resources, processor binding at graph startup

ST ERICSSON

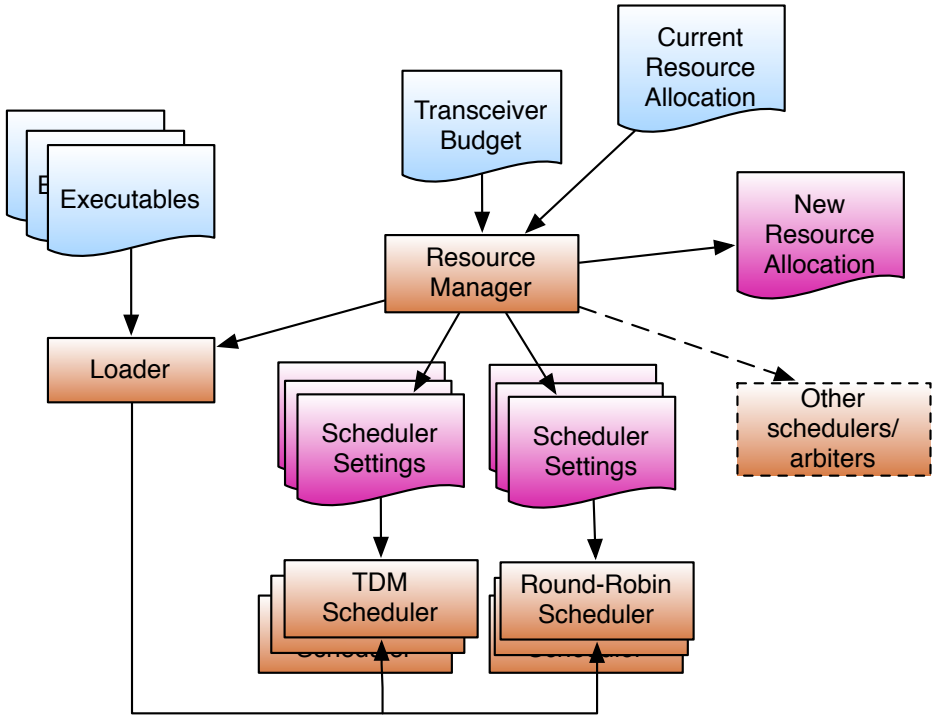
Software Architecture for SDR

Compile-Time (Budgeting)
For each graph



Compute Static ordering per processor
Buffer sizes, Run-time scheduler settings

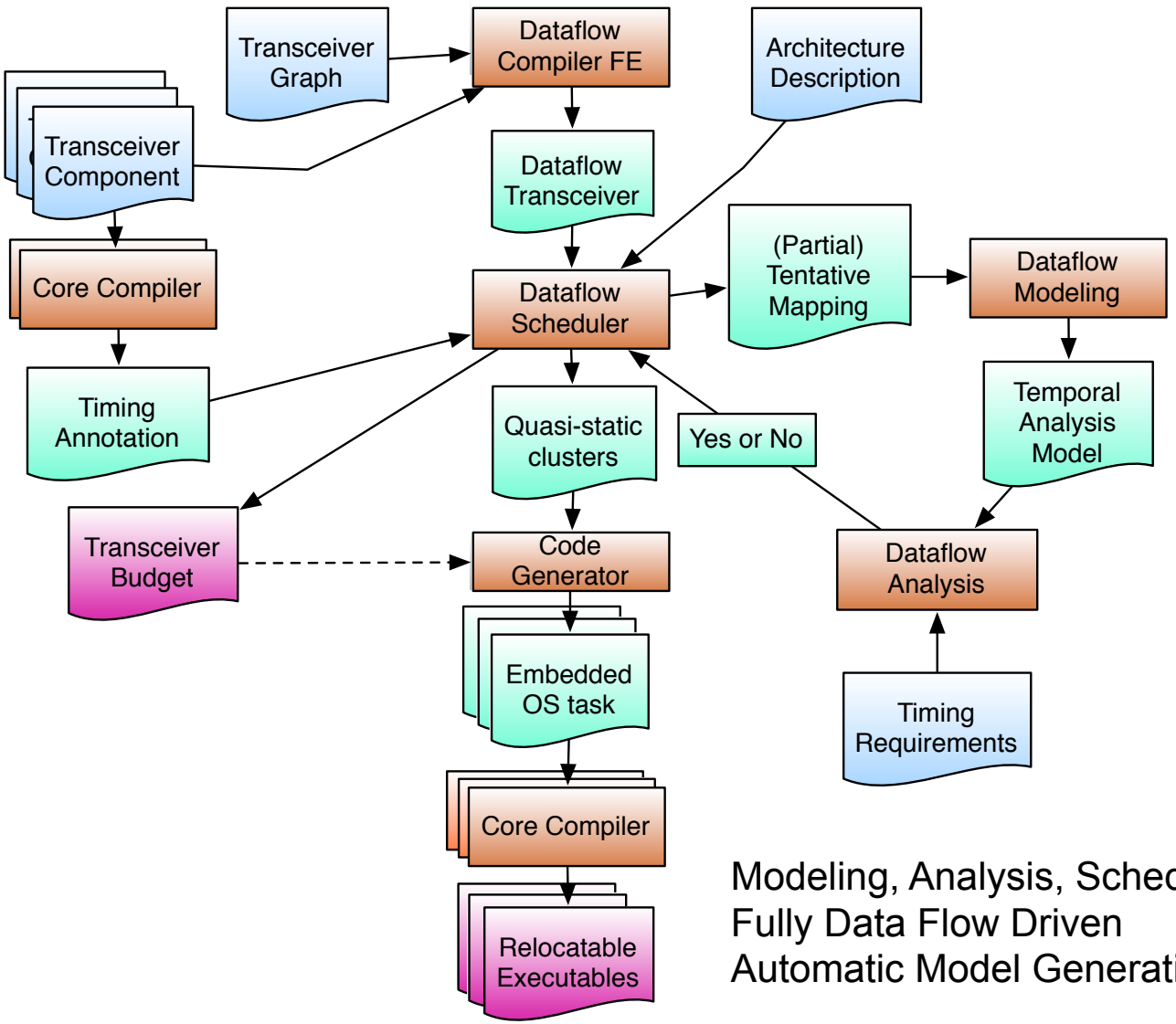
Run-Time (Admission Control)
For each graph start request



Perform admission control, actor to processor binding, load tasks, configure run-time schedulers

ST ERICSSON

Programming Flow in Detail

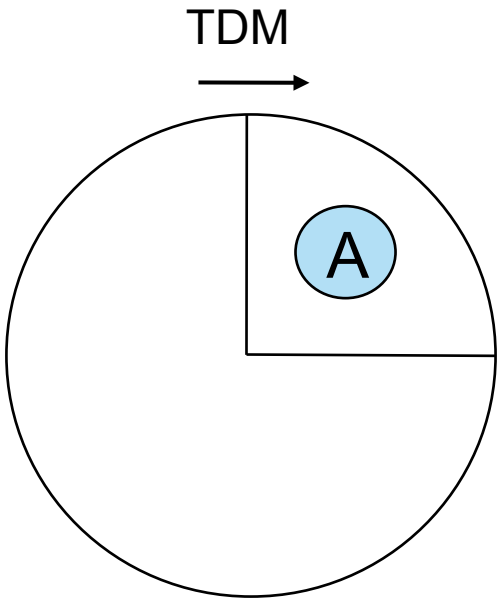


Modeling, Analysis, Scheduling
Fully Data Flow Driven
Automatic Model Generation

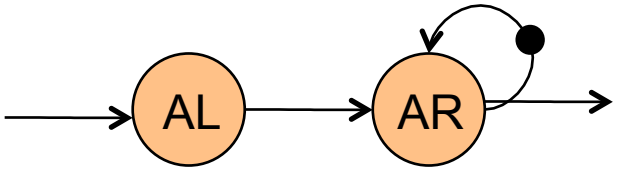
ST ERICSSON

Dynamic Scheduler Modeling: TDM

TDM is a budget scheduler: guaranteed resources per period.
Latency rate model [Wiggers@SCOPES07]: approximation for any starvation-free scheduler, with varying accuracy, depending on the scheduler.



Latency-rate server data flow model

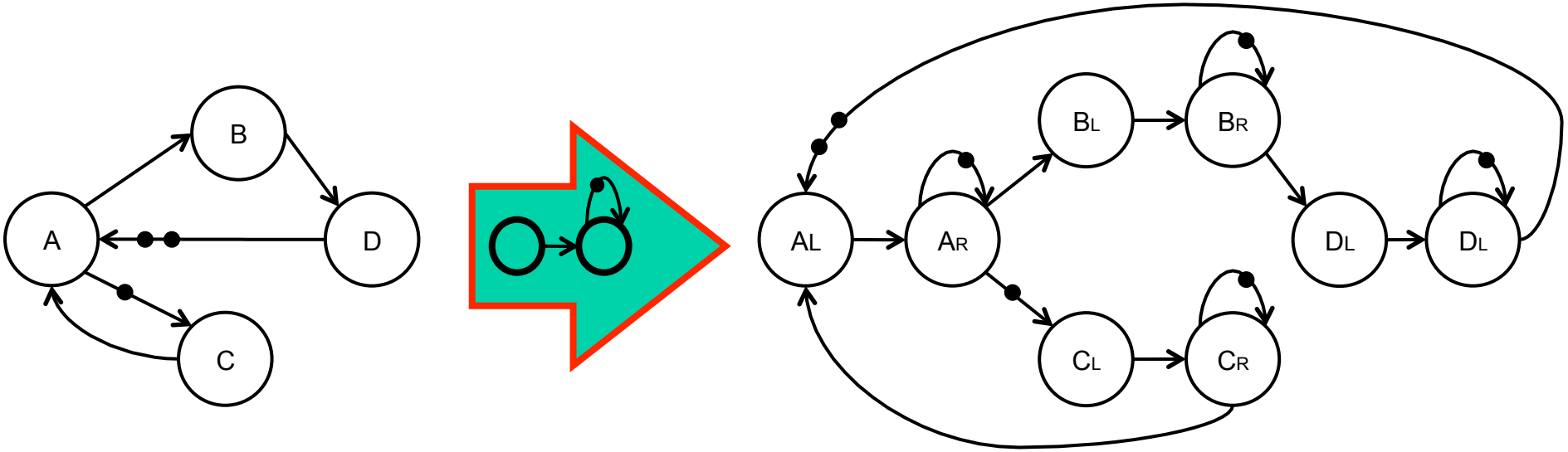


t_{AL}=P-S(A) t_{AR}=P.T(A)/S(A)

P: Period of the TDM scheduler
S(A): Slice allocated to A
T(A) : Worst-case Execution time of A

ST ERICSSON

DF Modeling: Composition of TDM arbitrations



Latency-rate server model can be used for any starvation-free/budget schedulers.
It can for some cases be rather pessimistic.

ST ERICSSON

Data flow Modeling: Problem with the LR-Model

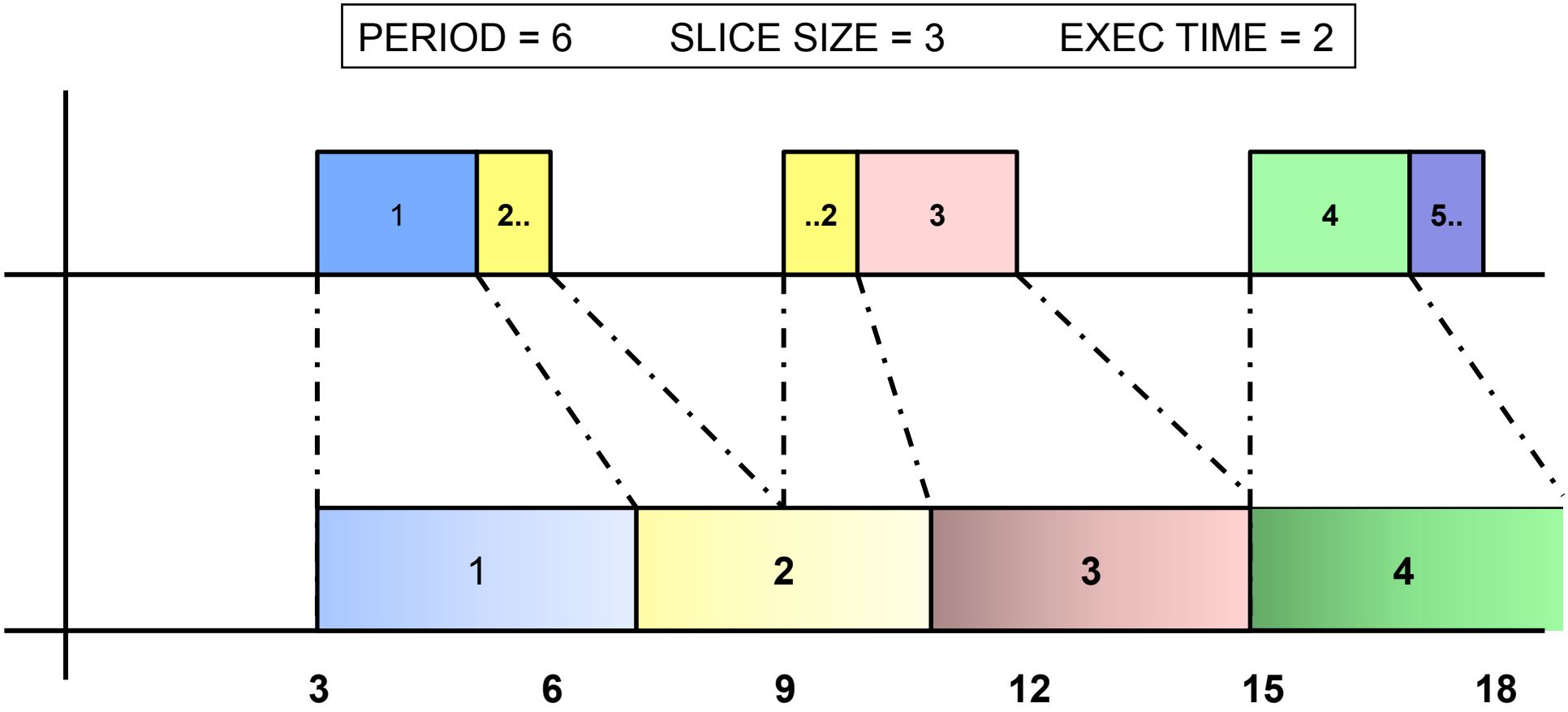


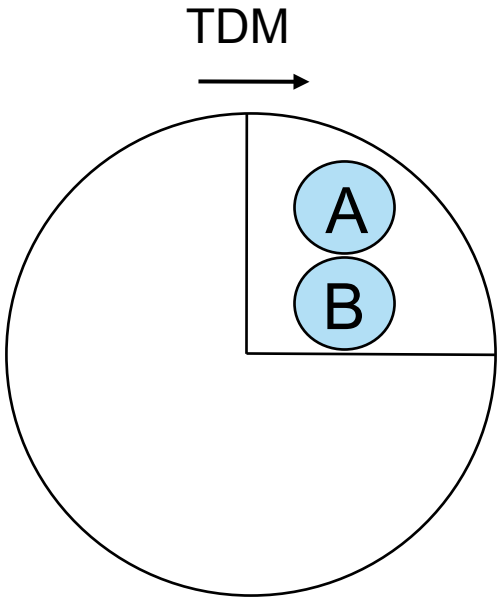
Fig: The LR-model over-estimates the worst-case temporal behavior of TDM arbitration by a factor of **(P/S)**

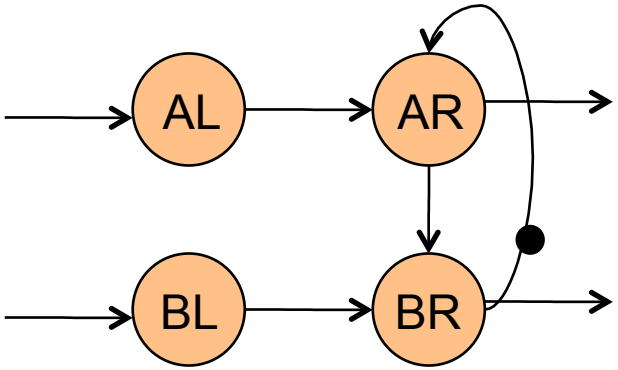
But do not fear. A model with precise worst-case is on the way!

ST ERICSSON

Modeling TDM combined with Static Order

We can compose a data flow analysis for a cluster of statically-ordered actors that share a slice on a TDM scheduler :

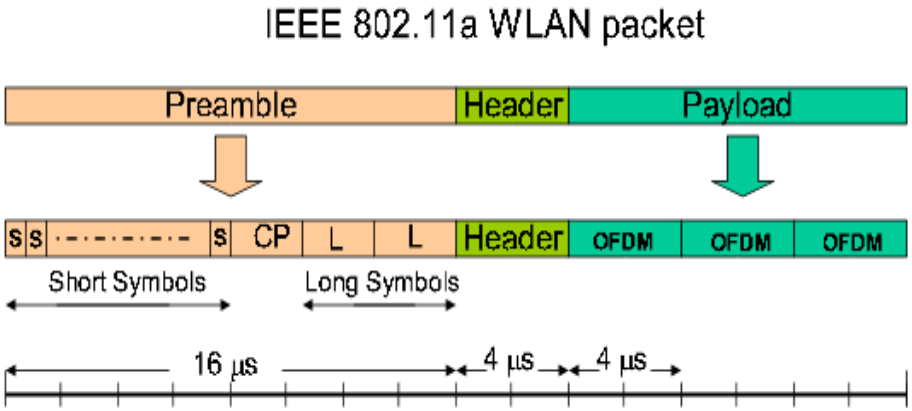


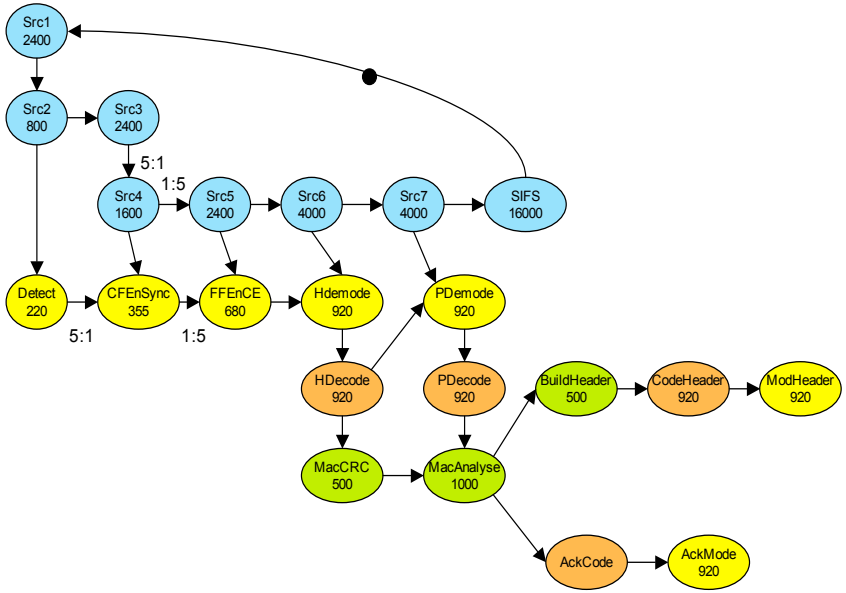


Latency component does not affect local (intra-cluster) communication.

ST ERICSSON

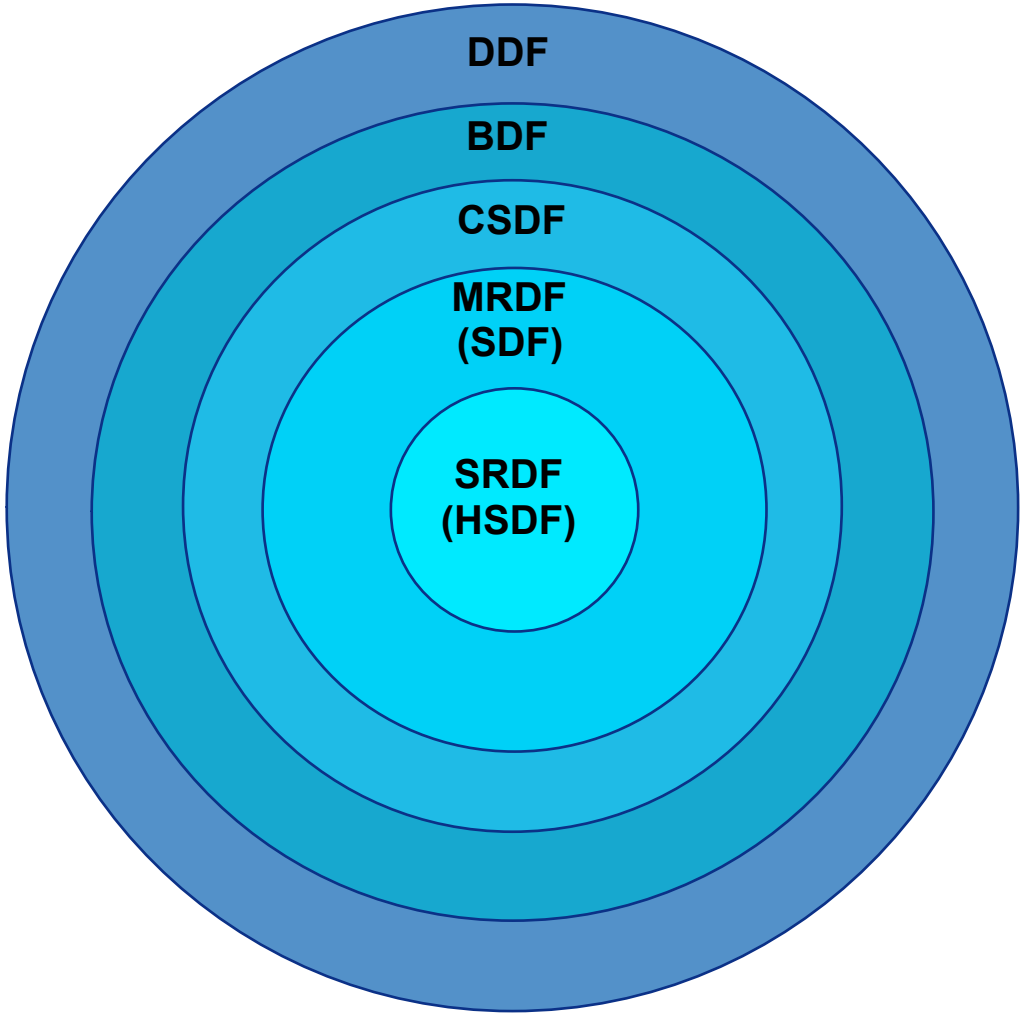
WLAN Packet structure and processing

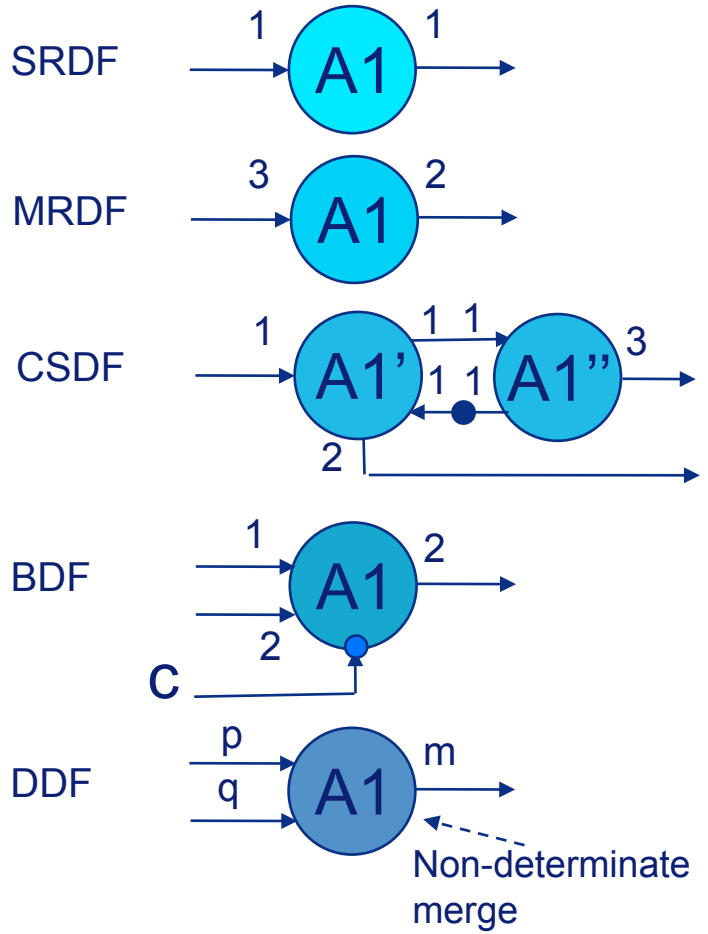




Can Static Data flow handle this?
We can **manually** design a **worst case model for analysis…**
…But that doesn't work for specification, DF compilation, code, generation.
It is **difficult**, **time-consuming, error prone**…
…And how do we guarantee that the model is correct?

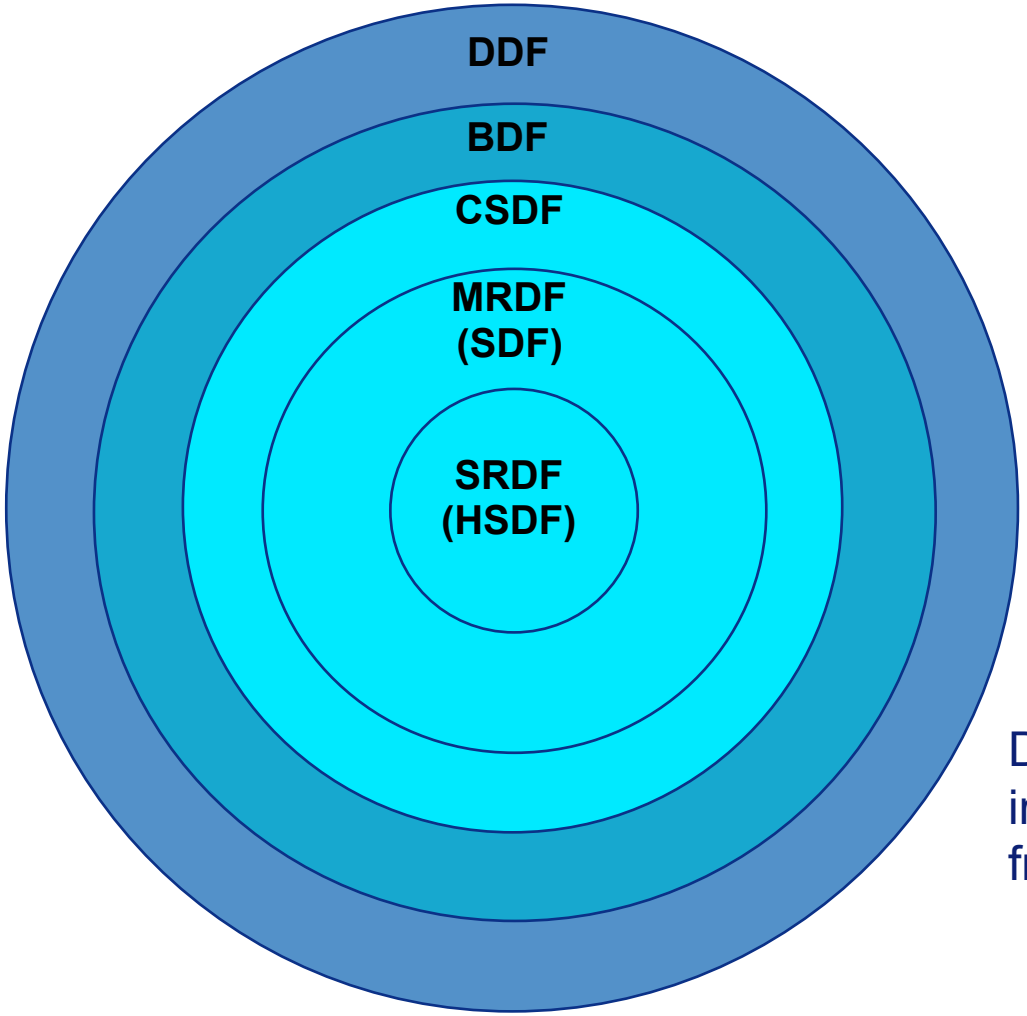
The right flavor of data flow: Expressivity





ST ERICSSON

The right flavor of data flow: Analyzability



Max algebra can be used to derive properties SRDF graph

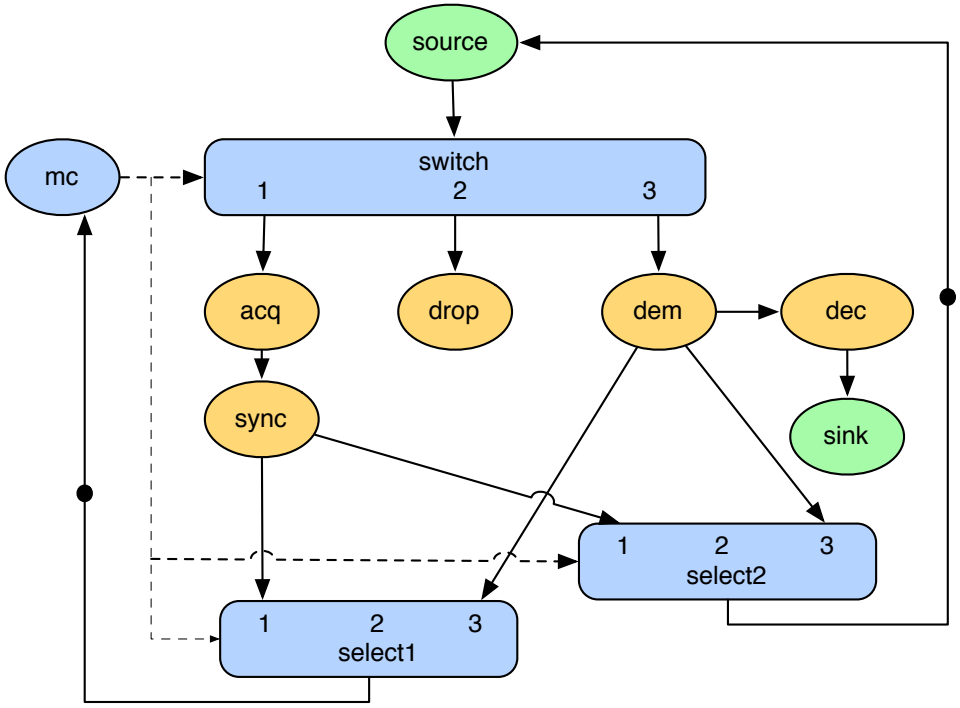
- deadlock free
- monotonic
- Self-timed behavior bounded by static-periodic schedule with max rate
- Static periodic schedule can be built from linear constraints
  - Linear/Convex Programming!

CSDF can be transformed into SRDF
MRDF can be transformed into SRDF

DDF and BDF are Turing complete, impossible to check even for deadlock freedom in the general case.

ST ERICSSON

DF model for Radio: Mode-Controlled Data-flow



DVB-T Receiver

3 Modes: Sync, Decode, Drop

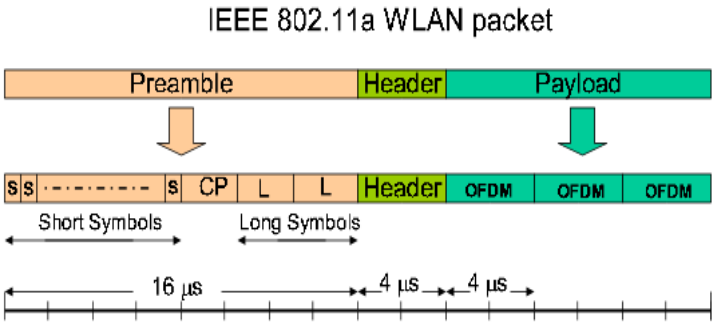
Extension of Static Dataflow. Allows (limited) data-dependent behavior.

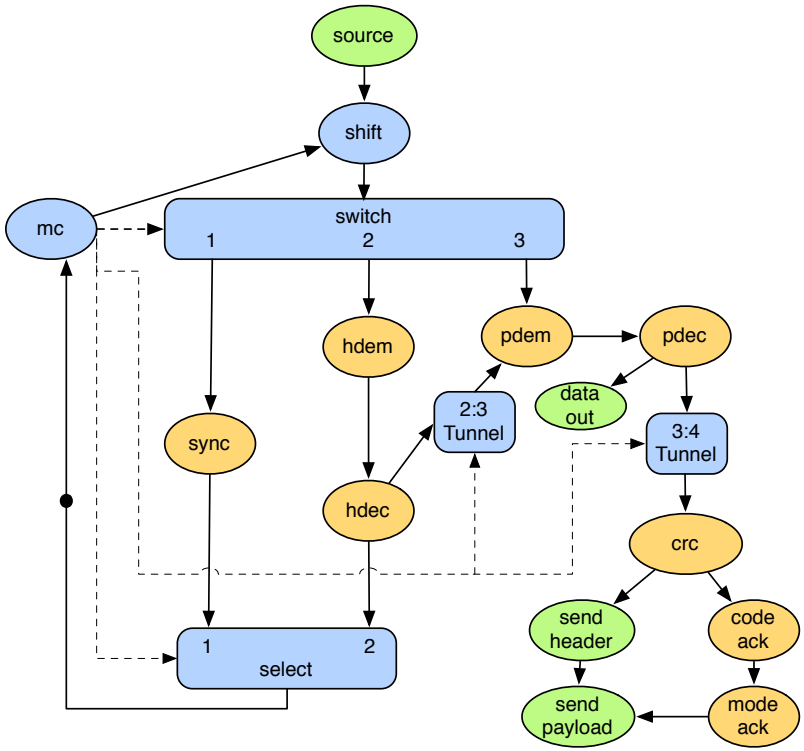
Properties are similar to Scenario-Aware Data Flow (TUE)

Boolean data flow with strict construction rules & valid control sequences defined

ST ERICSSON

Our Computation model: Mode-Controlled Data-flow





Extension of Static Dataflow. Allows (limited) data-dependent behavior.

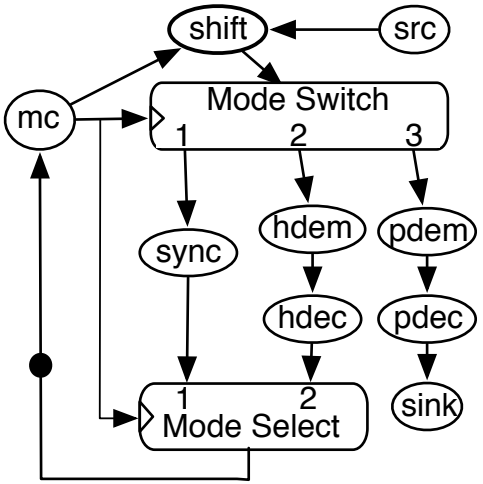
Boolean data flow with strict construction rules & valid control sequences defined

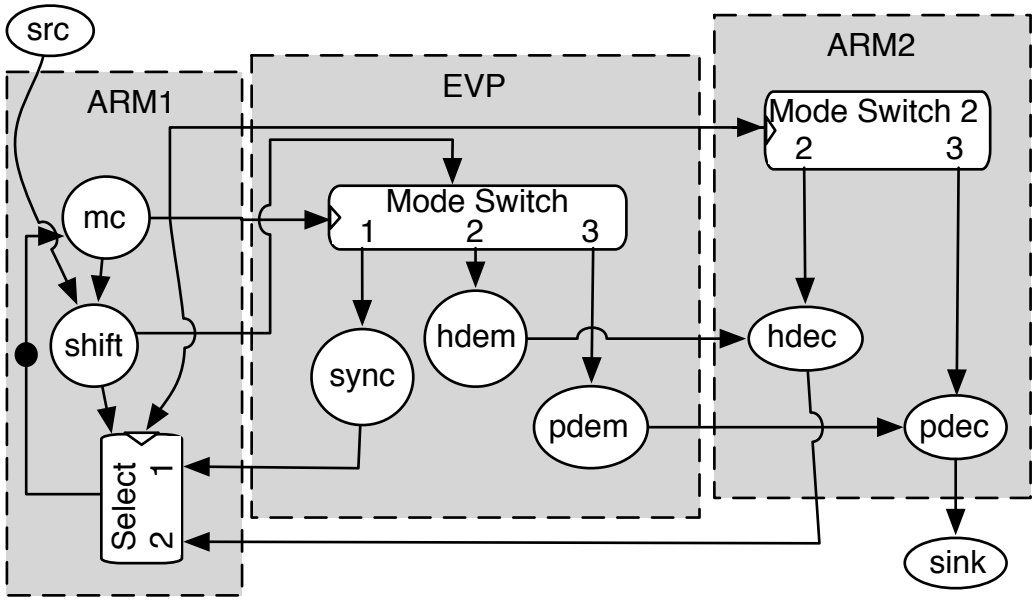
- **Analysis:** Monotonic, Strict, deadlock analysis, periodic bound per mode exists
  - Based on bounding self timed execution per mode and computing transitions
- **Scheduling:** Quasi-static ordering of actors possible, bounded buffers exist

ST ERICSSON

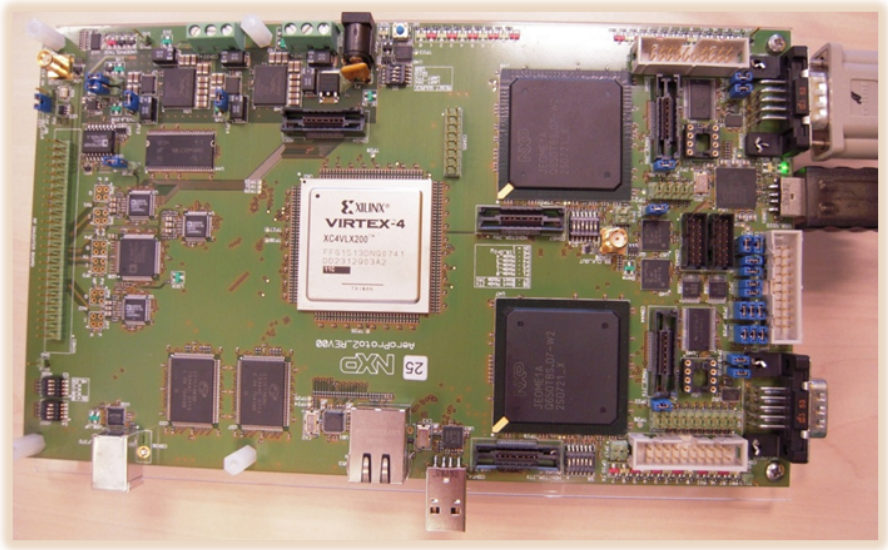
Quasi-static ordering (extension for MCDF)

Order actors inside a cluster as much as possible at compile-time
The only run-time decision is mode switching
Synchronization among clusters is handled by FIFOs





Demonstrator (2009)



Collaboration ST-Ericsson/Nokia/NXP.

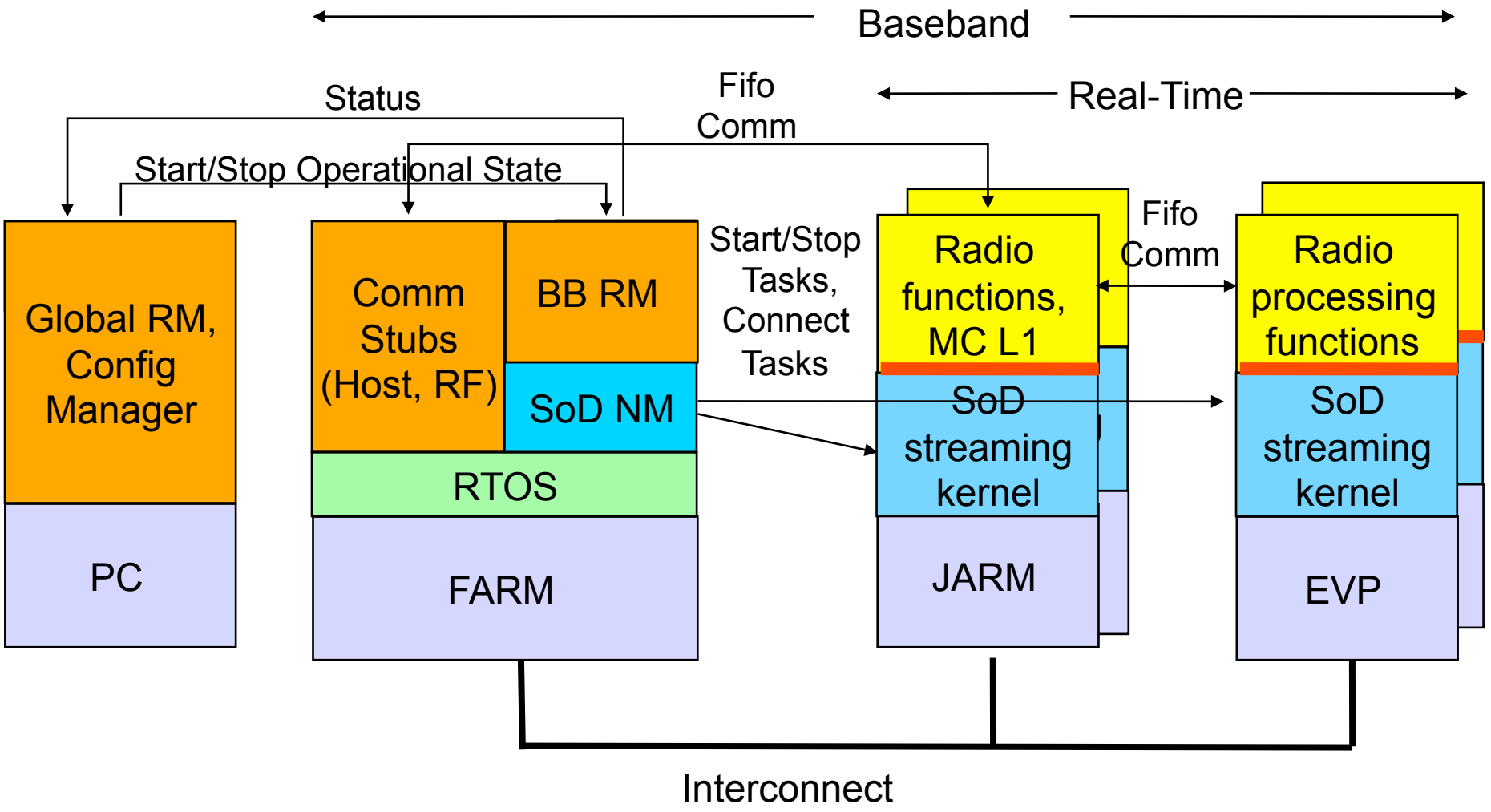
HW: Dresden Prototype Board for LTE – 3 ARMs, 2 EVPs

All run-time components implemented, including:

- Predictable local schedulers;
- Fifo-based communication and synchronization, self-timed execution
- Multi-RAT resource manager w/ run-time task and memory mapping
- EVP code relocation w/ run-time loader.

ST ERICSSON

Software Architecture - Run-time of Demonstrator



Conclusions

Messages:

- Data flow has many attractive properties as a real-time analysis model for iterative applications distributed on a multiprocessor
- Data flow has many attractive properties as a concurrent programming model
- Budget scheduling is essential for independent analysis and independent behavior of applications
- Automatic generation of the analysis model from the implementation code is essential to automation
- The right flavor of data flow is domain-specific

Future Scope:

- Better modal analysis, better modeling, better scheduling techniques
- Tool maturity
- Link between data flow language and data flow analysis, language design

ST ERICSSON


LET'S
CREATE
IT
THANK YOU
ST
ERICSSON