Radio Communications Research Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ALOE Framework and Waveform Design Workshop

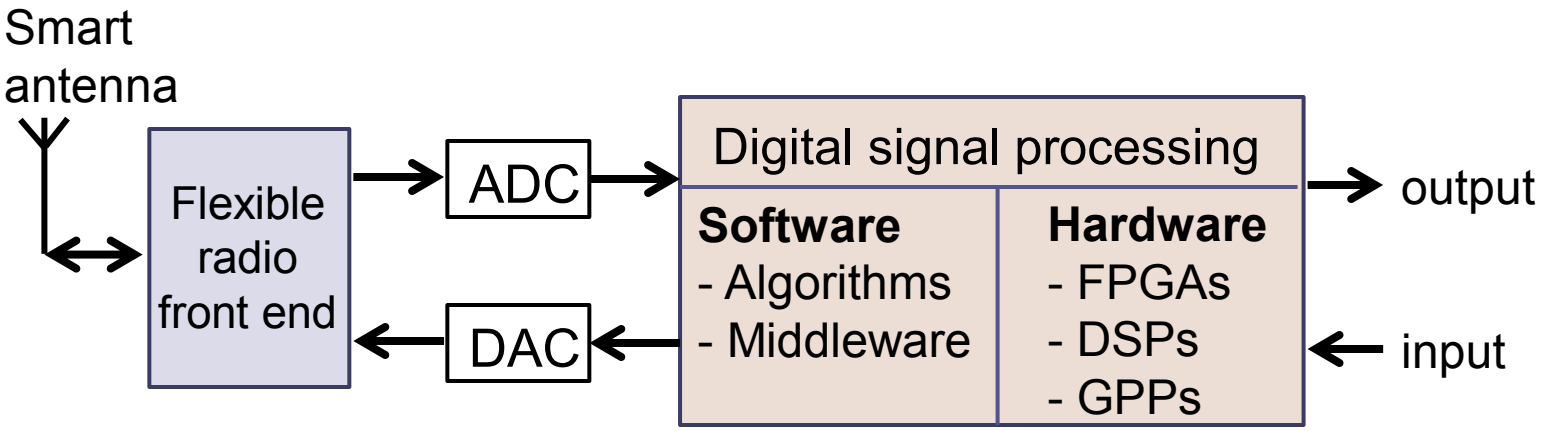
Vuk Marojevic
Ismael Gomez
Antoni Gelonch

SDR – WInnComm – Europe 2013

Outline

1. Context
2. ALOE Concepts and Framework
3. Computing Resource Management
4. Waveform Development and Deployment
5. Conclusions

SDR Model



The software radio or SDR provides a flexible radio architecture that allows changing the radio personality, possibly in real-time

---

J. H. Reed, *Software Radio – A Modern Approach To Radio Engineering*, 2002 Prentice Hall PTR, Upper Saddle River, NJ.

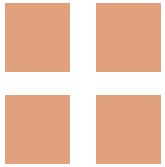
Software

- **Waveforms (SDR applications)**
  - DSP algorithms
  - Radio's physical layer behavior
- **Middleware (SDR framework)**
  - Software layer between applications and hardware
  - Execution environment for waveforms
  - Individual hardware and software development
  - Waveform loading and unloading → reconfiguration
  - Portability and reuse of components

Hardware

TODAY

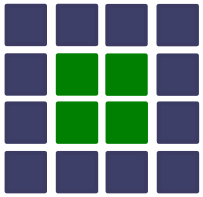
multicores



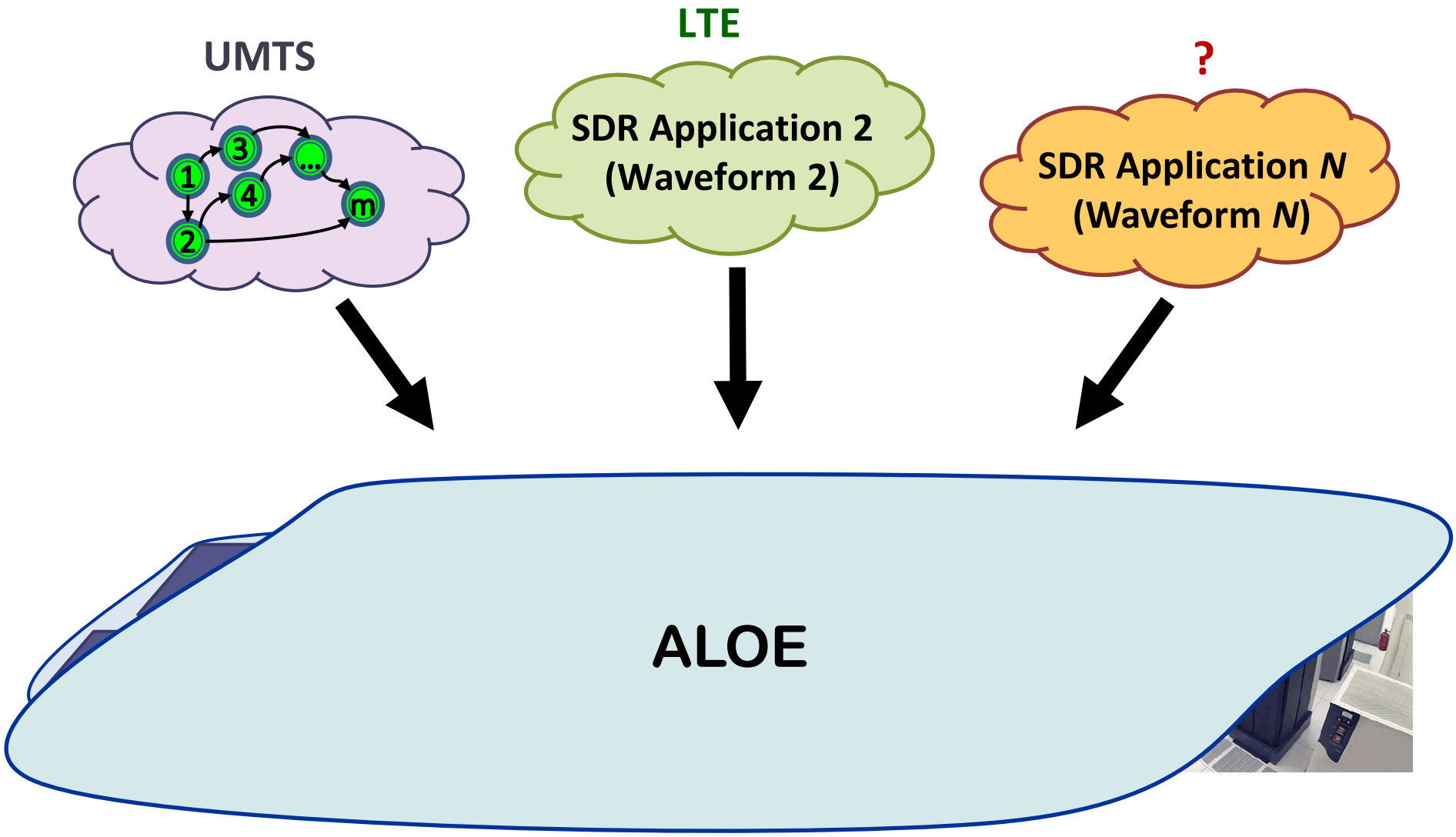
small clusters
(heterogeneous)

TOMORROW

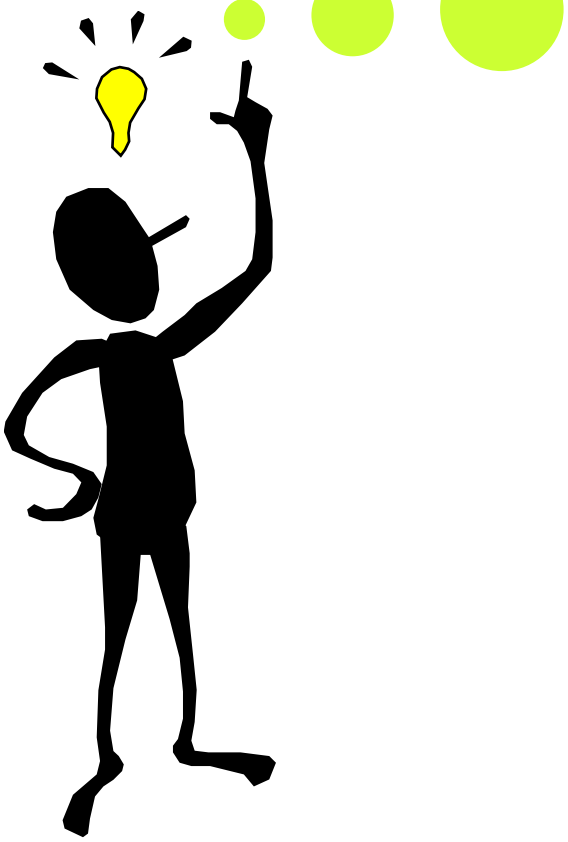
many-cores



ALOE Context



SDR Computing



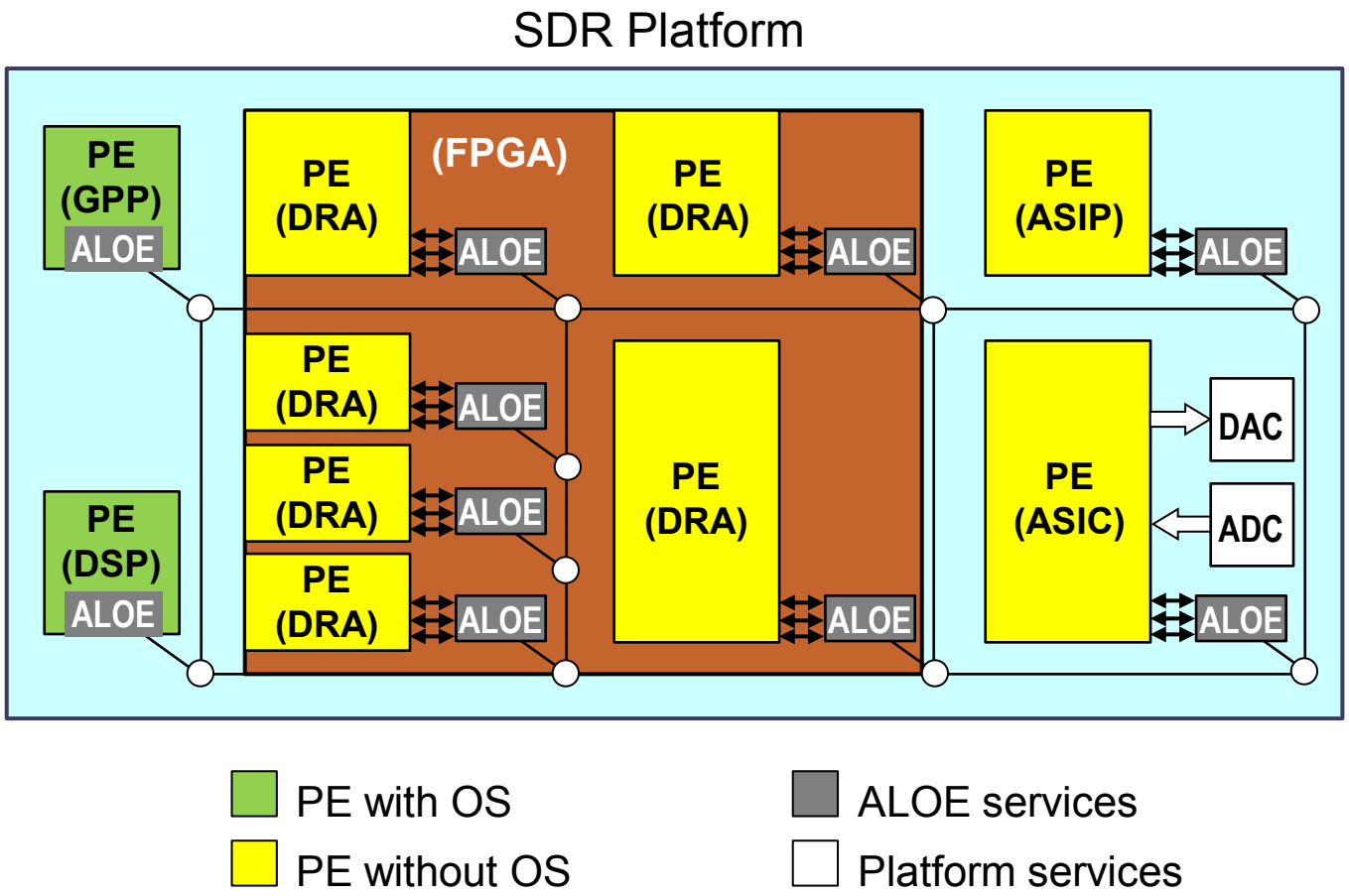
1. Multiprocessing
2. Lightweight
3. Platform Independence
4. Computing Resource Management

2004…

ALOE Tools

Open source

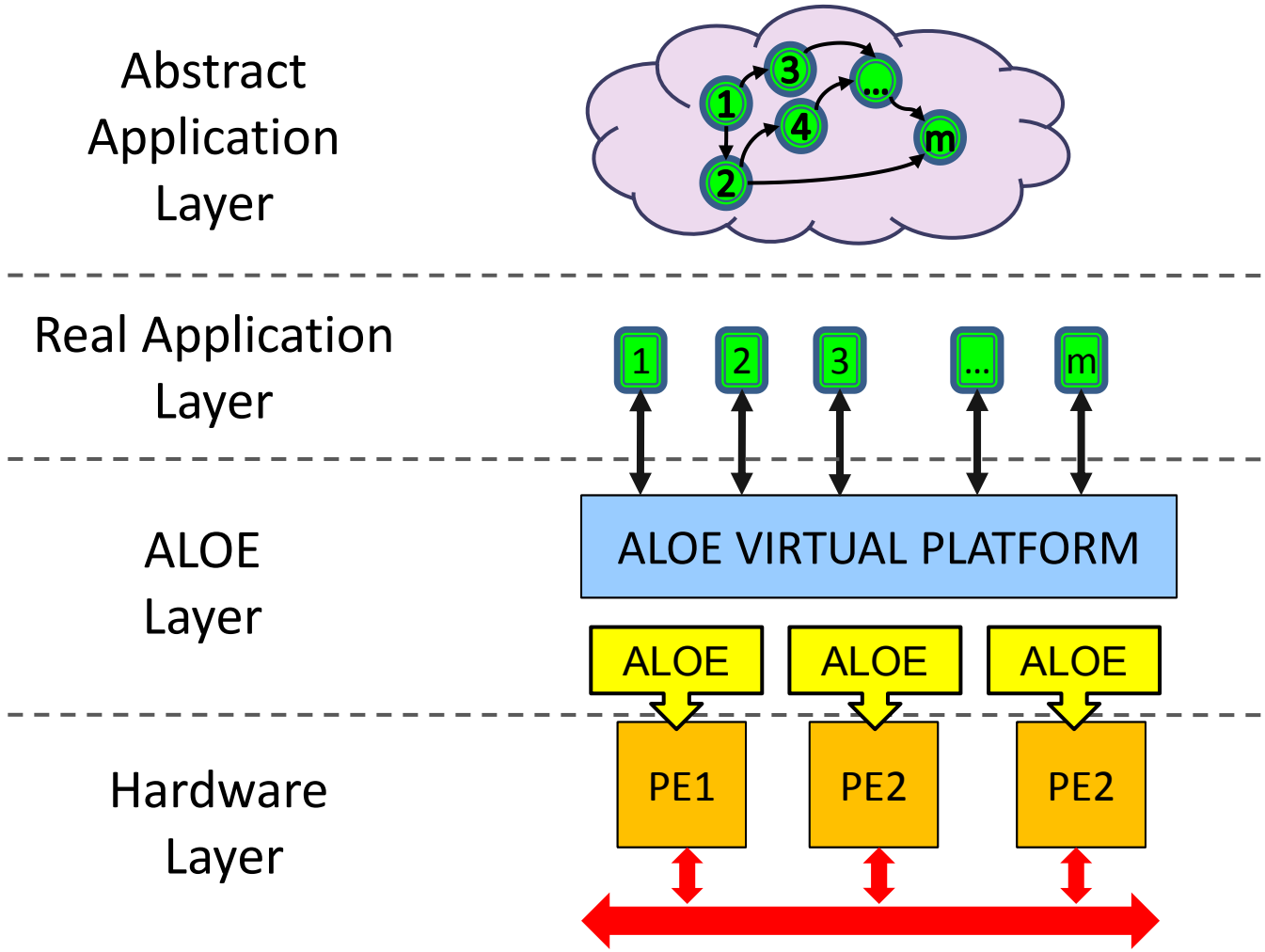
Heterogeneous Multiprocessing



PE: processing element

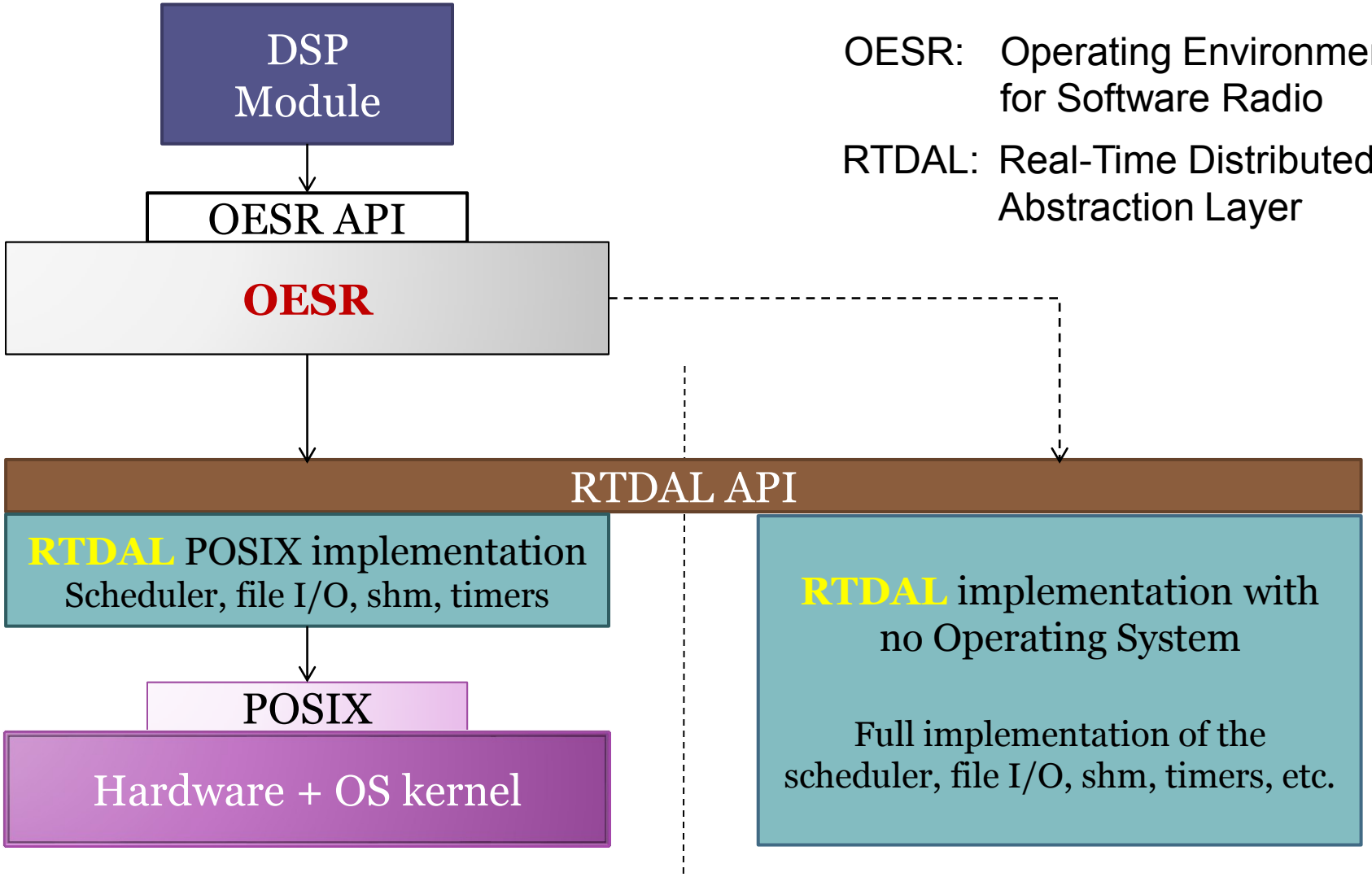
DRA: dynamically reconfigurable area

ALOE Layers



PE: Processing Element

ALOE Architecture



Real-Time Distributed Abstraction Layer (RTDAL)

- Interprocessor communication
- Synchronization
- Scheduling
  - pipelined execution, partitioned scheduling
  - 1 thread per processing core
- RTDAL API
  - Task creation and management
  - Interfaces
  - ADC/DAC abstraction
  - Time functions

Operating Environment for Software Radio (OESR)

- Automatic mapping of waveforms
- Location-transparent inter-module communications
- Configuration and visualization of variables & parameters
- Logs, counters, …

Computing Resource Management

Wireless Communications Characteristics

- Continuous data transmission and reception
- Real-time services → real-time processing
- RAT/mode/QoS target → processing demands
- Heterogeneous multiprocessor platforms
- Limited computing resources
- Dynamic reconfigurations

RAT: Radio Access Technology

Computing Resource Management

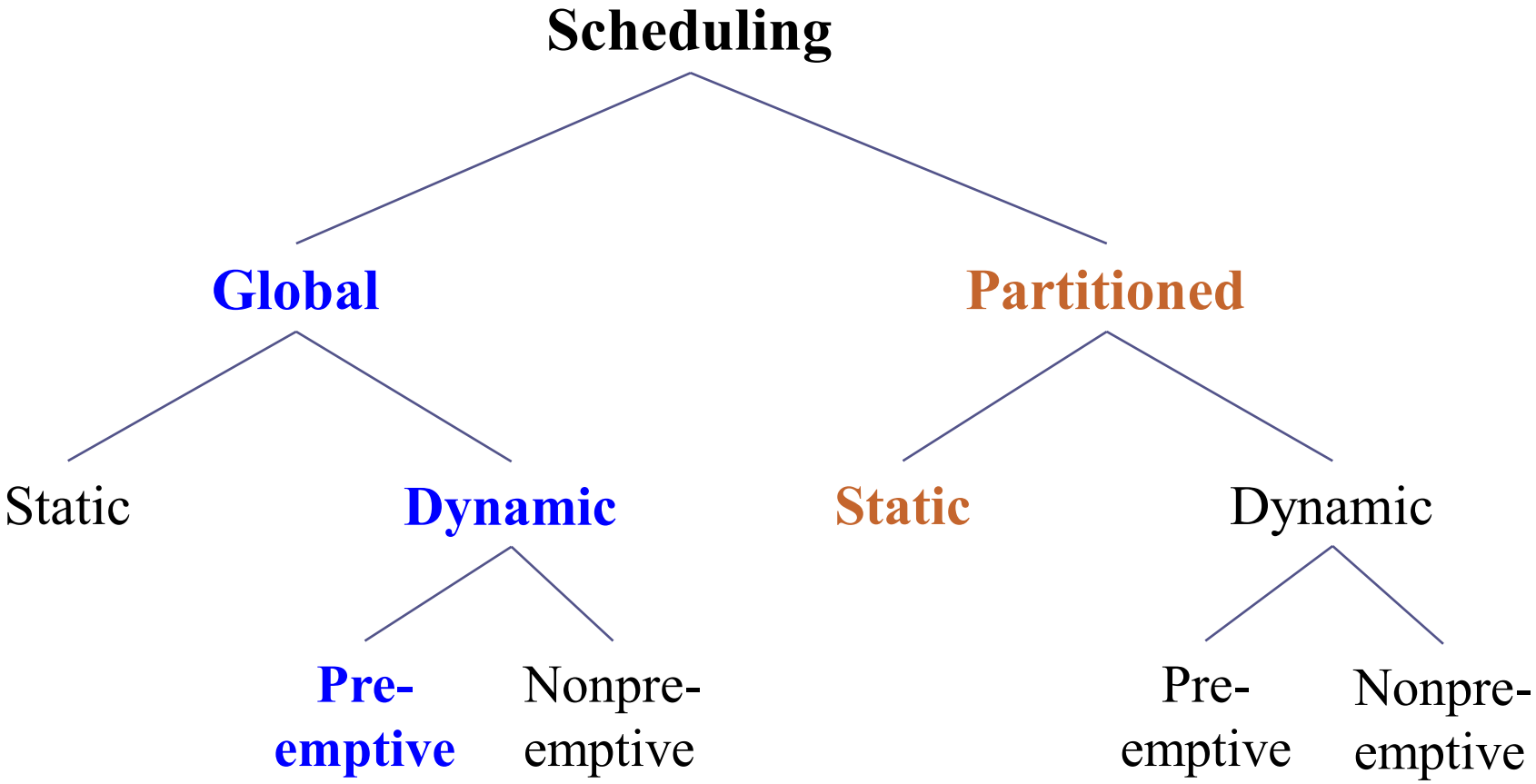
"Provide sufficient computing resource to waveforms for real-time processing"

Real-time constraints:

- *Minimum **throughput***
- *Maximum **latency***

Scheduling

Scheduling is the method by which threads, processes or data flows are given access to system resources. [Wikipedia]



Static vs. Dynamic Scheduling (I)

Static	Dynamic
Offline (compile time), before execution	Online (runtime), at execution
Deterministic performance	Nondeterministic performance
Avoid migrations → less overhead & fewer cache misses	Migrations → overhead & cache misses
Avoid task locks → less system calls → less overhead	Task locks → more system calls → more overhead
Regular, periodic tasks with a priori information	Irregular, aperiodic tasks with unknown characteristics a priori
Runtime rescheduling costly	Easy to add new task at runtime

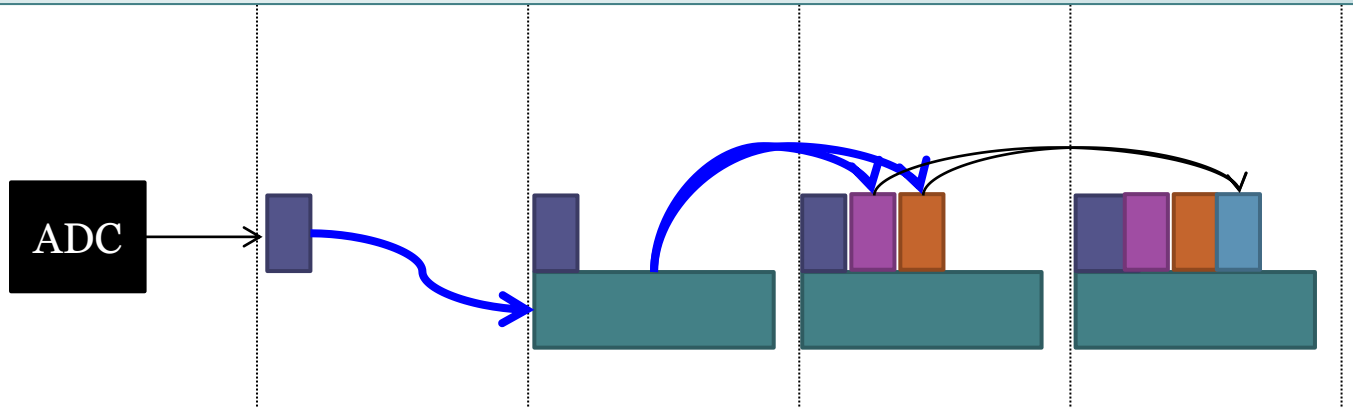
Static vs. Dynamic Scheduling (II)

- Scheduling overhead increases…
  - …with the waveform granularity
  - …with the number of processing elements
  - …inversely to task execution time
- *Global-dynamic-preemptive* scheduling
  + flexible
  - may incur significant resource overhead
- *Partitioned-static* scheduling scales better

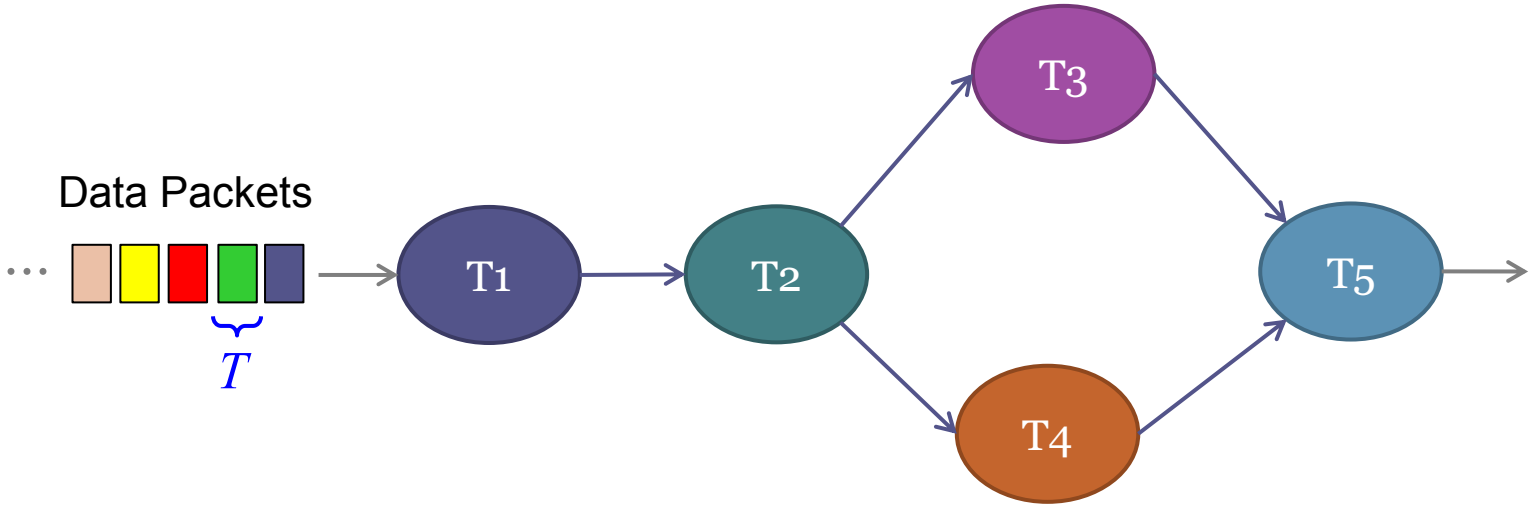
V. Marojevic, I. Gomez, A. Gelonch, "Evaluation of computing resource management methods for SDR clouds," *SDR-WInnComm Conf.*, Washington DC, 2013.

ALOE Resource Management

- **Pipelining**
- **Partitioned scheduling**: static and cooperative
  - Low overhead
  - Easy to implement
  - Scalable
- **Heterogeneous platforms** (w/o shared memory)
- Requires task-to-processor **mapping**



Pipelining



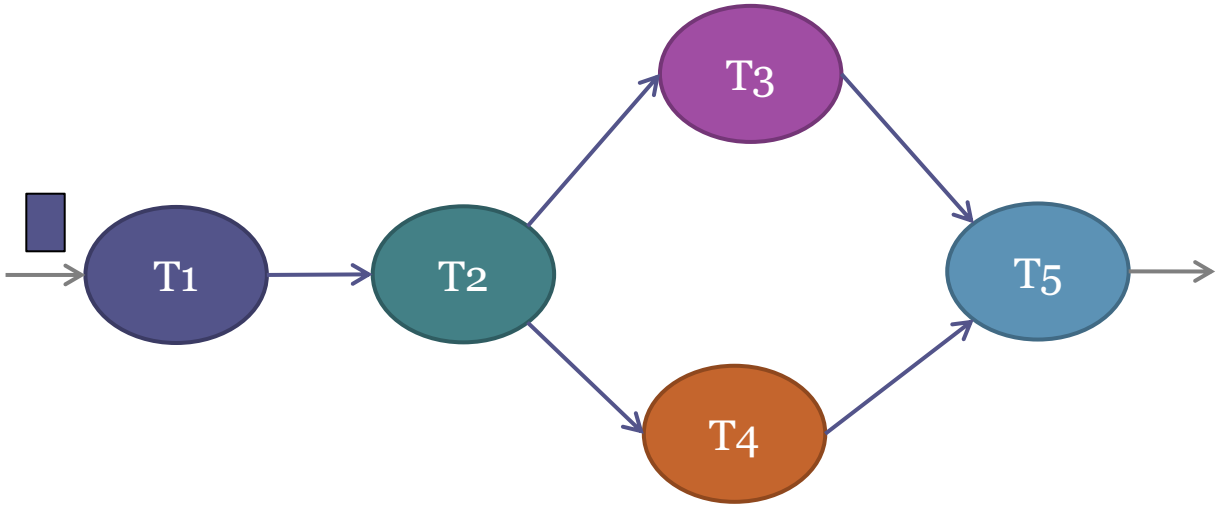
Data packet: x samples

Packet arrival rate: $1/T$ Hz → ***Throughput requirement***: x/T samples/s

Process 1 input data packet every T seconds

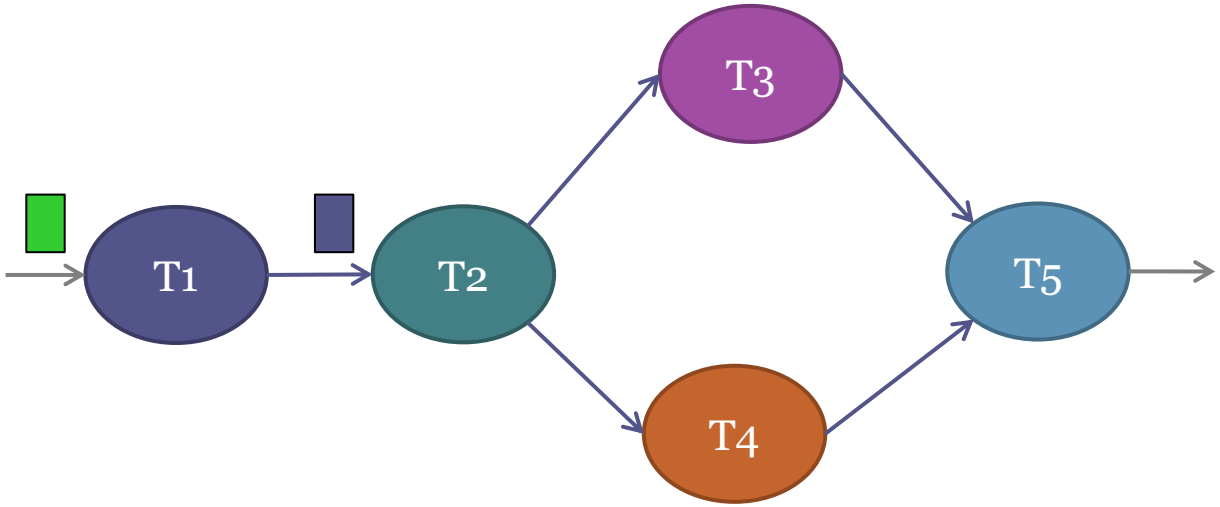
Pipelining

$t = 0$



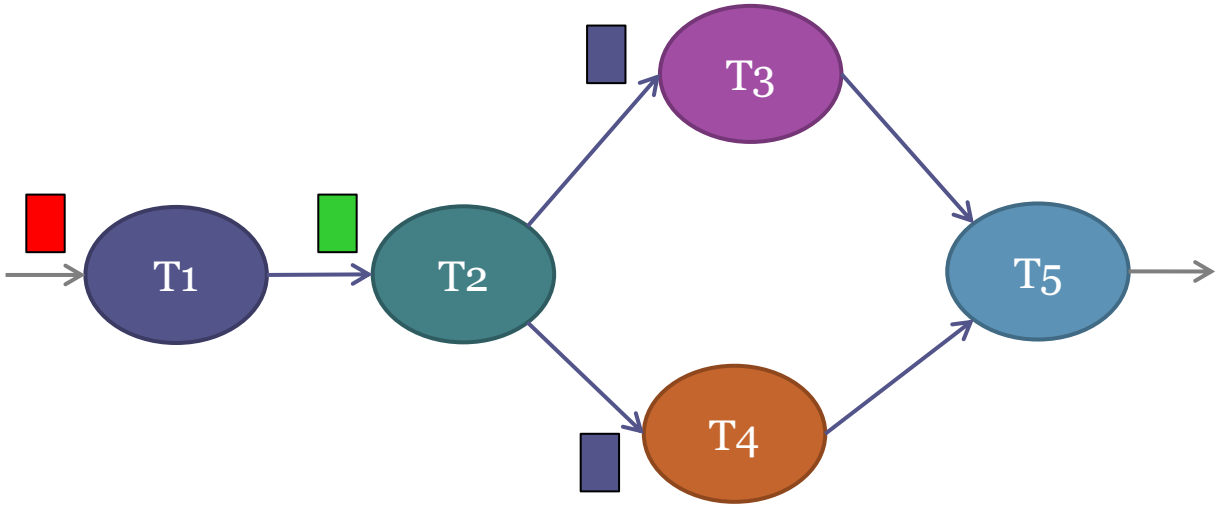
Pipelining

$t = T$



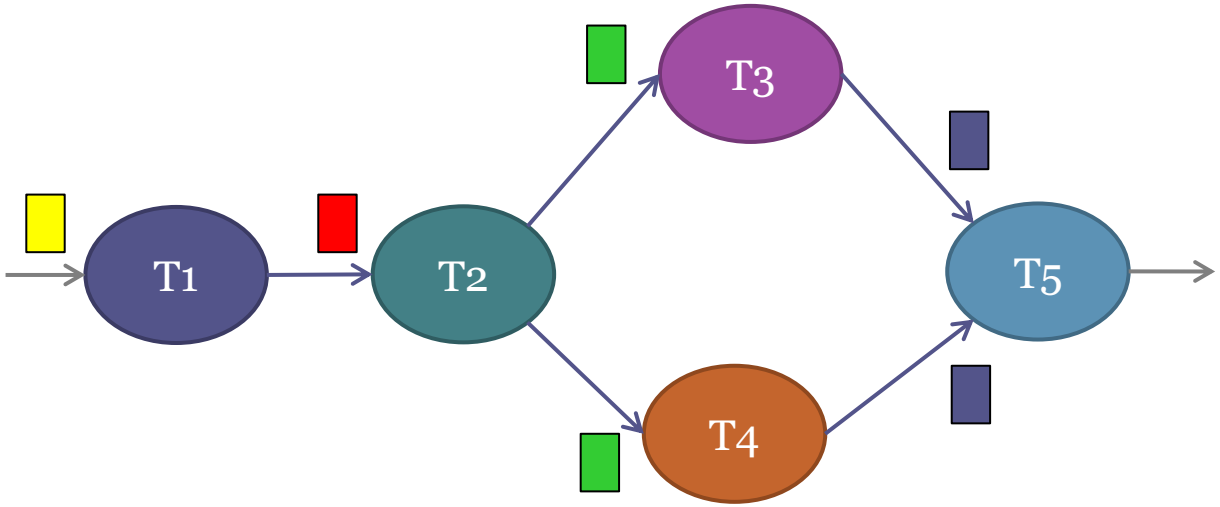
Pipelining

$t = 2T$



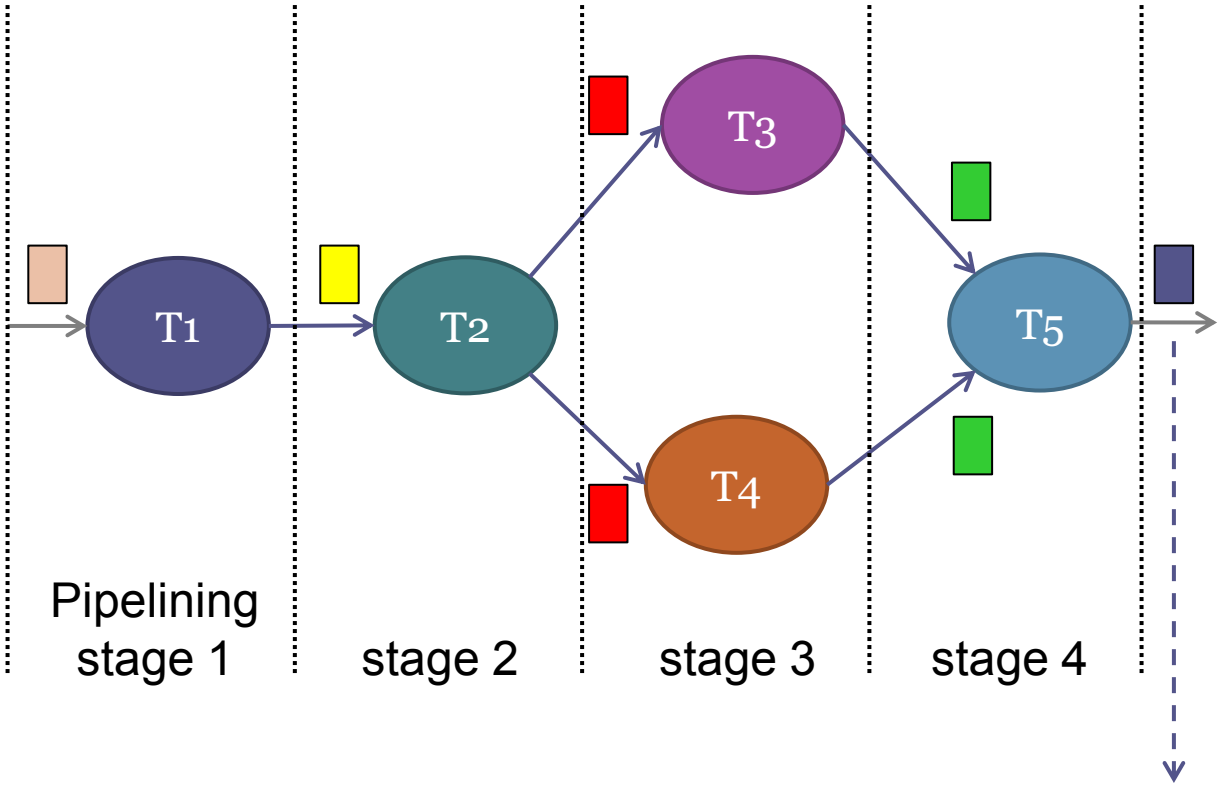
Pipelining

$t = 3T$



Pipelining

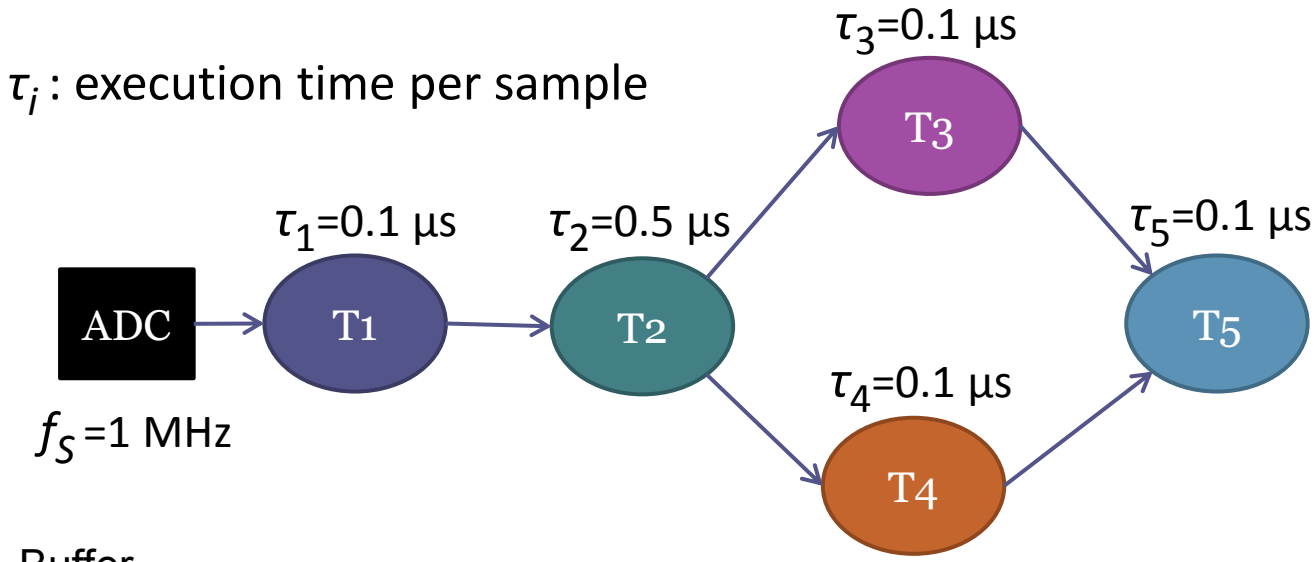
$t = 4T$

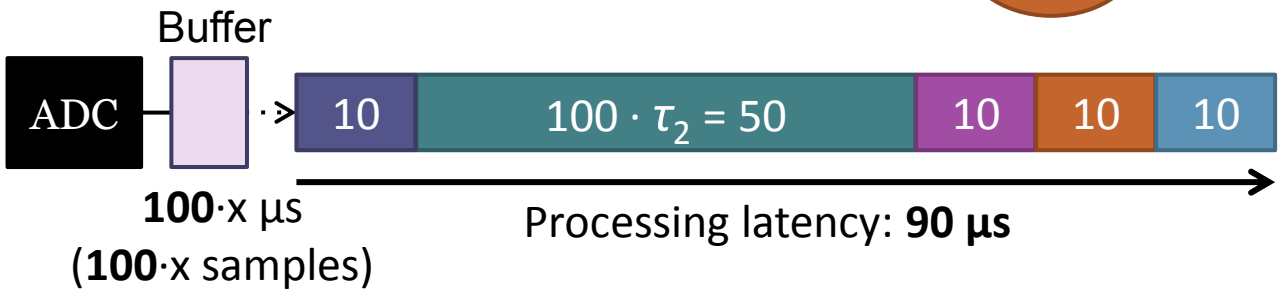


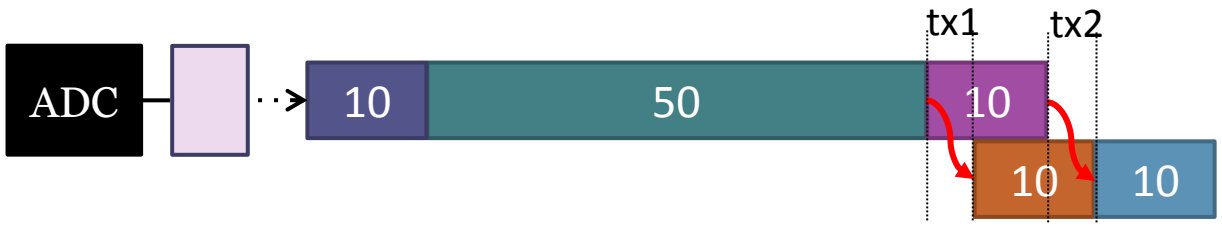
Processing latency: 4 time slots ($4 \cdot T$)

Processing throughput: 1 packet every T seconds (x/T input samples/s)

Scheduling Example w/o Pipeline

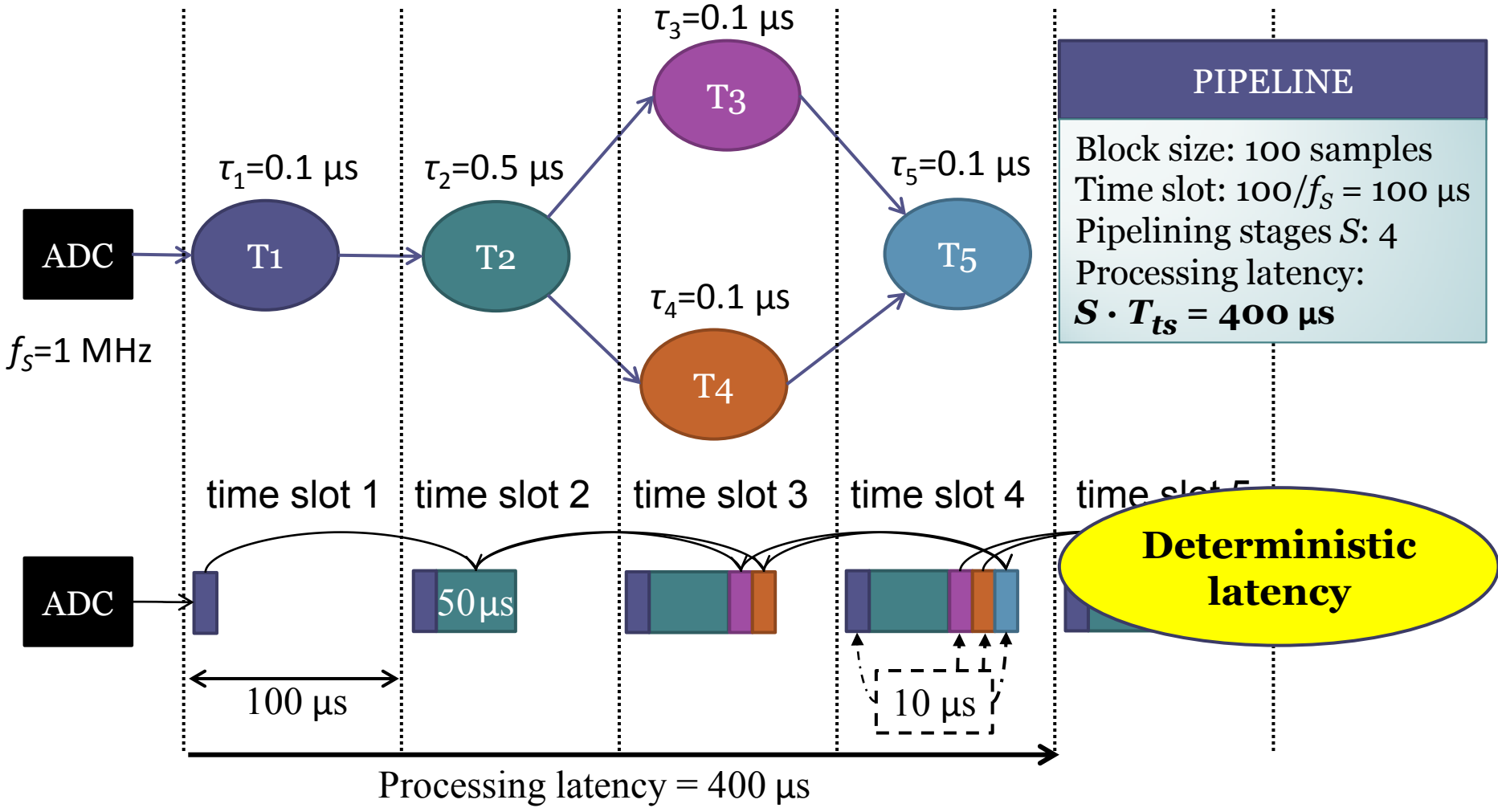






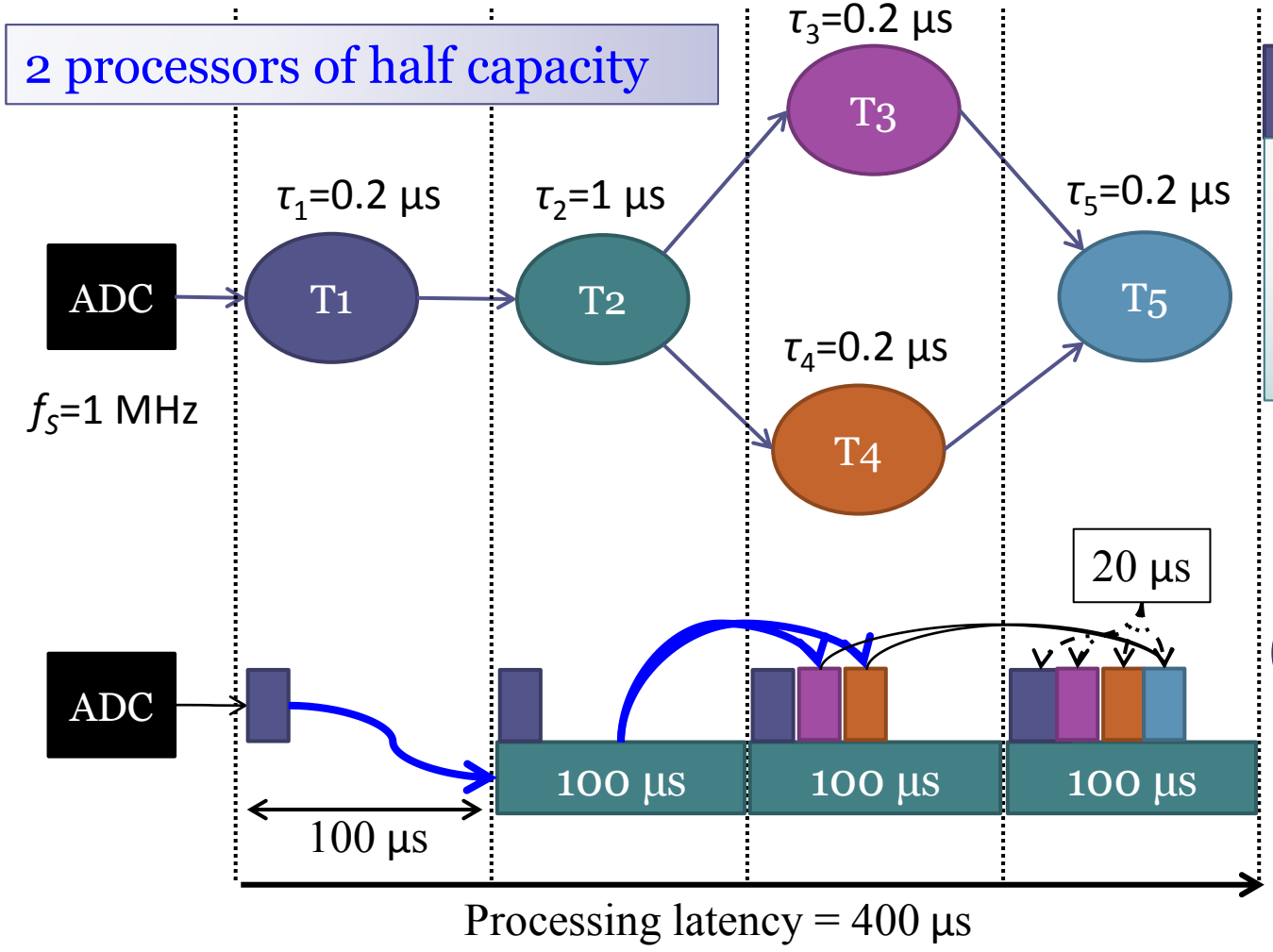
Latency: *f*(platform, mapping)

Pipelined Scheduling (I)



Pipelined execution: removes precedence constraints, simplifies scheduling

Pipelined Scheduling (II)



PIPELINE

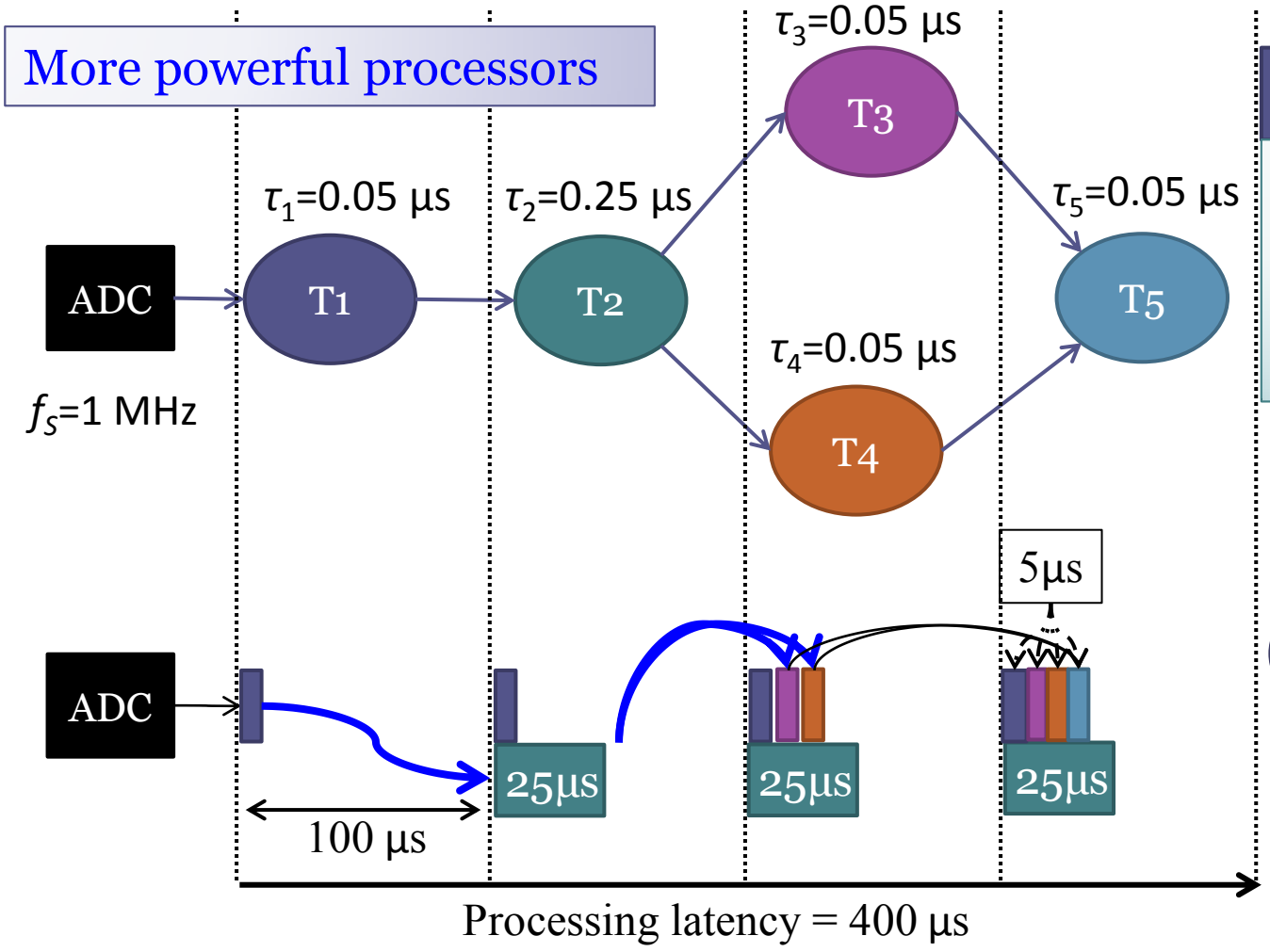
Block size: 100 samples
Time slot: $100/f_S$ = 100 µs
Pipelining stages S: 4
Processing latency:
$\boldsymbol{S \cdot T_{ts}}$ **= 400 µs**

Equal latency and throughtput

Scheduling performance: platform-independent

Pipelined Scheduling (III)

More powerful processors



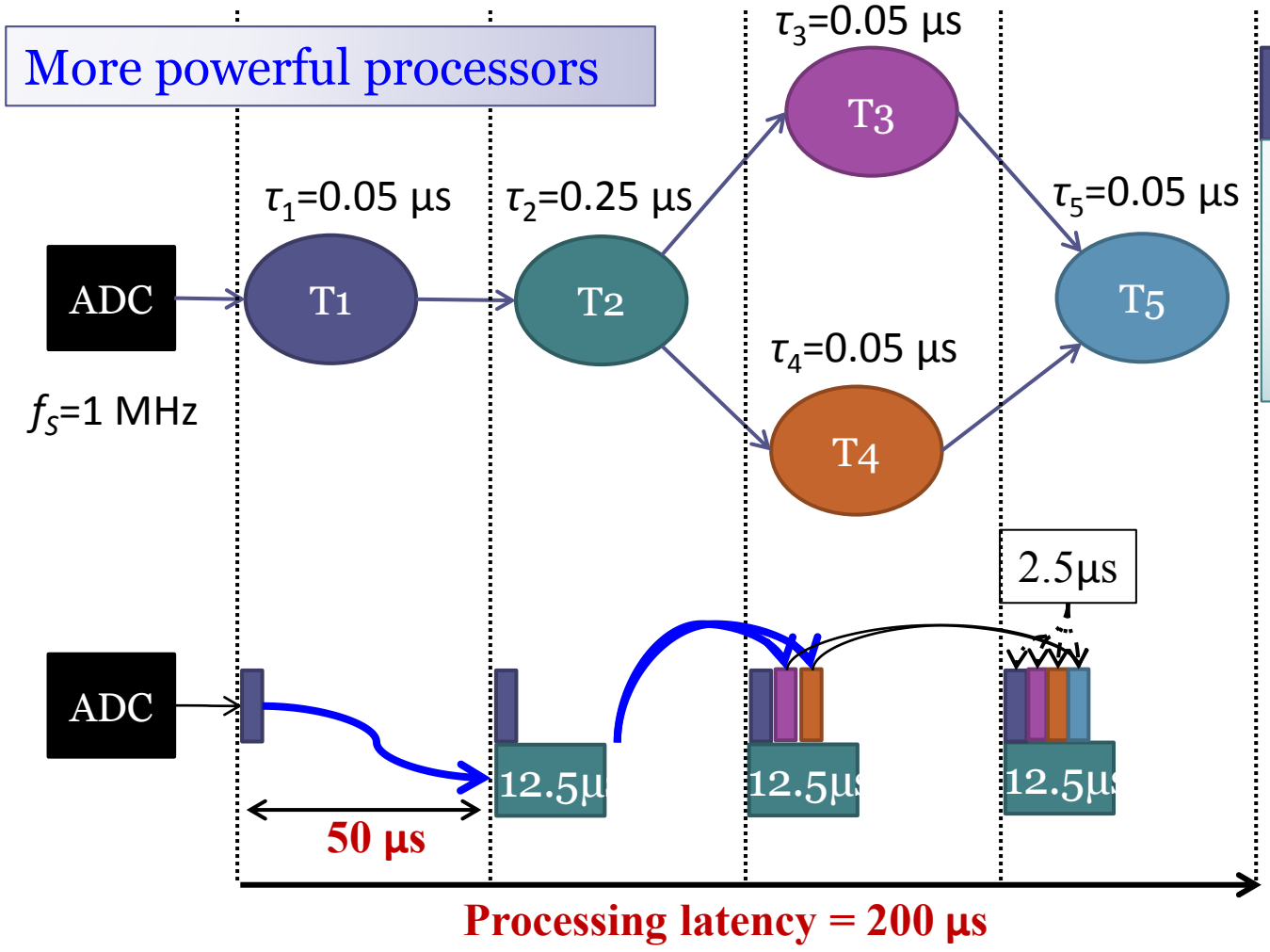
PIPELINE

Block size: 100 samples
Time slot: $100/f_S$ = 100 µs
Pipelining stages S: 4
Processing latency:
$\boldsymbol{S \cdot T_{ts}}$ **= 400 µs**

Equal latency and throughtput

Scheduling performance: platform-independent

Pipelined Scheduling: Latency Control (I)



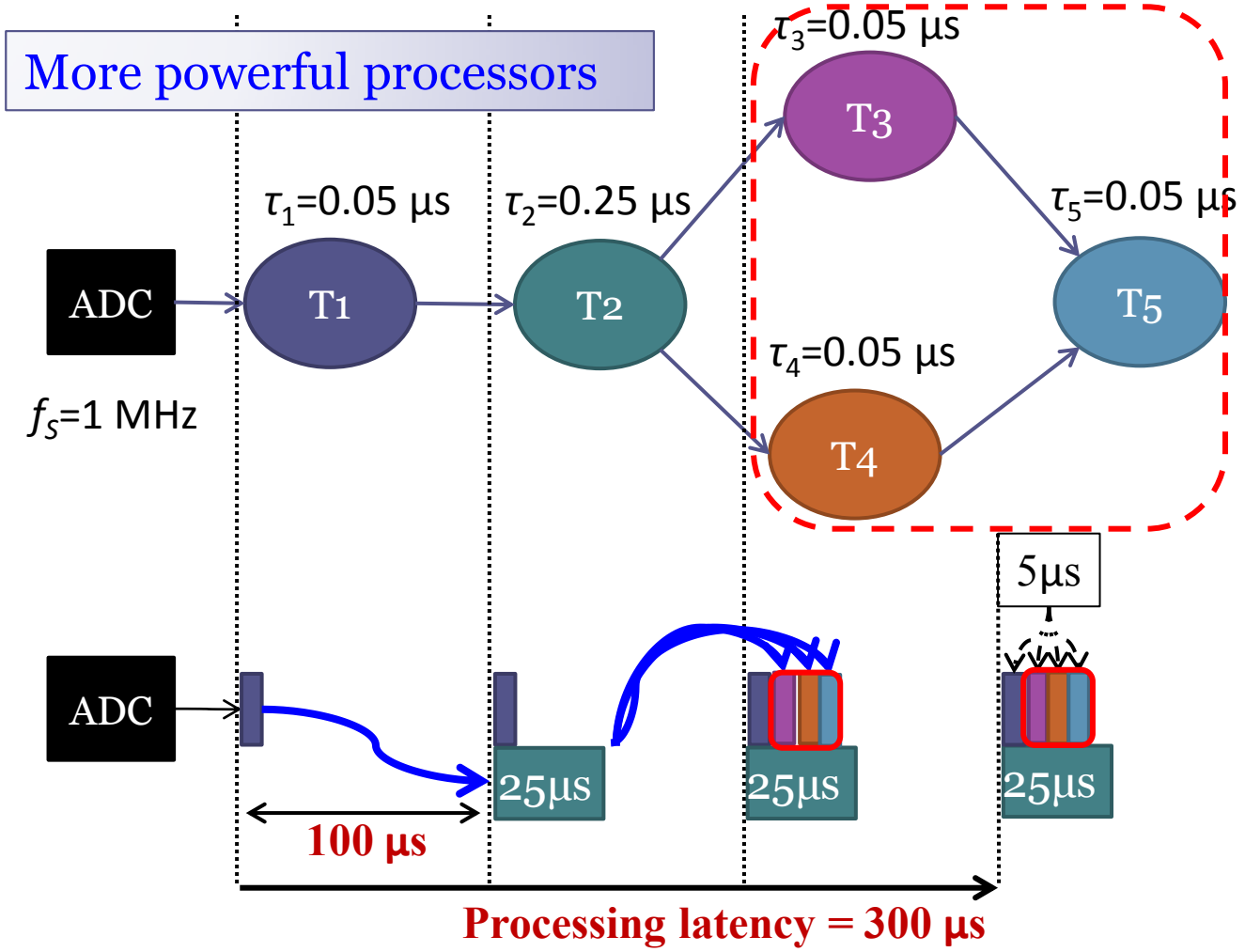
PIPELINE

Block size: **50** samples
Time slot: $50/f_S$ = **50 µs**
Pipelining stages S: 4
Processing latency:
$\boldsymbol{S \cdot T_{ts}}$ = **200 µs**

...

Scheduling performance: platform-independent

Pipelined Scheduling: Latency Control (II)



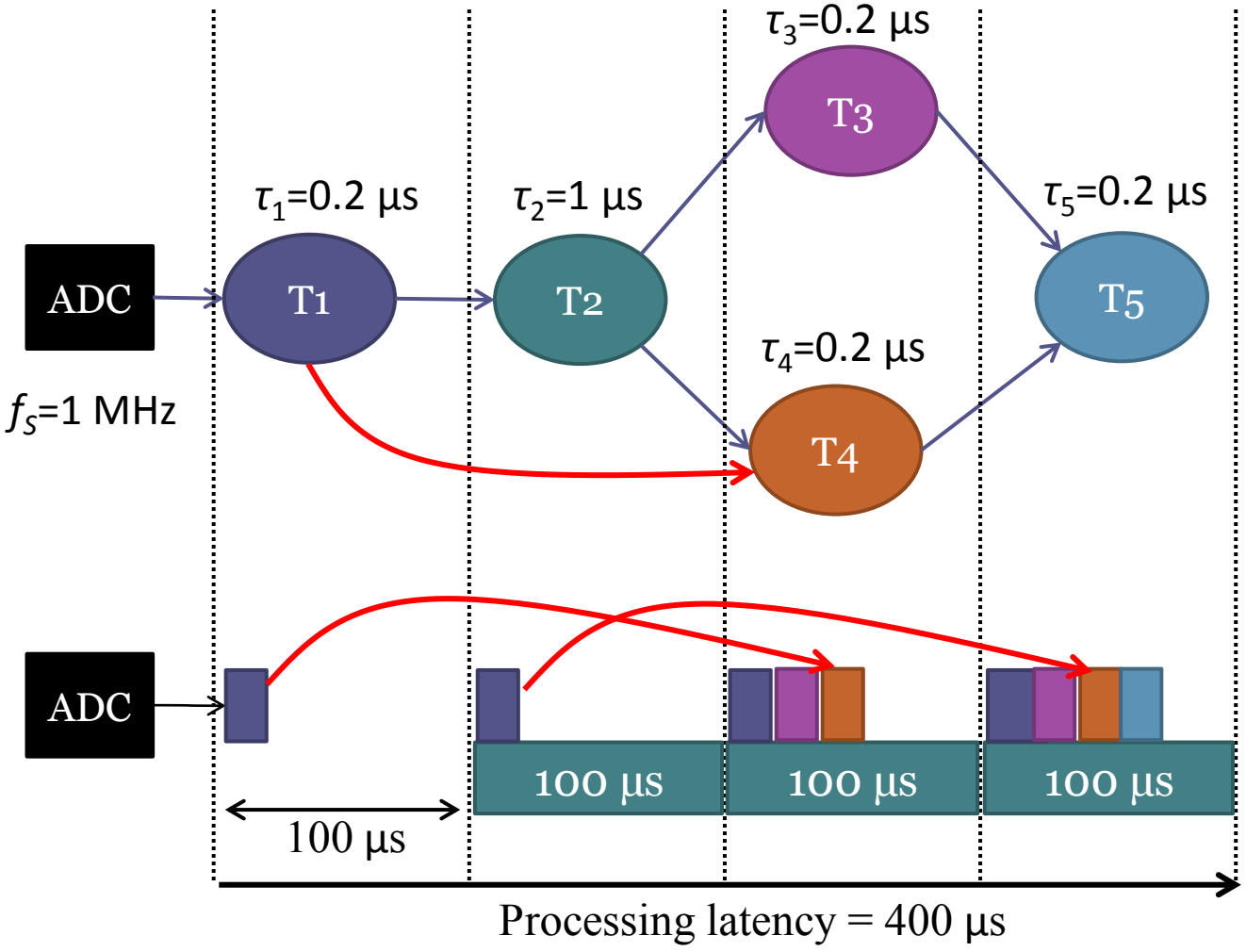
PIPELINE

Block size: **50** samples
Time slot: $50/f_S$ = **50 µs**
Pipelining stages S: 3
Processing latency:
$\boldsymbol{S \cdot T_{ts}}$ = **300 µs**

...

Scheduling performance: platform-independent

Pipelined Scheduling: Control Flow



PIPELINE

Block size: 100 samples
Period T: $1/f_S$ = 100 µs
Pipelining stages S: 4
Processing latency:
$\boldsymbol{S \cdot T}$ **= 400 µs**

...

Scheduling performance: platform-independent

Mapping

- Application and platform models
- Any mapping algorithm
- Two general-purpose algorithms:
  - t_w-mapping: $O(m \cdot n^{w+1})$
  - g_w-mapping: $O(m \cdot n^w)$
- Cost function

$$\text{Cost} = \frac{\text{processing requirement}}{\text{available processing power}} + \frac{\text{bandwidth requirement}}{\text{available bandwidth}}$$

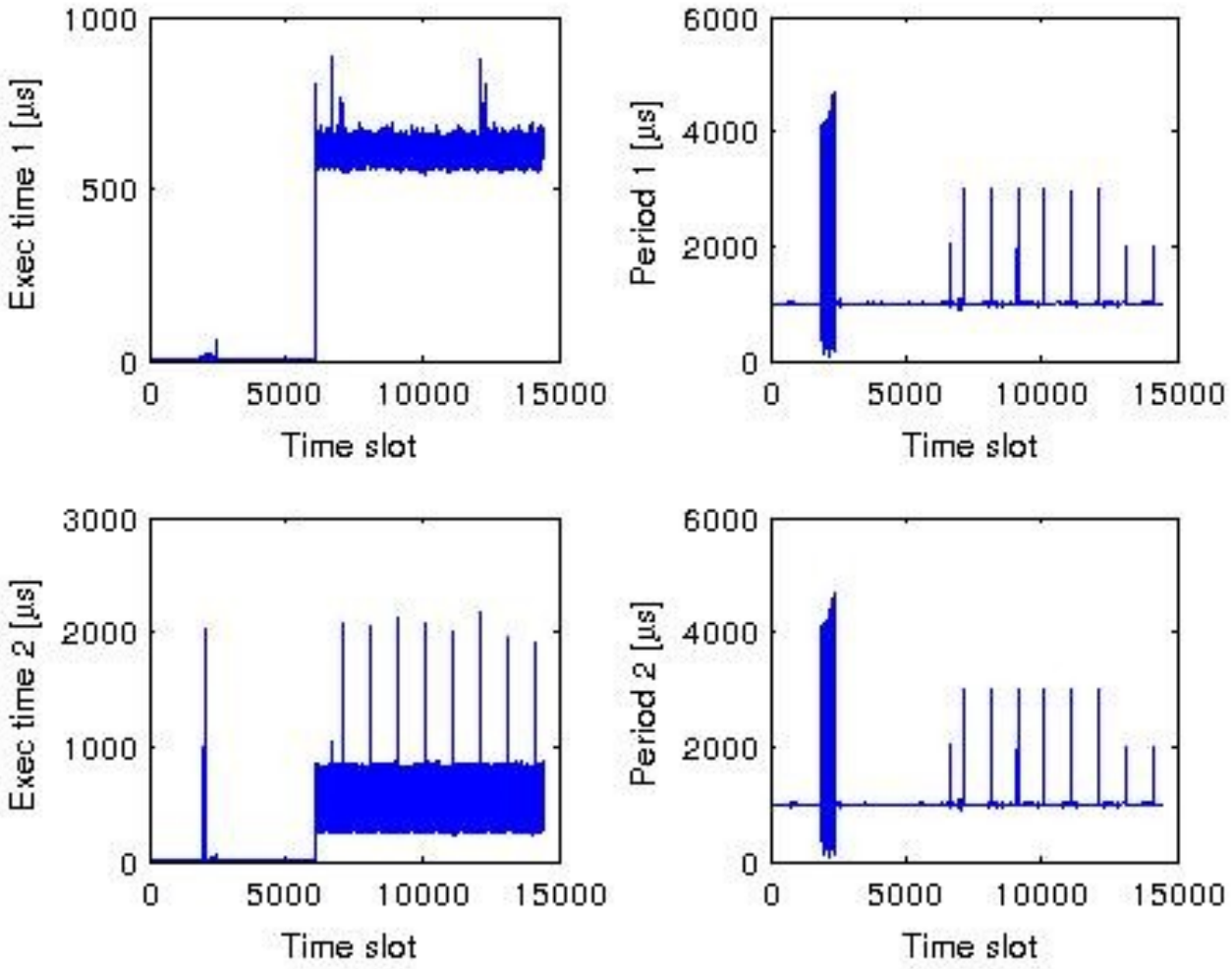
balance processing load *minimize data flows*

V. Marojevic, "Computing Resource Management in Software-Defined and Cognitive Radios," doctoral dissertation, Dept. Signal Theory and Communications, UPC, 2009.

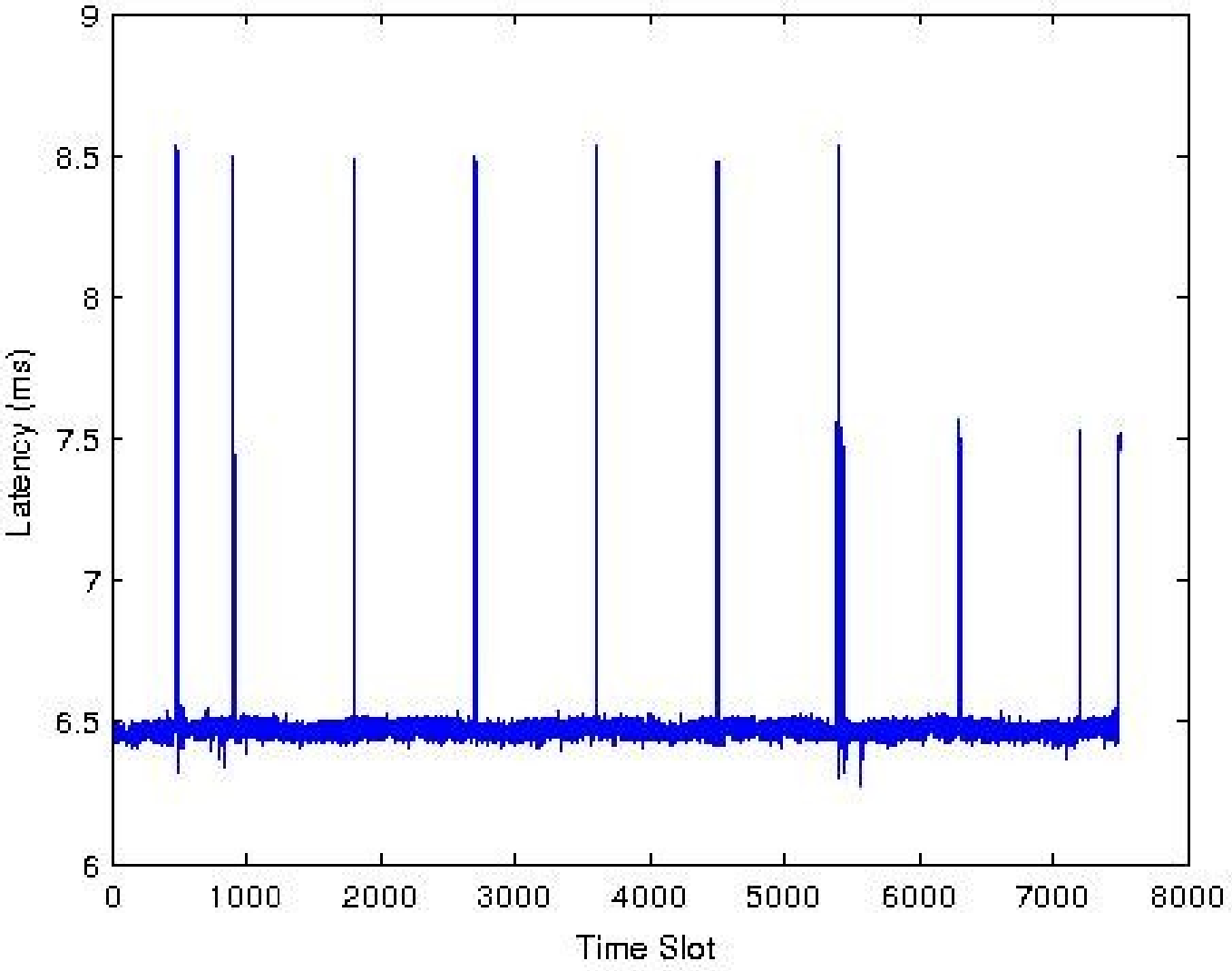
DEMO 1: Computing Resource Management

- LTE 1.4 MHz
- Time slot: 1 ms
- Sampling frequency: 1.92 MHz
- Each time slot, 1920 complex samples are sent to/ received from the USRP
- Receiver has 7 pipeline stages: 7 ms latency

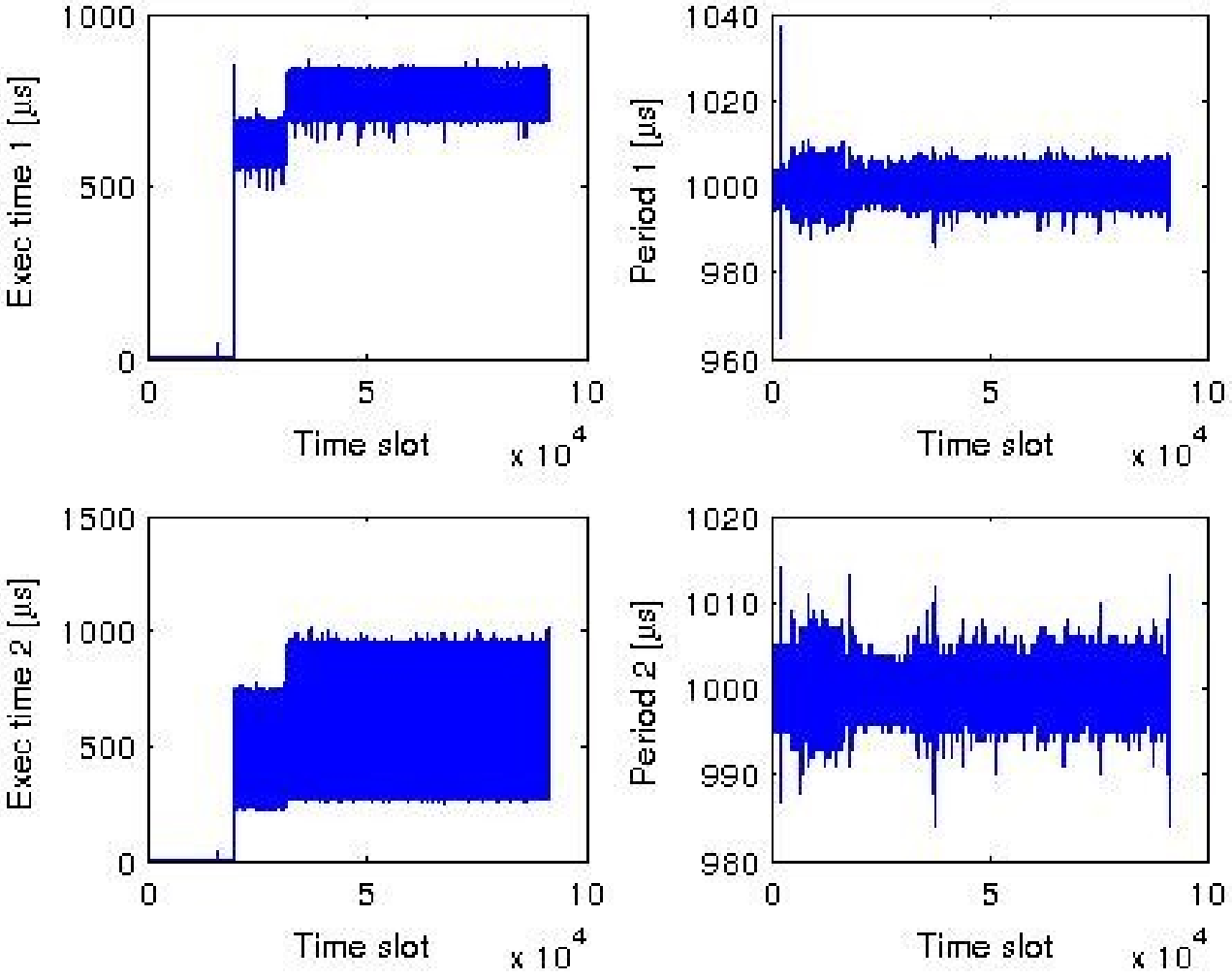
Soft Real-Time: Execution Trace



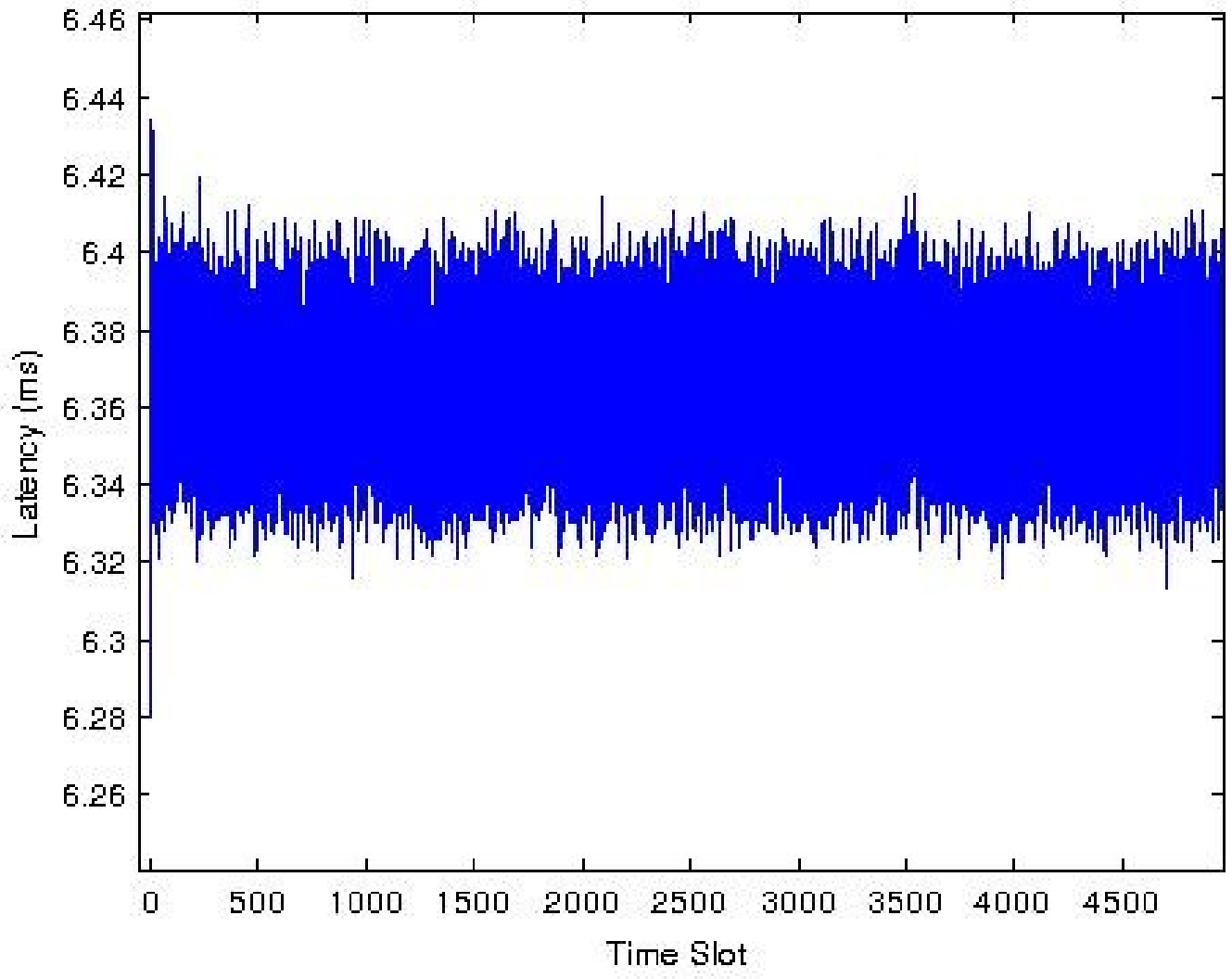
Soft Real-Time: Execution Trace



Hard Real-Time: Execution Trace

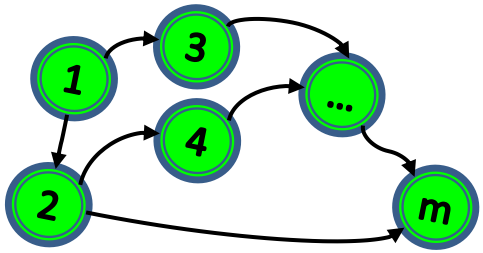


Hard Real-Time: Latency



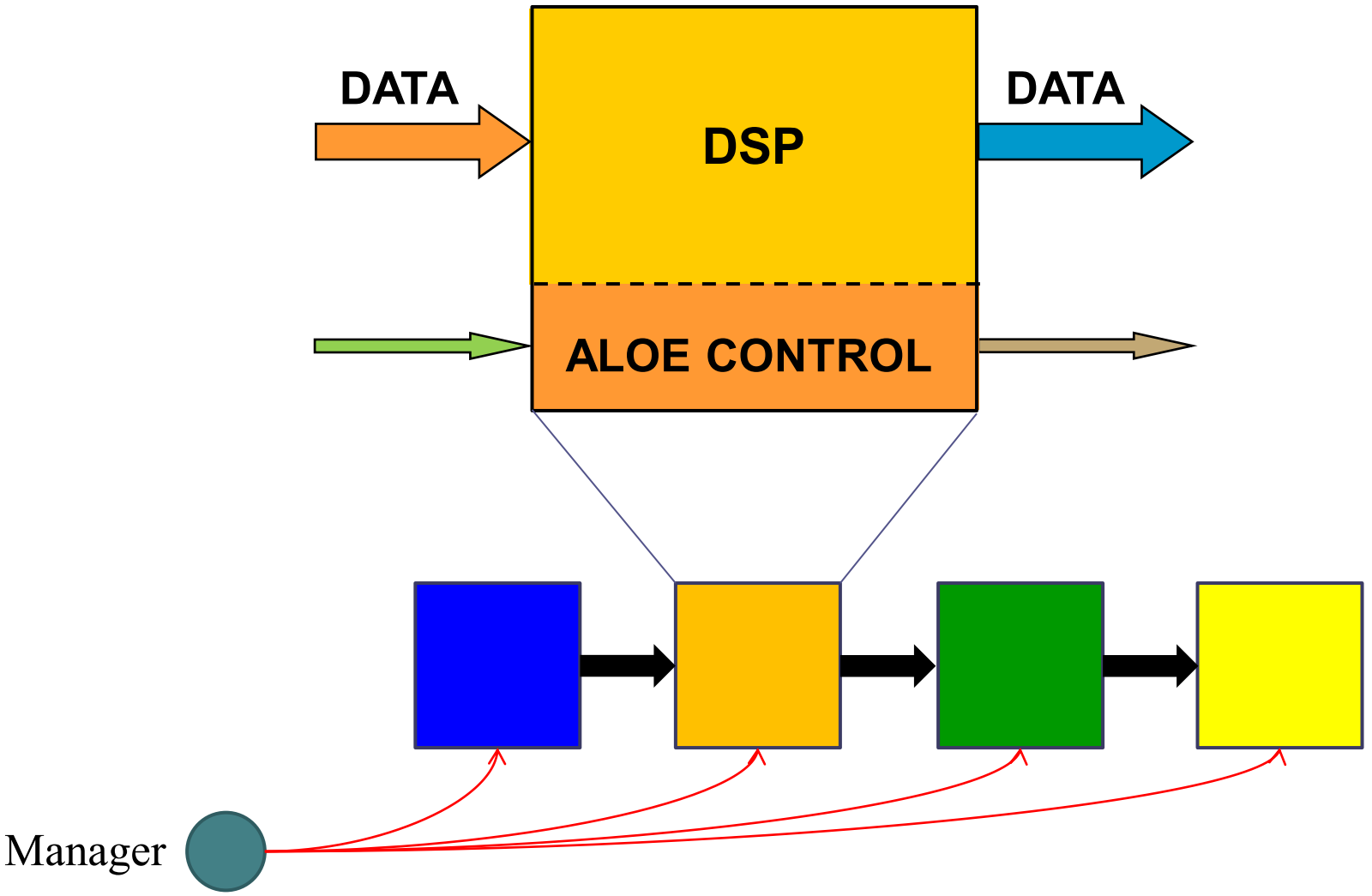
Waveform Design and Deployment

ALOE Waveform

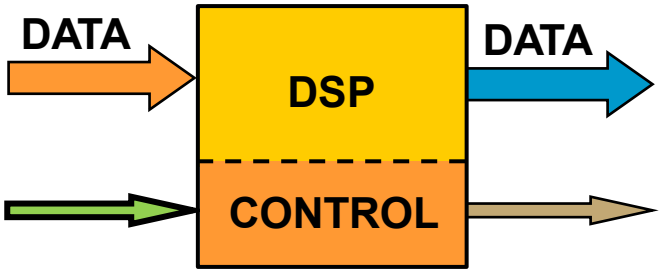


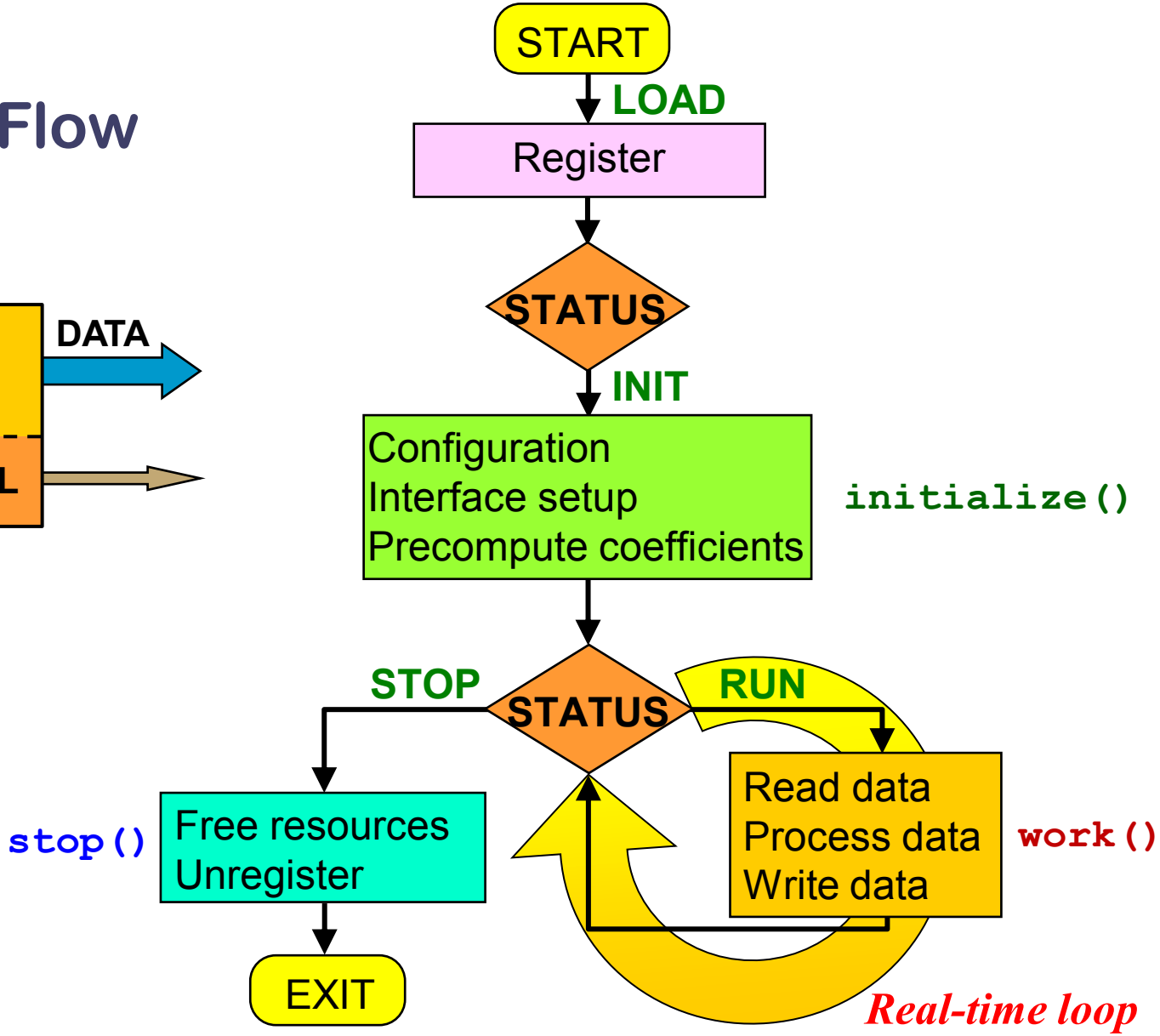
- **ALOE waveform**
  - Processing modules
  - Connections
  - Parameters
- **Module**
  - Computing requirements
  - Configuration parameters

Waveform Module



Module Execution Flow





Waveform Design and Deployment

Development

Implementation of DSP algorithms

CRC

Code Blk Concat.

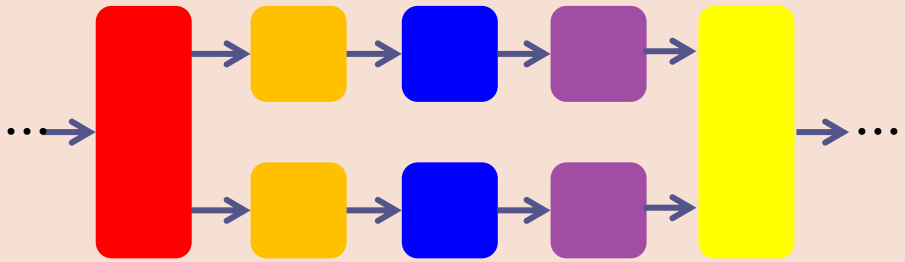
Turbo Coder

Code Blk Segm.

Rate Matching

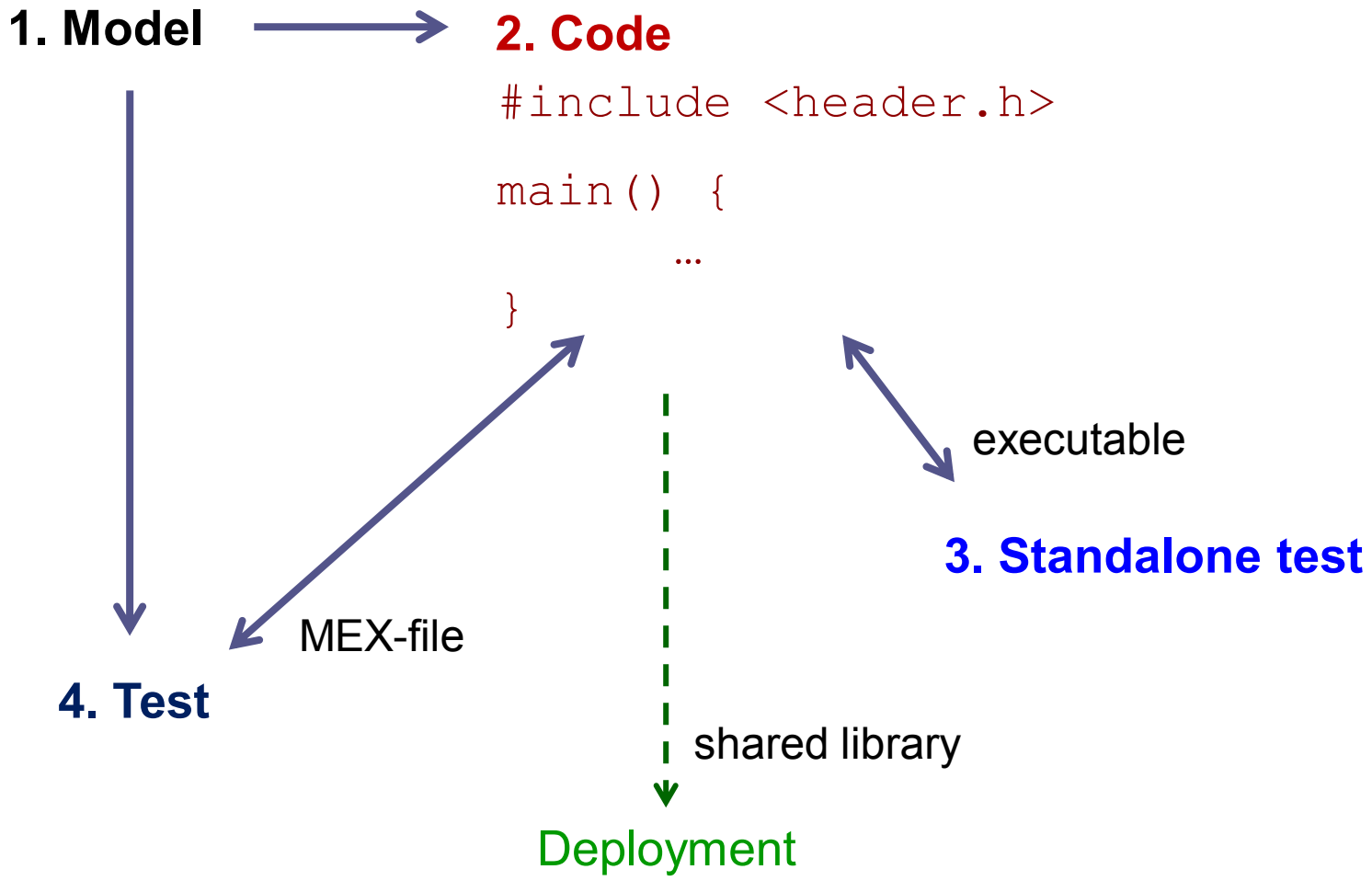
Deployment

Waveform creation and execution



- Parameters
- Execution time slot
- Pipelining stages
- …

Module Development



Module Template

initialize()

```
int initialize() {
        ...
        param_get_int_name("dft_len",&dft_len);
        dft_plan = compute_dft_plan(dft_len);
        return 0;
}
```

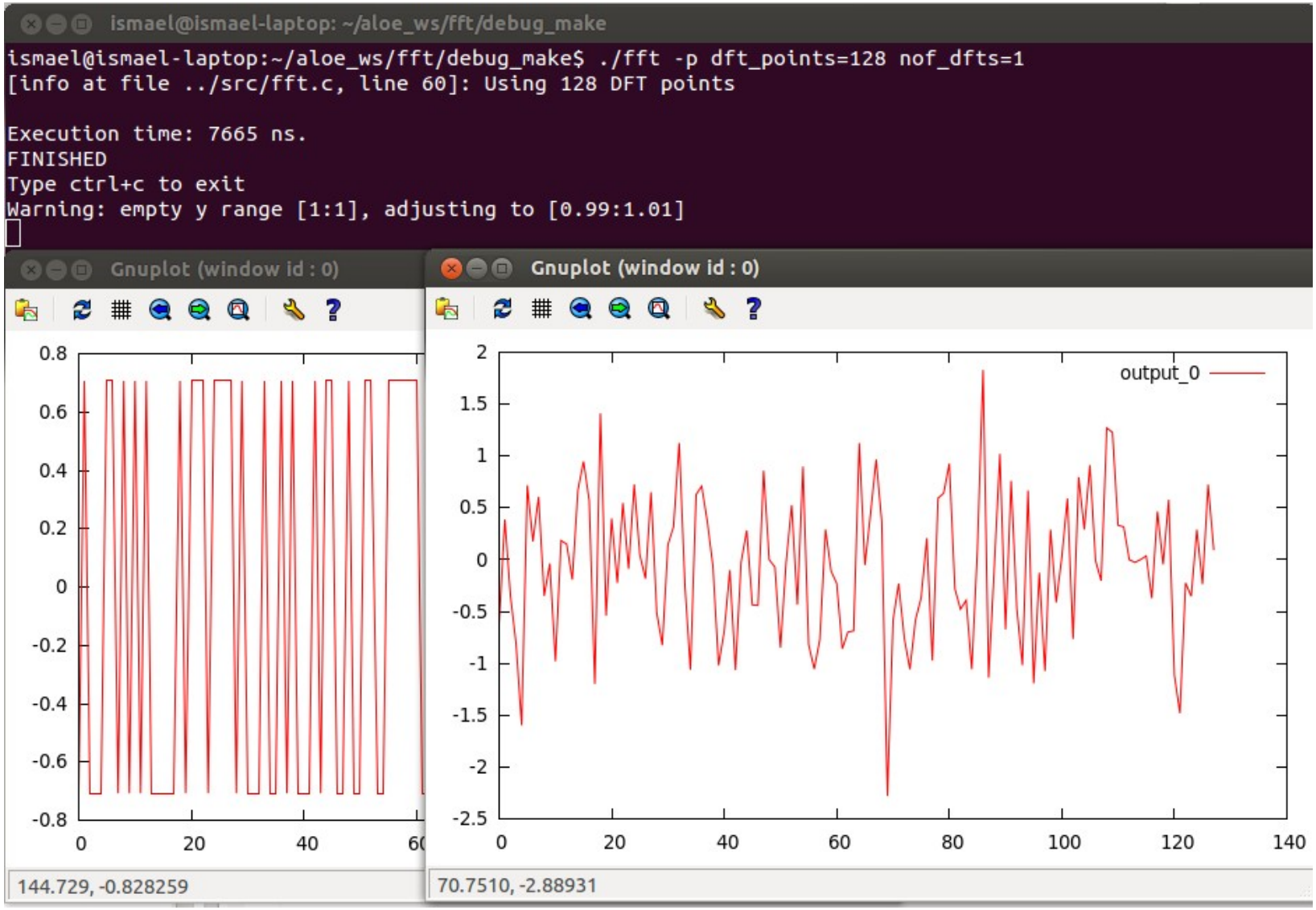
work()

```
int work(void **inp, void **out) {
        complex_t *input = inp[0];
        complex_t *output = out[0];
        int nsamples = get_input_samples(0);
        param_get_int_name("runtime_param",&my_param);
        run_dft(dft_plan, input, output);
        for (int i=0;i<nsamples;i++) {
                ...
        }
        return 0;
}
```

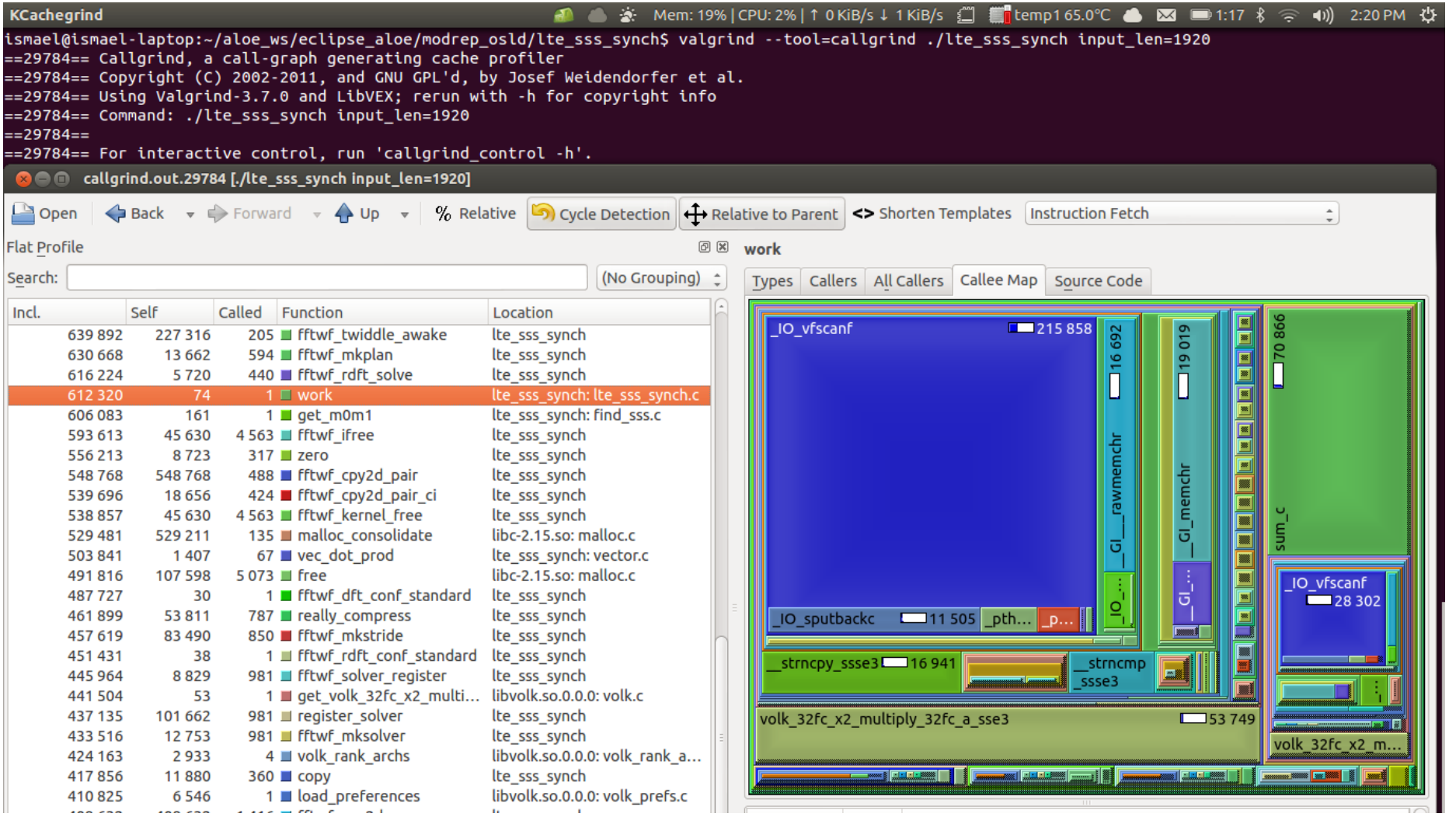
stop()

```
int stop() {
        destroy_dft_plan(dft_plan);
}
```

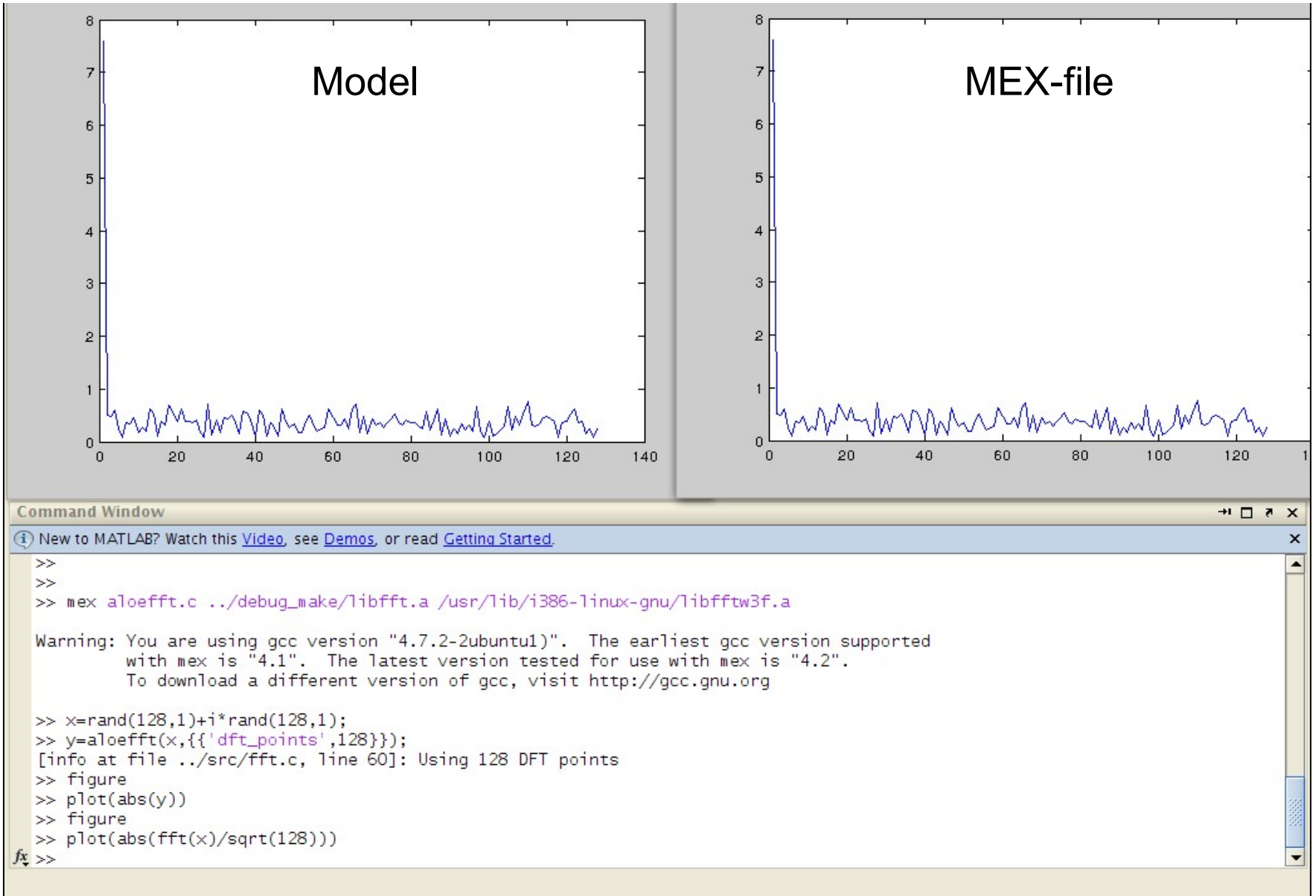
Standalone Execution: Debugging



Standalone Execution: Profiling (e.g. Valgrind)



Verification



DEMO 2: Module Development

Waveform Design and Deployment

Development

Implementation of DSP algorithms

CRC

Code Blk Concat.

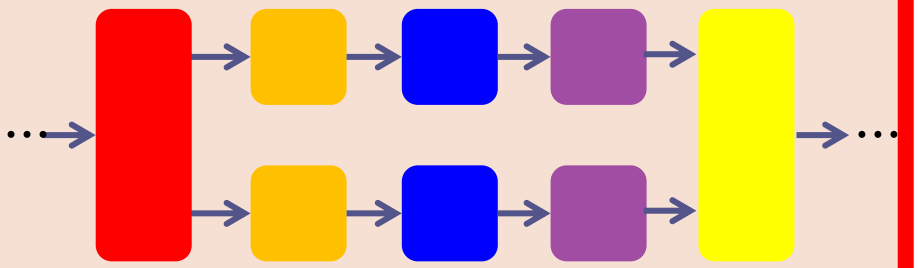
Turbo Coder

Code Blk Segm.

Rate Matching

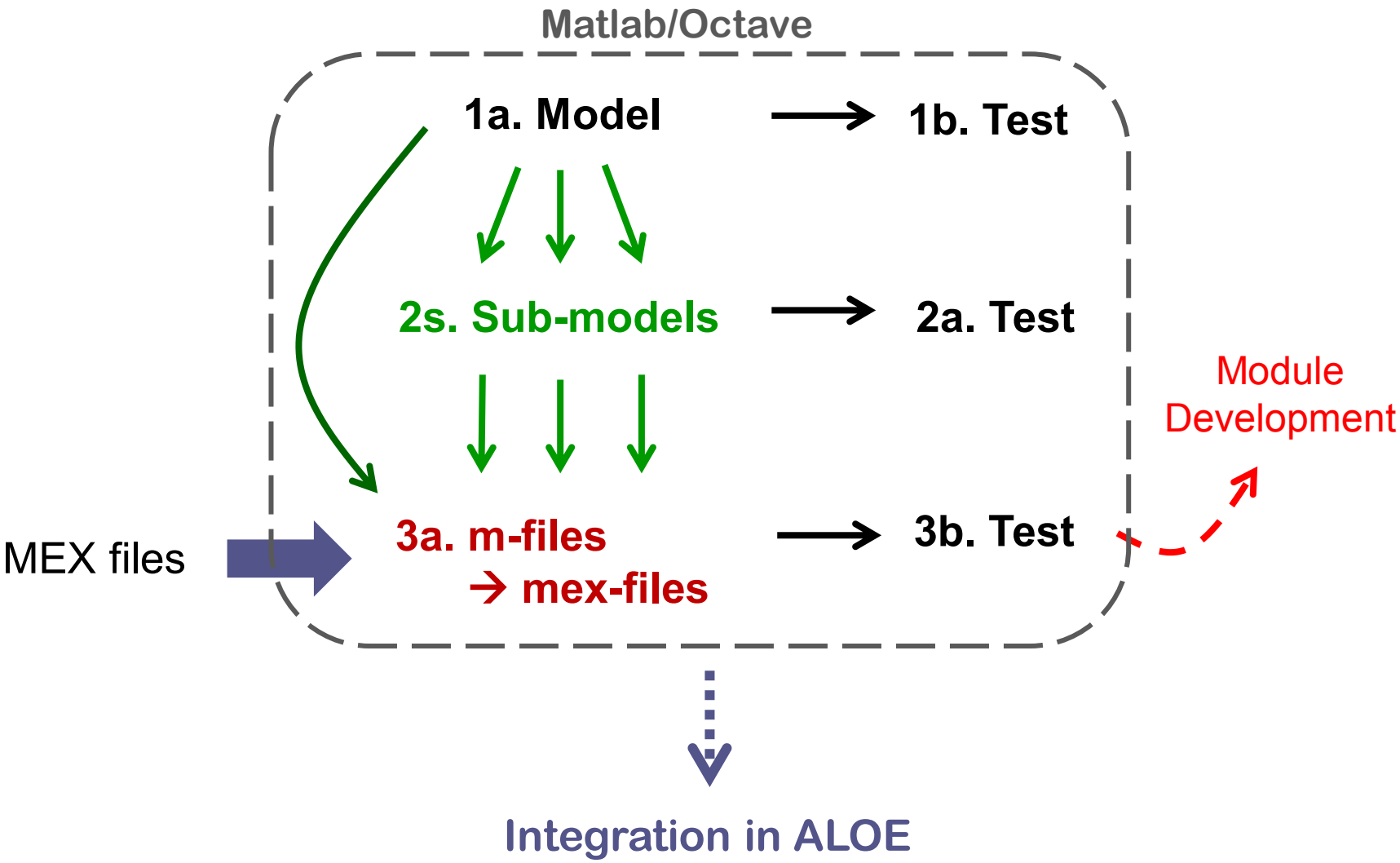
Deployment

Waveform creation and execution



- Parameters
- Execution time slot
- Pipelining stages
- …

Waveform Definition and Testing

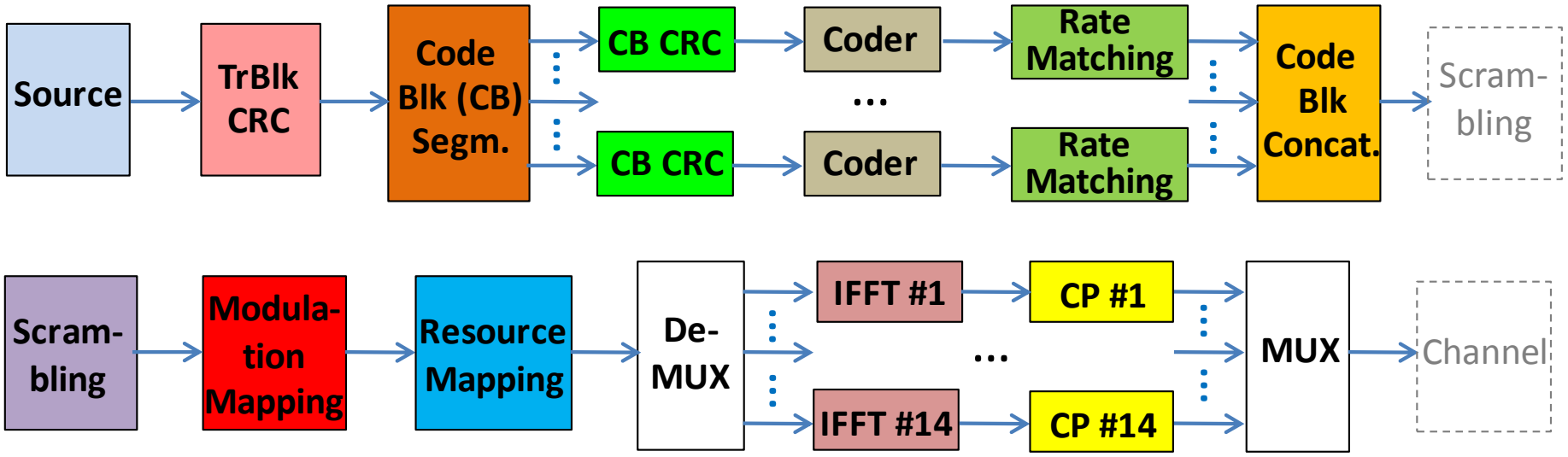


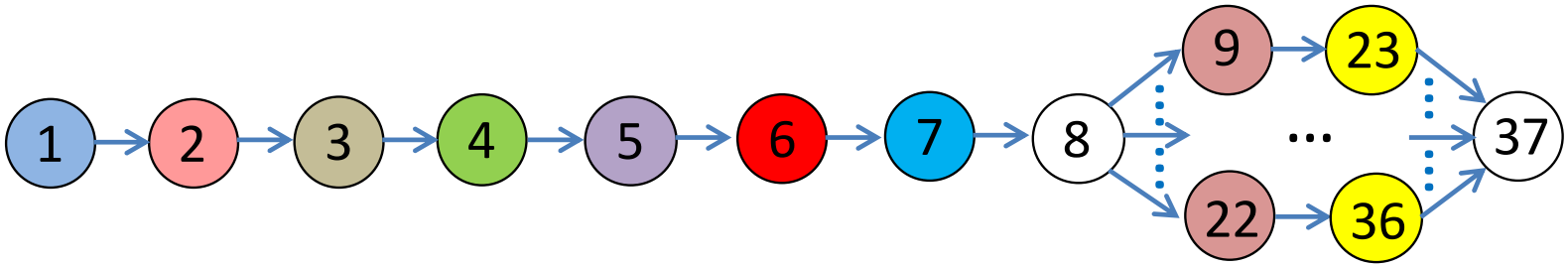
Application Description File (.app)

Field	Options	Description
modules	name	Unique ID
	binary	name and rel. location of the binary
	mopts	Processing requirements (for mapping)
	Variables	Configuration parameter values
interfaces	**{**src=<source>; dest=<destination>**}**, **{**...**}**, ...	Connection of modules: - *src*: output interface(s) of source module - *dest*: input interface(s) of destination module
join_stages	**(<**m1>,<m2>,...**)**, **(**...**)**	Executes modules listed in each pair of brackets in a single pipelining stage

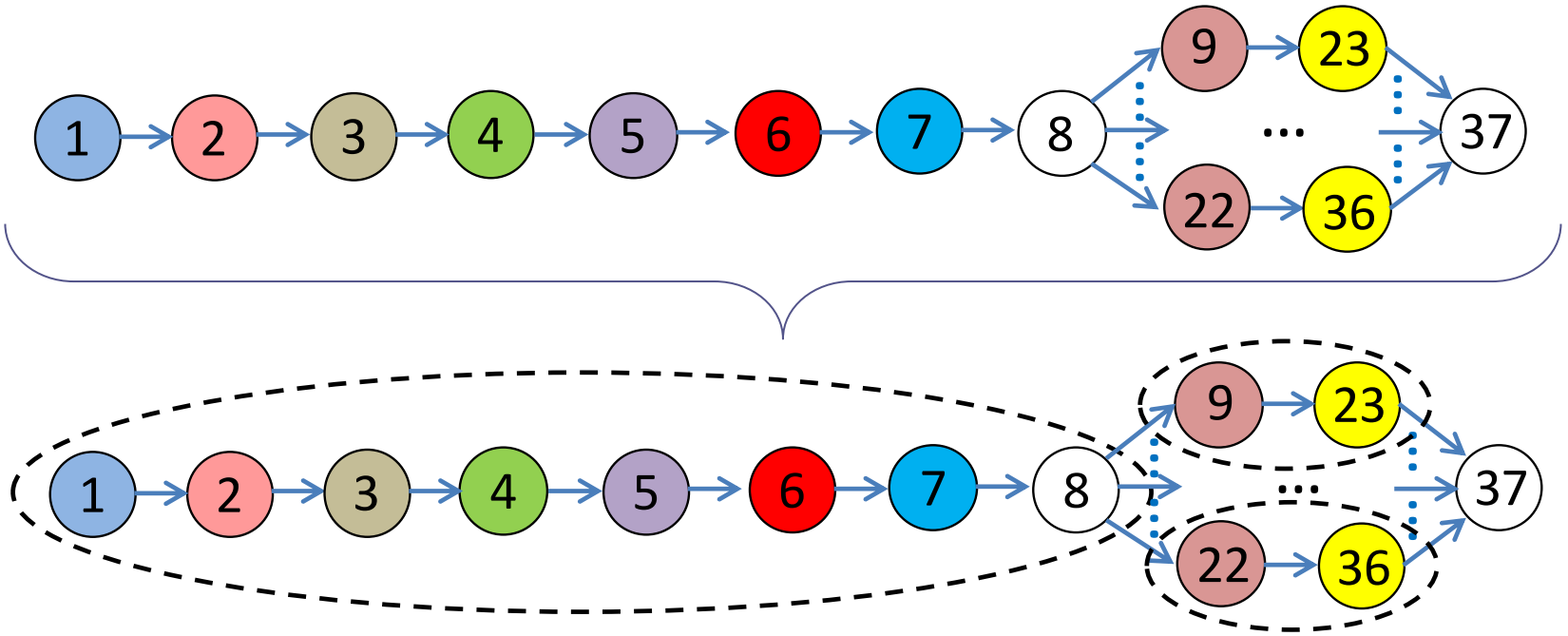
Pipelining Stages

PDSCH - Tx





Join Stages

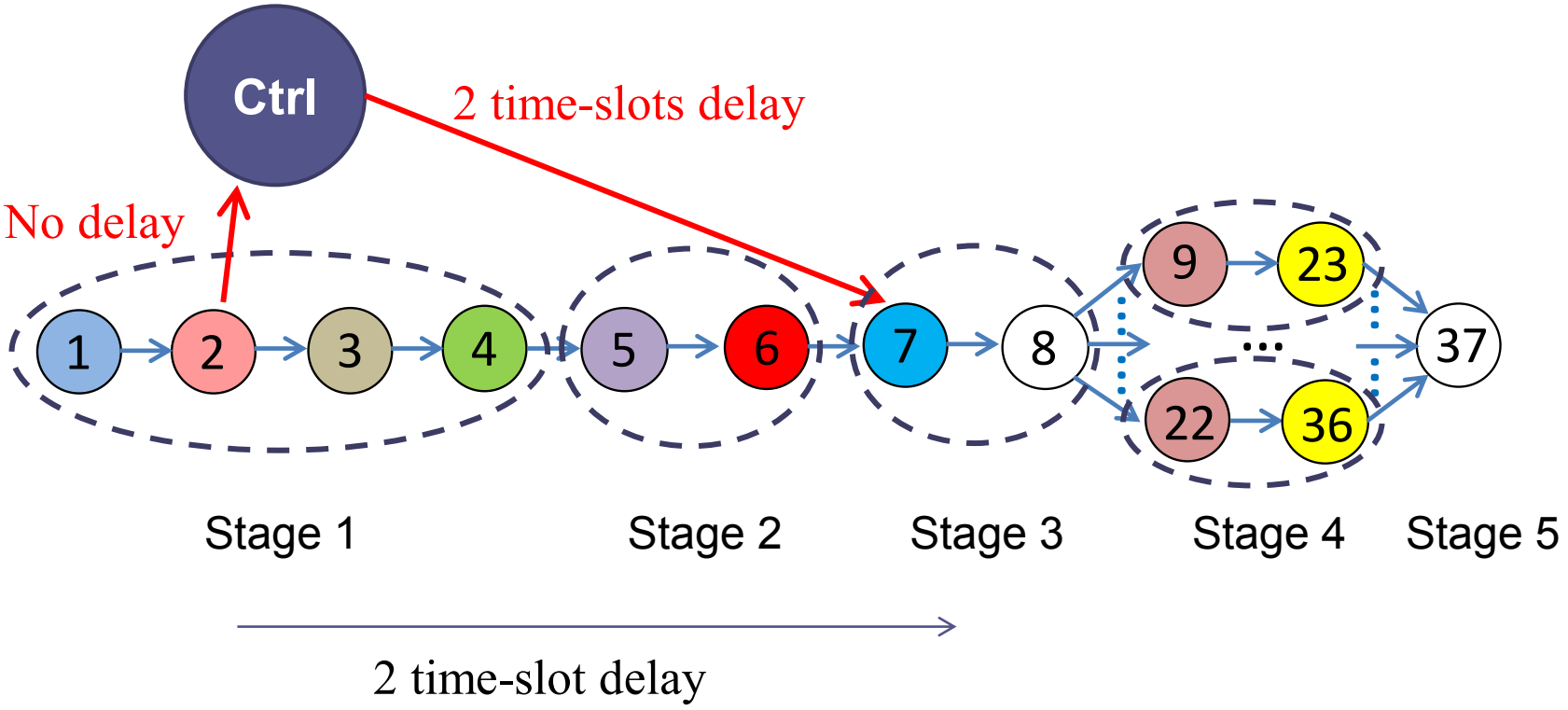



```
join stages=
(
        (M1, M2, M3, M4, M5, M6, M7, M8),
        (M9, M23),
        …
         (M22, M36)
);
```

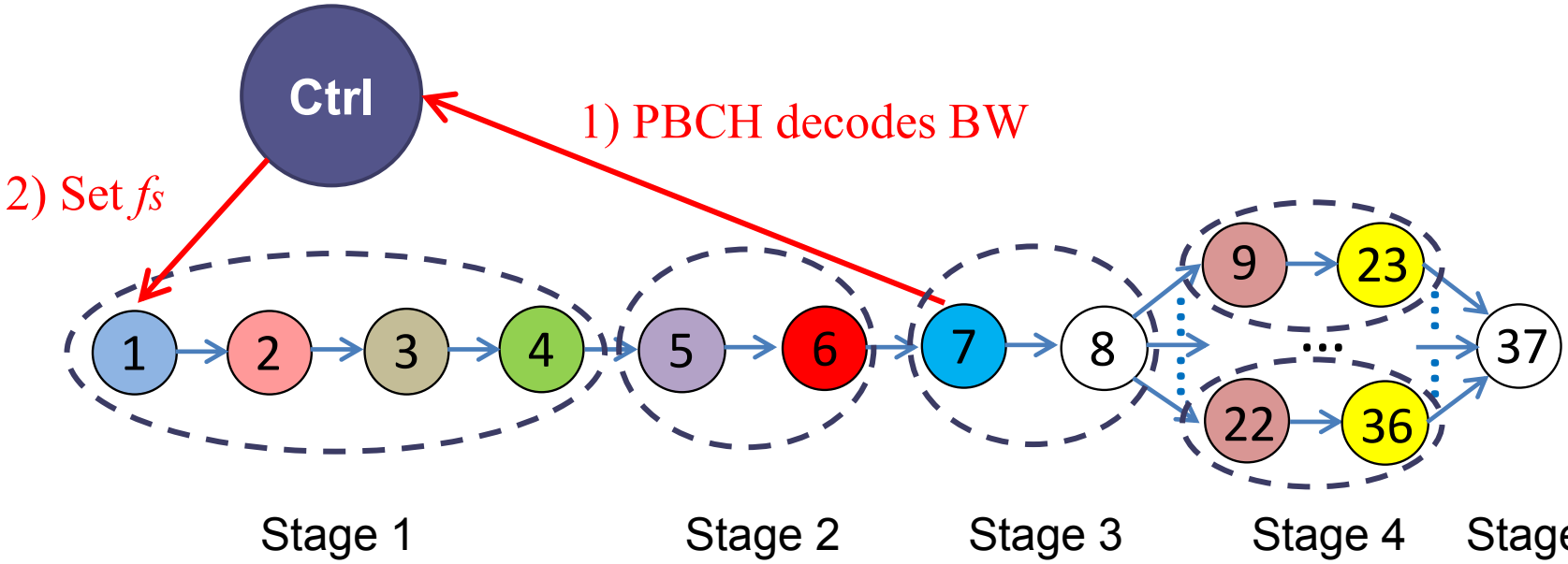
Waveform Deployment Tools

1. Waveform description file
   - Collection of sub-waveform description files
2. Decouple Tx-Rx
   - Tx writes to file, Rx reads from file
   - Modify file with Matlab (add channel noise/distortion)
3. Debug mode
   - Logging service
4. Real-time execution:
   - UHD support
   - Execution statistics

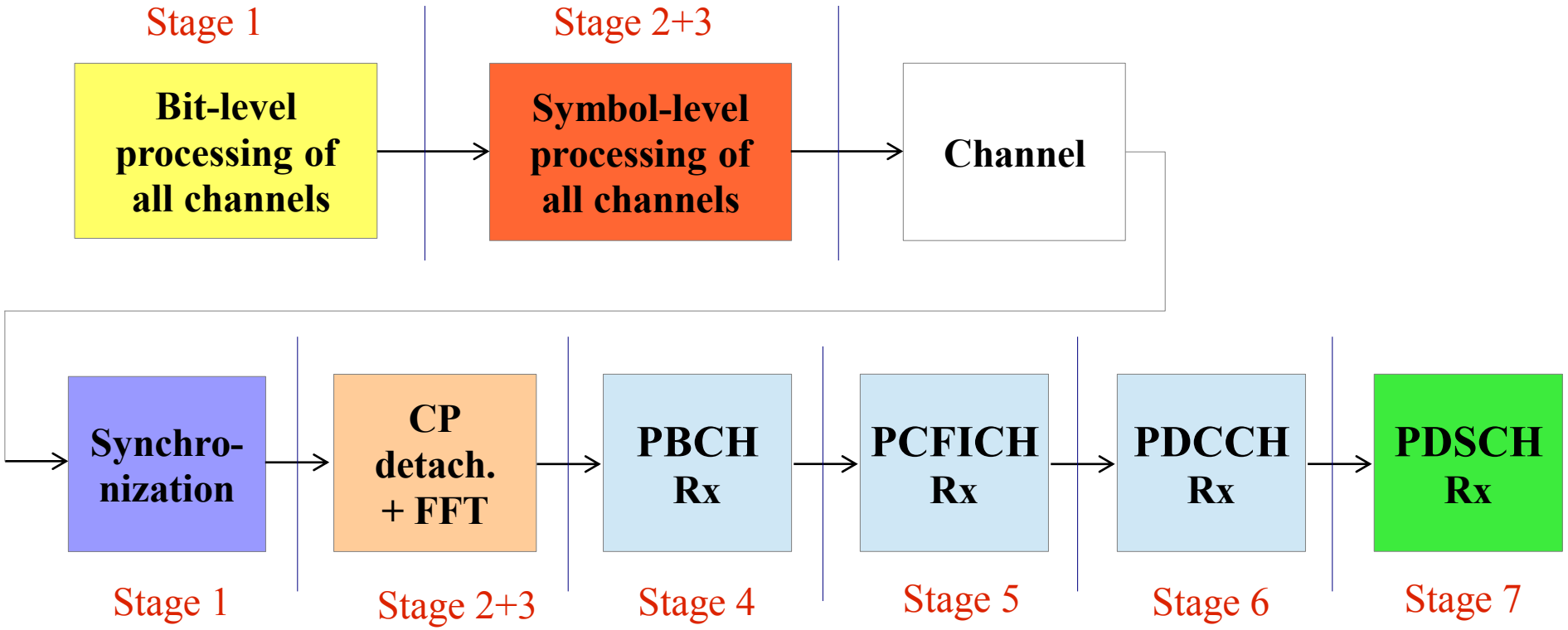
Control Module (I)

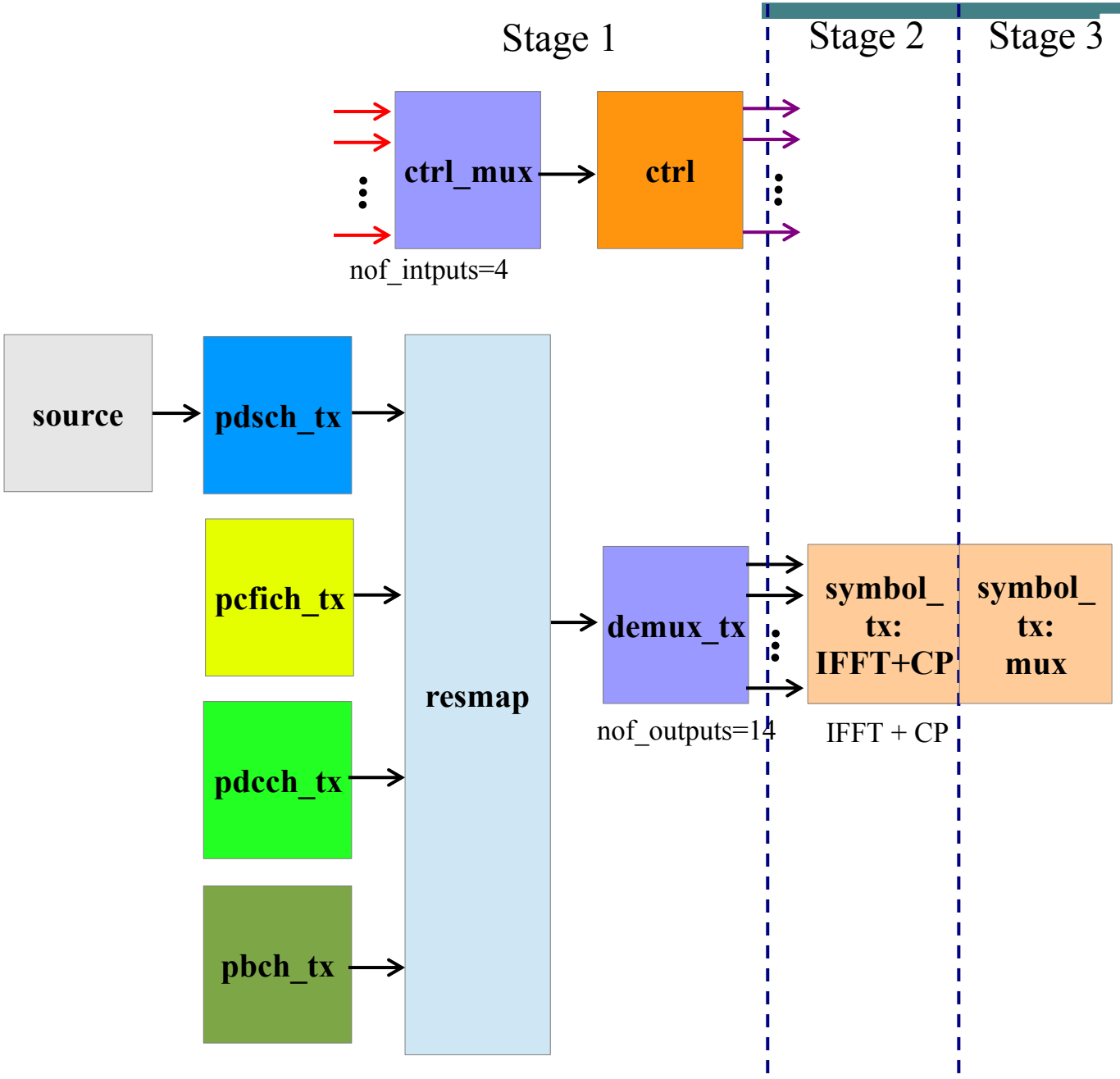


Control Module (II)

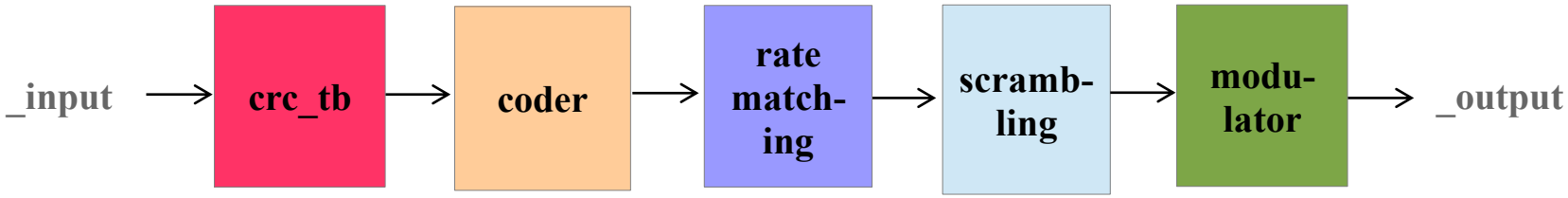


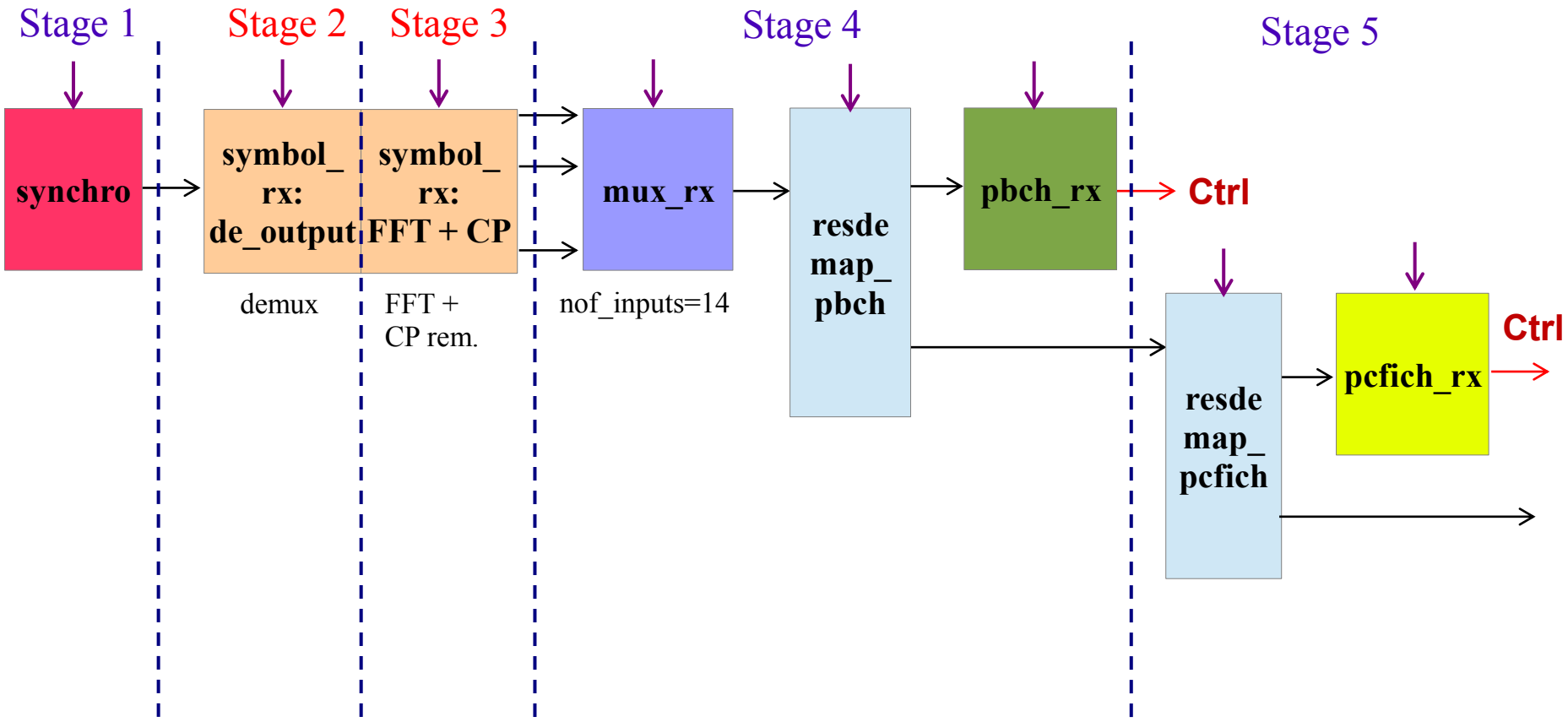
LTE DL Waveform

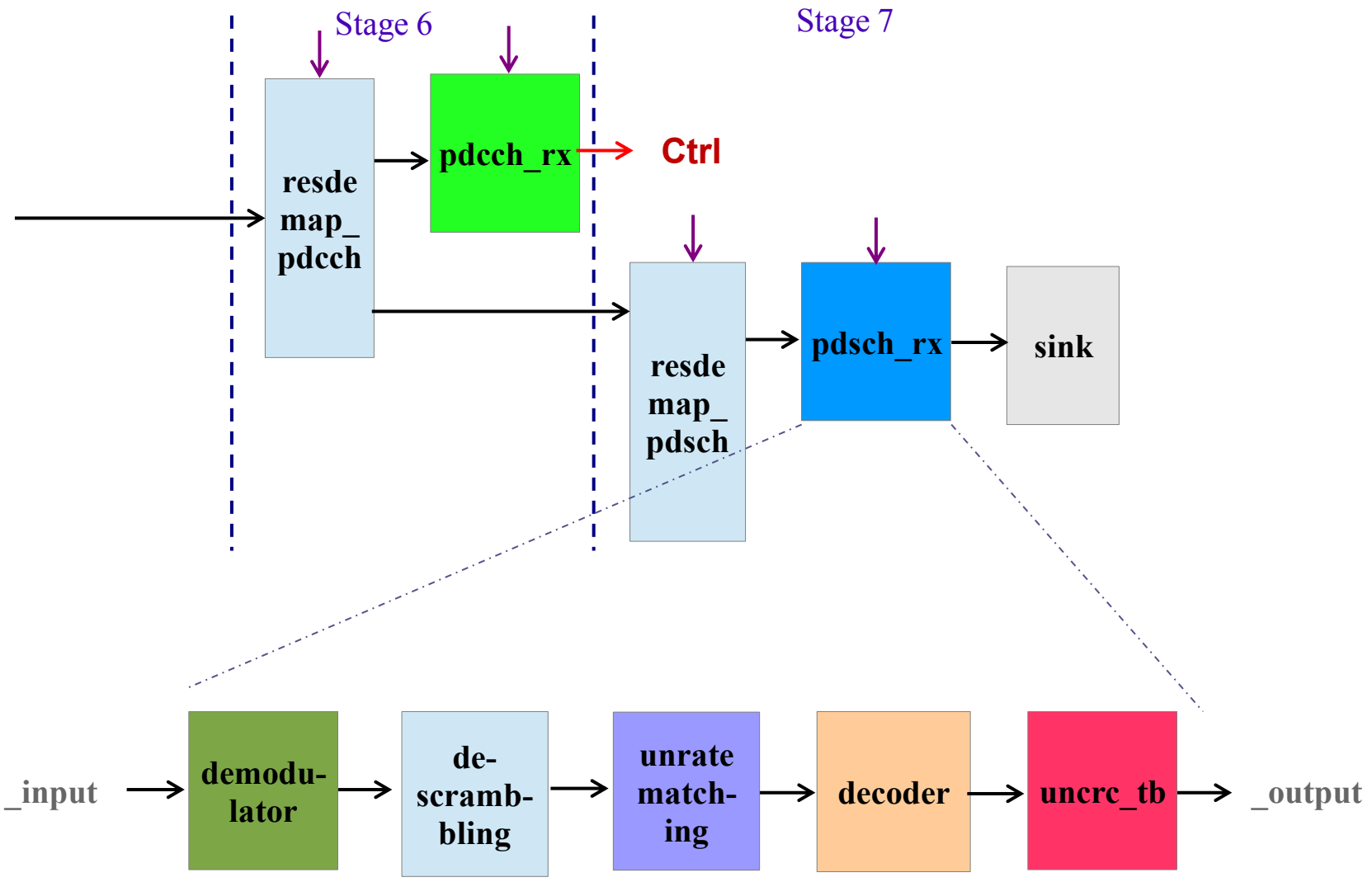



Stage 1
Stage 2
Stage 3
ctrl_mux
ctrl
nof_intputs=4
source
pdsch_tx
pcfich_tx
pdcch_tx
pbch_tx
resmap
demux_tx
nof_outputs=14
symbol_ tx: IFFT+CP
symbol_ tx: mux
IFFT + CP

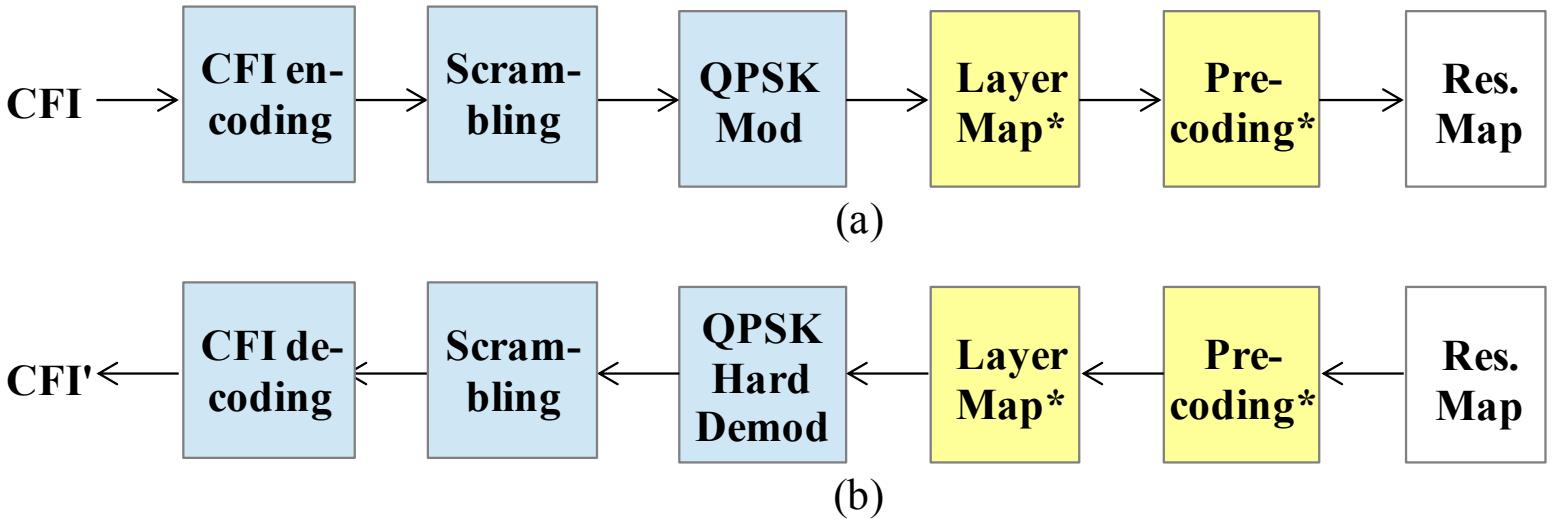
pdsch_tx


_input
crc_tb
coder
rate match-ing
scramb-ling
modu-lator
_output

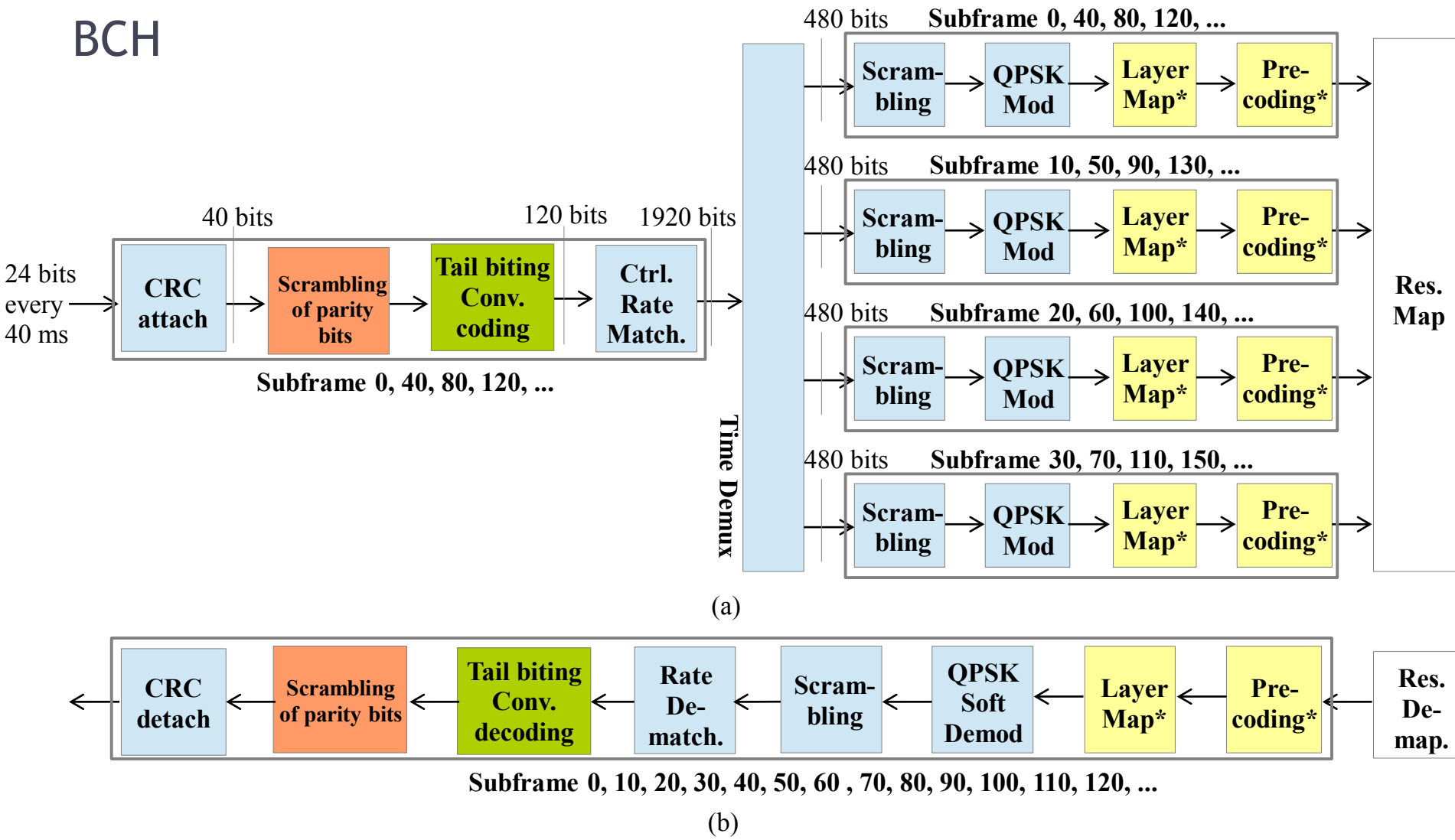

Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
synchro
symbol_ rx: de_output
symbol_ rx: FFT + CP
mux_rx
resde map_ pbch
pbch_rx
Ctrl
resde map_ pcfich
pcfich_rx
Ctrl
demux
FFT + CP rem.
nof_inputs=14


Stage 6
Stage 7
resde map_ pdcch
pdcch_rx
Ctrl
resde map_ pdsch
pdsch_rx
sink
_input
demodu- lator
de- scramb- bling
unrate match- ing
decoder
uncrc_tb
_output

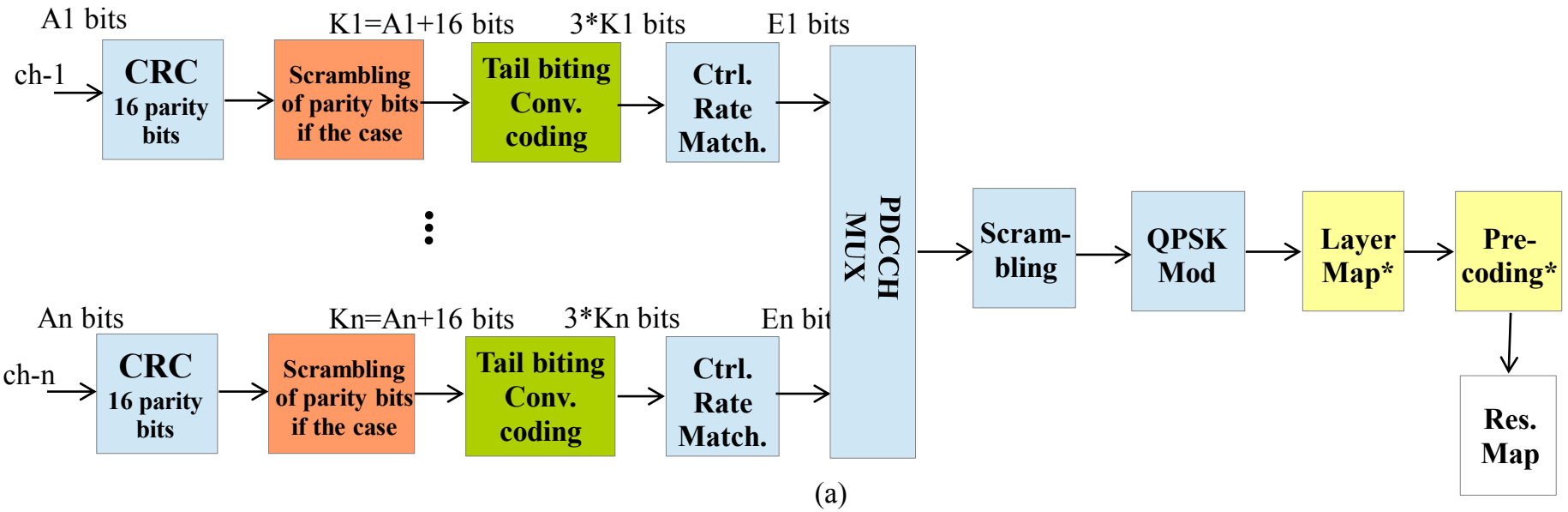
PCFICH


CFI
CFI en-coding
Scram-bling
QPSK Mod
Layer Map*
Pre-coding*
Res. Map
(a)
CFI'
CFI de-coding
Scram-bling
QPSK Hard Demod
Layer Map*
Pre-coding*
Res. Map
(b)

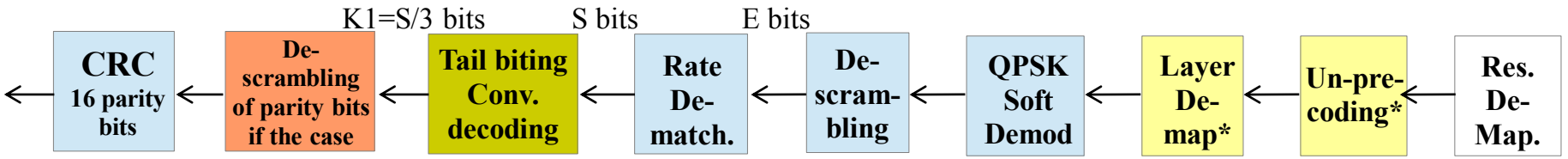
BCH


24 bits
every
40 ms
CRC attach
40 bits
Scrambling of parity bits
Tail biting Conv. coding
120 bits
Ctrl. Rate Match.
1920 bits
Subframe 0, 40, 80, 120, ...
Time Demux
480 bits
Subframe 0, 40, 80, 120, ...
Scram-bling
QPSK Mod
Layer Map*
Pre-coding*
480 bits
Subframe 10, 50, 90, 130, ...
480 bits
Subframe 20, 60, 100, 140, ...
480 bits
Subframe 30, 70, 110, 150, ...
Res. Map
(a)
CRC detach
Scrambling of parity bits
Tail biting Conv. decoding
Rate De-match.
Scram-bling
QPSK Soft Demod
Layer Map*
Pre-coding*
Res. De-map.
Subframe 0, 10, 20, 30, 40, 50, 60 , 70, 80, 90, 100, 110, 120, ...
(b)

PDCCH


A1 bits
ch-1
CRC
16 parity bits
K1=A1+16 bits
Scrambling of parity bits if the case
3*K1 bits
Tail biting Conv. coding
E1 bits
Ctrl. Rate Match.
An bits
ch-n
CRC
16 parity bits
Kn=An+16 bits
Scrambling of parity bits if the case
3*Kn bits
Tail biting Conv. coding
En bit
Ctrl. Rate Match.
PDCCH MUX
Scram-bling
QPSK Mod
Layer Map*
Pre-coding*
Res. Map

(a)


CRC
16 parity bits
De-scrambling of parity bits if the case
K1=S/3 bits
Tail biting Conv. decoding
S bits
Rate De-match.
E bits
De-scram-bling
QPSK Soft Demod
Layer De-map*
Un-pre-coding*
Res. De-Map.

(b)

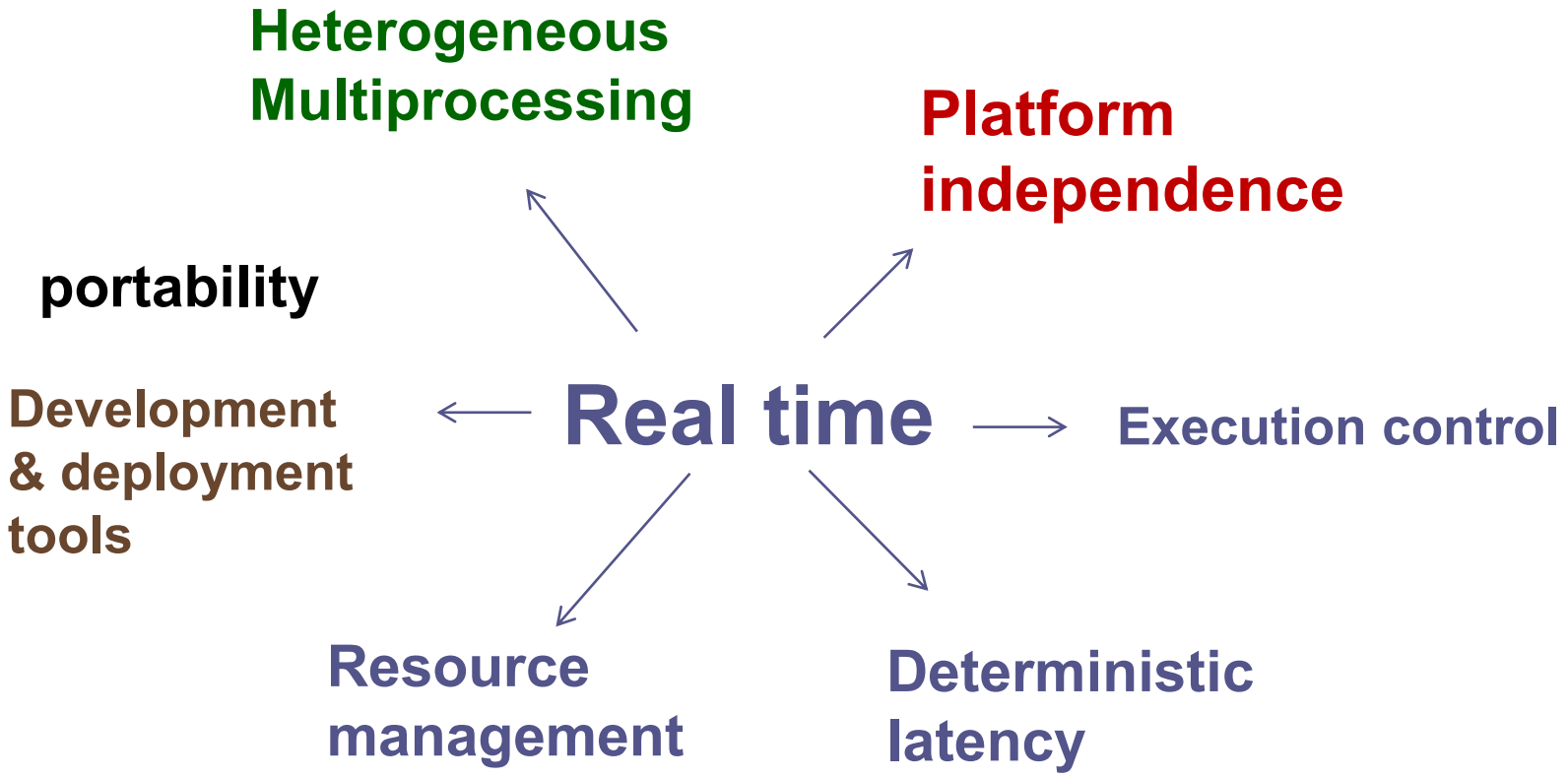
DEMO 3: Waveform Deployment

Conclusions

SDR Frameworks

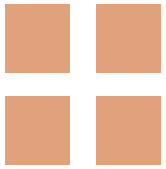
- **SCA (Software Communication Architecture)**
  - Military
  - Research and education (e.g. OSSIE)
- **GNU Radio**
  - Research and education (PC, multicore)
- **…**

ALOE Characteristics

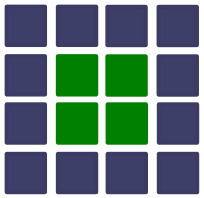


Conclusions

multicores



many-cores



small clusters
(heterogeneous)

Future Work

- GNU Radio/ SCA compatibility
- Add and test new schedulers:
  - Dynamic, provided by the RTOS
  - Hybrid (static-dynamic)
- Tools
  - Waveform development and deployment
  - Graphical User Interface for ALOE++
  - …

Call for Participation

- **FlexNets** (Flexible Wireless Communications Systems & Networks)
  - http://flexnets.upc.edu/trac/
    - ALOE releases
    - Computing resource management framework
    - Waveforms
    - Educational material
- **OSLD** (Open source LTE deployment)
  - https://sites.google.com/site/osldproject/home
  - https://github.com/flexnets
    - ALOE++
    - DSP modules library
    - Development Tools
- **Mailing lists:** https://groups.google.com/group/flexnets