



MINISTÈRE
DES ARMÉES
*Liberté
Égalité
Fraternité*



The AFF3CT framework for building digital communication chains

Olivier Aumage

Inria – LaBRI, Bordeaux, France

FOSDEM 2025



1.

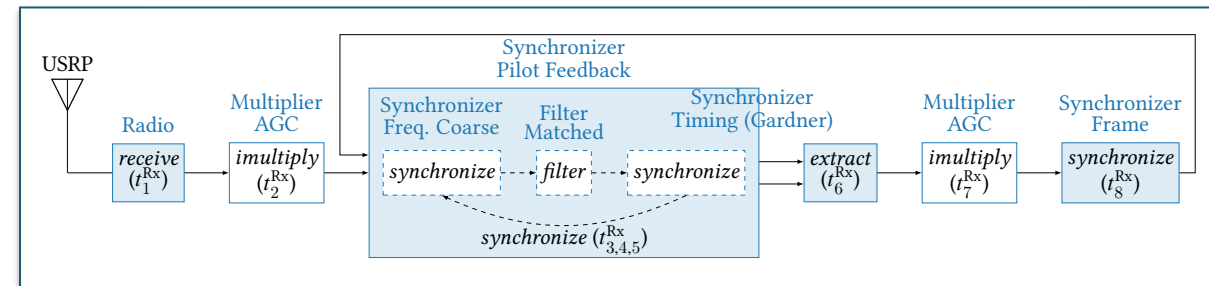
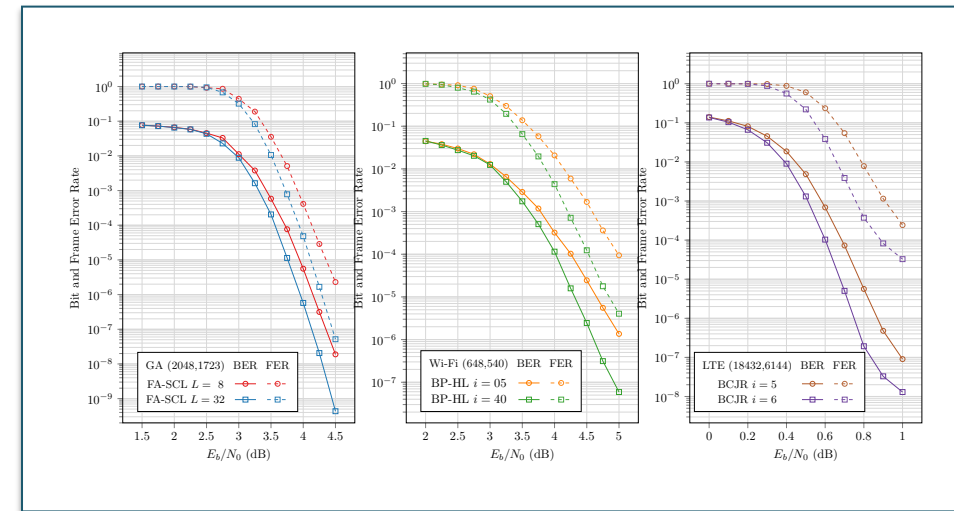
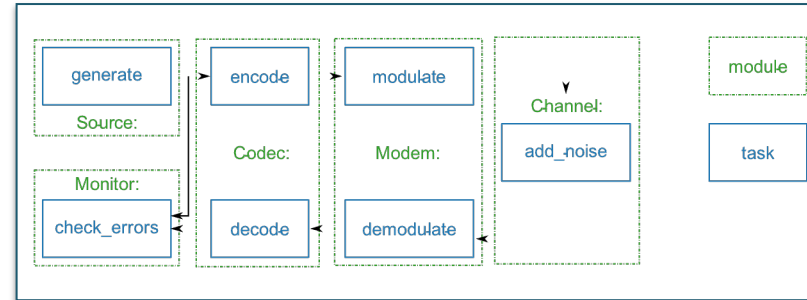
Context and People Involved

AFF3CT

Modular Digital Communication

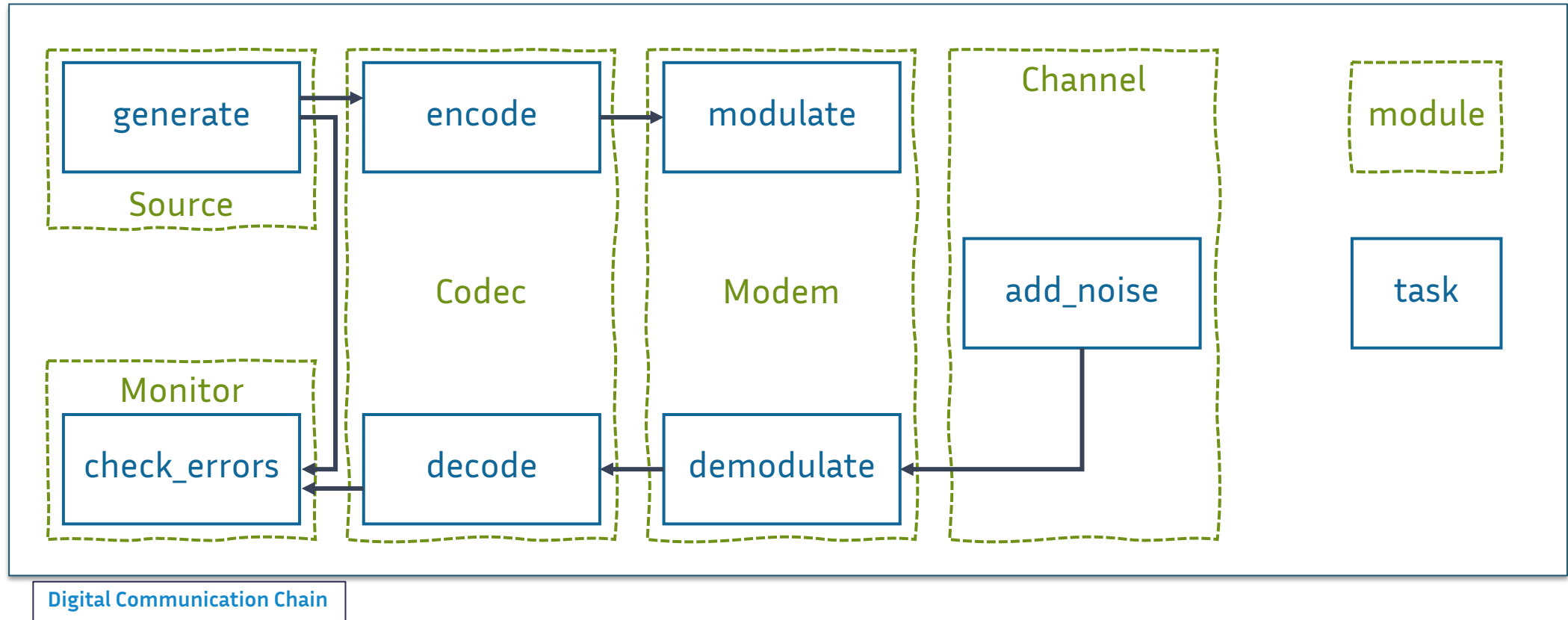
- **Software Defined Radio (SDR)**
 - Digital communication chains
 - (5G phones, Wi-Fi, DVB-RCS, etc.)
 - Codec algorithmics
 - ECC, modulation, channel models
 - Design, validation, production
 - Transition from ASICs to software
- **10-Year+ Project**
 - Started around 2013

<https://aff3ct.github.io/>



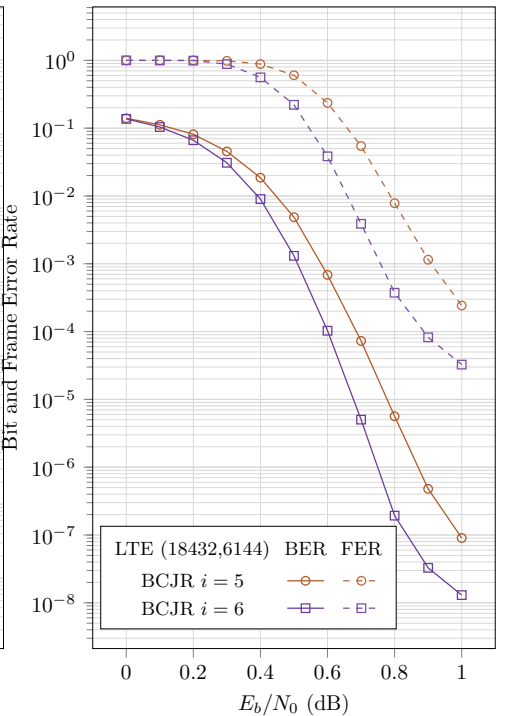
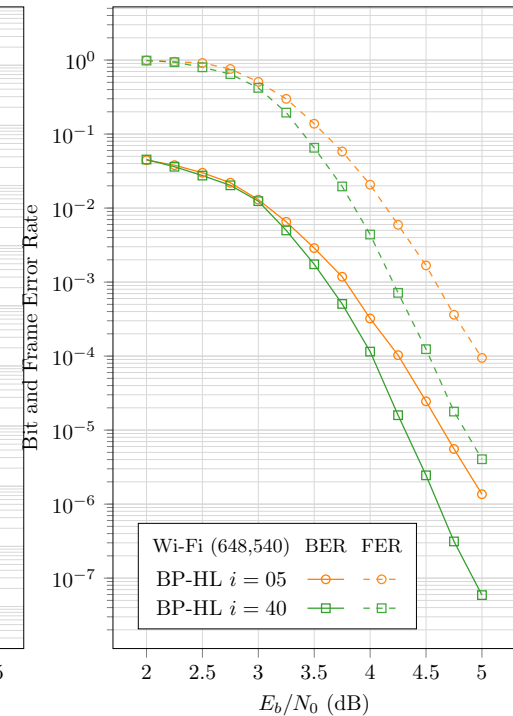
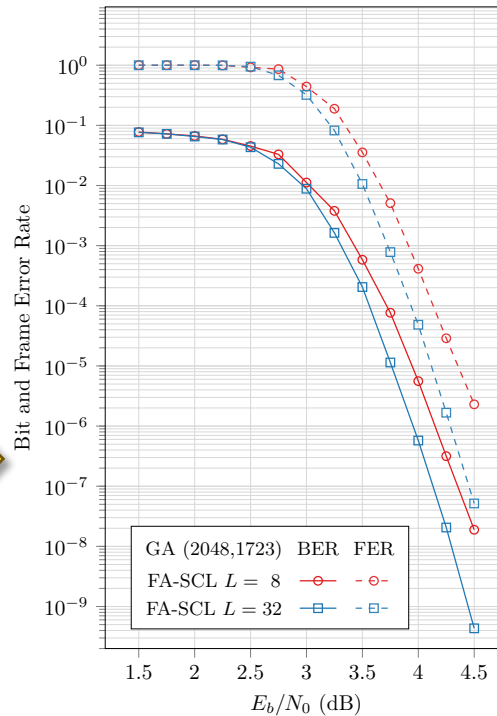
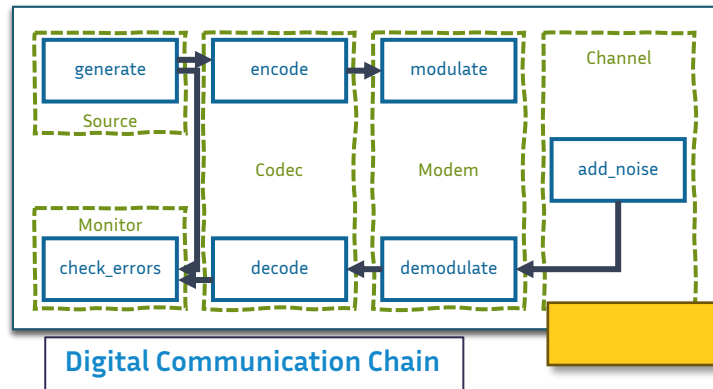
AFF3CT

Digital Communication Programming Environment for SDR



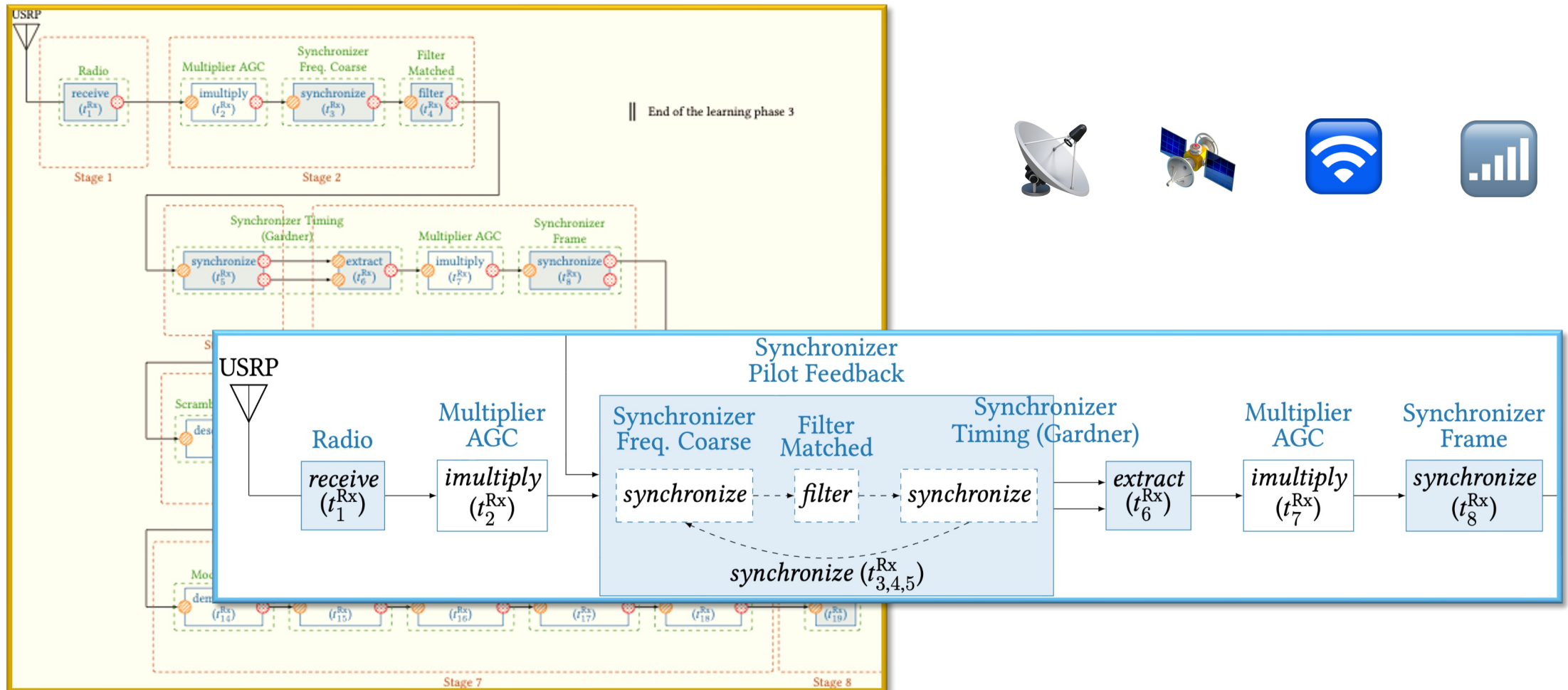
AFF3CT

Digital Communication Programming Environment – Bit Error Rate / Frame Error Rate Simulation

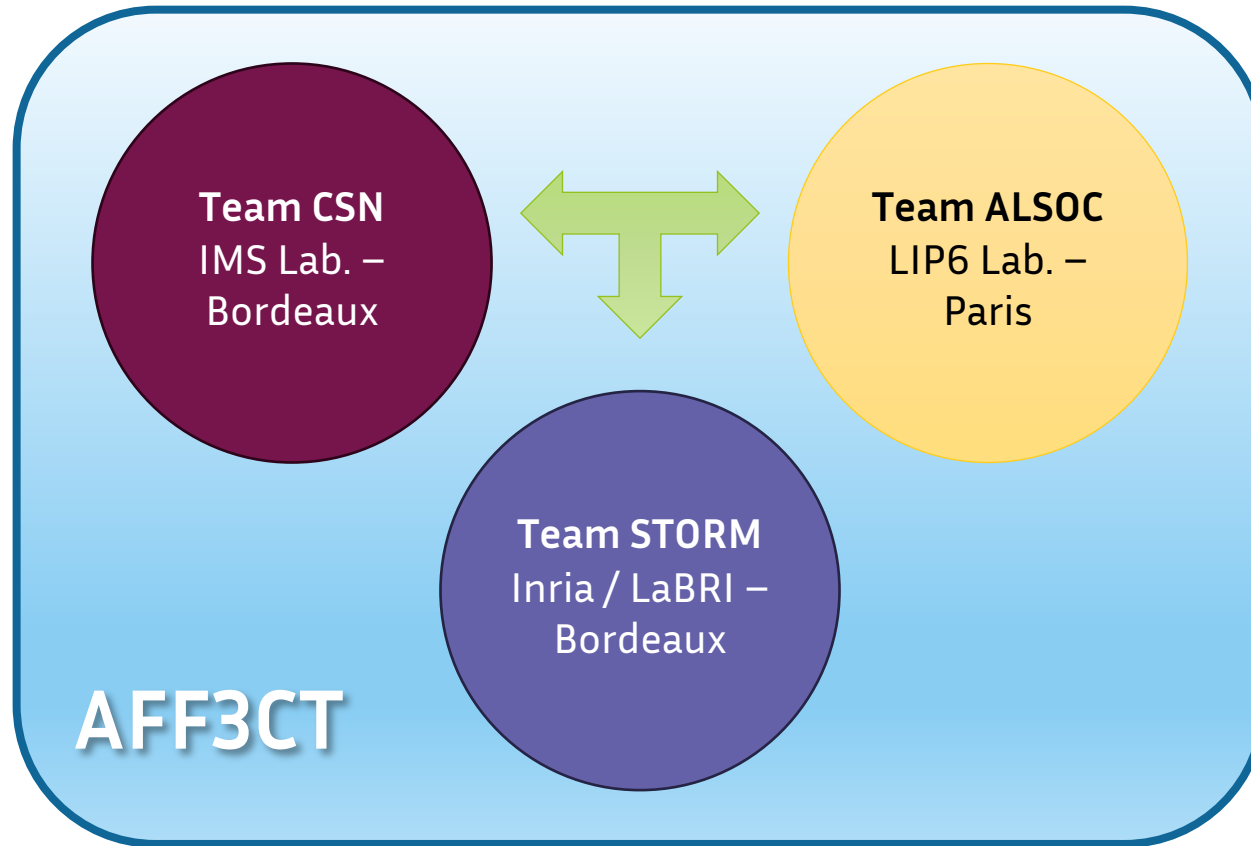


AFF3CT

Digital Communication Programming Environment – Real SDR Chains Exploitation



Involved Academic Research Teams



Team STORM

Inria & LaBRI, Bordeaux

Research Axes

- **Abstraction**

- Programming models
- Programming interfaces

- **Optimization**

- Performance, energy
- Load-balancing, scheduling
- Fault tolerance & checkpointing
- Code transformation
- Trade-off exploration

- **Analysis**

- Verification
- Visualization
- Teaching



Applications & Libraries

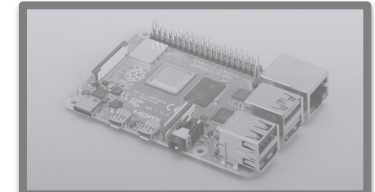
Abstraction

Optimization

Analysis

Computing architectures

Multicore CPU, SIMD, GPU, FPGA



Team CSN

IMS Laboratory, Bordeaux

Research Axes

- **Theme 1 : Conception of analogic and mixed circuits**
 - Building blocks
 - Frequency generation systems, clock & data recovery systems
 - Numerical compensation for PA, ADC, DAC, DDS
- **Theme 2 : Algorithmic / architecture adequation**
 - Methodology and conception of dedicated and/or programmable architectures
 - Applications in numerical communications numériques, neural networks, ...
- **Theme 3 : Numerical Communications**
 - Conception of algorithms for numerical communication systems
 - Channel coding, synchronization, equalization, ...

Team ALSOC

LIP6 Laboratory, Paris

Research Axes

Hardware and Software for Embedded Systems

- System on Chip, Multiprocessors, Manycores, SIMD, GPU
- Embedded floating point format, Embedded operating system
- Test & Verification
- Compilation, optimization, Synchronous Data Flow

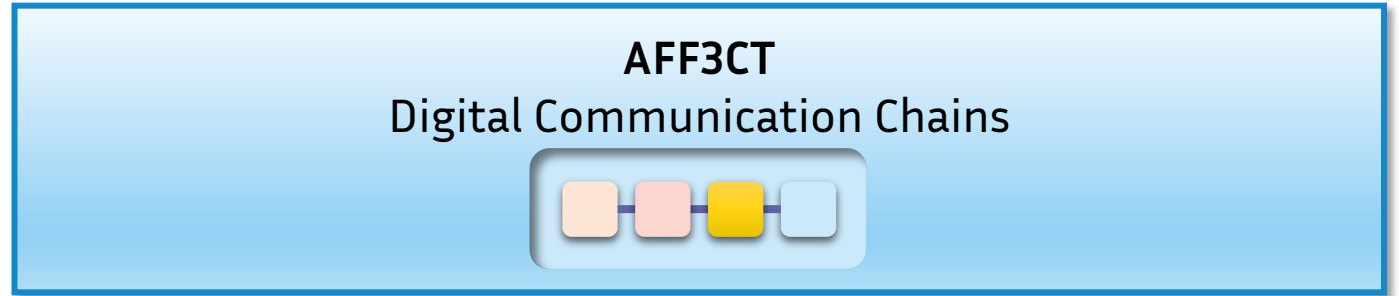
2.

AFF3CT Architecture

AFF3CT Architecture

Software Layout

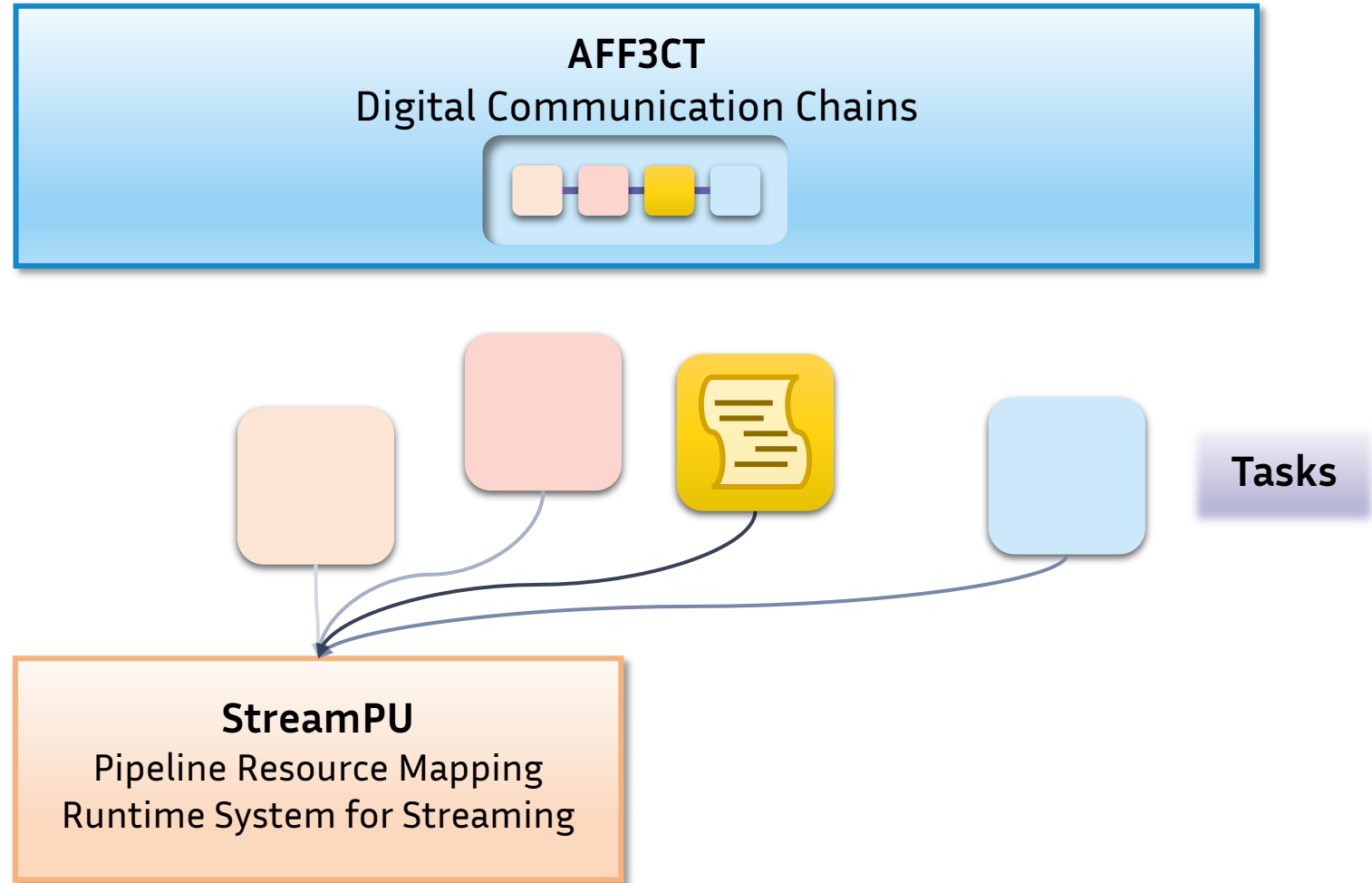
- **AFF3CT**
 - Digital Communication
 - BER/FER Simulator
 - C++ Library of chain modules



AFF3CT Architecture

Software Layout

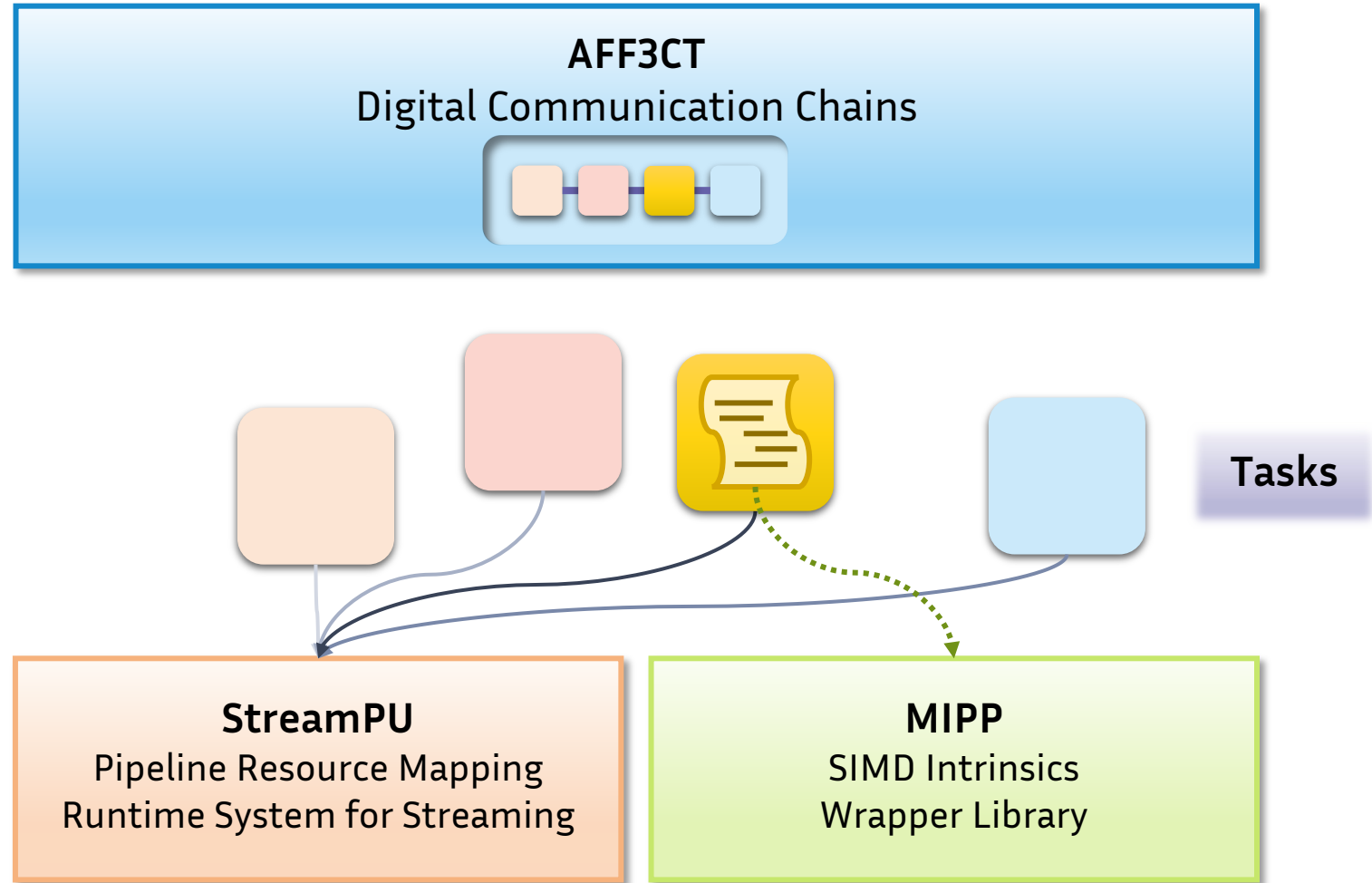
- **AFF3CT**
 - Digital Communication
 - BER/FER Simulator
 - C++ Library of chain modules
- **StreamPU**
 - Runtime system
 - Task scheduling
 - Streaming optimization



AFF3CT Architecture

Software Layout

- **AFF3CT**
 - Digital Communication
 - BER/FER Simulator
 - C++ Library of chain modules
- **StreamPU**
 - Runtime system
 - Task scheduling
 - Streaming optimization
- **MIPP**
 - SIMD Intrinsic programming
 - Hardware abstraction
 - C++ Library



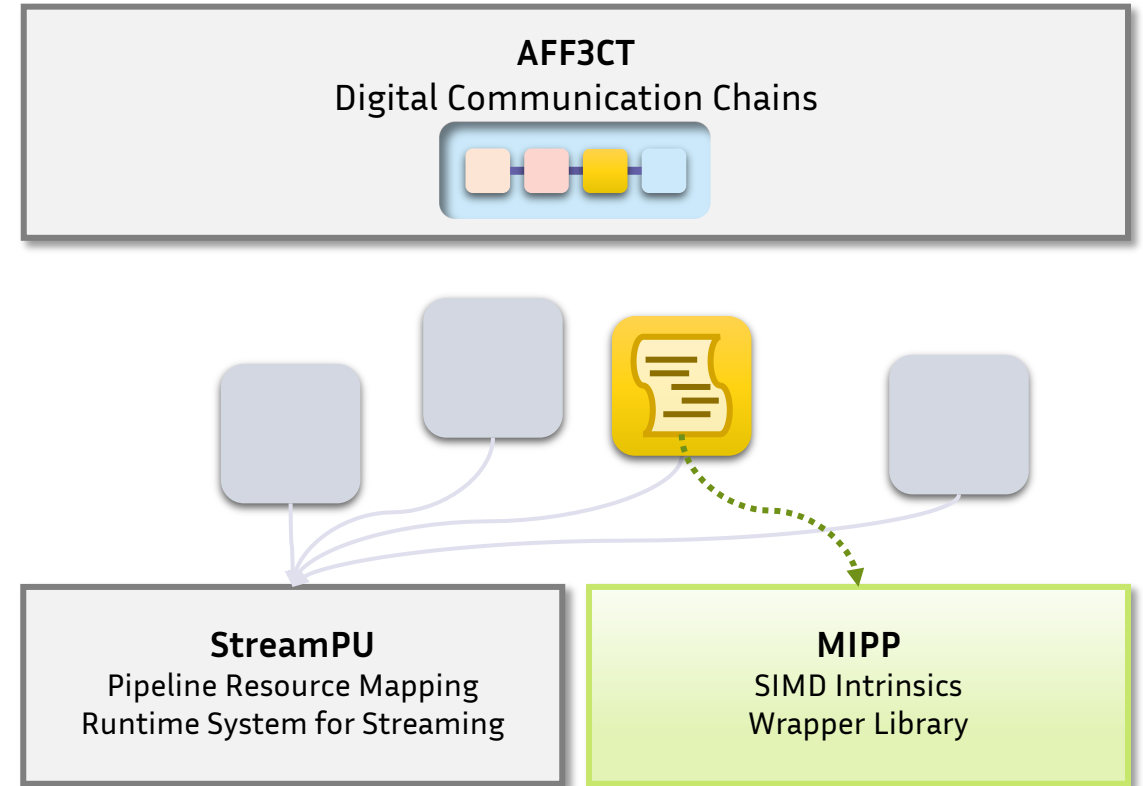
3.

SIMD Architecture Programming with the MIPP Library

Architecture — MIPP

Abstraction for SIMD Programming

- **C++ wrapper Library**
 - Header-only
 - Templates



MIPP – MyIntrinsics++

C++ 11 Wrapper Library

- **SIMD Abstraction Layer**
 - Architecture-independent programming interface
 - Built on machine-dependent *intrinsics* routines
- **Characteristics**
 - Supported instruction sets
 - Intel & AMD: SSE, AVX / AVX2, AVX-512
 - ARM: NEON, SVE (partial)
 - Supported data types
 - Floating point: FP32, FP64
 - Signed integers: 8-bit ... 64-bit
 - Unsigned integers: 8-bit ... 64-bit
- **Header-only (*wrapper* library)**
 - C++ templates
 - Function inlining
 - **Abstraction handled at compile-time**

```
mipp::Reg<float> r1;          // r1 = | unknown | unknown | unknown | unknown |
r1 = {1.0, 2.0, 3.0, 4.0};    // r1 = |   +1.0 |   +2.0 |   +3.0 |   +4.0 |
mipp::Reg<float> r2, r3;
r2 = 2.0;                     // r2 = |   +2.0 |   +2.0 |   +2.0 |   +2.0 |
r3 = r1 + r2;                  // r3 = |   +3.0 |   +4.0 |   +5.0 |   +6.0 |
```

MIPP – MyIntrinsics++

C++ 11 Wrapper Library

- **SIMD Abstraction Layer**
 - Architecture-independent programming interface
 - Built on machine-dependent *intrinsics* routines
- **Characteristics**
 - Supported instruction sets
 - Intel & AMD: SSE, AVX / **AVX2**, AVX-512
 - ARM: NEON, SVE (partial)
 - Supported data types
 - Floating point: FP32, FP64
 - Signed integers: 8-bit ... 64-bit
 - Unsigned integers: 8-bit ... 64-bit
- **Header-only (*wrapper* library)**
 - C++ templates
 - Function inlining
 - **Abstraction handled at compile-time**

```
mipp::Reg<float> r1;          // r1 = | unknown | unknown | unknown | unknown |
r1 = {1.0, 2.0, 3.0, 4.0}; // r1 = |   +1.0 |   +2.0 |   +3.0 |   +4.0 |
mipp::Reg<float> r2, r3;
r2 = 2.0;                  // r2 = |   +2.0 |   +2.0 |   +2.0 |   +2.0 |
r3 = r1 + r2;                // r3 = |   +3.0 |   +4.0 |   +5.0 |   +6.0 |
```

```
/* . . . */

template <>
inline reg set1<float>(const float val) {
    return mm256_set1_ps(val);
}
```

```
/* . . . */
```

mipp_impl_AVX.hxx

MIPP – MyIntrinsics++

C++ 11 Wrapper Library

- **SIMD Abstraction Layer**
 - Architecture-independent programming interface
 - Built on machine-dependent *intrinsics* routines
- **Characteristics**
 - Supported instruction sets
 - Intel & AMD: SSE, AVX / **AVX2**, AVX-512
 - ARM: NEON, SVE (partial)
 - Supported data types
 - Floating point: FP32, FP64
 - Signed integers: 8-bit ... 64-bit
 - Unsigned integers: 8-bit ... 64-bit
- **Header-only (*wrapper* library)**
 - C++ templates
 - Function inlining
 - **Abstraction handled at compile-time**

```
mipp::Reg<float> r1;          // r1 = | unknown | unknown | unknown | unknown |
r1 = {1.0, 2.0, 3.0, 4.0}; // r1 = |   +1.0 |   +2.0 |   +3.0 |   +4.0 |
mipp::Reg<float> r2, r3;
r2 = 2.0;                    // r2 = |   +2.0 |   +2.0 |   +2.0 |   +2.0 |
r3 = r1 + r2;                // r3 = |   +3.0 |   +4.0 |   +5.0 |   +6.0 |
```

```
/* . . . */
template <>
inline reg add<float>(const reg v1, const reg v2) {
    return mm256_add_ps(v1, v2);
}
/* . . . */
```

mipp_impl_AVX.hxx

MIPP – MyIntrinsics++

C++ 11 Wrapper Library

- **SIMD Abstraction Layer**
 - Architecture-independent programming interface
 - Built on machine-dependent *intrinsics* routines
- **Characteristics**
 - Supported instruction sets
 - Intel & AMD: SSE, AVX / **AVX2**, AVX-512
 - ARM: **NEON**, SVE (partial)
 - Supported data types
 - Floating point: FP32, FP64
 - Signed integers: 8-bit ... 64-bit
 - Unsigned integers: 8-bit ... 64-bit
- **Header-only (*wrapper* library)**
 - C++ templates
 - Function inlining
 - **Abstraction handled at compile-time**

```
mipp::Reg<float> r1;          // r1 = | unknown | unknown | unknown | unknown |
r1 = {1.0, 2.0, 3.0, 4.0}; // r1 = |   +1.0 |   +2.0 |   +3.0 |   +4.0 |
mipp::Reg<float> r2, r3;
r2 = 2.0;                    // r2 = |   +2.0 |   +2.0 |   +2.0 |   +2.0 |
r3 = r1 + r2;                // r3 = |   +3.0 |   +4.0 |   +5.0 |   +6.0 |
```

```
/* . . . */
template <>
inline reg add<float>(const reg v1, const reg v2) {
    return mm256_add_ps(v1, v2);
}
/* . . . */
```

mipp_impl_AVX.hxx

```
/* . . . */
template <>
inline reg add<float>(const reg v1, const reg v2) {
    return vaddq_f32(v1, v2);
}
/* . . . */
```

mipp_impl_NEON.hxx

MIPP – MyIntrinsics++

C++ 11 Wrapper Library

- **SIMD Abstraction Layer**
 - Architecture-independent programming interface
 - Built on machine-dependent *intrinsics* routines
- **Characteristics**
 - Supported instruction sets
 - Intel & AMD: SSE, AVX / AVX2, AVX-512
 - ARM: NEON, SVE (partial)
 - Supported data types
 - Floating point: FP32, FP64
 - Signed integers: 8-bit ... 64-bit
 - Unsigned integers: 8-bit ... 64-bit
- **Header-only (*wrapper* library)**
 - C++ templates
 - Function inlining
 - **Abstraction handled at compile-time**

```
#include <cstdlib>
#include <mipp.h>

int main() {
    const int n = 32000;
    mipp::vector<float> vA(n);
    mipp::vector<float> vB(n);
    mipp::vector<float> vC(n);

    for (int i = 0; i < n; i++) vA[i] = rand() % 10;
    for (int i = 0; i < n; i++) vB[i] = rand() % 10;

    mipp::Reg<float> rA, rB, rC;

    for (int i = 0; i < n; i += mipp::N<float>()) {
        rA.load(&vA[i]);
        rB.load(&vB[i]);
        rC = rA + rB;
        rC.store(&vC[i]);
    }

    return 0;
}
```

Addition of Two Vectors

MIPP – MyIntrinsics++

C++ 11 Wrapper Library

- **SIMD Abstraction Layer**
 - Architecture-independent programming interface
 - Built on machine-dependent *intrinsics* routines
- **Characteristics**
 - Supported instruction sets
 - Intel & AMD: SSE, AVX / AVX2, AVX-512
 - ARM: NEON, SVE (partial)
 - **RISC-V** "V" extension (on-going work)
 - Supported data types
 - Floating point: FP32, FP64
 - Signed integers: 8-bit ... 64-bit
 - Unsigned integers: 8-bit ... 64-bit
- **Header-only (*wrapper* library)**
 - C++ templates
 - Function inlining
 - **Abstraction handled at compile-time**

```
#include <cstdlib>
#include <mipp.h>

int main() {
    const int n = 32000;
    mipp::vector<float> vA(n);
    mipp::vector<float> vB(n);
    mipp::vector<float> vC(n);

    for (int i = 0; i < n; i++) vA[i] = rand() % 10;
    for (int i = 0; i < n; i++) vB[i] = rand() % 10;

    mipp::Reg<float> rA, rB, rC;

    for (int i = 0; i < n; i += mipp::N<float>()) {
        rA.load(&vA[i]);
        rB.load(&vB[i]);
        rC = rA + rB;
        rC.store(&vC[i]);
    }

    return 0;
}
```

Addition of Two Vectors

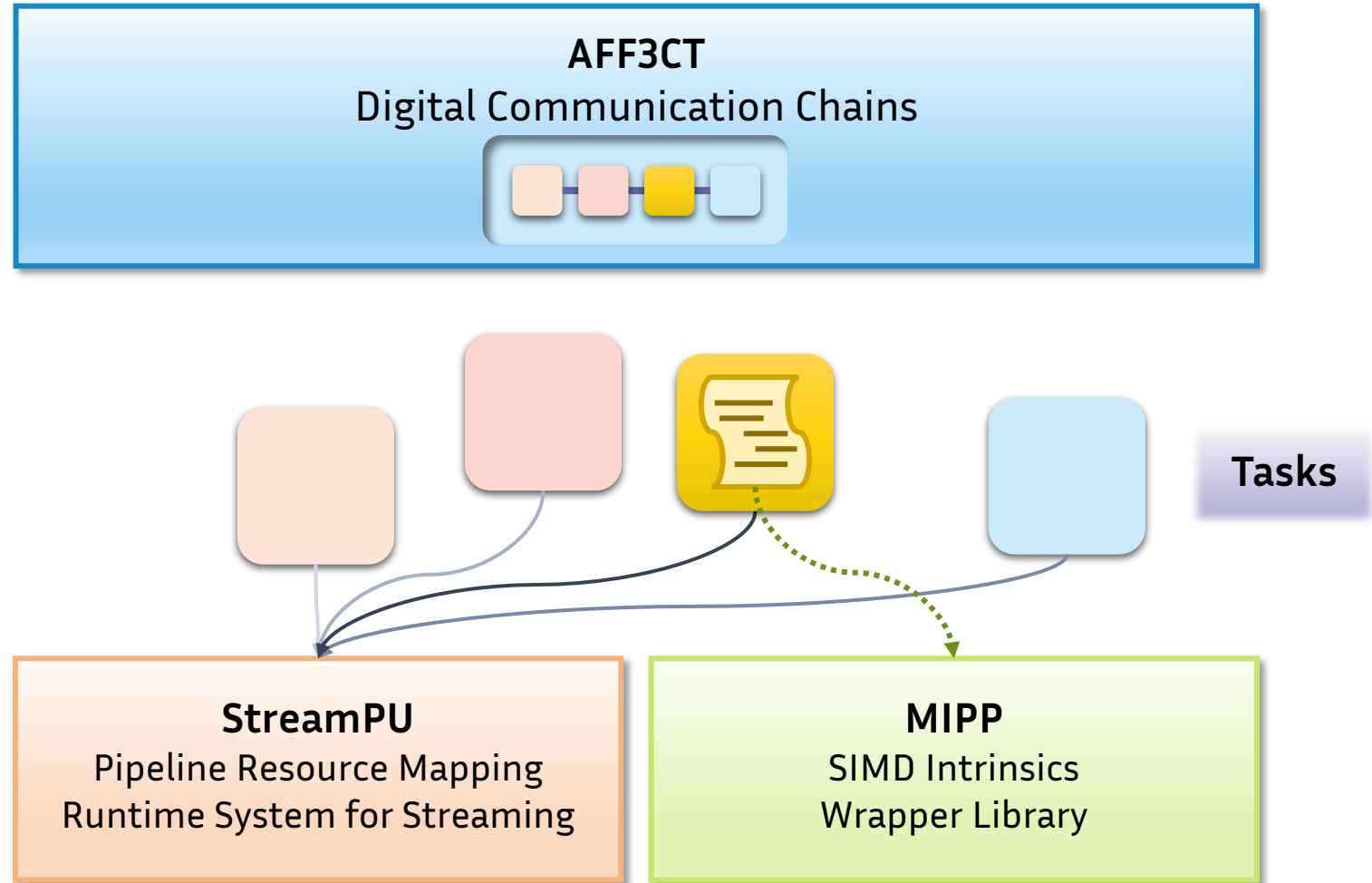
4.

Task-based Streaming on Multicore with the StreamPU Runtime

AFF3CT Architecture

Software Layout

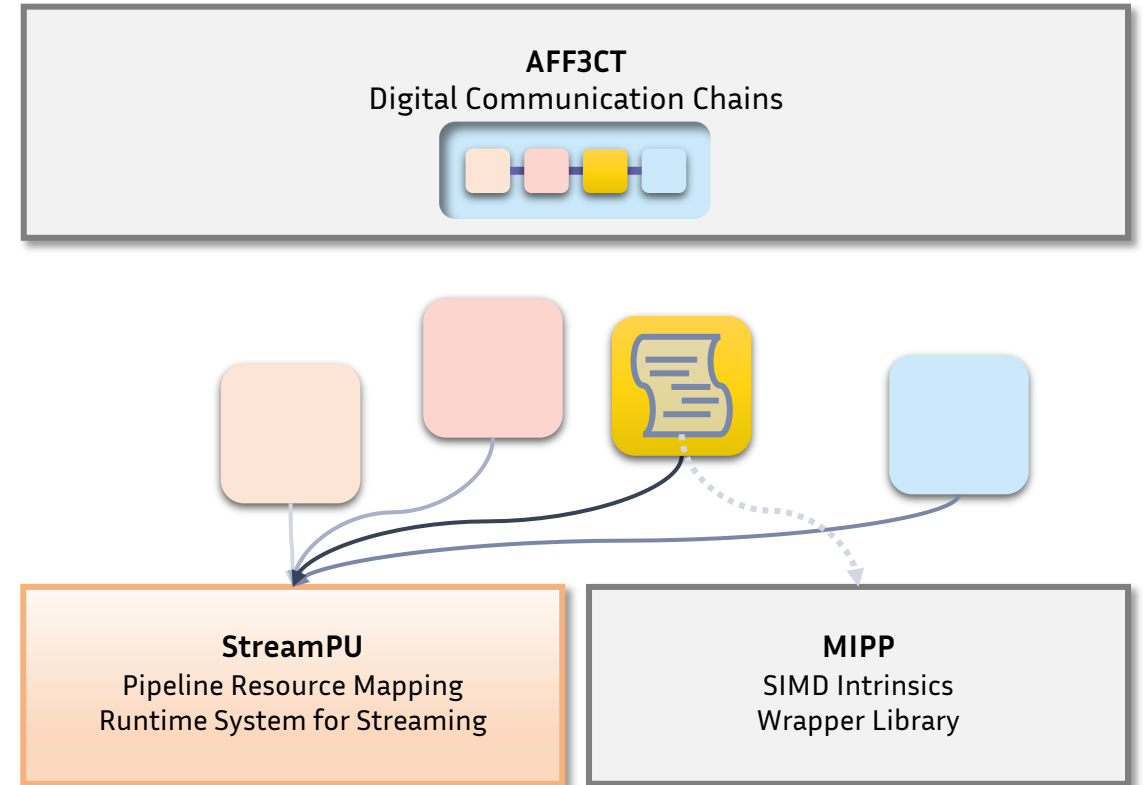
- **AFF3CT**
 - Digital Communication
 - BER/FER Simulator
 - C++ Library of chain modules
- **StreamPU**
 - Runtime system
 - Task scheduling
 - Streaming optimization
- **MIPP**
 - SIMD Intrinsic programming
 - Hardware abstraction
 - C++ Library



Architecture — StreamPU

Streaming Runtime System

- **Multicore Parallelism**
 - C++ threads
 - Chain replication
 - Scheduling & optimization



StreamPU API

C++ Embedded Language for Building Streaming Chains

StreamPU API

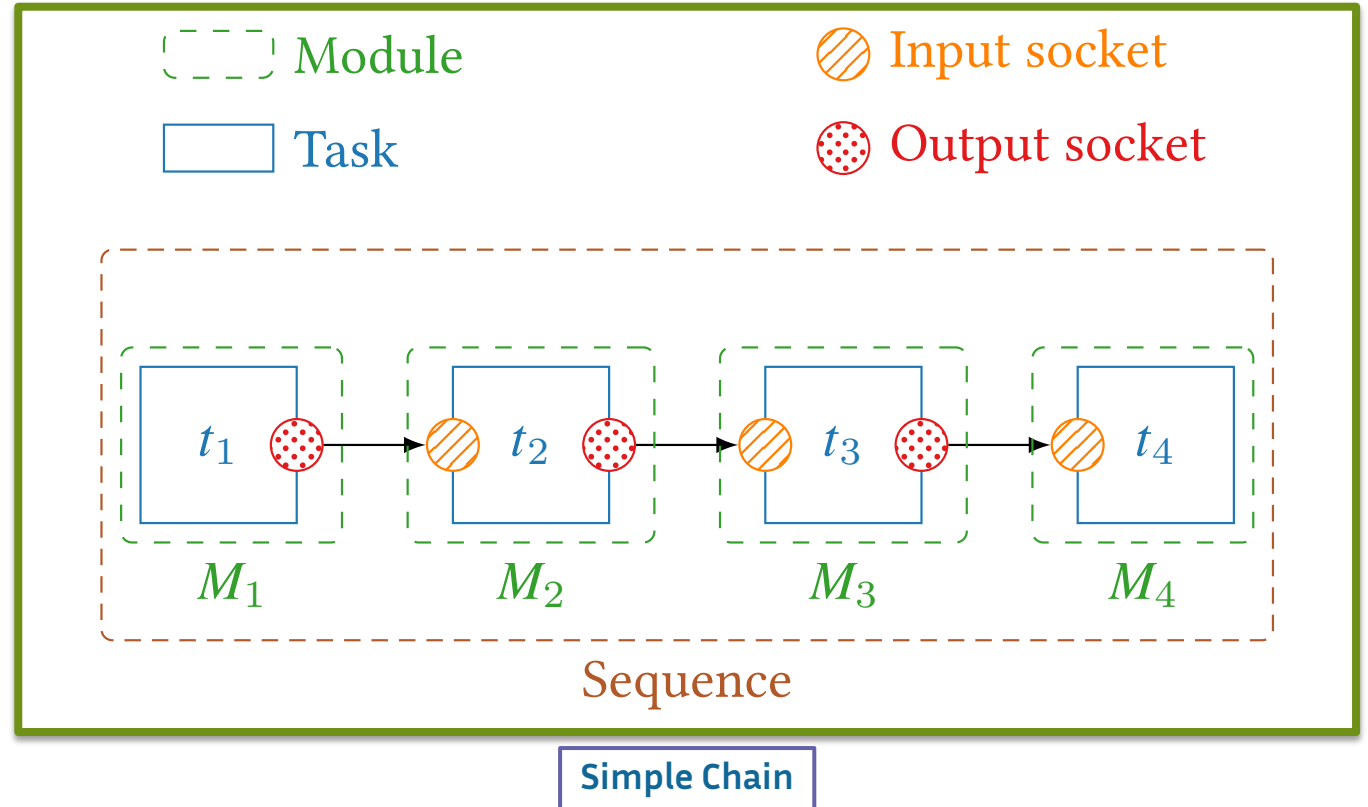
Basic building blocks

```
// 1) create the module objects
M1 m1(); M2 m2(); M3 m3(); M4 m4();

// 2) bind the tasks
m2["t2::in"] = m1["t1::out"];
m3["t3::in"] = m2["t2::out"];
m4["t4::in"] = m3["t3::out"];

// 3) create the sequence (stop
//      automatically at t4 task)
Sequence seq(m1["t1"]);

// 4) execute the sequence (tasks
//      graph is executed 100 times)
unsigned int exe_counter = 0;
seq.exec([&exe_counter]() {
    return ++exe_counter >= 100;
});
```



StreamPU API

Basic building blocks

// 1) create the module objects

```
M1 m1(); /* ... */ M7 m7();
```

```
std::vector<TYPE> some_data(SIZE);
```

// 2) bind the tasks

```
m1["t1::in" ] = some_data;
```

```
m3["t3::in" ] = some_data;
```

```
m2["t2::in" ] = m1["t1::out"];
```

```
m2["t4::in1" ] = m2["t2::out"];
```

```
m2["t4::in2" ] = m3["t3::out"];
```

```
m4["t5::in" ] = m2["t4::out"];
```

```
m5["t6::in" ] = m2["t4::out"];
```

```
m6["t7::in" ] = m4["t5::out"];
```

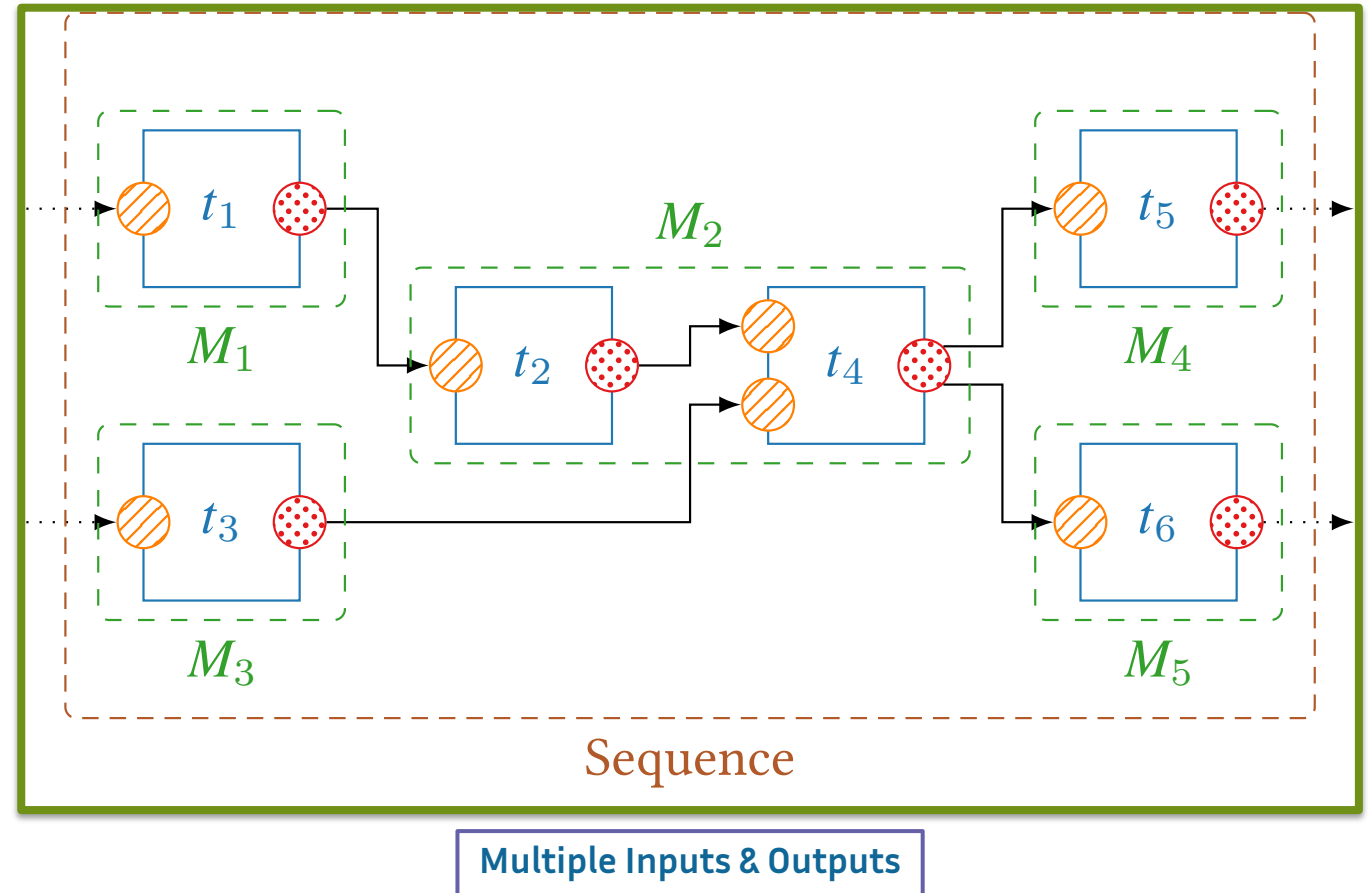
```
m7["t8::in" ] = m5["t6::out"];
```

// 3) create the sequence

```
Sequence seq(  
    { m1["t1"], m3["t3"] }, // first tasks  
    { m4["t5"], m5["t6"] }); // last tasks
```

// 4) execute the sequence (no stop)

```
seq.exec([]() { return false; });
```



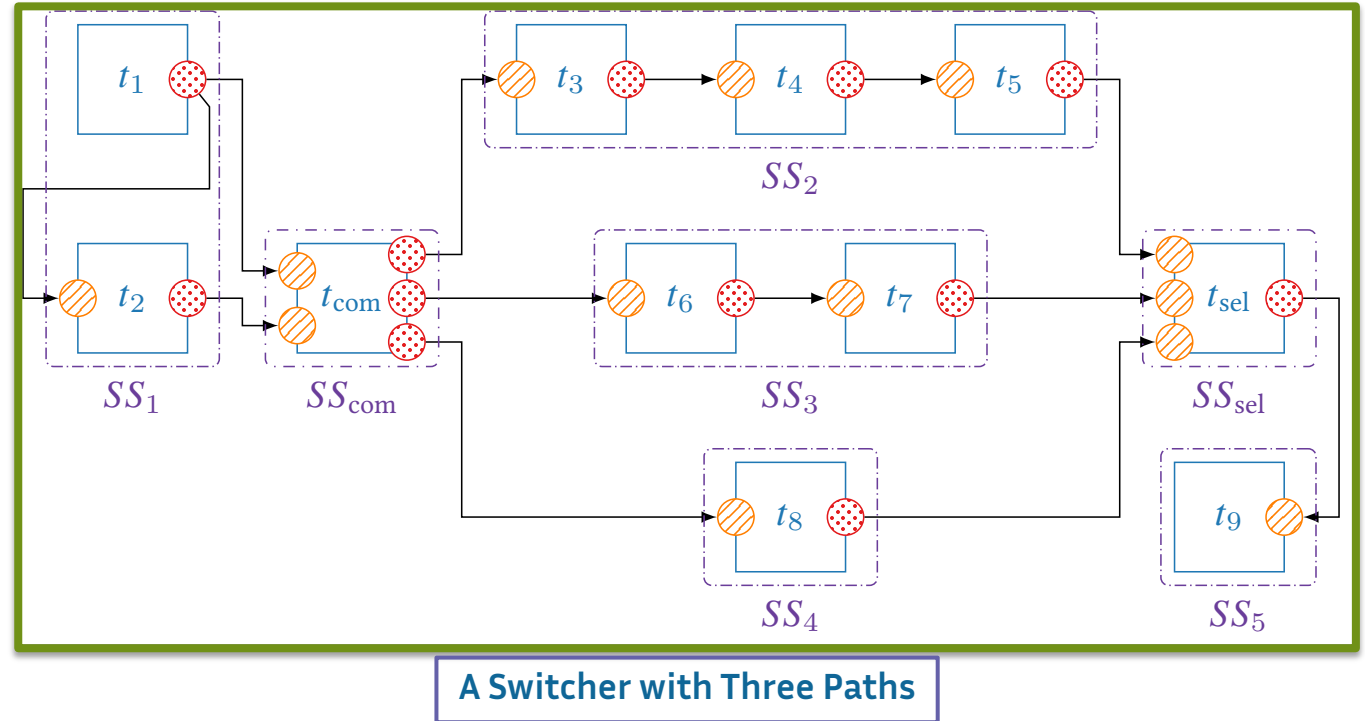
StreamPU API

Conditionals: "switcher" objects

```
M1 m1(); /* ... */ M9 m9();
Switcher sw(3); // 3 exclusive paths

sw["tcom::in1"] = m1[ "t1::out" ];
m2[ "t2::in" ] = m1[ "t1::out" ];
sw["tcom::in2"] = m2[ "t2::out" ];
// sub-seq. 2, executed if tcom::in2 = 0
m3[ "t3::in" ] = sw["tcom::out1"];
m4[ "t4::in" ] = m3[ "t3::out" ];
m5[ "t5::in" ] = m4[ "t4::out" ];
// sub-seq. 3, executed if tcom::in2 = 1
m6[ "t6::in" ] = sw["tcom::out1"];
m7[ "t7::in" ] = m6[ "t6::out" ];
// sub-seq. 4, executed if tcom::in2 = 2
m8[ "t8::in" ] = sw["tcom::out1"];
// merge exclusive paths
sw["tsel::in1"] = m5[ "t5::out" ];
sw["tsel::in2"] = m7[ "t7::out" ];
sw["tsel::in3"] = m8[ "t8::out" ];
// last task binding
m6[ "t6::in" ] = sw["tsel::out" ];
```

```
Sequence seq(m1["t1"]);
seq.exec([]() { return false; });
```

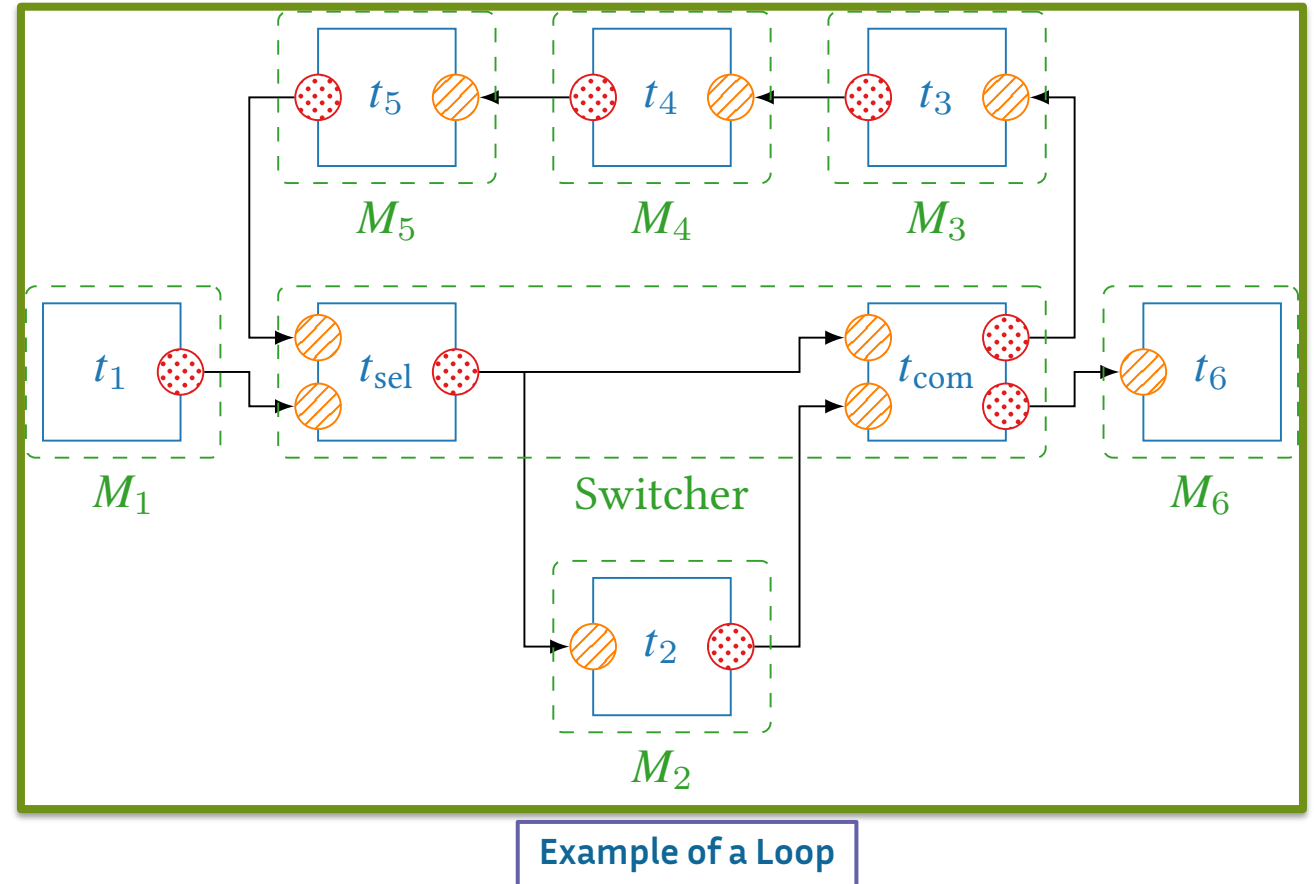


StreamPU API

Conditionals & repetitions: loops

```
M1 m1(); /* ... */ M6 m6();  
Switcher sw(2); // 2 exclusive paths  
  
sw["tsel::in2"] = m1["t1::out"];  
m2["t2::in"] = sw["tsel::out"];  
sw["tcom::in1"] = sw["tsel::out"];  
sw["tcom::in2"] = m2["t2::out"];  
// sub-seq. 3, executed if tcom::in2 = 0  
m3["t3::in"] = sw["tcom::out1"];  
m4["t4::in"] = m3["t3::out"];  
m5["t5::in"] = m4["t4::out"];  
sw["tsel::in1"] = m5["t5::out"];  
// sub-seq. 4, executed if tcom::in2 = 1  
m6["t6::in"] = sw["tcom::out2"];
```

Sequence seq(m1["t1"]);
seq.exec([]() { return false; });



StreamPU API

Multithreading and pipelining

```
// 1) creation of the module objects
M1 m1(); M2 m2(); M3 m3(); M4 m4(); M5 m5(); M6 m6();

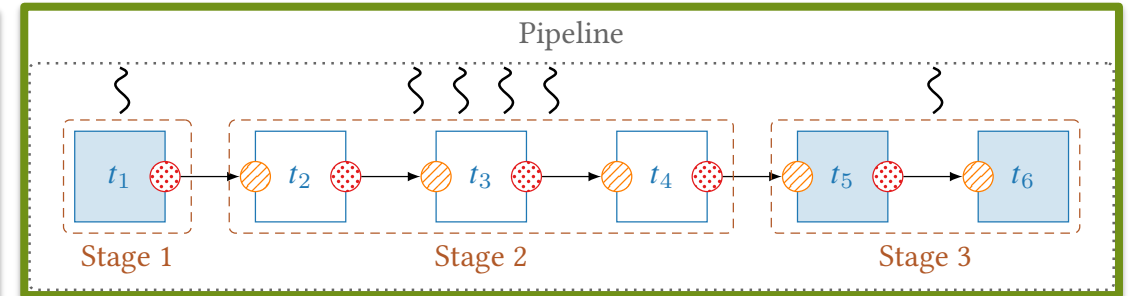
// 2) binding of the tasks
m2["t2::in"]=m1["t1::out"];
m3["t3::in"]=m2["t2::out"];
m4["t4::in"]=m3["t3::out"];
m5["t5::in"]=m4["t4::out"];
m6["t6::in"]=m5["t5::out"];

// 3) creation of the pipeline (= sequences and pipeline analyses)
runtime::Pipeline pip(
  m1["t1"], // first task of the graph
  { { { m1["t1"] }, { m1["t1"] }, }, // first & last tasks of stage 1
    { { m2["t2"] }, { m4["t4"] }, }, // first & last tasks of stage 2
    { { m5["t5"] }, { m6["t6"] }, }, // first & last tasks of stage 3

    { 1, 4, 1 }, // number of threads per stage

    {
      1 }, // pin thread '1' of stage 1 to core '1'
    { 3, 4, 5, 6 }, // pin threads '1-4' of stage 2 to cores '3-6'
    {
      8 } }; // pin thread '1' of stage 3 to core '8'

// 4) execution of the pipeline, it is indefinitely executed in loop
pip.exec([]() { return false; });
```



Example of a Pipeline

StreamPU API

Multithreading and pipelining: replication of parallel sequences

```
// 1) creation of the module objects
M1 m1(); M2 m2(); M3 m3(); M4 m4(); M5 m5(); M6 m6();

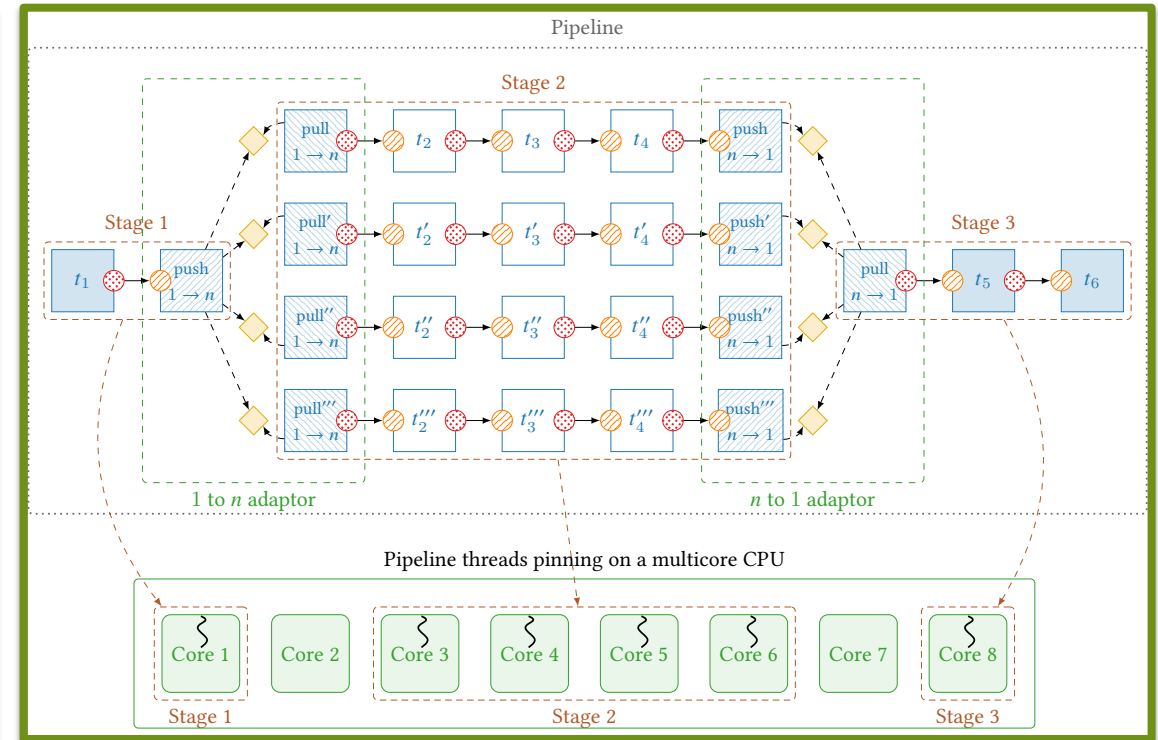
// 2) binding of the tasks
m2["t2::in"] = m1["t1::out"];
m3["t3::in"] = m2["t2::out"];
m4["t4::in"] = m3["t3::out"];
m5["t5::in"] = m4["t4::out"];
m6["t6::in"] = m5["t5::out"];

// 3) creation of the pipeline (= sequences and pipeline analyses)
runtime::Pipeline pip(
    m1["t1"], // first task of the graph
    { { { m1["t1"] }, { m1["t1"] } }, }, // first & last tasks of stage 1
    { { { m2["t2"] }, { m4["t4"] } }, }, // first & last tasks of stage 2
    { { { m5["t5"] }, { m6["t6"] } }, }, // first & last tasks of stage 3

    { 1, 4, 1 }, // number of threads per stage

    {
        { 1 }, // pin thread '1' of stage 1 to core '1'
        { 3, 4, 5, 6 }, // pin threads '1-4' of stage 2 to cores '3-6'
        { 8 } // pin thread '1' of stage 3 to core '8'
    });

// 4) execution of the pipeline, it is indefinitely executed in loop
pip.exec([]() { return false; });
```



Example of a Pipeline

StreamPU API

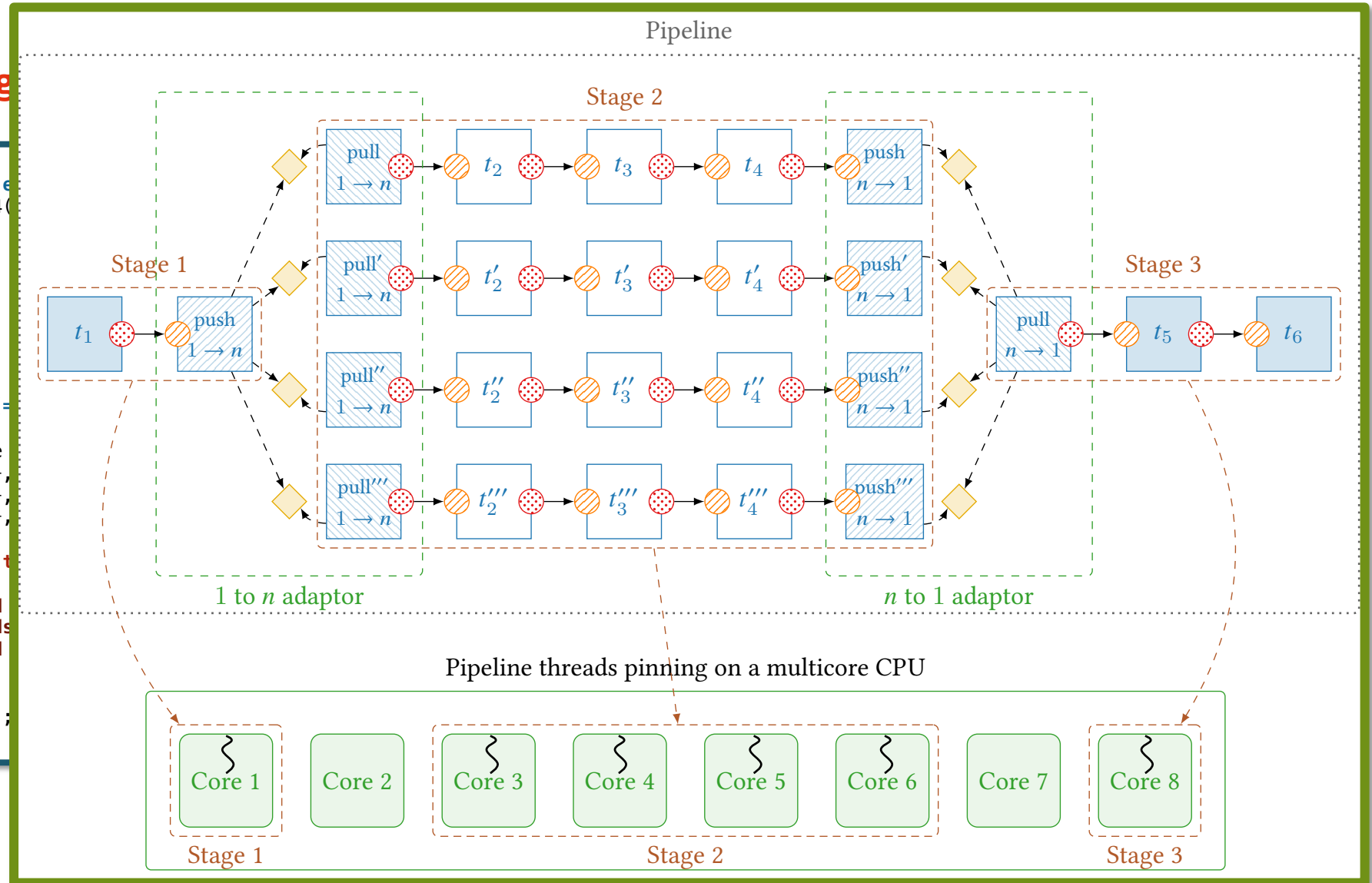
C++ Embedded Language

```
// 1) creation of the module objects
M1 m1(); M2 m2(); M3 m3(); M4 m4();

// 2) binding of the tasks
m2["t2::in"] = m1["t1::out"];
m3["t3::in"] = m2["t2::out"];
m4["t4::in"] = m3["t3::out"];
m5["t5::in"] = m4["t4::out"];
m6["t6::in"] = m5["t5::out"];

// 3) creation of the pipeline (=
runtime::Pipeline pip(
    m1["t1"], // first task of the
    { { { m1["t1"] }, { m1["t1"] } },
      { { m2["t2"] }, { m4["t4"] } },
      { { m5["t5"] }, { m6["t6"] } },
    },
    { 1, 4, 1 }, // number of threads
    { { 1 }, // pin thread
      { 3, 4, 5, 6 }, // pin threads
      { 8 } }; // pin threads

// 4) execution of the pipeline,
pip.exec([]() { return false; });
```



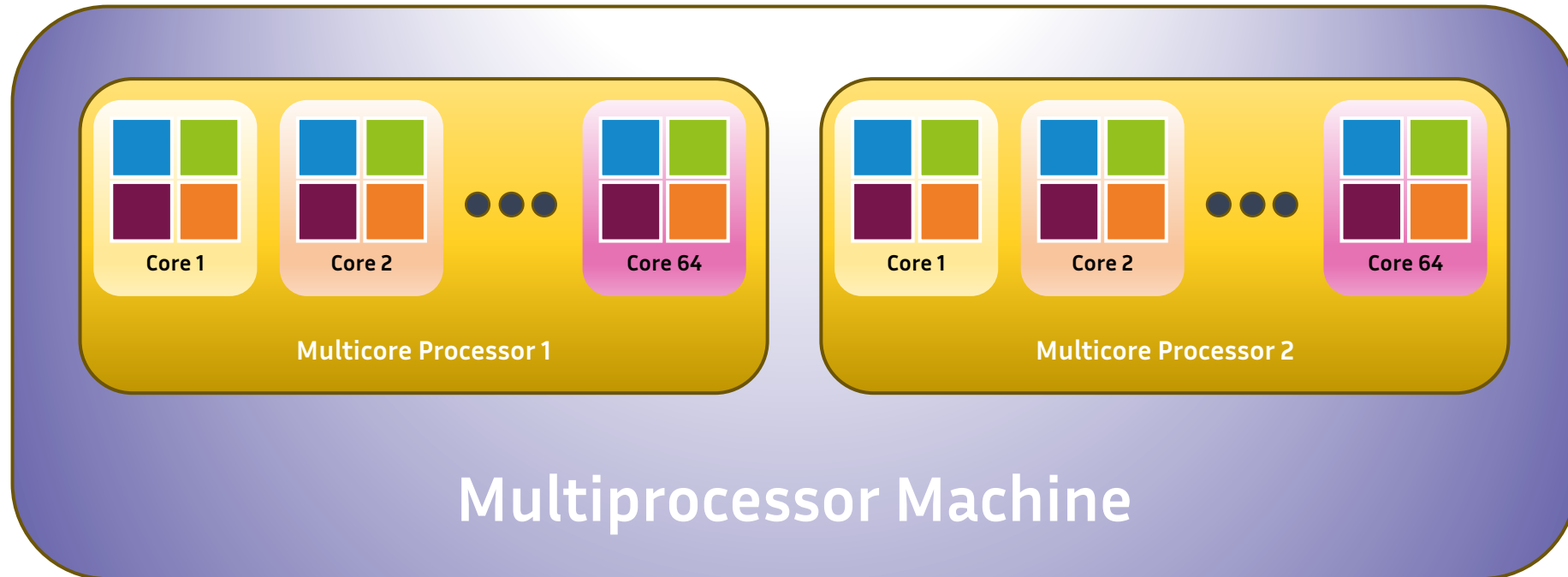
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

StreamPU – Multicore Parallelism

Pipeline Resource Mapping

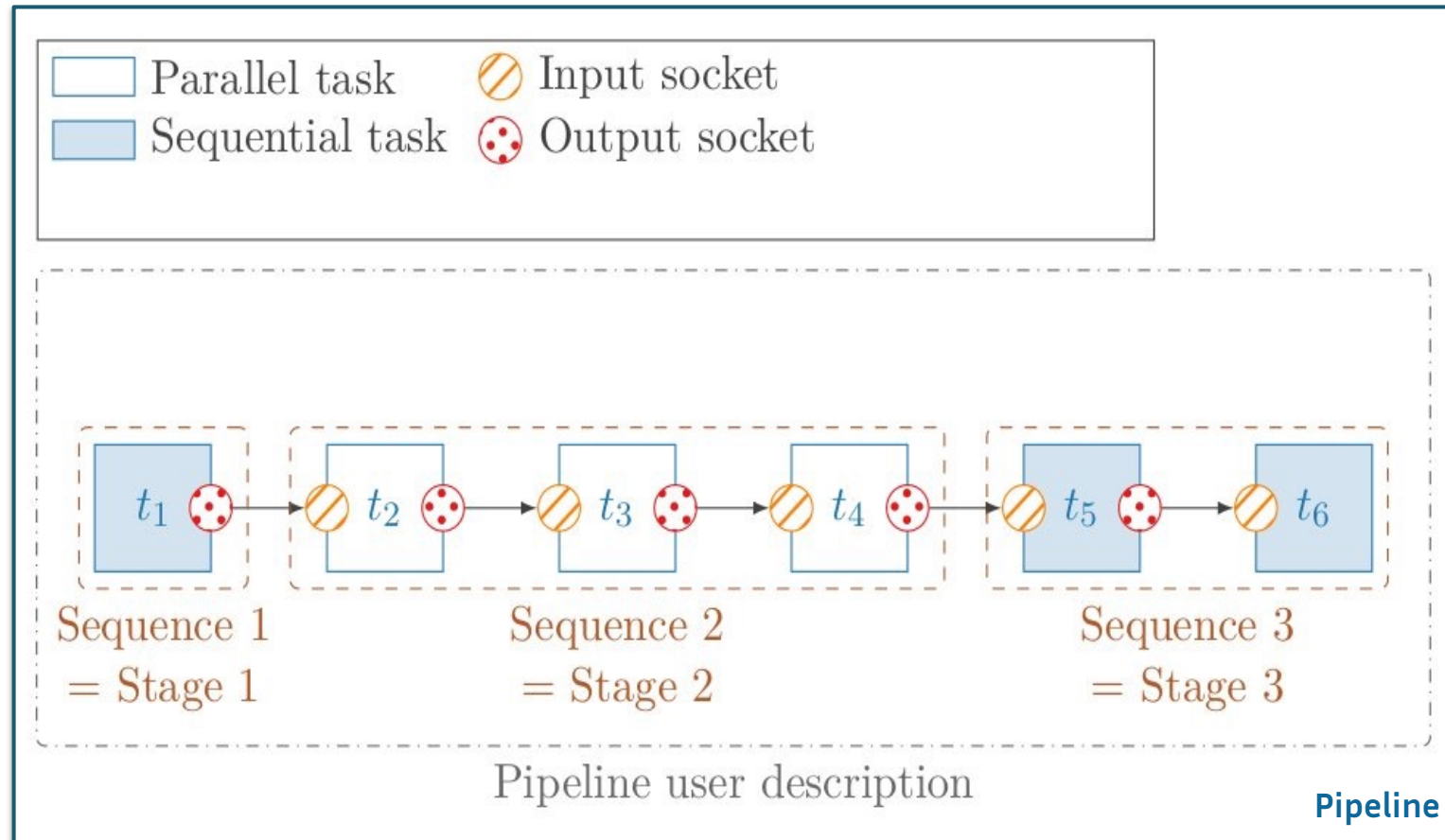
PhD Thesis – Diane Orhan



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

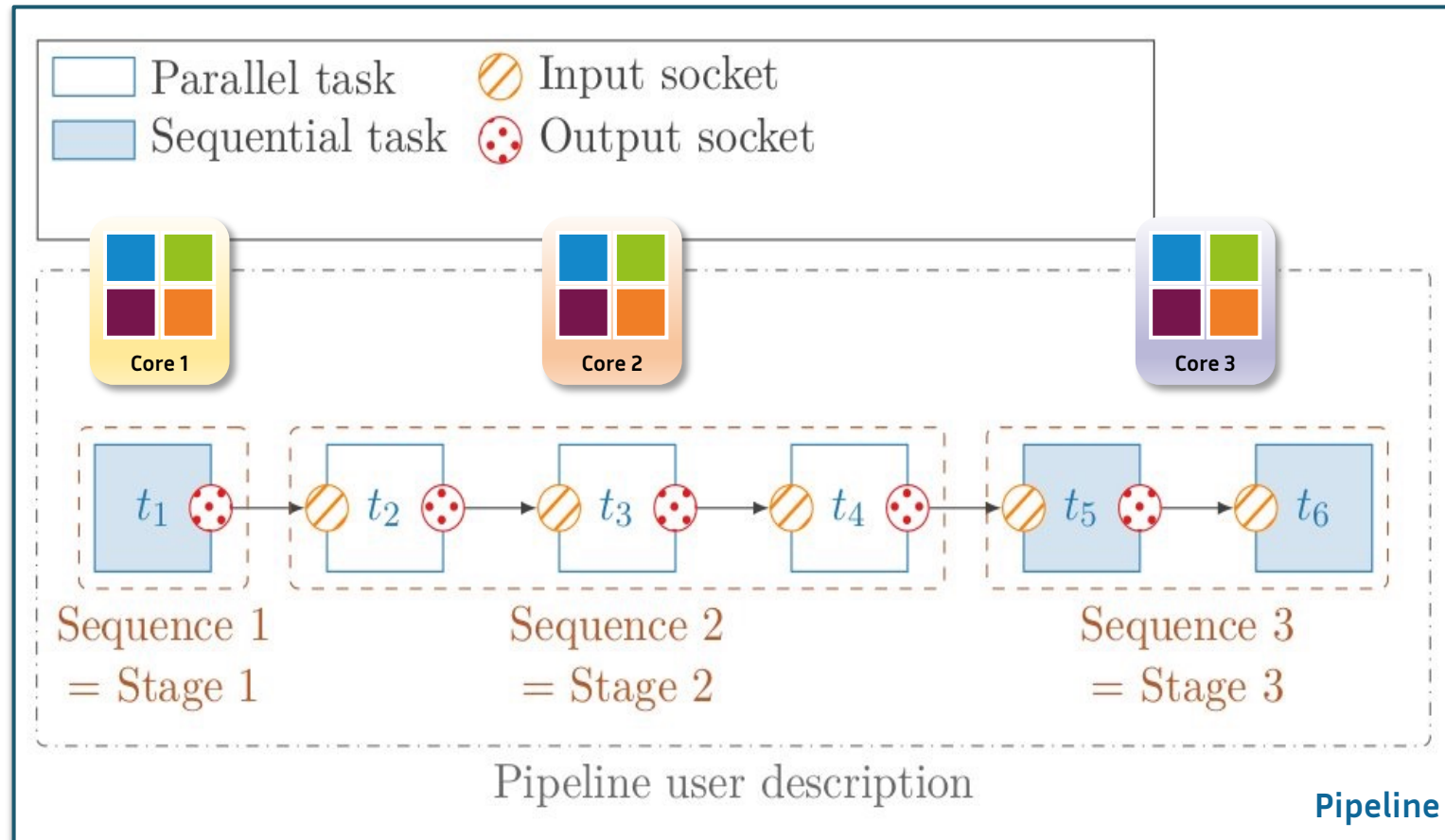
PhD Thesis – Diane Orhan



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

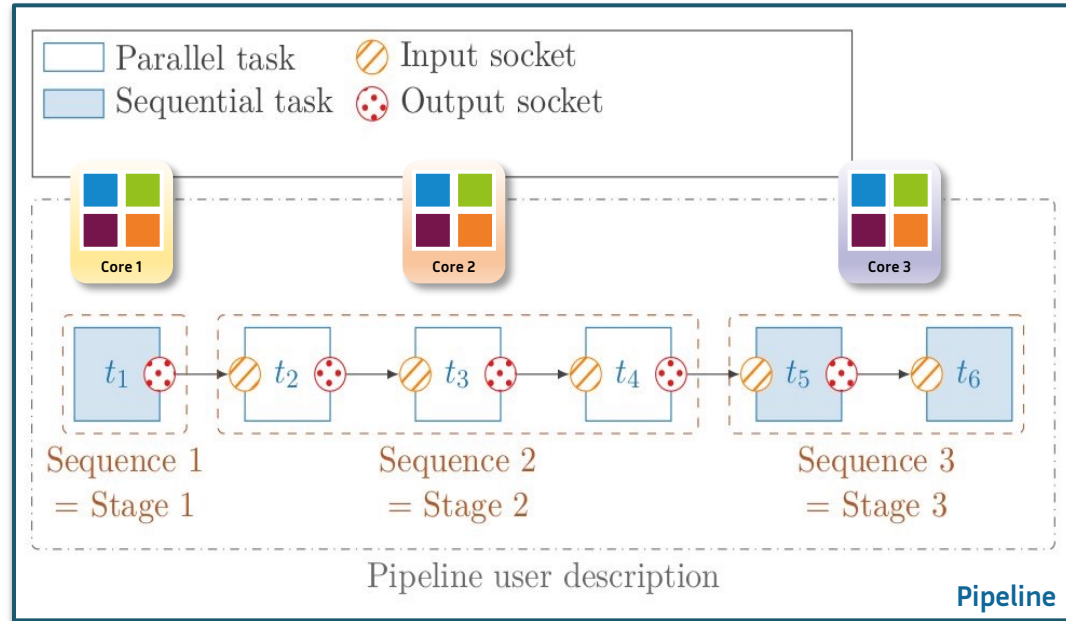
PhD Thesis – Diane Orhan



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

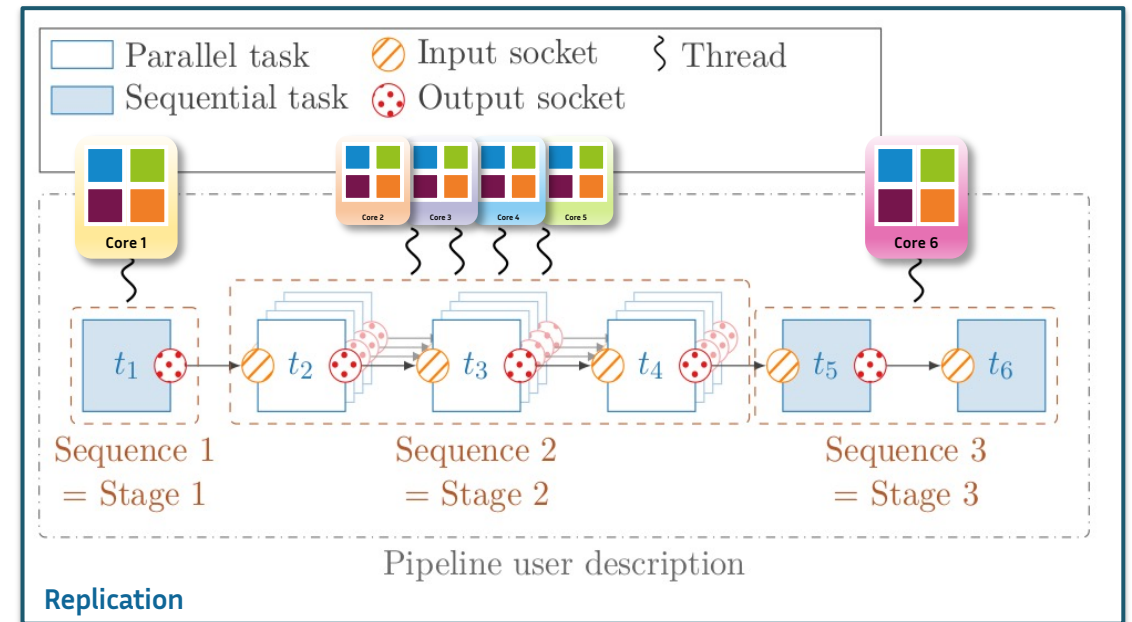
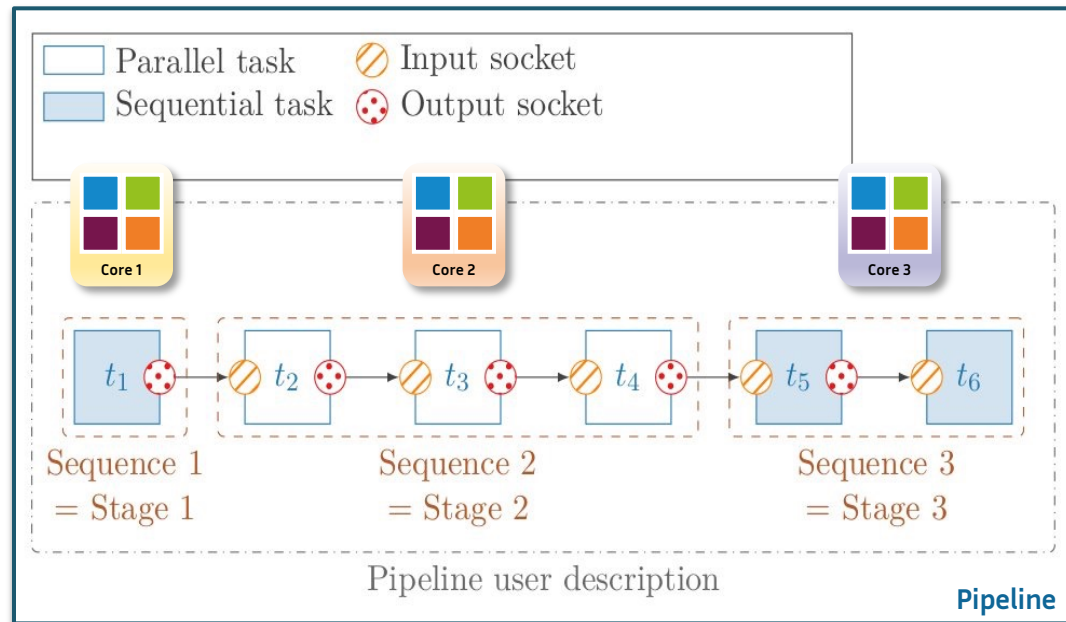
PhD Thesis – Diane Orhan



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

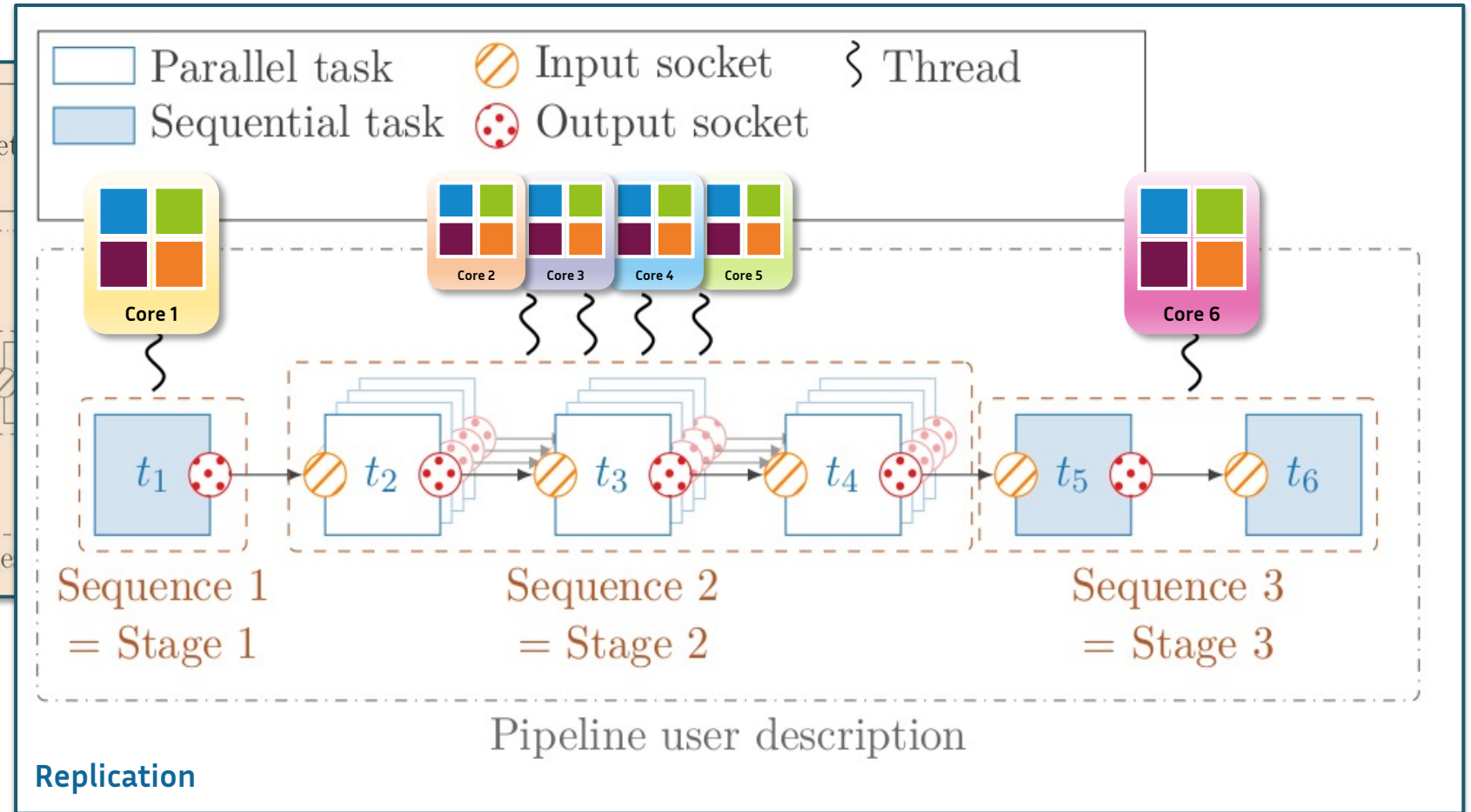
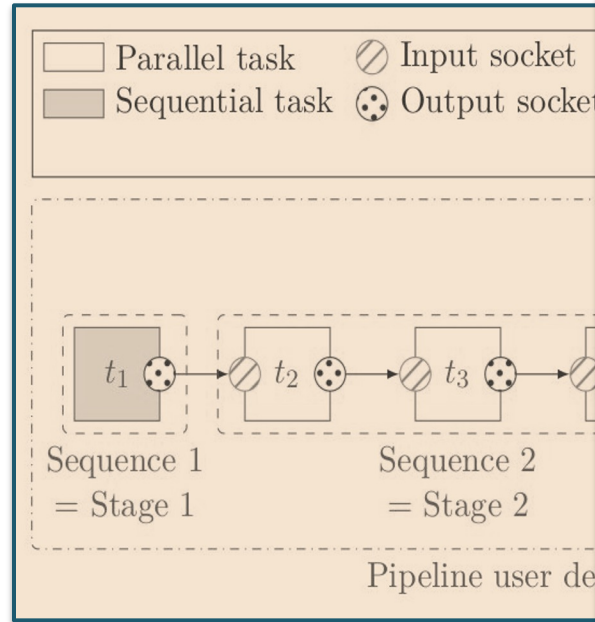
PhD Thesis – Diane Orhan



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

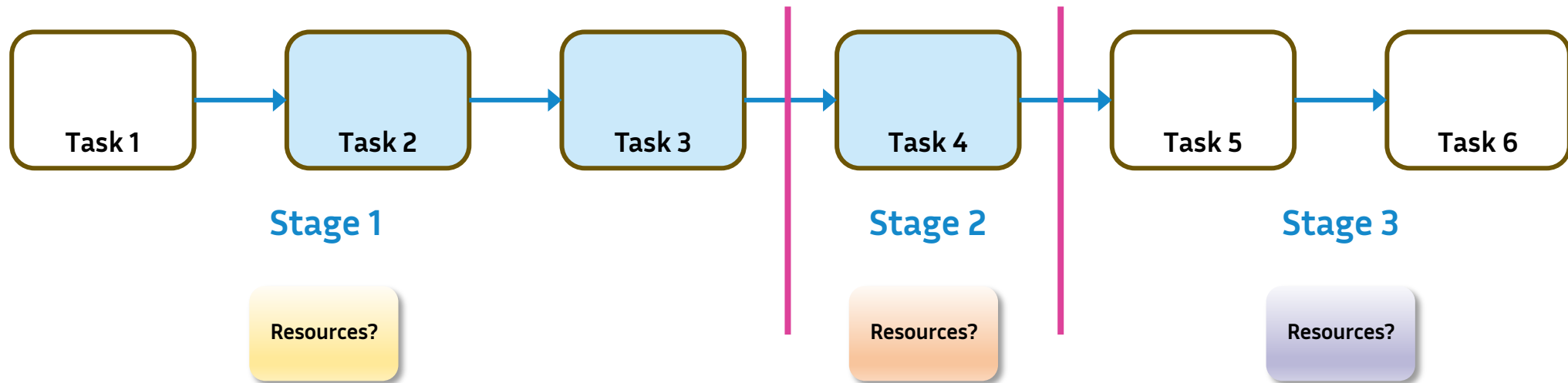
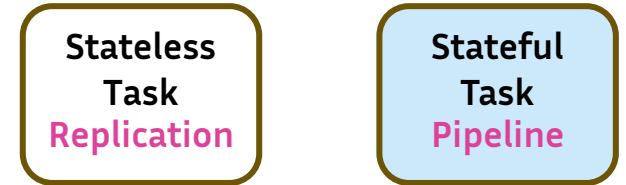
PhD Thesis – Diane Orhan



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

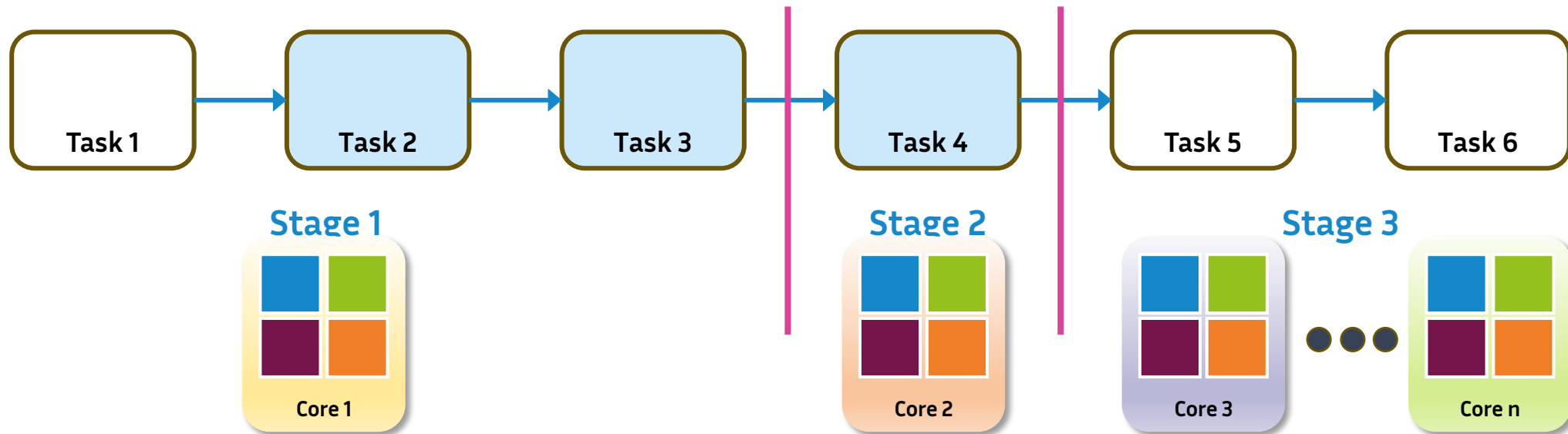
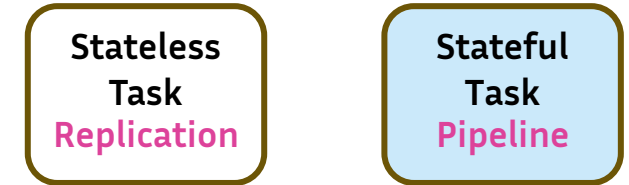
PhD Thesis – Diane Orhan



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan



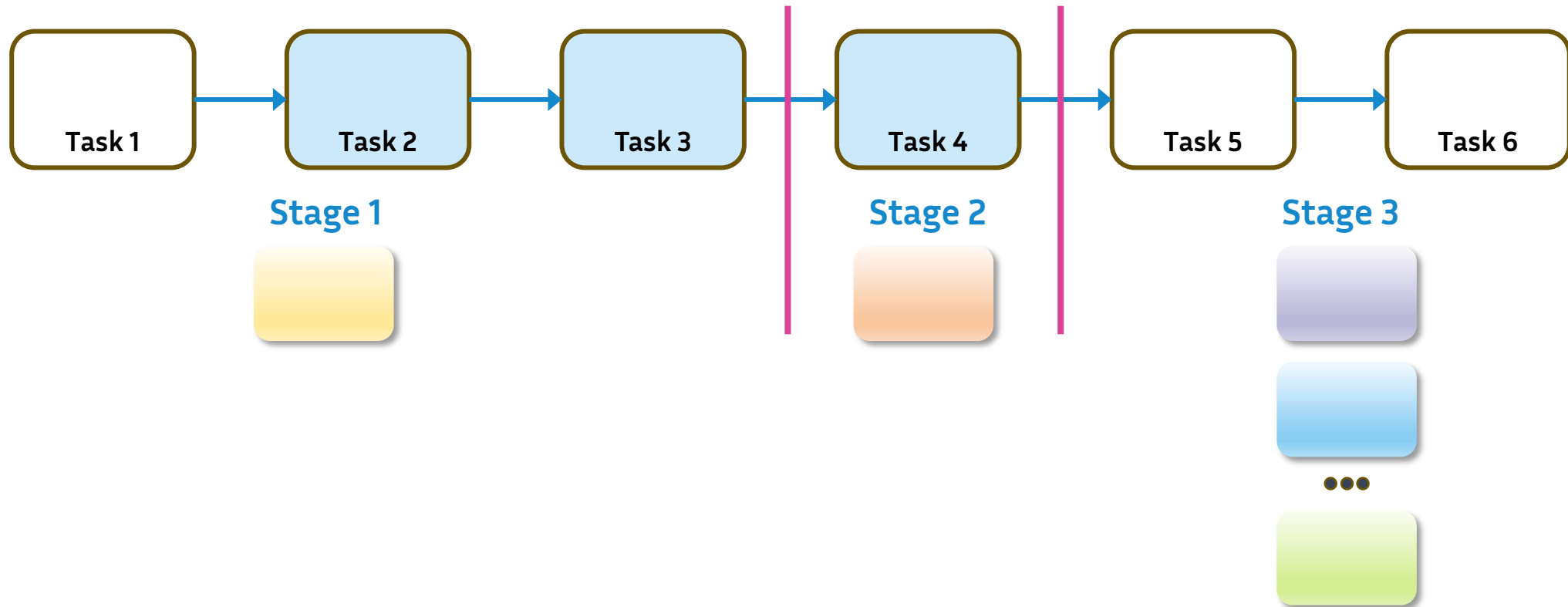
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



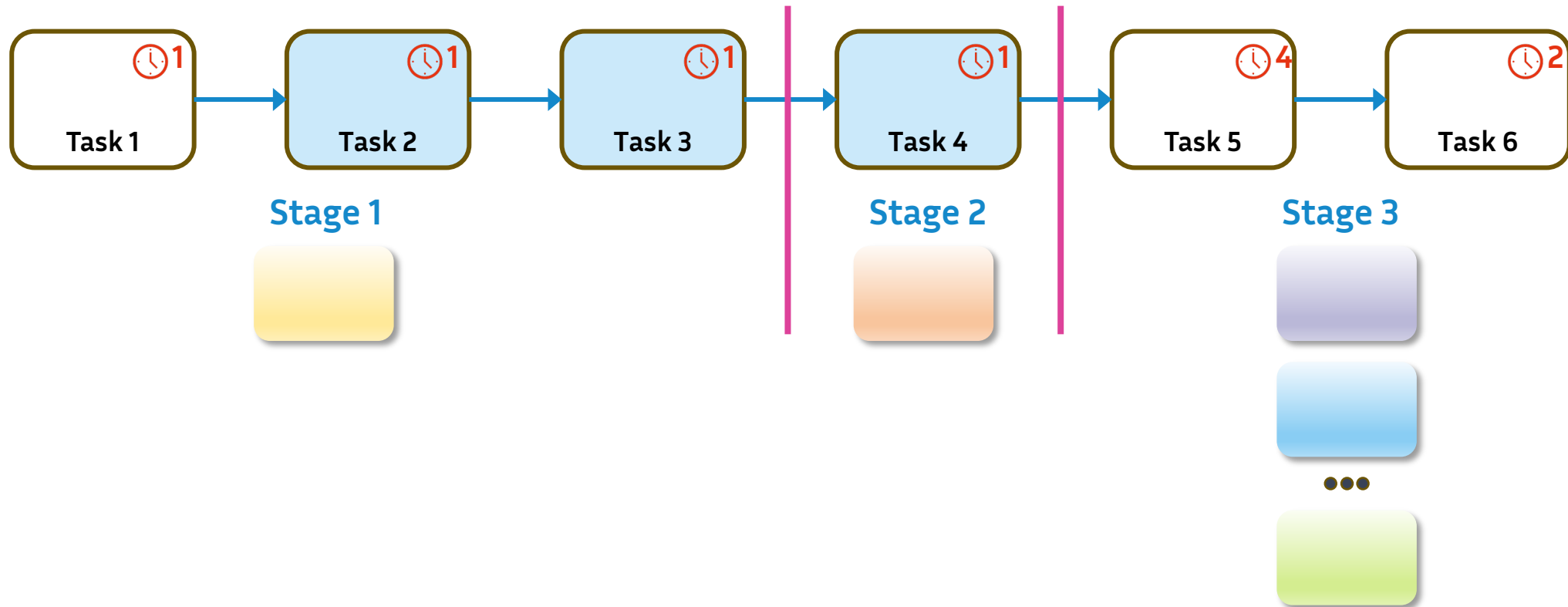
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



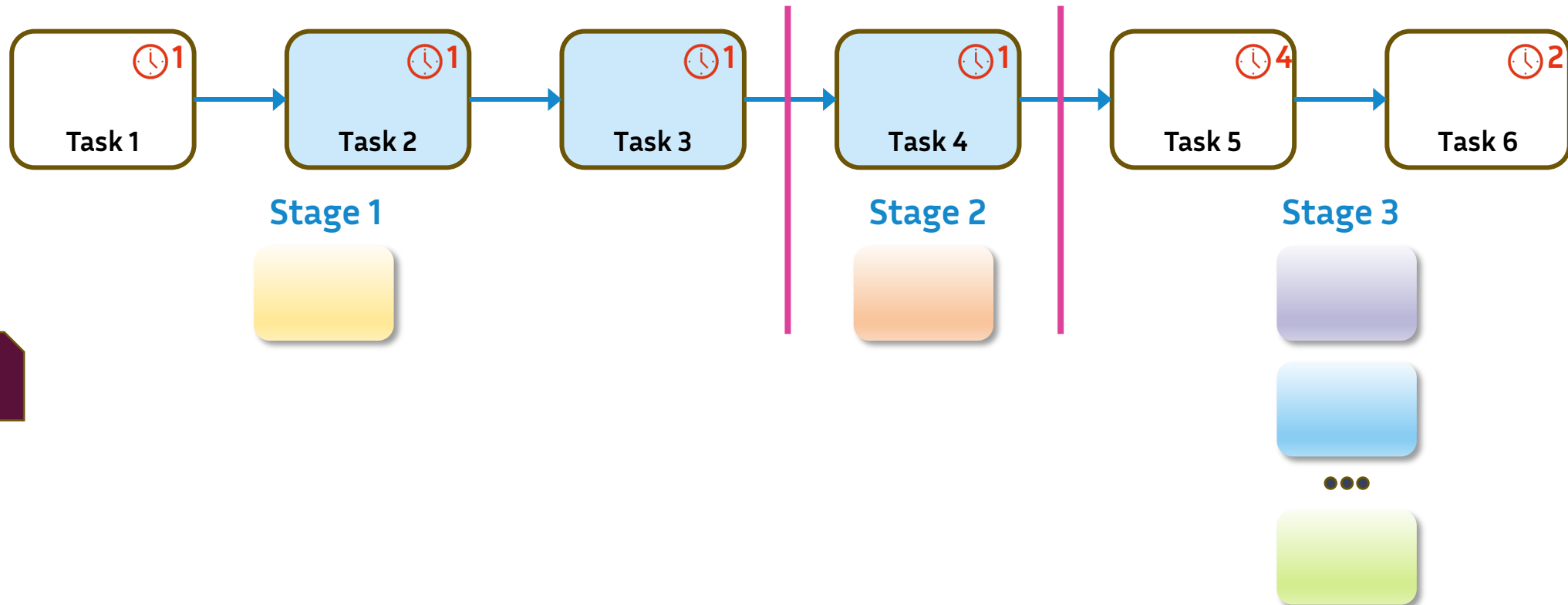
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



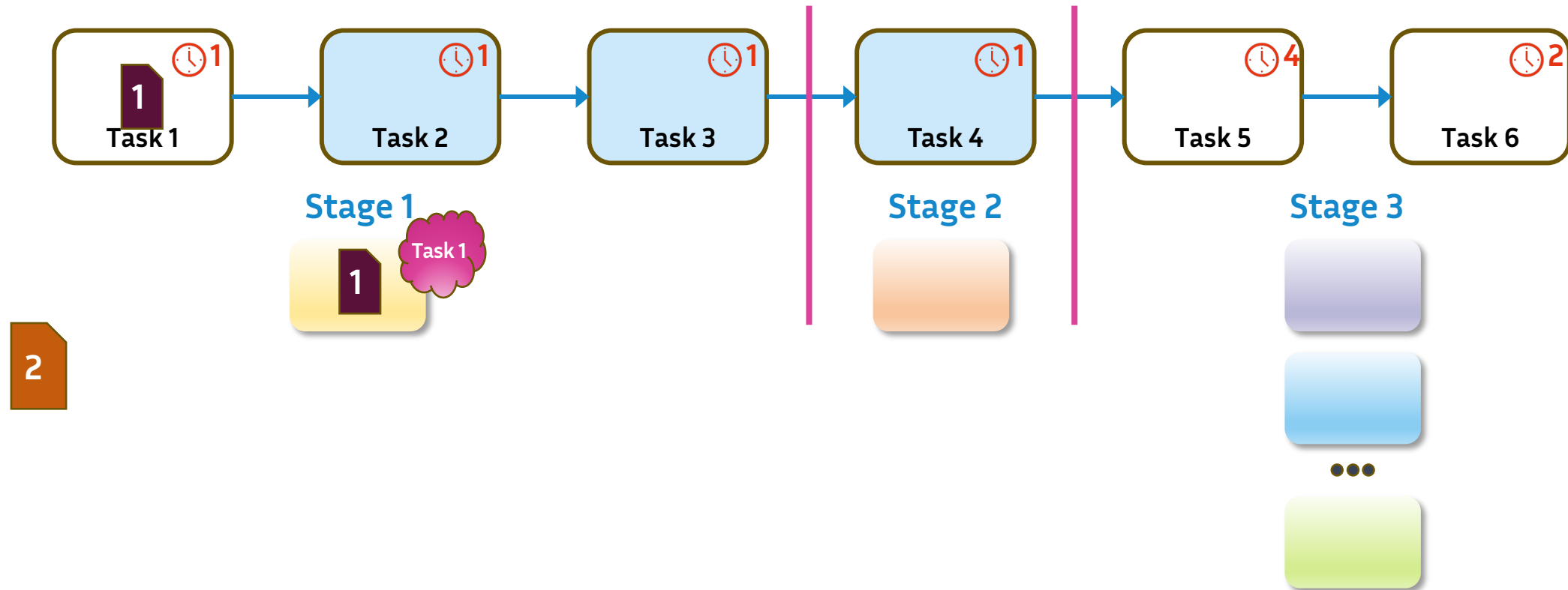
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



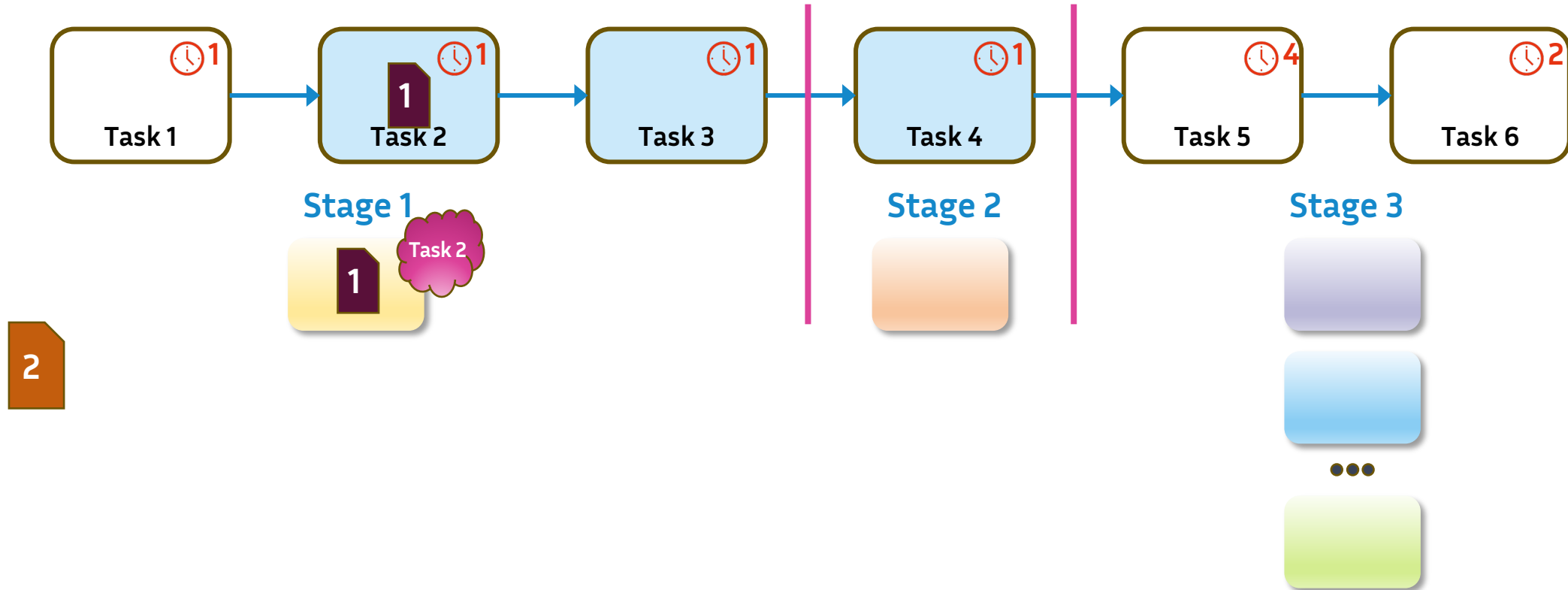
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



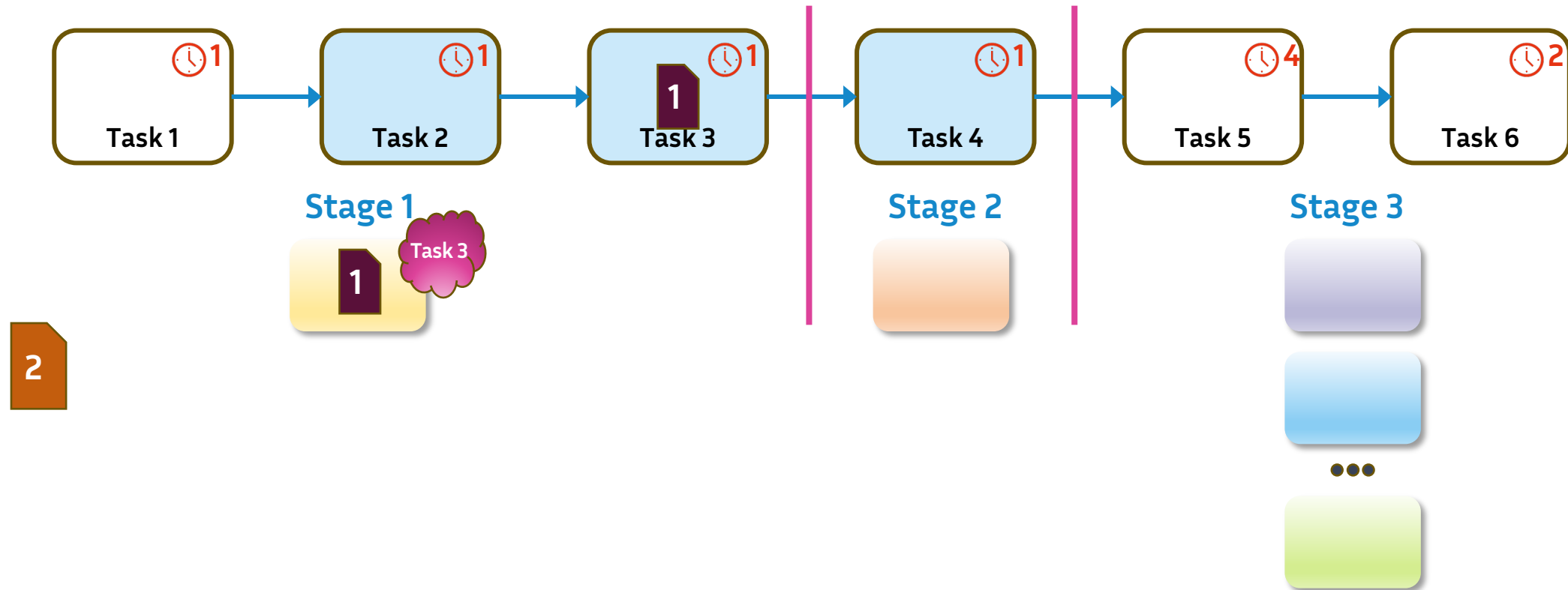
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



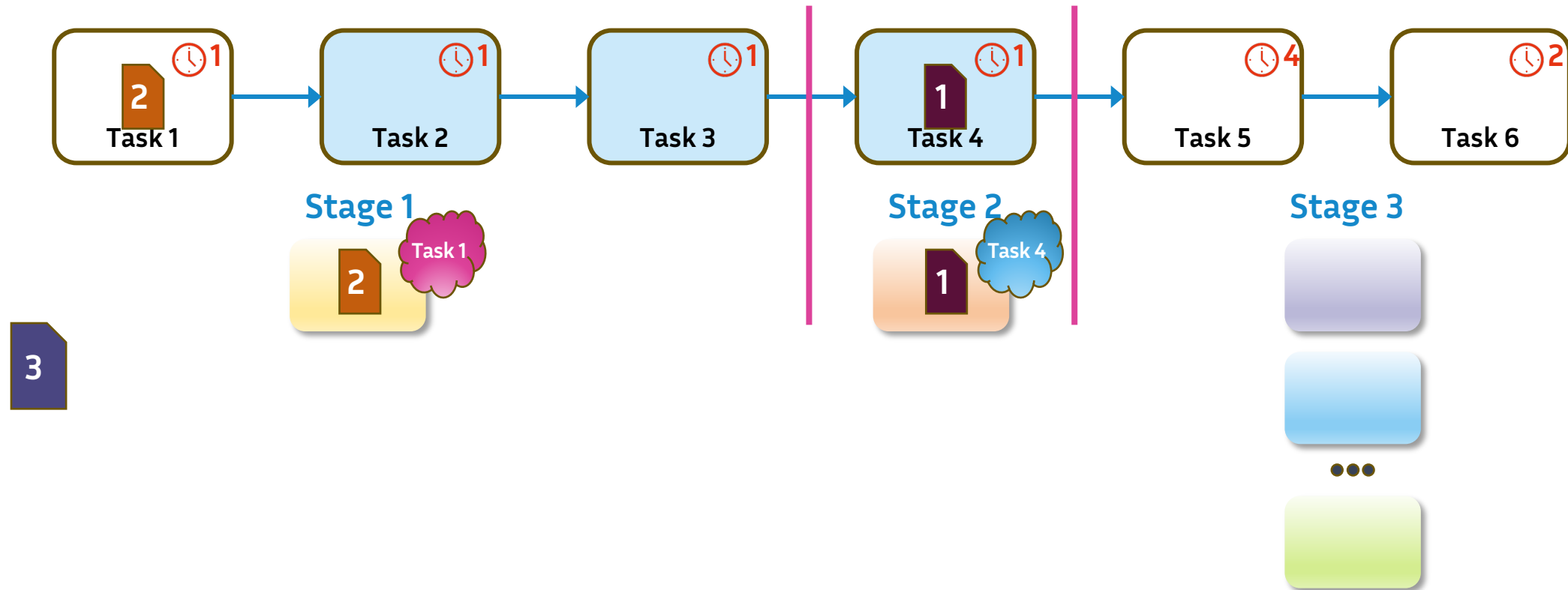
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



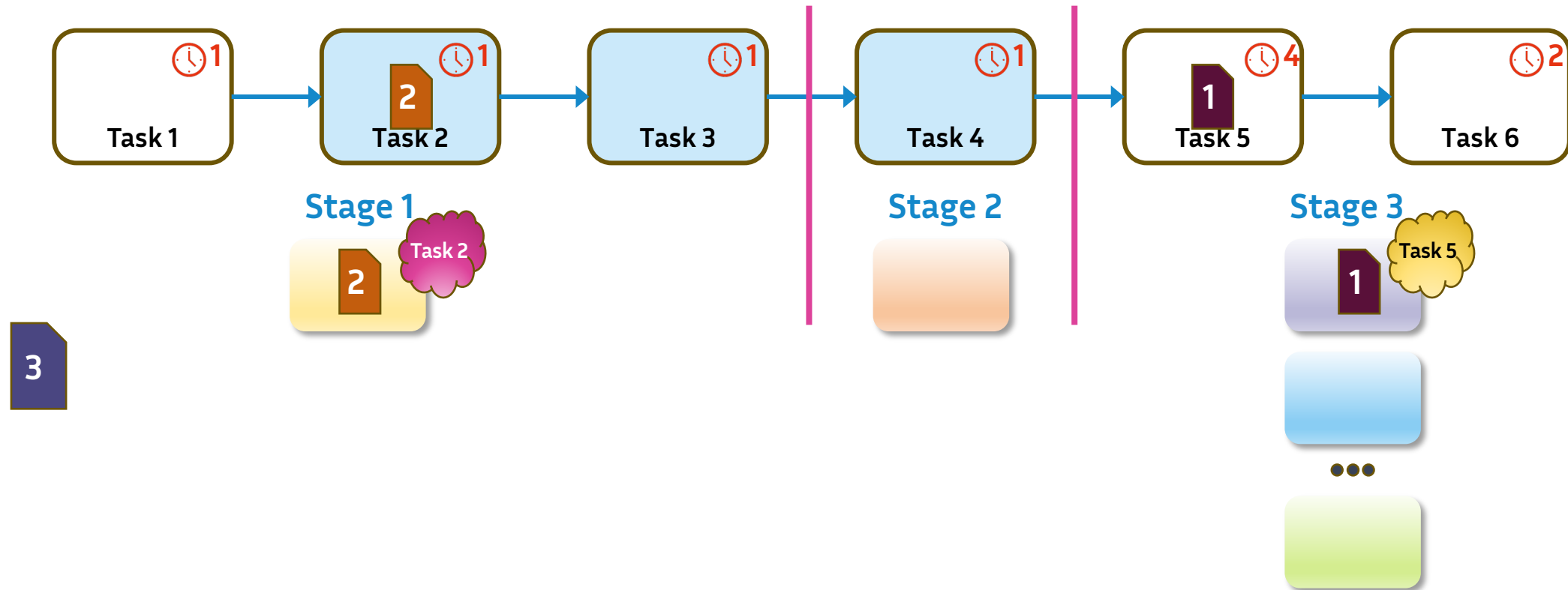
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



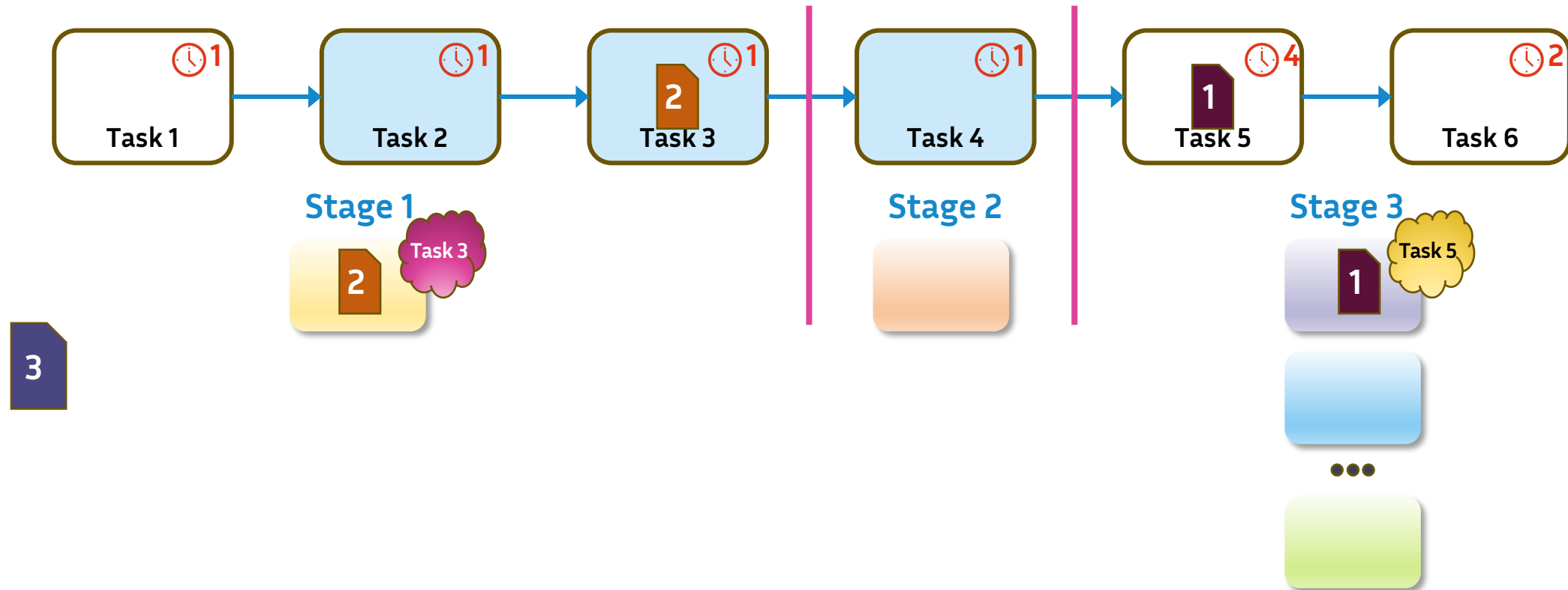
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



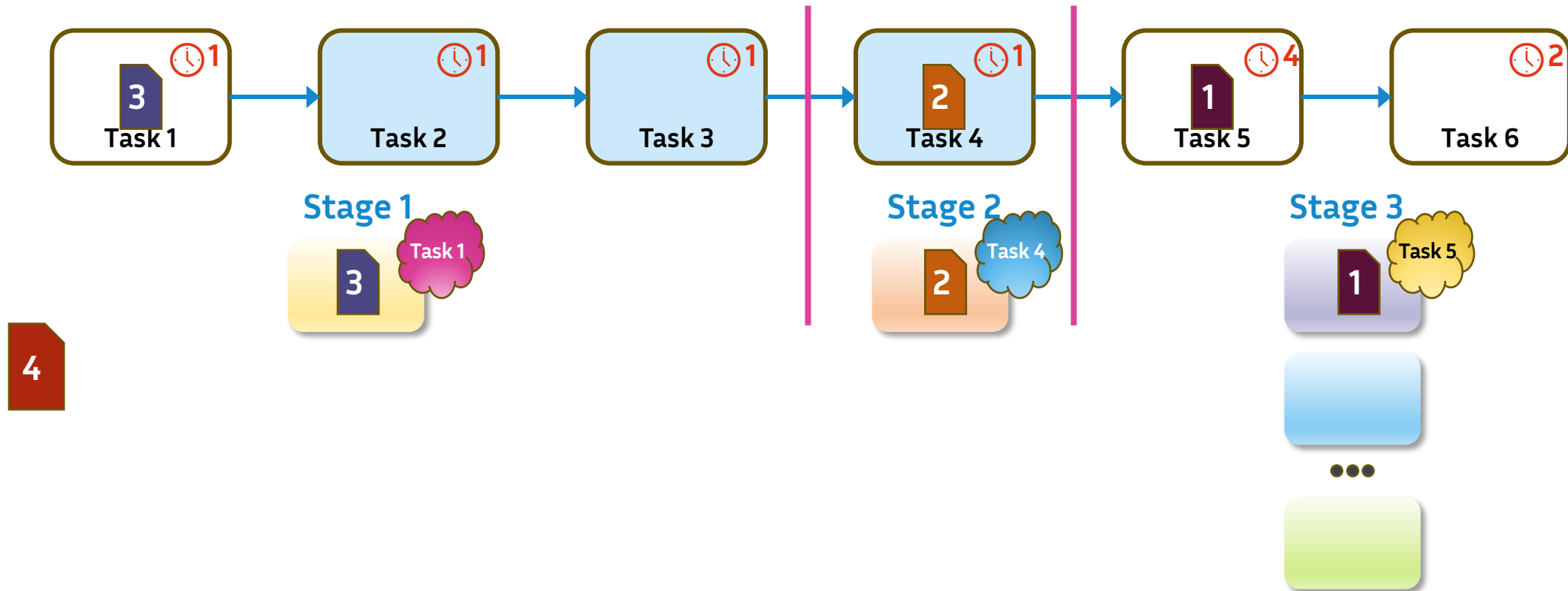
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

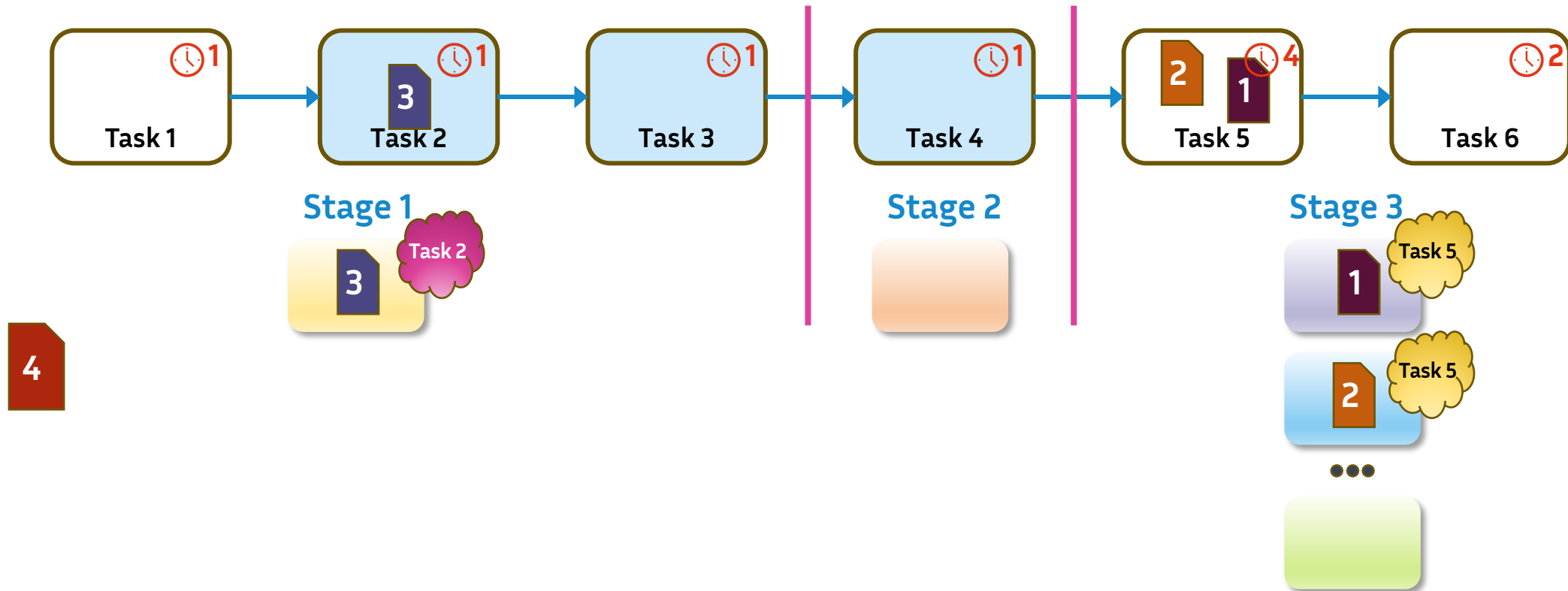
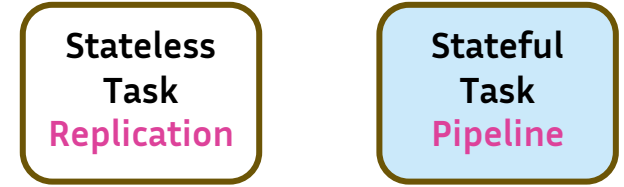
Stateful
Task
Pipeline



StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan



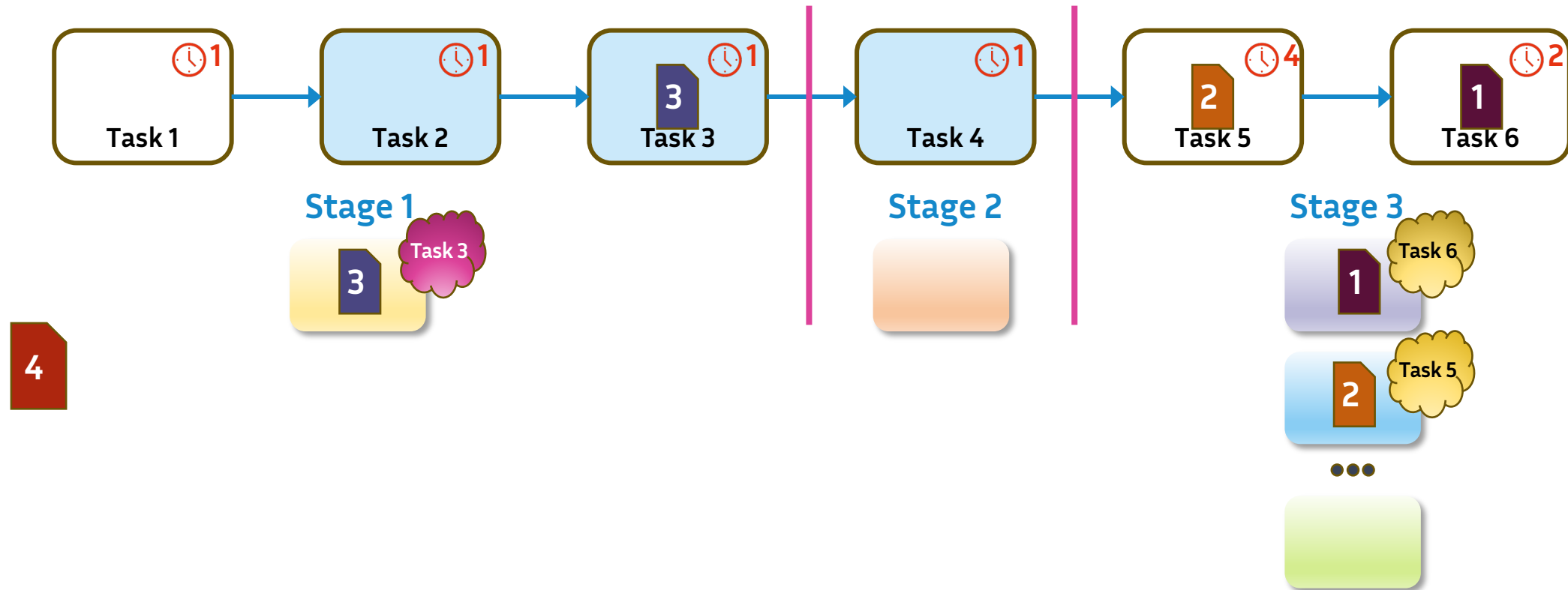
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



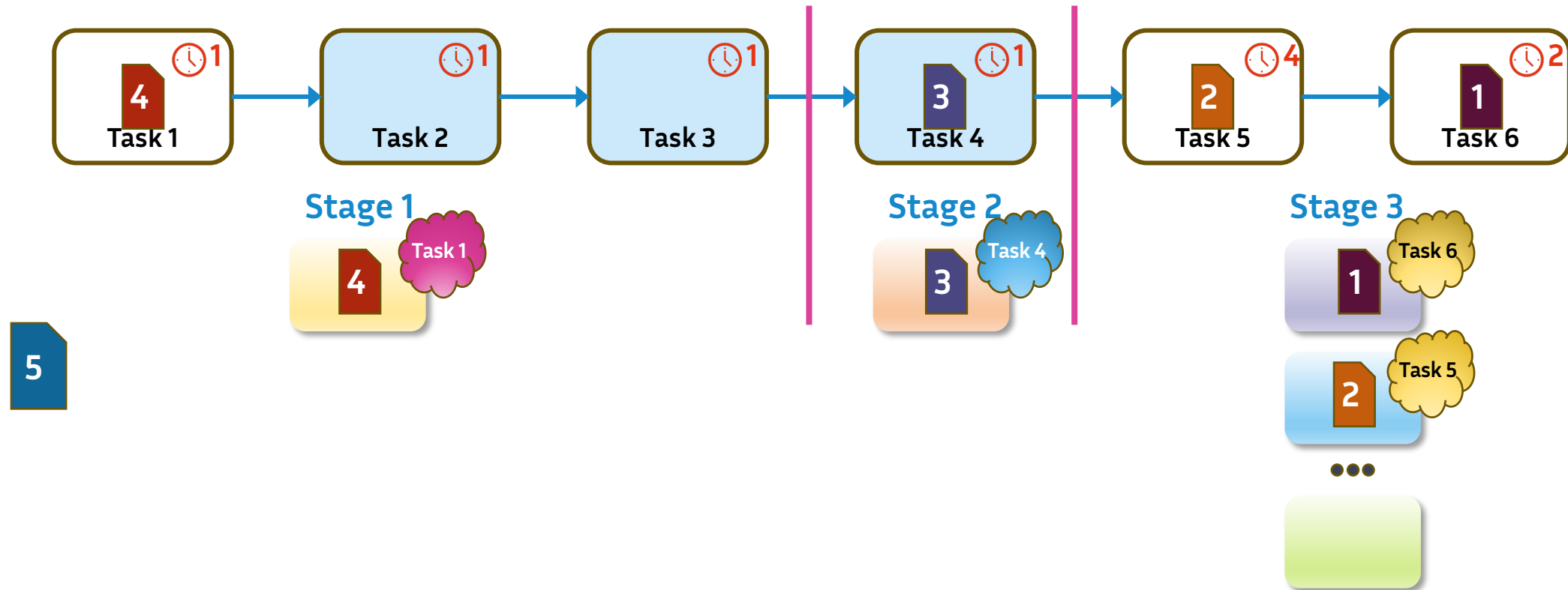
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



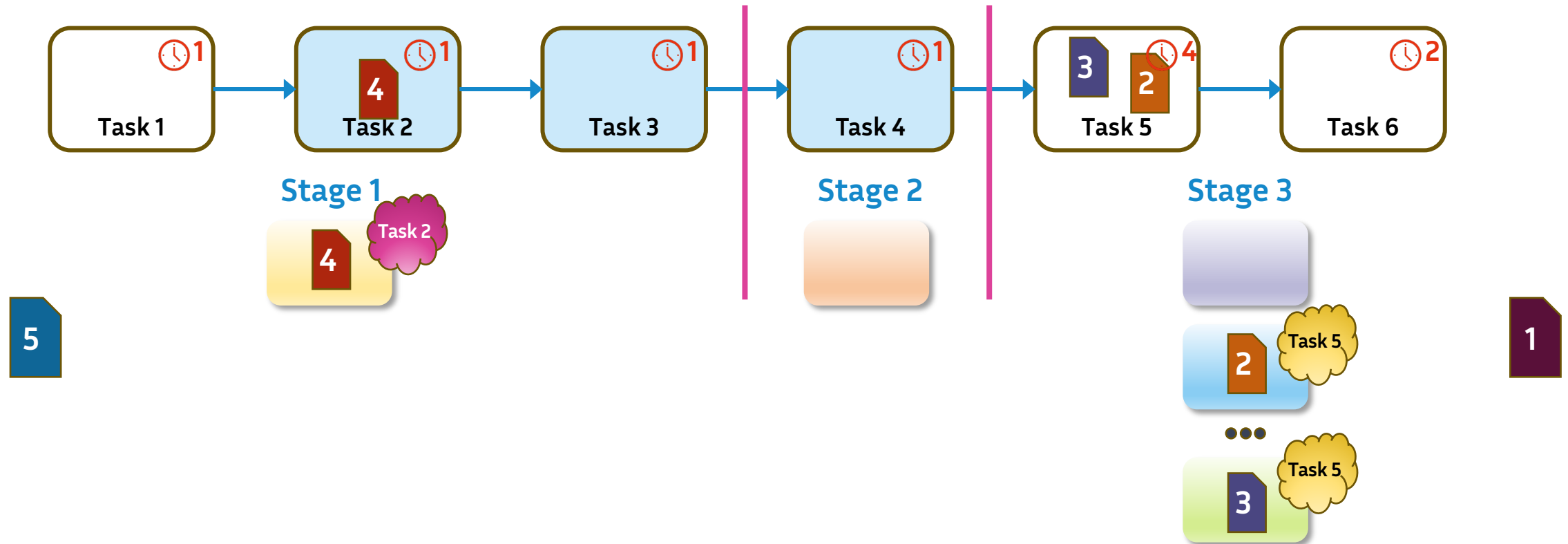
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



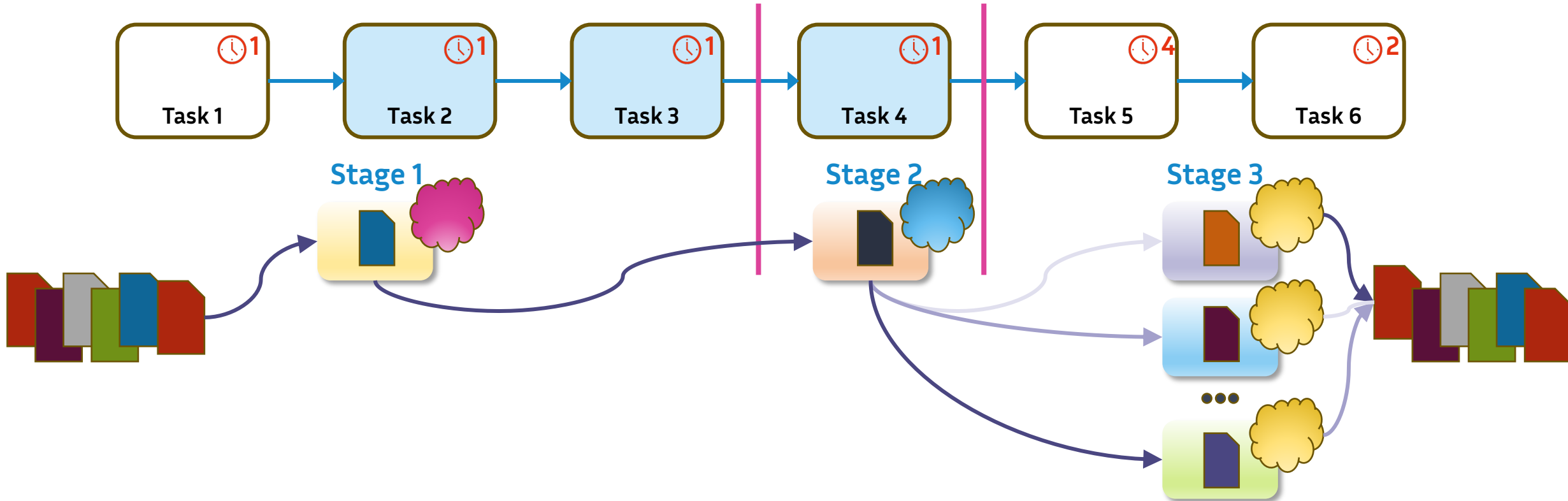
StreamPU – Multicore Parallelism

Pipeline Resource Mapping

PhD Thesis – Diane Orhan

Stateless
Task
Replication

Stateful
Task
Pipeline



StreamPU – Evaluation on a DVB-S2 Chain

Pipeline Resource Mapping: OTAC Algorithm / Optimal maximal-packing for TAsk Chains

PhD Thesis – Diane Orhan

Goals

1. Maximize the throughput, i.e. minimize the largest stage pipeline weight automatically
2. Minimize the number of used resources

Hypotheses

- Identical and homogeneous resources
- Synchronization or data movement overhead negligible

OTAC: Optimal Scheduling for Pipelined and Replicated Task Chains for Software-Defined Radio

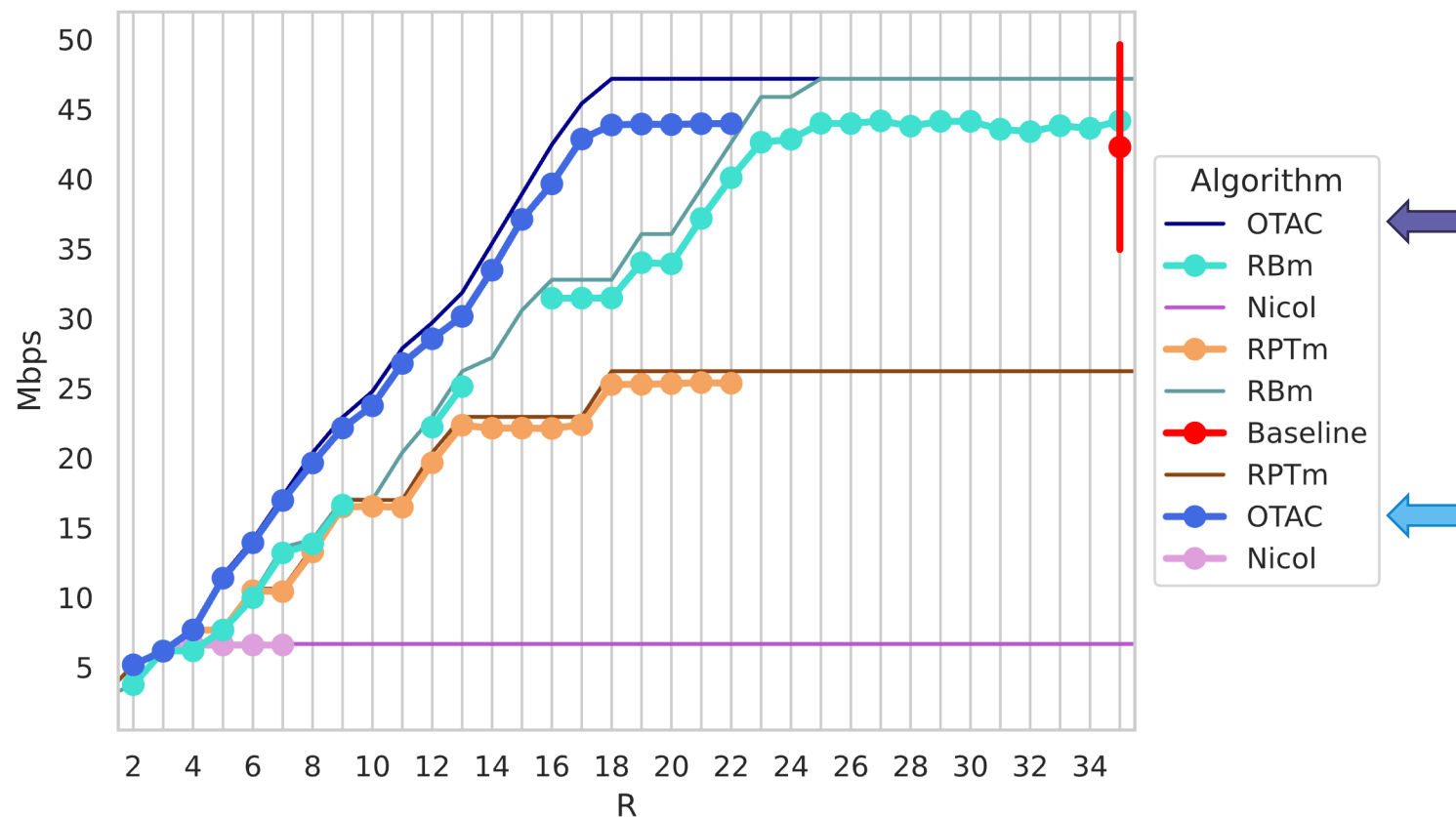
Orhan, Lima Pilla, Barthou, Cassagne, Aumage, Tajan, Jego, Leroux

<https://hal.science/hal-04228117>

StreamPU – Evaluation on a DVB-S2 Chain

Pipeline Resource Mapping: OTAC Algorithm / Optimal maximal-packing for TAsk Chains

PhD Thesis – Diane Orhan



	Simulated T.	Avg T.	Res.
OTAC	47.17	43.86	18
RPTm	26.21	25.26	18
RBm	47.17	43.98	25
Nicol	6.66	6.58	4
Baseline	47.17	42.3	35

- Simulated and average measured throughput
- Number of resources needed for the best results achieved

Target architecture: 2x (Intel® Xeon™ Platinum 8168 CPU)

5.

Example with the DVB-S2 Chain

AFF3CT Example – DVB-S2 Decoding

Industrial Project

Airbus D&S and IMS Laboratory

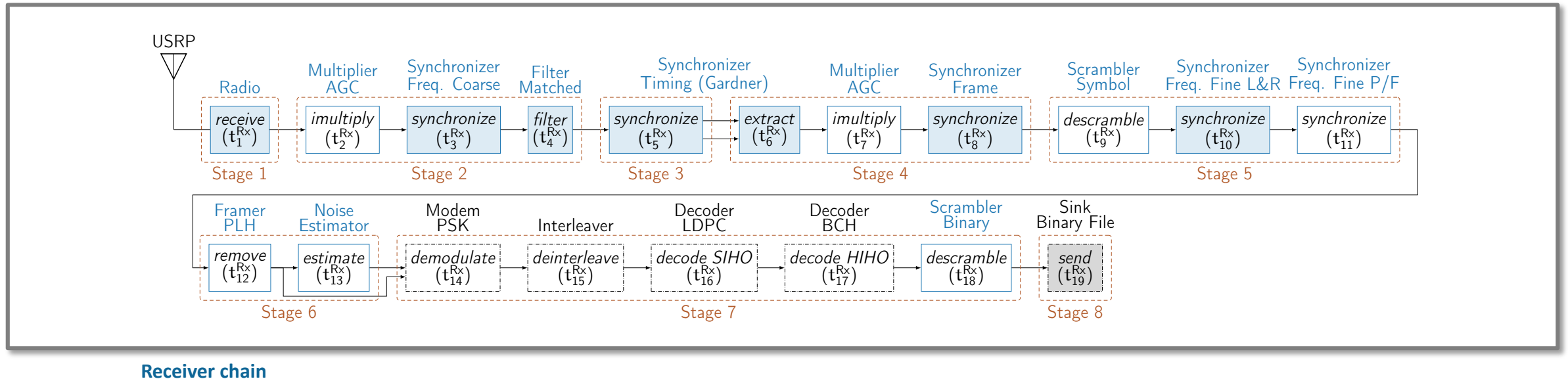
- Real-time DVB-S2 sender / receiver using AFF3CT

AFF3CT Example – DVB-S2 Decoding

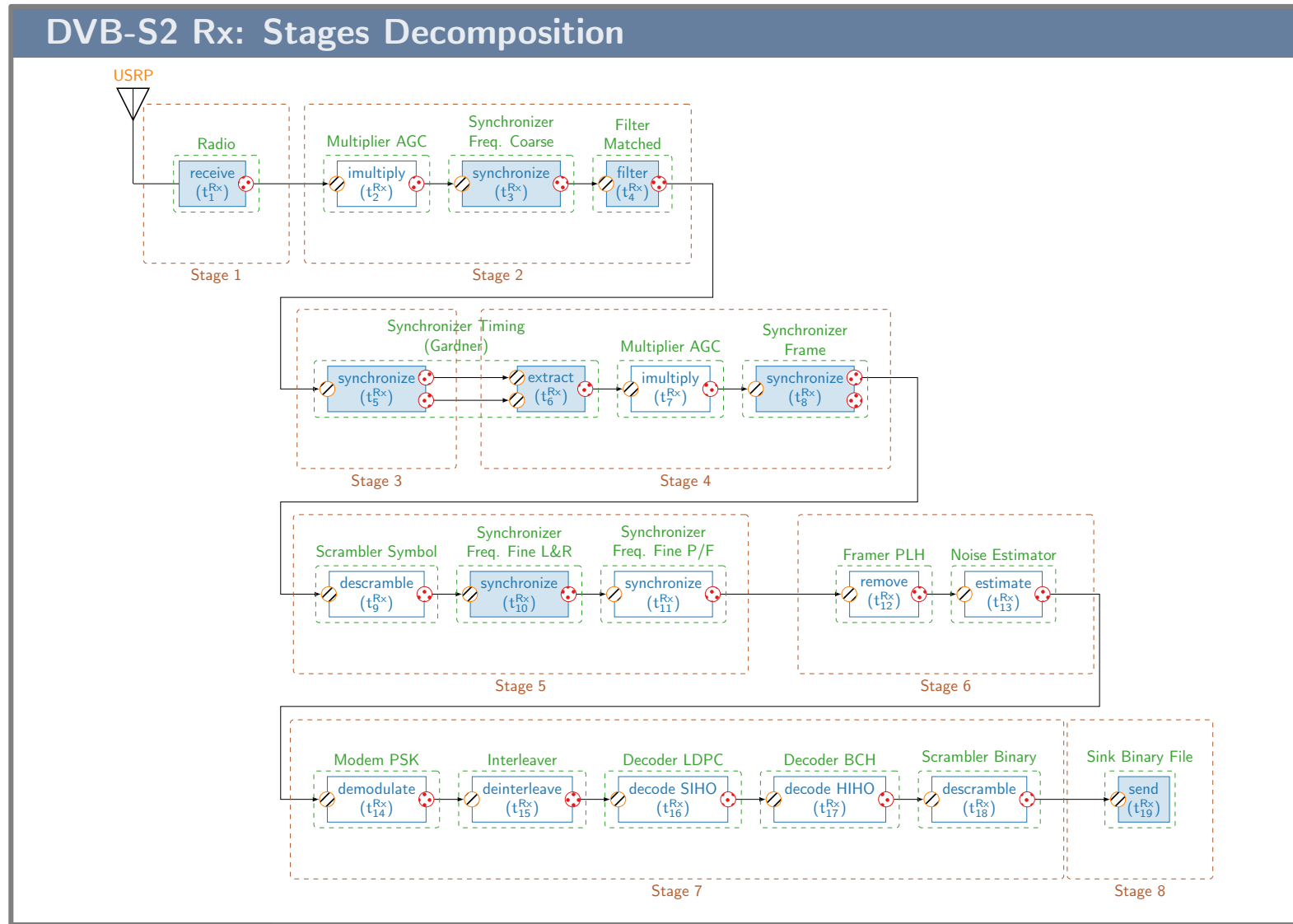
Industrial Project

Airbus D&S and IMS Laboratory

- Real-time DVB-S2 sender / receiver using AFF3CT

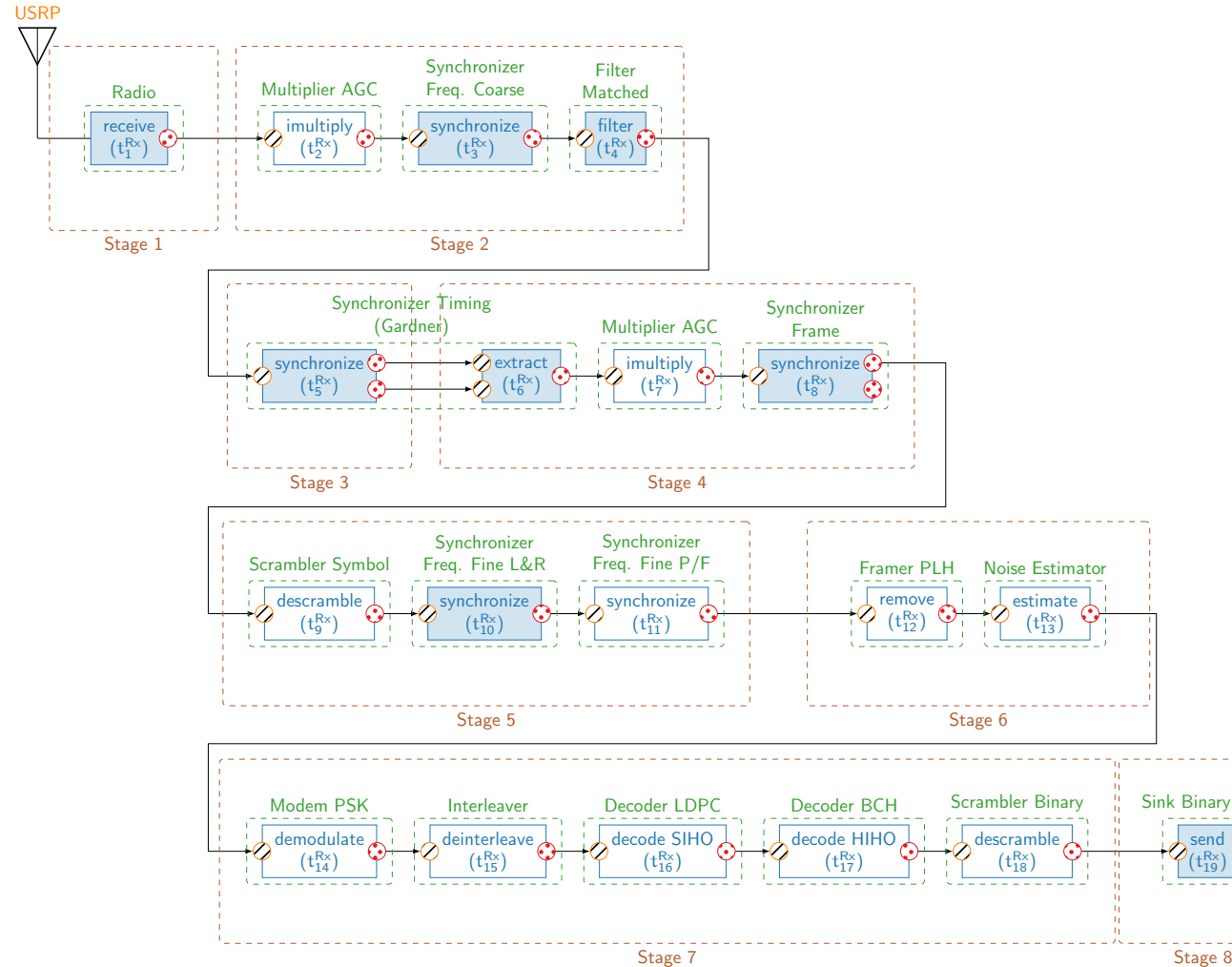


AFF3CT Example – DVB-S2 Decoding



AFF3CT Example – DVB-S2 Decoding

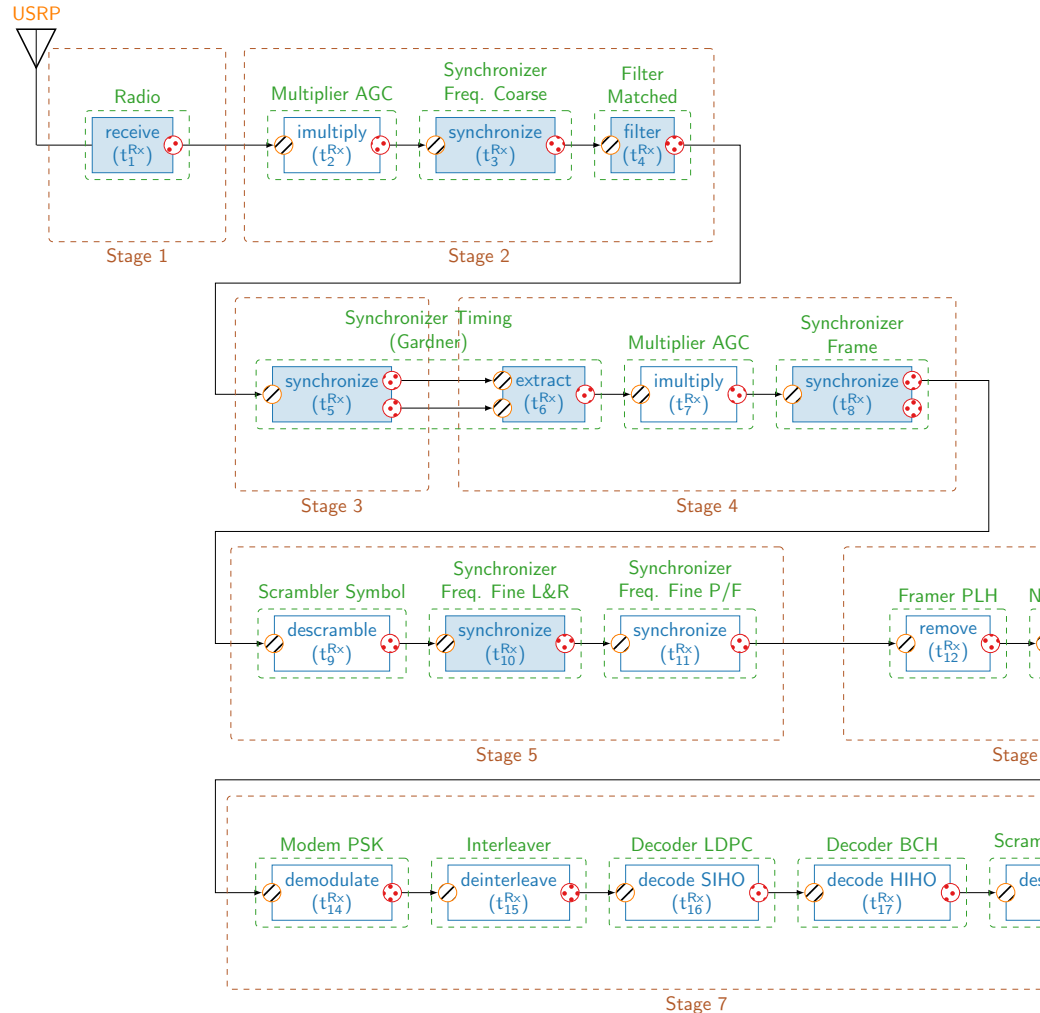
DVB-S2 Rx: Stages Decomposition



Stages and Tasks	Throughput (Mb/s)	Latency (μ s)	Time (%)
Radio - receive (t_1^{Rx})	431.83	527.32	0.94
Stage 1	431.83	527.32	0.94
Multiplier AGC - multiply (t_2^{Rx})	367.45	619.71	1.11
Synch. Freq. Coarse - synchronize (t_3^{Rx})	841.32	270.66	0.48
Filter Matched - filter (t_4^{Rx})	116.41	1956.08	3.49
Stage 2	80.00	2846.45	5.08
Synch. Timing - synchronize (t_5^{Rx})	55.42	4108.52	7.34
Stage 3	55.42	4108.52	7.34
Synch. Timing - extract (t_6^{Rx})	281.83	807.97	1.44
Multiplier AGC - multiply (t_7^{Rx})	685.51	332.18	0.59
Synch. Frame - synchronize (t_8^{Rx})	159.41	1428.51	2.55
Stage 4	88.65	2568.66	4.58
Scrambler Symbol - descramble (t_9^{Rx})	1682.89	135.31	0.24
Synch. Freq. Fine L&R - synchronize (t_{10}^{Rx})	1246.85	182.63	0.33
Synch. Freq. Fine P/F - synchronize (t_{11}^{Rx})	112.56	2022.98	3.61
Stage 5	97.27	2340.92	4.18
Framer PLH - remove (t_{12}^{Rx})	1008.60	225.77	0.40
Noise Estimator - estimate (t_{13}^{Rx})	550.06	413.98	0.74
Stage 6	355.94	639.75	1.14
Modem PSK - demodulate (t_{14}^{Rx})	40.47	5626.34	10.05
Interleaver - deinterleave (t_{15}^{Rx})	1347.25	169.02	0.30
Decoder LDPC - decode SIHO (t_{16}^{Rx})	164.21	1386.74	2.48
Decoder BCH - decode HIHO (t_{17}^{Rx})	6.92	32905.37	58.79
Scrambler Binary - descramble (t_{18}^{Rx})	91.11	2499.41	4.47
Stage 7	5.35	42586.88	76.09
Sink Binary File - send (t_{19}^{Rx})	1838.31	123.87	0.22
Stage 8	1838.31	123.87	0.22
Total	4.09	55742.37	99.57

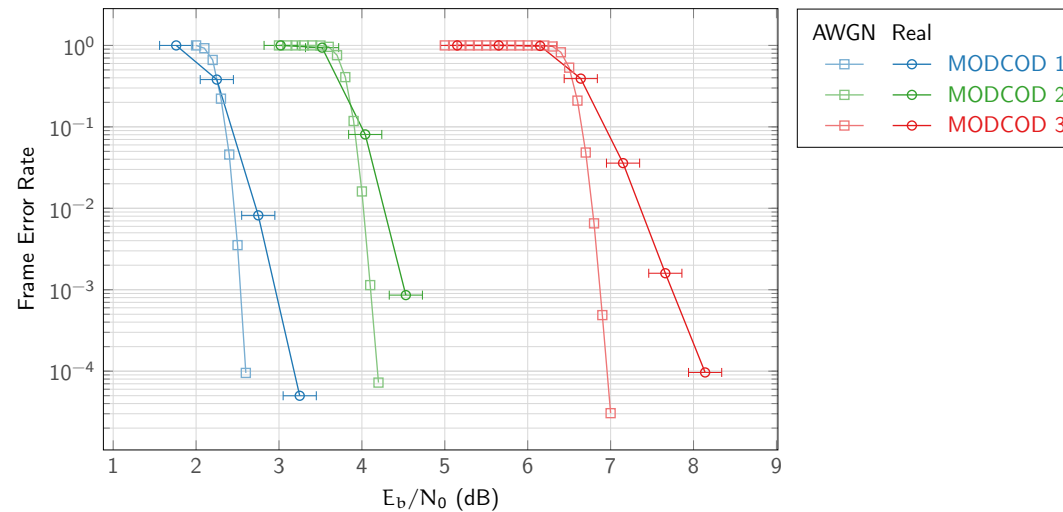
AFF3CT Example – DVB-S2 Decoding

DVB-S2 Rx: Stages Decomposition



Stages and Tasks	Throughput (Mb/s)	Latency (μ s)	Time (%)
Synch. Timing - synchronize (t_5^{Rx})	55.42	4108.52	7.34
Stage 3	55.42	4108.52	7.34
Synch. Timing - extract (t_6^{Rx})	281.83	807.97	1.44
Multiplier AGC - multiply (t_7^{Rx})	685.51	332.18	0.59
Synch. Frame - synchronize (t_8^{Rx})	159.41	1428.51	2.55
Stage 4	88.65	2568.66	4.58
Scrambler Symbol - descramble (t_9^{Rx})	1682.89	135.31	0.24
Synch. Freq. Fine L&R - synchronize (t_{10}^{Rx})	1246.85	182.63	0.33
Synch. Freq. Fine P/F - synchronize (t_{11}^{Rx})	112.56	2022.98	3.61
Stage 5	97.27	2340.92	4.18
Framer PLH - remove (t_{12}^{Rx})	1008.60	225.77	0.40
Noise Estimator - estimate (t_{13}^{Rx})	550.06	413.98	0.74
Stage 6	355.94	639.75	1.14
Modem PSK - demodulate (t_{14}^{Rx})	40.47	5626.34	10.05
Interleaver - deinterleave (t_{15}^{Rx})	1347.25	169.02	0.30
Decoder LDPC - decode SIHO (t_{16}^{Rx})	164.21	1386.74	2.48
Decoder BCH - decode HIHO (t_{17}^{Rx})	6.92	32905.37	58.79
Scrambler Binary - descramble (t_{18}^{Rx})	91.11	2499.41	4.47
Stage 7	5.35	42586.88	76.09
Sink Binary File - send (t_{19}^{Rx})	1838.31	123.87	0.22
Stage 8	1838.31	123.87	0.22
Total	4.09	55742.37	99.57

DVB-S2 Rx: Decoding Performances & Throughputs



Configuration	Throughput (Mb/s)				Latency (ms)	
	Sequential		Parallel			
	x86	ARM	x86	ARM	x86	ARM
MODCOD 1	3.4	1.0	37	19	–	37
MODCOD 2	4.1	1.4	55	28	56	41
MODCOD 3	4.0	1.1	80	42	–	51

- x86: 2× Intel[®] Xeon[™] Platinum 8168 CPUs, 24 cores @ 2.7 GHz
- ARM: 2× Cavium ThunderX2[®] CN9975 CPUs, 28 cores @ 2.0 GHz

6.

Targeting Heterogeneous Multicores

StreamPU – Heterogeneous Processors

Performance vs power efficiency cores

Yacine Idouar

Adrien Cassagne

Julien Sopena

StreamPU – Heterogeneous Processors

Performance vs power efficiency cores

Yacine Idouar
Adrien Cassagne
Julien Sopena

Targeted System: Apple M1 Ultra



- ▶ **20 CPU cores**
 - ▷ 4 clusters of 4 p-cores @ 3.230 GHz
 - ▷ 2 clusters of 2 e-cores @ 2.064 GHz
 - ▷ ARMv8.5-A ISA with 128-bit NEON SIMD
- ▶ 5 nm TSMC chiplet design
 - ▷ Fusion of two M1 Max
- ▶ RAM bandwidth: 800 GB/s
- ▶ Fedora Asahi Linux OS
 - ▷ Kernel 6.6
 - ▷ **Thread pinning***

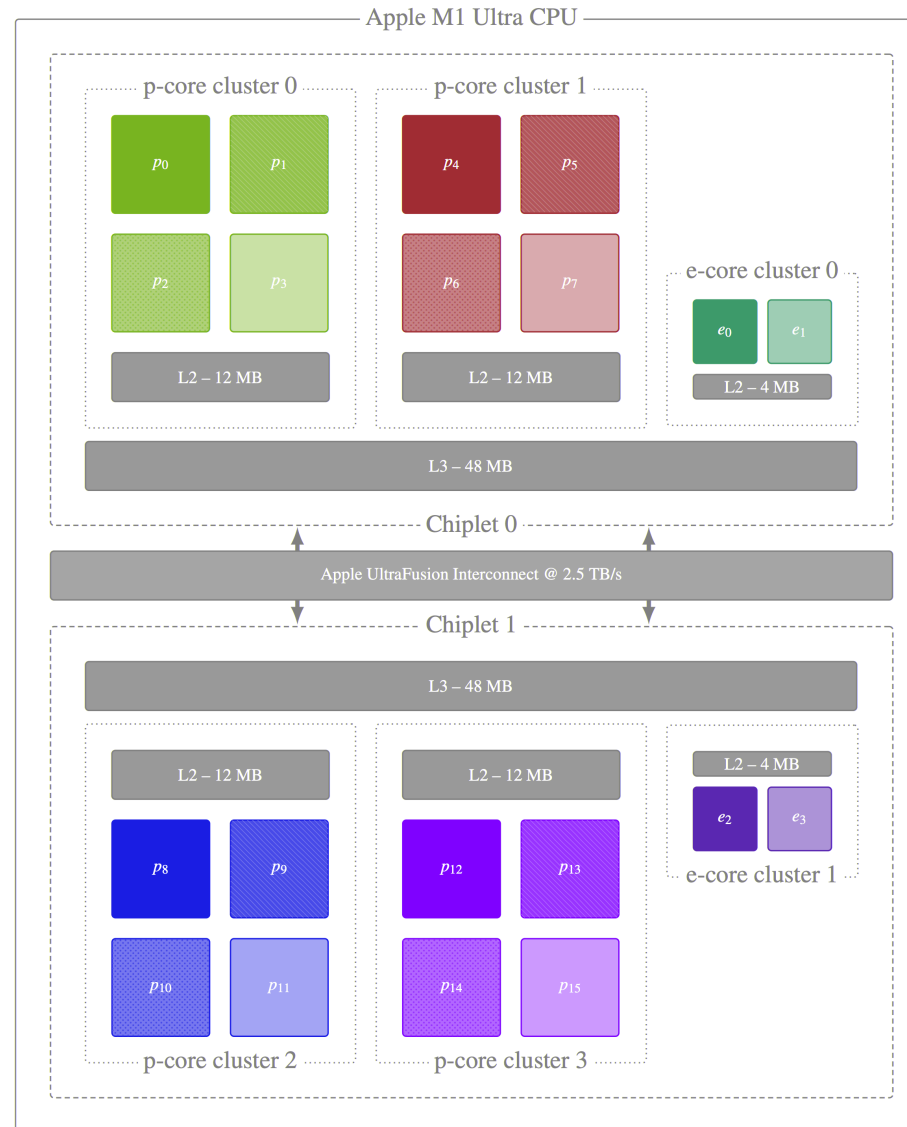
*: Running Linux is required as macOS does not provide a working thread pinning mechanism

the app

3%
10%
1.5%
20%

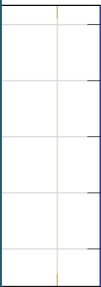


Targeted System: Apple M1 Ultra



- ▶ **20 CPU cores**
 - ▷ 4 clusters of 4 @ 3.230 GHz
 - ▷ 2 clusters of 2 @ 2.064 GHz
 - ▷ ARMv8.5-A IS 128-bit NEON
- ▶ 5 nm TSMC chip
 - ▷ Fusion of two
- ▶ RAM bandwidth:
- ▶ Fedora Asahi Lin
 - ▷ Kernel 6.6
 - ▷ **Thread pinni**

*: Running Linux is required as macOS does not provide a working thread pinning mechanism



e N .

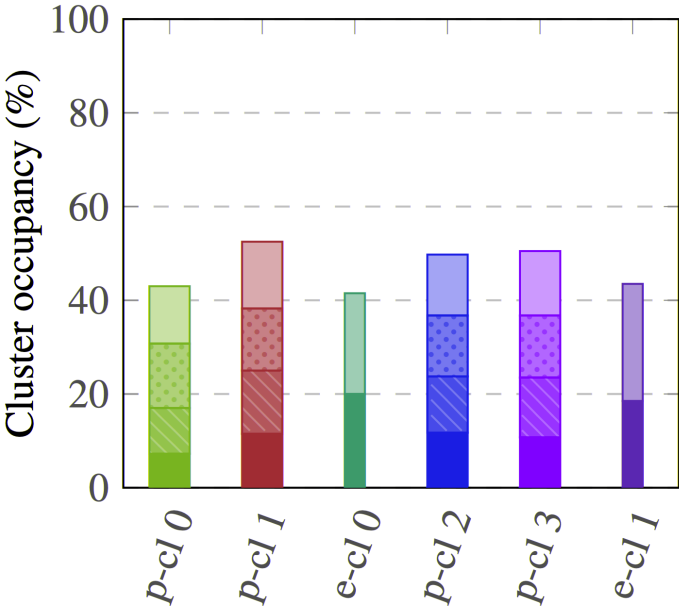
ad pinning
ations

ality

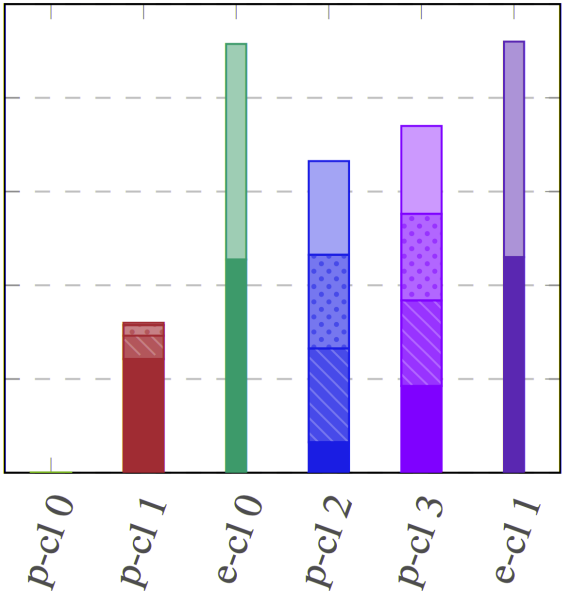
ft idle)

Policies

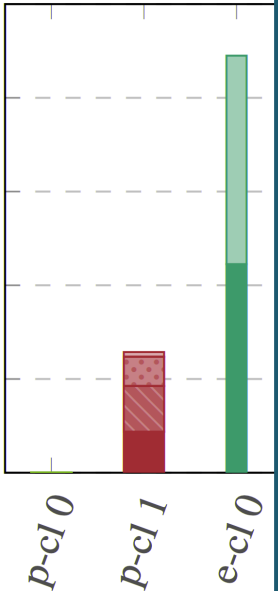
ay or may
ry policy)



S0: OS scheduling



S1: Pinning for throughput



S2: Pinning for energy efficiency

Occupancy of the M1 Ultra CPU clusters depending on three different thread scheduling strategies. S0: Linux 6.6 scheduler. S1: Manual thread pinning to maximize throughput. S2: Manual thread pinning to minimize the app energy consumption.

Strategy	Throughput (Mb/s)	Power (W)	Energy/fragment (mJ)
S0	54.5	32	8.0
S1	56.0	30	7.3
S2	53.6	26	6.6

Compared to S0 strategy

- S1: Throughput gain & Energy efficiency
- S2: Throughput gain & Energy efficiency

7.

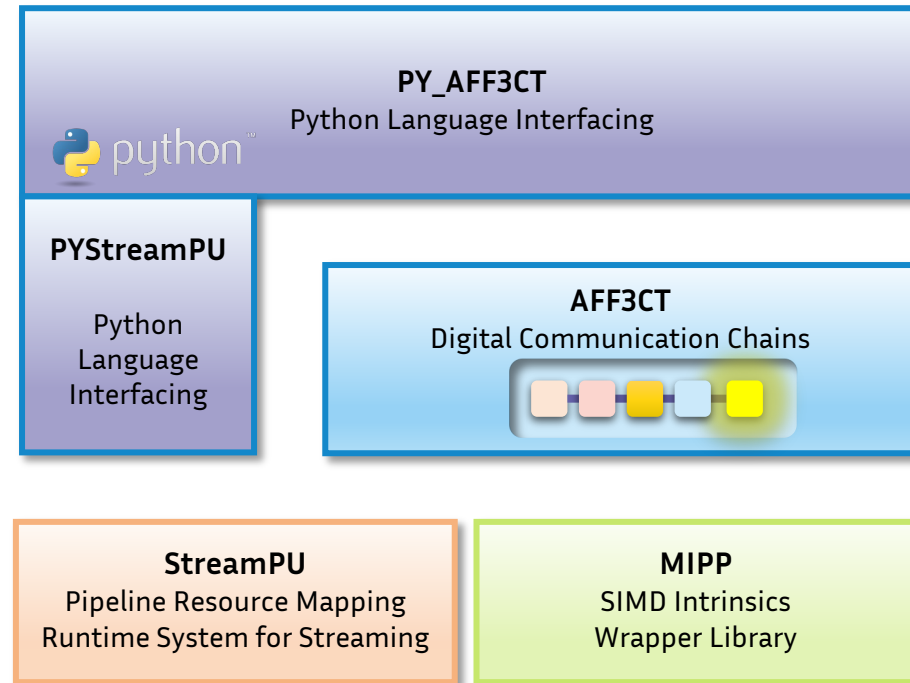
High-level Programming with PY-AFF3CT



Python Programming Interface

High Level Programming

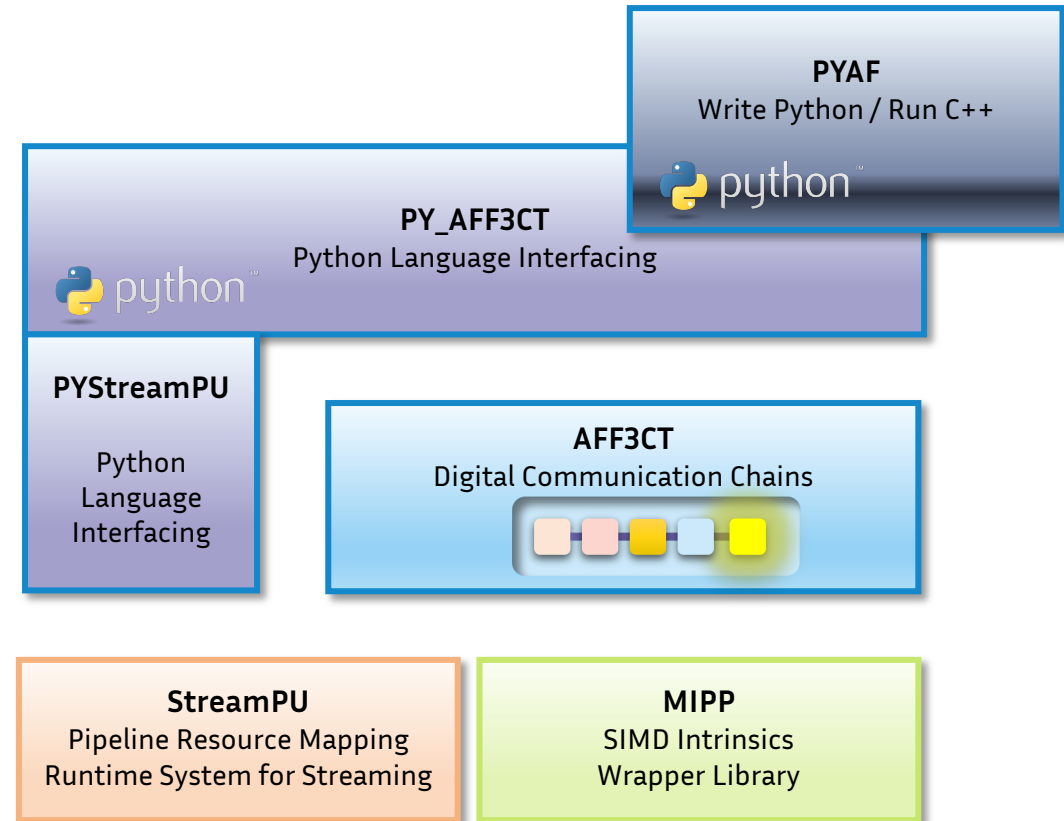
- **PY_AFF3CT**
 - Python access to AFF3CT modules
 - Based on pybind11



Python Programming Interface

High Level Programming

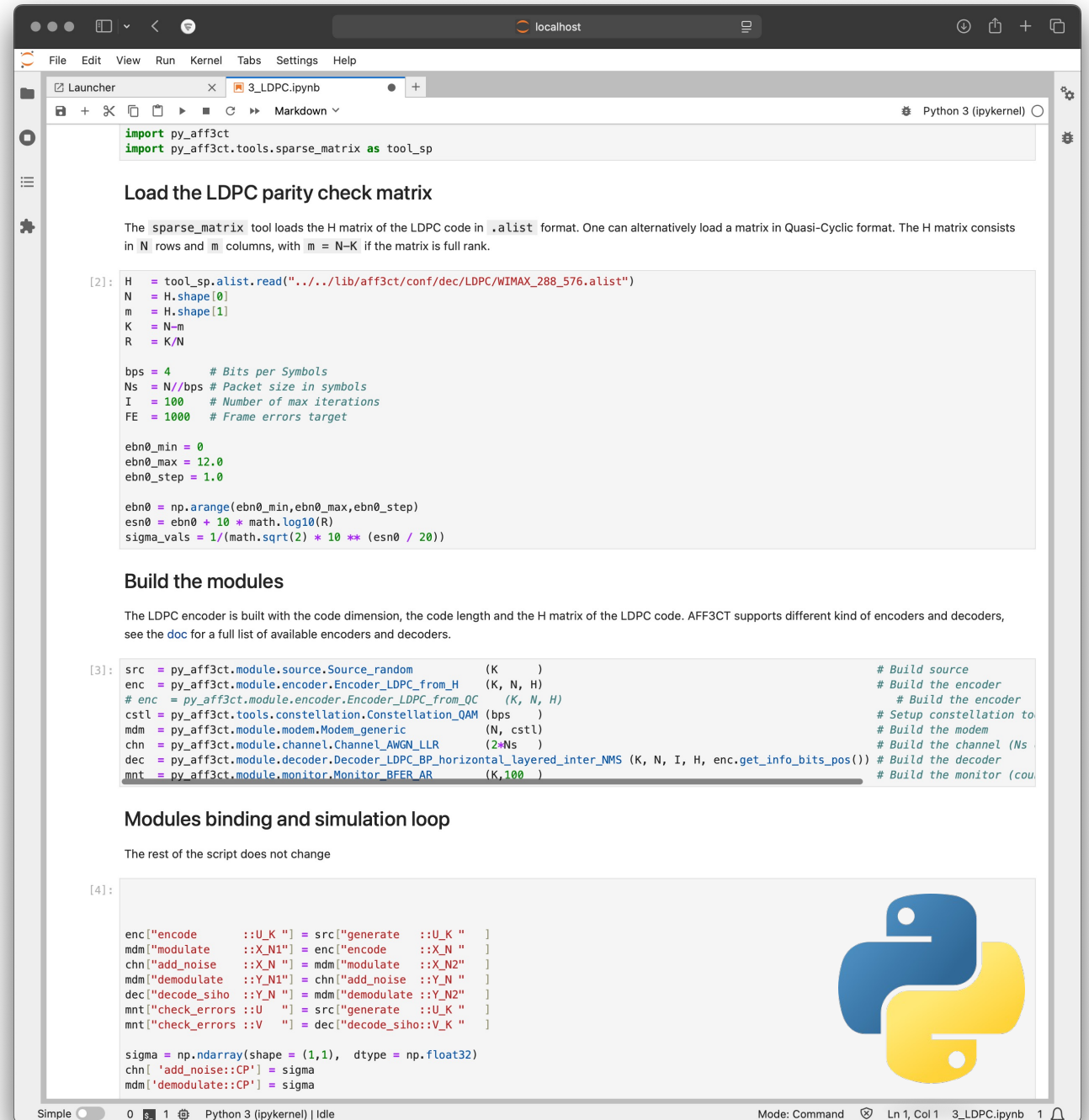
- **PY_AFF3CT**
 - Python access to AFF3CT modules
 - Based on pybind11
- **PYAF**
 - Insert custom C++ modules in PY_AFF3CT chains



Python Programming

High Level Programming

- PY_AFF3CT
 - Python access to AFF3CT modules
 - Based on pybind11
- PYAF
 - Insert custom C++ modules in PY_AFF3CT chains



The screenshot shows a Jupyter Notebook window titled '3_LDPC.ipynb' running on a 'Python 3 (ipykernel)' environment. The notebook contains two code cells. The first cell, labeled '[2]:', imports the 'py_aff3ct' module and its 'tools.sparse_matrix' as 'tool_sp'. It then defines a function to load the LDPC parity check matrix from a file named 'WIMAX_288_576 alist'. The function calculates various parameters like 'bps', 'Ns', 'I', 'FE', 'ebn0_min', 'ebn0_max', 'ebn0_step', 'ebn0', 'esn0', and 'sigma_vals'. The second cell, labeled '[3]:', builds the modules by creating instances of 'Source_random', 'Encoder_LDPC_from_H', 'Constellation_QAM', 'Modem_generic', 'Channel_AWGN_LLRL', 'Decoder_LDPC_BP_horizontal_layered_inter_NMS', and 'Monitor_BFER_AR'. The notebook interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing 'Simple' mode, '0' lines, '1' column, and 'Python 3 (ipykernel) | Idle'.

```
import py_aff3ct
import py_aff3ct.tools.sparse_matrix as tool_sp

Load the LDPC parity check matrix

The sparse_matrix tool loads the H matrix of the LDPC code in .alist format. One can alternatively load a matrix in Quasi-Cyclic format. The H matrix consists in N rows and m columns, with m = N-K if the matrix is full rank.

[2]: H = tool_sp.alist.read("../lib/aff3ct/conf/dec/LDPC/WIMAX_288_576.alist")
      N = H.shape[0]
      m = H.shape[1]
      K = N-m
      R = K/N

      bps = 4 # Bits per Symbols
      Ns = N//bps # Packet size in symbols
      I = 100 # Number of max iterations
      FE = 1000 # Frame errors target

      ebn0_min = 0
      ebn0_max = 12.0
      ebn0_step = 1.0

      ebn0 = np.arange(ebn0_min, ebn0_max, ebn0_step)
      esn0 = ebn0 + 10 * math.log10(R)
      sigma_vals = 1/(math.sqrt(2) * 10 ** (esn0 / 20))

Build the modules

The LDPC encoder is built with the code dimension, the code length and the H matrix of the LDPC code. AFF3CT supports different kind of encoders and decoders, see the doc for a full list of available encoders and decoders.

[3]: src = py_aff3ct.module.source.Source_random (K) # Build source
      enc = py_aff3ct.module.encoder.Encoder_LDPC_from_H (K, N, H) # Build the encoder
      # enc = py_aff3ct.module.encoder.Encoder_LDPC_from_QC (K, N, H) # Build the encoder
      cstl = py_aff3ct.tools.constellation.Constellation_QAM (bps) # Setup constellation to
      mdm = py_aff3ct.module.modem.Modem_generic (N, cstl) # Build the modem
      chn = py_aff3ct.module.channel.Channel_AWGN_LLRL (2*Ns) # Build the channel (Ns
      dec = py_aff3ct.module.decoder.Decoder_LDPC_BP_horizontal_layered_inter_NMS (K, N, I, H, enc.get_info_bits_pos()) # Build the decoder
      mnt = py_aff3ct.module.monitor.Monitor_BFER_AR (K, 100) # Build the monitor (cou

Modules binding and simulation loop

The rest of the script does not change

[4]: enc["encode ::U_K "] = src["generate ::U_K "]
      mdm["modulate ::X_N1"] = enc["encode ::X_N "]
      chn["add_noise ::X_N "] = mdm["modulate ::X_N2"]
      mdm["demodulate ::Y_N1"] = chn["add_noise ::Y_N "]
      dec["decode_siho ::Y_N "] = mdm["demodulate ::Y_N2"]
      mnt["check_errors ::U "] = src["generate ::U_K "]
      mnt["check_errors ::V "] = dec["decode_siho::V_K "]

      sigma = np.ndarray(shape = (1,1), dtype = np.float32)
      chn["add_noise::CP"] = sigma
      mdm["demodulate::CP"] = sigma
```

Python

High Level Prog

- **PY_AFF3CT**

- Python acc
- Based on p

- **PYAF**

- Insert cust
in PY_AFF3

The screenshot shows a Jupyter Notebook titled '3_LDPC.ipynb' running on a 'localhost' browser. The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar with icons for file operations and execution. The notebook content is as follows:

```
import py_aff3ct
import py_aff3ct.tools.sparse_matrix as tool_sp
```

Load the LDPC parity check matrix

The `sparse_matrix` tool loads the H matrix of the LDPC code in `.alist` format. One can alternatively load a matrix in Quasi-Cyclic format. The H matrix consists in `N` rows and `m` columns, with `m = N-K` if the matrix is full rank.

```
[2]: H = tool_sp.alist.read("../lib/aff3ct/conf/dec/LDPC/WIMAX_288_576.alist")
      N = H.shape[0]
      m = H.shape[1]
      K = N-m
      R = K/N

      bps = 4 # Bits per Symbols
      Ns = N//bps # Packet size in symbols
      I = 100 # Number of max iterations
      FE = 1000 # Frame errors target

      ebn0_min = 0
      ebn0_max = 12.0
      ebn0_step = 1.0

      ebn0 = np.arange(ebn0_min, ebn0_max, ebn0_step)
      esn0 = ebn0 + 10 * math.log10(R)
      sigma_vals = 1/(math.sqrt(2) * 10 ** (esn0 / 20))
```

Build the modules

The LDPC encoder is built with the code dimension, the code length and the H matrix of the LDPC code. AFF3CT supports different kind of encoders and decoders, see the [doc](#) for a full list of available encoders and decoders.

```
[3]: src = py_aff3ct.module.source.Source_random (K) # Build source
      enc = py_aff3ct.module.encoder.Encoder_LDPC_from_H (K, N, H) # Build the encoder
      # enc = py_aff3ct.module.encoder.Encoder_LDPC_from_QC (K, N, H) # Build the encoder
      cstl = py_aff3ct.tools.constellation.Constellation_QAM (bps) # Setup constellation to
      mdm = py_aff3ct.module.modem.Modem_generic (N, cstl) # Build the modem
      chn = py_aff3ct.module.channel.Channel_AWGN_LLNR (2*Ns) # Build the channel (Ns
      dec = py_aff3ct.module.decoder.Decoder_LDPC_BP_horizontal_layered_inter_NMS (K, N, I, H, enc.get_info_bits_pos()) # Build the decoder
      mnt = py_aff3ct.module.monitor.Monitor_BFER_AR (K, 100) # Build the monitor (cou
```


Python

High Level Progm

- PY_AFF3CT
 - Python acc
 - Based on p
- PYAF
 - Insert cust
 - in PY_AFF3

```
Rs = 4 # Number of symbols per symbols
Ns = N//bps # Packet size in symbols
I = 100 # Number of max iterations
FE = 1000 # Frame errors target

ebn0_min = 0
ebn0_max = 12.0
ebn0_step = 1.0

ebn0 = np.arange(ebn0_min,ebn0_max,ebn0_step)
esn0 = ebn0 + 10 * math.log10(R)
sigma_vals = 1/(math.sqrt(2) * 10 ** (esn0 / 20))
```

Build the modules

The LDPC encoder is built with the code dimension, the code length and the H matrix of the LDPC code. AFF3CT supports different kind of encoders and decoders, see the [doc](#) for a full list of available encoders and decoders.

```
[3]: src = py_aff3ct.module.source.Source_random (K) # Build source
enc = py_aff3ct.module.encoder.Encoder_LDPC_from_H (K, N, H) # Build the encoder
# enc = py_aff3ct.module.encoder.Encoder_LDPC_from_QC (K, N, H) # Build the encoder
cstl = py_aff3ct.tools.constellation.Constellation_QAM (bps) # Setup constellation to
mdm = py_aff3ct.module.modem.Modem_generic (N, cstl) # Build the modem
chn = py_aff3ct.module.channel.Channel_AWGN_LLR (2*Ns) # Build the channel (Ns
dec = py_aff3ct.module.decoder.Decoder_LDPC_BP_horizontal_layered_inter_NMS (K, N, I, H, enc.get_info_bits_pos()) # Build the decoder
mnt = py_aff3ct.module.monitor.Monitor_BFER_AR (K,100) # Build the monitor (cou
```

Modules binding and simulation loop

The rest of the script does not change

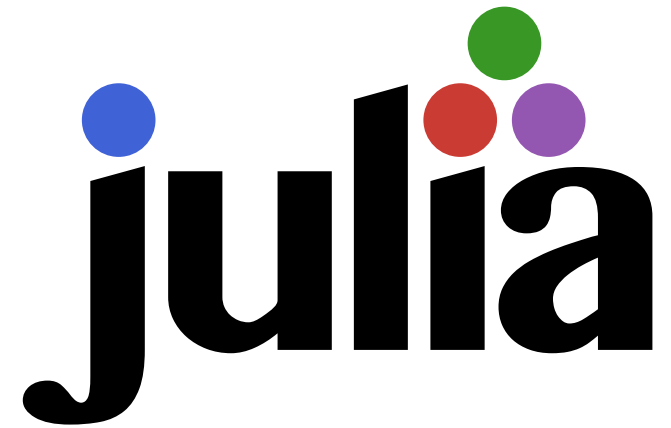
```
[4]: enc["encode ::U_K "] = src["generate ::U_K "]
mdm["modulate ::X_N1"] = enc["encode ::X_N "]
chn["add_noise ::X_N "] = mdm["modulate ::X_N2"]
mdm["demodulate ::Y_N1"] = chn["add_noise ::Y_N "]
dec["decode_siho ::Y_N "] = mdm["demodulate ::Y_N2"]
mnt["check_errors ::U "] = src["generate ::U_K "]
mnt["check_errors ::V "] = dec["decode_siho::V_K "]

sigma = np.ndarray(shape = (1,1), dtype = np.float32)
chn['add_noise::CP'] = sigma
mdm['demodulate::CP'] = sigma
```



8.

Experimental High-level Programming with Julia





Why Julia?

2 Julia

- **High level language** with meta-programming
- Just-in-time (JIT) compilation based on LLVM
- Looks like Python but with higher performance
 - No global interpreter lock (GIL) → Enable **multi-thread integration!**
- Good candidate to **prototype new algorithms** in AFF3CT
 - Easier than doing it in C++



Interfacing with Julia

2 Julia

Name	Language	Stars	Supp. J. versions	Last com.	Threads	Except.	J. $\leftarrow$ C++	C++ $\leftarrow$ J.
API C JULIA	C	—	$\mathcal{V} \leq 1.10$	—	~	✗	✓	~
CXX.JL	C++??	754 ★	$1.01 \leq \mathcal{V} \leq 1.03$	May 2024	✗	✗	✓	✗
CXXWRAP.JL	C++17	415 ★	$\mathcal{V} \leq 1.10$	Aug. 2024	✗	✗	✓	~
CXXINTERFACE.JL	C++??	45 ★	$\mathcal{V} \leq 1.10$	Jan. 2022	✗	✗	✓	✗
JLUNA	C++20	243 ★	$1.07 \leq \mathcal{V} \leq 1.10$	Apr. 2024	✓	✓	✓	✓

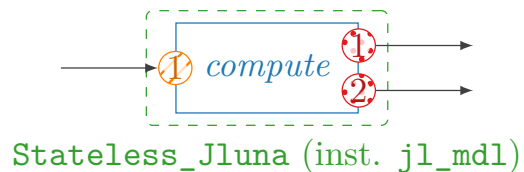
- **List of existing solutions** to interface with 
- **Select Jluna** as it seems to fit the best to be integrated in STREAMPU
 - Written in C++ for C++ applications
 - Support threads with its own pool of threads
 - Can invoke  code from C++ side
 - Can invoke C++ code from  side



New Julia Stateless Class

2 Julia

- `module::Stateless_Jluna` class
 - Behavior is quasi-similar to `module::Stateless` class
- Very **basic code example**: a task that reads input socket and writes the data into two output sockets



```
1 // C++ code
2 module::Stateless_Jluna jl_md1;
3 runtime::Task& t = jl_md1.create_task("compute");
4 jl_md1.create_socket_in <int8_t>(t, "in", N );
5 jl_md1.create_socket_out<int8_t>(t, "out1", N / 2);
6 jl_md1.create_socket_out<int8_t>(t, "out2", N / 2);
7 jl_md1.create_codelet_file(t, "compute.jl");
```

```
1 # Julia code ("compute.jl" file contents)
2 function compute(sck_in, sck_out1, sck_out2,
3                 rnt_frame_id, rnt_n_frames_per_wave)
4     # for each element in 'sck_in'
5     for n in eachindex(sck_in)
6         if n % 2 == 0
7             sck_out1[n / 2] = sck_in[n] # even
8         else
9             sck_out2[n / 2] = sck_in[n] # odd
10        end
11    end
12    return 0 # return val in 'status' sck
13 end
```



Benchmark Protocol

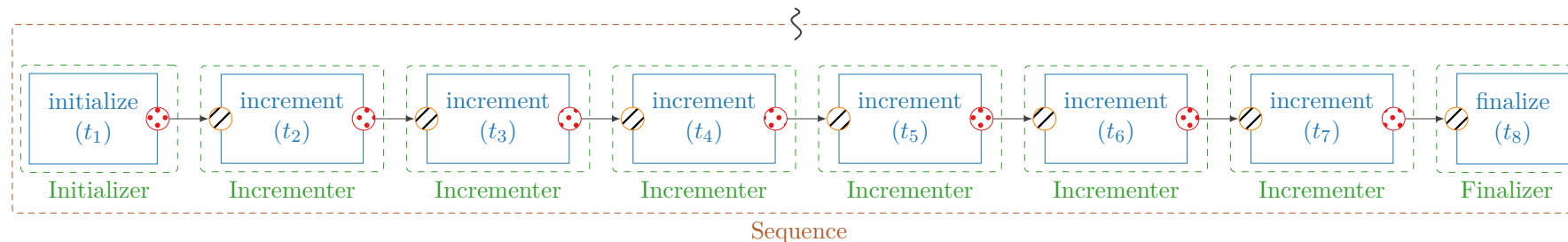
2 Julia

Software & hardware configuration:

- **AMD Ryzen 7840U CPU** (Zen4)
- **Ubuntu 24.04 LTS** (kernel 6.8)
- **C++ GNU Compiler v13.2.0**
 - `-O3 -mavx2 -funroll-loops`
- **JULIA v1.10.4 & JLUNA 7b08c4f**

Benchmark application:

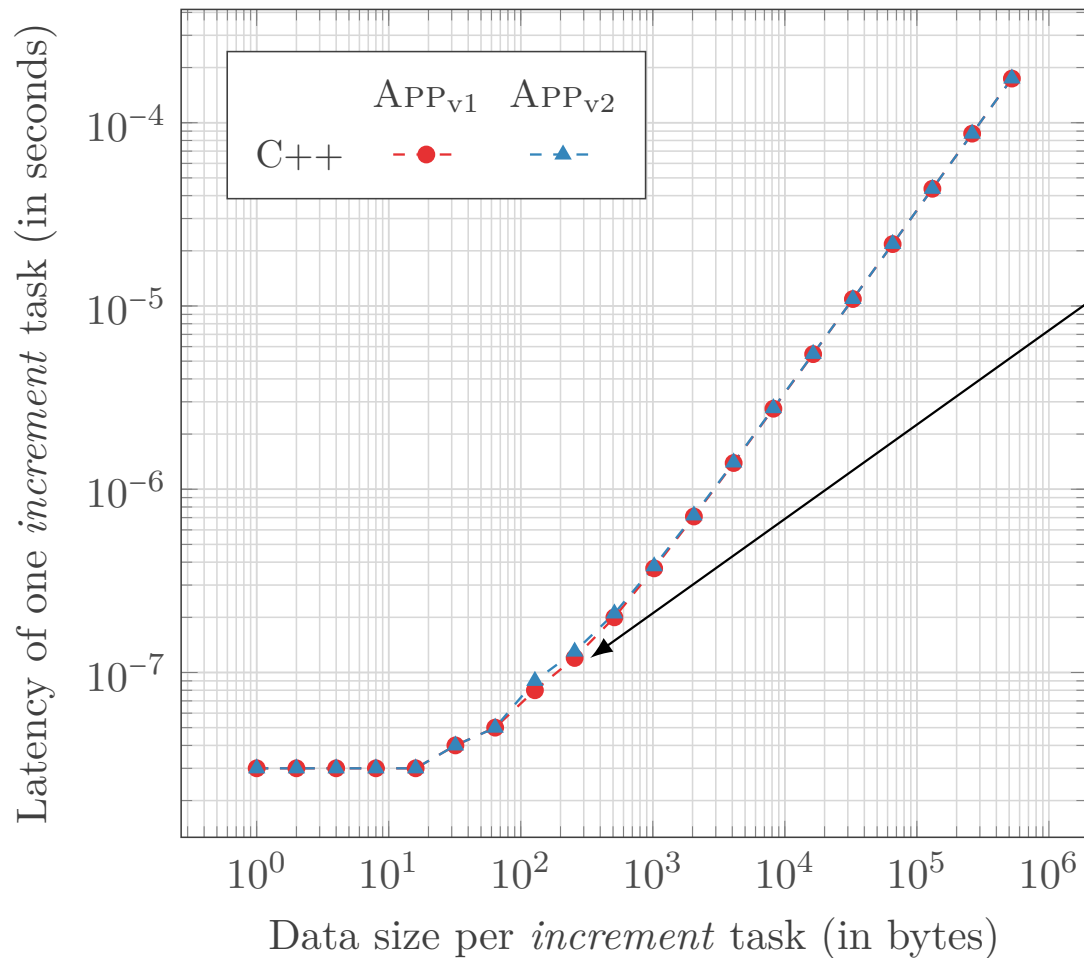
- **6 increment tasks chained**
 - **Run in loop**, memory bound
- Two variants has been evaluated
 - APP_{v1}: increment tasks with input/output sockets
 - APP_{v2}: increment tasks with forward sockets (= 0-copy)





Experimental Results

2 Julia



Implem. type of the *increment* tasks:

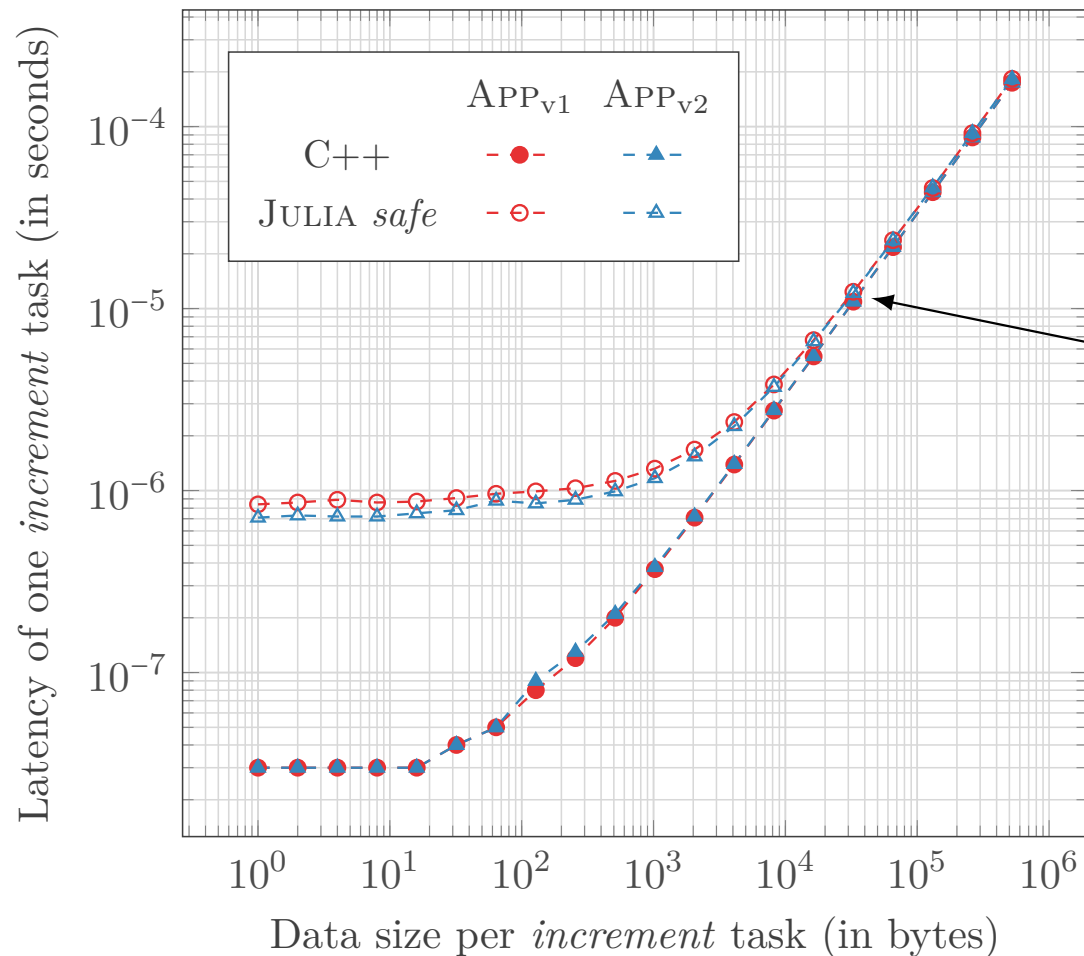
- In C++

— Efficient for task latency > 0.1 μ s



Experimental Results

2 Julia



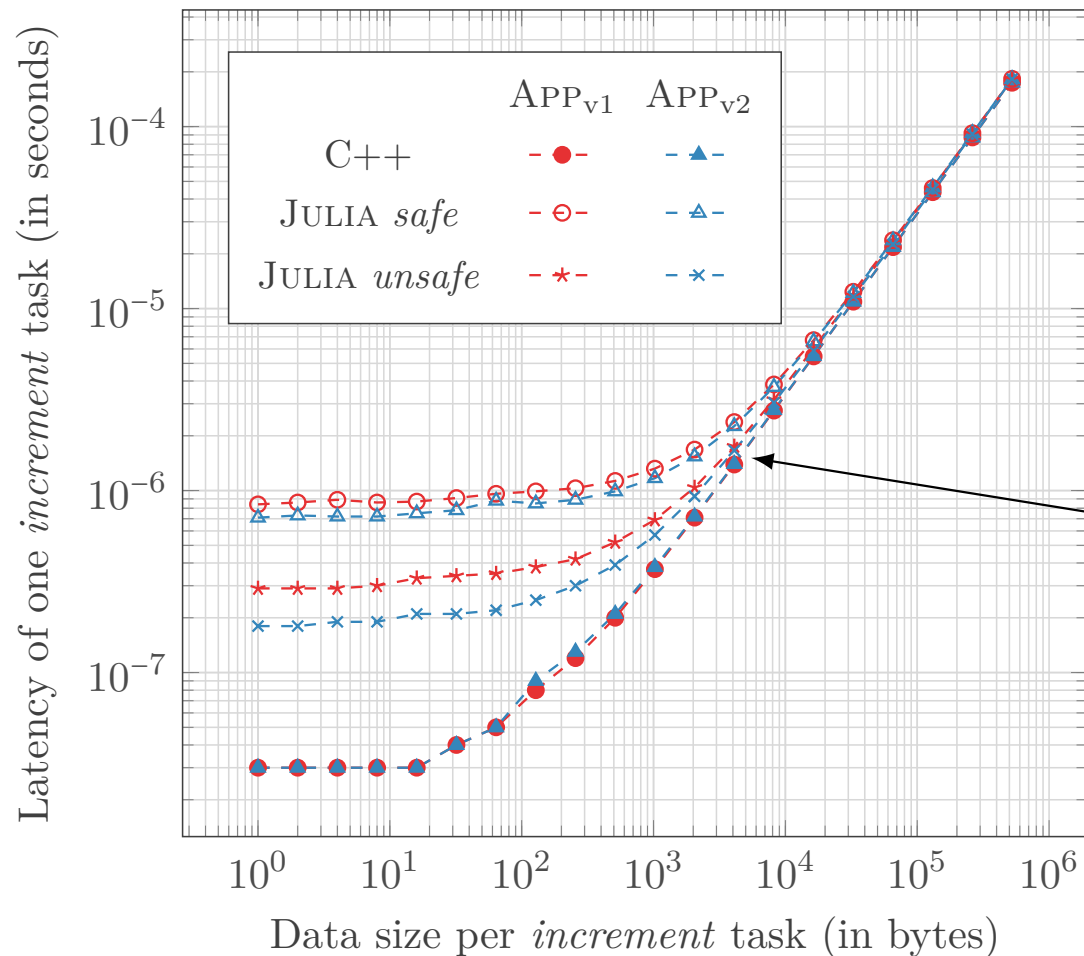
Implem. type of the *increment* tasks:

- In C++
 - Efficient for task latency $> 0.1 \mu s$
- In JULIA with errors management (safe)
 - Match C++ perf for latency $> 10 \mu s$



Experimental Results

2 Julia



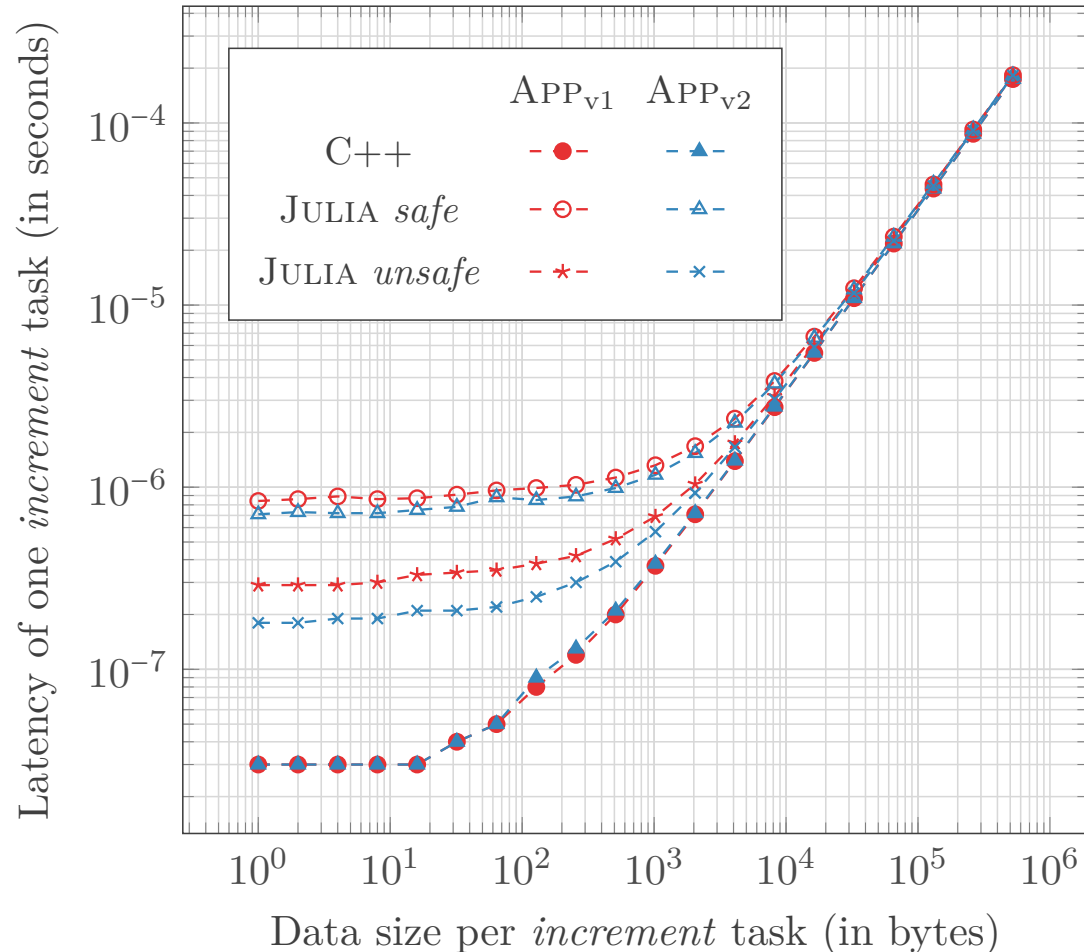
Implem. type of the *increment* tasks:

- In C++
 - Efficient for task latency $> 0.1 \mu s$
- In JULIA with errors management (safe)
 - Match C++ perf for latency $> 10 \mu s$
- In JULIA **for production** (unsafe)
 - Match C++ perf for latency $> 1.5 \mu s$



Experimental Results

2 Julia



Implem. type of the *increment* tasks:

- In C++
 - Efficient for task latency $> 0.1 \mu s$
- In JULIA with errors management (safe)
 - Match C++ perf for latency $> 10 \mu s$
- In JULIA **for production** (unsafe)
 - Match C++ perf for latency $> 1.5 \mu s$
- **julia** tasks can be **relevant for many signal processing algorithms**
 - Most of the AFF3CT modules have a latency higher than $1.5 \mu s$!



What's Next?

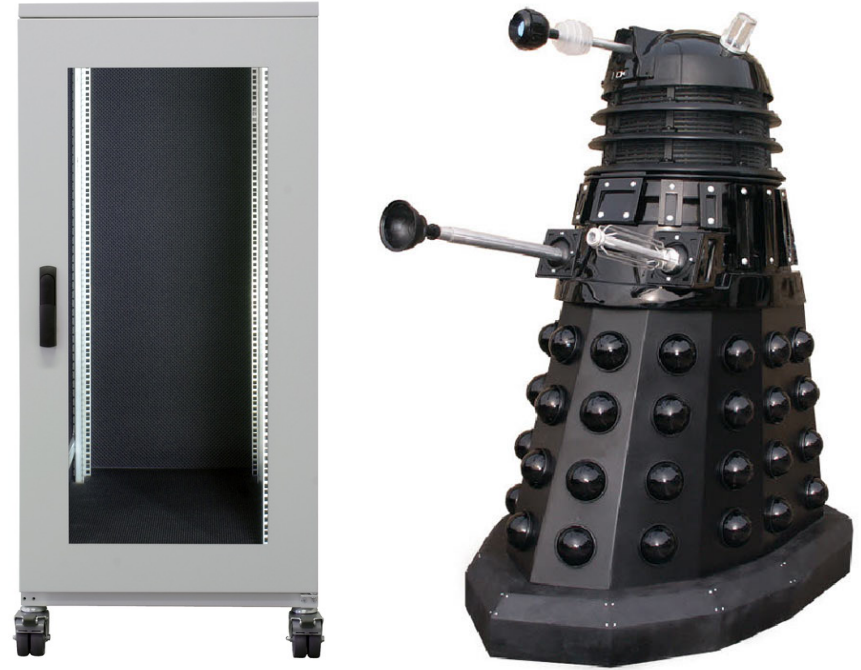
2 Julia

- Implement **stateful task** in **julia** → Need to manage the *clone* pattern
- Support **multi-threaded executions** of **julia** tasks (pipeline & replication)
- Integrate STREAMPU × **julia** to AFF3CT
- Implement a **representative signal processing task** in **julia** and in AFF3CT to validate the entire process

9.

Outreach initiatives – The DAL3K Heterogeneous Cluster at LIP6 for Architecture Enthusiast

<https://dalek.proj.lip6.fr/>





A Cluster Designed for Researchers

3 DAL3K

The DAL3K cluster

- A cluster of **heterogeneous nodes**
 - Enable AFF3CT eval a various architectures
- Made from low power mini-PC nodes: What an idea?!
 - Absolutely not designed for clustering → **Challenging!**
- Advantages
 - Components from the public at large → **Cheap!**
 - **Fast availability** of the chips and **easy to upgrade**
- For who?
 - The AFF3CT team in general
 - LIP6 researchers and **CS architecture enthusiasts**





Partitions

3 DAL3K

- 4 heterogeneous partitions of 4 nodes each
- NPUs and some iGPUs are not detailed below

	Processor (CPU)					System RAM Memory			
	Vendor	Model	Archi.	Cores	TDP	Type	MT/s	Chn.	Amount
Partition 1	AMD	Ryzen 9 7945HX	Zen 4	16 (32)	75 W	DDR5	5200	2	96 GB
Partition 2									
Partition 3	AMD	Ryzen AI 9 HX 370	Zen 5	12 (24)	54 W	LPDDR5X	7500	4	32 GB
Partition 4	Intel	Core Ultra 9 185H	Meteor Lake	16 (22)	115 W	DDR5	5600	2	32 GB

Graphical Process Unit (GPU)								
	Vendor	Model	Archi.	VRAM Memory			Cores	TDP
				Type	Bus	Amount		
Partition 1	Nvidia	GeForce RTX 4090	Ada Lovelace	GDDR6X	384-bit	24 GB	128	450 W
Partition 2	AMD	Radeon RX 7900 RTX	RDNA 3	GDDR6	320-bit	24 GB	84	300 W
Partition 3	AMD	Radeon 890M	RDNA 3.5	Unified with CPU RAM			16	–
Partition 4	Intel	Arc A770	Alchemist	GDDR6	256-bit	16 GB	32	225 W



Infrastructure

3 DAL3K

- **Half rack 25U** on wheels with glass for demonstrations
- Network: Unifi Switch Pro Max **48 ports**
 - 4 × SFP+ 10 Gbps (optical fiber)
 - 16 × Ethernet 2.5 Gbps
 - 32 × Ethernet 1 Gbps
- **Frontend node**
 - 14 cores Intel Core i9-13900H (Raptor Lake-H), 96 GB DDR5
 - 2 × Ethernet 2.5 Gbps ports
 - 2 × SFP+ 10 Gbps ports
 - 6 TB PCIe 4 NVMe SSD for NFS
- Energy consumption (estimated)
 - Idle: 734 Watts
 - Maximum: **6320 Watts** (requires two standard sockets)



DALEK Cluster

Summary

<https://dalek.proj.lip6.fr/>

Contact

- Adrien Cassagne, LIP6
- adrien.cassagne@lip6.fr

Number of nodes	1	4	4	4	4
CPU	Intel Core i9-13900H	AMD Ryzen 9 7945HX	AMD Ryzen 9 7945HX	Intel Core Ultra 9 185H	AMD Ryzen AI 9 HX 370
iGPU	Intel Iris Xe Graphics 96EU Mobile	AMD Radeon 610M	AMD Radeon 610M	Intel Arc Graphics 128EU Mobile	AMD Radeon 890M
GPU (or eGPU)	-	Gainward GeForce RTX 4090 Phantom GS 24 GB (PCIe 4)	Sapphire PULSE AMD Radeon RX 7900 XTX 24 GB (PCIe 4)	ASRock Intel Arc A770 Challenger 16 GB OC (Oculink)	-

10.

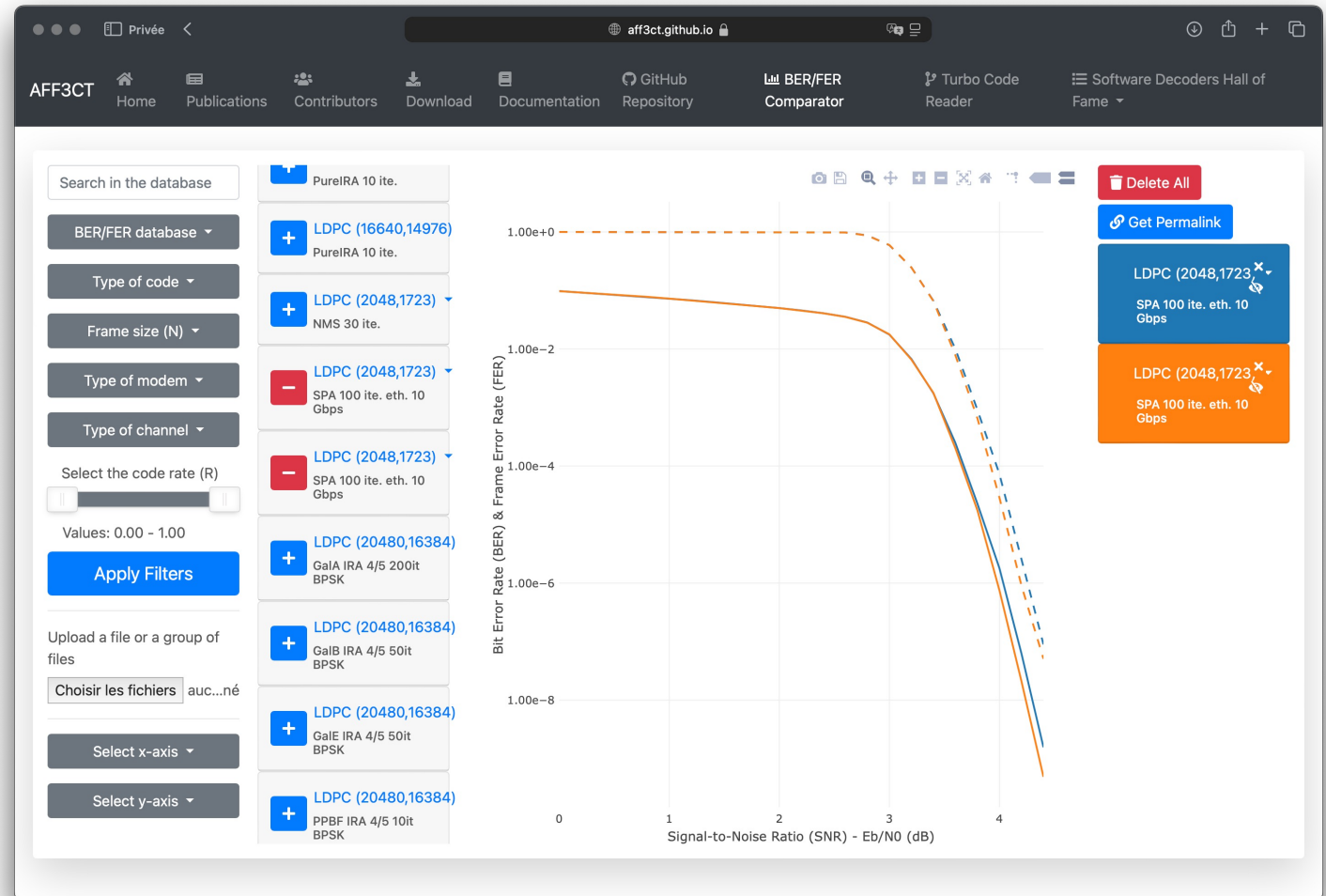
Outreach initiatives – AFF3CT's Online BER / FER Comparator

<https://aff3ct.github.io/comparator.html>

AFF3CT Online BER-FER Comparator

Compare FEC decoding performance with database entries

aff3ct.github.io/comparator.html



11.

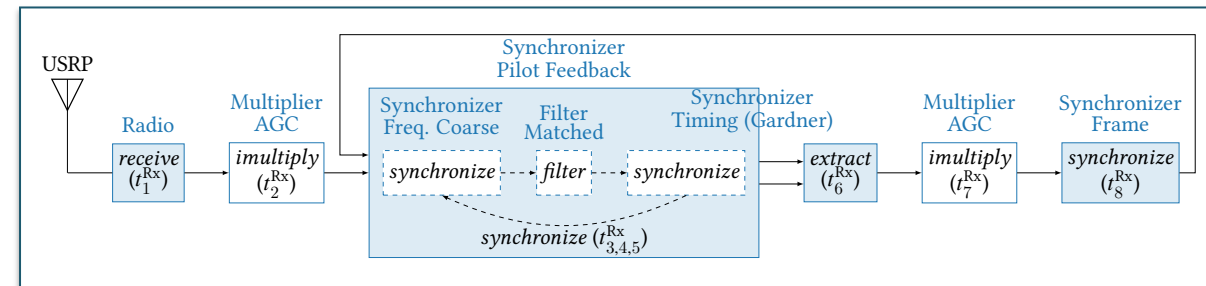
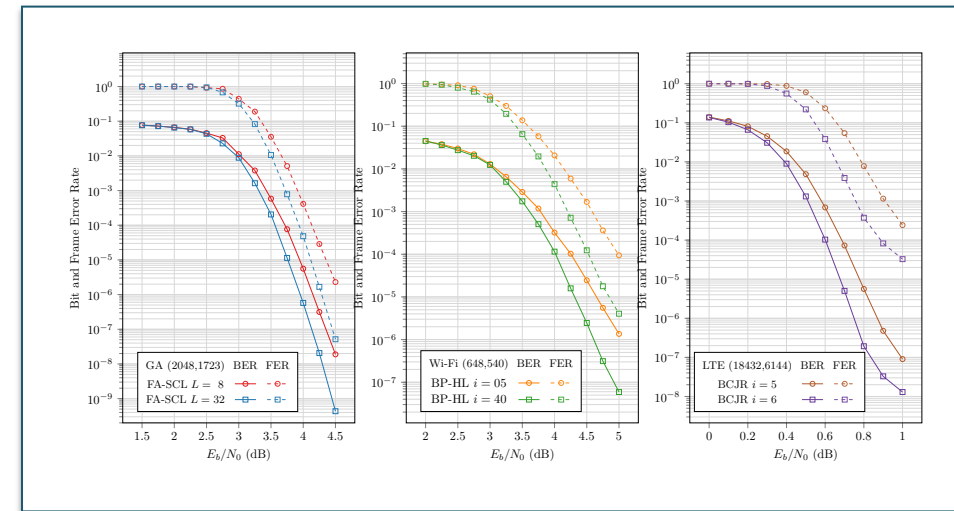
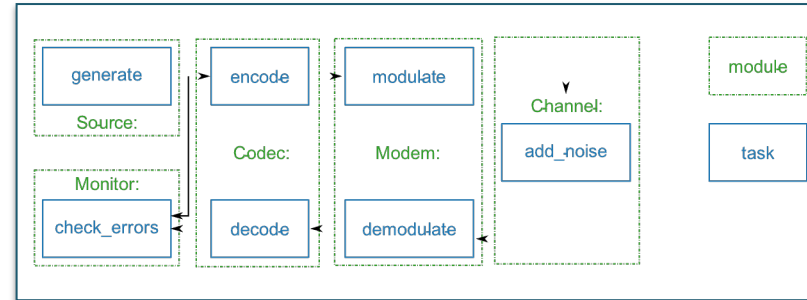
Conclusion

AFF3CT

Modular Digital Communication

- **Software Defined Radio (SDR)**
 - Digital communication chains
 - (5G phones, Wi-Fi, DVB-RCS, etc.)
 - Codec algorithmics
 - ECC, modulation, channel models
 - Design, validation, production
 - Transition from ASICs to software
- **Multicore parallelism**
 - Vector processing
 - Pipeline Resource Mapping

<https://aff3ct.github.io/>



AFF3CT

Small, focused team of developers

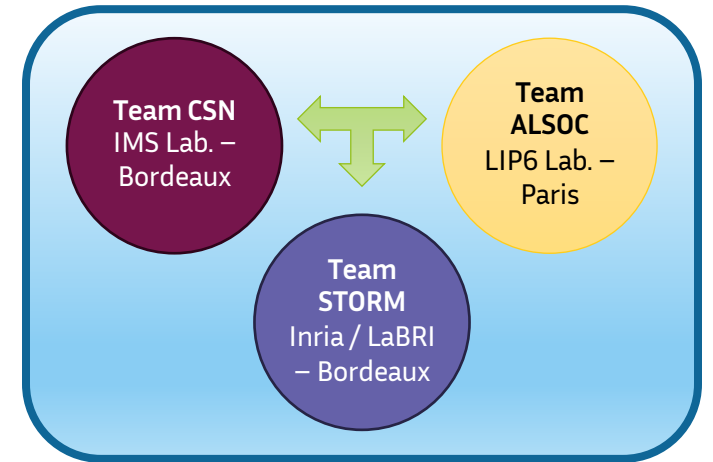
- Inria Bordeaux, IMS Laboratory, LIP6 Laboratory

<https://aff3ct.github.io/>

- Open Source, MIT License

Subprojects

- BER/FER online Comparator
 - <https://aff3ct.github.io/comparator.html>
- PY-AFF3CT
 - https://github.com/aff3ct/py_aff3ct
- StreamPU: task-based streaming runtime
 - <https://github.com/aff3ct/streampu>
- MIPP: C++ SIMD wrapper
 - <https://github.com/aff3ct/MIPP>
- DVB-S2 Chain
 - <https://github.com/aff3ct/dvbs2>



DAL3K Cluster

<https://dalek.proj.lip6.fr/>



Thanks!

<https://aff3ct.github.io/>



