Designing Interruptible Protocols in Distributed Systems

Leonid Nosovitskiy (ln60@st-andrews.ac.uk)

Adam Barwell (adb23@st-andrews.ac.uk)

University of St Andrews

What’s an Interrupt in Distributed System?

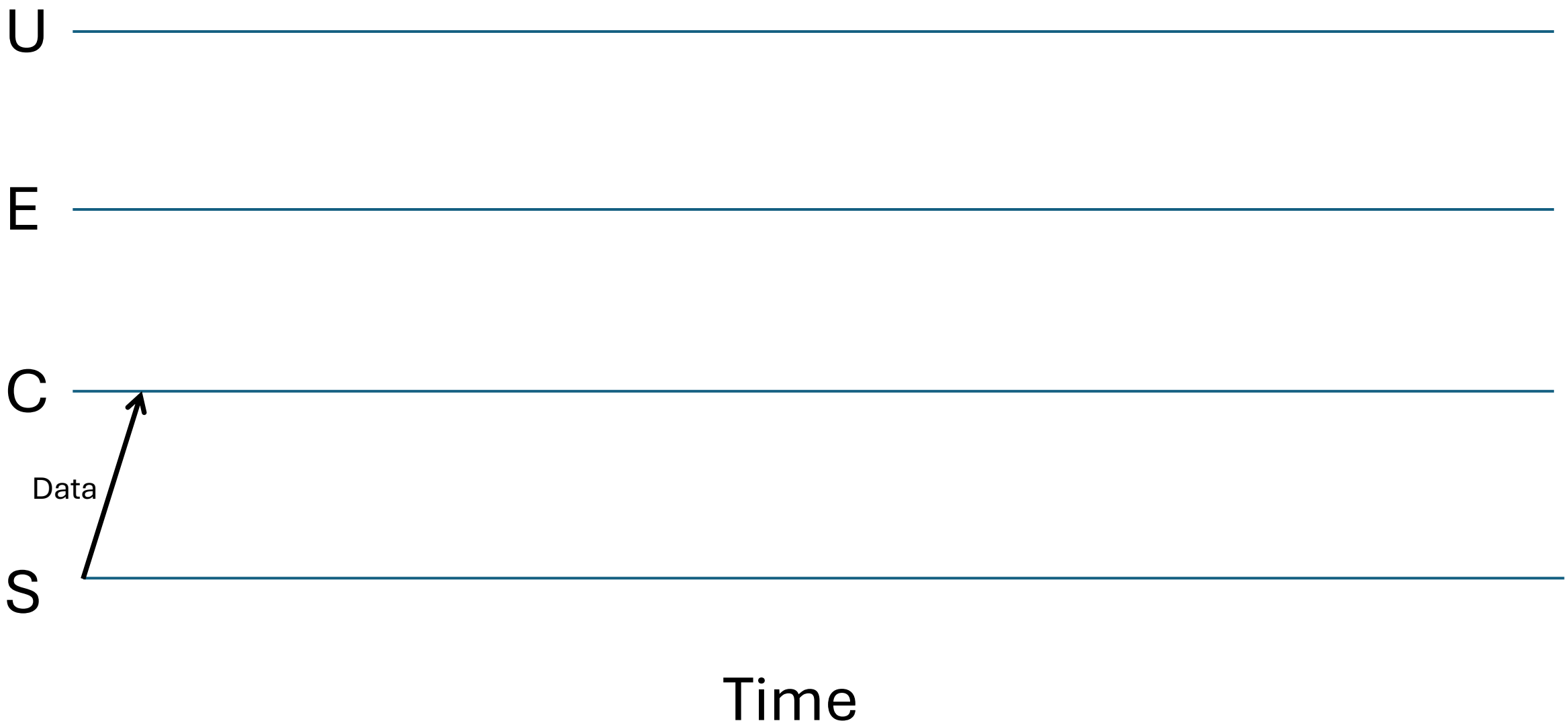
Interrupt – an asynchronous event that forces a node(s) to pause current execution in its current protocol step and immediately handle higher priority process.

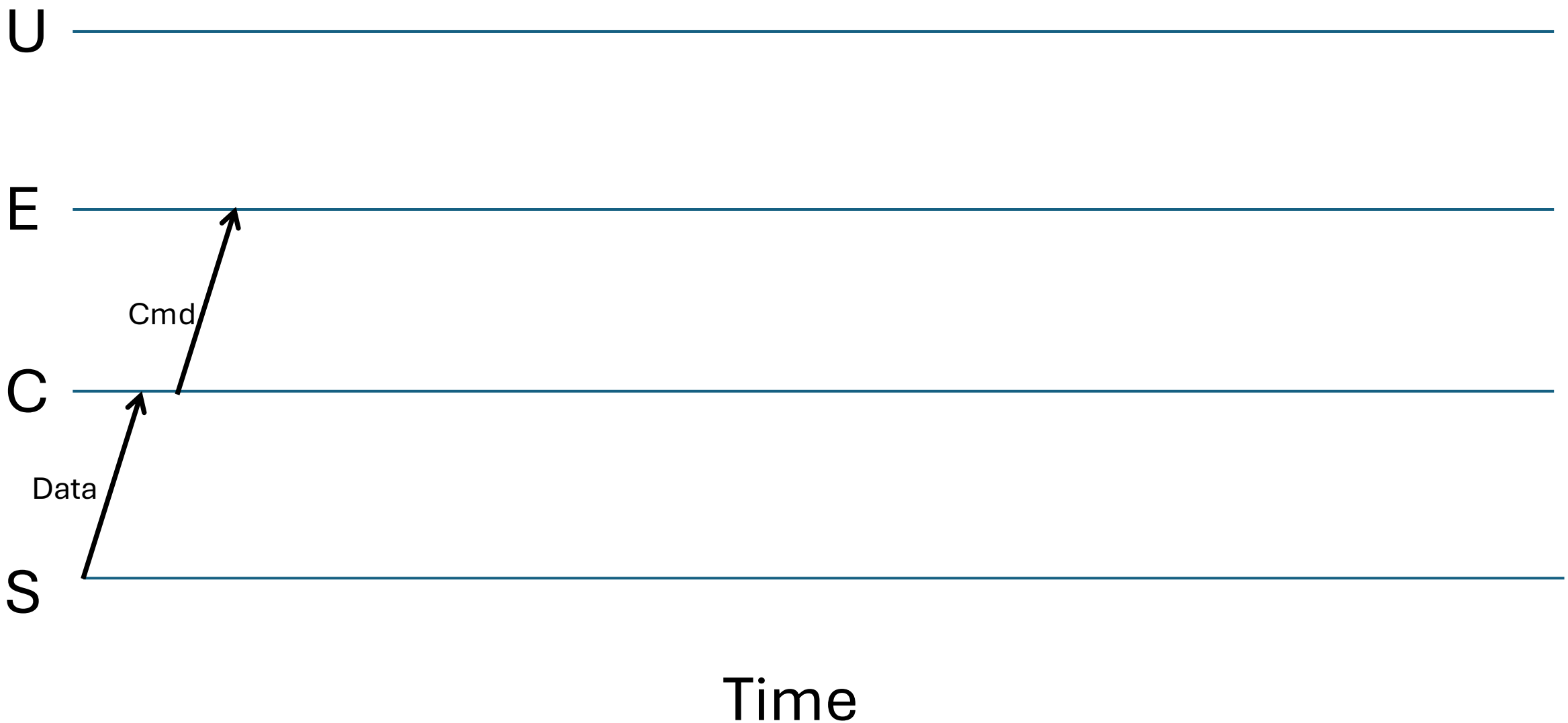
Examples

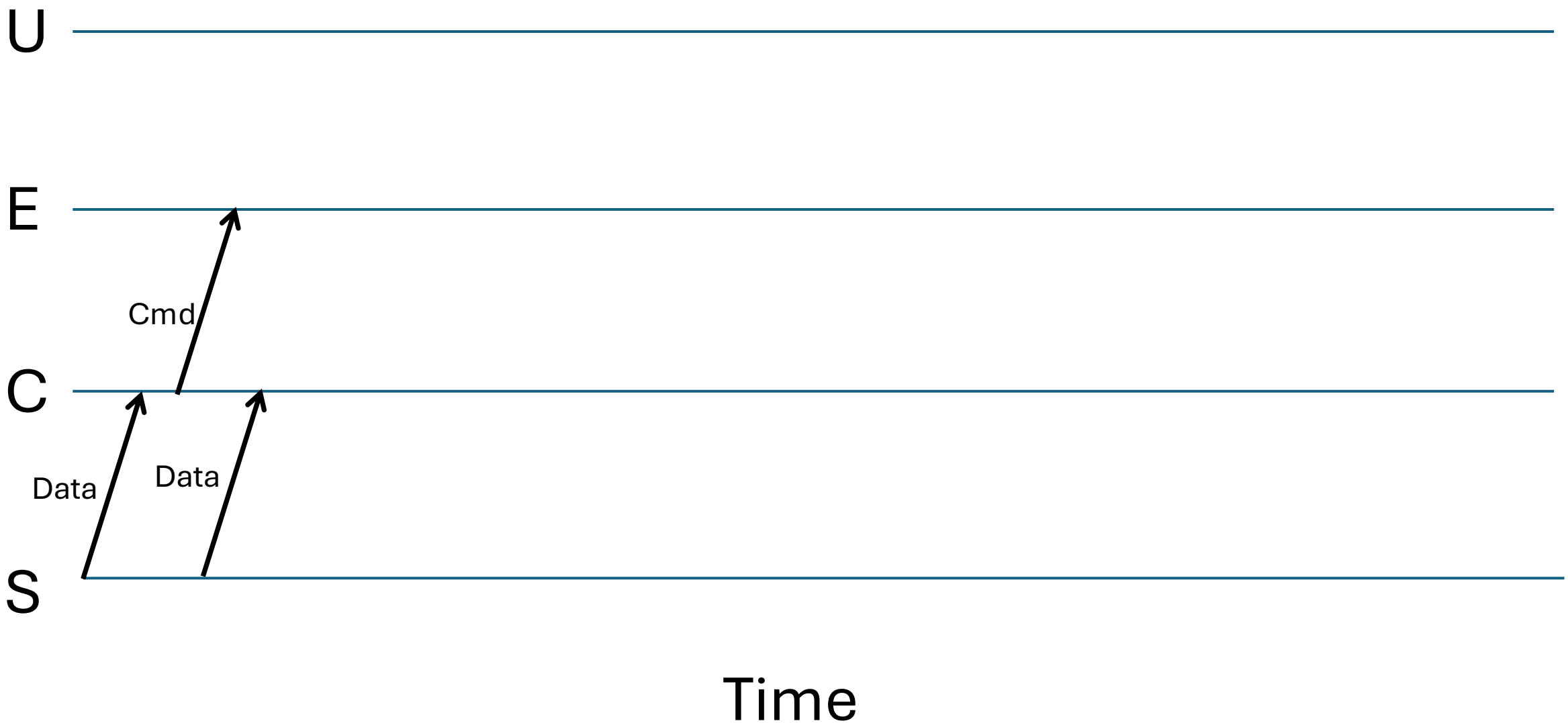
- Parallel Farms – an interrupt may terminate stragglers' current execution to pass a smaller load.
- Financial Information eXchange (FIX) protocol – during order execution an interrupt may change order requirements.
- Session Initiation Protocol (SIP) – an interrupt may abort a conversation.

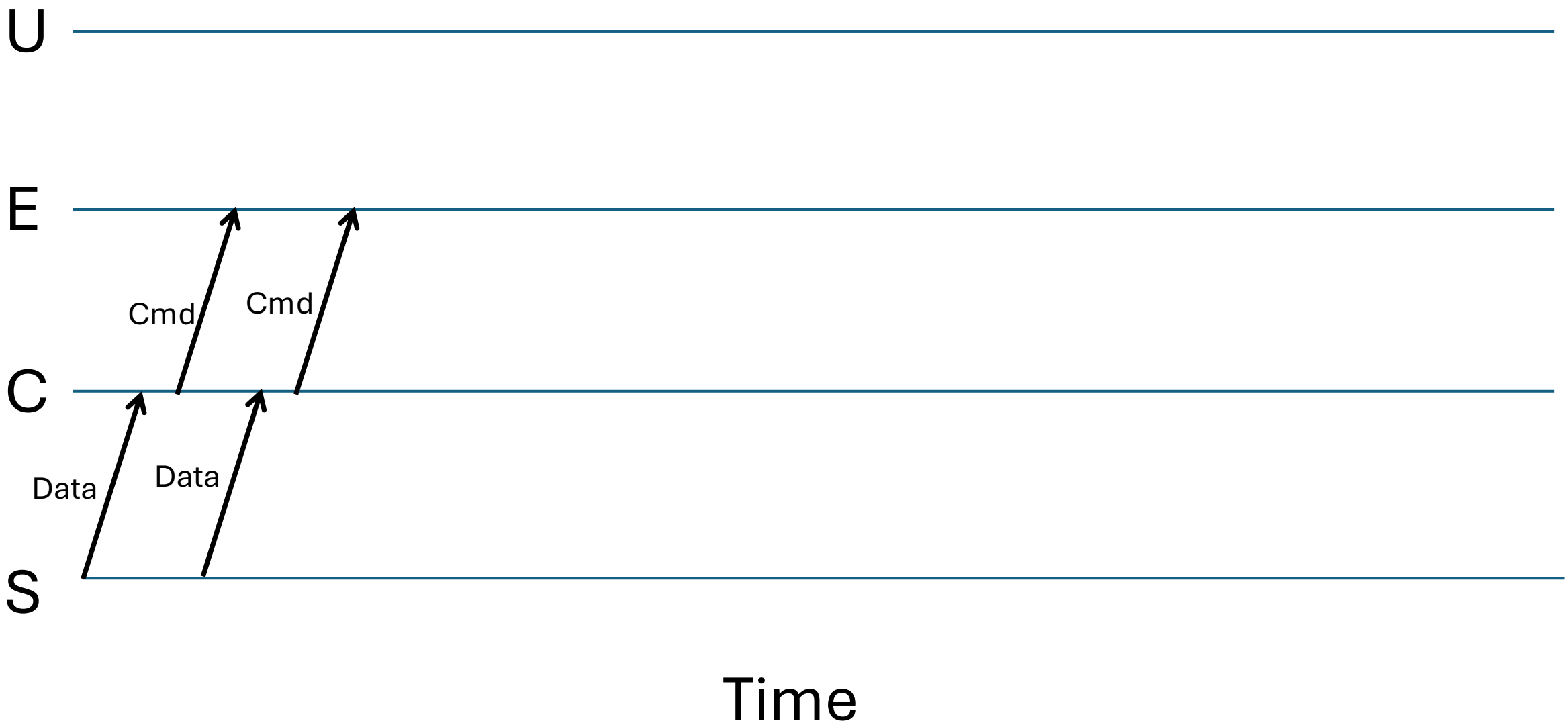
Sensor Example

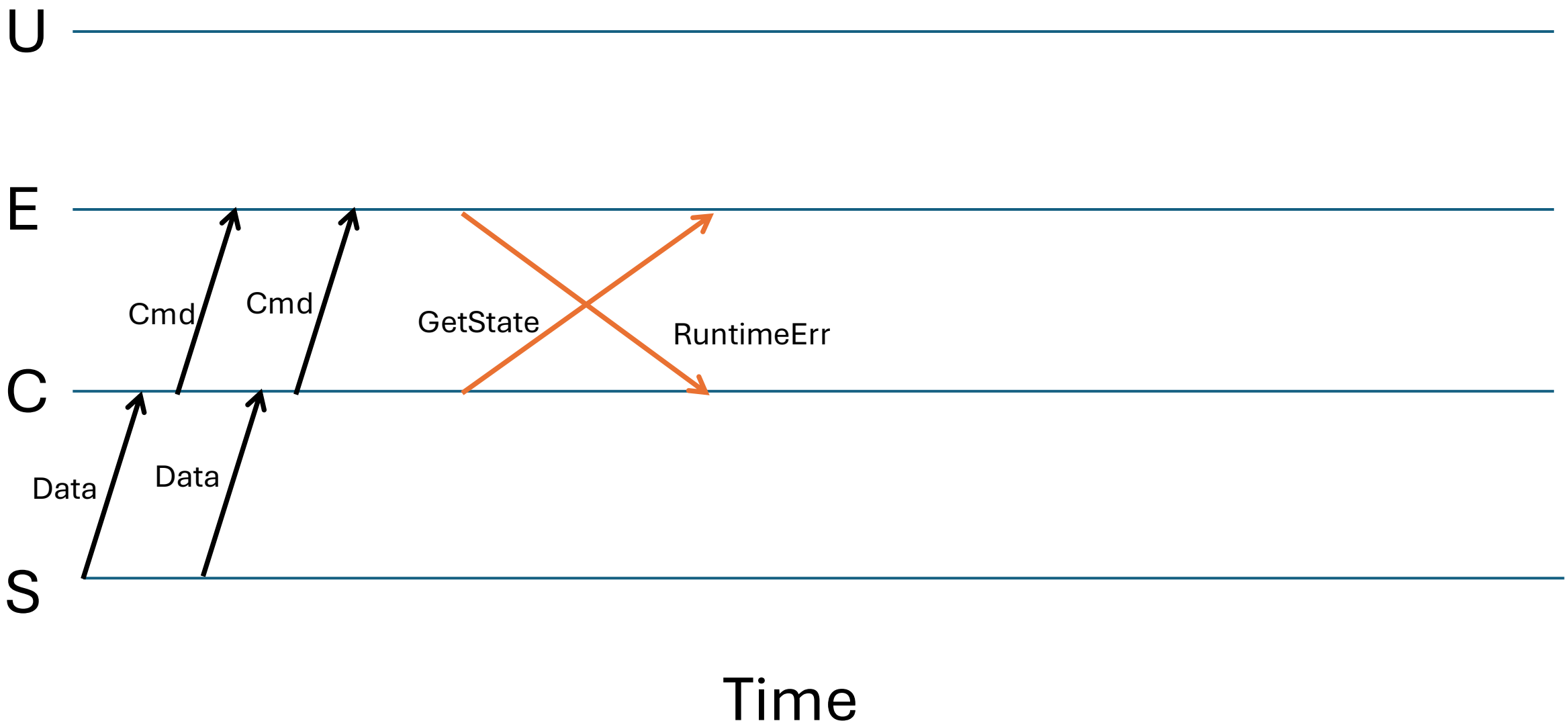
- User (U), Sensor (S), Controller (C), Executor (E)
- Usual continuous infinite flow Sensor → Controller → Executor
- Controller can interrupt to get current state from Executor before sending non-trivial adjustment and then recover to main flow. (Priority 1)
- Executor can interrupt with run-time error requesting an instruction from controller and then recover to main flow. (Priority 1)
- User can interrupt to stop the whole process in the main flow and lower-priority interrupt flow. (Priority 2)

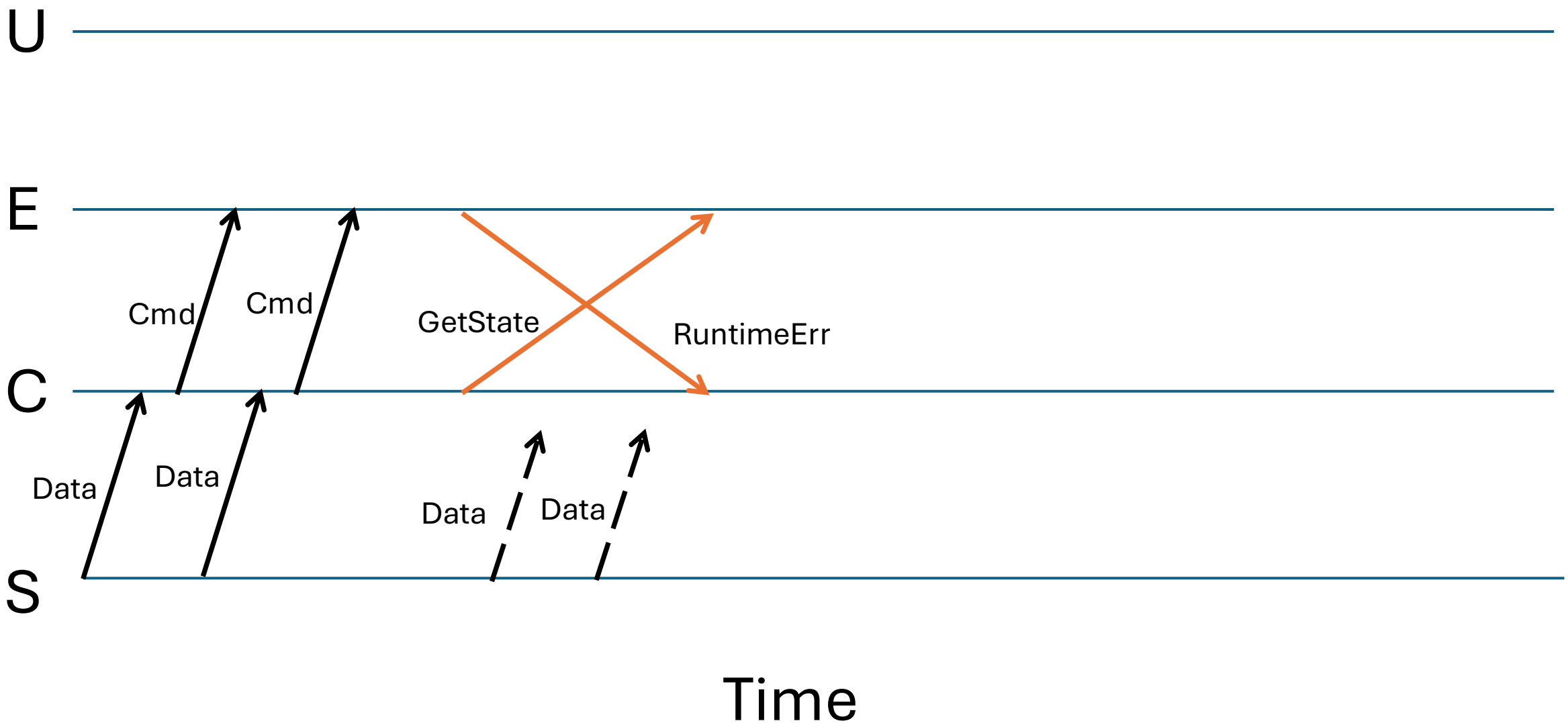

U
E
C
Data
S
Time

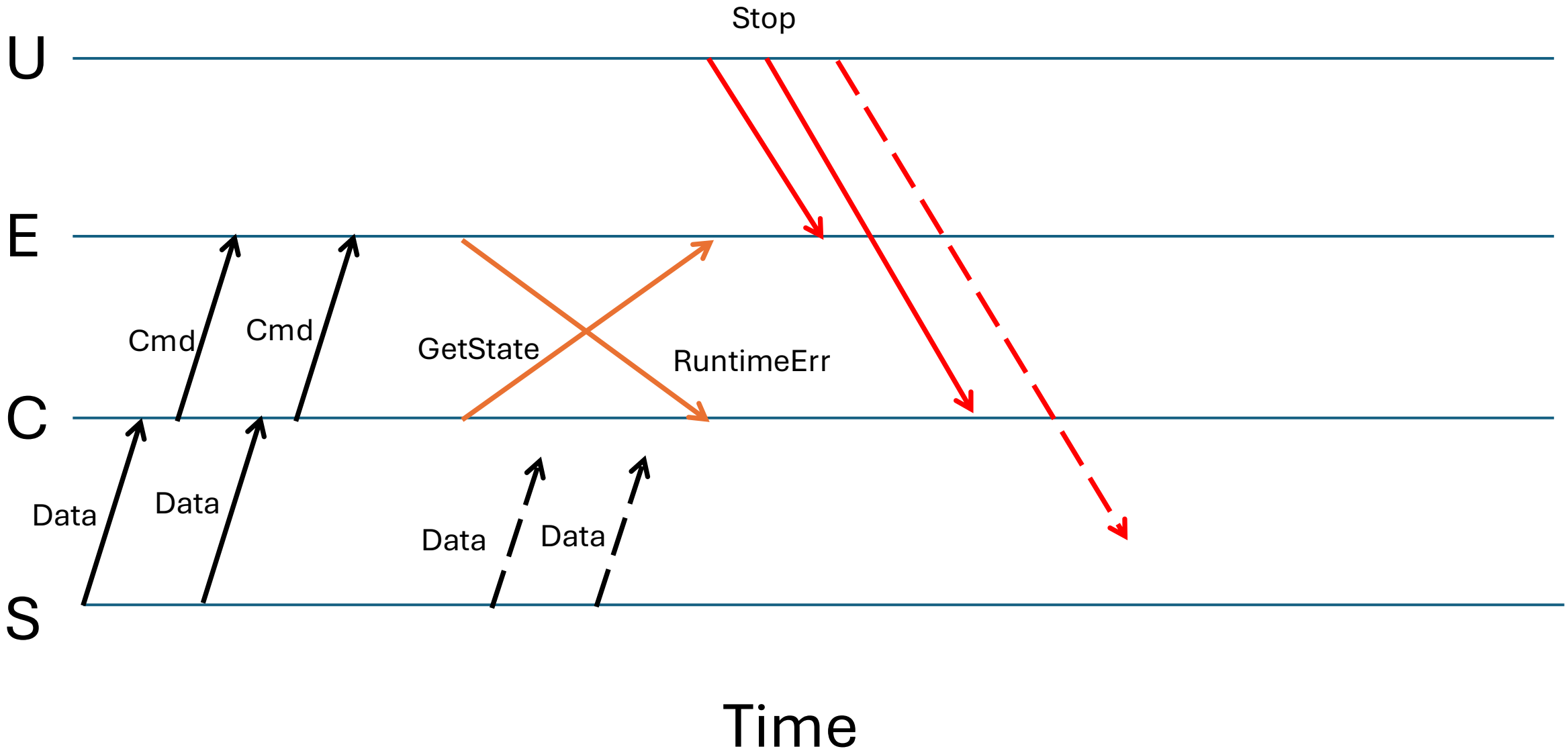

U
E
C
S
Cmd
Data
Time


U
E
C
S
Cmd
Data
Data
Time


U
E
C
S
Cmd
Cmd
Data
Data
Time


U
E
C
S
Cmd
Cmd
GetState
RuntimeErr
Data
Data
Time


U
E
C
S
Cmd
Cmd
GetState
RuntimeErr
Data
Data
Data
Data
Time


Stop
U
E
C
S
Cmd
Cmd
GetState
RuntimeErr
Data
Data
Data
Data
Time

What do we need?

- Cohesiveness – when an interrupt is raised , all should react appropriately. (e.g. the sensor needs to stop sending data)
- Mutual exclusion – same priority events can not arrive in the same time from global view.
- Consistency – when recovering to main flow, participants need to be in the same global state.
- Predictability – we need to know what events to expect.

Possible Solutions – Event Driven Strategies

Solutions such as epoch numbers, leaders, quorums etc…

Pros:

- Better scalability
- Lower latency
- Better resource utilization

Cons:

- More complex reasoning
- Harder to verify design automatically
- Synchronization overhead

Possible Solutions – Token Passing Round Based Strategy

Only one participant gets to talk at a time, each decision is propagated in a specific order

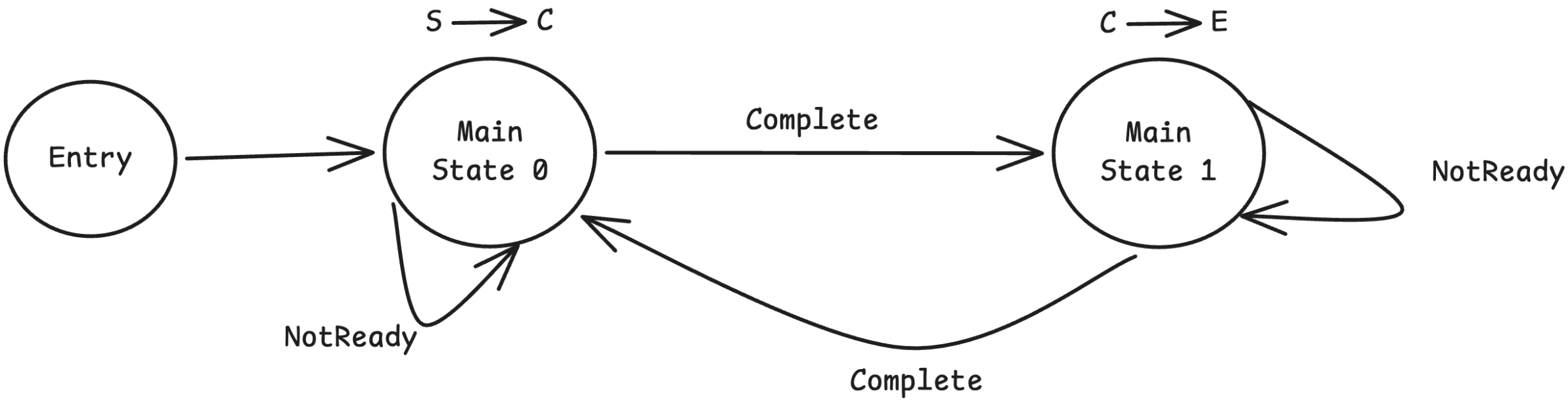
Pros:

- Robust consistency
- Explicit synchronization
- Control

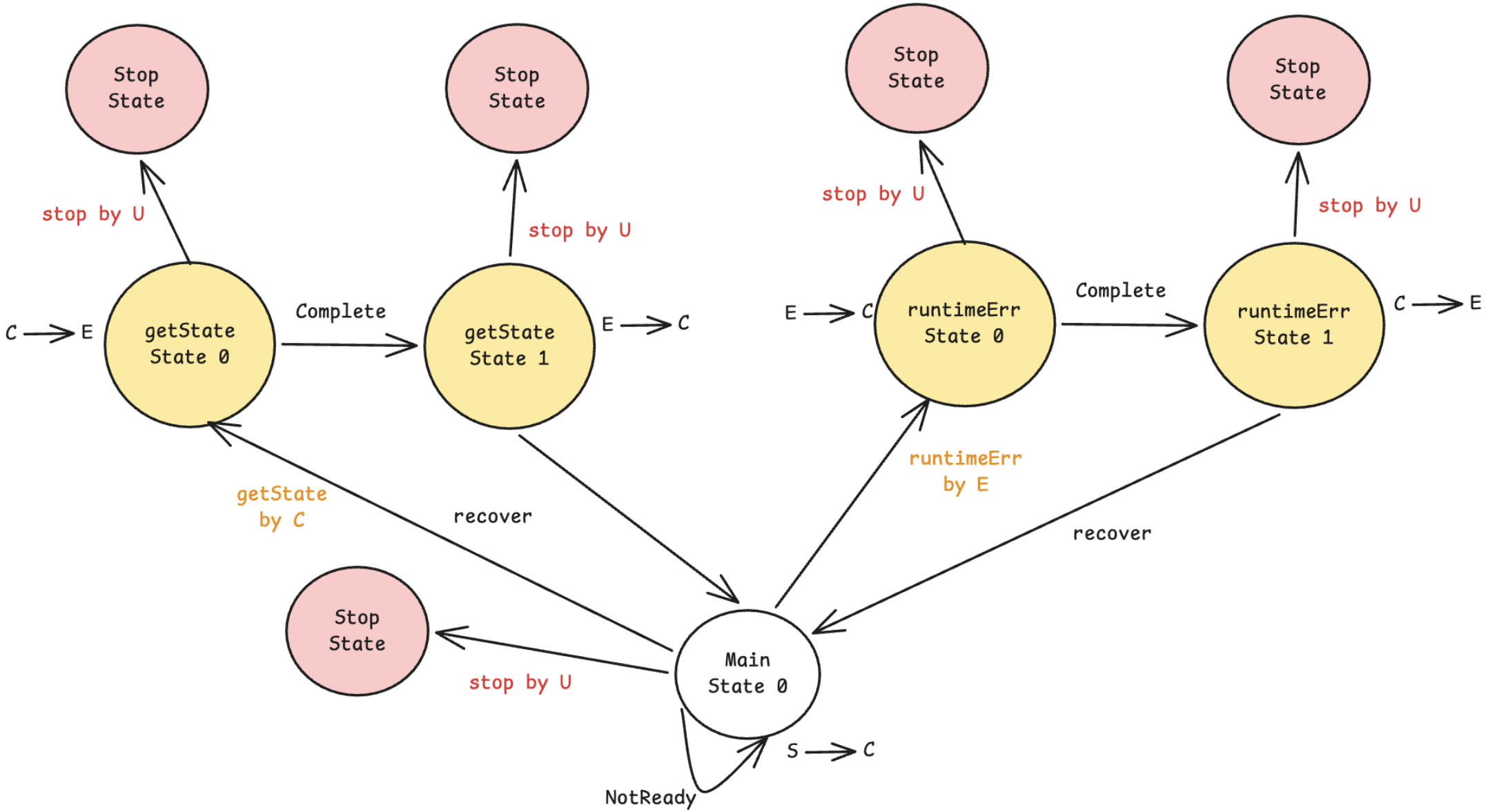
Cons:

- Higher latency for interrupts
- Propagation overhead (overhead for large number of participants)
- Link failure risk

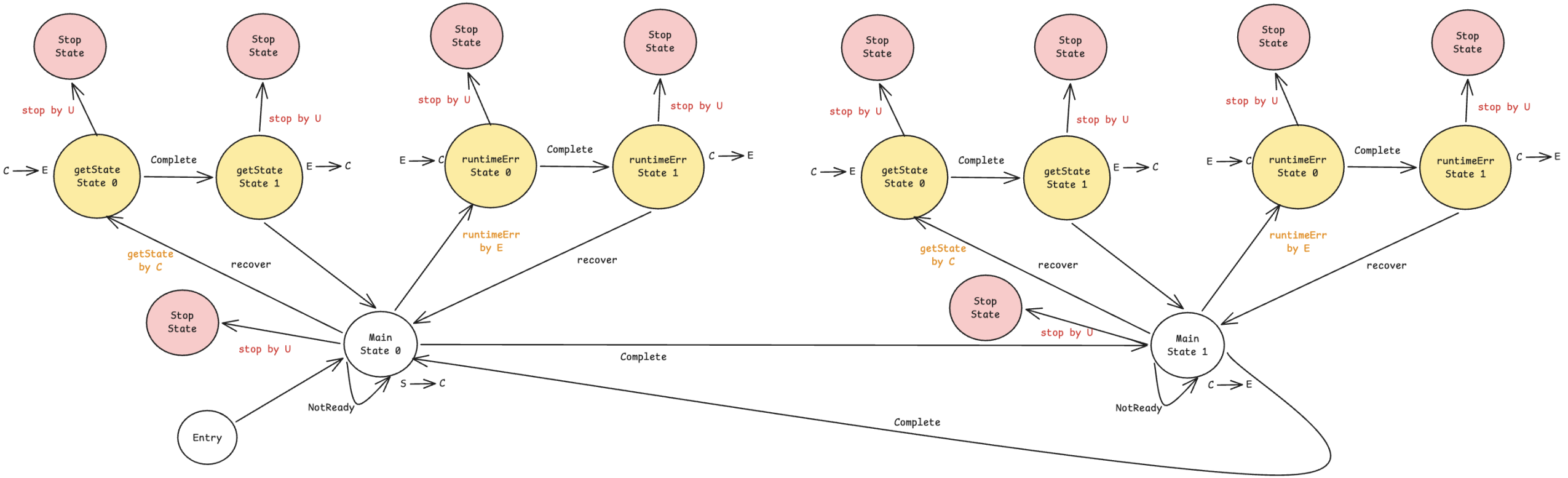
Global FSM


S → C
C → E
Entry
Main
State 0
Complete
Main
State 1
NotReady
NotReady
Complete

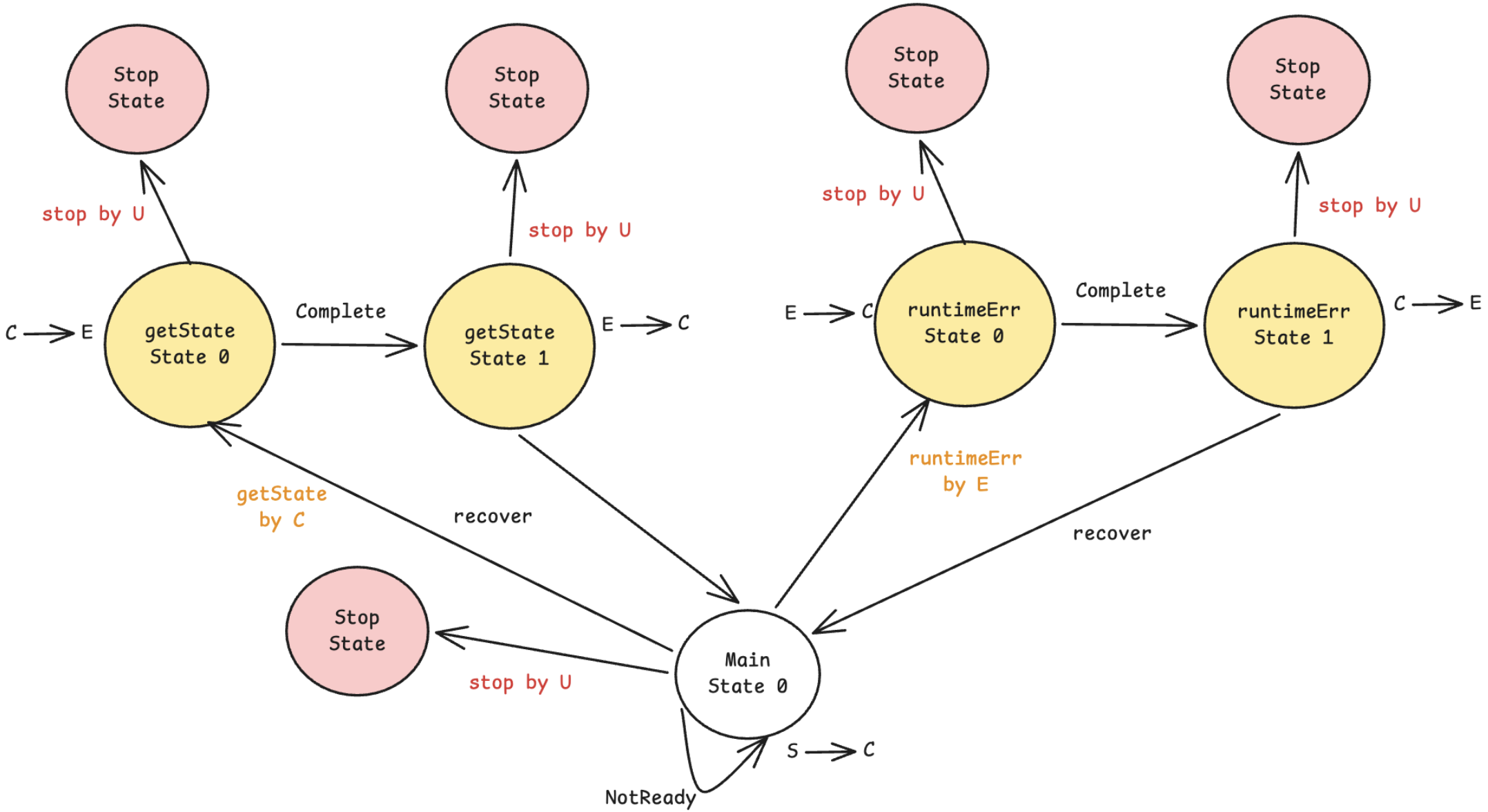
Global FSM Main State 0



Full Global FSM



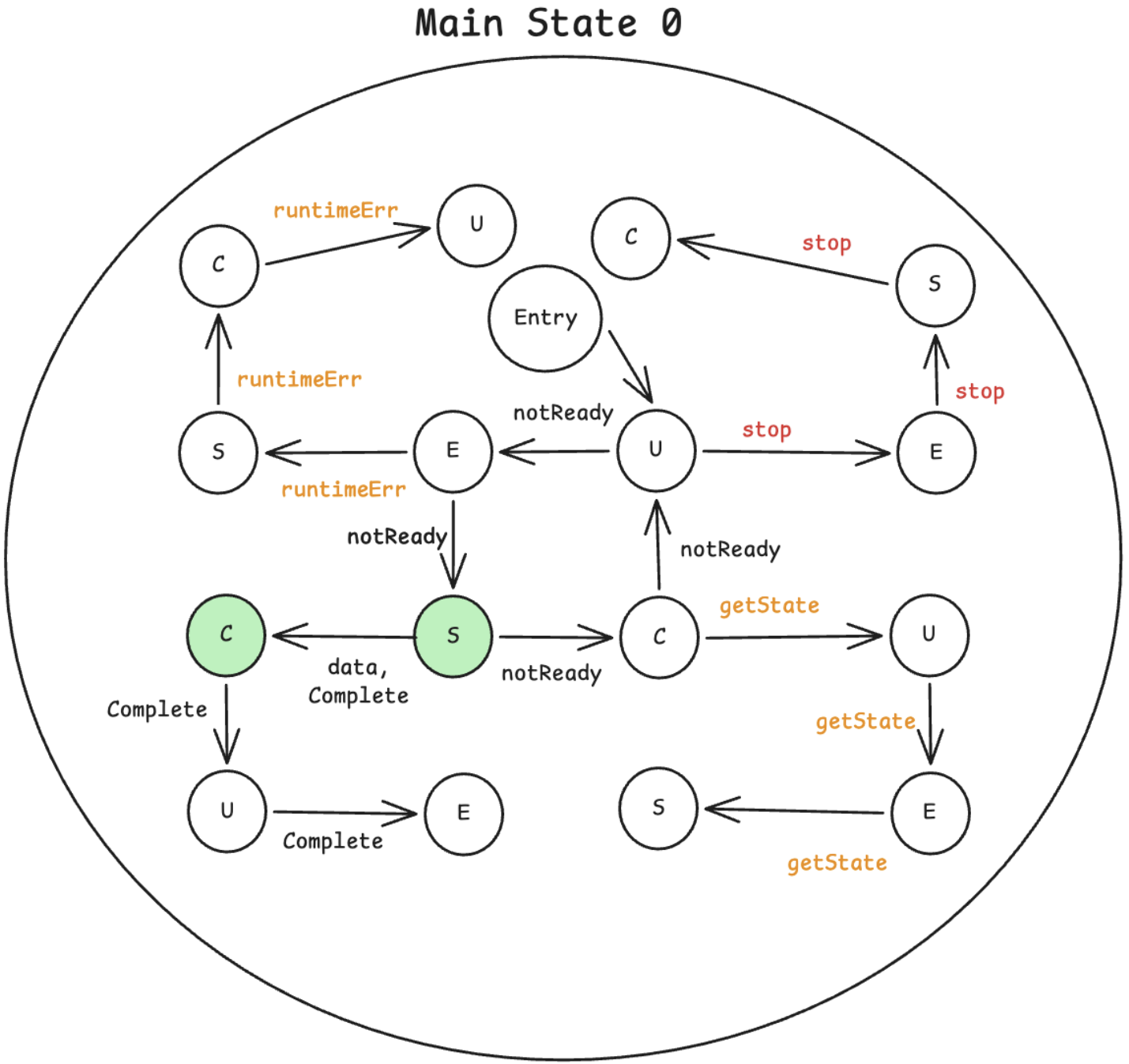
Global FSM Main State 0



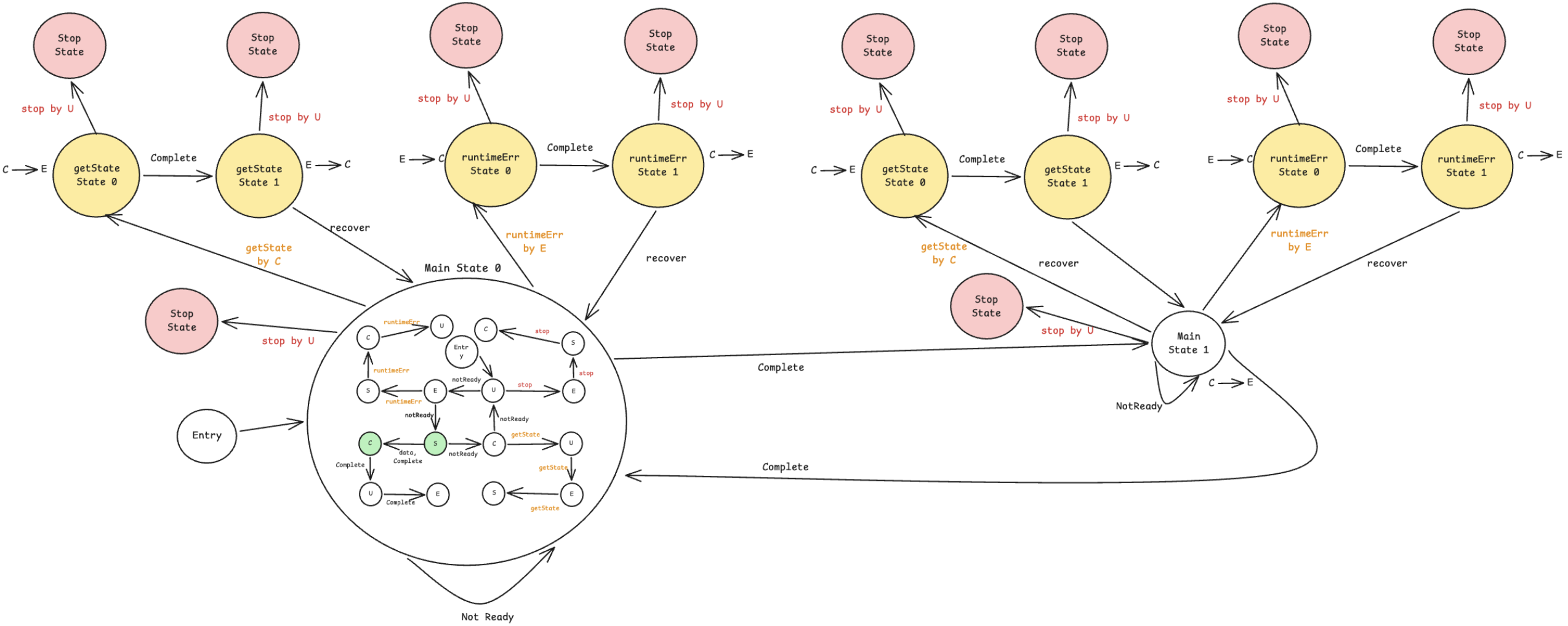
Inner FSM

Established sequence

U → E → S → C



Overall Picture



This is too difficult to do manually

- Designing such protocol is too intricate and error prone.
- We can not be sure that the implementation will be correct.

Multiparty Session Types (MPST)

- Formal framework which allows to verify protocol specification for **liveness, deadlock freedom, code conformance to the protocol**.
- Protocol Specification → Verification → **Correct by Construction**
  Code Generation → Code Conformance Check

Has constructs such as **recursion, transmissions and choice of transmissions**.

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec main_flow {
        @GeneralName mainData
        data from Sensor to Controller;
        @GeneralName mainCmd
        cmd from Controller to Executor;
        continue main_flow;
    }
}
```

Multiparty Session Types (MPST)

- Formal framework which allows to verify protocol specification for **liveness, deadlock freedom, code conformance to the protocol**.
- Protocol Specification → Verification → **Correct by Construction**
  Code Generation → Code Conformance Check

Has constructs such as **recursion, transmissions and choice of transmissions**.

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec main_flow {
        @GeneralName mainData
        data from Sensor to Controller;
        @GeneralName mainCmd
        cmd from Controller to Executor;
        continue main_flow;
    }
}
```

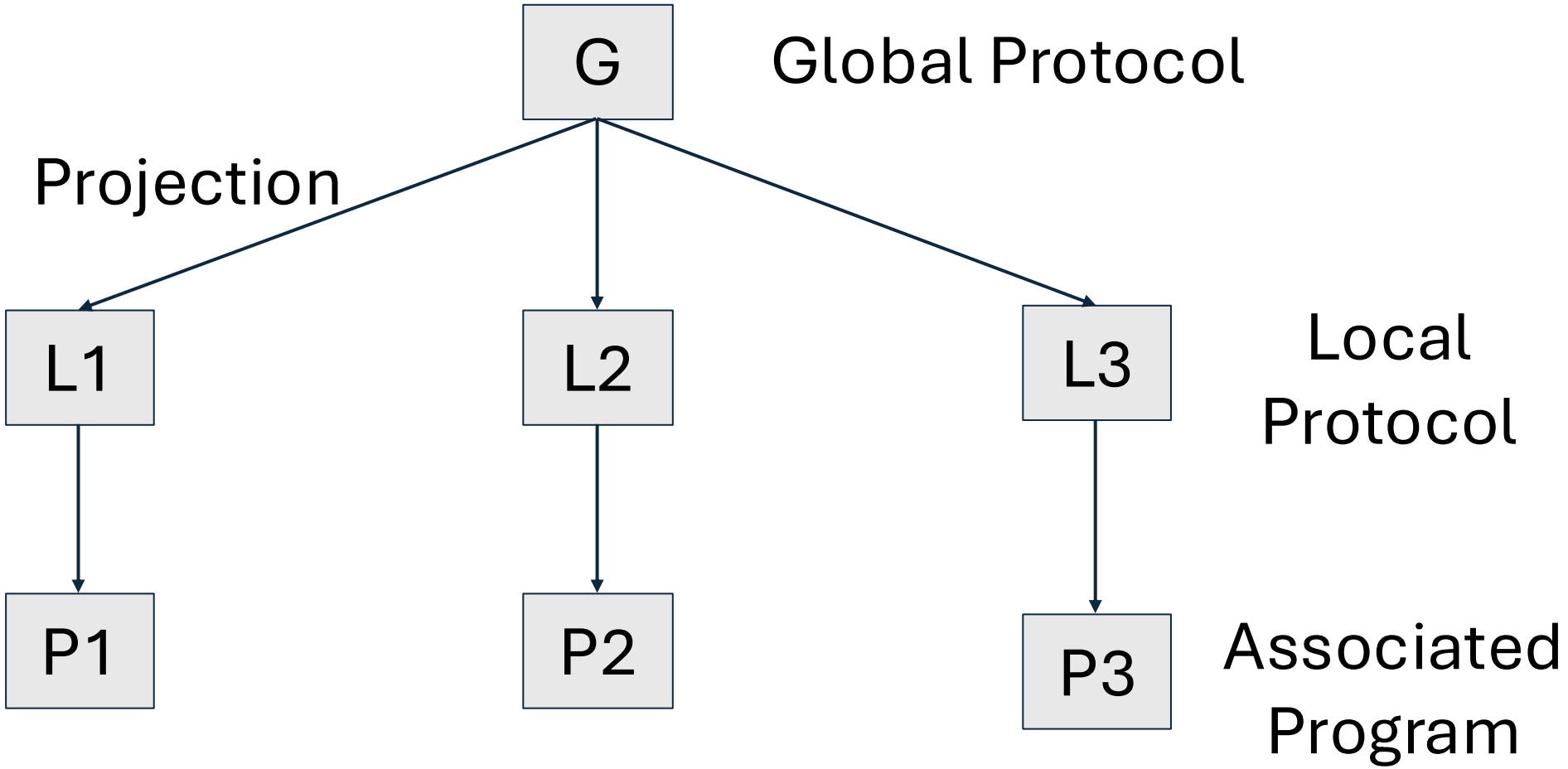
Multiparty Session Types (MPST)

- Formal framework which allows to verify protocol specification for **liveness, deadlock freedom, code conformance to the protocol**.
- Protocol Specification → Verification → **Correct by Construction**
  Code Generation → Code Conformance Check

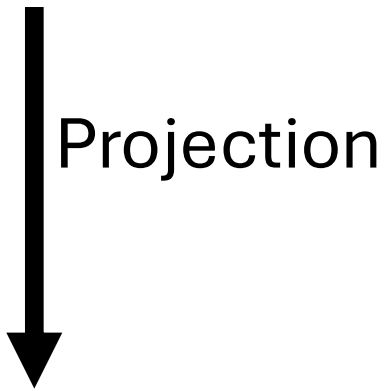
Has constructs such as **recursion, transmissions and choice of transmissions**.

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec main_flow {
        @GeneralName mainData
        data from Sensor to Controller;
        @GeneralName mainCmd
        cmd from Controller to Executor;
        continue main_flow;
    }
}
```

Top-Down Approach



```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec main_flow {
        @GeneralName mainData
        data from Sensor to Controller;
        @GeneralName mainCmd
        cmd from Controller to Executor;
        continue main_flow;
    }
}
```




```
projection of reliable role Sensor{
    rec RecT0 {
        data from Sensor to Controller;
        continue RecT0;
    }
}
```

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec main_flow {
        @GeneralName mainData
        data from Sensor to Controller;
        @GeneralName mainCmd
        cmd from Controller to Executor;
        continue main_flow;
    }
}
```

Projection

```
projection of reliable role Sensor{
    rec RecT0 {
        data from Sensor to Controller;
        continue RecT0;
    }
}
```

Generation

```
abstract class Sensor {
    type sensorStateType
    val sensorState: sensorStateType
    case class Data()

    def mainDataSending(state: sensorState.type): Data

    type Sensor[Ch0 <: OutChannel[Data]] = Rec[RecT0, Out[Ch0, Data] >>: Loop[RecT0]]
    def sensor(ch0: OutChannel[Data]): Sensor[ch0.type] = {
        rec(RecT0) {
            println("-- Sensor entering recursion body; t = RecT0")
            println(s"-- Sensor sending Data on ch0 ($ch0)")
            send(ch0, mainDataSending(sensorState)) >> {
                println(s"-- Sensor sent Data on ch0 ($ch0)")
                loop(RecT0)
            }
        }
    }
```

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec main_flow {
        @GeneralName mainData
        data from Sensor to Controller;
        @GeneralName mainCmd
        cmd from Controller to Executor;
        continue main_flow;
    }
}
```

Projection

```
projection of reliable role Sensor{
    rec RecT0 {
        data from Sensor to Controller;
        continue RecT0;
    }
}
```

Generation

```
abstract class Sensor {
    type sensorStateType
    val sensorState: sensorStateType
    case class Data()

    def mainDataSending(state: sensorState.type): Data

    type Sensor[Ch0 <: OutChannel[Data]] = Rec[RecT0, Out[Ch0, Data] >>: Loop[RecT0]]
    def sensor(ch0: OutChannel[Data]): Sensor[ch0.type] = {
        rec(RecT0) {
            println("-- Sensor entering recursion body; t = RecT0")
            println(s"-- Sensor sending Data on ch0 ($ch0)")
            send(ch0, mainDataSending(sensorState)) >> {
                println(s"-- Sensor sent Data on ch0 ($ch0)")
                loop(RecT0)
            }
        }
    }
```

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec main_flow {
        @GeneralName mainData
        data from Sensor to Controller;
        @GeneralName mainCmd
        cmd from Controller to Executor;
        continue main_flow;
    }
}
```

Projection

```
projection of reliable role Sensor{
    rec RecT0 {
        data from Sensor to Controller;
        continue RecT0;
    }
}
```

Generation

```
abstract class Sensor {
    type sensorStateType
    val sensorState: sensorStateType
    case class Data()

    def mainDataSending(state: sensorState.type): Data

    type Sensor[Ch0 <: OutChannel[Data]] = Rec[RecT0, Out[Ch0, Data] >>: Loop[RecT0]]
    def sensor(ch0: OutChannel[Data]): Sensor[ch0.type] = {
        rec(RecT0) {
            println("-- Sensor entering recursion body; t = RecT0")
            println(s"-- Sensor sending Data on ch0 ($ch0)")
            send(ch0, mainDataSending(sensorState)) >> {
                println(s"-- Sensor sent Data on ch0 ($ch0)")
                loop(RecT0)
            }
        }
    }
```

Implementation

```
abstract class Sensor {
    type sensorStateType
    val sensorState: sensorStateType
    case class Data()

    def mainDataSending(state: sensorState.type): Data

    type Sensor[Ch0 <: OutChannel[Data]] = Rec[RecT0, Out[Ch0, Data] >>: Loop[RecT0]]
    def sensor(ch0: OutChannel[Data]): Sensor[ch0.type] = {
        rec(RecT0) {
            println("-- Sensor entering recursion body; t = RecT0")
            println(s"-- Sensor sending Data on ch0 ($ch0)")
            send(ch0, mainDataSending(sensorState)) >> {
                println(s"-- Sensor sent Data on ch0 ($ch0)")
                loop(RecT0)
            }
        }
    }
```

Abstract Skeleton to Concrete Implementation

```
class ConcreteSensor extends  Sensor {

    override def mainDataSending(state: sensorState.type): Data = {
        ???
    }
```

Interruptible Blocks Abstraction

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    interruptible priority2 {
        interruptible priority1 {
            rec main_flow {
                data from Sensor to Controller;
                cmd from Controller to Executor;
                continue main_flow;
            }
        } with {
            getState by Controller () => {
                request from Controller to Executor;
                state from Executor to Controller;
                recover;
            };
            runtimeErr by Executor () => {
                error from Executor to Controller;
                repairCmd from Controller to Executor;
                recover;
            };
        };
    } with {
        stop by User;
    };
}
```

Interruptible Blocks Abstraction

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    interruptible priority2 {
        interruptible priority1 {
            rec main_flow {
                data from Sensor to Controller;
                cmd from Controller to Executor;
                continue main_flow;
            }
        } with {
            getState by Controller () => {
                request from Controller to Executor;
                state from Executor to Controller;
                recover;
            };
            runtimeErr by Executor () => {
                error from Executor to Controller;
                repairCmd from Controller to Executor;
                recover;
            };
        };
    } with {
        stop by User;
    };
}
```

Interruptible Blocks Abstraction

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    interruptible priority2 {
        interruptible priority1 {
            rec main_flow {
                data from Sensor to Controller;
                cmd from Controller to Executor;
                continue main_flow;
            }
        } with {
            getState by Controller () => {
                request from Controller to Executor;
                state from Executor to Controller;
                recover;
            };
            runtimeErr by Executor () => {
                error from Executor to Controller;
                repairCmd from Controller to Executor;
                recover;
            };
        };
    } with {
        stop by User;
    };
}
```

Interruptible Blocks Abstraction

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    interruptible priority2 {
        interruptible priority1 {
            rec main_flow {
                data from Sensor to Controller;
                cmd from Controller to Executor;
                continue main_flow;
            }
        } with {
            getState by Controller () => {
                request from Controller to Executor;
                state from Executor to Controller;
                recover;
            };
            runtimeErr by Executor () => {
                error from Executor to Controller;
                repairCmd from Controller to Executor;
                recover;
            };
        };
    } with {
        stop by User;
    };
}
```

Interruptible Blocks Abstraction

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    interruptible priority2 {
        interruptible priority1 {
            rec main_flow {
                data from Sensor to Controller;
                cmd from Controller to Executor;
                continue main_flow;
            }
        } with {
            getState by Controller () => {
                request from Controller to Executor;
                state from Executor to Controller;
                recover;
            };
            runtimeErr by Executor () => {
                error from Executor to Controller;
                repairCmd from Controller to Executor;
                recover;
            };
        };
    } with {
        stop by User;
    };
}
```

WE CAN ENCODE IT!

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    rec priority1_0 {
        choice at User {
            stop_priority2_User from User to Executor;
            stop_priority2_User from Executor to Sensor;
            stop_priority2_User from Sensor to Controller;
        } or {
            hB from User to Executor;
            choice at Executor {
                runtimeErr_priority1_Executor from Executor to Sensor;
                runtimeErr_priority1_Executor from Sensor to Controller;
                runtimeErr_priority1_Executor from Controller to User;
                rec priority1_1 {
                    choice at User {
                        stop_priority2_User from User to Executor;
                        stop_priority2_User from Executor to Controller;
                        stop_priority2_User from Controller to Sensor;
                    } or {
                        hB from User to Executor;
                        choice at Executor {
                            hB from Executor to Controller;
                            hB from Controller to Sensor;
                            hB from Sensor to User;
                            continue priority1_1;
                        } or {
                            error from Executor to Controller;
                            hBNext_ygD8z7QImJ7dxAEG from Controller to Sensor;
                            hBNext_ygD8z7QImJ7dxAEG from Sensor to User;
                            rec priority1_2 {
                                choice at User {
                                    stop_priority2_User from User to Controller;
                                    stop_priority2_User from Controller to Executor;
                                    stop_priority2_User from Executor to Sensor;
                                } or {
                                    hB from User to Controller;
                                    choice at Controller {
                                        hB from Controller to Executor;
                                        hB from Executor to Sensor;
                                        hB from Sensor to User;
                                        continue priority1_2;
                                    } or {
                                        repairCmd from Controller to Executor;
                                        hBNext_Fbar7ZkxQxnp10CB from Executor to Sensor;
                                        hBNext_Fbar7ZkxQxnp10CB from Sensor to User;
                                        continue priority1_0;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    } or {
        hB from Executor to Sensor;
        choice at Sensor {
            hB from Sensor to Controller;
            choice at Controller {
                getState_priority1_Controller from Controller to User;
                getState_priority1_Controller from User to Executor;
                getState_priority1_Controller from Executor to Sensor;
                rec priority1_1 {
                    choice at User {
                        stop_priority2_User from User to Controller;
                        stop_priority2_User from Controller to Executor;
                        stop_priority2_User from Executor to Sensor;
                    } or {
                        hB from User to Controller;
                        choice at Controller {
                            hB from Controller to Executor;
                            hB from Executor to Sensor;
                            hB from Sensor to User;
                            continue priority1_1;
                        } or {
                            request from Controller to Executor;
                            hBNext_H1i5FFsk0QjXrDiI from Executor to Sensor;
                            hBNext_H1i5FFsk0QjXrDiI from Sensor to User;
                            rec priority1_2 {
                                choice at User {
                                    stop_priority2_User from User to Executor;
                                    stop_priority2_User from Executor to Controller;
                                    stop_priority2_User from Controller to Sensor;
                                } or {
                                    hB from User to Executor;
                                    choice at Executor {
                                        hB from Executor to Controller;
                                        hB from Controller to Sensor;
                                        hB from Sensor to User;
                                        continue priority1_2;
                                    } or {
                                        state from Executor to Controller;
                                        hBNext_S6aXNcpTdl7Wpwnt from Controller to Sensor;
                                        hBNext_S6aXNcpTdl7Wpwnt from Sensor to User;
                                        continue priority1_0;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    } or {
        hB from Controller to User;
        hB from User to Executor;
        hB from Executor to Sensor;
        continue priority1_0;
    }
} or {
    data from Sensor to Controller;
    hBNext_Om6weQ85rIfJTzhW from Controller to User;
    hBNext_Om6weQ85rIfJTzhW from User to Executor;
    rec priority1_1 {
        choice at User {
            stop_priority2_User from User to Controller;
            stop_priority2_User from Controller to Executor;
        } or {
            hB from User to Controller;
            choice at Controller {
                getState_priority1_Controller from Controller to Executor;
                getState_priority1_Controller from Executor to User;
                rec priority1_2 {
                    choice at User {
                        stop_priority2_User from User to Controller;
                        stop_priority2_User from Controller to Executor;
                    } or {
                        hB from User to Controller;
                        choice at Controller {
                            hB from Controller to Executor;
                            hB from Executor to User;
                            continue priority1_2;
                        } or {
                            request from Controller to Executor;
                            hBNext_H1i5FFsk0QjXrDiI from Executor to User;
                            rec priority1_3 {
                                choice at User {
                                    stop_priority2_User from User to Executor;
                                    stop_priority2_User from Executor to Controller;
                                } or {
                                    hB from User to Executor;
                                    choice at Executor {
                                        hB from Executor to Controller;
                                        hB from Controller to User;
                                        continue priority1_3;
                                    } or {
                                        state from Executor to Controller;
                                        hBNext_S6aXNcpTdl7Wpwnt from Controller to User;
                                        continue priority1_1;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        } or {
            hB from Executor to User;
            hB from User to Controller;
            continue priority1_1;
        }
    } or {
        cmd from Controller to Executor;
        hBNext_BNxbITYyig3LGJ35 from Executor to User;
        continue priority1_0;
    }
}
```

Annotations

Callback functions that simplify enrichment of skeleton abstract classes.

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    interruptible priority2 {
        interruptible priority1 {
            rec main_flow {
                @GeneralName mainData
                data from Sensor to Controller;
                @GeneralName mainCmd
                cmd from Controller to Executor;
                continue main_flow;
            }
        } with {
            getState by Controller () => {
                @GeneralName requestState
                request from Controller to Executor;
                @GeneralName returnState
                state from Executor to Controller;
                recover;
            };
            runtimeErr by Executor () => {
                @GeneralName notifyError
                error from Executor to Controller;
                @GeneralName repairResponse
                repairCmd from Controller to Executor;
                recover;
            };
        };
    } with {
        stop by User;
    };
}
```

Annotations

Callback functions that simplify enrichment of skeleton abstract classes.

```
global protocol Sensor(reliable role User,reliable role Executor,
reliable role Sensor,reliable role Controller){
    interruptible priority2 {
        interruptible priority1 {
            rec main_flow {
                @GeneralName mainData
                data from Sensor to Controller;
                @GeneralName mainCmd
                cmd from Controller to Executor;
                continue main_flow;
            }
        } with {
            getState by Controller () => {
                @GeneralName requestState
                request from Controller to Executor;
                @GeneralName returnState
                state from Executor to Controller;
                recover;
            };
            runtimeErr by Executor () => {
                @GeneralName notifyError
                error from Executor to Controller;
                @GeneralName repairResponse
                repairCmd from Controller to Executor;
                recover;
            };
        };
    } with {
        stop by User;
    };
}
```

Types of Interrupts

- Fire and Notify
- Preemptive Interrupts
- Priority Interrupts
- Recoverable Interrupts
- Rollback Interrupts
- Masking

Conclusion

- Interruptible protocols and use cases
- Possible design solutions
- MPST and interruptible protocols encoding
- Protocol FSM abstraction with annotations