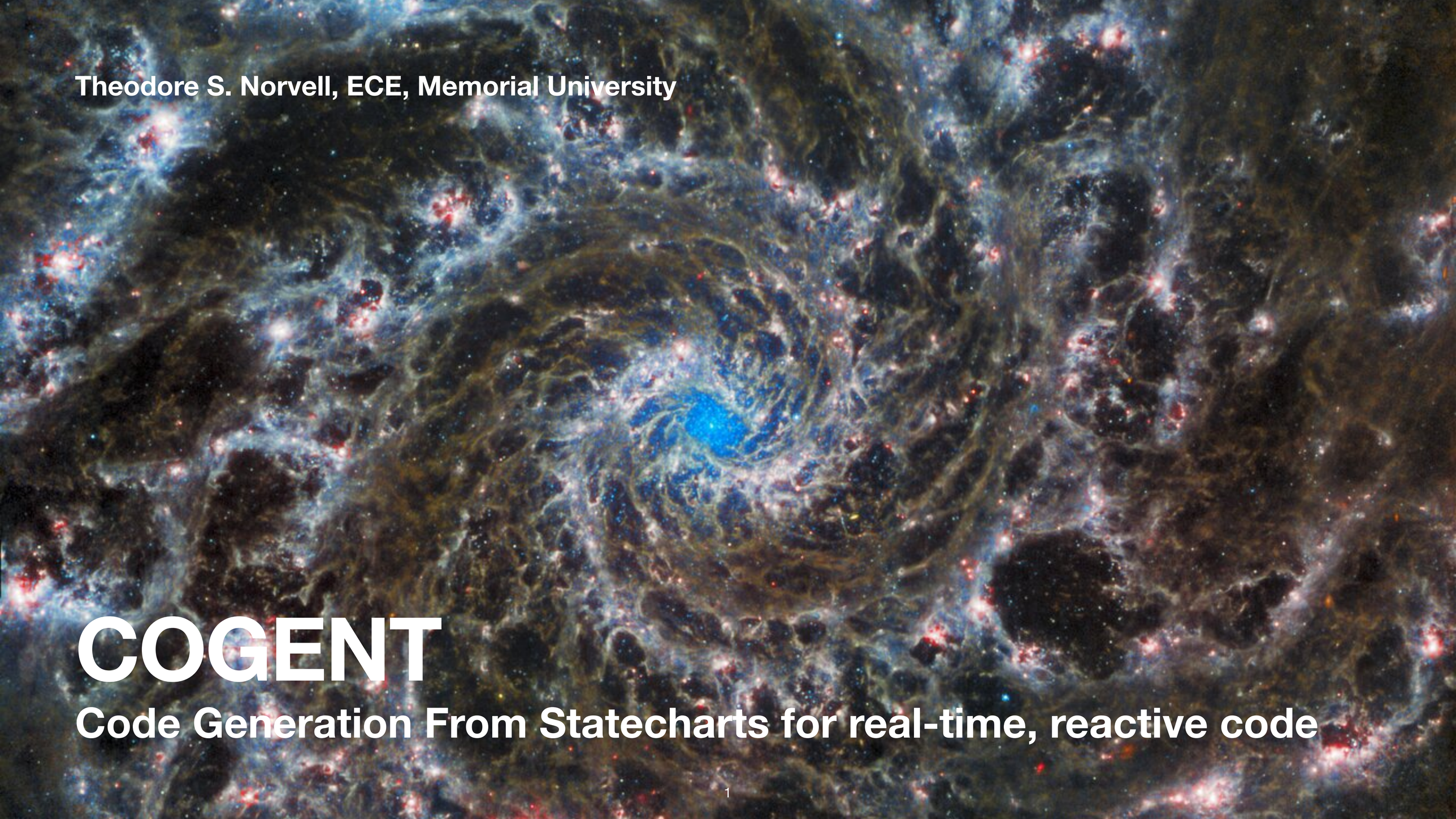


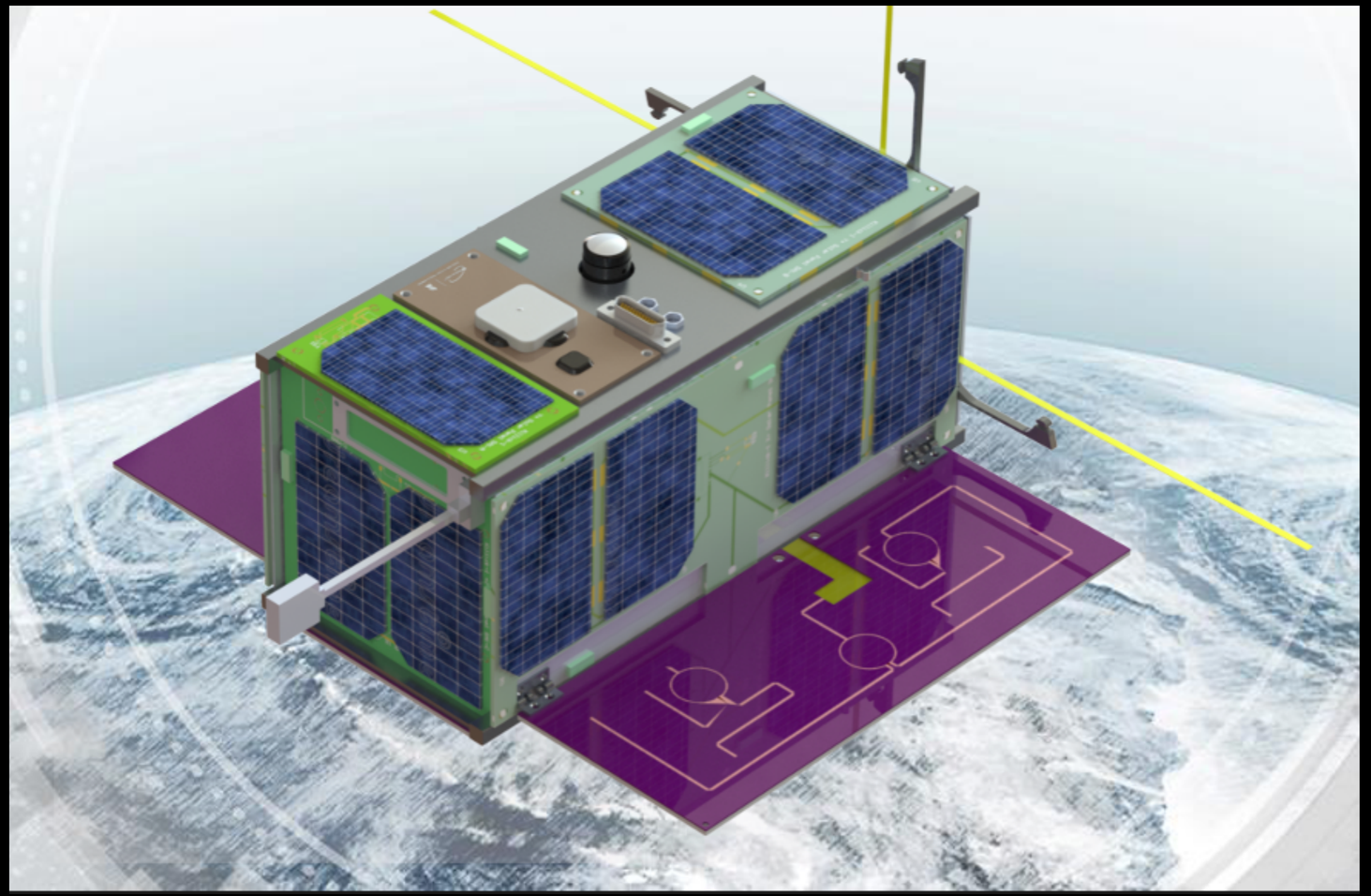
Theodore S. Norvell, ECE, Memorial University

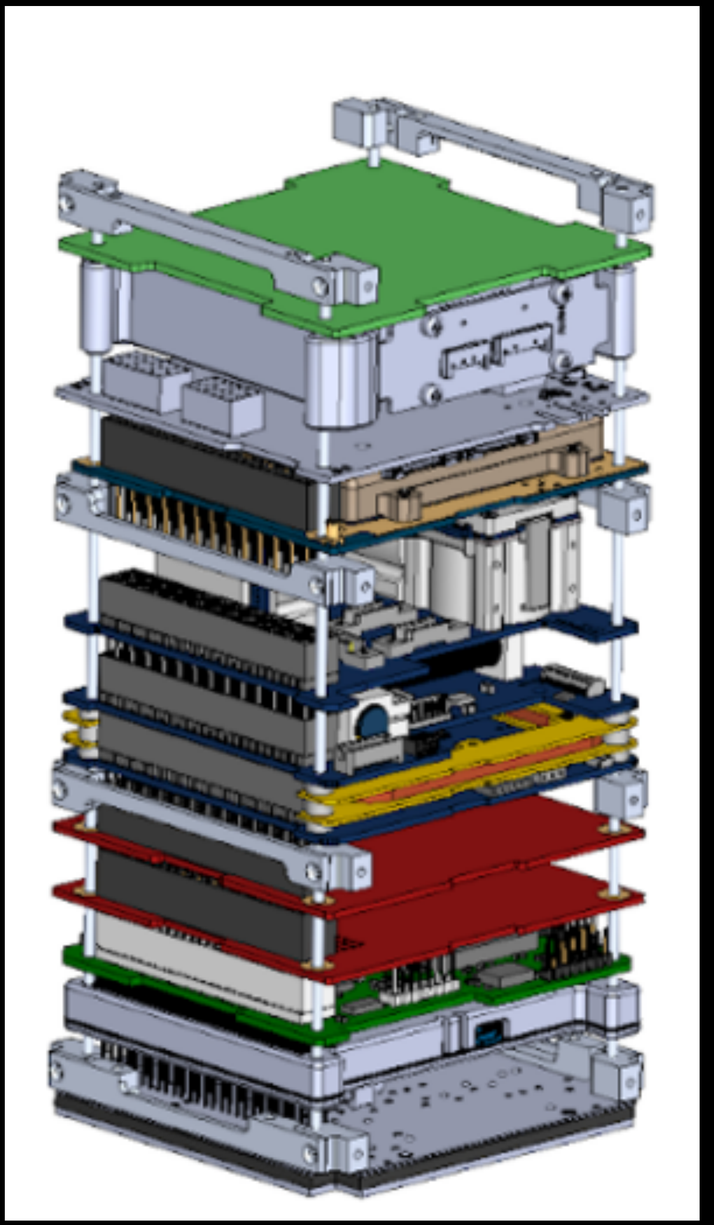
COGENT

Code Generation From Statecharts for real-time, reactive code

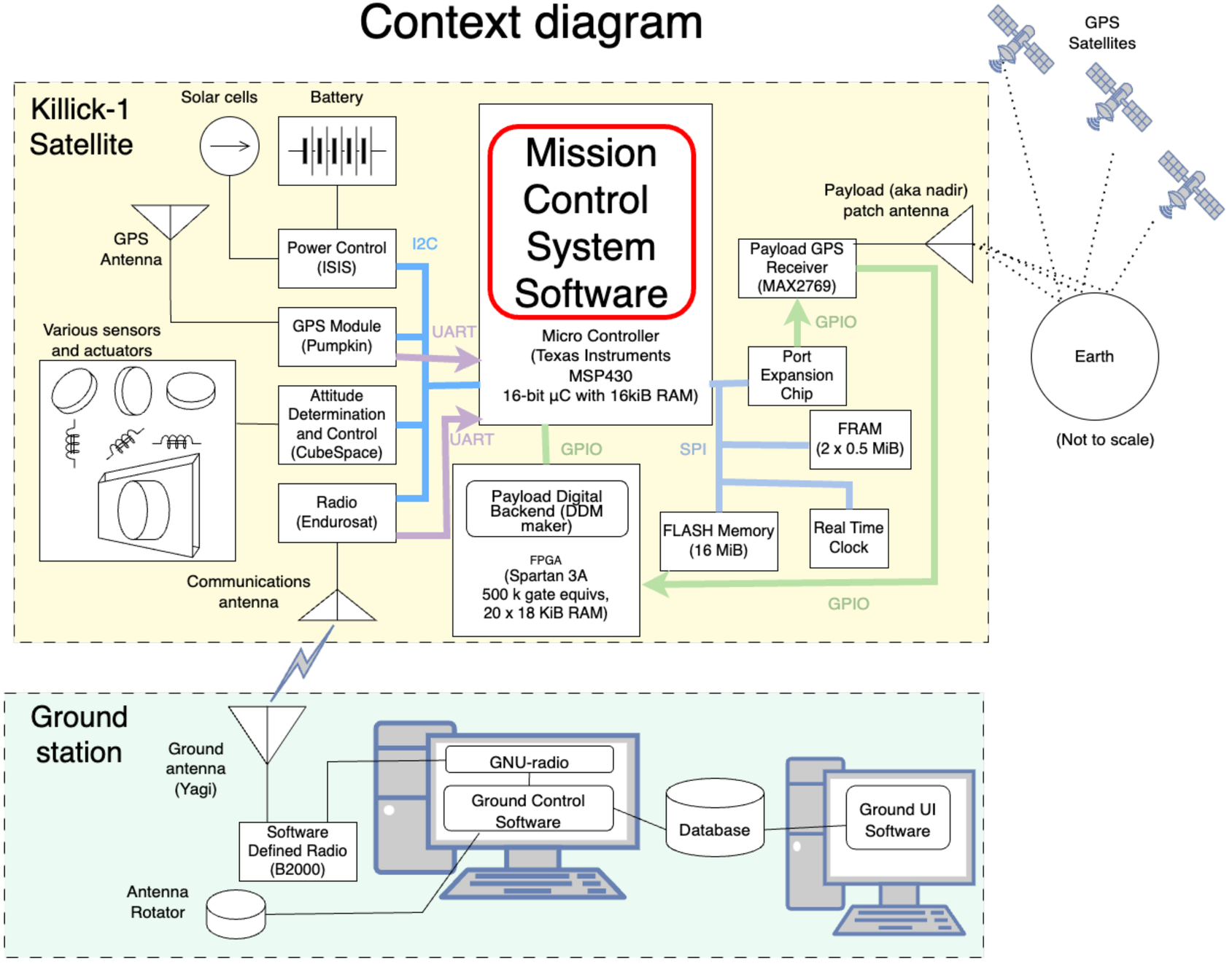
Context: The onboard mission control software for the Killick-1 satellite

https://github.com/theodore-norvell/cogent

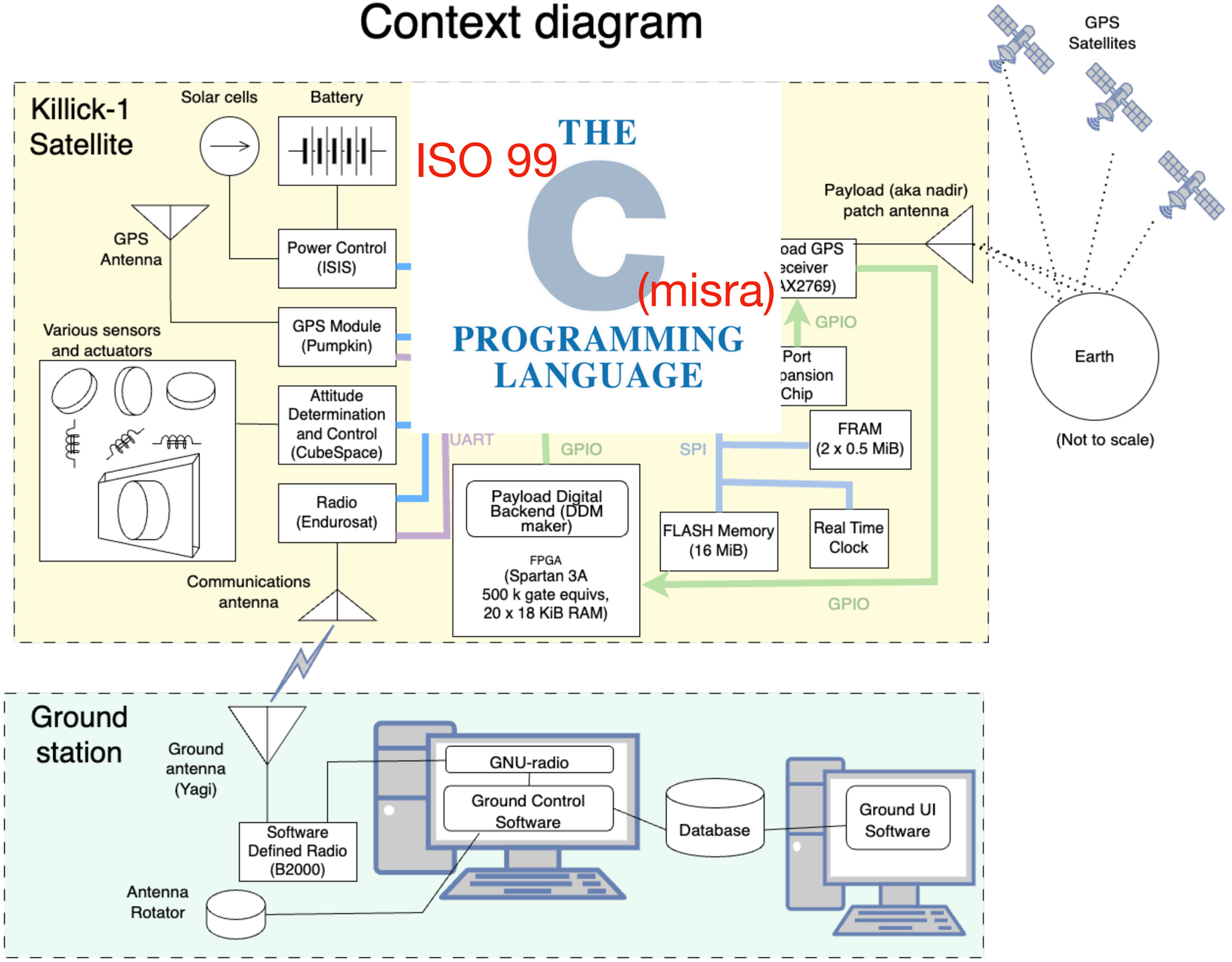




Context diagram



Context diagram



Problem: How can we describe behaviour?

https://github.com/theodore-norvell/cogent

How do you tell a story?

Structured programming v. unstructured flowcharts

or

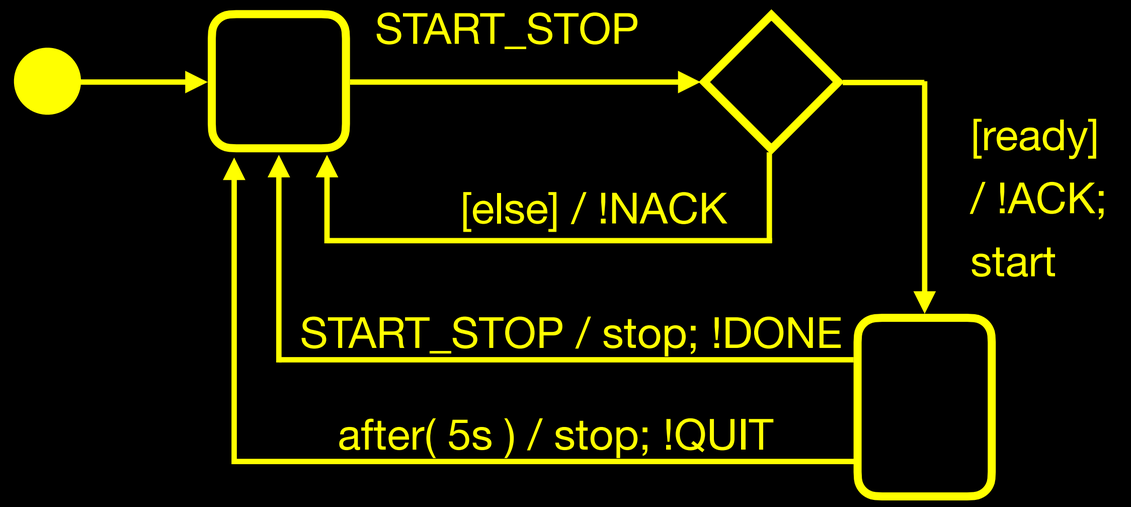
EBNF grammars v. recursive transition nets

https://github.com/theodore-norvell/cogent

How do you tell a story about interaction?

Structured narrative v. states and reactions

```
loop
    wait
        START_STOP ->
            if ready( )
                send ACK; start( )
                wait
                    START_STOP -> stop() ; send DONE
                    AFTER(5s) -> stop() ; send QUIT
            else
                send NAK
```



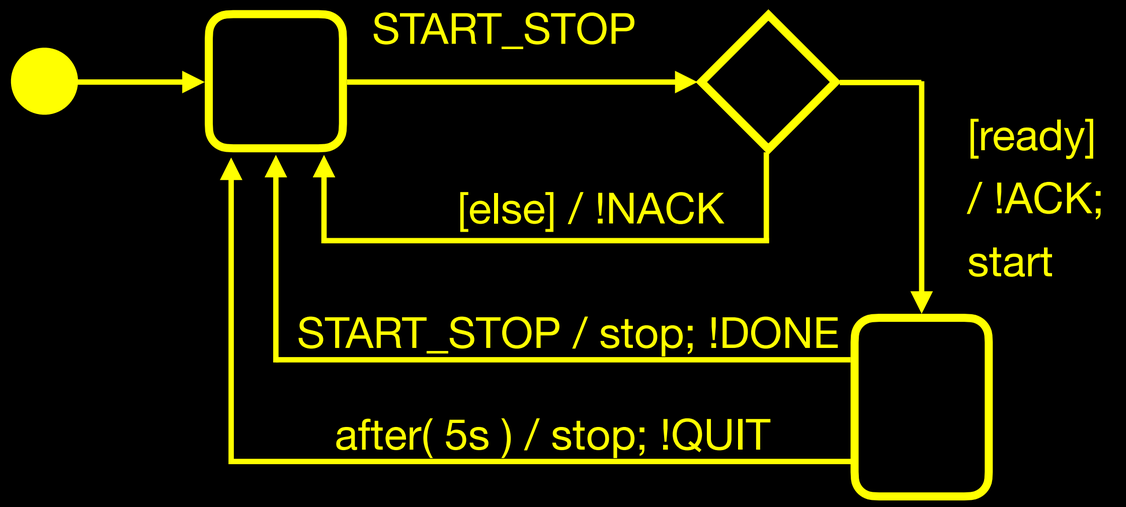
https://github.com/theodore-norvell/cogent

How do you tell a story about interaction?

Structured narrative v. states and reactions

Process Algebra

```
loop
    wait
        START_STOP ->
            if ready( )
                send ACK; start( )
                wait
                    START_STOP -> stop() ; send DONE
                    AFTER(5s) -> stop() ; send QUIT
            else
                send NAK
```



State Machine

https://github.com/theodore-norvell/cogent

Which is better?

Structured narrative v. states and reactions

https://github.com/theodore-norvell/cogent

Which is better?

Structured narrative v. states and reactions

Whatever gets you through the night
It's all right, it's all right
It's your money or your life
It's all right, it's all right
Don't need a sword to cut through' flowers
Oh no, oh no
– John Lennon

https://github.com/theodore-norvell/cogent

Which is better?

Structured narrative v. states and reactions

- Me at NECEC 2015 : Process algebra is better
  - It's code. It's structured. It's familiar. It's hierarchical. It allows nestable concurrency.
  - I showed how to create a process algebra library in fancy OO languages such as JavaScript, Python, Java, C++.
  - I used monads and lambda expressions! Not practical in C.
- Me in 2021: We're programming the tiny OBC of a satellite in C.
  - Narrative approach is possible in C.
  - But I didn't know that. It does not have nestable concurrency.
  - So state machine it is.

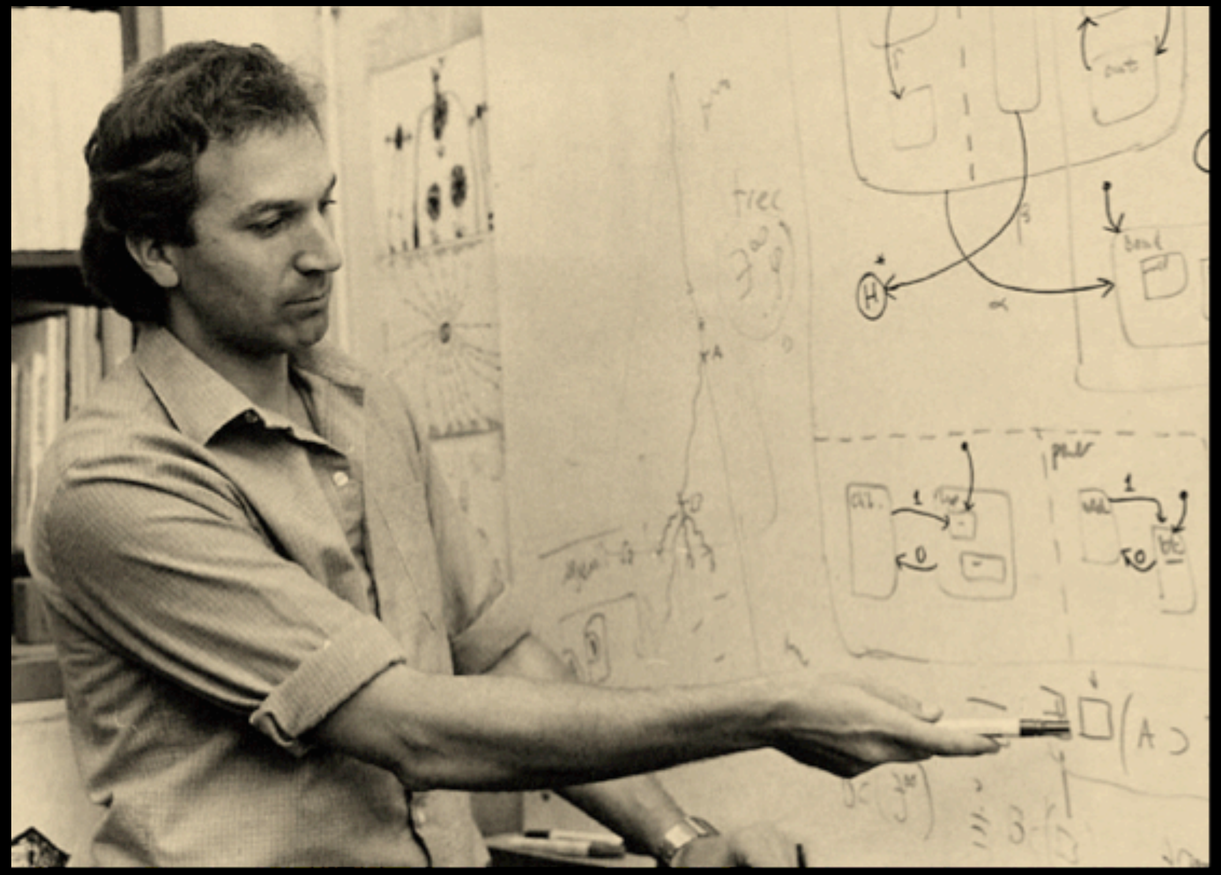
https://github.com/theodore-norvell/cogent

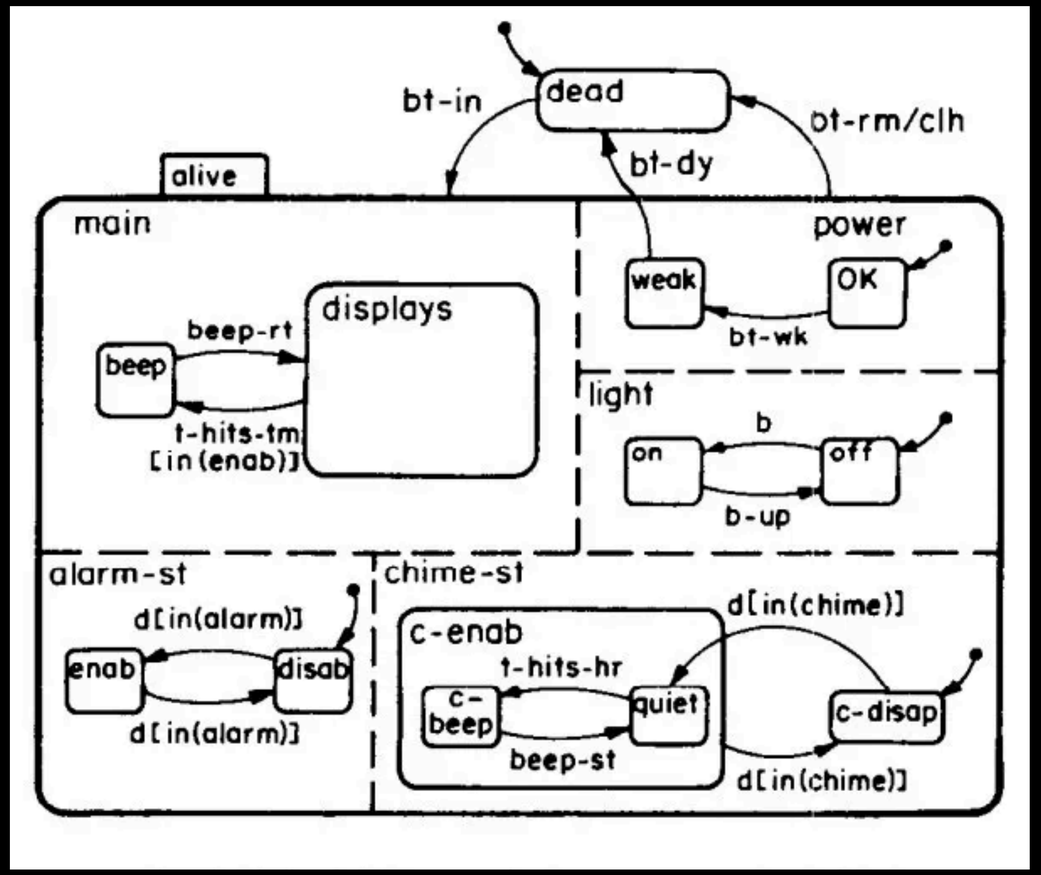
What's the *best* kind of state machine?
And why doesn't everyone use it?

https://github.com/theodore-norvell/cogent

Statecharts

David Harel 1987

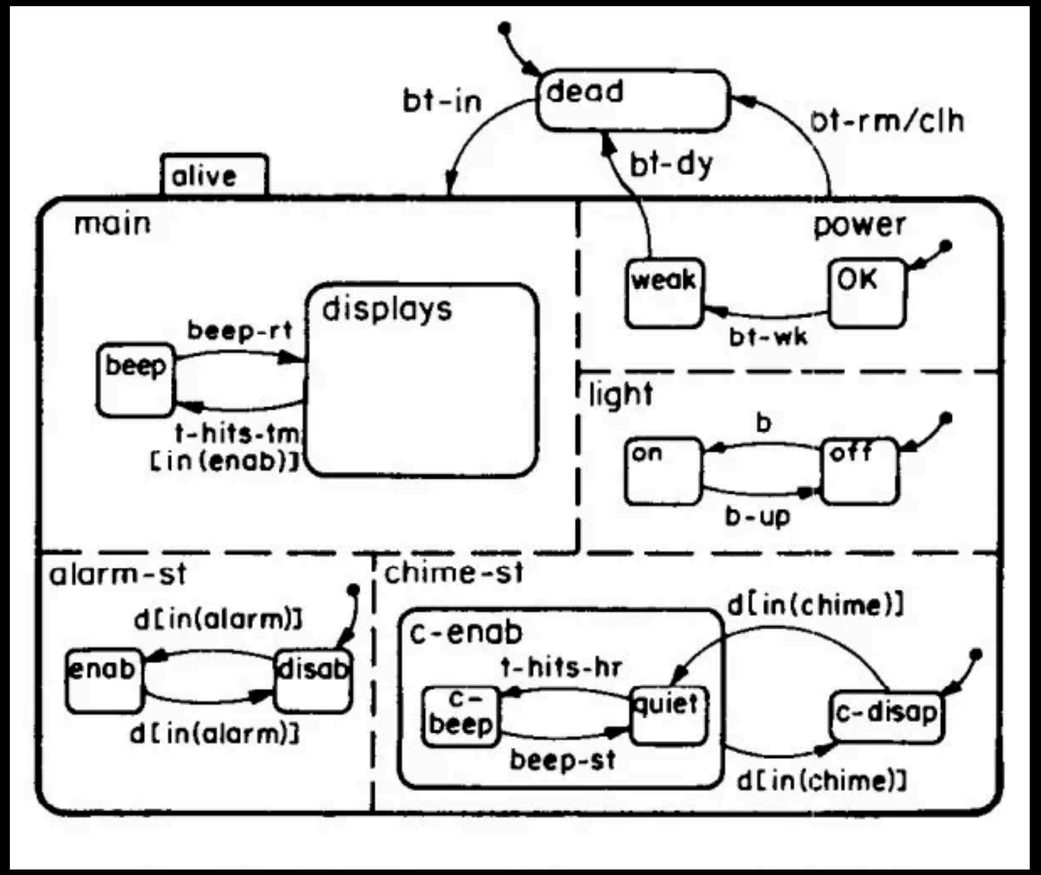




https://github.com/theodore-norvell/cogent

Statecharts
David Harel 1987

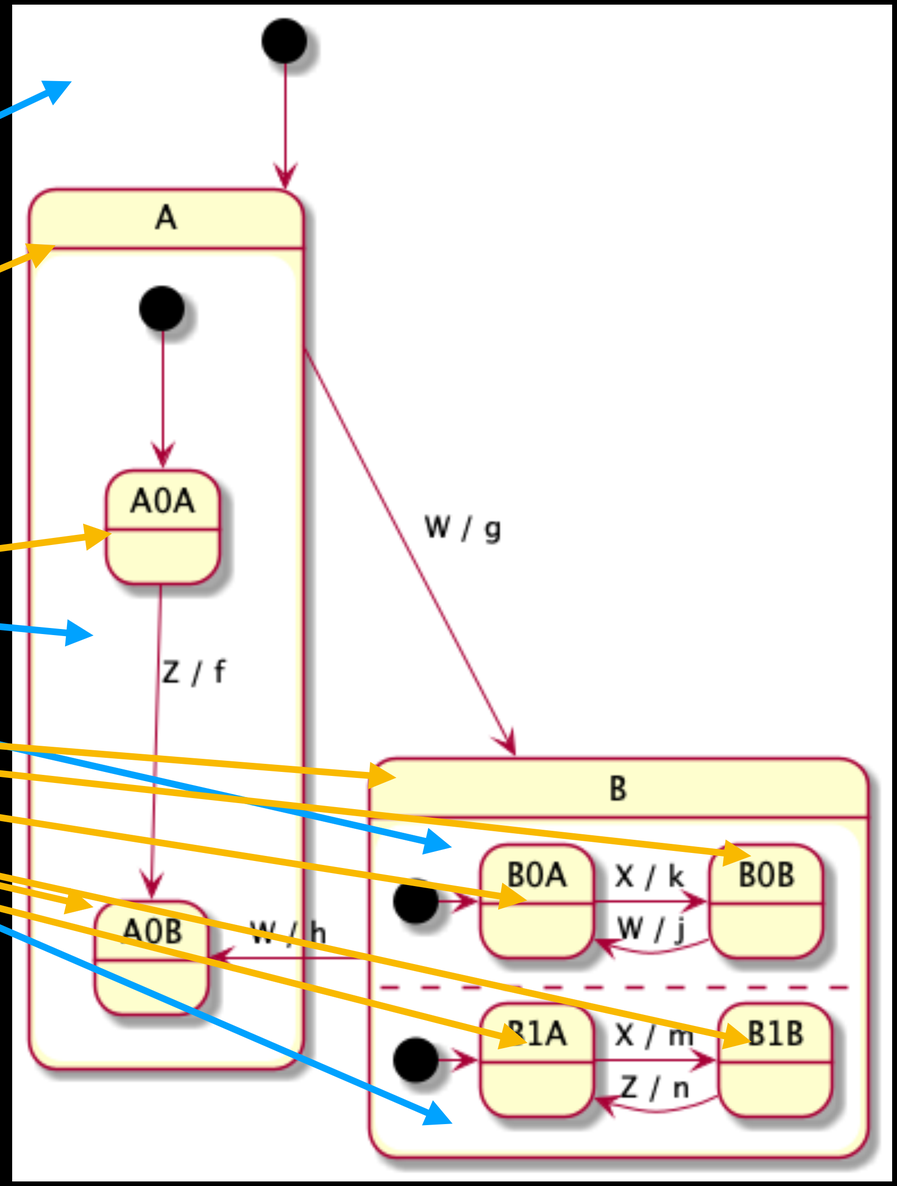
- Standardized by UML
- State Charts combine
  - State Machines
  - Hierarchy
  - Nestable concurrency
- Concurrency is non-preemtive



https://github.com/theodore-norvell/cogent

Statecharts
David Harel 1987

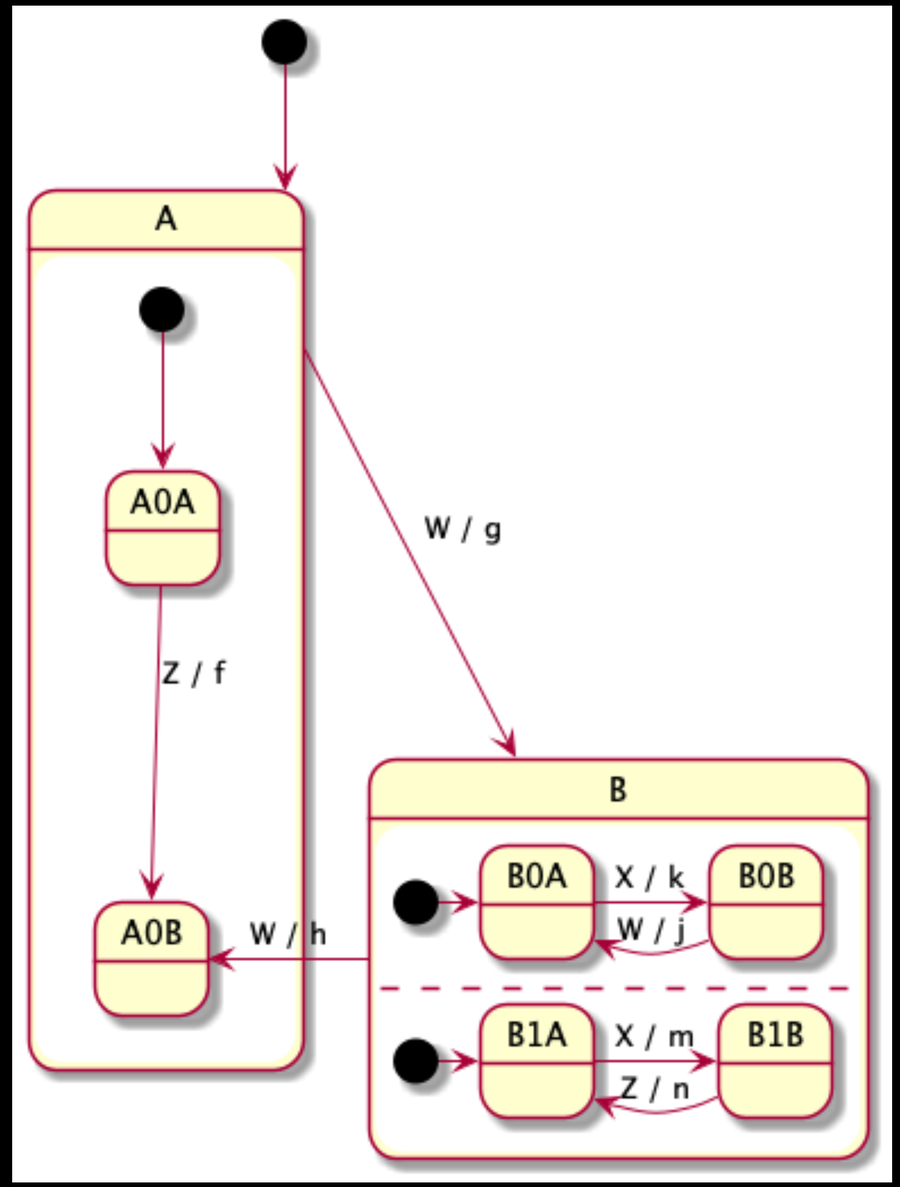
- State charts have
  - Regions
  - States



https://github.com/theodore-norvell/cogent

Statecharts
David Harel 1987

- The basic rules
  - The top region is always active
  - In an active region, 1 state is active
  - In an active state, all regions are active
  - If a state or region is active, so is its parent

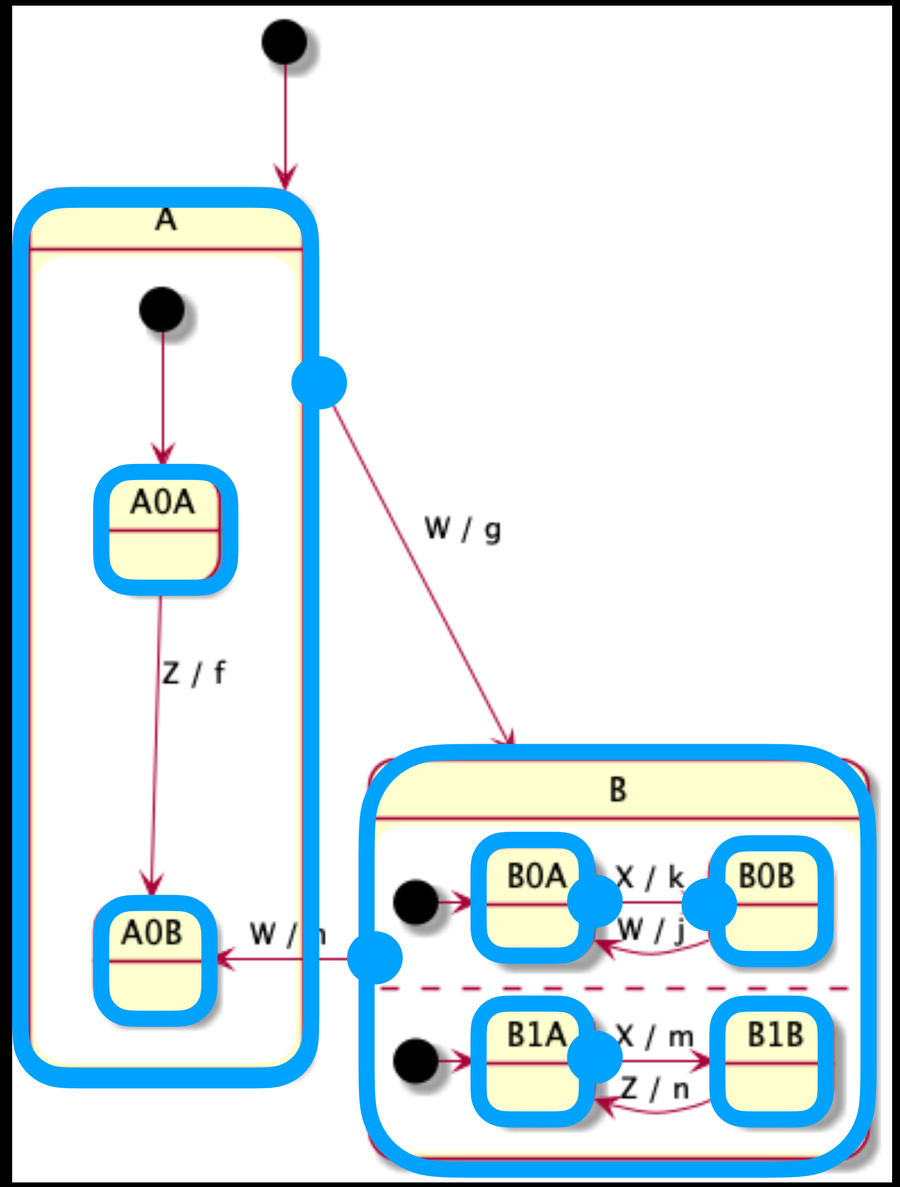


https://github.com/theodore-norvell/cogent

Statecharts

David Harel 1987

Events	Reactions
W	g
X	k; m or m; k
W	j
W	h



https://github.com/theodore-norvell/cogent

And why doesn't everyone use it?

https://github.com/theodore-norvell/cogent

Problems

- Statecharts are not a programming language, they are a modelling notation.
  - Models can be tricky to translate by hand to code in (e.g.) C.
  - Once translated, it is not easy for code and model to stay in sync.
- Statecharts are diagrams and diagrams are awkward
  - Diagrams don't fit well with tools such as "git diff" that show software evolution and help deal with merge conflicts.

https://github.com/theodore-norvell/cogent

Solutions

- Statecharts are not a programming language, they are a modelling notation.
  - Models can be tricky to translate by hand to code in (e.g.) C.
  - Once translated, it is not easy for code and model to stay in sync.
  - Automatically translate to code. (Not a new idea.)
- Statecharts are diagrams and diagrams are awkward
  - Diagrams don't fit well with tools such as "git diff" that show software evolution and help deal with merge conflicts.

https://github.com/theodore-norvell/cogent

Solutions

- Statecharts are not a programming language, they are a modelling notation.
  - Models can be tricky to translate by hand to code in (e.g.) C.
  - Once translated, it is not easy for code and model to stay in sync.
  - **Automatically translate to code. (Not a new idea.)**
- Statecharts are diagrams and diagrams are awkward
  - Diagrams don't fit well with tools such as "git diff" that show software evolution and help deal with merge conflicts.
  - **Use a simple textual representation of the diagrams.**

https://github.com/theodore-norvell/cogent

Solutions

- **Automatically translate to code. (Not a new idea.)**

  "Code as diagrams"

- **Use a simple textual representation of the diagrams.**

  "Diagrams as code"

https://github.com/theodore-norvell/cogent

Solutions

"Code as diagrams"

"Code as diagrams as code"

"Diagrams as code"

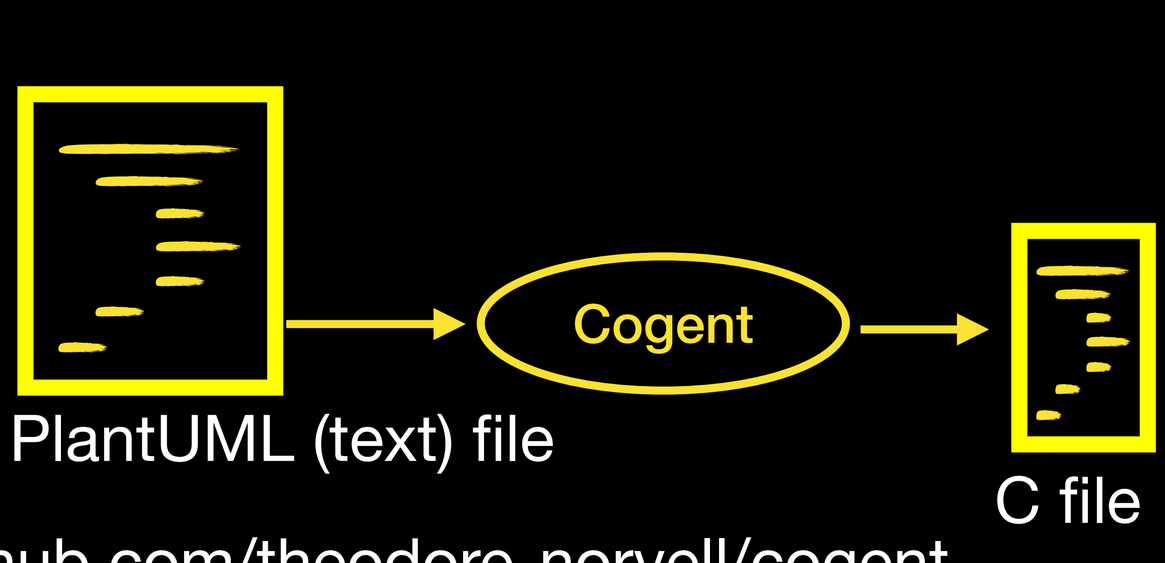
https://github.com/theodore-norvell/cogent

Solution: What is Cogent?

https://github.com/theodore-norvell/cogent

Cogent

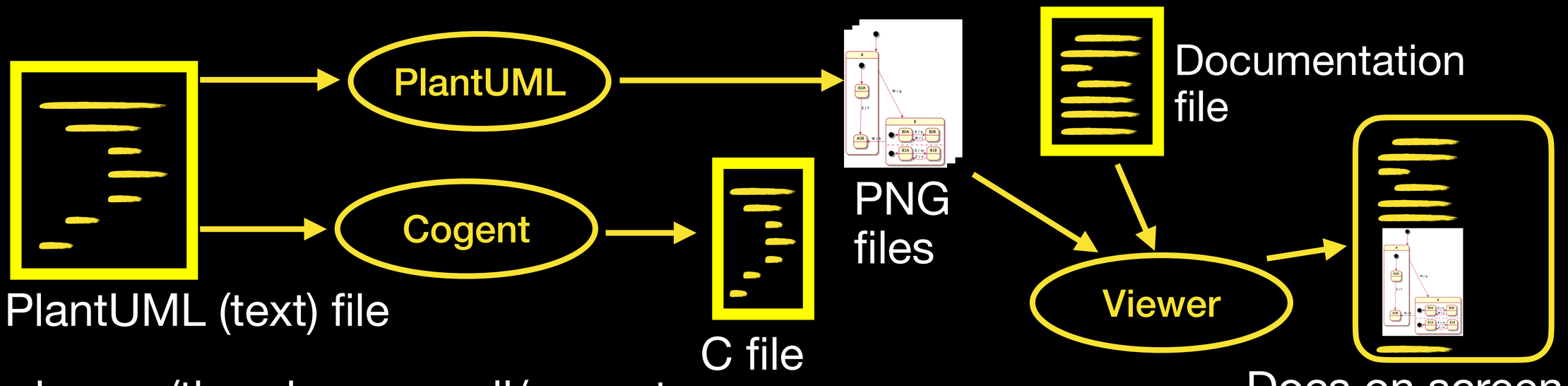
- **A simple translator from statecharts to (mostly) Misra compliant C.**
- **The input is in the PlantUML language**



https://github.com/theodore-norvell/cogent

Cogent

- **A simple translator from statecharts to (mostly) Misra compliant C.**
- **The input is in the PlantUML language**
- **The pre-existing PlantUML tool creates the diagrams as image files**



https://github.com/theodore-norvell/cogent

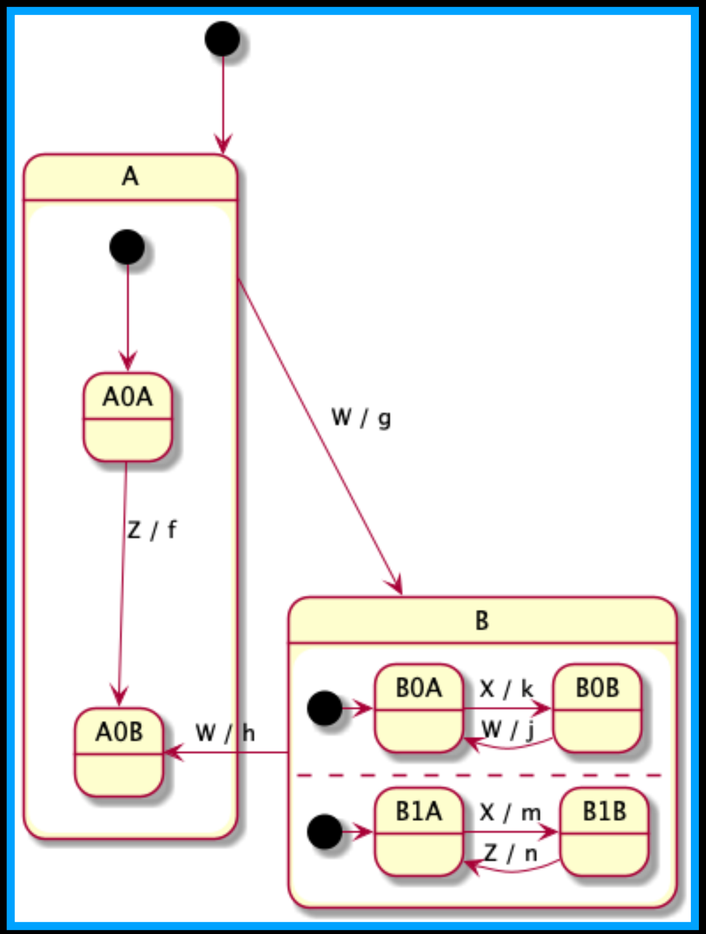
Input language
PlantUML

PlantUML is widely used tool for generating diagrams from textual descriptions.

It defines a set of related domain specific languages

It has a language for Statecharts

```
@startuml
    state A {
        state A0A
        state A0B
        [*] --> A0A
        A0A --> A0B : Z / f
    }
    state B {
        state B0A
        state B0B
        [*] -> B0A
        B0A -> B0B : X / k
        B0B -> B0A : W / j
        --
        state B1A
        state B1B
        [*] -> B1A
        B1A -> B1B : X / m
        B1B -> B1A : Z / n
    }
    [*] --> A
    A -> B : W / g
    B -> A0B : W / h
@enduml
```



https://github.com/theodore-norvell/cogent

Input language
PlantUML

Plant UML allows any text on transitions

Cogent imposes a syntax and interpretation on this text

`GO`	Triggered when a GO event arrives.
`after( 100 ms) [ ready and willing ]`	Triggered any time after 100 ms if conditions ready and willing are both true
`[ ! blocked ]`	Triggered as soon as condition blocked is false
`GO [ ! blocked ] / start ; !ACK`	Causes actions start and send_ACK.

https://github.com/theodore-norvell/cogent

Generated code interface

```
.
.
.
void initStateMachine_figs( TIME_T now) {
    .
    .
    .
}

bool_t dispatchEvent_figs( event_t *event_p, TIME_T now ) {
    .
    .
    .
}
.
.
.
```

Developer supplies:

- **Suitable event type**
- **Suitable TIME_T type**
- **Event dispatch loop**
- **One event kind per event trigger**
- **One procedure per condition**
- **One procedure per action**

https://github.com/theodore-norvell/cogent

Generated code

Fast and RAM efficient. No function pointers.

```
bool_t dispatchEvent_figs( event_t *event_p, TIME_T now ) {
    bool_t handledRoot = false ;
    switch( currentChild_a[ G_INDEX_root] ) {
            case L_INDEX_A : {
                bool_t handled_A = false ;
                /* Code for OR state 'A' */{
                    switch( currentChild_a[ G_INDEX_A] ) {
                        case L_INDEX_A0A : {
                            /* Code for basic state 'A0A' */{
                                /* Event handling code for state A0A */
                                switch( eventClassOf(event_p) ) {
                                    case EVENT(Z) : {
                                        status_t status = OK_STATUS ;
                                        handled_A = true ;
                                        /* Transition from A0A to A0B. */
                                        exit_A0A( -1 ) ;
                                        /* Code for action NamedAction(f). */
                                        LOG_ACTION_START( "f")
                                        status = ACTION(f)( event_p, status ) ;
                                        LOG_ACTION_DONE( "f")
                                        enter_A0B( -1, now ) ;
                                        handled_A = true ;
                                    } break ;
                                    default : { }
                                }
                            }/* End of basic state 'A0A' */
                        } break ;
                        case L_INDEX_A0B : { ...Code for A0B... } break ;
                        default : { assertUnreachable() ; }
                    }
                    if( ! handled_A  ) {
                        /* Event handling code for state A */
                        switch( eventClassOf(event_p) ) {
                            case EVENT(W) : {
                                status_t status = OK_STATUS ;
                                handled_A = true ;
                                /* Transition from A to B. */
                                exit_A( -1 ) ;
                                /* Code for action NamedAction(g). */
                                LOG_ACTION_START( "g")
                                status = ACTION(g)( event_p, status ) ;
                                LOG_ACTION_DONE( "g")
                                enter_B( -1, now ) ;
                            } break ;
                            default : { }
                        }
                    }
                }/* End of OR state 'A' */
                handled_Root = handled_A ;
            } break ;
            case L_INDEX_B : {
                bool_t handled_B = false ;
                ...code for both regions of B...
                ...code for B itself if handle_B is false...
                }
                handled_Root = handled_B ;
            } break ;
            default : { assertUnreachable() ; }
        }
        /* State root has no outgoing transitions. */
    }/* End of OR state 'root' */
    return handledRoot;
}
```

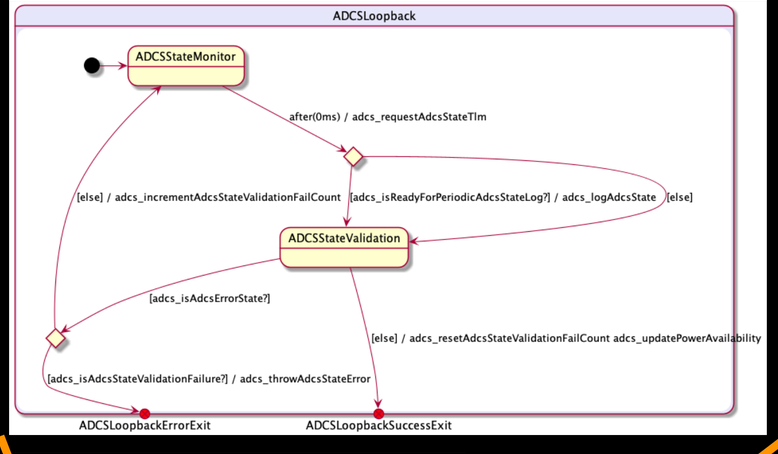
https://github.com/theodore-norvell/cogent

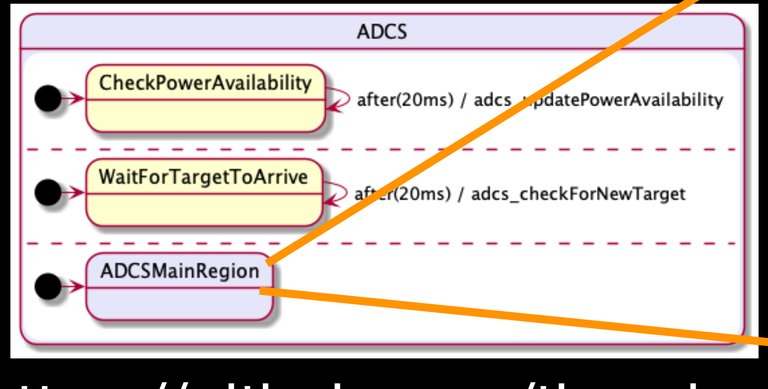
Scalability

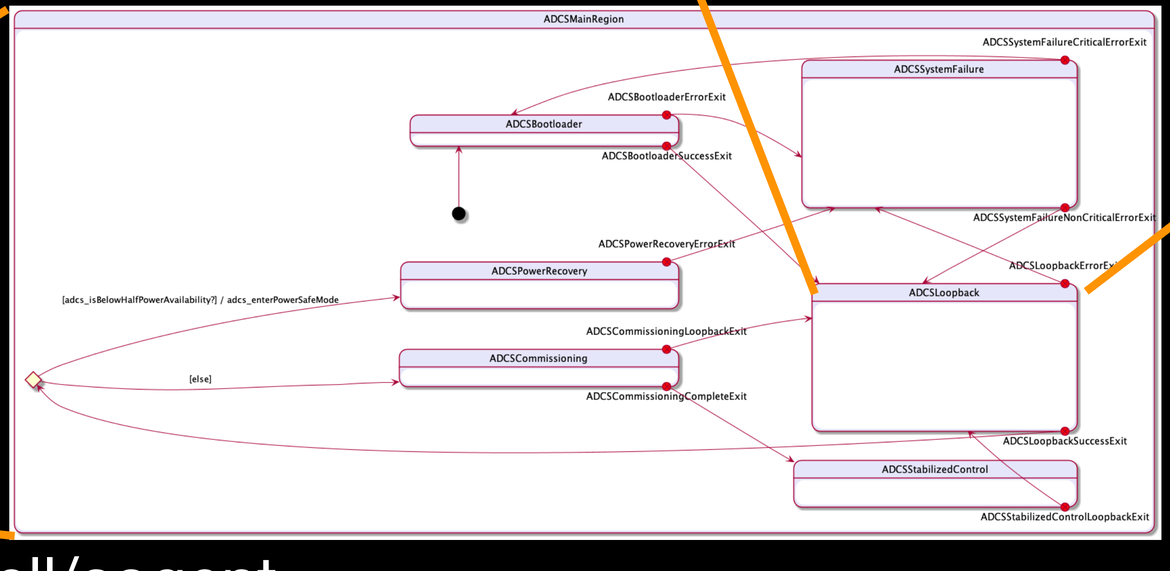
How can we deal with complex machines?

"Submachines"

- Allow state machine diagram and PUML code to be broken into smaller pieces
- Cogent stitches the submachines together







https://github.com/theodore-norvell/cogent

The past, present, and future

https://github.com/theodore-norvell/cogent

The past

- Many tools and libraries have supported translation of Statecharts and other kinds of state machine diagrams to code
- None have really caught on
- Most are not suitable for embedded code

https://github.com/theodore-norvell/cogent

The present

- Cogent is being used in the Killick-1 successfully
- Students seem to have learned to use statecharts effectively
- It may be used in the next satellite project
- It is open source and cost free

https://github.com/theodore-norvell/cogent

Assessment

- Pro
  - Statechart diagrams are fairly easy to create, understand, and review
  - Textual form has little baggage and if "diff friendly"
  - Cogent produces usable code
  - Concurrency with little nondeterminism and surprises
  - Provides a clear abstraction boundary
- Con
  - Treatment of data is awkward. E.g. communication between actions.

https://github.com/theodore-norvell/cogent

The future

- More coverage of UML statecharts, e.g. entry and exit actions
- Using text input has good points
  - But it is still a bit awkward
  - PUML has some limitations
- I'd still like to see a graphical editor
  - that takes a hybrid approach
  - with a graphical diff/merge tool

https://github.com/theodore-norvell/cogent