EVENT DRIVEN PROGRAMMING

- Embedded Real Time Systems
- Ron Barker

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Review of Last Week

STATE MACHINE IMPLEMENTATIONS

Generic State Tables

- Commonly Accepted "state of art" FSM Implementation
- State Tables are basically Truth Tables in which each vector is the address of a state handler.
- State Tables are represented asTwo Dimensional Arrays
  - Events in the horizontal axis
  - States in the Vertical Axis
- Numeric Values represent States / Event
- State Transition is implemented in a dispatch function:
  - Calculates the offset of state handler in table
  - Calls via function pointer that state handles

Generic State Tables

	Events			
	UP	DOWN	ARM	TICK
States				
Setting	Setting_UP, setting	Setting_DOWN setting	Setting_ARM(), timing	Empty(), setting
Timing	Timing_UP, timing	Timing_DOWN(), timing	Timing_ARM(), setting	timing_TICK() timing

Practical Exercise

- Bomb 2 – Check Out in Redmine

- Compile – Run
- Analyse

PRACTICAL EXERCISE
BLINKY : STATE TABLE MODEL

Blinky Reloaded

- Rewrite Blinky as a State Machine
  - Use State Table Method
  - Use Bomb2 as guide

Generic State Tables-The Good

- Maps state table representation directly to handlers
- Provides relatively good performance for event dispatching (single instance of dispatching)
- Event Process promotes reuse of code
- Tables can be stored in ROM to accommodate resource constrained devices

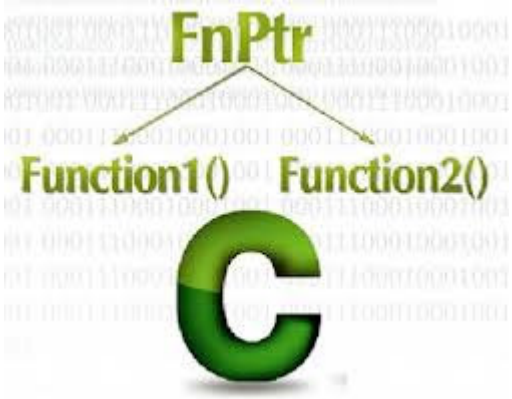
Generic State Tables-The Bad…

- Require States and Signals as ENUMS
- States and Signals – indices into state table .. must be contiguous and start with 0
- Initialisation of State Table is difficult to maintain:
- Adding a new state requires add and initialisation of a new row

STATE TABLES, FSM
FUNCTION POINTERS

Function Pointers

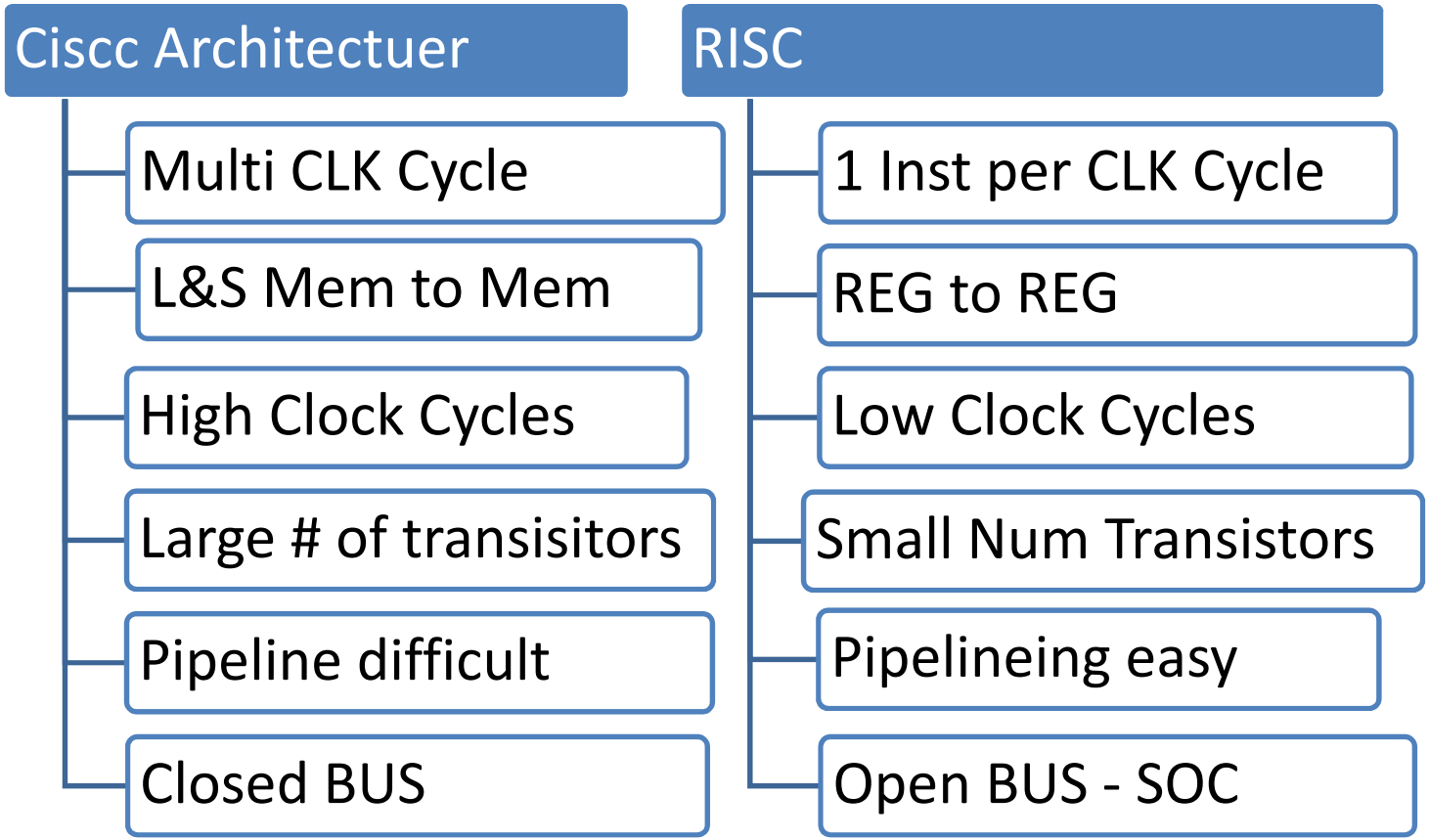
- Function pointer ->executable code within memory
- Enable late binding!!!
- FSM rely heavily on *2()
- State Machines are the “killer apps” for *2()
- Have a direct impact on CPU Architecture



CPU Architecture Review

- Von Neuman
- Complex Instruction Set
  - CISC architecture goal
    - complete a task in as few lines of assembly
    - compiler is simple
  - HW instructions were efficient compared to SW
  - ASM instructions represent multi operations
- Harvard
- Reduced Instruction Set
  - RISC Architecture
    - Simple instructions that Execute within one clock cycle
  - Compiler is complex

Comparison of CISC - RISC



CPU Impact for *2()

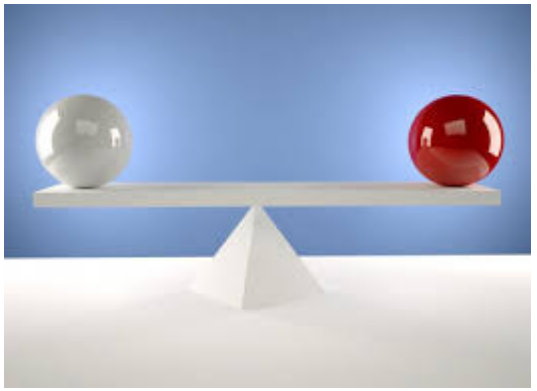
- RISC Seperates Load and Store
  - Operand remains in register until new value
  - Reduces the amount of work that the computer must perform.
- CISC-style "MULT" command is executed,
- The CPU automatically erases the registers
- Requires Register reload from memory

MODELING + FRAMEWORKS = STABLE STATE MACHINES

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

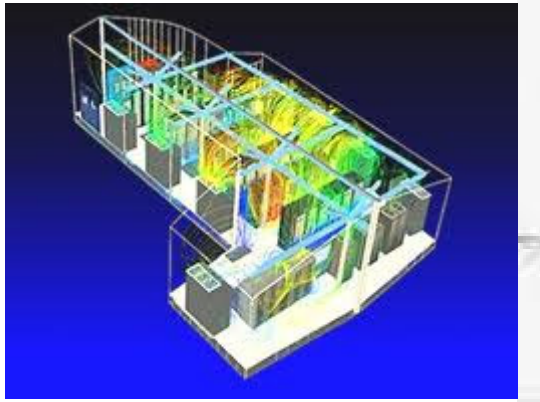
Steps to Stable State Machines

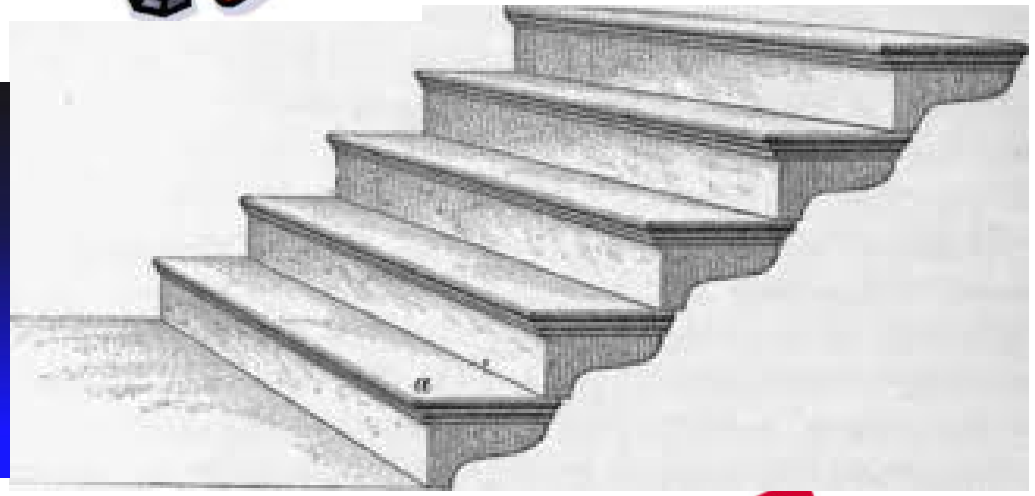
Deploy



Build with Frameworks

Design with UML

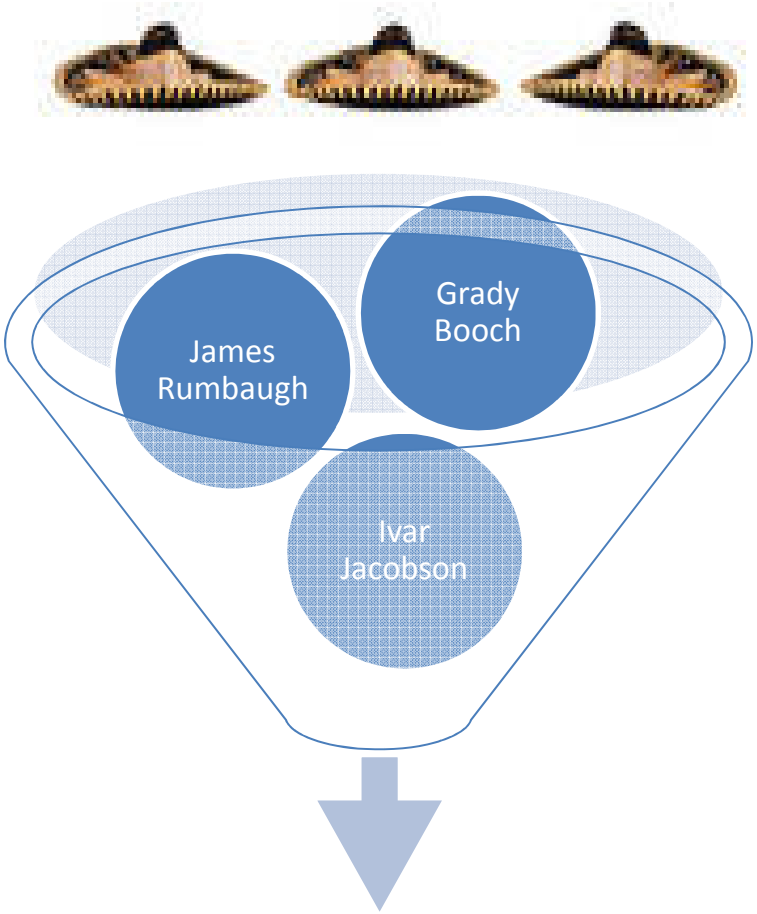




HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

UML MODELING BASICS

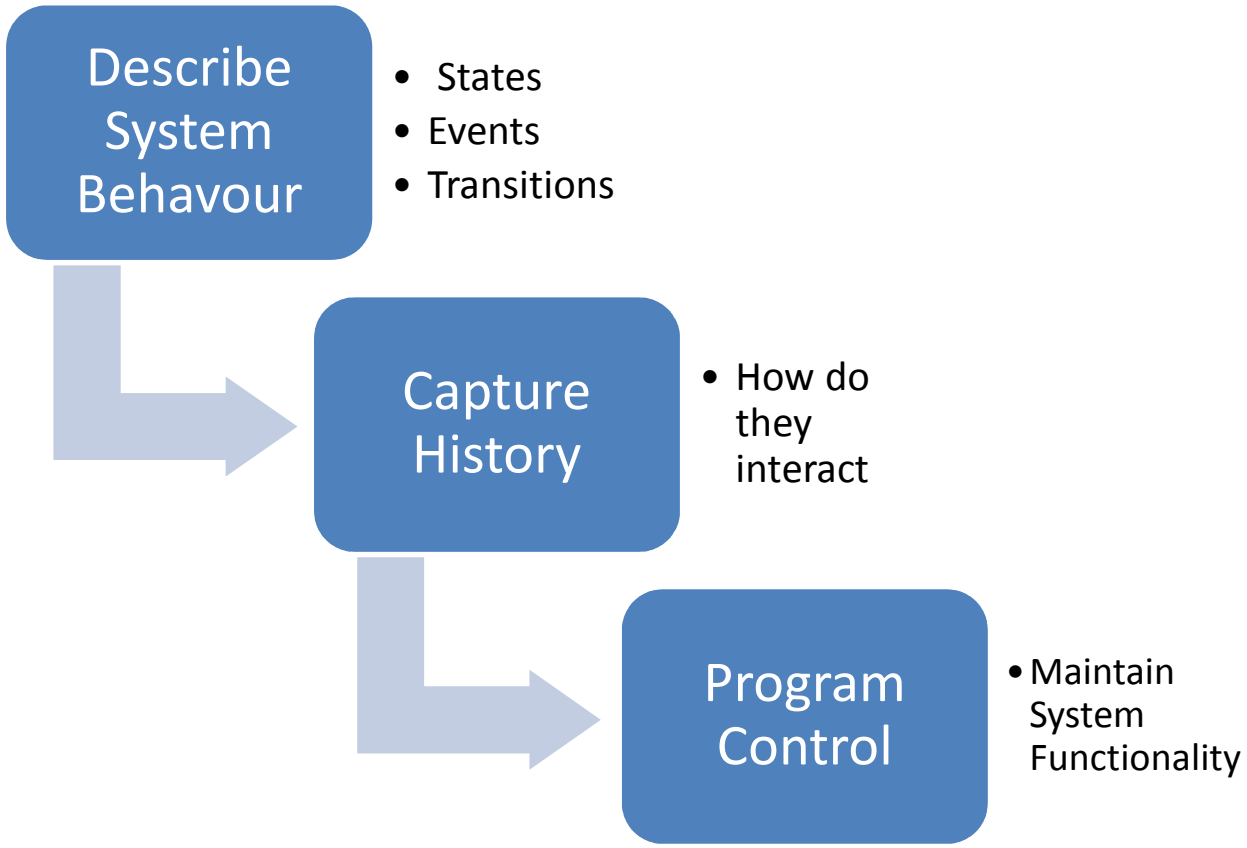
UML y Los tres amigos



UML State Charts

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Achieving Control --The Goal of State Modeling



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

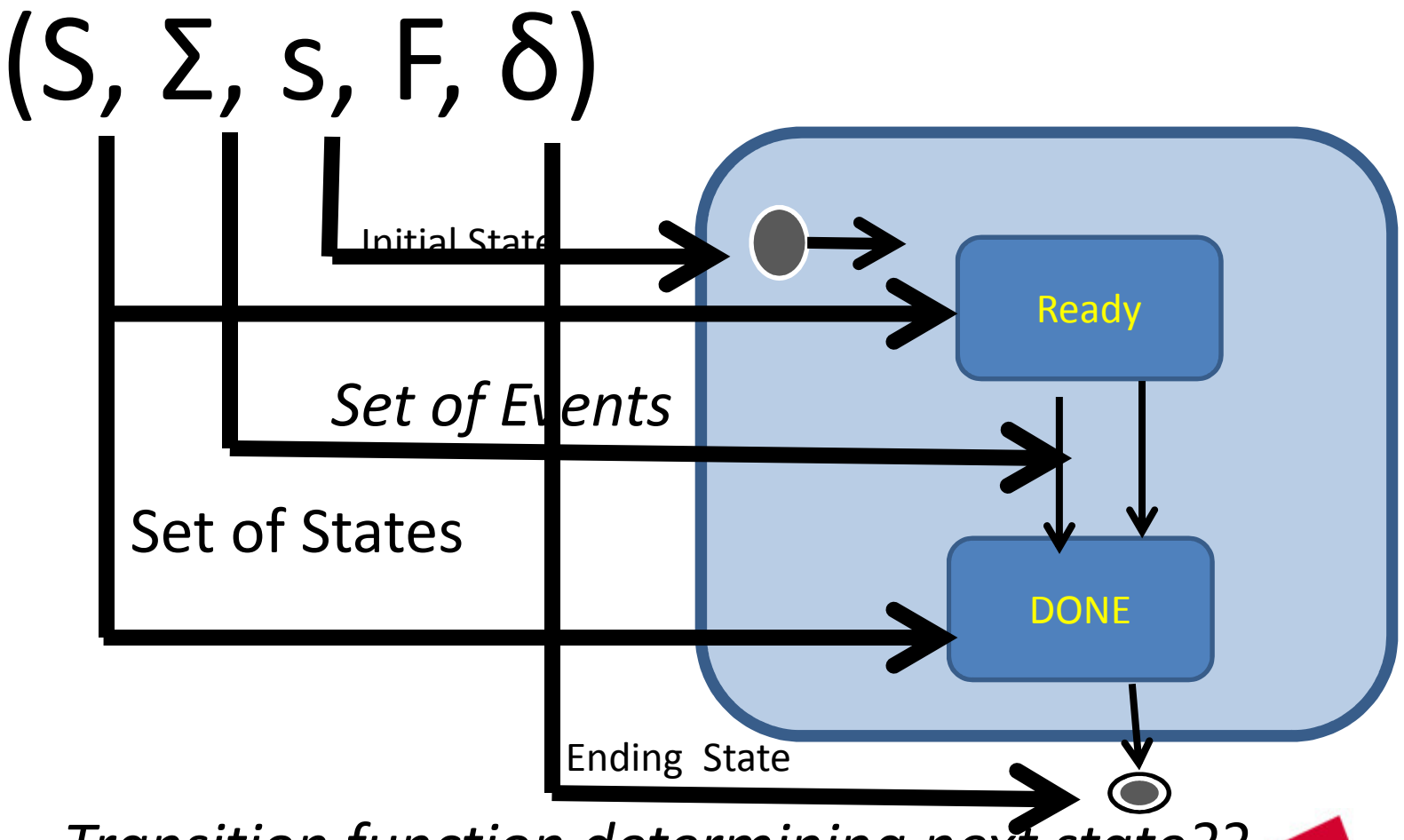
Modeling with UML

- UML State Charts are derived from Harel State charts
- UML Start Charts are only Blueprints

- They are NOT FLOWCHARTS!!!!!
- They Provide no implementation

JOB OF UML State Charts

realization of the mathematical concept of a finite automaton



Transition function determining next state??

Basic Formal UML Constructs

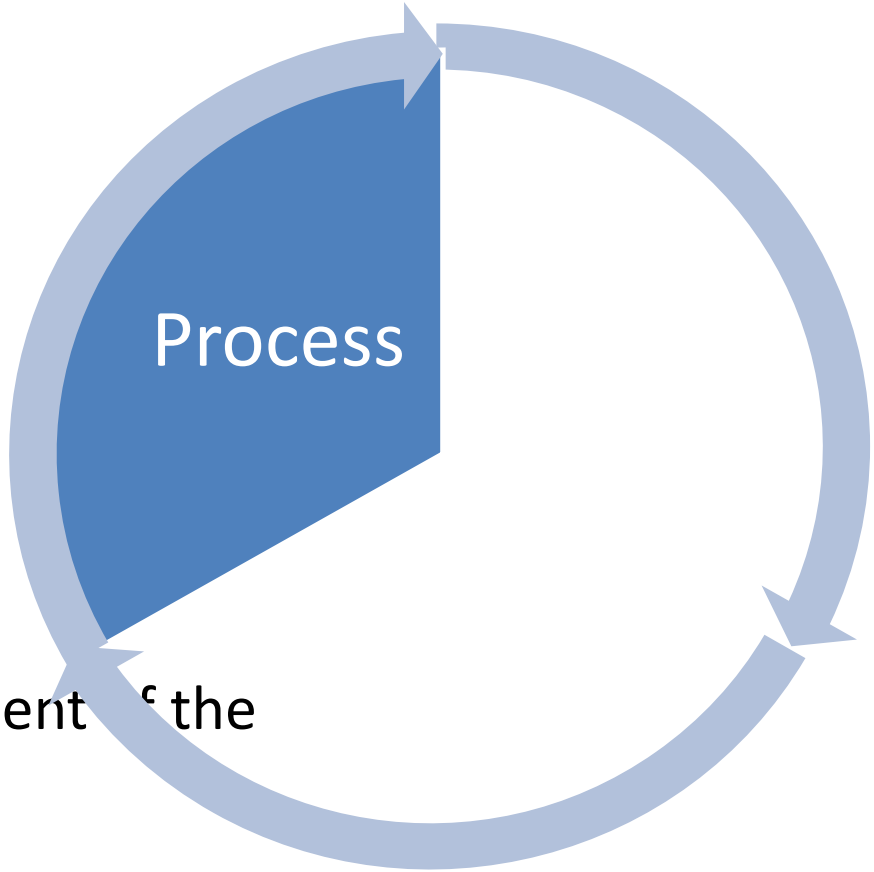
- Events
- States and Transitions
- Guards
- Actions
  - Entry
  - Exit
  - Other as required by application

Events

- UML Events
  - UML Refers to Type of Event Occurrence
    - An abstraction such as
      - Keyboard Event
      - Time Event
      - AD Event
  - UML Event do not refer to specific INSTANCES of an event
    - Key A Pressed
- UML Events may be
  - Internal
  - External

States

- UML State Charts
  - Capature System History
  - Decompose System Behaviour
  - Recognises Events
  - Handles Events
  - Executes Transitions



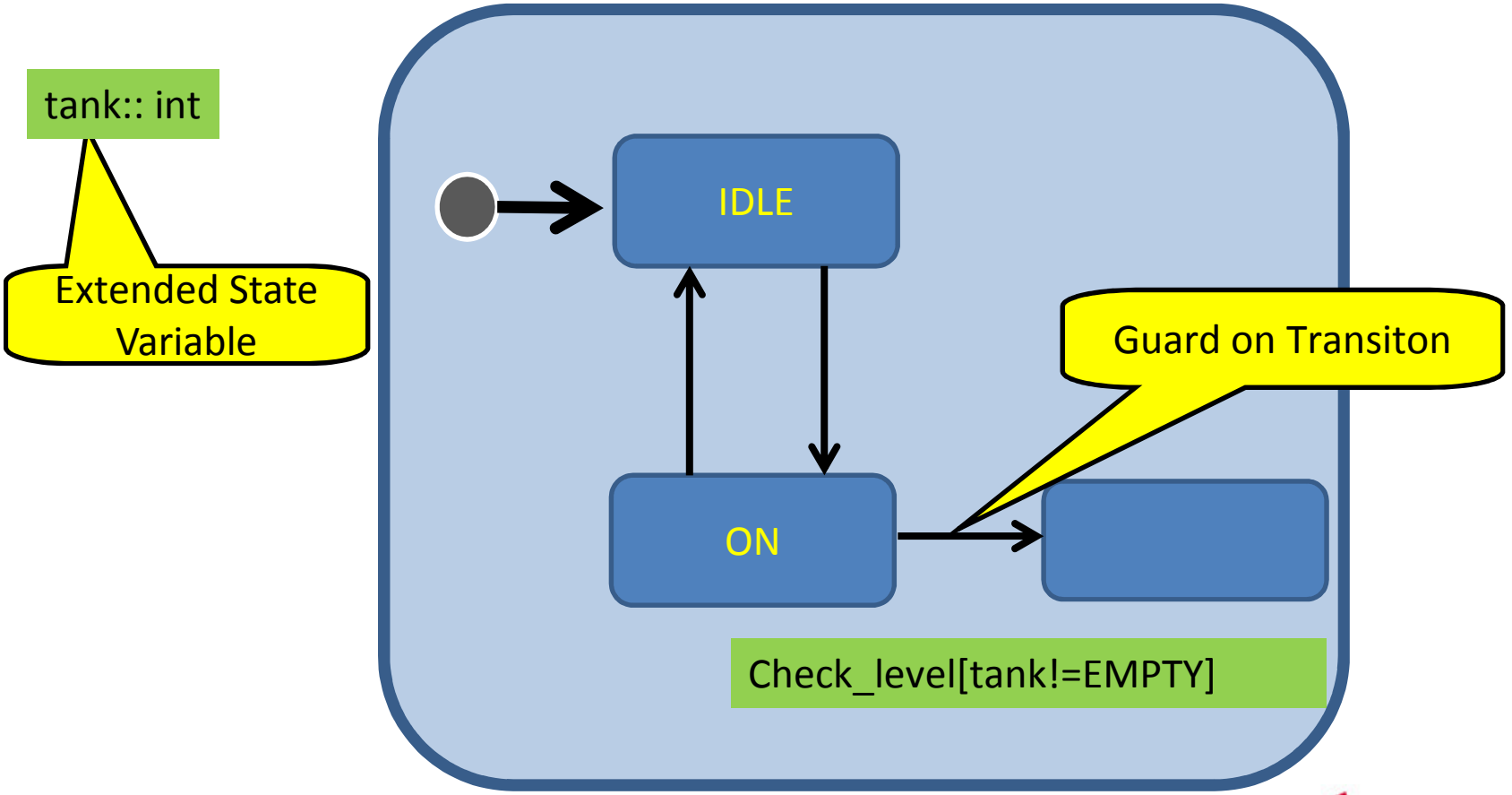
- Represents the „Process“ segment of the
  - Capture
  - Dispatch
  - Process

Guards

Help keep the solution manageable over the life cycle of the product

- Guard Conditions
  - Couples quantitative and qualitative aspects
  - E.g., Coffee Machine: The guard condition [tank_level >0]/BREW
- Introduces complexity and can lead to architectual decay (if/else)
- *Caveat Emptor!* – Creeping Guards lead to convoluted code
- Better to expand system through new states

Conditional Transition Execution-Guards



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Actions

- Entry / Exit Action
  - Optional actions
  - Associated with STATE and not the Transition
  - Analagous with C++ Constuctors /Destructores
    - » E.g. Coffee Machine: Pot removed (EVENT) in BREWING (STATE) results in exit action: Switch-off heat element
- Major Design Question:
  - Where best placed – level of heirarchy
    - » Must Each STATE have all of these...
    - » Or..other suggestions – hint: OO principle of??

Event-driven programming without an underlying FSM model can lead programmers to produce error prone, difficult to extend and excessively complex application code

FRAMEWORK BASICS

Building the State Machine

- DO IT YOURSELF
  - Blinky et alia

- USE A FRAMEWORK

Frame Works for Real Time Event Driven Systems

- What is a “framework
- Some Hype:

  has been developing the Real-Time Framework (RTF), a novel middleware technology for a high-level development of scalable real-time online services through a variety of parallelization and distribution techniques. RTF is implemented as a cross-platform C++ library and embedded into the service-oriented architecture

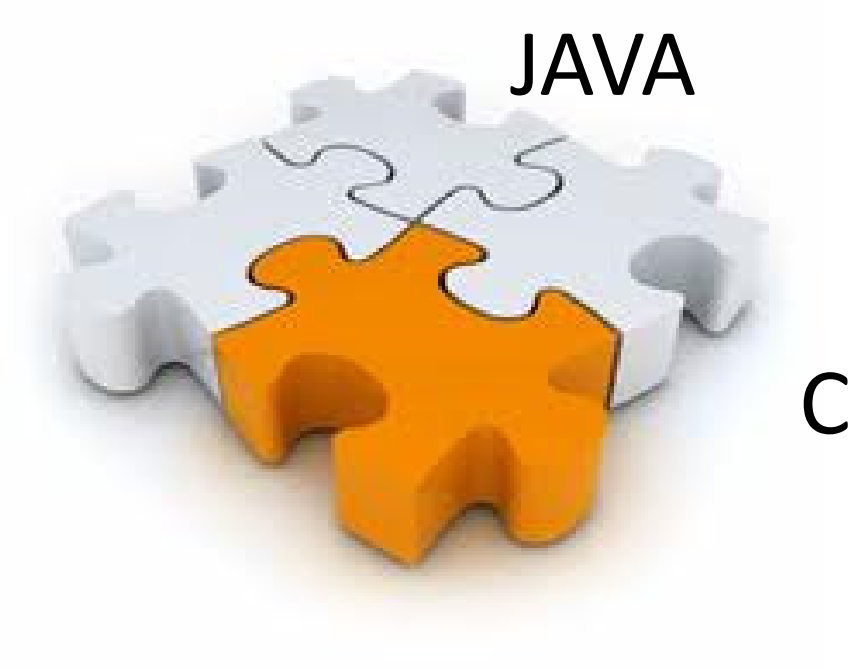
- A Framework is just a Tool(kit)
- NOT AN API

The Purpose of Frameworks

- Provide Re-usable Infrastructu for:
  - Controlling Event Capture
  - Abstracting Event Dispatch
  - Simplifying Event Process
- Provides Transparency for
  - Inversion of Control
  - Eliminate the need for a fore ground loop

UML – Frameworks –OO
The Perfect Fit?

UML



Framework

C++/Java seems “natural” in UML Frameworks

- Partitions state behaviour and localises it in specific classes
- Efficient state-transition – simply reassign pointer – essence of C++
- Good performance:
  - Late binding
  - Eliminates indexing then invoking a pointer
- Allows fine granularity & customisation of event handler
- Memory efficient
- Does not require ENUMs of State/Signals

Where are the Problems?

- Based on Polymorphism – OO Language is MUST
- Compromises encapsulation – “friends” might be required and are common in practice!
- Adding states results in sub classing abstract state class
- Adding new events required new event handlers in the abstract state class
- Not hierarchical

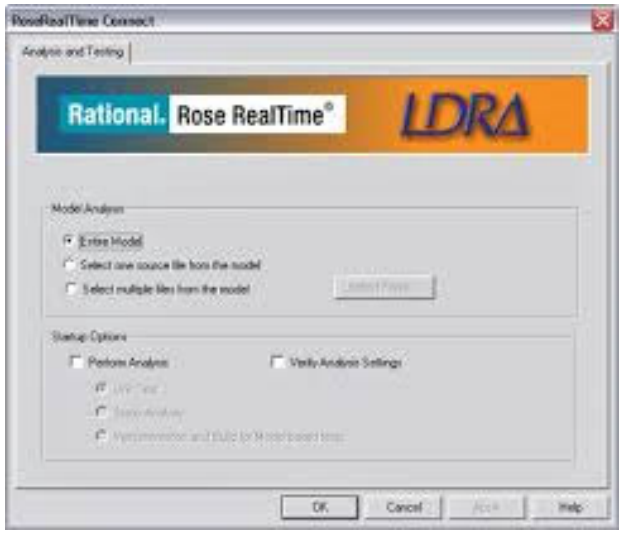
The Ugly

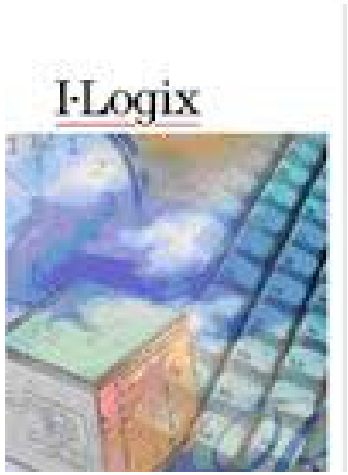
- C++ Exception Handling
  - Is fundamentally at odds with RTC semantics
  - Usually corrupts the extended state variables (why?)
  - Each RTC step must be considered atomic
- What effect does Exception Throw have on Stack?
  - Why is this important for Event Driven Systems
  - Poor Inversion of Control and Execution Context
    - History is in state variable
    - not preserved across the stack

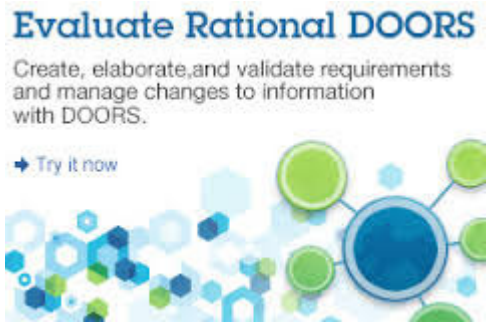
MODELING + FRAMEWORKS = STABLE STATE MACHINES

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Classical OO Frameworks







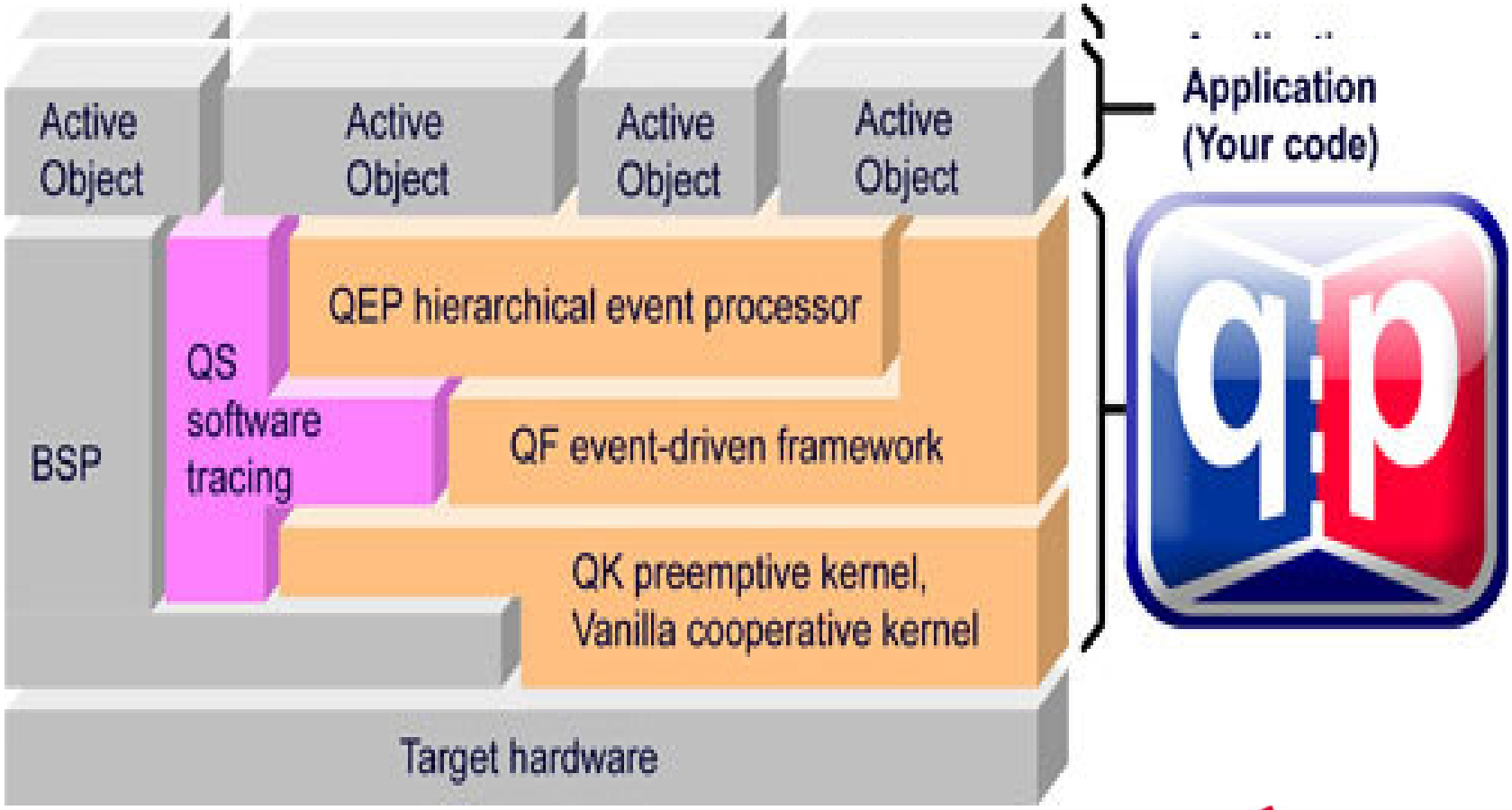
OO Frameworks Require System Resources

- Abstraction levels cost – time – money
- Not appropriate for resource constrained devices
- Sorry - No free lunch

The Challenge

- Create a “resource aware” / “skinny” framework:
  - Implements a “generic” Event Processor :
    - The “F” element (S, Σ, s, F, δ)
    - Use Nested Switched efficiently (single leve
    - Eliminate State Tables and overhead
  - Does not Require an OO Languag
  - Supports HSM
  - Enforces Inversion of Control
  - Conforms to UML Precepts

Quantum Processor Framework



Quantum Modeler

- Aspects of UML State Chart Modelling with QM
  - Effective mapping of (S, Σ, s, F, δ)
  - Provides a „UML conform“ framework-code generater
  - Follows a minimalistic approach

MODELING WITH QM

Practical Exercise – Bomb4

- Base Model file in bomb4 directory
- Add Packages
  - Bomb4 Stereotype Components
  - Events Stereotype Events
- Component Bomb4 – add class
  - Superclass FSM
  - Add extended State variables
- Component Events – add class
  - Super Class Evt
- Add Operation – Bomb4_ctor

Model SM Bomb4

- Insert Initial Transition
- Insert States
  - Timing
  - Setting
- Insert Transitions
  - TICK
  - UP / DOWN
  - ARM
- Insert UML – Option “Entry”

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Generate Framework Bomb4

- QM Meta Keyword $define
  - $define(xxxx:.yyyy)
    - Generates code for the entity yyyy in component xxxx
  - $define(Bomb4::Bomb4)
    - generates FSM bomb4
  - $define(Bomb4::Bomb4_ctor)
    - Generates Constructor
- QM Meta Keyword $declare
  - $declare(Bomb4::Bomb4)
    - Generates forward declarations

Add Code to Model

- Use Code in Bomb2
  - E.g. Transition UP in State Setting
    - if (me->timeout < 60) {++me->timeout;BSP_display(me->timeout);}
- Add Main + BSP
- Replace Bomb2_dispatch
  - Use QFsm_dispatch_(…)
- Generate Framework
- Run Application