Abstract State Machines Method

Bridging the Gap bw Specification and Design

Egon Börger

Dipartimento di Informatica, Universita di Pisa
http://www.di.unipi.it/~boerger

Overall Research & Technology Transfer Goal

Couple specification & design by

rigorous high-level (hw/sw co-) modeling which is linked seamlessly, in a way the practitioner can verify and validate, to executable code

– Develop succinct ground models with precise, unambiguous, yet understandable meaning

  to support implementation independent, application oriented system analysis: verification/validation

- Refine models into a hierarchy of intermediate models, modularizing orthogonal design decisions (“for change”) and justifying them as correct,
  - linking the ground model to the implementation
  - documenting the entire design for reuse and maintenance

Key Strategy for Hierarchy of Models: Divide et Impera

- Separation of Different Concerns
  - Separating orthogonal design decisions
    - to keep design space open (specify for change: avoiding premature design decisions & documenting design decisions to enhance maintenance)
    - to structure design space (rigorous interfaces for system (de)composition, laying the ground for the system architecture)
  - Separating design from analysis
    - Separating validation (by simulation) from verification
    - Separating verification levels (degrees of detail)
      - reasoning for human inspection (design justification)
      - rule based reasoning systems
        - » interactive systems
        - » automatic tools: model checkers, automatic theorem provers
- Linking system levels by abstraction and refinement

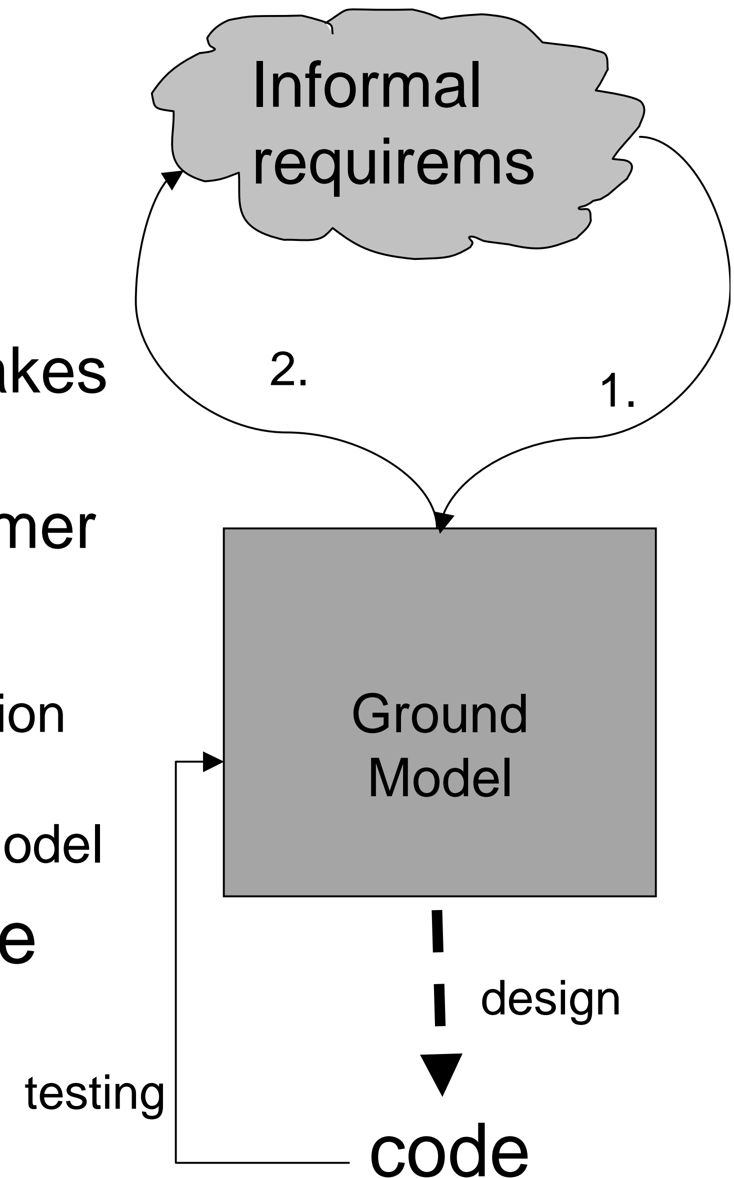
What is provided by ground models for requirements

1. **Requirements capture**
   documenting relevant application domain knowledge for designer

2. **Requirements inspection** makes correctness & completeness checkable for system user & customer
   - Verification of properties
   - Validation: mental/machine simulation (of user scenarios or components) supported by operational nature of model

3. Makes requirements traceable by relating them to design

4. Provides **test plan** basis



Properties of Ground Models

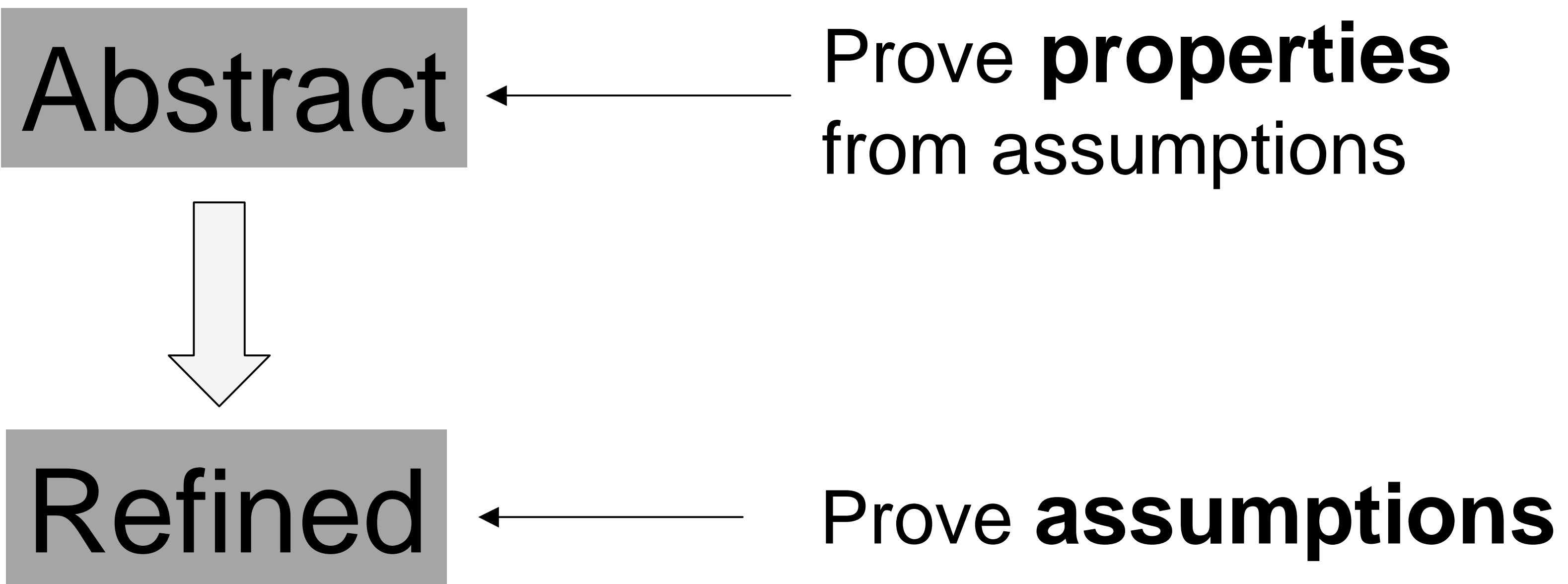
- precise at the right level of detailing, yet:
  - flexible:
    - adaptable to different application domains
    - easily modifiable/extendable to serve as prototype & for reuse
  - simple & concise:
    - to be understandable by domain expert & system user
    - for mathematical analyzability of consistency, completeness,minimality
  - resembling the structure of the real-world problem (oo credo!)
    - to be falsifiable (by experiment) and thus validatable (NB. No infinite purely mathematical justification chain is possible)
- abstract, yet:
  - complete: containing all semantically relevant params as interface
  - operational: to support process oriented understanding & simulation
- rigorous foundation for reliable tool development & prototyping

Calibrating degree of formality
wrt application (system user or customer)

- Language must be appropriate (natural) for application domain
  - easy to understand/use for practitioner, supporting
    - concentration on problem instead of notation
    - manipulations for execution & analysis of terms, consistency, completeness, etc.
  - tunable to (and apt to integrate) any
    - data oriented application (e.g. using entity relationship model)
    - function oriented application (e.g. using flow diagrams)
    - control oriented application (automata: sequential, multiple agents, time sensitive, …)
- Spec must resemble structure of the real-world problem & provide
  - data model (conceptual, application oriented)
  - function model (defining dynamics by rule executing agents)
  - interface to
    - user for communication with data/fct model by dialogue or batch operation
    - environment (neighboring systems/applications)

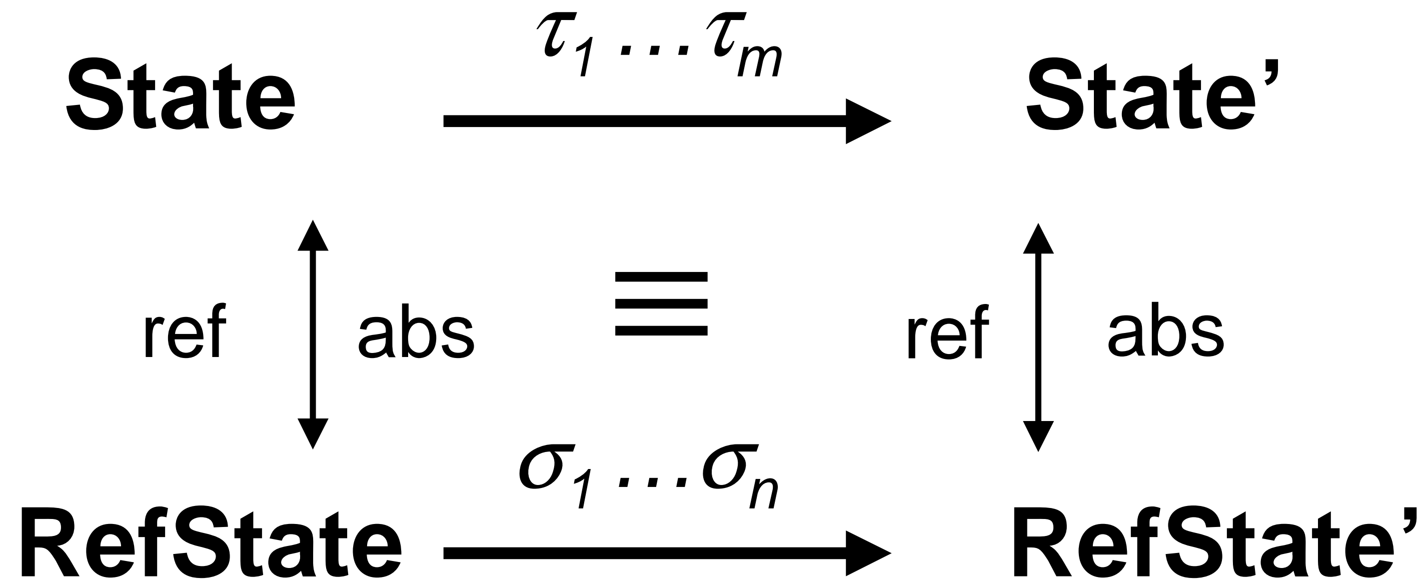
Verify implementation to meet design properties

Method: divide & conquer (ancient math paradigm)



Use correctness of refinement

The Scheme for a Correct Refinement/Abstraction Step



defined

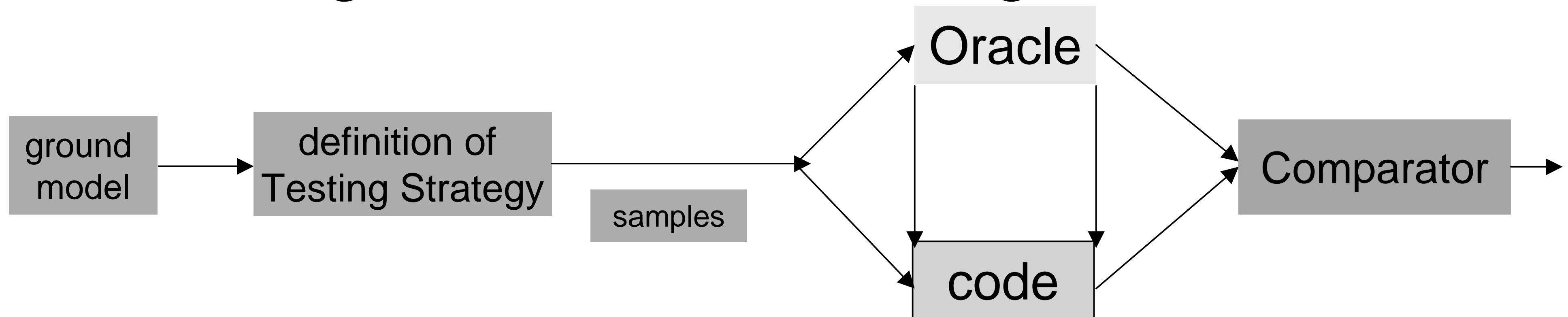
- relating the locations of interest
- in states of interest
- reached by comp segments of interest

The refinement task

- Find/formulate the right *refinement /abstraction* that
  - faithfully reflects the intended design decision (or reengineering idea)
  - can be justified to implement the given model correctly (or abstract from the given code), namely through
    - **Verification**
    - **Validation** testing model-based runtime assertions to show that design assumptions hold in the implementation
- Effect: enhancement of
  - communication of designs and system documentation (report of analysis)
  - effective reuse (exploiting orthogonalities, hierarchical levels)
  - system maintenance based upon accurate, precise, richly indexed & easily searchable documentation

See E.B.: High Level System Design and Analysis using ASMs LNCS 1012 (1999), 1-43 (Survey)

Using ASMs for test case generation



Creative, application domain driven selection/definition of test cases: guided by ground model support user to specify relevant env parts & props to be checked, to discover req gaps

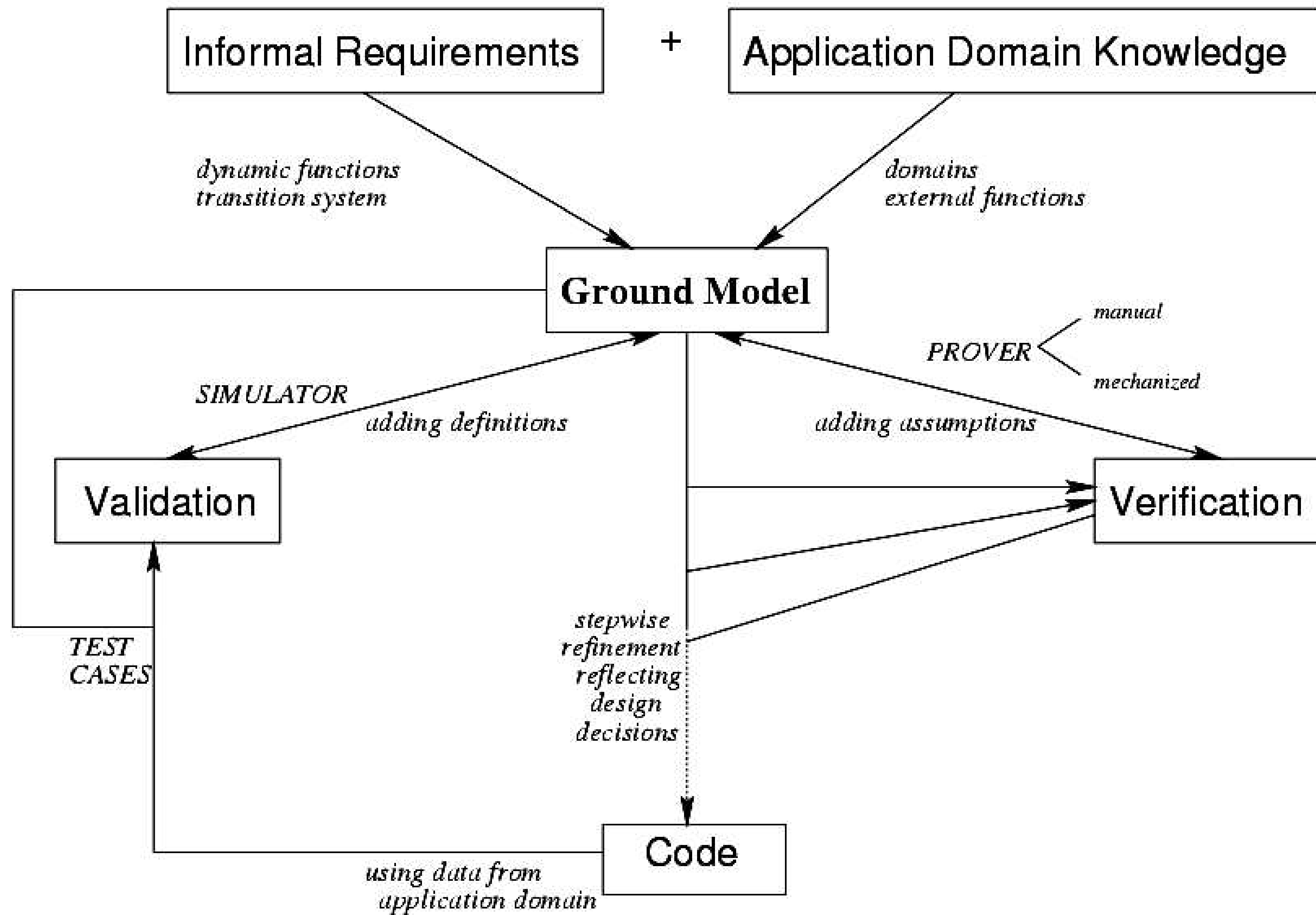
Definition of oracle to determine expected results for samples : by running the ground model

Definition of comparator using the refinement of oracle to code: determine states of interest to be related (spied) & locations of interest to be watched & when their comparison is considered successful (equivalence relation)

Using ASMs for maintenance

- Documentation: accurate, precise, richly indexed & easily searchable
  - reading off the relevant design features from the
    - ground model description (independent from the language chosen for the implementation)
    - refinement step descriptions
    - model-based runtime assertions appearing in the test reports
- Support
  - examine the model for fault analysis
  - use the model to recognize where to correct bugs which have been reported
- Versioning
  - reuse the model (exploiting orthogonalities, hierarchical levels)

From Specification via Design to Code: an Iterative Process



The ASM language: truly abstract "code"

- Def. A (basic) ASM is a finite set of "rules" of the form

  If Condition Then Updates

  with Updates a finite set of $f(t_1,\ldots,t_n):=t$ with arbitrary functions,

  Condition a Boolean valued expression (1st order formula).

  See separation of basic events (guarded assignments) from scheduling in event-B
- In the current state (structure) S:
  - determine all the fireable rules in S (s.t. Cond is true in S)
  - compute all exprs t_i,t occuring in updates $f(t_1,\ldots,t_n):=t$
  - execute simultaneously all these function updates
- The updating yields the next state S'
- NB. The parallelism of basic ASMs can be extended to synchronous or asynchronous multi-agent ASMs

Standard Notation choose/forall in Basic ASMs

- Supporting non-determinism by non-controlled selection functions:

  choose x satisfying Cond

  R

  where Cond is a Boolean valued expression and R a rule.

- Supporting synchronous parallelism by simultaneous execution of function updates:

  forall x satisfying Cond

  R

  to simultaneously execute R(x), for all x satisfying Cond in the given state S, to form the next state S'

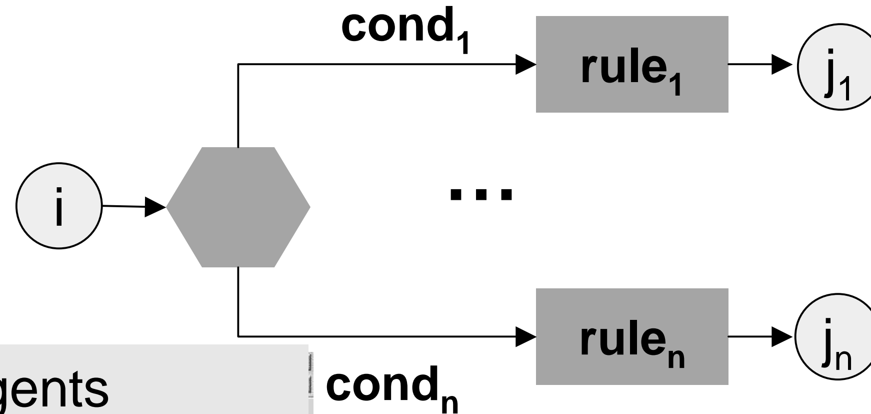
ExI: Creating new parallel processes (from Occam model)

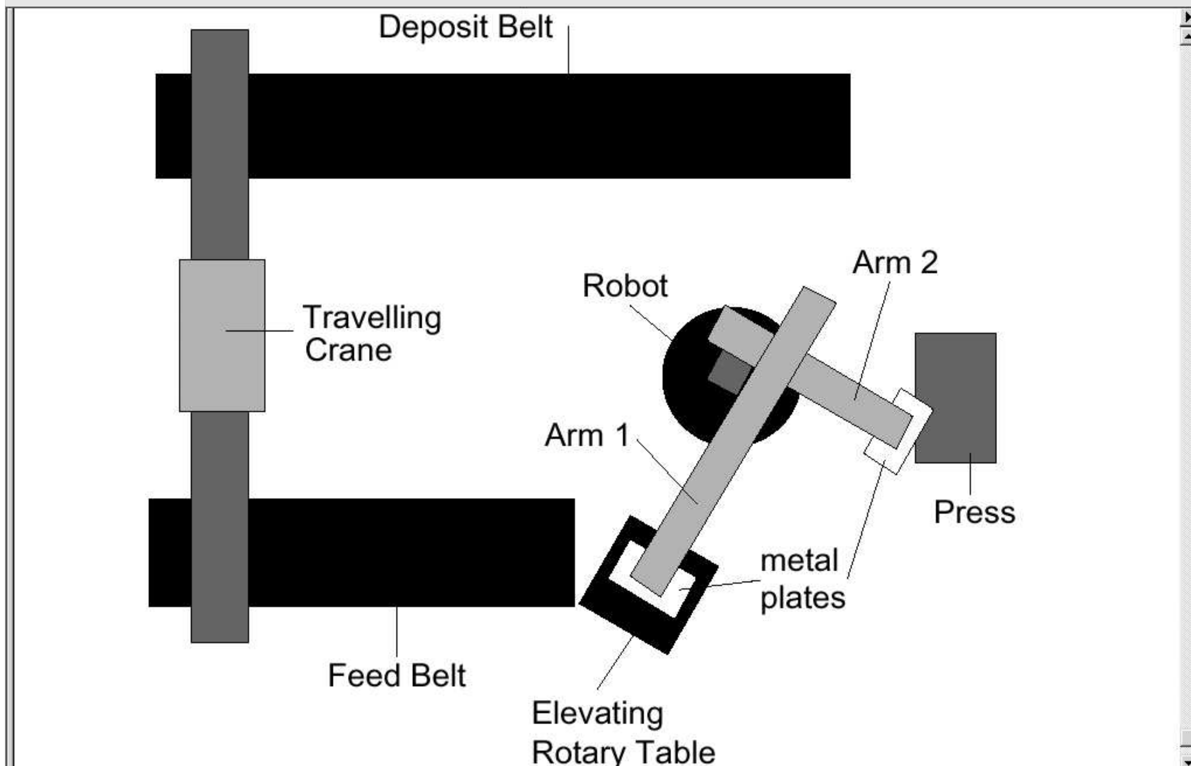
if instr(pos(a))=par $S_1 ... S_n$ — pos(a) = current “pc” value of agent a
and running(a)
then let $x_1, ..., x_n$ = new(AGENT) in — create n new agents
forall $1 \leq i \leq n$
running(x_i):= true — start each new agent to run
pos(x_i):= next(pos(a),i) — place each agent on its code
env(x_i):= env(a) — equip each agent with its env, inherited from a
parent(x_i):= a — record for each agent the parent process to whom to report upon termination
count(a):=n — parent process records how many subagents have to report to him before he will proceed
running(a):= false — parent process remains idle until all created subprocesses have terminated and reported to him

See AsmBook Ch.6.5

ExI: Co-Design FSMs Sangiovanni-Vincentelli

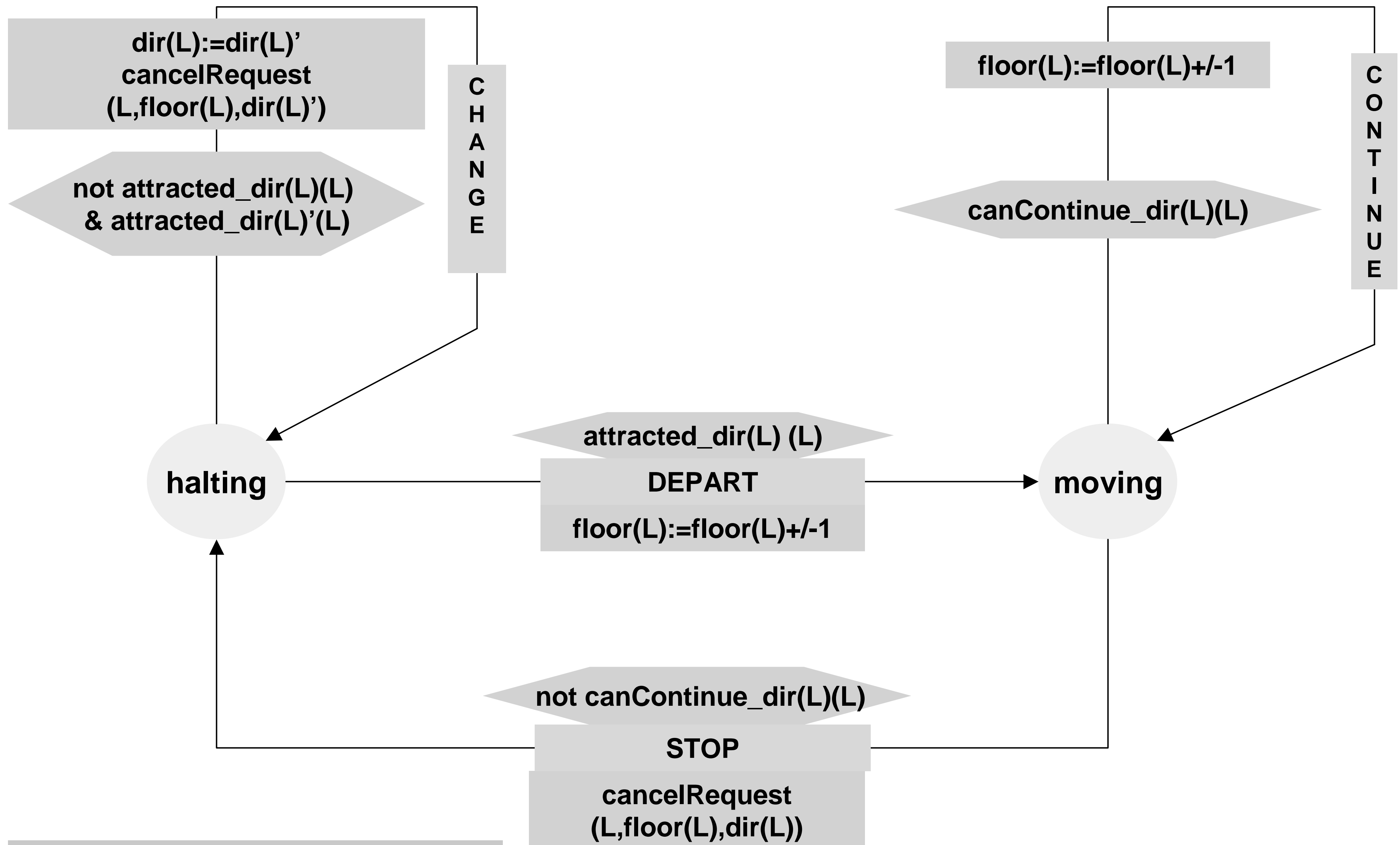
ASMs of rules of the form





Often with global agent scheduler and/or with timing conditions for agents performing durative instead of atomic actions

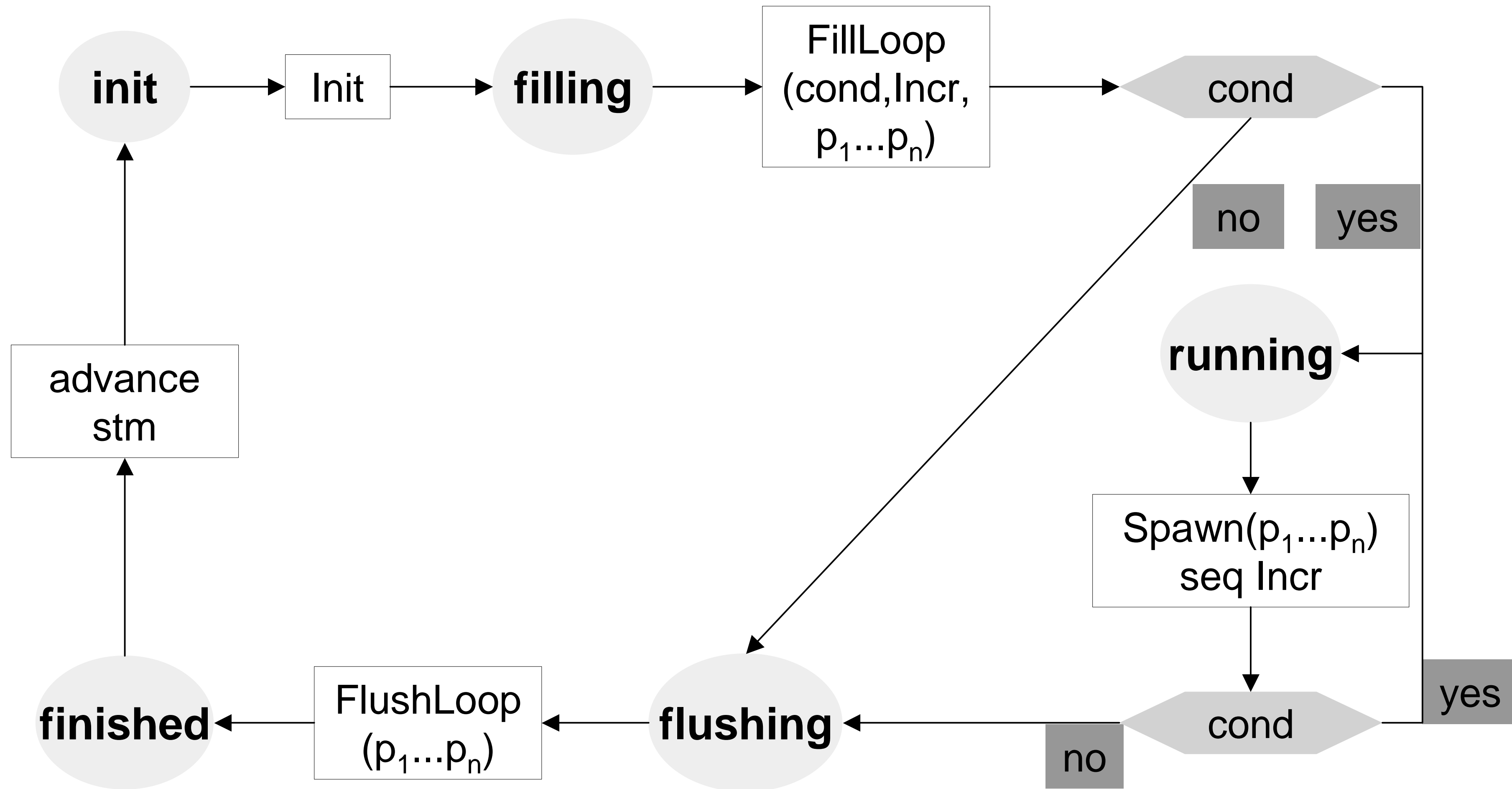
Verifiable Co-Design Lift: control state ASM agents L

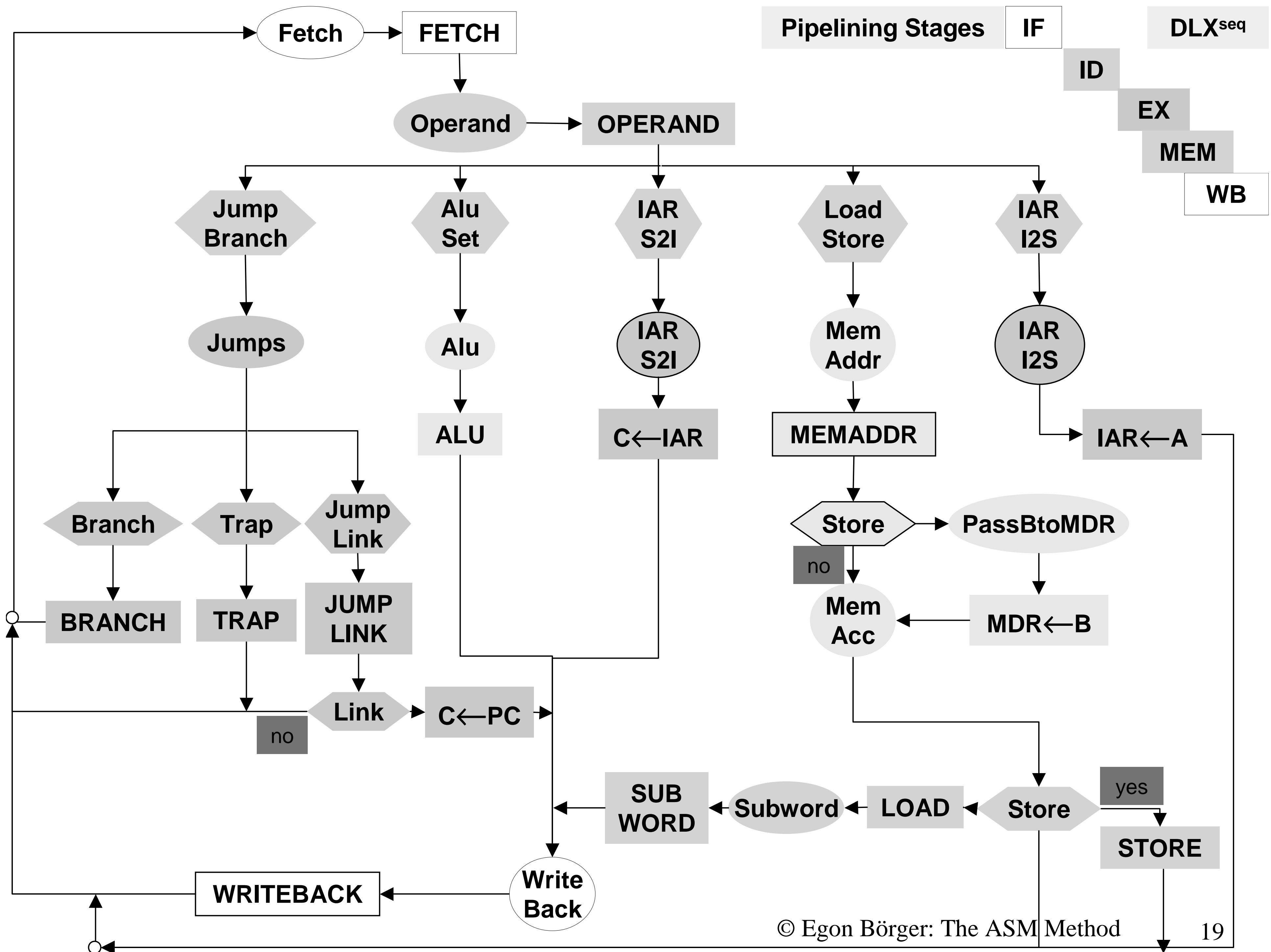


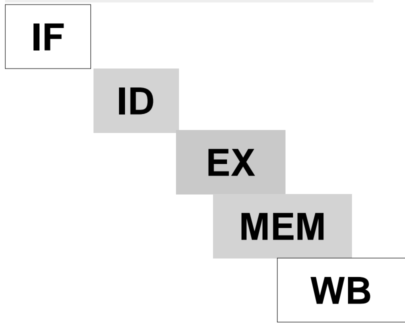
See AsmBook Ch.2.3

Control structure of execution semantics of SpecC pipe statements pipe(Init,cond,Incr,$p_1,\ldots,p_n$)

(Mueller, Doemer, Gerstlauer ISSS 2002)




Fetch
FETCH
Pipelining Stages
IF
ID
EX
MEM
WB
DLX^seq
Operand
OPERAND
Jump Branch
Alu Set
IAR S2I
Load Store
IAR I2S
Jumps
Alu
IAR S2I
Mem Addr
IAR I2S
ALU
C←IAR
MEMADDR
IAR←A
Branch
Trap
Jump Link
Store
PassBtoMDR
no
BRANCH
TRAP
JUMP LINK
Mem Acc
MDR←B
Link
C←PC
no
SUB WORD
Subword
LOAD
Store
yes
STORE
WRITEBACK
Write Back



DLX^{par} Rules

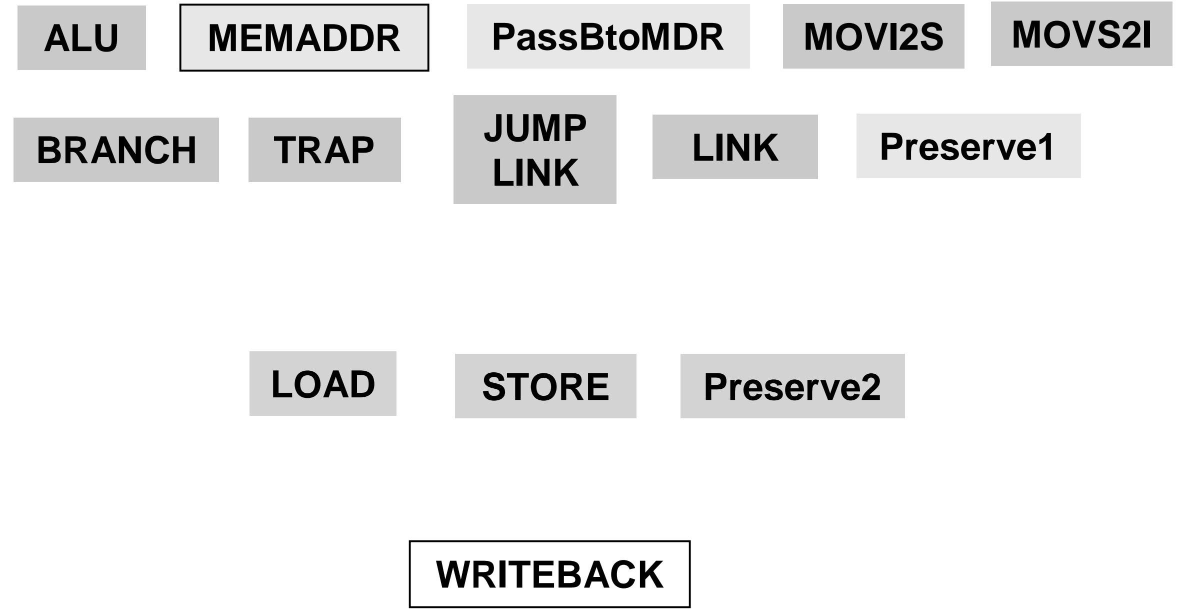
FETCH

Verified stepwise refinement see AsmBook Ch.3.3

- Ideally all rules fire simultaneously
  - one per instruction in its successive stages

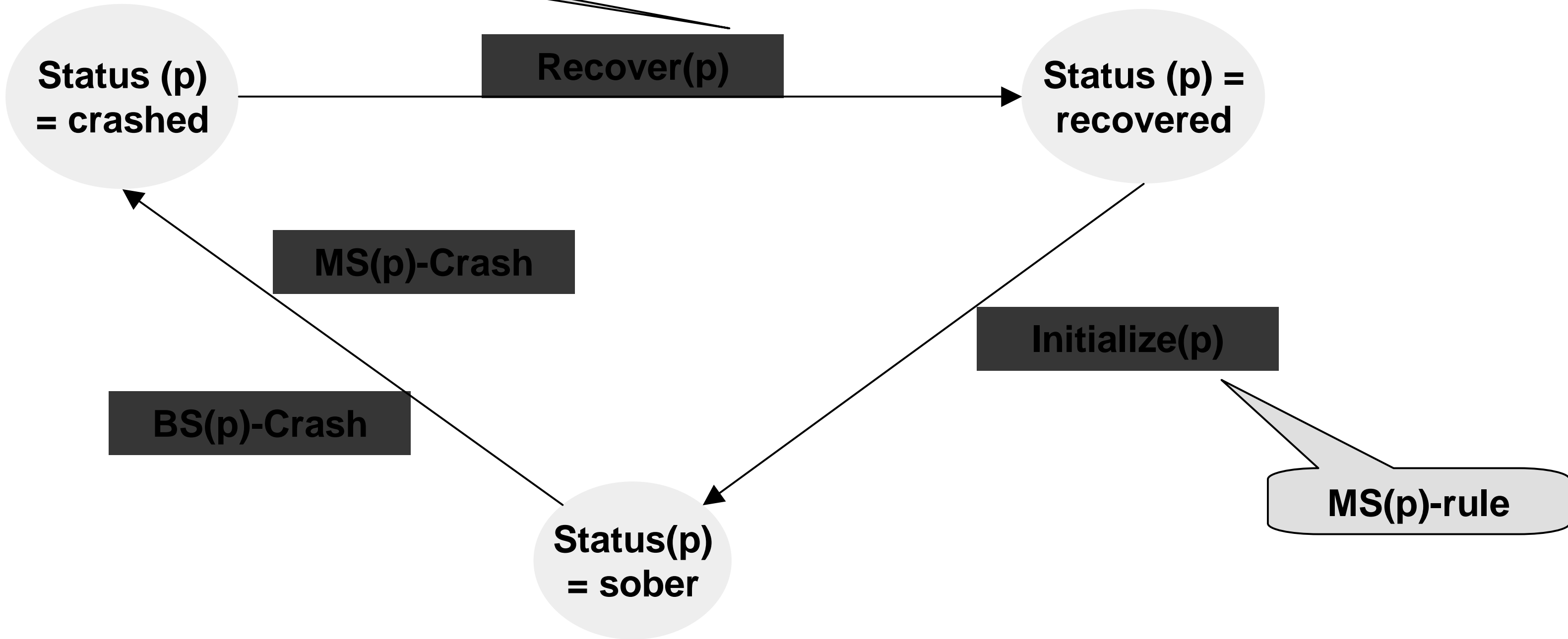
OPERAND Preserve

- The problem: how to guarantee that no conflicts arise when an instruction exec uses data which have to be computed by a preceding instruction whose pipelined execution is not yet terminated



Fault-tolerant Group Membership Protocol Verification

Each real-time processor p with multiple agents Clock(p), CurProc(p), Membership Server MS(p), Broadcast Server BS(p), Scheduler(p), …



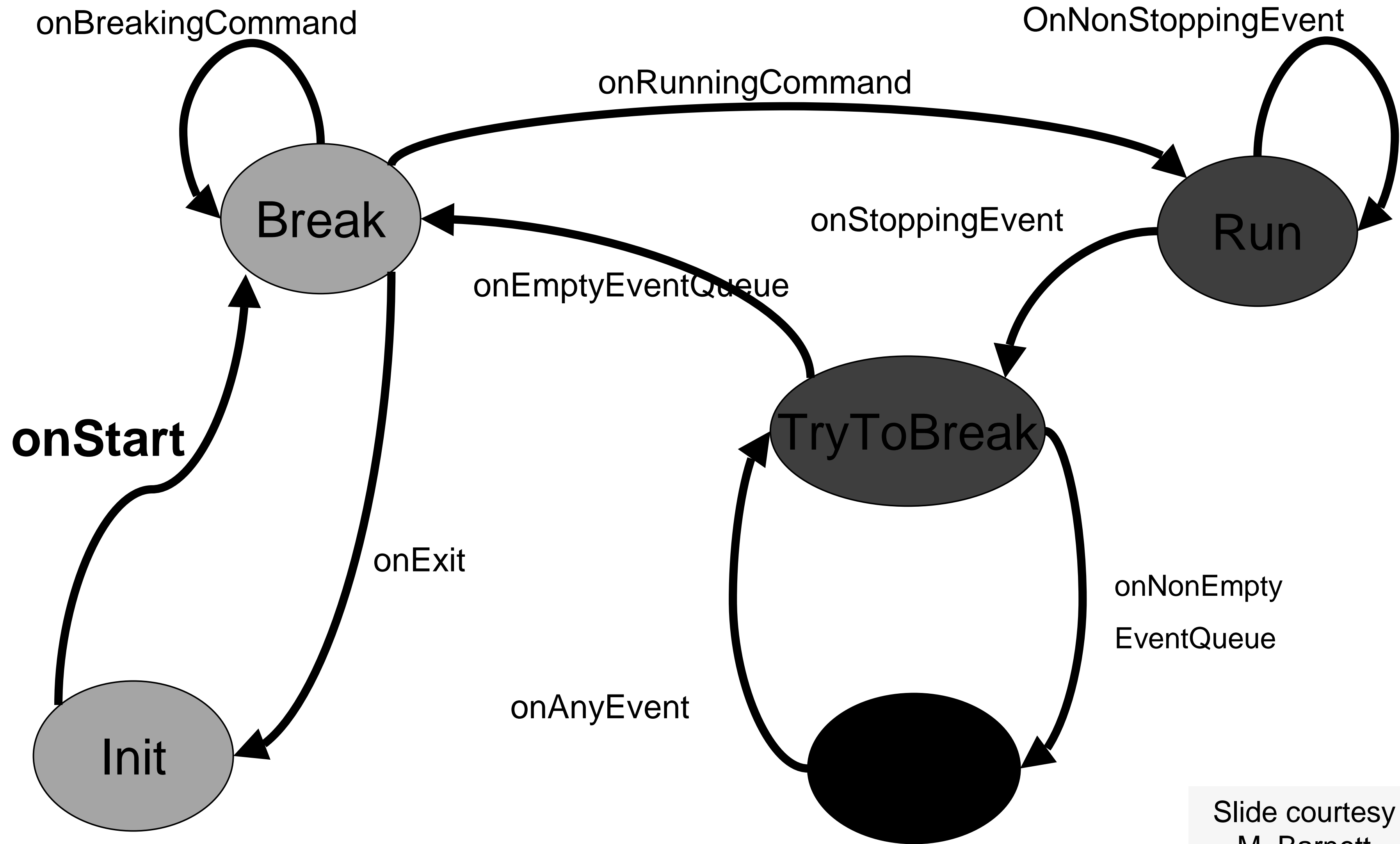
See Asm Book Ch. 6.3

Two reasons for crashing (processor interrupt): p missed the deadline of a
- **broadcast** : **BS(p) was scheduled too late**
-- **new-group msg** because **MS(p) was scheduled too late** to handle it

Clock(p) > BCastTime(p)

Clock(p) > Timestamp(CurMsg(p))

Illustrating sequential submachine refinements refining the control state ASM model for a debugger

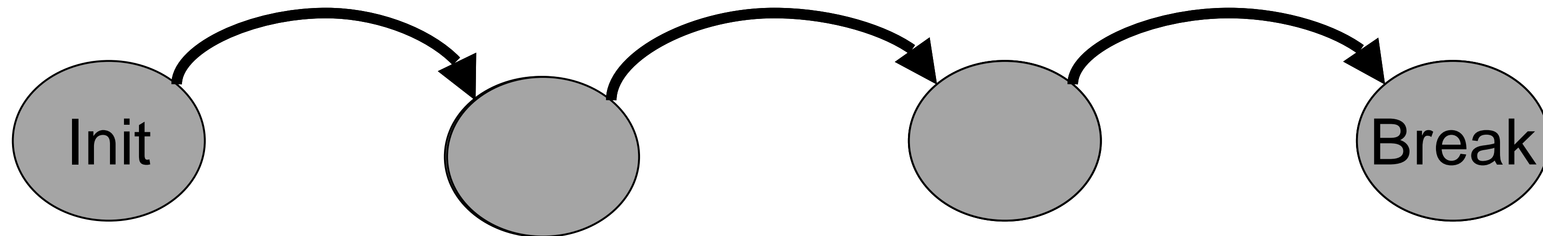


Slide courtesy
M. Barnett
M. Veanes

See AsmBook Ch.3.3

Sequential submachine refinement of machine onStart into a sequence of three submachines

`initializeCOM` `createNewShell` `setDbgCallback`



debugger

env

dbg services

Env
Process=Null
Thread =Null
Frame =Null
BPs ={}
...

callbacks

Slide courtesy
M. Barnett
M. Veanes

Parallel and sequential refinement of callback(LoadModule)

callback(LoadModule (proc,mod)) =

displayMessage("Loaded module: " ++ mod.name()) record mod in shell

forall bp **in** shell.BPs bind all breakpoints to the mod (in any order)

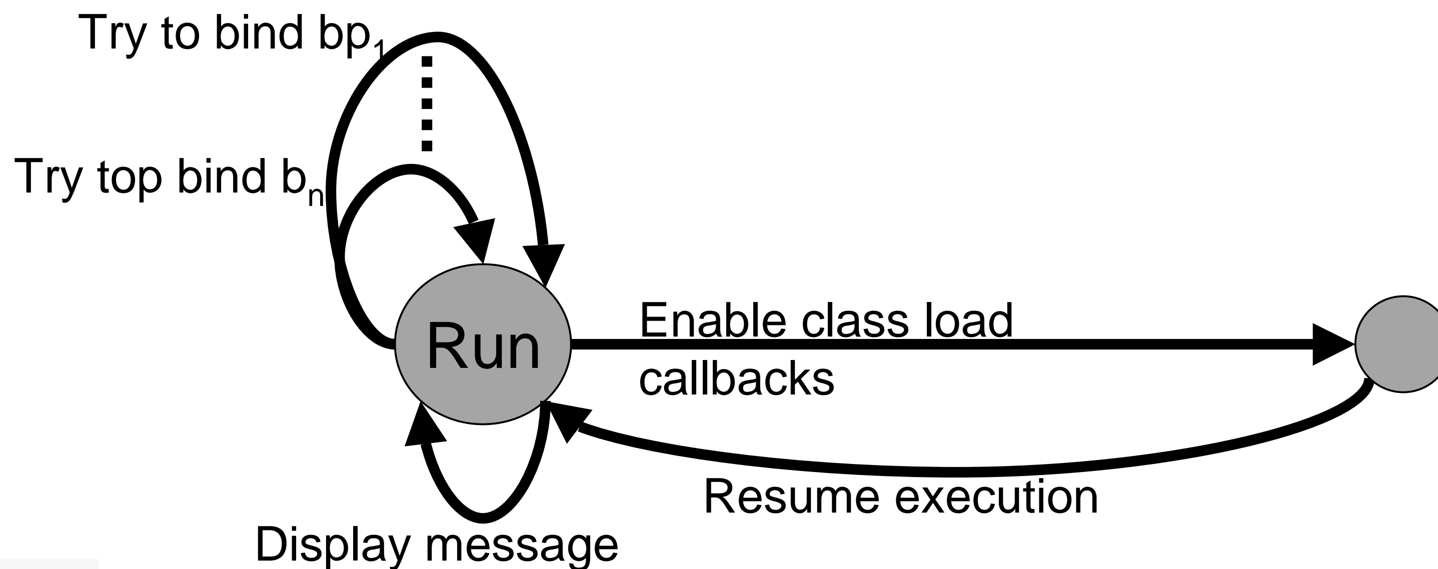
bp.bind(mod)

seq

mod.enableClassLoadCallbacks()

proc.resume() continue via external call

Analogously for UnloadModule



See Asm Book Ch. 3.1

Slide courtesy
M. Barnett
M. Veanes

The practical benefits of ASMs

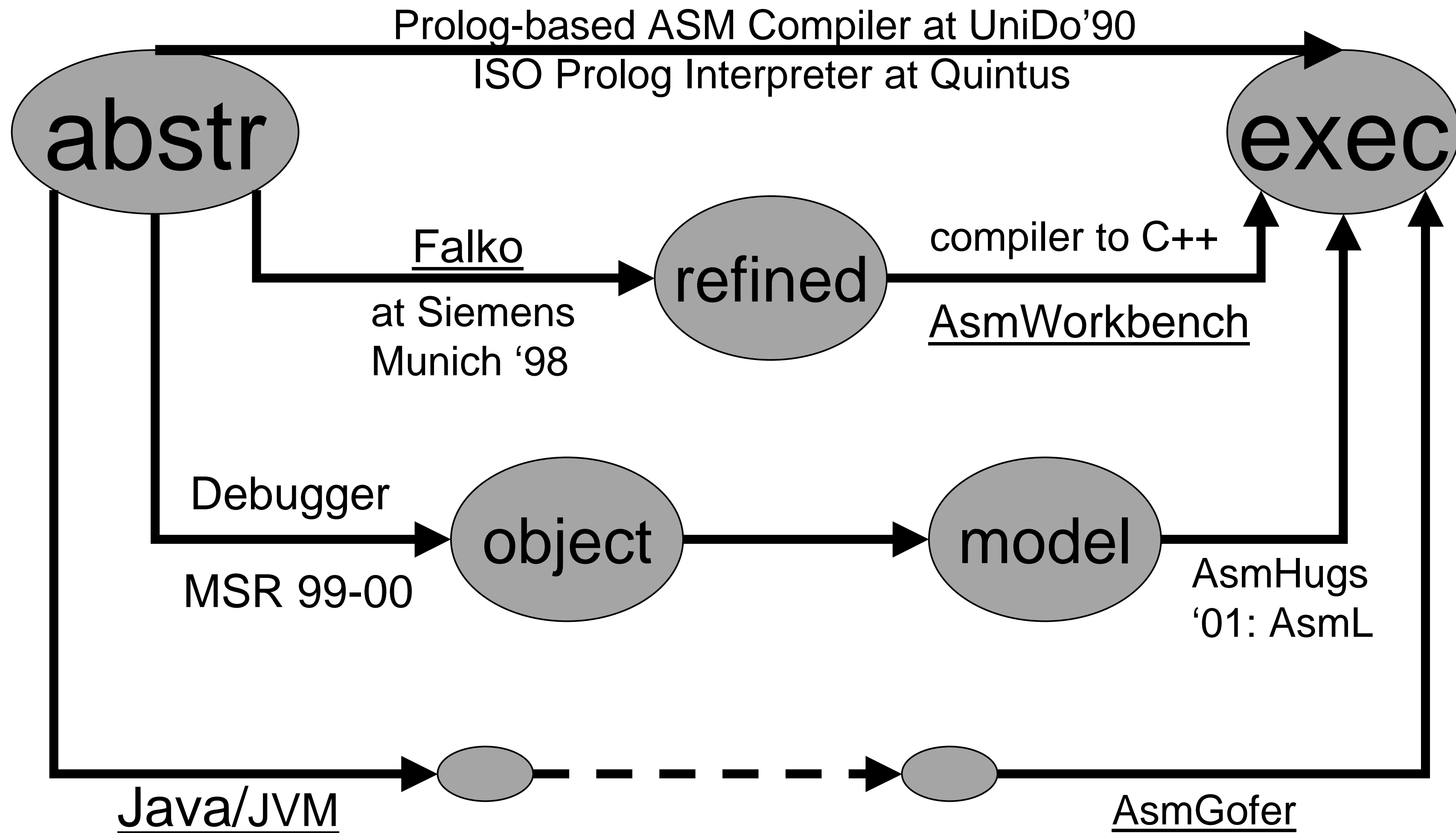
- The definition provides rigorous yet practical (process oriented & easy to use) semantics for pseudocode on arbitrary data structures, which
  - has clear notion of state & state transition,
  - can be made executable (in various natural ways, compatible with cross-language interoperable implementations, e.g. in .NET),
  - is tunable to desired level of abstraction with
    - well defined interfaces (rigor without formal overkill)
    - a most general refinement notion supporting a method of stepwise development by traceable links bw different abstraction levels

The parallel ASM execution model

- easens specification of “macro” steps (refinement & modularization)
- avoids unnecessary sequentialization of independent actions
- easens parallel/distributed implementations

Validation of ASM behavior

Make models executable by **implementing the abstractions**



Examples for Design & Verification of ASM Hierarchies

Architectures: Pipelining of microprocessors, model for VHDL,…

Control Systems: Production Cell (model checked), Steam Boiler (refinements to C++ code) Light Control (executable requirements model)

Compiler correctness

ISO Prolog to WAM: 12 refinement steps, KIV verified

backtracking, structure of predicates, structure of clauses, structure of terms & substitution, optimizations

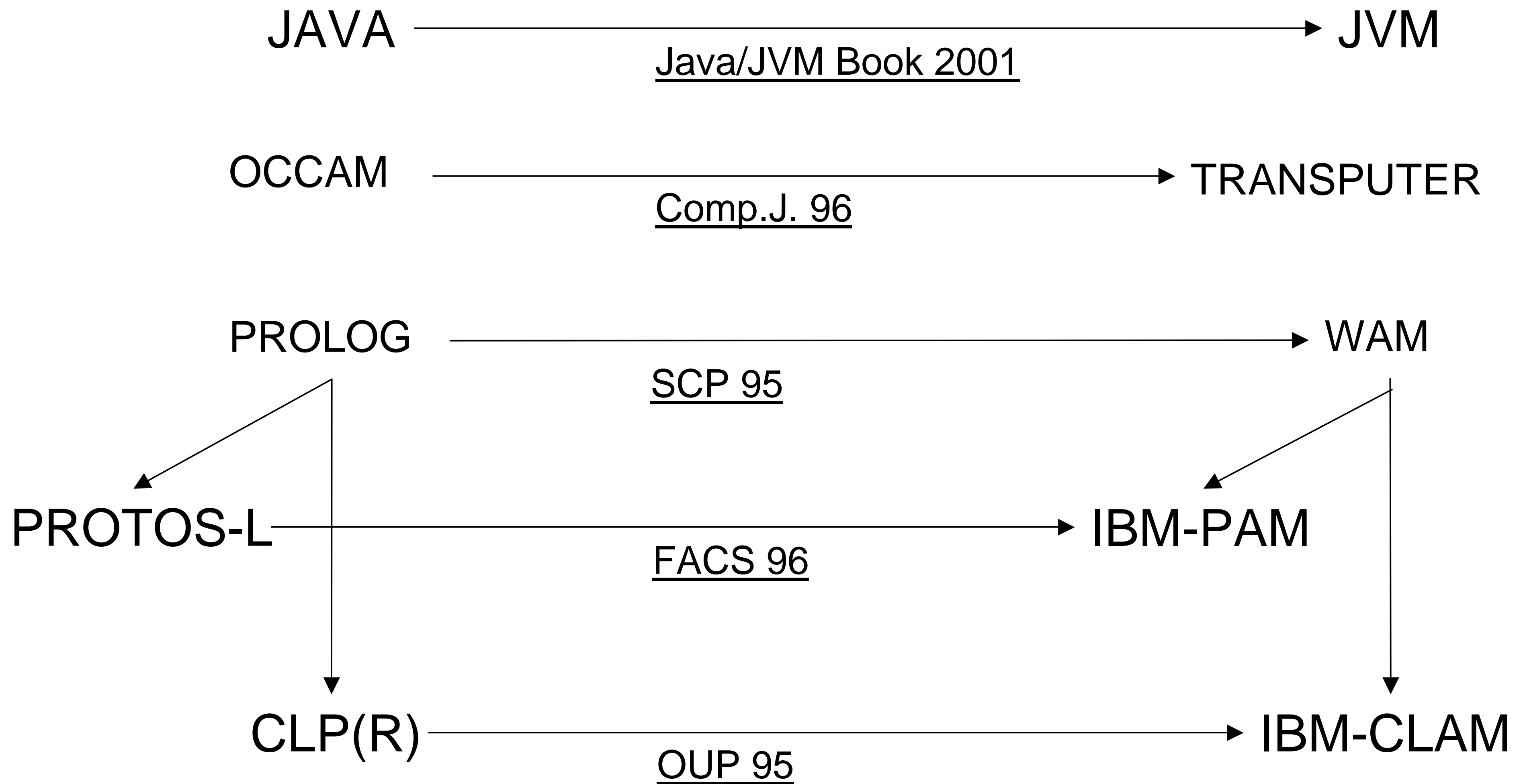
Occam to Transputer :15 models exhibiting channels, sequentialization of parallel procedures, pgm ctrl structure, env, transputer datapath and workspace, relocatable code (relative instr addresses & resolving labels)

Java to JVM: language and security driven decomposition into

5 horizontal sublanguage levels (imperative, modules, oo, exceptions, concurrency) and

4 vertical JVM levels for trustful execution, checking defensively at run time and diligently at link time, loading (modular compositional structuring)

Reusability of ASM Specifications and Verifications



Reuse of layered components (submachines) and of lemmas

References

ASM Book E. Börger, R. Stärk

Abstract State Machines

A Method for High-Level System Design and Analysis

Springer-Verlag 2003

web page http://www.di.unipi.it/AsmBook

ASM Case Study Book

R. Stärk, J. Schmid, E. Börger

Java and the Java Virtual Machine: Definition, Verification, Validation

Springer-Verlag 2001

web page http://www.inf.ethz.ch/~jbook