Event-B Decomposition

Reminder

Event-B machine consists of

- Variables (e.g., *authorised*, *location*,…)

Invariants

- Predicate logic
- Also used for type inference

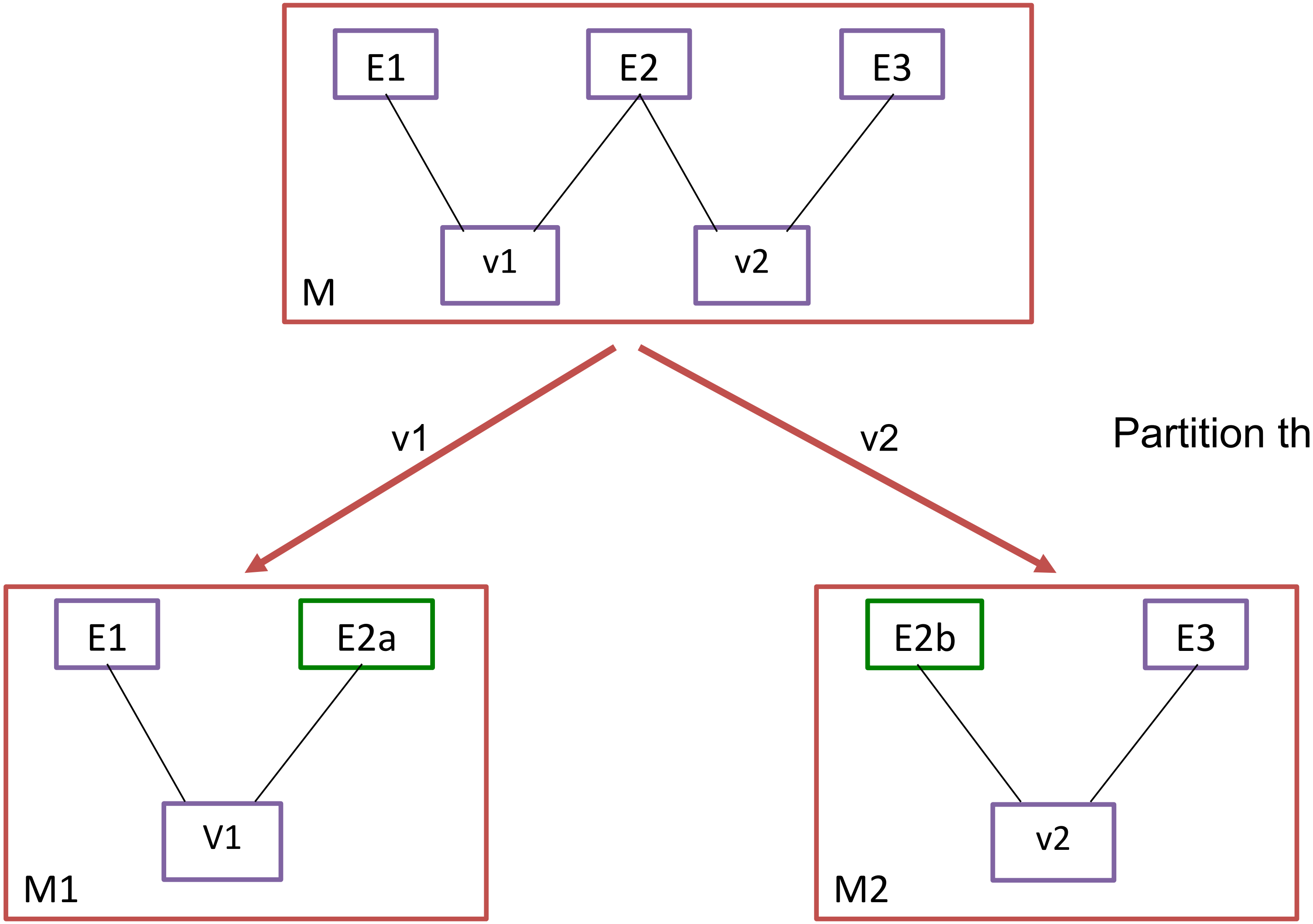
Events

- Acting on variables, expected to maintain invariants
- Specified by parameters, guards, actions

Model Decomposition styles

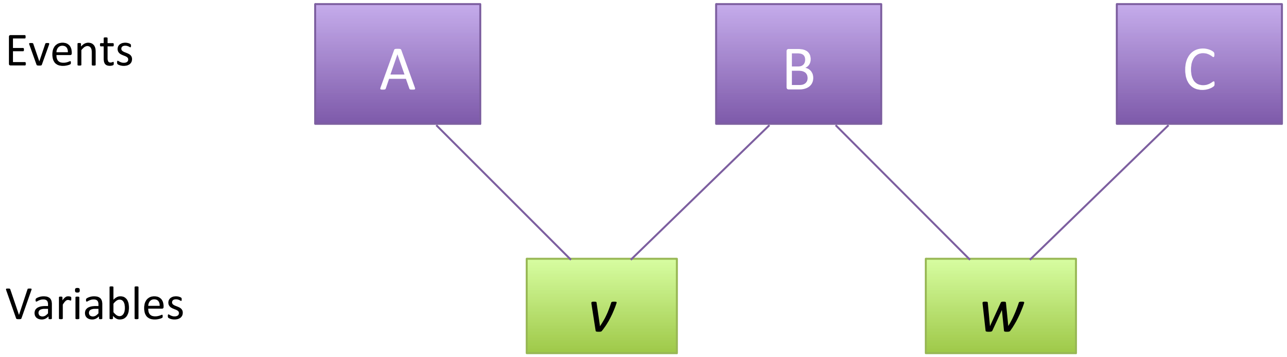
- Shared Event
  - Sub-models interact through synchronisation over shared events
  - Shared events can have common parameters
- Shared Variable
  - Sub-models interact through shared variables
  - Events are independent
- Both styles supported by a decomposition plug-in

Shared Event Decomposition



Partition the variables

Shared Event Decomposition – by example



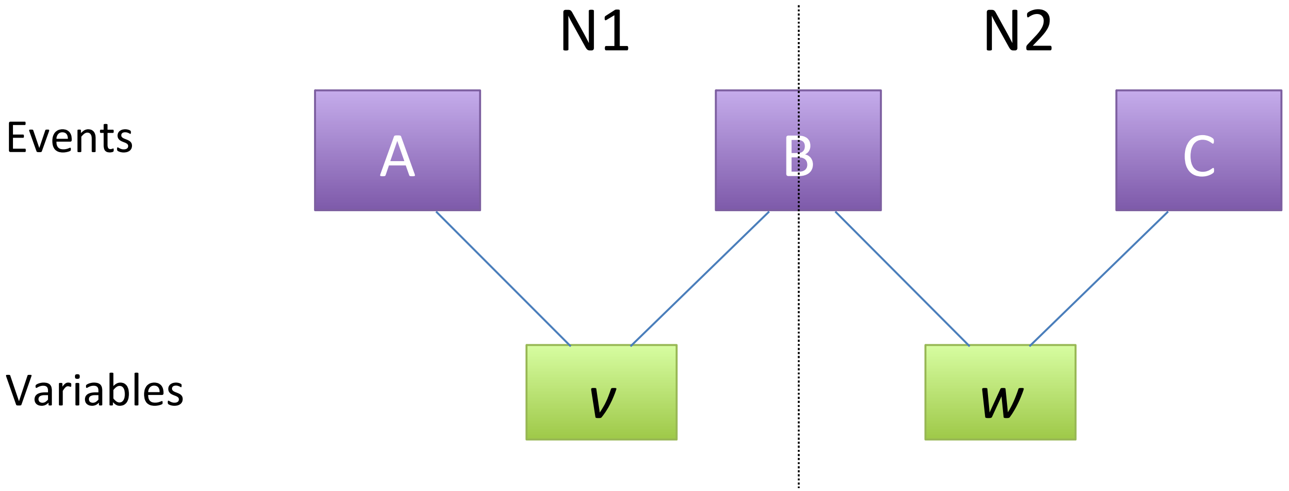
A ≜ v := v+1

B ≜ **when** v>0 ∧ w<M

then v := v-1 || w := w+1 **end**

C ≜ **when** w>0 **then** w := w-1 **end**

Partitioning the variables



A ≜ v := v+1

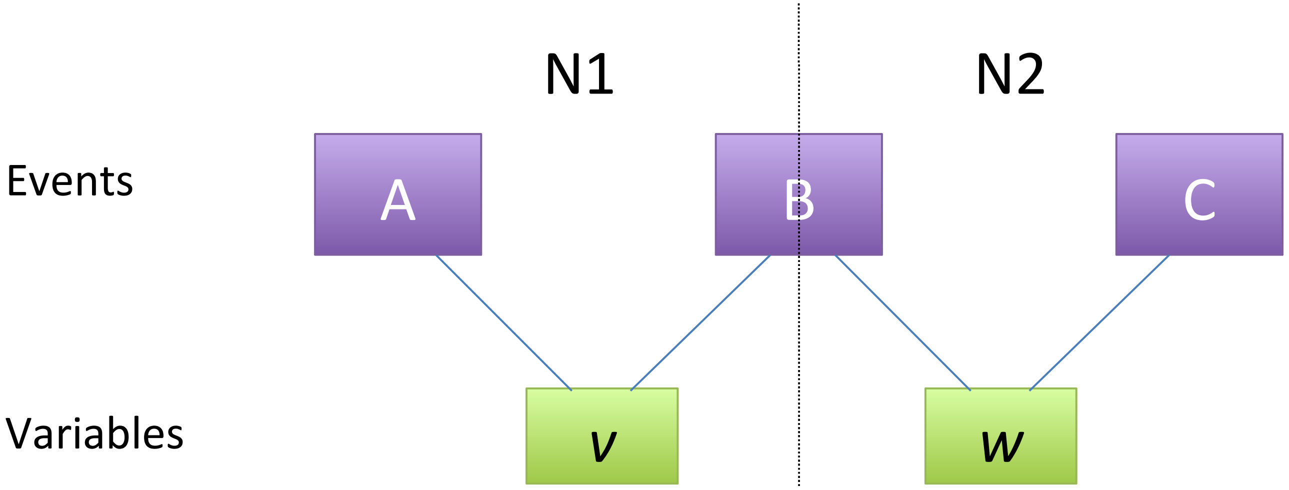
B ≜ **when** v>0 ∧ w<M

then v := v-1 || w := w+1 **end**

C ≜ **when** w>0 **then** w := w-1 **end**

B event needs to be split into *v*-part and *w*-part

Parallel Event Split



B ≙ **when** v>0 ∧ w<M

then v := v-1 || w := w+1 **end**

B is split into two parallel events operating on independent variables:

B1 ≙ **when** v>0

then v := v-1 **end**

B2 ≙ **when** w<M

then w := w+1 **end**

Synchronised events with parameter passing

B $\triangleq$ **any** x **where** $0 < x \leq v$

then v := v-x || w := w+x **end**

B can be split into 2 events that have x in common:

B1 $\triangleq$ **any** x **where** $0 < x \leq v$

then v := v-x **end**

B2 $\triangleq$ **any** x **where** $x \in \mathbb{Z}$

then w := w+x **end**

B1 constrains the value for *x* by $0 < x \leq v$ (output)

B2 just constrains the value of *x* to a type (input)

Partitioning variables

```
E = any  p  where
    GRD1( x, p )
    GRD2( y, p )
then
    x := EXP1( x, p )
    y := EXP2( y, p )
end
```

```
Ex =  any  p where
        GRD1( x, p )
then
        x := EXP1( x, p )
end
Ey = any  p  where
        GRD2( y, p )
then
        y := EXP2( y, p )
end
```

Pre-partitioning

```
E = any  p  where
    GRD1( x, p, f(y) )
    GRD2( y, p )
then
    x := EXP1( x, p, f(y) )
    y := EXP2( y, p )
end
```

```
E = any  p, q  where
    q = f(y)
    GRD1( x, p, q )
    GRD2( y, p )
then
    x := EXP1( x, p, q )
    y := EXP2( y, p )
end
```

Transform E to help the split into *x*-part and *y*-part

Composition and Decomposition

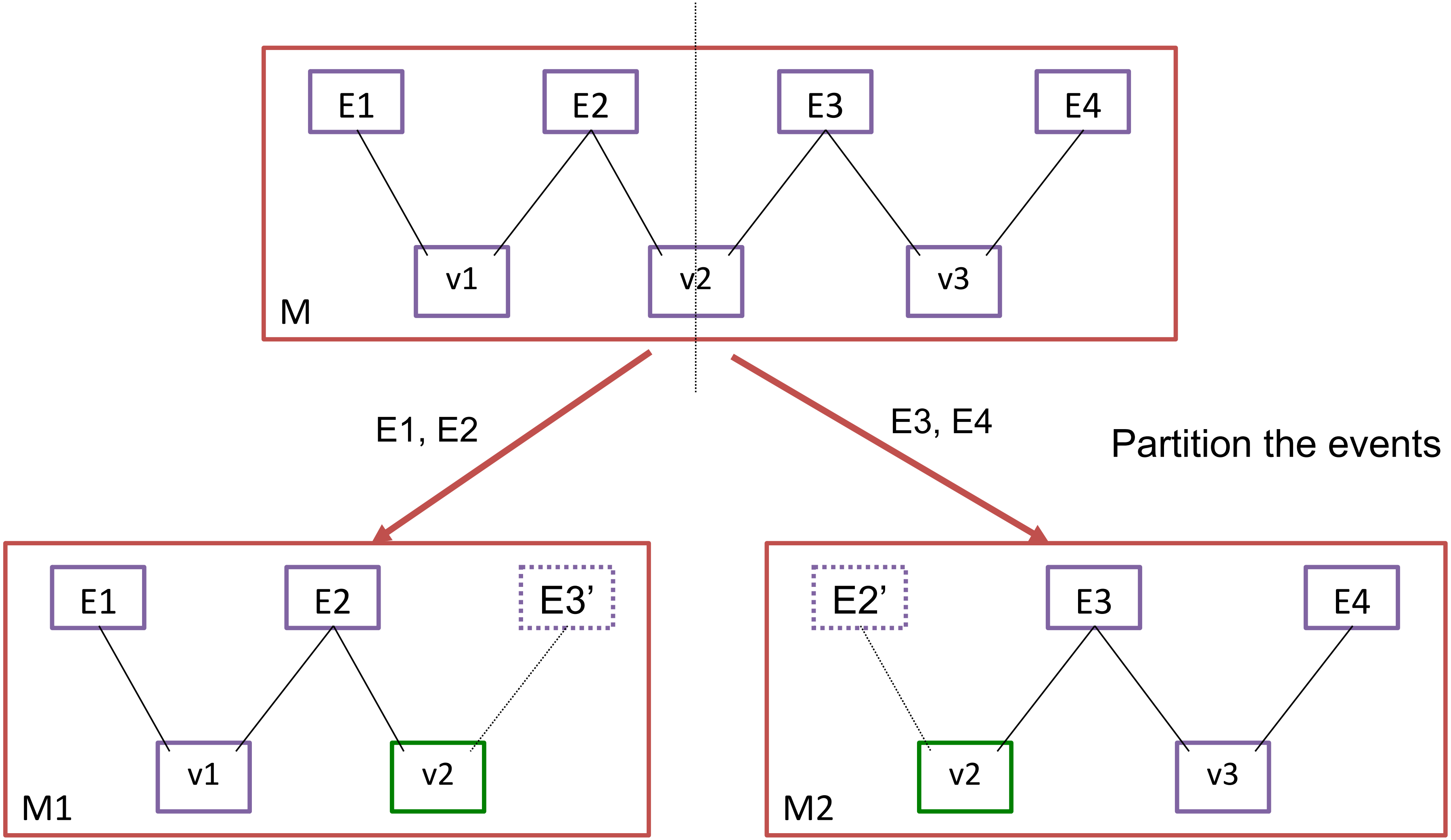
- Decomposition: from M, decomposition plug-in generates:
  - machines L, P
  - composed machine M’
- M’ is a wrapper for L || P
- Consistency of decomposition:
  - prove M’ refines M

```
composed machine
M’
refines  M
Includes  L, P
events
      A  =  L.A
      B  =  L.B || P .B
      C  =  P.C
end
```

Shared event composition

- Shared event composition operator for Event-B machines is syntactically simple
  - combine guards and combine actions of events to be synchronised
  - no shared state variables
  - common event parameters represent values to be agreed by both parties on synchronisation
- Corresponds to parallel composition in CSP
  - processes interact via synchronised channels
  - monotonic: subsystems can be refined independently

Shared Variable Decomposition



Partition the events

E2' and E3' are *external* events

Terminology of Decomposition

Private variables are only referred to in events of the parent machine.

Shared variables are accessed by events of other machines.

For the Shared Variable Style

External events simulate the way that shared variables are updated by other machines.

Internal events update shared and private variables.

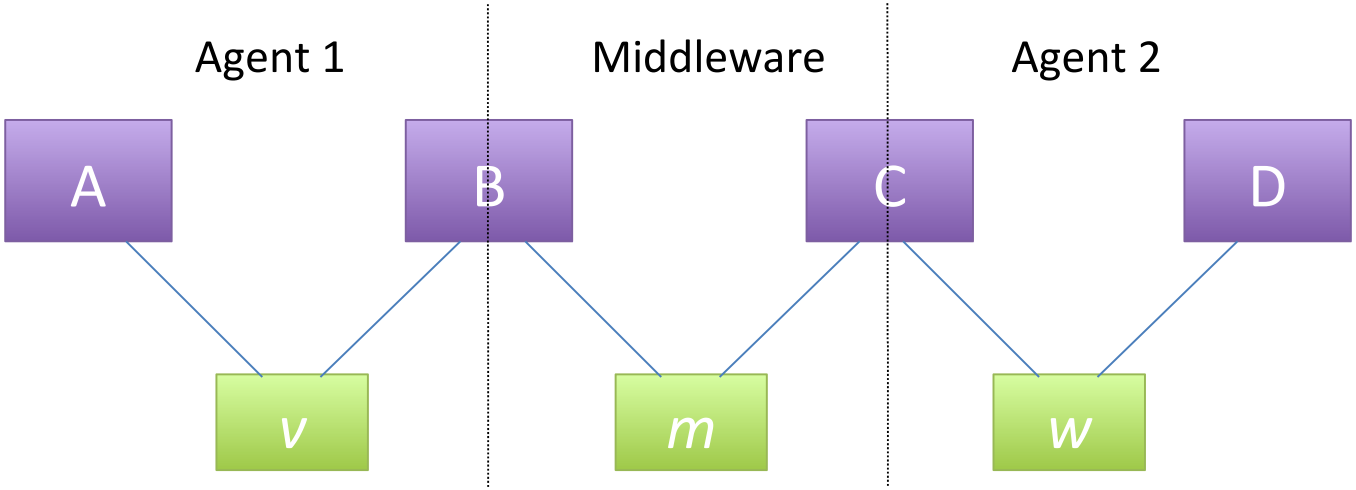
Refinement after decomposition

- Shared event: can refine sub-model provided
- Common parameters of shared events are consistently maintained
- Shared variable: can refine sub-model provided
    - External events are not refined (rely condition)
    - Private events in M1 that affect shared variables must refine some external event of M2, e.g., E3 refines E3’
    - Shared variables are not refined.
    - Invariants used in refinement are preserved by external events

Observation on Decomposition

- The decomposition itself is straightforward
  - Essentially a syntactic partitioning of events
- The more challenging part is refining the abstract model to a sufficiently detailed model to allow the syntactic decomposition to take place
- Our code generation approach makes use of Shared Event Decomposition

Asynchronous distributed system



For distributed systems, agents do not interact directly.

Instead they interact via some middleware, e.g., the Internet