

# Systems Design Laboratory

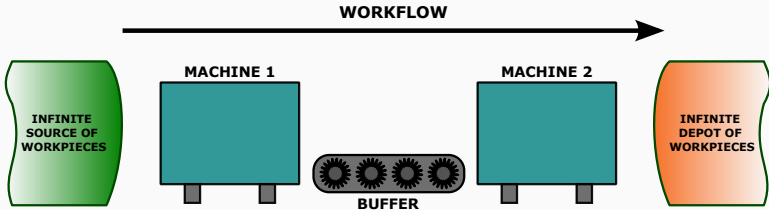
A Manufacturing Process

---

**Matteo Zavatteri**

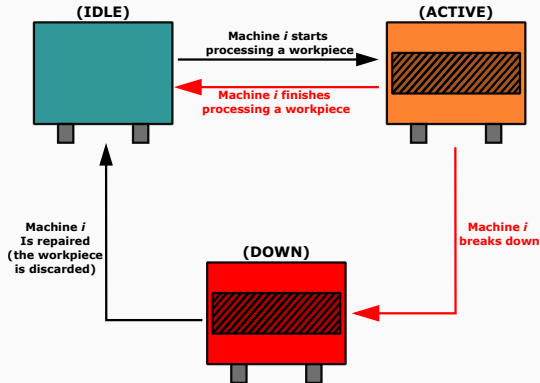
Department of Computer Science, University of Verona, ITALY

# A Manufacturing Process



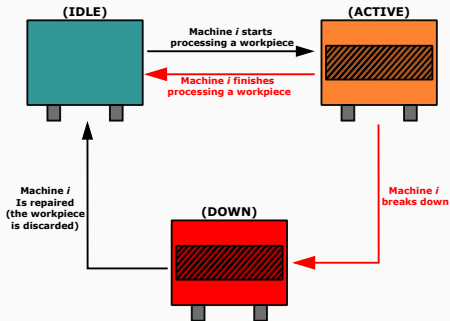
- The workflow is intended “left to right”
- Two machines processing workpieces
  - Machine 1 has an infinite source of workpieces
  - Machine 2 has an infinite depot of workpieces
- A Buffer (e.g., a conveyor) passing workpieces from Machine 1 to Machine 2

## Machine $i = 1, 2$



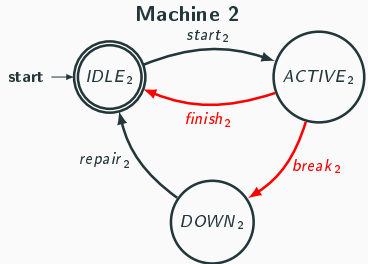
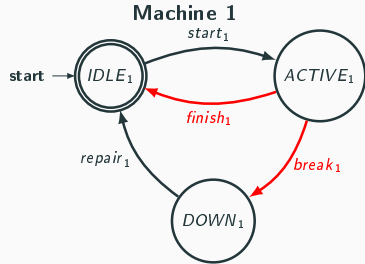
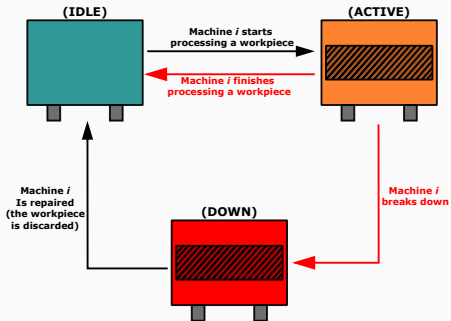
- Machine  $i$  starts and finishes processing workpieces (exactly like the Machine-Warehouse example)
- Machine  $i$  can also break down. If it does, it can be repaired and the workpiece being processed is discarded
- Machine  $i$  can't be prevented from finishing or breaking (why?)
- Initially, Machine  $i$  is IDLE.

# Automaton for Machine $i = 1, 2$

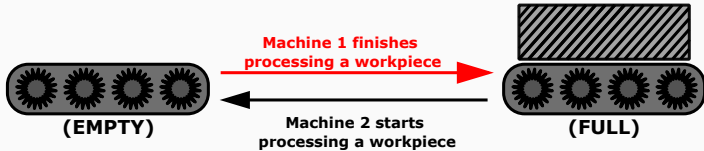


- States?
- Transitions?
- Event controllability?

# Automaton for Machine $i = 1, 2$



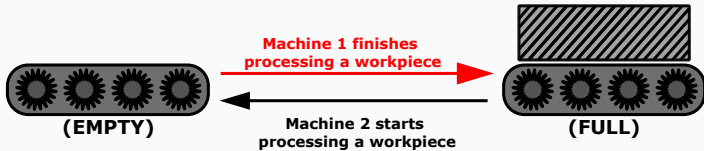
# Buffer $B$



- Buffer has a capacity of 1 workpieces
- Buffer is synchronized with Machine 1 and Machine 2
- Buffer fills when Machine 1 finishes processing a workpiece
- Buffer empties when Machine 2 starts processing a workpiece
- Initially, the Buffer is empty

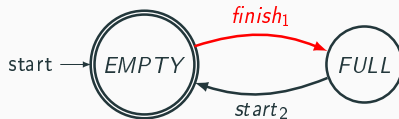
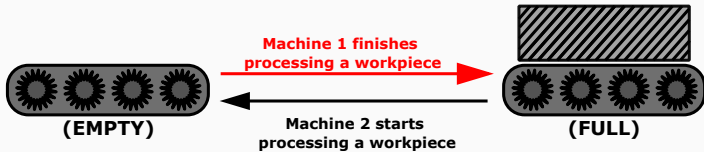
In other words, Machine 1 puts workpieces on the buffer, whereas Machine 2 removes workpieces from the buffer

# Automaton for Buffer $B$



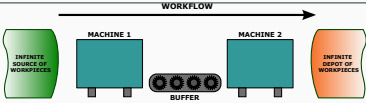
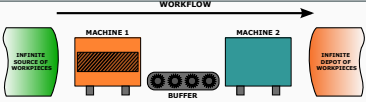
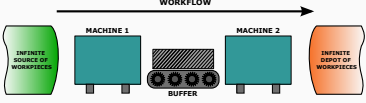
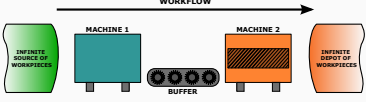
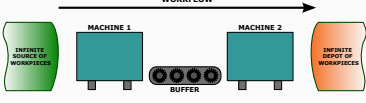
- States?
- Transitions?
- Event controllability?

# Automaton for Buffer $B$

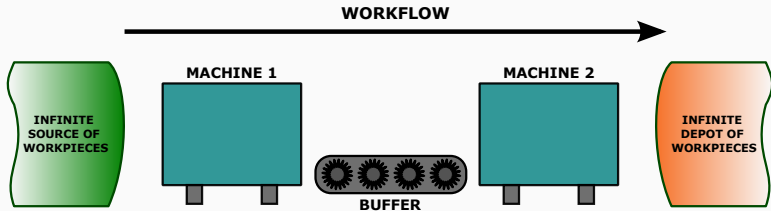




# A Manufacturing Process - Usecase Example

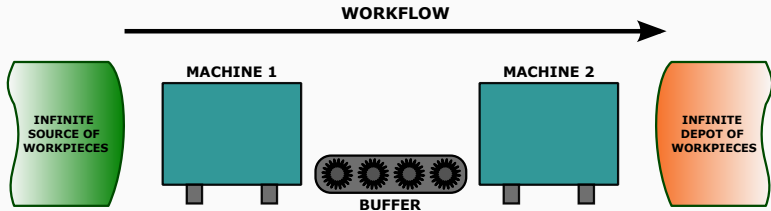
Graphical Representation	Description
	Whole system is idle
	$M_1$ starts processing a workpiece
	$M_1$ finishes processing the workpiece
	$M_2$ starts processing the workpiece
	$M_2$ finishes processing the workpiece

# What about source and depot?

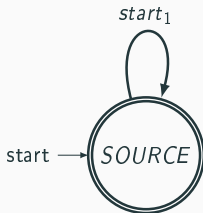


- Can you think about two automata to model them?
- States?
- Transitions?
- Event controllability?

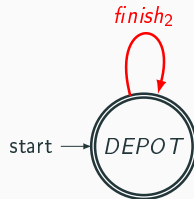
# What about source and depot?



Source

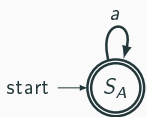
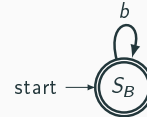
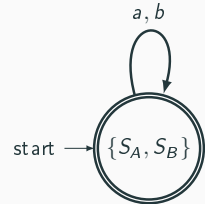
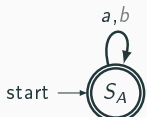
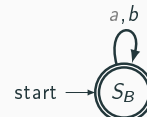
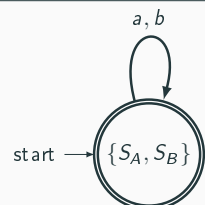


Depot

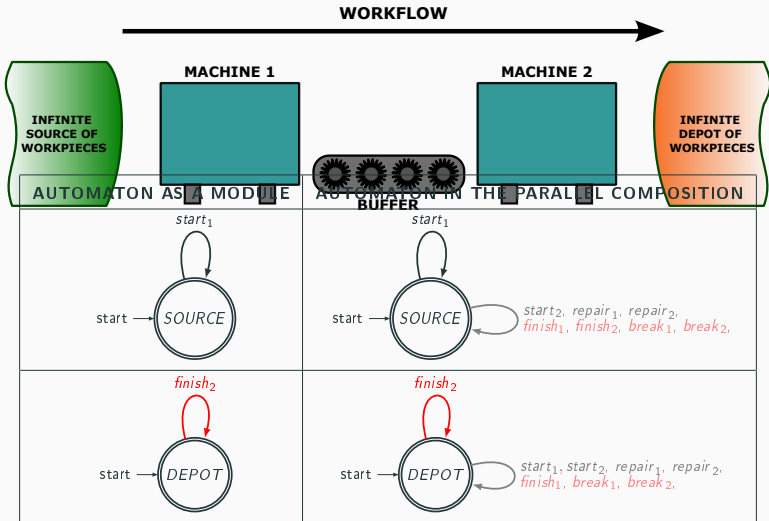


Do we really need them?

# Recall on the Equivalence Between Parallel and Product

Automaton $A$	Automaton $B$	Automaton $A \parallel B$
 <p>start <math>\rightarrow S_A</math></p> <p><math>\Sigma_A := \{a\}</math></p>	 <p>start <math>\rightarrow S_B</math></p> <p><math>\Sigma_B := \{b\}</math></p>	 <p>start <math>\rightarrow \{S_A, S_B\}</math></p> <p><math>\Sigma_{A \parallel B} := \Sigma_A \cup \Sigma_B = \{a, b\}</math></p>
Automaton $A'$	Automaton $B'$	Automaton $A' \times B'$
 <p>start <math>\rightarrow S_A</math></p> <p><math>\Sigma_{A'} := \{a, b\}</math></p>	 <p>start <math>\rightarrow S_B</math></p> <p><math>\Sigma_{B'} := \{a, b\}</math></p>	 <p>start <math>\rightarrow \{S_A, S_B\}</math></p> <p><math>\Sigma_{A' \times B'} := \Sigma_{A'} \cup \Sigma_{B'} = \{a, b\}</math></p>

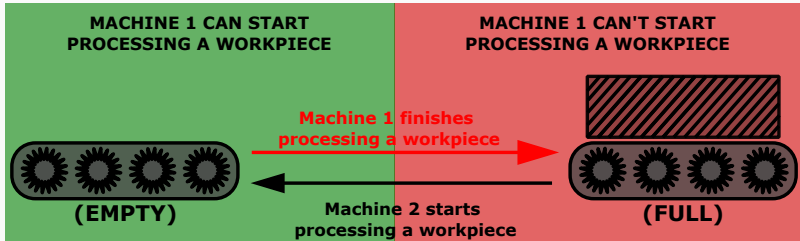
# What about source and depot?



$(\Sigma^*)$ . Thus, we do not need them.

## Requirement $R_1$ - Essential Desired Behavior

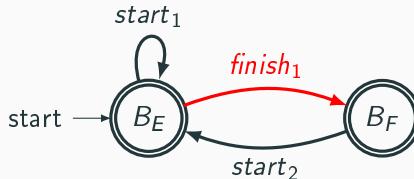
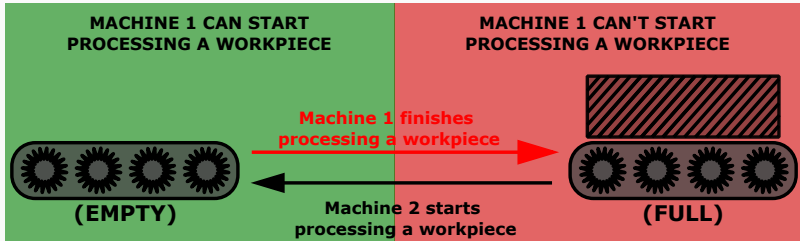
Requirement 1: Machine 1 can start processing a workpiece only if the Buffer is empty



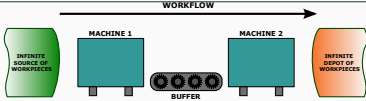
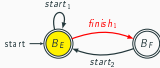
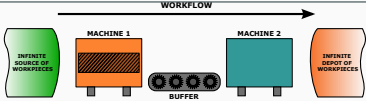
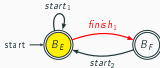
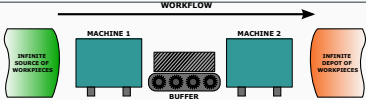


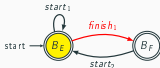
What should the automaton look like?

# Requirement $R_1$ - Essential Desired Behavior

Requirement 1: Machine 1 can start processing a workpiece only if the Buffer is empty

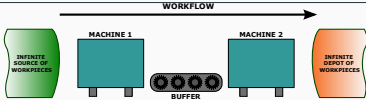
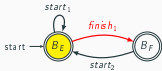
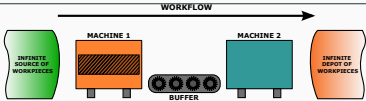

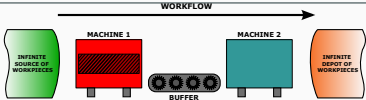
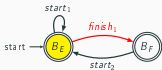
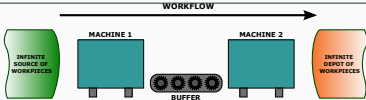
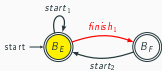


# Requirement $R_1$ - Usecase 1

Graphical Representation	Requirement	Description
 <p>WORKFLOW</p>		Whole system is idle
 <p>WORKFLOW</p>		$M_1$ starts processing a workpiece
 <p>WORKFLOW</p>		$M_1$ finishes processing the workpiece
 <p>WORKFLOW</p>		$M_2$ starts processing the workpiece

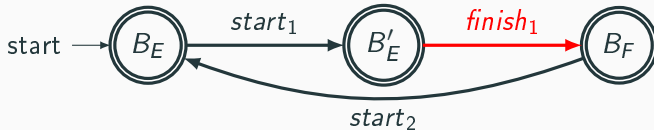
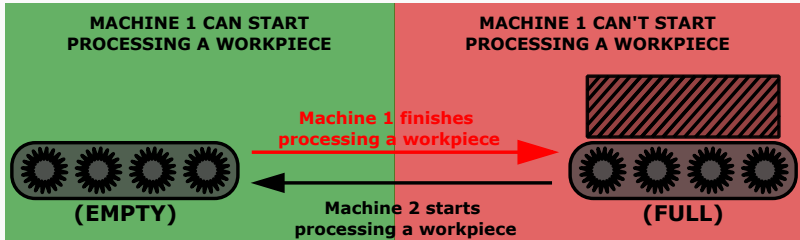


# Requirement $R_1$ - Usecase 2

Graphical Representation	Requirement	Description
 <p>WORKFLOW</p>		Whole system is idle
 <p>WORKFLOW</p>		$M_1$ starts processing a workpiece
 <p>WORKFLOW</p>		$M_1$ breaks down
 <p>WORKFLOW</p>		$M_1$ is repaired and it can start again

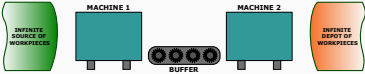
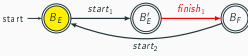



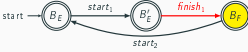


# What about this version for $R_1$ ? Right or wrong?

Requirement 1: Machine 1 can start processing a workpiece only if the Buffer is empty



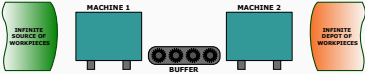
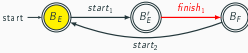



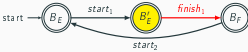

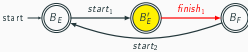
Could this be another automaton modeling  $R_1$ ?

# Right or wrong? - Usecase 1

<p>WORKFLOW →</p> 		<p>Whole system is idle</p>
<p>WORKFLOW →</p> 		<p><math>M_1</math> starts processing a workpiece</p>
<p>WORKFLOW →</p> 		<p><math>M_1</math> finishes processing the workpiece</p>
<p>WORKFLOW →</p> 		<p><math>M_2</math> starts processing the workpiece</p>

Seems working, right?  
...right?

# Right or wrong? - Usecase 2

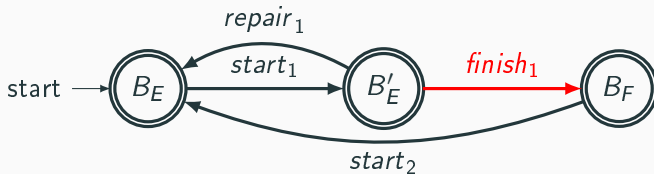
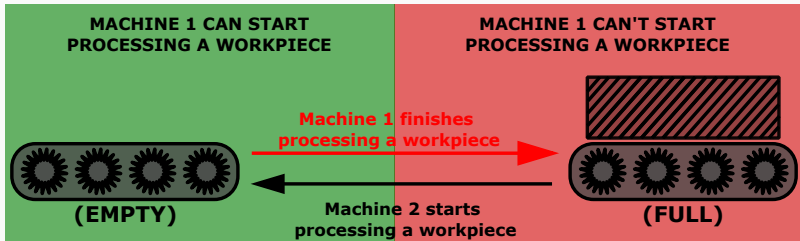
<p>WORKFLOW →</p> 		<p>Whole system is idle</p>
<p>WORKFLOW →</p> 		<p><math>M_1</math> starts processing a workpiece</p>
<p>WORKFLOW →</p> 		<p><math>M_1</math> breaks down</p>
<p>WORKFLOW →</p> 		<p><math>M_1</math> is repaired and it cannot start again</p>

Wrong! But still OK if Machine 1 never breaks (Usecase 1)

What's missing?

## Alternative $R_1$

Requirement 1: Machine 1 can start processing a workpiece only if the Buffer is empty

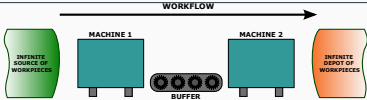
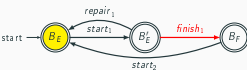
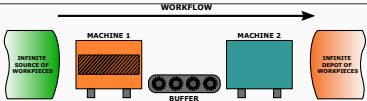
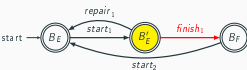
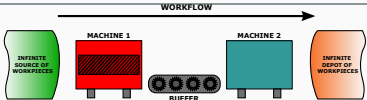
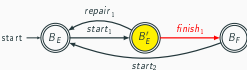
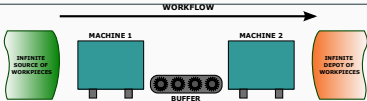
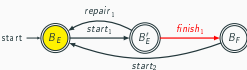


Right! Can we simplify it?

# Alternative $R_1$ - Usecase 1

Graphical Representation	Requirement	Description
<p>WORKFLOW</p>		Whole system is idle
<p>WORKFLOW</p>		$M_1$ starts processing a workpiece
<p>WORKFLOW</p>		$M_1$ finishes processing the workpiece
<p>WORKFLOW</p>		$M_2$ starts processing the workpiece

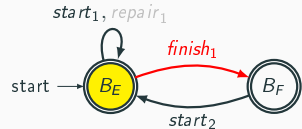
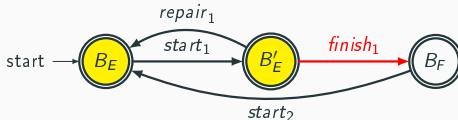
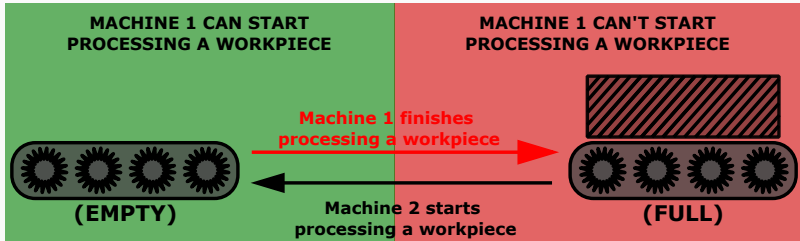
# Alternative $R_1$ - Usecase 2

Graphical Representation	Requirement	Description
 <p>Workflow diagram showing Machine 1 and Machine 2 in a production line. Machine 1 is blue, Machine 2 is blue, and the buffer is empty. An infinite source of workpieces is on the left and an infinite depot is on the right.</p>	 <p>Petri net diagram for the initial state. Place <math>B_E</math> is yellow. Transitions <math>start_1</math>, <math>start_2</math>, and <math>finish_1</math> are labeled. Place <math>B_F</math> is empty.</p>	Whole system is idle
 <p>Workflow diagram showing Machine 1 is orange with a hatched pattern, indicating it is processing a workpiece. Machine 2 is blue, and the buffer is empty.</p>	 <p>Petri net diagram for the state where Machine 1 is processing. Place <math>B'_E</math> is yellow. Transitions <math>start_1</math>, <math>start_2</math>, and <math>finish_1</math> are labeled.</p>	$M_1$ starts processing a workpiece
 <p>Workflow diagram showing Machine 1 is red with a hatched pattern, indicating it has broken down. Machine 2 is blue, and the buffer is empty.</p>	 <p>Petri net diagram for the state where Machine 1 has broken down. Place <math>B'_E</math> is yellow. Transitions <math>start_1</math>, <math>start_2</math>, and <math>finish_1</math> are labeled.</p>	$M_1$ breaks down
 <p>Workflow diagram showing Machine 1 is blue, indicating it has been repaired. Machine 2 is blue, and the buffer is empty.</p>	 <p>Petri net diagram for the state where Machine 1 has been repaired. Place <math>B_E</math> is yellow. Transitions <math>start_1</math>, <math>start_2</math>, and <math>finish_1</math> are labeled.</p>	$M_1$ is repaired and it can start again

Correct! Can we simplify the requirement?

## Alternative $R_1$ - Simplification

Requirement 1: Machine 1 can start processing a workpiece only if the Buffer is empty

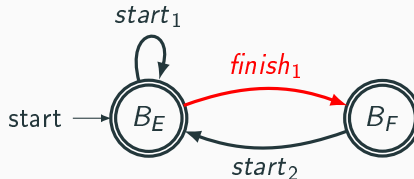
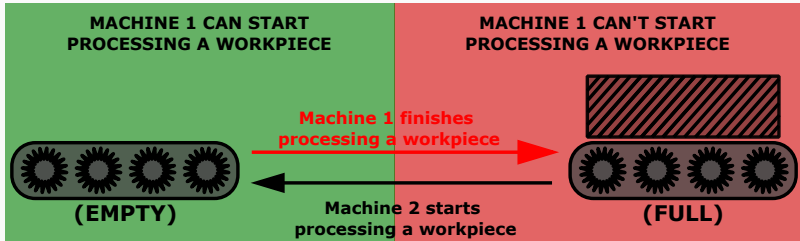


Can Machine 1 be repaired if it didn't even start?  
(=can we further simplify the requirement?)



# Requirement $R_1$ - Essential Desired Behavior

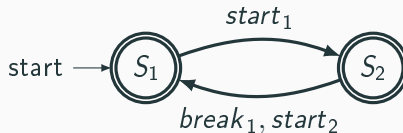
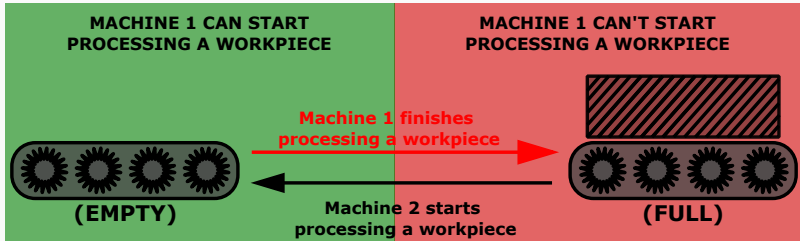
Requirement 1: Machine 1 can start processing a workpiece only if the Buffer is empty



That's it! You got it!

## Another alternative version for $R_1$ - Right or wrong?

Requirement 1: Machine 1 can start processing a workpiece only if the Buffer is empty

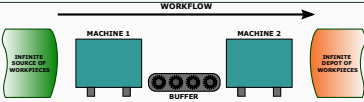
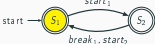
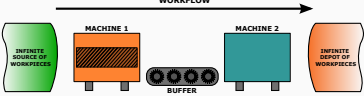

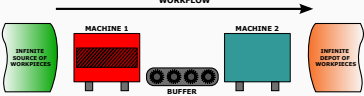

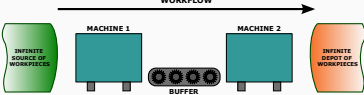



What about this one (no longer related to the buffer automaton)?

# Alternative $R_1$ - Usecase 1

Graphical Representation	Requirement	Description
<p>WORKFLOW</p>		Whole system is idle
<p>WORKFLOW</p>		$M_1$ starts processing a workpiece
<p>WORKFLOW</p>		$M_1$ finishes processing the workpiece
<p>WORKFLOW</p>		$M_2$ starts processing the workpiece

# Alternative $R_1$ - Usecase 2

Graphical Representation	Requirement	Description
 <p>Workflow diagram showing Machine 1 (blue) idle, Buffer (empty) idle, and Machine 2 (blue) idle. An arrow labeled 'WORKFLOW' points from left to right.</p>	 <pre> graph LR     start((start)) -- start1 --&gt; S1((S1))     S1 -- break1, start2 --&gt; S2(((S2)))     style S1 fill:#ffff00     style S2 fill:#d3d3d3         </pre>	Whole system is idle
 <p>Workflow diagram showing Machine 1 (orange with diagonal lines) processing a workpiece, Buffer (empty) idle, and Machine 2 (blue) idle. An arrow labeled 'WORKFLOW' points from left to right.</p>	 <pre> graph LR     start((start)) -- start1 --&gt; S1(((S1)))     S1 -- break1, start2 --&gt; S2(((S2)))     style S1 fill:#d3d3d3     style S2 fill:#d3d3d3         </pre>	$M_1$ starts processing a workpiece
 <p>Workflow diagram showing Machine 1 (red with diagonal lines) broken down, Buffer (empty) idle, and Machine 2 (blue) idle. An arrow labeled 'WORKFLOW' points from left to right.</p>	 <pre> graph LR     start((start)) -- start1 --&gt; S1((S1))     S1 -- break1, start2 --&gt; S2(((S2)))     style S1 fill:#ffff00     style S2 fill:#d3d3d3         </pre>	$M_1$ breaks down
 <p>Workflow diagram showing Machine 1 (blue) repaired and ready, Buffer (empty) idle, and Machine 2 (blue) idle. An arrow labeled 'WORKFLOW' points from left to right.</p>	 <pre> graph LR     start((start)) -- start1 --&gt; S1((S1))     S1 -- break1, start2 --&gt; S2(((S2)))     style S1 fill:#ffff00     style S2 fill:#d3d3d3         </pre>	$M_1$ is repaired and can start again

Correct!

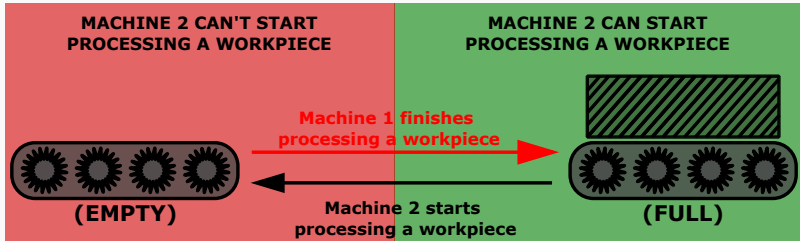
# Automata for $R_1$ - Summary of Equivalent Versions

Version	Automaton	Modeling Intuition
Version 1	<pre> graph LR     start((start)) -- start_1 --&gt; BE((B_E))     BE -- finish_1 --&gt; BF(((B_F)))     BF -- start_2 --&gt; BE     style start fill:none,stroke:none     style BE fill:none,stroke:none     style BF fill:none,stroke:none         </pre>	A modified copy of Buffer
Version 2	<pre> graph LR     start((start)) -- start_1 --&gt; BE(((B_E)))     BE -- repair_1 --&gt; BEP(((B'_E)))     BEP -- finish_1 --&gt; BF(((B_F)))     BF -- start_2 --&gt; BE     style start fill:none,stroke:none     style BE fill:none,stroke:none     style BEP fill:none,stroke:none     style BF fill:none,stroke:none         </pre>	Still a copy of buffer in some sense
Version 3	<pre> graph LR     start((start)) -- start_1 --&gt; S1(((S_1)))     S1 -- break_1, start_2 --&gt; S2(((S_2)))     S2 -- start_1 --&gt; S1     style start fill:none,stroke:none     style S1 fill:none,stroke:none     style S2 fill:none,stroke:none         </pre>	Not from a copy of the buffer

Homework: check that the effect of each version of  $R_1$  on the plant is the same.

## Requirement $R_2$ - Automaton

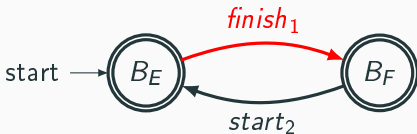
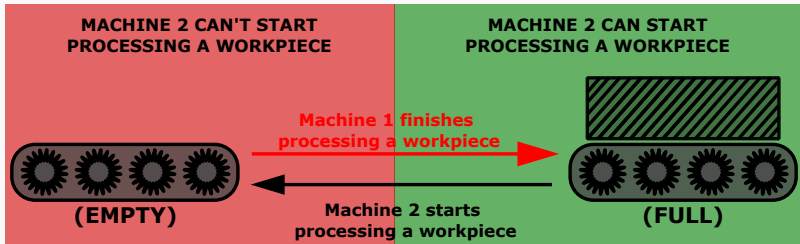
Requirement 2: Machine 2 can start processing a workpiece only if the Buffer is full



- States?
- Transitions?
- Event controllability?

## Requirement $R_2$ - Essential Desired Behavior

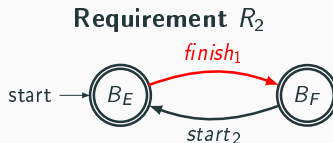
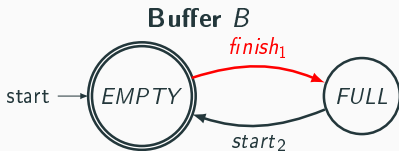
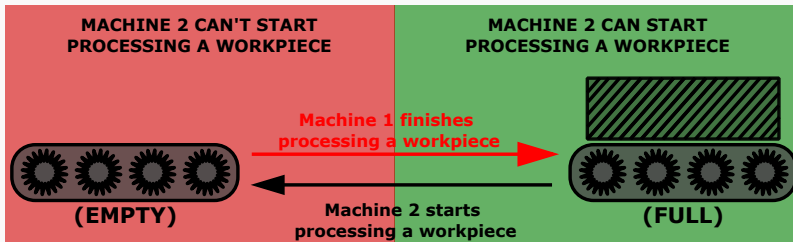
Requirement 2: Machine 2 can start processing a workpiece only if the Buffer is full



Doesn't it look familiar?

## Requirement $R_2$ - Essential Desired Behavior

Requirement 2: Machine 2 can start processing a workpiece only if the Buffer is full

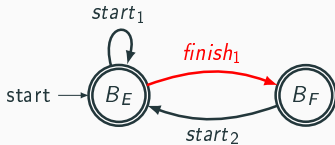


$R_2$  is already enforced by the plant. Note that  $B \parallel R_2 = B \times R_2 = B$ .

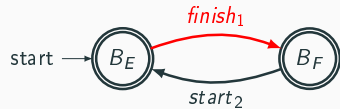


# Requirement $R_{1,2}$ - Parallel composition point of view

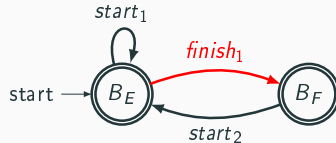
Requirement  $R_1$  (v.1): Machine 1 can start processing a workpiece only if the Buffer is empty



Requirement  $R_2$ : Machine 2 can start processing a workpiece only if the Buffer is full

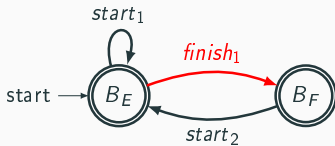


Requirement  $R_{1,2} := R_1 \parallel R_2$

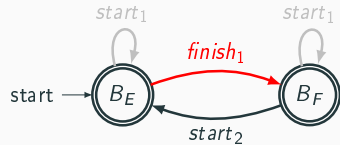


# Requirement $R_{1,2}$ - Product point of view

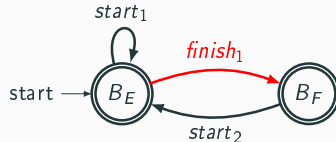
Requirement  $R_1$  (v.1): Machine 1 can start processing a workpiece only if the Buffer is empty



Requirement  $R_2$ : Machine 2 can start processing a workpiece only if the Buffer is full

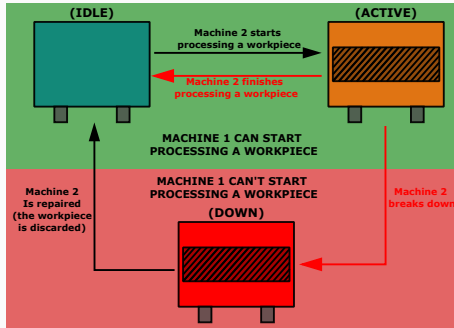


Requirement  $R_{1,2} := R_1 \times R_2$



# Requirement $R_3$ - Essential Desired Behavior

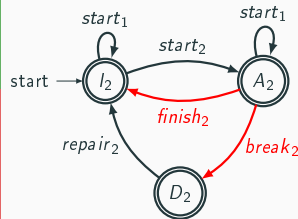
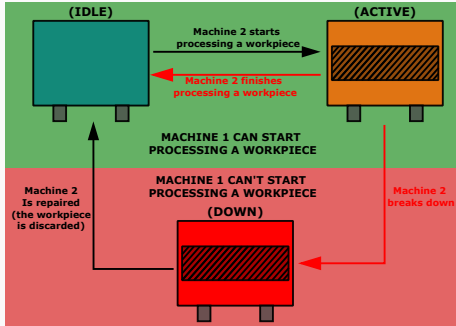
Requirement 3: Machine 1 can't start processing a workpiece if Machine 2 is down.



- States?
- Transitions?
- Event controllability?

## Requirement $R_3$ - Attempt 1

Requirement 3: Machine 1 can't start processing a workpiece if Machine 2 is down.

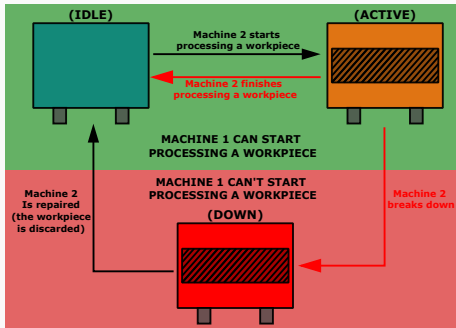


Correct, but maybe not "so essential".

Can we get a smaller automaton?

## Requirement $R_3$ - Attempt 2

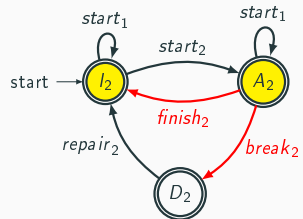
Requirement 3: Machine 1 can't start processing a workpiece if Machine 2 is down.



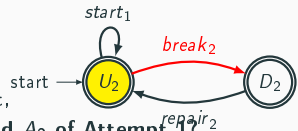
Note: to capture the essence of the requirement, we do not need  $start_2$  and  $finish_2$ .

Can you now see why we can merge states  $I_1$  and  $A_2$  of Attempt 1?

Requirement  $R_3$  - Attempt 1



Requirement  $R_3$  - Attempt 2



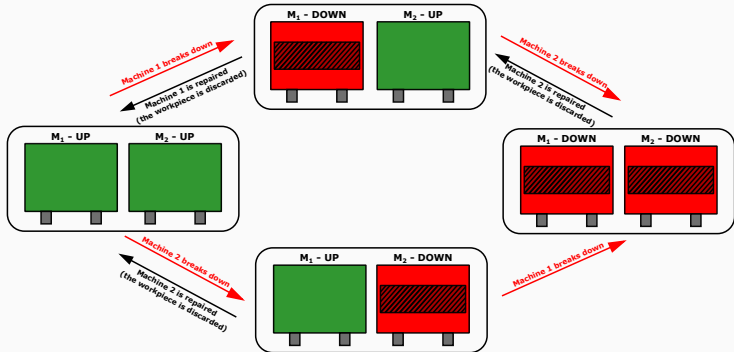
# Automata for $R_3$ - Summary of Equivalent Versions

Version	Automaton	Modeling Intuition
Version 1		A modified copy of Machine 2
Version 2		Still a copy of Machine 2 in some sense

**Homework:** check that the effect of each version of  $R_3$  on the plant is the same.

# Requirement $R_4$ - Attempt 1

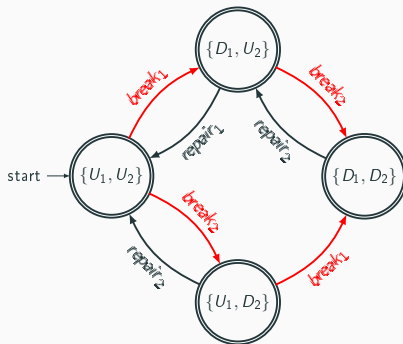
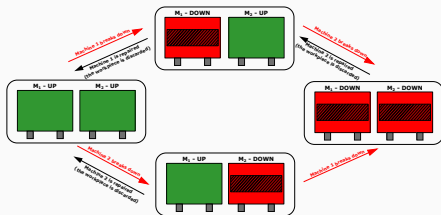
Requirement 4: If both Machines are down, then Machine 2 is repaired before Machine 1.



- States?
- Transitions?
- Event controllability?

# Requirement $R_4$ - Attempt 1

Requirement 4: If both Machines are down, then Machine 2 is repaired before Machine 1.



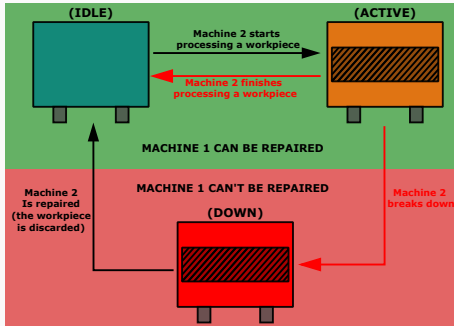
Easy and correct.

Can we improve on the essentiality of the requirement?



# Requirement $R_4$ - Attempt 2 - Desired Behavior

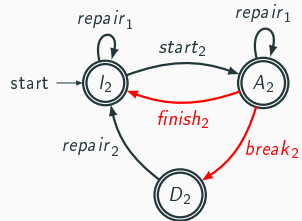
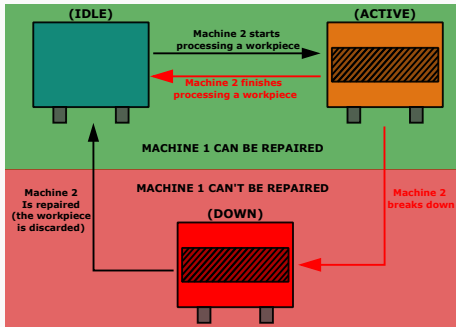
Requirement 4: If both Machines are down, then Machine 2 is repaired before Machine 1.



- States?
- Transitions?
- Event controllability?

# Requirement $R_4$ - Attempt 2 - Automaton

Requirement 4: If both Machines are down, then Machine 2 is repaired before Machine 1.



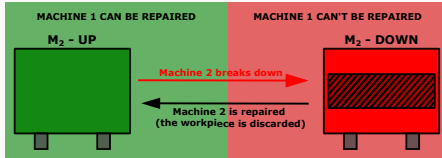
Easy and correct.

Can we improve on the essentiality of the requirement?

Rationale: When Machine 2 is down if we repair Machine 1 it means that Machine 1 is down as well.

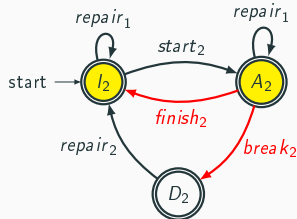
# Requirement $R_4$ - Attempt 3 - Automaton

Requirement 4: If both Machines are down, then Machine 2 is repaired before Machine 1.

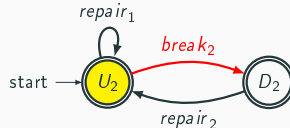


Recall the concept of  
“Machine is UP”  
(=Machine is NOT down)

Requirement  $R_4$  - Attempt 2



Requirement  $R_4$  - Attempt 3



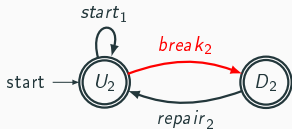
# Automata for $R_4$ - Summary of Equivalent Versions

Version	Automaton	Modeling Intuition
Version 1		A modified copy on a restriction of $M_1 \parallel M_1$
Version 2		A modified copy of $M_2$
Version 3		Still a restricted copy of $M_2$ in some sense

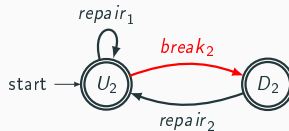
Homework: check that the effect of each version of  $R_4$  on the plant is the same.

## Requirement $R_{3,4}$ - Parallel composition point of view

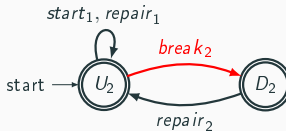
Requirement  $R_3$  (v.2): Machine 1 can't start processing a workpiece if Machine 2 is down.



Requirement  $R_4$  (v.3): If both Machines are down, then Machine 2 is repaired before Machine 1.

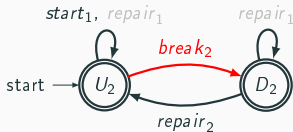


Requirement  $R_{3,4} := R_3 \parallel R_4$

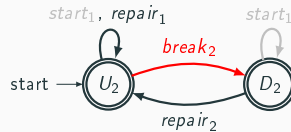


# Requirement $R_{3,4}$ - Product composition point of view

Requirement  $R_3$  (v.2): Machine 1 can't start processing a workpiece if Machine 2 is down.



Requirement  $R_4$  (v.3): If both Machines are down, then Machine 2 is repaired before Machine 1.



Requirement  $R_{3,4} := R_3 \times R_4$

