Design Formalisms--Petri Nets

1. Why design formalisms.
2. Background---Finite State Machines.
3. Limitations of FSM
4. Petri Nets
5. Using Petri Nets.

Why design formalisms?

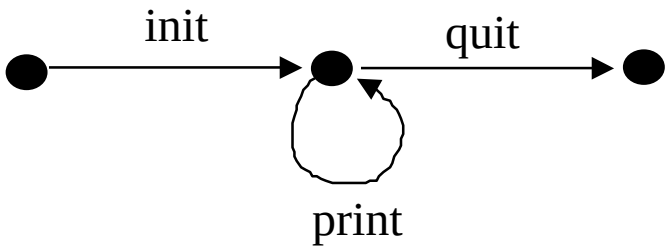
- Designers need to communicate with other designers
- Designers need to communicate with requirements writers, and with customers.
- Formal representations of design can be
  - *Automatically Analyzed:* Does this design have any performance bottlenecks? Can it lead to deadlocks?
  - *Manually Inspected* (as discussed earlier).
  - *Measured.* Measures like coupling and cohesion can be derived from designs.
  - *Verified.* Formal properties (such as if/how it meets customer requirements) can be verified from the design.
  - *Used for Testing.* Design representations can be used to create test scripts, etc.

Petri Nets are for modeling concurrent systems.

Finite State Machines

Example 1:

"A Print Spooler will first be initialized; it will then process any number print requests until a "quit" command is reached, then it will halt".



Example 2:

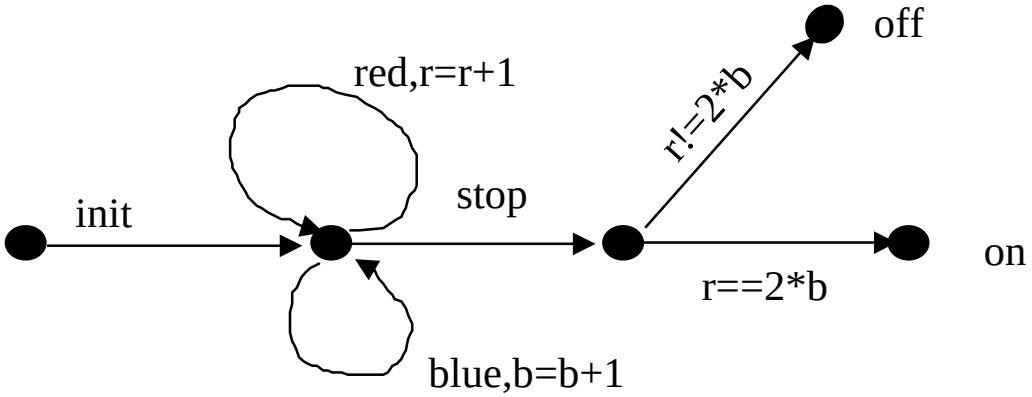
"When the button has been pushed an odd number of times, the light is on; otherwise it is off. ".

Example 3:

"When the number of times the button has been pushed is a prime number, the light will be on".

Example 4:

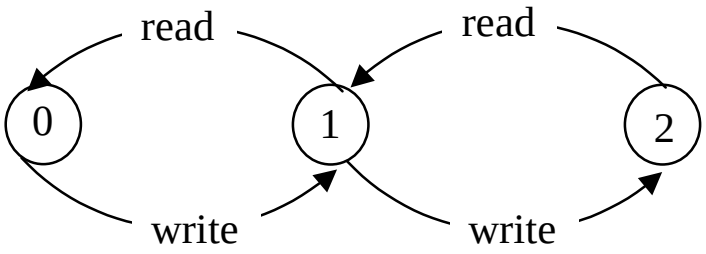
"After the init button is pushed, if the red button has been pushed twice as many times as the blue button, when the stop button is pushed, the light should be on. Otherwise off".



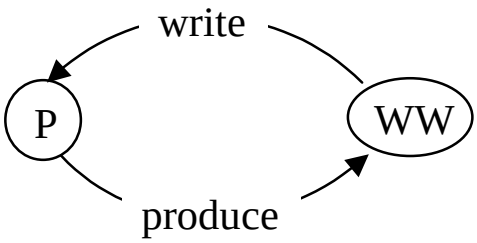
Double Buffer Example.

Producer and consumer are two asynchronous processes.

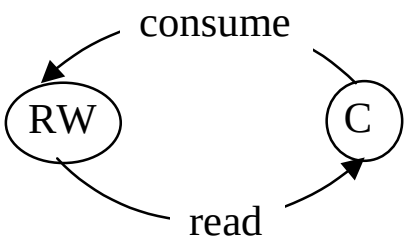
producer puts messages into a 2 slot buffer. A consumer reads messages from this buffer. If the buffer is empty, the consumer waits; if full the producer waits.



Buffer



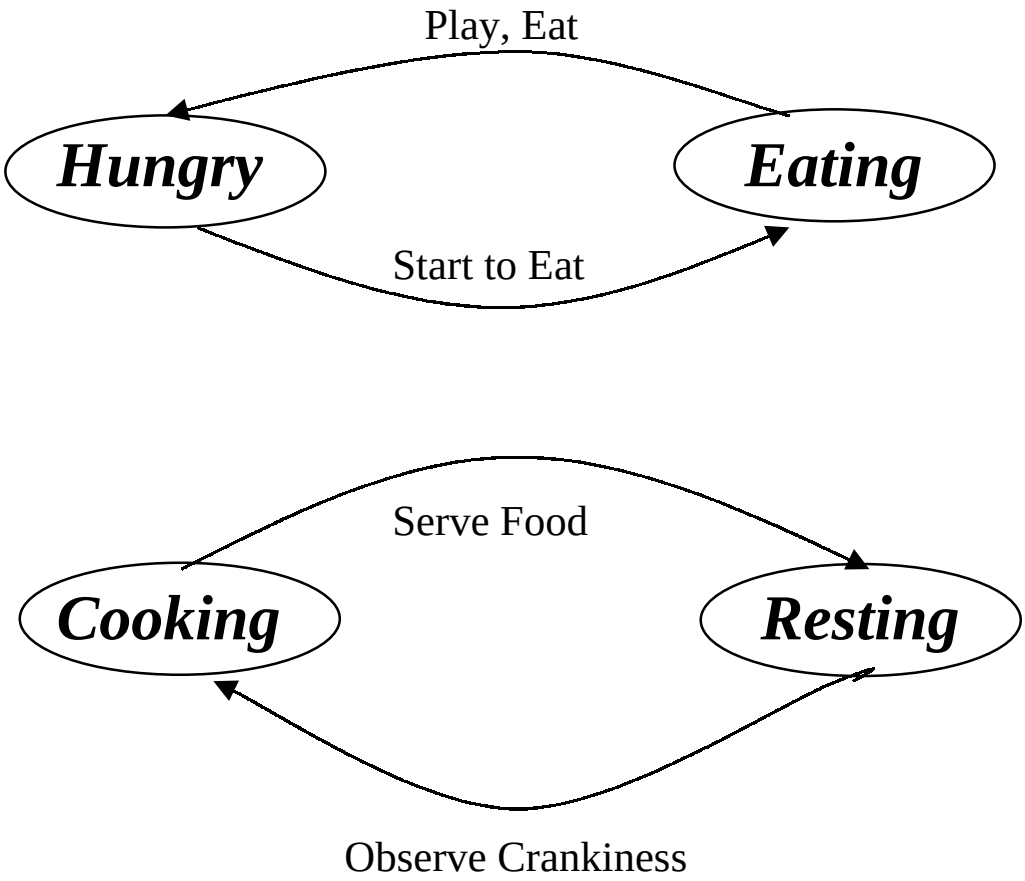
Producer



Consumer

- How does this work? (How can I grow more fingers?)
- Can these machines be combined?
  - What will the resulting machine look like?
  - Will it be an accurate model of reality?

Kids and Dad.



Petri Nets

Consists of:

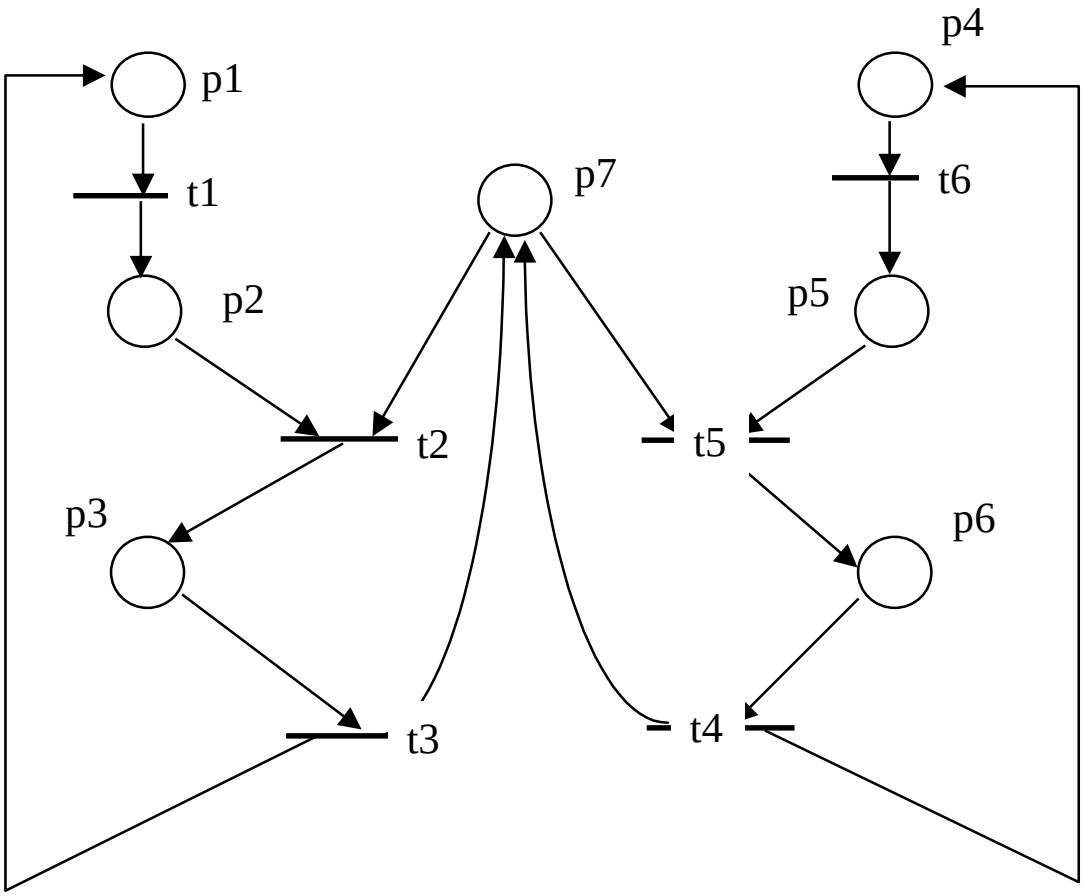
a) Finite set of *places*,

b) finite set of *transitions*,

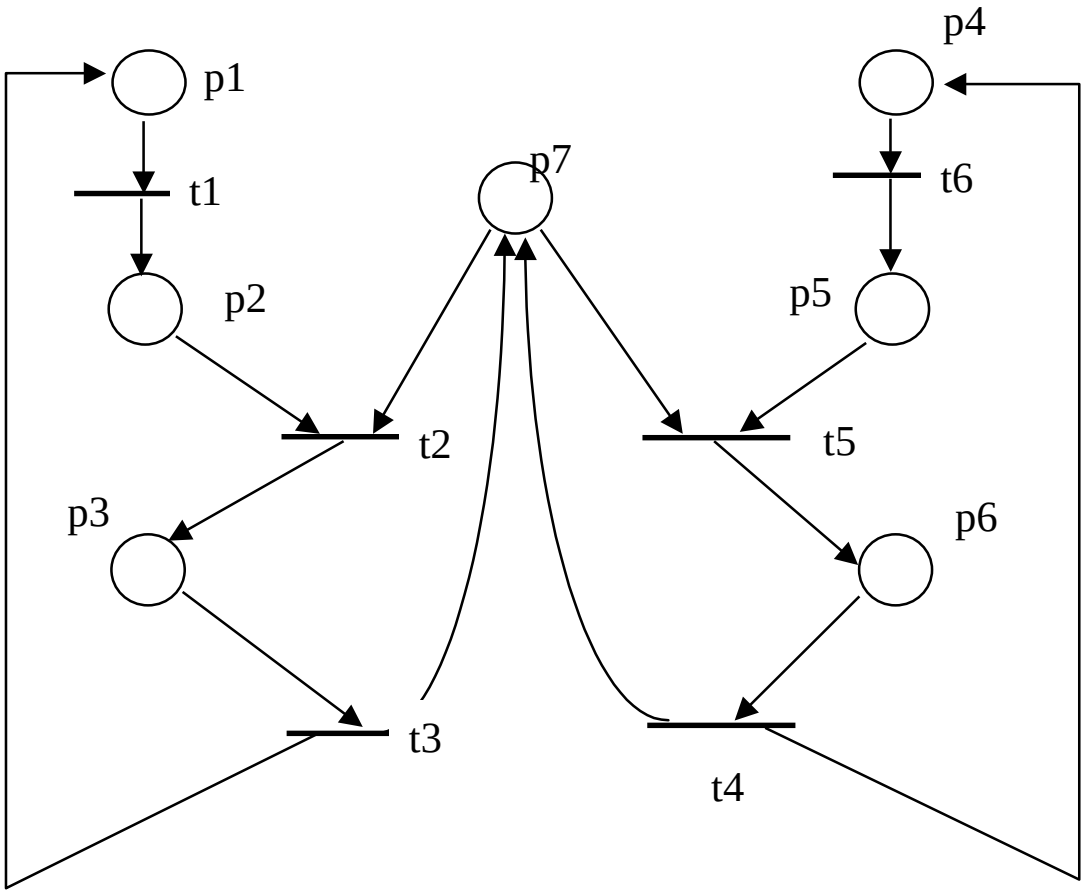
c) finite set of *arrows* connecting places to transitions or transitions to places.

A petri net is given a *state* by *marking* its places with a token.

A place with a token is *marked*.



How it works.



If an arrow comes into a place **p** from a transition **t**, **p** is **t**'s *output place*; If to **t** from **p**, it is **t's *i*nput place.**

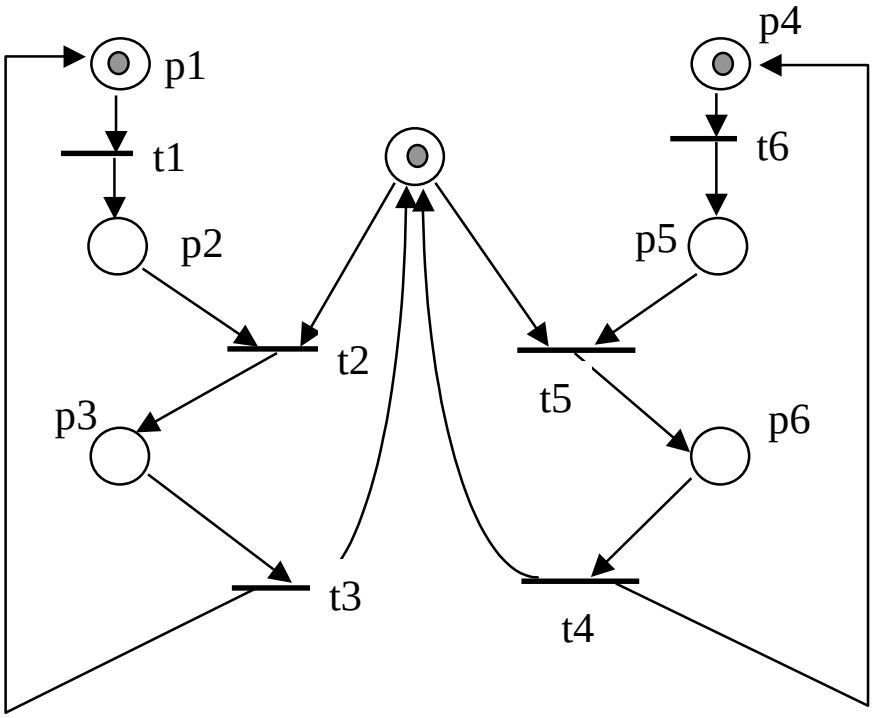
If all of a transition **t**'s input places have a token, the transiton *fires*, removes these tokens, and puts a token in each of **t's** output places.

A *firing sequence* is a sequence of possible firings, starting with an initial marking.

Note:

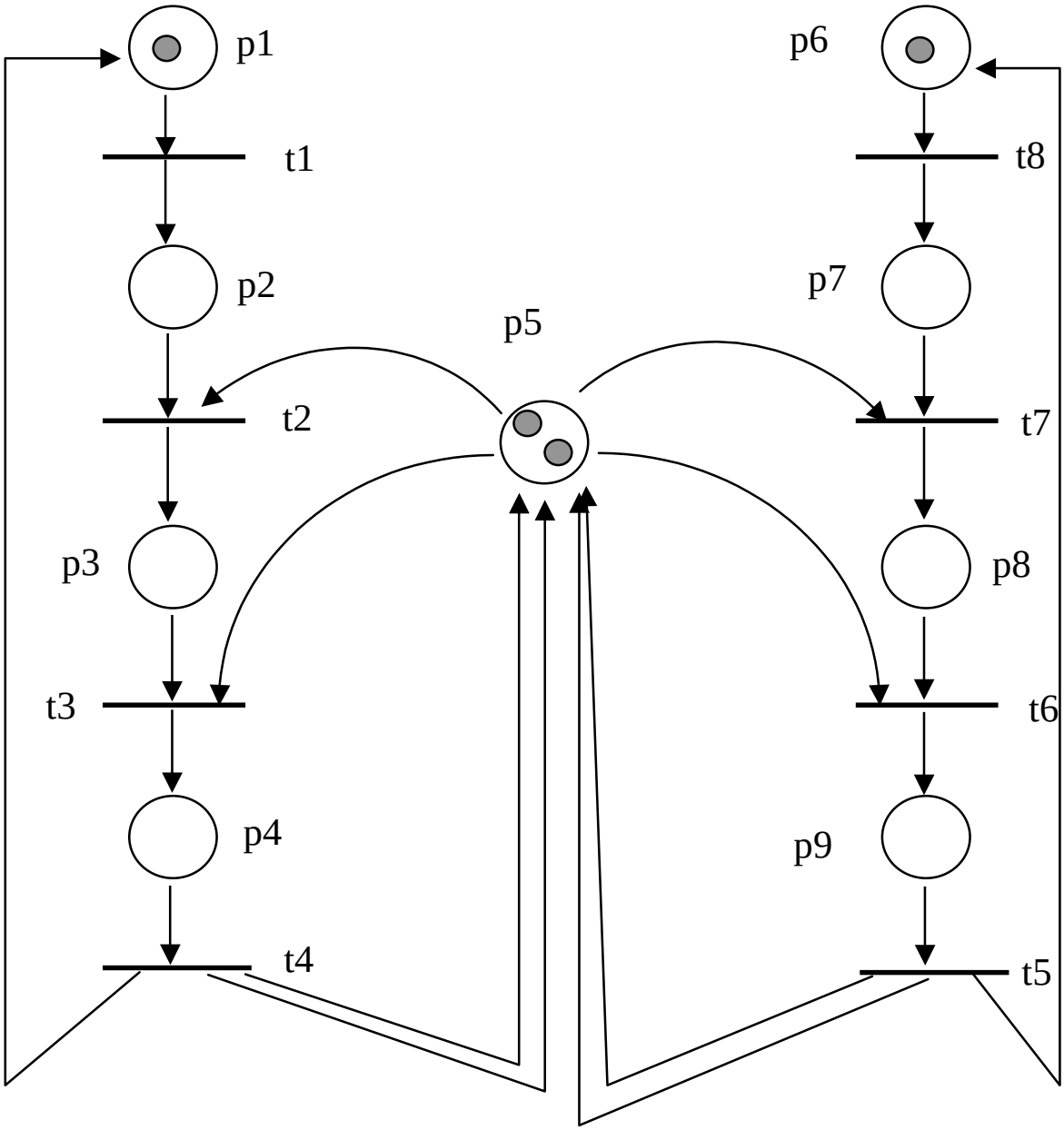
1) Several transitions may be enabled simultaneously.
2) They may happen in any order.
3) One transition may "compete" with another.

Petri net working example.

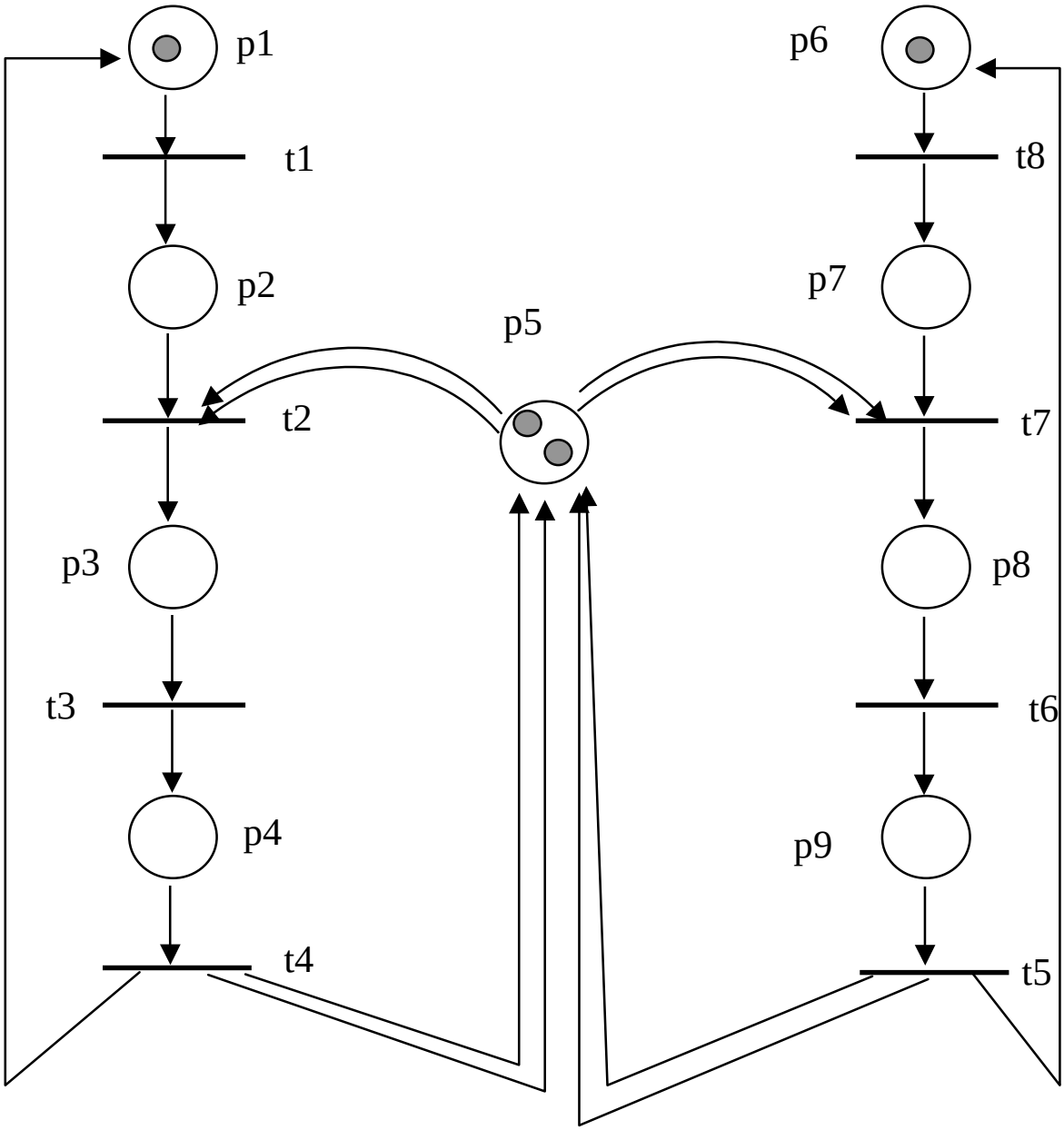


- What transitions (firing sequences) are possible?
- Will all of them happen?
- What transitions can happen together (***concurrent***)
- What transitions cannot (***conflict***)
- Can some transitions be prevented for ever (***starvation?)***
- What real world phenomena are being modeled here?

A bad, really bad, Petri Net

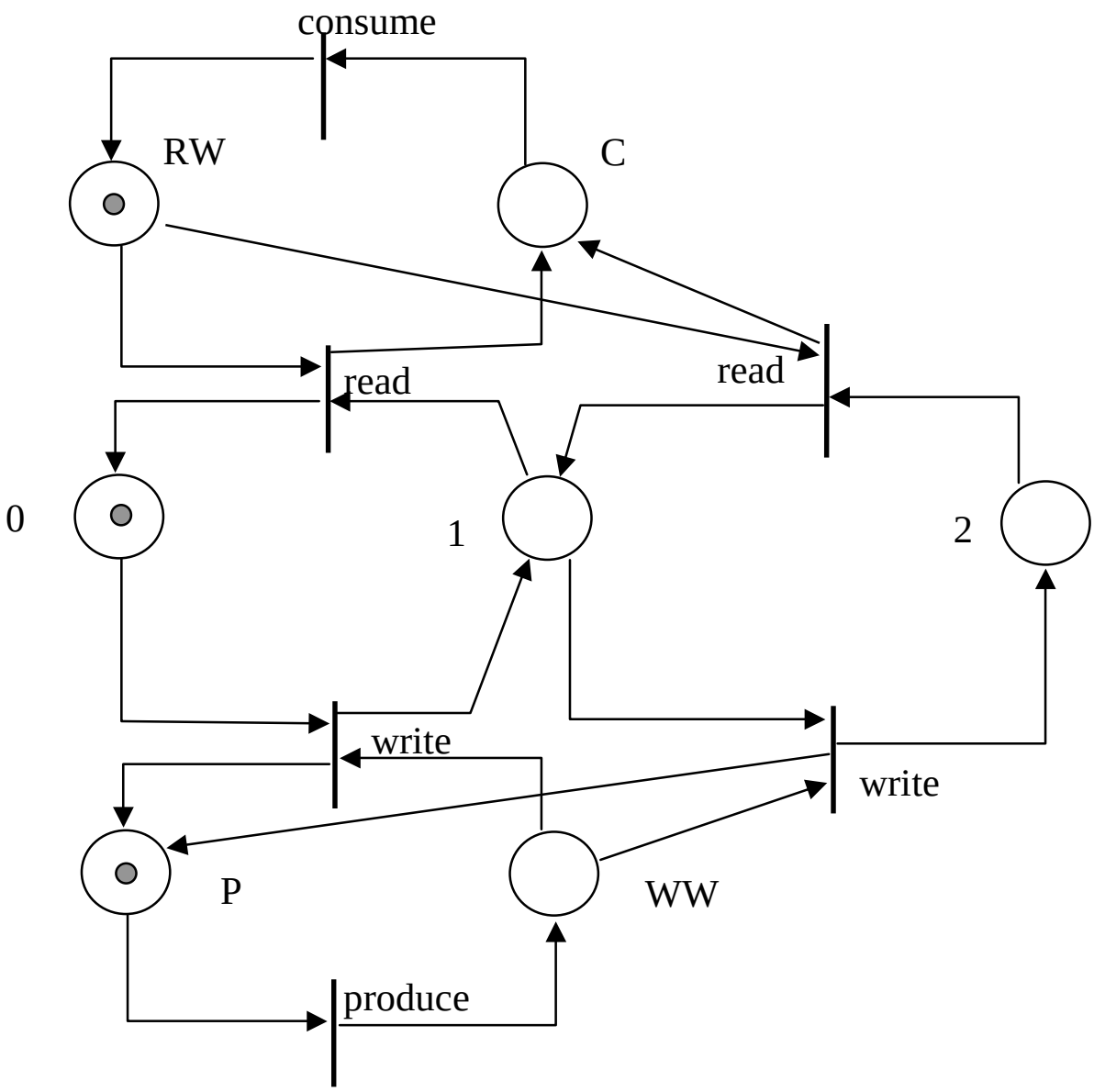


Fix



11

The Buffer Problem as Petri Net



Important: Study this carefully and convince yourself it really works!

Conclusions

Design formalisms are a helpful medium for communication between stakeholders.

They can also be used to *analyze*, *measure*, and *verify* designs.

Finite state machines are easy to understand, and well-known.

However, they cannot model certain types of situations that arise in concurrent systems.

Petri Nets are useful in such situations: they consist of *places* and *transitions*, with *arrows* connecting places to transitions.

Petri nets represent a group of finite state machine states via *markings*

Petri net transitions are *nondeteministic*, and potentially *concurrent.*

Petri nets can model *concurrency*, *resource contention*, *deadlock*, and *starvation.*